UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS

METODOLOGÍA PARA LA CONVERSIÓN DEL MODELO ENTIDAD RELACIÓN AL MODELO ORIENTADO A OBJETOS.

AUTORES:

WAKERNAGEL W. RIVERA F.

ROBERTO C. ALVAREZ R.

TESIS PREVIO A LA OBTENCIÓN DEL TITULO DE INGENIERO EN SISTEMAS

Ambato - Ecuador

Febrero - 2004

AGRADECIMIENTO

Nuestro profundo agradecimiento por su valiosa ayuda a la Universidad Técnica de Ambato y de manera especial a nuestra Facultad de Ingeniería en Sistemas, como a su personal Docente y Administrativo, quienes nos enseñaron a ser elementos valiosos y útiles para la de la sociedad, de un modo muy exclusivo al Ing. Edison Alvarez Director y al Ing. Oswaldo Paredes Asesor de nuestra Tesis; ya que con su constante apoyo, permitieron desarrollar y llevar a un feliz término la culminación de este trabajo.

Auguramos nuestros sinceros deseos para desempeñarnos exitosamente en nuestra futura vida profesional y poner muy en alto a tan prestigiosa Institución.

Wakernagel Rivera

Roberto Alvarez

DEDICATORIA

A Dios, quiero dedicarle todo mi esfuerzo y el trabajo alcanzado, a quien agradezco imperecederamente por darme la oportunidad de vivir, y por poder compartir con las personas que más amo.

A mis padres Humberto Rivera y Victoria Freire quienes con su apoyo moral, económico e incondicional han logrado ver plasmado el anhelo que siempre quisieron para su hijo.

A mi esposa Yadira y a mis hijas Michelle y Yuliana las mismas ha quienes quiero agradecerles por aconsejarme y apoyarme en todo lo necesario para alcanzar todos mis proyectos propuestos, y por compartir en la familia ese gran amor, cariño y comprensión de un hogar.

A mis Hermanos, Walker, Wimper e Hipatia quienes me apoyaron incondicionalmente día a día, a la culminación de mi meta tan anhelada y quienes se convirtieron en la base fundamental de la familia.

Wakernagel.

Dedico este trabajo con inmensa emoción, a mi padre Manuel Alvarez a mis hermanas Rosita Alvarez, Cecilia Alvarez y Ximena Alvarez, a mis amigos, amigas y a una persona muy especial el Dr Ángel Gustavo Haro Jácome fueron quienes influyeron en mis estudios en los buenos y malos momentos quienes me impulsaron a seguir adelante y tener éxitos en mi vida estudiantil.

También quiero dedicar este trabajo a maestros y juventud estudiosa de la Universidad Técnica de Ambato y sobre todo a los lectores de este trabajo el cual aspiro que estas páginas se conviertan en un instrumento útil para el conocimiento de la Informática.

Roberto

DECLARACION, AUTENTICIDAD Y RESPONSABILIDAD

Nosotros, Wakernagel Walpole Rivera Freire, C.I. 180284091-6 y Roberto Cristóbal Alvarez Rivera, C.I. 1803062932.

Declaramos que la investigación enmarcada en el diseño de esta Tesis es absolutamente original, auténtico y personal. Por tal motivo, declaro que el contenido, efectos legales y académicos que se desprenden del trabajo de la Tesis son y serán de nuestra sola y exclusiva responsabilidad legal y académica.

_____ _____

Wakernagel Walpole Rivera Freire Roberto Cristóbal Alvarez Rivera

INTRODUCCIÓN

En esta Tesis, se da una idea del campo de las Bases de Datos Orientada a Objetos, el material en este volumen es una Investigación desarrollada sobre este tema. Fue escogido primordialmente por su amplitud, ideas principales y enfoques tanto relacionales como Orientado a Objetos.

El campo de las Bases de Datos Orientadas a Objetos se ha introducido como una nueva área de investigación. Los campos de lenguajes de programación, inteligencia artificial e ingeniería de software han contribuido con el uso de la tecnología orientada a objetos en el área de las Bases de Datos. El desafío del área de Bases de Datos es integrarlos en un diseño de sistema simple que mantenga el equipo deseado para cada campo. El resultado de realizar la Conversión es la característica central de este Trabajo para obtener una mejor idea y comprensión de Bases de Datos Orientas a Objetos. Describiéndose por capítulos la investigación realizada que consta Capitulo I (INTRODUCCIÓN), describe los antecedentes, justificación y objetivos: General y Específicos del tema a tratar; se aclara el alcance y limitaciones de la investigación. Capitulo II definiciones en forma rápida del MER y el MOO, Capitulo III un estudio profundo del Modelo Entidad Relación, Capitulo IV un amplio estudio de Bases de Datos Orientadas a Objetos, Capitulo V desarrollo de la Metodología de Transformación del MER al Modelo Orientado a Objetos y un caso de estudio y en el Capítulo VI se describen las conclusiones a las que se ha llegado mediante la investigación, así como sus recomendaciones.

ÍNDICE

CONTENIDO…… **Págs.**

INDICE DE FIGURAS

INDICE DE TABLAS

BIBLIOGRAFÍA

- BOOCH Grady (1999), “El lenguaje de unificado de modelado”, Primera Edición.
- KENDALL Kenneth E. y KENDALL Julie E., (1991), “Análisis y Diseño de Sistemas”.
- RUMBAUGH James.(1999), “Modelado y Diseño Orientado a Objetos”, Tercera Edición.
- PRESSMAN Roger. (2002), “Ingeniería de Software un enfoque práctico”, Quinta Edición.
- SANCHEZ Wigberto, “Sistemas de Bases de Datos Integradas”, Primera Edición.
- http://lucas.hispalinux.es/Tutoriales/doc-modelado-sistemas-UML/multiple-html/index.html
- http://www.bd.cesma.usb.ve/ci5311/programa.html
- http://www.cs.cinvestav.mx/BDChapa/seck/indice.html
- http://www.conocimientosweb.net/dcmt/topic1.html

CAPITULO I

INTRODUCCIÓN

1.1.- ANTECEDENTES

A finales de la década de 1960, cuando las bases de datos entraron por primera vez en el mercado del software, los diseñadores de bases de datos actuaban como artesanos, con herramientas muy primitivas: diagramas de bloques y estructuras de registros eran los formatos comunes para las especificaciones, y el diseño de bases de datos se confundía frecuentemente con la implantación de las bases de datos. Esta situación ahora ha cambiado: los métodos y modelos de diseño de bases de datos han evolucionado paralelamente con el progreso de la tecnología en los sistemas de bases de datos y sistemas de bases datos distribuidas. Se ha entrado en la era de los sistemas relacionales de bases de datos, que ofrecen poderosos lenguajes de consulta, herramientas para el desarrollo de aplicaciones e interfaces amables con los usuarios. La tecnología de bases de datos cuenta ya con un marco teórico, que incluye la teoría relacional de datos, procesamiento y optimización de consultas, control de concurrencia, gestión de transacciones y recuperación.

Las bases de datos orientadas a objetos se crearon para tratar de satisfacer las necesidades de estas nuevas aplicaciones. La orientación a objetos ofrece flexibilidad

para manejar algunos de estos requisitos y no está limitada por los tipos de datos y los lenguajes de consulta de los sistemas de bases de datos tradicionales. Una característica clave de las bases de datos orientadas a objetos es la potencia que proporcionan al diseñador al permitirle especificar tanto la estructura de objetos complejos, como las operaciones que se pueden aplicar sobre dichos objetos.

Otro motivo para la creación de las bases de datos orientadas a objetos es el creciente uso de los lenguajes orientados a objetos para desarrollar aplicaciones. Las bases de datos se han convertido en piezas fundamentales de muchos sistemas de información y las bases de datos tradicionales son difíciles de utilizar cuando las aplicaciones que acceden a ellas están escritas en un lenguaje de programación orientado a objetos como C++, Smalltalk o Java. Las bases de datos orientadas a objetos se han diseñado para que se puedan integrar directamente con aplicaciones desarrolladas con lenguajes orientados a objetos, habiendo adoptado muchos de los conceptos de estos lenguajes.

Según ha avanzado la tecnología de bases de datos, se han desarrollado las metodologías y técnicas de diseño. Se ha alcanzado un consenso, por ejemplo, sobre la descomposición del proceso de diseño en fases, sobre los principales objetivos de cada fase y sobre las técnicas para conseguir estos objetivos.

En el proceso de abstracción que conduce a la creación de una base de datos desempeña una función prioritaria el **MODELO DE DATOS.** El modelo de datos, como abstracción del universo de discurso, es el enfoque utilizado para la representación de las entidades y sus características dentro de la base de datos.

Los modelos más extendidos son el modelo entidad-relación y el orientado a objetos existiendo además otro tipo de modelos. El modelo entidad relación (MER) se basa en una percepción del mundo compuesta por objetos, llamados entidades, y relaciones entre ellos. Las entidades se diferencian unas de otras a través de atributos. El modelo orientado a objetos también se basa en objetos, los cuales contienen valores y métodos, entendidos como órdenes que actúan sobre los valores, en niveles de anidamiento. Los objetos se agrupan en clases, relacionándose mediante el envío de mensajes.

1.2.-JUSTIFICACION

Los modeladores de bases de datos han visto la necesidad de realizar un modelado en el menor tiempo posible para lo cual utilizarían el Modelo Entidad Relación y tener una perspectiva orientada a objetos.

El modelo de objetos ha recibido la influencia de una serie de factores, no sólo de la programación orientada a objetos. Además, el modelo de objetos ha demostrado ser un concepto unificador en la informática, aplicable no sólo a los lenguajes de programación, sino también al diseño de interfaces de usuario, bases de datos distribuidas e incluso arquitecturas de ordenadores. La razón para este gran atractivo es simplemente que una orientación a objetos ayuda a combatir la complejidad inherente de muchos tipos de sistema diferentes.

La necesidad de minimizar el tiempo de desarrollo de las bases de datos, así como los costos de mantenimiento y mejora de programas de aplicación.

Al momento existen metodologías de conversión inconclusas por lo que es necesario indagar una nueva metodología y realizar un estudio a fondo de la misma.

Debido a la escasez bibliográfica respecto de estos temas, y a la falta de profundidad con la que muchas veces son tratados, se pretende establecer una base sólida de consulta para que estudiantes y personas involucradas en el área de sistemas, hagan uso de esta tesis para participar satisfactoriamente en proyectos relacionados con el modelado de bases de datos

1.3 OBJETIVO GENERAL

- Desarrollar una metodología de conversión del Modelo Entidad Relación al Modelo Orientado a Objetos.

1.4 OBJETIVOS ESPECÍFICOS

- Realizar un análisis general sobre la situación actual del modelado de Bases de Datos Relacionales y Orientadas a Objetos.
- Documentar los conceptos del Modelo Entidad Relación y el Modelo Orientado a Objetos.
- Determinar los fundamentos del Modelo Orientado a Objetos a través de sus definiciones, características y objetivos.

1.5 DELIMITACIONES.

Esta investigación se encuentra limitada por la insuficiente información documentada existente en nuestro medio, y de cierta información encontrada en Internet. También se limita a las experiencias obtenidas durante el proceso de educación en la Facultad de Ingeniería en Sistemas.

Se describen los diferentes aspectos que son necesarios durante el Modelado de Bases de Datos permitiendo ilustrar a la base de datos en términos de Objetos, y no directamente en tablas.

Se pretende establecer una base sólida de consulta para que estudiantes y personas involucradas en el área de sistemas, hagan uso de este trabajo y así poder participar satisfactoriamente en proyectos relacionados con la construcción de SGBDOO (Sistemas de gestión de Bases de Datos Orientados a objetos).

CAPÍTULO II

FUNDAMENTO TEÓRICO

2.1 MODELOS DE BASES DE DATOS

2.1.1 Definición

Un modelo es un conjunto de herramientas conceptuales para describir datos, sus relaciones, su significado y sus restricciones de consistencia.

El modelado es la actividad más delicada e importante en la realización de una aplicación con base de datos

Los diseñadores construyen muchos tipos de modelos para diversos propósitos antes de construir las cosas. Los ejemplos incluyen modelos arquitectónicos para mostrarse a los clientes, modelos de aeroplanos a escala para pruebas en túneles de viento, esbozos a lápiz para componer pinturas al óleo, planos de partes de máquinas, argumentos de comerciales y borradores de libros. Los modelos sirven para varios propósitos:

- Probar una entidad física antes de construirla. Los constructores medievales no conocían la física moderna, pero construyeron modelos a escala de las catedrales góticas para probar las fuerzas en las estructuras. Los modelos de aeroplanos, carros y botes han sido probados en túneles de viento y tanques de agua para mejorar su aerodinámica. Avances recientes en la computación

permiten la simulación de muchas estructuras físicas sin tener que construir los modelos físicos. No solamente es mas barata la simulación, sino que proporciona información que es tan inaccesible o fugaz para ser medida en un modelo físico. Ambos modelos, el físico y el computacional, son usualmente más baratos que el construir un sistema completo y permiten que las fallas sean corregidas en etapas tempranas.

