PONTIFICIA **UNIVERSIDAD CATÓLICA** DEL PERÚ

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE UN COMPARADOR Y SINCRONIZADOR DE BASES DE DATOS RELACIONALES DE DISTINTOS MANEJADORES

Tesis para optar por el Título de Ingeniero Informático

Giancarlo Roberto Calderón Garay

ASESOR: Ing. Claudia María del Pilar Zapata Del Río

Lima, Abril del 2009

Resumen

El presente proyecto consiste en el análisis, diseño e implementación de un sincronizador de bases de datos relacionales de distintos manejadores, cuya finalidad es realizar la comparación de objetos entre dos bases de datos y sincronizar dichos objetos de acuerdo a las diferencias encontradas.

Para la gestión del proyecto se ha seguido las prácticas recomendadas por PMI y se han considerado aquellos procesos de gestión necesarios para el desarrollo del proyecto.

Para el desarrollo de la herramienta se optado por utilizar la metodología AUP (cuyas siglas en inglés significan Agile Unified Process), cuyas fases y disciplinas se han adaptado mejor al desarrollo de la aplicación.

La arquitectura seleccionada ha permitido que la aplicación pueda trabajar con distintos manejadores de bases de datos relacionales. Para llevar a cabo este objetivo se ha implementado un componente que se encarga de realizar la abstracción de los manejadores e interactúa con las demás capas de manera transparente.

La implementación de esta componente se ha realizado a través de archivos XML que, con una estructura definida, permiten que la aplicación consulte la metadata de la base de datos y construya las sentencias SQL para la sincronización de objetos. De esta manera, si se desea incorporar otra base de datos sólo se necesita definir el contenido de la plantilla XML y los parámetros de conexión que van definidos en un archivo de configuración.

El proceso de sincronización implementado se puede realizar de dos maneras: mediante una comparación previa de los objetos a sincronizar o mediante un asistente de sincronización, el cual permite al usuario seleccionar de manera más personalizada qué objetos desea sincronizar sin importar qué diferencias existen con los objetos de la base de datos destino. En ambos escenarios, se genera un archivo de bitácora del proceso de sincronización donde se puede verificar los resultados del proceso.

TEMA DE TESIS PARA OPTAR EL TÍTULO DE INGENIERO INFORMÁTICO

TÍTULO:	**ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE UN COMPARADOR Y SINCRONIZADOR DE BASES DE DATOS RELACIONALES DE DISTINTOS MANEJADORES**
ÁREA:	TECNOLOGÍAS DE INFORMACIÓN
PROPONENTE:	Ing. Claudia María del Pilar Zapata Del Río
ASESOR:	Ing. Claudia María del Pilar Zapata Del Río
ALUMNO:	Giancarlo Calderón Garay
CÓDIGO:	20030224
TEMA N°:	316
FECHA:	San Miguel, 12 de agosto del 2008

DESCRIPCIÓN

En la actualidad, los sistemas de información han permitido a las organizaciones automatizar sus procesos y almacenar la información de sus transacciones diarias en bases de datos relacionales de manera que se asegure la confidencialidad, integridad y disponibilidad de la información hacia los usuarios finales.

Para llevar a cabo la implementación de un sistema de información es muy importante realizar el diseño y creación de objetos relacionados a bases de datos como tablas, procedimientos almacenados e índices. Dado que las aplicaciones cambian regularmente por razones como nuevas necesidades, crecimiento de las organizaciones, entre otros como parte de un proceso evolutivo, la tarea de gestionar y controlar los cambios en la base de datos se convierte en una labor crítica para todo administrador de base de datos.

Para lograr que las bases de datos tengan modelos actualizados acorde con las nuevas versiones de las aplicaciones, usualmente se recurre a métodos manuales de sincronización los cuales pueden derivar a cometer distintos errores y causar inconsistencia en la información y estructura de las bases de datos, originando así malestar en el usuario final por la posible pérdida de información o por continuos errores en las aplicaciones que consulta a la base de datos.

El sincronizador de bases de datos propuesto, será una herramienta diseñada para apoyar la labor de los administradores de bases de datos en las tareas de sincronización. Ofrece servicios como la comparación a nivel de estructura de las bases de datos y generación de scripts de sincronización.

OBJETIVO

El objetivo del presente proyecto es realizar el análisis, diseño e implementación de un comparador y sincronizador de bases de datos relacionales, en un entorno amigable para el usuario, que permita llevar a cabo las tareas de comparación y sincronización de la estructura de bases de datos de distintos manejadores.

OBJETIVOS ESPECÍFICOS

- Elaborar una arquitectura que permita que la aplicación pueda conectarse a distintas bases de datos relacionales y obtener su *metadata*.

- Elaborar el análisis y diseño de la aplicación que soporte la arquitectura planteada y sea una base para la implementación.

- Diseñar mecanismos de traducción de encabezados de disparadores y de encabezados de procedimientos almacenados entre manejadores de bases de datos distintos.

- Implementar las funcionalidades de comparación y sincronización de bases de datos relacionales de acuerdo al análisis, diseño y arquitectura planteados.

ALCANCE

La aplicación permitirá sincronizar la estructura de distintos objetos de una base de datos hacia otra, pudiendo pertenecer ambas al mismo modelo físico. Entre estos objetos se tienen: tablas, columnas, índices, llaves primarias, llaves foráneas, check constraints y vistas. Además se permitirá la sincronización de sólo los encabezados de triggers y procedimientos almacenados.

Además realizará la comparación a nivel de estructura de los objetos antes mencionados indicando en un cuadro resumen las diferencias encontradas y de acuerdo a los resultados de la comparación, el usuario puede sincronizar los objetos seleccionados o generar un script de sincronización que posteriormente puede ser ejecutado.

La aplicación se probará para conexiones a SQL Server y Oracle pero su diseño genérico permitirá agregar conexiones a otros motores de bases de datos. Los procesos de comparación y sincronización pueden realizarse entre bases de datos de distintos fabricantes.

Finalmente se podrán obtener un reporte con los resultados de la comparación, así como una bitácora con el resultado de la sincronización que permita visualizar los errores encontrados en caso de haber ocurrido alguno.

A mis padres Jorge y Gladys y a mi hermano Rodrigo por todo su apoyo incondicional durante mis años de estudio y por depositar en mí su confianza para lograr mis objetivos propuestos.

Para mis abuelos que me protegen y velan por mí desde el cielo y que siempre han sido un motor para llevar a cabo mis metas.

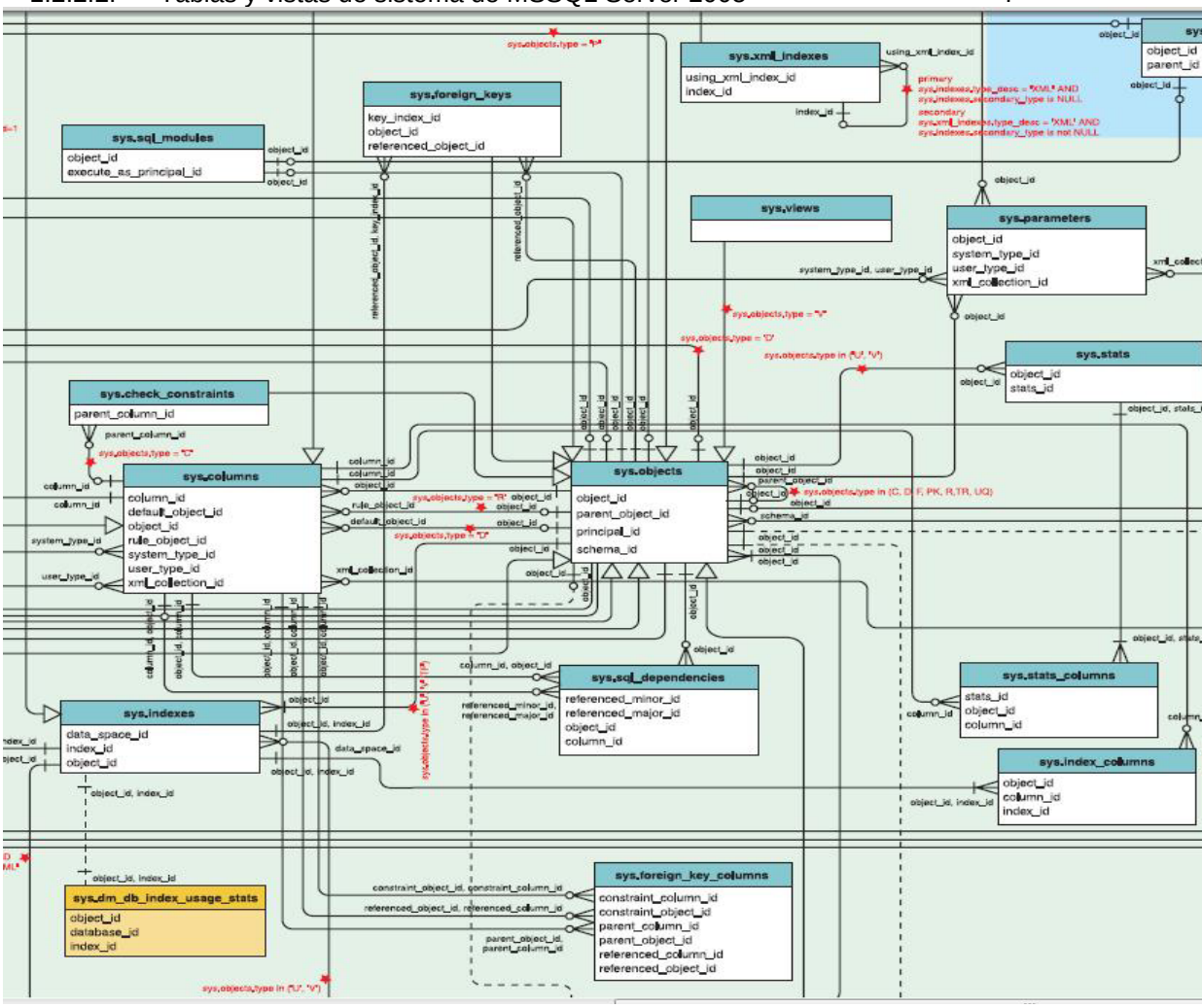
Agradecimientos

- Un agradecimiento especial a la Ing. Claudia Zapata por todos sus consejos y recomendaciones durante el desarrollo de la presente tesis, tanto como profesora y como asesora y que ha permitido que el proyecto se culminé con éxito.

- A todos mis profesores que durante mis 5 años de estudios no sólo me han brindado conocimiento, sino me han inculcado valores y el deseo de siempre aprender y no sucumbir ante las adversidades.

- A todos mis amigos y compañeros de clases con los que he vivido muchas experiencias tanto en lo académico como lo personal y que han aportado a mi crecimiento profesional y como persona.

Índice General



Índice de ilustraciones

Índice de Tablas

Introducción

El presente proyecto tiene como objetivo el análisis, diseño e implementación de un comparador y sincronizador de bases de datos que permita identificar las diferencias y similitudes entre distintos objetos dentro de una base de datos y a su vez, permita sincronizar la estructura de dichos objetos.

En la actualidad los procesos de sincronización suelen realizarse de manera manual, lo que origina errores que pueden perjudicar el desempeño de la base de datos y por ende, el de los sistemas que dependen de dicha base de datos.

Los escenarios más frecuentes en donde se lleva a cabo este proceso son cuando se realizan pases a producción que traen consigo cambios en la estructura de la base de datos y ambientes en donde se utilizar bases de datos distribuidas para el procesamiento y almacenamiento de información en distintas sucursales.

El presente trabajo ha sido dividido en 5 capítulos los cuales se describen a continuación. En el primer capítulo se detallarán los conceptos necesarios para el entendimiento del problema, las alternativas existentes, el plan de proyecto y la descripción de la solución a implementar.

En el segundo capítulo se explicará la metodología de desarrollo a utilizar así como el análisis de la solución que incluye la viabilidad del proyecto, identificación de requerimientos y el análisis derivado de dichos requerimientos.

En el tercer capítulo se detallará el diseño de la arquitectura seleccionada para la solución, la interfaz gráfica del aplicativo y los componentes identificados en la arquitectura.

En el cuarto capítulo se explicarán los criterios de selección de las tecnologías a usar en el desarrollo de la solución y el esquema de pruebas para la verificación y validación de la solución.

Finalmente, en el quinto capítulo de mencionarán cuáles han sido las observaciones, conclusiones y recomendaciones que se derivan del presente proyecto.

1. Generalidades

A continuación se explican los conceptos necesarios para el entender el problema que se desea resolver a través del presente proyecto, las soluciones existentes en la actualidad, luego se mostrará el esquema seguido para la realización del proyecto y finalmente se realizará una descripción de la solución a implementar.

1.1. Definición del problema

En la actualidad, la implementación de un sistema de información ha permitido a las organizaciones automatizar sus procesos y almacenar la información de sus transacciones diarias en bases de datos de manera que se asegure la confidencialidad, integridad y disponibilidad de la información hacia los usuarios finales.

Para llevar a cabo la implementación de un sistema de información es muy importante realizar el diseño y creación de objetos relacionados a bases de datos como tablas, procedimientos almacenados e índices. Dado que las aplicaciones cambian regularmente por razones como nuevas necesidades, crecimiento de las organizaciones, entre otros como parte de un proceso evolutivo, la tarea de

gestionar y controlar los cambios en la base de datos se convierte en una labor crítica para todo administrador de base de datos.

Para ilustrar la situación podemos considerar un proyecto de implementación en donde los desarrolladores cuentan con una base de datos de prueba (comúnmente llamado base de datos de desarrollo), el cual empieza como una copia de la base de datos de producción. A lo largo del proceso de desarrollo es común que se realicen cambios en el entorno de prueba y luego de una etapa de validaciones y pruebas, dichas modificaciones tienen que ser sincronizadas a la base de datos de producción de manera transparente, es decir sin alterar el correcto funcionamiento actual de dicha base de datos.

Para realizar el proceso de migración mencionado en el párrafo anterior, no sería lo más recomendable eliminar la actual base de datos de producción y reemplazarla por la base de datos de desarrollo actualizada debido a que se perdería información crítica. Ante esta situación el administrador de la base de datos usualmente genera de manera manual un archivo que contiene los cambios necesarios para actualizar la base de datos de producción, pero dicho proceso tiene distintos problemas que de manifestarse pueden traer como consecuencia el incorrecto funcionamiento del sistema de información y el malestar de los usuarios que utilizan a diario dicho sistema.

Otro caso típico en donde el proceso de sincronización es crítico se da cuando se desea sincronizar una base de datos distribuida, en donde la información está distribuida físicamente en distintos dispositivos de almacenamiento. En esta situación, se desea que los cambios que se den en la estructura de una imagen de la base de datos, también sea notificada hacia las otras imágenes de manera que todas las bases de datos estén sincronizadas. La situación antes mencionada se presenta en casos de que una organización distribuya su base de datos por cada sucursal y en donde la sincronización sea un proceso importante para mantener la integridad de la información.

Entre los principales inconvenientes que se pueden presentar en una sincronización manual están: la decisión de cuando generar el archivo de cambios, verificar si dicho archivo contiene todos los cambios necesarios debido a que la falta de inclusión de modificaciones que pueden repercutir de manera negativa en el funcionamiento del sistema y finalmente, la complejidad de la estructura de la base

de datos puede convertir el proceso de la migración en una labor tediosa y larga si se realiza de manera manual que, en el peor de los casos, puede tomar semanas y llevar a las consecuencias mencionadas anteriormente.

En resumen, se ha podido identificar que el problema principal es la falta de alguna herramienta automatizada que permita a los administradores de bases de datos comparar los cambios en las estructuras de las bases de datos y realizar el proceso de sincronización de manera rápida y eficiente. Con dicha herramienta, los cambios serán aplicados de una manera segura y permitirán el correcto funcionamiento del sistema permitiendo a los usuarios realizar sus actividades diarias sin ningún inconveniente.

1.2. Marco conceptual del problema

En esta sección se definirán los conceptos necesarios para el entendimiento del problema así como una descripción de los modelos de sincronización utilizados por los administradores de bases de datos.

1.2.1. Definiciones

Se empezará definiendo los conceptos relacionados a la administración de bases de datos para dar un mejor entendimiento del problema. Al final de cada definición se indicará la referencia correspondiente. [MAN2007], [TSA1990] y [WEB002].

1. **Base de datos.** Colección de datos persistentes que pueden compartirse e interrelacionarse (Fuente: [MAN2007]).
2. **Base de datos distribuida.** Una base de datos distribuida es una base de datos que está bajo el control de un sistema de administración de bases de datos (DBMS) cuyos dispositivos de almacenamiento se encuentran distribuidos en diferentes computadoras o equipos en un mismo espacio físico o distribuidos a lo largo de una red de computadoras interconectadas (Fuente: [MAN2007]).
3. **Sincronización de bases de datos.** Proceso por el cual se preparan los cambios apropiados para actualizar la *metadata* y/o data de una base de datos mediante la generación de scripts de actualización (Fuente: [WEB002]).

4. **SQL (Structured Query Language).** Estándar de la industria de los lenguajes de bases de datos que incluye sentencias para la definición de bases de datos, manipulación de bases de datos y control de las bases de datos (Fuente: [WEB002]).
5. **Tabla.** Arreglo bidimensional de datos. Una tabla está formada por la parte del encabezado y la parte del cuerpo (Fuente: [WEB002]).
6. **Índice.** Estructura de archivos secundaria que provee una ruta alternativa hacia los datos. Dicha estructura permite realizar las consultas de una manera más rápida (Fuente: [WEB002]).
7. **Llave Primaria.** Identificador único de una tabla de la base de datos. Puede estar compuesta por una o más columnas de la tabla en cuestión (Fuente: [WEB002]).
8. **Llave Foránea.** Es el campo de una tabla que se empareja con la llave primaria de otra tabla. El propósito de uso de la llave foránea es permitir la relación entre diferentes tablas (Fuente: [WEB002]).
9. **Procedimiento almacenado.** Son consultas pre-compiladas hacia la base de datos que mejoran la seguridad, eficiencia y usabilidad de las aplicaciones cliente/servidor de bases de datos (Fuente: [WEB002]).
10. **Disparador.** Es un procedimiento almacenado que puede ser configurado para automáticamente se ejecute cuando ciertos eventos ocurren en una tabla. Dichos eventos pueden ser la inserción, actualización o eliminación de registros (Fuente: [WEB002]).
11. **Sistema de Administración de Base de Datos (DBMS).** Es el software o aplicación encargado de proveer las herramientas necesarias para almacenar, consultar, agregar, modificar y eliminar información dentro de la base de datos. Entre los ejemplos más conocidos tenemos SQL Server, Oracle y Microsoft Access (Fuente: [MAN2007]).
12. **Sistema de Administración de Base de Datos Distribuido (DDBMS).** Es el conjunto de componentes que apoya las consultas de datos residentes en múltiples ubicaciones. Un DBMS distribuido encuentras datos remotos, optimiza consultas globales y coordina transacciones en múltiples ubicaciones (Fuente: [TSA1990]).
13. **Etapa de producción.** Es la etapa en el ciclo de desarrollo de un sistema en donde se pone puesta en marcha el sistema implementado, una vez realizada las pruebas necesarias, y los usuarios empiezan a utilizar dicho sistema para registrar las operaciones del día a día (Fuente: [TSA1990]).

14. **Base de datos de Producción.** Base de datos utilizada para almacenar la información originada por los procesos del negocio del día (Fuente: [MAN2007]).
15. **Base de datos de Desarrollo.** Base de datos utilizada en entornos de desarrollo por los programadores y desarrolladores durante el proceso de implementación de una solución informática (Fuente: [MAN2007]).

1.2.2. Estructura de las tablas de sistema de los manejadores de bases de datos

A continuación se muestra una breve descripción de la estructura de las tablas de sistema de dos manejadores de bases de datos distintos: MSSQL Server 2005 y Oracle 10g con el fin de resaltar que la aplicación tendrá que adaptarse a las diferentes estructuras de los diccionarios de datos de cada manejador.

1.2.2.1. Tablas y vistas de sistema de Oracle 10g

La figura 1.1 (obtenida de la referencia [WEB016]) muestra un esquema simple del diccionario de datos de una base de datos Oracle 10g. Cabe mencionar que no todas las tablas y/o vistas se muestran. Como se aprecia, existe cierto grado de redundancia en la información de las vistas lo que permite que las consultas al diccionario de datos sean más sencillas de elaborar.

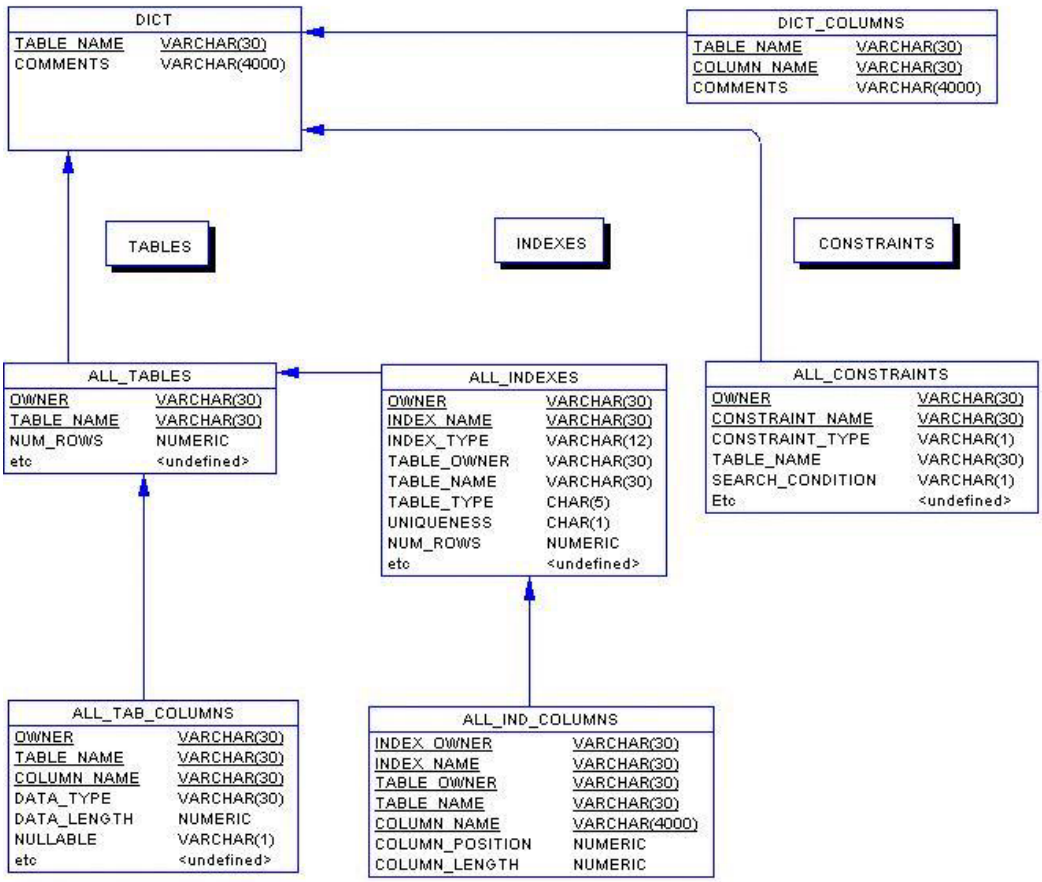


Figura 1.1 Tablas y vistas de sistema de Oracle 10g

1.2.2.2. Tablas y vistas de sistema de MSSQL Server 2005

En la figura 1.2 se muestra el diagrama del diccionario de datos de MSSQL Server 2005 (extraído de la referencia [WEB015]). Se recalca que se ha recortado la imagen relacionada al diccionario de datos del gráfico original, el cual muestra un diagrama completo de todas las tablas y vistas de sistema. A diferencia de Oracle, las relaciones entre las vistas de sistema se manejan mediantes identificadores y no existe la redundancia mostrada en el diccionario de datos de Oracle.

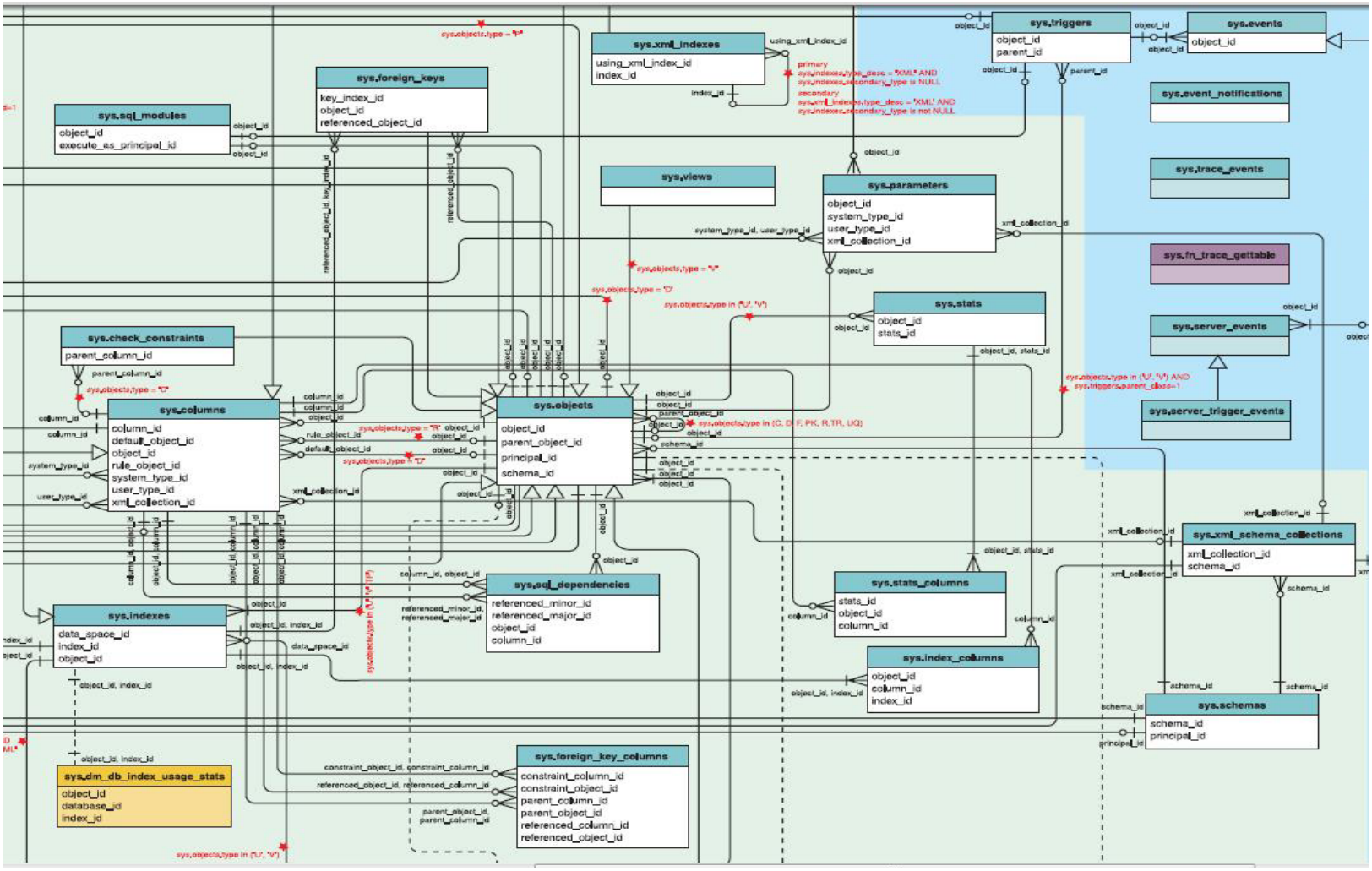


Figura 1.2 Tablas y vistas de sistema de MSSQL Server 2005

1.2.3. Modelos de sincronización de bases de datos

A continuación se presentan los modelos de sincronización existentes, los cuales no siempre implican un proceso automatizado. Estos modelos son utilizados por los administradores de bases de datos para realizar la migración de un entorno de desarrollo a un entorno de producción. Luego de cada explicación se adjunta una imagen que muestra de manera visual los pasos del modelo. La información fue extraída del documento “Improved Database Development” (Ver referencia [PDF003]).

1.2.3.1. Modelo de fusión ad-hoc

Este modelo se basa en aplicar las modificaciones a la base de datos de producción en intervalos de tiempo específicos. De esta manera, cada cierto tiempo todas las instancias de bases de datos de los desarrolladores se sincronizan con la base de datos real. El proceso para la migración es el siguiente:

1. Se restaura la base de datos de producción en la máquina de cada desarrollador, previo al proceso de desarrollo.
2. El desarrollador realiza las modificaciones necesarias en su base de datos local.
3. Se establece el momento de la sincronización de las bases de datos de los desarrolladores. En este punto, cada desarrollador compara su base de datos con la última copia de la base de datos de producción y en caso de que algún objeto haya cambiado o necesite agregar uno nuevo, se produce la sincronización. De esta manera cada cambio se produce de manera incremental.
4. Si es requerido, se graba una imagen de la nueva base de datos en un sistema de control de versiones.
5. La nueva versión de base de datos es comparada con una copia de las base de datos de producción. El resultado de este proceso es un archivo con las sentencias SQL necesarias para la sincronización final.
6. Se valida el archivo generado en el punto anterior aplicándolo sobre una copia de la base de datos de producción. Luego de realizar las pruebas y

modificaciones necesarias, se graba una imagen de esa copia en el sistema de control de versiones.

7. Este proceso se repite cada vez que los desarrolladores requieran liberar una nueva versión del sistema o aplicación.
8. Antes de ejecutar el archivo de sincronización, se genera una copia de respaldo de la base de datos de producción.
9. Finalmente se ejecuta el archivo de sincronización sobre la base de datos de producción.

Un resumen del proceso descrito se muestra en la figura 1.3. Esta figura fue obtenida de la referencia [PDF003].

