UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA
FACULTAD DE CIENCIAS QUIMICAS E INGENIERIA
MAESTRIA Y DOCTORADO EN CIENCIAS E INGENIERIA

APLICACIÓN PARA LA REPLICACION DE DATOS EN SISTEMAS DISTRIBUIDOS CON CONECTIVIDAD LIMITADA.

TESIS PARA OBTENER EL GRADO DE

MAESTRO EN INGENIERÍA

Presenta:

Pablo Adamir Coronel Prado

Director de tesis:

Dr. Manuel Castañón Puga

Co-director de tesis:

Dr. Ricardo Rosales Cisneros

TIJUANA, BAJA CALIFORNIA, MÉXICO DICIEMBRE DE 2017

A mi familia
Quienes con su dedicación, ejemplo
Y todo su esfuerzo, me han formado
Y educado con amor y humildad,
Y me han ayudado a salir adelante
Todos mis triunfos son para ellos

Universidad Autónoma de Baja California

FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

COORDINACIÓN DE POSGRADO E INVESTIGACIÓN

FOLIO No. 228

Tijuana, B. C., a 24 de noviembre de 2017

C. Pablo Adamir Coronel Prado
Pasante de: Maestro en Ingeniería
Presente

El tema de trabajo y/o tesis para su examen profesional, en la Opción TESIS

Es propuesto, por los C. Dres. Manuel Castañón Puga y Ricardo Fernando Rosales Cisneros

Quienes serán los responsables de la calidad de trabajo que usted presente, referido al tema APLICACIÓN PARA LA REPLICACIÓN DE DATOS EN SISTEMAS DISTRIBUIDOS CON CONECTIVIDAD LIMITADA.

el cual deberá usted desarrollar, de acuerdo con el siguiente orden:

I.- INTRODUCCION
II.- MARCO TEORICO
III.- METODOLOGIA
IV.- DESARROLLO DE LA APLICACION
V.- PRUEBAS Y RESULTADOS
VI.- CONCLUSIONES Y TRABAJO FUTURO

UNIVERSIDAD AUTONOMA DE BAJA CALIFORNIA

Dr. Manuel Castañón Puga
Director de Tesis

Dr. José Luis González Vázquez
Sub-Director Secretario

FACULTAD DE CIENCIAS QUIMICAS E INGENIERIA

Dr. Ricardo Fernando Rosales Cisneros
Co-Director de Tesis

Dr. Luis Enrique Palafox Maestre
Director

Agradecimientos.

Deseo agradecer a la Universidad Autónoma de Baja California (UABC), por darme la oportunidad de ingresar al Programa de Maestría y Doctorado en Ciencias e Ingeniería.

A mi director de tesis, Dr. Manuel Castañon Puga, por el tiempo dedicado, el apoyo brindado, su paciencia, ayuda y guía para que mi trabajo fuera concluido con éxito.

A mi jefe, el M.I Edaniel Figueroa, por darme la oportunidad de continuar mejorando continuamente, por permitirme continuar con mis estudios y superarme tanto personal y profesionalmente, por brindarme su apoyo y conocimiento para concluir con mi trabajo de manera exitosa.

A mi hermano y a mi madre, por estar siempre impulsándome a mejorar, a crecer en lo académico y en lo profesional.

Resumen

En este documento se propone una solución para la homologación de información en sistemas con bases de datos distribuidas, y de esta manera, proporcionar la correcta operación de un sistema sin necesidad de depender de una base de datos centralizada.

Como solución a dicha problemática, se documenta la construcción una herramienta capaz de mantener una base de datos en un sistema local homologada con la información en una base de datos centralizada.

Como caso de estudio, se identificó en una empresa mexicana ubicada en la ciudad de Tijuana, un sistema de punto de venta el cual funciona con una base de datos remota, pero que presenta diversos problemas externos. La principal problemática es la perdida continua de comunicación, dejando el sistema en la sucursal inoperable. Se analizó la situación actual de la organización y se planteó una solución de replicación de datos con la finalidad de proveer un sistema independiente de una base de datos central.

Abstract

In this document, we propose a solution for the homologation of information in systems with distributed databases, and in this way to provide, the correct operation of a system without the need to depend on a centralized database.

As a solution to this problem, we documented the construction as a tool capable of maintaining a database in a local system homologated with the information in a centralized database.

As a case study, was identified in a Mexican company, located in the city of Tijuana, a point of sale system which works with a remote database, but presents several external problems, the main problem is the consecutive loss of communication, leaving the system in the branch inoperable. We analyzed the current situation of the organization, and we proposed a data replication solution to provide an independent operation of a central database.

INDICE

INDICE DE FIGURAS

INDICE DE CUADROS

Capítulo 1

1. Introducción

Con la heterogeneidad de la red, es vital que los problemas de disponibilidad de información sean los menores posibles y que los problemas de movilidad se consideren en el diseño de cualquier sistema. La satisfacción de la creciente demanda de servicio de los usuarios depende de la disponibilidad y calidad del servicio solicitado. Los proveedores de servicios siempre buscan la manera de como maximizar sus ganancias, sin embargo, es necesario tener en cuenta la satisfacción de los usuarios proporcionando soluciones que les permitan maximizar los beneficios del servicio.

Uno de los grandes desafíos para los sistemas de computación en la nube es proporcionar de la calidad del servicio (QoS). Por lo tanto, se debe proporcionar a los usuarios una solución altamente dinámica y adaptable para el servicio y el acceso a los datos, así como mantener un nivel aceptable de satisfacción de la QoS al usuario[1].

En empresas con múltiples sucursales, es de suma importancia el poder continuar con la operación del día aun cuando existan problemas de comunicación los dispositivos que se encuentran en la sucursal y el servidor central en el cual se encuentra alojada toda la información.

En la actualidad, existe la necesidad de compartir y hacer uso de la información en tiempo real; Un ejemplo de esto, es la implementación de sistemas de información centralizada, la cual consiste en poder acceder y hacer uso de la información sin importar el lugar en el que esta se encuentre. Esto se logra mediante la implementación de una arquitectura cliente servidor, en la cual se tiene un servidor central, que es donde se concentra toda la información, y los dispositivos clientes son quienes harán uso de esta información. Pero dicha arquitectura tiene sus desventajas, una de ellas es la inoperabilidad del sistema cuando se pierde la comunicación entre los dispositivos, ya que para que sea un sistema completamente funcional, es necesario poder manipular en su totalidad la información.

Se tomará como referencia el sistema “Punto de venta Mail”, que es la parte fundamental para el manejo de inventario del sistema FACMAIL ®, un sistema desarrollado en conjunto entre la empresa Tecnología en Comunicaciones e Identificaciones de México S.A. de C. V., Punto de

Desarrollo de Software S.A. de C.V., el Instituto Politécnico Nacional y la Universidad Autónoma de Baja California.

Al tratarse de un proyecto de innovación tecnológica, es necesario plantear soluciones que utilicen y complementen las herramientas digitales más comunes disponibles en la actualidad.

En este documento se describe el desarrollo e implementación de una herramienta que es capaz de mantener la información de todos los dispositivos que hacen uso de un sistema actualizado, aun cuando estos tengan una conexión entre ellos intermitente, o limitada, y sin que el usuario tenga conocimiento sobre gestores de bases de datos.

1.1 Planteamiento del problema

El sistema FACMAIL está integrado por diversas aplicaciones que apoyan los procesos internos de las empresas: un sistema de facturación electrónica, un sistema de contabilidad electrónica, un sistema de ventas y un sistema de inventarios, los cuales se intercomunican y comparten información en tiempo real.

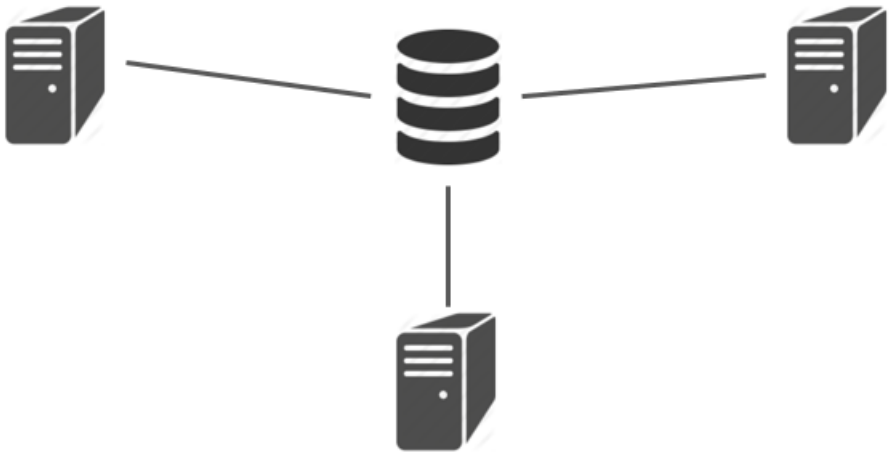


Figura 1.1 Arquitectura convencional de servidor de base de datos

Estos sistemas utilizan una arquitectura basada en Cliente/Servidor (véase Figura 1.1). Dicha arquitectura presenta una dependencia total del sistema hacia el servidor central. Si la comunicación entre ellos se interrumpe, el sistema deja de ser operable hasta que la comunicación se restablece.

Al no existir una comunicación al servidor central (véase Figura 1.2), la operación dentro de la empresa se detiene por completo, ya que no se tiene acceso a la información, ni si quiera a la información propia. Por ejemplo, una sucursal fuera de la matriz no tiene control ni uso alguno sobre su inventario y mucho menos sobre el inventario general.

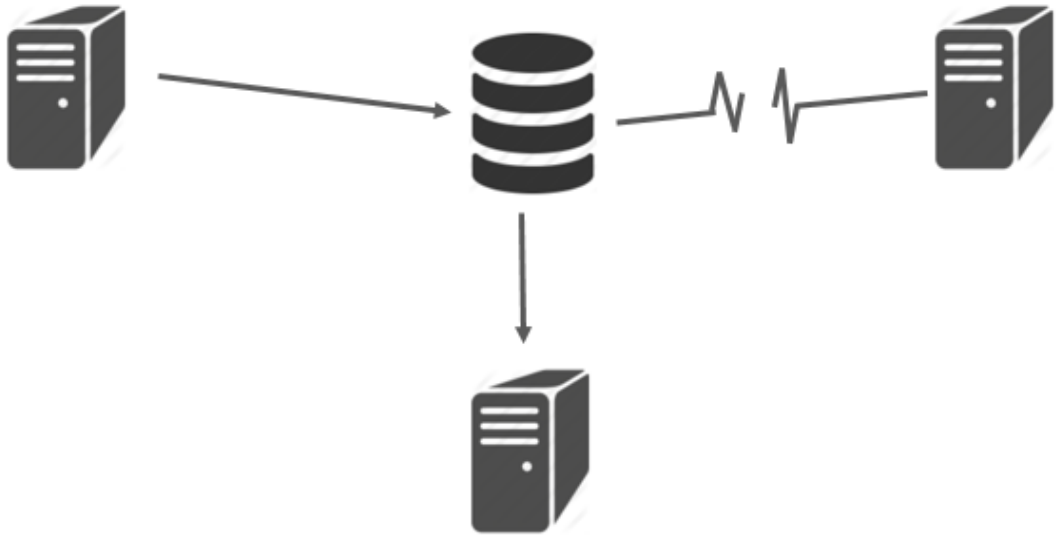


Figura 1.2 Perdida de comunicación entre dispositivo y servidor

A pesar de ser un modelo muy convencional, en la actualidad aún presenta una limitante, la continua comunicación con el servidor. Aún con los avances tecnológicos, no es posible asegurar que el servicio se encontrará siempre disponible ya que aún existen causas ajenas que afectan la operación de este modelo, por ejemplo, un ataque cibernético a los servidores centrales que pueden comprometer la disponibilidad de los servicios otorgados.

1.3 Justificación

FACMAIL es un sistema basado en una estructura de negocio funcional, por lo que no debería de presentar ninguna limitante, es decir, debería de ser capaz de funcionar aun cuando uno de los principales requerimientos (conexión continua) no se encuentre disponible temporalmente.

Dicho sistema exige la manipulación constante de la información, por lo que la arquitectura definida para dicho sistema no debería de ser una barrera para el correcto funcionamiento del mismo. En general, cualquier proyecto informático debe de ser robusto, confiable y completamente funcional.

Lo anterior nos llevó a analizar con profundidad y a desarrollar una herramienta capaz de sincronizar la información entre ambos dispositivos (siempre y cuando el modelo de datos sea el mismo). El enfoque es que sea fácil de utilizar por un usuario común, es decir, sin necesidad de utilizar un gestor de base de datos, y sin que tenga conocimiento alguno de la arquitectura de dicho sistema.

1.4 Hipótesis

Desarrollando una aplicación que sea capaz de replicar la información desde un punto A al punto B, será posible de aprovechar por completo los recursos del sistema, aun cuando existe una limitante en la conexión entre las interfaces implicadas.

1.5 Antecedentes

Los grandes avances tecnológicos de la última década, han permitido que los sistemas distribuidos sean completamente confiables y rentables, permitiendo manipular grandes cantidades de información aun cuando esta se encuentre a cientos de kilómetros de distancia [2]–[4].

En dichos sistemas la información suele alojarse en más de un servidor, centralizando todo el conjunto de datos en un servidor central, al cual se puede acceder de manera directa desde cualquier equipo dentro de la misma red de comunicación, ya sea desde un cliente o desde un servidor en una ubicación distinta [5], [6].

La sincronización de información se conoce como replicación, y la replicación como tal se refiere a la homologación de información en más de una base de datos, ya sea en un sistema de base de datos centralizado, o como en un sistema de bases de datos distribuido [5].

La replicación de datos es un conjunto de tecnologías que permiten realizar él envió/recepción tanto información como objetos desde una base de datos a otra, sin importar la plataforma, ni la región geográfica en la que esta se encuentre [2], [5].

Uno de los grandes problemas es que dichos sistemas deben de presentar una comunicación continua hacia el servidor central, si la comunicación es interrumpida, detiene la operación del mismo [5], [7].

Ha existido un gran interés en los problemas que la replicación de datos conlleva, aunque últimamente el enfoque que se la ha dado es sobre las aplicaciones móviles, por la importancia y la necesidad de mantener la información lo más integra posible [8]–[10].

1.6 Objetivos

Objetivo general

El objetivo general es el desarrollar e implementar una herramienta dentro del sistema FACMAIL, con el cual se mantenga la información actualizada de todas las interfaces que formen parte de dicho sistema, inicialmente enfocado en el punto de venta y el proceso para realizar una venta de un producto, y sin contar con comunicación al servidor central (véase Figura 1.3)

Figura 1.3 Sucursales no dependientes del servidor central

Objetivos específicos

1. Centralizar toda la información de cada una de las sucursales de manera efectiva.
2. Que los inventarios de las sucursales no dependan de una conexión continua.
3. Sincronizar la información de manera correcta entre las distintas interfaces del sistema, desde cualquier red, ya sea pública o una intranet.

1.7 Metas

1. Instalar en un ambiente de producción.
2. Sincronizar la información entre dos interfaces de la aplicación.
3. Realizar operaciones en la aplicación sin comunicación con el servidor central.

1.8 Alcances del trabajo

El alcance de este trabajo está delimitado a la replicación de datos de las tablas implicadas en el inventario de la base de datos del sistema FACMAIL-Punto de venta, de manera que se pueda operar de manera efectiva el sistema sin tener una comunicación constante con el servidor central.

Fase 1 Definir una metodología para el desarrollo de la aplicación.

Se define cual es la mejor metodología para llevar a cabo la solución, diseño e implementación de la aplicación. Se define la solución con suficiente detalle para permitir su interpretación y realización física así como el diseño de la aplicación, clases y métodos necesarios.