- Comunicación con los clientes. Los arquitectos y diseñadores de productos construyen modelos para mostrarlos a sus clientes. Las maquetas son productos de demostración que imitan una parte o todo el comportamiento externo de un sistema.
- Visualización. Los argumentos de las películas, programas de televisión y comerciales permiten a los escritores ver como fluyen sus ideas. Las transiciones toscas, finales inadecuados y segmentos innecesarios pueden ser modificados antes de que se inicie el escrito detallado. Los bosquejos de los artistas les permiten enfocar sus ideas y hacer cambios antes de realizarlos con pintura o en piedra.
- Reducción de la complejidad. Quizá la principal razón para modelar, que incorpora todas las razones previas, sea la de trabajar con sistemas que son tan complejos como para entenderlos directamente. La mente humana puede hacer frente solamente a una cantidad limitada de información a la vez. Los modelos reducen la complejidad al separar un número pequeño de cosas importantes con las cuales trabajar al mismo tiempo.

2.1.2 Tipos de modelado de datos

- Modelos basados en registros
- Modelos basados en objetos

2.1.2.1 Modelos basados en registros

- **Jerárquico**

  Datos en registros relacionados por *apuntadores* y organizados como colecciones de árboles

- **De redes**

  Datos en registros relacionados por *apuntadores* y organizados en gráficas arbitrarias

- **Relacional**

  Datos en tablas relacionados por el *contenido* de ciertas columnas

2.1.2.2 Modelos basados en objetos

- **Entidad-relación**

  Datos organizados en conjuntos inter-relacionados de objetos (entidades) con atributos asociados.

- **Orientado a objetos**

  Datos como *instancias* de *clases* de objetos con *métodos* asociados

En el caso de nuestro estudio nos basaremos en los Modelos Basados a Objetos de los cuales daremos una breve definición.

2.2 MODELO ENTIDAD-RELACIÓN

2.2.1 Definición

Técnica de análisis basada en la identificación de las entidades y de las relaciones que se dan entre ellas en la parte de realidad que pretendemos modelar.
El modelo E/R permite representar de forma abstracta los datos que se pretenden almacenar en la base de datos.

2.2.2 Elementos del modelo E/R

- **Entidad**: Objeto, real o abstracto, distinguible de otros objetos. Al grupo de entidades con cualidades similares acerca de los cuales se almacena información se le denomina TIPO (o, simplemente, conjunto de entidades).

  p.ej. Un libro concreto o un escritor

- **Entidades fuertes y entidades débiles**

  Un tipo de entidad es fuerte si la existencia de sus ocurrencias no depende de ningún otro tipo. En caso contrario, se dice que el tipo de entidad es débil.

- **Atributo**: Propiedad asociada a un conjunto de entidades (esto es, mediante los atributos representamos propiedades de los objetos). Para cada atributo hay un conjunto de valores permitidos llamado DOMINIO.

  p.ej. Del libro: Título, edición, número de páginas…

  Del escritor: Nombre, apellidos, fecha de nacimiento…

- **Clave:** Conjunto de atributos que permite identificar unívocamente a una entidad dentro de un conjunto de entidades. p.ej. Del escritor: (nombre, apellidos, fecha de nacimiento)

  - **Superclave:** Conjunto de atributos que permite identificar unívocamente a una entidad dentro de un conjunto de entidades.
  - **Clave candidata:** Superclave con un número mínimo de atributos.
  - **Clave primaria:** Clave candidata elegida por el diseñador de la base de datos para identificar unívocamente a las distintas entidades de un tipo.
  - **Clave alternativa:** Cualquiera de las claves candidatas no elegidas por el diseñador de la base de datos

- **Relación (conexión o asociación)**:

  Conexión semántica entre dos conjuntos de entidades.

  p.ej. Relación entre los escritores y los libros que han escrito.

- **Supertipo** Tipo de entidad que incluye uno o más subgrupos distintos de ocurrencias que deben ser representados en el modelo de datos.

- **Subtipo** Cada uno de los subgrupos de ocurrencias de un tipo de entidad que se han de representar en el modelo de datos**.**

- **Especialización** Proceso de extraer diferencias entre las ocurrencias de un tipo de entidad para distinguir los subtipos que lo forman.

- **Generalización** Proceso de encontrar la parte común de las ocurrencias de distintos tipos de entidad para extraer el supertipo que los engloba**.**

- **Agregación** Proceso en el cual consiste en construir un conjunto de entidades sobre la base de un conjunto de relaciones entre entidades.

2.3 MODELO ORIENTADO A OBJETOS

El término **orientación a objetos** (**OxO**) es utilizado para referirse a un conjunto creciente de tecnologías de *software* que van desde los lenguajes de programación

orientado a objetos, pasando por los sistemas de bases de datos orientado a objetos, las interfaces gráficas orientado a objetos y los sistemas operativos orientado a objetos, hasta llegar a la Ingeniería de Software orientado a objetos y su aplicación a los sistemas de información orientado a objetos. Estas tecnologías proveen conceptos, métodos, técnicas y herramientas para:

1. Modelar el mundo real en una forma lo más cercana posible a la perspectiva del usuario.
2. Interactuar fácilmente con un entorno computacional utilizando metáforas familiares al usuario.
3. Construir componentes de software reutilizables, así como librerías de módulos de software fácilmente extensibles.
4. Modificar y extender fácilmente las implementaciones de los componentes sin tener que recodificar los programas desde el inicio.

Podemos definir la Orientación a Objetos como un enfoque o paradigma de desarrollo de software que se caracteriza por su capacidad para modelar (representar) las entidades de un dominio de aplicación o universo del discurso, en una forma muy natural y directa, en términos de objetos. En este enfoque, a cada entidad del dominio de aplicación - dominio del problema o sistema de actividades - se le asocia un constructo o símbolo de modelado conocido como objeto. Un objeto es una unidad de software que representa o modela el estado y la dinámica que tiene una entidad. Los objetos simulan no sólo la estructura del dominio de aplicación, sino también su comportamiento. A través de un modelo computacional conocido como pase de

mensajes, los objetos se comunican entre sí y cooperan para simular la dinámica del dominio de aplicación.

Objeto.

Es un elemento auto contenido utilizado por el programa. Los valores que almacena un objeto se denominan atributos, variables o propiedades. Los objetos pueden realizar acciones, que se denominan métodos, servicios, funciones, procedimientos u operaciones. Los objetos tienen un gran sentido de la privacidad, por lo que sólo dan información sobre si y mismos a través de los métodos que poseen para compartir su información. También ocultan la implementación de sus procedimientos, aunque es muy sencillo pedirles que los ejecuten. Los usuarios y los programas de aplicación no pueden ver qué hay dentro de los métodos, sólo pueden ver los resultados de ejecutarlos. A esto es a lo que se denomina ocultación de información o encapsulamiento de datos. Cada objeto presenta una interface pública al resto de objetos que pueden utilizarlo. Una de las mayores ventajas del encapsulamiento es que mientras que la interface pública sea la misma, se puede cambiar la implementación de los métodos sin que sea necesario informar al resto de objetos que los utilizan. Para pedir datos a un objeto o que este realice una acción se le debe enviar un mensaje. Un programa orientado a objetos es un conjunto de objetos que tienen atributos y métodos. Los objetos interactúan enviándose mensajes. La clave, por supuesto, es averiguar qué objetos necesita el programa y cuáles deben ser sus atributos y sus métodos.

Elementos del Modelo de Objetos

En el modelo de objetos se pueden destacar cuatro elementos fundamentales:

- **Abstracción**
- **Encapsulamiento**
- **Modularidad**
- **Jerarquía**

Un modelo que carezca de cualquiera de los elementos anteriores, no será un modelo orientado a objetos.

Además de estos elementos fundamentales, existen otros que representan aspectos deseables en los modelos orientados a objetos:

- **Tipificación**
- **Concurrencia**
- **Persistencia**

Abstracción.

La abstracción es una de las vías fundamentales por la que los humanos combatimos la complejidad. La abstracción surge de un reconocimiento de las similitudes entre ciertos objetos, situaciones o procesos del mundo real, y la decisión de concentrarse en esas similitudes e ignorar las diferencias que puedan existir.

La abstracción denota las características esenciales de un objeto que lo distinguen de todos los demás tipos de objeto y proporciona así fronteras conceptuales nítidamente definidas respecto a la perspectiva del observador.

En si la Abstracción consiste en centrarse en los aspectos esenciales de una entidad e ignorar sus propiedades accidentales. En el desarrollo de sistemas esto significa centrarse en lo que es y lo que hace un objeto antes de decidir cómo debería ser implementado. La capacidad de utilizar herencia y polimorfismo proporciona una potencia adicional. El uso de la abstracción durante el análisis significa tratar solamente conceptos del dominio de la aplicación y no tomar decisiones de diseño o de implementación antes de haber comprendido el problema. Un uso adecuado de la abstracción permite utilizar el mismo modelo para el análisis, diseño de alto nivel, estructura del programa, estructura de una base de datos y documentación. Un estilo de diseño independiente del lenguaje pospone los detalles de programación hasta la fase final, relativamente mecánica del desarrollo.

Encapsulamiento.

La abstracción y el encapsulamiento son conceptos complementarios: la abstracción se centra en el comportamiento observable de un objeto, mientras el encapsulamiento se centra en la implementación que da lugar a este comportamiento. El encapsulamiento se consigue a menudo mediante la ocultación de información, que es el proceso de ocultar todos los secretos de un objeto que no contribuyen a sus características esenciales; normalmente, la estructura de un objeto está oculta, así como sus métodos.

En la práctica, cada clase debe tener dos partes: un interfaz y una implementación. El interfaz de una clase captura sólo su vista externa, abarcando la abstracción que se ha hecho del comportamiento común de todas las instancias de la clase. La implementación

de una clase comprende la representación de la abstracción así como los mecanismos que consiguen el comportamiento deseado.

Según esto: el encapsulamiento es el proceso de almacenar en un mismo compartimento los elementos de una abstracción que constituyen su estructura y su comportamiento; sirve para separar el interfaz contractual de una abstracción y su implantación.

Un encapsulamiento inteligente hace que sean locales las decisiones de diseño que probablemente cambien. A medida que evoluciona un sistema, sus desarrolladores pueden descubrir que en una utilización real, ciertas operaciones llevan más tiempo que el admisible o que algunos objetos consumen más recursos que los disponibles. En estas situaciones, la representación de un objeto suele cambiarse para poder aplicar algoritmos más eficientes o para que el consumo de recursos por parte de algunas operaciones sea inferior. Esta capacidad para cambiar la representación de una abstracción sin alterar la abstracción propiamente dicha, y por consiguiente sin que se necesiten modificaciones en otros objetos, es el beneficio esencial de la encapsulación.

Denominado también *ocultamiento de información* consiste en separar los aspectos externos del objeto, a los cuales pueden acceder otros objetos, de los detalles internos de implementación del mismo, que quedan ocultos para los demás. El encapsulamiento evita que el programa sea tan interdependiente que un pequeño cambio tenga efectos secundarios masivos. La implementación de un objeto se puede modificar sin afectar a las aplicaciones que la utilizan. Quizás sea necesario modificar la implementación de un objeto para mejorar el rendimiento, corregir un error, consolidar el código o para hacer un transporte a otra plataforma. El encapsulamiento no es exclusivo de los lenguajes

orientados a objetos pero la capacidad de combinar la estructura de datos y el comportamiento de una única entidad hace que el encapsulamiento sea mas potente y claro que en los lenguajes convencionales que separan las estructuras de datos y el comportamiento.

Modularidad.

La fragmentación de un programa complejo en componentes individuales puede reducir la complejidad de éste. Especialmente cuando se trata de aplicaciones muy grandes, en las que puede haber cientos de clases, el uso de módulos es esencial para ayudar a manejar la complejidad. Además, la modularidad permite crear fronteras bien definidas y documentadas dentro del programa.

La modularización consiste en dividir un programa en módulos que pueden compilarse separadamente, pero que tienen conexiones con otros módulos.

La mayoría de los lenguajes que soportan el módulo como un concepto adicional distinguen entre el interfaz y su implementación. De esta forma, la interfaz y el encapsulamiento de un objeto van unidos.