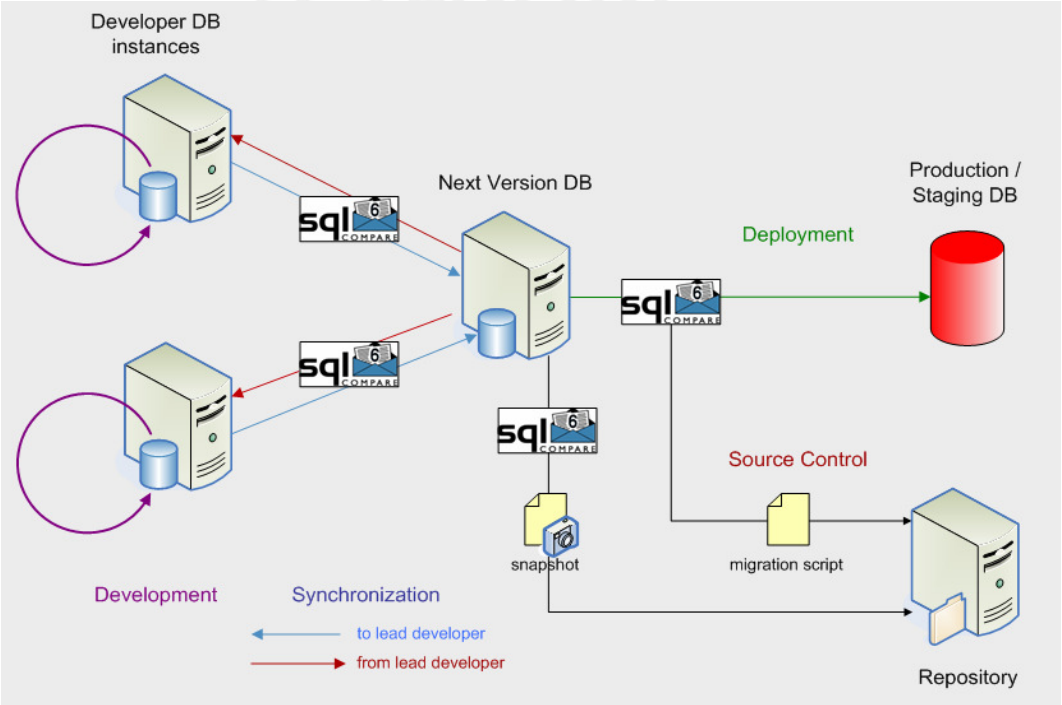


Figura 1.3 Modelo de fusión ad-hoc

1.2.3.2. Modelo de control de versiones a nivel de objetos

Este modelo se basa en guardar un historial de cambios a nivel de objetos dentro de una base de datos. De esta manera, este modelo permite al administrador de base de datos tener un control más detallado sobre los cambios hechos al diseño de base de datos. El proceso que sugiere el modelo es el siguiente:

1. Mediante una herramienta de administración de base de datos, generar los scripts de creación de los objetos de la base de datos de producción. Luego, organizarlos en carpetas diferentes para facilitar el rastreo de modificaciones.
2. Añadir las carpetas creadas en el sistema de control de versiones. De esta manera, los desarrolladores pueden utilizar las características propias de un controlador de versiones como aplicar los cambios hechos, obtener la última versión de los objetos y ver un historial de los cambios producidos.
3. En este punto, los archivos grabados en el repositorio de versiones servirán como base para empezar el proceso de desarrollo.
4. Mediante el sistema de control de versiones, cada desarrollador obtiene la última versión de los archivos de los objetos de la base de datos y los compara con la instancia local, la cual de ser necesaria es sincronizada.
5. Cada desarrollador actualiza los objetos en su base de datos local, y luego de un proceso de verificación y aprobación, sincroniza el cambio en el repositorio.
6. Cada cierto intervalo de tiempo, el desarrollador debe obtener la última versión de los objetos de la base de datos del repositorio y sincronizarlos con los de su base de datos local. De esta manera, se asegura de siempre tener los últimos archivos.
7. Cuando es requerido por el equipo de desarrolladores, los archivos del repositorio son marcados para ser migrados a la base de datos de producción. Entonces, se genera el archivo de actualización mediante la sincronización de los archivos del repositorio contra una copia de la base de datos de producción. Se ejecuta este archivo contra dicha copia para pasar por un proceso de validación.
8. Este proceso se repite cada vez que los desarrolladores requieran liberar una nueva versión del sistema o aplicación.
9. Antes de ejecutar el archivo de sincronización, se genera una copia de respaldo de la base de datos de producción.
10. Finalmente se ejecuta el archivo de sincronización sobre la base de datos de producción.

El resumen del proceso de este modelo se aprecia en la figura 1.4. Esta figura fue tomada del documento referenciado en [PDF003].

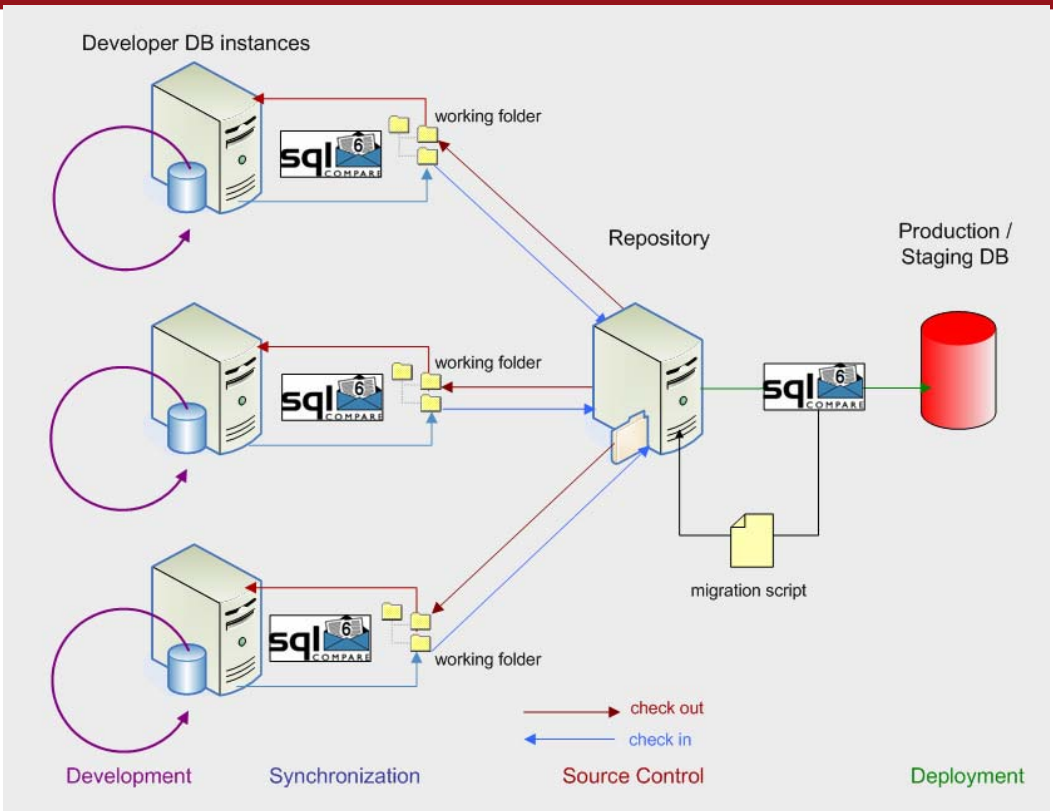


Figura 1.4 Modelo de control de versiones a nivel de objetos

1.2.3.3. Modelo de desarrollo off-line

A diferencia de los 2 modelos anteriores, este modelo se basa en modificar directamente los archivos de creación de la base de datos. La creación de una nueva base de datos de desarrollo se llevaría a cabo cuando se dichos scripts de creación sean ejecutados. De esta manera, se da cierto grado de libertad para desarrollar sin la necesidad de acceder a una base de datos debido a la modificación directa de los scripts de creación. A pesar de eso, siempre es necesario realizar una sincronización periódica con la base de datos real o de producción de manera que se pueda asegurar que los cambios serán aplicados sobre la última versión de la base de datos de producción. El proceso de migración propuesto por este modelo es el siguiente:

1. Antes de que el proceso de desarrollo empiece, se debe obtener los scripts de los objetos de la base de datos del sistema de control de versiones y actualizarlos en la máquina local del desarrollador.
2. Mediante un editor de textos SQL, el desarrollador modificar los scripts en su máquina local.

3. Una vez desarrollados los cambios, se realiza la comparación de los scripts modificados contra una base de datos de prueba y si es necesarios se sincroniza dicha base de datos.
4. Una vez validados los scripts ante bases de datos de prueba, se realiza la actualización de dichos archivos en el sistema de control de versiones.
5. Cuando se tenga listo una versión de los scripts de actualización, se compara contra una copia de la base de datos de producción y se genera el archivo de sincronización en caso de ser necesario. Este archivo es validado contra dicha copia y luego es almacenado en el sistema de control de versiones, previo a su ejecución en la base de datos de producción.
6. Una vez realizada la validación, se obtiene una copia de respaldo de la actual base de datos de producción y se ejecuta el archivo de sincronización ante dicha base de datos.

El proceso descrito se muestra resumido en la figura 1.5. Esta figura fue tomada del documento en la referencia [PDF003].

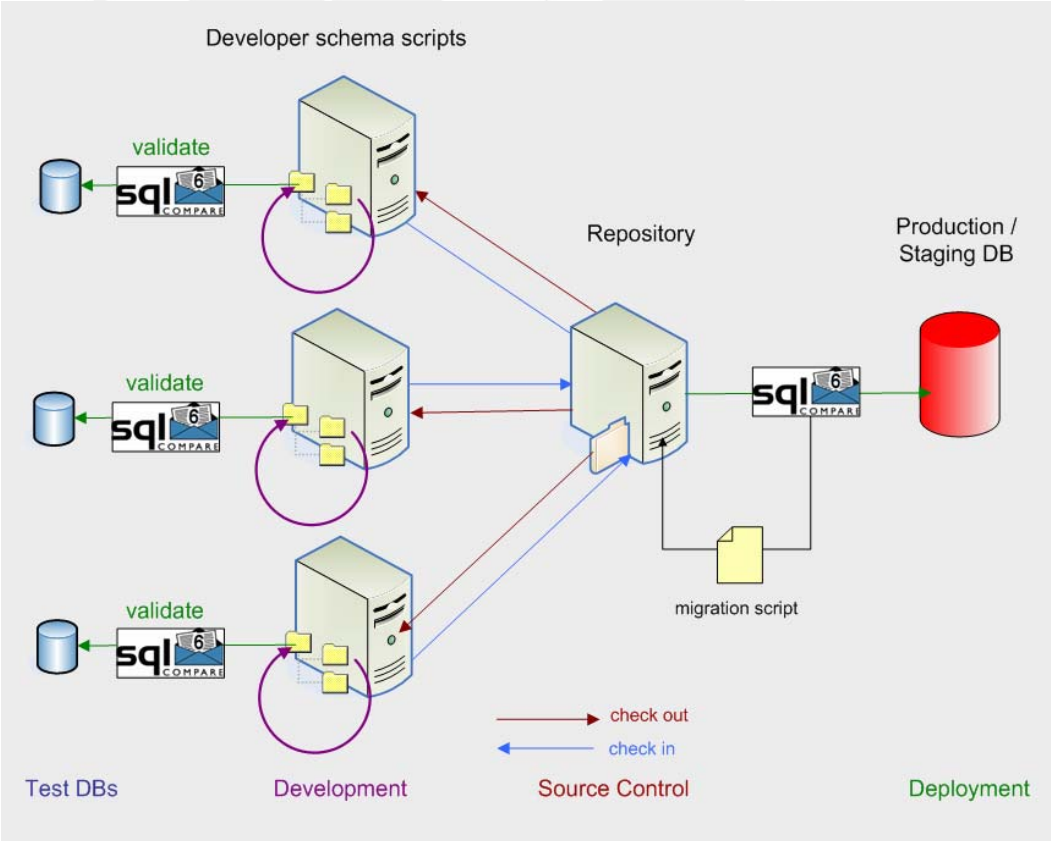


Figura 1.5 Modelo de desarrollo off-line

Los modelos presentados son concebidos de manera general, es decir en teoría son aplicables a distintos manejadores de bases de datos. Solo se ha podido identificar de acuerdo a la referencia bibliográfica [PDF003], que los 3 modelos se pueden aplicar sin ningún inconveniente a una base de datos MSSQL Server.

La aplicación a desarrollar seguirá el modelo de fusión ad-hoc para realizar la sincronización de los objetos de la base de datos debido a que la sincronización se llevará a cabo sin necesidad de cerrar la base de datos y se brindará al usuario la opción de generar los scripts de sincronización, los cuales pueden ser guardados en un sistema de repositorio de archivos.

1.3. Plan del proyecto

Para la gestión del Proyecto de Fin de Carrera, se tomará como base lo descrito en el documento “A Guide to the Project Management Body of Knowledge” del PMI (ver Referencia [PMI2004]). Considerando el alcance del proyecto, se considerarán los procesos de dirección de proyectos necesarios los cuales serán adaptados al proyecto.

1.3.1. Procesos de dirección de proyectos

A continuación se listan los procesos escogidos (clasificados por Grupos de Procesos) y una breve justificación del porqué de su elección.

A. Grupo de Procesos de Iniciación

- *Desarrollo preliminar del alcance del proyecto*
  Este proceso es necesario pues se define preliminarmente cuál va a ser el alcance del proyecto, es decir cuánto va a abarcar la solución que se plantea ante el problema de sincronización. En la figura 1.6. se muestra la secuencia de pasos de este grupo de proceso.

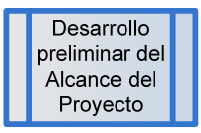


Figura 1.6 Grupo de Proceso de Iniciación

B. Grupo de Procesos de Planificación

- *Desarrollo del Plan de Dirección del Proyecto*

  Es indispensable contar con este documento pues es la principal fuente para determinar cuáles son los procesos de dirección de proyectos involucrados.

- *Definición del Alcance*

  Se desarrolla de manera más detallada el alcance del proyecto teniendo en cuenta cuáles son los procesos de plan de dirección de proyectos que van a ser considerados y los requerimientos obtenidos de la fase de identificación de los requisitos en la implementación de la solución.

- *Crear el WBS (Work Breakdown Structure)*

  Tomando en cuenta las fases de desarrollo de acuerdo a la metodología de desarrollo descrita más adelante, se definen cuáles van a ser las actividades a realizar y los entregables a presentar.

- *Definición de actividades*

  Con el WBS construido, podemos definir e identificar las actividades necesarias para la realización de los entregables identificados en el WBS.

- *Secuencia de las actividades*

  Una vez listadas y clasificadas las actividades a realizar en el WBS, se puede estimar la secuencia de dichas actividades tomando en cuenta las fases de implementación.

- *Estimación de recursos y duración de actividades*

  Este proceso es crítico pues permite estimar cuánto esfuerzo hay que dedicar a cada actividad de acuerdo al grado de complejidad. Para el presente proyecto, son críticas las actividades relacionadas al análisis y diseño.

- *Desarrollo del Calendario de Actividades*

  Se plasma lo anterior referente a la planificación de las actividades en un diagrama de Gantt para poder apreciar de manera gráfica las actividades a realizar con su respectiva duración.

- *Identificación de Riesgos*

  Teniendo en cuenta que parte del Proyecto de Fin de Carrera se va a desarrollar durante época de clases, es posible que existan retrasos en el desarrollo de algunas actividades, por lo cual identificar los riesgos asociados es importante para administrar mejor el tiempo.

- *Análisis cuantitativo y cualitativo de los riesgos*

  Priorizar los riegos encontrados permitirá definir los planes de contingencia más adecuados y contrarrestar los posibles retrasos en el proyecto.

- *Plan de respuesta ante los riesgos*

  Identificados y clasificados los riesgos por prioridad e impacto en el proyecto, es importante desarrollar los planes de contingencia necesarios para evitar retrasos en los entregables que perjudiquen el desarrollo normal del proyecto.

En la figura 1.7. se aprecia la secuencia de actividades del grupo de procesos de Planificación.

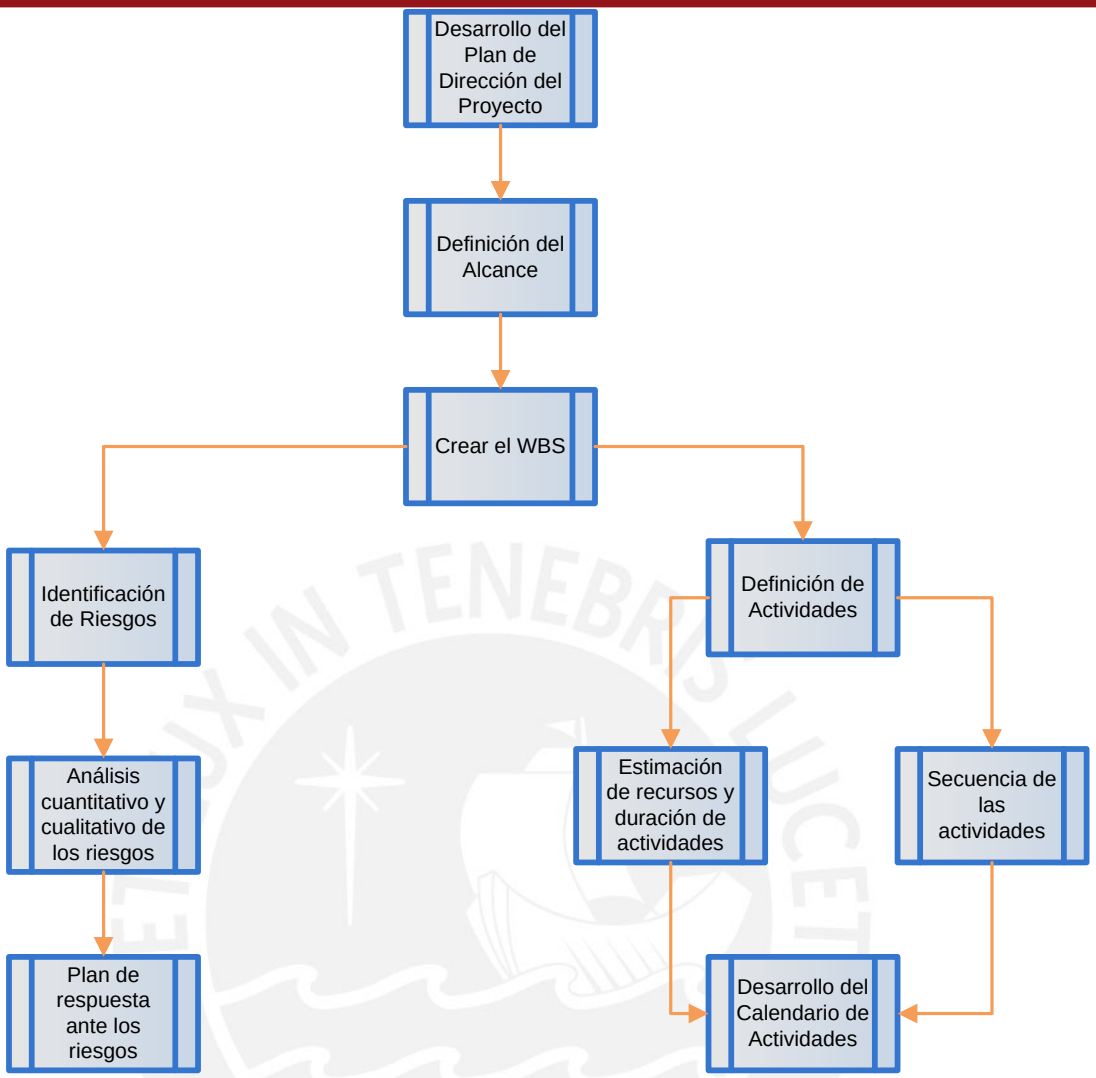


Figura 1.7 Grupo de Procesos de Planificación

C. Grupo de Procesos de Ejecución

- *Dirección y Manejo de la Ejecución del Proyecto*

  Proceso necesario para llevar un control del desarrollo de los procesos listados en el plan de proyecto, teniendo como indicador los entregables que se definan en el WBS. Además se podrá obtener información del avance de cada entregable y ver el estado en el que se encuentran. El proceso se muestra en la figura 1.8.

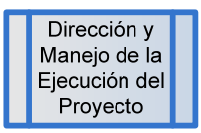


Figura 1.8 Grupo de Procesos de Ejecución

D. Grupo de Procesos de Control y Monitoreo

- *Monitoreo y Control del Trabajo del Proyecto*

  Este proceso incluye que los riesgos sean identificados a tiempo y que se ejecuten los planes de contingencia definidos. Además, este proceso provee información del desempeño de las tareas o actividades realizadas tomando en cuenta el alcance, la calendarización de las actividades (duración), costos y riesgos. Para controlar el avance del proyecto se propone el uso de la Gestión del Valor Ganado que permitirá medir en base al WBS y al diagrama GANTT cuánto del proyecto se tiene avanzado en comparación a una estimación inicial de lo que se debería haber avanzado.

- *Control de Cambios*

  En todo proyecto es importante detectar los factores que pueden ocasionar cambios, tanto beneficiosos como perjudiciales, en el proyecto y manejar el impacto que tengan en el mismo.

- *Control del Alcance*

  Es importante controlar los cambios que pueda sufrir el alcance, ya sea por los riesgos detectados o la estimación de tiempo y costo para las actividades descritas en el WBS.

- *Control del Calendario del Proyecto*

  Además del control del alcance, es importante verificar que las actividades se estén desarrollando en el tiempo establecido y controlar los cambios de tiempo de duración de algunas actividades (debido posiblemente a retrasos).

- *Monitoreo y Control de Riesgos*

  Es crítico que durante el desarrollo del proyecto se identifiquen los riesgos y se ejecuten los planes de contingencia de manera rápida. Además se puede ir midiendo el grado de efectividad de dichos planes ante la aparición de nuevos riesgos, que podrían llevar a la reformulación de tales planes.

La secuencia de actividades se muestra en la figura 1.9.

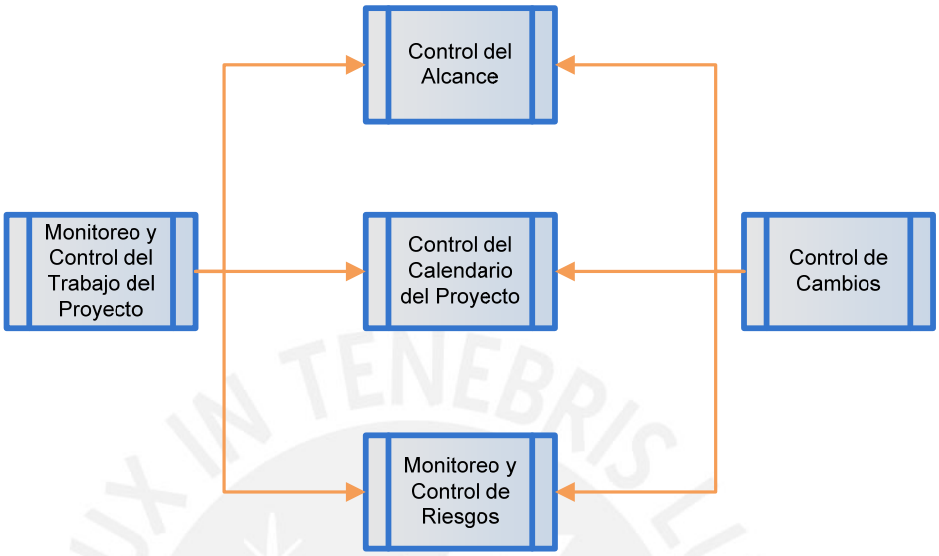


Figura 1.9 Grupo de Procesos de Control y Monitoreo

E. Grupo de Procesos de Cierre

- *Clausura del Proyecto*

  Este el proceso necesario para finalizar el Proyecto de Fin de Carrera, en la cual se da entrega de la tesis a la Facultad de Ciencias de Ingeniería en la fecha establecida. El proceso se muestra en la figura 1.10.

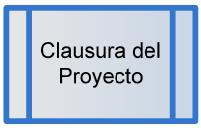


Figura 1.10 Grupo de Procesos de Cierre

1.3.2. WBS del proyecto

Para el presente proyecto se ha confeccionado el siguiente WBS, mostrado en la figura 1.11, correspondiente a las actividades a realizar para el desarrollo del producto. La metodología para implementar la solución es AUP (Agile Unified

Process), el cual consta de las fases de Incepción, Elaboración, Construcción y Transición. Esta es una metodología de desarrollo de software iterativo incremental de manera que la fase relacionada a la implementación se realizará en 3 iteraciones cuyos hitos serán la entrega de una versión preliminar de la herramienta. El detalle y justificación de la metodología se describirá en el capítulo 2.

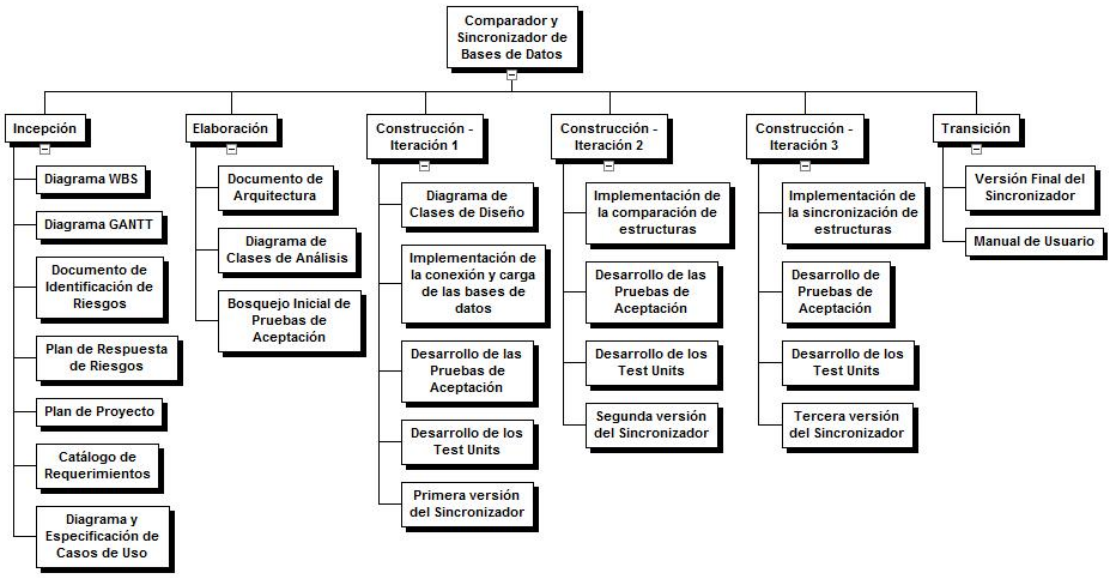


Figura 1.11 WBS del Proyecto

1.3.3. Listado y secuencia de las actividades

A continuación se muestra el diagrama GANTT, mostrando las actividades en la figura 1.12 y el calendario de actividades en la figura 1.13.

	Task Name	Duration	Start	Finish	Predecessors
1	**Comparador y Sincronizador de Bases de Datos**	**760 hrs**	**01/02/2008**	**11/07/2008**	
2	**Incepción**	**78 hrs**	**01/02/2008**	**17/02/2008**	
3	Desarrollo del Diagrama WBS	6 hrs	01/02/2008	01/02/2008	
4	Desarrollo del Diagrama GANTT	6 hrs	02/02/2008	03/02/2008	3
5	Documento de Riesgos del Proyecto	6 hrs	03/02/2008	04/02/2008	
6	Plan de Respuestas de Riesgos	12 hrs	05/02/2008	08/02/2008	5
7	Plan de Proyecto	8 hrs	08/02/2008	10/02/2008	3,5,6
8	Catálogo de Requerimientos	10 hrs	10/02/2008	11/02/2008	
9	Diagrama y Especificación de Casos de Uso	20 hrs	12/02/2008	17/02/2008	8
10	**Elaboración**	**45 hrs**	**17/02/2008**	**26/02/2008**	**7,9**
11	Diagrama de Clases de Análisis	21 hrs	17/02/2008	22/02/2008	9
12	Documento de Arquitectura	12 hrs	22/02/2008	24/02/2008	9
13	Bosquejo Inicial de las Pruebas de Aceptación	12 hrs	24/02/2008	26/02/2008	9,11,12
14	**Construcción - Iteración 1**	**227 hrs**	**27/02/2008**	**15/04/2008**	**11,12,13**
15	Diagrama de Clases de Diseño	22 hrs	27/02/2008	03/03/2008	11,12
16	Implementación de la conexión y carga de bases de datos	80 hrs	03/03/2008	21/03/2008	15
17	Pruebas de Aceptación	25 hrs	21/03/2008	25/03/2008	9,16
18	Desarrollo de Test Units de la iteración 1	45 hrs	04/04/2008	13/04/2008	16
19	Primera versión del Sincronizador	15 hrs	13/04/2008	15/04/2008	16,17,18
20	**Construcción - Iteración 2**	**178 hrs**	**26/04/2008**	**01/06/2008**	**19**
21	Implementación de la comparación de estructuras	90 hrs	26/04/2008	13/05/2008	15
22	Pruebas de Aceptación	29 hrs	14/05/2008	20/05/2008	9,21
23	Desarrollo de Test Units de la iteración 2	44 hrs	20/05/2008	30/05/2008	21
24	Segunda versión del Sincronizador	16 hrs	30/05/2008	01/06/2008	21,22,23
25	**Construcción - Iteración 3**	**129 hrs**	**09/06/2008**	**06/07/2008**	**24**
26	Implementación de la sincronización de estructuras	90 hrs	09/06/2008	28/06/2008	15
27	Pruebas de Aceptación	12 hrs	28/06/2008	30/06/2008	9,26
28	Desarrollo de Test Units de la iteración 3	20 hrs	30/06/2008	04/07/2008	26
29	Tercera versión del Sincronizador	10 hrs	05/07/2008	06/07/2008	26,27,28
30	**Transición**	**18 hrs**	**06/07/2008**	**11/07/2008**	**29**
31	Versión Final del Sincronizador	10 hrs	06/07/2008	08/07/2008	29
32	Manual de Usuario	8 hrs	08/07/2008	11/07/2008	31

Figura 1.12 Diagrama GANTT – Lista de Actividades

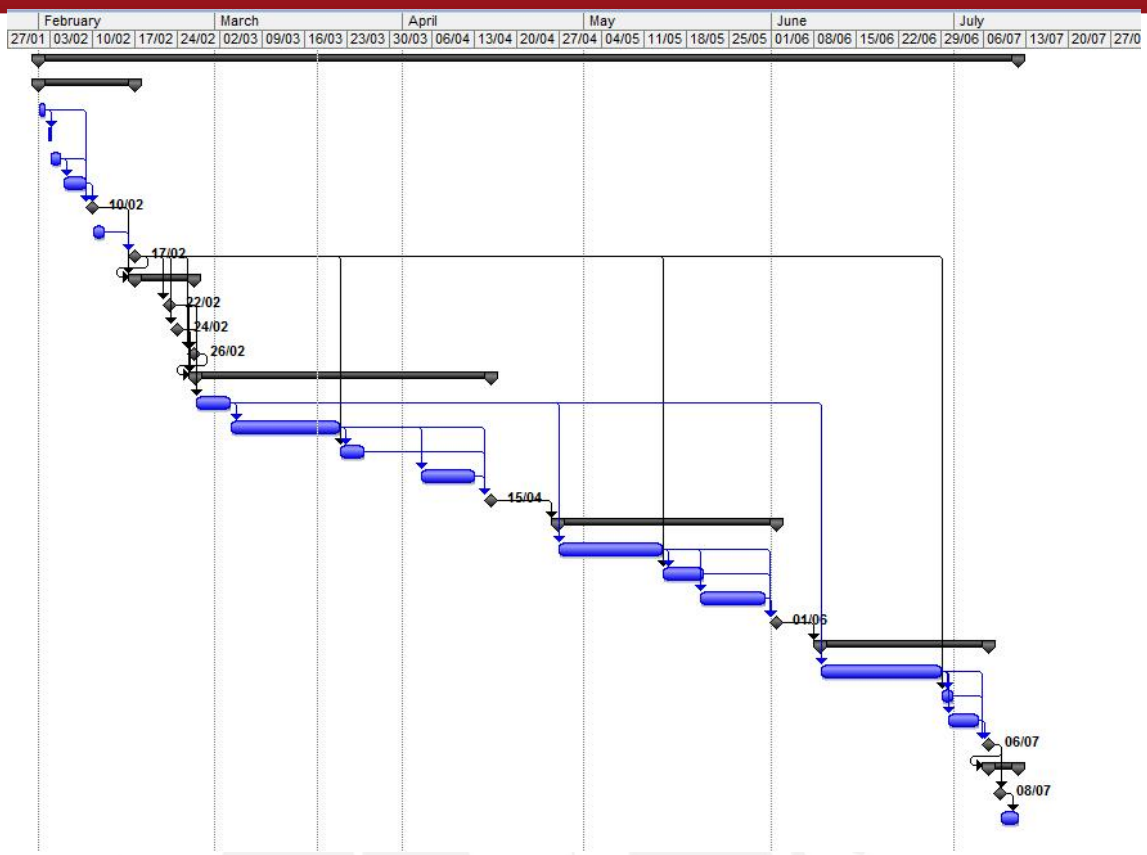


Figura 1.13 Diagrama GANTT – Calendario de Actividades

A lo largo del desarrollo del proyecto, se realizarán los procesos de monitoreo y control de cambios descritos en el acápite Metodologías para Gestión de Proyectos.

