Universidad de las Ciencias Informáticas
Facultad 4

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN INFORMÁTICA

Solución de conflictos durante la sincronización de datos para el Sistema de Gestión Integral de Aduanas (GINA).

AUTOR
Fernando Nápoles Gámez

TUTOR
Ing. Alain E. Rodríguez Arias

Ciudad de la Habana, 2010.

"Año 52 de la Revolución"

DECLARACIÓN DE AUTORÍA

Declaro ser el autor de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Fernando Nápoles Gámez **Ing. Alain E. Rodríguez Arias**

_____ _____

Firma del Autor Firma del Tutor

"Si salgo llego, si llego entro, si entro triunfo."

Fidel Castro Ruz

…A Fidel, Raúl, y a la Revolución Cubana por la oportunidad de formarme y enseñarme que el camino, aunque difícil, es alcanzable y por hacerme ver cuánta responsabilidad tenemos en el presente y futuro de nuestra Cuba…

…A mi padre, por la atención prestada desde lejos, por sus correcciones y útiles sugerencias…

…A mi madre por sus ánimos y constante apoyo, preocupación y confianza...

…A Alain, por su guía y apoyo…

…A Michel Díaz Llerena, por las oportunas ideas brindadas…

…A Pimentel, Mena, Damián y Eliecer, por su atención...

…A Julio, por sus enseñanzas…

…A la UCI, por acogerme y mostrarme que el conocimiento es infinito y que lo que no dominamos aun es inmenso. ..

…A todos los profes, seños, maestros y auxiliares que me han formado: los del circulo infantil, de la primaria, secundaria, preuniversitario y la universidad…

…A Luis y Yeni por siempre estar ahí…

…A Yoel, por sus ocurrencias...

…A Adrian y Luis por los momentos de la "Liga"…

…A Alexis por el "Most Wanted"...

…A Roque, por enseñarme el "método" con el que todas las "cargas" se hacen más "ligeras" si aprendes a manejar las más "grandes". ..

…A todos los compañeros de brigada a través de estos 5 años…

A la memoria de mi abuelo Lucindo, por haber sido parte indispensable de mi niñez.

A mi madre Arlenis, por su ternura , cuidado y exigencia , por educarme lo mejor posible , por hacerme feliz sin ser ricos y mostrarme que nada vale más que la voluntad , los sentimientos , los principios morales y el alma de la personas.

A mi padre Exiquio, por su apoyo y exigencia constantes, gracias a lo cual pude entender muchas cosas de la vida y logré forjar el carácter necesario para enfrentar retos mayores.

A mi hermana Yani (tata), por ser mi amiga.

A mi nené Eli, por ser la luz de mi vida, por darme ese amor de novelas, tan limpio y apasionante .Por esperarme y amarme hasta ahora a pesar de la distancia. Por mostrarme que el orgullo no es superior al amor si se quiere de veras. Por enseñarme a escuchar.

El presente trabajo está encaminado a la solución de conflictos durante la sincronización de las bases de datos para el Sistema de Gestión Integral de Aduanas (GINA) utilizando como sistema de gestión de bases de datos Oracle 11g, debido a que está concebido actualmente como un sistema centralizado que debe ser capaz de funcionar como un sistema distribuido en situaciones excepcionales, donde en cada servidor se inserta información referente a un elemento determinado.

Al efectuarse el intercambio de información entre las bases de datos de los servidores, se produce el registro de información múltiple perteneciente a un mismo elemento, provocando redundancia, inconsistencia y pérdida de los datos, expresándose en la ocurrencia de conflictos en los registros almacenados.

Para resolver el problema se identifican los tipos de conflictos que ocurren, la estrategia de identificación unívoca de elementos que haga posible la detección de estos y se implementa el mecanismo automático de resolución de los mismos teniendo en cuenta dos tipos de nodos: central y secundario.

Palabras Claves:

Sincronización, sistema distribuido, nodos, resolución de conflictos, mecanismo automático.

Índice

INTRODUCCIÓN

La Aduana General de la República (AGR) es una organización destinada a supervisar el tráfico internacional de medios de transportes, mercancías y viajeros.

Como parte de los programas de informatización de sus variados y complejos procesos aduanales, la AGR ha venido desarrollando, apoyada por el Centro de Automatización para la Información y la Dirección (CADI) y con la colaboración de la Universidad de las Ciencias Informáticas, soluciones que van respondiendo a sus intereses de integración y eficiencia. Con tal fin se crea el sistema de gestión integral para los procesos de una aduana (GINA), que debe ajustarse a los crecientes requerimientos debido a las altas prestaciones que brinda y la necesidad de garantizar mayor disponibilidad. Unido a la información que se maneja, que por su carácter "secreto", no debe estar en el mismo servidor centralizada. El despliegue se efectúa desde niveles superiores a niveles inferiores como se observa en el esquema de la red nacional de la Aduana General de la República, Fig. 1.

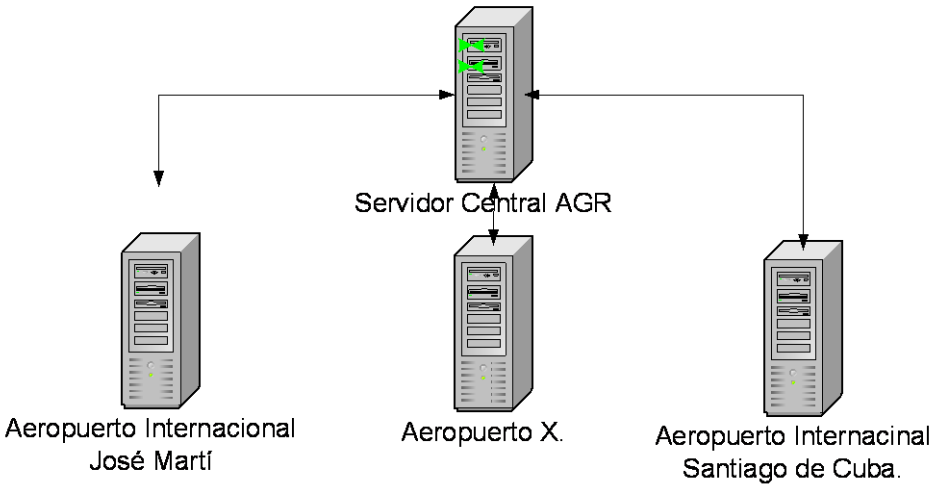

Fig. 1. Esquema de la red nacional de la Aduana General de la República.

En la concepción del sistema GINA, que debe funcionar como un sistema integral para la gestión de todos los procesos de una aduana, se determinó que toda la información se debe almacenar en una base de datos única. Como resultado del proceso de despliegue de las nuevas soluciones informáticas y la descentralización de la información para alcanzar mayor disponibilidad y seguridad de la información, el sistema se concibe además como un sistema distribuido con la presencia de un servidor central que funciona como puente entre los demás servidores y se encarga de manejar la información de estos, y que en situaciones excepcionales debe funcionar de manera que un nodo desconectado del servidor central debe ser capaz de continuar prestando servicios en forma autónoma y registrar las operaciones que se vayan realizando para posteriormente actualizar el servidor central y luego por réplica a todos los servidores de respaldo y los restantes que forman el sistema distribuido y que manejan información común.

Esto introduce una problemática al registrarse elementos nuevos en las operaciones como personas o medios de transporte que nunca han existido en el sistema. Si el elemento nuevo registra operaciones en el servidor local de una aduana que está funcionando desconectada y antes que ese servidor se reconecte también dicho elemento registra operaciones en otros nodos que estén conectados directamente o desconectados en el momento que todos los nodos se pongan activos hay que homogeneizar toda la información referente al elemento nuevo bajo un único identificador y con la precedencia entre las operaciones que ha realizado contra la aduana como entidad única.

También puede ocurrir que un elemento nuevo se registre bajo las mismas condiciones de desconexión pero en subsistemas diferentes que tienen intereses diferentes con respecto al concepto nuevo e igualmente en el momento que se vuelvan a conectar los nodos es necesario que el servidor central quede registrado un único elemento que contenga todas las informaciones provenientes de todos los subsistemas donde se insertó.

En la concepción del GINA, se ha identificado que pueden ocurrir conflictos en la información que se almacena entre los servidores de la red nacional de la Aduana General de la Réplica, el servidor central y los servidores locales o de respaldo, debido a que no existe el correcto tratamiento de la información durante sincronización entre dichos servidores.

Teniendo en cuenta la problemática presente, nos plantemos la solución de la misma concibiendo la formulación del **problema a resolver** de la siguiente manera:

¿Cómo garantizar la consistencia y evitar la redundancia de datos, durante la sincronización de las bases de datos de la red nacional de aduanas en Cuba?

El **objeto de estudio** está constituido por:

Manipulación de datos. Sistemas y algoritmos de replicación de Bases de Datos.

Persiguiendo como **objetivo general** desarrollar la solución de conflictos en la información registrada para el mismo elemento en diferentes servidores, durante la sincronización de las bases de datos para el Sistema de Gestión de Bases de Datos (SGBD) Oracle 11g. Centrándose para la investigación como **campo de acción** en la manipulación de datos, sincronización de bases de datos y resolución de conflictos para SGBD Oracle 11g en el sistema GINA.

Como **posibles resultados** se tienen**:**

- Clasificación y descripción de los tipos de conflictos que ocurren en el entorno descrito en la situación problemática.
- Descripción de la estrategia de identificación unívoca de un mismo elemento proveniente de diferentes servidores.
- Solución a cada tipo de conflicto descrito.
- Integración de la solución dentro del entorno de sincronización de datos identificado en la Aduana General de la República de Cuba.
- Implementación de la solución en un entorno que simule la situación actual de despliegue de la Aduana General de la República de Cuba para SGBD Oracle 11g.
- Evaluación de la solución con un caso de estudio predeterminado.

Para dar cumplimento al objetivo general se han concebido como **objetivos específicos**:

1. Detectar la ocurrencia de un conflicto en la situación descrita.

2. Identificar los tipos de conflicto que pueden ocurrir en el entorno descrito.
3. Definir la estrategia de identificación unívoca de elementos.
4. Obtener el estado del arte de las soluciones de resolución de conflictos y sincronización para el Sistema de Gestión de Bases de Datos (SGBD) Oracle y aplicaciones empresariales a nivel mundial y nacional.
5. Elaborar la propuesta de solución con su correspondiente justificación técnica para Oracle 11g.
6. Proponer la forma de integración de la solución propuesta dentro del entorno de sincronización de datos identificado para la Aduana General de la República de Cuba.
7. Implementar la solución propuesta para Oracle 11g en un caso de estudio que simule la situación actual de despliegue de la Aduana General de la República de Cuba.

El trabajo está estructurado en tres capítulos, tal y como se describe a continuación:

Capítulo 1: Se enuncian los principales conceptos relacionados con la resolución de conflictos y sincronización de bases de datos. Se realiza un estudio de las soluciones existentes. Se presentan tecnologías para ejecutar la sincronización de los datos. Se caracteriza la situación actual de sincronización de bases de datos distribuidas y resolución de conflictos en la AGR.

Capítulo 2: Se propone la solución de conflictos a utilizar por el sistema GINA basada en la estrategia de identificación unívoca de elementos que permita la detección, clasificación y resolución de los conflictos y se argumenta de forma teórica el por qué de la propuesta.

Capítulo 3: Se presenta de forma práctica la solución de conflictos propuesta basada en la estrategia de identificación unívoca de elementos que permita el tratamiento de dichos conflictos, además de las funcionalidades implementadas y la validación de la solución. .

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este capítulo se aborda el estado del arte de la réplica de datos a varios niveles: internacional, nacional y local, vinculado a la resolución de conflictos Se hace mención a un grupo de **conceptos** que son importantes para el desarrollo de nuestra investigación. Se incluyen las teorías y tecnologías más empleadas. Se abarca lo relacionado a las soluciones existentes con el objetivo conocer sus potencialidades, para aprovechar facilidades o efectuar una futura selección.

1.2 Fragmentación y sincronización de datos

Para comprender el objeto de estudio del trabajo se fundamentan algunos conceptos que nos acercarán al contenido de este trabajo.

Las técnicas de réplica y fragmentación se pueden aplicar sucesivamente a la misma relación de partida. Un fragmento se puede replicar y a su vez esa réplica ser fragmentada, para luego replicar alguno de esos fragmentos. (1)

1.2.1 Fragmentación horizontal y vertical.

Para fragmentar horizontalmente los datos, una Tabla T se divide mediante operaciones de selección en subconjuntos T1, T2,…Tn. Cada fragmento se ubica en un nodo o BD, y se reconstruyen con operaciones de unión.

En la fragmentación vertical, el proceso de división se realiza mediante operaciones de proyección, donde cada fragmento deberá incluir la llave o identificador de la tabla T (id). Este tipo de fragmentación se realiza generalmente atendiendo a la frecuencia de uso de la información, y su reconstrucción se realiza con una operación de join (palabra reservada del lenguaje de consulta, utilizada para formar criterios de selección entre dos o más tablas). (2)

1.2.2 Sincronización

Se refiere a sincronización como la coherencia total de un conjunto o porción de datos almacenados en BD diferentes. (3)

1.2.2.1 Tipos de sincronización

1. Adicionar Diferencias: Este método permite adicionar sólo las diferencias (tuplas o registros) de una fuente de datos origen hacia una fuente de datos destino.
2. Intercambio: Este método garantiza que los datos de una fuente origen se adicionen en la fuente destino y viceversa. Esta sincronización requiere un método de resolución de conflictos que permita tomar una decisión en caso de que ocurra una anomalía. Es muy utilizado en escenarios donde se permita trabajar a nodos del cluster en modo de desconexión que necesiten recuperarse después sin perder sus datos locales que a la vez deben sumarse a la base de datos central.
3. Copia Total: Este método establece una copia total de una fuente de datos origen hacia una fuente destino. Los datos de la fuente destino se pierden y son reemplazados por la tabla origen. Suele ser bastante costoso para la réplica instantánea y es muy utilizado para recuperaciones de fuentes de datos que inician desde cero o adición de nuevas fuentes a sincronizaciones ya establecidas.(3)

1.2.3 Flujo de Datos

Se entiende por flujo de datos, al transporte de información de un lugar a otro ya sea en el mismo ordenador o de una PC a otra. Este flujo de datos se denomina también Streams. El concepto de Streams o Flujo de Datos abarca desde leer o escribir datos en un ordenador hasta replicar datos de una PC en otra.

1.3 Réplica de datos

El problema de la réplica de datos entre servidores ha sido estudiado por las compañías desarrolladoras de gestores de datos, con el objetivo de garantizar la disponibilidad de la información.

Un sitio es una copia de la estructura de la BD, la cual contiene todos los datos o parte de ellos y una aplicación para comunicarse con esta.