Una de las ventajas más importantes que vienen de la modularización de los programas es la rebaja de los costes del software ya que los módulos se diseñan y se revisan independientemente, con lo que se pueden utilizar para varios programas con el ahorro de tiempo correspondiente. Para que sea posible esto, es necesario que los diferentes módulos con los que se trabaja no sufran un gran acoplamiento entre ellos. La mejor

solución es utilizar interfaces sencillas y ocultar tanto como sea posible la implantación del módulo.

Se entiende modularidad como la propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y débilmente acoplados.

De esta forma, los principios de abstracción, encapsulamiento y modularidad están relacionados, no son independientes entre ellos: un objeto proporciona una frontera bien definida alrededor de una sola abstracción, y tanto el encapsulamiento como la modularidad proporcionan barreras que rodean a esta abstracción.

Jerarquía.

Frecuentemente un conjunto de abstracciones forma una jerarquía, y la identificación de esas jerarquías en el diseño simplifica en gran medida la comprensión del problema. Por tanto se puede entender la jerarquía como una clasificación u ordenación de abstracciones.

En un modelo de objetos las dos jerarquías más importantes son la estructura de clases y la estructura de objetos.

La herencia es la jerarquía de clases más importante y es un elemento esencial de los sistemas orientados a objetos. Básicamente, la herencia define una relación entre clases, en la que una clase comparte la estructura de comportamiento definida en una o más clases: *herencia simple o herencia múltiple*. De esta forma, la herencia representa así una jerarquía de abstracciones, en la que una subclase hereda de una o más superclases.

Normalmente, una subclase aumenta o redefine la estructura y el comportamiento de sus superclases. Semánticamente, la herencia denota una relación “es un”.

A medida que se desarrolla la jerarquía de herencias, la estructura y comportamiento comunes a diferentes clases tenderá a migrar hacia superclases comunes. Por esta razón se habla a menudo de la herencia como una jerarquía de *generalización/especialización*.

Las superclases representan abstracciones en las que los campos y métodos de la superclase sufren añadidos, modificaciones o incluso ocultaciones.

El ignorar las jerarquías de clase que existen puede conducir a diseños deformados y poco elegantes.

Tipos (tipificación)

Los tipos son la puesta en vigor de la clase de los objetos, de modo que los objetos de tipos distintos no pueden intercambiarse o, como mucho, pueden intercambiarse sólo de formas muy restringidas.

Los tipos permiten expresar las abstracciones de manera que el lenguaje de programación en el que se implantan puede utilizarse para apoyar las decisiones de diseño.

Los lenguajes de programación pueden tener comprobación estricta o comprobación débil de los tipos, aún así, no dejan de ser lenguajes orientados a objetos.

En lenguajes con comprobación estricta de tipos, los errores a la concordancia de tipos pueden detectarse en tiempo de compilación, mientras que en lenguajes con una comprobación débil de tipos, un cliente puede enviar cualquier mensaje a cualquier

clase aunque ésta no sepa como responder a ese mensaje (produciéndose errores de ejecución, en la mayor parte de los casos).

Otro aspecto a resaltar en lo que se refiere a los tipos son los conceptos de ligadura estática y dinámica. Mientras que la comprobación estricta o débil de los tipos se refería a la consistencia de ellos, la ligadura se refiere al momento en el que los nombres se ligan con sus tipos. La ligadura estática significa que se fijan los tipos de todas las variables y expresiones en tiempo de compilación; la ligadura dinámica significa que los tipos de las variables y expresiones no se conocen hasta el tiempo de ejecución.

Al ser la comprobación estricta de tipos y la ligadura conceptos independientes, un lenguaje puede tener comprobación estricta de tipos y tipos estáticos (Ada), puede tener comprobación estricta de tipos pero soportar enlace dinámico (C++), o no tener tipos y admitir la ligadura dinámica (Smalltalk).

Concurrencia.

En muchos de los problemas que surgen hoy en día, es lógico pensar en el uso de un conjunto distribuido de ordenadores o utilizar procesadores capaces de realizar multitarea. Un solo proceso (*hilo de control*) es de donde parten las acciones dinámicas independientes dentro del sistema. Todo programa tiene al menos un hilo de control, pero un sistema que implique concurrencia puede tener muchos de tales hilos; algunos temporales y otros que permanecen a lo largo del ciclo de vida del sistema.

El diseño de características para la concurrencia en lenguajes de POO no es muy diferente de hacerlo en otros tipos de lenguajes. Se trate o no de un lenguaje orientado a

objetos, continúan existiendo los mismos problemas tradicionales: interbloqueo, bloqueo activo, inanición, exclusión mutua...

Persistencia

Un objeto de software ocupa una cierta cantidad de espacio, y existe durante una cierta cantidad de tiempo: hay un espacio continuo de existencia de un objeto.

La persistencia es la propiedad de un objeto por la que su existencia trasciende el tiempo, es decir, el objeto continúa existiendo después de que su creador deja de existir) y/o el espacio, la posición del objeto varía con respecto al espacio de direcciones en el que fue creado.

Clase.

Es un patrón o plantilla en la que se basan objetos que son similares. Cuando un programa crea un objeto de una clase, proporciona datos para sus variables y el objeto puede entonces utilizar los métodos que se han escrito para la clase. Todos los objetos creados a partir de la misma clase comparten los mismos procedimientos para sus métodos, también tienen los mismos tipos para sus datos, pero los valores pueden diferir.

Una clase también es un tipo de datos. De hecho una clase es una implementación de lo que se conoce como un tipo abstracto de datos. El que una clase sea también un tipo de datos significa que una clase se puede utilizar como tipo de datos de un atributo.

Tipos de clases. En los programas orientados a objetos hay tres tipos de clases: clases de control, clases entidad y clases interface.

- Las clases de control gestionan el flujo de operación de un programa.
- Las clases entidad son las que se utilizan para crear objetos que manejan datos (por ejemplo, clases para personas, objetos tangibles o eventos).
- Las clases interface son las que manejan la entrada y la salida de información (por ejemplo, las ventanas gráficas y los menús utilizados por un programa).

En los programas orientados a objetos, las clases entidad no hacen su propia entrada/salida. El teclado es manejado por objetos interface que recogen los datos y los envían a los objetos entidad para que los almacenen y los procesen. La salida impresa y por pantalla la formatea un objeto interface para obtener los datos a visualizar de los objetos entidad. Cuando los objetos entidad forman parte de la base de datos, es el SGBD el que se encarga de la entrada/salida a ficheros. El resto de la entrada/salida la manejan los programas de aplicación o las utilidades del SGBD. Muchos programas orientados a objetos tienen un cuarto tipo de clase: la clase contenedor. Estas clases contienen, o manejan, múltiples objetos creados a partir del mismo tipo de clase. También se conocen como agregaciones. Las clases contenedor mantienen los objetos en algún orden, los listan e incluso pueden permitir búsquedas en ellos.

Muchos SGBD orientados a objetos llaman a sus clases contenedor *extents* (extensiones) y su objetivo es permitir el acceso a todos los objetos creados a partir de la misma clase.

Jerarquía de clases.

En una base de datos existen objetos que responden a los mismos mensajes, utilizan los mismos métodos y tienen variables del mismo nombre y tipo. Sería inútil definir cada uno de estos objetos por separado por lo tanto se agrupan los objetos similares para que formen una clase, a cada uno de estos objetos se le llama instancia de su clase. Todos los objetos de su clase comparten una definición común, aunque difieran en los valores asignados a las variables.

Así que básicamente las bases de datos orientados a objetos tienen la finalidad de agrupar aquellos elementos que sean semejantes en las entidades para formar un clase, dejando por separado aquellas que no lo son en otra clase

Método: es una implementación específica de una operación ejecutable por una cierta clase.

Tipos de métodos.

Hay varios tipos de métodos que son comunes a la mayoría de las clases:

- **Constructores**. Un constructor es un método que tiene el mismo nombre que la clase. Se ejecuta cuando se crea un objeto de una clase. Por lo tanto, un constructor contiene instrucciones para inicializar las variables de un objeto.
- **Destructores.** Un destructor es un método que se utiliza para destruir un objeto. No todos los lenguajes orientados a objetos poseen destructores.

- **Accesores.** Un accesor es un método que devuelve el valor de un atributo privado de otro objeto. Así es cómo los objetos externos pueden acceder a los datos encapsulados.
- **Mutadores.** Un mutador es un método que almacena un nuevo valor en un atributo.

De este modo es cómo objetos externos pueden modificar los datos encapsulados.

Además, cada clase tendría otros métodos dependiendo del comportamiento específico que deba poseer.

Sobrecarga de métodos. Una de las características de las clases es que pueden tener métodos sobrecargados, que son métodos que tienen el mismo nombre pero que necesitan distintos datos para operar. Ya que los datos son distintos, las interfaces públicas de los métodos serán diferentes. Por ejemplo, consideremos una clase contenedor, TodosLosEmpleados, que agrega todos los objetos creados de la clase Empleado. Para que la clase contenedor sea útil, debe proporcionar alguna forma de buscar objetos de empleados específicos. Se puede querer buscar por número de empleado, por el nombre y los apellidos o por el número de teléfono. La clase contenedor TodosLosEmpleados tendría tres métodos llamados encuentra. Uno de los métodos requiere un entero como parámetro (el número de empleado), el segundo requiere dos cadenas (el nombre y los apellidos) y el tercero requiere una sola cadena (el número de teléfono). Aunque los tres métodos tienen el mismo nombre, poseen distintas interfaces públicas. La ventaja de la sobrecarga de los métodos es que

presentan una interface consistente al programador: siempre que quiera localizar a un empleado, debe utilizar el método encuentra.

Instancia

Se dice que cada objeto es una instancia de su clase. Toda clase describe un conjunto posiblemente finito de objetos individuales. Toda instancia de la clase posee su propio valor para cada uno de los atributos pero comparte los nombres de los atributos y las operaciones con las demás instancias de la clase. Todo objeto contiene una referencia implícita a su propia clase; ‘‘sabe la clase de cosa que es’’. Los objetos contienen los valores de los atributos (que lo distinguen de otros objetos) y una identidad

Operación

Es una acción o una transformación que se lleva a cabo o que se aplica a un objeto. Mover, justificar a la derecha son ejemplos de operaciones que se aplican a un objeto párrafo.

Nombres de clases, atributos y métodos.

En el mundo de la orientación a objetos hay cierta uniformidad en el modo de dar nombres a clases, atributos y métodos.

Los nombres de las clases empiezan por una letra mayúscula seguida de minúsculas.

Si el nombre tiene más de una palabra, se puede usar el carácter subrayado para separar palabras o bien empezar cada una con una letra mayúscula (Materia prima o MateriaPrima).

Los nombres de los atributos y de los métodos empiezan por minúscula y si tienen más de una palabra, utilizan el subrayado o la mayúscula (num empleado o numEmpleado).
Los métodos accesores empiezan por la palabra get seguida del nombre del atributo al que acceden (getNumEmpleado).
Los métodos mutadores empiezan por la palabra set seguida del nombre del atributo cuyo valor modifican (setNumEmpleado).

Herencia de atributos.

En ocasiones se necesita trabajar con clases que son similares pero no idénticas. Para ello es muy útil una de las características del paradigma orientado a objetos: la herencia. Una clase puede tener varias subclases que representan ocurrencias más específicas de la superclase. Por ejemplo, podemos tener la clase (superclase) Animal con sus atributos (nombre común, nombre científico, fecha de nacimiento y género) y las subclases Mamífero, Reptil y Pez, cada una con unos atributos específicos (Mamífero: peso, altura del hombro, raza y color; Reptil: longitud actual y longitud máxima; Pez: color). Por el hecho de ser subclases de Animal, heredan sus atributos. La relación que mantienen las subclases con la superclase es del tipo “es un”: un mamífero es un animal, un reptil es un animal y un pez es un animal. No todas las clases de una jerarquía se utilizan para crear objetos. Por ejemplo, nunca se crean objetos de la clase Animal, sino que se crean objetos de las clases Mamífero, Reptil o Pez. La clase Animal sólo se utiliza para recoger los atributos y métodos que son comunes a las tres subclases. Se dice que estas

clases son abstractas o virtuales. Las clases que se utilizan para crear objetos se denominan clases concretas.

Herencia múltiple.

Cuando una clase hereda de más de una superclase se tiene herencia múltiple.

Interfaces.

Algunos lenguajes orientados a objetos no soportan la herencia múltiple. En lugar de eso permiten que una clase se derive de una sola clase pero permiten que la clase implemente múltiples interfaces. Una interface es una especificación para una clase sin instrucciones en los métodos. Cada clase que implemente la interface proporcionaría las instrucciones para cada método de la misma. Una interface puede contener atributos y métodos, o bien solo atributos, o bien solo métodos.