Con respecto a la fecha de entrega del Proyecto de Tesis, esta fecha corresponde a la entrega del proyecto completo (es decir cuando se está cursando Proyecto de Tesis 2) durante el presente semestre. La fecha programada es un aproximado, por lo cual cuando se obtenga la fecha real de entrega se actualizará el diagrama de GANTT.

La estimación de costos y el plan de compra y adquisiciones serán descritos en la sección de análisis de costo y beneficio y viabilidad del proyecto perteneciente al Capítulo 2.

1.3.4. Identificación de Riesgos del Proyecto

De acuerdo a la naturaleza del proyecto se ha podido establecer los siguientes riesgos que pueden obstaculizar el desarrollo del mismo a lo largo de las distintas fases del proyecto.

Los riesgos serán presentados en la tabla 1.1, en donde figurará el nombre del riesgo, si la probabilidad que suceda es baja, media o alta y si es un riesgo de alto, medio o bajo impacto en el desarrollo de la solución. En base a esta información se podrá confeccionar un adecuado plan de respuesta ante los mismos.

RIESGO	PROBABILIDAD DE QUE SUCEDA	IMPACTO
Las actividades definidas en el diagrama GANTT no han sido distribuidas de manera equitativa.	MEDIA	ALTO
No realizar el seguimiento de las tareas y actividades definidas.	MEDIA	ALTO
No controlar ni hacer seguimiento a los riegos.	MEDIA	ALTO
Realizar más esfuerzo de lo estimado debido a la falta de conocimiento de algunas herramientas o tecnologías.	ALTA	MEDIO
Los requerimientos son demasiados ambiciosos para el tiempo disponible.	BAJA	ALTO
Los requerimientos cambian demasiado a lo largo del proceso de desarrollo.	MEDIA	ALTO
Elaborar un diseño demasiado sencillo y no cubre todas las funcionalidades.	BAJA	MEDIO
Elaborar un diseño demasiado complejo que exija más tiempo en complicaciones ocasionadas.	MEDIA	MEDIO
No se han especificado todas las interfaces u clases necesarias.	MEDIA	MEDIO
Se desarrollan funciones innecesarias no descritas en el modelo de diseño.	ALTA	BAJO
El diseño no respeta la arquitectura planteada.	MEDIA	ALTO
La arquitectura escogida no abarca el cumplimiento de todos los requerimientos.	MEDIA	ALTO
La curva de aprendizaje del entorno de programación es demasiada larga y demanda bastante tiempo.	BAJA	BAJO
Las librerías provistas por el lenguaje de programación no permiten implementar las funcionalidades requeridas.	BAJA	BAJO
El esquema de pruebas desarrollado no permite probar todas las funcionalidades de la aplicación.	BAJA	MEDIO

El conocimiento del problema no es lo suficiente y está enfocado de una manera distinta.	BAJA	ALTO
Haber dejado de lado artefactos de software necesarios para un mejor diseño de la solución.	BAJA	BAJO
No poder terminar la solución en las fechas indicadas o tener retrasos en las liberaciones de la aplicación.	MEDIA	ALTO

Tabla 1.1 Riesgos del Proyecto

Como se aprecia en el cuadro anterior, los riesgos relacionados a la planificación del proyecto tiene un impacto mucho mayor en el desarrollo de la solución debido a que puede originar que las actividades planificadas no se realicen en el tiempo adecuado y se tenga que dejar de largo algunas de ellas.

Así mismo, los riesgos relacionados a la implementación tienen poca probabilidad de aparición, debido a que los actuales lenguajes de programación y sus respectivos IDEs brindan una serie de librerías y funcionalidades que permiten que la programación no sea un obstáculo para el desarrollo de la solución.

Finalmente, se puede concluir además que el impacto de un mal diseño o una mala arquitectura pueden repercutir de manera negativa en el desarrollo del producto, pues de este modelamiento deriva la implementación de la herramienta, la cual según el calendario de actividades, es la actividad que toma más tiempo en llevarse a cabo. Es por esta razón, que debe ponerse bastante cuidado y énfasis en el modelamiento de requerimientos, análisis y diseño.

1.3.5. Plan de respuesta ante los riesgos

Identificados los riesgos y su impacto sobre el desarrollo del producto, se ha elaborado el siguiente plan de respuesta para minimizar el impacto o en el mejor de los casos evitarlos.

a. En lo referente a los riegos relacionados a la etapa de planificación, se ha decidido invertir más tiempo en definir los procesos necesarios para el desarrollo del proyecto. Si bien es un tiempo que se puede utilizar para la investigación de uso de herramientas, se compensa con el tiempo que se puede ahorrar al momento de la fase de construcción del sistema para evitar realizar funcionalidades innecesarias o que no cumplan con los requerimientos.

b. Ante un posible cambio en los requerimientos, se procederá a analizarlos y el proceso de desarrollo se detendrá mientras dure el análisis de dicho requerimiento. De acuerdo al impacto del requerimiento, se continuará con el desarrollo cambiando de manera ágil los modelos de análisis y diseño en caso de que sean necesarios.
c. La arquitectura de software seleccionada será puesta a prueba antes de pasar al diseño, de manera que se eviten incoherencias entre la arquitectura y el diseño. Para esto se implementará un caso de uso y corroborar que la información entre capas fluya de manera esperada.
d. Se validará de manera ágil el diseño de la solución implementando los casos de uso más sencillos y probándolas con las pruebas de aceptación. De esta manera, se podrá identificar si realmente el diseño propuesto es capaz de soportar los requerimientos.
e. Se invertirá tiempo en la investigación de las librerías necesarias para el desarrollo de la solución. Los criterios de selección de las librerías serán los siguientes: facilidad de uso, documentación suficiente y que tengan las funcionalidades solicitadas. Entre las librerías más importantes están las encargadas de realizar una conexión a base de datos, generar reportes y ejecutar sentencias SQL.
f. Como parte de la metodología de desarrollo de la solución, la implementación de las pruebas se harán en paralelo con la codificación, de manera que se puedan probar de manera rápida y ágil las funcionalidades implementadas previas a las liberaciones por cada iteración.

1.4. Estado del arte

En primer lugar se mencionará el nombre de las aplicaciones encontradas junto con una pequeña descripción de las principales características del producto. Posteriormente, se explicará el proceso manual de la sincronización de bases de datos. Finalmente, se presentará en una tabla de doble entrada la información de los productos y sus características, de manera que se pueda realizar una comparación de los beneficios que ofrecen estos productos.

1.4.1. Aplicaciones actuales

Cross-Database Studio 6.0. Aplicación que ofrece las funcionalidades de comparación, sincronización y replicación entre distintos tipos de bases de datos. Posee una amplia gama de propiedades y reglas para los procesos de comparación y replicación.

En los escenarios en donde las bases de datos de origen y destino sean de distintos tipos, el usuario no se ve en la necesidad de configurar manualmente las relaciones entre los tipos de datos y objetos de ambas bases de datos debido a que el programa realiza dichas conversiones una vez que el usuario haya seleccionado los elementos a comparar correspondientes.

Entre otra característica importante se tiene que la aplicación permite al usuario programar los procesos antes mencionados para ejecutarse en un determinado momento. Además, no es necesario instalar distintos controladores ODBC de acuerdo al tipo de base de datos que se esté utilizando dado que el programa trae en su paquete de instalación un juego de los principales controladores reduciendo así la labor al usuario de instalar y configurar manualmente dichos controladores.

Es importante mencionar que la comparación, sincronización y replicación se puede realizar a nivel de estructura del objeto de la base de datos o a nivel de información (solo en el caso de tablas y vistas). A nivel de estructura se pueden realizar dichas operaciones en tablas, vistas, índices, llaves primarias, llaves foráneas y procedimientos almacenados. De igual manera la replicación se puede llevar a cabo a nivel de estructura e información.

Finalmente, el programa ofrece la posibilidad de emitir los resultados de la comparación de objetos en archivos con formato HTML y XML.

A continuación en la figura 1.14 que resume lo expresado anteriormente sobre el producto. Este gráfico fue tomado de la página web oficial del producto (Ver referencia [WEB001]).

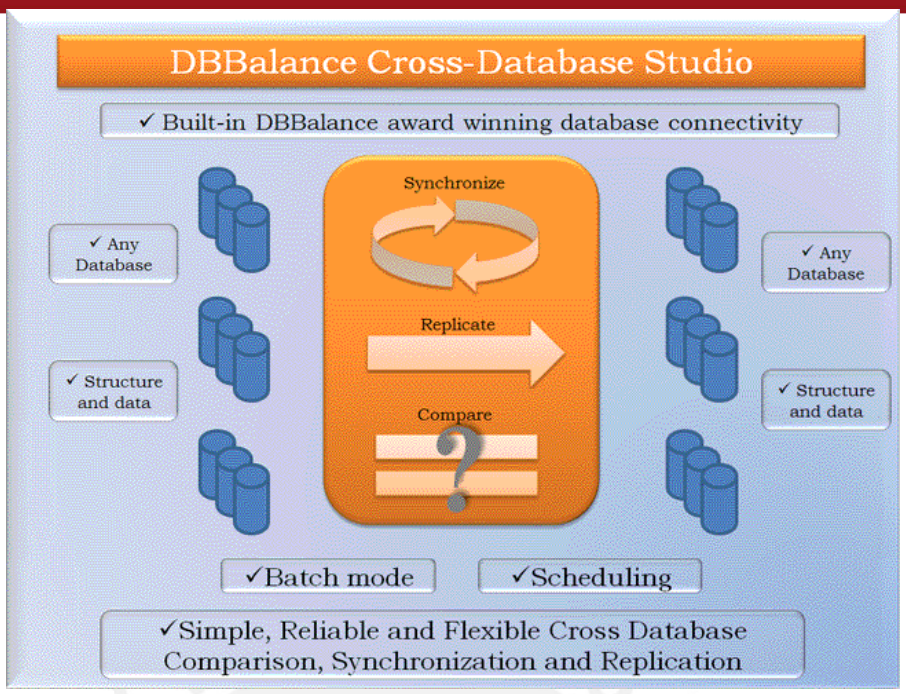


Figura 1.14 Funcionalidades de Cross Database Studio

Como se aprecia en la figura anterior, la aplicación permite realizar las operaciones de sincronización, replicación y comparación de estructura y datos entre bases de datos de distintos fabricantes, incluso permitiendo la programación de dichas labores en ciertas horas definidas por el usuario.

SQL Compare. Aplicación de la empresa Redgate que permite realizar los procesos de comparación y sincronización entre bases de datos SQL Server. Dichos procesos solo se hacen a nivel de estructura debido a que la empresa cuenta con otro software exclusivamente para la comparación y sincronización de información, SQL Data Compare.

A diferencia del anterior software, SQL Compare puede realizar la comparación y sincronización a nivel de archivos de sentencias SQL, además de realizarlo con bases de datos que están en pleno funcionamiento.

Los procesos de comparación y sincronización pueden ser llevados a cabo en tablas, vistas, procedimientos almacenados, funciones de usuario, esquemas de objetos XML y archivos de ensamblador CLR.

Otra característica importante de este producto es que permite grabar imágenes instantáneas (snapshots) de bases de datos previo a la sincronización, ofreciendo

la posibilidad de realizar procesos de recuperación de información (rollback) en caso de presentarse algún error. Además, dichas imágenes pueden ser utilizadas en procesos de auditoría.

Finalmente, SQL Compare permite la emisión de reportes en archivos Excel y HTML mostrando los cambios realizados en la base de datos luego de realizarse el proceso de sincronización, permitiendo así llevar pistas de auditoría sobre la base de datos.

La información sobre este producto fue obtenida de la página oficial. (Ver referencia [WEB003].

Database Restyle. Componente construido con tecnología .NET que puede ser integrado con una aplicación también desarrollada bajo .NET (pudiendo ser una aplicación de consola, con uso de ventanas o proyectos WEB), cuya función principal es realizar la sincronización desde la aplicación. El producto está desarrollado para trabajar sobre bases de datos SQL Server 2005.

La sincronización origina una nueva estructura de la base de datos la cual es grabada en un archivo XML y puede ser distribuido en diferentes formas (sea en un archivo o como recurso). Finalmente, dicho archivo es aplicado a la base de datos para realizar los cambios necesarios.

Debido a su integración con tecnología .NET, los desarrolladores puede crear la estructura de la base de datos a partir de un modelo LINQ to SQL (modelo de programación que permite modelar bases de datos relacionales utilizando clases de .NET). Tomando esta estructura como base, también se pueden realizar las operaciones de creación o actualización de bases de datos mediante la sincronización.

Además, este componente ofrece la posibilidad construir una base de datos usando programación. Se puede partir de modelos de objetos de negocio y mediante la generación de procedimientos permitir la sincronización entre dicho modelos y la base de datos.

A continuación se presenta la figura 1.15 que resume las principales características del producto. Esta imagen fue tomada de la página web oficial del producto (Ver referencia [WEB005]).

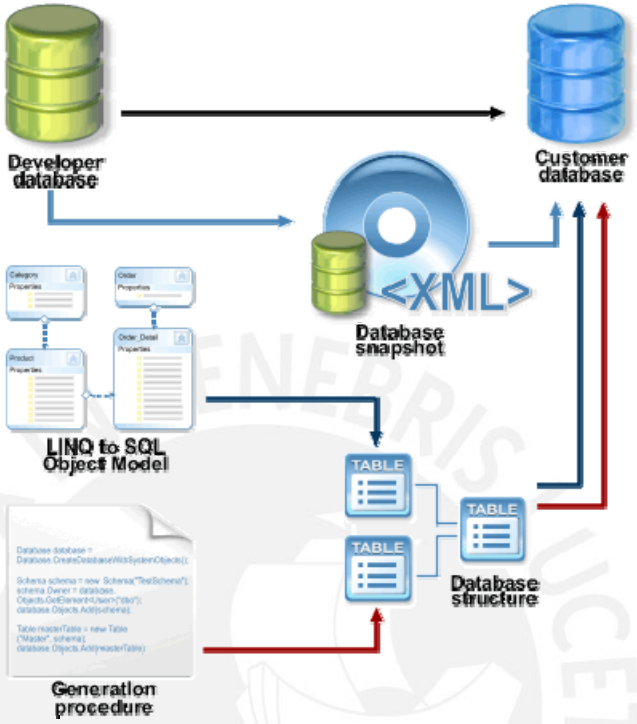


Figura 1.15 Funcionamiento del componente Database Restyle

Database Workbench. Aplicación para el desarrollo y diseño de bases de datos que, aparte de tener las funcionalidades de comparación y migración entre distintas bases de datos, posee distintas herramientas que facilitan al administrador de base de datos su labor en tareas de mantenimiento y desarrollo como diseño lógico y físico de bases de datos, procesos de depuración, importación y exportación de datos y herramientas de impresión de documentación de los objetos de la base de datos.

Esta aplicación actualmente soporta distintas bases de datos como SQL Server, Oracle, MySQL, Nexus BD entre otras.

En lo que se refiere a su herramienta de comparación, esta ofrece la posibilidad de realizar dicho proceso en tablas, vistas, índices, llaves de integridad y triggers, pudiendo estas 3 últimas ser obviadas en el proceso de comparación de acuerdo a las preferencias del usuario.

Los resultados de la comparación se visualizan de una manera fácil de entender y permite entrar a detalle en los resultados encontrados. En caso de encontrar un objeto distinto entre las bases de datos, la herramienta ofrece 4 posibilidades para realizar la sincronización: crear el objeto en el destino, remover el objeto del destino, modificar el objeto en el destino o realizar un proceso de fusión entre los objetos comparados (agregando los componentes necesarios como columnas, índices, entre otros).

Cabe mencionar, que la herramienta de comparación no permite aplicar los cambios directamente en la base de datos destino, solo genera un archivo con las sentencias SQL para realizar dichas modificaciones.

En la figura 1.16 se puede apreciar la presentación de los resultados de la comparación entre objetos de distintas bases de datos.

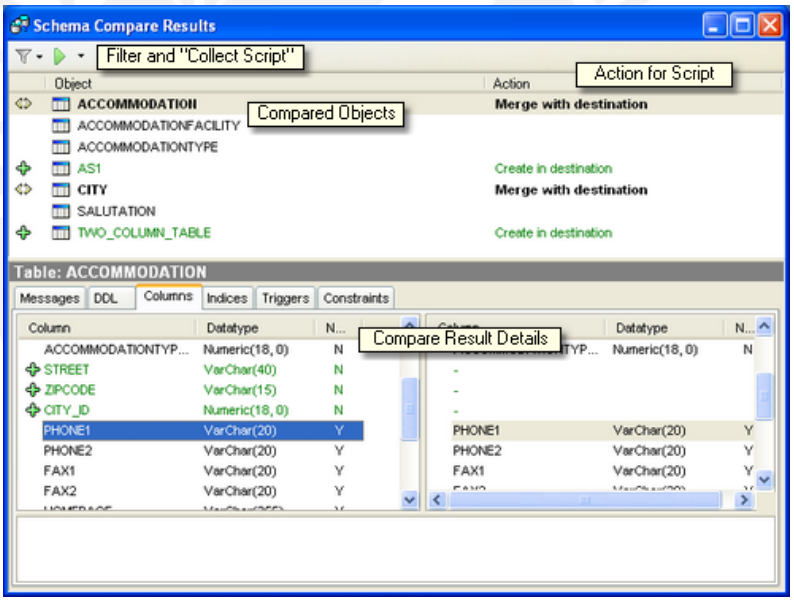


Figura 1.16 Resultado de la comparación mediante la herramienta Database Workbench

La información y el gráfico sobre este producto fueron obtenidos de la página oficial del mismo (Ver referencia [WEB006]).

Easy SQL Schema Compare. Herramienta que cubre las funcionalidades de detectar, analizar y sincronizar entre distintas bases de datos SQL Server.

Entre las características más resaltantes de esta herramienta están que permite visualizar las diferencias entre los elementos que se están comparando, generación

del archivo de sentencias SQL para realizar la sincronización y un asistente amigable para el usuario para realizar el proceso de sincronización.

Dado que su funcionamiento está restringido a bases de datos SQL Server, posee un editor de sentencias similar al utilizado en la herramienta Query Analyzer de Microsoft, permitiendo diferencias las distintas regiones dentro del archivo de sentencias SQL.

La información de este producto fue obtenido de la referencias [WEB008].

UDB Workbench. Herramienta especializada para trabajar en ambientes de desarrollo de bases de datos DB2 UDB. Ofrece distintas características como un editor de SQL con autocompletado y resaltado de sintaxis, ejecución de scripts, edición, filtrado, agrupamiento de resultados de consultas y finalmente, comparación y sincronización de distintos esquemas de bases de datos.

Permite la comparación y sincronización tanto a nivel de objetos e información dentro de una base de datos, resaltando con distintos colores las diferencias entre los objetos que están siendo comparados. Además brinda la posibilidad de generar el archivo con sentencias SQL para realizar la sincronización y previa a su ejecución, alerta al usuario de distintos tipos de advertencias como truncamiento de la información, conversión incompatible de tipos de datos entre otros.

Finalmente, brinda la posibilidad al usuario de realizar la sincronización de manera manual mediante el archivo SQL de sincronización. Esto permite al usuario revisarlo y ejecutarlo de manera manual.

La información de este producto fue obtenido de la página oficial del mismo (Ver referencia [WEB009].

DB Difference Expert. Herramienta especializada en realizar la comparación de distintas bases de datos, esquemas u objetos que pueden residir en distintos servidores Oracle.

Entre los objetos válidos dentro del proceso de comparación están tablas, índices, llaves de integridad, vistas, paquetes, procedimientos almacenados, funciones y tipos definidos por el usuario y triggers.

Los resultados de la comparación pueden ser mostrados de 3 distintas maneras permitiendo al usuario cierta flexibilidad al momento de visualizar los resultados: un estilo de árbol con nodos expandibles, código SQL que permite además generar el archivo de sentencias para sincronizar la base de datos o mediante un reporte de comparación optimizado para su impresión.

Una característica particular de este producto es que permite crear y mantener un repositorio central en donde almacena distintas imágenes de la base de datos luego de pasar por el proceso de comparación y sincronización permitiendo de esta manera, poder realizar una auditoría a los objetos que han ido cambiando a lo largo de un periodo de tiempo. Con esta funcionalidad, esta aplicación puede comparar la base de datos con dichas imágenes almacenadas en su repositorio.

Esta información fue obtenida del documento en la referencia [PPT001].

1.4.2. Proceso manual de sincronización

En la presente sección se explica el procedimiento manual para realizar la sincronización de una base de datos. Los pasos para realizar dicho procedimiento son:

1. *Analizar los cambios*

Los pasos en esta fase son los siguientes:

- Analizar manualmente el esquema SQL modificado para encontrar todos los cambios que se han realizado en los distintos objetos, de manera que se puedan propagar dichos cambios hacia la otra base de datos.
- Revisar si los cambios encontrados no afectan en términos de dependencia a otros objetos de la base de datos.
- Analizar si los cambios propuestos afectan el funcionamiento del sistema. Si esto ocurre se debe coordinar con los desarrolladores para resolver el inconveniente.

2. *Generar los scripts de sincronización y/o migración*

Una vez terminada la fase de análisis, se siguen los siguientes pasos:

- Basado en los cambios encontrados, se debe codificar de manera manual los scripts de sincronización con las sentencias SQL para actualizar el esquema destino en donde se quieren reflejar los cambios.
- Asegurarse que las dependencias entre los objetos de la base de datos están siendo manejados de manera adecuada para asegurar que el sistema siga funcionando como lo ha estado haciendo previo a los cambios.
- Probar los scripts para asegurar la integridad y funcionalidad de la base de datos.

3. Ejecutar y salvar los scripts con fines de mantenimiento

Finalmente, los pasos para esta fase son las siguientes:

- Realizar una copia de seguridad de la base de datos destino, en caso de presentarse alguna falla al momento de la sincronización.
- Ejecutar los scripts de sincronización mediante el uso de una herramienta para ejecutar sentencias SQL de acuerdo al manejador de base de datos. (Por ejemplo, se puede utilizar la herramienta Query Analyzer de Microsoft en caso de contar con una base de datos SQL Server 2000).
- Generar un script de toda la base de datos destino después de ejecutado los scripts de sincronización y almacenarlo en un sistema de versiones para futuras referencias.
- Almacenar los scripts de sincronización para efectos de auditoría (Tener evidencia de todos los cambios realizados a la base de datos).

Este proceso se realiza cada vez que se presentan cambios en la estructura o datos de una base de datos y trae consigo los siguientes inconvenientes:

- Exceso consumo de tiempo en identificar de manera manual toda la lista de cambios de la base de datos, lo cual está propenso a error.
- La codificación manual de los scripts de sincronización tomando en cuenta todas las dependencias está propenso a error y es ineficiente por el consumo de tiempo.

A continuación se muestra en la tabla 1.2 un cuadro comparativo de las características que poseen las herramientas anteriormente descritas.

Soluciones / Características	Cross-Database Studio	SQL Compare	Database Restyle	Database Workbench	Easy SQL Schema Compare	UDB Workbench	DB Difference Expert
Multiplataforma (comparar bases de datos de distintos fabricantes):							
- MSSQL Server	x	x	x	x	x		
- Oracle	x			x			x
- MySql	x			x			
- Microsoft Access	x						
- UDB Database						x	
Permiten realizar una sincronización inmediata de bases de datos a partir de los resultados de la comparación.	x		x		x		
Permiten generar un archivo de sentencias SQL para realizar la sincronización de manera manual.	x	x		x	x	x	x
Permiten realizar migración o replicación de bases de datos.	x			x			
Comparan los siguientes objetos:							
- Tablas y vistas	x	x	x	x	x	x	x
- Índices	x	x	x	x		x	x
- Constraints (llaves de referencia)	x	x	x	x	x	x	x
- Procedimientos almacenados	x	x			x	x	x
- Disparadores (triggers)	x	x		x		x	x
- Funciones definidas por el usuario		x			x		x
- Tipos de datos definidos							x
- Objetos XML		x			x		
Sincronizan los siguientes objetos:							
- Tablas y vistas	x	x	x	x	x	x	x
- Índices	x	x	x	x		x	x
- Constraints (llaves de referencia)	x	x	x	x	x	x	x
- Procedimientos almacenados	x	x			x	x	x
- Disparadores (triggers)	x	x		x		x	x
- Funciones definidas por el usuario		x			x		x
- Tipo de datos definidos							x
- Objetos XML		x			x		
Realizan la comparación a nivel de estructura de los objetos.	x	x	x	x	x	x	x
Realizan, además, la comparación a nivel de información.	x						

Exportan los resultados de la comparación a HTML, XML o Excel.	x	x					x
Permiten programar tiempos para realizar la comparación y sincronización.	x						
Almacenan imágenes de la base de datos previo a los cambios de la sincronización en un repositorio.		x	x				x

Tabla 1.2 Cuadro de comparación de las herramientas encontradas

1.5. Descripción y sustentación de la solución

La solución que se plantea para resolver el problema de la falta de una herramienta automatizada para realizar la sincronización de bases de datos es la implementación de un comparador y sincronizador a nivel de estructura de bases de datos. Esta solución permitirá, además de identificar cuáles han sido las modificaciones entre dos versiones de una base de datos, la sincronización automática de dichos cambios permitiendo así la actualización segura de la base de datos.

Si bien la solución permite una sincronización inmediata a partir de los resultados de la comparación, también contempla la posibilidad de generar un archivo con sentencias SQL para sincronizar de manera manual una base de datos. Dicho archivo luego puede ser ejecutado desde cualquier editor de sentencias SQL dependiendo del sistema de administración de bases de datos que se tenga instalado.

La solución permitirá comparar y sincronizar los siguientes objetos de bases de datos: tablas, índices, llaves primarias y foráneas, check constraints, vistas, definiciones de los triggers y parámetros de los procedimientos almacenados. El propósito es identificar las diferencias entre la información de la metadata de distintas versiones de bases de datos y poder sincronizar dicha información si se cree conveniente.

La solución se desarrollará bajo una arquitectura que permita conectarse a distintos DBMS sin necesidad de recompilar todo el código fuente en caso de que se desee agregar un nuevo DBMS. Inicialmente se propone que la aplicación sea probada con 3 manejadores de bases de datos. Dichos manejadores son: MSSQL Server 2000, Oracle 10g y MySql 5.0. De esta manera se ofrece un producto multiplataforma que puede adecuarse a distintos ambientes ofreciendo las mismas funcionalidades.

La solución permitirá visualizar las bases de datos a ser comparadas en forma de esquema de árbol, en donde el usuario podrá seleccionar los objetos a ser comparados y mediante una serie de opciones, como ignorar mayúsculas o

minúsculas, no tomar en cuenta el orden de las columnas entre otras, podrá configurar la comparación de acuerdo a las necesidades del caso.

Los resultados de la comparación se mostrarán de una manera amigable para el usuario, en donde se resaltarán con distintos colores cuáles han sido los elementos que han sido agregados, modificados o eliminados entre versiones distintas de base de datos y de acuerdo a esa información, el usuario podrá elegir que acciones tomar con respecto a la sincronización.

Además de lo mencionado en el párrafo anterior, los resultados de la comparación podrán ser exportados a archivos en formato PDF y HTML de manera que el usuario pueda guardarlos en medios de almacenamiento para propósitos de ayuda o para mantener un historial de cambios.

La solución mostrará advertencias previo al proceso de sincronización informando al usuario de posibles incompatibilidades de tipos de datos o inconsistencia de datos en lo que se refiere a la sincronización de información (típicos casos de inconsistencia por duplicidad de llaves primarias). Así mismo, al finalizar el proceso de sincronización, se grabará un archivo de bitácora con los resultados del proceso el cual puede ser visualizado inmediatamente después de terminado el proceso de sincronización.

Con todo lo descrito anteriormente, el buen uso de esta herramienta por parte de los administradores de bases de datos y desarrolladores les permitirá reducir el tiempo utilizado en los procesos de actualización y sincronización de bases de datos y estar menos propensos a errores relacionados a sintaxis de las sentencias SQL y consistencia de datos.

2. Análisis

En este capítulo se definirán los aspectos relacionados con la metodología de desarrollo de la aplicación, el análisis de viabilidad proyecto y finalmente se verá el análisis de la solución que incluye el diagrama de clases de análisis.

2.1. Definición de la metodología de la solución

En esta sección se describirá los aspectos a la elección de la metodología para el desarrollo de la solución así como el desarrollo de la misma.

2.1.1. Elección de la metodología

Para implementar la solución propuesta se ha optado por una metodología que permita llevar a cabo un proceso de desarrollo de manera ágil y que incluya las especificaciones y documentación necesarias para modelar los requisitos obtenidos a partir del problema ya descrito y desarrollar los modelos de análisis y diseño de la solución. De esta manera, se busca tener una base sólida al momento de

implementar la solución y elaborar un plan de pruebas que permita validar la herramienta contra los requerimientos solicitados.

Siendo la premisa principal el tener un modelo de proceso de la solución ágil en el sentido de la establecer los requerimientos, modelar la solución e implementarlo de acuerdo a dicho modelamiento, se ha optado por escoger la metodología AUP (Agile Unified Process) para el desarrollo de la herramienta propuesta.

Esta metodología es una versión simplificada de la metodología RUP (Rational Unified Process), que tiene como principios desarrollar un software utilizando técnicas de modelamiento ágiles y documentación necesarios y suficientemente buenos para el entendimiento del problema y el desarrollo de la solución.

A continuación se describen algunos principios del desarrollo ágil de software (Agile Software Development) que son apropiados para el desarrollo de la herramienta y que justifican la elección de AUP como la metodología de la solución.

- **Evolución de requerimientos.** Si bien al inicio de todo desarrollo de software se busca capturar todos los requerimientos, existen factores externos que ocasionan que dichos requerimientos cambien y evolucionen o aparezcan otros de acuerdo a nuevas necesidades. Esto a su vez conlleva a realizar cambios en los modelos de análisis y diseño para adecuarlos a los nuevos requerimientos. Tomando en consideración lo descrito anteriormente, se buscar capturar esos nuevos requerimientos y darles la apropiada prioridad para no perjudicar el trabajo que pudo haber avanzado durante ese tiempo.
- **Desarrollar el producto en pequeñas iteraciones y liberar versiones del producto en tiempos incrementales.** Con esta característica, se espera ir implementando por iteración una cierta cantidad de funcionalidades de manera que se puedan realizar pruebas y analizar si se están cumpliendo con los requisitos.
- **Realización de pruebas a lo largo del ciclo de desarrollo del software.** A diferencia de metodologías tradicionales, el desarrollo ágil de software propone realizar las pruebas a lo largo de todo el desarrollo mediante la implementación de unidades de pruebas que pueden ser reutilizadas, asegurando de que todas las características están funcionando correctamente durante la liberación de pequeñas versiones y de la liberación del producto final.

Los artefactos de software necesarios para el entendimiento de la solución estarán basados en los diagramas UML correspondientes, los cuales serán descritos en cada fase de la metodología.

2.1.2. Desarrollo de la metodología

Como se mencionó anteriormente, la metodología para desarrollar la herramienta es AUP, cuyo ciclo de vida se muestra en la figura 2.1. Posteriormente, se explicará la derivación de su proceso mediante las fases y disciplinas que posee la metodología.

