FACULTAD DE CIENCIAS DE LA INGENIERÍA

ESCUELA DE INFORMÁTICA

CARRERA DE INGENIERÍA EN SISTEMAS

Proyecto de Investigación previo a la obtención del título de Ingeniero en Sistemas.

Título del Proyecto de Investigación:

"DETECCIÓN Y CORRECCIÓN DE INCONSISTENCIAS EN ESQUEMAS LÓGICOS DE BASE DE DATOS"

Autora:

Pin Zamora Angélica Claribel

Director de Proyecto de Investigación:

Phd. Puris Cáceres Amilkar Yudier.

Quevedo – Los Ríos - Ecuador

2016

DECLRACIÓN DE AUTORÍA Y CESIÓN DE DERECHOS

Yo, **Pin Zamora Angélica Claribel**, declaro que la investigación aquí descrita es de mi autoría; que no ha sido previamente presentado para ningún agrado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyan en este documento.

La Universidad Técnica Estatal de Quevedo, puede hacer uso de los derechos correspondiente a este documento, según lo establecido por la ley de propiedad intelectual, por su reglamento y por la normativa institucional vigente.

Autor

Pin Zamora Angélica Claribel

CERTIFICACIÓN

El suscrito **Phd. Amilkar Yudier Puris Cáceres** Docente de la Universidad Técnica Estatal de Quevedo, certifica que la estudiante Sra. **Pin Zamora Angélica Claribel**, realizó el Proyecto de investigación titulado: **"DETECCIÓN Y CORRECCIÓN DE INCONSISTENCIAS EN ESQUEMAS LÓGICOS DE BASE DE DATOS",** previo a la obtención del Título de Ingeniero en Sistemas, bajo mi dirección, habiendo cumplido con las disposiciones reglamentarias establecidas para el efecto.

Phd. Puris Cáceres Amilkar Yudier
DIRECTOR DEL PROYECTO DE INVESTIGACIÓN

FACULTAD DE CIENCIAS DE LA INGENIERÍA

ESCUELA DE INFORMÁTICA

CARRERA DE INGENIERÍA EN SISTEMAS

PROYECTO DE INVESTIGACIÓN

TITULO:

"DETECCIÓN Y CORRECCIÓN DE INCONSISTENCIAS EN ESQUEMAS LÓGICOS DE BASE DE DATOS"

Presentado a la Comisión Académica como requisito previo a la obtención del título de Ingeniero en Sistemas.

Aprobado por:

_____

Presidente del Tribunal

Ing. Carlos Márquez de la Plata

_____ _____

Miembro del Tribunal

Ing. Iván Jaramillo Chuqui

Miembro del Tribunal

Phd. Pavel Novoa Hernández

QUEVEDO - LOS RÍOS - ECUADOR

2016

AGRADECIMIENTO

Esta tesis se la dedico a mi Dios quién supo guiarme por el buen camino, proporcionándome fuerzas para seguir adelante y no desmayar ante las adversidades que se me presentaron durante el camino.

A mi madre, pilar fundamental en mi vida, por sus consejos, comprensión, amor, ayuda en los momentos difíciles.

A mi esposo por impulsarme a concluir esta meta brindándome su apoyo incondicional, mis hermanos por estar siempre presentes, acompañándome para poderme realizar.

Y en especial a mis hijos que son el motor fundamental y las ganas de salir adelante día a día para ofrecerles un mañana mejor.

A mi familia en general porque siempre creyeron en mí, en que todo lo que me propusiera lo lograría.

DEDICATORIA

Primero quiero agradecer a Dios por brindarme sabiduría y haberme guiado para concluir un peldaño más de mi meta. En segundo lugar a mi madre, mujer emprendedora y llena de temple que supo guiarme en mi diario caminar.

A mi esposo e hijos que son mi soporte, mi apoyo, mi luz, mi guía, mi todo para seguir adelante y no bajar los brazos en los momentos difíciles; a cada uno de los que son parte de mi familia porque todos han aportado con un granito de arena a mi formación.

RESUMEN Y PALABRAS CLAVES

En el presente proyecto de investigación se plantea una propuesta de una herramienta para la detección y corrección de inconsistencias en esquemas lógicos de base de datos. En el cual se desarrolló mediante el análisis de cada uno de los métodos existentes para la detección de restricciones de cardinalidad inconsistentes en el esquema conceptual de una base de datos, donde se logró determinar una de las inconsistencias que se puede presentar y corregir mediante la implementación del algoritmo llamado "inconsistencia de referencia cíclicas", razón por la cual se considera posible la realización de esta nueva propuesta.

Una vez determinado el tipo de inconsistencia a solucionar con la implementación del algoritmo se procedió a buscar la complejidad computacional donde se determinó a n como la cantidad de nodos, m la cantidad de aristas y c la cantidad de ciclo.

El algoritmo de detección y corrección de inconsistencias lógicas de base de datos fue implementando en una herramienta ERECASE, el cual permitió realizar las pruebas para validar el correcto funcionamiento del algoritmo.

Palabras Claves: Inconsistencias, Restricciones de cardinalidad, complejidad computacional

ABSTRACT AND KEYWORDS

The present research project proposes a proposal of a tool for the detection and correction of inconsistencies in logical database schemas. In which it developed through the analysis of each of the existing methods for the detection of inconsistent cardinality restrictions in the conceptual schema of a database, where it managed to determine one of the inconsistencies that can occur and correct through the implementation of the algorithm called "inconsistency of cyclical reference", reason why is considered possible the realization of this new proposal.

Once determined the type of inconsistency to solve with the implementation of the algorithm we proceeded to search the computational complexity where determined to n as the number of nodes, m the number of edges, and c the amount of cycle.

The algorithm of detecting and correcting logical inconsistencies of database was implemented in an ERECASE tool, which allowed testing to validate the proper operation of the algorithm.

Keywords: inconsistencies, constraints of cardinality, computational complexity

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CÓDIGO DUBLING

Título:	"DETECCIÓN Y CORRECCIÓN DE INCONSISTENCIAS EN ESQUEMAS LÓGICOS DE BASE DE DATOS"
Autor:	Pin Zamora Angélica Claribel
Palabras Claves:	Base de Datos, Algoritmos, Inconsistencias de base de datos
Fecha de Publicación:	
Editorial:	
Resumen:	En el presente proyecto de investigación se plantea una propuesta de una herramienta para la detección y corrección de inconsistencias en esquemas lógicos de base de datos. En el cual se desarrolló mediante el análisis de cada uno de los métodos existentes para la detección de restricciones de cardinalidad inconsistentes en el esquema conceptual de una base de datos, donde se logró determinar uno de las inconsistencias que se puede presentar y corregir mediante la implementación del algoritmo llamado "inconsistencia de referencia cíclicas", razón por la cual se considera posible la realización de esta nueva propuesta. El algoritmo de detección y corrección de inconsistencias lógicas de base de datos fue implementando en una herramienta ERECASE, el cual permitió realizar las pruebas para validar el correcto funcionamiento del algoritmo.
Descripción:	67 hojas : dimensiones 21cm x 29.7cm
URI:	

INTRODUCCIÓN

En los últimos tiempos ha existido un gran desarrollo en las aplicaciones de Base de Datos, por esto su diseño se ha convertido en una actividad muy común realizada por cualquier persona aunque no siempre con la preparación necesaria. El diseño de bases de datos está fuertemente influido por la elección de un modelo de datos adecuado para representar los datos. Un modelo de datos es una serie de conceptos que pueden utilizarse para describir un conjunto de datos y operaciones para manipular los mismos. Cuando un modelo de datos describe un conjunto de conceptos de una realidad determinada, se llama modelo conceptual de datos.

Se han propuesto varios modelos conceptuales como alternativas, entre estos se encuentran los modelos semánticos de datos, el modelo estructurado, el modelo funcional y los modelos binarios. Todos estos modelos conceptuales están basados en el uso de mecanismo de abstracción por lo cual es posible definir correspondencias entre ellos. En particular el modelo Entidad- Relación (ER, Entity-Relationship) emergió como la principal estructura formal para la representación conceptual de datos, convirtiéndose y estableciéndote como estándar internacional.

En los intentos de dar un enfoque más sistemático a la resolución de problemas de modelación se ha enfatizado en la automatización mediante el empleo de herramientas CASE(Computer Aided Software Engeneering). Si bien es cierto que algunas de ellas adoptan enfoques avanzados también lo es que muchas no pasan de ser simples utilidades para dibujar. En general no disponen siquiera de un soporte metodológico, o no son lo suficientemente estricta para su aplicación, con lo cual el diseñador no encuentra el camino correcto para realizar su tarea. Además los modelos que generalmente soportan son modelos lógicos, que suelen incluir bastante consideraciones físicas, aunque la notación gráfica empleada se un dialecto del modelo ER. En este trabajo se presenta un herramienta CASE, nombrada ERECASE, que contiene un conjunto de construcciones de este modelo para lograr una mejor expresividad en el diagrama, la realización de validaciones en los esquemas lógico y conceptual, la transformación a un modelo relacional partiendo de un esquema conceptual, y la generación de código SQL para la creación de la base de datos en un Sistema de Gestión de Base de datos (SGBD) determinado.

CAPÍTULO I

CONTEXTUALIZACIÓN DE LA INVESTIGACIÓN

1.1. Problema de Investigación

1.1.1. Planteamiento del Problema

El diseño de una base de datos es un proceso, que a pesar de la experiencia el diseñador y del empleo de herramientas CASE (Ingeniería asistida por computadora), no está exento de errores e insuficiencias, lo que puede poner en peligro la calidad de los proyectos generados, en particular del esquema físico. Por tanto es conveniente detectar estos errores e insuficiencias en las fases iniciales del diseño.

La inconsistencia en los SGBD es muy común en los modelos actuales debido a la redundancia de datos, dificultando el acceso a datos y la obtención de la información de forma inmediata, provocando que el software que utiliza la información tenga problemas de eficiencia y eficacia.

Algunos de estos métodos sólo deciden si el conjunto de limitaciones relacionada con el esquema conceptual puede cumplirse o no, otros métodos permiten determinar los conjuntos de limitaciones de cardinalidad que hacen que el esquema conceptual sea inconsistente y en algunos casos se plantea al diseñador un plan de posibles correcciones a realizar en el esquema conceptual. Con relación a la detección de inconsistencias en esquemas lógicos.

1.1.2. Formulación del Problema

¿Cómo detectar y corregir inconsistencias en el esquema lógico de base de datos?

1.1.3. Sistematización del Problema

- ¿Qué tipo de inconsistencias se encuentran en el diseño de un esquema lógico?
- ¿Cómo detectar las inconsistencias en el esquema lógico?
- ¿Qué complejidad temporal tiene el algoritmo implementado?

1.2. Objetivos

1.2.1. General

Desarrollar una herramienta informática utilizando un algoritmo que permita la detección y corrección de inconsistencias en el esquema lógico de la base de datos.

1.2.2. Específicos

- Analizar los métodos existentes para detección de restricciones de cardinalidad inconsistentes en el esquema conceptual de una base de datos.
- Determinar la complejidad computacional del algoritmo que permita la corrección de inconsistencias en un esquema lógico de base de datos.
- Implementar el algoritmo de detección y corrección de inconsistencias en la herramienta ERECASE.

1.3. Justificación

Los resultados de esta investigación serán incluidos en una herramienta de diseño de bases datos, y que permitirá dotar a dicha herramienta de la capacidad de detectar y corregir inconsistencias en el esquema lógico de una base de datos, logrando obtener un esquema lógico sin inconsistencias y por consiguiente sin problemas de implementación.

A diferencia de otras herramientas similares, posee las siguientes características. Una variedad significativa de construcciones del modelo Entidad – Relación que permite una mayor expresividad en modelo conceptual. Entre estas construcciones se incluyen las entidades fuertes y débiles. Interrelaciones, validación estructural del esquema conceptual, detección e inconsistencia en el esquema lógico y generación del esquema físico para varios sistemas de base de datos relacionales. La herramienta va ser utilizada en la Facultad de Ciencias de la Ingeniería con los alumnos de la carrera de Ingeniería en Sistemas.