Se entiende la réplica de datos como un proceso que permite copiar y distribuir idénticamente tablas desde una BD hacia uno o múltiples sitios de una red. Este proceso asegura que los datos estén siempre disponibles en el lugar necesario para ser utilizados en el momento indicado. Un sistema maneja ***Réplicas de Datos*** si una relación dada(o, en términos más generales, un fragmento dado de una relación) se puede representar en el nivel físico mediante varias copias almacenadas o réplicas, en muchos sitios distintos.(4)

Generalmente las situaciones se presentan en alguno de los siguientes entornos de red:

- ***Homogéneos:*** Réplica de datos entre servidores de datos con el mismo gestor y sobre mismo sistema operativo.
- ***Homogéneos con diferentes plataformas:*** Réplica de datos entre servidores de datos con el mismo gestor pero donde los sistemas operativos son diferentes. Ejemplo Oracle Enterprise Edition instalados en plataformas Windows y Linux.
- ***Heterogéneos:*** Réplica de datos entre servidores de datos con diferentes gestores de datos y el mismo sistema operativo. Ejemplo PostgreSql vs Oracle o SQL Server vs Oracle, ambos corriendo sobre el mismo sistema operativo.
- ***Heterogéneos con diferentes plataformas:*** Réplica de datos entre servidores de datos con diferentes gestores de datos y diferentes sistemas operativos. Ejemplo: PostgreSql vs Oracle o SQL Server vs Oracle uno en Windows y otro en Linux. (5)

1.3.1 Entornos de réplica

La réplica de datos puede aplicarse de dos maneras diferentes de acuerdo a las necesidades propias de las personas o instituciones que la lleven a cabo, estas son:

Maestro-Esclavo: También es conocido como de solo lectura, porque permite a un solo sitio (maestro) realizar consultas de escritura sobre los demás, mientras estos solo pueden hacer consultas de lectura (esclavos).

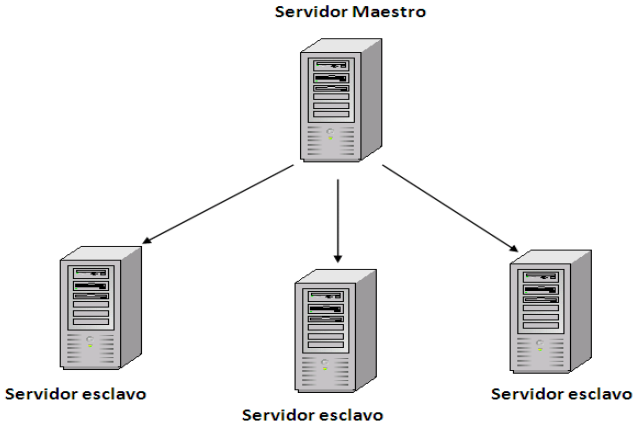

Fig. 2. Entorno de Réplica Maestro – Esclavo

Maestro-Maestro*:* También llamado par-a-par o réplica de camino de n, porque permite múltiples sitios (maestros), actuando como pares iguales. Cada sitio es un sitio maestro, y se comunica con otros sitios maestros. Esta capacidad tiene también un severo impacto en el desempeño debido a la necesidad de sincronizar los cambios entre todas las partes que intervienen en la réplica. Este tipo de entorno puede ser usado para mantener sitios recuperables ante posibles desastres o caídas, así como para proveer sistemas con alta disponibilidad y para balancear la carga de consultas a través de las distintas ubicaciones.

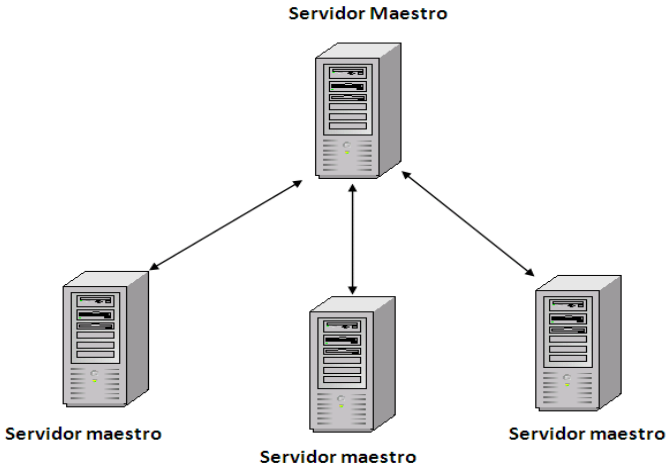

Fig. 3. Entorno de Réplica Maestro – Maestro

La situación descrita se ha identificado pertenece a un entorno multimaestro o maestro-maestro.

1.3.1.1 Tipos de replicación

Los tipos de replicación están relacionados con el momento en que se realiza la actualización de los datos cuando se efectúa una transacción de escritura en la base de datos.

Replicación Asíncrona: En el caso de la replicación asíncrona, las actualizaciones a una réplica son propagadas hacia los demás en algún momento posterior, no dentro de la misma transacción, por lo tanto

la replicación asíncrona presenta un retardo de tiempo o latencia, durante el cual es posible que las réplicas no sean idénticas.

Replicación Síncrona: En el caso de la replicación síncrona, si se actualiza una réplica dada, todas las demás réplicas del mismo fragmento de datos también se actualizan dentro de la misma transacción; lo que implica que (desde un punto de vista lógico) solo existe una versión de los datos. La mayoría de los productos implementan la replicación síncrona por medio de procedimientos disparados (posiblemente ocultos y manejados por el sistema). Sin embargo la replicación síncrona tiene la desventaja de que impone una sobrecarga sobre las transacciones que actualizan cualquier réplica (también puede haber problemas de disponibilidad).(6)

La replicación asíncrona se ajusta a las características de la situación.

1.3.2 Identificación unívoca de entidades.

Atributos

Los atributos son las propiedades que describen a cada entidad en un conjunto de entidades. Un conjunto de entidades dentro de una entidad, tiene valores específicos asignados para cada uno de sus atributos, de esta forma mediante la selección de un subconjunto de estos es posible su identificación unívoca. .

Ejemplos:
A la colección de entidades Alumnos, con el siguiente conjunto de atributos en común, (id, nombre, edad, semestre), pertenecen las entidades:

- (1, Ernesto, 18 años, 2)
- (2, Juan, 19 años, 5)
- (3, Ana, 20 años, 2)

Cada una de las entidades pertenecientes a este conjunto se diferencia de las demás por el valor de sus atributos. Es evidente que dos o más entidades diferentes pueden tener los mismos valores para algunos de sus atributos, pero nunca para todos.

En particular, los atributos identificativos son aquellos que permiten diferenciar a una instancia de la entidad de otra distinta. Por ejemplo, el atributo identificativo que distingue a un alumno de otro es su número de id.

Para cada atributo, existe un dominio del mismo, este hace referencia al tipo de datos que será almacenado o a restricciones en los valores que el atributo puede tomar (Cadenas de caracteres, números, solo dos letras, solo números mayores que cero, solo números enteros). Cuando una entidad no tiene un valor para un atributo dado, este toma el valor nulo, bien sea que no se conoce, que no existe o que no se sabe nada al respecto del mismo.

1.3.2.1 Claves

Es un subconjunto del conjunto de atributos comunes en una colección de entidades, que permite identificar unívocamente cada una de las entidades pertenecientes a dicha colección. Asimismo, permiten distinguir entre sí las relaciones de un conjunto de relaciones.

Dentro de los conjuntos de entidades existen los siguientes tipos de claves:

Superclave: Es un subconjunto de atributos que permite distinguir unívocamente cada una de las entidades de un conjunto de entidades. Si otro atributo unido al anterior subconjunto, el resultado seguirá siendo una superclave.

- **Clave candidata**: Dada una superclave, si esta deja de serlo removiendo únicamente uno de los atributos que la componen, entonces esta es una clave candidata.
  - **Clave primaria:** Es una clave candidata, elegida por el diseñador de la base de datos, para identificar unívocamente las entidades en un conjunto de entidades.
  - **Clave única:** Tiene la misma función de la clave primaria pero a diferencia de ella, puede aceptar valores nulos.

Los valores de los atributos de una clave, no pueden ser todos iguales para dos o más entidades.

Para distinguir una **tupla** de otra, se recurre al concepto de "**llave primaria**", o sea a un conjunto de atributos que permiten identificar unívocamente una **tupla** en una relación. En una relación puede haber más combinaciones de atributos que permitan identificar unívocamente una **tupla** ("llaves candidatas"), pero entre estas se elegirá una sola para utilizar como llave primaria.

Los atributos de la llave primaria no pueden asumir el valor nulo (que significa un valor no determinado), en tanto que ya no permitirían identificar una **tupla** concreta en una relación. Esta propiedad de las relaciones y de sus llaves primarias está bajo el nombre de **integridad de las entidades** (*entity integrity*). (7)

1.3.3 Conflictos de replicación de datos

Existen varios tipos de conflictos a tener en cuenta, al tratar la replicación de datos. Los conflictos pueden ocurrir cuando estamos trabajando en un ambiente de replicación que permite actualizaciones concurrentes sobre los mismos datos en múltiples sitios. (Replicación asíncrona multimaestro).

Estos conflictos son:

- **Conflicto de actualización:** Este conflicto ocurre cuando dos transacciones originadas desde distintos sitios actualizan el mismo registro, en forma cercana en el tiempo.
- **Conflicto de unicidad:** Sucede cuando la replicación de un registro intenta violar una restricción de integridad, ya sea por llave primaria o única (*Primary Key* o *Unique).* Por ejemplo considere lo que sucede cuando dos transacciones originadas de dos sitios diferentes, cada una inserta un registro, en su respectiva tabla, con el mismo valor de clave primaria.
- **Conflicto de supresión:** Un conflicto de supresión ocurre cuando dos transacciones originadas de sitios diferentes, una de ellas intenta borrar un registro, y la otra actualizar o borrar el mismo registro, ya que en este caso el registro no existe, tanto para ser actualizado como para ser eliminado.
- **Conflicto de orden:** Los conflictos de orden pueden ocurrir en ambientes de replicación con tres o más sitios maestros. Si la propagación al sitio maestro X, está bloqueada por alguna razón,

entonces la replicación de modificaciones en datos puede seguir siendo propagada a través de los otros sitios maestros; al finalizar la propagación, estas modificaciones debieron ser propagadas al sitio X en un orden diferente a como ocurrieron en los otros sitios maestros, pudiendo producirse un conflicto. (8)

1.3.3.1 Tipos de conflictos presentes

En la situación que nos compete podemos identificar dos tipos de conflictos principales, que se podrían ubicar dentro de la clasificación de conflictos de unicidad expuestos anteriormente, estos son:

TIPO 1: conflicto de elementos en un mismo subsistema:

Suceden dentro de un mismo subsistema del sistema GINA. Estos elementos tienen el mismo significado para el negocio ya que representan al mismo concepto dentro de la base de datos. Un ejemplo de ello puede darse en el subsistema Control de Personas, que trabaja directamente vinculado con el esquema Persona de la base de datos, ya que puede darse la aparición de dos personas que representan a la misma pero que poseen datos distintos y se debe decidir qué información conformará a la persona definitiva, garantizando que esté en el elemento definitivo la información relevante de ambos. Este tipo de conflicto está relacionado con los conflictos de unicidad descritos anteriormente debido a que se refiere a la violación de restricciones de integridad de identificación como la clave primaria o la clave única, y su relación con la necesidad de identificar unívocamente un elemento dentro del contexto global de la Aduana General de la República.

TIPO 2: conflicto de elementos en distintos subsistemas:

Suceden en el marco entre varios subsistemas o esquemas del sistema GINA. Suceden al darse la situación de que el elemento no brinda la misma información respecto al contexto o el significado que representa en dependencia del subsistema o el esquema donde esté situado. Este tipo de conflicto se refiere a la necesidad de que toda la información de un elemento determinado existente en los diferentes subsistemas, en dependencia del significado que representen para este, esté debidamente referenciada por claves foráneas a través de un único identificador válido que permita relacionar toda esa información al elemento correcto. Un ejemplo de ello es la interacción entre los subsistemas de Control de personas y

Medios de Transporte Internacional (MTI), pues aunque ambos inciden sobre elementos comunes como las personas en distintos escenarios, se necesita contar con la forma de identificarlos de manera única ante la posibilidad de reconocerlos como pasajeros o tripulantes y como una persona de interés para la aduana , todo ello aunque ocurra en distintos nodos y unificar esa información para tener claridad de las operaciones de estos elementos ante la Aduana General de La República como entidad

1.3.4 Resolución de conflictos

La manera que se pueden resolver cada uno de los clásicos conflictos se explica a continuación. Para evitar conflictos de:

Actualización. Para este caso existen tres formas aceptadas de resolverse:

- **Prioridad:** Cada servidor obtiene una prioridad única, y el servidor de mayor prioridad gana respecto a los demás servidores.
- **Timestamp**: La más nueva o la más antigua de las modificaciones es la considerada correcta, y por defecto, si no se eligió ninguno de los criterios gana la más nueva.
- **Particionamiento de datos:** Se garantiza que cada registro sea manipulado por un único servidor, lo que simplifica la arquitectura.

Unicidad: También existen tres formas aceptadas para resolverse:

Brindar un rango distinto de números para los generadores de claves en cada nodo.

Agregar el identificador del servidor a la clave primaria.

Supresión: Para evitar este tipo de conflictos, una posible solución es que los sitios marquen lógicamente los registros a ser borrados y que periódicamente el sitio maestro corra un proceso que realice el “delete” físico de los datos, es decir que desde los sitios replicados no se puede ejecutar una sentencia “delete”.

Orden: Este tipo de conflicto suele resolverse asignándole distintas prioridades a los sitios maestros, de forma que se ordenen las transacciones de acuerdo a estas prioridades.

En la nuestra situación en relación a la solución de los dos tipos de conflictos principales identificados podemos decir que todas de estas prácticas anteriormente enunciadas para evitar conflictos podrían ser válidas en la situación pero no son suficientes pues en el caso de los conflictos de unicidad por lo general se selecciona uno de los elementos en conflicto y se desechan los demás, por lo contrario en nuestro caso toda la información contenida en los distintos elementos es relevante y por tanto se debe hacer una integración entre ellos y conformar un único elemento que contenga o represente todas operaciones realizadas sobre él y trasladar ese nuevo elemento a los demás nodos para su conocimiento .

Formas de Resolución de Conflictos:

Existen varias formas de resolver los conflictos pero en este caso nos referimos a la manera que pueden ser tratados en un sistema informático. Para ello hay que tener presente cómo se desea que el sistema gestione los conflictos que en un momento anterior han sido identificados. El tiempo de respuesta según las prioridades o necesidades del negocio deben considerase y hasta qué punto es posible el uso de una de estas formas y su influencia en los resultados finales que se esperan. El tratamiento a los conflictos es un tema crucial para los sistemas que deban enfrentar esta tarea.

Estas formas son:

Resolución automática:

En esta forma de resolución se le da al sistema la responsabilidad de resolver de forma nativa los conflictos identificados, para ello es necesario dotar el sistema de métodos y estrategias de resolución de estos conflictos adecuadas y relacionadas con el negocio donde se encuentra o pertenece.