Fase 2 Verificaciones sobre el catálogo de información para las validaciones necesarias.

Definir las consultas sobre el catálogo de información de la base de datos CubeERP_0.2.2 para la validación de los modelos de datos antes de realizar la replicación de datos entre ellos.

Fase 3 Desarrollo de la aplicación

En esta fase se desarrolla la aplicación implementando las verificaciones sobre el catálogo de información del modelo de datos CubeERP_0.2.2 para la validación de la estructura de los modelos de datos a replicar.

Fase 4 Implementación y pruebas.

En esta fase se asegura que la solución funcione de acuerdo a los requerimientos y que los usuarios puedan operarlo de una manera completamente transparente.

1.9 Como está organizada esta tesis

El presenta trabajo consta de seis capítulos. En el primer capítulo se encuentra una descripción del caso de estudio, un resumen breve de sus antecedentes, objetivos, delimitaciones y alcances del mismo. Durante el segundo capítulo hace una revisión sobre el estado de la técnica, y los fundamentos teóricos que sirven de base para las técnicas, lenguajes de programación, propuestas de solución y el desarrollo de la aplicación. El tercer capítulo describe la metodología y la forma de trabajo seleccionada para el desarrollo del proyecto. En el cuarto capítulo, se explica la fase de desarrollo de la aplicación. En el quinto y penúltimo capítulo, se describen las pruebas realizadas, explicando la función en base a la metodología propuesta y los resultados obtenidos. Por último, en el sexto capítulo se presentan las conclusiones, y las oportunidades de continuidad del trabajo.

2. Marco teórico

2.1 Sistema de base de datos

Un sistema de base de datos es un sistema computarizado mediante el cual podemos almacenar y consultar información en un sistema computarizado (véase Figura 2.1) . Su finalidad es almacenar y permitir a los usuarios recuperar y actualizar la información en base a peticiones [11].

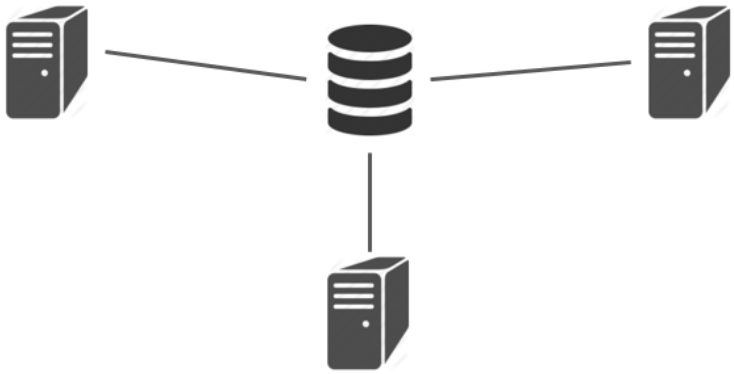


Figura 2.1 Servidor de base de datos centralizado

2.2 Bases de datos

Una base de datos es un repositorio de información, diseñado para almacenar, consultar y mantener la información de manera eficiente. Existen distintos tipos de bases de datos, las cuales se adecuan según a las necesidades que existan en la industria.

Una base de datos puede especializarse para almacenar un tipo de dato en específico, ya sean datos binarios, documentos, imágenes, videos, datos relacionales, datos multidimensionales, datos transaccionales, analíticos o geográficos, solo por mencionar algunos [12].

Los datos pueden ser almacenados de distintas formas, puede ser tabular, jerárquicamente, o mediante grafos. Cuando los datos son almacenados en forma tabular, se le conoce como base de datos relacional, cuando los datos se organizan en una estructura de árbol, es conocido como base de datos jerárquica, y por último, cuando los datos se almacén en forma de grafos los cuales representan relaciones entre objetos, se le conoce como base de datos en red [12].

Una base de datos en una colección organizada de símbolos leíbles por una computadora, las cuales se pueden interpretar como un conjunto de datos con una relación completa. También se puede interpretar como una colección de variables, debido a que pueden ser actualizadas en cualquier momento [13].

Una base de datos es una colección de registros interrelacionados, la cual se puede componer por datos o metadatos [14].

2.3 Metadatos

Los metadatos, se definen como la información que describen el orden y la estructura de un registro dentro de la base de datos [14].

Todos los manejadores de bases de datos proporcionan una función de diccionario de datos, dicho diccionario de datos puede ser visto como una base de datos del sistema, en la cual se almacena la definición de otros objetos del sistema, a este diccionario se le conoce como Metadatos o Descriptor.

En dicho diccionario podemos encontrar información de los diversos esquemas contenidos dentro de nuestro servidor de base de datos y cada una de sus transformaciones (externos, conceptuales, etc.)

2.4 Manejador de Bases de datos

También se le conoce como *Database management system* (DBMS), es un conjunto de herramientas de software las cuales permiten tener el control, el acceso, la administración y el mantenimiento de una base de datos [12].

Un manejador de base de datos es un conjunto de programas utilizados para definir, administrar y procesar bases de datos y sus respectivas aplicaciones, en pocas palabras, un DBMS es la herramienta que se utiliza para construir la estructura de una BD y manipular la información contenida en la misma [14].

2.5 Base de datos relacional

Una base de datos relacional se define como un conjunto de símbolos organizados dentro de una colección de relaciones [13].

Una base de datos relacional es un modelo simple y elegante, el cual tiene sus bases fundamentadas en teoría matemática y predicados de cálculo, es uno de los modelos para bases de datos más usados hoy en día.

- Se almacena la información en estructuras simples (tablas)
- Se accede a la información utilizando un lenguaje de alto nivel, conocido como *Data Manipulation Language* (DML).
- Es independiente del almacenamiento físico.

Con una estructura de datos simple, es fácil mantener la independencia de la información lógica. Con un lenguaje de alto nivel es fácil mantener la independencia de la información física [12].

2.6 Lenguaje SQL

Structured Query Language (SQL) es un lenguaje de alto nivel el cual permite a los usuarios manipular datos relacionales. Una de las ventajas de SQL, es que el usuario solo necesita especificar qué información es la que se necesita sin tener la necesidad de saber cómo obtenerla. El sistema administrador de la base de datos, es el responsable de proveer la ruta de acceso para obtener la información necesaria.

SQL es el lenguaje universal para las bases de datos relacionales, contiene palabras clave las cuales vienen del idioma inglés, que son fáciles de utilizar y comprender.

SQL tiene tres categorías basadas en la funcionalidad implicada:

- DDL – *Data Definition Language.*
- DML – *Data Manipulation Language.*
- DCL – *Data Control Language*. [12]

Los comandos SQL se encuentran agrupados en categorías: DDL, DML, DCL y *Transaction Control Language* (TCL) [5].

2.7 Lenguaje de definición de datos DDL

Data Definition Language, es un vocabulario utilizador para definir estructuras de datos en SQL. Se utilizan estas sentencias para crear, modificar o eliminar estructuras en una instancia de SQL [15].

DDL es utilizado para definir, modificar o eliminar objetos de la base de datos [12].

Los comandos DDL son aquellos que nos permiten definir o alterar la estructura de la base de datos [5].

En el lenguaje de definición encontramos las sentencias que nos permiten realizar modificaciones sobre el modelo físico de la base de datos, ejemplo, la modificación un objeto como una tabla, procedimiento almacenado o una vista la realizamos con la sentencia Alter, de la misma manera encontramos la sentencia *Create* que nos sirve para la creación de nuevos objetos, índices o *triggers*, y así como podemos crear objetos, también podemos eliminarlos con la sentencia *Drop*.

2.8. Lenguaje de manipulación de datos DML

Data Manipulation Language, es utilizado para modificar o consultar la información almacenada en la base de datos [12].

Los comandos DML son aquellos que nos permiten manipular la información, crear, leer, actualizar o eliminar [5].

En el lenguaje de manipulación de datos, podemos encontrar las sentencias que nos permiten manipular la información almacenada, con dichas sentencias podemos insertar, seleccionar, actualizar y eliminar registros, y también podemos encontrar sentencias para procesos específicos como la declaración de variables y cursores.

2.9. Lenguaje de control de datos DCL

Data Control Language, es utilizado para conceder o remover autorizaciones a objetos o procesos de una base de datos [12].

Los comandos DCL no permiten tener el control del acceso a la información, ya sea sobre los privilegios de un usuario, tabla o campo [5].

En el lenguaje de control de datos encontramos las sentencias que nos permiten otorgar y revocar permisos y derechos tantos a usuarios como a procesos dentro del mismo servidor, e inclusive a bases de datos.

2.10. Lenguaje de control transacciones TCL

Los comandos TCL nos permiten utilizar transacciones. Una transacción es un conjunto de comandos que deben de ser ejecutados de manera atómica, como una sola unidad, para asegurar que todos los comandos se ejecuten o ninguno de ellos lo haga [5].

En el lenguaje de control de transacciones, encontramos las sentencias que nos permiten trabajar con un conjunto de operaciones dentro de un mismo bloque el cual definimos como transacción, las sentencias que forman parte a este lenguaje son el *Begin, Commit, Rollback* y *Save*.

2.11. Modelo de datos

El modelo de datos es la representación conceptual de una base de datos, mediante la cual podemos conocer su estructura, las relaciones entre los datos [11].

El modelo de datos nos permite entender e interpretar de una forma abstracta como se representan los datos en un sistema de información o un sistema de gestión. Básicamente es la descripción física de del contenedor o base de datos.

Se puede definir modelo de datos como un conjunto de conceptos, reglas y convenciones que permiten describir los datos del universo, así como especificar y manipular los datos que se desean almacenar [16].

2.12. Esquema de datos (Information Schema)

El esquema de información es un conjunto de tablas cuyo contenido refleja la definición de los demás esquemas del catálogo, es decir, en dicho catalogo está contenida la representación lógica de una base de datos.

Un esquema de datos es el descriptor de una base de datos individual, puede existir n cantidad de catálogos, los cuales pueden estar a su vez subdivididos en n cantidad de esquemas, sin embargo, cada catalogo debe de incluir exactamente un esquema denominado INFORMATION_SCHEMA, el cual es el encargado de almacenar la estructura lógica de nuestra base de datos [11].

2.13. Catálogos de sistema de bases de datos

El catálogo de información del sistema, nos provee de otra opción para acceder a los metadatos de una base de datos independiente de las tablas del sistema.

Dichos catálogos están nombrados como INFORMATION_SCHEMA dentro del gestor de base de datos de SQL, en dichos catálogos podemos encontrar las llaves, columnas, tipo de dato,

privilegios, referencias, tablas etc. De aquellas bases de datos distintas a la base de datos del gestor SQL [17].

La vista de *Information Schema* es uno de los distintos métodos que se puede encontrar en SQL Server para obtener información de los catálogos de metadatos. Las vistas de *Information Schema* proveen un sistema de tablas independientes de los metadatos contenidos en los catálogos de SQL Server [18].

```
SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, COLUMN_DEFAULT
FROM AdventureWorks2012.INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = N'Product';
GO
```

Cuadro 2.1 Uso del information schema (Fuente: C.J)

Las vistas de *Information Schema* están definidos bajo un esquema de notación especial INFORMATION_SCHEMA [11]. (Véase Figura 2.1)

2.14. Replicación de datos

La replicación de datos es la sincronización entre dos o más instancias de SQL Server véase , usualmente, copiando la información desde un Publicador (*Publisher*) quien es el origen de la información hacia los suscriptores, quienes serán los que alojaran un bloque de la información total [19].

La replicación de datos es una manera de satisfacer la integración de información de manera continua. La replicación permite realizar la sincronización de datos desde distintos puntos hacia un servidor central otorgando disponibilidad sobre la información, administrando de manera eficiente la información conforme esta crece, además de la explotación en tiempo real.

La replicación de datos en tiempo real enriquecen los sistemas de información, ya que garantizan la disponibilidad de la información sin importar en donde se encuentre [20].

Figura 2.2 Replicación de datos de base A hacia base B

3. Metodología

3.1. Metodología de desarrollo de software.

En la actualidad, existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Por una parte tenemos aquellas propuestas más tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán.

Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos [21].

Existen propuestas que se centran en los procesos de desarrollo, actividades, artefactos y restricciones, las cuales se conocen como metodologías tradicionales, y por otra parte, tenemos metodologías que se centran en otras dimensiones, por ejemplo en el factor humano o el producto software, estas metodologías son conocidas como metodologías agiles [22].

3.1.1 Metodología de desarrollo ágil.

Una metodología de desarrollo ágil es un esquema de trabajo en el cual se le da un mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas, a diferencia de las metodologías tradicionales en donde se realiza un solo entregable y todo queda estipulado bajo un contrato véase Figura 3.1. Este proceso ha mostrado su efectividad en proyectos con requisitos muy cambiantes y cuando se exige la reducción drástica de los tiempos de desarrollo sin descuidar la calidad del producto final [22].

Las metodologías agiles se basan en un manifiesto.

- Importancia al individuo e interacciones del equipo de desarrollo sobre el proceso y las herramientas.
- Desarrollas software que funciona es más importante que conseguir una buena documentación.
- La colaboración con el cliente más que la negociación de un contrato.

- Se flexible a los cambios y no apegar a un plan de desarrollo.

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Figura 3.1 Comparativa entre metodologías (Fuente: Canos, Letelier)

Entre las metodologías Agiles podemos encontrar *Scrum*, Crystal Methodologies, Dynamic Systems Development Method, Adaptive Software Development, Feature-Driven Development y Lean Development [22].

3.1.2 Metodología de desarrollo Scrum

Es un esquema de trabajo mediante el cual un equipo puede resolver problemas complejos de manera iterativa, entregando un producto del mayor valor posible al final de la iteración.

Scrum no es un proceso o técnica para la construcción de un producto, es un esquema de trabajo en el cual se pueden implementar diversos procesos y técnicas para cumplir con el objetivo [23].

3.2. El estado de la técnica

3.2.1 Tecnologías disponibles

En este apartado se describen algunas tecnologías que se encuentran en el mercado que contemplan la replicación de datos utilizando distintas tecnologías y metodologías en particular, se describen de manera breve a continuación:

MICROSOFT SYNC FRAMEWORK Es una plataforma de sincronización que permite la colaboración y el acceso a sistemas, aplicaciones, servicios y dispositivos que se encuentren fuera de línea. Esta plataforma proporciona tecnologías y herramientas que habilitan la compartición de datos y la

manipulación fuera de línea. Al utilizar Sync Framework los desarrolladores puede construir ambientes de sincronización que se pueden integrar en cualquier aplicación con almacenamiento de información utilizando cualquier protocolo de cualquier red [24].

3.2.2 Productos disponibles en el mercado

En la actualidad existen diversas herramientas que permiten realizar la sincronización y replicación de información entre bases de datos homologadas, dichas herramientas tienen precios que varían desde unos cientos hasta unos miles de dólares.

SQL DATA COMPARE Herramienta comercial para realizar la comparación y homologación de información entre dos bases de datos con el mismo esquema, utilizando una herramienta gráfica completamente automatizada o generando un script a ejecutar en la BD destino [25].

SQL ADMIN STUDIO Herramienta gratuita de la compañía Simego con la cual se puede administrar, comparar y sincronizar información entre bases de datos [26].

VISUAL STUDIO (Compare and Synchronize Database Schemas) Herramienta incluida dentro de la paquetería de Visual Studio con la cual se puede realizar la comparación de esquemas de bases de datos, así como de la sincronización de la información ente dichas bases de datos [27].

3.3 Metodología

Para el desarrollo de este proyecto, se seleccionó una metodología incremental basada en iteraciones y revisiones, similar a la metodología SCRUM, no se cuenta con la cantidad de integrantes necesarios para que sea considerada como metodología SRUM véase Figura 3.2.