CAPÍTULO II

FUNDAMENTACIÓN TEÓRICA DE LA INVESTIGACIÓN

2.1.ESQUEMA REFERENCIAL DEL MARCO TEÓRICO

ALGORITMO

Es un conjunto de pasos lógicos y estructurados que permiten dar solución a un problema. [1]

La importancia de un algoritmo radica en desarrollar un razonamiento lógico matemático a través de la comprensión y aplicación de metodologías para la resolución de problemáticas. [1]

BASE DE DATOS

Evolucionaron a partir de sistemas de archivos para computadoras, aun cuando la administración de sistema de base de archivos se creó mucho antes, de modo que el sistema de base de datos ayuda a eliminar casi todas las diferencias de la administración de datos de un sistema de archivos. [2]

MODELO JERÀRJICO

Se desarrolló en el año 1960, para manejar grandes cantidades de datos para complejos proyectos de manofactura. Su estructura lógica básica está representado por un árbol invertido. [2]

SEGMENTO

Es el equivalente de un tipo de registro de un sistema de archivos. El modelo jerárquico describe un conjunto de relaciones de uno a muchos. [2]

PROGRAMAS PARA DESARROLLAR BASE DE DATOS

En el mercado existen ciertos programas que ayudan al desarrollador a generar código fuente, con lo que se puede ahorrar tiempo de aprendizaje y programación, pero dichos programas no son 100% compatibles con todas las bases de datos, no pueden generar código fuente para cualquier lenguaje de programación o simplemente generan demasiado código basura. Crear un programa que pueda generar código fuente para controlar las base de datos MySQL, SQL Server y Access, usando los lenguajes de programación A.S.P., J.S.P. y P.H.P [3]

ASP.NET

Es un modelo de desarrollo Web unificado que incluye los servicios necesarios para crear aplicaciones Web empresariales con el código mínimo. ASP.NET forma parte de .NET Framework y al codificar las aplicaciones ASP.NET tiene acceso a las clases en .NET Framework. El código de las aplicaciones puede escribirse en cualquier lenguaje compatible con el Common Language Runtime (CLR), entre ellos Microsoft Visual Basic, C#, JScript .NET y J#. Estos lenguajes permiten desarrollar aplicaciones ASP.NET que se benefician del Common Language Runtime, seguridad de tipos, herencia, etc. [4]

2.2. MARCO CONCEPTUAL

2.2.1. ALGORITMO

Un algoritmo es una serie de pasos finitos, definidos y organizados que describe el proceso que se debe seguir, para dar solución a un, problema especifico

EJECUTAR o probar un algoritmo es realizar las tareas que fijan esos pasos, en el orden especificado y utilizando los recursos. [5]

ESTRUCTURA DE UN ALGORITMO

Todo algoritmo consta de tres secciones principales:

- Entrada: Es la introducción de datos para ser transformados.
- Proceso: Es el conjunto de operaciones a realizar para dar solución al problema.
- Salida: Son los resultados obtenidos a través del proceso. [1]

ANÁLISIS DE UN ALGORITMO

- La característica básica que debe tener un algoritmo es que sea correcto, es decir, que produzca el resultado deseado en tiempo finito. Adicionalmente puede interesar que sea claro, que esté bien estructurado, que sea fácil de usar, que sea fácil de implementar y que sea eficiente. [6]

COSTE DE TIEMPO

- A la cantidad de tiempo que requiere la ejecución de un cierto algoritmo se le suele llamar coste en tiempo mientras que a la cantidad de memoria que requiere se le suele llamar coste en espacio. [6]

PRUEBA DE ESCRITORIO

- Es a la comprobación que se hace de un algoritmo para saber si está bien hecho. Esta prueba consiste en tomar datos específicos como entrada y seguir la secuencia indicada en el algoritmo hasta obtener un resultado, el análisis de estos resultados indicará si el algoritmo está correcto o si por el contrario hay necesidad de corregirlo o hacerle ajustes.
- Es Importante realizar una prueba de escritorio para eliminar variables no necesarias, crear variables faltantes, ocupar los ciclos adecuados. Y principalmente legar al objetivo del problema. [7]

Precisión

El algoritmo debe indicar el orden exacto de la ejecución de cada tarea. [5]

Definitud o determinismo

Si se sigue el algoritmo dos o más veces con los mismos datos de entrada, se debe obtener los mismos datos de salida. Si se utiliza un conjunto de datos análogos se obtienen resultados comparables y con la misma seguridad en su valor. [5]

Finitud

El algoritmo deber terminar en algún momento y debe usar una cantidad de recursos finita. Cuando se implementa un algoritmo en un computador digital, los recursos con los que se cuenta son tiempo de proceso y memoria. [5]

Siempre es necesario repasar el proceso que escribimos para resolver un problema, para hacer que sea económico es decir, que observe la economía de tiempo y de memoria para evitar pérdidas derivadas del mal uso de ellas. Un proceso es eficiente cuando tiene

la menor cantidad de pasos posibles, y el mejor uso de memoria (variables) para que, al ejecutarlo se obtenga la solución buscada. [5]

Hay algunas formas de escribir los algoritmos y facilitar luego la traducción a lenguajes de programación. Existen un conjunto de símbolos y reglas que se utilizan para describir de manera explícita un proceso. Hay formatos gráficos, para producir diagramas de seguimiento de un proceso (pueden ser diagramas de flujo o DFD, o diagramas estructurados, denominados de Chapín) y una forma de escribir las ordenes con palabras especialmente elegidas, que se llama pseudocódigo. [5]

Pasos

a) **Definición del problema**

Esta fase está representado por el enunciado del problema, si lo hay, que debe ser comprendido y delimitado, para lo cual necesita saber un concepto claro y preciso. con la siguiente etapa. Hay casos en que es necesario acotar el juego de datos, cuando no está especificado y puede anular el proceso posterior. Por ejemplo, si hay una división, el denominador debe acotarse a que sea diferente de cero, aunque el problema no lo especifique, para evitar un error y perder el control. [8]

b) **Análisis del problema**

Una vez que se ha comprendido el funcionamiento del computador, es necesario conocer: cuáles son los datos de entrada, cuál es la información que se desea visualizar (salida) y si hay métodos y/o fórmulas necesarios para procesar los datos. [5]

c) **Definición los pasos**

Después proceder a definir los pasos que se crean necesarios para resolver el problema, en forma ordenada, y precisa. [5]

Las características de un buen proceso de solución son:

- Debe haber un punto particular de inicio.
- Debe escribirse en pasos simples y de resolución única.
- El proceso completo tiene que ser definido, no debe permitir dobles interpretaciones ni ambigüedades.

- Debe ser general, es decir, soportar la mayor cantidad de variantes que se puedan presentar en la definición del problema.
- Debe ser finito (limitado) en tamaño y tiempo de ejecución.
- Debe poder arribarse a la solución solicitada.

Una vez conseguido el proceso, se ha armado el algoritmo, y debe escribirse en un lenguaje de programación para enviar las ordenes a un procesador. [5]

Los procesos pueden escribirse de diferentes formas facilitando la tarea. Una forma es un lenguaje similar al nuestro que se llama Pseudocódigo y usa algunas palabras especiales (escribir, ingresar, mostrar, etc.). También se usan gráficos como los llamados diagramas de Chapin, o los diagramas de Flujo que marcan el sentido en que se desarrolla el proceso. [5]

En este caso, se utilizará una herramienta denominada DFD que permite pasar estos diagramas a un proceso que los prueba tanto en su ejecución como en el tratamiento de sus variables y salidas. [5]

DIAGRAMA

Es un dibujo que marca las acciones, variables y sentido del proceso. DFD es un utilitario para probar los procesos sin escribir los programas en ningún lenguaje de programación. [5]

2.2.2. Base de datos

Una base de datos es un conjunto organizado de información. Se utilizará una base de datos para algo tan simple como llevar una agenda personal o para algo tan difícil como la gestión de una empresa. [9]

Las bases de datos aparecen en la década de los 70. Anteriormente, los programas debían crear y manipular unos sistemas de ficheros o archivos, mediante los cuales se almacenaba la información de forma estructurada. Este sistema presentaba bastantes inconvenientes: [9]

Redundancia e inconsistencia de los datos: Un sistema de ficheros está compuesto de un conjunto de ficheros individuales en los cuales es posible que se repita información. Por ejemplo, en un sistema de ficheros creado para almacenar los datos de los clientes de

una empresa, es posible que el teléfono de contacto de dicha empresa se repita en más de un fichero. Esta situación generaría dos inconvenientes. Por un lado, aumentaría la necesidad de almacenamiento y, por tanto, el coste del sistema. Por otro, si en algún momento se quiere cambiar ese dato, tendría que hacerlo en todos los ficheros en los que ese dato se encuentra almacenado. Con frecuencia, se olvida actualizar un dato en un sitio concreto y se producen inconsistencias en el sistema de ficheros. [9]

- Dependencia físico-lógica: Los sistemas de ficheros se diseñan para un conjunto de programas determinados. Al realizar cualquier cambio en la estructura de los ficheros que componen el sistema de ficheros, será necesario realizarlo en todos los programas que manejan dichos ficheros.
- No concurrencia: Es imposible que, en un sistema de ficheros, dos usuarios puedan modificar datos simultáneamente.
- Problemas con la seguridad de los datos: Es muy difícil implantar restricciones de seguridad en un sistema de archivos.
- Problemas de integridad: En los sistemas de ficheros es frecuente que se produzcan problemas de integridad. Por ejemplo, en un sistema de ficheros que controle las ventas de una empresa, se almacena en un archivo los datos de los clientes y, en otro, los pedidos realizados por dichos clientes. Se producirían problemas de integridad, si, por ejemplo, se da de alta un pedido para un cliente que no existe en el fichero correspondiente. Es difícil controlar que esto no suceda en un sistema de ficheros.

Para dar una definición más exacta de lo que es una base de datos se puede decir que es un conjunto de ficheros relacionados entre sí, cuyo contenido pueden compartir los usuarios, con características de integridad máxima y redundancia mínima. [9]

La integridad es un concepto bastante amplio que se debe definir para un sistema determinado. No obstante, de forma general, es posible especificar unas reglas que deben de cumplirse para que un sistema mantenga su integridad máxima: [9]

- No se puede relacionar un registro de un fichero con un registro inexistente de otro fichero. Es decir, no se podrá insertar un pedido de un cliente que no está dado de alta cómo cliente.

- A veces, tampoco es posible dejar un registro de un fichero sin relacionar con alguno del otro fichero. Es decir, en ocasiones, puede ser necesario que no exista ningún cliente en el sistema si este cliente no ha realizado aún algún pedido. Esta condición no siempre se cumple y depende de las especificaciones que hayan dado a la hora de diseñar el sistema.
- A la hora de hacer cambios en los distintos ficheros, las altas, las bajas o las modificaciones es necesario tener en cuenta las dos condiciones anteriores. Por ejemplo, si se necesita eliminar a un cliente de la base de datos, verificar qué pedidos tiene este cliente para eliminarlos también.

Redundancia significa repetición. Es inevitable que en una base de datos no se repita información, sin embargo, es necesario que esta repetición de información sea mínima. En el caso de los clientes y los pedidos, es estrictamente necesario que en cada pedido se incluya un dato del cliente que realiza ese pedido, cómo se comentó antes, lo ideal es el DNI. Lo que sería redundante es incluir más de un dato en esta relación. Es decir, si en el archivo del pedido se inserta, además del DNI, el nombre y apellidos del cliente o cualquier otra información, se incluye redundancia en el sistema. Es importante también, elegir el dato adecuado para relacionar ficheros. En el ejemplo que se está viendo, el dato más adecuado sería el DNI porque es único y porque está bien definido. Otros campos no están bien definidos, por ejemplo, el nombre y los apellidos de un cliente se escribe de varias maneras: utilizar mayúsculas y minúsculas, incluir más o menos espacios en blanco y hasta incluir abreviaturas. Si se elige los apellidos, por ejemplo, para relacionar dos ficheros y en uno de ellos estuvieran escritos con mayúsculas y, en el otro, con minúsculas, el sistema podría considerarlos distintos y provocar fallos en las aplicaciones. [9]

Otra de las características más representativas de las bases de datos es la compartición de datos. Es decir, las bases de datos están pensadas para ser utilizadas por varios usuarios, con frecuencia, al mismo tiempo. Para que esto sea posible es necesario establecer sistemas que permitan: [9]

- Gestionar los permisos: cada usuario sólo puede acceder y manipular aquellos datos para los que está autorizado.
- Gestionar la concurrencia: cuando dos usuarios realizan operaciones sobre una base de datos simultáneamente es necesario que el sistema evite que se

produzcan errores en la misma. Con frecuencia, lo que se hace es bloquear a uno de los usuarios hasta que el otro termine de operar.