La ventaja de esta forma de resolución es la baja latencia entre la ocurrencia de un conflicto determinado y su resolución, además de que no depende de la intervención de un operador lo que hace el proceso más rápido.

La principal limitación es poder darle al sistema todos los criterios de decisión para los variados conflictos detectados, pues pueden presentarse disímiles casos.

Resolución manual:

La resolución manual es brindada por el sistema a través de una interfaz generalmente, donde se muestran los conflictos ocurridos y se le da la responsabilidad a un trabajador que interactúe con el sistema de decidir cuál será la solución a cada uno. El sistema debe en estos casos brindar algunas opciones diferentes que puedan ser seleccionadas.

La ventaja de esta forma de resolución es que permite brindar una solución más general para los conflictos, proponiendo varios casos básicos u otros al no cumplirse una condición determinada.

Su principal limitación es la intervención de un trabajador que tome las decisiones y esto puede conllevar a una mayor latencia entre la ocurrencia de la transacción, la identificación de un conflicto y su solución.

1.3.6 Reglas para la Replicación

Una regla es un objeto de la base de datos que le permite a un cliente realizar una acción cuando un acontecimiento ocurre y una condición es satisfecha. La misma está constituida por varios componentes.

Define que elementos de la base de datos en cuestión se van a replicar, pues deben cumplir con las condiciones especificadas para ello.

Componentes de una Regla

Una **regla** consta de los siguientes componentes:

- Condición de Regla.
- Contexto de Evaluación de la Regla (Opcional)
- Contexto de Acción de la Regla (Opcional)

1.4 Tecnologías actuales

Existen varias tecnologías que tratan lo relacionado a la réplica de datos, pero la más relevante y novedosa de estas es Oracle Streams.

1.4.1 Tecnología Streams de Oracle

Oracle Streams, es una tecnología para compartir información, detectando que información es relevante y quienes la utilizan. Esta tecnología es utilizada por Oracle 11g para propagar los cambios en un ambiente replicado. Se basa en tres acciones aplicadas a la información, Captura, Propagación y Consumo. Para la replicación la Captura se relaciona con un mecanismo que toma los cambios de los Redo Log. El almacenamiento se vincula cuando los cambios capturados son enviados al área de almacenamiento y estos cambios son propagados a las áreas de almacenamiento de los equipos remotos. El Consumo es la acción que se encarga de aplicar los cambios almacenados en la base de datos destino. Las tablas replicadas pueden ser diferentes, Oracle Streams se encarga de transformar la información para ajustarla a la base de datos de cada sitio replicado. El Proceso de Captura configurado para recolectar cambios realizados, recupera los mismos desde los Redo Log, formatea la información como Registros Lógicos de Cambios (LCR Logical Change Records por sus siglas en inglés) y coloca los LCRs en el área de almacenamiento de la base de datos local. Los LCRs son entonces propagados desde el área o cola de almacenamiento de la base de datos origen, hacia las áreas de almacenamiento de las bases de datos destino de la replicación. Los componentes de consumo de las bases de datos destino, son configurados para recibir los cambios y aplicarlos. La principal limitación de esta tecnología es la no abstracción con respecto a la estructura de la base de datos, pues su configuración es específica para cada tabla de la base de datos, definiendo el código de forma estática, por lo que no se cuenta con una variante que sea dinámica y permita la reutilización del código y se ajuste a los cambios en la estructura de la base de datos.

1.4.2 Soluciones existentes.

Dentro de las soluciones propuestas para la réplica de datos las más interesantes son:

- Réplica bidireccional basada en control de cambios.
- Sistema de réplica para bases de datos distribuidas en PostgreSql.
- Herramienta de replicación Asíncrona Multimáster para PostgreSql
- Réplica usando la tecnología Streams de Oracle 11g.
- Solución para la réplica de datos del Proyecto SIGEP.

A continuación fundamentamos un poco cada una de estas soluciones:

1.4.3.1 Réplica bidireccional basada en control de cambios

Esta solución fue propuesta por el proyecto Identidad de la Facultad 1 de la Universidad de las Ciencias Informáticas, se caracteriza fundamentalmente por ser bidireccional y basado en control de cambios y no en colas lo que evita las transmisiones redundantes. Soporta ambientes heterogéneos, no importa el gestor de base de datos. Soporta particionamiento horizontal de los datos, está implementado sobre la plataforma .NET y posee una topología flexible. No soporta el entorno de replicación maestro-esclavo. La principal limitación de esta solución es la plataforma en que es desarrollada, por el hecho de no ser libre y requerir licencias para su uso y depender en gran medida de la intervención de un operador para efectuar las tareas principales como la sincronización de los cambios.

1.4.3.2 Sistema de réplica para bases de datos distribuidas en PostgreSql.

Es una aplicación de escritorio programada en Python y como lenguaje para la interfaz gráfica la librería gtk y la librería glade. Realiza la réplica de datos en un solo entorno de réplica que es maestro- esclavo, y solo está implementada para el gestor de base de datos PostgreSql, como herramienta de replicación utilizan Slony I. Sus principales funcionalidades son:

- Configurar la réplica de espejo.
- Configurar la réplica fragmentada.
- Reporte de réplica.
- Realizar mantenimiento.

Por ser desarrollada para el Sistema de Gestión de Base de Datos PostgreSql no representa utilidad para la solución del problema, aunque se tendrán en cuenta detalles generales relacionados con los mecanismos en su implementación.

1.4.3.3 Herramienta de replicación Asíncrona Multimaestro para PostgreSql.

Esta herramienta es desarrollada por el grupo EIPAD de la Facultad 4 de la Universidad de las Ciencias Informáticas y es denominada Chronos. Chronos es una herramienta de réplica asíncrona para entornos

multimaestro que pretende satisfacer la mayor parte de los requerimientos. Ha sido diseñada a partir de necesidades particulares de un escenario de réplica, asumiendo las principales potencialidades de las soluciones similares en la actualidad para PostgreSql e implementando nuevas mejoras que la distinguen del resto. Chronos puede realizar replicación asíncrona multimaestro instantánea y programada. Esta implementada en lenguaje de programación C++.

Por ser desarrollada para el Sistema de Gestión de Base de Datos PostgreSql no representa utilidad para la solución del problema, aunque se tendrán en cuenta detalles generales relacionados con los mecanismos en su implementación.

1.4.3.4 Réplica usando la tecnología Streams de Oracle 11g.

Esta solución se basa en una propuesta de configuración e implementación para el proceso de Replicación de Datos, entre las bases de datos nacionales que soportan el sistema de Circulados Nacionales del Ministerio del Interior, usando la tecnología Streams de Oracle 11g. El uso de una implementación adecuada de la tecnología Streams de Oracle 11g, permitió elevar el desempeño del proceso de Replicación de datos perteneciente al sistema de Circulados Nacionales. Muestra las grandes ventajas de uso y factibilidad de Oracle Streams. La principal limitación de esta tecnología radica en que es tratada con la implementación de procedimientos y funciones a nivel de la base de datos. Aunque tiene una interfaz de administración como parte del Oracle Enterprise Manager, esta es algo limitada. Es válido resaltar que tiene incorporado procedimientos relacionados con la solución de conflictos de actualización, que pueden ser configurados según las necesidades.

1.4.3.5 Solución para la réplica de datos del Proyecto SIGEP.

Reko es la solución desarrollada por el Proyecto Prisiones de la Facultad 4 de la Universidad de las Ciencias Informáticas. Este software de réplica de datos surge como respuesta a la necesidad de mantener actualizadas un conjunto de bases de datos. Se fundamenta en la copia de información de una localización a otra. Pretende cubrir las principales necesidades relacionadas con la distribución de datos entre los gestores más populares como la protección, recuperación, sincronización, transferencia de datos entre diversas localizaciones o la centralización de la información en una única localización. Actualmente es compatible con SGBD Oracle y PostgreSql.

Permite además la representación de complejos escenarios de replicación con herramientas para la definición de localizaciones o nodos y la selección de los datos de replicación. Cuenta además con una herramienta Web para la administración y monitoreo de los datos de replicación.

Características

Tipos de replicación: Soporta la replicación de transacciones a través de Hibernate y la replicación de acciones a través de triggers.

Gestores soportados: Se integra actualmente con los gestores de bases de datos Oracle y PostgreSql.

Selección de datos: Provee la facilidad de seleccionar el subconjunto de tablas y datos a ser replicados de la base de datos mediante la definición de filtros aplicables a las tablas y la selección de usuarios propios del gestor.

Transmisión de datos: La transferencia de datos de replicación puede realizarse a soporte para replicación sobre TCP/IP, FTP, HTTP o por ficheros de forma manual. Los archivos de gran tamaño son enviados por FTP permitiendo resumir la trasmisión caso de interrupciones en la red.

Configuración entre nodos: El mecanismo de registro entre nodos de replicación se basa únicamente en un identificador (id) del nodo lo que permite la abstracción de los datos físicos de cada nodo como son el IP y el protocolo de comunicación. Este mecanismo permite que los nodos puedan moverse por distintas subredes y mantener sus datos sincronizados. Por ejemplo, mantener sincronizada la base de datos de una laptop con la base de datos central a través de Internet.

Seguridad: Los nodos de replicación manejan credenciales entre ellos para verificar la autenticidad de los datos transferidos. El envío de datos se realiza utilizando protocolos de comunicación seguros como son *Protocolo seguro de transferencia de hipertexto* o Hypertext Transfer Protocol Secure (HTTPS), por sus siglas en inglés, y Secure Socket Layer **(**SSL), por sus siglas en inglés.

Monitoreo: Permite conocer el estado de los datos transmitidos, puede realizarse en tiempo real a través de la Web así como dar un seguimiento al funcionamiento interno del mecanismo.

Interfaz visual Web: La administración y configuración de la réplica se realiza a través de una interfaz Web, por lo que es administrable vía remota usando solamente un navegador.

Independiente de la plataforma: El software de réplica puede ser instalado en cualquier sistema operativo y no necesita de un ambiente gráfico en el mismo para su funcionamiento.

Tolerancia a fallos: Detecta errores de conexión. Mantiene los datos de réplica en un estado estable en caso de desconexión. Al restablecerse la conexión, automáticamente sincroniza los datos entre las bases de datos.

Reko representa de gran utilidad para la transferencia de datos y su posible vinculación con solución del problema por la arquitectura que tiene definida y su relación con el SGBD Oracle 11g, sobre el cual se trabaja, esta permite abstraerse del la estructura de la base datos en distintos momentos, al ser agregadas tablas en la base de datos, solo es necesario agregarlas en la configuración del nodo en cuestión a nivel de aplicación lo que nos permite no depender del trabajo manual y directo en la base de datos.

1.5 Conclusiones del capítulo.

Actualmente en la Aduana General de la República no se cuenta con una forma de mantener la consistencia de la información referente a un elemento de la base de datos al efectuarse la sincronización en un ambiente de bases de datos distribuidas.

La situación descrita representa un entorno de réplica multi-maestro y asíncrono, donde la solución podrá estar vinculada al mecanismo de propagación que se utilice dada la estructura de los elementos que formen la arquitectura del mismo. Asimismo el tratamiento de los conflictos que se presenten, podría establecerse como un mecanismo que se encargue de esta tarea una vez ejecutado del proceso de aplicación de los datos por parte del software o mecanismo de réplica, el cual se encarga de efectuar la transferencia de los cambios ocurridos en las bases de datos sincronizadas. De esta forma se considera es posible brindar una solución flexible y que garantice la abstracción de los datos y el dinamismo de la misma.

Referente a la necesidad de identificar unívocamente un registro en la base de datos en la situación descrita, es conocido que el modelo entidad-relación brinda elementos para la identificación unívoca de una tupla perteneciente a una tabla de la base de datos, a través de del uso de claves primarias, claves candidatas y claves únicas, haciendo uso también de las claves naturales y claves subrogadas. Todos estos elementos serán usados para la identificación de los elementos en la base de datos, lo que es el primer paso para la detección de la ocurrencia de un conflicto para su posterior tratamiento.

Tanto los conflictos básicos como su resolución pueden estar presente en la situación descrita, pero en algunos casos no se ajustan o no son suficientes para resolver el problema, por lo que se han caracterizado los conflictos particulares existentes y la resolución de los mismos se hará usando algunas vías ya propuestas para evitarlos, así como modificaciones de estas que respondan a las necesidades de la solución.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

2.1 Introducción

En el presente capítulo se propondrá la solución correspondiente a la unificación de la información, la estrategia de identificación unívoca de elementos, y el tratamiento a conflictos identificados durante la sincronización de las bases de datos de la Aduana General de la República.

2.2 Integración con el entorno multimaestro-asíncrono. Sincronización de las bases de datos

Como proceso inicial en presencia de la necesidad de propagar los datos de un nodo a otro, y establecer un punto de partida donde se cuenten con los mismos datos iniciales en las bases de datos involucradas, se hace uso de una solución desarrollada por el proyecto SIGEP (Reko), realizando una configuración específica en correspondencia con la topología del entorno que se simula.

Esta configuración se efectuó para poder simular correctamente la situación real, no obstante la solución es independiente del mecanismo de propagación que se use.

El caso de estudio propuesto está compuesto por tres servidores de bases de datos que se identificarán como los nodos de bases de datos distribuidas en la red nacional de aduanas, cumpliendo con el entorno multimaestro reconocido. Estos nodos son los que se observan seguidamente:

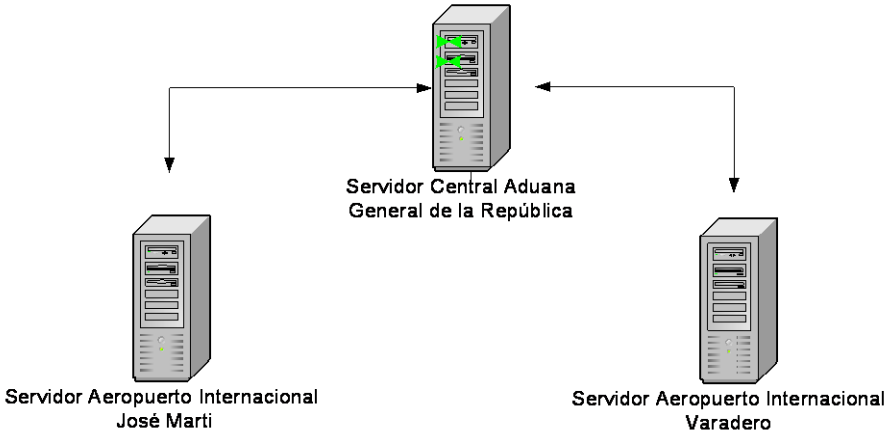

Fig. 5. Esquema de simulación

1. Nodo # 1 : Servidor Central Aduana General de la República
2. Nodo # 2 : Servidor Aeropuerto Internacional José Martí
3. Nodo # 3 : Servidor Aeropuerto Internacional de Varadero

De esta forma la solución estará integrada al entorno multimaestro-asíncrono identificado, ya que se establecerá en cada nodo que conforma el entorno, debido a que en todos ellos puede efectuarse modificaciones de la información y por tanto darse la ocurrencia de conflictos. La solución se introduciría en cada nodo que forma el entorno con la particularidad de que se predefine comportamientos distintos en correspondencia del tipo de nodo done se realiza, si es nodo central o si es nodo secundario.