Se toma como requerimiento inicial el desarrollo de una aplicación para la replicación de datos, la cual se realizará en N cantidad de iteraciones, durante cada iteración se estará añadiendo una nueva funcionalidad al sistema (procedimiento, método, validación) la cual consideraremos nuestro entregable del sprint.

Figura 3.2 Representación gráfica de Scrum

Este modelo se caracteriza por:

• Adoptar una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto.

• Basar la calidad del resultado más en el conocimiento tácito de las personas en equipos auto organizados, que en la calidad de los procesos empleados.

• Solapamiento de las diferentes fases del desarrollo, en lugar de realizar una tras otra en un ciclo secuencial o de cascada.

Esta metodología obliga a todos los involucrados en el proyecto a sostener reuniones periódicas (diarias) que provean un panorama de los avances en el ciclo en el que se encuentran. Además, se establece desde un inicio, con qué frecuencia se declarará el sprint como terminado para que forme parte del incremental, que en un final será el producto terminado [28].

3.4. Descripción del caso de estudio y planeación del desarrollo.

Para la elaboración de la solución se seleccionó una metodología incremental basada en SCRUM, la cual permite adaptarse fácilmente a los estándares de programación y diseño de empresa objeto de estudio.

Se decidió utilizar esta metodología por la flexibilidad que presenta a la hora de la construcción del producto, el cual se especifica de manera general, pero evoluciona a lo largo del proyecto.

El proyecto fue dividido en pequeñas partes que a su vez comprenden una entrega funcional de la solución.

- Definir el flujo de la aplicación.
- Identificar las consultas sobre el catálogo de información para la validación de los modelos de datos.
- Definir los métodos de validación del modelo de datos.
- Diseñar las clases necesarias para la validación del modelo de datos.
- Diseñar la aplicación.

3.5. Desarrollo de prototipos

Cada iteración del desarrollo constara de una serie de actividades que permitirán asegurar la calidad de la solución a fin de cumplir con cada una de las entregas establecidas.

Análisis de la solución. Durante esta etapa se determinan con precisión cada una de las tareas que deberá realizar la solución y como se logrará realizarlas. Y provee la información necesaria para las siguientes etapas.

Diseño de la solución. En base a la selección de la etapa anterior, se determinó el flujo del proceso que se requiere para realizar cada una de las tareas con apoyo de la técnica de diagramas de flujo listos para pasarse a la etapa de desarrollo.

Desarrollo de la solución. Etapa que consiste propiamente en el proceso de desarrollo del producto de software, siguiendo los requerimientos establecidos en la etapa anterior.

Pruebas. Durante esta etapa, la versión del software creada se someterá a diversas pruebas de calidad, funcionalidad y de concepto, con la finalidad de detectar las debilidades y errores para ser corregidos. Los resultados de las pruebas serán registrados y anexados a este documento.

3.6 Herramientas para el desarrollo

Durante la etapa de construcción del producto de software se utilizaron herramientas de desarrollo para cumplir con las actividades a realizar en base al requerimiento inicial, la documentación, el seguimiento de tareas y el desarrollo de software.

Microsoft SQL Server 2012 El manejador de base de datos Microsoft SQL Server 2012 se utilizó como el caso de estudio principal.

ER-Studio Herramienta utilizada para consultar el modelo de la base de datos, así como el sub-modelo que se desea sincronizar entre ambas bases de datos.

Microsoft Visual Studio 2012 (C#) Ambiente de desarrollo utilizado para la construcción del sistema de replicación de datos.

4 Desarrollo de la aplicación

El desarrollo de la aplicación se realizó en dos fases en base a las necesidades de funcionalidad. Primeramente, fue necesario obtener acceso a los catálogos de sistemas de bases de datos y realizar un análisis de los mismos, para identificar las tablas y vistas en donde se encuentra alojada la información de cada uno de los objetos de la base de datos que se está utilizando y conocer cómo se encuentra almacenada la relación existente entre múltiples objetos, así como las vistas y tablas que nos permitan identificar y conocer las propiedades del objeto en cuestión.

Y como segunda etapa se trata de trasladar dicha información a lógica de programación utilizando un entorno de desarrollo de Visual Studio con el lenguaje de programación C# para implementar toda la serie de reglas necesarias para realizar la evaluación de la información almacenada en las bases de datos que se están intentando homologar.

4.1 Arquitectura de la solución

Se diseñó una aplicación pensando en el usuario final, que esta tiene que ser lo más fácil de utilizar posible. La aplicación está compuesta por un conjunto de formas (vistas) que definen y solicitan al usuario la información relevante para determinar cuál es la base de datos que se estará tomando como base, y cuál es la base de datos hacia donde se desea trasladar la información, así como los usuarios y contraseñas necesarios para establecer la comunicación con ambos servidores. Véase Figura 4.1.

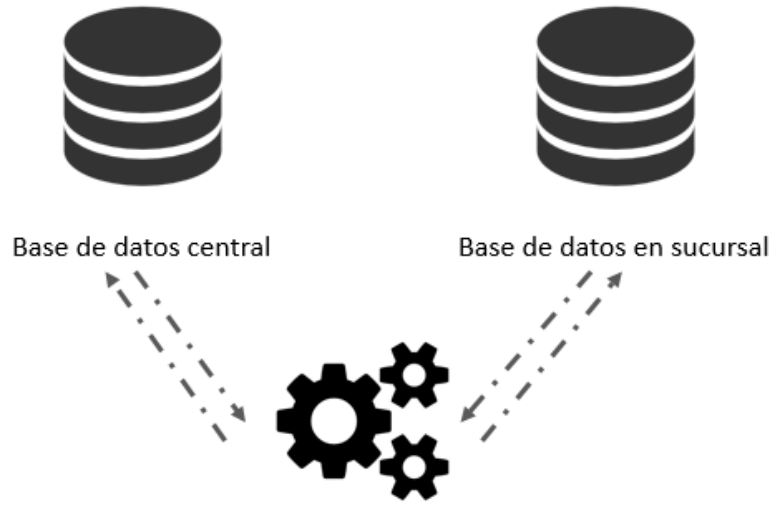


Figura 4.1 Arquitectura de la aplicación

La configuración de las cadenas de conexión es una de las partes fundamentales de la aplicación debido a que sin ellas no es posible realizar el análisis, verificación y comparación de las bases de datos.

Como configuración adicional, el usuario necesita especificar la tabla que se estará utilizando como base, mediante el catálogo de sistemas de bases de datos la aplicación determinara todas las tablas relacionadas a nuestra tabla base, y evaluara cuales objetos representan un catálogo de sistema, una tabla de encabezado o una tabla de detalle según el estándar que lleva la empresa en sus modelos de datos.

La aplicación va notificando al usuario en que paso se encuentra y cada una de las operaciones que se están realizando.

4.2 Mecanismo para evaluar modelo de datos

Tal como se ha venido mencionando, el mecanismo utilizado para evaluar los modelos de datos depende enteramente del catálogo de sistemas de bases de datos, el usuario proporciona la tabla base y mediante distintas consultas a dicho catálogo de sistemas, se comienzan a determinar las reglas necesarias para poder realizar una homologación de información, así como la cantidad de tablas que se requiere evaluar por las relaciones que puedan existir.

4.2.1 Consultas al catálogo de sistemas de bases de datos

El catálogo de sistemas de bases de datos ayuda a determinar las distintas propiedades de cada uno de los objetos existentes de una base de datos, así como las relaciones que existen entre distintos objetos.

Las propiedades que se evalúan mediante el catálogo de sistemas de bases de datos son si el modelo de datos es el mismo especificado en el origen y en el destino, es decir, si ambas bases de datos cuentan con la misma cantidad de tablas, cuando se cumple dicha condición, se procede a determinar el sub-modelo al que corresponde la tabla base para evaluar si las tablas del sub-modelo

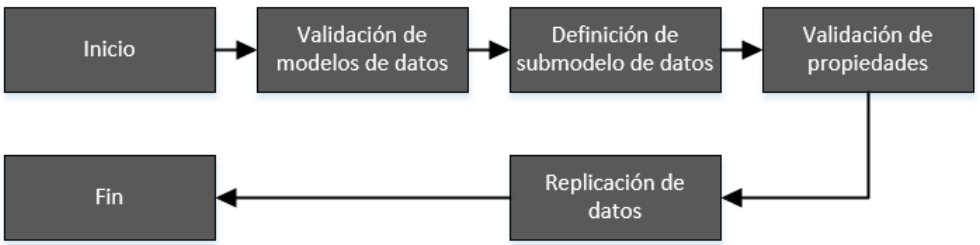


Figura 4.2 Pasos para la replicación de datos

tienen las mismas propiedades en ambos servidores, si la cantidad de columnas es la misma y si el tipo de dato de las columnas corresponde en ambos modelos véase Figura 4.2.

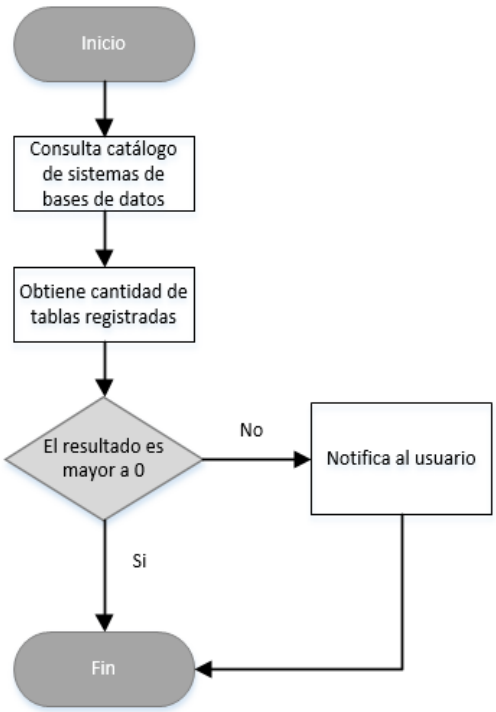
Verificación de modelos de datos Utilizando el catálogo de sistemas de bases de datos, se generó una consulta que nos permite realizar una validación rápida del modelo de datos véase Cuadro 4.1, de manera inicial si la cantidad de tablas es la misma entre la base de datos origen y la base de datos destino, ya que si hubiese alguna diferencia el sistema puede que no realice una homologación de la información de manera concreta.

```
SELECT  tc2.TABLE_NAME , ( SELECT COUNT(1) FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = tc2.TABLE_NAME) AS COLUMN_COUNT
FROM    INFORMATION_SCHEMA.TABLES tc2 WHERE TABLE_TYPE = 'BASE TABLE'
```

Cuadro 4.1 Obtención de objetos del modelo de datos

La lógica trasladada al lenguaje de programación es realizar la consulta sobre el catálogo de sistemas de bases de datos, obtener la cantidad de tablas existentes bajo la base de datos especificada, y si dicha cantidad es mayor a 0, el sistema continuara con el siguiente paso, de lo contrario, el sistema notificara al usuario con un mensaje sencillo y entendible, véase Figura 4.3.

Figura 4.3 Diagrama de flujo de validación de modelos de datos



El método a utilizar no requiere de ningún parámetro, y el resultado que se regresa es un valor de tipo entero, el cual representa la cantidad de tablas existentes en el modelo de datos especificado, para realizar la validación entre ambos modelos se requiere instanciar el objeto por cada una de las conexiones establecidas y posteriormente hacer la llamada.

Determinación de sub-modelo de datos Utilizando la tabla base especificada por el usuario se realiza una consulta sobre el catálogo de sistemas de bases de datos para determinar el sub-modelo al que corresponde la tabla especificada, es decir, determinar todos los objetos que tienen una relación directa/indirecta con la tabla especificada para asegurar la integridad de la información véase Cuadro 4.2.

Cuadro 4.2 Determinación de sub-modelo de datos

```
SELECT      t.TABLE_NAME AS PRIMARY_TABLE ,
            tc.CONSTRAINT_NAME ,
            tc.CONSTRAINT_TYPE ,
            rc.CONSTRAINT_NAME ,
            tc2.TABLE_NAME ,
            (SELECT    COUNT(1)
              FROM      INFORMATION_SCHEMA.COLUMNS
              WHERE     TABLE_NAME = tc2.TABLE_NAME
            ) AS COLUMNS_COUNT
    FROM    INFORMATION_SCHEMA.TABLES t
            INNER JOIN INFORMATION_SCHEMA.TABLE_CONSTRAINTS tc
                                ON tc.TABLE_NAME = t.TABLE_NAME
                                AND tc.CONSTRAINT_TYPE = 'PRIMARY KEY'
            INNER JOIN INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS rc
                                ON tc.CONSTRAINT_NAME = rc.UNIQUE_CONSTRAINT_NAME
            INNER JOIN INFORMATION_SCHEMA.TABLE_CONSTRAINTS tc2
                                ON rc.CONSTRAINT_NAME = tc2.CONSTRAINT_NAME
    WHERE   t.TABLE_NAME = @NombreTabla
    UNION
    SELECT  t.TABLE_NAME ,
            tc.CONSTRAINT_NAME ,
            tc.CONSTRAINT_TYPE ,
            rc.UNIQUE_CONSTRAINT_NAME ,
            tcs.TABLE_NAME ,
            ( SELECT    COUNT(1)
              FROM      INFORMATION_SCHEMA.COLUMNS
              WHERE     TABLE_NAME = tcs.TABLE_NAME
            ) AS COLUMNS_COUNT
    FROM    INFORMATION_SCHEMA.TABLES t
            LEFT JOIN INFORMATION_SCHEMA.TABLE_CONSTRAINTS tc
                                ON tc.TABLE_NAME = t.TABLE_NAME
            LEFT JOIN INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS rc
                                ON rc.CONSTRAINT_NAME = tc.CONSTRAINT_NAME
            LEFT JOIN INFORMATION_SCHEMA.TABLE_CONSTRAINTS tcs
                                ON tcs.CONSTRAINT_NAME = rc.UNIQUE_CONSTRAINT_NAME
    WHERE   t.TABLE_NAME = @NombreTabla
```

La lógica trasladada al lenguaje de programación consiste en realizar la consulta especificando la tabla que se está utilizando como base y generar el sub-modelo de datos al cual corresponde, el resultado de este sub-modelo se almacena en memoria ya que posteriormente será evaluado con un mayor detalle, en caso de que el resultado sea nulo, se le notificara al usuario que hubo algún problema ya que al no obtener información puede que la tabla base no exista en el modelo de datos o que exista algún problema de acceso a la información véase Figura 4.4.