Modelo entidad relación

Un diagrama o modelo entidad-relación es una herramienta para el modelado de datos de un sistema de información. Estos modelos expresan entidades relevantes para un sistema de información así como sus interrelaciones y propiedades. [10]

Entidades

Representa una “cosa” u "objeto" del mundo real con existencia independiente, es decir, se diferencia unívocamente de cualquier otro objeto o cosa, incluso siendo del mismo tipo, o una misma entidad. Una casa (Aunque sea exactamente igual a otra, aún se diferenciará en su dirección). Una entidad puede ser un objeto con existencia física como: una persona, un animal, una casa, etc. (entidad concreta), o un objeto con existencia conceptual como: un puesto de trabajo, una asignatura de clases, un nombre, etc. [10]

Conjunto de entidades

Las entidades que poseen las mismas propiedades forman conjunto de entidades. [11]

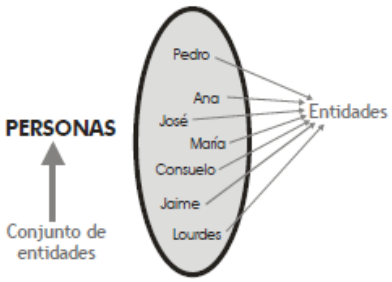

Figura 1. Ejemplo de entidad y conjunto de entidad

Representación gráfica de las entidades

En el modelo entidad relación

Tipos de entidades

- **Regulares.** Son las entidades normales que tienen existencia por sí mismas sin depender de otras. Su representación gráfica es la indicada arriba. [12]
- **Débiles.** Su existencia depende de otras. Por ejemplo la entidad tarea laboral sólo podrá tener existencia si existe la entidad trabajo. [12]

Figura 2. Entidad débil

Atributos

Los atributos son las propiedades que describen a cada entidad en un conjunto de entidades. Un conjunto de entidades dentro de una entidad, tiene valores específicos asignados para cada uno de sus atributos, de esta forma, es posible su identificación unívoca. [10]

Claves

Es un subconjunto del conjunto de atributos comunes en una colección de entidades, que permite identificar únicamente cada una de las entidades pertenecientes a dicha colección. [10]

Relaciones

Representan asociaciones entre entidades. Es el elemento del modelo que permite relacionar en sí los datos del modelo. Por ejemplo, en el caso de que se tenga una entidad personas y otra entidad trabajos. Ambas se realizan ya que las personas trabajan y los trabajos son realizados por personas: [12]

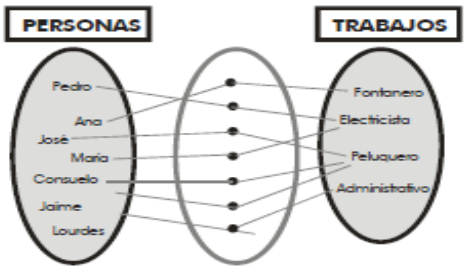

Figura 3. Ejemplo de relación

Representación gráfica

La representación gráfica de las entidades se realiza con un rombo al que se le unen líneas que se dirigen a las entidades, las relaciones tienen nombre (se suele usar un verbo). En el ejemplo anterior podría usarse como nombre de relación, trabajar. [12]

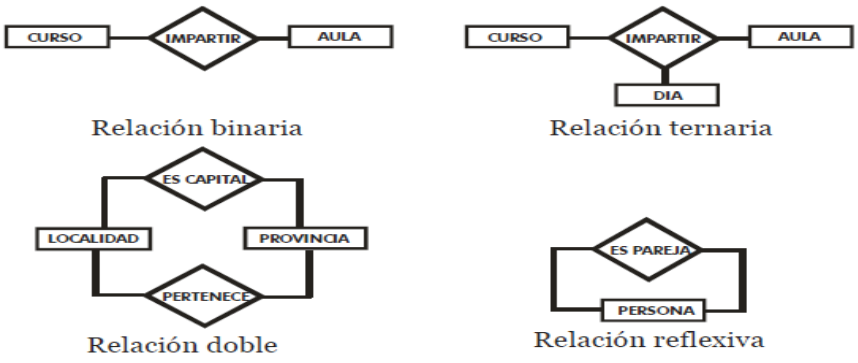

Figura 4. Ejemplos de relaciones

Identificador

Se trata de uno o más campos cuyos valores son únicos en cada ejemplar de una entidad. Se indican subrayando el nombre del identificador. [12]

Para que un atributo sea considerado un buen identificador tiene que cumplir:

1. Deben distinguir a cada ejemplar teniendo en cuenta las entidades que utiliza el modelo. No tiene que ser un identificador absoluto. [12]
2. Todos los ejemplares de una entidad deben tener el mismo identificador. [12]
3. Cuando un atributo es importante aun cuando no tenga una entidad concreta asociada, entonces se trata de una entidad y no de un atributo. [12]

Entidades isa

Son relaciones de tipo isa (es un) aquellas en las que una entidad se descompone en entidades especializadas. Hay dos tipos de entidades isa: especializaciones y generalizaciones. [12]

Las especializaciones consisten en que una entidad se divide en entidades más concretas. La entidad general comparte con las especializadas sus atributos. Se observa una especialización cuando hay ejemplares para los que no tienen sentido algunos de los atributos, mientras que para otros sí. [12]

Se denomina generalización si se agrupan varias entidades en una o más entidades generales. Se observa una generalización si en varias entidades se observan atributos iguales, lo que significa que hay una entidad superior que posee esos atributos. [12]

En cualquier caso la representación en el modelo es la misma, se representan con un triángulo que tiene el texto ISA. Ejemplo:

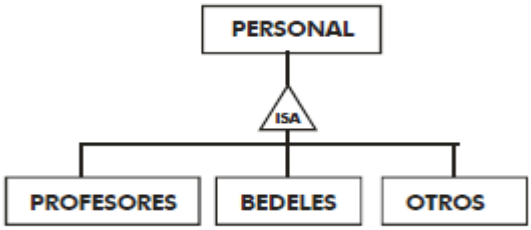

Figura 5. Entidades ISA

En estas relaciones se habla también de herencia, ya que tanto los profesores como los bedeles como los otros, heredan atributos de la entidad personal (se habla de la superentidad personal y de la subentidad profesores). [12]

Se puede colocar un círculo (como el del número cero) en lado de la superentidad para indicar que es opcional la especialización, de otro modo se tomará como obligatoria (el personal tiene que ser alguna de esas tres cosas). [12]

Se puede indicar también exclusividad. Esto ocurre cuando entre varias líneas hacia una relación, las entidades sólo pueden tomar una. Se representa con un ángulo en el diagrama:

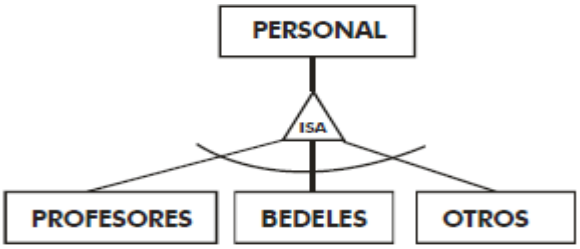

Figura 6. Entidades ISA con exclusividad

En el diagrama el ángulo indica que el personal sólo puede ser o profesor o bedel u otros. No puede ser dos cosas a la vez. [12]

Pasos para el diseño

1. Encontrar entidades (conjuntos de entidades)
2. Identificar atributos de las entidades
3. Buscar identificadores
4. Especificar las relaciones y cardinalidades
5. Identificar entidades débiles
6. Especializar y generalizar entidades donde sea posible

Inconsistencias en base de datos

Sólo se produce cuando existe redundancia de datos. La inconsistencia consiste en que no todas las copias redundantes contienen la misma información. Así, si existen diferentes modos de obtener la misma información, y esas formas pueden conducir a datos almacenados en distintos sitios. El problema surge al modificar esa información, si sólo cambiamos esos valores en algunos de los lugares en que se guardan, las consultas que hagamos más tarde podrán dar como resultado respuestas inconsistentes (es decir, diferentes). Puede darse el caso de que dos aplicaciones diferentes proporcionen resultados distintos para el mismo dato. [13]

Inconsistencias en esquemas lógicos

Para introducir el problema del tipo de inconsistencia que trata este artículo, se mostrarán varios ejemplos y se hará una caracterización del caso más general. Los diagramas Entidad Relación que se muestran utilizan la notación de Elmasri, en la cual el rectángulo representa los conjuntos de entidades y el rombo a los conjuntos de interrelaciones. La cardinalidad máxima de una interrelación se indica con un “1” o con la letra “M”. Para la cardinalidad mínima una línea simple indica participación opcional y una línea doble participación obligatoria de las entidades en la interrelación. En la notación de Elmasri la cardinalidad máxima y la cardinalidad mínima se expresan mediante la notación look across y look here respectivamente. La transformación del esquema conceptual al esquema lógico se hace siguiendo las reglas de transformación

de Elmasri, aunque se pudieran haber aplicado otras como las propuestas por Teorey, Jajodia, De Miguel o Ponniah. [14]

Inconsistencia en esquemas conceptual de base de datos

La transformación de esquemas conceptuales a esquemas lógicos llevada a cabo por herramientas de diseño de bases de datos puede traer como resultado esquemas lógicos que presenten algún tipo de inconsistencia y por consiguiente los esquemas físicos tendrán problemas de implementación. En este trabajo se analiza un tipo de inconsistencia (nombrada por los autores como “inconsistencia de referencias cíclicas”) que puede presentarse en el esquema lógico de la base de datos y se propone un algoritmo para su detección y corrección. Este algoritmo puede ser implementado en herramientas de diseño de bases de datos. [14]

En este sentido se han publicado numerosos trabajos cuyo objetivo es detectar inconsistencias en esquemas conceptuales basados en las restricciones de cardinalidad.

Algunos de estos métodos sólo deciden si el conjunto de restricciones asociado al esquema conceptual puede ser satisfecho o no, otros métodos permiten determinar los conjuntos de restricciones de cardinalidad que hacen que el esquema conceptual sea inconsistente y en algunos casos se propone al diseñador un plan de posibles correcciones a realizar en el esquema conceptual. [14]

Método existente para detección de restricciones de cardinalidad inconsistentes en el esquema conceptual de una base de datos.

Si las entidades A, B y C están relacionadas en una interrelación ternaria R, se puede determinar la cardinalidad para la participación de cada entidad en la interrelación. Para cada combinación particular de B y C, si existe sólo un valor de A, entonces A participa como una interrelación “uno”. Si puede haber más que un valor de A para una combinación particular B-C, entonces A participa como una interrelación “muchos”. Lo mismo ocurre con B y su interrelación A-C, y con C y su interrelación A-B. [14]

En un diagrama E/R, las líneas conectan los rectángulos de las entidades a los diamantes de las interrelaciones para mostrar sus asociaciones. [14]

Existen diversos métodos para mostrar la cardinalidad de la interrelación:

- **Método 1**, mostrar la cardinalidad poniendo 1, N o M sobre la línea que une la entidad a la interrelación, dependiendo del número máximo de entidades asociadas [15].

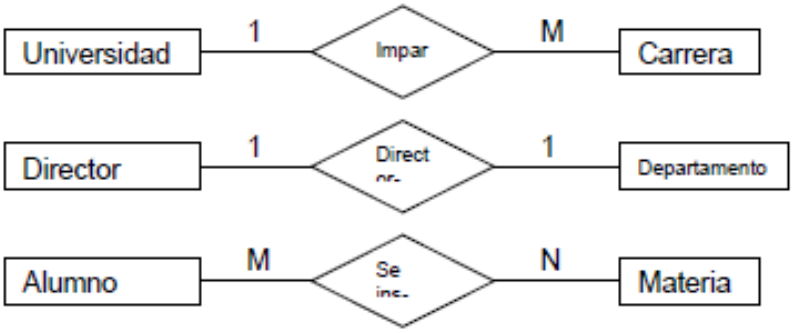

Figura 7. Ejemplo de Método 1

- **Método 2**, mostrar la cardinalidad con flecha simple o doble en la línea que une la entidad a la interrelación, dependiendo del número máximo de entidades asociadas [15].

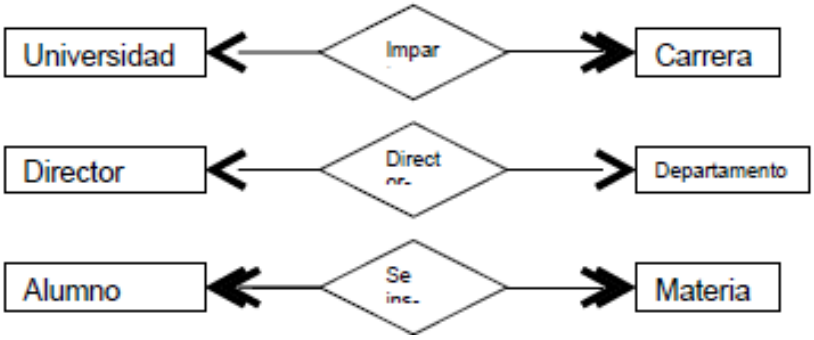

Figura 8. Ejemplo Método 2

- **Método 3**, mostrar la cardinalidad con o sin flecha en la línea que une la entidad a la interrelación, dependiendo del número máximo de entidades asociadas [15].