2.3 Estrategia de identificación unívoca

Para iniciar el tratamiento de conflictos primero debemos identificar elementos como las personas de forma única en el contexto global de la Aduana General de la República y no solo en el contexto del servidor o subsistema donde sean insertadas.

Siguiendo las prácticas generales de identificación de registros en una base de datos, donde los atributos y sus distintas combinaciones permiten identificar la presencia única de un elemento lo primero que se tendrá en cuenta son los atributos que están definidos como parte de las restricciones para cada entidad de la base de datos, reflejadas en claves primarias y únicas.

Estos atributos pueden no pertenecer a la misma tabla de la base de datos, entonces se necesita haya una relación de llave extranjera o foránea entre ambas tablas para que se pueda establecer de forma correcta la restricción de identificación a través de los distintos registros en distintas tablas y sus dependencias.

Se propone por consiguiente aprovechar las facilidades que se derivan al hacer uso de las restricciones de integridad tanto referencial como de identificación brindadas por el Sistema de Gestión de Base de Datos, específicamente a través de claves únicas que representen claves naturales o de negocio y las claves foráneas garantizando que no exista en la base de datos ningún registro que dependa de otro que no se encuentre en la misma, utilizando estas para detectar la existencia en la base de datos de elementos repetidos por cada entidad y dado que ante estas situaciones el Sistema de Gestión de Bases

de Datos, para proteger o garantizar la consistencia de la base de datos, no permite la inserción de un elemento que tenga los mismos valores en los atributos definidos como clave única, produciéndose un error en tiempo de ejecución. Así se filtraría la ocurrencia de este tipo de error en la base de datos, almacenando los datos del registro que produjo dicho error y detectando de esta manera la presencia de un conflicto.

La estrategia general para la identificación unívoca de elementos consta de los siguientes pasos.

- Determinar qué entidades o tablas de la base de datos están involucradas en el proceso de réplica.
- Detectar la ocurrencia de un conflicto de unicidad en las tablas antes determinadas.
- Capturar los datos relacionados con la tupla o registro que provocó el conflicto de violación de restricción de identificación, incluyendo la tabla o entidad a la que pertenece.
- Capturar el nombre de la restricción establecida en la base de datos que fue violentada y los atributos asociados a esta, los cuales conforman la clave única o primaria de la tabla.
- Guardar los datos capturados como metadatos para su posterior utilización.

2.4 Conflictos que se pueden presentar. Clasificación.

Los conflictos de unicidad que tienen lugar en la situación que nos ocupa los podemos agrupar en casos según sus características. Estos son:

Caso 1: Identificador igual, elementos iguales

En este caso se parte de la identificación de los elementos a través de los atributos que puedan definir la existencia única de los mismos, a través de una clave única, siendo clave natural. El conflicto ocurre al poseer dos registros los mismos valores de clave única y además los mismos valores de clave primaria, entiéndase por esta última la clave definida en la tabla a través de una secuencia lo que representa una clave subrogada. Al confirmarse la semejanza de estos registros se procede a comparar la similitud de los restantes campos y determinar las diferencias entre ambos, para efectuar la solución de estos en correspondencia con las diferencias encontradas.

Caso 2: Identificador igual, elementos distintos.

Sucede al poseer los dos registros en conflicto, de distintos valores en la clave única, los mismos valores de clave primaria, lo que representa en nuestra situación una clave subrogada por medio del establecimiento de una secuencia. Al tener el mismo valor de clave primaria se presenta un conflicto. En esta situación se puede resolver creando un nuevo elemento donde se le asigne un nuevo identificador al último elemento insertado.

Caso 3: Identificador distinto, elementos iguales.

En este caso se necesita aplicar la forma de identificación de cada elemento a través de los atributos que puedan determinar la existencia única de esos elementos, a través de una clave única, siendo clave natural. El conflicto ocurre al poseer dos registros los mismos valores de clave única y distintos valores de clave primaria, entiéndase por esta última la clave definida en la tabla a través de una secuencia lo que representa una clave subrogada. Una vez identificados los elementos se puede proceder a comparar la similitud de los restantes campos y determinar las diferencias entre ambos, para efectuar la solución de los mismos en correspondencia con las diferencias encontradas.

2.4.1 Formas de solución de conflictos.

Se considera se debe brindar la resolución de conflictos automática y facilitar las bases para la resolución manual, teniendo en cuenta las ventajas de ambas, asignando la responsabilidad al sistema de ofrecer la solución a todos los conflictos elementales automáticamente sin intervención del usuario mediante el establecimiento de un grupo de reglas estándar , y bridando las facilidades o mecanismos que permitan posteriormente poner en práctica la resolución manual de los conflictos donde, se pueda determinar por cada conflicto cuál será la solución que se le dará, lo que permitirá dar mayor rango de decisión sobre un conflicto concreto.

Para poder aplicar los criterios de resolución de conflictos se elaborarán un conjunto de reglas que permitan dar solución a los conflictos principales.

Anteriormente se ha explicado que los conflictos presentes en la situación se identifican como parte de los conflictos de unicidad, puesto que parten de la necesidad de identificar a un elemento unívocamente y lo

que se relaciona directamente con los términos de violación de las restricciones de integridad, ya sea por clave única o clave foránea.

Para garantizar que no se den conflictos de unicidad clásicos, se implementará una de las soluciones que se proponen para ello. En este caso establecer un rango distinto a las secuencias por cada servidor o nodo.

Rangos para cada nodo:

- Nodo Central Aduana General de la República (id: nodo1)

  Valor de inicio de la secuencia: 1

  Valor de incremento: 5

  Primeros 10 números: 1, 6, 11, 16, 21, 26, 31, 36, 41, 46

- Nodo Aeropuerto Internacional José Martí (id: nodo2)

  Valor de inicio de la secuencia: 2

  Valor de incremento: 5

  Primeros 10 números: 2, 7, 12, 17, 22, 27, 32, 37, 42, 47

- Nodo Aeropuerto Internacional Varadero (id: nodo3)

  Valor de inicio de la secuencia: 3

  Valor de incremento: 5

  Primeros 10 números: 3, 8, 13, 18, 23, 28, 33, 38, 43, 48

Es válido resaltar que este rango de números está limitado para una cantidad menor o igual a 10 nodos, si es necesario ponerlo en práctica en una mayor cantidad de nodos sería necesario cambiar el valor de incremento de la secuencia por 10. De esta forma el rango es mayor.

2.4.2 Criterios de resolución de un conflicto.

De forma general se han identificado criterios de resolución de un conflicto. Estos son.

- Determinar si los dos elementos son el mismo, en ese momento se identifica la ocurrencia de un conflicto.
- Determinar el tipo de conflicto que se presenta
- Conocer el nodo donde ocurre el conflicto
- Establecer comportamientos para cada tipo de nodo: central y secundario.
- Detección de las diferencias de claves primarias de los elementos.
- Conocer la diferencia de todos los atributos referentes a cada elemento.

2.4.3 Reglas

Se considera que las reglas a implementar deben ser las siguientes, cumpliendo con una condición y permitiendo realizar alguna acción o tomar una decisión.

1. Si posee el mismo identificador y no se trata del mismo elemento, asignarle un nuevo identificador al último elemento recibido e insertarlo como un nuevo elemento.
2. Si posee el mismo identificador y tratarse del mismo elemento, verificar las diferencias entre los atributos de ambos elementos.
3. Si son iguales los atributos de esos elementos, se mantiene el elemento local y se desecha el remoto.
4. Si son distintos los atributos, en el caso de los nodos secundarios, se actualizan los pertenecientes al elemento local con las diferencias que poseía el elemento remoto, exceptuando en este proceso los campos nulos del remoto, garantizando de esta forma que siempre se tenga en los nodos secundarios la información identificada por el central como correcta y se mantengan los datos que el nodo central no posee.

5. Si son distintos los atributos, en el caso del servidor central, se actualizan los campos nulos del elemento local con los datos del elemento remoto, garantizando así que los datos del servidor central no sean sobre escritos o eliminados y se tenga presente los datos que el local no posea y el remoto sí.
6. Si posee distinto identificador, y no se trata del mismo elemento, se mantiene el elemento local intacto y se inserta el elemento remoto.
7. Si posee distinto identificador y se trata del mismo elemento, se verifican las diferencias entre los atributos.
8. Si son iguales los atributos de esos elementos, en el caso de los servidores secundarios se inserta el elemento con sus datos dado que son iguales, y con el nuevo identificador procedente del servidor central, se modifican los valores en los campos de clave foránea en las tablas que tienen relación con la afectada para asociar al elemento con nuevo identificador la información que estaba relacionada al anterior. Por último se elimina el elemento que posee el valor de identificador modificado.
9. Si son iguales los atributos de esos elementos, en el caso del servidor central, se mantiene el elemento local y se desecha el remoto.
10. Si son distintos los atributos, en el caso de los servidores o nodos secundarios, se inserta un nuevo elemento con el identificador del elemento remoto y con una mezcla de la información común y las diferencias que poseía este último con respecto al local, exceptuando en este proceso los campos nulos del remoto, garantizando de esta forma que siempre se tenga en los nodos secundarios la información identificada por el central como correcta y se incorporen los datos que el nodo central no posee. Posteriormente se modifican los valores en los campos de clave foránea en las tablas que tienen relación con la afectada para asociar al elemento con nuevo identificador la información que estaba relacionada al anterior. Por último se elimina el elemento que posee el valor de identificador local.
11. Si son distintos los atributos, en el caso del servidor central, se actualizan los campos nulos del elemento local con los datos del elemento remoto, cuando el local posee campos nulos, garantizando así que los datos del servidor central no sean sobre escritos o eliminados y se tenga presente los datos que el local no posea y el remoto sí. Posteriormente se modifican los valores en los campos de clave foránea, en las tablas que tienen relación con la afectada, modificando en

todas las ocurrencias el valor del identificador perteneciente al registro remoto por el valor del identificador del registro local, para asociar así al elemento local la información que estaba relacionada al elemento remoto.

2.5 Unificación de la información referente a un elemento.

Como se demuestra en la definición de las reglas anteriores, por lo general en el caso de que un elemento no sea el correcto o se tenga una versión del mismo que deba ser tenida en cuenta, se actualiza su identificador y con ello los datos que lo caracterizan o se inserta un nuevo elemento con toda la información. Como esos datos son relevantes en la tupla de la tabla a la que pertenece, y en cualquier otro lugar donde se utilicen son enlazados por el identificador de dicha tupla como clave foránea, entonces se necesita primero obtener la jerarquía de las tablas para luego de insertar el elemento con el nuevo identificador , actualizar seguidamente con este nuevo valor de identificador todos los valores en los campos que conformen las claves foráneas de las tablas que tengas relación con ella y que apuntan a esta clave primaria, así toda la información referente al elemento con el anterior identificador quedará relacionada con este pero a través del nuevo identificador. Como la opción **on_update_cascade** no está habilitada para las claves primarias, por lo que no es posible modificar primero el valor de la clave primaria y luego por cascada los de las claves foráneas relacionados a esta, y además no es posible modificar un valor de clave primaria siempre que este tenga asociado dependencias, es necesario actualizar los valores de las claves foráneas que apuntan a una clave primaria partiendo de los niveles más bajos en la jerarquía de las tablas, desde las tablas hijas hasta las tablas padres. Sería necesario entonces hacer un recorrido en árbol de la jerarquía de las tablas y realizar estas modificaciones de forma recursiva.

Ejemplo 1:

Solución de conflicto TIPO 2: elementos en distintos subsistemas. CASO: 3

Nodo Secundario.

Registro de la tabla persona del esquema PERSONA en la base de datos del nodo central:

ID_PERSONA	TRATAMIENTO	PRIMER_NOMBE	SEGUNDO_NOMBRE	PRIMER_APELLIDO	SEGUNDO_APELLIDO	TIPO

1	NULL	Yasser	Antonio	Romero	Pineiro	1

Registro de la tabla persona del esquema PERSONA en la base de datos del nodo secundario:

ID_PERSONA	TRATAMIENTO	PRIMER_NOMBE	SEGUNDO_NOMBRE	PRIMER_APELLIDO	SEGUNDO_APELLIDO	TIPO
10	NULL	Yasser	Antonio	Romero	Pineiro	1

Registro de la tabla lcf_persona_vinculada del esquema LCF en la base de datos del nodo secundario

ID_RESULTADO_CONTROL	ID_PERSONA
5	10

En este caso en el nodo secundario se tiene una información asociada a la persona que posee como ID_PERSONA el valor 10.

Ejecutando lo antes explicado, se detectaría que la persona es la misma y como la tupla que se recibe en el nodo secundario proviene el nodo central se debería actualizar el identificador de la persona, asignado el valor 1 a ID_PERSONA, en lugar de 10 en la tupla del nodo secundario. Pero para poder efectuar esto, es necesario primero insertar el elemento en la tabla PERSONA con el nuevo valor de identificador en el campo ID_PERSONA y luego actualizar este valor en los hijos de la tabla en cuestión. Al realizar esta operación se actualizan todas las ocurrencias de ID_PERSONA, que se encuentra como clave foránea en las tablas que tienen relación con la tabla PERSONA donde estaba el valor 10 , se asigna el valor 1. Así queda la información relacionada a la persona 10, vinculada ahora a la misma persona pero con otro identificador, el 1.

Se evalúa si existe diferencia en los restantes campos y al no suceder se mantienen con los datos intactos.

Estado de las tuplas luego de efectuar la operación:

Registro de la tabla persona del esquema PERSONA en la base de datos del nodo secundario:

ID_PERSONA	TRATAMIENTO	PRIMER_NOMBE	SEGUNDO_NOMBRE	PRIMER_APELLIDO	SEGUNDO_APELLIDO	TIPO

1	NULL	Yasser	Antonio	Romero	Pineiro	1

Registro de la tabla lcf_persona_vinculada en la base de datos del nodo secundario

ID_RESULTADO_CONTROL	ID_PERSONA
5	**1**

Nodo Central

Si esta situación se diera en el nodo central, se mantiene intacto el elemento local, pues solo son diferentes los identificadores y al tener prioridad la información del nodo central se mantiene el identificador de este.

De todas formas es necesario asociar la información que arriba relacionada con el elemento de identificador 10 al elemento con identificador 1. Para ello, la tupla perteneciente al elemento remoto de la tabla lcf_persona_vinculada que hace referencia al elemento persona con identificador 10, al arribar provoca un conflicto de clave foránea pues esa persona no existe en la base de datos, de esa manera se almacena como un conflicto de este tipo, guardando los datos asociados en las tablas temporales. Luego cuando al resolver el conflicto de unicidad donde se actualiza el valor de las claves foráneas con el nuevo valor del identificador en todas las tablas hijas, se actualizan igualmente esos valores en las tablas temporales pertenecientes a la correspondiente tabla hija donde está almacenado el registro conflictivo, modificando así el valor 10 por el 1 en el campo ID_PERSONA. Posteriormente esos registros ya con el identificador del elemento local 1 son recuperados e insertados en las tablas originales a las que pertenecen. Así puede recuperar la información de un elemento que llegue al nodo central, si los identificadores son distintos.