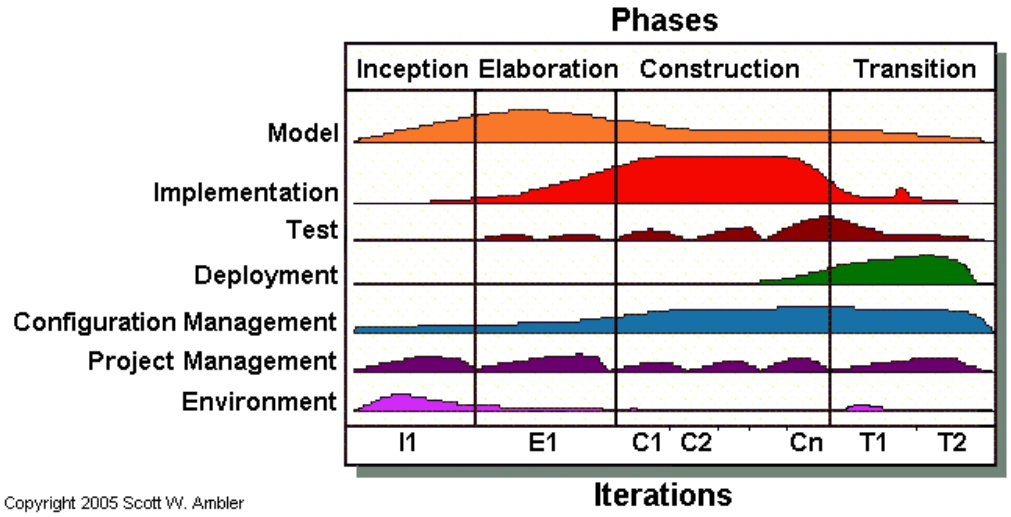


Figura 2.1 Ciclo de vida de AUP

2.1.2.1. Fase de Incepción

El objetivo de esta fase es la de establecer a un alto nivel los requerimientos de la aplicación, los cuales van a ser modelados en un diagrama de casos de uso y su correspondiente especificación.

Además, durante esta fase inicial se realizarán las estimaciones de costo y programación de tareas (las cuales ya han sido establecidas en el diagrama de GANTT y el diagrama WBS durante la etapa de planificación). Con dicha información se pueden establecer las iteraciones de desarrollo para la fase de construcción, las cuales serán descritas posteriormente.

Adicionalmente, se deberá establecer una lista de los riesgos del proyecto ordenados por prioridad de tal manera que los riesgos de alta prioridad sean mitigados durante las primeras iteraciones y evitar obstaculizar el desarrollo del producto durante las fases posteriores.

La definición de riesgos ya fue establecida en el Capítulo 1 de la presente tesis, en el acápite de Plan del Proyecto

Finalmente, se tendrá que establecer que requerimientos de hardware y software son necesarios en corto y largo plazo para desarrollar la aplicación.

2.1.2.2. Fase de Elaboración

El principal objetivo de esta fase es elaborar y probar la arquitectura del sistema. Dicha arquitectura deberá satisfacer los requerimientos ya definidos y servirá como base para la fase de construcción. Para ilustrar la arquitectura del sistema, se utilizarán los diagramas de componentes para mostrar la distribución del sistema a través de distintas capas y como estas interactúan entre ellas.

2.1.2.3. Fase de Construcción

Esta fase corresponde al desarrollo en sí de la aplicación. Previo a la codificación, se tendrá que definir el modelo de análisis y diseño de la solución, para lo cual se usarán los diagramas de clases de análisis y los diagramas de clases de diseño con sus correspondientes descripciones y especificaciones.

Para esta fase se han establecido 3 iteraciones de acuerdo a lo establecido en la fase de incepción y a la planificación del proyecto:

Iteración N° 1. Comprende la codificación de las funcionalidades relacionados a la conexión hacia las fuentes de datos origen y destino, así como la carga de la estructura de ambas bases de datos.

Iteración N° 2. Comprende la codificación de las funcionalidades relacionadas a la comparación de las estructuras de las bases de datos.

Iteración N° 3. Comprende la codificación de las funcionalidades relacionadas a la sincronización de las estructuras de las bases de datos.

En paralelo con la codificación, se implementarán las pruebas necesarias para probar cada funcionalidad, de manera que una vez terminada una iteración, las funcionalidades implementadas puedan ser corroboradas y en caso de que sea necesario, se hagan las correcciones pertinentes. Para llevar a cabo este trabajo, se realizarán los siguientes tipos de pruebas:

- **Pruebas de aceptación.** Permitirán corroborar que los requerimientos han sido resueltos de manera satisfactoria.

- **Unidades de Prueba (Unit Test)**. Son pruebas de caja blanca que verifican que el código para implementar una funcionalidad sea el correcto. Estas pruebas verifican el comportamiento del sistema a un bajo nivel (nivel de código), así como verificar que el diseño sea correcto. Estos tipos de pruebas suelen ser independientes de las funcionalidades pues verifican el código fuente. Para esto tipo de pruebas se utilizarán **JUnits** (Unit Test correspondientes al lenguaje Java).

La elección de ambos tipos de pruebas se justifica en el hecho de que son complementarias y necesarias. Las pruebas de aceptación permitirán verificar que los requerimientos se estén implementando, mientras que las unidades de prueba verificarán que el código no contenga errores.

2.1.2.4. Fase de Transición

En esta etapa, se continuarán con el conjunto de pruebas definidos en las fases anteriores previo a la puesta en producción de la herramienta. En caso de ser necesario, se realizarán algunos arreglos a la codificación de acuerdo al resultado de las pruebas.

2.2. Identificación de requerimientos

Los requerimientos que se listarán a continuación se obtuvieron mediante una serie de preguntas a dos administradores de bases de datos relacionadas a las necesidades e inconvenientes que tiene al realizar una sincronización de bases de datos de manera manual y en base a esta información se pudo elaborar el catálogo de requerimientos.

Adicionalmente, se observó que necesidades cubría las aplicaciones descritas en el Estado del Arte, con el fin de contrastarlas con los requerimientos obtenidos anteriormente y consolidar un catálogo de requerimientos final.

A continuación se presentará la lista de requerimientos funcionales y no funcionales. Posteriormente, se explicará el diagrama de casos de uso utilizado para modelar los requerimientos.

2.2.1. Requerimientos funcionales

a) Relacionados a la conexión de base de datos y carga de datos.

1. La aplicación permitirá establecer dos conexiones a distintas bases de datos, las cuales pueden ser de los siguientes fabricantes: SQL Server, Oracle y MySql.
2. La aplicación permitirá cargar la estructura de ambas bases de datos mostrando lo siguiente:

- Información de tablas: columnas, llaves primarias, llaves foráneas, check constraints y triggers.
- Información de vistas: columnas, índices y definición de la vista.
- Información de procedimientos almacenados: parámetros y definición.

b) Relacionados a la comparación de objetos de base de datos.

1. La aplicación permitirá comparar objetos de bases de datos a nivel de esquemas. Entre ellos están: tablas, vistas y procedimientos almacenados. Así mismo, se podrá ver el detalle de la comparación de

cada objeto (Por ejemplo, si 2 tablas son diferentes se puede detallar las diferencias a nivel de columnas, llaves y triggers).

2. La aplicación permitirá comparar las columnas de dos tablas al siguiente nivel de detalle: nombre de la columna, tipo de dato, longitud, precisión numérica, escala numérica, si acepta nulos y el valor por defecto de la columna.
3. La aplicación permitirá comparar los índices de dos tablas al siguiente nivel de detalle: nombre del índice, tipo de índice, unicidad y el nombre de las columnas a las que hace referencia el índice.
4. La aplicación permitirá comparar las llaves primarias de dos tablas al siguiente nivel de detalle: nombre de la llave primaria y el nombre de las columnas a las que hace referencia la llave primaria.
5. La aplicación permitirá comparar las llaves foráneas de dos tablas al siguiente nivel de detalle: nombre de la llave foránea, si está habilitada la llave, si la llave es del tipo diferido, si la llave valida la consistencia de todos los registros de la tabla, si realiza acciones en cascada al actualizar y eliminar un registro, el nombre de la tabla referida y el nombre de las columnas referentes y referidas.
6. La aplicación permitirá comparar los check constraints de dos tablas al siguiente nivel de detalle: nombre del check constraint, si está habilitada la llave, la regla que define el constraint y el nombre de las columnas a las que hace referencia.
7. La aplicación permitirá comparar la declaración de los triggers de dos tablas al siguiente detalle: si el trigger se dispara antes o después de un evento y los tipos de eventos por los cuales se dispara el trigger.
8. La aplicación permitirá comparar las columnas de dos vistas al siguiente nivel de detalle: nombre de la columna, tipo de dato, longitud, precisión numérica, escala numérica, si acepta nulos y el valor por defecto de la columna.
9. La aplicación permitirá comparar los índices de dos vistas al siguiente nivel de detalle: nombre del índice, tipo de índice, unicidad y el nombre de las columnas a las que hace referencia el índice.
10. La aplicación permitirá comparar la definición de dos vistas.
11. La aplicación permitirá comparar los parámetros de dos procedimientos almacenados a nivel de tipo y cantidad de parámetros utilizados.
12. La aplicación permitirá generar un reporte de comparación.

c) Relacionados a la sincronización.

1. La aplicación permitirá sincronizar todos los elementos de una tabla (columnas, llaves y triggers) de un esquema origen a otro destino.
2. La aplicación permitirá sincronizar las columnas de una tabla.
3. La aplicación permitirá sincronizar los índices de una tabla.
4. La aplicación permitirá sincronizar la llave primaria de una tabla.
5. La aplicación permitirá sincronizar las llaves foráneas de una tabla.
6. La aplicación permitirá sincronizar los check constraints de una tabla.
7. La aplicación permitirá sincronizar la declaración de los triggers de una tabla.
8. La aplicación permitirá sincronizar los índices de una vista.
9. La aplicación permitirá sincronizar los parámetros de un procedimiento almacenado.
10. La aplicación permitirá crear un script con las sentencias necesarias para realizar la sincronización.
11. La aplicación permitirá grabar en una bitácora cada proceso de sincronización.

Adicionalmente se han incluido funcionalidades referentes a la sincronización de objetos en un manejador Oracle, en el cual la sincronización se realiza mediante una llamada a un procedimiento almacenado para obtener la sentencia DDL de creación del objeto. Las funcionalidades adicionales son:

a) La aplicación permitirá comparar cualquier objeto de una base de datos Oracle.
b) La aplicación permitirá generar el script de sincronización de cualquier objeto de una base de datos Oracle.

2.2.2. Requerimientos no funcionales

1. La aplicación deberá presentar una interfaz gráfica fácil de usar e intuitiva para el administrador de base de datos (usuario que utilizará la aplicación).
2. La aplicación soportará las conexiones a distintos manejadores de bases de datos.

Como se observa, la aplicación va a permitir realizar una conexión a bases de datos de distintos fabricantes como MSSQL Server, Oracle, MySql entre otros. Una vez realizada la conexión, la aplicación mostrará la estructura de las bases de datos a sincronizar permitiendo al usuario escoger qué elementos desea comparar.

Una vez que el usuario haya decidido los elementos a comparar, la aplicación mostrará al usuario las diferencias entre los componentes seleccionados, brindándole la opción de sincronizar tales elementos entre las bases de datos o armar un script con las sentencias SQL para su posterior ejecución.

Finalmente, la aplicación permitirá generar un reporte en formato HTML o PDF con el resultado de la comparación. Adicionalmente se grabará una bitácora mostrando el resultado de la sincronización que permita obtener información del proceso realizado. Para ilustrar los requerimientos antes mencionados se ha elaborado el siguiente diagrama de casos de uso mostrado en la figura 2.2.

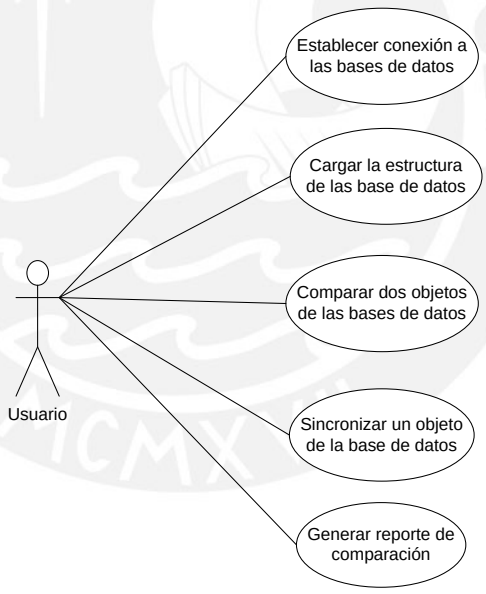


Figura 2.2 Diagrama de casos de uso

La especificación detallada de estos casos de uso se encuentra en la sección de Anexos (Anexo B: Especificación de los casos de uso).

2.3. Análisis de la solución

En la presente sección se presentará la viabilidad del proyecto, el análisis costo beneficio y finalmente la definición del sistema representando por el diagrama de

clases de análisis y su correspondiente sustentación de cómo cumple con los requerimientos definidos.

2.3.1. Viabilidad del proyecto

Para realizar el estudio de la viabilidad de la solución se tomó en cuenta 3 factores que repercuten de manera directa al desarrollo de la solución, con la finalidad que se pueda establecer el riesgo del proyecto.

Antes de realizar el análisis de la viabilidad, se definirán los roles profesionales que participarán en el proyecto, así como el tiempo estimado por cada uno. Este análisis permitirá definir una base para el estudio de la viabilidad del proyecto. Los roles son mostrados a continuación en la tabla 2.1.

Rol	Descripción	Tiempo estimado (en horas)
Analista	Realiza actividades de captura de requerimientos y análisis de la solución. Además se encarga de definir la arquitectura de la solución así como el diagrama de clases.	80
Programador	Realiza la implementación de la solución en base al diagrama de clases definido. Además se encarga de definir el esquema de pruebas para probar las funcionalidades de la solución.	250
Ejecutor de pruebas	Realiza la ejecución del esquema de pruebas definido. Esto incluye las pruebas de aceptación como pruebas unitarias,	170

Tabla 2.1 Roles profesionales del proyecto

a. Viabilidad económica

En este rubro se debe considerar los costos a incurrir en el desarrollo del proyecto, los cuales debido a la naturaleza de la solución están relacionados al software necesario para la fase de implementación de la aplicación. Entre los principales costos se tienen los siguientes:

- Costo relacionado a la compra de una licencia para operar un servidor SQL Server 2000. La edición sugerida para este software es la versión Deployment.
- Costo relacionado a la compra de una licencia para operar un servidor Oracle 10g.
- Costo relacionado a un IDE como entorno de programación.
- Costos relacionados al uso de energía eléctrica a lo largo de todo el proyecto.
- Costos relacionados al uso de herramientas CASE para la elaboración de los diagramas UML necesarios en las distintas fases de la metodología propuesta.

Como se aprecia, la mayoría de los costos están relacionados a la compra de software necesario para la implementación. Debido a que los costos relacionados a la compra de licencia de software son elevados en el mercado peruano, se ha optado por las siguientes alternativas que hacen más factible la realización del proyecto:

- Para el punto referente a la compra de licencia de un servidor SQL Server, se ha optado por la alternativa de coordinar con la dirección de informática de la universidad el uso de una base de datos de prueba, la cual pueda ser utilizada durante la fase de construcción de la aplicación.
- Para el punto referente a la compra de la licencia de un servidor Oracle, se ha optado por hacer uso de la versión gratuita de la versión 10g (correspondiente a la edición Express), la cual contiene los componentes necesarios para el desarrollo de la aplicación. Otra alternativa, es hacer uso de la versión gratuita de Oracle para Windows Vista, en caso de que las pruebas se hagan sobre un servidor Oracle en una máquina cuyo sistema operativo sea Windows Vista.
- Para el uso de un servidor MySql, no se presenta ningún costo relacionado a la compra de licencia, pues es un programa del tipo Open Source, el cual su instalación y uso no presenta costo alguno.
- Para el uso de las herramientas CASE, se ha optado por utilizar las herramientas provistas en los laboratorios de la universidad. Para la presente

solución, se hará uso de Microsoft Visio para la elaboración de los diagramas UML necesarios.

- Finalmente, el costo del IDE se verá mitigado por el uso de un IDE gratuito como lo es Netbeans, el cual es de libre distribución.

En resumen, se ha optado por alternativas que reduzcan el costo por compra y uso de software como la adquisición de software de libre distribución o uso de herramientas provistas por la universidad, teniendo como únicos costos los relacionados al uso de la energía eléctrica de los equipos en donde se desarrolle la solución.

b. Viabilidad técnica

Las restricciones de índole técnico están orientadas al uso de las tecnologías necesarias para desarrollar la solución. Para el presente proyecto, estas restricciones están relacionadas a la disponibilidad de las herramientas y del lenguaje de programación seleccionados para implementar todos los requerimientos definidos. Entre las restricciones más importantes se tienen:

- Disponibilidad de los servidores para realizar pruebas de conexión a bases de datos durante la fase de construcción.
- Librerías y APIs necesarios para lograr la conexión desde la aplicación hacia las bases de datos ya mencionadas (SQL Server, Oracle y MySql).
- Librerías necesarias para la conversión de tipos de datos entre las distintas bases de datos.
- Características del lenguaje de programación escogido para facilitar la conversión mencionada en el punto anterior.
- El IDE a utilizar contenga los componentes gráficos para implementar una interface agradable para el usuario.

c. Viabilidad legal

En lo referente a la viabilidad legal, se evitará cualquier posibilidad de infracción legal relacionado a la compra de software mediante el uso de software de libre distribución como los mencionados en la parte a) de esta sección. De esta manera, se espera brindar una solución con ningún problema legal referente a uso de licencias.

2.3.2. Análisis costo – beneficio

Como se mencionó en la sección anterior, los costos en los que se va a incurrir para el desarrollo de la implementación son evitados mediante el uso de software gratuito y de libre distribución.

Para establecer este análisis es necesario identificar los beneficios y ventajas que ofrecerá la herramienta a los usuarios. Como se mencionó en la descripción de la solución (Acápite 5 del Capítulo 1), el buen uso de esta herramienta permitirá que desarrolladores y administradores realicen la tarea de sincronizar la base de datos de manera rápida y segura, evitando cometer errores, los cuales pueden ser difíciles de encontrar en caso de que sea una sincronización que abarque muchas entidades de la base de datos.

Tomando en cuenta lo expuesto anteriormente, se puede decir que el proyecto es viable en los aspectos antes mencionados y los resultados que esperan obtener van a permitir que el usuario final tenga una herramienta confiable y automatizada para realizar la sincronización de bases de datos.

2.3.3. Definición del sistema

Para realizar la definición del sistema, se ha utilizado el diagrama de clases de análisis para representar las entidades necesarias en el modelamiento de la aplicación. Este esquema servirá como base para desarrollar las clases de diseño, generar los algoritmos para realizar la comparación y sincronización y, posteriormente, pasar a la fase de construcción de la aplicación. Para explicar

mejor el modelo, se ha dividido en tres paquetes: Paquete de Conexión a Base de datos, Paquete de Abstracción y Paquete de Lógica de Negocio.

2.3.3.1. Paquete de Abstracción del DBMS

En lo que respecta a la capa de Abstracción del DBMS, las clases que la conforman se encargan de implementar una interfaz que permite separar la lógica de los objetos de negocio y el tipo de DBMS que se esté utilizando.

La clase principal corresponde a la plantilla utilizada por las clases de negocio para obtener información sobre cada DBMS siguiendo algunos criterios definidos. Básicamente los métodos que ofrece esta clase permiten obtener la sentencia SQL para lectura de metadata, el catálogo de equivalencias entre DBMS y el formato de la sentencia DDL para la sincronización por cada objeto de la base de datos.

Las otras clases corresponden a las entidades encargadas de la lectura y almacenamiento de parámetros necesarios para la configuración de la aplicación, como por ejemplo el formato de la cadena de conexión hacia un DBMS en particular así como la ruta de la librería necesaria para establecer una conexión. Para visualizar un mejor detalle se puede referir a la figura 2.3.

Las clases que conforman este paquete son el soporte principal para la implementación de una arquitectura independiente del DBMS a utilizar, cumpliendo de esta manera con uno de los objetivos planteados inicialmente.

2.3.3.2. Paquete de Lógica de Negocio

Como se aprecia, existe una clase Base de Datos que interactúa con las clases de lógica de negocio (como la clase Esquema), las clases de conexión y las clases de abstracción del DBMS. Esta clase es el punto de partida para realizar el análisis pues interactúa con todas las capas de la aplicación.

Casi la totalidad de clases presentes (ver figura 2.4) en el diagrama representa un objeto de base de datos a el cual puede ser comparado y sincronizado. Para llevar a cabo esto, se cuenta con una interfaz que define las cabeceras de los métodos de

comparación, sincronización y carga de estructura del objeto de base de datos. El propósito es que todas las clases implementen su propia lógica de cómo compararse con objetos de su misma categoría, por ejemplo el objeto Columna implementa una lógica diferente para compararse que la utilizada por el objeto Llave Foránea.

De manera análoga, cada objeto implementa la lógica de cómo sincronizarse hacia la otra base de datos, ya sea de manera directa o generando un script de sincronización. Los atributos de cada clase representan los campos que el usuario va a poder comparar tal y como se especificaron en los requerimientos y pueden ser consultados además en la especificación de los casos de uso.

2.3.3.3. Paquete de Conexión a Base de datos

Como se muestra en la figura 2.5, se cuenta con una clase para manejar la conexión hacia la base de datos que contiene los métodos para establecer una conexión y ejecutar sentencias DML y DDL. Con estos métodos la aplicación será capaz de obtener la estructura de la base de datos mediante consultas SQL al diccionario de datos de la base de datos.

Esta clase interactúa con las librerías de conexión a base de datos propias del lenguaje de programación escogido para establecer una conexión a base de datos. Además se cuenta con una clase para manejar los tipos de parámetros utilizados en una sentencia SQL.

Con la definición de este paquete se ha buscado resolver la conectividad entre la aplicación y la base de datos a sincronizar permitiendo ejecutar sentencias SQL para la obtención de la metadata y sentencias DDL para realizar la sincronización una vez ya armadas y preparadas por las clases del paquete de abstracción.

En resumen, con el modelo presentado se ha logrado cubrir con todas las funcionalidades requeridas y armar una base robusta para la etapa de diseño e implementación, dando más énfasis en las funcionalidades de comparación y sincronización.

A continuación se muestra el diagrama de clases de análisis descrito en los párrafos anteriores. La descripción de cada clase, sus atributos y operaciones está especificada en el Anexo C: Especificación de las clases de Análisis.

Figura 2.3 Diagrama de clases de análisis (Paquete de Abstracción del DBMS)

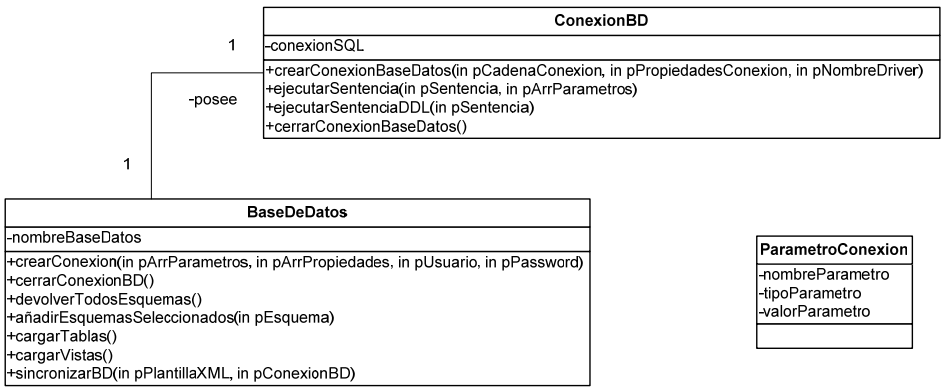


Figura 2.4 Diagrama de clases de análisis (Paquete de Conexión)

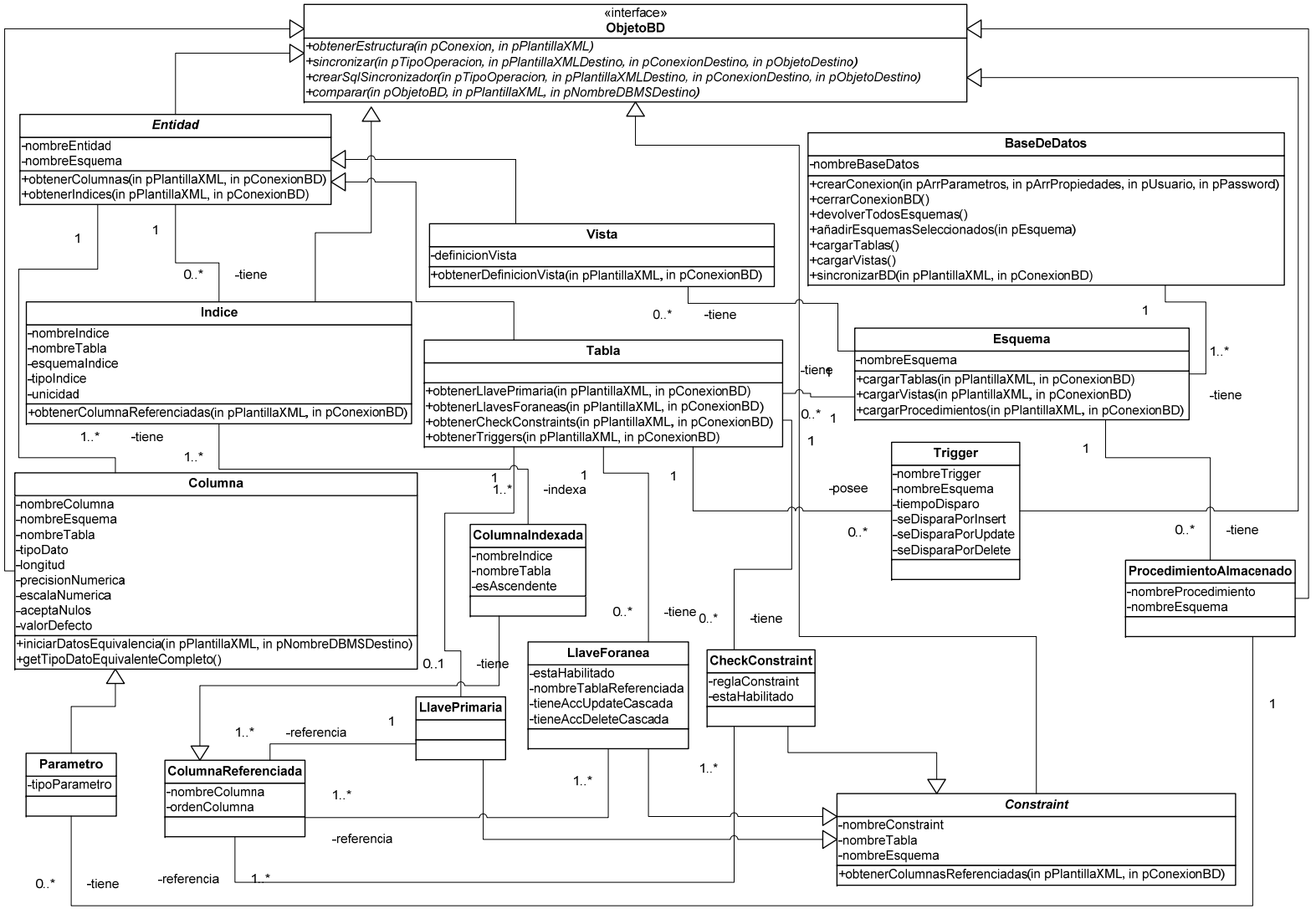


Figura 2.5 Diagrama de clases de análisis (Paquete de Lógica de Negocio)

3. Diseño

En este capítulo se discutirá los aspectos relacionados al diseño de la aplicación como la definición de la arquitectura, los prototipos de pantallas, diagramas de secuencia y definición de los archivos de configuración y las plantillas de la capa de Abstracción.

3.1. Arquitectura de la solución

En la presente sección se hará una descripción de dos arquitecturas que podrían aplicarse para el diseño de alto nivel de la aplicación. Posteriormente, se justificará la selección de una de ellas para el diseño de la solución.

3.1.1. Arquitectura dependiente del manejador de base de datos

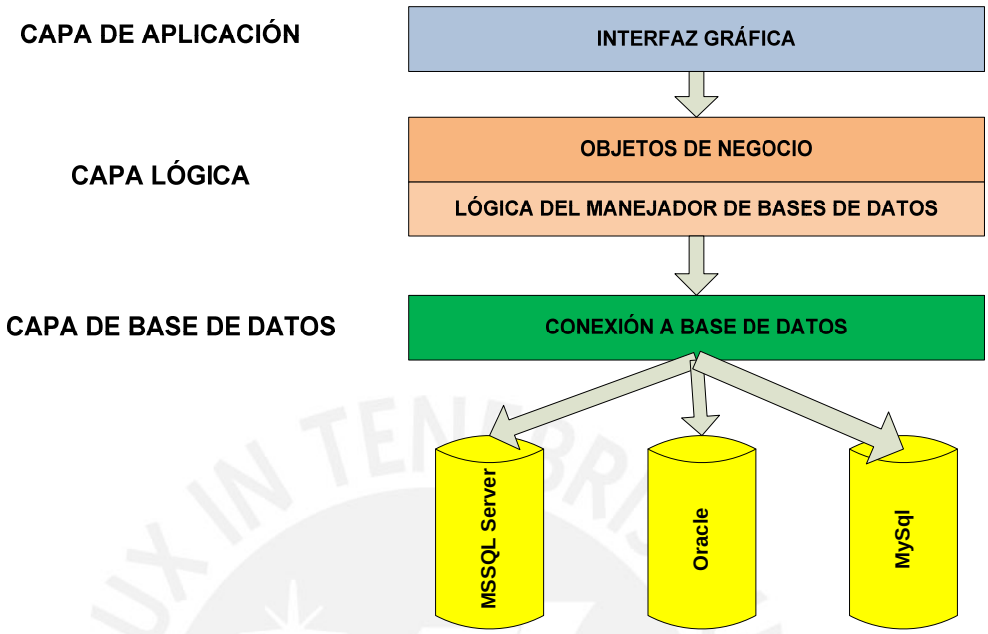


Figura 3.1 Arquitectura dependiente del manejador de base de datos

Como se aprecia en la figura 3.1, esta arquitectura propone 3 capas las cuales pasarán a ser descritas a continuación:

a) **Capa de aplicación.** Esta capa contiene las funcionalidades para que el usuario interactúe con la aplicación mediante la interfaz gráfica de la solución. Dichas funcionalidades están relacionadas a mostrar al usuario la estructura de la base de datos en un árbol jerárquico, el cuadro de resultados de la comparación de objetos resaltando los cambios y la visualización de los scripts de sincronización.

   Esta capa va a ser uso de las funcionalidades de la capa lógica para mostrar los resultados de la comparación y sincronización realizados por los objetos de negocio.

b) **Capa lógica.** Esta capa contiene los objetos de negocio definidos en el diagrama de clases de análisis, los cuales contienen las funcionalidades de cargado de estructura, comparación y sincronización de los objetos de bases de datos que representan.

Como se analizó en el Marco Conceptual del Problema (acápite 2 del capítulo 1), los manejadores de bases de datos a los cuales la aplicación podrá conectarse, tienen diferentes arquitecturas lo que involucra que la implementación del diccionario de datos de cada manejador difiera entre ellos. Esta arquitectura propone que los objetos de negocio implementen una lógica diferente por cada manejador que se desee probar, debido a las diferencias antes mencionadas.

Para ilustrar mejor la situación anterior, se puede proponer el siguiente ejemplo: para mostrar la estructura de la base de datos, es necesario saber los nombres de las tablas que conforman dicha base de datos. Siguiendo la arquitectura propuesta, el pseudocódigo para realizar esta función dentro de la clase Tabla sería:

Clase Tabla
….
Método obtenerColumnas(tipoBaseDatos, esquemaBaseDatos){
En caso de que tipoBaseDatos sea:

1. SQL Server:
   sentenciaSQL = “SELECT NAME FROM SYSBOJECTS WHERE XTYPE = ‘U’ AND USER_NAME(UID) = esquemaBaseDatos ORDER BY NAME”
2. Oracle:
   sentenciaSQL = “SELECT TABLE_NAME FROM DBA_TABLES WHERE OWNER = esquemaBaseDatos ORDER BY TABLE_NAME”
3. MySql:
   sentenciaSQL = “SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = esquemaBaseDatos ORDER BY TABLE_NAME”
   Llamar EjecutarSentenciaSQL(sentenciaSQL)
   }

c) **Capa de Base de Datos.** En esta capa se encuentras las funcionalidades relacionadas a establecer la conexión a la base de datos, así como los métodos para ejecutar sentencias DML (para realizar la carga de estructura de la base de datos con consultas al diccionario de datos) y sentencias DDL (para realizar la sincronización).

