Universidad Central "Marta Abreu" de las Villas
Facultad de Matemática Física y Computación
Departamento de Ciencia de la Computación

Modelo de replicación de datos para el Sistema de Control y Cobro de Multas

Tesis presentada en opción al título académico de
Máster en Ciencia de la Computación

Autora: *Lic. Vivian Milagros Romero Buchillón*

Tutores: *Dr. Abel Rodríguez Morffi*
Dra. Luisa Manuela González González

Santa Clara
2011

"El progreso consiste en el cambio"

Miguel de Unamuno

AGRADECIMIENTOS

A mis padres por la buena educación que me han dado y por siempre estar de mi lado.

A mis tutores, el Dr. Abel Rodríguez Morffi y la Dra. Luisa Manuela González González, por tantas buenas enseñanzas ofrecidas y por dedicarme gran parte de su valioso tiempo.

A mi esposo, por su comprensión y ayuda.

A todos mis compañeros de trabajo.

A todas las personas que de una forma u otra han sido parte de este logro.

DEDICATORIA

A mis padres, por mi educación, por mi formación, por mi vida actual.
A mi hija Liana, que ha sido el mejor suceso de mi vida.
A mi esposo, Jorge, por brindarme siempre su apoyo.

RESUMEN

Desde hace algunos años, el estado cubano se encuentra inmerso en un proceso de informatización de la sociedad mediante la aplicación las Tecnologías de la Información y la Comunicación para lograr la soberanía tecnológica. Tradicionalmente se le ha prestado interés al enfrentamiento a las actividades delictivas, las ilegalidades y las indisciplinas sociales. Una de las actividades que actúa como reguladora de la disciplina social es la imposición y el cobro de multas por contravenciones. La gestión de multas se realiza a nivel municipal a través del Sistema de Control y Cobro de Multas (SCCM) pero los datos sobre esto también son de interés a nivel provincial y nacional.

En este trabajo se crea e implementa un modelo de replicación que garantiza autonomía a los sitios donde se gestionan las multas y disponibilidad de los datos a todos los niveles en el menor tiempo posible.

El modelo creado se aplicó en el subsistema de replicación del SCCM y se obtuvo resultados satisfactorios en su implantación en Villa Clara. Se demostró que SymmetricDS es un software apropiado para la replicación bidireccional de datos del modelo propuesto.

ABSTRACT

Since many years, the Cuban government has been focusing on the use of IT improvements in order to achieve technological sovereignty. Traditionally, delinquency, illegalities and social indiscipline have been ruled and contraventions are penalized by means of fines. The management of payments is made at municipal offices, though the data about fines are also important at province and national level.

The present work addresses the design and implementation of a replication model that guarantee site autonomy for municipal offices where fines are rewarded and immediate data availability for provinces and nation.

The implementation of this model using the SymmetricDS tool revealed the model is correct and the tool is very suitable for bidirectional replication under the proposed model.

TABLA DE CONTENIDOS

Pág.

INTRODUCCIÓN

Desde hace algunos años, el estado cubano se encuentra inmerso en un proceso de informatización de la sociedad, basado en la aplicación ordenada y masiva de las Tecnologías de la Información y la Comunicación y orientado a lograr la soberanía tecnológica, por lo que surge la necesidad de desarrollar sistemas eficientes y a la altura de las nuevas tecnologías que se introducen como parte de los avances económicos que tiene el país.

Tradicionalmente se le ha prestado particular interés al enfrentamiento a las actividades delictivas, las ilegalidades y las indisciplinas sociales, sobre todo a partir del inicio del período especial cuando estas aumentaron considerablemente. Una de las actividades que actúa como reguladora de la disciplina social es la imposición y el cobro de multas por contravenciones.

La actividad de multas en Cuba se realiza en las Oficinas de Control y Cobro de Multas (OCCM) que existen en cada municipio; estas se subordinan a la Dirección Municipal de Finanzas y Precios (DMFP) en cada gobierno municipal, así como a la Dirección Provincial de Finanzas y Precios (DPFP) en cada gobierno provincial. La DPFP se subordina a la Dirección Nacional de Multas (DNM) dentro del Ministerio de Finanzas y Precios (MFP), el cual traza las directrices de la actividad, además de administrar y distribuir los recursos destinados para las mismas.

A principios del año 2010, sólo 42 de las 169 OCCM del país estaban automatizadas por lo que el Proceso de Control y Cobro de Multas (PCCM) en las 127 restantes se realizaba de forma manual. Esto provocó que:

- La DNM gastara anualmente varios miles de pesos en la compra de los modelos en papel necesarios para la realización de la actividad. En el año 2010 el gasto fue de 92300,00 pesos cubanos y 109500,00 pesos convertibles.
- La DNM gastara 124015,00 pesos cubanos en el envío de multas a través de correo postal de una OCCM a otra.
- El proceso de obtención de informes de consolidación en las DPFP del PCCM de las OCCM pertenecientes a esta, se tornara complicado y podría tardar varios días.
- La obtención de informes de consolidación en la DNM del PCCM de todas las OCCM del país se convirtiera en un proceso muy complejo que podría tardar semanas.

- La determinación de las multas sin pagar que tiene una persona fuera un proceso incompleto al no tener toda la información del país en un lugar accesible, lo que condujo a que no se detectaran todas las multas sin pagar de los infractores. Sólo en 2010 se cancelaron más de 40 000 multas por prescripción, o sea, porque no se localizó al infractor durante un período de más de un año. Esto ocurrió principalmente cuando la persona a la que se le impuso la multa no vive en el municipio donde radica la OCCM donde se originó la multa. Esta cifra cuantifica un total de 8 millones de pesos; sin embargo, se ha comprobado que en la mayoría de los casos, los infractores estaban localizables e incluso pagaron las multas impuestas en otros municipios.
- La imposibilidad por parte de las instancias superiores de tener mejor control y supervisión del trabajo que realizan las OCCM, lo que en ocasiones ha inducido a la ocurrencia de hechos de corrupción y malversación en las oficinas de multas.

Por otro lado, el sistema implantado en las oficinas que están informatizadas, tiene como principal limitante que su alcance es municipal, o sea, solamente se tiene acceso a la información de la OCCM donde se encuentra. Esto imposibilita la interacción con otras OCCM y con las instancias superiores. Además, las entidades externas al MFP tales como: la Aduana General de la República (AGR), la Policía Nacional Revolucionaria (PNR), los gobiernos territoriales, la Dirección Nacional de Tránsito (DNT), entre otras, no pueden obtener información sobre las multas que debe un infractor. En el caso de la aduana, por ejemplo, no se permite que un ciudadano salga definitivamente del país sin haber saldado sus deudas con el estado cubano y esta información en lo que a multas se refiere no se puede obtener de forma automática.

Todo lo anteriormente expuesto resume sintéticamente la *situación problemática* que fundamenta el *problema científico a resolver* consistente en la necesidad de manejar información local en las OCCM y permitir el intercambio bidireccional de estas con las DPFP correspondientes, así como entre las DPFP y la DNM; de manera que en las DPFP se encuentre toda la información de la provincia y en la DNM la del país.

En correspondencia con lo planteado anteriormente y derivado de la construcción del marco teórico–referencial de la investigación, se formuló la siguiente *hipótesis* de la investigación: Mediante el diseño e implementación de un modelo de replicación de datos para el Sistema Automatizado para el Control y Cobros de Multas se podrá gestionar toda la información de multas del país en los distintos niveles sin que las oficinas pierdan autonomía sobre sus datos.

El *objetivo general* de este trabajo es crear e implementar un modelo de replicación que garantice la distribución de datos para el control y cobro de multas a todos los niveles, favoreciendo la toma de decisiones.

Para lograr este objetivo se plantean los siguientes *objetivos específicos* de la investigación:

1. Describir el proceso de control y cobro de las multas con el fin de identificar los principales aspectos relacionados con la distribución de los datos.
2. Estudiar los principales aspectos relacionados con la replicación para implementar la distribución de datos.
3. Seleccionar una herramienta de software libre para la replicación de datos.
4. Implementar un modelo de replicación para el control y cobro de las multas.

Preguntas de la investigación:

1. ¿Qué características debe tener un modelo de replicación que facilite la gestión distribuida de los datos para el control y cobro de las multas según los procedimientos establecidos para ello?
2. ¿Qué Sistemas de Gestión de Bases de Datos de software libre permiten la creación de bases de datos distribuidas?
3. ¿Qué herramientas pueden ayudar a la implementación de un modelo de replicación entre servidores donde estos puedan ser publicadores y/o suscriptores simultáneamente?
4. ¿Cuáles herramientas cumplen con los requisitos necesarios para la implementación del modelo de replicación de datos para el problema de control y cobro de las multas?

Para el desarrollo de la investigación se utilizaron métodos y técnicas empíricas como: análisis de documentos, análisis comparativos, observación y criterio de expertos. Además se emplearon métodos teóricos como el analítico sintético, inductivo deductivo, la modelación y el enfoque sistémico estructural. También el análisis lógico, la analogía, la reflexión y otros procesos mentales que también le son inherentes a toda actividad de investigación científica.

El *valor teórico* de la investigación radica en el diseño del modelo de replicación de datos y en la selección y uso de una herramienta de software libre para su implementación.

El *valor metodológico* se manifiesta en la consistencia del modelo de replicación de datos diseñado y la herramienta de replicación utilizada que facilitan su uso para problemas con características similares.

Su *valor práctico* radica en la factibilidad y pertinencia demostrada de haber aplicado el modelo de replicación de datos en el sistema automatizado para el control y cobro multas y la posibilidades de aplicar dicho modelo en la solución de problemas similares, así como el conocimiento obtenido de la herramienta utilizada para la replicación y su posible utilización en otras situaciones.

La tesis está estructurada en tres capítulos:

En el capítulo 1 se presenta un estudio sobre bases de datos distribuidas y la distribución de datos mediante replicación; se hace una valoración de algunas herramientas de software libre para la replicación de datos.

En el capítulo 2 se explica el modelo de replicación para el control y cobro de las multas.

En el capítulo 3 se realiza una presentación de la herramienta seleccionada para implementar el modelo de replicación descrito.

Esta tesis de maestría culmina con las conclusiones, recomendaciones generales derivadas del proceso de investigación realizado y referencias bibliográficas.

CAPÍTULO 1: BASES DE DATOS DISTRIBUIDAS Y REPLICACIÓN DE DATOS

En este capítulo se refieren los aspectos que son particularmente interesantes en Bases de Datos Distribuidas (BDD) y la replicación de datos. Se presenta una valoración de algunas herramientas de software libre para la replicación de datos.

1.1 Introducción a las bases de datos en ambientes distribuidos

El término Base de Datos Distribuida (BDD) tiene diferentes definiciones, para el propósito de este trabajo es suficiente la dada por (Őzsu and Valduriez, 1999). Una BDD es una colección de múltiples Bases de Datos (BD), lógicamente interrelacionadas distribuidas sobre una red de computadoras. La distribución involucra el hecho de que los datos no residen necesariamente en el mismo sitio, pero poseen propiedades comunes que los vinculan, y se facilita su acceso a través de una interfaz común. Es necesario destacar que los enlaces entre los datos se llevan a cabo en una red de comunicación, lo que implica generalmente que los sitios estén ubicados en diferentes áreas geográficas y con capacidad de procesamiento autónomo.

Para el diseño de BDDs se han definido dos grandes estrategias (Ceri et al., 1987), el enfoque ascendente (*Top-Down)* y el descendente (*Bottom-Up).* En el enfoque *ascendente* se comienza diseñando el esquema global, luego se concibe la fragmentación de la BD y la localización de los fragmentos en los sitios. Se completa ejecutando, en cada sitio, el diseño físico de los datos. Por otro lado el enfoque *descendente* se basa en la integración de esquemas ya creados en un esquema global a partir de las BD existentes.

Los Sistemas de Bases de Datos Distribuidas (SBDD) representan más naturalmente la estructura geográficamente descentralizada de una organización, aumentan la disponibilidad de los datos, reducen el tráfico de comunicación y es justificable, además, por el abaratamiento actual de los costos en el equipamiento y la infraestructura de comunicaciones de las redes de computadoras y su objetivo fundamental es integrar la manipulación de datos para que sean presentados al usuario como una única colección de datos global y coherente (Őzsu and Valduriez, 1999). El diseño de una BDD posee las fases del diseño centralizado y cuenta, además, con dos nuevos problemas que caracterizan el proceso de distribución de datos, e incluyen la determinación de: cómo dividir la base de

datos en componentes para localizarlos en diferentes sitios, qué cantidad de datos deben ser replicados y cómo deben ser ubicados los fragmentos replicados.

Un Sistema de Gestión de Bases de Datos Distribuidas (SGBDD) es un Sistema de Gestión de Bases de Datos que gestiona la BD distribuida.

El diseño de la BDD tiene como objetivo lograr un mejor desempeño y requiere de su propia teoría, metodología y herramientas de apoyo. Las diferentes técnicas de distribución de datos generalmente se basan en la semántica de los datos, y se rigen por principios idénticos que las BD centralizadas, incorporando otros detalles particulares, como la fragmentación de las entidades y su posterior localización en los diferentes sitios de la red. La fragmentación es muy útil a los fines de mejorar los tiempos de respuesta y garantizar el paralelismo de un sistema (Johansson et al., 2000, Őzsu and Valduriez, 1999, Baião et al., 2004, Irún-Briz et al., 2005, Lin and Veeravalli, 2006, Coulon et al., 2005). En el diseño de la distribución influyen muchos factores relacionados con la BD, las aplicaciones que acceden a la misma, la comunicación de la red, y sobre el sistema de computadoras que la soporta; lo que hace que sea muy complicada la formulación de un problema de distribución. Por tanto, es necesario decidir cómo fragmentar y distribuir los datos sobre los diferentes sitios y cuáles de estos datos deben ser replicados.

La distribución de la BD requiere determinar la fragmentación y la localización. La fragmentación es el proceso de dividir una relación en pequeñas porciones llamadas fragmentos (Meghini and Thanos, March 1991). Las razones principales para la fragmentación son el incremento del nivel de concurrencia y el desempeño del sistema. Existen dos alternativas para fragmentar datos: fragmentación horizontal (FH) y fragmentación vertical (FV). La combinación de las anteriores resulta en una fragmentación híbrida o mixta. Es importante seguir tres reglas, las cuales aseguran que la BD no tenga cambios semánticos durante la fragmentación: completitud, reconstrucción y disjuntura. Los dos principios básicos de las BDD (Ceri et al., 1987, Ma et al., 2005, Őzsu and Valduriez, 1999, Ma and Schewe, 2005) son reducir el intercambio de datos entre los sitios y eliminar datos irrelevantes en la ejecución de solicitudes. Ambos sirven de guía al proceso de diseño de BDD, el cual ha sido dividido en cuatro pasos fundamentales (Baião et al., 2002, Bellatreche et al., 2000, Ceri et al., 1987, Hababeh et al., 2004, Huang and Chen, 2001, Navathe et al., 1995, Őzsu and Valduriez, 1999, Schewe, 2002):

1. Diseño del esquema conceptual, que describe la base de datos integrada, es decir, todos los datos que son utilizados por las aplicaciones que tienen acceso a las bases de datos.
2. Diseño de la fragmentación, que determina la forma en que se subdividen las relaciones globales en fragmentos horizontales, verticales o mixtos
3. Diseño de la asignación de los fragmentos, que decide la forma en que se mapean los fragmentos a las imágenes físicas.
4. Diseño físico de la base de datos, que mapea el esquema conceptual a las áreas de almacenamiento y determina los métodos de acceso a las BDs.

Los pasos 2 y 3 caracterizan al diseño de las BDD, mientras que los 1 y 4 son comunes a las Base de Datos Centralizadas (BDC) y las BDD. En este trabajo, para el diseño de BDDs se siguió un enfoque ascendente, en el que se parte de cero y se avanza en el desarrollo del trabajo. Los pasos a realizar mediante esta estrategia son los siguientes:

Análisis de requisitos**:** En esta etapa se determinan los requisitos para obtener tanto los datos como las necesidades de procesamientos de los usuarios. Igualmente se deberán fijar los requisitos del sistema, los objetivos que debe cumplir en cuanto a rendimiento, seguridad, disponibilidad y flexibilidad teniendo en cuenta aspectos económicos. Al finalizar esta etapa se poseen los objetivos que servirán como entrada para dos actividades: Diseño conceptual y diseño de vistas.

Diseño de vistas**:** En esta etapa se definen las interfaces del usuario con el sistema, se determinan las aplicaciones que usarán la BD así como datos estadísticos o estimaciones de las mismas sobre frecuencia de acceso de cada aplicación a cada tabla, que permitirá poseer información que ayudará a optimizar ciertas partes y crear un diseño conceptual más eficiente. Al finalizar esta etapa se debe poseer toda la información de acceso y la definición de los esquemas externos.

Diseño conceptual**:** En esta etapa se suele realizar la integración de las vistas del usuario. Como resultado de la ejecución de estas dos últimas etapas se tiene un esquema conceptual global, información de acceso y los esquemas externos que servirán de entrada para la próxima etapa.

Diseño de la distribución**:** Esta etapa es representativa en el diseño de BBDD ya que es la etapa que la diferencia del diseño de BDC. La misma consiste en obtener diferentes esquemas conceptuales locales a partir del esquema conceptual global y la información de acceso. Aquí se deben considerar dos actividades importantes:

• Fragmentación: consiste en decidir cómo dividir la BD y en qué partes.
• Asignación: consiste en ubicar los fragmentos obtenidos en los distintos sitios.

Diseño físico. A partir de los esquemas conceptuales locales y la información de acceso obtenidos en las etapas anteriores se debe obtener el esquema físico.

Monitorización y ajustes**.** Este paso se realiza para llevar un control del proceso e intentar reparar los errores o desviaciones que se produzcan en el proceso.

1.1.1 Fragmentación

Para la construcción del esquema local que se ubicará en cada sitio de la red, es común tomar del esquema global de datos las relaciones definidas y dividirlas en subrelaciones. Estas subrelaciones forman los fragmentos que luego serán distribuidos entre los esquemas locales (Őzsu and Valduriez, 1991) (Ceri et al., 1983).

Cuando se plantea el concepto de fragmentación de una BD deben tenerse en cuenta una serie de factores, a saber:

- ¿Por qué fragmentar?
- ¿Cómo se debería fragmentar?
- ¿Cuánto se debería fragmentar?
- ¿Es posible evaluar la correctitud de la descomposición?
- ¿Cómo deben ubicarse los fragmentos?

Cada uno de estos factores debe ser analizado, primero en general, y luego para cada problema en cuestión, a fin de encontrar el mejor esquema de distribución de datos.

1.1.2 ¿Por qué fragmentar?

Desde el punto de vista de la distribución de datos, no hay razones para fragmentar la información. En general, las BDD, aplican el esquema de distribución a “tabla completa”. Cuando se habla de fragmentación lo importante es definir cuál debe ser la unidad de fragmentación. La elección de una relación del esquema global como unidad de fragmentación no es razonable. Usualmente una relación posee diferentes vistas desde

diferentes usuarios, lo que plantea una serie de subconjuntos de esta relación. Es más natural considerar estos subconjuntos como unidades de distribución.

Otro factor, que hace que la relación completa no sea una unidad de fragmentación aceptable, está vinculado con el concepto anterior. Una relación en particular puede necesitarse en varias localidades de la red. Esto implica que un volumen importante de datos se encuentre repetido varias veces innecesariamente, lo que significará potenciales causas de problemas de mantenimiento e integridad de las copias y demoras en la ejecución de actualizaciones de la información. Además, la descomposición de la relación en fragmentos, permitirá que un número mayor de transacciones se ejecuten concurrentemente en el sistema.

Existe una serie de razones para llevar a cabo la fragmentación:

- Utilización: En general, las aplicaciones funcionan con vistas que normalmente son subconjuntos de relaciones. Por tanto, es lógico considerar como unidad de distribución a esos subconjuntos de relaciones.
- Eficiencia: Los datos se almacenan cerca del lugar en el que son utilizados con mayor frecuencia. Además, los datos que las aplicaciones locales no necesitan no se almacenan en ese sitio.
- Paralelismo: La descomposición de una relación en fragmentos permite que una transacción pueda ser dividida en subconsultas. Cada subconsulta operará sobre el fragmento adecuado. En definitiva, se aumenta el grado de concurrencia.
- Seguridad: Los datos no requeridos por las aplicaciones locales no se almacenan en ese sitio, por lo que no están disponibles para los usuarios no autorizados.

1.1.3 ¿Cómo fragmentar?