Polimorfismo

Significa que una operación puede comportarse de modos distintos en distintas clases teniendo el mismo nombre de método. La operación *mover* se puede comportar distinto en las clases párrafo y pieza de ajedrez. En términos prácticos, el polimorfismo permite referirse a objetos de diferentes clases por medio del mismo elemento de programa y realizar la misma operación de formas diferentes, de acuerdo al objeto a que se hace referencia en cada momento. En el mundo real una operación es simplemente, una abstracción de comportamiento análogo entre distintas clases de objetos. Cada objeto

sabe llevar a cabo sus propias operaciones. Sin embargo, en un lenguaje orientado a objetos es este el que selecciona automáticamente el método correcto para implementar una operación basándose en el nombre de la operación y en la clase del objeto que está siendo afectado. El usuario de una operación no necesita ser consciente del número de métodos que existen para implementar una cierta operación polimórfica. Se pueden añadir clases sin modificar el código existente, siempre y cuando se proporcione métodos para todas las operaciones aplicables a las nuevas clases.

Combinación de datos y comportamientos.

El enfoque orientado a objetos tiene una sola jerarquía, jerarquía de clases. Unifica la jerarquía de estructuras de datos y jerarquía de procedimientos que presentan los enfoques convencionales. Cuando un objeto invoca una operación no necesita considerar cuántas implementaciones existen de una operación dada. El polimorfismo de operadores traslada la carga de decidir que implementación hay que utilizar llevándola del código que hace la llamada a la jerarquía de clases. El mantenimiento es mas sencillo porque el código que hace la llamada no necesita ser modificado cuando se añade una clase nueva. En el contexto de un sistema orientado a objetos la jerarquía de estructuras de datos es idéntica a la jerarquía de herencia de operaciones.

CAPÍTULO III

ESTUDIO DE MODELOS RELACIONALES

3.1 MODELO ENTIDAD RELACIÓN

El modelo entidad relación es el modelo conceptual más utilizado para el diseño conceptual de bases de datos. Fue introducido por Peter Chen en 1976. El modelo entidad relación está formado por un conjunto de conceptos que permiten describir la realidad mediante un conjunto de representaciones gráficas y lingüísticas.

Originalmente, el modelo entidad relación sólo incluía los conceptos de entidad, relación y atributo. Más tarde, se añadieron otros conceptos, como los atributos compuestos y las jerarquías de generalización, en lo que se ha denominado modelo entidad relación extendido. En este estudio vamos a utilizar la notación de Codd.

Entidad

Cualquier tipo de objeto o concepto sobre el que se recoge información: cosa, persona, concepto abstracto o suceso. Por ejemplo: coches, casas, empleados, clientes, empresas, conciertos, excursiones, etc. Las entidades se representan gráficamente mediante rectángulos y su nombre aparece en el interior, como se muestra en la Fig 1.Un nombre de entidad sólo puede aparecer una vez en el esquema conceptual.

Hay dos tipos de entidades: fuertes y débiles. Una *entidad débil* es una entidad cuya existencia depende de la existencia de otra entidad. Una *entidad fuerte* es una entidad que no es débil.

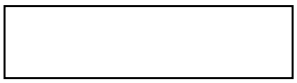


Fig 1 Representación de Entidad

Relación

Es una correspondencia o asociación entre dos o más entidades. Cada relación tiene un nombre que describe su función. Las relaciones se representan gráficamente mediante rombos y su nombre aparece en el interior, como se muestra en la Fig 2.

Las entidades que están involucradas en una determinada relación se denominan entidades participantes. El número de participantes en una relación es lo que se denomina grado de la relación. Por lo tanto, una relación en la que participan dos entidades es una relación binaria; si son tres las entidades participantes, la relación es ternaria; etc.

Una relación recursiva es una relación donde la misma entidad participa más de una vez en la relación con distintos papeles. El nombre de estos papeles es importante para determinar la función de cada participación.

La cardinalidad con la que una entidad participa en una relación especifica el número mínimo y el número máximo de correspondencias en las que puede tomar parte cada ocurrencia de dicha entidad. La participación de una entidad en una relación es

obligatoria (total) si la existencia de cada una de sus ocurrencias requiere la existencia de, al menos, una ocurrencia de la otra entidad participante. Si no, la participación es opcional (parcial). Las reglas que definen la cardinalidad de las relaciones son las reglas de negocio.

A veces, surgen problemas cuando se está diseñado un esquema conceptual. Estos problemas, denominados trampas, suelen producirse a causa de una mala interpretación en el significado de alguna relación, por lo que es importante comprobar que el esquema conceptual carece de dichas trampas. En general, para encontrar las trampas, hay que asegurarse de que se entiende completamente el significado de cada relación. Si no se entienden las relaciones, se puede crear un esquema que no represente fielmente la realidad.

Una de las trampas que pueden encontrarse ocurre cuando el esquema representa una relación entre entidades, pero el camino entre algunas de sus ocurrencias es ambiguo. El modo de resolverla es reestructurando el esquema para representar la asociación entre las entidades correctamente.

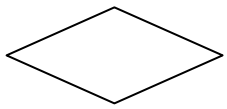


Fig 2 Representación de una Relación

Atributo

Es una característica de interés o un hecho sobre una entidad o sobre una relación. Los atributos representan las propiedades básicas de las entidades y de las relaciones. Toda

la información extensiva es portada por los atributos. Gráficamente, se representan mediante bolitas que cuelgan de las entidades o relaciones a las que pertenecen.

Cada atributo tiene un conjunto de valores asociados denominado dominio. El dominio define todos los valores posibles que puede tomar un atributo. Puede haber varios atributos definidos sobre un mismo dominio.

Los atributos pueden ser simples o compuestos. Un atributo simple es un atributo que tiene un solo componente, que no se puede dividir en partes más pequeñas que tengan un significado propio como se indica en la Fig 3. Un atributo compuesto es un atributo con varios componentes, cada uno con un significado por sí mismo. Un grupo de atributos se representa mediante un atributo compuesto cuando tienen afinidad en cuanto a su significado, o en cuanto a su uso. Un atributo compuesto se representa gráficamente mediante un óvalo como en la Fig 4.

Los atributos también pueden clasificarse en monovalentes o polivalentes. Un atributo monovalente es aquel que tiene un solo valor para cada ocurrencia de la entidad o relación a la que pertenece. Un atributo polivalente es aquel que tiene varios valores para cada ocurrencia de la entidad o relación a la que pertenece. A estos atributos también se les denomina multivaluados, y pueden tener un número máximo y un número mínimo de valores. La cardinalidad de un atributo indica el número mínimo y el número máximo de valores que puede tomar para cada ocurrencia de la entidad o relación a la que pertenece. El valor por omisión es (1).

Por último, los atributos pueden ser derivados. Un atributo derivado es aquel que representa un valor que se puede obtener a partir del valor de uno o varios atributos, que no necesariamente deben pertenecer a la misma entidad o relación.

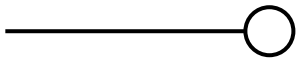


Fig 3 Representación de Atributo

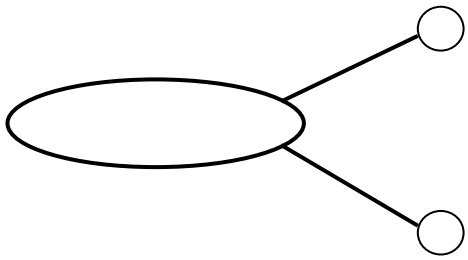


Fig 4 Representación de un Atributo compuesto

Identificador

Un identificador de una entidad es un atributo o conjunto de atributos que determina de modo único cada ocurrencia de esa entidad como en la Fig 5. Un identificador de una entidad debe cumplir dos condiciones:

1. No pueden existir dos ocurrencias de la entidad con el mismo valor del identificador.
2. Si se omite cualquier atributo del identificador, la condición anterior deja de cumplirse.

Toda entidad tiene al menos un identificador y puede tener varios identificadores alternativos. Las relaciones no tienen identificadores.

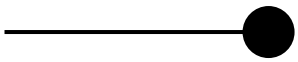


Fig 5 Representación de un identificador

Jerarquía de generalización

Una entidad E es una generalización de un grupo de entidades E^1,E^2,...E^n, si cada ocurrencia de cada una de esas entidades es también una ocurrencia de E. Todas las propiedades de la entidad genérica E son heredadas por las subentidades.

Cada jerarquía es total o parcial, y exclusiva o superpuesta. Una jerarquía es total si cada ocurrencia de la entidad genérica corresponde al menos con una ocurrencia de alguna subentidad. Es parcial si existe alguna ocurrencia de la entidad genérica que no corresponde con ninguna ocurrencia de ninguna subentidad. Una jerarquía es exclusiva si cada ocurrencia de la entidad genérica corresponde, como mucho, con una ocurrencia de una sola de las subentidades. Es superpuesta si existe alguna ocurrencia de la entidad genérica que corresponde a ocurrencias de dos o más subentidades diferentes.

Un subconjunto es un caso particular de generalización con una sola entidad como subentidad. Un subconjunto siempre es una jerarquía parcial y exclusiva.

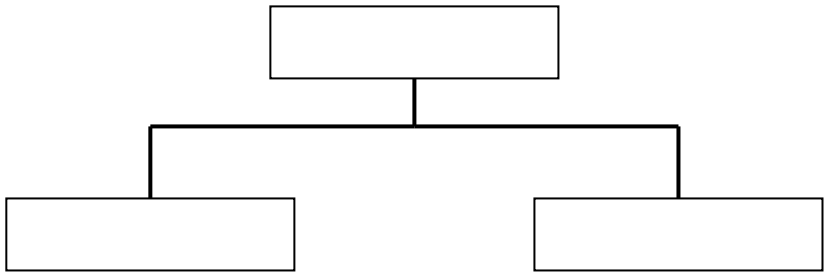


Fig 6 Representación de Jerarquía de generalización

3.1.1.- METODOLOGÍA DE DISEÑO CONCEPTUAL

El primer paso en el diseño de una base de datos es la producción del esquema conceptual. Normalmente, se construyen varios esquemas conceptuales, cada uno para representar las distintas visiones que los usuarios tienen de la información. Cada una de estas visiones suelen corresponder a las diferentes áreas funcionales de la empresa como, por ejemplo, producción, ventas, recursos humanos, etc.

Estas visiones de la información, denominadas vistas, se pueden identificar de varias formas. Una opción consiste en examinar los diagramas de flujo de datos, que se pueden haber producido previamente, para identificar cada una de las áreas funcionales. La otra opción consiste en entrevistar a los usuarios, examinar los procedimientos, los informes y los formularios, y también observar el funcionamiento de la empresa.

A los esquemas conceptuales correspondientes a cada vista de usuario se les denomina esquemas conceptuales locales. Cada uno de estos esquemas se compone de entidades, relaciones, atributos, dominios de atributos e identificadores. El esquema conceptual también tendrá una documentación, que se irá produciendo durante su desarrollo. Las tareas a realizar en el diseño conceptual son las siguientes:

1. Identificar las entidades.
2. Identificar las relaciones.
3. Identificar los atributos y asociarlos a entidades y relaciones.
4. Determinar los dominios de los atributos.
5. Determinar los identificadores.
6. Determinar las jerarquías de generalización (si las hay).

7. Dibujar el diagrama entidad-relación.
8. Revisar el esquema conceptual local con el usuario.

3.1.1.1. Identificar las entidades

En primer lugar hay que definir los principales objetos que interesan al usuario. Estos objetos serán las entidades. Una forma de identificar las entidades es examinar las especificaciones de requisitos de usuario. En estas especificaciones se buscan los nombres o los sintagmas nominales que se mencionan (por ejemplo: número de empleado, nombre de empleado, número de inmueble, dirección del inmueble, alquiler, número de habitaciones). También se buscan objetos importantes como personas, lugares o conceptos de interés, excluyendo aquellos nombres que sólo son propiedades de otros objetos. Por ejemplo, se pueden agrupar el número de empleado y el nombre de empleado en una entidad denominada empleado, y agrupar número de inmueble, dirección del inmueble, alquiler y número de habitaciones en otra entidad denominada inmueble.

Otra forma de identificar las entidades es buscar aquellos objetos que existen por sí mismos. Por ejemplo, empleado es una entidad porque los empleados existen, sepamos o no sus nombres, direcciones y teléfonos. Siempre que sea posible, el usuario debe colaborar en la identificación de las entidades.

A veces, es difícil identificar las entidades por la forma en que aparecen en las especificaciones de requisitos. Los usuarios, a veces, hablan utilizando ejemplos o

analogías. En lugar de hablar de empleados en general, hablan de personas concretas, o bien, hablan de los puestos que ocupan esas personas.