Esta capa provee métodos para realizar la ejecución de sentencias SQL independientes del manejador de base de datos que se esté utilizando, solo siendo dependiente de los parámetros necesarios para establecer la conexión. Para llevar esto a cabo se van a necesitar de librerías que provean una interfaz o API para lograr la comunicación con las bases de datos.

La principal desventaja de esta arquitectura es la poca escalabilidad de la aplicación para poder incorporar otras bases de datos, pues va a ser necesario realizar modificaciones a todas clases dentro de la capa lógica debido a su dependencia con el manejador de base de datos.

Como consecuencia, esta desventaja va a ocasionar que cada vez que se necesite agregar lógica para un manejador de base de datos distinto, se va a tener que modificar el código que a la larga puede resultar poco mantenible.

3.1.2. Arquitectura independiente del manejador de base de datos

La principal diferencia con la arquitectura anterior está en la capa lógica presenta una separación más definida entre los objetos de negocio y la lógica del manejador de bases de datos, permitiendo a la aplicación trabajar con distintos manejadores de bases de datos aparte de los ya mencionados y aumentar la escalabilidad de la aplicación. Dicha arquitectura se muestra en la figura 3.2.

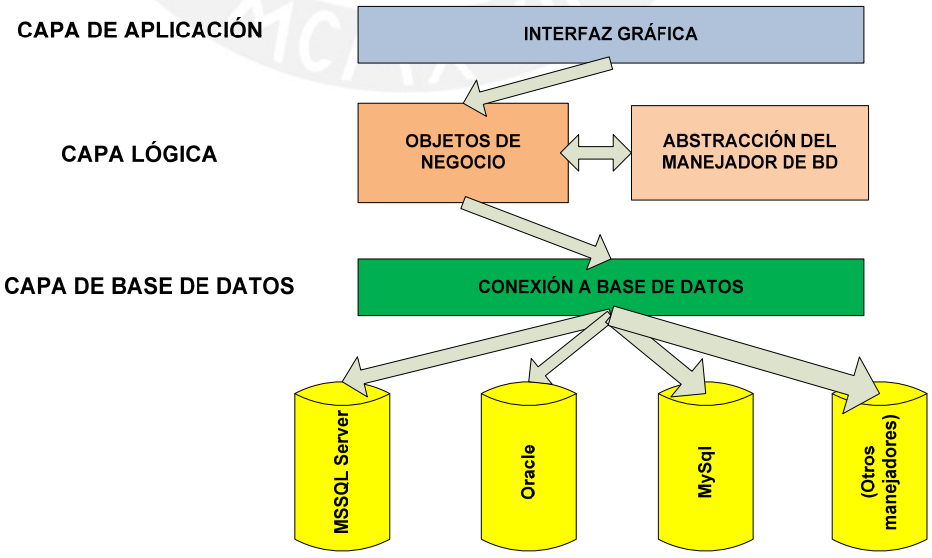


Figura 3.2 Arquitectura independiente del manejador de base de datos

El componente de abstracción del manejador de base de datos se encargará de proveer una interfaz para comunicarse con cualquier manejador de base de datos.

Este componente proveerá los métodos al componente de objetos de negocio que permitan armar las sentencias SQL para obtener la metadata de los diferentes objetos de base de datos a comparar así como un formato para poder ejecutar la sentencia DDL para realizar la sincronización.

De manera similar, para obtener la información de la estructura de la base de datos el componente de abstracción del manejador de base de datos contará con una plantilla que servirá como formato para obtener la información necesaria del diccionario de datos por cada manejador de base de datos.

A partir de cada formato los objetos de negocio, de acuerdo al manejador de base de datos, obtendrán el nombre de columnas y tablas del diccionario de datos necesarios y construirán la sentencia SQL adecuada.

De manera análoga, la plantilla permitirá armar la sentencia de sincronización de acuerdo al manejador de base de datos.

Con este diseño, la desventaja de la arquitectura anterior queda suprimida dado a que permite la conexión a otros manejadores de bases de datos sin necesidad de modificar la implementación de los objetos de negocio. Pero, la desventaja de esta arquitectura es que necesariamente se requiere que se arme una nueva plantilla por cada manejador que se desee agregar.

3.1.3. Arquitectura seleccionada

Para el desarrollo de la solución se hará uso de la arquitectura independiente del manejador de base de datos, debido a su diseño permite que la solución tenga más escalabilidad para soportar diferentes manejadores de datos. Se hará uso de plantillas para que, de acuerdo al manejador de base de datos, se pueda consultar al diccionario de datos de manera transparente y de igual manera realizar la sincronización.

A continuación se muestra el diagrama de componentes correspondientes a la arquitectura seleccionada en la figura 3.3.

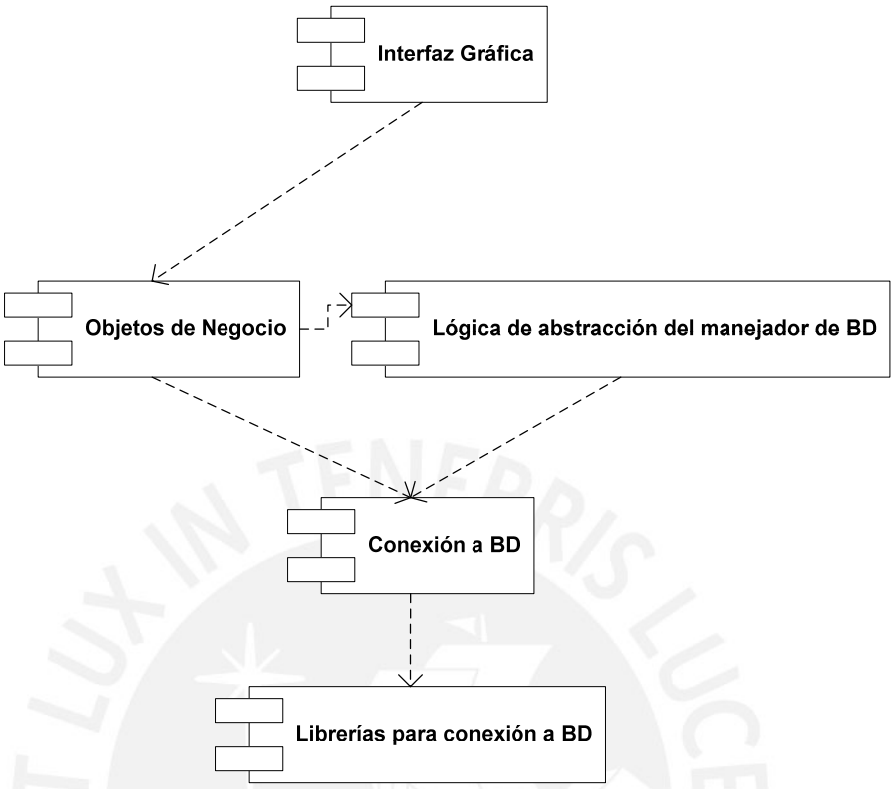


Figura 3.3 Diagrama de componentes de la arquitectura

En la figura 3.3, se muestran cómo se van a agrupar los objetos de la solución a lo largo de 5 paquetes, que conforman los componentes de la arquitectura seleccionada. Las funcionalidades abarcadas por cada paquete corresponden de manera paralela a las capas de la arquitectura.

El paquete de Interfaz Gráfica contiene las funcionalidades descritas en la capa de aplicación. El paquete de Objetos de Negocio contiene los objetos genéricos que representan los objetos de base de datos e invocan las funcionalidades del paquete de Lógica de abstracción del manejador de base de datos para construir las sentencias SQL para consultar al diccionario de datos y sincronizar dichos objetos. Finalmente el paquete Conexión a base de datos hace uso de librerías de conexión a diferentes bases de datos para establecer una conexión y ejecutar sentencias SQL.

3.2. Diseño de la interfaz gráfica

Para el diseño de la interfaz gráfica, se ha considerado poner énfasis en la simplicidad y amigabilidad de la aplicación de manera que el usuario no tenga dificultades en interactuar con la herramienta.

Se ha dispuesto el uso de controles GUI sencillos como cajas de texto, botones y tablas para desarrollar la capa de interfaz gráfica.

A continuación se muestran algunos prototipos de pantalla de la aplicación y una breve explicación de cada uno, resaltando las características antes mencionadas.

3.2.1. Pantallas de conexión a base de datos

Figura 3.4 Pantalla de conexión hacia una base de datos MSSQL Server 2005

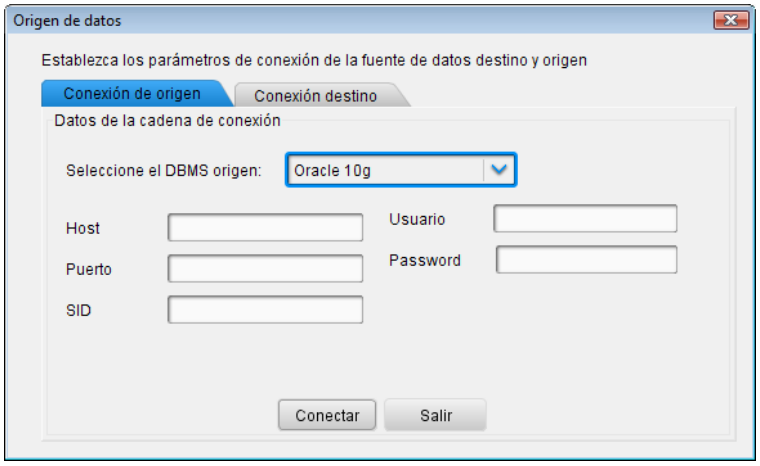


Figura 3.5 Pantalla de conexión hacia una base de datos Oracle 10g

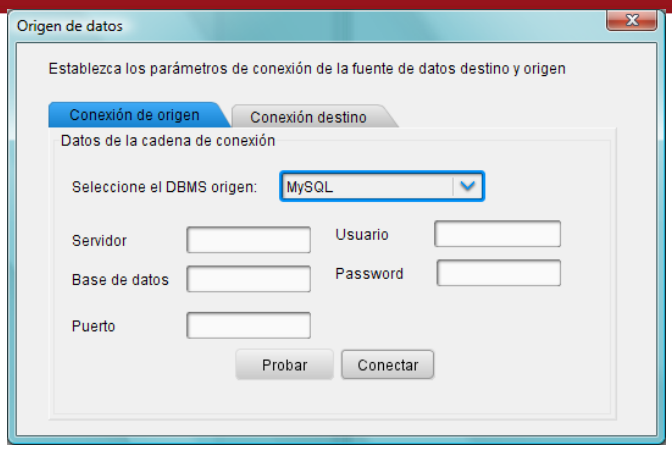


Figura 3.6 Pantalla de conexión hacia una base de datos MySql

El propósito de las pantallas mostradas en las figuras 3.4, 3.5 y 3.6 es permitir al usuario ingresar los parámetros de conexión a base de datos dependiendo del manejador. Como se aprecia el usuario puede seleccionar de una lista el manejador de base de datos y la aplicación muestra los campos correspondientes a los parámetros necesarios para establecer una conexión con dicho manejador.

3.2.2. Pantalla Principal. Estructura de base de datos cargada.

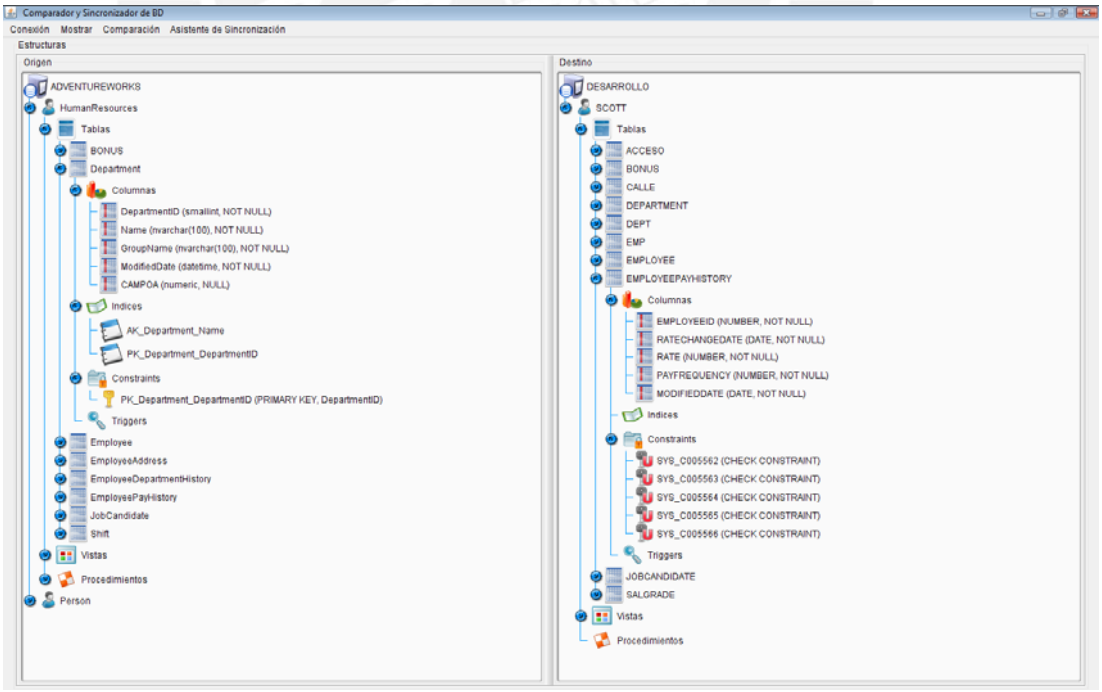


Figura 3.7 Pantalla principal

Como se aprecia en la figura 3.7, al momento de cargar la estructura de la base de datos se mostrará en la raíz el nombre de la base de datos cargada con 3 nodos

principales: Tablas, Vistas y Procedimientos Almacenados. Estos nodos se presentarán por cada esquema de la base de datos que se haya seleccionado al momento de realizar la conexión.

Como se observa, en cada nodo de la rama de Tablas, se muestran 4 sub-nodos correspondientes a la información de Columnas, Índices y Constraints (que incluye la llave primaria, llaves foráneas y check constraints si los hubiera).

Por cada nodo de la rama de Vistas se muestran 2 sub-nodos correspondientes a las Columnas e Índices de las vistas.

Finalmente, el nodo de procedimientos almacenados contiene por cada rama, un procedimiento almacenado mostrándose el nombre en el árbol de la estructura de la base de datos.

3.2.3. Pantalla de Comparación de Tablas

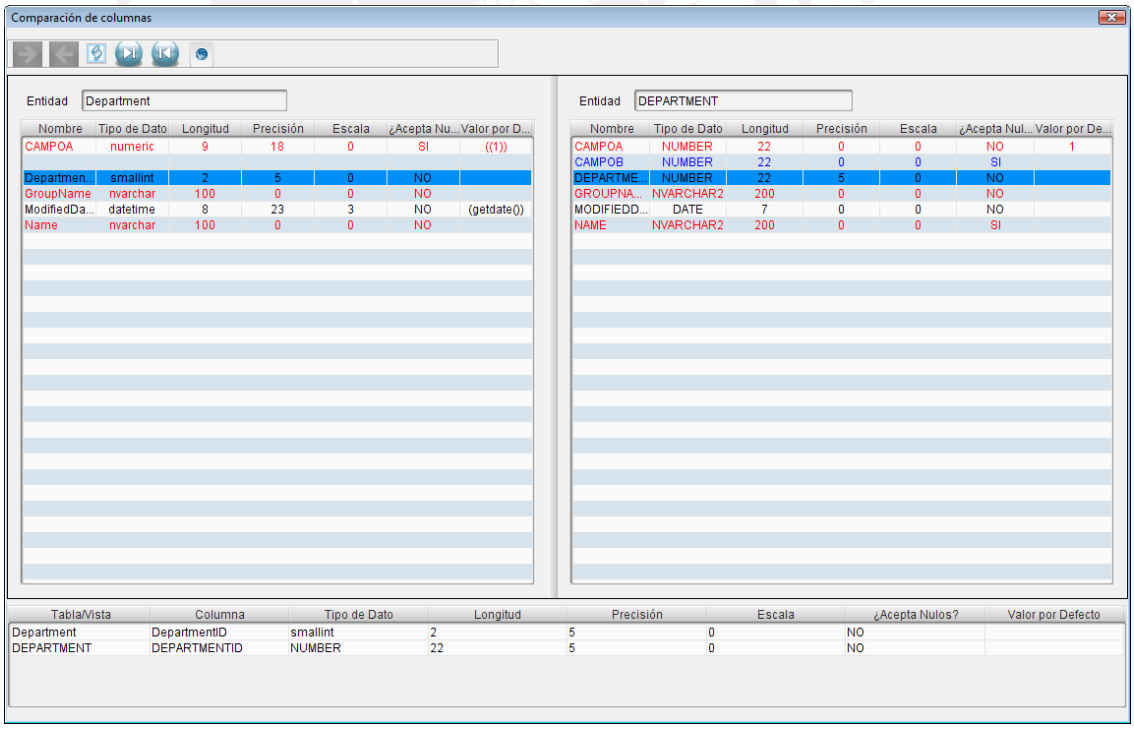


Figura 3.8 Pantalla de comparación de tablas por columnas

Esta pantalla muestra el resultado de la comparación de dos tablas, en donde se muestra la información de las columnas de cada una de ellas. En este ejemplo, se muestra el nombre, tipo de dato, longitud, precisión, escala, si acepta nulos y el

valor por defecto de cada columna. Para ayudar al usuario a resaltar los cambios en la estructura de base de datos, se utilizan 3 colores de acuerdo a la siguiente leyenda:

- **Negro:** Si no existen diferencias entre las características a comparar.
- **Azul:** Si una de las características no existe en la otra entidad.
- **Rojo:** Si la característica existe, pero difiere en una de sus propiedades (por ejemplo, en la pantalla mostrada, la longitud del tipo de datos de la columna *ContactName* presente en ambas tablas es diferente).

Este mismo esquema se utilizará para realizar la comparación de índices, llaves primarias y foráneas, check constraints, triggers y los parámetros de los procedimientos almacenados.

En la parte superior, se aprecian los botones de sincronización rápida, generación de script, sincronización de más de una característica y generación de reporte de comparación.

Los botones de sincronización rápida permiten al usuario sincronizar de un lado a otro la característica seleccionada de manera rápida, reflejándose el cambio de manera inmediata. Si está activada la opción de generación de script, se mostrará en una pantalla la sentencia SQL de sincronización de acuerdo al manejador de base de datos correspondiente.

Si el usuario desea podrá guardar dicha sentencia en un archivo de texto. Los dos siguientes botones corresponden a una sincronización múltiple, en donde se muestra una pantalla donde el usuario puede seleccionar más de una característica a sincronizar o generar las sentencias SQL de sincronización. Finalmente, el último botón permite generar un reporte de comparación de acuerdo a los resultados mostrados.

En la parte inferior de la pantalla se muestra un cuadro resumen de la comparación para que el usuario pueda apreciar las diferencias de manera más fácil.

Para realizar la sincronización de un objeto de la base de datos se pueden utilizar 2 maneras, que a continuación se describen:

a. **Sincronización directa.** En esta forma, el usuario puede escoger que elementos desea sincronizar sin haber realizado algún tipo de comparación previa. Este proceso se realizará a través de un asistente personalizado. A continuación en las figuras 3.9, 3.10, 3.11, 3.12 y 3.13 se muestran los pasos a seguir para llevar a cabo esta operación.

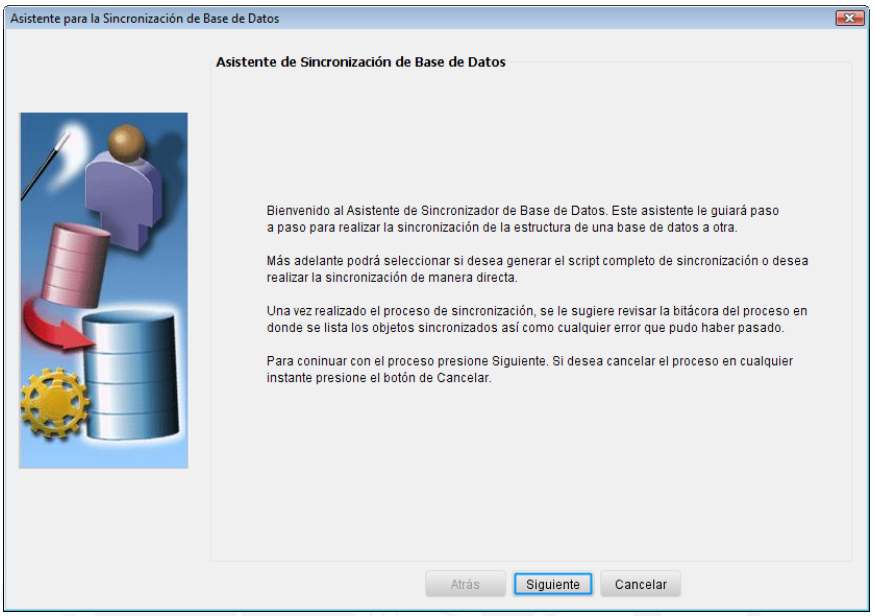


Figura 3.9 Pantalla de bienvenida al asistente

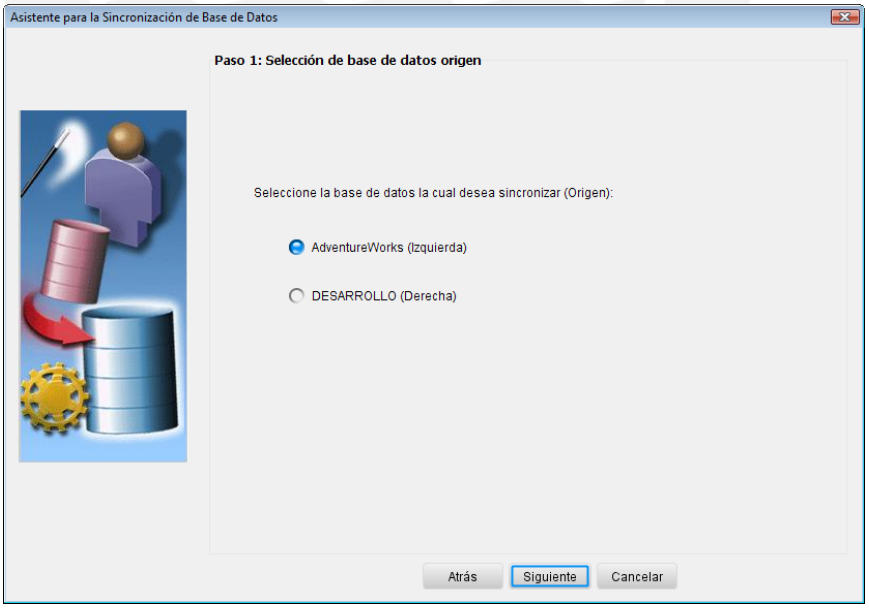


Figura 3.10 Pantalla de Selección de base de datos origen

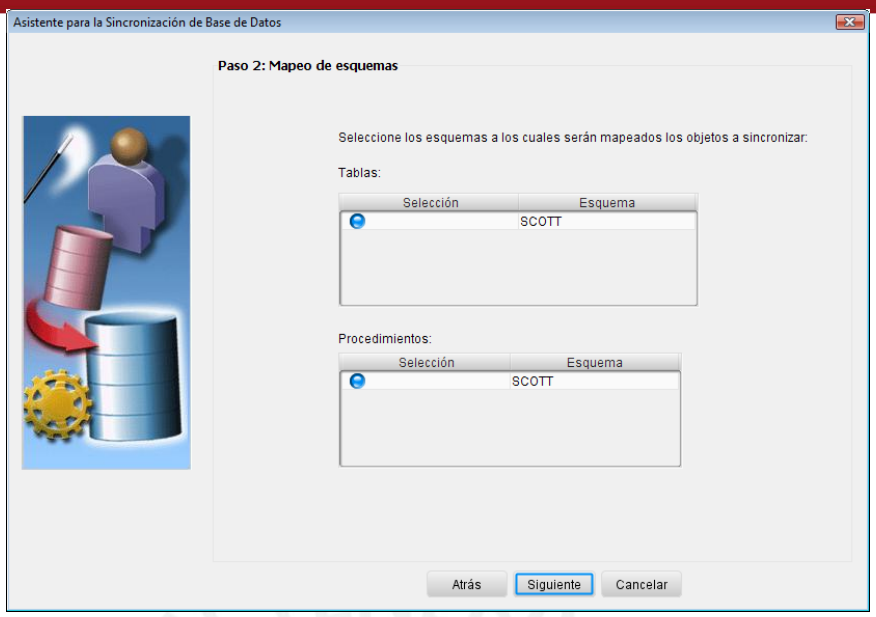


Figura 3.11 Pantalla de Mapeo de Esquemas

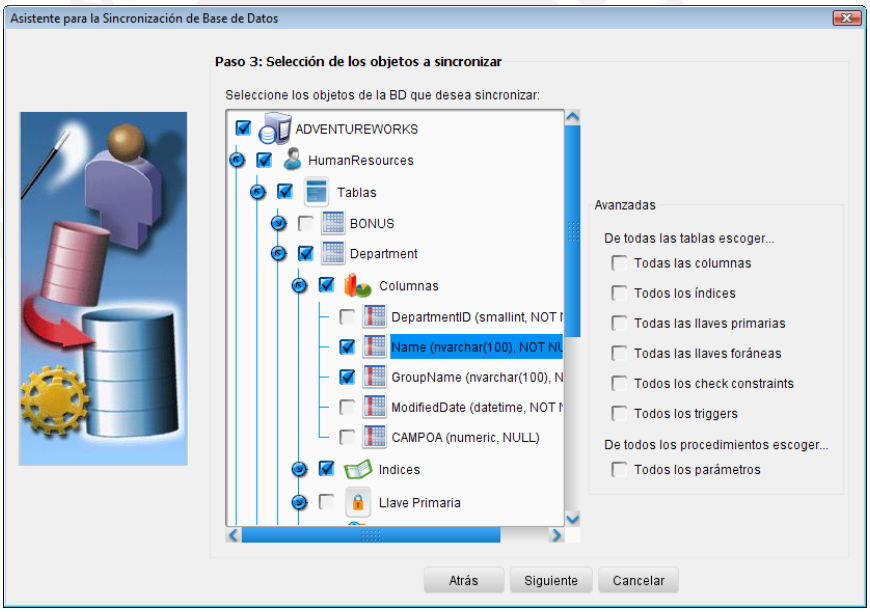


Figura 3.12 Pantalla de Selección de Objetos de BD a sincronizar

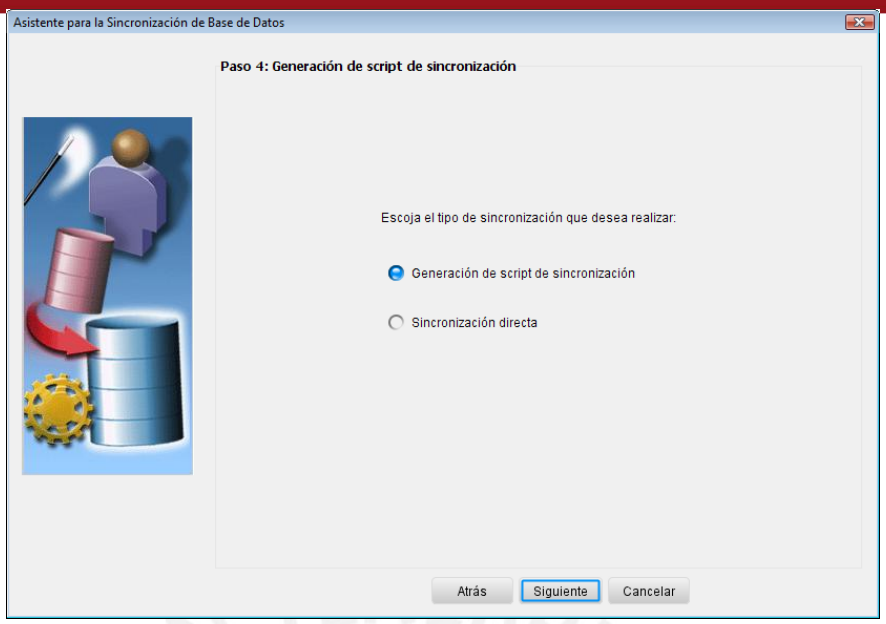


Figura 3.13 Pantalla de Tipo de sincronización a realizar

b. **Sincronización indirecta.** Para este tipo de sincronización, el usuario debe haber realizado una comparación previa de lo que desea sincronizar (como se muestra en la figura 3.8) y de acuerdo a su criterio, seleccionará aquellos elementos que desee sincronizar. Este tipo de sincronización solo se aplica sobre un mismo tipo de objeto (columna, llave primaria, etc.) a diferencia del caso anterior en donde se pueden mezclar distintos objetos.

Cabe mencionar que en ambos casos, está la posibilidad de realizar la sincronización de manera inmediata o solo generar el script de sincronización.

3.2.4. Diagramas de secuencia

En esta sección, se presentarán 3 diagramas de secuencia correspondientes a la carga de metadata de un objeto de base de datos, la comparación y sincronización de un objeto. El propósito es mostrar la interacción entre las distintas capas de la aplicación y el usuario. Es necesario mencionar que la secuencia de pasos es parecida para el caso de distintos objetos de la base de datos, por lo que se ha escogido un objeto para demostrar la interacción de las capas. Junto con cada diagrama se adjunta una breve explicación.

3.2.4.1. Diagrama de secuencia de la operación Cargar Tablas

Como se aprecia en la figura 3.14, la operación de Cargar Tablas empieza cuando el usuario selecciona la opción de Mostrar la estructura de la base de datos.

Por cada esquema que se tenga en la base de datos, se carga la información de las tablas. Posteriormente, por cada tabla es necesario cargar la información de las columnas, índices y constraints (llave primaria, llaves foráneas y check constraints).

Estos objetos pertenecientes a la capa de Lógica de Negocio interactúan con la capa de Abstracción mediante llamadas a métodos de la Plantilla para obtener la sentencia SQL necesaria para consultar el diccionario de datos de la base de datos. Estos métodos, de acuerdo al tipo de DBMS, devuelven al objeto de negocio la sentencia SQL de la consulta, para luego ser enviada al objeto de Conexión con el propósito de ejecutarse y devolver un conjunto de resultados (conocido como un ResultSet).

Una vez ejecutada la sentencia SQL, se crean los objetos de lógica de negocio con la información recopilada. En este caso se aprecia que el objeto Tabla va creando sus objetos Columnas, Llave Primaria, Llaves Foráneas y Check Constraints de acuerdo al conjunto de resultados que le devuelve la ejecución de la sentencia SQL.