Hay, básicamente, dos formas para fragmentar una relación: dividirla horizontalmente o verticalmente. Una división horizontal consiste en el particionamiento en tuplas de una relación global en subconjuntos, donde cada subconjunto puede contener datos que cumplen una condición y se puede definir expresando cada fragmento como una operación de selección sobre la relación global. Un ejemplo, en el control y cobro de multas, es la relación de contraventores de una provincia ubicaría las tuplas de contraventores de cada municipio de residencia juntas formando una unidad de fragmentación como se observa en la figura 1.1.

id_contraventor bigint	nombre character varying(50)	primer_apellido character varying(50)	segundo_apellido character varying(50)	municipio_reside character varying(4)	id_zona integer
9216108486610506739	MARILIN	TORRES	RUIZ	0512	848096946
9220633216877389413	MARLEN	PEREZ	SANCHEZ	0512	575455610
9215421324583266441	ALAIN	OSES	CRUZ	0512	
9219399720197342479	OMAR	PEREZ	PEREZ	0512	1820160247
9218298918883157290	JULIO	DELGADO	NUÑEZ	0512	1335711237
9220759113158833455	NORGE	GUERRA	RDGUEZ	0512	1000780088
9223234211403178509	ALFREDO	TAPANES	NIEBLA	0512	388413748
9215284234645951049	RAMON	PEREZ	ALVAREZ	0512	
9218180462961496466	MAYKEL	ALLEGUE	DIAZ	0512	1820160247
9215133716268254630	RAMON	MACHADO	FUENTES	0512	1293848626
9223351845409310803	MARIO	AVALOS	BATISTA	0512	1820160247
308795049515245076	ROBERTO	MONTIEL	GARCIA	0511	
8898009462800167110	ABEL	FDEZ	BARNACH	0511	
14932511534584815	CARLOS	ESPINOSA	IBAÑES	0511	
19794887649824790	AGAPITO G	FUSTE	MARTINEZ	0511	

id_contraventor bigint	nombre character varying(50)	primer_apellido character varying(50)	segundo_apellido character varying(50)	municipio_reside character varying(4)	id_zona integer
9216108486610506739	MARILIN	TORRES	RUIZ	0512	848096946
9220633216877389413	MARLEN	PEREZ	SANCHEZ	0512	575455610
9215421324583266441	ALAIN	OSES	CRUZ	0512	
9219399720197342479	OMAR	PEREZ	PEREZ	0512	1820160247
9218298918883157290	JULIO	DELGADO	NUÑEZ	0512	1335711237
9220759113158833455	NORGE	GUERRA	RDGUEZ	0512	1000780088
9223234211403178509	ALFREDO	TAPANES	NIEBLA	0512	388413748
9215284234645951049	RAMON	PEREZ	ALVAREZ	0512	
9218180462961496466	MAYKEL	ALLEGUE	DIAZ	0512	1820160247
9215133716268254630	RAMON	MACHADO	FUENTES	0512	1293848626
9223351845409310803	MARIO	AVALOS	BATISTA	0512	1820160247

id_contraventor bigint	nombre character varying(50)	primer_apellido character varying(50)	segundo_apellido character varying(50)	municipio_reside character varying(4)	id_zona integer
8898009462800167110	ABEL	FDEZ	BARNACH	0511	
14932511534584815	CARLOS	ESPINOSA	IBAÑES	0511	
19794887649824790	AGAPITO G	FUSTE	MARTINEZ	0511	
308795049515245076	ROBERTO	MONTIEL	GARCIA	0511	

Figura 1.1 Fragmentación horizontal en la tabla contraventor

La fragmentación vertical consiste en particionar separando los atributos de una relación. Así, es posible ubicar solo algunos atributos juntos generando unidades de fragmentación. El requisito que debe establecerse cuando se separan los atributos, es que mediante una operación del álgebra relacional debe ser posible obtener la relación original. Para lograr el requisito se debe mantener el identificador o clave primaria de la relación en cada

fragmento, véase la figura 1.2. Este esquema de fragmentación vertical es inherentemente más complejo que el esquema de fragmentación horizontal.

id_contraventor bigint	nombre character varying(50)	primer_apellido character varying(50)	segundo_apellido character varying(50)	municipio_reside character varying(4)	id_zona integer
9216108486610506739	MARILIN	TORRES	RUIZ	0512	848096946
9220633216877389413	MARLEN	PEREZ	SANCHEZ	0512	575455610
9215421324583266441	ALAIN	OSES	CRUZ	0512	
9219399720197342479	OMAR	PEREZ	PEREZ	0512	1820160247
9218298918883157290	JULIO	DELGADO	NUÑEZ	0512	1335711237
9220759113158833455	NORGE	GUERRA	RDGUEZ	0512	1000780088
9223234211403178509	ALFREDO	TAPANES	NIEBLA	0512	388413748
9215284234645951049	RAMON	PEREZ	ALVAREZ	0512	
9218180462961496466	MAYKEL	ALLEGUE	DIAZ	0512	1820160247
9215133716268254630	RAMON	MACHADO	FUENTES	0512	1293848626
9223351845409310803	MARIO	AVALOS	BATISTA	0512	1820160247
308795049515245076	ROBERTO	MONTIEL	GARCIA	0511	
8898009462800167110	ABEL	FDEZ	BARNACH	0511	
14932511534584815	CARLOS	ESPINOSA	IBAÑES	0511	
19794887649824790	AGAPITO G	FUSTE	MARTINEZ	0511	

id_contraventor bigint	nombre character varying(50)	primer_apellido character varying(50)	segundo_apellido character varying(50)	municipio_reside character varying(4)
9216108486610506739	MARILIN	TORRES	RUIZ	0512
9220633216877389413	MARLEN	PEREZ	SANCHEZ	0512
9215421324583266441	ALAIN	OSES	CRUZ	0512
9219399720197342479	OMAR	PEREZ	PEREZ	0512
9218298918883157290	JULIO	DELGADO	NUÑEZ	0512
9220759113158833455	NORGE	GUERRA	RDGUEZ	0512
9223234211403178509	ALFREDO	TAPANES	NIEBLA	0512
9215284234645951049	RAMON	PEREZ	ALVAREZ	0512
9218180462961496466	MAYKEL	ALLEGUE	DIAZ	0512
9215133716268254630	RAMON	MACHADO	FUENTES	0512
9223351845409310803	MARIO	AVALOS	BATISTA	0512
308795049515245076	ROBERTO	MONTIEL	GARCIA	0511
8898009462800167110	ABEL	FDEZ	BARNACH	0511
14932511534584815	CARLOS	ESPINOSA	IBAÑES	0511
19794887649824790	AGAPITO G	FUSTE	MARTINEZ	0511

Figura 1.2 Fragmentación vertical en la tabla contraventor

Existe, además, un esquema de fragmentación denominado híbrido o mixto. En algunos casos una fragmentación horizontal o vertical no es suficiente para satisfacer los requerimientos de una aplicación particular. En esos casos, se aplica simultáneamente fragmentación horizontal y vertical sobre la relación.

1.1.4 ¿Cuánto fragmentar?

La extensión en que la BD debería ser fragmentada es una decisión importante que afecta el rendimiento de las consultas. El grado de fragmentación va desde un extremo en que la BD no se fragmenta, al otro, donde se fragmenta hasta cada tupla (horizontal) o cada atributo (vertical). En general, si se plantea un esquema donde cada unidad de fragmentación es demasiado pequeño aumentan los efectos adversos. Lo que se necesita es encontrar un nivel de fragmentación óptimo entre los dos extremos. Este nivel puede ser solo definido respecto a cada aplicación particular de BD. Para determinar cuánto fragmentar se deben tener en cuenta una serie de parámetros que dependen directamente del problema que se está estudiando. Por lo tanto, no es posible establecer, a priori y en forma genérica, el grado de fragmentación. Un aspecto interesante es simular el comportamiento de la BDD con diferentes esquemas de fragmentación para determinar aquel que mejor se adapte al problema que se trate.

1.1.5 ¿Cómo evaluar la correctitud de la descomposición?

Es posible aplicar un conjunto de reglas que determinen si la fragmentación de la BD generó o no problemas semánticos en la misma. Estas reglas son:

- Completitud: si una relación R es descompuesta en fragmentos R_1, R_2,..., R_n cada elemento de datos que puede ser encontrado en R también puede ser encontrado en uno o más fragmentos. Los datos encontrados en R deben ser mapeados en los fragmentos sin ninguna pérdida.
- Reconstructibilidad: si la relación R se descompone en fragmentos R_1, R_2,..., R_n es posible definir un operador del álgebra relacional que, aplicado sobre los fragmentos permita encontrar nuevamente a R. En el caso de fragmentación horizontal el operador será el de unión (union), en tanto que el acople (join) será el utilizado para la fragmentación vertical. Con estos operadores es posible reconstruir la relación original. La reconstructibilidad de una relación desde sus fragmentos asegura que se preserven las dependencias existentes entre los datos.

- Disjuntura: si la relación R se descompone en fragmentos R_1, R_2,..., R_n y si el elemento de dato d_i se encuentra en el fragmento R_j, no lo está en otro fragmento R_k con $k \neq j$. Se debe notar que si la descomposición es vertical el atributo que es clave primaria se encuentra repetido en todos los fragmentos, lo que significaría no cumplir con esta regla. En la partición vertical la disjuntura se aplica a los atributos de una relación que no forman la clave primaria.

1.1.6 ¿Cómo deben ubicarse los fragmentos?

La ubicación de datos a lo largo de una red de computadoras es un problema que ha sido estudiado extensivamente. Solo una pequeña proporción de los estudios realizados abarcan la distribución de datos de una BD (Buretta, 1997) (Carey and Livny., 1991) (Jim Gray et al., 1996). Asúmase que hay un conjunto de fragmentos $F = \{ F_1, F_2, ..., F_n \}$ y una red que consiste de los sitios $S = \{ S_1, S_2, ..., S_m \}$ en los cuales se va a ejecutar un conjunto $Q = \{ q_1, q_2, ..., q_q \}$ de consultas. El problema de asignación consiste en determinar la distribución *óptima* de F en S.

La optimización puede ser definida en función del costo mínimo, que consiste en evaluar cuánto costará en comunicación consultar y modificar los datos en todos los lugares donde aparezcan; o por el rendimiento, donde la estrategia de asignación se diseña para mantener una métrica de rendimiento, se evalúa en función del tiempo de respuesta y del rendimiento del sistema en cada sitio de la red.

1.1.7 Asignación

La asignación es el proceso mediante el cual se decide dónde se ubicarán los fragmentos obtenidos en la etapa de fragmentación. También se decide si se harán replicas de los mismos. Hacer réplicas tiene sentido por:

- Confiabilidad: Mayor seguridad ante pérdida de datos.
- Disponibilidad: Mayor tolerancia a fallos ante caídas de los sistemas de computadoras.
- Aumento del paralelismo: Mayor eficiencia en las consultas de lectura al posibilitarse su descomposición en subconsultas y la paralelización de éstas; pero presenta el

inconveniente de las consultas de escritura, que conllevan las actualizaciones de todas las copias de la red.

La ubicación de los datos es influenciada, además, por el esquema de replicación de información elegido. En conjunto con este concepto, las técnicas que preservan integridad en la información y que aseguran la mejor actualización de los datos también inciden en la definición del mejor esquema.

1.1.8 ¿Por qué usar BDD?

La cantidad de innovaciones tecnológicas que han surgido en los últimos años ha promovido un cambio en la forma de enfocar las aplicaciones computacionales. También el continuo descenso de los costos del hardware, en contraposición al incremento de la complejidad y costos del software a gran escala, han estimulado el interés en el desarrollo e implementación de los SBDD, los cuales han resuelto de manera sutil, la aparente dicotomía existente entre dos puntos de vista del procesamiento de datos: los Sistemas de Bases de Datos Centralizadas (SBDC) y la tecnología de redes de computadoras, representativas de integración y distribución respectivamente.

Existen varias razones técnicas para distribuir datos, en particular, los sistemas distribuidos se adaptan mejor a las necesidades de las organizaciones descentralizadas ya que reflejan más adecuadamente su estructura y tienen importantes ventajas con respecto a los centralizados. La descentralización se justifica desde el punto de vista tecnológico, ya que permite autonomía local y promueve la evolución de los sistemas y los cambios en los requerimientos de los usuarios, proporciona una arquitectura de sistemas simple, flexible y tolerante a fallos; además de ofrecer buenos rendimientos.

Otra razón a considerar a favor de la descentralización es la distribución de los accesos a la memoria, tanto de entrada como de salida, donde se almacena finalmente la información. Igualmente, las redes de computadoras trabajan cada vez a mayores velocidades, abriendo una puerta a la distribución del trabajo y la información (Őzsu and Valduriez, 1999), pero al mismo tiempo añaden nuevas complejidades a su diseño e implementación.

Como se puede observar, en el problema planteado en esta investigación, se tienen organizaciones geográficamente distribuidas (OCCM, DPFP y DNM), por lo que las vías centralizadas no son una alternativa factible y se impone la necesidad de la migración hacia

SBDD, para aprovechar la posibilidad que ofrecen las BDD de difundir datos sobre diferentes sitios de una red de computadoras y ejecutar aplicaciones que requieran el acceso a datos ubicados en más de un sitio. En este caso, cada una de las organizaciones (OCCM, DPFP y DNM) necesita mantener los datos que son relevantes para sus operaciones y las OCCM deben mantener su autonomía sobre los datos allí generados. En esta tesis se defiende el principio de que un sistema distribuido muestra la estructura de la BD como un espejo de la estructura de la organización, con lo cual se incrementa la localidad de referencia y se reduce drásticamente el tráfico en la red (Date, 2000).

1.2 Replicación de datos

La replicación de una BD es tradicionalmente vista como una forma para incrementar la disponibilidad y rendimiento de las BDD. Esta permite que ciertos datos de la BD sean almacenados en más de un sitio, aumentando así la disponibilidad de los datos y mejorando el funcionamiento de las consultas globales a la BD (Elmasri and Navathe, 2002).

En un entorno distribuido la replicación de datos puede darse de varias maneras. En el caso más extremo, con una replicación total de la BD, se mejora notablemente la disponibilidad de la información, dado que es altamente probable que algún sitio siempre permanezca activo. También se mejora el rendimiento de las consultas que se hacen sobre la BD, dado que cualquier sitio dispone de información necesaria. La desventaja de la replicación completa es que puede reducir drásticamente la rapidez de las operaciones de actualización, por el motivo que una sola actualización lógica se deberá ejecutar en todas y cada una de las copias de la BD.

El extremo opuesto al esquema de replicación total lo constituye la no replicación de los datos. Cada fragmento se almacena en un solo sitio. Esta solución conlleva ventajas y desventajas opuestas al caso anterior.

En general, estas soluciones no resultan aplicables como respuesta a un problema de BDD, salvo para situaciones muy puntuales. Por este motivo, se considera que una replicación parcial de datos es la mejor solución (Elmasri and Navathe, 2002). Esta forma de replicación tiene una amplia gama de soluciones, como por ejemplo que todos los datos se encuentren replicados al menos en dos sitios diferentes, o que los datos denominados “importantes” se encuentren en todos los sitios de la red, etc. Se denomina esquema de replicación al formato seleccionado.

1.2.1 Caracterización de la replicación de datos

La replicación de datos es 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 y es mucho más que la simple copia de datos entre varias localidades. La misma ha sido utilizada tradicionalmente como el mecanismo básico para incrementar la disponibilidad y el desempeño de una BDD (Kemme and Alonso, 2000).

La replicación debe estar acompañada del análisis, diseño, implementación, administración y monitoreo que garantice la consistencia de los datos a lo largo de múltiples administradores de recursos en ambientes distribuidos. Por este motivo, un servicio de replicación de datos debería proveer las siguientes funcionalidades (Őzsu and Valduriez, 1991):

- Ser escalable. Con respecto a la replicación, la escalabilidad significa la habilidad de replicar volúmenes de datos pequeños o grandes a lo largo de recursos heterogéneos (hardware, redes, sistemas operativos).

- Proveer transformación de datos y mapeo de servicios. Estos servicios permiten que los esquemas de datos diferentes coexistan sin perder su semántica. Por ejemplo, las copias pueden ser idénticas o semánticamente equivalentes. Las copias idénticas podrían tener la misma plataforma, el mismo contenido de información y el mismo tipo de datos; en tanto que copias semánticamente equivalentes podrían tener el mismo contenido de información pero diferentes plataformas y, posiblemente, diferentes tipos de datos.

- Soportar replicación en modo sincrónico (tiempo real) o asincrónico.

- Suministrar un mecanismo que describa los datos y objetos que se van a replicar (diccionario de datos).

- Facilitar un mecanismo para inicializar un sitio de procesamiento, esto es para indicar la recepción de datos replicados.

- Sustentar administración end-to-end de seguridad y calidad de servicios. Por ejemplo, el servicio debe garantizar que no ocurra corrupción en los datos durante la proceso de replicación. En otras palabras, los datos pueden cambiar de formato pero no de contenido.
- Mantener un mecanismo de bitácora que administre cualquier esfuerzo de replicación fallido.
- Tener un mecanismo de recuperación automático.

Un sistema de replicación es aquel sistema que soporta la réplica de datos, y debe caracterizarse principalmente por:

- Efectividad: depende de la forma en la que los datos sean distribuidos y almacenados. A mayor efectividad, mayor será la disponibilidad de datos para ejecutar procesos paralelos.
- Alta disponibilidad: es la razón de tiempo prudente en la que un servicio puede ser accedido. En el mejor de los casos puede ser de un 100%, a pesar de los fallos que se puedan presentar en el servidor, ya que debe existir un servidor adicional que posea alguna técnica de replicación que lo pueda suplantar en caso de ser necesario.
- Tolerancia a fallos: garantiza un comportamiento correcto, donde efectivamente puede existir un número finito de fallos y tipos de fallos.
- Bien coordinado: cada una de las partes que intervienen en la réplica de datos llegan a un consenso para realizar las invocaciones de los servicios a los objetos, que al final de la transacción debe realizarse tal y como fue solicitada. Para esto debe utilizar algún tipo de ordenamiento y así se evitan posibles conflictos durante el proceso.

La réplica de datos es 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.

1.2.2 Tecnologías de replicación

La replicación es una tecnología compleja y no puede existir una solución única para cubrir todas las necesidades. Se ofrecen diferentes tecnologías de replicación que se pueden adaptar y combinar para responder lo más fielmente posible a las necesidades de las aplicaciones. Cada tecnología tiene ventajas e inconvenientes. Los tres criterios principales para seleccionar una tecnología de replicación son: la coherencia de los datos replicados, la autonomía de los sitios, y el particionamiento de datos para evitar conflictos. Por la complejidad de garantizar estos criterios, no es posible optimizar los tres al mismo tiempo.

Coherencia de los datos replicados

La coherencia de las operaciones distribuidas, como la replicación, es mucho más complicada de mantener en comparación con la coherencia de las transacciones locales, por lo que es suficiente con respetar las propiedades ACAD (*Atomicidad, Coherencia, Aislamiento y Durabilidad*). Existen dos tipos principales de coherencia:

- Homogeneidad de las transacciones: La homogeneidad de las transacciones en la replicación obliga a que los datos sean idénticos en todos los sitios que participan en la replicación, como si la transacción fuera a ejecutarse sobre todos los sitios.
- Convergencia de los datos: La convergencia de los datos significa que todos los sitios que participan en la replicación deben tener el mismo juego de datos, que no es necesariamente el que se obtendría si todas las réplicas se hubieran efectuado sobre el mismo servidor.

Autonomía de los sitios

La autonomía de los sitios se mide por el impacto que tiene una operación efectuada sobre un sitio en el resto de los sitios. La autonomía es perfecta cuando un sitio puede evolucionar totalmente de manera libre e independiente. El sitio autónomo no se preocupa por las operaciones que pueden intervenir en los otros sitios. Por ejemplo, cuando la replicación garantiza la convergencia de los datos, la autonomía de los sitios está en su máxima expresión, ya que cada servidor de BD puede evolucionar libremente en relación a los otros sitios. A la inversa, la coherencia transaccional inmediata impone una autonomía casi nula de los sitios que participan en ella, ya que una transacción debe ser aprobada por todos los

servidores antes de ser validada. Si un solo servidor no puede, entonces la transacción no se valida en ninguno.

Particionamiento de los datos

Es posible repartir los datos sobre varios sitios con el objetivo de que cada sitio trabaje con sus propios datos, estrictamente distinto de los demás. De esta manera, las transacciones que intervienen sobre cada sitio sólo ponen en juego los datos del sitio y se conserva la coherencia global.

El particionamiento de los datos evita todo conflicto entre los datos y funciona con una coherencia de datos latente ya que cada sitio sólo modifica su propio juego de datos; la aplicación de esta coherencia es menos laboriosa que la coherencia transaccional inmediata que se basa en un proceso de validación en dos fases. Este tipo de particionamiento no se debe confundir con el particionamiento de tablas. En el marco de la replicación, se define una partición lógica, mientras que en el marco de una tabla particionada, se define una partición física de la tabla.

1.2.3 Metodología de replicación de datos

La tendencia actual consiste en migrar la BDC hacia un ambiente distribuido. Es importante seguir una metodología correcta de diseño de una BDD para poder obtener un esquema que sea escalable y adaptable a los cambios de tecnología. Cuando se distribuyen datos, el objetivo es crear una BDD que resida en varias localidades, de manera que se adapte de la mejor forma posible para cubrir las necesidades del problema a resolver. Estas necesidades pueden girar en torno al desempeño del sistema, la disponibilidad, seguridad e integridad de los datos, la tolerancia a fallos, entre otras. De esta manera, la forma de distribución de datos debe ser analizada cuidadosamente buscando una opción que optimice y maximice la utilización todos los factores que se deban considerar para un problema particular. El factor principal para la determinación de la distribución de datos va a estar dado, básicamente, por los requerimientos de usuario (Sommerville, 2002). Cuestiones, como por ejemplo el desempeño del sistema, pueden llevar a generar mayor replicación en los datos, a fin de colocarlos “cerca” del usuario, evitando el costo de la transmisión en la red. Obviamente, esta apreciación puede tornarse rápidamente en un inconveniente. Si la replicación de datos

aumenta, los protocolos de compromiso van a demorar más en completar su ejecución, o mantener las copias actualizadas “en línea” resulta más costoso.