Para liarlo aún más, los usuarios usan, muchas veces, sinónimos y homónimos. Dos palabras son sinónimos cuando tienen el mismo significado. Los homónimos ocurren cuando la misma palabra puede tener distintos significados dependiendo del contexto.

No siempre es obvio saber si un objeto es una entidad, una relación o un atributo. Por ejemplo ¿cómo se podría clasificar matrimonio? Pues de cualquiera de las tres formas. El análisis es subjetivo, por lo que distintos diseñadores pueden hacer distintas interpretaciones, aunque todas igualmente válidas. Todo depende de la opinión y la experiencia de cada uno. Los diseñadores de bases de datos deben tener una visión selectiva y clasificar las cosas que observan dentro del contexto de la empresa u organización. A partir de unas especificaciones de usuario es posible que no se pueda deducir un conjunto único de entidades, pero después de varias iteraciones del proceso de análisis, se llegará a obtener un conjunto de entidades que sean adecuadas para el sistema que se ha de construir.

Conforme se van identificando las entidades, se les dan nombres que tengan un significado y que sean obvias para el usuario. Los nombres de las entidades y sus descripciones se anotan en el diccionario de datos. Cuando sea posible, se debe anotar también el número aproximado de ocurrencias de cada entidad. Si una entidad se conoce por varios nombres, éstos se deben anotar en el diccionario de datos como alias o sinónimos.

3.1.1.2. Identificar las relaciones

Una vez definidas las entidades, se deben definir las relaciones existentes entre ellas. Del mismo modo que para identificar las entidades se buscaban nombres en las especificaciones de requisitos, para identificar las relaciones se suelen buscar las expresiones verbales (por ejemplo: oficina tiene empleados, empleado gestiona inmueble, cliente visita inmueble). Si las especificaciones de requisitos reflejan estas relaciones es porque son importantes para la empresa y, por lo tanto, se deben reflejar en el esquema conceptual.

Pero sólo interesan las relaciones que son necesarias. En el ejemplo anterior, se han identificado las relaciones empleado gestiona inmueble y cliente visita inmueble. Se podría pensar en incluir una relación entre empleado y cliente: empleado atiende a cliente, pero observando las especificaciones de requisitos no parece que haya interés en modelar tal relación.

La mayoría de las relaciones son binarias (entre dos entidades), pero no hay que olvidar que también puede haber relaciones en las que participen más de dos entidades, así como relaciones recursivas.

Es muy importante repasar las especificaciones para comprobar que todas las relaciones, explícitas o implícitas, se han encontrado. Si se tienen pocas entidades, se puede comprobar por parejas si hay alguna relación entre ellas. De todos modos, las relaciones que no se identifican ahora se suelen encontrar cuando se valida el esquema con las transacciones que debe soportar.

Una vez identificadas todas las relaciones, hay que determinar la cardinalidad mínima y máxima con la que participa cada entidad en cada una de ellas. De este modo, el esquema representa de un modo más explícito la semántica de las relaciones. La cardinalidad es un tipo de restricción que se utiliza para comprobar y mantener la calidad de los datos. Estas restricciones son aserciones sobre las entidades que se pueden aplicar cuando se actualiza la base de datos para determinar si las actualizaciones violan o no las reglas establecidas sobre la semántica de los datos.

Conforme se van identificando las relaciones, se les van asignando nombres que tengan significado para el usuario. En el diccionario de datos se anotan los nombres de las relaciones, su descripción y las cardinalidades con las que participan las entidades en ellas.

3.1.1.3. Identificar los atributos y asociarlos a entidades y relaciones

Al igual que con las entidades, se buscan nombres en las especificaciones de requisitos. Son atributos los nombres que identifican propiedades, cualidades, identificadores o características de entidades o relaciones.

Lo más sencillo es preguntarse, para cada entidad y cada relación, ¿qué información se quiere saber de ...? La respuesta a esta pregunta se debe encontrar en las especificaciones de requisitos. Pero, en ocasiones, será necesario preguntar a los usuarios para que aclaren los requisitos. Desgraciadamente, los usuarios pueden dar respuestas a esta pregunta que también contengan otros conceptos, por lo que hay que considerar sus respuestas con mucho cuidado.

Al identificar los atributos, hay que tener en cuenta si son simples o compuestos. Por ejemplo, el atributo dirección puede ser simple, teniendo la dirección completa como un solo valor: `San Rafael 45, Almazora'; o puede ser un atributo compuesto, formado por la calle (`San Rafael'), el número (`45') y la población (`Almazora'). El escoger entre atributo simple o compuesto depende de los requisitos del usuario. Si el usuario no necesita acceder a cada uno de los componentes de la dirección por separado, se puede representar como un atributo simple. Pero si el usuario quiere acceder a los componentes de forma individual, entonces se debe representar como un atributo compuesto.

También se deben identificar los atributos derivados o calculados, que son aquellos cuyo valor se puede calcular a partir de los valores de otros atributos. Por ejemplo, el número de empleados de cada oficina, la edad de los empleados o el número de inmuebles que gestiona cada empleado. Algunos diseñadores no representan los atributos derivados en los esquemas conceptuales. Si se hace, se debe indicar claramente que el atributo es derivado y a partir de qué atributos se obtiene su valor. Donde hay que considerar los atributos derivados es en el diseño físico.

Cuando se están identificando los atributos, se puede descubrir alguna entidad que no se ha identificado previamente, por lo que hay que volver al principio introduciendo esta entidad y viendo si se relaciona con otras entidades.

Es muy útil elaborar una lista de atributos e ir eliminándolos de la lista conforme se vayan asociando a una entidad o relación. De este modo, uno se puede asegurar de que

cada atributo se asocia a una sola entidad o relación, y que cuando la lista se ha acabado, se han asociado todos los atributos.

Hay que tener mucho cuidado cuando parece que un mismo atributo se debe asociar a varias entidades. Esto puede ser por una de las siguientes causas:

- Se han identificado varias entidades, como director, supervisor y administrativo, cuando, de hecho, pueden representarse como una sola entidad denominada empleado. En este caso, se puede escoger entre introducir una jerarquía de generalización, o dejar las entidades que representan cada uno de los puestos de empleado.
- Se ha identificado una relación entre entidades. En este caso, se debe asociar el atributo a una sola de las entidades y hay que asegurarse de que la relación ya se había identificado previamente. Si no es así, se debe actualizar la documentación para recoger la nueva relación.

Conforme se van identificando los atributos, se les asignan nombres que tengan significado para el usuario. De cada atributo se debe anotar la siguiente información:

- Nombre y descripción del atributo.
- Alias o sinónimos por los que se conoce al atributo.
- Tipo de dato y longitud.
- Valores por defecto del atributo (si se especifican).
- Si el atributo siempre va a tener un valor (si admite o no nulos).
- Si el atributo es compuesto y, en su caso, qué atributos simples lo forman.

- Si el atributo es derivado y, en su caso, cómo se calcula su valor.
- Si el atributo es multievaluado.

3.1.1.4. Determinar los dominios de los atributos

El dominio de un atributo es el conjunto de valores que puede tomar el atributo. Por ejemplo el dominio de los números de oficina son las tiras de hasta tres caracteres en donde el primero es una letra y el siguiente o los dos siguientes son dígitos en el rango de 1 a 99; el dominio de los números de teléfono y los números de fax son las tiras de 9 dígitos.

Un esquema conceptual está completo si incluye los dominios de cada atributo: los valores permitidos para cada atributo, su tamaño y su formato. También se puede incluir información adicional sobre los dominios como, por ejemplo, las operaciones que se pueden realizar sobre cada atributo, qué atributos pueden compararse entre sí o qué atributos pueden combinarse con otros. Aunque sería muy interesante que el sistema final respetara todas estas indicaciones sobre los dominios, esto es todavía una línea abierta de investigación.

Toda la información sobre los dominios se debe anotar también en el diccionario de datos.

3.1.1.5. Determinar los identificadores

Cada entidad tiene al menos un identificador. En este paso, se trata de encontrar todos los identificadores de cada una de las entidades. Los identificadores pueden ser simples

o compuestos. De cada entidad se escogerá uno de los identificadores como clave primaria en la fase del diseño lógico.

Cuando se determinan los identificadores es fácil darse cuenta de si una entidad es fuerte o débil. Si una entidad tiene al menos un identificador, es fuerte (otras denominaciones son padre, propietaria o dominante). Si una entidad no tiene atributos que le sirvan de identificador, es débil (otras denominaciones son hijo, dependiente o subordinada).

Todos los identificadores de las entidades se deben anotar en el diccionario de datos.

3.1.1.6. Determinar las jerarquías de generalización

En este paso hay que observar las entidades que se han identificado hasta el momento. Hay que ver si es necesario reflejar las diferencias entre distintas ocurrencias de una entidad, con lo que surgirán nuevas subentidades de esta entidad genérica; o bien, si hay entidades que tienen características en común y que realmente son subentidades de una nueva entidad genérica.

En cada jerarquía hay que determinar si es total o parcial y exclusiva o superpuesta.

3.1.1.7. Dibujar el diagrama entidad-relación

Una vez identificados todos los conceptos, se puede dibujar el diagrama entidad-relación correspondiente a una de las vistas de los usuarios. Se obtiene así un esquema conceptual local.

3.1.1.8. Revisar el esquema conceptual local con el usuario

Antes de dar por finalizada la fase del diseño conceptual, se debe revisar el esquema conceptual local con el usuario. Este esquema está formado por el diagrama entidad-relación y toda la documentación que describe el esquema. Si se encuentra alguna anomalía, hay que corregirla haciendo cambios oportunos, por lo que posiblemente haya que repetir alguno de los pasos anteriores. Este proceso debe repetirse hasta que se esté seguro de que el esquema conceptual es lo que la empresa está tratando de modelar.

3.1.2 DISEÑO LÓGICO DE BASES DE DATOS

A continuación describiremos los pasos para llevar a cabo el diseño lógico. Ya que aquí se trata el diseño de bases de datos relacionales, en esta etapa se obtiene un conjunto de relaciones (tablas) que representen los datos de interés.

El objetivo del diseño lógico es convertir los esquemas conceptuales locales en un esquema lógico global que se ajuste al modelo de SGBD sobre el que se vaya a implementar el sistema. Mientras que el objetivo fundamental del diseño conceptual es la compleción y expresividad de los esquemas conceptuales locales, el objetivo del diseño lógico es obtener una representación que use, del modo más eficiente posible, los recursos que el modelo de SGBD posee para estructurar los datos y para modelar las restricciones

Los modelos de bases de datos más extendidos son el modelo relacional, el modelo de red y el modelo jerárquico.

El modelo relacional carecen de ciertos rasgos de abstracción que se usan en los modelos conceptuales. Por lo tanto, un primer paso en la fase del diseño lógico consistirá en la conversión de esos mecanismos de representación de alto nivel en términos de las estructuras de bajo nivel disponibles en el modelo relacional.

3.1.2.1 METODOLOGÍA DE DISEÑO LÓGICO EN EL MODELO RELACIONAL

La metodología que se va a seguir para el diseño lógico en el modelo relacional tiene dos fases, cada una de ellas compuesta por varios pasos que se detallan a continuación.

- Construir y validar los esquemas lógicos locales para cada vista de usuario.
    1. Convertir los esquemas conceptuales locales en esquemas lógicos locales.
    2. Derivar un conjunto de relaciones (tablas) para cada esquema lógico local.
    3. Validar cada esquema mediante la normalización.
    4. Validar cada esquema frente a las transacciones del usuario.
    5. Dibujar el diagrama entidad-relación.
    6. Definir las restricciones de integridad.
    7. Revisar cada esquema lógico local con el usuario correspondiente.
- Construir y validar el esquema lógico global.
    1. Mezclar los esquemas lógicos locales en un esquema lógico global.
    2. Validar el esquema lógico global.

3. Estudiar el crecimiento futuro.
4. Dibujar el diagrama entidad-relación final.
5. Revisar el esquema lógico global con los usuarios.

En la primera fase, se construyen los esquemas lógicos locales para cada vista de usuario y se validan. En esta fase se refinan los esquemas conceptuales creados durante el diseño conceptual, eliminando las estructuras de datos que no se pueden implementar de manera directa sobre el modelo que soporta el SGBD, en el caso que nos ocupa, el modelo relacional. Una vez hecho esto, se obtiene un primer esquema lógico que se valida mediante la normalización y frente a las transacciones que el sistema debe llevar a cabo, tal y como se refleja en las especificaciones de requisitos de usuario. El esquema lógico ya validado se puede utilizar como base para el desarrollo de prototipos. Una vez finalizada esta fase, se dispone de un esquema lógico para cada vista de usuario que es correcto, comprensible y sin ambigüedad.