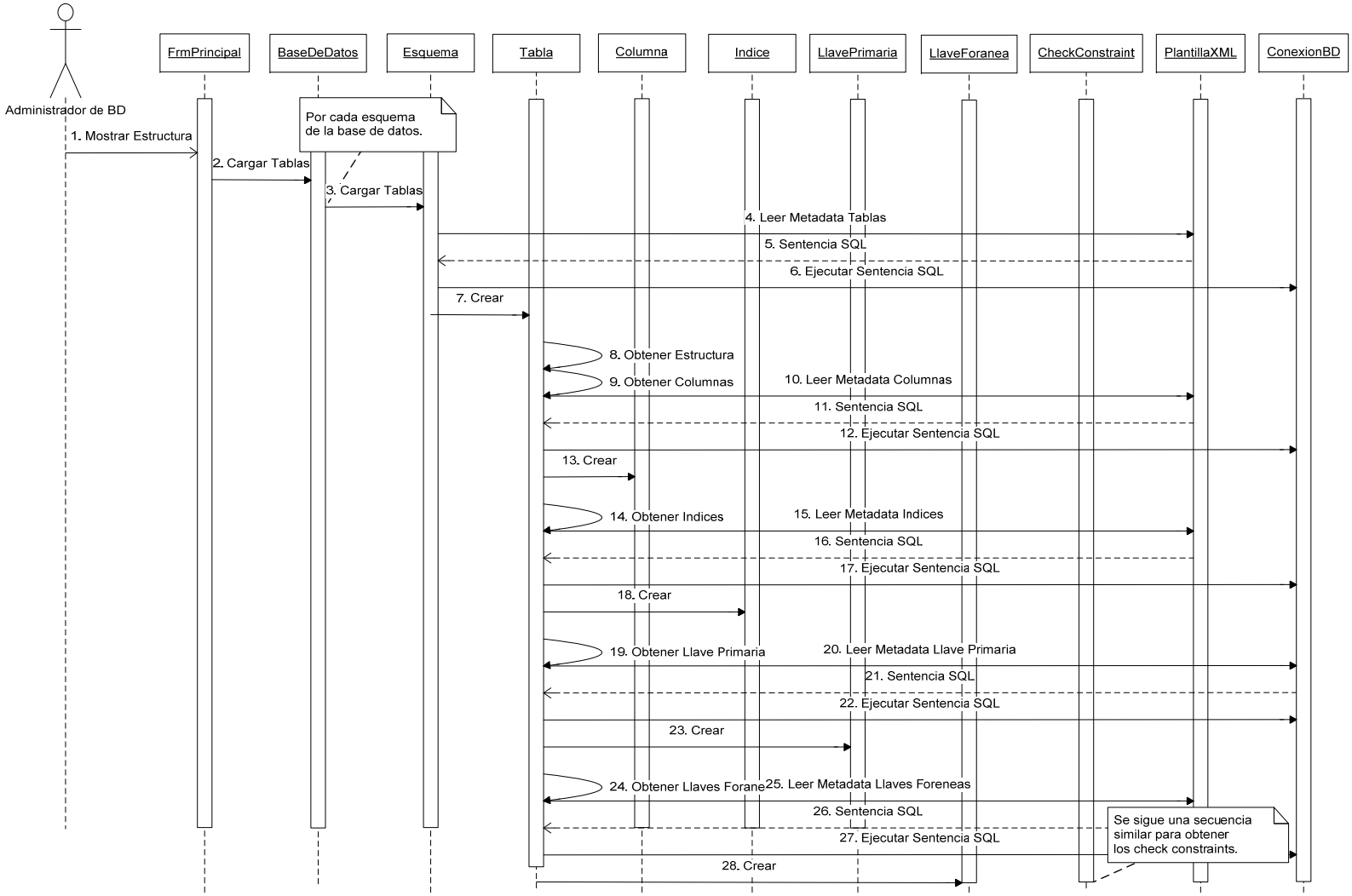


Figura 3.14 Diagrama de la operación de Cargar Tablas

3.2.4.2. Diagrama de secuencia de la operación Comparar Columnas

En este diagrama, la operación inicia cuando el usuario selecciona dos tablas y la aplicación realiza la comparación en base a las columnas que poseen ambas tablas.

Desde la ventana de comparación, se llama al método de Comparar perteneciente al objeto Columna cuyo resultado va a permitir a la ventana mostrar un pintado visual diferente en caso de existir o no diferencias entre las columnas a comparar.

Para este caso particular, el objeto Columna hace uso de un par de métodos de la Plantilla para obtener el tipo de dato equivalente en caso de estar comparando columnas de diferentes DBMS. Así mismo, obtiene información sobre los parámetros que necesita comparar pues para ciertos tipos de datos algunos datos son irrelevantes y no necesitan ser comparados. Esta secuencia de pasos demuestra la interacción de los objetos de la capa de Lógica de Negocio con la capa de Abstracción.

Finalmente, una vez realizado las comparaciones de los atributos de las columnas, se devuelve un resultado a la ventana para que realice el pintado correspondiente y muestre el resultado de la comparación. La figura 3.15 demuestra la secuencia de pasos correspondientes.

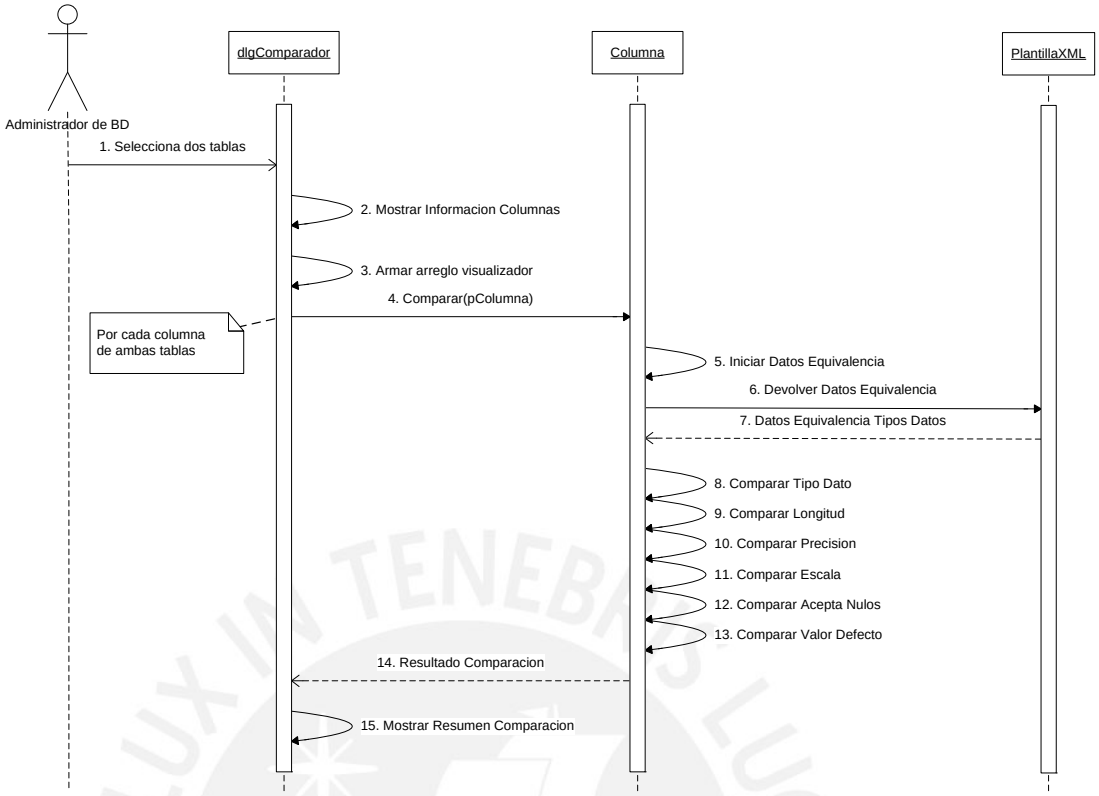


Figura 3.15 Diagrama de la operación de Comparar columnas

3.2.4.3. Diagrama de secuencia de la operación Sincronizar Columnas

Como se aprecia en la figura 3.16, primero el usuario selecciona la(s) columna(s) que desea sincronizar. Desde la ventana se hace la llamada al método Sincronizar del objeto Columna (el cual fue seleccionado por el usuario) para realizar en sí el proceso de sincronización.

La interacción entre la capa de negocio y la capa de abstracción se da cuando el objeto Columna hace uso del método de lectura del formato de la sentencia de sincronización del objeto Plantilla que, de acuerdo al tipo de DBMS destino, devuelve el formato de la sentencia DDL para realizar la sincronización.

Una vez devuelto el formato, el objeto de negocio se encarga de reemplazar los parámetros de la cadena con sus atributos para terminar de armar la sentencia DDL de sincronización. Posteriormente esta sentencia es enviada al objeto Conexión para su ejecución.

Finalmente, en la ventana se refrescan las estructuras para mostrar al usuario la información actualizada.

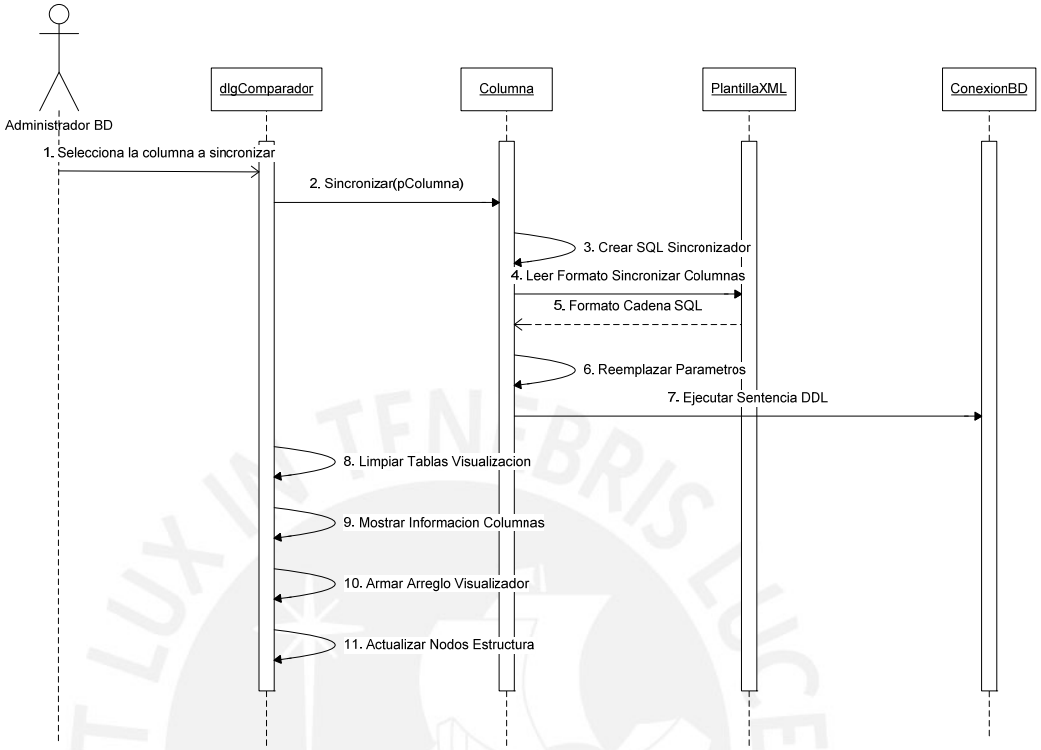


Figura 3.16 Diagrama de la operación de Sincronizar Columnas

3.2.5. Formato del Archivo de Configuración

A continuación se explican los parámetros definidos en el archivo de configuración, que forman parte de la implementación de la capa de Abstracción del DBMS.

El archivo de configuración estará formato en varias secciones cada una de ellas correspondientes a un DBMS en particular. Dentro de cada sección se van a definir los siguientes parámetros, los cuales son necesarios para establecer una conexión con dicho DBMS.

Nombre del DBMS. Este parámetro es la cabecera de la sección. Debe ir dentro de corchetes para identificar el inicio de la sección. Por ejemplo [MSSQL].

ArchivoPlantilla. Indica la ruta en donde se ubica la plantilla XML para el DBMS.

NombreDriver. Indica el nombre del driver necesario para establecer una conexión hacia dicho DBMS.

ListaParametros. Contiene una lista de nombre de parámetros usados para establecer una conexión. Los nombres deben ir separados por comas.

Obligatorios. Contiene una lista de los parámetros obligatorios para la conexión. Para esta lista se utilizará la siguiente nomenclatura: PARAMETRO1, PARAMETRO2 y así sucesivamente. Cada uno corresponde al nombre del parámetro definido en el campo ListaParametros en el mismo orden en que fueron definidos.

FormatoCadena. Este campo contiene el formato de la cadena de conexión. Se utilizará la nomenclatura anterior para definir como se distribuyen los parámetros dentro de la cadena de conexión.

NombrePropiedades. Este campo contiene el nombre de los parámetros que no son obligatorios. A diferencia de los parámetros obligatorios que deben ir definidos en la cadena de conexión, este tipo de parámetros se agregan a la cadena como propiedades adicionales.

ParametroNombreBD. Este campo representa que parámetro almacena el nombre de la base de datos.

RutaLibreriaSQL. Este campo representa la ruta en donde almacena la librería que va a permitir a la aplicación conectarse al DBMS definido en la sección. Esto permitirá a la aplicación que, en tiempo de ejecución, cargue la librería de la ruta indicada, registre el driver correspondiente, lea el formato de la cadena de conexión y arme la cadena con los parámetros ingresados por el usuario.

A continuación se muestra un ejemplo de una sección del archivo de configuración.

```
[MSSQL2000]
ArchivoPlantilla=C:\Users\Giancarlo\SincronizadorBD\plantillaMSSQL2000.xml
NombreDriver=com.microsoft.sqlserver.jdbc.SQLServerDriver
ListaParametros=Servidor,Puerto,Base de Datos,Instancia
Obligatorios=PARAMETRO1,PARAMETRO2,PARAMETRO3
```

FormatoCadena=jdbc:sqlserver://PARAMETRO1:PARAMETRO2;databaseName IGUAL PARAMETRO3
NombrePropiedades=instanceName
ParametroNombreBD=PARAMETRO3
RutaLibreriaSQL=C:\Users\Giancarlo\SincronizadorBD\dist\lib\sqljdbc.jar

3.2.6. Formato de las Plantillas

A continuación se describen las secciones de la plantilla XML utilizada por las clases correspondientes a la capa de abstracción del DBMS. La clase que se encarga de implementar las funcionalidades de lectura es la clase **PlantillaXML**.

3.2.6.1. Sección de parámetros generales

Esta sección contiene parámetros relacionados al armado de sentencias SQL para un DBMS y parámetros de configuración. Los parámetros contemplados son: si el DBMS utiliza esquemas, el separador de alias en las sentencias SELECT, si se utiliza comillas simples para los alias, el símbolo del OUTER JOIN, el operador de desigualdad, si se usa ordenamiento de columnas en la construcción de índices, el separador de sentencias SQL, el separador de los eventos en la construcción de un trigger, los DBMS compatibles para sincronizar triggers y procedimientos almacenados y si en la definición de parámetros de los procedimientos se deben ignorar los atributos como precisión o escala numérica. El formato es el siguiente:

```
<utiliza_esquemas>…</utiliza_esquemas>
<separador_alias>…</separador_alias>
<usa_comilla_simple>…</usa_comilla_simple>
<sintax_outer_join>…</sintax_outer_join>
<operador_diferente>…</operador_diferente>
<usa_ordenamiento_indices>…</usa_ordenamiento_indices>
<separador_sentencia>…</separador_sentencia>
<separador_evento_trigger>…</separador_evento_trigger>
<sincroniza_trigger_dbms>…</sincroniza_trigger_dbms>
<sincroniza_procedimiento_dbms>…</sincroniza_procedimiento_dbms>
<ignorar_atributos_parametros>…</ignorar_atributos_parametros>
```

3.2.6.2. Sección de lectura de metadata

Para la lectura de metadata se realizarán consultas al diccionario de datos del DBMS mediante sentencias SELECT. Con esta premisa, se ha armado la siguiente estructura para construir la sentencia SQL necesaria.

```
<seccion_select>
<columna_1 existeMTD='SI'>…</columna_1>
<columna_2 existeMTD='SI'>…</columna_2>
…
</seccion_select>
<seccion_from>TABLA1, TABLA2, …</seccion_from>
<seccion_where>
<condicion conParametro='SI' outerJoin='NO' tipoDato='CADENA'
nombrePropiedad='getNombreEsquema' operadorIgual='SI'>
        <valor_izquierdo>…</valor_izquierdo>
        <valor_derecho></valor_derecho>
        </condicion>
<condicion conParametro='NO' outerJoin='NO' tipoDato='' nombrePropiedad='' operadorIgual='SI'>
                <valor_izquierdo>…</valor_izquierdo>
                <valor_derecho>…</valor_derecho>
        </condicion>
        …
</seccion_where>
<seccion_orderBy>
<columna tipoOrdenamiento = 'ASC'>…</columna>
<columna tipoOrdenamiento = 'DESC'>…</columna>
…
</seccion_orderBy>
```

En la parte de la sentencia correspondiente al SELECT, se tienen tantas etiquetas como columnas se necesiten extraer para un objeto en específico. Por cada etiqueta, se ha definido el atributo existeMTD que indica si existe la columna de donde se pueda extraer la información necesaria. En caso de que no se pueda, esa columna se rellena con el valor NULL.

En la sección del FROM, se listan las tablas de donde se va a extraer la información. La forma de listar es similar a la utilizada cuando se define el FROM de una sentencia SQL.

En la sección del WHERE, se tienen tantas etiquetas como condiciones se necesiten. Se pueden definir dos tipos de condiciones dependiendo del atributo conParametro: una condición en donde un valor va a ser igualado a una variable y una condición en donde se igualen dos valores provenientes de las columnas de las tablas definidas.

En el primer escenario, es necesario llenar los atributos tipoDato y nombrePropiedad. El atributo tipoDato indica el tipo de datos de la variable a enlazar pudiendo ser CADENA, ENTERO, DECIMAL o FECHA. El atributo nombrePropiedad indica el nombre de la propiedad del objeto de donde se va a obtener el valor a enlazar con dicha variable. Para llevar a cabo esta funcionalidad, se recurre a la reflexión de clases, concepto que está relacionado a la obtención de información sobre un objeto en particular.

El atributo outerJoin indica si en la condición se va a utilizar el operador de OUTER JOIN mientras que el atributo operadorIgual indica si es que se va a evaluar una igualdad o desigualdad dentro de la condición.

Por cada condición, se definen dos elementos, el valor izquierdo y el valor derecho que corresponden a los valores a evaluar en la condición. En caso de que la condición utilice parámetros, se obvia el contenido del valor derecho.

Finalmente, en la sección de ORDER BY, se definen tantas etiquetas como columnas se necesiten se deseen que intervengan en algún tipo de ordenamiento. Por cada columna se define el atributo tipoOrdenamiento que indica si el ordenamiento es Ascendente o Descendente.

Para identificar las secciones por cada objeto de base de datos, se utilizará la siguiente estructura.

Para los esquemas:

```
<metadata_esquema existeTablaMTD = 'SI'>
 [Sección SELECT, FROM, WHERE, ORDER BY]
</metadata_esquema>
```

Para las tablas:

```
<metadata_tabla>
        <sección_tablas existeTablaMTD = 'SI'>
 [Sección SELECT, FROM, WHERE, ORDER BY]
        </sección_tablas>
        <sección_columnas existeTablaMTD = 'SI'>
         [Sección SELECT, FROM, WHERE, ORDER BY]
        </sección_columnas>
        <sección_indices existeTablaMTD = 'SI'>
                <sección_cabecera>
                 [Sección SELECT, FROM, WHERE, ORDER BY]
```

```
                    </sección_cabecera>
                    <sección_columnas>
                     [Sección SELECT, FROM, WHERE, ORDER BY]
                    </sección_columnas>
          </sección_indices>
          <sección_llave_primaria existeTablaMTD = 'SI'>
                    <sección_cabecera>
                     [Sección SELECT, FROM, WHERE, ORDER BY]
                    </sección_cabecera>
                    <sección_columnas>
                     [Sección SELECT, FROM, WHERE, ORDER BY]
                    </sección_columnas>
          </sección_llave_primaria>
          <sección_llaves_foraneas existeTablaMTD = 'SI'>
                    <sección_cabecera>
                     [Sección SELECT, FROM, WHERE, ORDER BY]
                    </sección_cabecera>
                    <sección_columnas>
                     [Sección SELECT, FROM, WHERE, ORDER BY]
                    </sección_columnas>
          </sección_llaves_foraneas>
          <sección_check_constraints existeTablaMTD = 'SI'>
                    <sección_cabecera>
                     [Sección SELECT, FROM, WHERE, ORDER BY]
                    </sección_cabecera>
                    <sección_columnas>
                     [Sección SELECT, FROM, WHERE, ORDER BY]
                    </sección_columnas>
          </sección_check_constraints>
          <sección_triggers existeTablaMTD = 'SI'>
                     [Sección SELECT, FROM, WHERE, ORDER BY]
          </sección_triggers>
</metadata_tabla>
```

Para las vistas

```
<metadata_vista>
          <sección_definicion existeTablaMTD = 'SI'>
           [Sección SELECT, FROM, WHERE, ORDER BY]
          </sección_definicion>
          <sección_columnas existeTablaMTD = 'SI' usaMTDTablas='SI'>
           [Sección SELECT, FROM, WHERE, ORDER BY]
          </sección_columnas>
          <sección_indices existeTablaMTD = 'SI' usaMTDTablas='SI'>
                    <sección_cabecera>
                     [Sección SELECT, FROM, WHERE, ORDER BY]
                    </sección_cabecera>
                    <sección_columnas>
                     [Sección SELECT, FROM, WHERE, ORDER BY]
                    </sección_columnas>
          </sección_indices>
</metadata_vista>
```

Para los procedimientos almacenados

```
<metadata_procedimiento>
          <sección_cabecera existeTablaMTD = 'SI'>
 [Sección SELECT, FROM, WHERE, ORDER BY]
</sección_cabecera>
<sección_parametros existeTablaMTD = 'SI'>
 [Sección SELECT, FROM, WHERE, ORDER BY]
</sección_parametros>
</metadata_procedimiento>
```

Como se aprecia, por cada sección se ha definido el atributo **existeTablaMTD** que indica si efectivamente, existe alguna tabla en el diccionario de datos de donde se pueda extraer la información necesaria. En caso de no haber tabla alguna, no se podrá leer la información de dicho objeto lo cual impediría la comparación y sincronización.

Existe un atributo particular para las vistas (**usaMTDTablas**) el cual indica si la información de las vistas puede ser obtenida de las tablas del diccionario de datos correspondientes a las tablas. Este escenario ocurre con frecuencia en algunos DBMS en donde en la información sobre tablas y vistas se almacenan en una misma tabla de sistema.

3.2.6.3. Sección de catálogo de equivalencias

En esta sección se define el catálogo de tipos de datos entre diferentes DBMS, así como algunos parámetros sobre la comparación y sincronización para el caso de columnas.

La estructura es la siguiente:

```
<catalogo_equivalencia>
        <DBMS1>
<TD1 compararLongitud='NO' compararPrecision='NO' compararEscala='NO' valorPrecision=''
valorEscala=''>TDEQa</TD1>
<TD2 compararLongitud='SI' compararPrecision='NO' compararEscala='NO' valorPrecision=''
valorEscala=''>TDEQb</TD2>
…
        </DBMS1>
<DBMS2>
<TD1 compararLongitud='NO' compararPrecision='NO' compararEscala='NO' valorPrecision=''
valorEscala=''>TDEQc</TD1>
<TD2 compararLongitud='NO' compararPrecision='NO' compararEscala='NO' valorPrecision=''
valorEscala=''>TDEQd</TD2>
…
        </DBMS2>
        …
</catalogo_equivalencia>
```

Como se aprecia, se tiene que definir una equivalencia por cada tipo de dato del DBMS para todos los DBMS configurados (presentes en el archivo de configuración). Esto también incluye la definición de los tipos de datos para el mismo DBMS, pues existen parámetros que deben configurarse para la comparación y sincronización.

Para el caso de la comparación de los tipos de datos, se hace uso de los 3 primeros atributos que indican si para ese tipo de dato se va a comparar la longitud, la precisión y escala numérica. Esto es necesario debido a que para ciertos tipos de datos, estas propiedades son irrelevantes y solo contienen información sobre cómo se almacenan internamente en la base de datos (por ejemplo para los tipos de datos de Fechas).

Para el caso de la sincronización se utilizan los dos siguientes atributos que indican cual es la precisión y escala correspondientes que hacen equivalentes a los tipos de datos. En caso de que estos datos estén vacíos pero si se utilicen la precisión y escala numérica como parte de la comparación, se tomarán los valores correspondientes a la columna origen. Para ilustrar lo descrito anteriormente se tiene el siguiente ejemplo que muestra la equivalencia de un tipo de datos entre MSSQL Server y Oracle.

```
<catalogo_equivalencia>
        <Oracle>
        <BIGINT compararLongitud='NO' compararPrecision='SI' compararEscala='SI'
        valorPrecision='19' valorEscala='0'>NUMBER
        </BIGINT>
```

En este ejemplo, cuando se compare un tipo de dato BIGINT (que corresponde a MSSQL Server) contra un tipo de dato de Oracle, la primera comparación se realizará a nivel de tipo de dato de acuerdo a la equivalencia. Luego se procederá a comparar la precisión de ambos tipos de datos y luego la escala.

Al momento de realizar la sincronización, el tipo de dato BIGINT se transformará en NUMBER(19,0) para Oracle.

3.2.6.4. Sección de formato de sincronización

Para realizar la sincronización de un objeto es necesario conocer cuál es el formato de la sentencia DDL correspondiente. Por tal motivo, se ha definido la siguiente estructura que permitirá conocer dicho formato:

```
<OBJETOBD>
        <nuevo>
                <formato>[SENTENCIA SQL]</formato>
```

```
                <parametro origen='NO'>…</parametro>
                …
        </nuevo>
        <modificacion >
                <formato>[SENTENCIA SQL]</formato>
                <parametro origen='SI'>…</parametro>
…
        </modificacion>
</OBJETOBD>
```

Por cada objeto de la base de datos se definirá la estructura anterior, pudiendo presentarse dos escenarios: la adición de un objeto o la modificación. En la sección de formato se ingresará la sentencia DDL correspondiente tomando en cuenta la siguiente consideración: se incluirá un signo de interrogación (?) por cada variable necesaria en la sentencia. Los valores de dichas variables se obtendrán de la información contenida en las etiquetas de parámetros que contienen el nombre de las propiedades del objeto de negocio que devuelve el valor de la variable (similar al caso de la lectura de metadata de los objetos de la base de datos).

El siguiente ejemplo muestra lo explicado en el párrafo anterior:

```
<columnas>
        <nuevo cadenaDefault='DEFAULT'>
                <formato>ALTER TABLE ?.? ADD ? ? NULL</formato>
                <parametro origen='NO'>getNombreEsquema</parametro>
                <parametro origen='NO'>getNombreTabla</parametro>
                <parametro origen='SI'>getNombreColumna</parametro>
                <parametro origen='SI'>getTipoDatoEquivalenteCompleto</parametro>
        </nuevo>
        <modificacion cadenaDefault='SET DEFAULT'>
                <formato>ALTER TABLE ?.? ALTER COLUMN ? ? ?</formato>
                <parametro origen='NO'>getNombreEsquema</parametro>
                <parametro origen='NO'>getNombreTabla</parametro>
                <parametro origen='SI'>getNombreColumna</parametro>
                <parametro origen='SI'>getTipoDatoEquivalenteCompleto</parametro>
                <parametro origen='SI'>getCadenaNulo</parametro>
        </modificacion>
</columnas>
```

Como se aprecia, se tiene la sentencia DDL para añadir y modificar una columna cuyas variables serán obtenidas de las propiedades del objeto Columna definidas en las etiquetas de parámetros. Además se hace uso del atributo origen para indicar si la propiedad será invocada del objeto origen o del objeto destino. En este ejemplo en particular, el nombre del esquema y el nombre de la tabla serán obtenidas de la columna sobre la cual se van a hacer las modificaciones.

Adicionalmente, para la funcionalidad extra de sincronización de objetos de base de datos Oracle se ha usado el procedimiento **dbms_metadata.get_dll** el cual recibe 3 parámetros: tipo de objeto, nombre del objeto y el nombre del esquema.
Este procedimiento permite crear la sentencia DDL de cualquier objeto de una base de datos permitiendo así la creación de cualquier objeto a otra base de datos. La aplicación solo permite crear el script de sincronización.

Cabe señalar que esta funcionalidad solo permite crear nuevos objetos en una base de datos Oracle, mas no actualizar alguna propiedad de un objeto ya existente. A continuación se muestra un ejemplo de la llamada de este procedimiento:

select dbms_metadata.get_ddl('TABLE','DEPT','SCOTT') from dual;

Para poder listar todos los objetos de una base de datos Oracle se utilizó una consulta a la tabla de objetos de base de datos el cual es dba_objects. En esta tabla se almacena la información básica de cada objeto de una base de datos Oracle.

El procedimiento para la sincronización de objetos Oracle es el siguiente:

a) Realizar una consulta a la tabla dba_objects de todos los objetos de una base de datos de acuerdo al esquema seleccionado.
b) La aplicación realizará la comparación de existencia de los objetos entre las bases de datos origen y destino.
c) Finalmente, el usuario selecciona que objeto sincronizar y la aplicación usando el procedimiento almacenado mencionado anteriormente genera el script de sincronización.

4. Construcción

En este capítulo se explicarán los detalles relacionados a la fase de construcción de la solución tales como tecnologías a utilizar y los tipos de pruebas a utilizar para verificar las funcionalidades del aplicativo.

4.1 Fase de construcción

En la presente sección se discutirá la elección de la tecnología a utilizar para realizar la implementación de la solución. Esto incluye plataformas (o máquinas virtuales), lenguajes de programación, librerías y soporte para pruebas.

4.1.1. Criterios y características

Entre los candidatos para la elección de las tecnologías se ha escogido .NET de Microsoft y Java, siendo C# el lenguaje de programación escogido para .NET.

Para realizar la comparación de tecnologías se han definido distintas características que van a permitir escoger la mejor opción de acuerdo a las necesidades de la solución. Por cada característica se ha definido un puntaje específico que va a permitir medir de manera más adecuada que tecnología es la más apropiada.

A continuación se muestra la lista de características con sus respectivos puntajes.

CARACTERÍSTICAS	PUNTAJE
Facilidad para abstraer la conexión a diferentes DBMS	20
Soporta programación orientada a objetos	18
Provee librerías para conexión a base de datos	15
Permite reflexión de clases	15
Permite lectura de archivos XML	15
Provee librerías para realización de pruebas unitarias	12
Uso de componentes gráficos (ventanas, componentes GUI, etc.)	12
Facilidad y amigabilidad de uso del lenguaje de programación	10
Multiplataforma	8

Tabla 4.1 Características a evaluar para la elección de la tecnología

Si bien ambas tecnologías soportan las características mencionadas, lo que se pretende medir es la facilidad de uso, configuración y eficiencia (uso de recursos) de cada una de ellas.

Con respecto al cuadro anterior, se observa que el criterio más importante para la elección de una tecnología es la facilidad para abstraer la conexión y ejecución de sentencias SQL a diferentes DBMS. La abstracción del DBMS involucra los siguientes factores:

- Configuración de la aplicación para realizar la conexión a un DBMS.
- Proveer de una librería base para realizar operaciones básicas sobre un DBMS (abrir una conexión, cerrar una conexión y ejecutar sentencias DML y DDL).
- Factibilidad para cargar en tiempo de ejecución librerías que provean interfaces para establecer una conexión a un DBMS en particular. Por ejemplo, permitir que se cargue una librería específica para realizar la conexión a una base de datos Oracle, el cual va a ser utilizada por la librería base. La función de estas librerías es de ser un API entre la aplicación y una base de datos en particular.

A continuación se muestran las herramientas necesarias para la solución provistas por las 2 tecnologías propuestas.

HERRAMIENTA	.NET	JAVA
Lenguaje de programación	C#	Java
Puente para conexión a base de datos	ODBC	JDBC Driver (Nativo)
Framework para realización de pruebas unitarias	NUnit	JUnit
IDE	Visual Studio 2005	Netbeans 6.0

Tabla 4.2 Herramientas necesarias para la solución

En el caso de .NET se han analizado los criterios y herramientas tomando en cuenta la versión 2 del framework .NET. De igual manera, para el caso de Java se ha considerado la versión 6 del JDK (Java SE Development Kit).