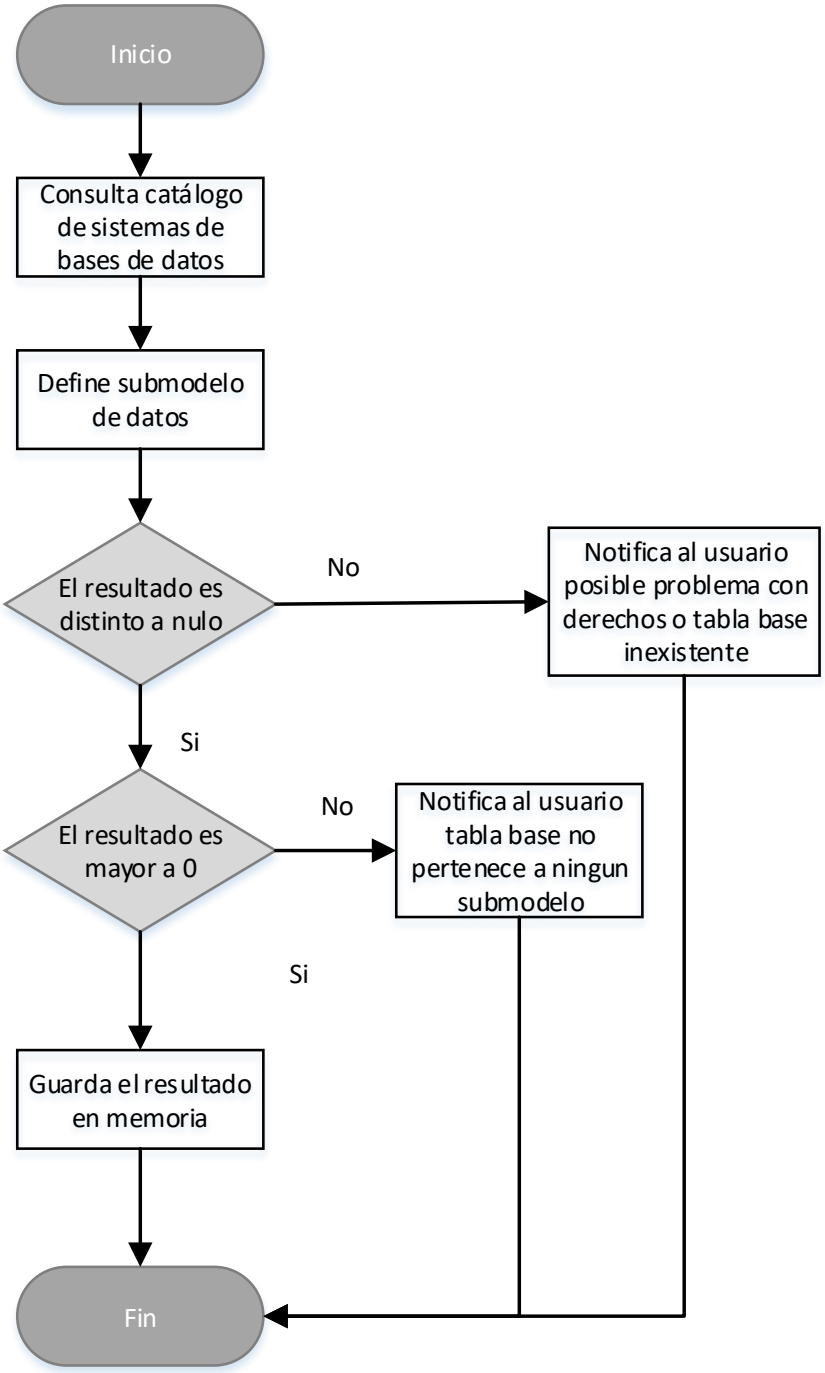


Figura 4.4 Diagrama de flujo de determinación de sub-modelo de datos

El método a utilizar recibe como parámetro el nombre de la tabla sobre la cual se obtendrá el sub-modelo al que esta corresponde y el resultado que se obtiene es un *typed data set* llamado DsSubmodeloTablas con las propiedades generales de cada una de las tablas correspondientes al sub-modelo véase Figura 4.5 Dataset resultadoFigura 4.5.

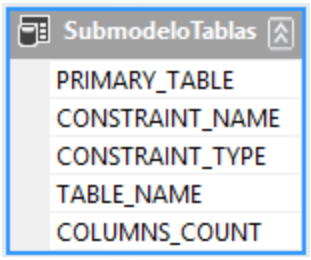


Figura 4.5 Dataset resultado

La información almacenada en este *dataset* es la que posteriormente nos servirá para evaluar las propiedades de cada una de las tablas dentro del sub-modelo y poder determinar si ambos sub-modelos son equivalentes además que desde este momento ya se podrían identificar que objetos forman parte del catálogo de sistemas y cuales representan una tabla encabezado o detalle.

Determinación de las propiedades de los objetos del sub-modelo En base al resultado previamente obtenido, se realiza de nueva cuenta una consulta al catálogo de sistemas de bases de datos para determinar las propiedades de cada uno de los objetos (tablas) pertenecientes al sub-modelo obtenido a partir de la tabla base especificada véase Cuadro 4.3.

```
SELECT  t.TABLE_NAME ,
        c.COLUMN_NAME ,
        c.ORDINAL_POSITION ,
        c.IS_NULLABLE ,
        c.DATA_TYPE ,
        c.COLLATION_NAME
FROM    INFORMATION_SCHEMA.TABLES t
        INNER JOIN INFORMATION_SCHEMA.COLUMNS c ON c.TABLE_NAME = t.TABLE_NAME
WHERE   t.TABLE_NAME = @NombreTabla
```

Cuadro 4.3 Determinación de las propiedades de los objetos del sub-modelo

La lógica trasladada al lenguaje de programación consiste en realizar la consulta *N* cantidad de veces según el resultado de tablas obtenidas por el sub-modelo, mediante un ciclo se comenzará a evaluar cada una de las tablas y cada una de sus propiedades, si alguna de las propiedades no es equitativa en ambos modelos, el sistema notifica al usuario que se encontró una inconsistencia en los modelos de datos por lo cual no puede continuar véase **Error! Reference source not found.**.

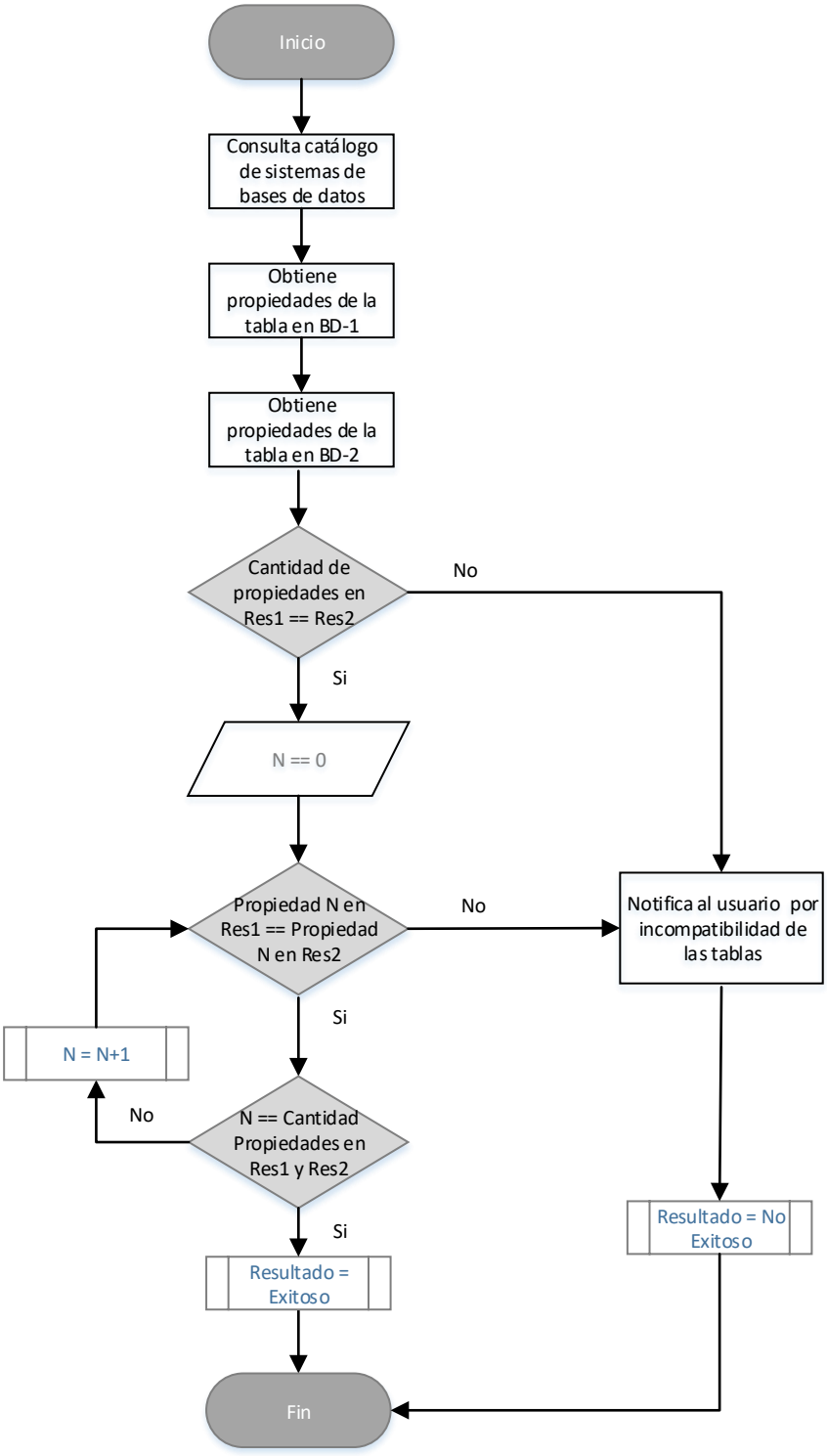


Figura 4 6 Diagrama de flujo de la determinación de las propiedades de los objetos de un sub-modelo

Determinación de catálogos de sistema En base a los resultados del sub-modelo de datos, se realiza una consulta sobre el catálogo de sistemas de bases de datos (véase Cuadro 4.4) para determinar que objetos corresponden a un catálogo de sistema, y que objetos corresponden a una tabla encabezado o un detalle, para que eventualmente la información se replique en ese mismo orden sin comprometer la integridad de la misma.

```
SELECT  c.TABLE_NAME ,
        c.COLUMN_NAME ,
        c.DATA_TYPE ,
        pk.CONSTRAINT_TYPE
FROM    INFORMATION_SCHEMA.COLUMNS c
        LEFT JOIN ( SELECT  ku.TABLE_CATALOG ,
                            ku.TABLE_SCHEMA ,
                            ku.TABLE_NAME ,
                            ku.COLUMN_NAME ,
                            tc.CONSTRAINT_TYPE
                    FROM    INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS tc
                            INNER JOIN INFORMATION_SCHEMA.KEY_COLUMN_USAGE AS ku ON
tc.CONSTRAINT_TYPE = 'PRIMARY KEY'
                                                                    AND tc.CONSTRAINT_NAME
= ku.CONSTRAINT_NAME
                  ) pk ON c.TABLE_CATALOG = pk.TABLE_CATALOG
                          AND c.TABLE_SCHEMA = pk.TABLE_SCHEMA
                          AND c.TABLE_NAME = pk.TABLE_NAME
                          AND c.COLUMN_NAME = pk.COLUMN_NAME
WHERE   c.TABLE_NAME = @NombreTabla
        AND pk.COLUMN_NAME IS NOT NULL
ORDER BY c.TABLE_SCHEMA ,
        c.TABLE_NAME ,
        c.ORDINAL_POSITION
```

Cuadro 4.4 Determinación de catálogos de sistema

Primeramente, se obtiene el nombre el nombre de cada una de las llaves que forman parte de una tabla en específico, y con este resultado se procede a evaluar el tipo de dato que se utiliza para dicha llave, para poder determinar de manera más precisa según el estándar utilizado por la empresa si se trata de un catálogo de sistema o de una tabla de la base de datos véase Cuadro 4.5

```
SELECT cc.TABLE_NAME ,
       cc.COLUMN_NAME ,
       C.DATA_TYPE ,
       C2.REFERENCED_TABLE ,
       C2.REFERENCED_COLUMN
FROM    INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE cc
INNER JOIN INFORMATION_SCHEMA.TABLE_CONSTRAINTS tc ON tc.CONSTRAINT_NAME =
cc.CONSTRAINT_NAME
INNER JOIN INFORMATION_SCHEMA.COLUMNS C ON C.TABLE_NAME = cc.TABLE_NAME AND
C.COLUMN_NAME = cc.COLUMN_NAME
LEFT JOIN ( SELECT  fk.name ,
    OBJECT_NAME(fk.parent_object_id) 'PARENT_TABLE' ,
    c1.name 'PARENT_COLUMN' ,
    OBJECT_NAME(fk.referenced_object_id) 'REFERENCED_TABLE' ,
    C2.name 'REFERENCED_COLUMN'
FROM    sys.foreign_keys fk
    INNER JOIN sys.foreign_key_columns fkc ON fkc.constraint_object_id =
fk.object_id
    INNER JOIN sys.columns c1 ON fkc.parent_column_id = c1.column_id
                                 AND fkc.parent_object_id = c1.object_id
    INNER JOIN sys.columns C2 ON fkc.referenced_column_id = C2.column_id
                                 AND fkc.referenced_object_id = C2.object_id
) AS C2 ON cc.COLUMN_NAME = C2.[PARENT_COLUMN]
        AND cc.TABLE_NAME = C2.PARENT_TABLE
WHERE   cc.CONSTRAINT_NAME = @LlaveReferencia
```

Cuadro 4.5 Obtención de llaves en la tabla

La lógica trasladada al lenguaje de programación se basa en el estándar seguido por la empresa para la definición de sus modelos de datos, es decir, que esta consulta no es completamente versátil, depende mucho del modelo y el estándar seguido. La empresa utiliza valores enteros para definir la llave primaria de una tabla de tipo catálogo de sistema véase Figura 4.7.

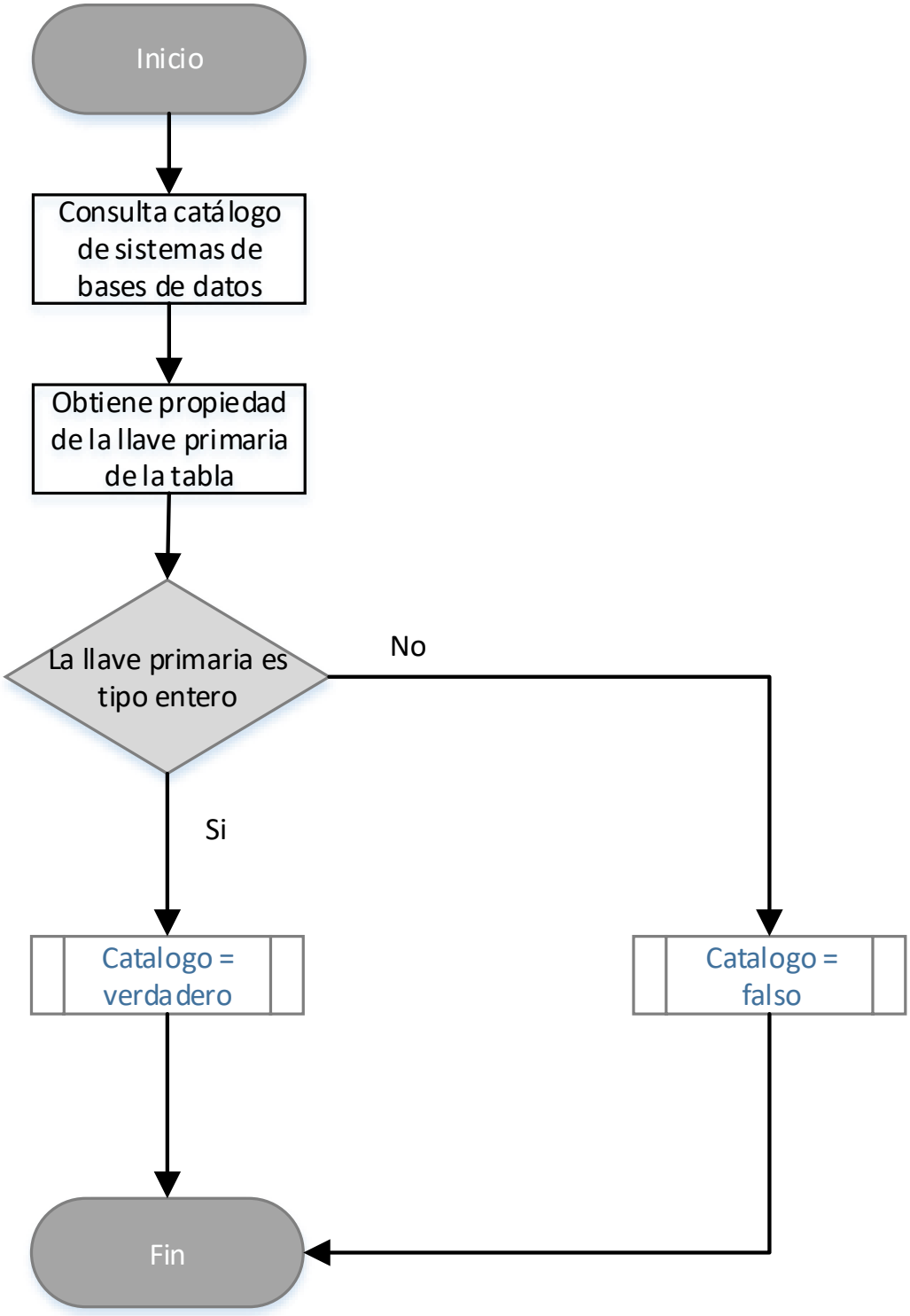


Figura 4.7 Diagrama de flujo de determinación de catálogos de sistema

4.3 Descripción del proyecto

Se crearon una serie de clases las cuales actúan como intermediarios entre la aplicación y los servidores de SQL, ya que en estas clases es en donde se encuentra toda la lógica de programación para realizar la correcta validación previa a la replicación accediendo de manera directa al catálogo de sistemas de bases de datos correspondiente.