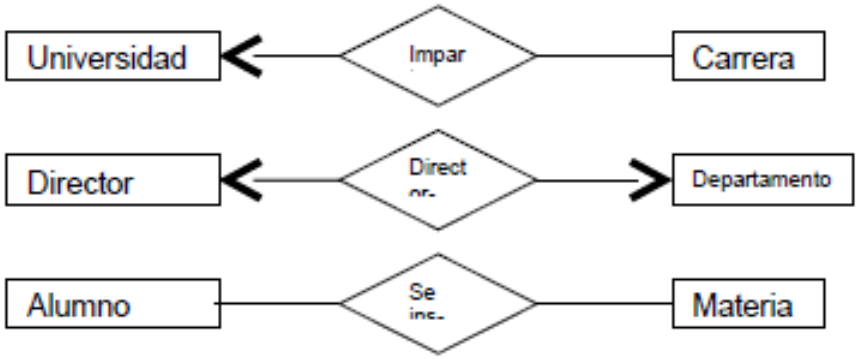

Figura 9. Ejemplo Método 3

- **Método 4**, mostrar la cardinalidad con o sin punto en la línea que une la entidad a la interrelación, dependiendo del número máximo de entidades asociadas [15].

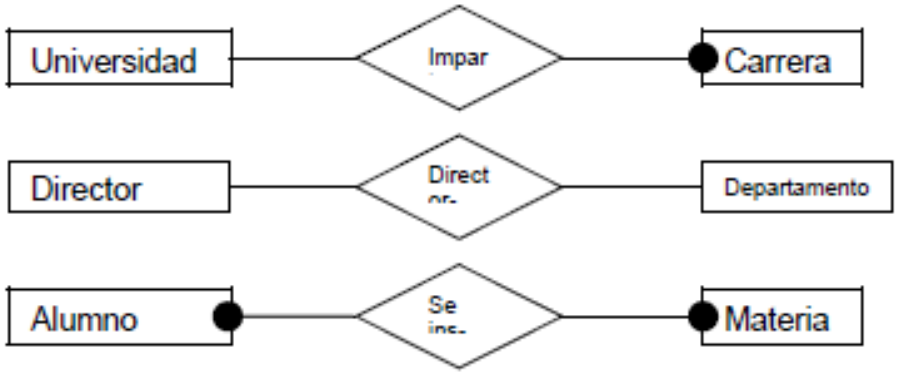

Figura 10. Ejemplo Método 4

- **Método 5**, mostrar la cardinalidad con o sin patas de gallo en la línea que une la entidad a la interrelación, dependiendo del número máximo de entidades asociadas [15].

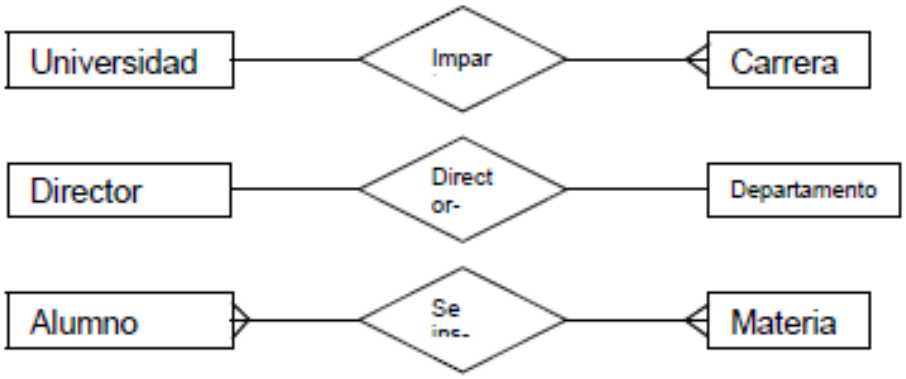

Figura 11. Ejemplo Método 5

Aunque estos no son los únicos métodos para representar las interrelaciones entre entidades, son de los métodos más empleados, sobre todo por herramientas CASE y de diseño de BD, existen también algunos métodos que no incluyen los diamantes, sólo las líneas que conectan a las entidades. [15]

2.3. MARCO REFERENCIAL

El proceso de diseño de una base de datos comienza con la ingeniería de requisitos. Una actividad clave durante este proceso es la creación de un esquema conceptual. Los esquemas conceptuales describen las propiedades que el sistema de información debe tener y sirve como base para las siguientes fases de su diseño. Aunque la modelación conceptual representa sólo una pequeña parte del desarrollo de un sistema, tiene un impacto importante en el resultado final, ya que los errores o insuficiencias que se cometen en esta fase son trasladados hacia las siguientes fases del diseño, lo que puede traer como consecuencia la obtención de esquemas físicos que presenten errores o inconsistencias. Por tanto, es deseable descubrir estos errores en las fases iniciales del diseño. En este sentido se han desarrollado métodos que permiten detectar inconsistencias en los esquemas conceptuales basados en las restricciones de cardinalidad.

CAPÍTULO III

METODOLOGÍA DE LA INVESTIGACIÓN

3.1. Localización

El algoritmo de detección y corrección de inconsistencias en esquemas lógicos de base de datos se llevará a cabo en la Universidad Técnica Estatal de Quevedo, provincia de Los Ríos.

3.2. Tipo de Investigación.

3.2.1. Bibliográfica – documental

Se realizó como su nombre su indica, apoyándose en fuentes de carácter documental, esto es, en documentos de cualquier especie tales como, las obtenidas a través de fuentes bibliográficas, basadas en la consulta de libros, artículos, revistas y periódicos, en documentos que se encuentran en archivos como cartas, oficios, circulares y expedientes.

3.2.2. Exploratoria

Este proyecto de investigación permitirá introducir una nueva propuesta dentro de los algoritmos para la detección de inconsistencias de base de datos puesto que el algoritmo para la detección y corrección de inconsistencias no había sido estudiado en este campo. Encontrando así mediante el estudio de otros algoritmos los procedimientos adecuados para desarrollar una nueva propuesta.

3.3. Métodos de Investigación

Para el desarrollo de este trabajo se utilizó el siguiente método de investigación:

- **Método deductivo.** se podrá deducir los problemas que originan las inconsistencias de base de datos, además identificará algunos factores claves que ayudarán a desarrollar las estrategias aplicables en esta investigación.
- **Método analítico.** permitirá interpretar una situación de una manera analítica-crítica y así poder identificar las relaciones inconsistencias y soluciones del objeto de estudio. Para de esta manera comprender mejor el objeto de estudio y lograr alcanzar el fin propuesto.

3.4. Fuentes de recopilación de información.

Para la recopilación de información de este proyecto de investigación se utilizaron fuentes de información secundaria puesto que dentro del campo de la de base de datos se han venido realizando varios estudios sobre las inconsistencias de base de datos ya existentes y el desarrollo de nuevas propuestas.

3.5. Diseño de la Investigación.

3.5.1. Diseño Cuasi-Experimental

Para el presente proyecto de investigación, se empleará el diseño cuasi-experimental una técnica de investigación la cual permite evaluar el impacto que tendrá el algoritmo de detección y corrección de errores en inconsistencias de base de datos.

Los cuasi-experimentos permiten realizar investigaciones dentro de un marco de restricciones, particularmente la falta de aleatorización. No obstante, y teniendo siempre presente la limitación en cuanto al valor predictivo de este tipo de estudios, las relaciones causales son valiosas porque proporcionan el conocimiento de cómo manipular nuestro mundo sistemáticamente.

3.6. Recursos Humanos y materiales.

Los recursos y materiales empleados durante este proyecto son:

Tabla 1. Recursos y materiales

Talento Humano	**Docentes** ✓ Phd. Amilkar Yudier Puris Cáceres	
	Investigadora y responsable de la Investigación ✓ Angélica Claribel Pin Zamora	
Recursos Físicos	Detalle	Costo
	✓Útiles de oficina	20,00
	✓Internet	30,00
	✓Fuentes de consulta	0,00
Recursos de Software	✓Microsoft Office	35,00
	✓Visual Studio 2010	570,00
	✓Adobe Master Collection CS5	2600,00
Recursos de Hardware	✓Computadoras	1000,00
	✓Impresora	300,00
	TOTAL	**4555,00**

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

4.1. Resultados

El objetivo principal es explicar la implementación del algoritmo de detección y corrección de inconsistencias, utilizando para dicho fin el diagrama de clases del módulo implementado, se desarrolla una explicación del algoritmo de forma general, descripción de las clases usadas en la implementación, el flujo de actividades del módulo lo que permite tener una mayor visión de cómo se desarrolla el algoritmo para la detección, al igual que la corrección luego que el usuario selecciona la opción de llevar a cabo la misma.

Se muestran imágenes de las diversas situaciones que se podrían presentar, es decir de los diversos tipos de ciclos que pueden surgir en el modelo relacional, mostrando en cada momento las acciones que puede realizar el usuario al encontrarse con las inconsistencias ya sea ignorar o corregirlas, el modelo resultante en cualquiera de las dos opciones, así como el código SQL obtenido.

4.1.1. Inconsistencias en el esquema lógico de una base de datos

Se utilizaron varios ejemplos de esquemas conceptuales como caso de estudios y fueron aplicados cada uno de los métodos discutidos en el capítulo anterior.

Se encontró que algunos esquemas conceptuales son coherentes, al ser transformado al esquema lógico correspondiente presentando un tipo de inconsistencia que da como resultado esquemas con problemas de aplicación. Esto llevó a un estudio para la caracterización de esta incoherencia y el desarrollo de un algoritmo que detecta y corrige, obteniendo así un esquema lógico coherente.

Para resolver el problema de las inconsistencias de base de datos, se plantearon diversos casos de estudio y se hizo el arreglo para requisitos particulares del caso más general. Se utilizaron diagramas de entidad-relación en la que el rectángulo representa los conjuntos de entidades y el diamante a los conjuntos de relaciones. donde la cardinalidad se indica mediante "1" o la letra "M" y la cardinalidad mínima para una sola línea indicando participación opcional y línea doble para una participación obligatoria de las entidades en la relación

En la Figura 12 se muestra un diagrama Entidad-Relación modela representando el hecho de que un supervisor puede vigilar a varios empleados y que un empleado no

puede supervisar a ninguna persona.

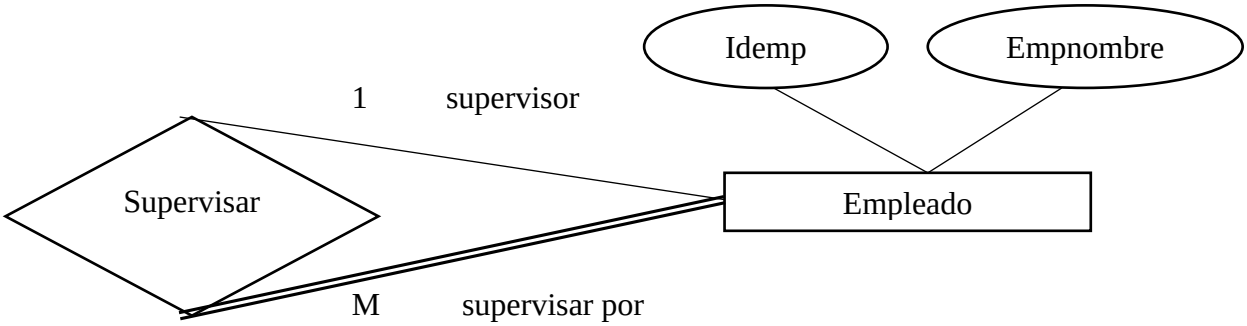

Figura 12. Interrelación recursiva en un diagrama ER.

Al transformar el esquema conceptual se logra obtener el siguiente esquema de relación, en el cual se adoptará que las llaves primarias estarán subrayadas y las llaves foránea s en cursivas:

EMPLEADO (IDEMP, EMPNOMBRE, *SUPERVISOR*)

En el esquema anterior la llave foránea SUPERVISOR no permite valores nulos debido a la participación obligatoria de la entidad EMPLEADO en la interrelación supervise. Al generar el código SQL correspondiente al esquema de relación EMPLEADO se obtiene lo siguiente:

```
CREATE TABLE EMPLEADO (
   IDEMP            INTEGER NOT NULL,
   EMPNOMBRE        CHAR(30),
   SUPERVISOR       INTEGER NOT NULL,
   CONSTRAINT       empleado_pk1        PRIMARY KEY(IDEMP),
   CONSTRAINT       empleado_fk1        FOREIGN KEY(SUPERVISOR)
   REFERENCES EMPLEADO (IDEMP));
```

Como se observa no es posible insertar un empleado sin antes haber ingresado al supervisor, por lo que se presenta un problema de implementación. Es importante destacar que el problema de implementación expuesto es consecuencia de un esquema de relación inconsistente.

En el diagrama de la Figura 13 se modelan los hechos de que los empleados tienen que pertenecer a un departamento y que un departamento tiene que ser dirigido por un

empleado.