Ejemplo 2:

Solución del conflicto Tipo 1: elementos en el mismo subsistema . Caso 1.

Nodo Secundario

Registro de la tabla persona del esquema PERSONA en la base de datos del nodo central:

ID_PERSONA	TRATAMIENTO	PRIMER_NOMBE	SEGUNDO_NOMBRE	PRIMER_APELLIDO	SEGUNDO_APELLIDO	TIPO
1	null	Yasser	Antonio	Romero	Pineiro	1

Registro de la tabla persona del esquema PERSONA en la base de datos del nodo secundario:

ID_PERSONA	TRATAMIENTO	PRIMER_NOMBE	SEGUNDO_NOMBRE	PRIMER_APELLIDO	SEGUNDO_APELLIDO	TIPO
1	especial	Yasser	Antonio	Romero	Pineiro	2

En este caso en el nodo secundario tiene la misma persona que posee como ID_PERSONA el valor 1.

Ejecutando lo antes explicado, se detectaría que la persona es la misma y como el identificador de la persona es el mismo no se modifica.

Se evalúa la existencia de diferencias entre los valores de los restantes campos, y al detectarse alguna, según la prioridad establecida para el servidor central, se procede a actualizar los campos al elemento local con las diferencias que poseía el elemento remoto, exceptuando en este proceso los campos nulos del remoto, garantizando de esta forma que siempre se tenga en los nodos secundarios la información identificada por el central como correcta y se mantengan los datos que el nodo central no posee.

Estado de las tuplas luego de efectuar la operación:

Registro de la tabla persona del esquema PERSONA en la base de datos del nodo secundario:

ID_PERSONA	TRATAMIENTO	PRIMER_NOMBE	SEGUNDO_NOMBRE	PRIMER_APELLIDO	SEGUNDO_APELLIDO	TIPO
1	especial	Yasser	Antonio	Romero	Pineiro	1

En caso de que lo anterior sucediera en el **nodo central** el resultado sería el mismo pues se actualizarían en la tupla del elemento local, solo los campos que posean valor nulo, garantizando se almacene la información del elemento remoto que el local no posea.

2.6 Funcionalidades

Se pretende implementar las siguientes funcionalidades dentro de un paquete de procedimientos que va a permitir una mejor organización de las mismas.

- Detectar conflicto

Ante la ocurrencia de un conflicto de unicidad, detectarlo y capturar la tupla que lo provocó así como la tabla a la que pertenece. Se realizará en la implementación de un trigger ante la necesidad de capturar un evento del Sistema de Gestión de Base de Datos Oracle 11g.

- Preparar soporte a la solución

Esta funcionalidad ejecutaría todas las funciones de configuración necesarias para dejar listo al sistema para la ejecución del mecanismo de detección y resolución de conflictos.

- Crear tablas temporales

Se necesita crear tablas temporales con la misma estructura de datos de las tablas originales donde ocurre un conflicto, agregándole un campo id_error para almacenar las tuplas que provocaron el conflicto y el identificador del error.

- Crear tabla temporal de conflictos

Es evidente que se debe guardar la información referente a los conflictos ocurridos, para mejor manejo de estos, haciéndose uso de metadatos o datos de los datos, registrándose esencialmente el identificador del conflicto que lo vincula a la tupla que lo produjo, así como otros datos de interés como la tabla donde se sucedió, el nombre y el tipo de la restricción de integridad que se violó, y la sentencia SQL que se ha ejecutado.

- Almacenar conflicto

Se guardan los datos relacionados a los conflictos, así como las tupla que lo produjo, en las estructuras creadas con ese fin.

- Hallar tablas a replicar

Es necesario conocer las tablas que están involucradas en la replicación porque de ellas dependen otras funcionalidades.

- Obtener restricciones

Conociendo las tablas que están involucradas en la replicación, se deben manejar los campos que conforman o pertenecen a las restricciones (constraints) definidas para cada tabla, de tal forma que se tenga por cada tabla las restricciones así como el nombre de los campos que la forman.

- Crear tabla temporal de campos por restricciones.

Para almacenar los datos obtenidos en la funcionalidad anterior

- Obtener registro conflictivo

Se obtiene según el conflicto, la tupla que se quiso insertar y que provocara el conflicto.

- Obtener registro en conflicto

Se obtiene según la tupla que se quiso insertar y que provocara el conflicto, la tupla original que contenía ya al elemento que se quería insertar.

- Clasificar Conflicto

Dado un conflicto determinado, poder clasificarlo en correspondencia con las características de la tuplas en conflicto y los casos que pueden ocurrir.

- Resolver conflictos

Teniendo un conflicto, su clasificación, el nodo donde se presenta y las tuplas en conflicto darle solución al mismo.

- Tratar conflictos

Mecanismo que con una frecuencia determinada de tiempo activaría las funciones de resolución de conflictos.

Estas funcionalidades serán implementadas combinando procedimientos almacenados, funciones y triggers en PLSQL.

2.7 Conclusiones del Capítulo

La solución se propone para un caso de estudio con las características del entorno de replicación propio de la situación. La misma está concebida en tres elementos principales: la forma en que los datos serán propagados a través de la red, la estrategia de identificación unívoca de elementos en el ambiente de una base de datos distribuida ligado a la identificación de la ocurrencia de un conflicto y la resolución de estos en correspondencia con el tipo identificado y el nodo donde tiene lugar: nodo central, nodo secundario. Se implementarán las funcionalidades identificadas en una combinación de procedimientos, funciones en lenguaje PL/SQL de forma genérica y dinámica que brinde mayor flexibilidad y adaptación de la solución a futuros cambios en la base de datos.

Para la propagación de datos se propone asimilar el software Reko, desarrollado por el proyecto SIGEP, estableciendo las configuraciones necesarias para adaptarlo a las necesidades de la situación, aunque la forma en la que los datos sean transmitidos es independiente de la propuesta de solución y podría ser asimilada cualquier otra que por sus características se decida.

Referente a la identificación unívoca de elementos se propone hacer uso de una combinación de atributos estableciendo claves primarias y únicas que permitan alcanzar el objetivo de identificar correctamente un elemento único en la base de datos.

La resolución de conflictos se propone establecerla de forma automática para escenarios determinados a través de la definición de reglas que identifiquen el tipo de conflicto y la acción en cada caso. Para cumplir con ese propósito se ha fijado la preferencia de la información del nodo central, ya sea al recibirla o enviarla.

La solución se concibe como un mecanismo automático de detección y resolución de conflictos.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS

3.1 Introducción

En el presente capítulo se presenta la solución práctica y validación del mecanismo de detección y resolución de conflictos que ocurren durante la sincronización de las bases de datos de la Aduana General de la República.

3.2 Preparando condiciones para el funcionamiento del mecanismo.

Para el correcto funcionamiento de la propuesta que se implementó se deben garantizar varios aspectos que incluyen una adecuada configuración y el establecimiento de las estructuras necesarias sobre las cuales el mecanismo podrá trabajar.

3.2.1 Simulación del caso de estudio

El mecanismo debe funcionar en un ambiente distribuido de base de datos, como lo requieren las condiciones de despliegue del esquema de la red nacional de aduanas como se observa en el esquema mostrado en la introducción de este trabajo.

El entorno de réplica se ha identificado como Multimaestro-Asíncrono donde la propagación de los datos ocurre en ambas direcciones, tanto desde el nodo maestro hasta los esclavos como desde estos al primero, modificándose la información en todos los nodos, de tal forma que funcionan todos como maestros de datos, con el característica especial de que se establece un maestro central que sirve como puente para la transferencia de datos entre los demás nodos, y permitiendo así que los otros nodos maestros se comuniquen solo con este y se efectúe de forma correcta el tratamiento de la información.

Por lo antes explicado se diseña una simulación del caso de estudio de tal forma que cuente con las características del entorno seleccionado así como las de la situación real planteada en la situación problemática.

El esquema de este caso de estudio está conformado por tres nodos como se muestra en el esquema de simulación propuesto para efectuar la sincronización.

Para el establecimiento de la simulación del caso de estudio se instalaron tres servidores, dotándolos con Sistema Operativo Ubuntu GNU/Linux 9.04, además de Sistema de Gestión de Base de Datos (SGBD) Oracle 11g.

Teniendo en cuenta las características del entorno y la propuesta de solución, se implementa la solución en función de dos tipos de nodos, según sus responsabilidades e importancia en el esquema simulado. Estos tipos de nodos son: **nodo central, nodo secundario**. Como nodo central se tiene la AGR y como nodos secundarios el Internacional José Martí y el Internacional de Varadero.

El mecanismo de detección y resolución de conflictos tendrá distinto comportamiento para estos dos tipos de nodo puesto que se establece en él, hasta el momento, prioridad para la información que se encuentra en el nodo central.

3.2.3 Punto de partida del mecanismo.

El mecanismo se diseña como una solución automática que se activa luego de arribar datos replicados al nodo en cuestión. Esto ocurre independientemente del mecanismo de propagación de datos que se use. En este caso sucede luego del proceso de aplicación de los cambios ejecutado por Reko, la solución de réplica que se usa para poder propagar los datos y simular correctamente la situación.

Para garantizar el funcionamiento del mecanismo es necesario se tenga una tabla de control donde se especifiquen las tablas que estarán bajo este proceso de réplica. Es importante que el nombre de estas tablas tenga incluido el nombre del esquema de tal forma que esté compuesto por [nombre_esquema.nombre_tabla], para de esta forma garantizar se conozca la ubicación exacta de la tabla y no haya confusiones si existe esta en varios esquemas. Además se podrá acceder indistintamente a la información de un elemento que se encuentre en varios subsistemas. Así esta información es aprovechada por el mecanismo con el objetivo de alcanzar el mayor dinamismo y transparencia necesarios. Se debe especificar el nombre de esta tabla de control que contenga la información.

3.2.4 Estructuras necesarias para el funcionamiento del mecanismo.

Primeramente se crean tablas temporales correspondientes a cada una de las tablas sobre las cuales el mecanismo trabajará en la detección de los conflictos. Estas tablas poseen la misma estructura que las tablas originales con el objetivo de poder almacenar en ellas el mismo tipo de información que estas. Además se le agrega una columna *id_error* donde se almacenará el valor de una secuencia que identificará de manera única un conflicto en la base de datos.

EL nombre de estas tablas seguirá el siguiente estándar:

Estándar de nombre de tablas temporales:

Nombre de las tablas: [NOMBRE_TABLA]_T

Con el objetivo de efectuar la creación de estas tablas temporales de forma dinámica se creó un procedimiento almacenado nombrado ***tables_to_replicate***. Este procedimiento es ejecutado cada vez sea llamado el mecanismo pues es posible que entre una ejecución y otra del mismo se haya agregado otra tabla como objeto de replicación y por tanto el mecanismo debe incorporarla a sus objetos de chequeo.

Implementación del procedimiento. Ver Anexo # 1

3.2.4.1 Tablas de Control. Metadatos.

Cada una de las siguientes tablas hace posible el funcionamiento del mecanismo, pues en ellas se almacenan de forma automática datos y metadatos (datos de los datos) imprescindibles y críticos para realizar la tarea de detección y posterior resolución de los conflictos.

Estas constituyen las fuentes o destinos de muchas funcionalidades que se encuentran enlazadas dentro del mecanismo, las cuales serán enunciadas más adelante en este capítulo.

Estas tablas son creadas por el procedimiento ***create_tables***. Ver Anexo # 2

Las tablas son las siguientes:

Cuadro 1: Descripción tabla *ORAERROR*

Nombre de la tabla: ORAERROR		
Descripción	Almacena el identificador de un conflicto así como datos necesarios asociados a él	
Atributo	**Tipo**	**Descripción**
id_error	number(20)	Identificador del conflicto
table_name	varchar2(40)	Nombre de la tabla donde ocurre el conflicto
log_date	timestamp(6)	Momento en que sucede el conflicto
log_usr	varchar2(30)	Usuario que realiza la operación
error_n	number(10)	Número de error según Oracle
error_msg	varchar2(100)	Mensaje del error
sql_text	varchar2(100)	SQL de la operación que se realizó.
constraint_voliated	Varchar2(40)	Nombre de la restricción violada

Cuadro 2: Descripción tabla *LOG_GENERIC_CONFLICT*

Nombre de la tabla: LOG_GENERIC_CONFLICT		
Descripción	Almacena las tuplas remotas que han provocado un conflicto	
Atributo	**Tipo**	**Descripción**
id_conflict	number(20)	Identificador del conflicto
conflict_record	xmltype	Registro remoto que provocó el conflicto

Cuadro 3: Descripción tabla *ORIGINAL_RECORD*

Nombre de la tabla: ORIGINAL RECORD		
Descripción	Almacena las tuplas locales que han provocado un conflicto	
Atributo	**Tipo**	**Descripción**
id_conflict	number(20)	Identificador del conflicto
original_record	xmltype	Registro local que provocó el conflicto

Cuadro 4: Descripción tabla *TABLE_PARENTS*

Nombre de la tabla: TABLE_PARENTS		
Descripción	Almacena la relación de todas las tablas hijas, presentes en el esquema y relacionadas con el mecanismo, con sus respectivas tablas padres.	
Atributo	**Tipo**	**Descripción**
nombre_tabla	varchar2(40)	Nombre de una tabla hija
tabla_padre	varchar2(40)	Nombre de la tabla padre

Cuadro 5: Descripción tabla *CONSTRAINTS_COL_TEMP*

Nombre de la tabla: CONSTRAINTS_COL_TEMP		
Descripción	Almacena el nombre de las columnas que conforman cada una de las restricciones o claves de una tabla, así como el tipo de restricción: primaria, única, foránea.	
Atributo	**Tipo**	**Descripción**
campo	varchar2(40)	Nombre de la columna
nombre_constraint	varchar2(3)	Nombre de la restricción
tipo_constraint	varchar2(40)	Tipo de restricción (P,U,R)
nombre_tabla	varchar2(40)	Nombre de la tabla

3.3 Descripción del mecanismo.

A continuación se detallará la concepción de mecanismo, sus componentes, y cómo da respuesta a las necesidades planteadas en la investigación.

3.3.1 Estructura del mecanismo

El mecanismo está concebido en tres componentes esenciales. El primero de ellos es un trigger que se encarga de chequear las tablas involucradas ante la ocurrencia de un conflicto y detectarlo, además de capturar y almacenar los datos necesarios en las estructuras creadas para ese fin. El segundo componente es un paquete de funciones donde se encuentran la mayoría de las funciones y procedimientos relacionados con el mecanismo y su funcionalidad de resolver los conflictos detectados. El tercer componente es una tarea programada o job el cual se encarga de activar la función principal que ejecuta a su vez los procedimientos necesarios a través la resolución automática de los conflictos.