4.3.1 Solución de la aplicación

La solución de la aplicación está hecha en Visual Studio 2012, ya que esta era la herramienta disponible en el momento en que se comenzó a trabajar, pero fácilmente puede ser migrada a una versión más reciente de Visual Studio véase Figura 4.8Figura 4.9.

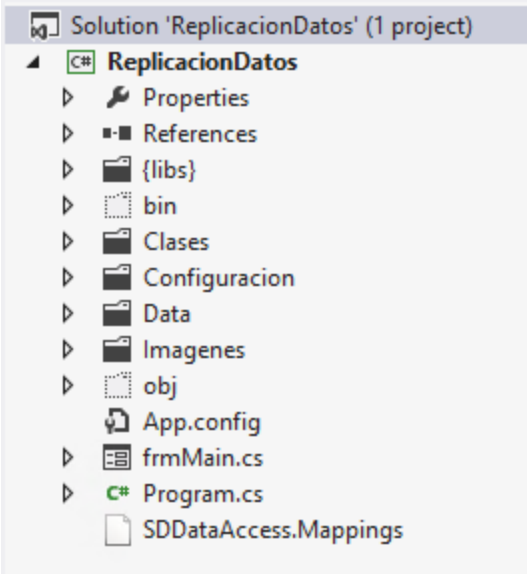


Figura 4.8 Solución de Visual Studio de la aplicación

Dentro de la solución podemos encontrar distintas carpetas las cuales contienen lo necesario para realizar la replicación de datos, desde librerías, hasta clases.

Properties Dentro de esta carpeta podemos encontrar todo lo relacionado con la configuración de la aplicación, desde el detalle de las librerías, versión de la aplicación, hasta la configuración de la misma definida por el usuario.

Libs Dentro de esta carpeta se incluyeron todas las librerías necesarias para el correcto funcionamiento de la aplicación, ya que son librerías propias que no se encuentran en ningún gestor de librerías como *Nuget*.

Clases Aquí es donde se encuentran contenidas las clases necesarias para la replicación de la información, la validación y verificación de los modelos de datos, así como clases adicionales para el aseguramiento de la información, además de clases contenedoras que ayudan a almacenar la información en memoria para su evaluación posterior.

Configuración En esta carpeta se encuentra contenida la forma mediante la cual el usuario interactuara con el archivo de configuración del sistema, así como la lógica de la misma.

Data Aloja objetos de tipo *Typed Data Set*, los cuales ayudan en las tareas de verificación del modelo, ya que una vez que se realizar la ejecución sobre el catálogo de sistemas de bases de datos, el resultado se almacena en estos *datasets* que posteriormente se utilizan para validar información adicional del modelo de datos, como sus propiedades.

Imágenes Carpeta de recursos para mejorar la apariencia de la aplicación.

Raíz En la raíz del proyecto están contenidos los archivos principales como la forma inicial, en la cual se hacen las llamadas a las clases correspondientes necesarias de la validación y replicación de la información entre modelos de datos, así como un archivo de tipo *mapping* el cual ayuda a realizar consultas a las bases de datos configuradas.

4.3.2 Configuración de la aplicación

La configuración de los servidores y bases de datos se almacena en las propiedades del proyecto véase Figura 4.9, pero dicha información pasa por una clase que almacena la información de manera cifrada ya que parte de la configuración es información que no debe de estar expuesta como usuario y contraseña de un servidor SQL o la *ip* publica que pudiese tener.

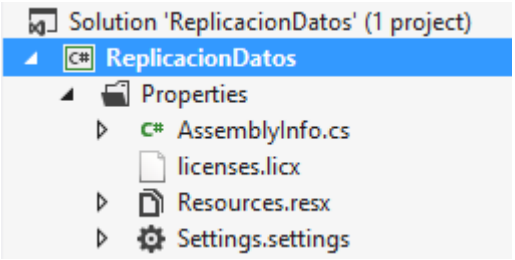


Figura 4.9 Configuración de la aplicación

La información se almacena en *Settings.Settings* el cual es un archivo de configuración del proyecto véase Figura 4.10, y aunque a simple vista podemos visualizar el nombre de la propiedad de configuración, el valor no se almacena en texto legible.

Name	Type	Scope	Value
ServidorLocal	string	User	
ServidorRemoto	string	User	
BDLocal	string	User	
BdRemoto	string	User	
UserLocal	string	User	

Figura 4.10 Archivo de configuración de la aplicación

En el momento en que el usuario realizar alguna modificación ya sea en el nombre de la base de datos, servidor o usuario y contraseña, al actualizar la información la aplicación de manera interna hace una llamada a una clase que se encarga de cifrar la información de manera completamente transparente para el usuario, una vez que se ha cifrado, se procede a guardar la información véase Cuadro 4.6.

```
Properties.Settings.Default.PassRemoto = SeguridadDatos.Encriptar(txtPasRemoto.Text);
Properties.Settings.Default.ServidorLocal = SeguridadDatos.Encriptar(txtServerLocal.Text);
Properties.Settings.Default.ServidorRemoto = SeguridadDatos.Encriptar(txtServerRemoto.Text);
ConnectionSring();
Properties.Settings.Default.Save();
```

Cuadro 4.6 Implementación de seguridad en configuración de aplicación

Al acceder al archivo de configuración *Settings.Settings* podemos asegurarnos que la información se está guardando de una manera que no es legible para el usuario véase Cuadro 4.7.

```
<?xml version="1.0" encoding="utf-8"?>
    <configuration>
        <userSettings>
            <ReplicacionDatos.Properties.Settings>
                <setting name="ServidorLocal" serializeAs="String">
                    <value>LzjWJV9LncN4mVT7IXlC8g==</value>
                </setting>
                <setting name="ServidorRemoto" serializeAs="String">
                    <value>LzjWJV9LncN4mVT7IXlC8g==</value>
                </setting>
                <setting name="BDLocal" serializeAs="String">
                    <value>xdvgW7/eYxWboIwkvtLag1+wu0vgeDOj</value>
                </setting>
```

Cuadro 4.7 Legibilidad del archivo de configuración

Al acceder a la interfaz de configuración el usuario no se dará cuenta de que su información se encuentra protegida mediante un método de cifrado, ya que el sistema de manera transparente remueve el cifrado de la configuración véase Cuadro 4.8.

```
txtBDLocal.Text = SeguridadDatos.Desencriptar(Properties.Settings.Default.BDLocal);
txtBDRemoto.Text = SeguridadDatos.Desencriptar(Properties.Settings.Default.BdRemoto);
txtPasLocal.Text = SeguridadDatos.Desencriptar(Properties.Settings.Default.PassLocal);
txtPasRemoto.Text = SeguridadDatos.Desencriptar(Properties.Settings.Default.PassRemoto);
txtServerLocal.Text = SeguridadDatos.Desencriptar(Properties.Settings.Default.ServidorLocal);
```

Cuadro 4.8 Remover el cifrado de la configuración

Una vez que el sistema ha removido el cifrado de la información, el usuario la visualiza de manera natural véase *Figura 4.11*.

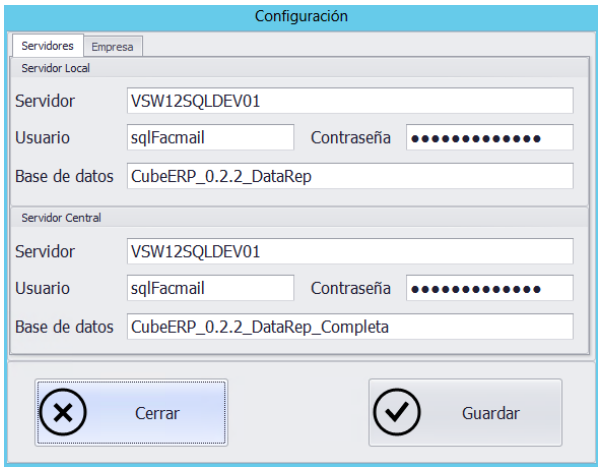


Figura 4.11 Despliegue de la información de configuración

4.3.3 Implementación de lógica de catálogo de sistemas de bases de datos

Verificación de modelos de datos La verificación de modelos de datos se realiza consultando el catálogo de sistemas de bases de datos por cada una de las conexiones definidas por el usuario mediante un método que evalúa si la cantidad de tablas en la base de datos A es correspondiente a la cantidad de tablas de la base de datos B, de esta manera se puede realizar una validación rápida sobre ambos modelos de datos y determinar si será posible o no realizar la replicación de datos entre estos modelos.

El método definido no recibe ningún parámetro, obtiene los parámetros mediante el archivo de configuración de la aplicación, y el valor que regresa es un verdadero en caso de que ambos modelos de datos correspondan correctamente o un falso en caso de que sean modelos de datos distintos véase Cuadro 4.9.

```
private bool CantidadTablas()
{
    int _TablasLocales = 0, _TablasRemotas = 0;
    MensajeHilo("Validando modelo de datos");
    _TablasLocales = _consultaLocal.CantidadTablas();
    MensajeHilo(string.Format("Servidor Local, Tablas {0}", _TablasLocales));
    _TablasRemotas =  _consultaRemota.CantidadTablas();
    MensajeHilo(string.Format("Servidor Remoto, Tablas {0}", _TablasRemotas));
    if (_TablasLocales != _TablasRemotas)
        MensajeHilo("Verifique los modelos de datos, no coinciden");
    return (_TablasLocales == _TablasRemotas);
}
```

Cuadro 4.9 Cantidad de tablas entre modelos de datos

En caso de que los modelos de datos correspondan, se procede a realizar una evaluación rápida entre las tablas de ambos modelos y si cuentan con la misma cantidad de columnas se considera como si el modelo de datos fuera equivalente, aun cuando las propiedades de los objetos no se han evaluado véase Cuadro 4.10.

```
private bool ValidaModeloColumnas()
{
    MensajeHilo("Estableciendo comunicación con servidores");
    List<Estructuras.TablaColumnas> _listaTablasCol = new List<Estructuras.TablaColumnas>();
    List<Estructuras.TablaEvaluada> _listaTablaEvaluada = new List<Estructuras.TablaEvaluada>();
    List<int> _indices = new List<int>();
    _listaTablasCol = TablasColumnas();
    int _tablasEvaluar = (_listaTablasCol.Count / 2);
    _indices = RandomIndex.GenerateRandom(_tablasEvaluar, _listaTablasCol.Count);
    int _indice = 1;
    foreach(Estructuras.TablaColumnas _tabla in _listaTablasCol)//foreach (int _indice in _indices)
    {
        _listaTablaEvaluada.Add(new Estructuras.TablaEvaluada
        {
            _nombreTabla = _listaTablasCol[_indice - 1]._nombreTabla,
            _valida = (_listaTablasCol[_indice - 1]._columnasRemoto == _listaTablasCol[_indice - 1]._columnasLocal)
        });
        _indice++;
    }
    var _resultado = _listaTablaEvaluada.Where(y => y._valida == false).Select(x => x._nombreTabla).ToList();
    if (_resultado.Count > 0)
    {
        MensajeHilo(string.Format("{0} Tablas con diferencias {0}", Environment.NewLine));
        foreach (string _itemEvaluado in _resultado)
            MensajeHilo(string.Format("    {0}", _itemEvaluado, false));
    }
    return (_resultado.Count == 0);
}
```

Cuadro 4.10 Validación de modelos de datos

Determinación de sub-modelo de datos y verificación de propiedades En este caso de estudio el sub-modelo de datos se acoto, y se configuro en la aplicación que únicamente evaluara el sub-modelo de datos para las tablas de venta e inventario físico véase Cuadro 4.11.

```
private bool ValidaSubModelo()
{
    MensajeHilo("Validando esquemas de submodelos..");
    var _subModelos = Properties.Settings.Default.Submodelos.Split(',').ToList();
    List<Estructuras.TablaError> _listaTablasInvalidas = new List<Estructuras.TablaError>();
    foreach (string _submodelo in _subModelos)
    {
        try
        {
            string _subModeloEvaluar = _submodelo.Replace(" ", string.Empty);
            MensajeHilo(string.Format("Validando submodelo {0}", _subModeloEvaluar));
            List<Estructuras.Tabla> _listaTablasSubModeloRemoto = new List<Estructuras.Tabla>();
            List<Estructuras.Tabla> _listaTablasSubModeloLocal = new List<Estructuras.Tabla>();
            Data.DS_Sql.DsSubmodeloTablas _dsSubmodelo = new Data.DS_Sql.DsSubmodeloTablas();
            _dsSubmodelo = _consultaRemota.ObtieneSubmodelo(_subModeloEvaluar);
            DefinicionTablas(_listaTablasSubModeloRemoto, _dsSubmodelo, _consultaRemota, _subModeloEvaluar);
            _dsSubmodelo.Clear();
            _dsSubmodelo = _consultaLocal.ObtieneSubmodelo(_subModeloEvaluar);
            DefinicionTablas(_listaTablasSubModeloLocal, _dsSubmodelo, _consultaLocal, _subModeloEvaluar);
            _dsSubmodelo.Dispose();
            if (_listaTablasSubModeloRemoto.Count == _listaTablasSubModeloLocal.Count)
            {
                foreach (Estructuras.Tabla item in _listaTablasSubModeloLocal)
                {
                    Estructuras.Tabla _itemRemoto = _listaTablasSubModeloRemoto.Find(x => x._nombreTabla == item._nombreTabla);
                    if (_itemRemoto != null)
                    {
                        if (!item._columnasTabla.CompareList(_itemRemoto._columnasTabla, item._nombreTabla, item._SubModelo))
                        {
                            _listaTablasInvalidas.Add(ComparaClases._Elementos[0]);
                            MensajeHilo(string.Format("{0}{1} presenta diferencias en {2} de {3}", Environment.NewLine,
                                ComparaClases._Elementos[0]._nombreTabla,
                                ComparaClases._Elementos[0]._columnasTablaError[0]._nombreColumna,
                                ComparaClases._Elementos[0]._columnasTablaError[0]._propiedad), false);
                        }
                        else
                            _listaTablasSubmodelos.Add(item);
                    }
                }
            }
        }
        catch (Exception ex)
        {}
    }
    return (_listaTablasInvalidas.Count == 0);
```

Cuadro 4.11 Determinación de sub-modelo y sus propiedades

Una vez que se obtiene el sub-modelo sobre el cual se comenzara a trabajar se almacena en una estructura de datos definida para que posteriormente el sistema pueda evaluar por cada uno de los objetos del sub-modelo cada una de sus propiedades, y si existe alguna diferencia entre una tabla o propiedad entre ambos modelos, notifica al usuario y considera que los sub-modelos de datos no son compatibles entre sí.