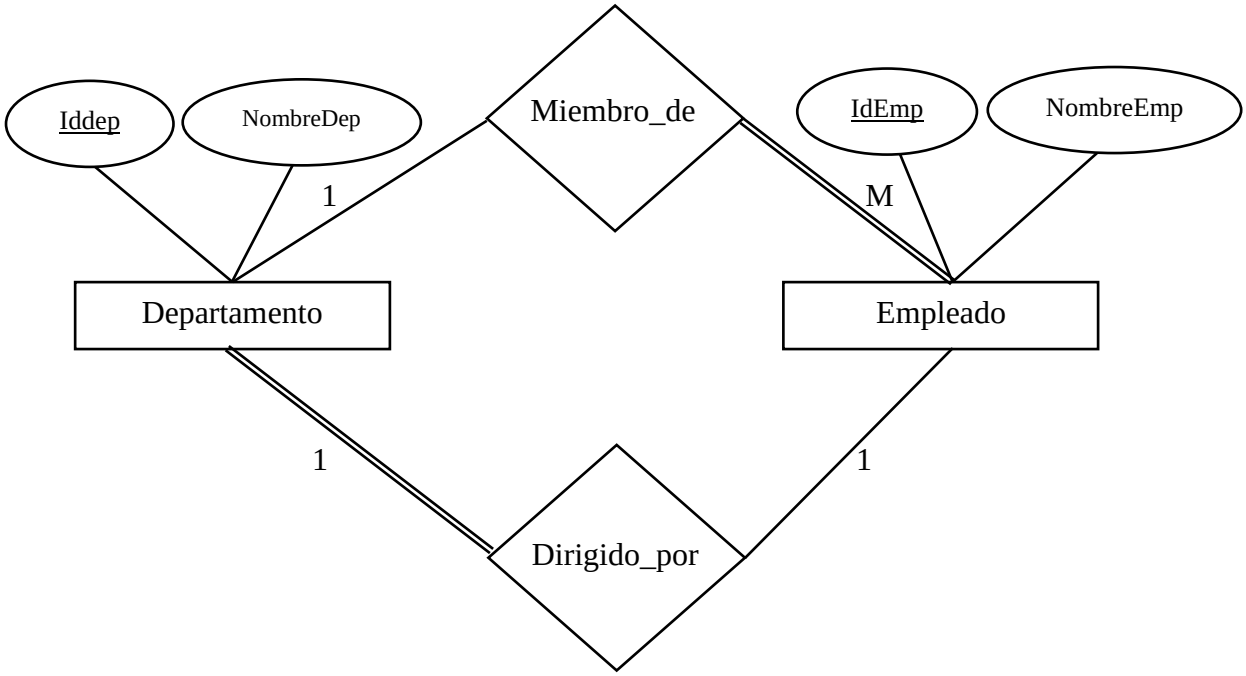

Figura 13. Diagrama ER que forma un ciclo entre dos entidades.

Al transformar el esquema conceptual partiendo del diagrama ER de la Figura 14 se logran obtener los siguientes esquemas de relación:

EMPLEADO (IDEMP, NOMBREEMP, *IDDE*)

DEPARTAMENTO (IDDEP, NOMBREDEP, *DIRECTOR*)

En este caso, debido a la participación obligatoria de los conjuntos entidad DEPARTAMENTO y EMPLEADO en las interrelaciones DIRIGIDO_POR y MEMBRO_DE respectivamente, las llaves foráneas correspondientes no permiten valores nulos, lo cual se expresa a través del siguiente código SQL:

```
CREATE TABLE EMPLEADO(
   IDEMP              INTEGER NOT NULL,
   NOMBREEMP          CHAR(30),
   IDDEP              INTEGER NOT NULL,
   CONSTRAINT     empleado_pk1        PRIMARY KEY(IDEMP),
   CONSTRAINT     empleado_fk1        FOREIGN KEY(IDDEP)
   REFERENCES DEPARTAMENTO(IDDEP));
CREATE TABLE DEPARTAMENTO(
```

```
IDDEP                   INTEGER NOT NULL,
NOMBREDEP                CHAR(30),
DIRECTOR                 INTEGER NOT NULL UNIQUE, CONSTRAINT
departamento_pk1        PRIMARY KEY(IDDEP), CONSTRAINT
departamento_fk1         FOREIGN KEY(DIRECTOR) REFERENCES
EMPLEADO(IDEMP));
```

Con la restricción de integridad NOT NULL sobre el atributo EMPID es necesario primero insertar el registro de un empleado en la tabla EMPLEADO; por otro lado con la restricción de integridad NOT NULL sobre el atributo IDDEP en la tabla EMPLEADO, no es posible insertar el registro de un empleado determinado a menos que se haya insertado el departamento al que pertenece. Como se puede observar hay una dependencia mutua que genera un conflicto que no permite la inserción registros a ninguna de las tablas. Este tipo de inconsistencia ocasiona un problema de diseño dado por la presencia de llaves foráneas con referencias mutuas y que no admiten valores nulos.

En la Figura 14 se diseñan los siguientes hechos: los empleados pertenecen a un departamento; un departamento financia un proyecto y un proyecto tiene que ser apoyado por un empleado.

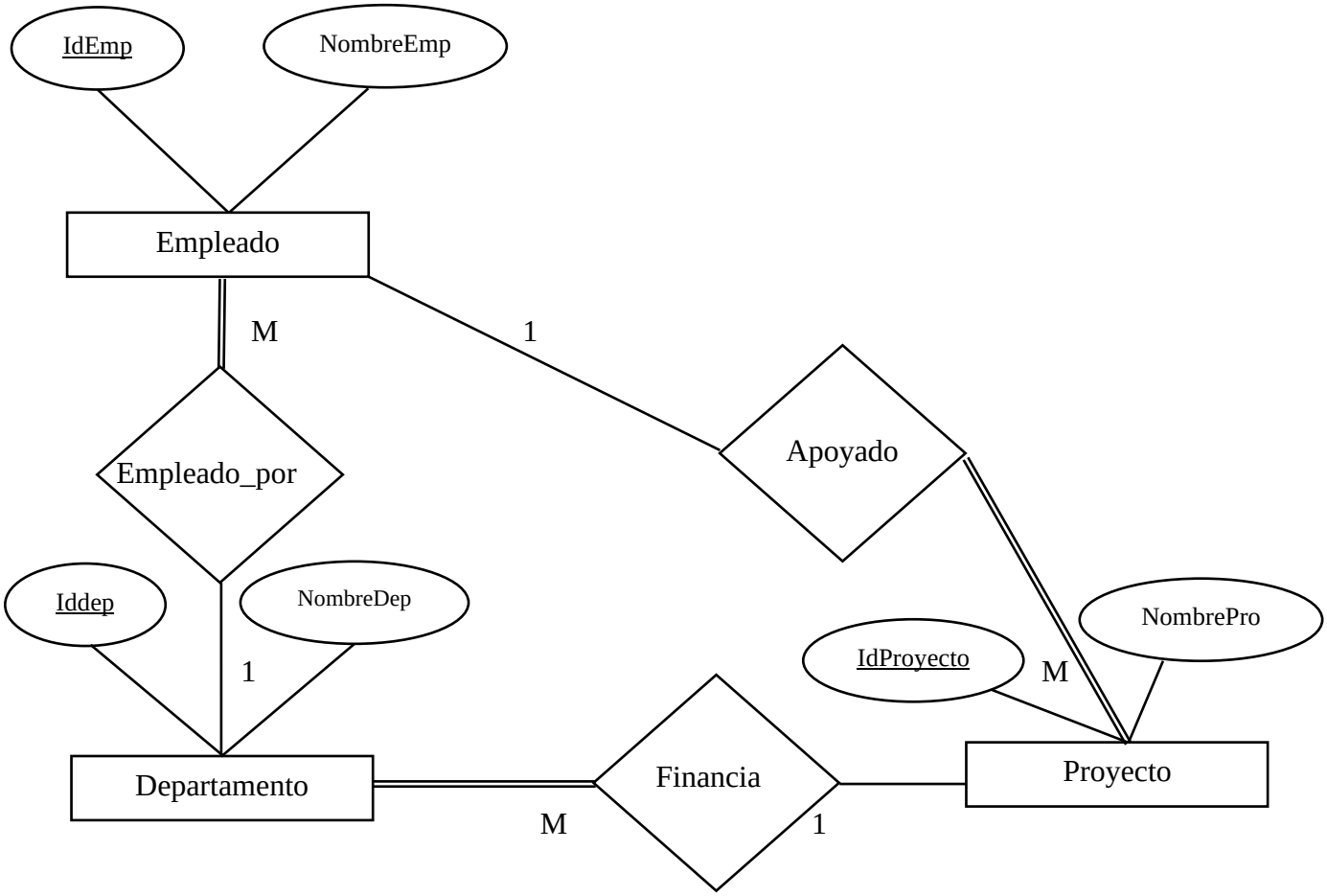

Figura 14. Diagrama ER de un ciclo entre tres conjuntos de entidades.

Transformando el esquema conceptual se obtienen los siguientes esquemas de relaciones:

EMPLEADO (IDEMP, NOMBREEMP, *IDDEP*)

DEPARTAMENTO (IDDEP, NOMBREDEP, *IDPROYECTO*)

PROYECTO (IDPROYECTO, NOMBREPRO, *IDEMP*)

Como se puede observar que las llaves foráneas de los esquemas obtenidos no admiten valores nulos debido a la participación necesaria del grupo de entidades del lado "mucho" de la interrelación.

```
CREATE TABLE EMPLEADO (
  IDEMP          INTEGER NOT NULL, NOMBREEMP   CHAR(30),
  IDDEP          INTEGER NOT NULL,
  CONSTRAINT     empleado_pk1      PRIMARY KEY(IDEMP),
  CONSTRAINT     empleado_fk1      FOREIGN KEY(IDDEP)
  REFERENCES DEPARTAMENTO(IDDEP));
```

```
CREATE TABLE DEPARTAMENTO (
   IDDEP          INTEGER NOT NULL, NOMBREDEP    CHAR(30),
   IDPROYECTO     INTEGER NOT NULL,
   CONSTRAINT     departamento_pk1      PRIMARY KEY(IDDEP),
   CONSTRAINT     departamento_fk1      FOREIGN KEY(IDPROYECTO)
   REFERENCES PROYECTO (IDPROYECTO));
CREATE TABLE PROYECTO (
   IDPROYECTO  INTEGER NOT NULL, NOMBREPRO    CHAR(30),
   IDEMP       INTEGER NOT NULL,
   CONSTRAINT     proyecto_pk1     PRIMARY KEY(IDPROYECTO),
   CONSTRAINT     proyecto_fk1     FOREIGN KEY(IDEMP) REFERENCES
   EMPLEADO (IDEMP));
```

Como se puede observar en el código SQL, las llaves foráneas de los esquemas existe una referencia cíclica entre las tablas y al mismo tiempo la restricción NOT NULL, de manera que no es posible insertar un registro en ninguna de las tres tablas, por lo que se presenta como un problema de implementación debido a la inconsistencias de esquemas.

Los ejemplos mostrados anteriormente permiten concluir que a pesar de que un diagrama ER sea estructuralmente válido este pudiera generar un esquema lógico con Inconsistencias en dependencia del conjunto de reglas de transformación que se le apliquen a los esquemas conceptuales.

4.1.2. Algoritmo de detección y corrección de inconsistencias en esquemas lógicos

4.1.2.1. Diseño e implementación del algoritmo.

Para la detección y corrección de inconsistencias se creó un módulo el cual consta de 11 clases como se puede ver en la Figura 15, las cuales están relacionadas entre sí de una u otra forma. Las clases fundamentales son TGrafo y TschemasCicles ya que es donde se concentra todo el trabajo con el grafo como estructura base y la detección y corrección de inconsistencias respectivamente. Las otras clases son utilizadas por las dos clases anteriormente mencionadas. Antes de explicar la

implementación de las clases se debe presentar el diagrama de clases que se diseñó para tal fin.