Se puede obtener replicación de datos considerando diferentes criterios. Uno de ellos, quizá el más simple, consiste en clasificar la replicación por el costo de latencia que existe para lograr la consistencia de los datos a lo largo de todas las réplicas. De esta manera se puede hablar de dos tipos de actualización de replicación: sincrónica o asincrónica. La replicación sincrónica provee un mecanismo que asegura que todas las réplicas se mantengan actualizadas en línea. De esta forma, la latencia para lograr consistencia de datos se reduce a cero. La implementación clásica del protocolo de compromiso de dos fases (2PC) garantiza este tipo de replicación (Elmasri and Navathe, 2002). Con este esquema de actualización de réplicas es posible garantizar las propiedades ACAD de una transacción. Por el contrario, la replicación asincrónica se limita a asentar las modificaciones en una copia o réplica, dejando para más adelante la actualización del resto. De esta forma, un esquema con estas características debe aceptar inconsistencias temporales en los datos.

1.2.3.1 Esquema de actualización sincrónica de replicación

Un esquema de actualización *sincrónica* de replicación mantiene todas las copias exactamente sincronizadas en todas las localidades modificando todas las réplicas como parte de una transacción atómica (Jim Gray et al., 1996). Éste permite una ejecución serializable, debido a que no se presentan anomalías de concurrencia. En contraste, este esquema reduce el desempeño de una actualización e incrementa el tiempo de respuesta de una transacción porque la acción determinada por esta debe resolverse en múltiples sitios, requiriendo mayor cantidad de mensajes. Las actualizaciones se aplican a todas las réplicas de un objeto de datos como parte de la transacción original. No se presentan inconsistencias debido a que no se presentan anomalías de serialización y no es necesaria la reconciliación de los datos ya que todos permanecen iguales. Los bloqueos sobre los datos permiten detectar anomalías potenciales y las convierten estas situaciones en una espera que se aplica sobre la transacción o directamente en un caso de deadlock o fallos en las transacciones, para abortar dicha transacción y salvar la anomalía.

La utilización de este esquema permite, en todo momento, que cualquier lectura sobre una copia del dato obtenga como resultado una información correcta. En el problema de esta investigación no necesariamente todos los sitios de la red tendrán una conexión permanente con la misma. En este caso, un esquema *sincrónico* no es viable, dada la imposibilidad de actualizar una copia de datos que reside en un sitio que se encuentra desconectado.

Un criterio de corrección aplicable sobre el esquema *sincrónico* se denomina seriabilidad de una copia (1CSR) (Bernstein et al., 1987), o equivalencia de una copia (Őzsu and Valduriez, 1996, Őzsu and Valduriez, 1999) que afirma que el valor de todas las copias de un objeto de dato deben ser idénticas cuando finaliza la transacción que la modifica. El protocolo típico de control de réplica que permite obtener este criterio se denomina Read-One/Write-All (ROWA). Este protocolo es simple y directo, pero requiere que todas las copias de los datos que debe actualizar la transacción se encuentren disponibles; esto es, si un sitio conteniendo alguna copia falla, la transacción será bloqueada hasta que ese sitio se recupere. De esta forma se reduce la disponibilidad de la BD.

Resumiendo, el esquema de actualización de replicación sincrónica proporciona las siguientes ventajas (Len, 2001):

- Serialidad: Debido a que todas las actualizaciones dentro del proceso de replicación ocurren como una unidad lógica de trabajo, las transacciones exhiben lo que comúnmente se llaman propiedades ACAD, generando, de esta forma, ejecución serializable, sin inconsistencias de ningún tipo.

- Consistencia firme: Esta segunda ventaja se desprende de la anterior. La actualización original se demora hasta que todas las copias se actualicen, originando una transacción global. Así, cuando una transacción termina, todas las copias tienen el mismo valor, logrando que sea cero la latencia antes que se logre la consistencia de los datos.

Sin embargo, estas ventajas tienen los siguientes costos:

- Degradación del desempeño: debido a que se necesita sincronizar cada acceso a datos con las demás réplicas durante la ejecución de la transacción, lo convierten en un esquema muy costoso.

- Mayores tiempos de respuesta: resulta un corolario del caso anterior. Esta sobrecarga de mensajes tiende a reducir el desempeño de actualización e incrementar los tiempos

de respuestas transaccionales, no pudiendo darle una respuesta al usuario hasta que la actualización haya sido cometida por completo.

- Difícilmente escalable: la mayor desventaja es que el número de operaciones por transacción aumenta con el grado de replicación de los datos (O(N) donde N es el número de sitios), y debido a que la probabilidad de deadlock crece muy rápidamente en proporción con el tamaño de la transacción y con el número de sitios, resulta en un esquema inseguro para escalar más allá de un pequeño número de sitios.
- Inapropiado para ambientes móviles: por un lado reducen el desempeño de las actualizaciones con sincronizaciones extras, y por el otro, en un ambiente móvil los sitios están normalmente desconectados; de aquí que una actualización sincronizada no sea válida para este tipo de ambientes.

1.2.3.2 Esquema de actualización asincrónica de replicación

Un esquema de actualización a*sincrónica* de replicación permite que una réplica sea actualizada por la transacción original. Las actualizaciones de las restantes se propagan asincrónicamente, generalmente con una transacción para cada sitio o localidad. La *actualización asincrónica* de réplicas proporciona las siguientes ventajas:

- Mejor desempeño: el tamaño de las transacciones es reducido, sólo se limitan a actualizar la copia primaria de datos y, por consiguiente, son menores los bloqueos necesarios. Se mejora, de esta forma, el desempeño global del sistema.
- Mayor facilidad de escalabilidad: una conclusión del caso anterior, el número de deadlock es menor que en los esquemas sincrónicos debido a que las transacciones son más cortas. Sin embargo, cada actualización genera N-1 transacciones para actualizar las demás réplicas, esto lleva a que el número de transacciones concurrentes aumente en O(N) (donde N es el número de sitios).
- Aptos para ambientes móviles: estos esquemas dan una respuesta inmediata al usuario, los cambios de la propagación son realizados luego; este tipo de respuesta es de gran importancia dada la proliferación de aplicaciones para usuarios móviles, donde una copia no está siempre conectada al resto del sistema y no tiene sentido esperar hasta que las

actualizaciones se hayan cometido para permitir al usuario ver los cambios. Los algoritmos basados en este tipo de esquemas son una buena opción para aplicaciones móviles, ajustándose a las necesidades de estos ambientes, donde la sincronización ocurre después que la transacción de actualización se compromete. (Jim Gray et al., 1996)

Por otro lado, hay que tener en cuenta las desventajas surgidas con este tipo de esquemas:

- Consistencia débil: debido a que cada transacción se ejecuta localmente e independientemente, el sistema no requiere protocolos de compromiso. Consecuentemente, este esquema debilita la propiedad de consistencia mutua y proporciona una consistencia más débil, en la cual la latencia antes de lograr la consistencia de los datos es siempre mayor que cero. Como conclusión, siempre existe algún grado de retraso entre el tiempo de comprometer la copia origen y el tiempo para disponerla en las demás réplicas, permitiendo que las copias puedan divergir y se produzcan posibles inconsistencias en los datos.

- Acceso a "datos viejos": surge como consecuencia de la consistencia débil. Existe la posibilidad de que la propagación asincrónica pueda causar que una transacción de actualización obtenga valores viejos de algún dato, resultando una ejecución que genera un estado inconsistente de la base de datos.

Además de los aportes realizados por productos comerciales, se ha realizado un trabajo considerable con el fin de desarrollar estrategias de replicación *asincrónica*. Algunos modelos propuestos implementan consistencia débil de datos (Pu and Leff, 1991) (Krishnakumar and Bernstein, 1991), otros presentan paradigmas que resultan en soluciones económicas en términos de desempeño (Chen and Pu, 1992) (Sidell et al., 1996) o estrategias que se podrían denominar epidémicas (Agrawal and Abbadi, 1997). Las soluciones más recientes desarrollan soluciones *asincrónicas* que garantizan seriabilidad (Pacitti et al., 1999) (Breitbart et al., 1999). Todos se basan en aproximaciones de copia primaria, esquemas maestro-esclavo (master-slave). La idea básica es restringir la ubicación de las copias primarias y secundarias y, de esta forma, controlar el orden en que se aplican las actualizaciones. Estas soluciones pueden proveer una gran respuesta en tiempo (no se requiere intercambio de mensajes durante la ejecución de la transacción) y se garantiza

seriabilidad. El costo que se paga es que las posibles configuraciones del sistema y la forma de ejecutar transacciones quedan muy restringidas.

1.3 Herramientas de replicación de datos

PostgreSQL, por sus características, es el SGBD libre seleccionado por el Cuba para migrar las soluciones de BD a software libre. La réplica en este gestor se puede lograr a través de diferentes mecanismos para tales fines. Sin embargo, resulta difícil seleccionar un mecanismo de replicación para un propósito general, capaz de ser puesto en marcha en cualquier ambiente de replicación. Por tal motivo en esta investigación se abordan los mecanismos de replicación libres que soporta PostgreSQL como una de sus fuentes de datos.

1.3.1 Slony-I

Es un sistema de replicación "maestro a múltiples esclavos" que soporta la réplica en cascada y permite a un esclavo ser a su vez maestro para otro servidor. Incluye los tipos de funcionalidades necesarias para replicar BDs grandes a un número razonablemente limitado de sistemas esclavos. En este contexto, razonable es un número no superior a 10 servidores. Si el número de servidores crece más allá de 10, el costo de las comunicaciones aumentará prohibitivamente, y las ventajas incrementales de tener servidores múltiples fallarán en ese punto.

Además, permite indicar qué cambios replicar de un servidor a otro. Slony-I implementa la réplica asincrónica, usando disparadores para determinar las actualizaciones de las tablas, donde un solo origen (maestro) se puede replegar a los suscriptores múltiples (esclavos) incluyendo suscriptores conectados en cascada. Slony I realiza una réplica de espejos, exactamente igual al origen de datos, no es posible actualizar los datos a medida que se produce algún cambio en ellos.

1.3.2 Pgpool

Es un programa intermediario (middleware) que trabaja entre PostgreSQL y servidores de una BD PostgreSQL cliente. Trabaja entre los servidores de datos y el cliente para reducir el

consumo (overhead) de establecimiento de la conexión. Este trabaja hasta con dos servidores, si el principal cae automáticamente trabaja con el otro. Permite la replicación ya que puede gestionar múltiples servidores PostgreSQL, el uso de la función de replicación en tiempo real permite crear una copia de seguridad en dos o más discos físicos, por lo que el servicio puede continuar sin parar los servidores en caso de un fallo de disco. Existe un límite en el número máximo de conexiones simultáneas con PostgreSQL, y las conexiones se rechazan después de esta cantidad.

1.3.3 PgCluster

Es un mecanismo de replicación sincrónico y multimaestro. Su replicación es física, lo que significa que para replicar utiliza los archivos del gestor y no los objetos lógicos del mismo. Presenta un servidor para balancear la carga de las peticiones y permitir que la replicación se realice lo más rápido posible. Tiene un mecanismo de detección de errores que permite aislar a los servidores que tengan problema de conexión y continuar con el mecanismo de replicación sin afectar a los demás servidores.
Este sistema permite que la replicación y las BDs siempre estén disponibles aunque existan fallos. Emplea un sistema de recuperación mediante el cual al iniciarse una BD en este modo recupera todos los datos de la BD maestra y al terminar este proceso satisfactoriamente es que se incluye en el sistema de replicación. Si existen fallos en las BDs se pueden recuperar los datos, al menos, de una copia en buen estado. Cuenta con la desventaja de que la replicación es sincrónica; si la conexión falla pueden ocurrir errores. Además, la replicación es de forma física por lo que pueden ocurrir errores en la trasferencia de archivos.

1.3.4 Pyreplica

Es un mecanismo asincrónico, multiplataforma y desarrollado en Python. Es de fácil instalación y administración aunque se realiza a través de comandos en una terminal. Permite detención de conflictos y replicación maestro-esclavo y multimaestro. Posee la desventaja de no soportar objetos grandes y de no contar con una herramienta gráfica para su administración.

1.3.5 DBReplicator

Es una suite de software y API diseñada para realizar la replicación asincrónica. Está desarrollada en Java. Soporta múltiples gestores como fuentes de datos, y puede ser usado a través de líneas de comandos o una interfaz gráfica. La interfaz gráfica es de escritorio y posee algunas limitaciones con respecto al manejo de las llaves primarias y las restricciones entre BDs heterogéneas. La actualización manual puede afectar el funcionamiento de la réplica, y existen problemas con las columnas auto-incrementales que son llaves primarias.

1.3.6 SymmetricDS

Es un software de sincronización de datos independiente de la BD y pensado para su uso en Internet. Utiliza tecnologías Web y tecnologías de BD para replicar tablas entre BDs relacionales casi en tiempo real. Este software ha sido diseñado para poder escalarse a un gran número de BDs, trabajar sobre conexiones de pequeño ancho de banda, y resistir períodos de inoperatividad de la red. Se instala o bien de modo autónomo, como una aplicación Web dentro del servidor de aplicaciones Java, o puede ser incorporado a otra aplicación Java.
SymmetricDs está escrito en Java 5 y require Java SE Runtime Environment (JRE) o Java SE Development Kit (JDK) version 5.0 o superior. Soporta la sincronización entre diferentes plataformas de base de datos mediante el concepto de dialectos de BDs. Un dialecto de BD es una capa de abstracción con la cual interactúa SymmetricDS para aislar la lógica de sincronización de los detalles de implementación específicos de cada base de datos.

1.4 Consideraciones parciales

Las BDD son cada vez más usadas por las empresas y suponen una ventaja competitiva frente a los sistemas centralizados, siempre y cuando la empresa en cuestión tenga necesidad de usar una BD de este tipo. Lo más habitual es disponer de varias sedes y tener que manejar información común, para lo cual las BDD son especialmente útiles. Tanto la fragmentación como la asignación requieren variada información acerca de los esquemas de

la BD, las aplicaciones, los sitios y las redes de comunicación, pero cada cual ignora cómo la otra captura y usa esta información.

Debido a esta tendencia a la descentralización, el interés por el uso de la replicación para la distribución de datos ha ido creciendo ya que se necesita acceder de forma coherente a los mismos datos como si estuvieran centralizados, y se trabaja con grandes volúmenes de datos que no pueden ser fácilmente recuperados o restituidos en caso de fallas. La replicación permite compartir información entre BD y plataformas heterogéneas, y modificar y reconciliar esa información; también garantiza la disponibilidad de los datos correctos cuándo y dónde se precisen.

La selección del mecanismo de replicación adecuado para poder configurar un ambiente de replicación genérico es complejo, pues cada mecanismo posee características enfocadas a resolver un problema en específico. Sin embargo, se pueden clasificar los mecanismos antes mencionados teniendo en cuenta un grupo de características que facilitan la implantación de cualquier solución de réplica.

Según el estudio realizado, se propone como herramienta de replicación SymmetricDS por su simplicidad y capacidad para realizar la réplica en ambos entornos, además de su flexibilidad y sencillez. Su facilidad para instalarlo, administrarlo y adaptarlo son otros aspectos a tener en cuenta también para su selección. Resulta eficiente al causar bajo impacto en la utilización de la memoria y la red. Es un software de replicación de datos asíncrona que permite subscriptores múltiples y sincronización bidireccional. Por la facilidad para conectarse con PostgreSQL mediante la utilidad psycopg2, además de ser de código abierto, publicado bajo la licencia GNU/LGPL y de su capacidad para desempeñarse eficientemente en múltiples plataformas. Está desarrollado en Java que es el lenguaje de programación seleccionado para desarrollar el sistema automatizado para el control y cobro de multas, por lo que no se requeriría de un cambio de plataforma.

CAPÍTULO 2: MODELO DE REPLICACIÓN DE DATOS

En este capítulo se presenta el modelo de replicación de datos para el Sistema de Control y Cobro de Multas que garantiza que todos los servidores que se encuentran en las capitales provinciales (DPFP) contengan los datos de sus municipios, las multas impuestas en las OCCM, y a su vez en el servidor nacional se registren todas las multas impuestas en el país.

2.1 Proceso de control y cobro de multas

Las OCCM son las encargadas de efectuar las gestiones de cobro sobre:

a) Multas por contravenciones personales.
b) Multas impuestas por la PNR, la Fiscalía General de la República y las fiscalías provinciales y municipales a tenor de lo establecido en el inciso 3 del artículo 8 del Código Penal.
c) Multas impuestas por los tribunales populares de acuerdo con lo establecido en el Código Penal.
d) Convenios de pagos aplazados de multas por contravenciones personales o pagos a plazos acordados por los tribunales populares en el caso de multas impuestas al amparo del Código Penal
e) Multas Institucionales que se impongan mediante boletas.

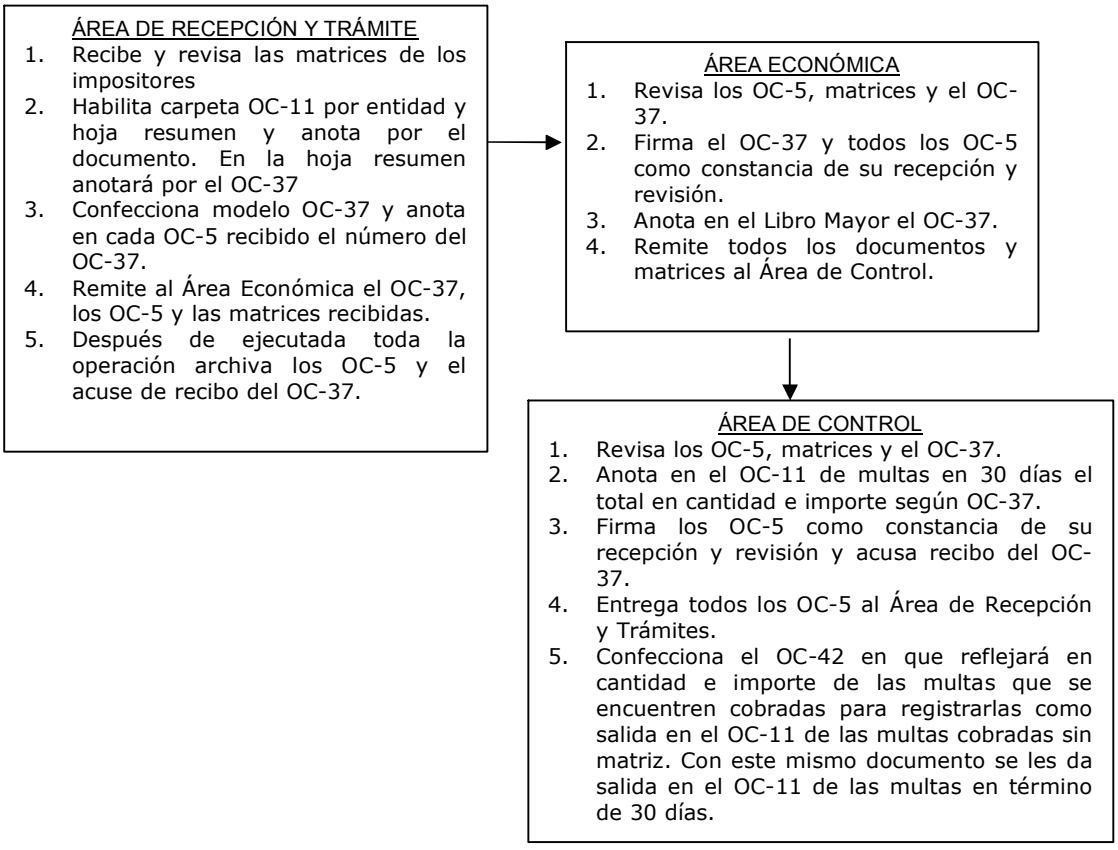
En las OCCM no se gestionan pero se efectúa el cobro de las multas impuestas a tenor del Código de Vialidad y Tránsito y las multas Fiscales. Las oficinas tienen la responsabilidad de aplicar con rigor y agilidad los procedimientos de cobros de que disponen, de forma tal que su gestión comienza desde que se reciben las matrices de multas remitidas por los órganos y organismos del Estado facultados para imponerlas hasta su cobro, envío a otro municipio o cancelación, según proceda y garantizar con ello que no se pierda el principio de educar al pueblo en el respeto de la Ley. Los importes cobrados en las OCCM son ingresados al presupuesto del municipio correspondiente.

Están establecidos términos para efectuar los traslados a otras OCCM, la cancelación de las multas impuestas y la duplicación del valor de las multas por contravenciones personales, entre otros.