Determinación de catálogos de sistema Durante la verificación de las propiedades de los objetos de un sub-modelo, se identificaron aquellas tablas son catálogos de sistema según el estándar seguido en la empresa véase Cuadro 4.12, el siguiente paso consiste en evaluar los registros existentes en cada una de las tablas para posteriormente hacer la replicación de datos de un servidor a otro.

```
List<Estructuras.TablaPrimaryKey> _listaTablas = _tablas.AsEnumerable()
.Where(x => x._tablaReferenciada == "" && x._tipoDato.ToString().ToUpper() == "INT")
.Select(y => y).ToList();
comparaCatalogos(_listaTablas);
```

Cuadro 4.12 Determinación de catálogos de sistema

Determinación de información a replicar Cuando se han definido las tablas correspondientes a un catálogo de sistemas, la aplicación comienza a realizar el llenado correspondiente de los objetos obtenidos según el sub-modelo, una vez que ha llenado todos los objetos implicados en el sub-modelo, procede a evaluar cada una de las tablas del servidor A entre su equivalente en el servidor A, al encontrar diferencias se almacenan en una estructura de datos dinámica, que se construye como una copia exacta de la tabla que se está evaluando, para mantener las propiedades originales del objeto, véase Cuadro 4.13.

```
if (dtDiferenciasRemoto.Count > 0 || dtDiferenciasLocal.Count > 0)
 {
    MensajeHilo(string.Format("- Diferencias {0}", (dtDiferenciasLocal.Count + dtDiferenciasRemoto.Count)));
    DataTable _dtLocal = new DataTable();
    DataTable _dtRemoto = new DataTable();
    if (dtDiferenciasLocal.Count > 0)
    {
        _listDifLocal.Add(dtDiferenciasLocal.CopyToDataTable());
        _listDifLocal[_listDifLocal.Count - 1].TableName = _item._nombreTabla;
        _dtLocal.Merge(dtDiferenciasLocal.CopyToDataTable());
        _dtLocal.TableName = _item._nombreTabla;
        _EsInsertUpdate = true;
    }
    if (dtDiferenciasRemoto.Count > 0)
    {
        _listDifRemoto.Add(dtDiferenciasRemoto.CopyToDataTable());
        _listDifRemoto[_listDifRemoto.Count - 1].TableName = _item._nombreTabla;
        _dtRemoto.Merge(dtDiferenciasRemoto.CopyToDataTable());
        _dtRemoto.TableName = _item._nombreTabla;
        _EsInsertUpdate = true;
    }
    if (_EsInsertUpdate)
        PreparaQueryEjecucion(_item._nombreTabla, dtDiferenciasLocal, dtDiferenciasRemoto, _item._nombreColumna);
    if (dtDiferenciasLocal.Count > 0 || dtDiferenciasRemoto.Count > 0)
    {
        Clases.usTablaDatos _usControl = new usTablaDatos(_item._nombreTabla, _dtLocal, _dtRemoto);
        _usControl.Dock = DockStyle.Top;
        DoOnUIThread(delegate() { scrlGrids.Controls.Add(_usControl); });
    }
    _dtLocal.Dispose();
    _dtRemoto.Dispose();
 }
```

Cuadro 4.13 Comparación de la información de una tabla entre ambos modelos de datos

Replicación de la información Una vez que se han encontrado todas las diferencias en las tablas de un sub-modelo de datos en específico, el sistema comienza a realizar la evaluación para determinar si corresponde a una inserción de información o a una actualización sobre un registro ya existente véase Cuadro 4.14.

```
private void PreparaQueryEjecucion(string NombreTabla, List<DataRow> dtDiferenciasLocal,
            List<DataRow> dtDiferenciasRemoto, string _Llave = "")
 {
     try
     {
         if (dtDiferenciasLocal.Count > 0 && dtDiferenciasRemoto.Count == 0)
         {
             foreach (DataRow _Elemento in dtDiferenciasLocal)
             {
                 string _ConsultaInsert = (string.Format("Insert into {0} Values ('{1}')",
                                 NombreTabla, string.Join("','", _Elemento.ItemArray)));
                 _ConsultasRemoto.Add(_ConsultaInsert);
             }
         }

         if (dtDiferenciasRemoto.Count > 0)
         {   DataSet _ds = new DataSet();
             _ds = _consultaLocal.ColumnasUpdate(NombreTabla, _Llave);
             foreach (DataRow _Elemento in dtDiferenciasRemoto)
             {
                 var _elementoInicial = _Elemento.ItemArray.ToList();
                 _elementoInicial.Remove(_Elemento.ItemArray[0]);
                 var _elementos = _ds.Tables[0].AsEnumerable().Select(x => x.ItemArray[0]).ToList();
                 string _ConsultaUpdate = string.Format("UPDATE {0} SET ", NombreTabla) +
                                                         string.Join(", ", _elementos);
                 _ConsultaUpdate = string.Format(_ConsultaUpdate, _elementoInicial.ToArray())+
                                                 string.Format(" WHERE {0} = {1}",
                     _Llave,_Elemento.ItemArray[0].ToString());
                 _ConsultasLocal.Add(_ConsultaUpdate);
             }
         }
     }
```

Cuadro 4.14 Preparación de la información a replicar

Una vez que el sistema ha determinado que registros son para realizar una inserción y que registros son para realizar una actualización, se procede a realizar la ejecución de cada uno de los *queries* de manera individual para evitar cargar al servidor con múltiples peticiones a la vez véase Cuadro 4.15.

```
public bool EjecutaActualizaciones(string _Query)
{
    string _Consulta = _Query;
    SqlCommand dbCommand = new SqlCommand(_Consulta, con);
    dbCommand.CommandTimeout = 5;
    DataSet ds = null;
    try
    {
        ds = db.ExecuteDataSet(dbCommand);
    }
    catch (Exception ex)
    {
        throw ex;
    }
    return true;
}
```

Cuadro 4.15 Ejecución de las consultas determinadas

Básicamente lo que hace el método es recibir cada una de las cadenas definidas por el sistema ya sea inserción o una actualización y procede a realizar la ejecución sobre el servidor correspondiente.

5 Pruebas y resultados

5.1 Prototipos y experimentos

Las pruebas fueron realizadas en un ambiente controlado sobre un sistema operativo Microsoft Windows 10, un servidor con sistema operativo Microsoft Server 2012, SQL Server 2014, y el objetivo de las mismas es comprobar la funcionalidad de la aplicación de replicación de datos, asi como ejemplificar de manera que es lo que ocurre en cada uno de los procesos de verificación y validación de los modelos de datos, se utiliza como ejemplo, distintos modelos de datos para el sistema de FACMAIL-Punto de venta, bajo el sub-modelo de datos de Inventario.

Dichas bases de datos se montaron bajo el mismo servidor SQL.

5.1.1 Arquitectura de escenario de pruebas.

El ambiente de trabajo bajo el cual se realizaron las pruebas, se estableció sobre el mismo servidor SQL 2012, en donde fueron montadas las distintas bases de datos para comprobar la validación del modelo de datos, esquemas de datos (propiedades de las tablas) y la replicación de datos del sub-modelo de inventario.

5.1.2 Base de datos CubeERP_0.2.2 (Base de desarrollo)

Esta base de datos simula la base de datos del sistema de software que se encuentra en desarrollo, la cual con el paso del tiempo va cambiando a la par de los requerimientos del software

5.1.3 Base de datos CubeERP_0.2.2_DataRep (Base de replicación)

Esta base de datos simula la base de datos del sistema de software que se encuentra en un ambiente aislado completamente de todos los cambios tanto en el modelo como en la información contenida en la misma, la cual a diferencia de la base de datos de desarrollo no sufre ningún cambio en absoluto.

Se diseñaron una serie de escenarios para ejecutar las pruebas de funcionalidad como la validación del modelo de datos, la validación de los esquemas y la sincronización de información entre un modelo a otro.

La distribución de dichas bases de datos para el ambiente de pruebas, fue establecida en un mismo servidor, el cual fue utilizado para definir y realizar una serie de pruebas, las cuales serán descritas con más detalle.

5.2 Pruebas sobre modelos distintos.

Se realizaron pruebas utilizando bases de datos con algunas diferencias entre ellas, por ejemplo la primera prueba es de validación de modelos, conocer si ambas bases de datos son equivalentes es decir, si son del mismo sistema, independientemente si sus propiedades son las mismas o no, y las pruebas subsecuentes corresponden a validación de sub-modelo y propiedades de los objetos.

5.2.1 Prueba 1 Verificación de modelos de datos (Cantidad de tablas)

Se realizaron pruebas utilizando dos bases de datos completamente distintas, una de ellas pertenece al sistema FACMAIL-Punto de venta, mientras que la otra base de datos, pertenece a un sistema distinto.

Durante esta prueba se ejecutaron algunos scripts para validar que nuestros modelos de datos son completamente distintos, y que el sistema valido de manera correcta dichas diferencias, sin ejecutar ninguno de los procesos de replicación de información.

1. Se consulta el catálogo de información de la base de datos CubeERP_0.2.2 (véase **Error! Reference source not found.**). Esta consulta nos indica la cantidad de tablas con las que cuenta nuestro modelo de datos.

```
USE MASTER
SELECT TABLE_CATALOG ,TABLE_SCHEMA ,COUNT(TABLE_NAME) AS TABLES_COUNT
FROM sys.databases d INNER JOIN [CubeERP_0.2.2].INFORMATION_SCHEMA.TABLES i ON d.name = i.TABLE_CATALOG
WHERE TABLE_TYPE = 'Base Table'
GROUP BY TABLE_CATALOG ,
         TABLE_SCHEMA
```

Results | Messages

	TABLE_CATALOG	TABLE_SCHEMA	TABLES_COUNT
1	CubeERP_0.2.2	dbo	392

Figura 5.1 Catalogo de información de CubeERP_0.2.2

2. Posteriormente se ejecuta el mismo script sobre la base de datos del sistema FACMAIL.

3. Se realizó la configuración sobre la aplicación, apuntando a cada una de las bases de datos. (véase Figura 5.1).

Figura 5.1 Configuración de la aplicación prueba 1

4. Se guardó la configuración de las bases de datos en la aplicación.

5. Se procedió a ejecutar el proceso de replicación.

6. El sistema de manera correcta nos notificó de que los modelos de datos eran distintos

(Véase Figura 5.2)

Figura 5.2 Intento de replicación prueba 1

Resultados

Se observa como resultado que la aplicación valido y notifico al usuario de manera correcta cuando se detectó que el modelo de datos era distinto entre ambas bases de datos, local y remota.

Como se puede observar en la **Error! Reference source not found.**,se realiza una consulta desde la base de datos Master hacia el catálogo de información haciendo un cruce por el catálogo de objetos de tipo base de datos de nuestro DBMS para obtener la cantidad de tablas bajo cada uno de los catálogos (base de datos) deseados, en este caso la base de datos Facmail y CubeERP_0.2.2

Dicha consulta es la utilizada internamente por la aplicación para validar que los modelos de datos configurados realmente sean los mismos.

5.2.2 Prueba 2 Verificación de modelos de datos (Verificación del modelo)

Esta prueba simula cuando se tienen dos modelos de datos que cumplen con el primer criterio de validación, es decir, la misma cantidad de tablas en su modelo, y como el sistema valido de manera correcta que ambos modelos de datos, son distintos, por lo cual no se puede realizar una replicación de datos entre ambos, para este caso de pruebas, se seguirá utilizando la configuración previamente definida, conexión a la base de datos CubeERP_0.2.2 y la conexión a la base de datos de FACMAIL.

1. Se inicia la aplicación, la cual ya se encuentra previamente configurada, y se inicia el proceso de replicación de datos (véase Figura 5.3).

Figura 5.3 Intento de replicación de datos prueba 2

2. El sistema notifica al usuario de cada uno de los pasos que se han realizado, verificar la configuración de la aplicación, establecer la comunicación con los servidores, verificar los modelos de datos.

3. El sistema procede a obtener la información para la validación de cada una de las tablas de ambos modelos de datos CubeERP y Facmail, utilizando el catálogo de información de cada uno de los modelos. (véase Figura 5.4)

```
USE [CubeERP_0.2.2]
SELECT  tc2.TABLE_NAME , ( SELECT COUNT(1) FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = tc2.TABLE_NAME) AS COLUMN_COUNT
FROM    INFORMATION_SCHEMA.TABLES tc2 WHERE TABLE_TYPE = 'BASE TABLE'
```

Figura 5.4 Consulta al catálogo de información prueba 2

4. Dicha consulta nos despliega cada una de las tablas del modelo de datos especificado, en este caso CubeERP_0.2.2. (véase Figura 5.5)

Results | Messages

	TABLE_NAME	COLUMN_COUNT
1	TipoTransferencia	2
2	TipoProducto	2
3	Turno	7
4	XML_POLIZA_SAT_TRANSACCION	13
5	TipoVideo	2
6	RegimenFiscal	2

Figura 5.5 Resultado de consulta al catálogo de información

5. Internamente el sistema almacena la información obtenida con esta consulta, y procede a realizar el mismo proceso con la base de datos B, en este caso Facmail.

6. Una vez que se obtienen las cantidades de columnas de las tablas de ambas bases de datos, el sistema comienza con la validación de cada una de las tablas, tomando una tabla de la base de datos A y buscando su equivalente en la base de datos B.

7. El sistema comienza a notificar sobre aquellas tablas en las cuales encontró algún problema, en este caso, no encontró la tabla equivalente en el modelo de datos B, debido a que no existe o porque sus propiedades no son equivalentes véase Figura 5.6.

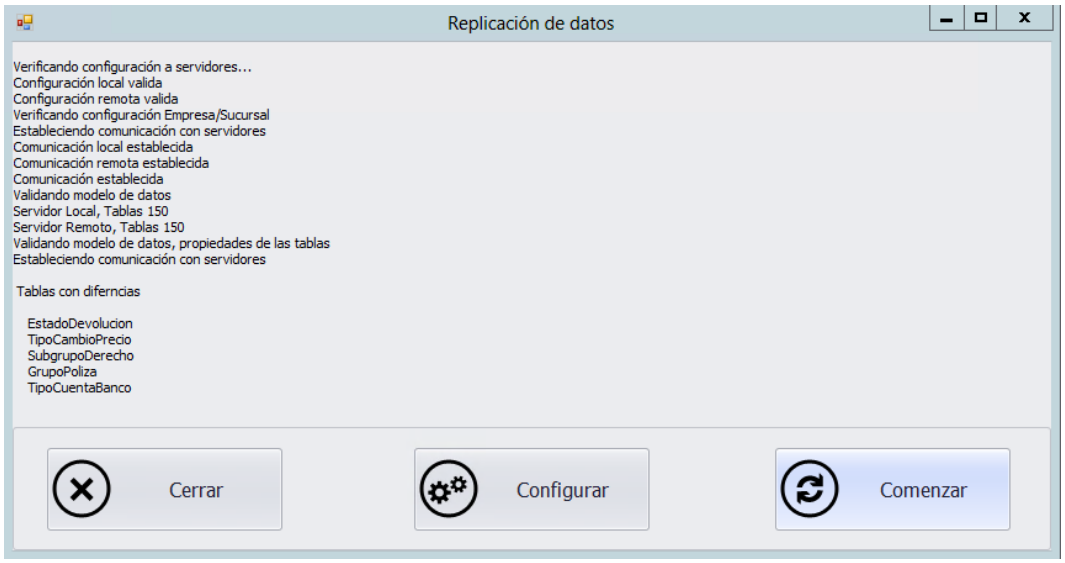


Figura 5.6 Notificaciones de aplicación al usuario

8. Al finalizar con el proceso, el sistema notifica al usuario con todas las tablas en las que se encontraron problemas, así como un mensaje de advertencia. (véase Figura 5.7)

Figura 5.7 Notificación final modelos incompatibles

9. El sistema finaliza con el proceso de validación de manera exitosa, sin realizar ningún paso o intento por replicar información.