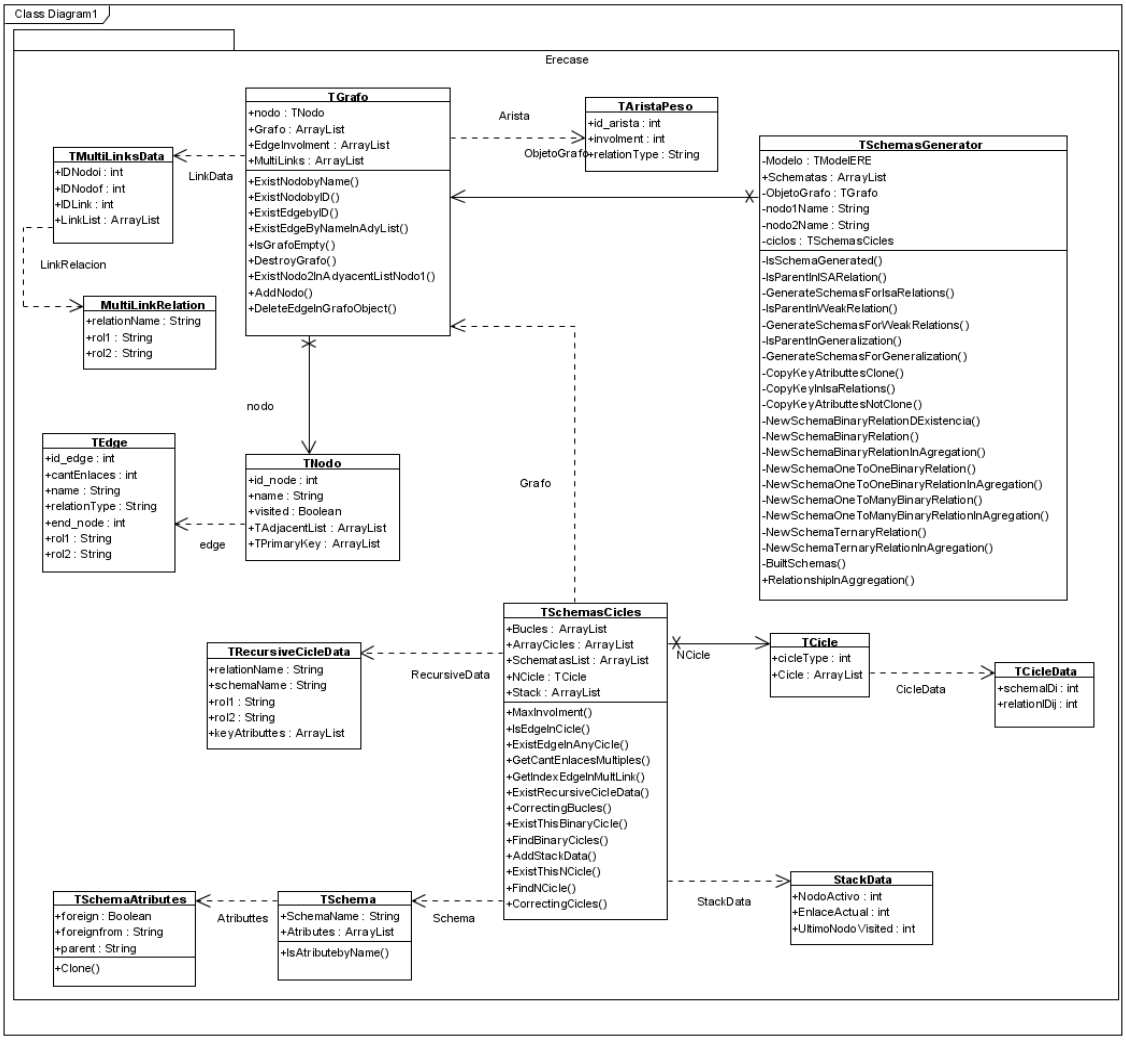

Figura 15. Diagrama de clases del módulo ValidationLogicalSchema

La clase TNodo es utilizada para almacenar los datos referentes al esquema del modelo lógico que representa cada nodo del grafo, entre los que están *nombre* del esquema, la lista de *atributos claves* del esquema, etc.

La clase TEdgees utilizada para almacenar los datos referentes a las arista, ya que la misma representa la relación existente entre dos esquemas del modelo lógico, aquí se guarda el *nombre* de la relación entre los dos nodos, el *nodo* final de la relación, etc.

La clase TmultiLinksData es utilizada para almacenar las relaciones múltiples entre dos nodos, es decir, cuando dos nodos presentan más de una relación en el mismo

sentido la primera relación se almacena en la clase TEdge pero las demás se etiquetan estableciendo que tengan el mismo *identificador* de arista al igual que el *involment*, con esto se optimiza el algoritmo ya que en los recorridos y en la corrección las relaciones múltiples en la misma dirección entre dos nodos son tratados como una sola relación.

La clase MultiLinkRelation tiene como atributo el arreglo de elementos del tipo TmultiLinksData lo cual permite tener todos los enlaces múltiples almacenados en una estructura de fácil manejo para la posterior corrección.

La clase TRecursiveCicleData es utilizada para guardar los datos necesarios de una relación recursiva, que luego son almacenados en la lista Bucles para la respectiva corrección.

La clase TCicleData es utilizada para guardar los datos necesarios de cada elemento que participa en un ciclo detectado como son el *nodo inicial y la arista* que sale de dicho nodo, luego son almacenados en la lista Tcicle.

La clase TCicle es utilizada para almacenar los elementos que participan en un ciclo el cual puede ser de longitud = 2 o longitud > 2, este ciclo se almacena en la lista *ArrayCicles* el cual es utilizado con posterioridad para la respectiva corrección.

La clase TAristaPeso es utilizada para almacenar el *identificador* y el *involment* de cada arista.

La clase TGrafo es utilizada para representar el grafo dirigido y contiene los métodos necesarios para el trabajo con el mismo. La estructura de datos usada para la representación del grafo dirigido es una lista de listas la cual es un atributo de la clase, en dicha estructura, la lista inicial es la que almacena todos los nodos, considerando los mismos como nodos iniciales de un enlace, almacenando en el mismo datos tales como el *nombre* del nodo, *identificador*, la *lista de atributos llaves*, la otra lista es la de adyacencia de cada nodo en particular, almacenándose en la misma los datos referentes al enlace entre el nodo inicial y el nodo final como son el *nombre del enlace, identificador*.

La lista de adyacencia permite conocer a que nodos se puede llegar desde un nodo en particular y que nodos son nodos terminales, es decir son los nodos que no tienen lista de adyacencia que solo se pueda llegar a él pero de él no se pueda llegar a ningún otro nodo.

Esta estructura de datos permite una representación más exacta del modelo relacional, a la vez que facilita cualquier recorrido y trabajo con el grafo. A medida que se va obteniendo un nuevo ciclo en el grafo este se almacena en una lista de ciclos, de la cual se va extrayendo cada ciclo a la vez y efectuando la corrección correspondiente, este proceso se continúa iterando mientras exista un ciclo por extraer de la lista.

La clase TSchemasCicles es utilizada para el trabajo con el grafo y la lista de esquemas, en esta clase es donde se realiza la detección y corrección de inconsistencias del modelo relacional.

Para los ciclos de longitud = 1, la detección y corrección se realiza de forma diferente a como se realiza para los ciclos de longitud > 1 , ya que se recorre la lista de ciclos de longitud = 1 y se va corrigiendo hasta eliminar todos los ciclos (bucles), a su vez la detección de ciclos de longitud = 2 y los de longitud > 2 es diferente, pero la corrección es la misma ya que no importa la longitud del ciclo, el criterio a seguir es obtener la arista de *involment* (participación) máximo (el *involment* es el peso correspondiente a cada arista del grafo el cual no es más que la cantidad de ciclos simples en los que participa dicha arista), al encontrar la arista se pasa a corregir esta interrelación logrando eliminar cada ciclo donde la arista se encuentre involucrada, este criterio optimiza el algoritmo ya que al eliminar la arista con mayor participación se eliminan varios ciclos dando la posibilidad de realizar el menor número de recorridos y de correcciones posible logrando así generar la menor cantidad de nuevos esquemas.

La clase *StackData* es utilizada para ir adicionando o eliminando a conveniencia los elementos del grafo que se van recorriendo en la detección de ciclos, es la estructura que permite el trabajo para la detección de ciclos.

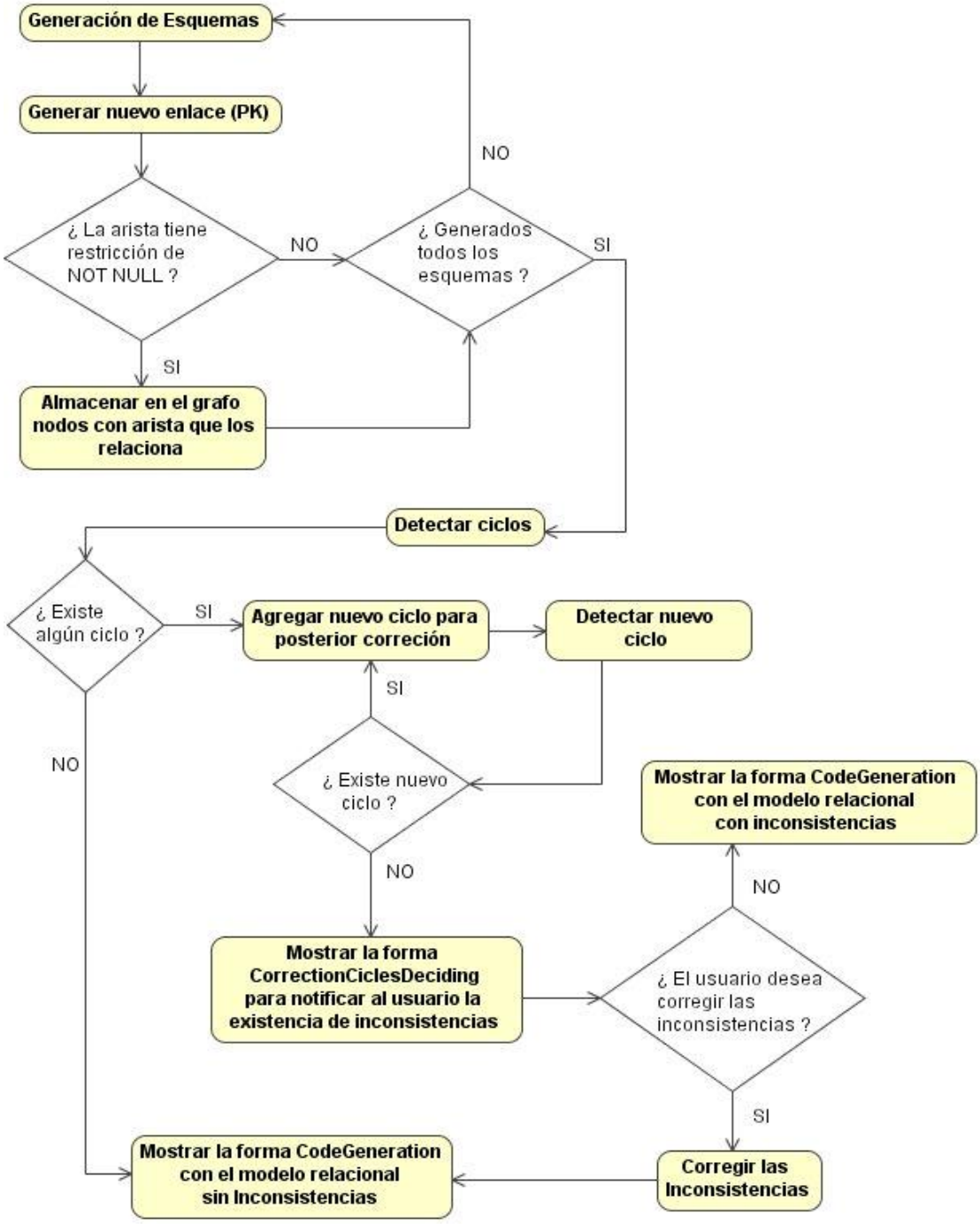

Figura 16. Diagrama de flujo del módulo *ValidationLogicalSchema*

4.1.3. Análisis de la complejidad computacional

Según (Cerruela García et al., 2002) el intervalo para predecir la cantidad de ciclos en un grafo no dirigido puede ser calculado a través de la fórmula.

$$\eta = N - E + 1$$

Donde N es la cantidad de nodos y E la cantidad de aristas obteniendo el siguiente intervalo para acotar los ciclos.

$$[\eta , 2^{\eta} - 1]$$

Por tanto se puede decir que la cantidad de ciclos en el grafo dirigido será mucho menor que la cota superior de un grafo no dirigido, por lo que esta fórmula permite acotar la cantidad de ciclos en el grafo teniendo en cuenta la cantidad de nodos y aristas.

Haciendo un análisis del algoritmo de detección se llega a la complejidad computacional de

$$O[(n^2*m^3)+(c*n*m^2)] = O[Max((n^2*m^3) , (c*n*m^2))]$$

Donde n representa los cantidad de nodos, m la cantidad de aristas y c la cantidad de ciclos detectados.

Para mayor comprensión aquí se muestra el análisis del método de detección.