4.1.2. Comparación y selección de la tecnología

Con los criterios definidos y las herramientas especificadas, se ha podido elaborar un cuadro comparativo entre las tecnologías propuestas y analizar cuál de ellas es conveniente para la implementación de la solución.

CARACTERÍSTICA	.NET	JAVA
Facilidad para abstraer la conexión a diferentes DBMS	13	18
Soporta programación orientada a objetos	18	18
Provee librerías para conexión a base de datos	15	15
Permite reflexión de clases	10	14
Permite lectura de archivos XML	14	12
Provee librerías para realización de pruebas unitarias	10	10
Uso de componentes gráficos (ventanas, componentes GUI, etc.)	12	10
Facilidad y amigabilidad de uso del lenguaje de programación	8	9

Multiplataforma	4	8
TOTAL	**104**	**114**

Tabla 4.3 Cuadro comparativo de tecnologías

Como se aprecia del cuadro anterior, una de las principales ventajas que tiene Java sobre .NET es la facilidad para realizar una abstracción del DBMS. Para el caso de Java solo es necesario descargar la librería correspondiente, cargarla en tiempo de ejecución y registrar el Driver correspondiente. En el caso de .NET es necesario instalar un controlador ODBC por cada DBMS, lo cual puede ser tedioso en algunos casos.

En lo que respecta al uso de reflexión, Java provee interfaces y clases más intuitivas para llevar a cabo la reflexión, a diferencia de .NET en donde se tienen que seguir pasos adicionales de configuración y la forma de usar la reflexión no llega a ser tan intuitiva como Java.

Por el contrario, la lectura de archivos XML por parte de .NET se hace de una manera más eficiente que en Java, pues se define un cursor que va recorriendo el archivo (como si fuera un archivo de texto), mientras que en Java para buscar una etiqueta en particular recorre todo el archivo buscando dicha etiqueta y no de manera secuencial. Esto produce uso de más de los recursos del sistema.

Finalmente, la aplicación desarrollada en Java va a poder ser utilizada en distintos ambientes y plataformas (Windows, Unix, etc.) tomando en cuenta que en la actualidad muchas empresas están migrando sus plataformas a Unix (servidores de bases de datos y computadoras de escritorio). Esto provee de una ventaja adicional para Java, pues las aplicaciones .NET solo pueden ser utilizadas sobre plataformas Windows.

Con todo lo descrito anteriormente, para el desarrollo de la solución se ha escogido Java como tecnología a utilizar básicamente por su facilidad para realizar una abstracción del DBMS.

4.2 Pruebas

En esta sección se definirán los tipos de pruebas a utilizar y una breve justificación de cada una de ellas. El catálogo de pruebas estará adjunto en la sección de Anexos.

Los tipos de pruebas escogidos para realizar la implementación de la solución son las pruebas de aceptación y las pruebas unitarias. A continuación se definirán cada una de ellas con una breve justificación de su elección y un ejemplo.

4.2.1 Pruebas de aceptación

Este tipo de pruebas consiste en verificar la conformidad del sistema frente a un requerimiento del usuario. Esta verificación se realizará de acuerdo a ciertos criterios que el usuario establezca inicialmente y el sistema deberá mostrar la información esperada de acuerdo a dichos criterios. (Ver referencia [AMB2002] sobre este tipo de pruebas).

Para la solución, este tipo de pruebas es importante debido a la necesidad de verificar que se esté realizando la sincronización de los objetos de base de datos de acuerdo a la configuración que decida el usuario. Los cambios pueden ser apreciados desde en el mismo sistema (al momento de refrescarse la estructura de la base de datos) o desde una herramienta de administración de base de datos que permita visualizar la estructura de la base de datos.

Además este tipo de pruebas también permite verificar que los resultados de la comparación sean correctos al visualizarse en pantalla, tomando en cuenta que se pueden comparar diferentes DBMS siendo necesario el uso de un catálogo de equivalencias de tipos de datos.

Resumiendo lo expresado anteriormente, las pruebas de aceptación van a permitir validar la comparación y sincronización de los objetos de base de datos seleccionados estableciendo condiciones y criterios iniciales. Ambos procesos serán probados tomando en cuenta 2 escenarios: la base de datos origen y destino son del mismo fabricante y que ambas bases de datos sean de diferentes

fabricantes. Para el primer escenario se probará sobre una base de datos MSSQL Server 2000 y en el segundo caso sobre bases de datos MSSQL Server 2000 y Oracle 10g.

A continuación se muestran 2 ejemplos que reflejan las pruebas de la comparación y sincronización.

Identificador	**TA001**
Descripción	Comparación de la columna *EmployeeName* de la tabla *Employee* de la base de datos *Northwind* y de la tabla *Employee* de la base de datos *NorthwindMod*.
Componente a validar	Interfaz de comparación de columnas
Parámetros iniciales	1. Se establece la fuente de datos origen con una base de datos SQL Server: - Servidor: GIANCARLO - Base de datos: *Northwind* - Usuario: sa - Contraseña: sa 2. Se establece la fuente de datos destino con una base de datos SQL Server: - Servidor: GIANCARLO - Base de datos: *NorthwindMod* - Usuario: sa - Contraseña: sa 3. En la tabla *Employee* de la BD *Northwind*, la columna a comparar contiene los siguientes atributos: Tipo de dato: VARCHAR Longitud: 50 Precisión: 0 Escala: 0 Admite Nulos: Sí Valor por defecto: - En la tabla *Employee* de la BD *NorthwindMod* los atributos de la columna son: Tipo de datos: VARCHAR Longitud: 35 Precisión: 0 Escala: 0 Admite Nulos: No Valor por defecto: - 4. Se carga la estructura de ambas bases de datos mediante el sistema.
Instrucciones	1. Se escoge la tabla *Employee* de la base de datos *Northwind* y *NorthwindMod*. 2. Se selecciona “Comparar columnas”. 3. Seleccionar la columna *EmployeeName* de la tabla *Employees* de la base de datos *Northwind*.

Resultados esperados	En la ventana de resumen de comparación, la fila correspondiente a la columna *EmployeeName* debe aparecer pintada de color rojo indicando que existen diferencias sobre la misma columna.

Tabla 4.4 Prueba de aceptación TA001

Identificador	**TA002**
Descripción	Sincronización de la tabla *Employee* a nivel de columnas hacia la BD *NorthwindMod*.
Componente a validar	Requerimiento: Sincronización de tablas a nivel de columnas
Parámetros iniciales	1. Se establece la fuente de datos origen con una base de datos SQL Server: - Servidor: GIANCARLO - Base de datos: *Northwind* - Usuario: sa - Contraseña: sa 2. Se establece la fuente de datos destino con una base de datos SQL Server: - Servidor: GIANCARLO - Base de datos: *NorthwindMod* - Usuario: sa - Contraseña: sa 3. En la BD Northwind la tabla *Employee* cuenta con la siguiente estructura de columnas: *IdEmployee*: INT, NOT NULL *EmployeeName*: VARCHAR(50) NOT NULL *HireDate*: DATETIME NULL En la BD *NorthwindMod* la tabla *Employee* cuenta con la siguiente estructura: *IdEmployee*: INT, NOT NULL *EmployeeName*: VARCHAR(30) NULL
Instrucciones	1. Se escoge la tabla *Employee* de ambas bases de datos. 2. Se selecciona “Sincronizar columnas”. 3. Se selecciona “Realizar sincronización directa”.
Resultados esperados	Debe aparecer un mensaje de confirmación indicando que la sincronización se realizó con éxito. Además se debe revisar que la estructura de la tabla *Employee* de la BD *NorthwindMod* sea la siguiente: *IdEmployee*: INT, NOT NULL *EmployeeName*: VARCHAR(50) NOT NULL *HireDate*: DATETIME NULL

Tabla 4.5 Prueba de aceptación TA002

4.2.2 Pruebas unitarias

Este tipo de pruebas permiten verificar que el comportamiento de los métodos implementados de las clases sea el esperado. Esta prueba se basa en indicar los inputs necesarios para el método y el resultado se compara con el esperado (calculado de manera manual). Si ambos valores de retorno son iguales, entonces se ha pasado la prueba unitaria. (La definición de este tipo de pruebas fue sacada de la referencia [WEB013]).

Debido a que se va a utilizar Java como lenguaje de programación, se ha escogido el framework de JUnits para la implementación de las pruebas unitarias, debido a su fácil integración con aplicaciones Java así como su integración con Netbeans 6.0.

Las pruebas unitarias se realizarán sobres los métodos que carguen la metadata de la base de datos, es decir aquellos métodos que lean de la plantilla XML la información necesaria para armar la sentencia SELECT necesaria, así como sobre los métodos que armen las sentencias DDL para realizar la sincronización.

Adicionalmente, este tipo de pruebas se realizarán sobre los métodos que cargan los arreglos con los objetos de base de datos (tablas, índices, columnas y constraints) con el objetivo de verificar si se han cargado correctamente cada objeto de la BD.

Para ilustrar lo explicado anteriormente se presentan dos ejemplos de las pruebas unitarias.

```java
/**
 * Test of leerMetadataTablas method, of class PlantillaXML.
 */
@Test
public void leerMetadataTablas() throws Exception {
    System.out.println("Prueba unitaria del método leerMetadataTablas().");
    System.out.println("Se probará crear la sentencia SQL para obtener el nombre de las tablas de un esquema de " +
            "una base de datos MSSSQL Server 2000.");
    System.out.println("La sentencia SQL esperada es: ");
    System.out.println("SELECT NAME FROM SYSOBJECTS WHERE USER_NAME(?) = ? AND TYPE = 'U' ORDER BY NAME");
    System.out.println("Inicio de la prueba...");
    ArrayList<ParametroConexion> pArrParametros = new ArrayList<ParametroConexion>();
    Object pObjetoNegocio = new Esquema("dbo");
    PlantillaXML instance = new PlantillaXML("C:\\Users\\Giancarlo\\SincronizadorBD\\plantillaMSSQL.xml");
    String expResult = "SELECT NAME AS 'nombreTabla' FROM SYSOBJECTS WHERE USER_NAME(UID) = ? " +
            "AND TYPE = 'U' ORDER BY NAME ASC";
    String result = instance.leerMetadataTablas(pArrParametros, pObjetoNegocio);
    assertEquals(expResult, result);
}
```

Figura 4.1 Prueba unitaria del método leerMetadataTablas ()

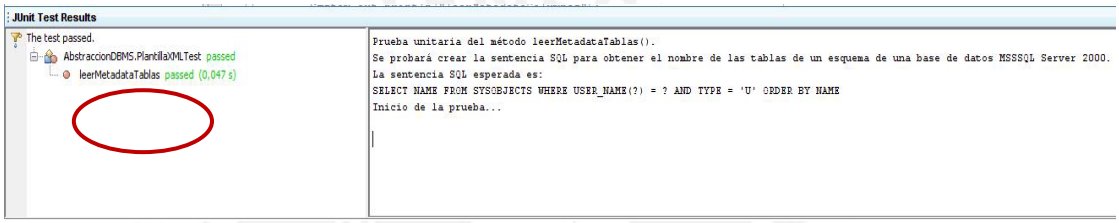


Figura 4.2 Resultados de la prueba unitaria del método leerMetadataTablas ()

```java
@Test
public void leerMetadataColumnas() throws Exception {
    System.out.println("Prueba unitaria del método leerMetadataColumnas().");
    System.out.println("Se probará crear la sentencia SQL para obtener el nombre de las columnas de una tabla de " +
            "una base de datos Oracle 10g.");
    System.out.println("La sentencia SQL esperada es: ");
    System.out.println("SELECT " +
            "DATA_DEFAULT  " + '"' + "valorDefecto" + '"' + "," +
            "COLUMN_NAME  " + '"' + "nombreColumna" + '"' + "," +
            "DATA_TYPE  " + '"' + "tipoDato" + '"' + "," +
            "DATA_LENGTH  " + '"' + "longitud" + '"' + "," +
            "DATA_PRECISION  " + '"' + "precisionNumerica" + '"' + "," +
            "DATA_SCALE  " + '"' + "escalaNumerica" + '"' + "," +
            "DECODE(NULLABE,'Y',1,0) " + '"' + "aceptaNulos" + '"' +
            " FROM DBA_TAB_COLUMNS" +
            " WHERE" + " TABLE_NAME = ? AND" + " OWNER = ?" + " ORDER BY COLUMN_ID ASC");
    System.out.println("Inicio de la prueba...");
    ArrayList<ParametroConexion> pArrParametros = new ArrayList<ParametroConexion>();
    PlantillaXML instance = new PlantillaXML("C:\\Users\\Giancarlo\\SincronizadorBD\\plantillaOracle.xml");
    Object pObjetoNegocio = new Tabla("dbo", "Customer", instance, new ConexionBD());
    String expResult = "SELECT " +
            "DATA_DEFAULT  " + '"' + "valorDefecto" + '"' + "," +
            "COLUMN_NAME  " + '"' + "nombreColumna" + '"' + "," +
            "DATA_TYPE  " + '"' + "tipoDato" + '"' + "," +
            "DATA_LENGTH  " + '"' + "longitud" + '"' + "," +
            "DATA_PRECISION  " + '"' + "precisionNumerica" + '"' + "," +
            "DATA_SCALE  " + '"' + "escalaNumerica" + '"' + "," +
            "DECODE(NULLABLE,'Y',1,0)  " + '"' + "aceptaNulos" + '"' +
            " FROM DBA_TAB_COLUMNS" +
            " WHERE" +
            " TABLE_NAME = ? AND" +
            " OWNER = ?" +
            " ORDER BY COLUMN_ID ASC";
    String result = instance.leerMetadataColumnas(pArrParametros, pObjetoNegocio);
    assertEquals(expResult, result);
}
```

Figura 4.3 Resultados de la prueba unitaria del método leerMetadataColumnas ()

Como se aprecia en las figuras 4.1 y 4.3, las pruebas unitarias se realizan por métodos, en donde se define algunos parámetros iniciales y posteriormente se invoca al método a probar. Luego mediante la llamada **assertEquals** se evalúa si el resultado esperado es igual al obtenido. Si esto se cumple, el método ha pasado la prueba unitaria. Cabe resaltar que el IDE Netbeans provee de una serie de opciones para elaborar plantillas que facilitan la implementación de las pruebas unitarias. Este plug-in permite enfocarse más en la obtención del resultado y la validez de la prueba.

5. Observaciones, conclusiones y recomendaciones

En este capítulo, se enunciarán cuáles han sido las observaciones, conclusiones y recomendaciones a las que se ha llegado con al desarrollo del presente proyecto.

5.1. Observaciones

Las observaciones más resaltantes del desarrollo del producto son:

a. Si bien la solución ha sido diseñada bajo una arquitectura independiente de la base de datos, existen ciertas restricciones con respecto a la implementación de la capa de abstracción de la solución. Entre tales restricciones se tienen:

- Las consultas al diccionario de datos mediante sentencias SQL no pueden ser demasiado complejas, debido a la falta de inclusión de algunos operadores tales como el operador EXISTS, BETWEEN, IN entre otros.
- Donde sea necesario usar estos operadores, se deberá definir una vista en la base de datos para agrupar la información de manera tal que la capa de abstracción pueda leerla de manera correcta y segura.

b. Los objetos que puedan compararse y sincronizarse va a depender mucho de la información obtenida del diccionario de datos del DBMS y la disponibilidad que se tenga de dicha información. En caso de no contar con dicha disponibilidad no permitirá realizar la comparación y sincronización de todos los objetos que se pretende abarcar con la solución.

5.2. Conclusiones

Las conclusiones a las que se ha llegado con el presente proyecto son las siguientes:

a. Se ha logrado implementar una solución automatizada que permite realizar la sincronización de los objetos más importantes dentro de una base de datos como lo son las tablas, columnas, constraints, vistas y las cabeceras de disparadores y procedimientos almacenados.

b. Mediante el uso de una metodología ágil como AUP se ha logrado definir las fases del desarrollo de la aplicación más adecuadas, evitando así documentación no relevante para la implementación de la solución.

c. Una correcta gestión del proyecto y continuo monitoreo de las actividades a partir de los diagramas de GANTT y el WBS han permitido que las fases de implementación se lleven a cabo dentro de periodos de tiempo razonables de acuerdo a los tiempos previstos y al uso de recursos.

d. La solución ha sido diseñada bajo una arquitectura independiente de la base de datos lo que permite comparar y sincronizar diferentes DBMS sin necesidad de cambiar la codificación de la aplicación.

e. El correcto uso de las tecnologías escogidas para el desarrollo de la solución ha permitido asegurar que la aplicación sea implementada de acuerdo a los requerimientos no funcionales definidos y a la arquitectura seleccionada.

f. Esta herramienta ha sido diseñada para que usuarios con los conocimientos adecuados sobre administración de bases de datos lo utilicen en sus labores relacionadas a migración y sincronización de bases de datos. Dichas labores

se presentan cuando se desarrollan actualizaciones a los sistemas de producción o cuando se tienen bases de datos distribuidas.

5.3. Recomendaciones y trabajos futuros

Como se mencionó anteriormente, esta aplicación fue diseñada para que usuarios expertos en la administración de base de datos la utilicen dentro de su ámbito de trabajo para realizar sincronizaciones de objetos de las bases de datos de diferentes entornos. Además, dado a que se realizan consultas al diccionario de la base de datos, es necesario registrarse en el sistema con un usuario administrador.

Debido a que la aplicación no contempla la sincronización del cuerpo de los triggers y procedimientos almacenados, la implementación de un traductor de lenguajes procedurales entre distintos DBMS podría ampliar el uso del sincronizador de bases de datos. Dicho traductor puede ser utilizado dentro de la aplicación para comparar y sincronizar de manera más detallada los disparadores y procedimientos almacenados. Para su implementación sería necesario implementar las gramáticas de dichos lenguajes para posteriormente seguir con la traducción de bloques SQL de un DBMS a otro.

Este aporte sería muy importante para lograr una sincronización casi completa de una base de datos brindándole al usuario más opciones para realizar dicha labor.

Bibliografía

1. [MAN2007] MANNINO, MICHAEL V., Administración de bases de datos: Diseño y desarrollo de aplicaciones, Tercera Edición, McGraw-Hill, México, 2007.

2. [TSA1990]. TSAI, ALICE Y.H., Sistemas de bases de datos: administración y uso, Primera edición, Prentice-Hall, México, 1990.

3. [PMI2004] PROJECT MANAGEMENT INSTITUTE, A Guide to the Project Management Body of Knowledge, Tercera Edición, PMI Standard, Estados Unidos de América, 2004.

4. [PMI2006] PROJECT MANAGEMENT INSTITUTE, Practice Standard for Work Breakdown Structures, Segunda Edición, Global Standard, Estados Unidos de América, 2006.

5. [DEL2007] DELANEY, KALEN, Inside Microsoft SQL Server 2005: The Storage Engine, Primera Edición, Microsoft Press, Estados Unidos de América, 2007.

6. [WEB001] DBBALANCE Cross Database Solutions, Cross-Database Studio 6.0, http://www.dbbalance.com/db_studio.htm

7. [WEB002] Database Glossary. Important database design terms in database design defined, http://www.databasedesign-resource.com/database-glossary.html

8. [WEB003] Compare database schemas, synchronize database schemas with SQL Compare, http://www.red-gate.com/products/SQL_Compare/index.htm

9. [PDF001] The Oracle DBMS Architecture: A Technical Introduction - http://www.naspa.com/files/CDKIT/CD01/1997/February/T9702004.PDF

10. [PDF002] MySql Conceptual Architecture - www.swen.uwaterloo.ca/~mrbannon/cs798/assignment_01/mysql.pdf

11. [PDF003] Improved database development, http://www.red-gate.com/products/SQL_Compare/SQL_Compare.pdf

12. [WEB004] Coordinating database development change, http://www.red-gate.com/products/solutions_for_sql/coordinating_database_development_changes.htm

13. [WEB005] Database Restyle – Database Synchronization Component Designed for MS SQL 2005, http://www.perpetuumsoft.com/Product.aspx?lang=en&pid=55

14. [WEB006] Database Workbench 3 Documentation - Index, http://www.upscene.com/documentation/dbw3/index.html

15. [WEB007] Upscene: Database tools for developers. Database developer tools, http://www.upscene.com/index.htm?./products/dbw/index.htm

16. [WEB008] Easy SQL Tools | Schema Compare, http://www.easysqltools.com/EasySQLSchemaCompare.aspx

17. [WEB009] UDB Workbench – Features, http://www.easysqltools.com/EasySQLSchemaCompare.aspx

18. [WEB010] DB Tools for Oracle, http://www.softtreetech.com/monitor/

19. [WEB011] Compare and Synchronize SQL Server Databases - http://www.swissql.com/products/database-compare-synchronize-tool/synchronize-sql-server-databases.html

20. [WEB012] SQL Server – Generic Architecture Image - http://blog.sqlauthority.com/2007/12/08/sql-server-generic-architecture-image/

21. [PPT001] DB Difference Expert, http://www.softtreetech.com/monitor/demos/dbDiff.ppt

22. [AMB2005] AMBLER, SCOTT. The elements of UML 2.0 Style. Primera edición, Cambridge University Press, Estados Unidos de América, 2005.

23. [AMB2002] AMBLER, SCOTT. JEFFRIES, RON. Agile Modeling: Effective Practices for Extreme Programming and the Unified Process. Primera edición, John Wiley & Sons Inc, Estados Unidos de América, 2002.

24. [VER2000] VERMEULEN, ALAN. AMBLER, SCOTT. BUMGARDNER, GREG. The Elements of Java Style. Primera edición, Cambridge University Press, Estados Unidos de América, 2000.

25. [WEB013] The Agile Unified Process (AUP) Home Page. http://www.ambysoft.com/unifiedprocess/agileUP.html

26. [WEB014] Análisis de sistemas de información. http://docente.ucol.mx/grismu/public_html/analis.htm

27. [WEB015] Agile Alliance: 10 Key Principles of Agile Development. http://kw-agiledevelopment.blogspot.com/2007/02/10-things-you-need-to-know-about-agile.html

28. [PDF004] An Introduction to Software Architecture. www.cs.cmu.edu/afs/cs/project/vit/ftp/pdf/intro_softarch.pdf

29. [WEB016] There's something about SQL! http://blogs.technet.com/beatrice/default.aspx?p=2

30. [WEB017] Oracle Data Dictionary Design Data Model. http://www.databaseanswers.org/data_models/oracle_data_dictionary/index.htm

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE UN COMPARADOR Y SINCRONIZADOR DE BASES DE DATOS RELACIONALES DE DISTINTOS MANEJADORES

ANEXOS

Giancarlo Roberto Calderón Garay

ASESOR: Ing. Claudia María del Pilar Zapata Del Río

Lima, Abril del 2009

Anexos

Anexo A: Arquitecturas de manejadores de bases de datos

A continuación se describe la arquitectura de los manejadores de bases de datos que la solución soportará mediante una breve explicación de cada arquitectura y su correspondiente representación gráfica. De esta manera se tendrá una noción de cómo está diseñado cada manejador y cómo la solución podrá interactuar con dichos manejadores.

Arquitectura de MSSQL Server 2000

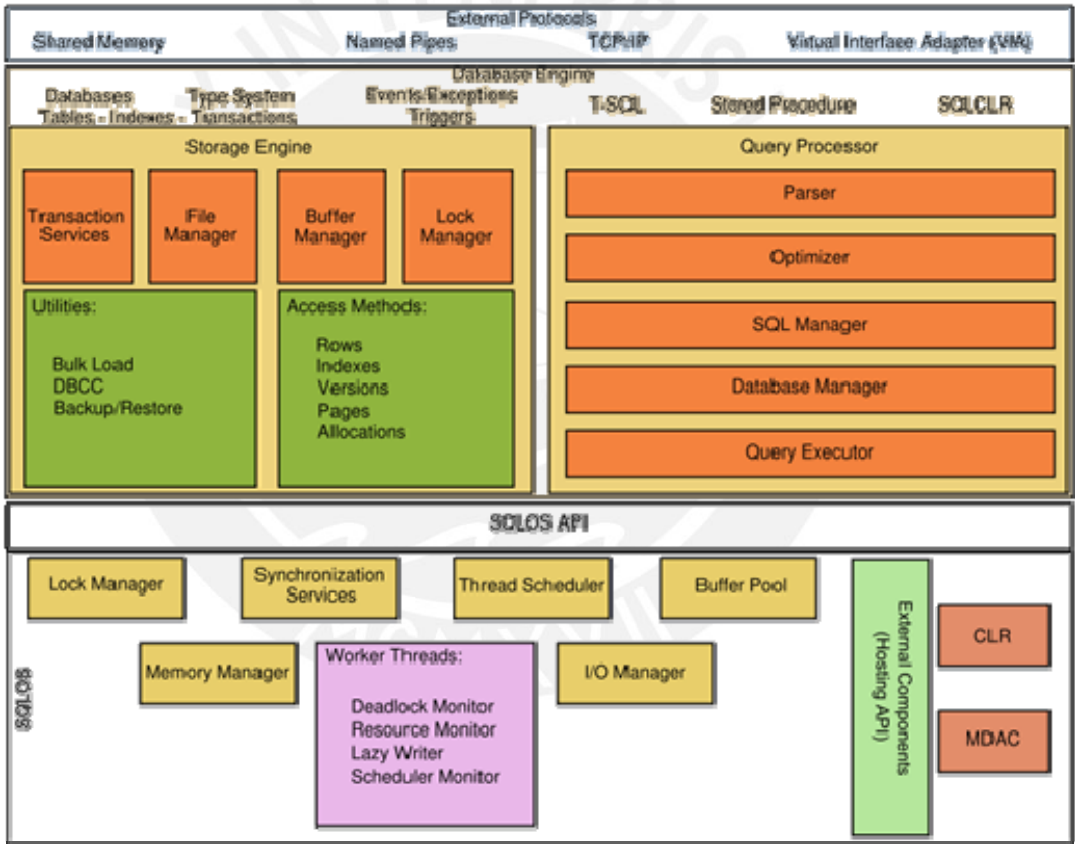


Figura A.1 Arquitectura de SQL Server

Como se aprecia en la figura 1.1 tomada de la referencia [WEB012], la arquitectura de MSSQL Server se divide en 3 capas bien definidas, cuya descripción según [DEL2007] es la siguiente:

a. *Capa SQLOS*

Es la capa base dentro de la arquitectura de MSSQL Server. Esta capa se encarga de implementar funciones realizadas comúnmente por el Sistema Operativo como manejo de memoria, programación de hilos, manejo del flujo de entrada y salida, manejo de un *pool* de conexiones, primitivas de sincronización y detección de *deadlocks*.

Debido a la alta especialización de los requerimientos de MSSQL Server, este manejador implementa su propio sistema de manejo de hilos, dividiendo todas las operaciones en una serie de tareas (Tasks) los cuales se encargan de procesar todos los pedidos de los clientes. Cada tarea es ejecutada por un hilo, cuya asociación permanece hasta la finalización de la tarea.

Una labor importante de esta capa es reducir las operaciones de entrada y salida al disco. Para cumplir con este objetivo, MSSQL Server mantiene un buffer especializado para almacenar páginas de datos desde el disco (similar a una memoria cache) acelerando así las operaciones de lectura y escritura del procesador de sentencias y otras estructuras internas. SQLOS monitorea siempre este buffer para poder determinar cuándo enviar estos datos al disco y dejar espacio libre para los nuevos datos requeridos.

b. *Capa del Motor Relacional (Relational Engine)*

Esta capa implementa las funciones relacionadas al almacenamiento de la información. Entre sus funciones está la implementación de los tipos de datos que pueden ser utilizados en tablas, así como que tipos de objetos se pueden almacenar en la base de datos (como tablas, índices, logs, etc.). Implementa, además, cómo los datos van a ser almacenados en los dispositivos físicos, así como los métodos para extraer dichos datos. Finalmente, esta capa se encargar de proveer el procesador de sentencias, que se encarga de extraer los datos mediante la traducción de la sentencia SQL en una secuencia de operaciones para extraer dicha data, previa optimización de dicha sentencia para hacer menor de uso de recursos de sistema.

c. Capa de Protocolo

Esta capa implementa la interface externa hacia un manejador de MSSQL Server. Todas las operaciones realizadas hacia SQL Server son enviadas en un formato especial llamado Flujo Tabular de Datos (Tabular Data Stream TDS). Los paquetes TDS pueden ser cubiertos por otros protocolos dependientes del tipo de transporte físico como TCP/IP, Pipes nombrados o memoria compartida.

Esta última capa se encarga de proveer el medio de comunicación hacia una base de datos SQL Server, mediante el cual la aplicación podrá establecer una conexión y obtener la metadata. Posteriormente, mediante sentencias SQL se podrá realizar la sincronización para modificar la estructura de los objetos de base de datos seleccionados.

Arquitectura de Oracle

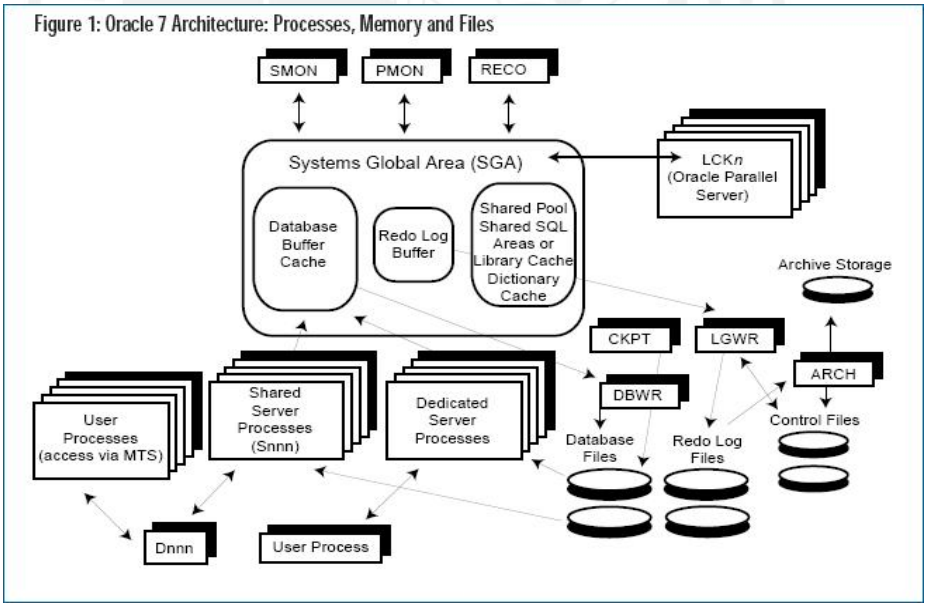


Figura A.2 Arquitectura de Oracle

La información que a continuación se describe fue tomada del documento *"The Oracle DBMS Architecture: A Technical Introduction"* (Ver referencia [PDF001]).