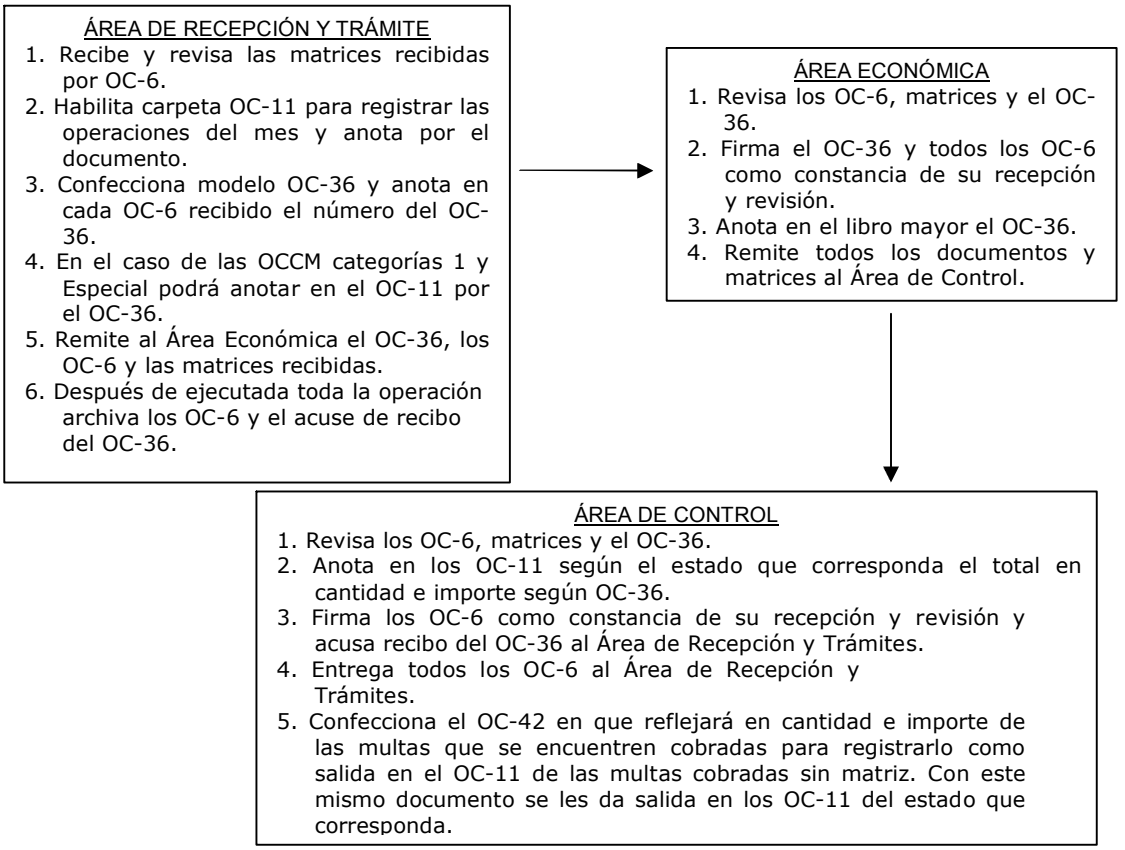
2.1.1 Procedimientos para las diferentes operaciones que se realizan en las OCCM

A continuación se detallan los procedimientos para realizar las diferentes operaciones que se realizan en las OCCM como son: multas recibidas de las entidades impositores (véase el procedimiento 1), multas por traslados (véase el procedimiento 2), traslado (véase el procedimiento 3) y devolución (véase el procedimiento 4) de matrices, cobros (véase el procedimiento 5) y otros procedimientos (véase el procedimiento 6).

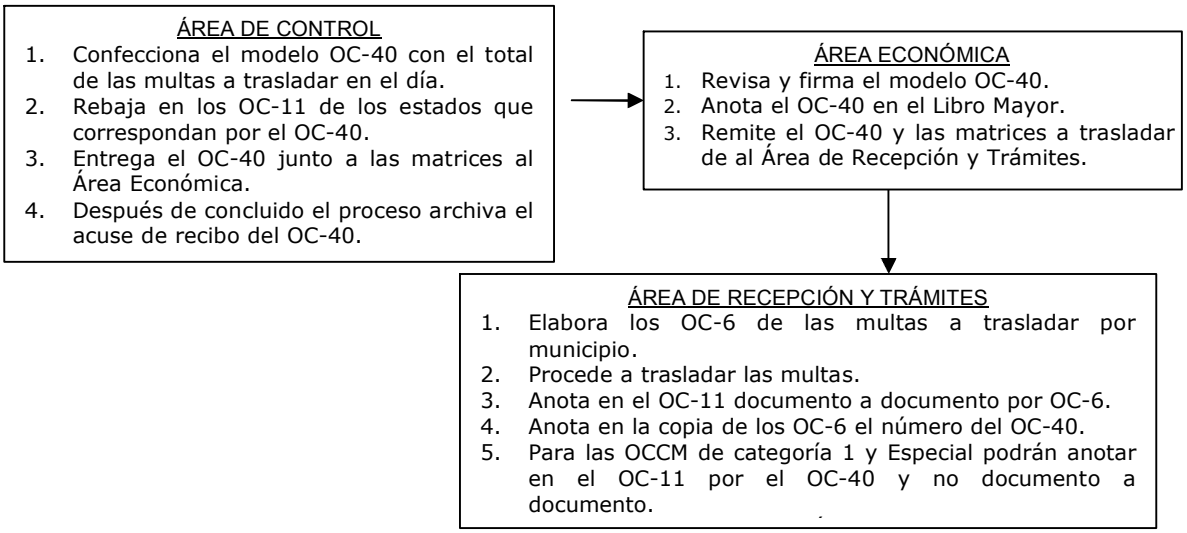
Procedimiento 1: Multas recibidas de las entidades impositoras



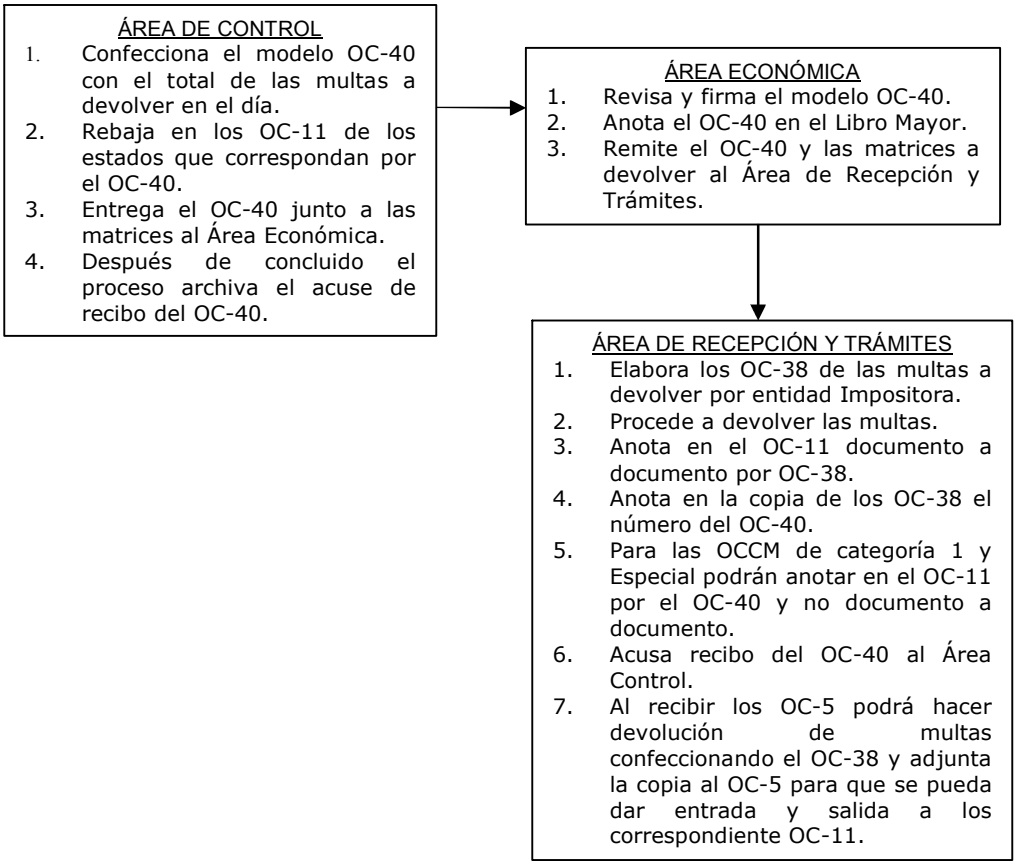
Procedimiento 2: Multas recibidas por traslados



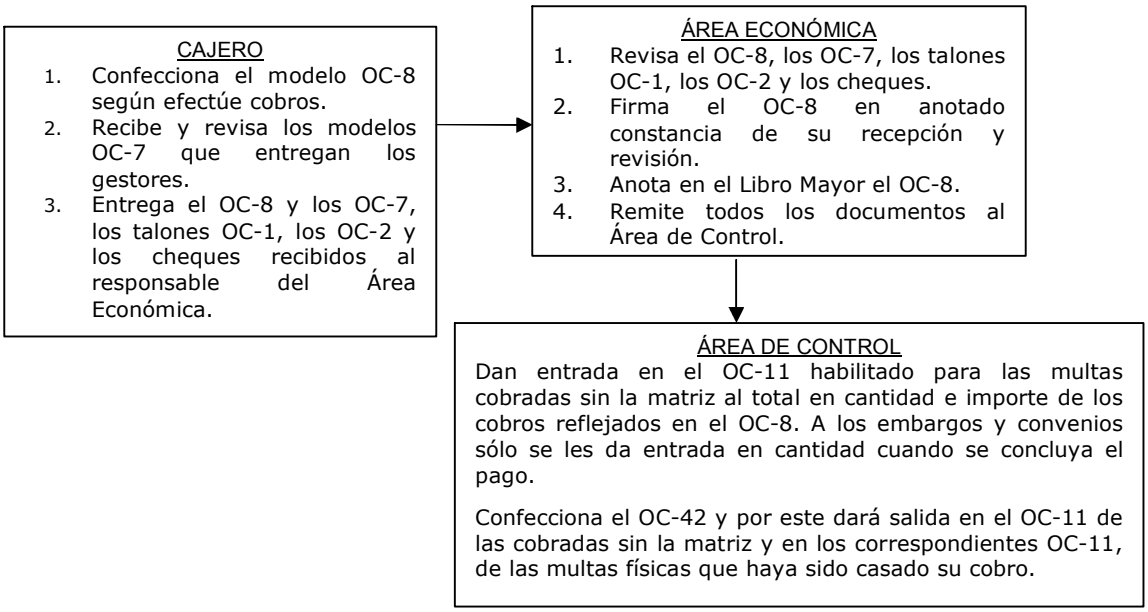
Procedimiento 3: Traslado de matrices



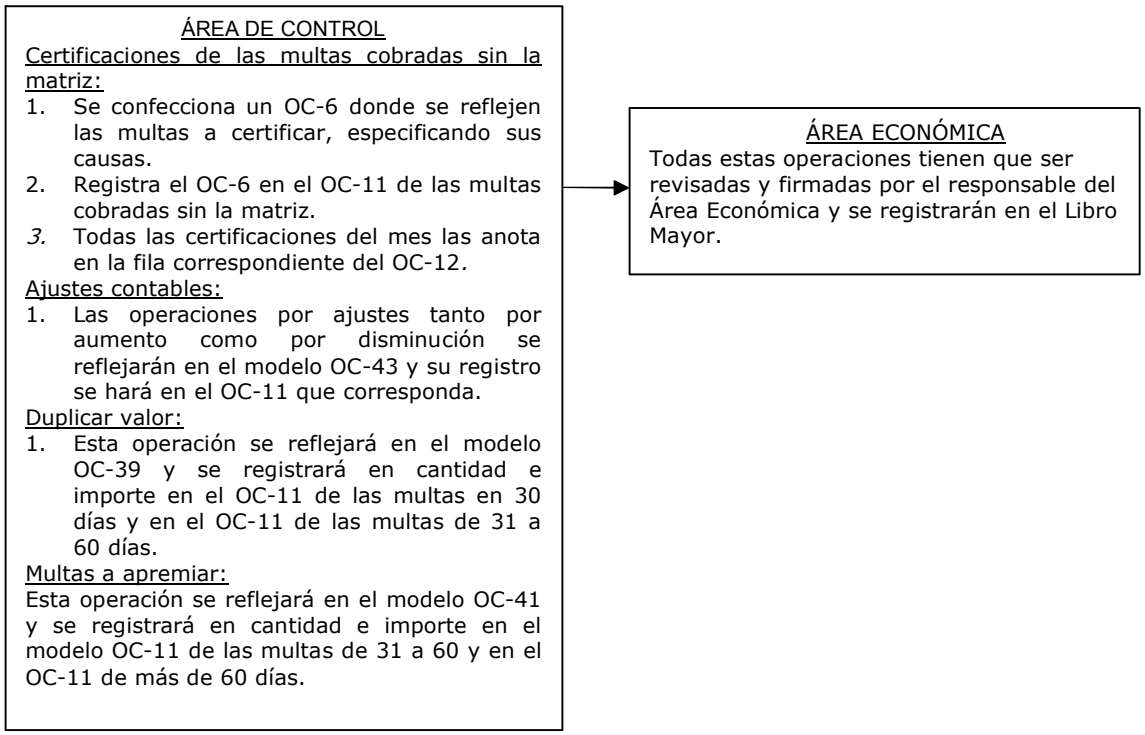
Procedimiento 4: Devolución de matrices



Procedimiento 5: Cobros



Procedimiento 6: Otros procedimientos



2.1.2 Objetivo, funciones y procedimiento de las estafetas provinciales.

El objetivo de las estafetas provinciales es centralizar, a través de las estafetas de las DPFP, la responsabilidad de recepcionar y enviar los traslados de multas en el país, verificando que estos cumplan con los requerimientos establecidos en el Sistema de Multas, Control y Cobro, desde cada provincia hacia sus respectivas OCCM y desde cada provincia hacia el resto de las provincias del país y desde estas hacia sus respectivas OCCM.

Las estafetas tienen dentro de sus funciones garantizar la adecuada comprobación y supervisión de los traslados y devoluciones de multas a las entidades, la rapidez de las operaciones, seguridad y control además de las siguientes:

1. Recepciona de las OCCM y demás provincias del país las matrices de multas que son objeto de traslados con sus respectivos OC-6 y que se ajusten a los conceptos siguientes:

- Multas cuyos contraventores no son residentes en el municipio donde se impuso la multa y que su pago no se realizó dentro del término establecido.
- Multas que tramitadas con la Oficina de Identificaciones y Registros del Ministerio del Interior, les certifican una nueva dirección, fuera del municipio.
- Multas impuestas por los Tribunales Municipales Populares y Provinciales, de una provincia o municipio que no es el de residencia del sancionado y que su comunicación por impago se realiza al Tribunal Popular Municipal o Provincial que impuso la sanción. En este caso el Expediente Único para la comunicación al tribunal deberá acompañar a la matriz con todos los documentos establecidos.
- Multas que se cobraron, por causas que autoriza el Sistema de Multas, Control y Cobro, en el municipio en que fueron impuestas y que fue trasladada al municipio de residencia del contraventor, comprobándose mediante la gestión el cobro realizado. Este traslado debe contener claramente en el OC-6 los datos del comprobante de pago, la fecha del cobro y el importe.
- Multas que por los conceptos autorizados en el sistema les son devueltas a los órganos u organismos que las impusieron, previa aprobación del jefe de la Estafeta Provincial.
- Cuando se traslade una multa recibida de organismo y que el contraventor no sea residente en el municipio donde se recepcionó la multa, el tiempo máximo para trasladar esta multa es 5 días posterior a la fecha de recepción; de trasladarse con posterioridad, por causas plenamente justificadas y no antes de 10 días de su recepción, junto al traslado deberá acompañarse una nota del jefe del Departamento de Multas de la provincia explicando las causas que motivaron se efectúe el traslado fuera de los términos establecidos. Si el jefe del Departamento de Multas provincial determina que el traslado tarde se hizo por responsabilidad de la OCCM exigirá la correspondiente responsabilidad material al jefe de la OCCM emisora y procederá a realizar el rechazo correspondiente.

2. Recepciona, revisa, asienta, comprueba y clasifica los traslados aceptados, así como los recibidos para después de concluido todo el proceso, distribuirlos en las correspondientes valijas y entregar las recibidas a las OCCM destinatarias.
3. Lleva los registros estadísticos habilitados por OCCM y provincia para los traslados y para las recibidas de forma que le permita comprobar que las trasladadas y las recibidas

hasta al cierre de cada mes coinciden con lo reflejado en el modelo OC-12 por cada una de las OCCM, en cantidad e importe.

4. La fecha de cierre para las estafetas provinciales se fija el último día de cada mes. Las OCCM seguirán trasladando de la forma que se establece en el Sistema de Multas.

2.1.3 Procedimientos por los que se rigen las OCCM y los organismos impositores

Las relaciones entre las OCCM y los organismos Impositores tienen tres niveles: municipal, provincial y central. Dichas relaciones podrán contactarse mediante acciones de trabajo, las cuales estarán delimitadas en función del nivel de subordinación.

A nivel de municipio se realizan las siguientes acciones:

1. Dentro de los primeros quince (15) días de cada mes, se realiza la conciliación entre el organismo impositor y la OCCM, donde ambas partes aportan los siguientes datos:

Por el organismo o entidad impositora:

1.1. Cantidad de multas anuladas, las que lleva a la conciliación físicamente para su debida comprobación por la OCCM y rebaja del saldo en OC-1.

1.2. El expediente de los modelos OC-5 para su confrontación, a modo de contrapartida, con los modelos OC-5 entregados en el período a la OCCM.

1.3. El organismo impositor está en el deber de dar seguimiento a las multas que dentro del pendiente de cobro no se ha hecho imposible cobrar, por no lograrse la ubicación del contraventor.

Por la OCCM:

1.4 Elabora el acta de conciliación en original y copia, original para el organismo y copia para la OCCM, en el cual se hace constar el saldo inicial en la entrega de talones OC-1, más los talones entregados en el mes, menos las multas impuestas, menos las multas anuladas, hasta llegar al saldo final con que termina el organismo en la existencia de talones, para analizar la posible necesidad de una nueva entrega de talones.

1.5 En el acta de conciliación se reflejan los problemas presentados en la calidad de la imposición, haciendo constar la cantidad de devoluciones y sus causas, así como el número de resoluciones emitidas para cancelar multas; esto se hace con el objetivo

de que el organismo cuente con elementos para el análisis con los inspectores de la calidad del trabajo.

1.6 Las OCCM incluyen la incidencia que tenga el organismo con que se concilia, en las multas pendientes en apremio que aún no se logra su cobro; específicamente los no localizables, para su posible ubicación.

2. Las OCCM concilian mensualmente las multas del Código Penal cobradas, canceladas y cobradas sin matriz, con la PNR y la Fiscalía.

A nivel provincial se realizan las siguientes acciones:

1. Bimestralmente, entre los días diez y veinte, la conciliación entre los organismos impositores provinciales y las DPFP, en la cual se deberá aportar por ambas partes los siguientes datos:

El organismo o entidad impositora provincial:

1.1 Cantidad de multas impuestas por su organismo en cada municipio, comparando con la información emitida por las OCCM.

1.2 Confronta las informaciones sobre las multas devueltas por mala confección y por Carné de Identidad negativo; multas canceladas por resolución del organismo impositor, y multas anuladas.

1.3 Valora las medidas tomadas con los municipios que mayor incidencia tienen en las devoluciones, entregas fuera de fecha y multas anuladas. También se entrega la plantilla actualizada de los cuerpos de inspección, así como la fuerza real facultada a imponer multas.

1.4 Informa la cantidad de multas no entregadas a las OCCM, por municipios, y las medidas pecuniarias tomadas.

La DPFP:

1.5 Informa la cantidad de multas impuestas por el organismo, por municipios.

1.6 Aporta los datos de las multas canceladas por resolución del organismo o entidad impositora, por cada municipio.

1.7 Informa las devoluciones realizadas por sus diferentes conceptos, así como cualquier problema presentado por el organismo o entidad impositora que fuera informado por una OCCM.

1.8 Informa el consumo de los talones de imposición y la existencia física, así como cualquier problema que se presente en los controles del mismo.

1.9 Informa el nivel excesivo que se presenta en las anulaciones de multas.

Se realizan a nivel central las siguientes acciones:

1. Trimestralmente la conciliación a nivel de ministerios, donde se confrontan las siguientes informaciones:

El organismo o entidad impositoras nacionales

1.1. Se confrontan las informaciones referidas a:

1.1.1. Multas impuestas por provincias.

1.1.2. Multas devueltas por mala confección y por Carné de Identidad negativo.

1.1.3. Multas canceladas por resolución del organismo impositor.

1.1.4. Multas anuladas.

1.1.5. Multas impuestas por inspector y valor medio.

1.1.6. Valoración y medidas tomadas con las provincias que mayor incidencia tienen en las devoluciones, entregas fuera de fecha y multas anuladas.

1.1.7. Informa la cantidad de multas no entregadas a las OCCM.

El Ministerio de Finanzas y Precios

1.2. Se confrontan las informaciones referidas a:

1.2.1. Multas impuestas por provincias,

1.2.2. Multas devueltas por mala confección,

1.2.3. Multas impuestas por Carné de Identidad negativo.

1.2.4. Multas canceladas por resolución del organismo impositor, por prescripción.

1.2.5. Multas anuladas.

1.3. Informa el consumo de talones de cobro y la existencia física, así como cualquier problema que se presente en los controles del mismo.

1.4. Analiza la situación de las conciliaciones en las provincias, y resalta el nivel excesivo que se pueda presentar en las anulaciones de multas.

1.5. En el acta de conciliación se reflejan los problemas presentados en la calidad de la imposición, haciendo constar las devoluciones, canceladas y la cantidad de multas no entregadas a las OCCM.

2. Chequea en las OCCM y DPFP, los controles realizados a los organismos y entidades impositoras y a las unidades municipales de la PNR y órganos provinciales y municipales de instrucción, sobre los documentos que amparan el cobro de las multas de tránsito, así como el cobro y cancelaciones de las multas del Código Penal.