Resultados

Se observa como resultado que la aplicación valido y notifico al usuario de manera correcta cuando se detectó que el modelo de datos era distinto entre ambas bases de datos, local y remota.

Como se puede observar en la Figura 5.4, se realiza una consulta desde la base de datos Master hacia el catálogo de información de cada una de las bases de datos configuradas en la aplicación para obtener las tablas y contabilizar la cantidad de columnas que tiene cada una de las tablas en nuestros modelos de datos.

5.2.3 Prueba 3 Verificación de modelos de datos (Verificación de propiedades de las tablas)

Esta prueba muestra el funcionamiento del sistema cuando se cumplieron con las dos validaciones iniciales entre ambos modelos de datos, es decir, misma cantidad de tablas, y mismo modelo de datos. Una vez que se realizaron dichas validaciones, se proceden a validar las propiedades de cada una de las tablas de un sub-modelo dado, en este caso, el sub-modelo del inventario físico.

Se realizará una modificación sobre la misma tabla en el modelo de datos CubeERP_0.2.2_DataRep.

1. Primeramente, se procede a configurar el dispositivo usando los modelos de datos CubeERP_0.2.2 y CubeERP_0.2.2_DataRep. (véase Figura 5.8)

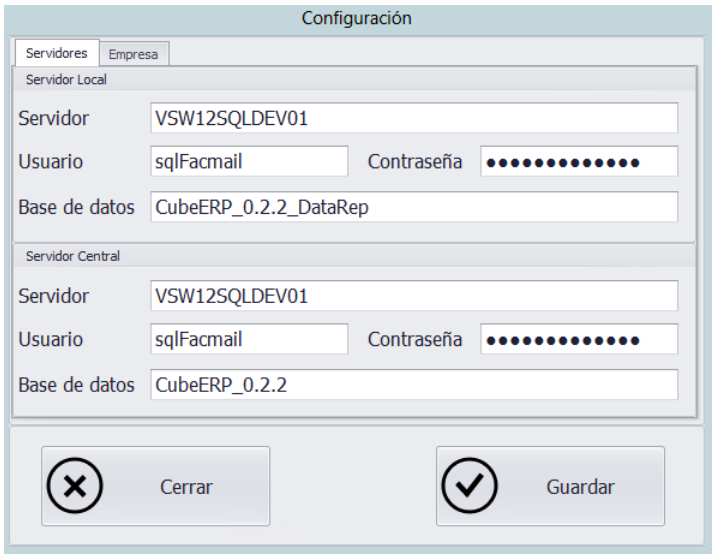


Figura 5.8 Configuración de la aplicación prueba 3

2. Se procede a guardar la información en la aplicación.

3. Se procede a comenzar con el proceso de sincronización.
4. El sistema notifica al usuario los procesos que va realizando. (véase Figura 5.9)

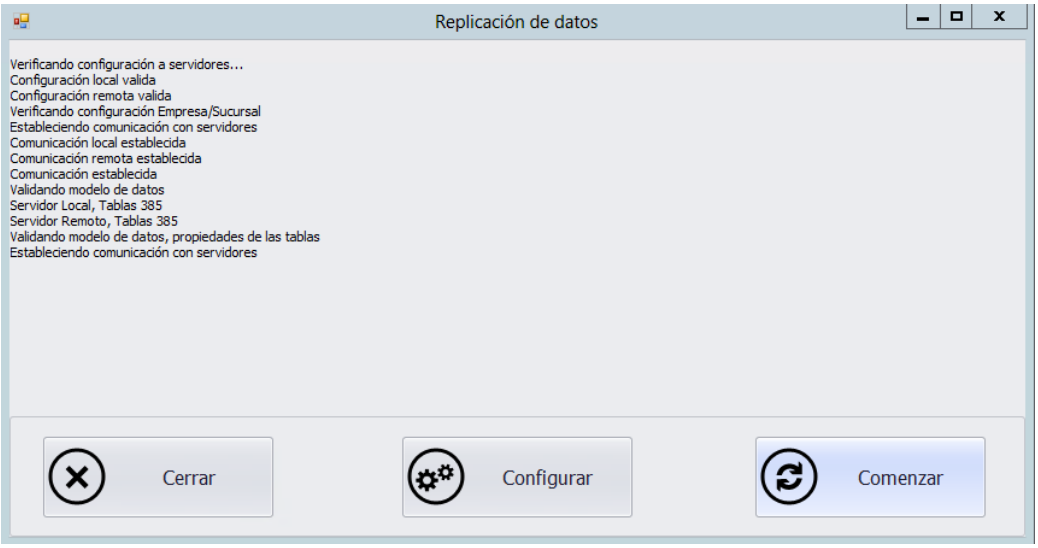


Figura 5.9 Notificaciones al usuario prueba 3

5. El sistema comienza a evaluar las tablas de cada uno de los sub-modelos, contra la información del catálogo de información de la base de datos. (véase Figura 5.10)

```
SELECT   t.TABLE_NAME , c.COLUMN_NAME ,c.ORDINAL_POSITION ,
         c.IS_NULLABLE, c.DATA_TYPE , c.COLLATION_NAME
FROM     INFORMATION_SCHEMA.TABLES t
         INNER JOIN INFORMATION_SCHEMA.COLUMNS c ON c.TABLE_NAME = t.TABLE_NAME
WHERE    t.TABLE_NAME = 'DetalleVenta'
```

Figura 5.10 Evaluación de sub-modelo prueba 3

6. Dicha consulta nos da como resulta la información de cada una de las columnas de la tabla, su nombre, tipo de dato, orden en el que se encuentra, si permite valores nulos o no. (Véase Figura 5.11)

Results | Messages

	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION	IS_NULLABLE	DATA_TYPE	COLLATION_NAME
1	DetalleVenta	DetalleVentaID	1	NO	uniqueidentifier	NULL
2	DetalleVenta	VentaID	2	NO	uniqueidentifier	NULL
3	DetalleVenta	AlmacenID	3	NO	uniqueidentifier	NULL
4	DetalleVenta	EmpresaID	4	NO	uniqueidentifier	NULL
5	DetalleVenta	SucursalID	5	NO	uniqueidentifier	NULL
6	DetalleVenta	UsuarioRegistroID	6	NO	uniqueidentifier	NULL

Figura 5.11 Resultado de la evaluación de sub-modelo prueba 3

7. El sistema comienza a notificar cuando encuentra diferencias en las propiedades de las tablas. (véase Figura 5.12)

Figura 5.12 Notificación final prueba 3

Resultados

Se observa como resultado que la aplicación valido y notifico al usuario de manera correcta cuando se detectó que había diferencias en las propiedades de alguna de las tablas entre ambos modelos de datos.

Como se puede observar en la Figura 5.10, se realiza una consulta al catálogo de información de la base de datos, para obtener la información de cada una de las columnas de cada tabla involucrada en el sub-modelo que se está evaluando.

5.3 Prueba de replicación

Esta prueba muestra el funcionamiento del sistema cuando se cumplieron con todas las validaciones entre ambos modelos de datos, es decir, misma cantidad de tablas, y mismo modelo de datos y propiedades entre objetos equivalentes. Una vez que se ha cumplido con todo esto, se procede a realizar una revisión sobre las tablas pertenecientes a los sub-modelos para encontrar diferencias y poder actualizar la información de manera correcta. En esta prueba se realizará una comparación utilizando la herramienta de *RedGate DataCompare*.

1. Se verifica que la configuración de la aplicación sea la correcta, y este apuntando a modelos de datos correspondientes véase Figura 5.13.

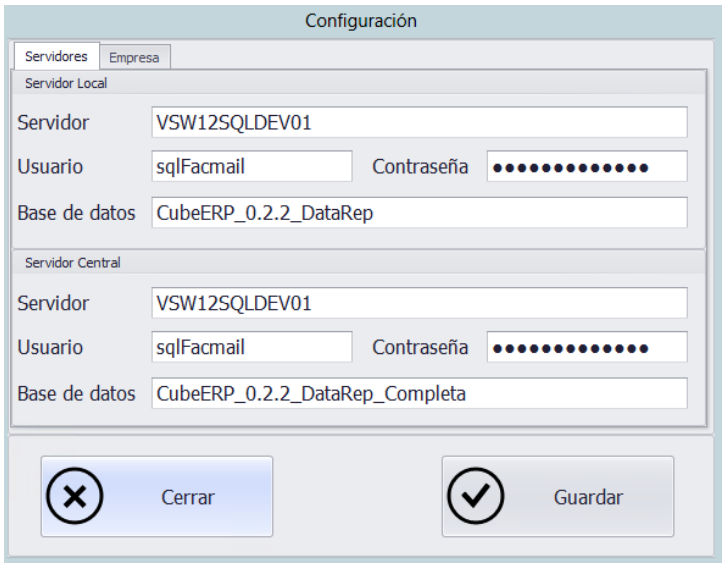


Figura 5.13 Configuración aplicación prueba 4

2. Una vez que se cumplan con las condiciones necesarias entre ambos modelos de datos, se procede con la replicación de información.

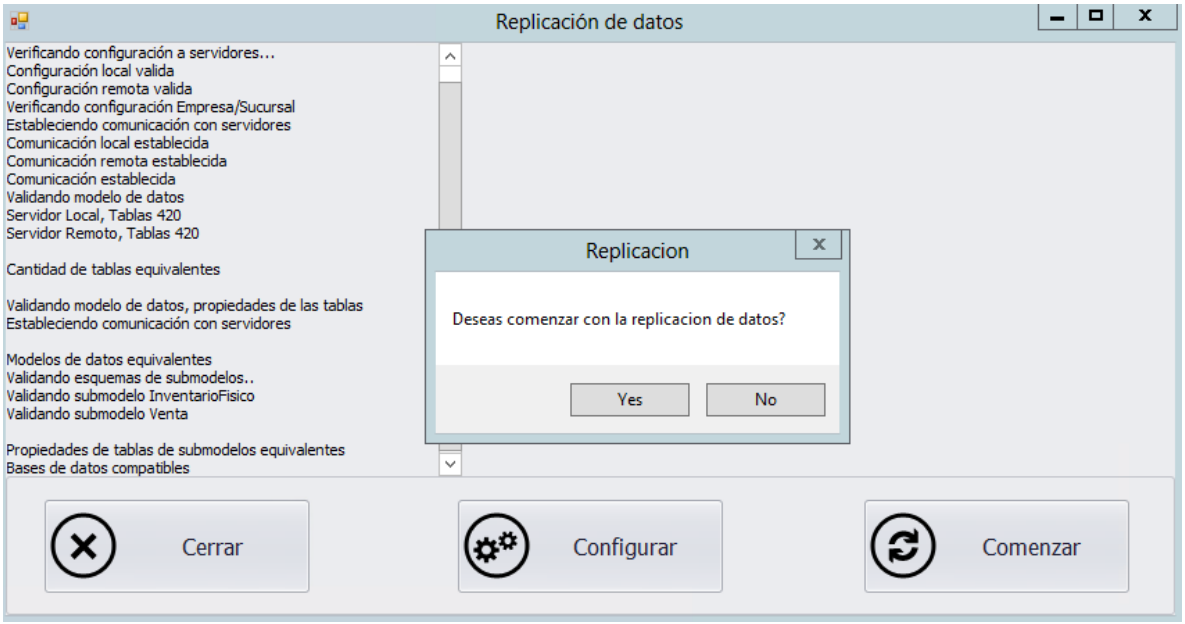


Figura 5.14 Notificación al usuario validaciones exitosas

3. Una vez que se haya encontrado alguna diferencia en una de las tablas, la aplicación procede a desplegar la tabla en la cual encontró información que no se encuentra en ambas bases de datos, la información del lado derecho representa la información en la base de datos del servidor central, y la información del lado derecho representa la tabla en la base de datos local véase Figura 5.15.

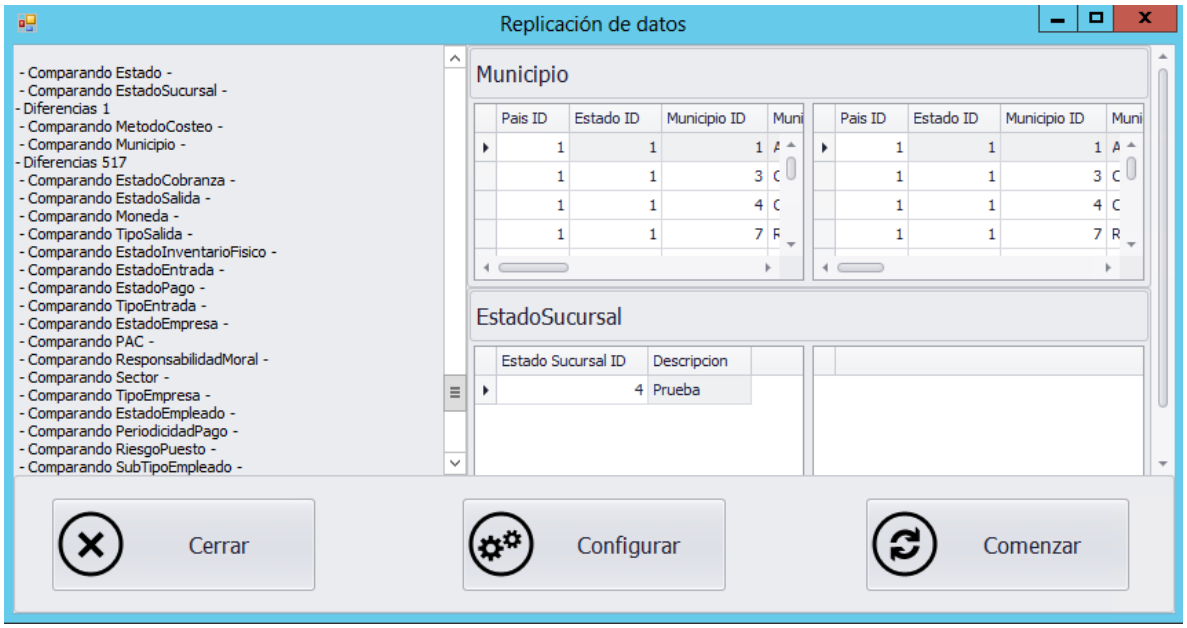


Figura 5.15 Diferencias en tablas

4. Este punto nos sirve para realizar la comparación entre ambas aplicaciones, *red gate* y la aplicación desarrollada. Según la aplicación desarrollada tenemos dos tablas en las que se encontraron diferencias véase Figura 5.16, las cuales son *EstadoSucursal* y *Municipio*

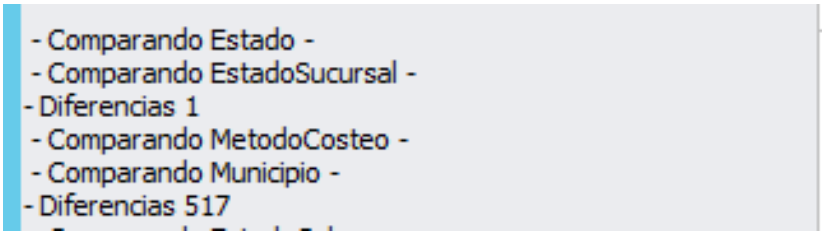


Figura 5.16 Tablas con sus diferencias

5. Procedemos a realizar la misma validación en la aplicación de *RedGate DataCompare*, y evaluaremos el resultado obtenido véase Figura 5.17.