3.3.2 Detección, captura y almacenamiento de un conflicto

El funcionamiento del mecanismo comienza al ser capaz de detectar la ocurrencia de un conflicto. Para poder hacerlo se determinó que la vía correcta, y como estrategia de identificación unívoca de los distintos elementos en la base datos, es aprovechar y hacer un adecuado uso de las restricciones de integridad, fundamentalmente de identificación que radican en el establecimiento de claves primarias y únicas. Así al detectar la violación de una clave única o primaria se está en presencia de un conflicto de unicidad. Para esto las restricciones en la base de datos deben estar adecuadamente establecidas, pues se necesita además que estas sean fundamentalmente claves naturales y no subrogadas pues las segundas no identifican al elemento en un contexto global o en un ambiente de base de datos distribuidas como es el caso.

Al ocurrir una violación de integridad Oracle lanza el error conocido por ora-00001 o 1. Por eso la detección, captura y almacenamiento de un conflicto de este tipo es realizado por un trigger nombrado *AFTER_ERROR* y que se activa al ocurrir un error en la base de datos filtrando el tipo de error de unicidad. En él se chequean solo las tablas que se relacionan con el proceso de réplica.

Cuadro 6: Descripción trigger *AFTER_ERROR*

Nombre del trigger : `AFTER ERROR`		
Descripción	Detección, captura y almacenamiento de un conflicto de unicidad.	
Momento	**EVENTO**	**SOBRE OBJETO**

after	servererror	on database

El resultado del trigger almacena los datos de la tupla remota que provocó el error en la tabla temporal asociada a la tabla original donde ocurrió el conflicto, además guarda el identificador del conflicto como id_error. Al mismo tiempo almacena información importante del conflicto en la tabla ***ORAERROR*** como el identificador , tabla donde sucede, momento, usuario, numero de error según la clasificación de Oracle, el mensaje de error, la sentencia sql que se ejecutó y el nombre de la clave o restricción que se violó. Estos datos se consultan posteriormente.

Implementación del trigger. Anexo # 3.

3.3.3 Resolución de un conflicto

La resolución de conflictos que se implementa es un mecanismo automático y dinámico en PL/SQL que se ejecuta cada 15 minutos en la base datos. Esto es posible mediante una tarea programada o job que ejecuta el procedimiento principal del mecanismo para resolver los conflictos anteriormente detectados y almacenados. Este procedimiento principal es nombrado ***handle_conflict*** y es el encargado de manejar todo lo relacionado con la resolución de los conflictos. En su interior son ejecutados a su vez los distintos procedimientos que conforman el paquete ***util_package*** *(anexo # 4)* en el orden en el que son necesarios, teniendo como procedimiento esencial el ***solve_conflict,*** donde se resuelven los conflictos.

Conforman el paquete ***util_package*** los siguientes procedimientos:

Cuadro 7: Descripción funciones y procedimientos del paquete util_package.

Nombre del procedimiento	**Descripción**
find_constraints_column_proc	Obtiene el nombre de las claves y su tipo(R, P, U) por cada tabla, los campos que pertenecen a cada una de las claves y el nombre de la tabla a la que están asociadas.
get_conflicts_x_table	Obtiene los conflictos sucedidos en una tabla determinada.
get_error_record	Obtiene el registro remoto que provocó un conflicto dado
get_all_columns	Obtiene el nombre las columnas pertenecientes a una tabla

	específica
get_conflict_orignal_record	Obtiene el registro local que provocó un conflicto dado
get_constraint_table	Dado el nombre de una clave obtiene los campos que la forman.
solve_conflict	Dado un conflicto, y los registros involucrados resuelve el conflicto.
recursive_cascade	Actualiza recursivamente las ocurrencias de un valor de clave primaria en las claves foráneas de las tablas hijas.

El procedimiento ***handle_conflict*** selecciona todos los conflictos ocurridos luego de su última ejecución, resolviendo cada uno de ellos, no sin antes clasificarlos y tener en cuenta el tipo de nodo en el que es ejecutado (central o secundario).

Los primeros procedimientos que son ejecutados son los que garantizan se almacenen los datos necesarios en las tablas de control creadas, estos serán ejecutados cada vez sea llamado el procedimiento principal, porque los datos y metadatos que se manejan pueden cambiar en el intervalo de las distintas ejecuciones del mismo.

Para llenar la tabla ***CONSTRAINTS_COL_TEMP***, que almacena la información referente a todos los campos que pertenecen a las distintas claves definidas en las tablas que se involucran en el mecanismo, se utiliza la función ***find_constraits_column***. Anexo # 5. Es necesario especificar el nombre de la tabla de control que posee las tablas que se tendrán en cuenta en el proceso de replica y por tanto en el mecanismo.

Para llenar la tabla ***TABLE_PARENTS***, que almacena la relación de las tablas involucradas en el mecanismo y su correspondiente padre incluyendo las columnas hija y padre respectivamente , para de esta manera posteriormente obtener la jerarquía de tablas que es necesaria para dar solución a un conflicto mediante el procedimiento ***solve_conflict*** del paquete ***util_package*** , se utilizan los procedimientos ***find_parents*** (anexo # 6), ***find_parents_aux*** (anexo # 7) ***y exist_table_name*** (anexo # 8).

3.4 Validación de la solución. Modelo de prueba

Se efectuaron 15 casos de prueba .Seguidamente se describen los principales casos de prueba de integración realizados a nivel de procedimientos para validar el funcionamiento correcto de las funciones del mecanismo.

Cuadro 8: Descripción caso de prueba **# 1**

Caso de Prueba # 1
Nombre Funcionalidad : AFTER_ERROR
Descripción: Prueba para la funcionalidad de capturar la ocurrencia de un conflicto.
Condiciones de Ejecución El elemento debe existir en la base de datos. La tabla del elemento local debe poseer clave única.
Entradas/Pasos de Ejecución: Se intenta insertar un elemento con datos válidos, con valor de clave única igual al almacenado y tipos de datos correctos para las columnas de la respectiva tabla local.
Resultado Esperado: Los tipos de datos del elemento son verificados antes de insertarlo en el las tablas temporales. Es notificado si no se realizó correctamente la inserción. Se almacena el conflicto.
Evaluación de la Prueba: Satisfactoria

Cuadro 9: Descripción caso de prueba **# 2**

Caso de Prueba # 2
Nombre Funcionalidad : find_constraints_col_temp
Descripción: Prueba para la funcionalidad de hallar las claves de las tablas y sus datos asociados.
Condiciones de Ejecución La tabla de control donde están las tablas relacionadas al mecanismo, cuyo nombre es entrado por parámetro, debe existir. Las tablas relacionadas en la tabla de control antes mencionada deben existir.

Entradas/Pasos de Ejecución: Se intenta introducir un nombre de la tabla de control válido. La tabla control y las tablas relacionadas en ella existen.
Resultado Esperado: Son insertados en la tabla ***CONSTRAINT_COL_TEMP*** las claves pertenecientes a cada tabla relacionada, así como los campos que forman cada una de ellas y otros datos necesarios.

Cuadro 10: Descripción caso de prueba **# 3**

Caso de Prueba # 3
Nombre Funcionalidad : get_conflict_original_record
Descripción: Prueba para la funcionalidad de hallar genéricamente el registro local conflictivo.
Condiciones de Ejecución El registro conflictivo remoto debe existir en la tabla temporal para ello. El registro conflictivo local bebe existir en la tabla original. Debe estar creada la tabla temporal original_record para almacenar la tupla genérica en un XML.
Entradas/Pasos de Ejecución: Se introduce un identificador válido del conflicto. El conflicto existe.
Resultado Esperado: Se obtiene la tupla local conflictiva asociada al registro remoto que provoco el conflicto, todo ello de forma genérica.
Evaluación de la Prueba: Satisfactoria

Cuadro 11: Descripción caso de prueba **# 4**

Caso de Prueba # 4
Nombre Funcionalidad : get_error_record
Descripción: Prueba para la funcionalidad de hallar genéricamente el registro remoto conflictivo.
Condiciones de Ejecución. El registro conflictivo remoto debe existir en la tabla temporal para ello.

Debe estar creada la tabla temporal log_generic_conflict para almacenar la tupla genérica en un XML.
Entradas/Pasos de Ejecución: Se introduce un identificador válido del conflicto. El conflicto existe.
Resultado Esperado: Se obtiene el registro remoto conflictivo, de forma genérica.
Evaluación de la Prueba: Satisfactoria

Cuadro 12: Descripción caso de prueba # **5**

Caso de Prueba # 5
Nombre Funcionalidad : solve_conflict
Descripción: Prueba para la funcionalidad de resolver un conflicto caso 1 (Identificador igual, elementos iguales), nodo central.
Condiciones de Ejecución. Se deben tener tanto el registro conflictivo remoto como el registro conflictivo local. El conflicto a solucionar debe existir en la tabla ORAERROR, así como los datos asociados a este en las estructuras auxiliares.
Entradas/Pasos de Ejecución: Se introduce valores válidos del conflicto. El conflicto existe. Se introduce el número que indica en qué nodo se resuelve el conflicto (1).
Resultado Esperado: El conflicto es resuelto.
Evaluación de la Prueba: Satisfactoria

Cuadro 13: Descripción caso de prueba # **6**

Caso de Prueba # 6
Nombre Funcionalidad : solve_conflict
Descripción: Prueba para la funcionalidad de resolver un conflicto caso 1 (Identificador igual, elementos iguales), nodo secundario.

Condiciones de Ejecución. Se deben tener tanto el registro conflictivo remoto como el registro conflictivo local. El conflicto a solucionar debe existir en la tabla ORAERROR, así como los datos asociados a este en las estructuras auxiliares.
Entradas/Pasos de Ejecución: Se introduce valores válidos del conflicto. El conflicto existe. Se introduce el número que indica en qué nodo se resuelve el conflicto (2).
Resultado Esperado: El conflicto es resuelto.
Evaluación de la Prueba: Satisfactoria

Cuadro 14: Descripción caso de prueba **# 7**

Caso de Prueba # 7
Nombre Funcionalidad : solve_conflict
Descripción: Prueba para la funcionalidad de resolver un conflicto caso 3 (Identificador distinto, elementos iguales), nodo central.
Condiciones de Ejecución. Se deben tener tanto el registro conflictivo remoto como el registro conflictivo local. El conflicto a solucionar debe existir en la tabla ORAERROR, así como los datos asociados a este en las estructuras auxiliares.
Entradas/Pasos de Ejecución: Se introduce valores válidos del conflicto. El conflicto existe. Se introduce el número que indica en qué nodo se resuelve el conflicto (1).
Resultado Esperado: El conflicto es resuelto.
Evaluación de la Prueba: Satisfactoria

Caso de Prueba # 8
Nombre Funcionalidad : solve_conflict

Descripción: Prueba para la funcionalidad de resolver un conflicto caso 3 (Identificador distinto, elementos iguales), nodo secundario.
Condiciones de Ejecución. Se deben tener tanto el registro conflictivo remoto como el registro conflictivo local. El conflicto a solucionar debe existir en la tabla ORAERROR, así como los datos asociados a este en las estructuras auxiliares.
Entradas/Pasos de Ejecución: Se introduce valores válidos del conflicto. El conflicto existe. Se introduce el número que indica en qué nodo se resuelve el conflicto (2).
Resultado Esperado: El conflicto es resuelto.
Evaluación de la Prueba: Satisfactoria

Cuadro 15: Descripción caso de prueba # **8**

Durante la realización de las pruebas se detectaron un total de 17 errores, lo que demuestra la efectividad de las pruebas, pues su objetivo y efectividad radica en encontrar la mayor cantidad de errores posibles. Dichos errores fueron solucionados concluyendo de esta forma las pruebas de manera satisfactoria.

3.5 Conclusiones del Capítulo

Mediante este capítulo se ha explicado y documentado la solución práctica del mecanismo de detección y resolución de conflictos en un ambiente que simula el caso de estudio de la red nacional de aduanas. Se mostrado la estructura y funcionamiento del mismo. Además se validó la solución desarrollada demostrando su funcionalidad y el cumplimiento de los objetivos trazados.

CONCLUSIONES

Mediante la realización del presente trabajo ha sido posible desarrollar la solución de conflictos en la información registrada para el mismo elemento en diferentes servidores, durante la sincronización de las bases de datos para el Sistema de Gestión de Bases de Datos (SGBD) Oracle 11g. El logro de estos resultados fue posible gracias a que:

- Fue detectada la ocurrencia de conflictos que tienen lugar en la situación problemática.
- Se identificaron los tipos de conflictos que pueden suceder en el entorno descrito.
- Se definió la estrategia de identificación unívoca de elementos.
- Se pudo obtener el estado del arte de las soluciones de resolución de conflictos y sincronización para el Sistema de Gestión de Bases de Datos (SGBD) Oracle.
- Fue elaborada la propuesta de solución con su correspondiente justificación técnica para Oracle 11g.
- Se propuso la forma de integración de la solución dentro del entorno de sincronización de datos identificada para la Aduana General de la República de Cuba.
- Se alcanzó implementar la solución propuesta para Oracle 11g en un caso de estudio que simula la situación actual de despliegue de la Aduana General de la República de Cuba, ajustándose al tiempo y recursos planificados.

RECOMENDACIONES

Ya concluido el presente trabajo y haber alcanzado sus objetivos se recomienda:

- Estudiar la posible ampliación de escenarios para la resolución automática de los conflictos.
- Aprovechar los resultados del presente trabajo para brindar una solución manual de conflictos, cubriendo la mayor cantidad de escenarios posibles.
- Determinar la ocurrencia y la posible solución a otros tipos de conflictos como los de actualización, eliminación y orden.
- Desplegar el mecanismo desarrollado en un ambiente real de la Aduana General de la República.

BIBLIOGRAFÍA

Referenciada

1. Bases de datos distribuidas y fragmentación. Departamento de O.I.E -U.P.M. 2009, [Consultado el: 10 de marzo 2010]. Disponible en: http://www.oei.eui.upm.es/Asignaturas/BD/DYOBD/DISTRIBUIDAS.pdf

2. Tipos de Fragmentación de Datos. Departamento de Computación de CINVESTAV. 2009, [Consultado el: 20 febrero 2010]. Disponible en: http://www.cs.cinvestav.mx/SC/prof_personal/adiaz/Disdb/Cap_3.html

3. Bases de datos distribuidas. 2008 [Consultado el: 5 de marzo 2010]. Disponible en: http://cmapspublic.ihmc.us/rid=1161027353218_44637313_464/2.pdf

4. OLARTE, C. A. Bases de Datos Distribuidas. 2009, [Consultado el: 1 de marzo 2010]. Disponible en: http://atlas.puj.edu.co/~caolarte/puj/cursos/cc100/files/clases/BDDistribuidas.pdf

5. OCAÑA, A. M. Replicación de Bases de Datos [Página Web]. [Consultado el: 05/02 de 2010]. Disponible en: http://lfc.uah.es/bda/BDA%20Replicacion.pdf

6. Advanced Replication 11g. Oracle Database. 2008, [Consultado el: 15 de marzo 2010]. Disponible en: http://www.filibeto.org/sun/lib/nonsun/oracle/11.1.0.6.0/B28359_01/server.111/b28326.pdf

7. ORACLE. Advanced Replication Management API Reference 11g. 2008, [Consultado el: 15 de marzo 2010]. Disponible en: http://www.filibeto.org/sun/lib/nonsun/oracle/11.1.0.6.0/B28359_01/server.111/b28327.pdf

8. Sistemas de Gestión de Bases de Datos. de 2010]. Disponible en: http://www.mailxmail.com/curso-procesamiento-datos-oracle/sistema-manejador-base-datos.itada

Consultada

Ocaña, A.M. *Replicación de Bases de Datos*. [Consultado 5 de Febrero 2010]; Disponible en: http://lfc.uah.es/bda/BDA%20Replicacion.pdf

Bases de Datos Distribuidas. [Consultado 28 de enero 2010]; Disponible en: http://html.rincondelvago.com/bases-de-datos-distribuidas_1.html.