Método Encontrar ciclos de longitud mayor que dos (Grafo)

{

SI existen elementos en el grafo

{

crear el primer elemento a agregar en la pila con el primer nodo del grafo

agregar a la pila el primer elemento del grafo

$O(n^2*m^3)+O(c*n*m^2)$

MIENTRAS la pila no esté vacía

{

$O(n)$

obtener índice en el grafo del nodo activo

$O(n*m^3)+O(c*m^2)$

SI se encuentra posible ciclo

{

obtener posición en la pila del último nodo visitado antes de llegar al nodo activo

crear lista ordenada para creación del ciclo

$O(n^2)$

MIENTRAS no se ha encontrado el nodo inicial del ciclo

{

O(n)

crear camino del ciclo

}

incorporar al ciclo nodo final

O(n)

obtener id de la arista

agregar datos nodo final y evitar así ciclos redundantes

$O(c*m^2)$

SI el ciclo ya existía

declarar ciclo encontrado

SI no se encontraba el ciclo por tanto hay que adicionarlo

{

crear ciclo de longitud > 2

O(m)

PARA CADA arista del ciclo

{

crear y agregar elemento al ciclo

}

$O(n*m^3)$

PARA TODAS las participaciones de las aristas

{

$O(n*m^2)$

PARA TODOS los elementos del ciclo

{

O(n*m)

aumentar la participación de las aristas en ciclos

}

}

agregar nuevo ciclo

}

O(n)

obtener índice del nodo en el grafo declarar como visitado

eliminar el elemento de la pila

}

O(n*m)

SINO

{

obtener próximo índice de lista de adyacencia de nodo activo como

nodo inicial del ciclo

O(n)

SINO se recorrió completo lista de adyacencia agregar nuevo enlace a la pila

{

crear y agregar nuevo elemento de la pila

}

O(n*m)

SINO SI no hay ciclo pero se recorrió lista de adyacencia completa

{

obtener el índice en el grafo del nodo activo crear variable de condición de parada del ciclo

eliminar nodos de la lista adyacencia recorrida completa y no se encontró ciclo alguno

O(n*m)

MIENTRAS índice de la pila sea >=0 y no verdadera condición de parada seguir

{

SI encontró elemento al cual se recorrió su lista de adyacencia

{

declarar como no visitado elementos de lista de adyacencia y eliminar de la pila

```
                                    }
                                    else
                                    {
                                        detener ciclo y marcar elemento como no
                                        visitado
                                    }
                                }
                            }
                        }
                        obtener el próximo índice de la pila
                }
}
}
```

Haciendo un análisis del algoritmo de corrección se llega a la complejidad computacional de

$O(c*n*m^3)$

Por tanto la complejidad del algoritmo en su totalidad sería

$O[Max(O[Max((n^2*m^3) , (c*n*m^2))], O(c*n*m^3))]$

4.1.4. Resultado de la implementación del algoritmo

A continuación se retoman los ejemplos explicados en el capítulo anterior para mostrar el funcionamiento del algoritmo en la herramienta.

En cada momento donde se mencione enlace el mismo en el grafo representa la llave foránea de un esquema que hace referencia a una llave foránea de otro esquema, que no necesariamente tiene que ser diferente, y cuando se mencione nodo este representa un esquema del modelo relacional.

Se analiza primero la problemática donde el Empleado tiene que pertenecer al Departamento y a su vez el Departamento debe ser dirigido por un Empleado, en este caso las restricciones de cardinalidad conducen a que el esquema presente inconsistencias de referencias cíclicas entre estos dos esquemas.

Para comenzar el proceso de transformación se debe seleccionar el menú

Esquemas y luego Generación de esquemas como se muestra en la Figura 17.

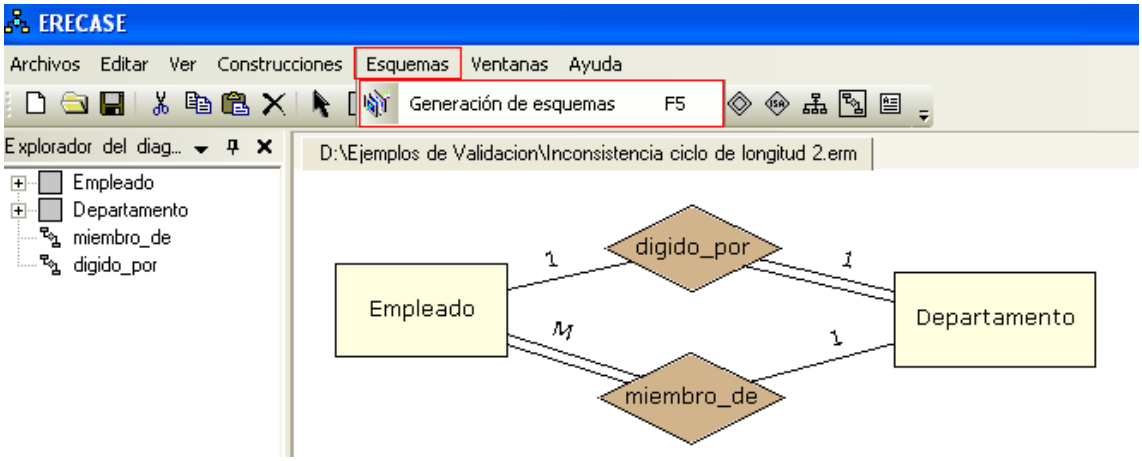

Figura 17. Modelo Conceptual con ciclo de longitud = 2

Antes de mostrar la ventana con los correspondientes esquemas lógicos y físicos asociados a este modelo conceptual se lleva a cabo la detección de inconsistencias de referencias cíclicas en el modelo. Se muestra el grafo dirigido en la Figura 18 donde se puede observar el ciclo de longitud = 2 en el que se indica la inconsistencia de referencias cíclicas entre los esquemas Departamento y Empleado.

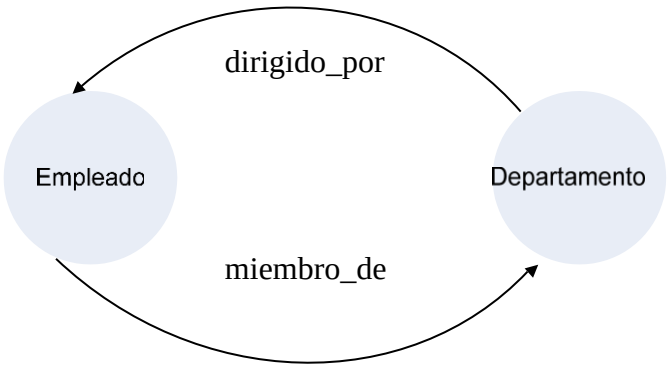

Figura 18. Grafo dirigido con ciclo de longitud =2

Al detectar el ciclo se muestra al diseñador la ventana de reporte de la presencias de inconsistencias en el modelo relacional de la Figura 18. El diseñador tiene la oportunidad de seleccionar generar un esquema lógico inconsistente o un esquema lógico sin problemas de implementación.

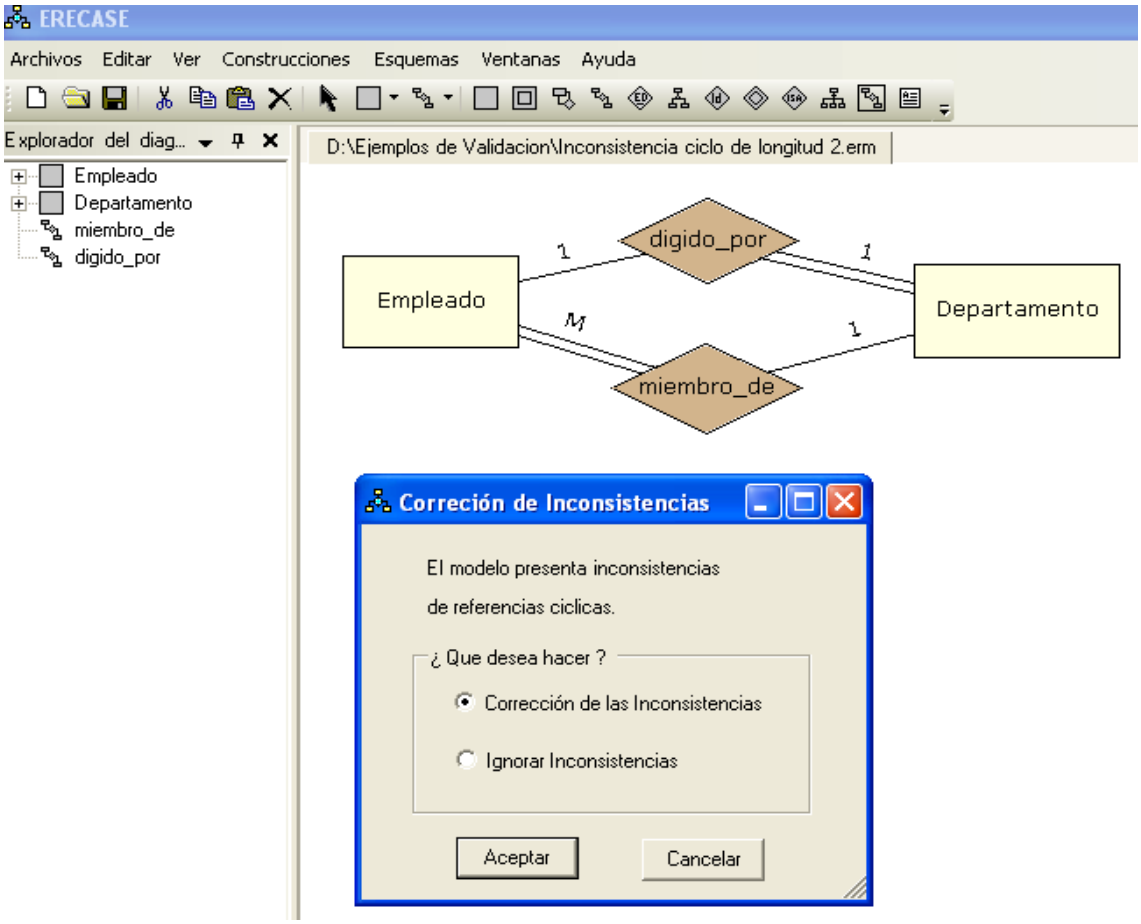

Figura 19. Reporte al diseñador de la presencia de inconsistencias en el esquema lógico para ciclos de longitud = 2

Luego de seleccionar la opción de corregir las inconsistencias se aplica el algoritmo de corrección obteniendo cambios en el grafo dirigido de la Figura 20, en este caso consiste en eliminar el enlace *miembro_de* y crear un nuevo nodo con el nombre del enlace eliminado

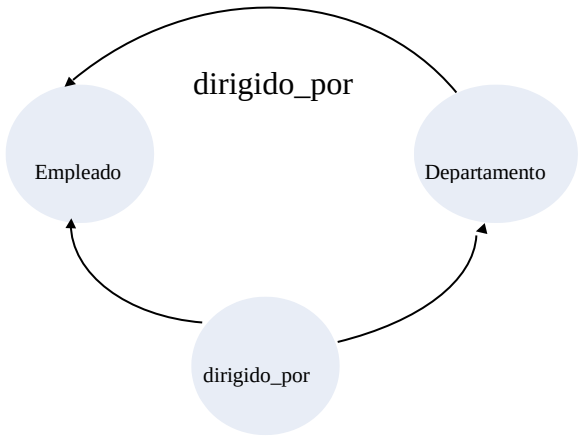

Figura 20. Grafo dirigido al aplicar algoritmo

A continuación se mostrará en la Figura 21 la ventana de esquemas lógicos y físicos con el modelo relacional sin inconsistencias.

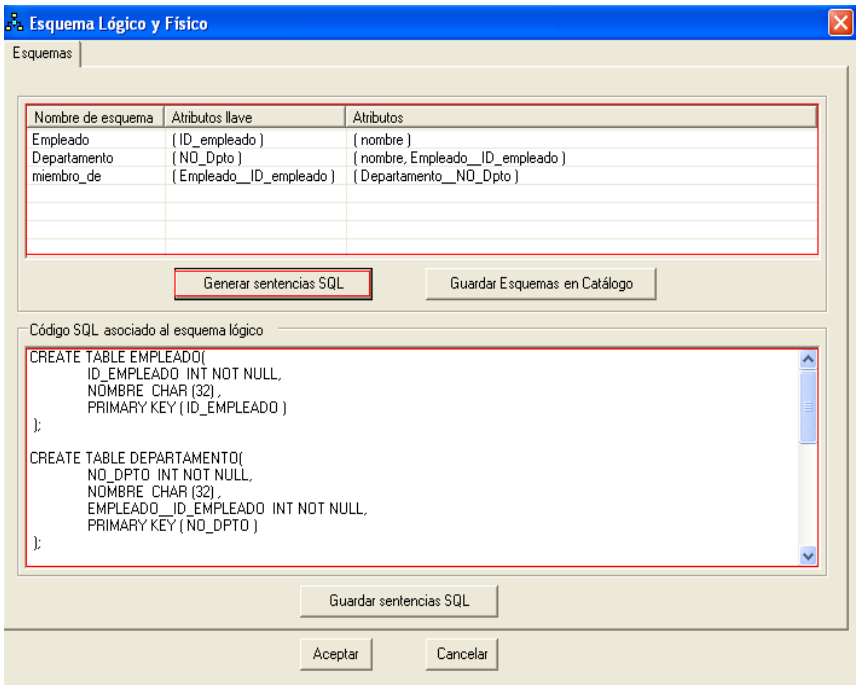

Figura 21. Esquema lógico obtenido al aplicar algoritmo de corrección al ciclo de longitud = 2

En la ventana de Corrección de la Figura 22 el usuario puede escoger la opción de ignorar las inconsistencias ante la presencia de las mismas en el modelo lógico para ciclos de longitud =2.