3.1.2.1.1. Convertir los esquemas conceptuales locales en esquemas lógicos locales

En este paso, se eliminan de cada esquema conceptual las estructuras de datos que los sistemas relacionales no modelan directamente:

(a)

Eliminar las relaciones de muchos a muchos, sustituyendo cada una de ellas por una nueva entidad intermedia y dos relaciones de uno a muchos de esta nueva entidad con las entidades originales. La nueva entidad será débil, ya que sus ocurrencias dependen de la existencia de ocurrencias en las entidades originales.

(b)

Eliminar las relaciones entre tres o más entidades, sustituyendo cada una de ellas por una nueva entidad (débil) intermedia que se relaciona con cada una de las entidades originales. La cardinalidad de estas nuevas relaciones binarias dependerá de su significado.

(c)

Eliminar las relaciones recursivas, sustituyendo cada una de ellas por una nueva entidad (débil) y dos relaciones binarias de esta nueva entidad con la entidad original. La cardinalidad de estas relaciones dependerá de su significado.

(d)

Eliminar las relaciones con atributos, sustituyendo cada una de ellas por una nueva entidad (débil) y las relaciones binarias correspondientes de esta nueva entidad con las entidades originales. La cardinalidad de estas relaciones dependerá del tipo de la relación original y de su significado.

(e)

Eliminar los atributos multivaluados, sustituyendo cada uno de ellos por una nueva entidad (débil) y una relación binaria de uno a muchos con la entidad original.

(f)

Revisar las relaciones de uno a uno, ya que es posible que se hayan identificado dos entidades que representen el mismo objeto (sinónimos). Si así fuera, ambas entidades deben integrarse en una sola.

(g)

Eliminar las relaciones redundantes. Una relación es redundante cuando se puede obtener la misma información que ella aporta mediante otras relaciones. El hecho de que haya dos caminos diferentes entre dos entidades no implica que uno de los caminos corresponda a una relación redundante, eso dependerá del significado de cada relación.

Una vez finalizado este paso, es más correcto referirse a los esquemas conceptuales locales refinados como esquemas lógicos locales, ya que se adaptan al modelo de base de datos que soporta el SGBD escogido.

3.1.2.1.2. Derivar un conjunto de relaciones (tablas) para cada esquema lógico local

En este paso, se obtiene un conjunto de relaciones (tablas) para cada uno de los esquemas lógicos locales en donde se representen las entidades y relaciones entre entidades, que se describen en cada una de las vistas que los usuarios tienen de la empresa. Cada relación de la base de datos tendrá un nombre, y el nombre de sus atributos aparecerá, a continuación, entre paréntesis. El atributo o atributos que forman la clave primaria se subrayan. Las claves ajenas, mecanismo que se utiliza para representar las relaciones entre entidades en el modelo relacional, se especifican aparte indicando la relación (tabla) a la que hacen referencia.

A continuación, se describe cómo las relaciones (tablas) del modelo relacional representan las entidades y relaciones que pueden aparecer en los esquemas lógicos.

(a)

Entidades fuertes. Crear una relación para cada entidad fuerte que incluya todos sus atributos simples. De los atributos compuestos incluir sólo sus componentes. Cada uno de los identificadores de la entidad será una clave candidata. De entre las claves candidatas hay que escoger la clave primaria; el resto serán claves alternativas. Para escoger la clave primaria entre las claves candidatas se pueden seguir estas indicaciones:

- Escoger la clave candidata que tenga menos atributos.
- Escoger la clave candidata cuyos valores no tengan probabilidad de cambiar en el futuro.
- Escoger la clave candidata cuyos valores no tengan probabilidad de perder la unicidad en el futuro.
- Escoger la clave candidata con el mínimo número de caracteres (si es de tipo texto).
- Escoger la clave candidata más fácil de utilizar desde el punto de vista de los usuarios.

(b)

Entidades débiles. Crear una relación para cada entidad débil incluyendo todos sus atributos simples. De los atributos compuestos incluir sólo sus componentes. Añadir una clave ajena a la entidad de la que depende. Para ello, se incluye la clave primaria de la relación que representa a la entidad padre en la nueva

relación creada para la entidad débil. A continuación, determinar la clave primaria de la nueva relación.

(c)

Relaciones binarias de uno a uno. Para cada relación binaria se incluyen los atributos de la clave primaria de la entidad padre en la relación (tabla) que representa a la entidad hijo, para actuar como una clave ajena. La entidad hijo es la que participa de forma total (obligatoria) en la relación, mientras que la entidad padre es la que participa de forma parcial (opcional). Si las dos entidades participan de forma total o parcial en la relación, la elección de padre e hijo es arbitraria. Además, en caso de que ambas entidades participen de forma total en la relación, se tiene la opción de integrar las dos entidades en una sola relación (tabla). Esto se suele hacer si una de las entidades no participa en ninguna otra relación.

(d)

Relaciones binarias de uno a muchos. Como en las relaciones de uno a uno, se incluyen los atributos de la clave primaria de la entidad padre en la relación (tabla) que representa a la entidad hijo, para actuar como una clave ajena. Pero ahora, la entidad padre es la de ``la parte del muchos" (cada padre tiene muchos hijos), mientras que la entidad hijo es la de ``la parte del uno" (cada hijo tiene un solo padre).

(e)

Jerarquías de generalización. En las jerarquías, se denomina entidad padre a la entidad genérica y entidades hijo a las subentidades. Hay tres opciones distintas para representar las jerarquías. La elección de la más adecuada se hará en función de su tipo (total/parcial, exclusiva/superpuesta).

1. Crear una relación por cada entidad. Las relaciones de las entidades hijo heredan como clave primaria la de la entidad padre. Por lo tanto, la clave primaria de las entidades hijo es también una clave ajena al padre. Esta opción sirve para cualquier tipo de jerarquía, total o parcial y exclusiva o superpuesta.
2. Crear una relación por cada entidad hijo, heredando los atributos de la entidad padre. Esta opción sólo sirve para jerarquías totales y exclusivas.
3. Integrar todas las entidades en una relación, incluyendo en ella los atributos de la entidad padre, los atributos de todos los hijos y un atributo discriminativo para indicar el caso al cual pertenece la entidad en consideración. Esta opción sirve para cualquier tipo de jerarquía. Si la jerarquía es superpuesta, el atributo discriminativo será multievaluado.

Una vez obtenidas las relaciones con sus atributos, claves primarias y claves ajenas, sólo queda actualizar el diccionario de datos con los nuevos atributos que se hayan identificado en este paso.

3.1.2.1.3. Validar cada esquema mediante la normalización

La normalización se utiliza para mejorar el esquema lógico, de modo que satisfaga ciertas restricciones que eviten la duplicidad de datos. La normalización garantiza que el esquema resultante se encuentra más próximo al modelo de la empresa, que es consistente y que tiene la mínima redundancia y la máxima estabilidad.

La normalización es un proceso que permite decidir a qué entidad pertenece cada atributo. Uno de los conceptos básicos del modelo relacional es que los atributos se agrupan en relaciones (tablas) porque están relacionados a nivel lógico. En la mayoría de las ocasiones, una base de datos normalizada no proporciona la máxima eficiencia, sin embargo, el objetivo ahora es conseguir una base de datos normalizada por las siguientes razones:

- Un esquema normalizado organiza los datos de acuerdo a sus dependencias funcionales, es decir, de acuerdo a sus relaciones lógicas.
- El esquema lógico no tiene porqué ser el esquema final. Debe representar lo que el diseñador entiende sobre la naturaleza y el significado de los datos de la empresa. Si se establecen unos objetivos en cuanto a prestaciones, el diseño físico cambiará el esquema lógico de modo adecuado. Una posibilidad es que algunas relaciones normalizadas se desnormalicen. Pero la desnormalización no implica que se haya malgastado tiempo normalizando, ya que mediante este proceso el diseñador aprende más sobre el significado de los datos. De hecho, la normalización obliga a entender completamente cada uno de los atributos que se han de representar en la base de datos.

- Un esquema normalizado es robusto y carece de redundancias, por lo que está libre de ciertas anomalías que éstas pueden provocar cuando se actualiza la base de datos.
- Los equipos informáticos de hoy en día son mucho más potentes, por lo que en ocasiones es más razonable implementar bases de datos fáciles de manejar (las normalizadas), a costa de un tiempo adicional de proceso.
- La normalización produce bases de datos con esquemas flexibles que pueden extenderse con facilidad.

El objetivo de este paso es obtener un conjunto de relaciones que se encuentren en la forma normal de Boyce-Codd. Para ello, hay que pasar por la primera, segunda y tercera formas normales que se describen a continuación.

3.1.2.1.3.1. Normalización

La normalización es una técnica para diseñar la estructura lógica de los datos de un sistema de información en el modelo relacional, desarrollada por E. F. Codd en 1972. Es una estrategia de diseño de abajo a arriba: se parte de los atributos y éstos se van agrupando en relaciones (tablas) según su afinidad. Aquí no se utilizará la normalización como una técnica de diseño de bases de datos, sino como una etapa posterior a la correspondencia entre el esquema conceptual y el esquema lógico, que elimine las dependencias entre atributos no deseadas. Las ventajas de la normalización son las siguientes:

- Evita anomalías en inserciones, modificaciones y borrados.
- Mejora la independencia de datos.
- No establece restricciones artificiales en la estructura de los datos.

Uno de los conceptos fundamentales en la normalización es el de *dependencia funcional*. Una dependencia funcional es una relación entre atributos de una misma relación (tabla). Si **x** e **y** son atributos de la relación R, se dice que **y** es funcionalmente dependiente de **x** (se denota por $\mathbf{x} \rightarrow \mathbf{y}$) si cada valor de **x** tiene asociado un solo valor de **y** (**x** e **y** pueden constar de uno o varios atributos) .A **x** se le denomina *determinante*, ya que **x** determina el valor de **y**. Se dice que el atributo **y** es *completamente dependiente* de **x** si depende funcionalmente de **x** y no depende de ningún subconjunto de **x**.

La dependencia funcional es una noción semántica. Si hay o no dependencias funcionales entre atributos no lo determina una serie abstracta de reglas, sino, más bien, los modelos mentales del usuario y las reglas de negocio de la organización o empresa para la que se desarrolla el sistema de información. Cada dependencia funcional es una clase especial de regla de integridad y representa una relación de uno a muchos.

En el proceso de normalización se debe ir comprobando que cada relación (tabla) cumple una serie de reglas que se basan en la clave primaria y las dependencias funcionales. Cada regla que se cumple aumenta el grado de normalización. Si una regla no se cumple, la relación se debe descomponer en varias relaciones que sí la cumplan.

La normalización se lleva a cabo en una serie pasos. Cada paso corresponde a una forma normal que tiene unas propiedades. Conforme se va avanzando en la normalización, las

relaciones tienen un formato más estricto (más fuerte) y, por lo tanto, son menos vulnerables a las anomalías de actualización. El modelo relacional sólo requiere un conjunto de relaciones en primera forma normal. Las restantes formas normales son opcionales. Sin embargo, para evitar las anomalías de actualización, es recomendable llegar al menos a la tercera forma normal.

Primera forma normal (1FN)

Una relación está en primera forma normal si, y sólo si, todos los dominios de la misma contienen valores atómicos, es decir, no hay grupos repetitivos. Si se ve la relación gráficamente como una tabla, estará en 1FN si tiene un solo valor en la intersección de cada fila con cada columna.

Si una relación no está en 1FN, hay que eliminar de ella los grupos repetitivos. Un grupo repetitivo será el atributo o grupo de atributos que tiene múltiples valores para cada tupla de la relación. Hay dos formas de eliminar los grupos repetitivos. En la primera, se repiten los atributos con un solo valor para cada valor del grupo repetitivo. De este modo, se introducen redundancias ya que se duplican valores, pero estas redundancias se eliminarán después mediante las restantes formas normales. La segunda forma de eliminar los grupos repetitivos consiste en poner cada uno de ellos en una relación aparte, heredando la clave primaria de la relación en la que se encontraban.

Un conjunto de relaciones se encuentra en 1FN si ninguna de ellas tiene grupos repetitivos.

Segunda forma normal (2FN)

Una relación está en segunda forma normal si, y sólo si, está en 1FN y, además, cada atributo que no está en la clave primaria es completamente dependiente de la clave primaria.

La 2FN se aplica a las relaciones que tienen claves primarias compuestas por dos o más atributos. Si una relación está en 1FN y su clave primaria es simple (tiene un solo atributo), entonces también está en 2FN. Las relaciones que no están en 2FN pueden sufrir anomalías cuando se realizan actualizaciones.