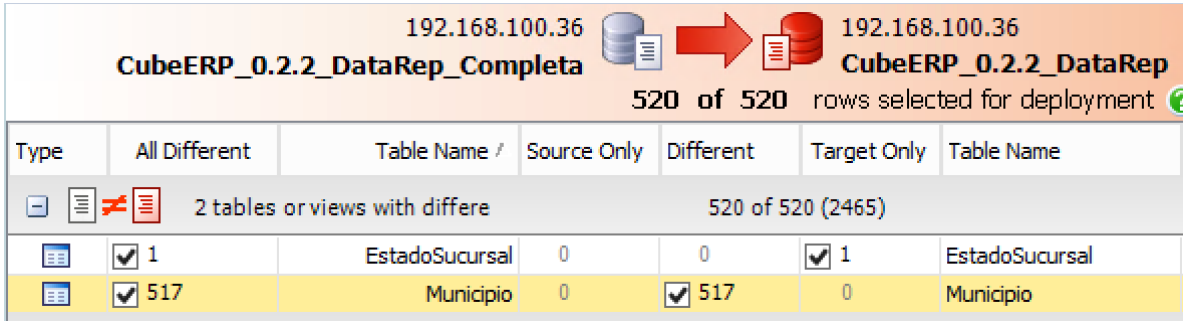


Figura 5.17 Comparación utilizando redgate

6. El resultado obtenido similar, aunque en la aplicación de *RedGate* podemos ver con un poco más de detalle el tipo de registro que es, si representa un *insert* o un *update*. En el caso de la aplicación desarrollada en este momento para obtener dicho detalle es necesario depurar la aplicación y verificar la variable de consultas a realizar véase Figura 5.18.

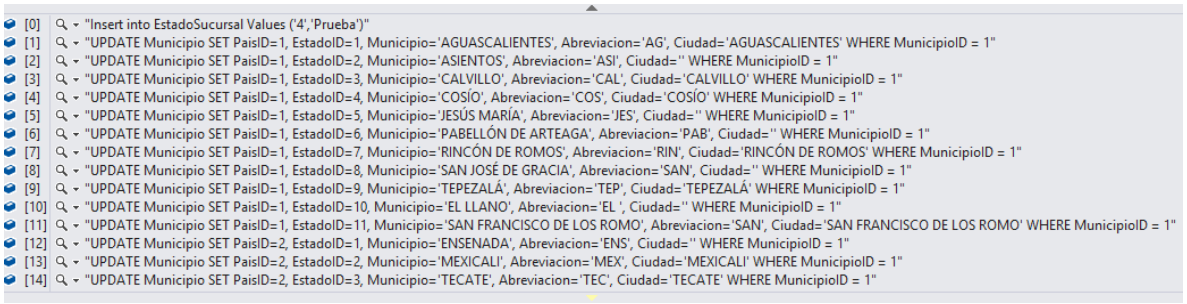


Figura 5.18 Conjunto de operaciones a realizar

7. El sistema procederá a ejecutar dichas consultas de manera automática, en la Figura 5.19 del lado izquierdo podemos observar el catálogo de estado sucursal de ambas bases de datos antes de que se ejecute la consulta de manera automática por el sistema, y en el lado derecho podemos observar como quedaron los catálogos una vez que se realizó la ejecución del comando de manera automática.

Figura 5.19 Comparación de los catálogos en ambos modelos antes y después

	EstadoSucursalID	Descripcion
1	1	Activo
2	2	Inactivo
3	3	Activa

	EstadoSucursalID	Descripcion
1	1	Activo
2	2	Inactivo
3	3	Activa
4	4	Prueba

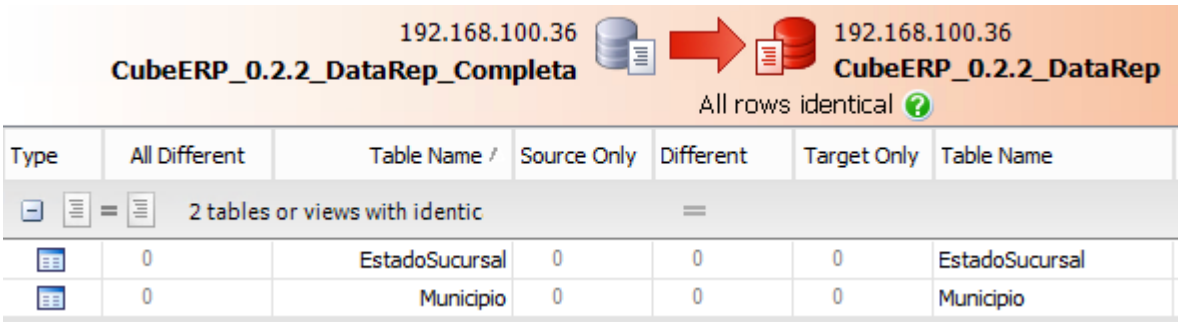
	EstadoSucursalID	Descripcion
1	1	Activo
2	2	Inactivo
3	3	Activa
4	4	Prueba

	EstadoSucursalID	Descripcion
1	1	Activo
2	2	Inactivo
3	3	Activa
4	4	Prueba

8. Una vez que el sistema ha finalizado véase Figura 5.20 procedemos a realizar de nueva cuenta la comparación con RedGate para validar que realmente se hayan actualizado los registros.

Figura 5.20 Notificación final replicación de datos

9. Una vez que se ha hecho la validación con la herramienta de *redgate* véase Figura 5.21, podemos ver que la herramienta cumple con la función para la cual fue diseñada.



Type	All Different	Table Name	Source Only	Different	Target Only	Table Name
2 tables or views with identic						
	0	EstadoSucursal	0	0	0	EstadoSucursal
	0	Municipio	0	0	0	Municipio

Figura 5.21 Resultado final utilizando Redgate

Resultados

Se observa como resultado que la aplicación valido y notifico al usuario de manera correcta cuando se detectó que había diferencias en la información de las tablas.

Posterior a la validación el sistema realizo la replicación de datos de manera precisa y efectiva, actualizando la información e ingresando la información que no se encontraba en alguna de las dos bases de datos.

Análisis de resultados

Cuadro 5.1 Resultados de las pruebas realizadas

Nombre de la prueba	Tiempo desarrollo	Resultado
Verificación de modelo de datos (cantidad de tablas)	1 sprint	Exitoso
Verificación de modelo de datos (compatibilidad de modelos)	1 sprint	Exitoso
Verificación de modelo de datos (validación de propiedades)	2 sprint	Exitoso
Replicación de información	4 sprint	Exitoso

En el Cuadro 5.1 se enlistan los resultados de las pruebas realizadas a la aplicación de replicación de datos utilizando un ambiente controlado de la empresa antes mencionada.

Si se realiza una comparación contra la herramienta de *redgate*, la herramienta desarrollada propone un ambiente más amigable para el usuario y tiene un enfoque práctico, es decir una vez que se ha realizado la configuración por primera vez un usuario común que desconozca de términos

de informática y de modelos de datos puede hacer uso de ella, lo contrario pasa con la herramienta de *redgate*, la cual tiene un enfoque más técnico.

6 Conclusiones y trabajo futuro

A continuación, se presentan algunas conclusiones a las que se llegó con la experiencia utilizando la aplicación y el desarrollo de la misma durante cada una de sus iteraciones de la metodología utilizada. También se hacen algunas recomendaciones para su correcto funcionamiento y se exponen algunas oportunidades de mejora para un trabajo futuro.

6.1 Conclusiones

Al termino del desarrollo de la aplicación descrita en el presente documento tenemos como resultado 4 métodos que hacen uso de los catálogos de sistema de bases de datos y combinados con métodos de validación y ejecución desarrollada en C# tenemos una herramienta que permite a los distintos usuarios del sistema de Facmail-Punto de venta mantener su información siempre disponible es decir, que no dependan de una comunicación constante al servidor en la nube y que al finalizar su operación pueda mantener su información siempre actualizada de la forma más sencilla para cualquier usuario aunque no conozca de bases de datos.

Del trabajo implementado podemos considerar 1 beneficio para la empresa:

1. Disponibilidad de la información para los usuarios: En este momento no es necesario que los usuarios finales del sistema Facmail-Punto de venta se encuentren comunicándose de manera constante con el servidor central, y que pueden operar su sistema de manera normal, y estar seguros que cuando se inicie la aplicación de replicación su información será enviada al servidor central y cualquier modificación que se requiera como parte de su información se descargara desde el servidor central hacia su servidor.

De la misma manera podemos considerar 5 desventajas para la empresa:

1. Verificación de errores en el sistema: Si el cliente desea que su sistema opere sin necesidad de estarse comunicando de manera constante al servidor central, hace que el trabajo de realizar una revisión o validación o una tarea de soporte se vuelva más compleja por el simple hecho de que el técnico no tendrá acceso a la información de una manera rápida, se

tienen que utilizar herramientas de un tercero para poder acceder a dicha información o inclusive realizar una visita de campo para que el técnico pueda realizar su trabajo.

2. Reglas en el sistema: Hace que la labor de programación se vuelva más compleja ya que hay que considerar todos los posibles escenarios y procesos que un usuario pudiese realizar, y cuáles de estos escenarios puede realizar, aunque no se encuentre en comunicación con el servidor central, por ejemplo, el apartado de un producto que no se encuentra en su sucursal, al no tener una comunicación con el servidor central, la información de las demás sucursales no es confiable en absoluto.
3. Volumen de soporte: Al no contar con la información en el momento, el soporte se puede acumular lo que puede generar que los clientes comiencen a desesperarse o en ocasiones a disgustarse por el tiempo de respuesta que se pueda tener.
4. Implementación del sistema ERP-Punto de venta: Al ser un sistema completamente en la nube, al intentar no depender de esto hace que la implementación sea más compleja ya que se requiere que el cliente tenga en su infraestructura un servidor capaz de alojar los servicios de consulta, la aplicación web y además la base de datos y que cuente con los recursos suficientes para soportar el tráfico que se pueda generar.
5. Versionamiento de bases de datos: Es necesario notificar a cada uno de los clientes cuando se ha realizado una modificación sobre el modelo de la base de datos, lo cual además de generar carga adicional de trabajo puede generar una molestia por parte del cliente.

6.2 Recomendaciones

Hacer uso de esta tecnología cuando no hay otra opción, es decir cuando se esté implementando el sistema ERP-Punto de venta en una zona en donde el internet sea demasiado caro ya que únicamente se encuentra internet satelital o cuando es una zona muy propensa a pérdidas de comunicación por distintas causas como accidentes frecuentes, huracanes o zona de sismos, donde realmente se requiera que el sistema siempre esté disponible para su operación.

Además que la instalación la realice una persona con conocimientos técnicos sobre el funcionamiento de servidores, redes y bases de datos, para que la configuración inicial sea la correcta, así como el uso de la aplicación la primera vez, ya que puede que la versión de la base de datos alojada en el servidor local no sea la misma que la alojada en el servidor central, lo cual generaría un trabajo adicional para que ambos modelos de datos se encuentren homologados y la aplicación pueda funcionar de manera correcta.

6.3 Trabajo futuro

Existen algunas posibilidades de mejora detectadas a lo largo del desarrollo de este proyecto.

Versionamiento de bases de datos Incluir procesos para que cuando se encuentren diferencias en el modelo de datos, se genere un script con los cambios necesarios para homologar los dos modelos de datos y se ejecuten de manera automática sin necesidad de que el usuario tenga que intervenir.

Replicación de datos automatizada Cuando se detecte que hay comunicación con el servidor central, realizar la replicación de datos de manera automática, sin que el usuario tenga que intervenir de alguna manera.

Instalador actualizable Generar un instalador versionado que se actualice cada que se realice una nueva publicación, que notifique al usuario cuando existe una versión más reciente y permita actualizar de manera automática, guardando la configuración previamente definida.

Referencias

[1] I. Al Ridhawi, N. Mostafa, and W. Masri, “Location-Aware Data Replication in Cloud Computing Systems,” pp. 20–27, 2015.

[2] S. Kamali, P. Ghodsnia, and K. Daudjee, “Dynamic data allocation with replication in distributed systems,” *30th IEEE Int. Perform. Comput. Commun. Conf.*, pp. 1–8, 2011.

[3] D. I. G. Amalarethinam and C. Balakrishnan, “An optimized strategy for replication in Peer-to-Peer Distributed databases,” *2012 IEEE Int. Conf. Comput. Intell. Comput. Res.*, pp. 1–4, 2012.

[4] O. Id, C. D. Manning, and P. Raghavan, “Online edition (c) 2009 Cambridge UP,” *Inf. Retr. Boston.*, no. c, pp. 1–18, 2009.

[5] R. Stephens, *Beginning Database Design Solutions*. 2009.

[6] M. Bsoul, A. E. Abdallah, K. Almakhadmeh, and N. Tahat, “A Round-based Data Replication Strategy,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 9219, no. c, p. 1, 2015.

[7] M. Younas and B. Eaglestone, “A Formal Verification Strategy for Crash Recovery in Web-Database Applications,” 2002.

[8] O. Wolfson, S. Jajodia, and Y. Huang, “An adaptive data replication algorithm,” *ACM Transactions on Database Systems*, vol. 22, no. 2. pp. 255–314, 1997.

[9] W. Lin and B. Veeravalli, “An adaptive object allocation and replication algorithm in distributed databases,” pp. 132–137, 2003.

[10] Y. F. Huang and J. H. Chen, “Fragment allocation in distributed database design,” *J. Inf. Sci. Eng.*, vol. 17, no. 3, pp. 491–506, 2001.

[11] D. C.J, *Introduccion a los sistemas de bases de datos*. 2001.

[12] N. Sharma, L. Perniu, R. F. Chong, A. Iyer, C. Nandan, A. Mitea, M. Nonvinkere, and M. Danubianu, “Database Fundamentals DB2,” *IBM Corp. 2010*, p. 282, 2010.

[13] H. Darwen, *An Introduction to Relational Database Theory*. 2014.

[14] A. G. Taylor, *SQL for Dummies - .pdf*. 2013.

[15] Microsoft, “Data Definition Language (DDL) Statements (Transact-SQL),” 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/ff848799.aspx.

[16] D. Bello Pena, *Modelizacion de datos*, Primera Ed. Lulu, 2009.

[17] R. Rankins and P. Bertucci, *SQL Server 2014 Unleashed*. Pearson Education, 2015.

[18] Microsoft, “Information Schema Views (Transact-SQL),” 2016. [Online]. Available: https://msdn.microsoft.com/es-mx/library/ms186778.aspx.

[19] B. S. Meine, *Fundamentals of SQL Server 2012 Replication Fundamentals of SQL Server 2012 Replication*. 2012.

[20] IBM, “Data Replication Real-time data for a real-time world.” [Online]. Available: https://www-01.ibm.com/software/data/replication/.

[21] P. Letelier and C. Penades, “Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP),” *Técnica Adm.*, vol. 5, no. 2, 2006.

[22] J. H. Canos, P. Letelier, and C. Penades, “Metodologias agiles de desarrollo de software SCRUM,” p. 8.

[23] K. Schwaber and J. Sutherland, “The Scrum Guide,” *Scrum.Org and ScrumInc*, no. July, p. 17, 2013.

[24] Microsoft, “Sync Framework,” 2008. [Online]. Available: https://msdn.microsoft.com/es-es/library/bb902854(v=sql.110).aspx.

[25] “Red gate, SQL Data Compare.” [Online]. Available: http://www.red-gate.com/products/sql-development/sql-data-compare/.

[26] “SIMEGO.” [Online]. Available: https://www.simego.com/products/sql-admin-studio.

[27] “VISUAL STUDIO.” [Online]. Available: https://msdn.microsoft.com/en-us/library/dd193250(v=vs.100).aspx.

[28] P. Juan and R. Claudia, *Gesti{ó}n de proyectos Scrum Manager*. 2011.