2.2 Arquitectura de soporte a los datos del sistema

Este sistema debe gestionar toda la información del PCCM a nivel nacional donde las oficinas de multas no deben perder autonomía sobre sus datos. Por tanto, se decidió que los datos se deben almacenar de forma distribuida, con sitios de procesamiento en la DNM en las DPFP y en las OCCM del país. En la Figura 2.1 se muestra la arquitectura de red que da soporte a los datos del sistema.

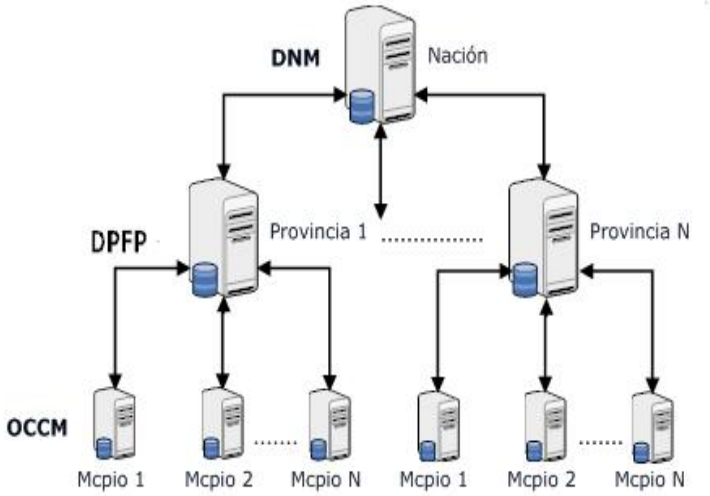


Figura 2.1 Arquitectura de soporte a los datos del sistema.

Como se puede apreciar, en cada municipio se ubicará un servidor en el que se almacenarán los datos de la OCCM de dicho municipio; cada municipio replicará sus datos a un servidor provincial ubicado en la DPFP a la que pertenece y cada provincia replicará sus datos a un servidor nacional ubicado en la DNM. De este modo, en cada DPFP se tendrán los datos de todas las OCCM de la provincia en cuestión y en la DNM se tendrán todos los datos del país. La replicación se realizará en ambos sentidos. Con esto se garantiza tener una BD nacional en la DNM y que el funcionamiento de las OCCM no dependa de la conectividad con la misma.

Para replicar datos de una OCCM a otra, estos primero pasan a la DPFP de la oficina origen, de aquí a la nacional y si es para una OCCM de esa provincia, esta misma los envía, de lo contrario de la oficina nacional pasa a la DPFP que le corresponde y de esta a la OCCM.

La replicación de los datos se realizará a través de una red de área extensa (WAN), mediante una línea dedicada; entre las OCCM y las DPFP los anchos de banda están entre 28 Kb/s, 128 Kb/s y 1 Mb y de las DPFP hacia la DNM. En todas las instancias, el SGBD que se utiliza es PostgreSQL.

El envío de información de los servidores municipales hacia los servidores provinciales se realiza cada 20 minutos y de los servidores provinciales al servidor nacional cada una hora, de igual forma en sentido inverso. Estos valores fueron definidos teniendo en cuenta los recursos de hardware con que cuentan las oficinas de multas en la actualidad y el volumen de datos que como promedio se produce en una OCCM diariamente. Pueden ser ajustados en la medida que mejoren dichos recursos o que aumente o disminuya el volumen de datos que se generen en las OCCM.

2.3 Modelo de replicación

El modelo de replicación que se seleccione debe garantizar la seguridad de la información y agilizar el tiempo de transferencia de los datos.

2.3.1 Componentes del Modelo de replicación

El modelo de replicación emplea la metáfora de la industria conocida como “Publicador – Suscriptor”. Este modelo consiste en Publicadores, Suscriptores y Distribuidores; las publicaciones y los artículos, y las suscripciones por inserción (*push*) o extracción (*pull*) y se compone de los siguientes objetos: el publicador, el distribuidor, el suscriptor, la publicación, el artículo y la suscripción.

La replicación de datos ocurre exclusivamente entre servidores de datos; en el caso de la presente investigación se trata de servidores. Los servidores pueden desempeñar uno o varios de los siguientes roles: publicador, distribuidor o suscriptor.

Publicador: Es el servidor que hace los datos disponibles para la réplica hacia otros servidores. Además de identificar los datos para su replicación, detecta los datos que han cambiado y mantiene la información sobre todas las publicaciones en el servidor. Este puede disponer de una o más publicaciones, de las cuales los suscriptores se suscriben a las publicaciones que necesitan, nunca a artículos individuales de una publicación, además,

detecta qué datos han cambiado durante la replicación y mantiene información acerca de todas las publicaciones del sitio.

Suscriptor: Son los servidores que almacenan las publicaciones y reciben actualizaciones. Un suscriptor puede, alternadamente, también publicar hacia otros suscriptores o devolver datos modificados al publicador. Los suscriptores reciben los datos replicados.

Distribuidor: Es un servidor que almacena la base de datos de distribución, datos históricos, transacciones y metadatos. La función del distribuidor varía según el método de replicación implementado. En ocasiones se configura el mismo publicador como distribuidor y se le denomina distribuidor local. En el resto de los casos el distribuidor será remoto, pudiendo coincidir en algún caso con un suscriptor. Para el modelo del presente trabajo será un distribuidor local.

Publicación: Es una colección de uno o más artículos que especifican un conjunto de datos u objetos relacionados lógicamente que se desean replicar conjuntamente. Un artículo puede ser una tabla completa, sólo ciertas columnas en caso de fragmentación vertical, o solamente ciertas filas para la fragmentación horizontal.

En la figura 2.2 se pueden observar los componentes elegidos para el diseño del modelo de replicación para el SCCM.

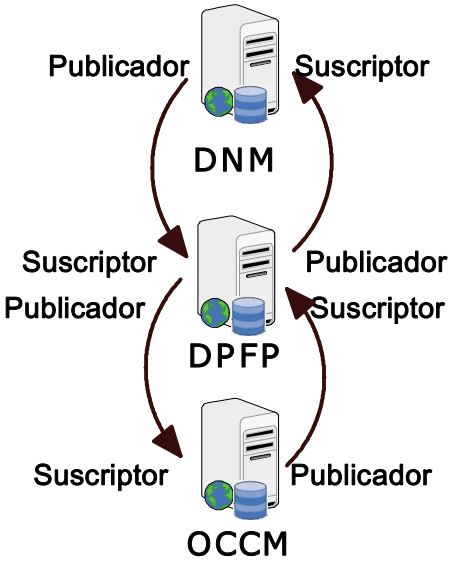


Figura 2.2 Componentes del modelo de replicación para el SCCM.

2.3.2 Escenarios típicos de la replicación

En una solución de replicación pudiera ser necesario utilizar varias publicaciones en una combinación de métodos y opciones. En la replicación los datos o transacciones fluyen del publicador al suscriptor pasando por el distribuidor. Por lo tanto, en su configuración mínima, una topología de replicación se compone de al menos dos o tres servidores que desempeñan los tres roles mencionados. Según la ubicación del servidor distribuidor se pudiera contar con las siguientes variantes:

1. El rol de distribuidor desempeñado por el publicador.
2. El rol de distribuidor desempeñado por el suscriptor.
3. Un servidor de distribución, independiente del publicador y del suscriptor.

En el modelo de esta tesis se utiliza la variante 1, donde el publicador actúa como distribuidor (véase la figura 2.2).

Figura 2.2 Rol de distribuidor desempeñado por el publicador

En la mayoría de las configuraciones, el peso fundamental de la replicación recae sobre el servidor de distribución. Por tanto, éste puede ser un criterio para determinar su ubicación, teniendo en cuenta las configuraciones o posibilidades físicas de los servidores, así como otras responsabilidades que pueden estar desempeñando como por ejemplo: servidor de dominio, servidor de páginas web entre otras. En el caso de esta tesis, los servidores fungirán como servidores Web y de BD.

Existe la posibilidad de contar con un servidor que se suscriba a una publicación y a la vez la publique para el resto de los suscriptores, esto puede ser muy útil cuando se cuente con una conexión muy costosa con el publicador principal. Por ejemplo, el publicador principal en las DPFP y los suscriptores en la OCCM. En casos como este se puede elegir un suscriptor, dígase el servidor de la DPFP, el cual se suscribe al publicador en la OCCM y a la vez actúa como servidor de publicación para el servidor de la DNM. A su vez, el servidor de la DNM

sirve de publicador a los servidores de las DPFP y estos a las OCCM. De este modo, los servidores de las OCCM realizan la función de publicadores, mientras que los de las DPFP y de la DNM fungen como publicador/suscriptor/distribuidor.

2.3.3 Tipos de replicación

Existen tres tipos básicos de replicación: instantánea, transaccional y de mezcla. El empleo de cada uno de estos responde a una necesidad bien concreta.

En la replicación instantánea los datos se copian exactamente tal y como aparecen en un momento determinado. Por consiguiente, no se requiere de un control continuo de los cambios. En este tipo de publicaciones se suelen replicar los objetos con menos frecuencia que en otros; puede llevar más tiempo propagar las modificaciones de datos a los suscriptores. En general, se recomienda utilizar la misma ante alguna de las siguientes situaciones: cuando la mayoría de los datos no cambian con frecuencia; se replican pequeñas cantidades de datos; los sitios con frecuencia están desconectados y es aceptable que la latencia o período de tiempo que transcurre entre la actualización de los datos en un sitio y en otro sea largo. En ocasiones se hace necesario utilizarla cuando están involucrados algunos tipos de datos predefinidos por los SGBD como text, ntext, e image, cuyas modificaciones no se registran en la bitácora de transacciones y por tanto no se pueden replicar utilizando el método de replicación transaccional.

Con la opción de actualización inmediata en el suscriptor le permite a los suscriptores actualizar datos solamente si el publicador los va a aceptar inmediatamente. Si el publicador los acepta, se propagan a otros suscriptores. El suscriptor debe estar conectado de forma estable y continua al publicador para poder realizar cambios en el suscriptor. Esta opción es útil en escenarios en los que tienen lugar unas cuantas modificaciones ocasionales en los servidores suscriptor.

En el caso de la replicación transaccional se propaga una instantánea inicial de datos a los suscriptores, y después, cuando se efectúan las modificaciones en el publicador, las transacciones individuales se propagan a los suscriptores. Se almacenan las transacciones que afectan a los objetos replicados y se propagan esos cambios a los suscriptores de forma continua o a intervalos programados. Al finalizar la propagación de los cambios, todos los suscriptores tendrán los mismos valores que el publicador. Esta suele utilizarse cuando: se desea que las modificaciones de datos se propaguen a los suscriptores, normalmente

pocos segundos después de producirse; se necesita que las transacciones sean atómicas, es decir que se apliquen todas o ninguna al suscriptor; los suscriptores se conectan en su mayoría al publicador; su aplicación no puede permitir un período de latencia largo para los suscriptores que reciban cambios.

Este tipo de replicación es útil en escenarios en los que los suscriptores pueden tratar a sus datos como de sólo lectura, pero necesitan cambios a los datos con una cantidad mínima de latencia. Con el uso de la opción de actualización inmediata en el suscriptor se pierde aún más la autonomía de sitio, pero se reduce el tiempo en el cual los sitios actualizan sus copias de los datos. Para hacer modificaciones en la BD del suscriptor, estas se realizan también en la BD publicador bajo el protocolo de compromiso en dos fases (2PC) por lo que si su modificación se compromete indica que es válida y luego, en cuestión de minutos o según la planificación hecha, estos cambios son duplicados a las demás BDs suscriptoras.

La replicación de mezcla proporciona el nivel más alto de la autonomía para cualquier solución de réplica. Los publicadores y los suscriptores pueden trabajar independientemente y volverse a conectar periódicamente para combinar sus resultados. Si se crea un conflicto por los cambios que se hacen al mismo elemento de datos en los sitios múltiples, esos cambios serán resueltos automáticamente. Esta permite que varios sitios funcionen en línea o desconectados de manera autónoma, y mezclar más adelante las modificaciones de datos realizadas en un resultado único y uniforme. La instantánea inicial se aplica a los suscriptores; a continuación se hace un seguimiento de los cambios realizados en los datos publicados en el publicador y en los suscriptores. Los datos se sincronizan entre los servidores a una hora programada o a petición. Las actualizaciones se realizan de manera independiente, sin protocolo de confirmación, en más de un servidor, así el publicador o más de un suscriptor pueden haber actualizado los mismos datos. Por lo tanto, pueden producirse conflictos al mezclar las modificaciones de datos. Es útil cuando: varios suscriptores necesitan actualizar datos en diferentes ocasiones y propagar los cambios al publicador y a otros suscriptores; cada sitio hace cambios solamente en sus datos pero necesitan tener la información de los otros sitios; los suscriptores necesitan recibir datos, realizar cambios sin conexión y sincronizar más adelante los cambios con el publicador y otros suscriptores; el requisito de periodo de latencia de la aplicación es largo o corto; la autonomía del sitio es un factor crucial.

Para seleccionar el tipo de replicación a utilizar para el modelo del presente trabajo se tuvieron en cuenta tanto factores relacionados con los requerimientos de la aplicación como

factores relacionados con el entorno de red. Dentro de los factores relacionados con los requerimientos de la aplicación se tuvo en cuenta:

- Autonomía.
- Consistencia transaccional.
- Latencia.

La autonomía de un sitio da la medida de cuánto puede operar el sitio desconectado de la BD publicadora. La consistencia transaccional de un sitio está dada por la necesidad de ejecutar o no inmediatamente todas las transacciones que se han ejecutado en el servidor, o si es suficiente con respetar el orden de las mismas. La latencia de un sitio se refiere al momento en que se deben sincronizar las copias de los datos, si necesitan los datos estar el 100% en sincronía o de qué tamaño es aceptable la espera si es admisible determinada latencia.

Entre los factores relacionados con el entorno de red están la velocidad de transmisión de datos y la confiabilidad. Por otra parte, en el caso que los servidores no permanezcan todo el día encendidos deben considerarse los horarios de disponibilidad de cada servidor. La consideración de estos factores han servido de guía en la configuración del ambiente de replicación, por lo que se le dio prioridad a la autonomía de los sitios (OCCM) que es donde se lleva el proceso fundamental y deben estar disponibles todo el tiempo; además se tuvo en cuenta que aunque la red es segura y rápida, pueden presentarse situaciones externas como fenómenos atmosféricos o interrupción prolongada en el servicio eléctrico que faciliten que los servidores en las DPFP, la DNM o en las propias OCCM estén desconectados. Dado todo esto se selecionó la replicación por mezcla.

2.3.4 Tipos de suscripciones

Los tipos de suscripción que se han modelado son:

Suscripción por inserción *(Push),* donde el publicador genera la publicación y esta se entrega a todos los suscriptores en forma inmediata. La tarea de distribución de la publicación está asignada a la misma entidad publicadora (servidor). En este trabajo se implementa para los nodos de las OCCM a las DPFP y de las DPFP a la DNM.

Suscripción por extracción *(Pull)*, en el que el suscriptor solicita la última publicación generada por el publicador. Esta petición se hace directamente al distribuidor, y es la

expresión más alta de asincronía. En este trabajo se implementa para los nodos de la DNM a las DPFP y de las DPFP a las OCCM.

2.3.5 Entornos de la 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 sólo lectura porque permite a un solo sitio (maestro) realizar consultas de escritura sobre los demás, mientras estos sólo pueden hacer consultas de lectura (esclavos).

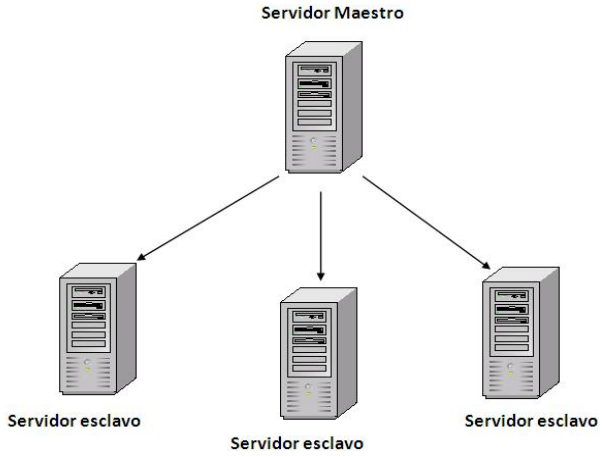


Figura 2.4 Replicación Maestro-Esclavo

Maestro-Maestro*:* También llamado par-a-par o réplica de camino de n, porque permite que múltiples sitios (maestros) actúen 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. Este es el entorno de replicación que se ha incluido en el presente trabajo y se ha representado gráficamente en la Figura 2.5

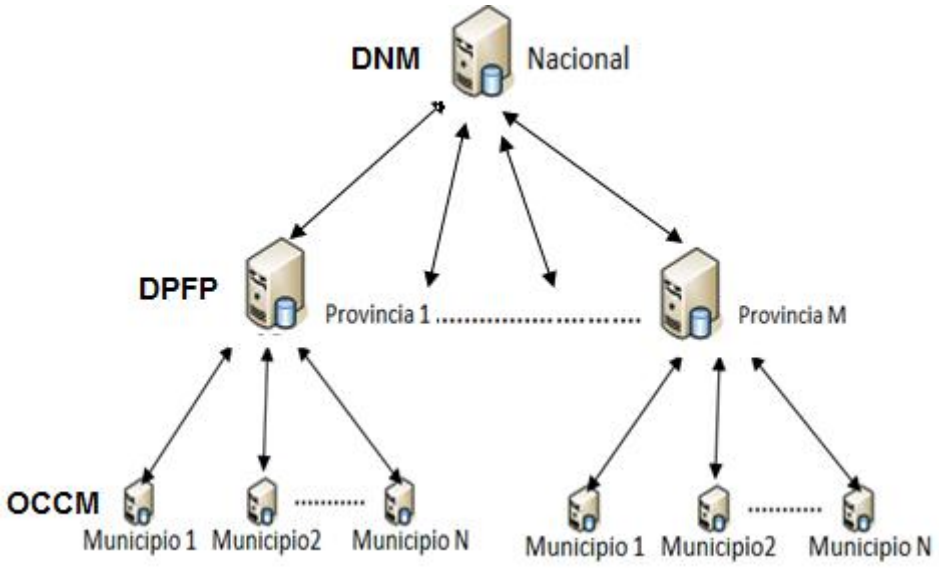


Figura 2.5 Replicación Maestro-Maestro

La BD de Control y Cobro de Multas cuenta con 164 tablas, de la nación hacia la provincia se replican 59, de provincia a nación 43, de provincia a municipio 62 y de municipio a provincia 38.

2.3.6 Ambientes de la réplica

Un ambiente de replicación es una configuración de dos o más sitios mediante un escenario par a par. Cada sitio es un par que contiene un motor de replicación y una BD simple o compartida. El motor de replicación puede residir en la misma computadora que la BD asociada, como es el caso de este trabajo, o en una computadora separada. En cada opción, la BD debe ser accesible por el cliente del motor de replicación. Cada sitio almacena sólo los datos que requieren los usuarios locales. En segundo plano, el motor de replicación gestiona los cambios realizados a la BD sincronizando las actualizaciones de los datos con otros sitios activos en la red.

Una red de replicación puede ser de los siguientes tipos:

- Homogénea: Se replican datos entre BD con SGBD y plataformas del mismo tipo.
- Homogénea con diferentes plataformas: Se replican datos entre BD con SGBD del mismo tipo y plataformas diferentes.

- Heterogénea: Se replican los datos entre BD en SGBD diferentes no importa si el sistema operativo sea el mismo o no.

Una red de replicación requiere una estructura básica TCP/IP que posibilite una comunicación efectiva y eficiente entre todos los sitios. La red de replicación por sí misma constituye estructuralmente una red virtual que se coloca por encima de la red física. Para el problema de este trabajo se tiene el SGBD PostgreSQL 8.4 y la misma plataforma en todos los sitios, por lo que es una red de replicación homogénea.

2.4 Conclusiones parciales

La replicación es un mecanismo utilizado para propagar y diseminar datos en un ambiente distribuido, con el objetivo de tener mejor desempeño y confiabilidad, mediante la reducción de la dependencia de un SBD centralizado.

Se creó un modelo de replicación que garantiza la distribución de datos para el SCCM que permite gestionar toda la información del PCCM a nivel nacional y provincial sin que las OCCM pierdan la autonomía sobre sus datos y que las OCCM interactúen entre sí y tengan la información que le es necesaria para su trabajo en el menor tiempo posible.

CAPÍTULO 3: IMPLEMENTACIÓN DEL MODELO DE RÉPLICA PARA EL SISTEMA DE CONTROL Y COBRO DE MULTAS

En este capítulo se muestra la implementación del modelo de replicación de datos para el SCCM mediante el uso de la herramienta SymmetricDS, de la cual se detalla su funcionamiento.

3.1 Características de SymmetricDS

Una instalación de SymmetricDS se denomina nodo. Un nodo es inicializado mediante un archivo properties, y es configurado insertando datos de configuración en una serie de tablas de BD. A continuación, el nodo crea triggers de BD en las tablas de aplicación especificadas, de modo que los eventos de BD son capturados para ser entregados a otros nodos SymmetricDS.