Un manejador Oracle compromete dos tipos de procesos: procesos de usuario o cliente y procesos de sistema Oracle. Los procesos de usuario son creados para

ejecutar algún código desde una aplicación. Además se encarga de manejar la comunicación con los procesos de servicio Oracle. Por otra parte, los procesos de sistema Oracle son llamados desde otros procesos Oracle para realizar funciones de acuerdo a lo especificado en el proceso invocador. Estos tipos de procesos se dividen en dos tipos: procesos de servidor y procesos de *background*.

a. *Procesos de Servidor*

La función de estos procesos es recibir los pedidos de los procesos de usuario. Primero reciben los pedidos en forma de sentencias SQL, revisa en el *shared pool* junto con SGA (estos conceptos serán explicados más adelante) si la sentencia ha pasado por el proceso de *parse*, revisa privilegios de acceso, obtiene la información ya sea del *datafile* correspondiente o del buffer cache de la instancia SGA y permite que dicha información esté disponible para el usuario. La comunicación entre los procesos de servidor y usuario se pueden dar de 3 maneras diferentes:

- **Servidor dedicado.** Cada proceso de servidor atiene un único proceso de usuario. A pesar de que el proceso de usuario está inactivo, el proceso dedicado de servidor aún existe. Este escenario es común cuando la aplicación y el servidor Oracle están en distintas máquinas.
- **Servidor multi-hilo.** En esta configuración, varios procesos de usuario pueden ser atendidos por una cantidad pequeña de procesos compartidos de servidor. Esta configuración requiere de un *listener* de red, que se encarga de escuchar los pedidos de los procesos de usuarios y conectarlos a un despachador de procesos que se encargan de asignar estos procesos al próximo proceso de servidor disponible.
- **Procesos de servidor compartido.** Similar al servidor dedicado con la diferencia de que cuando después de ejecutarse un pedido SQL, libera sus recursos para atender un diferente proceso de usuario.

b. *Procesos de background*

Algunos procesos de *background* importantes son los siguientes:

- **Monitor de Sistema** (SMON). Se encarga de tareas de recuperación al momento de iniciar la instancia Oracle, limpiar segmentos temporales no usados y recuperar transacciones "muertas" por conexiones cerradas abruptamente.
- **Monitor de Proceso** (PMON). Se encarga de tareas de recuperación cuando un proceso de usuario falla.
- **Escritor de bases de datos** (DBWR). Se encarga de copiar los datos de la cache del SGA hacia los archivos físicos de datos mediante un algoritmo que determina los datos menos usados recientemente.
- **El recuperador** (RECO). Se encarga de resolver conflictos entre transacciones distribuidas.
- **El archivador** (ARCH). Copia en línea la información de los archivos redo-log (los cuales almacenan las transacciones realizadas en la base de datos) hacia otro dispositivo de almacenamiento.

La principal estructura de memoria de la arquitectura Oracle es el SGA (*System Global Area*), el cual se compone de las siguientes 3 estructuras.

- **Buffer Cache de la base de datos.** Almacena la información utilizada más reciente de los archivos de disco.
- **Shared Pool.** Contiene definiciones para las sentencias SQL en tiempo de ejecución e información del diccionario de datos.
- **Buffer Redo-log.** Área utilizada por Oracle para almacenar los cambios ocurridos en la base de datos debido a sentencias SQL INSERT, UPDATE, DELETE, CREATE y DROP.

Arquitectura de MySql

La información sobre la arquitectura de MySql fue tomada del documento *"MySql Conceptual Architecture"* (Ver referencia [PDF002]).

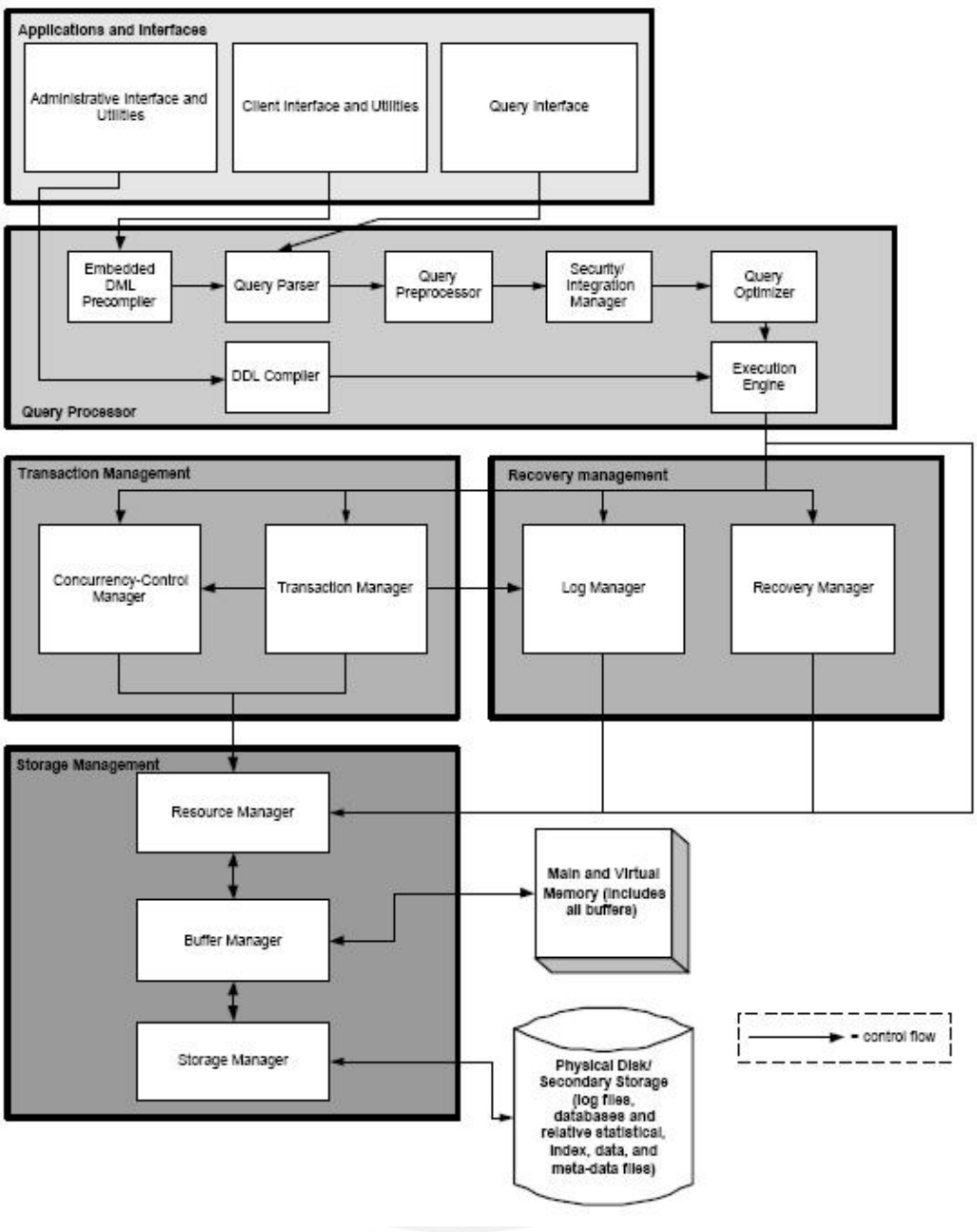


Figura 7.3 Arquitectura de MySql

La arquitectura de MySql se divide básicamente en tres capas: la capa de aplicación, la capa lógica y la capa física. A continuación se describe brevemente cada una de ellas.

1. Capa de aplicación

Esta capa permite que los clientes y usuarios se comuniquen con un manejador MySql. Como se muestra en la figura 1.3 existen distintos tipos de usuario que pueden interactuar como administradores, clientes y usuarios que realizan

sentencias SQL. Las interfaces cliente utilizan diferentes APIs para varios lenguajes de programación como C++, Java, Perl, PHP entre otros. Así mismo existen distintas herramientas y utilitarios que permiten que un usuario ejecute sentencias SQL directamente a una base de datos MySql y ver los resultados inmediatamente.

2. Capa lógica

Esta capa está formada por distintos subsistemas que a continuación son descritos:

a) *Procesador de consultas*. La función de este subsistema es realizar el proceso de parse y optimización de consultas realizadas mediante un lenguaje de manipulación de información (como SQL). Los componentes del procesador de consultas son:

- Pre-compilador embebido DML.
- Compilador DDL
- Parser de consultas
- Pre-procesador de consultas
- Administrador de seguridad e integración
- Optimizador de consultas
- Motor de ejecución de sentencias

b) *Administrador de Transacciones*. La función de este subsistema es asegurar que una transacción, definido como un conjunto de sentencias SQL que se deben ejecutar de manera atómica, sea ejecutada de manera segura y atómica. Además, se encarga de resolver los conflictos ocasionados por *deadlocks*. Adicionalmente, este componente es responsable de ejecutar los comandos COMMIT y ROLLBACK de una transacción.

Para realizar esta labor, el administrador de transacciones se apoya del administrador de control de concurrencia quien es responsable que las transacciones se ejecuten de manera separa e independiente.

c) *Administración de Recuperación.* Formado por dos componentes:

- **Administrador de Log (Log Manager).** Se encarga de almacenar cualquier operación hecha hacia la base de datos. Dichas operaciones son almacenadas en forma de sentencias SQL, las cuales serán ejecutas para regresar la base de datos a su último estado estable en caso de una caída de la base de datos.
- **Administrador de Recuperación (Recovery Manager).** Es el encargado de restaurar la base de datos a su último estado estable mediante la ejecución de las operaciones almacenadas en los logs. Esta información es adquirida desde el buffer de memoria.

d) *Administración de Almacenamiento*. Formado por los siguientes componentes:

- **Administrador de Almacenamiento (Storage Manager).** Se encarga de enviar los datos desde los archivos físicos hacia el Buffer Manager mediante los controladores del disco o vía el sistema operativo.
- **Buffer Manager.** Su rol corresponde a destinar los recursos de memoria para la consulta y manipulación de los datos. Además, se encarga de definir cuanta memoria destinar por buffer y cuantos buffer necesita destinar por pedido.
- **Administrador de Recursos (Resource Manager).** Se encarga de aceptar las consultas desde el motor de ejecución de sentencias y solicitar información desde el Buffer Manager, la cual luego es enviada a capas superiores.

3. Capa física

Esta capa se encarga de almacenar de manera física la información de la base de datos. Los principales tipos de almacenamiento son:

- *Data files.* Almacenan las tablas de los usuarios.
- *Diccionario de datos.* Almacena información sobre la estructura de la base de datos.
- *Índices.* Proveen acceso rápido a los registros de una tabla.

- *Datos estadísticos.* Almacena información estadística sobre la base de datos. Es utilizado por el procesador de consultas para seleccionar el camino más eficiente para ejecutar una sentencia.
- *Logs.* Almacena trazas de las sentencias ejecutadas de manera que el administrador de recuperación pueda usar dicha información para restablecer la base de datos en caso de alguna caída abrupta.

Anexo B: Especificación de los casos de uso

Establecer la conexión a una base de datos	
Descripción	Permite que el usuario establezca la conexión a una base de datos.
Pre-condición	Ninguna.
Post-condición	Se ha establecido la conexión a una base de datos.
Flujo básico	

1. El usuario selecciona “Establecer origen de datos”
2. El usuario selecciona “Configurar conexión origen” o “Configurar conexión destino”
3. El usuario selecciona el tipo de DBMS origen al cual desea conectarse.
4. Dependiendo del DBMS elegido, el sistema muestra los parámetros de entrada necesarios para establecer una conexión.
5. El usuario ingresa la información solicitada y selecciona “Conectar”.
6. El sistema verifica la información ingresada y si es correcta establece una conexión a la base de datos.
7. Si el DBMS soporta el uso de esquemas, el sistema muestra una pantalla listando los esquemas que se deseen comparar o sincronizar.
8. El usuario selecciona los esquemas de los cuales desea mostrar la información de los objetos de la base de datos pertenecientes a dichos esquemas.
9. El sistema muestra un mensaje de éxito del establecimiento de la conexión a la base de datos.
10. Fin del caso de uso.

Cargar la estructura de una base de datos

Descripción	Permite que el sistema muestre al usuario la estructura de la base de datos.
Pre-condición	El usuario debe haber establecido la conexión a la base de datos.
Post-condición	Se ha cargado la estructura de la base de datos.

Flujo básico

1. El usuario selecciona “Cargar estructura de la base de datos origen” o “Cargar estructura de la base de datos destino”.
2. El sistema realiza la carga de la estructura de la base de datos mostrando el avance en una barra de progreso.
3. El sistema, finalizada la carga de la estructura, muestra en un árbol la estructura de la base de datos origen agrupado por:
   - En caso de haber establecido de que el DBMS escogido soporte el uso de esquemas, la información se agrupará por los esquemas seleccionados.
   - A continuación se muestran 2 ramas correspondientes a las Tablas y Vistas
   - En la rama de Tablas, el sistema lista todas las tablas pertenecientes a la base de datos, en donde cada tabla tiene a su vez las siguientes ramas: Columnas, Índices, Llaves Primarias, Llaves foráneas y Check Constraints.
   - En la rama de Vistas, el sistema lista todas las vistas pertenecientes a la base de datos, en donde cada vista tiene a su vez las siguientes ramas: Columnas e Índices.
   - Finalmente, en la rama de Procedimientos Almacenados el sistema lista los procedimientos almacenados pertenecientes a las bases de datos, en donde cada procedimiento almacenado tiene a su vez la rama de Parámetros.
4. Fin del caso de uso.

Comparar dos objetos de la base de datos	
Descripción	Permite que el usuario realice la comparación de dos objetos de la base de datos.
Pre-condición	El usuario debe haber realizado la carga de las estructuras de las bases de datos origen y destino.
Post-condición	Se ha realizado la comparación de dos objetos de la base de datos.

Flujo básico
1. El usuario selecciona un objeto de la base de datos origen y destino que tengan la misma categoría. Entre las categorías que el usuario puede seleccionar están tablas, columnas, índices, llaves primarias, llaves foráneas, check constraints y vistas. 2. El usuario selecciona “Comparar estructuras”. 3. La aplicación muestra en un cuadro resumen las diferencias encontradas entre las estructuras de los objetos seleccionados indicando qué elementos difieren entre ambos objetos o si alguna característica en particular no se encuentra en una de ellas. 4. Fin del caso de uso.

Sincronizar un objeto de la base de datos	
Descripción	Permite que el usuario sincronice un objeto de la base de datos.
Pre-condición	El usuario debe haber realizado la carga de las estructuras de las bases de datos origen y destino. El usuario debe haber realizado la comparación de los objetos que desea sincronizar.
Post-condición	Se ha sincronizado un objeto de la base de datos.
Flujo básico: Sincronización directa	
1. El usuario selecciona cual va a ser la base de datos que va a sincronizarse. 2. El usuario selecciona una o más características del objeto que desea sincronizar. 3. El usuario selecciona “Sincronizar”. 4. El sistema muestra un mensaje de sincronización en caso de que la operación ya se haya realizado con éxito. Caso contrario muestra el mensaje de error correspondiente a la sentencia DDL generada para la sincronización. 5. Fin del caso de uso.	
Flujo Alternativo: Generación de script de sincronización	
1. El usuario selecciona cual va a ser la base de datos que va a sincronizarse.	

2. El usuario selecciona una o más características del objeto que desea sincronizar.
3. El usuario selecciona “Generar script de sincronización”.
4. El sistema muestra en un formulario el script generado a partir del proceso de sincronización.
5. El usuario selecciona “Guardar”.
6. El usuario ingresa el nombre del archivo y selecciona el directorio del archivo.
7. El usuario selecciona “Aceptar”.
8. El sistema muestra un mensaje de confirmación en caso de que el archivo haya sido creado exitosamente. Caso contrario, muestra el mensaje de error correspondiente.
9. Fin del caso de uso.

Generar reporte de comparación	
Descripción	Permite que el usuario genere un reporte de comparación.
Pre-condición	El usuario debe haber realizado la carga de las estructuras de las bases de datos origen y destino. El usuario debe haber realizado cualquier tipo de comparación.
Post-condición	Se ha generado un reporte de comparación.
Flujo básico	

1. El usuario selecciona el formato del archivo de reporte: HTML o PDF.
2. El usuario selecciona “Generar reporte de comparación”.
3. El usuario ingresa el nombre del archivo para el reporte de comparación.
4. El usuario selecciona “Aceptar”.
5. El sistema muestra un mensaje de confirmación en caso de que el archivo haya sido creado exitosamente. Caso contrario, muestra el mensaje de error correspondiente.
6. Fin del caso de uso.

Anexo C: Especificación de las clases de análisis

BaseDeDatos	
Esta clase representa una base de datos.	
Atributo	Descripción
Nombre Base de Datos	Representa el nombre de la base de datos.
Operaciones	Descripción
Crear Conexión	Método que se encarga de establecer una conexión a una base de datos recibiendo un conjunto de parámetros, un usuario y una contraseña.
Cerrar Conexión BD	Método que se encarga de cerrar una conexión a una base de datos.
Devolver Todos los Esquemas	Método que se encarga de devolver el nombre de todos los esquemas de la base de datos, si es que es soportado el uso de esquemas.
Añadir Esquemas Seleccionados	Método que se encarga de añadir los esquemas que el usuario ha seleccionado a un arreglo de esquemas.
Cargar Tablas	Método que se encarga de cargar las tablas de la base de datos.
Cargar Vistas	Método que se encarga de cargas las vistas de la base de datos.
Sincronizar BD	Método que se encarga de sincronizar toda la estructura de la base de datos hacia otra.

Esquema	
Esta clase representa un esquema de la base de datos.	
Atributo	Descripción
Nombre del Esquema	Nombre de un esquema de la base de datos.
Operaciones	Descripción
Cargar Tablas	Método por el cual se cargan todas las tablas de un esquema.
Cargar Vistas	Método por el cual se cargan todas las vistas de un esquema.
Cargar Procedimientos	Método por el cual se cargan todos los procedimientos de un esquema.

ObjetoBD	
Esta interface representa un objeto de la base de datos.	
Operaciones	Descripción
Obtener Estructura	Método que implementan todos los objetos de las bases de datos para obtener información sobre su metadata.
Sincronizar	Método que implementan todos los objetos de las bases de datos para sincronizarse contra otra base de datos.
Crear SQL Sincronizador	Método que implementan todos los objetos de las bases de datos para crear un archivo SQL con los cambios necesarios para sincronizarse contra otra base de datos.
Comparar	Método que implementan todos los objetos de las bases de datos para compararse contra otro objeto de su misma categoría.

Entidad	
Esta clase abstracta representa una entidad de la base de datos en donde se almacena información como tablas o vistas. Implementa la interface **ObjetoBD**.	
Atributo	Descripción
Nombre de la Entidad	Nombre de la tabla o vista de la base de datos.
Nombre del Esquema	Nombre del usuario dueño de entidad.
Operaciones	Descripción
Obtener Columnas	Método por el cual una tabla o vista obtiene información sobre sus columnas.
Obtener Índices	Método por el cual una tabla o vista obtiene información sobre sus índices.

Tabla	
Esta clase representa una tabla de la base de datos. Hereda de la clase abstracta **Entidad**.	
Operaciones	Descripción

Obtener Llave Primaria	Método por el cual una tabla obtiene información sobre su llave primaria.
Obtener Llaves Foráneas	Método por el cual una tabla obtiene información sobre sus llaves foráneas.
Obtener Check Constraints	Método por el cual una tabla obtiene información sobre sus check constraints.
Obtener Triggers	Método por el cual una tabla obtiene información sobre sus triggers.

Vista	
Esta clase representa una vista de la base de datos. Hereda de la clase abstracta **Entidad**.	
Atributo	Descripción
Definición de la Vista	Contiene la sentencia SQL que definió a la vista.
Operaciones	Descripción
Obtener Definición de la Vista	Método por el cual una vista obtiene la información sobre la sentencia SQL que la definió.

Columna	
Esta clase representa una columna de la base de datos. Implementa la interface **ObjetoBD**.	
Atributo	Descripción
Nombre de la Columna	Nombre de la columna.
Nombre de la Tabla	Nombre de la tabla a la que pertenece la columna.
Tipo de dato	Tipo de dato correspondiente a la columna.
Longitud	Longitud del tipo de dato de la columna. Este campo tiene más relevancia para tipos de datos cadenas.
Precisión Numérica	Precisión numérica del tipo de dato de la columna. Este campo es relevante cuando el tipo de dato es numérico.
Escala Numérica	Escala numérica del tipo de dato de la columna. Este campo es relevante cuando el tipo de dato es numérico.
Acepta nulos	Indica si la columna acepta valores nulos.
Valor por defecto	Valor por defecto si la columna no contiene dato alguno.

Índice	
Esta clase abstracta representa un índice de una tabla o vista. Implementa la interface **ObjetoBD**.	
Atributo	Descripción
Nombre del índice	Nombre del índice.
Nombre de la tabla	Nombre de la tabla asociada al índice.
Esquema del índice	Nombre del usuario creador del índice.
Tipo de índice	Indica el tipo de índice.
Unicidad	Indica si el índice es único o no.
Operaciones	Descripción
Obtener Columnas Referenciadas	Método que permite al índice obtener información relevante sobre las columnas a las que referencia.

Columna Referenciada	
Esta clase representa una columna que está siendo referenciada por un constraint de la base de datos.	
Atributo	Descripción
Nombre de la Columna	Nombre de la columna referenciada.
Orden	Orden de la columna dentro de la definición de la referencia.

Columna Indexada	
Esta clase representa una columna que está siendo referenciada por un índice de la base de datos. Hereda de la clase **Columna Referenciada**.	
Atributo	Descripción
Nombre del índice	Nombre del índice que referencia a esta columna.
Nombre de la tabla	Nombre de la tabla apuntada por el índice.
Es Ascendente	Indica si el índice se definió para esta columna para ordenar los datos de manera ascendente o descendente.

Constraint	
Esta clase abstracta representa un constraint de la base de datos. Implementa la interface **ObjetoBD**.	
Atributo	Descripción
Nombre del	Nombre del constraint.

constraint	
Nombre de la tabla	Nombre de la tabla referenciada por el constraint.
Nombre del esquema	Nombre del usuario creador del constraint.

Operaciones	Descripción
Obtener Columnas Referenciadas	Método por el cual el constraint puede obtener información relevante sobre las columnas que está referenciado sobre la tabla referenciada.

Llave Primaria
Esta clase representa la llave primaria de una tabla. Hereda de la clase abstracta **Constraint**.

Llave Foránea	
Esta clase representa una llave foránea de una tabla de la base de datos. Hereda de la clase **Constraint**.	
Atributo	Descripción
Está habilitado	Indica si la llave foránea está habilitada.
Nombre de la tabla referenciada.	Nombre de la tabla que referencia la llave foránea.
Tiene acción en cascada al actualizar	Indica si al actualizar la información de la tabla, la llave actualiza además las referencias a la tabla que definió la llave.
Tiene acción en cascada al eliminar	Indica si al eliminar la información de la tabla, la llave elimina toda referencia hacia la tabla que definió a la llave.

Check constraint	
Esta clase representa un check constraint de la base de datos. Hereda de la clase abstracta **Constraint**.	
Atributo	Descripción
Regla del constraint	Representa la regla o validación que realiza el constraint sobre la tabla referente.
Está habilitado	Indica si el constraint está habilitado.

Trigger

Esta clase representa un trigger de una tabla de la base de datos. Implementa la interface **ObjetoBD**.	
Atributo	Descripción
Nombre del Trigger	Nombre del trigger.
Nombre del Esquema	Nombre del esquema al que pertenece el trigger.
Tiempo de disparo	Indica si el trigger se dispara antes o después de un evento.
Se dispara por INSERT	Indica si el trigger se dispara por una sentencia INSERT.
Se dispara por UPDATE	Indica si el trigger se dispara por una sentencia UPDATE.
Se disparar por DELETE	Indica si el trigger se dispara por una sentencia DELETE.

Procedimiento Almacenado	
Esta clase representa un procedimiento almacenado de la base de datos. Implementa la interface **ObjetoBD**.	
Atributo	Descripción
Nombre del Procedimiento	Nombre del procedimiento almacenado.
Nombre del Esquema	Nombre del esquema al que pertenece el procedimiento.

Parámetro	
Esta clase representa un parámetro de un procedimiento almacenado. Hereda de la clase **Columna**.	
Atributo	Descripción
Tipo de parámetro	Representa el tipo de parámetro. (IN, OUT, IN/OUT)

ConexiónBD	
Esta clase representa la conexión a una base de datos.	
Atributo	Descripción
Conexión SQL	Representa la conexión a una base de datos. Este campo es proprio de la librería de conexión a bases de datos del entorno a usar.
Operaciones	Descripción

Crear conexión a Base de Datos	Método por el cual se crea una conexión a una base de datos.
Ejecutar sentencia	Método que permite ejecutar una sentencia DML hacia una tabla de la base de datos.
Ejecutar sentencia DDL	Método que permite ejecutar una sentencia DDL para modificar la estructura de la base de datos.
Cerrar conexión a Base de Datos	Método que permite cerrar una conexión a una base de datos.

ParámetroConexión	
Esta clase representa un parámetro que se enlazará en tiempo de ejecución dentro de una sentencia DML.	
Atributo	Descripción
Nombre del Parámetro	Representa el nombre del parámetro a utilizar. En el caso de la sentencias SELECT representa el nombre de la propiedad de un objeto, cuyo valor será obtenido en tiempo de ejecución.
Tipo de Parámetro	Representa el tipo de parámetro utilizado. Los valores permitidos son: Cadena, Entero, Decimal y Fecha.
Valor de Parámetro	Representa el valor del parámetro obtenido en tiempo de ejecución al invocarse a la propiedad del objeto especificado.

Configuración DBMS	
Esta clase representa el conjunto de parámetros de configuración de un DBMS.	
Atributo	Descripción
Nombre del DBMS	Representa el nombre del DBMS (Por ejemplo Oracle, MySql).
Archivo de la Plantilla	Representa la ubicación física del archivo de la plantilla del DBMS.
Nombre del Driver	Representa el nombre del Driver necesario para la configuración de la conexión a la base de datos.
Lista de Parámetros	Representa una lista con los valores de los parámetros necesarios para una conexión a la base de datos.
Campos obligatorios	Representa la lista de los parámetros obligatorios para la conexión.

Formato de cadena de conexión	Representa una cadena con el formato necesario para establecer la conexión a una base de datos.
Nombre de los parámetros	Representa una lista con los nombre de los parámetros.
Parámetro del Nombre de la base de datos	Representa el nombre del parámetro que contiene el nombre de la base de datos.
Ruta de la librería para conexión al DBMS	Representa la ubicación física de la librería necesaria para conectarse a un DBMS en particular.

Operaciones	Descripción
Armar Cadena de Conexión	Método que se encarga de armar la cadena de conexión con los parámetros especificados.
Armar Propiedades de Conexión	Método que se encarga de armar una serie de propiedades adicionales para la conexión a la base de datos.

Lectura Configuración	
Esta clase contiene métodos estáticos para la lectura del archivo de configuración de la aplicación.	
Operaciones	Descripción
Leer Archivo de Configuración	Método que se encarga de leer las entradas del archivo de configuración.
Leer Datos del DBMS	Método que se encarga de leer los datos de un DBMS dentro del archivo de configuración. Este método es llamado por el método anterior por cada entrada de un DBMS encontrado.

Plantilla XML	
Esta clase representa una plantilla en forma de archivo XML que contiene la información necesaria para extraer la metadata, equivalencia de tipos de datos y sentencias DDL de un DBMS. En resumen se encarga de proveer la abstracción del DBMS a los objetos de negocio.	
Atributo	Descripción
Plantilla XML	Representa el archivo físico XML.
Separador de Alias	Representa la cadena utilizada por el DBMS para usar

	'Alias' dentro de una sentencia SELECT.
Tipo Comilla	Representa el tipo de comilla utilizado para la definición de 'Alias'.
Operador de Outer Join	Representa el operador utilizado para realizar Outer Joins.
Utiliza Esquemas	Indica si el DBMS soporta el uso de esquemas.
Operador Diferente	Representa el operador utilizado para expresar una desigualdad dentro de un WHERE.

Operaciones	Descripción
Leer Metadata de Esquemas	Método que se encarga de devolver la sentencia SQL necesaria para extraer el nombre de los esquemas de una base de datos.
Leer Metadata de Tablas	Método que se encarga de devolver la sentencia SQL necesaria para extraer el nombre de las tablas de una base de datos.
Leer Metadata de Columnas	Método que se encarga de devolver la sentencia SQL necesaria para extraer los datos de las columnas de una tabla.
Leer Metadata de Índices	Método que se encarga de devolver la sentencia SQL necesaria para extraer los datos de un índice de una tabla.
Leer Metadata Columnas de Índices	Método que se encarga de devolver la sentencia SQL necesaria para extraer los datos las columnas de un índice.
Leer Metadata Llave Primaria	Método que se encarga de devolver la sentencia SQL necesaria para extraer los datos de una llave primaria.
Leer Metadata Columnas de Llave Primaria	Método que se encarga de devolver la sentencia SQL necesaria para extraer los datos de las columnas de una llave primaria.
Leer Metadata Llave Foránea	Método que se encarga de devolver la sentencia SQL necesaria para extraer los datos de las llave foráneas.
Leer Metadata Columnas de Llaves Foráneas	Método que se encarga de devolver la sentencia SQL necesaria para extraer los datos de las columnas de una llave foránea.

Leer Metadata Check Constraint	Método que se encarga de devolver la sentencia SQL necesaria para extraer los datos de los check constraints.
Leer Metadata Columnas de Check Constraints	Método que se encarga de devolver la sentencia SQL necesaria para extraer los datos de las columnas de una check constraint.
Leer Metadata de Vistas	Método que se encarga de devolver la sentencia SQL necesaria para extraer el nombre y la definición de las vistas de una base de datos.
Leer Metadata de Columnas de Vistas	Método que se encarga de devolver la sentencia SQL necesaria para extraer los datos de las columnas de una vista.
Leer Metadata de Índices de Vistas	Método que se encarga de devolver la sentencia SQL necesaria para extraer los datos de un índice de una vista.
Leer Metadata Columnas de Índices de Vistas	Método que se encarga de devolver la sentencia SQL necesaria para extraer los datos las columnas de un índice de una vista.
Leer Metadata de Triggers	Método que se encarga de devolver la sentencia SQL necesaria para extraer los datos de la declaración de un trigger.
Leer Metadata de Procedimientos	Método que se encarga de devolver la sentencia SQL necesaria para extraer los datos de un procedimiento almacenado.
Leer Metadata de Parámetros de Procedimientos	Método que se encarga de devolver la sentencia SQL necesaria para extraer los datos de un parámetro de un procedimiento almacenado.
Armar Variables de Sentencia	Método que se encarga de armar las variables dentro de una sentencia SQL.
Armar Columnas SELECT	Método que se encarga de armar la parte del SELECT de una consulta SQL.
Armar FROM	Método que se encarga de armar la parte del FROM de una consulta SQL.
Armar WHERE	Método que se encarga de armar la parte del WHERE de una consulta SQL.
Armar ORDER BY	Método que se encarga de armar la parte del

	ORDER BY de una consulta SQL.
Devolver Método de un Objeto	Método encargado de devolver en tiempo de ejecución el método de un objeto dado un nombre.
Devolver Equivalencia Tipo de Dato	Método encargado de devolver el equivalente de un tipo de datos de acuerdo al DBMS pasado como parámetro.
Leer Formato Sincronizar Columnas	Método encargado de devolver la sentencia DDL de sincronización de columnas de acuerdo al DBMS pasado como parámetro.
Leer Formato Sincronizar Índices	Método encargado de devolver la sentencia DDL de sincronización de índices de acuerdo al DBMS pasado como parámetro.
Leer Formato Sincronizar Llave Primaria	Método encargado de devolver la sentencia DDL de sincronización de llaves primarias de acuerdo al DBMS pasado como parámetro.
Leer Formato Sincronizar Llave Foránea	Método encargado de devolver la sentencia DDL de sincronización de llaves foráneas de acuerdo al DBMS pasado como parámetro.
Leer Formato Sincronizar Check Constraint	Método encargado de devolver la sentencia DDL de sincronización de check constraints de acuerdo al DBMS pasado como parámetro.
Leer Formato Sincronizar Índices Vistas	Método encargado de devolver la sentencia DDL de sincronización de índices de vistas de acuerdo al DBMS pasado como parámetro.
Leer Formato Sincronizar Triggers	Método encargado de devolver la sentencia DDL de sincronización de triggers de acuerdo al DBMS pasado como parámetro.
Leer Formato Sincronizar Procedimientos	Método encargado de devolver la sentencia DDL de sincronización de procedimientos de acuerdo al DBMS pasado como parámetro.