Documentación Oficial de Oracle. (2008) "Advanced Replication" [Consultado 06 de febrero 2010]; Disponible en: http://www.oracle.com/technology/documentation/database.html

OLARTE, C. A. Bases de Datos Distribuidas. 2009, [Consultado el: 1 de marzo 2010]. Disponible en: http://atlas.puj.edu.co/~caolarte/puj/cursos/cc100/files/clases/BDDistribuidas.pdf

Urbano, R. *Data Replication and Integration Guide*. 2008. 246 p.

MCLAUGHLIN, M. *Oracle Database 11g. PL/SQL Programming.* 2009. 866 p.

ORACLE. *Oracle XML DB Developer Guide.* 2008. 852 p.

Lizama Mué, Yadira; Concepción Milanés, Mario Michel. *Disponibilidad y accesibilidad a datos en la plataforma Génesis*. Ciudad de la Habana. Universidad de las Ciencias Informáticas. 2009

León López, José Luis; Mesa Rodríguez, Fernando. *Introducción de la tecnología Streams del SGBDR Oracle 11g, para garantizar la explotación del sistema de Circulados Nacionales*. Ciudad de la Habana. Universidad de las Ciencias Informáticas. 2009.

Dieguez Sánchez, Roberto Carlos; Fernández Banguela, Sergio. *Sistema de sincronización y gestión de nodos aislados para la Replicación de bases de datos en PostgreSql.* Ciudad de la Habana. Universidad de las Ciencias Informáticas. 2008.

ANEXOS

Anexo 1: Implementación procedimiento tables_to_replicate

```
CREATE OR REPLACE PROCEDURE tables_to_replicate
IS
/*Declarar variables*/
  str_string   VARCHAR2 (60);
  str_tbl_name  VARCHAR2 (30);
  pos      NUMBER (3);

  CURSOR micursor
  IS
   SELECT table_name
    FROM rep_control_name_table;
BEGIN OPEN micursor;

  LOOP
   FETCH micursor INTO str_string;

   EXIT WHEN micursor%NOTFOUND;

   SELECT NVL (INSTR (str_string, '.'), 0) INTO pos FROM DUAL;

   SELECT NVL (SUBSTR (str_string, pos + 1), 0) INTO str_tbl_name FROM DUAL;

   EXECUTE IMMEDIATE 'drop TABLE '||str_tbl_name||'_T' ;

   EXECUTE IMMEDIATE  'CREATE TABLE ' || str_tbl_name || '_T as select * from'
            || str_tbl_name;

   EXECUTE IMMEDIATE ' TRUNCATE TABLE ' || str_tbl_name || '_T drop storage';

   EXECUTE IMMEDIATE  ' ALTER TABLE '
            || str_tbl_name|| '_T ADD (ID_ERROR NUMBER (20))';

  END LOOP;

  CLOSE micursor;

END tables_to_replicate;
```

Anexo 2: Implementación procedimiento create_tables

```
CREATE OR REPLACE PROCEDURE create_tables
```

```
IS
/*Declarar variables*/
  pos  NUMBER (3);
BEGIN
  EXECUTE IMMEDIATE 'CREATE TABLE ORAERROR
    ( id_error number(20),
     table_name varchar2(40),
     log_date timestmap(6),
     log_usr varchar2(32),
     error_n number(10),
     error_msg varchar2(100),
     sql_text varchar2(100),
     constraint_violated varchar2(40)


     ) ';

  EXECUTE IMMEDIATE 'CREATE TABLE CONSTRAINTS_COL_TEMP
    (
     campo  varchar2(40),
     nombre_constraint  varchar2(3),
     tipo_constraint    varchar2(40),
     nombre_tabla varchar2(40)

     ) ';

  EXECUTE IMMEDIATE 'CREATE TABLE LOG_GENERIC_CONFLICT
    (
    id_conflict number(20),
    conflict_record xmltype

     ) ';

  EXECUTE IMMEDIATE 'CREATE TABLE ORIGINAL_RECORD
    (
    id_conflict number(20),
    original_record xmltype

     ) ';

  EXECUTE IMMEDIATE 'CREATE TABLE TABLE_PARENTS
    (
    nombre_tabla varchar2(60),
    tabla_padre varchar2(60)

     ) ';
END create_tables;
```

Anexo 3: Implementación trigger after_error

```
CREATE OR REPLACE TRIGGER after_error
  AFTER SERVERERROR ON DATABASE
DECLARE
  pos          NUMBER (3);
  pos1         NUMBER (3);
  pos2         NUMBER (3);
  PRAGMA AUTONOMOUS_TRANSACTION;
  ID          NUMBER (20);
  sql_text       ora_name_list_t;
  v_stmt        CLOB;
  str          VARCHAR2 (100);
  str_tbl_name_all  VARCHAR2 (64);
  str_tbl_name     VARCHAR2 (64);
  col          VARCHAR2 (100);
  param         VARCHAR2 (100);
  err          NUMBER (10, 0);
  n           NUMBER;
  wend         VARCHAR2 (20);
  flag         NUMBER (3);
  id_last_row      ROWID;
  parents        NUMBER (3);
  vsparents       NUMBER (3);
  constraint violat  VARCHAR2 (60);
  point_constraint  VARCHAR2 (3);
BEGIN
  SELECT oraerror_seq.NEXTVAL
   INTO ID
   FROM DUAL;

 --
-- pos2  := 0 ;
  wend := 'before'; n := ora_sql_txt (sql_text);
  --
  IF n >= 1
  THEN
   FOR i IN 1 .. n
   LOOP
     v_stmt := v_stmt || sql_text (i);
   END LOOP;
  END IF;

  --
  str := DBMS_LOB.SUBSTR (v_stmt, 1000, 1);

  SELECT INSTR (str, 'into')
   INTO pos                      -- donde inicia el into
   FROM DUAL;
```

```
  SELECT NVL (INSTR (str, 'values'), 0)
   INTO pos1                    -- donde comienza el value
   FROM DUAL;

  SELECT NVL (SUBSTR (str, pos + 4, pos1 - pos - 4), 0)
   INTO str_tbl_name_all
-- obtener el nombre de la tabla incluido los campos donde se insertaran si los tiene
el sql
   FROM DUAL;

  SELECT NVL (INSTR (str_tbl_name_all, '('), 0)
   INTO pos2              -- posiciÃ³n donde comienzan los campos
   FROM DUAL;

  str_tbl_name := str_tbl_name_all;
             --asignando para depurara solo el nombre de la tabla
  IF pos2 <> 0
  THEN
   SELECT NVL (SUBSTR (str_tbl_name_all, 0, pos2 - 1), 0)
    INTO str_tbl_name
    FROM DUAL;

   SELECT NVL (SUBSTR (str_tbl_name_all, pos2), 0)
    INTO col
    FROM DUAL;
  END IF;

  col := REPLACE (col, '(', ' ');
  col := REPLACE (col, ')', ' ');

  SELECT NVL (SUBSTR (str, pos1 + 6), 0)
   INTO param
   FROM DUAL;

  param := REPLACE (param, '(', ' ');
  param := REPLACE (param, ')', ' ');

  FOR n IN 1 .. ora_server_error_depth
  LOOP
   --
   IF ora_server_error (n) IN
      ('1')
   THEN
     err := ora_server_error (n);
     str_tbl_name := REPLACE (str_tbl_name, ' ', '');
               -- quitar los espacios dentro de nombre de tabla

     SELECT INSTR (str_tbl_name, 'mia')
      INTO flag                 -- donde cominza el into
```

```
 FROM DUAL;

IF /*col is not null or*/ pos2 <> 0
THEN
 DBMS_OUTPUT.put_line ('col no es null');

 EXECUTE IMMEDIATE  'insert into '
          || str_tbl_name
          || '_T ( '
          || col
          || ', id_error ) values ('
          || param
          || ', '
          || ID
          || ')';

 DBMS_OUTPUT.put_line ('salio de eso ' || str_tbl_name || '-'
            || col
            );
END IF;

IF /*col is null or */ pos2 = 0
THEN
 DBMS_OUTPUT.put_line ('col es null' || flag);
 DBMS_OUTPUT.put_line (  'entro '
            || str_tbl_name
            || '-'
            || col
            || '-'
            || param
            );

 EXECUTE IMMEDIATE  'insert into '
          || str_tbl_name
          || '_T values ( '
          || param
          || ', '
          || ID
          || ')';

 DBMS_OUTPUT.put_line (  'salio de eso '
            || str_tbl_name
            || '-'
            || col
            || '-'
            || param
            );
END IF;
```

```
      EXECUTE IMMEDIATE  ' select max(rowid) from '
               || str_tbl_name
               || '_T '
            INTO id_last_row;

      SELECT INSTR (ora_server_error_msg (n), '(')
       INTO parents                -- donde cominza el into
       FROM DUAL;

      SELECT INSTR (ora_server_error_msg (n), ')')
       INTO vsparents               -- donde cominza el into
       FROM DUAL;

      SELECT NVL (SUBSTR (ora_server_error_msg (n),
                 parents + 1,
                 vsparents - parents - 1
                ),
            0
           )
       INTO constraint_violat
       FROM DUAL;

      SELECT INSTR (constraint_violat, '.')
       INTO point_constraint            -- donde inicia el into
       FROM DUAL;

      SELECT NVL (SUBSTR (constraint_violat, point_constraint + 1), 0)
       INTO constraint_violat
       FROM DUAL;

      DBMS_OUTPUT.put_line ('paso por aqui ');
      str_tbl_name := UPPER (str_tbl_name);

      INSERT INTO oraerror
        VALUES (ID, str_tbl_name, CURRENT_TIMESTAMP, ora_login_user,
            ora_client_ip_address, err, ora_server_error_msg (n),
            wend, id_last_row, str, constraint_violat);

      COMMIT;
    END IF;
  END LOOP;
--
END;
```