3.1.1 Esquemas de notificación

Tras registrar un cambio que se ha producido en la BD, se notifican los nodos interesados en el cambio. La notificación de cambios se configura para realizar push o pull. Cuando varios nodos dirigen sus cambios a un nodo central, es eficiente realizar un push de los cambios, en vez de esperar a que el nodo central realice un pull de cada nodo origen. Cuando la configuración de red protege a un nodo con un firewall, una configuración pull permite que el nodo reciba cambios de datos que de otro modo podrían ser bloqueados utilizando push. La frecuencia de la notificación de cambios es de un minuto en la configuración predefinida.

3.1.2 Sincronización bidireccional de tablas

SymmetricDS permite la sincronización bidireccional, y evita incurrir en ciclos de actualización porque sólo registra cambios de datos fuera de la sincronización.

3.1.3 Canales de Datos

La sincronización de datos se define al nivel de tabla o de subconjunto de tablas. Cada tabla gestionada puede ser asignada a un canal que ayuda a controlar el flujo de datos. Un canal es una categoría de datos que puede ser habilitada, priorizada y sincronizada independientemente del resto de los canales.

3.1.4 Filtrado y re-enrutado de datos

Los datos pueden ser filtrados como son registrados, extraídos y cargados.

- El enrutamiento y filtrado de los datos se especifica mediante expresiones SQL en la configuración de los triggers. Las expresiones SQL son configuradas en el node_select para el enrutamiento en tiempo real y en el initial_load_select para la carga inicial de datos. Las expresiones SQL se utilizan para crear el SQL trigger que SymmetricDS instala para capturar los cambios de los datos. Usando esta técnica, los datos pueden enviarse a un nodo específico o a un grupo de nodos.
- Los datos pueden ser filtrados, ya sea cuando se cargan en la BD de destino o cuando se extraen una BD fuente. El filtro puede impedir que los datos lleguen a la BD en conjunto, reemplazando de manera efectiva la carga de datos por defecto.
- Se pueden excluir columnas de la sincronización para que nunca se registren cuando se cambia la tabla. Como los cambios de datos se cargan en la BD de destino, una clase que implementa IColumnFilter puede remover una columna de la sincronización.

3.1.5 Transporte HTTP(S)

De forma predeterminada, SymmetricDS utiliza HTTP o HTTPS en Internet en un estilo llamado Representation State Transfer (REST), que es ligero y fácil de manejar. También se proporciona una serie de filtros para aplicar la autenticación y para restringir el número de sincronizaciones simultáneas. La interfaz InternalTransportManager permite implementar otros transportadores. Las pruebas unitarias para SymmetricDS aprovechan InternalTransportManager que facilitan la ejecución de pruebas automatizadas a nivel local.

3.1.6 Administración Remota

Las funciones de administración están expuestas a través de Java Management Extensions (JMX), que se puede acceder desde la JConsole Java o a través de un servidor de aplicaciones. Las funciones incluyen la apertura de registro, carga de datos, depuración de datos antiguos, y la visualización de los lotes. También están disponibles un número de propiedades de configuración de tiempo de ejecución.

SymmetricDS tiene funcionalidades para enviar eventos SQL a través de los mismos mecanismos de sincronización que se usan para enviar datos. La carga de datos puede ser cualquier sentencia SQL. El evento se procesa y se reconoce como cualquier otro tipo de evento.

3.2 Configuración básica

Para mantener ejecutándose una instancia de SymmetricDS es necesario tener una identidad y conocer cómo conectarse a la BD que se manejará. Una forma básica para especificar esto es en el archivo symmetric.properties. Cuando SymmetricDS se inicia, lee esta configuración y el estado de la BD. Las tablas de configuración se crean automáticamente si no existen. Una configuración básica describe lo siguiente:

- Node Group: cada nodo pertenece a un grupo. Una categoría de nodos que sincroniza datos con uno o más grupos de nodos. Un uso común de grupos de nodos es para describir un nivel en la jerarquía de la sincronización de datos. Esta configuración se guarda en la tabla sym_node_group. Para el caso del SCCM se crearon tres grupos de nodos: Nacional, Provincial y Municipios. En el grupo Nacional se incluye la DNM, en el Provincial todas las DPFP del país y en Municipios las OCCM del país.
- Node Group Link: enlaza dos Node Group a través de la sincronización. Especifica el esquema de notificación, Push- P , Pull – W Wait for Pull o personalizado por el usuario, a utilizar para intercambiar datos entre dos grupos de nodos. Para establecer la sincronización entre nodos deben ser enlazados dos grupos de nodos entre ellos. La dirección de la sincronización es determinada por la especificación de un grupo de nodos fuente y uno destino. Si la sincronización ocurre en ambas direcciones, entonces se crean dos enlaces en direcciones opuestas. El grupo de nodos destino recibe los datos

intercambiados por un método pull o push. Un método push causa que el grupo de nodos fuente se conecte al destino, mientras que un pull causa una espera hasta que el destino se conecte. Los enlaces entre los grupos de nodo quedaron configurados de la siguiente manera:

Grupo Origen	Grupo Destino	Sistema de notificación para enviar datos al destino
Nacional	Provincia	W
Provincia	Nacional	P
Provincia	Municipios	W
Municipios	Provincias	P

- Channel: los datos se categorizan para sincronizarse independientemente. Los cambios en la BD se capturan en el orden en que ocurren y se conservan durante la sincronización con otros nodos. Algunos datos pueden necesitar una prioridad para la sincronización independientemente del orden normal de los acontecimientos. Los canales proporcionan un orden de procesamiento de más alto nivel para los datos, una limitación en la cantidad de datos, y el aislamiento de los errores. Al clasificar los datos en los canales, el usuario tiene más control y visibilidad sobre el flujo de los mismos. Se definieron canales de Nación a Provincia, de Provincia a Nación, de Provincia a Municipio y de Municipio a Provincia. Dentro de estos se tuvo en cuenta la dependencia entre las tablas que van a ser replicadas para establecer niveles, en el nivel 0 se agrupan las tablas que no tienen padre, en el nivel 1 las que su padre está en el nivel 0 y así las del nivel n para las que su padre se encuentra en el nivel n-1. También se le dio prioridad por canales independientes a las operaciones de traslado y notificación, creando para los traslados un canal de Nación a Provincia, de Provincia a Municipio y para la notificación de Provincia a Municipio y de Municipio a Provincia, estableciendo también niveles. Quedando 32 canales definidos de la siguiente forma:

  Canal Notificaciones de Nación a Provincia Nivel 0

  Canal Traslados de Nación a Provincia Nivel 0 hasta el nivel 2

  Canal de Provincia a Nación Nivel 0 hasta el nivel 2

  Canal de Provincia a Nación Nivel 0 hasta el nivel hasta el nivel 6

  Canal de Provincia a Municipio Nivel 0 hasta el nivel 2

  Canal Notificaciones de Provincia a Municipio Nivel 0

Canal Traslados de Provincia a Municipio Nivel 0 hasta el nivel 2

Canal Notificaciones de Provincia a Municipio Nivel 0

Canal de Municipio a Provincia Nivel 0 hasta el nivel 7

Canal Notificaciones de Municipio a Provincia Nivel 0

Canal de Municipio a Provincia Nivel 0 Logs

En el Anexo 1 se muestran los scripts a ejecutar para configurar las características antes mencionadas.

- Trigger: especifica qué cambios serán capturados en la BD. El núcleo del SymmetricDS son los triggers que definen qué datos se capturan. Los nodos en el grupo de nodos fuente capturarán los cambios de una tabla y los enviarán al grupo de nodos destino. Los cambios pueden incluir inserción, actualización o eliminación en la tabla, y estos datos pueden ser filtrados por una expresión condicional. La condición puede ser especificada para capturar los cambios de la fila en la BD. Si la condición se cumple, los cambios son capturados y enviados al destino. Si la condición no se cumple, se permite el cambio de los datos, pero estos no son capturados y enviados al destino. Las condiciones corren dentro del contexto del evento de trigger de la BD, y usa el lenguaje específico de la plataforma de la BD. Una condición puede ser especificada para una inserción, actualización y eliminación, y esta corre para cada fila del evento. Durante la puesta en marcha, los trigger se verifican contra la BD, y se instalan en las tablas que requieren que los cambios se capturen. El PullJob y PushJob comienzan a correr para sincronizar los cambios con otros nodos. Ejemplos de los triggers implementados se muestran y se explican en el Anexo 3.

3.3 Propiedades básicas

Cada nodo requiere propiedades que lo conectarán con la BD y con el nodo padre. Para dar a un nodo su identidad, se usan las siguientes propiedades:

- group.id: El grupo al cual pertenece el nodo. La sincronización se realiza entre grupos de nodos.
- external.id: El ID externo para el nodo; tiene un significado para el usuario y ofrece una integración en el sistema en que esté desplegado. El identificador externo se puede utilizar en expresiones condicionales. Es transparente para al usuario; cada nodo tiene

un número de secuencia única para el seguimiento de eventos de sincronización. Es posible asignar el mismo identificador externo a varios nodos, si se desea.

- my.url: El localizador URL donde el nodo puede ser contactado para sincronizar. Cuando un nodo se inicia por primera vez, este no tiene información sobre la sincronización. Este contacta con el servidor de registro para unirse a la red y recibir su configuración. La configuración de los nodos es almacenada en el servidor de registro, y el URL debe ser especificado en la propiedad.
- registration.url: El URL donde el nodo puede conectarse para registrarse y recibir su configuración.
- db.driver: Nombre de la clase del driver para JDBC.
- db.url: El URL de la JDBC usado para conectar la BD.
- db.user: El nombre del usuario para conectarse a la BD. Este puede ser encriptado.
- db.password: La clave del usuario de la BD. Esta también puede ser encriptada.

En el Anexo2 se muestra la configuración del fichero symmetric.properties para el nodo nacional, un nodo provincial y uno municipal.

3.4 Modelo de datos de SymmetricDS

Muchos de los conceptos de sincronización de datos pueden ser comprendidos examinando el modelo de datos. El modelo de datos de configuración es un conjunto de tablas actualizadas por el usuario como sea necesario para configurar el sistema. Por el contrario, el conjunto de tablas para el modelo de datos en tiempo de ejecución cambia constantemente conforme el sistema captura cambios en los datos y registra la actividad para entregarlos.

Todos los nombres de tabla tienen un prefijo configurable, para que puedan coexistir varias instancias de SymmetricDS en la misma BD. El prefijo por defecto es *sym_*. Este puede ser cambiado mediante la propiedad sync.table.prefix.

3.4.1 Modelo de datos de configuración

La configuración es introducida por el usuario en el modelo de datos para controlar el comportamiento de qué datos son sincronizados con qué nodos. Para facilitar la

administración de varias BD, los nodos se incluyen en grupos, y los grupos se enlazan entre sí para la sincronización. Un trigger captura datos para una tabla, que puede incluir condiciones y criterios para subconjuntos. Los triggers se agrupan en canales para la priorización y control del flujo de datos. A continuación se describen algunas de las tablas más importantes de este modelo.

Tabla node

Cada nodo tiene un identificador único (nodeid) que se utiliza en la comunicación, así como un dominio específico de identificador (externalid) que proporciona el contexto dentro del sistema local.

Nombre de columna	Tipo	Valor nulo	LLave	Valor predefinido	Descripción
node_id	varchar(50)	No	PK		Identificador único par el nodo
node_group_id	varchar(50)	No	FK		El node_group en el cual está incluido el nodo
external_id	varchar(50)	No			Un dominio identificador específico para el contexto dentro del sistema local.
sync_enabled	booleanint	Sí		0	Indica si este nodo debe enviar la sincronización
sync_url	varchar(2000)	Sí			La URL para contactar con el nodo para la sincronización
schema_version	varchar(50)	Sí			La versión del esquema de BD que este nodo gestiona. Útil para especificar la sincronización con la versión.
symmetric_version	varchar(50)	Sí			La version del SymmetricDS que está ejecuntándose en el nodo.
database_type	varchar(50)	Sí			El nombre del producto de BD de este nodo según lo informado de JDBC.
database_version	varchar(50)	Sí			La versión del producto de BD de este nodo según lo informado de JDBC.
heartbeat_time	timestamp	Sí			La marca de tiempo, cuando el nodo envía un pulso, que se intenta cada diez minutos de forma predeterminada.
timezone_offset	varchar(6)	Sí			El ajuste de zona horaria en formato

Nombre de columna	Tipo	Valor nulo	LLave	Valor predefinido	Descripción
					RFC822 en el momento del último pulso.
created_at_node_id	varchar(50)	Sí			El identificador del nodo donde se creó este nodo. Esto se llena con el id que se encuentran en node_identity cuando el registro fue abierto para el nodo.

Figura 3.1 Tabla node

Tabla node_identity

Después del registro, esta tabla tiene una fila que representa la identidad del nodo. Para un nodo de la raíz, la fila es introducida por el usuario.

Nombre de columna	Tipo	Valor nulo	LLave	Valor x Defecto	Descripción
node_id	varchar(50)	No	PK FK		Identificador único par el Nodo

Figura 3.2 Tabla node_identity

Tabla node_group

Un uso común del grupo de nodos es describir un nivel en una jerarquía de sincronización de datos.

Nombre de columna	Tipo	Valor nulo	LLave	Valor x Defecto	Descripción
node_group_id	varchar(50)	No	PK		Identificador único del grupo de nodos; usualmente es un nombre que identifica algo, como "provincias", "nacional" , "municipios"
description	varchar(50)	Sí			Una descripción del grupo.

Figura 3.3 Tabla node_group

Tabla node_group_link

Un grupo de nodos envía sus actualizaciones de datos a otro grupo de nodos mediante un pull, push, o una técnica personalizada.

Nombre de columna	Tipo	Valor nulo	LLave	Valor x Defecto	Descripción
source_node_group_id	varchar(50)	No	PK FK		El grupo de nodos donde los cambios de datos deben ser capturados.
target_node_group_id	varchar(50)	No	PK FK		El grupo de nodos donde los datos cambiados deben ser enviados.
data_event_action	char(1)	No		W	El sistema de notificación usado para enviar los cambios de datos al grupo de nodo destino. (P = Push, empuje, W = Espere por Pull)

Figura 3.4 Tabla node_group_link

Tabla channel

Una categoría de datos que pueden ser sincronizados de forma independiente de otros canales. Los canales permiten el control sobre el tipo de datos que fluyen.

Nombre de columna	Tipo	Valor nulo	LLave	Valor x Defecto	Descripción
channel_id	varchar(50)	No	PK		Un identificador único, usualmente un nombre que representa algo, por ejemplo “Actualiza nomencladores”, “Actualiza municipios”
processing_order	integer	No		1	Orden para procesar los datos del canal.
max_batch_size	integer	No		1000	El número máximo de eventos de datos a procesar dentro de un lote del canal.
enabled	booleanint	No		1	Indica si el canal está disponible o no
description	varchar(1000)	Sí			Descripción del tipo de datos que va por el canal

Figura 3.5 Tabla channel

Tabla Trigger

Los triggers (disparadores) de la BD provocan que los cambios en la BD se capturen automáticamente por SymmetricDS. Se genera la configuración de las acciones que se desencadenan y cómo se comportarán y se almacena en la tabla trigger.

Nombre de columna	Tipo	Valor nulo	LLave	Valor x Defecto	Descripción
trigger_id	integer	No	PK		Identificador único del disparador
source_schema_name	varchar(50)	Sí			El nombre del esquema donde reside la tabla origen. El disparador de BD se creará para el source_table_name en este esquema.
source_table_name	varchar(50)	No			El nombre de la tabla de origen que tiene el disparador para observar los cambios en los datos.
target_schema_name	varchar(50)	Sí			El nombre del esquema donde reside la tabla destino. Cuando los datos se envían a la BD remota, las instrucciones SQL que se utilizan para cargar los datos califican el target_table_name con este esquema.
target_table_name	varchar(50)	Sí			El nombre de la tabla destino que recibirá los cambios de datos.
source_node_group_id	varchar(50)	No			El grupo de nodos donde se instalará este disparador para observar los cambios de datos
target_node_group_id	varchar(50)	No			El grupo de nodos donde cambiarán los datos.
channel_id	varchar(50)	No			El canal por donde fluirán los cambios de datos.
sync_on_update	booleanint	No		1	Si se va a instalar o no un desencadenador de actualización.
sync_on_insert	booleanint	No		1	Si se va a instalar o no un disparador de inserción.
sync_on_delete	booleanint	No		1	Si se va a instalar o no un disparador de eliminación.
sync_on_incoming_batch	booleanint	No		0	Si los datos que llegan causan un disparador para ser enviados a otro destino. Se debe tener cuidado porque esto puede causar un ciclo infinito.
sync_column_level	booleanint	No		0	Establece la sincronización a nivel

Nombre de columna	Tipo	Valor nulo	LLave	Valor x Defecto	Descripción
					de columnas, que solamente los campos que cambiaron sean actualizados en la BD destino. Cuando las actualizaciones vienen de varias fuentes al mismo tiempo, esto permite que la fusión de los cambios no provoque conflictos.
name_for_update_trigger	varchar(30)	Sí			Reemplaza el nombre predeterminado para el disparador de actualización.
name_for_insert_trigger	varchar(30)	Sí			Reemplaza el nombre predeterminado para el disparador de inserción.
name_for_delete_trigger	varchar(30)	Sí			Reemplaza el nombre predeterminado para el disparador de eliminación.
sync_on_update_condition	varchar(1000)	Sí			Especifica una condición para que se ejecute disparador de actualización.
sync_on_insert_condition	varchar(1000)	Sí			Especifica una condición para que se ejecute disparador de inserción.
sync_on_delete_condition	varchar(1000)	Sí			Especifica una condición para que se ejecute disparador de eliminación.
initial_load_select	varchar(1000)	Sí			Especifica una cláusula Where para una carga inicial de esta tabla. La tabla tiene el alias de "t". Variables de sustitución es de $ (nodeid), $ (groupId), y $ (externalid).
node_select	varchar(1000)	Sí			Especifica una cláusula Where para seleccionar los nodos que recibirán los cambios. La tabla de nodo es un alias como "c".
tx_id_expression	varchar(1000)	Sí			Reemplaza la expresión predeterminada para el identificador de la transacción que agrupa a los cambios en los datos que se cometieron juntos.
excluded_column_names	varchar(1000)	Sí			Especifica una lista delimitada por comas de las columnas que no deben estar sincronizados de esta tabla.
initial_load_order	integer	No		1	Secuencia de orden de esta tabla cuando una se envía una carga

Nombre de columna	Tipo	Valor nulo	LLave	Valor x Defecto	Descripción
					inicial a un nodo.
create_time	timestamp	No			Marca de tiempo cuando se creó esta entrada.
inactive_time	timestamp	Sí			Marca de tiempo cuando esta entrada fue inactivada, lo que detiene la captura de los cambios de datos.
last_updated_by	varchar(50)	Sí			El usuario que actualizó por última vez esta entrada.
last_updated_time	timestamp	No			Marca de tiempo cuando un usuario actualizó por última vez esta entrada.

Figura 3.6 Tabla trigger

Tabla parameter

La tabla de parámetros proporciona una manera de manejar la mayoría de configuración de SymmetricDS desde la BD.

Nombre de columna	Tipo	Valor nulo	LLave	Valor x Defecto	Descripción
external_id	varchar(50)	No	PK		Objetivo para el parámetro en una identificación externa específica. Para seleccionar todos los nodos se debe utilizar el valor "ALL".
node_group_id	varchar(50)	No	PK FK		Objetivo para el parámetro en un grupo de nodos específicos de identificación. Para seleccionar todos los grupos se debe utilizar el valor "ALL".
param_key	varchar(100)	No	PK		El nombre del parámetro.
param_value	varchar(1000)	No			El valor del parámetro.

Figura 3.7 Tabla parameter

3.4.2 Modelo de datos de tiempo de ejecución

En tiempo de ejecución, la configuración se utiliza para capturar los cambios de datos y la ruta a los nodos. Los cambios de datos se colocan juntos en una sola unidad llamada lote

que puede ser cargado por otro nodo. Los lotes de salida se entregan a los nodos y se reconocen. Los lotes entrantes se reciben y se cargan. En la historia se registran los cambios de estado de proceso por lotes y estadísticas. Entre las tablas más significativas de este modelo se encuentran las siguientes.

Tabla outgoing_batch

Esta tabla se utiliza para rastrear el envío de una colección de datos a un nodo. Se crea un lote de salida por el lote saliente de servicio y teniendo en cuenta el estado de "NE" (nueva creación). Después de enviar el lote saliente a su nodo de destino, el estado se convierte en "SE" (enviado a nodo). El nodo responde con un estado de éxito de "OK " o un estado de error de "ER". Un error al enviar al nodo también se traduce en un estado de error de "ER", independientemente de si el nodo envía acuse de recibo. Véase la tabla 3.8 para reconocer los diferentes estados.

Nombre de columna	Tipo	Valor nulo	LLave	Valor x Defecto	Descripción
batch_id	integer	No	PK		Identificador único del lote
node_id	varchar(50)	No	PK		El nodo que envió a este lote.
channel_id	varchar(50)	Si			El canal que clasifica los datos en este lote
batch_type	varchar(2)	No		EV	Tipos de lotes, incluyen los eventos que se activan cuando las filas cambian (EV) y la carga inicial de que se envía a todas las tablas (IL).
Status	char(2)	Sí			El estado actual de los lotes, pueden ser de nueva creación (NE), enviado a un nodo (SE), reconocida como exitosa (OK), y el error (ER).
create_time	timestamp	Sí			La fecha y hora en que el lote fue recibido por primera vez.