Para pasar una relación en 1FN a 2FN hay que eliminar las dependencias parciales de la clave primaria. Para ello, se eliminan los atributos que son funcionalmente dependientes y se ponen en una nueva relación con una copia de su determinante (los atributos de la clave primaria de los que dependen).

Tercera forma normal (3FN)

Una relación está en tercera forma normal si, y sólo si, está en 2FN y, además, cada atributo que no está en la clave primaria no depende transitivamente de la clave primaria. La dependencia $\mathbf{x \rightarrow z}$ es transitiva si existen las dependencias $\mathbf{x \rightarrow y}$, $\mathbf{y \rightarrow z}$, siendo **x, y**, atributos o conjuntos de atributos de una misma relación.

Aunque las relaciones en 2FN tienen menos redundancias que las relaciones en 1FN, todavía pueden sufrir anomalías frente a las actualizaciones. Para pasar una relación de 2FN a 3FN hay que eliminar las dependencias transitivas. Para ello, se eliminan los

atributos que dependen transitivamente y se ponen en una nueva relación con una copia de su determinante (el atributo o atributos no clave de los que dependen).

Forma normal de Boyce-Codd (BCFN)

Una relación está en la forma normal de Boyce-Codd si, y sólo si, todo determinante es una clave candidata.

La 2FN y la 3FN eliminan las dependencias parciales y las dependencias transitivas de la clave primaria. Pero este tipo de dependencias todavía pueden existir sobre otras claves candidatas, si éstas existen. La BCFN es más fuerte que la 3FN, por lo tanto, toda relación en BCFN está en 3FN.

La violación de la BCFN es poco frecuente ya que se da bajo ciertas condiciones que raramente se presentan. Se debe comprobar si una relación viola la BCFN si tiene dos o más claves candidatas compuestas que tienen al menos un atributo en común

3.1.2.1.4. Validar cada esquema frente a las transacciones del usuario

El objetivo de este paso es validar cada esquema lógico local para garantizar que puede soportar las transacciones requeridas por los correspondientes usuarios. Estas transacciones se encontrarán en las especificaciones de requisitos de usuario. Lo que se debe hacer es tratar de realizar las transacciones de forma manual utilizando el diagrama entidad-relación, el diccionario de datos y las conexiones que establecen las claves ajenas de las relaciones (tablas). Si todas las transacciones se pueden realizar, el

esquema queda validado. Pero si alguna transacción no se puede realizar, seguramente será porque alguna entidad, relación o atributo no se ha incluido en el esquema.

3.1.2.1.5. Dibujar el diagrama entidad-relación

En este momento, se puede dibujar el diagrama entidad-relación final para cada vista de usuario que recoja la representación lógica de los datos desde su punto de vista. Este diagrama habrá sido validado mediante la normalización y frente a las transacciones de los usuarios.

3.1.2.1.6. Definir las restricciones de integridad

Las restricciones de integridad son reglas que se quieren imponer para proteger la base de datos, de modo que no pueda llegar a un estado inconsistente. Hay cinco tipos de restricciones de integridad.

(a)

Datos requeridos. Algunos atributos deben contener valores en todo momento, es decir, no admiten nulos.

(b)

Restricciones de dominios. Todos los atributos tienen un dominio asociado, que es el conjunto los valores que cada atributo puede tomar.

(c)

Integridad de entidades. El identificador de una entidad no puede ser nulo, por lo tanto, las claves primarias de las relaciones (tablas) no admiten nulos.

(d)

Integridad referencial. Una clave ajena enlaza cada tupla de la relación hijo con la tupla de la relación padre que tiene el mismo valor en su clave primaria. La integridad referencial dice que si una clave ajena tiene un valor (si es no nula), ese valor debe ser uno de los valores de la clave primaria a la que referencia. Hay varios aspectos a tener en cuenta sobre las claves ajenas para lograr que se cumpla la integridad referencial.

1. ¿Admite nulos la clave ajena? Cada clave ajena expresa una relación. Si la participación de la entidad hijo en la relación es total, entonces la clave ajena no admite nulos; si es parcial, la clave ajena debe aceptar nulos.
2. ¿Qué hacer cuando se quiere borrar una ocurrencia de la entidad padre que tiene algún hijo? O lo que es lo mismo, ¿qué hacer cuando se quiere borrar una tupla que está siendo referenciada por otra tupla a través de una clave ajena? Hay varias respuestas posibles:
    - *Restringir:* no se pueden borrar tuplas que están siendo referenciadas por otras tuplas.
    - *Propagar:* se borra la tupla deseada y se propaga el borrado a todas las tuplas que le hacen referencia.
    - *Anular:* se borra la tupla deseada y todas las referencias que tenía se ponen, automáticamente, a nulo (esta respuesta sólo es válida si la clave ajena acepta nulos).

- *Valor por defecto:* se borra la tupla deseada y todas las referencias toman, automáticamente, el valor por defecto (esta respuesta sólo es válida si se ha especificado un valor por defecto para la clave ajena).
- *No comprobar:* se borra la tupla deseada y no se hace nada para garantizar que se sigue cumpliendo la integridad referencial.

3. ¿Qué hacer cuando se quiere modificar la clave primaria de una tupla que está siendo referenciada por otra tupla a través de una clave ajena? Las respuestas posibles son las mismas que en el caso anterior. Cuando se escoge propagar, se actualiza la clave primaria en la tupla deseada y se propaga el cambio a los valores de clave ajena que le hacían referencia.

(e)

Reglas de negocio. Cualquier operación que se realice sobre los datos debe cumplir las restricciones que impone el funcionamiento de la empresa.

Todas las restricciones de integridad establecidas en este paso se deben reflejar en el diccionario de datos para que puedan ser tenidas en cuenta durante la fase del diseño físico.

3.1.2.1.7. Revisar cada esquema lógico local con el usuario correspondiente

Para garantizar que cada esquema lógico local es una fiel representación de la vista del usuario lo que se debe hacer es comprobar con él que lo reflejado en el esquema y en la documentación es correcto y está completo.

Relación entre el esquema lógico y los diagramas de flujo de datos

El esquema lógico refleja la estructura de los datos a almacenar que maneja la empresa. Un diagrama de flujo de datos muestra cómo se mueven los datos en la empresa y los almacenes en donde se guardan. Si se han utilizado diagramas de flujo de datos para modelar las especificaciones de requisitos de usuario, se pueden utilizar para comprobar la consistencia y completitud del esquema lógico desarrollado. Para ello:

- Cada almacén de datos debe corresponder con una o varias entidades completas.
- Los atributos en los flujos de datos deben corresponder a alguna entidad.

Los esquemas lógicos locales obtenidos hasta este momento se integrarán en un solo esquema lógico global en la siguiente fase para modelar los datos de toda la empresa.

3.1.2.1.8. Mezclar los esquemas lógicos locales en un esquema lógico global

En este paso, se deben integrar todos los esquemas locales en un solo esquema global. En un sistema pequeño, con dos o tres vistas de usuario y unas pocas entidades y relaciones, es relativamente sencillo comparar los esquemas locales, mezclarlos y resolver cualquier tipo de diferencia que pueda existir. Pero en los sistemas grandes, se debe seguir un proceso más sistemático para llevar a cabo este paso con éxito:

1. Revisar los nombres de las entidades y sus claves primarias.
2. Revisar los nombres de las relaciones.
3. Mezclar las entidades de las vistas locales.
4. Incluir (sin mezclar) las entidades que pertenecen a una sola vista de usuario.

5. Mezclar las relaciones de las vistas locales.
6. Incluir (sin mezclar) las relaciones que pertenecen a una sola vista de usuario.
7. Comprobar que no se ha omitido ninguna entidad ni relación.
8. Comprobar las claves ajenas.
9. Comprobar las restricciones de integridad.
10. Dibujar el esquema lógico global.
11. Actualizar la documentación.

3.1.2.1.9. Validar el esquema lógico global

Este proceso de validación se realiza, de nuevo, mediante la normalización y mediante la prueba frente a las transacciones de los usuarios. Pero ahora sólo hay que normalizar las relaciones que hayan cambiado al mezclar los esquemas lógicos locales y sólo hay que probar las transacciones que requieran acceso a áreas que hayan sufrido algún cambio.

3.1.2.1.10. Estudiar el crecimiento futuro

En este paso, se trata de comprobar que el esquema obtenido puede acomodar los futuros cambios en los requisitos con un impacto mínimo. Si el esquema lógico se puede extender fácilmente, cualquiera de los cambios previstos se podrá incorporar al mismo con un efecto mínimo sobre los usuarios existentes.

3.1.2.1.11. Dibujar el diagrama entidad-relación final

Una vez validado el esquema lógico global, ya se puede dibujar el diagrama entidad-relación que representa el modelo de los datos de la empresa que son de interés. La documentación que describe este modelo (incluyendo el esquema relacional y el diccionario de datos) se debe actualizar y completar.

3.1.2.1.12. Revisar el esquema lógico global con los usuarios

Una vez más, se debe revisar con los usuarios el esquema global y la documentación obtenida para asegurarse de que son una fiel representación de la empresa.

3.1.3. DISEÑO FÍSICO DE BASES DE DATOS

En este capítulo se describe la metodología de diseño físico para bases de datos relacionales. En esta etapa, se parte del esquema lógico global obtenido durante el diseño lógico y se obtiene una descripción de la implementación de la base de datos en memoria secundaria. Esta descripción es completamente dependiente del SGBD específico que se vaya a utilizar. En este capítulo se dan una serie de directrices para escoger las estructuras de almacenamiento de las relaciones base, decidir cuándo crear índices y cuándo desnormalizar el esquema lógico e introducir redundancias

El diseño de una base de datos se descompone en tres etapas: diseño conceptual, lógico y físico. La etapa del diseño lógico es independiente de los detalles de implementación y dependiente del tipo de SGBD que se vaya a utilizar. La salida de esta etapa es el

esquema lógico global y la documentación que lo describe. Todo ello es la entrada para la etapa que viene a continuación, el diseño físico.

Mientras que en el diseño lógico se especifica qué se guarda, en el diseño físico se especifica cómo se guarda. Para ello, el diseñador debe conocer muy bien toda la funcionalidad del SGBD concreto que se vaya a utilizar y también el sistema informático sobre el que éste va a trabajar. El diseño físico no es una etapa aislada, ya que algunas decisiones que se tomen durante su desarrollo.

3.1.3.1 Metodología de diseño físico para bases de datos relacionales

El objetivo de esta etapa es producir una descripción de la implementación de la base de datos en memoria secundaria. Esta descripción incluye las estructuras de almacenamiento y los métodos de acceso que se utilizarán para conseguir un acceso eficiente a los datos.

El diseño físico se divide de cuatro fases, cada una de ellas compuesta por una serie de pasos:

- Traducir el esquema lógico global para el SGBD específico.
    1. Diseñar las relaciones base para el SGBD específico.
    2. Diseñar las reglas de negocio para el SGBD específico.
- Diseñar la representación física.
    1. Analizar las transacciones.
    2. Escoger las organizaciones de ficheros.
    3. Escoger los índices secundarios.

4. Considerar la introducción de redundancias controladas.
5. Estimar la necesidad de espacio en disco.

- Diseñar los mecanismos de seguridad.
  1. Diseñar las vistas de los usuarios.
  2. Diseñar las reglas de acceso.
- Monitorizar y afinar el sistema.

3.1.3.1.1 Traducir el esquema lógico global

La primera fase del diseño lógico consiste en traducir el esquema lógico global en un esquema que se pueda implementar en el SGBD escogido. Para ello, es necesario conocer toda la funcionalidad que éste ofrece. Por ejemplo, el diseñador deberá saber:

- Si el sistema soporta la definición de claves primarias, claves ajenas y claves alternativas.
- Si el sistema soporta la definición de datos requeridos (es decir, si se pueden definir atributos como no nulos).
- Si el sistema soporta la definición de dominios.
- Si el sistema soporta la definición de reglas de negocio.
- Cómo se crean las relaciones base.

1. Diseñar las relaciones base para el SGBD

Las relaciones base se definen mediante el lenguaje de definición de datos del SGBD. Para ello, se utiliza la información producida durante el diseño lógico: el esquema

lógico global y el diccionario de datos. El esquema lógico consta de un conjunto de relaciones y, para cada una de ellas, se tiene:

- El nombre de la relación.
- La lista de atributos entre paréntesis.
- La clave primaria y las claves ajenas, si las tiene.
- Las reglas de integridad de las claves ajenas.