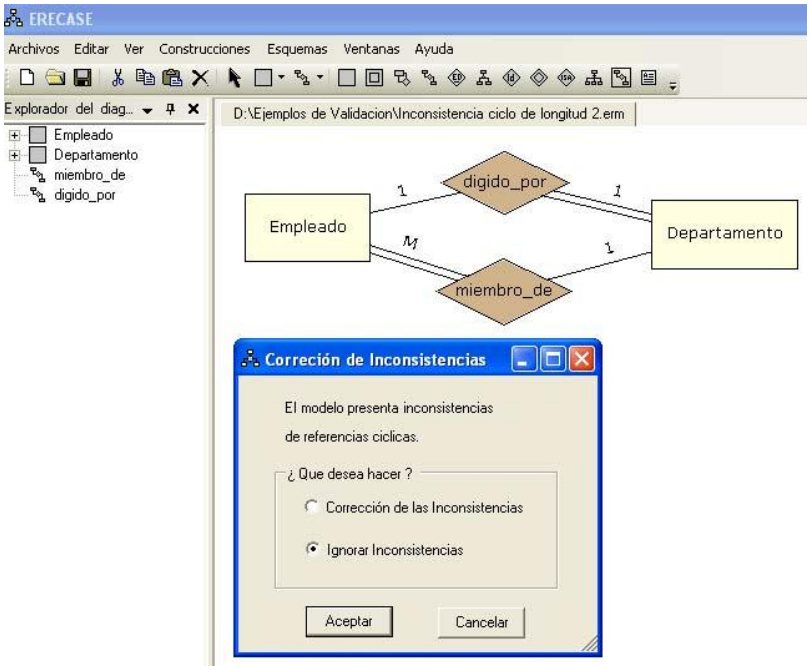

Figura 22. Selección de opción ignorar inconsistencias.

En este caso se mostrará la ventana de esquemas lógicos y físicos en la Figura 23 con el modelo lógico con inconsistencias.

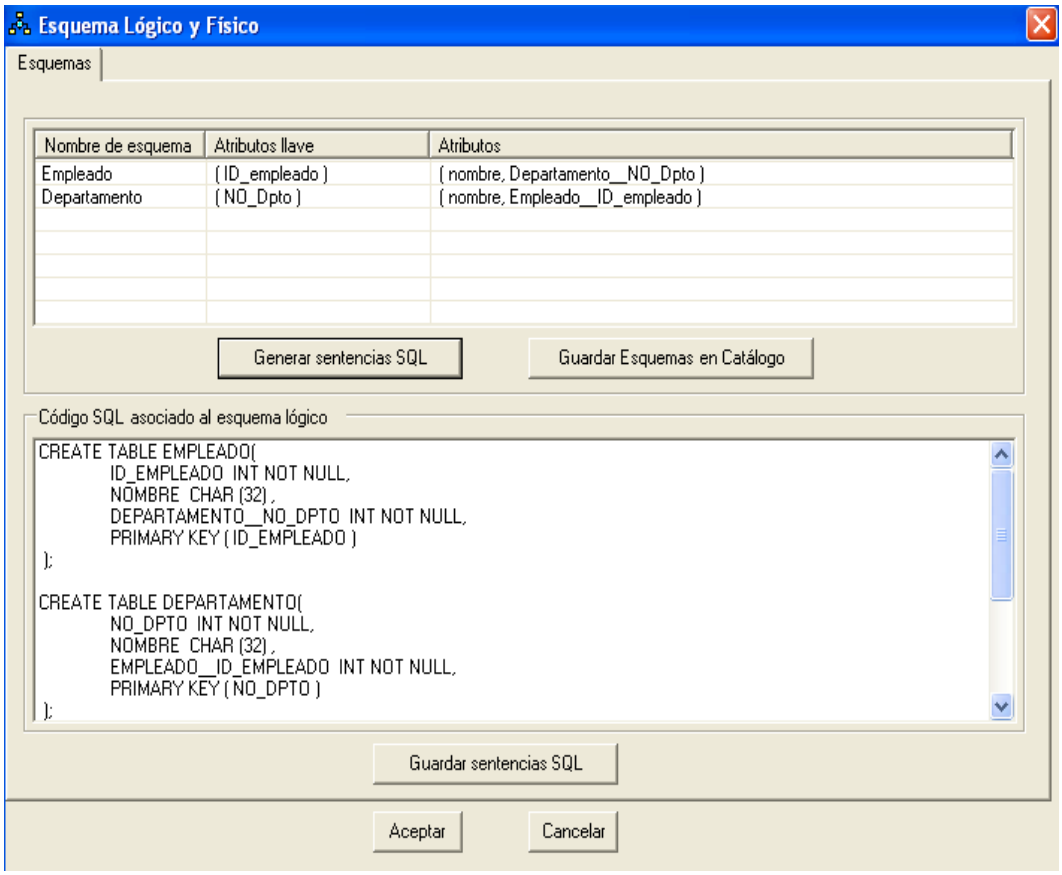

Figura 23 Esquema lógico obtenido al ignorar las inconsistencias.

Discusión

Se logró implementar un algoritmo de detección y corrección de inconsistencias en el modelo relacional. Este algoritmo fue diseñado y adaptado para poder ser usado por el ERECASE. Presenta un grafo dirigido como estructura para representar aquellos esquemas con relaciones entre sí que no admiten valores nulos. El mismo fue comprobado a través de varios casos de estudio obteniendo los resultados deseados

CAPÍTULO V

CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES

Mediante el análisis de cada uno de los métodos que existen se logró determinar una inconsistencia que se puede presentar en un esquema lógico de la base de datos, llamada "inconsistencia de referencias cíclicas".

Mediante la elaboración del algoritmo de detección y corrección de inconsistencias se logró determinar la complejidad computacional **O[Max(O[Max((n^2*m^3) , (c*n*m^2))], O(c*n*m^3))]** donde n representa los cantidad de nodos, m la cantidad de aristas y c la cantidad de ciclos detectados.

La implementación del algoritmo en la herramienta ERECASE permite la posibilidad de obtener a decisión del usuario un modelo lógico sin inconsistencias y por consiguiente sin problemas de implementación.

5.2. RECOMENDACIONES

Analizar otros métodos existentes de inconsistencias que se pueda presentar en un esquema lógico de la base de datos, que permitan mejorar los sistemas de gestión de Base de datos.

Mejorar el algoritmo matemático de detección y corrección de inconsistencias en esquemas lógicos de base de datos, para permitir una mayor precisión en la corrección y efectividad ante otro tipo de inconsistencia.

Utilizar la herramienta como material de apoyo en las unidades de aprendizaje Base de dato y diseño de base de datos, mejorando así el aprendizaje de los estudiantes y los métodos de enseñanza del docente.

5.3. BIBLIOGRAFÍA

Trabajos citados

[1] Baños Garcia Yesenia y Hernandez Lijera Araceli. (2012, Enero) Docencia. [Online]. http://www.uaeh.edu.mx/docencia/P_Presentaciones/prepa1/algoritmos.pdf

[2] CARLOS CORONEL, *Bases de Datos, Diseño, Implementacion y Administracion*. Mexico D.F, 2011.

[3] Eduardo Rene Chaves Reina, "Artículo Científico - Análisis, diseño y desarrollo de un generador de código fuente para gestión de información de MySQL, SQL Server y Access para los lenguajes Java, PHP y ASP.," in *Conferencia ESPE*, Sangolquì, 2012.

[4] MICROSOFT. (2010) [Online]. https://msdn.microsoft.com/es-es/library/4w3ex9c2(v=vs.100).aspx

[5] Silvia Kern, *Introducción a la informática Concepto de Algoritmos*. Agentina, 2008. [Online]. https://www.google.com.ec/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0ahUKEwjmhsjrnunLAhWGKh4KHQF9CnwQFggbMAA&url=http%3A%2F%2Fwww.utn.edu.ar%2Fdownload.aspx%3FidFile%3D13677&usg=AFQjCNHsjzJ_HOMXWcO_RDAeqtnHXBnhGg&cad=rja

[6] Amalia Duch. (Marzo, 2007) [Online]. https://www.cs.upc.edu/~duch/home/duch/analisis.pdf

[7] Pedro Pacheco. (2010, Diciembre) Blogspot. [Online]. http://sistemasteleinformaticos.blogspot.com/2010/12/pruebas-de-escritorio.html

[8] www.clubensayos.com. (2014, Oct.) www.clubensayos.com. [Online]. https://www.clubensayos.com/Temas-Variados/Fases-Del-Analisis-Del-Problema/2134927.html

[9] Ramírez Raquel Zambrano, *Sistema Gestores de Base de Datos.*, 2008. [Online]. http://www.csi-csif.es/andalucia/modules/mod_ense/revista/pdf/Numero_14/RAQUEL_ZAMBRANO_2.pdf

[10] Juan Esquivel Vaquera. (2010, agosto) wordpress. [Online]. https://j20v.files.wordpress.com/2010/08/modelo-entidad-relacion1.pdf

[11] LIMBER SANCHEZ MENDOZA. Webquest. [Online]. http://www.webquestcreator2.com/majwq/ver/verp/592

[12] Jorge Sánchez, *Diseño conceptual de Base de datos*.: Creative Commons, 2004. [Online]. http://jorgesanchez.net/bd/disenoBD.pdf

[13] http://www.conclase.net/. (2005) http://decimera.weebly.com/. [Online]. http://decimera.weebly.com/uploads/1/2/8/9/12896180/curso_mysql.pdf

[14] Abel Rodríguez, Norma Cabrera, Luisa González Carlos García, "Detección y corrección de inconsistencias de," *Scientific Electronic Library Online*, no. 55, p. 9, 2010.

[15] María de Lourdes Isabel Ponce Vásquez, *Base de Datos*. México, 2008. [Online]. https://es.scribd.com/doc/8585294/30/Restricciones-de-participacion-Cardinalidad-Minima

[16] Silvia Acid, Nicolás Marín, Juan Miguel Medina y Amparo Vila Olga Pons, *Introdución de los sistema de Base de Datos*. Madrid - España: Paraninfo, 2008.

[17] DIEGO R. LLANOS FERRARIS, *FUNDAMENTOS DE INFORMÁTICA Y PROGRMACIÓN EN C*. Madrid: Parainfo S.A, 2010.

[18] Carlos Coronel, *Base de Datos, Implementación y Administración*.: Cengage, 2011.

[19] Joan Vancells Flotats, *Algoritmos y programas*.

[20] A Pardo and M Ruiz, *Guía para el análisis de datos*. Madrid: McGraw-Hill, 2002.

[21] E Bonabeau, *From natural to Artificial systems, , in Swarm Intelligence*.: Oxford University Press, 1999.

[22] M. RICARDO Catherine, *Base de Datos*.: Mc Graw Hill, 2004.

[23] Cegarra José, *Metodología de la Investigación Científica y Tecnológica*. Madrid: Díaz Santos S.A, 2004.

[24] Carlos García González, Abel Rodrígez Morffi, Norma Cabrera González, and Luiza González González, *Diseño y validación estructural de esquemas conceptuales utilizando una herramienta case*., 2010, vol. 1. [Online]. http://rcci.uci.cu/index.php?journal=rcci&page=article&op=view&path%5B%5D=32&path%5B%5D=31

[25] PNI Trujillo Senati, *Programa Nacional de Informatica*., Agosto 2007. [Online]. https://es.scribd.com/doc/300201195/15/Modelo-Entidad-Relacion

[26] Carlos García, Abel Rodríguez, Norma Cabrera, and Luisa González, *Detection and correction of inconsistencies of cyclical references in database logical schemas*., Agosto 2010.

[27] Fernando Palmero. aprendizajealgoritmos. [Online]. http://aprendizajealgoritmos.blogspot.com/2011/02/que-es-un-algoritmo.html

[28] Jorge Sánchez. (2004) [Online]. http://www.jorgesanchez.net/bd/disenoBD.pdf