Figura 3.8 Tabla outgoing_batch

Tabla incoming_batch

El lote entrante se utiliza para el seguimiento del estado de carga de un lote saliente de otro nodo. Los datos se cargan y se ejecutan a nivel de lote. El estado de los lotes de entrada puede ser correctamente (OK) o error (ER).

Nombre de columna	Tipo	Valor nulo	LLave	Valor x Defecto	Descripción
batch_id	integer	No	PK		Identificador único del lote
node_id	varchar(50)	No	PK		El nodo que envió a este lote.
Status	char(2)	Sí			El estado actual de los lotes puede ser cargado correctamente (OK) o error (ER).
create_time	timestamp	Sí			La fecha y hora en que el lote fue recibido por primera vez.

Figura 3.9 Tabla incoming_batch

3.5 Arquitectura

La aplicación *SymmetricDS* permite que los cambios de entrada y de salida sean sincronizados con otras BD. El nodo que inicia la conexión es el cliente y el nodo que la recibe es el servidor. Dado que la sincronización está configurada para extraer o enviar en cualquier dirección, el mismo nodo puede actuar a la vez como cliente o como servidor en diferentes circunstancias. La aplicación consiste una serie de tareas, administradores, servlets, y servicios que se ofrecen juntos (véase la figura 3.10).

Como cliente, el nodo ejecuta el trabajo de envío y extracción en el hilo de un contador que se sincroniza con el nodo servidor. El envío usa los servicios de procesamiento por lotes, extracción, y envía datos a otro nodo. La respuesta a un envío es una lista de recibo de procesamiento por lotes que indica que el dato fue cargado. El trabajo de extracción usa el servicio para cargar datos que se envían desde otro nodo. Después que se cargan los datos, se hace una segunda conexión para enviar una lista de confirmación de recepción de procesamiento en lotes (véase la figura 3.10).

Como servidor, el nodo espera las conexiones entrantes que extraen, envían o confirman cambios en los datos. El servlet de envío usa los servicios para cargar los datos que se envían desde el nodo cliente. Después de cargar los datos, responde con una lista de confirmación de recibo de procesamiento en lotes. El servlet de extracción usa los servicios de procesamiento en lotes, extracción y envío de datos hacia el nodo cliente. EL servlet de confirmación de recibo usa los servicios para actualizar el estado de los datos que se cargaron en el nodo cliente (véase la figura 3.10).

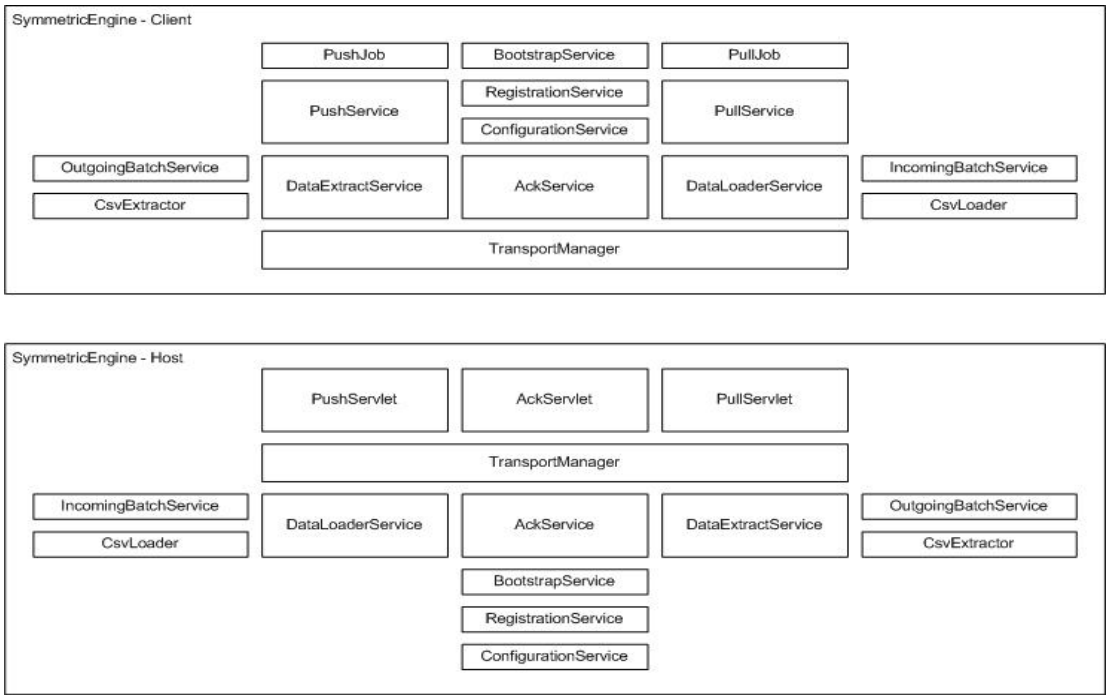


Figura 3.10 Esquema de funcionamiento

El administrador de Transporte maneja los flujos de datos que salen y entran entre los nodos. El transporte por defecto está basado en una simple implementación sobre el protocolo HTTP. También se suministra un transportador interno. Es posible además adicionar otras implementaciones, como la de un administrador de transporte basado en socket (véase la figura 3.10). La comunicación del nodo usando HTTP se representa en la figura 3.11.

La API SymmetricEngine puede ser usada para iniciar, de forma directa, solamente los servicios del cliente. La API SymmetricWebServer puede ser usada para iniciar, de forma directa, ambos servicios, cliente y servidor dentro de un contenedor Web Jetty.

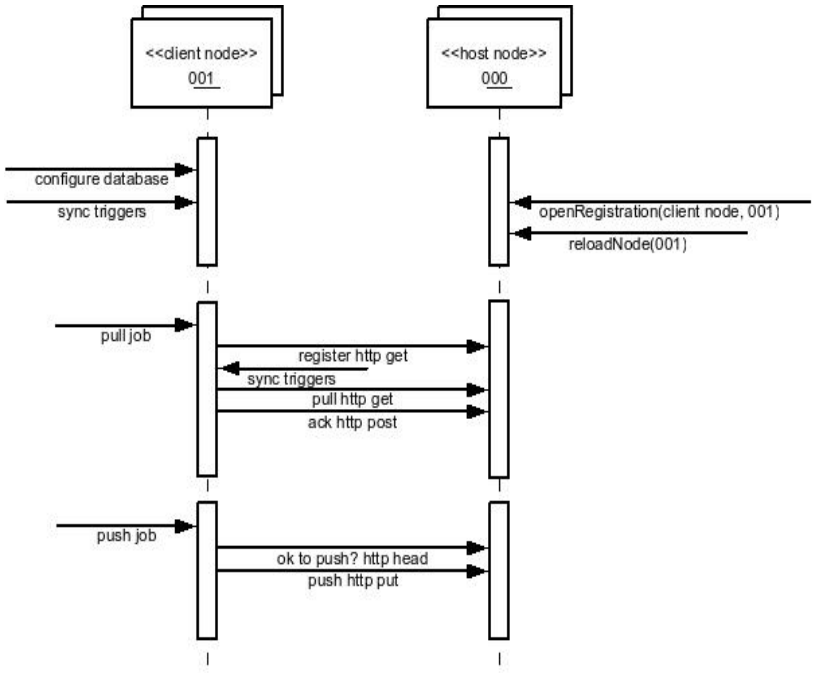


Figura 3.11 Comunicación entre nodos

3.5.1 Opciones de despliegue

Standalone

Un servicio independiente puede utilizar las opciones de la línea de comandos del sym para iniciar un servidor. Una instancia incorporada de Jetty se utiliza para atender las solicitudes Web de todos los servlets.

```
/symmetric/bin/sym --properties root.properties --port 8080 --server
```

En este ejemplo, se inicia el servidor SymmetricDS por el puerto 8080 con las propiedades de inicio que se encuentran en el archivo root.properties.

Embebido

Una aplicación Java con el SymmetricDS Java Archive (JAR) de la colección en su ruta de clases puede utilizar el SymmetricEngine para iniciar el servidor.

```
import org.jumpmind.symmetric.SymmetricEngine;
 public class StartSymmetricDSEngine {
     public static void main(String[] args) throws Exception {
        String workingDirectory = System.getProperty("user.dir");
        SymmetricEngine engine = new SymmetricEngine(
            "classpath://my-application.properties",
            "file://" + workingDirectory + "/my-environment.properties");
         // this will create the database, sync triggers, start jobs running
        engine.start();
                // this will stop the engine
        engine.stop();
    }
 }
```

En este ejemplo se inicia el servidor SymmetricDS en el puerto 8080 con las propiedades de inicio que se encuentran en dos lugares. El primer archivo, mi-application.properties, se empaqueta en la aplicación para proporcionar propiedades que anulan los valores predeterminados SymmetricDS. El segundo archivo, mi-environment.properties, se encuentra en un directorio de trabajo que reemplaza las propiedades específicas para el ambiente de desarrollo. Esto permite que la misma solicitud de desplegarse en entornos de desarrollo y producción usando diferentes mi-environment.properties.

Como aplicación Web

Esta opción significa empaquetar un archivo WAR y desplegarlo en un servidor Web. Esta la que se utilizó en la aplicación de esta tesis. Como un archivo de aplicación Web, un archivo WAR o EAR se despliega en un servidor de aplicaciones, tal como Tomcat, Jetty, o JBoss. La estructura de archivos debe tener un archivo web.xml en la carpeta WEB-INF, el archivo symmetric.properties en la carpeta WEB-INF/classes, y los archivos JAR necesarios en la carpeta WEB-INF/lib.

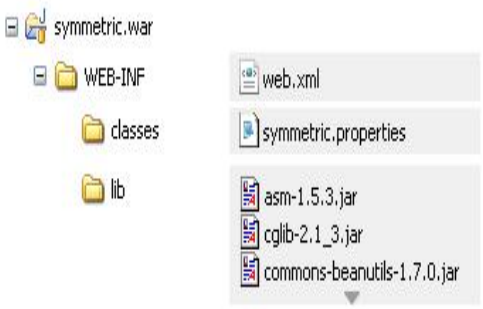


Figura 3.12 Estructura de archivo de la aplicación Web

El fichero WEB-INF/web.xml se configura con un SymmetricEngineContextLoaderListener el requerido SymmetricFilter, y la necesaria asignación de SymmetricServlet.

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    version="2.5">
    <display-name>sync</display-name>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <!-- You can optionally specify other Spring files to load into same context here
        <param-value>classpath:symmetric.xml</param-value>
    </context-param>

    <filter>
        <filter-name>SymmetricFilter</filter-name>
        <filter-class>
            org.jumpmind.symmetric.web.SymmetricFilter
        </filter-class>
    </filter>

    <filter-mapping>
        <filter-name>SymmetricFilter</filter-name>
        <servlet-name>/*</servlet-name>
    </filter-mapping>

    <listener>
        <listener-class>
            org.jumpmind.symmetric.SymmetricEngineContextLoaderListener
        </listener-class>
    </listener>

    <servlet>
        <servlet-name>SymmetricServlet</servlet-name>
        <servlet-class>
            org.jumpmind.symmetric.web.SymmetricServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>SymmetricServlet</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>

</web-app>
```

En este ejemplo se inicia todos los servlets SymmetricDS con filtros para comprimir el flujo, los nodos de autenticación, y rechazar los nodos cuando el servidor está demasiado ocupado. La propiedad web.base.servlet.path en symmetric.properties se puede establecer si el SymmetricServlet necesita coexistir con otros servlets.

Para realizar las pruebas del modelo de replicación implementado se configuró un servidor en la DPFP de Villa Clara para simular el servidor nacional, ya que este no existe actualmente y es necesario para poder probar la réplica desde los servidores provinciales al nacional, en cada OCCM se instalaron los servidores correspondientes, al igual que en la DPFP.

3.6 Conclusiones parciales

Las características mencionadas de SymmetricDS hacen que esta sea la herramienta ideal para la replicación de datos en el modelo propuesto. Además de otras ventajas, permite que el sistema de replicación se mantenga funcionando en caso de problemas con la conectividad ya sea entre los servidores municipales y los provinciales o entre estos últimos y el servidor nacional, guardando los datos que no han podido ser enviados satisfactoriamente y procede con el envío de estos cuando se restablece la conexión. Otra característica importante de SymmetricDS para el funcionamiento del SCCM es la de permitir la réplica bidireccional. El envío de información puede ser ajustado en la medida que cambien las condiciones en las OCCM.

SymmetricDS constituye una potente herramienta para la replicación de datos, ya sean homogéneos o no, independiente del gestor de base de datos que se utilice, siempre y cuando soporte la tecnología de trigger y driver JDBC. Es una herramienta en constante evolución con una amplia comunidad, fácil de aprender y con más de una posibilidad para su despliegue, lo que permite usar la forma más adecuada según las características del hardware que se posea. Su configuración es manejada centralmente.

La hipótesis de la investigación se comprobó a través de los resultados obtenidos con la aplicación del SCCM y los efectos obtenidos durante su utilización en la provincia de Villa Clara donde se configuró un servidor que simuló el servidor nacional al no existir éste, y se pudo probar la réplica entre los servidores provinciales al nacional, en cada OCCM, al igual que en la DPFP.

CONCLUSIONES

1. El estudio realizado para el desarrollo de este trabajo confirmó el crecimiento de la tendencia a la descentralización y el interés por el uso de la replicación para la distribución de datos debido a la necesidad de acceder de forma coherente a los mismos datos como si estuvieran centralizados. La replicación permite compartir, modificar y reconciliar información entre BD y garantiza la disponibilidad de los datos correctos cuándo y dónde se precisen.
2. Se creó un modelo de replicación que garantiza la distribución de datos para el Sistema de Control y Cobro de Multas que permite gestionar toda la información del PCCM a nivel nacional y provincial sin que las OCCM pierdan la autonomía sobre sus datos y que tengan la información que le es necesaria para su trabajo y que se encuentra en otras oficinas, en el menor tiempo posible.
3. El modelo diseñado se aplicó en el subsistema de replicación del Sistema Automatizado para el Control y Cobro de Multas y fue comprobado mediante la implantación del mismo en la DPFP de Villa Clara, donde se simuló un servidor nacional, ya que éste no existe actualmente, y los resultados obtenidos fueron satisfactorios.
4. Se demostró que el SymmetricDS, debido a su simplicidad y capacidad para realizar la réplica en ambos entornos, su flexibilidad y sencillez, su facilidad para instalar, administrar y configurar, causar bajo impacto en la utilización de la memoria y la red, por ser un software de replicación de datos asíncrona que permite subscriptores múltiples y sincronización bidireccional, entre otras, es la herramienta idónea para la replicación de datos del modelo propuesto.

RECOMENDACIONES

1. Realizar pruebas a medida que en las demás OCCM del país se vaya implantando el Sistema Automatizado para el Control y Cobro de Multas.
2. Desarrollar los subsistemas para la integración con otros sistemas y el de apoyo a la toma de decisiones.

REFERENCIAS BIBLIOGRÁFICAS

AGRAWAL, D. & ABBADI, A. E. (1997) Epidemic algorithms in replicated databases. *Of the ACM-SIGACT-SIGMOD-SIGART Int. Symp. On Principles of databse Systems. (PODS)*.

BAIÃO, F. A., MATTOSO, M. & ZAVERUCHA, G. (2002) A Framework for the Design of Distributed Databases. IN LITWIN, W. & LÉVY, G. (Eds.) *Records of the 4th International Meeting on Distributed Data & Structures 4 (WDAS 2002).* Paris, France, Carleton Scientific.

BAIÃO, F. A., MATTOSO, M. & ZAVERUCHA, G. (2004) A Distribution Design Methodology for Object DBMS. *Distributed and Parallel Databases,* 16**,** 45-90.

BELLATRECHE, L., KARLAPALEM, K. & SIMONET, A. (2000) Algorithms and support for horizontal class partitioning in object-oriented databases. *Distributed and Parallel Databases,* 8**,** 155-179.

BERNSTEIN, P. A., HADZILACOS, V. & GOODMAN, N. (1987) *Concurrency Control and Recovery in Database Systems*, Addison-Wesley.

BREITBART, Y., KOMONDOOR, R., RASTOGI, R., SESHADRI, S. & SILBERSCHATZ, A. (1999) Update Propagation Protocols For Replicated Databases. IN DELIS, A., FALOUTSOS, C. & GHANDEHARIZADEH, S. (Eds.) *Proceedings ACM SIGMOD International Conference on Management of Data 1999.* Philadelphia, Pennsylvania, USA, ACM Press.

BURETTA, M. (1997) *Data Replication. Tools and techniques for managing distributed information*, John Wiley & Sons, Inc. 1997.

CAREY, M. & LIVNY., M. (1991) Conflict Detection Tradeoff for Replicated Data. *ACM Transactions on Database Systems,* Vol. 16**,** 703-746.

CERI, S., NAVATHE, S. B. & WIEDERHOLD, G. (1983) Distribution Design of Logical Database Schemas. *IEEE Trans. Software Eng.,* 9**,** 487-504.

CERI, S. & PELAGATTI, G. (1984) *Distributed Databases: Principles and Systems*, McGraw-Hill Book Company.

CERI, S., PERNICI, B. & WIEDERHOLD, G. (1987) Distributed Database Design Methodologies. *IEEE Database Eng. Bull.,* 75**,** 533-546.

COULON, C., PACITTI, E. & VALDURIEZ, P. (2005) Optimistic Preventive Replication in a Database Cluster. *Proceedings of the 21èmes Journées Bases de Données Avancées (BDA).* France, INRIA.

CHEN, S.-W. & PU, C. (1992) A Structural Classification of Integrated Replica Control Mechanisms. Department of Computer Science
Columbia University
New York, NY 10027.

DATE, C. J. (2000) *Introducción a los Sistemas de Bases de Datos, Séptima edición,* México, Addison-Wesley.

ELMASRI, R. & NAVATHE, S. (2002) *Fundamentals of database systems. 3ra Edición*
Addison Wesley.

HABABEH, I. O., BOWRING, N. & RAMACHANDRAN, M. (2004) A Method for Fragment Allocation in Distributed Object Oriented Database Systems. *Proceedings of the 5th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking & Broadcasting (PGNet).* Liverpool, UK.

HUANG, Y.-F. & CHEN, J.-H. (2001) Fragment Allocation in Distributed Database Design. *Journal of Information Science and Engineering,* 17**,** 491-506.

IRÚN-BRIZ, L., DECKER, H., JUAN-MARIN, R., CASTRO-COMPANY, F., ARMENDÁRIZ-INIGO, J. E., BERNABÉU-AUBÁN, J. M. & MUÑOZ-ESCOÍ, F. D. (2005) MADIS: A slim middleware for database replication. *Proceedings of the 11st International Conference on Parallel Processing (EuroPar 2005). Lecture Notes in Computer Science.* Springer.

JIM GRAY, PAT HELLAND, PATRICK E. O'NEIL & SHASHA, D. (1996) The Dangers of Replication and a Solution. *SIGMOD '96.* Montreal, Quebec, Canada.

JOHANSSON, J. M., MARCH, S. T. & NAUMANN, J. D. (2000) The effects of parallel processing on update response time in distributed database design. IN ANG, S., KRCMAR, H., ORLIKOWSKI, W. J., WEILL, P. & DEGROSS, J. I. (Eds.) *Proceedings of the Twenty-First International Conference on Information Systems ICIS.* Brisbane, Australia, ACM.

KEMME, B. & ALONSO, G. (2000) A new approach to developing and implementing eager database replication protocols. *ACM Trans. Database Syst.,* 25**,** 333-379.

KRISHNAKUMAR, N. & BERNSTEIN, A. J. (1991) Bounded ignorance in replicated systems. *ACM SIGACT-SIGMOD-SGART Symp. On Principles of Database Systems*.

LEN, S. (2001) Actualización de Réplicas de datos en entornos de BDD. Facultad de Informática.

LIN, W. J. & VEERAVALLI, B. (2006) An object replication algorithm for real-time distributed databases. *Distributed and Parallel Databases.,* 19**,** 125-146.

MA, H. & SCHEWE, K.-D. (2005) Query optimisation as part of distribution design for complex value databases. IN KIYOKI, Y., KANGASSALO, H., JAAKKOLA, H. & HENNO, J. (Eds.) *Proceedings of the 15th European-Japanese Conference on Information Modelling and Knowledge Bases.* Tallinn University of Technology, Estonia.

MA, H., SCHEWE, K.-D. & WANG, Q. (2005) Distribution design for higher-order data models. Palmerston North, New Zealand, Massey University, Department of Information Systems.

MEGHINI, C. & THANOS, C. (March 1991) The Complexity of Operation on a Fragmented Relation. ACM Transaction on Database Systems. *Nro. 1,* Vol.16.

NAVATHE, S. B., KARLAPALEM, K. & RA, M. (1995) A mixed fragmentation methodology for initial distributed database design. Atlanta, GA, College of Computing, Georgia Institute of Technology.

ŐZSU, M. T. & VALDURIEZ, P. (1991) *Principles of Distributed Database Systems*, Prentice-Hall.

ŐZSU, M. T. & VALDURIEZ, P. (1996) Distributed and Parallel Database Systems. *ACM Comput. Surv.,* 28**,** 125-128.

ŐZSU, M. T. & VALDURIEZ, P. (1999a) *Principles of Distributed Database Systems, Second Edition,* Upper Saddle River, New Jersey., Prentice-Hall.

ŐZSU, M. T. & VALDURIEZ, P. (1999b) *Principles of Distributed Database Systems, Second Edition*.

ŐZSUS, M. T. & VALDURIEZ, P. (1999) *Principles of Distributed Database Systems, Second Edition*.

PACITTI, E., MINET, P. & SIMON, E. (1999) Fast Algorithms for Maintaining Replica Consistency in Lazy Master Replicated Databases. IN ATKINSON, M. P., ORLOWSKA, M. E., VALDURIEZ, P., ZDONIK, S. B. & BRODIE, M. L. (Eds.) *Proceedings of 25th International Conference on Very Large Data Bases VLDB'99.* Edinburgh, Scotland, UK, Morgan Kaufmann.

PU, C. & LEFF, A. (1991) Replica control in distributed systems: an asynchronous approach. *In Proc. Of the ACM SIGMOD Int. Conf. on Management of Data.* Denver, Colorado.

SCHEWE, K.-D. (2002) Fragmentation of object oriented and semi-structured data. IN HAAV, H.-M. & KALJA, A. (Eds.) *Databases and Information Systems II.* Kluwer Academic Publishers.

SIDELL, J., AOKI, P. M., SAH, A., STAELIN, C., STONEBRAKER, M. & YU, A. (1996) Data Replication in Mariposa. *ICDE '96 Proceedings of the Twelfth International Conference on Data Engineering*

SOMMERVILLE, I. (2002) *Ingeniería de Software. 6ta Edición.*, Adddison Wesley. 2002.

ANEXOS

Anexo 1. Scripts para la configuración básica