Anexo 4: Implementación paquete util_package

```
CREATE OR REPLACE PACKAGE util_package
AS
  PRAGMA SERIALLY_REUSABLE;
  colaux  VARCHAR2 (60);

  TYPE type_ref_cursor IS REF CURSOR;

  TYPE type_columns IS TABLE OF VARCHAR2 (40);


   FUNCTION find_constraints_column_func (
   tbl_name  IN  VARCHAR2,
   myowner  IN  VARCHAR2
  )
   RETURN type_ref_cursor;

  PROCEDURE find_constraints_column_proc (
   tbl_name  IN    VARCHAR2,
   myowner  IN    VARCHAR2,
   mycursor  OUT   type_ref_cursor
  );

  PROCEDURE get_conflicts_x_table (
   tbl_name  IN    VARCHAR2,
   mycursor  OUT   type_ref_cursor
  );

  PROCEDURE get_error_record (
   error_table_name  IN    VARCHAR2,
   id_error      IN    NUMBER,
   conflict_record  OUT   XMLTYPE
  );

  PROCEDURE get_all_columns (
   error_table_name  IN    VARCHAR2,
   columns_table   OUT   type_columns
  );

  PROCEDURE print_column (columns_table IN type_columns);

  PROCEDURE get_conflict_orignal_record (
   error_table_name  IN    VARCHAR2,
   id_error      IN    NUMBER,
   original      OUT   XMLTYPE,
   constraint_name  IN    VARCHAR2
  );

  PROCEDURE get_constraint_table (
   error_table_name  IN    VARCHAR2,
   constraint_name  IN    VARCHAR2,
```

```
   constraint_table  OUT   type_ref_cursor
  );

  PROCEDURE solve_conflic (
   original_record        IN  XMLTYPE,
   conflict_record        IN  XMLTYPE,
   id_conflict          IN  NUMBER,
   table_name          IN  VARCHAR2,
   columns_constraint_violated  IN  type_ref_cursor,
   all_constrain         IN  type_ref_cursor,
   columns_table         IN  type_columns,
   node             IN  NUMBER
  );
END util_package;
/

CREATE OR REPLACE PACKAGE BODY util_package
AS
  PRAGMA SERIALLY_REUSABLE;


--**************************
  PROCEDURE find_constraints_column_proc (
   tbl_name  IN    VARCHAR2,
   myowner  IN    VARCHAR2,
   mycursor  OUT   type_ref_cursor
  )
  IS
   tmp_cur  type_ref_cursor;
  BEGIN
   OPEN tmp_cur FOR
     SELECT DISTINCT a.table_name, a.column_name, a.constraint_name,
            b.constraint_type, b.r_constraint_name
         FROM user_cons_columns a, user_constraints b
         WHERE a.table_name IN (SELECT table_name
                     FROM all_tables
                     WHERE owner = myowner)
          AND a.table_name = b.table_name
          AND a.table_name = tbl_name
          AND a.constraint_name = b.constraint_name
          AND (  b.constraint_type = 'R'
            OR b.constraint_type = 'P'
            OR b.constraint_type = 'U'
            );

   mycursor := tmp_cur;
  END;

--**********************
  PROCEDURE get_conflicts_x_table (
```

```
   tbl_name  IN    VARCHAR2,
   mycursor  OUT   type_ref_cursor
  )
  IS
   tmp_cur  type_ref_cursor;
  BEGIN
   OPEN tmp_cur FOR
     SELECT DISTINCT id_error, table_name, id_row, constraint_violated
          FROM oraerror
          WHERE table_name = tbl_name;

   mycursor := tmp_cur;
  END;

--**********************
  PROCEDURE get_error_record (
   error_table_name  IN    VARCHAR2,
   id_error      IN    NUMBER,
   conflict_record  OUT   XMLTYPE
  )
  IS
   tmp_cur    type_ref_cursor;
   error_record  XMLTYPE;
  BEGIN
   SELECT conflict_record
    INTO error_record
    FROM log_generic_conflict p
    WHERE p.id_conflict = id_error;

   conflict_record := error_record;
  END;

--*************************
  PROCEDURE get_all_columns (
   error_table_name  IN    VARCHAR2,
   columns_table   OUT   type_columns
  )
  IS
   colmuns_table_x  type_columns
     := type_columns ('', '', '', '', '',  '', '', '', '', '', '','', '', '','', '',
''',
             '', '', '','', '', '', '', '', '', '', '', '',  '', '', '', '',
             '',  '', '', '',  '', ''
             );
   tbl_name     VARCHAR2 (40);
   cont       NUMBER (11);
   colm       type_ref_cursor;
  BEGIN
-- tbl_name := upper(error_table_name);
   OPEN colm FOR ' select column_name from user_tab_cols where table_name= :1'
```

```
   USING error_table_name;

   cont := 1;

   LOOP
     FETCH colm
     INTO tbl_name;

     EXIT WHEN colm%NOTFOUND;
     colmuns_table_x (cont) := tbl_name;
     DBMS_OUTPUT.put_line ('inserto ' || tbl_name);
     cont := cont + 1;
   END LOOP;

   columns_table := colmuns_table_x;
   DBMS_OUTPUT.put_line ('ASIGNO OK');
  END;


--**********************************
  PROCEDURE get_constraint_table (
   error_table_name  IN    VARCHAR2,
   constraint_name  IN    VARCHAR2,
   constraint_table  OUT   type_ref_cursor
  )
  IS
   constraint_x  type_ref_cursor;
  BEGIN
   OPEN constraint_x FOR ' select campo from constraints_col_temp where nombre_tabla=
:1 and nombre_constraint= :2 '
   USING error_table_name, constraint_name;

   constraint_table := constraint_x;
  END;

--**********************************
  PROCEDURE get_conflict_orignal_record (
   error_table_name  IN    VARCHAR2,
   id_error      IN    NUMBER,
   original      OUT   XMLTYPE,
   constraint_name  IN    VARCHAR2
  )
  IS
   constraint_table    util_package.type_ref_cursor;
   auxoriginal      util_package.type_ref_cursor;
   auxerror_record    util_package.type_ref_cursor;
   auxcolum        util_package.type_columns
     := type_columns ('', '', '', '', '',  '', '', '', '', '', '','', '', '','', '',
'',
            '', '', '','', '', '', '', '', '', '', '', '', '',  '', '', '', '',
```

```
            '',  '', '', '',  '', ''
            );
   conflict         XMLTYPE;
   varconflict       XMLTYPE;
   cust           XMLTYPE;
   constraint_row_table  constraints_col_temp%ROWTYPE;
   -- er error_record%rowtype;
   stmt_where        VARCHAR2 (500);
   stmt_auxwhere      VARCHAR2 (500);
   otro           util_package.type_ref_cursor;
   str           VARCHAR2 (100);
   campo          VARCHAR2 (60);
   PATH           VARCHAR2 (60);
   v_sql          VARCHAR2 (32767);
   cont           NUMBER (11);
   campo1          VARCHAR2 (60);
   pos           NUMBER (3);
   mylength         NUMBER (4);
   val           VARCHAR2 (60);
  BEGIN
   campo := ' O ';
   campo1 := ' O ';
   val := ' O ';
   stmt_where := ' ';
   stmt_auxwhere := ' ';
   PATH := ' O ';

   EXECUTE IMMEDIATE  ' SELECT DBURIType(''/REPLICA/''
            || error_table_name
            || 'MIA/ROW[ID_ERROR='
            || id_error
            || ' ]'').getXML() FROM DUAL'
         INTO conflict;

   INSERT INTO log_generic_conflict
      VALUES (id_error, conflict);

   OPEN constraint_table FOR  ' select * from constraints_col_temp where
nombre_constraint = '''
               || constraint_name
               || ''' and nombre_tabla = '''
               || error_table_name
               || ''' ';

   --OBTENIENDO LA TUPLA QUE SE TRATO DE INSERTAR Y PROVOCO EL CONFLICTO
   SELECT conflict_record
    INTO varconflict
    FROM log_generic_conflict p
    WHERE p.id_conflict = id_error;
```

```
LOOP
  FETCH constraint_table
  INTO constraint_row_table;

  EXIT WHEN constraint_table%NOTFOUND;

  SELECT CONCAT ('//', constraint_row_table.campo)
   INTO campo
   FROM DUAL;

  SELECT CONCAT (campo, '/text()')
   INTO campo
   FROM DUAL;

  cust := varconflict.EXTRACT (campo);
  val := cust.getnumberval ();
  DBMS_OUTPUT.put_line (' el valor es ' || val);

  SELECT CONCAT (constraint_row_table.campo, ' = ')
   INTO stmt auxwhere
   FROM DUAL;

  SELECT CONCAT (stmt_auxwhere, '"')
   INTO stmt_auxwhere
   FROM DUAL;

  SELECT CONCAT (stmt_auxwhere, val)
   INTO stmt_auxwhere
   FROM DUAL;

  SELECT CONCAT (stmt_auxwhere, '"')
   INTO stmt_auxwhere
   FROM DUAL;

  SELECT CONCAT (stmt_auxwhere, ' and')
   INTO stmt_auxwhere
   FROM DUAL;

  SELECT CONCAT (stmt_where, stmt_auxwhere)
   INTO stmt_where
   FROM DUAL;
END LOOP;

SELECT LENGTH (stmt_where)
 INTO mylength
 FROM DUAL;

DBMS_OUTPUT.put_line (stmt_where || ' y su largo es de ' || mylength);

SELECT NVL (SUBSTR (stmt_where, 0, mylength - 4), 0)
```

```
    INTO stmt_auxwhere
    FROM DUAL;

   DBMS_OUTPUT.put_line (stmt_auxwhere);

--//OBTENIENDO LA TUPLA ORIGINAL QUE EXISTIA Y DIO LUGAR A CONFLICTO
   EXECUTE IMMEDIATE  ' SELECT DBURIType(''/REPLICA/''
            || error_table_name
            || '/ROW['
            || stmt_auxwhere
            || ']'').getXML() FROM DUAL'
          INTO conflict;

   INSERT INTO original_record
      VALUES (id_error, conflict);

   --ASINANDO LA TUPLA OBTENIDA A LA VARIABLE POR REERENCIA.
   original := conflict;
  END;

--************************************
  PROCEDURE solve_conflic (
   original_record        IN  XMLTYPE,
   conflict_record        IN  XMLTYPE,
   id_conflict          IN  NUMBER,
   table_name          IN  VARCHAR2,
   columns_constraint_violated  IN  type_ref_cursor,
   all_constrain         IN  type_ref_cursor,
   columns_table         IN  type_columns,
   node             IN  NUMBER
  )
  IS
   original_aux   XMLTYPE;
   conflic_aux    XMLTYPE;
   val_original   VARCHAR2 (45);
   val_conflict   VARCHAR2 (45);
   same_id      NUMBER (1);
   sequence_column  VARCHAR2 (45);
  BEGIN
   IF table_name = 'PRUEBA'
   THEN
     sequence_column := 'ID_DOCUMENTO';
   END IF;

   SELECT CONCAT ('//', sequence_column)
    INTO campo
    FROM DUAL;

   SELECT CONCAT (campo, '/text()')
    INTO campo
```

```
    FROM DUAL;

   original_aux := original_record.EXTRACT (campo);
   val_original := original_aux.getnumberval ();
   conflic_aux := conflict_record.EXTRACT (campo);
   val_conflict := conflic_aux.getnumberval ();

   IF val_original = val_conflict
   THEN
     same_id := 1;
   ELSE
     same_id := 0;
   END IF;

  END;

--****************************************
PROCEDURE recursive_cascade (
  table_name      IN  VARCHAR2,
  sequence column  IN  VARCHAR2,
  val_original_aux  IN  VARCHAR2,
  val_conflict_aux  IN  VARCHAR2
)
IS
  myparents   util_package.type_ref_cursor;
  tabla     VARCHAR2 (40);
  var_type    VARCHAR2 (3);
  columna_hija  VARCHAR2 (40);
  v_sql     VARCHAR2 (160);
BEGIN
  OPEN myparents FOR  ' select nombre_tabla, columna_hija from parents where
tabla_padre = '''
           || table_name
           || ''' and columna_padre='''
           || sequence_column
           || ''' order by nombre_tabla';

  LOOP
   FETCH myparents
    INTO tabla, columna_hija;

   EXIT WHEN myparents%NOTFOUND;
   v_sql :=
      ' update '
     || tabla
     || ' set '
     || columna_hija
     || '= '
     || val_conflict_aux
     || ' where '
```

```
      || columna_hija
      || ' = '
      || val_original_aux
      || '';
    DBMS_OUTPUT.put_line (  ' tabla hija ; '
                || tabla
                || ', columna hija '
                || columna_hija
                || ' sql '
                || v_sql
                );

    IF column_is_pk (tabla, columna_hija)
      AND is_parent (tabla, columna_hija)
    THEN
      recursive_cascade (table_name,
                sequence_column,
                val_original_aux,
                val_conflict_aux
                );
    END IF;

    IF is_execute (v_sql)
    THEN
--column_is_pk(tabla,columna_hija ) and
      DBMS_OUTPUT.put_line ('****// operaciÃ³n satisfactoria' || v_sql);

      EXECUTE IMMEDIATE  'delete from '
                || table_name
                || ' where '
                || sequence_column
                || ' = '
                || val_original_aux
                || ' ';
    END IF;
--execute immediate ' update '||tabla||' set '||columna_hija||'=
'||val_conflict_aux||' where '||sequence_column||' = '||val_original_aux||'';

  --end if;
  END LOOP;
END;
END util_package;
/
```

Anexo 5: Implementación función find_constraints_column

```
CREATE OR REPLACE FUNCTION find_constraints_column
  RETURN NUMBER
```

```
IS
  /* declaracion de variables */
  acc_bal         NUMBER (11, 2);
  var_const_name    VARCHAR2 (60);
  var_tbl_name     VARCHAR2 (30);
  var_col_name     VARCHAR2 (30);
  var_constraint_type  VARCHAR2 (3);
  var_ref_const_name  VARCHAR2 (30);
  micursor       util_package.type_ref_cursor;
  str_tbl_name     VARCHAR2 (30);
  str_schema_name    VARCHAR2 (30);
  str_string      VARCHAR2 (80);
  pos          NUMBER (3);

  CURSOR otrocursor
  IS
   SELECT table_name
    FROM rep_control_name_table;
BEGIN
  acc_bal := 1;

--OPEN otrocursor;
  FOR x_row IN otrocursor
  LOOP
   str_string := x_row.table_name;

   SELECT NVL (INSTR (str_string, '.'), 0)
    INTO pos
    FROM DUAL;

   SELECT NVL (SUBSTR (str_string, pos + 1), 0)
    INTO str_tbl_name
    FROM DUAL;

   SELECT NVL (SUBSTR (str_string, 0, pos - 1), 0)
    INTO str_schema_name
    FROM DUAL;

   util_package.find_constraints_column_proc (str_tbl_name,
                        str_schema_name,
                        micursor
                       );

   LOOP
     FETCH micursor
     INTO var_tbl_name, var_col_name, var_const_name,
        var_constraint_type, var_ref_const_name;

     EXIT WHEN micursor%NOTFOUND;
```

```
      INSERT INTO constraints_col_temp
        VALUES (var_col_name, var_const_name, var_constraint_type,
            var_tbl_name);

    END LOOP;

    COMMIT;

    CLOSE micursor;
  END LOOP;

  RETURN (acc_bal);
END find_constraints_column;
```

Anexo 6: Implementación procedimiento find_parents

```
CREATE OR REPLACE PROCEDURE find_parents
IS
/*Declarar variables*/
  var_tbl_name     VARCHAR2 (30);
  var_ref_tbl_name   VARCHAR2 (30);
  var_const_name    VARCHAR2 (30);
  var_col_name     VARCHAR2 (30);
  var_ref_const_name  VARCHAR2 (30);
  var_ref_col_name   VARCHAR2 (30);
  var_table_name    VARCHAR2 (30);
  aux         util_package.type_ref_cursor;

  CURSOR micursor
  IS
/* Hallar nombre de tabla, atributo ,
 nombre de clave foranea
 y nombre de clave primaria que se referncia*/
   SELECT DISTINCT a.table_name, a.column_name, a.constraint_name,
           b.r_constraint_name
         FROM user_cons_columns a, user_constraints b
        WHERE a.table_name IN (SELECT table_name
                    FROM all_tables
                   WHERE owner = 'REPLICA')
          AND a.table_name = b.table_name
          AND a.constraint_name = b.constraint_name
          AND b.constraint_type = 'R'
       ORDER BY a.table_name;
BEGIN
  OPEN micursor;

  LOOP
   FETCH micursor
```

```
    INTO var_tbl_name, var_col_name, var_const_name, var_ref_const_name;

   EXIT WHEN micursor%NOTFOUND;

   /* Hallar dado el nombre de la clave primaria que se referencia
   la tabla y el atributo al que pertenece esa clave primaria*/
   EXECUTE IMMEDIATE ' SELECT E.TABLE_NAME, E.COLUMN_NAME FROM USER_CONS_COLUMNS E,
USER_CONSTRAINTS D WHERE D.CONSTRAINT_NAME = : 1
 AND E.TABLE_NAME =D.TABLE_NAME
 AND E.CONSTRAINT_NAME=D.CONSTRAINT_NAME '
          INTO var_ref_tbl_name, var_ref_col_name
         USING var_ref_const_name;



   INSERT INTO parents
      VALUES (var_tbl_name, var_ref_tbl_name);
  END LOOP;

  CLOSE micursor;
END find_parents;
```

Anexo 7: Implementación procedimiento find_aux_parents

```
CREATE OR REPLACE PROCEDURE find_parents_aux
IS
/*Declarar variables*/
  var_tbl_name    VARCHAR2 (30);
  var_table_name   VARCHAR2 (30);
  var_aux_tbl_name  VARCHAR2 (30);
  myaux       util_package.type_ref_cursor;

  CURSOR aux
  IS
   SELECT DISTINCT tabla_padre
         FROM parents
       ORDER BY tabla_padre;
BEGIN
  OPEN aux;

  LOOP
   FETCH aux
    INTO var_tbl_name;

   EXIT WHEN aux%NOTFOUND;

   IF exist_table_name (var_tbl_name)
   THEN
     DBMS_OUTPUT.put_line ('No existe');
   ELSE
     INSERT INTO parents
```

```
       VALUES (var_tbl_name, NULL);

     DBMS_OUTPUT.put_line ('Existe');
    END IF;
  END LOOP;

  CLOSE aux;
END find_parents_aux;
```

Anexo 8: Implementación procedimiento exist_table_name

```
CREATE OR REPLACE FUNCTION exist_table_name (tbl_name VARCHAR2)
  RETURN BOOLEAN
IS
  var_table_name  VARCHAR2 (60);
BEGIN
  SELECT nombre_tabla
   INTO var_table_name
   FROM parents
  WHERE nombre_tabla = tbl_name;

  RETURN TRUE;
EXCEPTION
  WHEN NO_DATA_FOUND
  THEN
   RETURN FALSE;
END;
```