```
--
-- GRUPOS DE NODOS
--

--------- Nodo Nacional
insert into sym_node_group (node_group_id, description) values ('nacional', 'Nacional');

--------- Nodos Provinciales
insert into sym_node_group (node_group_id, description) values ('provincias', 'Provincias');

--------- Nodos Municipales
insert into sym_node_group (node_group_id, description) values ('municipios', 'Municipios');


--
-- VINCULOS ENTRE GRUPOS DE NODOS
--

--------- Vinculos entre Nacion y Provincias
insert into sym_node_group_link (source_node_group_id, target_node_group_id, data_event_action)
values ('nacional', 'provincias', 'W');

insert into sym_node_group_link (source_node_group_id, target_node_group_id, data_event_action)
values ('provincias', 'nacional', 'P');

--------- Vinculos entre Provincias y Municipios
insert into sym_node_group_link (source_node_group_id, target_node_group_id, data_event_action)
values ('provincias', 'municipios', 'W');

insert into sym_node_group_link (source_node_group_id, target_node_group_id, data_event_action)
values ('municipios', 'provincias', 'P');

--
-- NODO NACIONAL
--

insert into sym_node (node_id, node_group_id, external_id, sync_enabled) values ('00',
'nacional', '00', 1);

insert into sym_node_identity values ('00');
```

```
--
-- CANALES
--
============== CANALES DE NACION A PROVINCIA ==================

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal N2P Nv 0', 1, 1000, 1, 'Canal de Nacion a Provincia Nivel 0');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal N2P Nv 1', 2, 1000, 1, 'Canal de Nacion a Provincia Nivel 1');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal N2P Nv 2', 3, 1000, 1, 'Canal de Nacion a Provincia Nivel 2');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal Trsl N2P Nv 0', 1, 1000, 1, 'Canal Tralados de Nacion a Provincia Nivel 0');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal Trsl N2P Nv 1', 2, 1000, 1, 'Canal Traslados de Nacion a Provincia Nivel 1');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal Trsl N2P Nv 2', 3, 1000, 1, 'Canal Traslados de Nacion a Provincia Nivel 2');


================ CANALES DE PROVINCIA A NACION ================

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal P2N Nv 0', 1, 1000, 1, 'Canal de Provincia a Nacion Nivel 0');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal P2N Nv 1', 2, 1000, 1, 'Canal de Provincia a Nacion Nivel 1');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal P2N Nv 2', 3, 1000, 1, 'Canal de Provincia a Nacion Nivel 2');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal P2N Nv 3', 4, 1000, 1, 'Canal de Provincia a Nacion Nivel 3');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal P2N Nv 4', 5, 1000, 1, 'Canal de Provincia a Nacion Nivel 4');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal P2N Nv 5', 6, 1000, 1, 'Canal de Provincia a Nacion Nivel 5');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal P2N Nv 6', 7, 1000, 1, 'Canal de Provincia a Nacion Nivel 6');
```

```
================== CANALES DE PROVINCIA A MUNICIPIO ================
insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal P2M Nv 0', 1, 1000, 1, 'Canal de Provincia a Municipio Nivel 0');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal P2M Nv 1', 2, 1000, 1, 'Canal de Provincia a Municipio Nivel 1');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal P2M Nv 2', 3, 1000, 1, 'Canal de Provincia a Municipio Nivel 2');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal Trsl P2M Nv 0', 1, 1000, 1, 'Canal Tralados de Provincia a Municipio Nivel 0');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal Trsl P2M Nv 1', 2, 1000, 1, 'Canal Tralados de Provincia a Municipio Nivel 1');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal Trsl P2M Nv 2', 3, 1000, 1, 'Canal Tralados de Provincia a Municipio Nivel 2');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal Not P2M Nv 0', 7, 1000, 1, 'Canal Notificaciones de Provincia a Municipio Nivel
0');

============= CANALES DE MUNICIPIO A PROVINCIA  ====================

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal M2P Nv 0', 1, 1000, 1, 'Canal de Municipio a Provincia Nivel 0');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal M2P Nv 1', 2, 1000, 1, 'Canal de Municipio a Provincia Nivel 1');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal M2P Nv 2', 3, 1000, 1, 'Canal de Municipio a Provincia Nivel 2');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal M2P Nv 3', 4, 1000, 1, 'Canal de Municipio a Provincia Nivel 3');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal M2P Nv 4', 5, 1000, 1, 'Canal de Municipio a Provincia Nivel 4');


insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal M2P Nv 5', 6, 1000, 1, 'Canal de Municipio a Provincia Nivel 5');

insert into sym_channel (channel_id, processing_order, max_batch_size, enabled, description)
values('Canal M2P Nv 6', 7, 1000, 1, 'Canal de Municipio a Provincia Nivel 6');
```

Anexo 2. Archivos symmetric.properties para la configuración de los nodos

Anexo 2.1 Nodo nacional

```
# The class name for the JDBC Driver

db.driver=org.postgresql.Driver

# The JDBC URL used to connect to the database

db.url=jdbc:postgresql://localhost:5432/dbmultas0100

# The user to login as who can create and update tables
db.user=postgres

# The password for the user to login as
db.password=desoft*09

my.url=http://localhost:8082/nac0100

group.id=nacional
external.id=0100
```

*En este ejemplo, el nodo nacional pertenece al grupo nacional y el external.id con que se identificará es el 0100, se conecta mediante JDBC a la BD "*dbmultas100" *que está ubicada localmente mediante el usuario "*postgres" *y la contraseña* "desoft*09". *La URL mediante la cual se pueden conectar a él es http://localhost:8082/nac0100.*

Anexo 2.2 Nodo provincial

```
# The class name for the JDBC Driver

db.driver=org.postgresql.Driver

# The JDBC URL used to connect to the database

db.url=jdbc:postgresql://localhost:5432/dbmultas05

# The user to login as who can create and update tables
db.user=postgres

# The password for the user to login as
db.password=desoft*09

# The HTTP URL of the root node to contact for registration
registration.url=http://localhost:8082/nac0100

my.url=http://localhost:8082/provi05

group.id=provincias
external.id=05
```

El extrenal.id que se le dio a los nodos provinciales está en correspondencia con el código de cada provincia que se encuentra en la tabla que contiene los datos de las provincias.

*En este ejemplo el nodo pertenece al grupo provincias y el external.id con que se identificará es el 05, se conecta mediante JDBC a la BD "*dbmultas05" *que está ubicada localmente mediante el usuario "*postgres" *y la contraseña* "desoft*09". *La URL mediante la cual se pueden conectar a él es http://localhost:8082/provi05 y se registra en http://localhost:8082/nac0100.*

Anexo 2.3 Nodo municipio

```
# The class name for the JDBC Driver

db.driver=org.postgresql.Driver

# The JDBC URL used to connect to the database
db.url=jdbc:postgresql://localhost:5432/dbmultas0501

# The user to login as who can create and update tables
db.user=postgres

# The password for the user to login as
db.password=desoft*09

# The HTTP URL of the root node to contact for registration
registration.url=http://localhost:8082/provi05

group.id=municipios
external.id=0501
```

Los external.id que se les dieron a los nodos municipales están en correspondencia con el código de cada OCCM que se encuentra en la tabla que contiene los datos de las mismas.

*En este ejemplo, el nodo pertenece al grupo municipios y el external.id con que se identificará es el 0501, se conecta mediante JDBC a la BD "*dbmultas0501" *que está ubicada localmente mediante el usuario "*postgres" *y la contraseña* "desoft*09". *La URL donde se registra en* http://localhost:8082/provi05.

Anexo 3. Ejemplos de triggers

Anexo 3.1 Clasificadores de nación a provincia nivel 0

De nación a provincia en el nivel 0 se replica 34 tablas, mostraremos ejemplo de 3.

```
insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_delete, initial_load_order,
last_updated_by,last_updated_time, create_time)
values('provincia', 'nacional', 'provincias', 'Canal N2P Nv 0', 1, 1, 1, 100,
'demo', current_timestamp, current_timestamp);
```

```
insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_delete, initial_load_order,
last_updated_by, last_updated_time, create_time)
values('tipo_ajuste', 'nacional', 'provincias', 'Canal N2P Nv 0', 1, 1, 1,
104, 'demo', current_timestamp, current_timestamp);
```

```
insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_delete, initial_load_order,
last_updated_by, last_updated_time, create_time)
values('tipo_calendario', 'nacional', 'provincias', 'Canal N2P Nv 0', 1, 1, 1,
135, 'demo', current_timestamp, current_timestamp);
```

Estas tablas se replican cuando ocurre una operación de inserción, actualización o eliminación sobre ellas.

Anexo 3.2 Clasificadores de nación a provincia nivel 1

De nación a provincia en el nivel 1 se replican 12 tablas, mostraremos el ejemplo de 3

```
insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_delete, initial_load_order,
last_updated_by, last_updated_time, create_time)
values('municipio', 'nacional', 'provincias', 'Canal N2P Nv 1', 1, 1, 1, 101,
'demo', current_timestamp, current_timestamp);
```

```
insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_delete, initial_load_order,
last_updated_by, last_updated_time, create_time)
values('concepto_ajuste', 'nacional', 'provincias', 'Canal N2P Nv 1', 1, 1, 1,
105, 'demo', current_timestamp, current_timestamp);
```

```
insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_delete, initial_load_order,
last_updated_by, last_updated_time, create_time)
values('calendario_detalle', 'nacional', 'provincias', 'Canal N2P Nv 1',
1, 1, 1, 112, 'demo', current_timestamp, current_timestamp);
```

Anexo 3.3 Clasificadores de nación a provincia nivel 2

De nación a provincia en el nivel 2 se replica 7 tablas.

```
insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_delete, initial_load_order,
last_updated_by, last_updated_time, create_time)
values('occm', 'nacional', 'provincias', 'Canal N2P Nv 2', 1, 1, 1, 103,
'demo', current_timestamp, current_timestamp);
```

```
insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_delete, initial_load_order,
last_updated_by, last_updated_time, create_time)
values('organismo_tipo_multa', 'nacional', 'provincias', 'Canal N2P Nv 2',
1, 1, 1, 108, 'demo', current_timestamp, current_timestamp);
```

Anexo 3.4 Clasificadores de provincia a municipio

Las mismas tablas una vez que llegan a las provincias bajan a los municipios, para esto en la propiedad sync_on_incoming_batch *de la tabla* sym_trigger *toma valor 1*.

```
insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_delete, sync_on_incoming_batch,
initial_load_order, last_updated_by, last_updated_time, create_time)
values('provincia', 'provincias', 'municipios', 'Canal P2M Nv 0', 1, 1, 1, 1,
105, 'demo', current_timestamp, current_timestamp);
```

```
insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_delete, sync_on_incoming_batch,
initial_load_order, last_updated_by, last_updated_time, create_time)
values('municipio', 'provincias', 'municipios', 'Canal P2M Nv 1', 1, 1, 1, 1,
110, 'demo', current_timestamp, current_timestamp);

insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_delete, sync_on_incoming_batch,
initial_load_order, last_updated_by, last_updated_time, create_time)
values('occm', 'provincias', 'municipios', 'Canal P2M Nv 2', 1, 1, 1, 1,
112, 'demo', current_timestamp, current_timestamp);
```

Anexo 3.5 Operaciones de municipio a provincia

En este trigger está presente la propiedad node_select, *para seleccionar los nodos que recibirán los cambios.*

```
insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_delete, node_select,
initial_load_order, last_updated_by, last_updated_time, create_time)
values('consejo_popular', 'municipios', 'provincias', 'Canal M2P Nv 0',
1, 1, 1, 'and external_id = (SELECT DISTINCT municipio.id_provincia
FROM current_occm Inner Join municipio ON
current_occm.id_occm = municipio.id_municipio)',102, 'demo',
current_timestamp, current_timestamp);
```

En este ejemplo node_select toma el valor

'and external_id = (SELECT DISTINCT municipio.id_provincia FROM current_occm Inner Join municipio ON current_occm.id_occm = municipio.id_municipio),

Esto significa que el nodo que se selecciona para replicar los datos es aquel que su external,id se corresponde con el código de la provincia de la OCCM actual, en la que se están ejecutando los cambios.

```
insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_delete, sync_on_update_condition,
sync_on_insert_condition, sync_on_delete_condition, node_select,
initial_load_order, last_updated_by, last_updated_time, create_time)
values('sub_submayor_general', 'municipios', 'provincias', 'Canal M2P Nv 2',
1, 1, 1,
'$(newTriggerValue).id_reporte=(SELECT sub_submayor_general.id_reporte FROM
sub_submayor_general WHERE (sub_submayor_general.mover =TRUE AND
sub_submayor_general.id_reporte = $(newTriggerValue).id_reporte))',
'$(newTriggerValue).id_reporte= (SELECT sub_submayor_general.id_reporte
FROM sub_submayor_general WHERE (sub_submayor_general.mover =TRUE AND
sub_submayor_general.id_reporte = $(newTriggerValue).id_reporte))',
'$(oldTriggerValue).id_reporte=(SELECT sub_submayor_general.id_reporte FROM
sub_submayor_general WHERE (sub_submayor_general.mover =TRUE AND
sub_submayor_general.id_reporte = $(oldTriggerValue).id_reporte))',
'and external_id = (SELECT DISTINCT municipio.id_provincia FROM current_occm
Inner Join municipio ON current_occm.id_occm = municipio.id_municipio)',
225, 'demo', current_timestamp,
current_timestamp);
```

En este trigger se especifica una condición para que se ejecuten los disparadores de eliminación, inserción y actualización, si esta condición se cumple entonces se ejecuta el trigger. Las siguientes variables son usadas:

- $(oldTriggerValue) *alias de la columna antes de la actualización o eliminación*
- $(newTriggerValue) *alias de columnas después de la actualización o inserción*

```
insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_delete,node_select, initial_load_order,
last_updated_by, excluded_column_names, last_updated_time, create_time)
values('contraventor', 'municipios', 'provincias', ' Canal M2P Nv 2 ', 1, 1, 1,
'and external_id = (SELECT DISTINCT municipio.id_provincia FROM current_occm
Inner Join municipio ON current_occm.id_occm = municipio.id_municipio)' ,106,
'demo', 'id_zona,id_centro',current_timestamp, current_timestamp);
```

En este caso con la utilización de la propiedad excluded_column_names *se excluyen de la replicación las columnas* id_zona *e* id_centro *en la tabla contraventor.*

Anexo 3.6 Operaciones de provincia a nación

```
insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_delete, sync_on_incoming_batch,
initial_load_order, last_updated_by, last_updated_time, create_time)
values('consejo_popular', 'provincias', 'nacional', ' Canal P2N Nv 0 ',
1, 1, 1, 1, 102, 'demo', current_timestamp, current_timestamp);

insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_delete, sync_on_incoming_batch,
sync_on_update_condition, sync_on_insert_condition, sync_on_delete_condition,
initial_load_order, last_updated_by, last_updated_time, create_time)
values('sub_submayor_general', 'provincias', 'nacional', ' Canal P2N Nv 2 ',
1, 1, 1,1,'$(newTriggerValue).id_reporte=
(SELECT sub_submayor_general.id_reporte FROM sub_submayor_general WHERE
(sub_submayor_general.mover =TRUE AND sub_submayor_general.id_reporte =
$(newTriggerValue).id_reporte))',
'$(newTriggerValue).id_reporte=(SELECT sub_submayor_general.id_reporte
FROM sub_submayor_general WHERE (sub_submayor_general.mover =TRUE AND
sub_submayor_general.id_reporte = $(newTriggerValue).id_reporte))',
'$(oldTriggerValue).id_reporte=(SELECT sub_submayor_general.id_reporte FROM
sub_submayor_general WHERE (sub_submayor_general.mover =TRUE AND
sub_submayor_general.id_reporte = $(oldTriggerValue).id_reporte))',
225, 'demo', current_timestamp, current_timestamp);


insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_delete,sync_on_incoming_batch,
initial_load_order, excluded_column_names, last_updated_by, last_updated_time,
create_time)
values('contraventor', 'provincias', 'nacional', 'Canal P2N Nv 2', 1, 1, 1,1,
106, 'demo','id_zona,id_centro ',current_timestamp, current_timestamp);
```

Anexo 3.7 Operaciones de nación a provincia

```
insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id, sync_on_insert,
sync_on_update, sync_on_incoming_batch, node_select, initial_load_order, excluded_column_names,
last_updated_by, last_updated_time, create_time)
values('contraventor', 'nacional', 'provincias', 'Canal Trsl N2P Nv 0', 1, 1, 1,
'and external_id = (select DISTINCT id_provincia from node_map where id_occm =
$(curTriggerValue).id_occm)', 250 ,'id_zona', 'demo', current_timestamp, current_timestamp);
```

```
insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_incoming_batch,sync_on_insert_condition,
sync_on_update_condition, node_select, initial_load_order, excluded_column_names,
last_updated_by, last_updated_time, create_time)
values('matriz', 'nacional', 'provincias', 'Canal Trsl N2P Nv 1', 1, 1, 1,
'$(newTriggerValue).cod_multa = (SELECT DISTINCT matriz.cod_multa FROM
mov_entrada_detalle Inner Join matriz ON mov_entrada_detalle.cod_multa = matriz.cod_multa
WHERE ((matriz.cod_multa = $(curTriggerValue).cod_multa) AND
(matriz.cod_multa = $(newTriggerValue).cod_multa)))',
'$(newTriggerValue).cod_multa=(SELECT DISTINCT matriz.cod_multa FROM mov_entrada_detalle
Inner Join matriz ON mov_entrada_detalle.cod_multa = matriz.cod_multa WHERE
((matriz.cod_multa = $(newTriggerValue).cod_multa) AND
(matriz.cod_multa = $(newTriggerValue).cod_multa)))',
'and external_id = (select DISTINCT id_provincia from node_map where
id_occm = $(curTriggerValue).id_occm)',270,'id_usuario,id_oc5' ,'demo',
current_timestamp, current_timestamp);
```

Anexo 3.8 Operaciones de provincia a municipio

```
insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_incoming_batch,
node_select, initial_load_order, excluded_column_names,last_updated_by,
last_updated_time, create_time)
values('contraventor', 'provincias', 'municipios', 'Canal Trsl P2M Nv 0', 1, 1, 1,
'and external_id = (select DISTINCT id_municipio from node_map where
id_occm = $(curTriggerValue).id_occm)', 250 ,'id_zona,id_centro', 'demo',
current_timestamp, current_timestamp);


insert into sym_trigger
(source_table_name, source_node_group_id, target_node_group_id, channel_id,
sync_on_insert, sync_on_update, sync_on_incoming_batch,
sync_on_insert_condition, sync_on_update_condition, node_select, initial_load_order,
excluded_column_names, last_updated_by, last_updated_time, create_time)
values('matriz', 'provincias', 'municipios', 'Canal Trsl P2M Nv 1', 1, 1, 1,
'$(newTriggerValue).cod_multa = (SELECT DISTINCT matriz.cod_multa FROM
mov_entrada_detalle Inner Join matriz ON mov_entrada_detalle.cod_multa = matriz.cod_multa
WHERE ((matriz.cod_multa = $(curTriggerValue).cod_multa) AND
(matriz.cod_multa = $(newTriggerValue).cod_multa)))',
'$(newTriggerValue).cod_multa=(SELECT DISTINCT matriz.cod_multa FROM
mov_entrada_detalle Inner Join matriz ON mov_entrada_detalle.cod_multa = matriz.cod_multa
WHERE ((matriz.cod_multa = $(newTriggerValue).cod_multa) AND
(matriz.cod_multa = $(newTriggerValue).cod_multa)))',
'and external_id = (select DISTINCT id_municipio from node_map where
id_occm = $(curTriggerValue).id_occm)',270,'id_usuario,id_oc5' ,
'demo', current_timestamp, current_timestamp);
```