En el diccionario de datos se describen los atributos y, para cada uno de ellos, se tiene:

- Su dominio: tipo de datos, longitud y restricciones de dominio.
- El valor por defecto, que es opcional.
- Si admite nulos.
- Si es derivado y, en caso de serlo, cómo se calcula su valor.

2. Diseñar las reglas de negocio para el SGBD

Las actualizaciones que se realizan sobre las relaciones de la base de datos deben observar ciertas restricciones que imponen las reglas de negocio de la empresa. Algunos SGBD proporcionan mecanismos que permiten definir estas restricciones y vigilan que no se violen.

Se puede definir una restricción en la sentencia CREATE TABLE de la relación

Otro modo de definir esta restricción es mediante un *disparador* (*trigger*):

Hay algunas restricciones que no las pueden manejar los SGBD, como por ejemplo ‘a las 20:30 del último día laborable de cada año archivar los inmuebles vendidos y

borrarlos'. Para estas restricciones habrá que escribir programas de aplicación específicos. Por otro lado, hay SGBD que no permiten la definición de restricciones, por lo que éstas deberán incluirse en los programas de aplicación.

Todas las restricciones que se definan deben estar documentadas. Si hay varias opciones posibles para implementarlas, hay que explicar porqué se ha escogido la opción implementada.

3.1.3.1.2. Diseñar la representación física

Uno de los objetivos principales del diseño físico es almacenar los datos de modo eficiente. Para medir la eficiencia hay varios factores que se deben tener en cuenta:

- *Productividad de transacciones.* Es el número de transacciones que se quiere procesar en un intervalo de tiempo.
- *Tiempo de respuesta.* Es el tiempo que tarda en ejecutarse una transacción. Desde el punto de vista del usuario, este tiempo debería ser el mínimo posible.
- *Espacio en disco.* Es la cantidad de espacio en disco que hace falta para los ficheros de la base de datos. Normalmente, el diseñador querrá minimizar este espacio.

Lo que suele suceder, es que todos estos factores no se pueden satisfacer a la vez. Por ejemplo, para conseguir un tiempo de respuesta mínimo, puede ser necesario aumentar la cantidad de datos almacenados, ocupando más espacio en disco. Por lo tanto, el

diseñador deberá ir ajustando estos factores para conseguir un equilibrio razonable. El diseño físico inicial no será el definitivo, sino que habrá que ir monitorizándolo para observar sus prestaciones e ir ajustándolo como sea oportuno. Muchos SGBD proporcionan herramientas para monitorizar y afinar el sistema.

Hay algunas estructuras de almacenamiento que son muy eficientes para cargar grandes cantidades de datos en la base de datos, pero no son eficientes para el resto de operaciones, por lo que se puede escoger dicha estructura de almacenamiento para inicializar la base de datos y cambiarla, a continuación, para su posterior operación. Los tipos de organizaciones de ficheros disponibles varían en cada SGBD. Algunos sistemas proporcionan más estructuras de almacenamiento que otros. Es muy importante que el diseñador del esquema físico sepa qué estructuras de almacenamiento le proporciona el SGBD y cómo las utiliza.

Para mejorar las prestaciones, el diseñador del esquema físico debe saber cómo interactúan los dispositivos involucrados y cómo esto afecta a las prestaciones:

- *Memoria principal.* Los accesos a memoria principal son mucho más rápidos que los accesos a memoria secundaria (decenas o centenas de miles de veces más rápidos). Generalmente, cuanta más memoria principal se tenga, más rápidas serán las aplicaciones. Sin embargo, es aconsejable tener al menos un 5% de la memoria disponible, pero no más de un 10%. Si no hay bastante memoria disponible para todos los procesos, el sistema operativo debe transferir páginas a disco para liberar memoria. Cuando estas páginas se vuelven a

necesitar, hay que volver a traerlas desde el disco. A veces, es necesario llevar procesos enteros a disco (*swapping*) para liberar memoria. El hacer estas transferencias con demasiada frecuencia empeora las prestaciones.

- *CPU.* La CPU controla los recursos del sistema y ejecuta los procesos de usuario. El principal objetivo con este dispositivo es lograr que no haya bloqueos de procesos para conseguirla. Si el sistema operativo, o los procesos de los usuarios, hacen muchas demandas de CPU, ésta se convierte en un cuello de botella. Esto suele ocurrir cuando hay muchas faltas de página o se realiza mucho swapping.
- *Entrada/salida a disco.* Los discos tienen una velocidad de entrada/salida. Cuando se requieren datos a una velocidad mayor que ésta, el disco se convierte en un cuello de botella. Dependiendo de cómo se organicen los datos en el disco, se conseguirá reducir la probabilidad de empeorar las prestaciones. Los principios básicos que se deberían seguir para repartir los datos en los discos son los siguientes:
    - Los ficheros del sistema operativo deben estar separados de los ficheros de la base de datos.
    - Los ficheros de datos deben estar separados de los ficheros de índices
    - Los ficheros con los diarios de operaciones deben estar separados del resto de los ficheros de la base de datos.
- *Red.* La red se convierte en un cuello de botella cuando tiene mucho tráfico y cuando hay muchas colisiones.

Cada uno de estos recursos afecta a los demás, de modo que una mejora en alguno de ellos puede provocar mejoras en otros.

1. Analizar las transacciones

Para realizar un buen diseño físico es necesario conocer las consultas y las transacciones que se van a ejecutar sobre la base de datos. Esto incluye tanto información cualitativa, como cuantitativa. Para cada transacción, hay que especificar:

- La frecuencia con que se va a ejecutar.
- Las relaciones y los atributos a los que accede la transacción, y el tipo de acceso: consulta, inserción, modificación o eliminación. Los atributos que se modifican no son buenos candidatos para construir estructuras de acceso.
- Los atributos que se utilizan en los predicados del `WHERE` de las sentencias SQL. Estos atributos pueden ser candidatos para construir estructuras de acceso dependiendo del tipo de predicado que se utilice.
- Si es una consulta, los atributos involucrados en el join de dos o más relaciones. Estos atributos pueden ser candidatos para construir estructuras de acceso.
- Las restricciones temporales impuestas sobre la transacción. Los atributos utilizados en los predicados de la transacción pueden ser candidatos para construir estructuras de acceso.

2. Escoger las organizaciones de ficheros

El objetivo de este paso es escoger la organización de ficheros óptima para cada relación. Por ejemplo, un fichero desordenado es una buena estructura cuando se va a cargar gran cantidad de datos en una relación al inicializarla, cuando la relación tiene pocas tuplas, también cuando en cada acceso se deben obtener todas las tuplas de la relación, o cuando la relación tiene una estructura de acceso adicional, como puede ser un índice. Por otra parte, los ficheros dispersos (hashing) son apropiados cuando se accede a las tuplas a través de los valores exactos de alguno de sus campos (condición de igualdad en el WHERE). Si la condición de búsqueda es distinta de la igualdad (búsqueda por rango, por patrón, etc.), la dispersión no es una buena opción. Hay otras organizaciones, como la ISAM o los árboles B+.

Las organizaciones de ficheros elegidas deben documentarse, justificando en cada caso la opción escogida.

3. Escoger los índices secundarios

Los índices secundarios permiten especificar caminos de acceso adicionales para las relaciones base. Si se accede a menudo a esta relación a través del atributo alquiler, se puede plantear la creación de un índice sobre dicho atributo para favorecer estos accesos. Pero hay que tener en cuenta que estos índices conllevan un coste de mantenimiento que hay que sopesar frente a la ganancia en prestaciones. A la hora de seleccionar los índices, se pueden seguir las siguientes indicaciones:

- Construir un índice sobre la clave primaria de cada relación base.
- No crear índices sobre relaciones pequeñas.
- Añadir un índice sobre los atributos que se utilizan para acceder con mucha frecuencia.
- Añadir un índice sobre las claves ajenas que se utilicen con frecuencia para hacer joins.
- Evitar los índices sobre atributos que se modifican a menudo.
- Evitar los índices sobre atributos poco selectivos (aquellos en los que la consulta selecciona una porción significativa de la relación).
- Evitar los índices sobre atributos formados por tiras de caracteres largas.

Los índices creados se deben documentar, explicando las razones de su elección.

4. Considerar la introducción de redundancias controladas

En ocasiones puede ser conveniente relajar las reglas de normalización introduciendo redundancias de forma controlada, con objeto de mejorar las prestaciones del sistema. En la etapa del diseño lógico se recomienda llegar, al menos, hasta la tercera forma normal para obtener un esquema con una estructura consistente y sin redundancias. Pero, a menudo, sucede que las bases de datos así normalizadas no proporcionan la máxima eficiencia, con lo que es necesario volver atrás y desnormalizar algunas relaciones, sacrificando los beneficios de la normalización para mejorar las prestaciones. Es importante hacer notar que la desnormalización sólo debe realizarse cuando se estime que el sistema no puede alcanzar las prestaciones deseadas. Y, desde

luego, la necesidad de desnormalizar en ocasiones no implica eliminar la normalización del diseño lógico: la normalización obliga al diseñador a entender completamente cada uno de los atributos que se han de representar en la base de datos. Por lo tanto, hay que tener en cuenta los siguientes factores:

- La desnormalización hace que la implementación sea más compleja.
- La desnormalización hace que se sacrifique la flexibilidad.
- La desnormalización puede hacer que los accesos a datos sean más rápidos, pero ralentiza las actualizaciones.

Por regla general, la desnormalización de una relación puede ser una opción viable cuando las prestaciones que se obtienen no son las deseadas y la relación se actualiza con poca frecuencia, pero se consulta muy a menudo. Las redundancias que se pueden incluir al desnormalizar son de varios tipos: se pueden introducir datos derivados (calculados a partir de otros datos), se pueden duplicar atributos o se pueden hacer joins de relaciones.

El incluir un atributo derivado dependerá del coste adicional de almacenarlo y mantenerlo consistente con los datos de los que se deriva, frente al coste de calcularlo cada vez que se necesita.

No se pueden establecer una serie de reglas que determinen cuándo desnormalizar relaciones, pero hay algunas situaciones muy comunes en donde puede considerarse esta posibilidad:

- *Combinar relaciones de uno a uno.* Cuando hay relaciones (tablas) involucradas en relaciones de uno a uno, se accede a ellas de manera conjunta con frecuencia y casi no se les accede separadamente, se pueden combinar en una sola relación (tabla).
- *Duplicar atributos no clave en relaciones de uno a muchos para reducir los joins.* Para evitar operaciones de join, se pueden incluir atributos de la relación (tabla) padre en la relación (tabla) hijo de las relaciones de uno a muchos.
- *Tablas de referencia.* Las tablas de referencia (*lookup*) son listas de valores, cada uno de los cuales tiene un código. Este tipo de tablas son un caso de relación de uno a muchos. De este modo, es muy fácil validar los datos, además de que se ahorra espacio escribiendo sólo el código y no la descripción para cada inmueble, además de ahorrar tiempo cuando se actualizan las descripciones. Si las tablas de referencia se utilizan a menudo en consultas críticas, se puede considerar la introducción de la descripción junto con el código en la relación (tabla) hijo, manteniendo la tabla de referencia para validación de datos.
- *Duplicar claves ajenas en relaciones de uno a muchos para reducir los joins.* Para evitar operaciones de join, se pueden incluir claves ajenas de una relación (tabla) en otra relación (tabla) con la que se relaciona (habrá que tener en cuenta ciertas restricciones).
- *Duplicar atributos en relaciones de muchos a muchos para reducir los joins.* Durante el diseño lógico se eliminan las relaciones de muchos a muchos introduciendo dos relaciones de uno a muchos. Esto hace que aparezca una

nueva relación (tabla) intermedia, de modo que si se quiere obtener la información de la relación de muchos a muchos, se tiene que realizar el join de tres relaciones (tablas). Para evitar algunos de estos joins se pueden incluir algunos de los atributos de las relaciones (tablas) originales en la relación (tabla) intermedia.

- *Introducir grupos repetitivos.* Los grupos repetitivos se eliminan en el primer paso de la normalización para conseguir la primera forma normal. Estos grupos se eliminan introduciendo una nueva relación (tabla), generando una relación de uno a muchos. A veces, puede ser conveniente reintroducir los grupos repetitivos para mejorar las prestaciones.

Todas las redundancias que se introduzcan en este paso se deben documentar y razonar. El esquema lógico se debe actualizar para reflejar los cambios introducidos.

5. Estimar la necesidad de espacio en disco

En caso de que se tenga que adquirir nuevo equipamiento informático, el diseñador debe estimar el espacio necesario en disco para la base de datos. Esta estimación depende del SGBD que se vaya a utilizar y del hardware. En general, se debe estimar el número de tuplas de cada relación y su tamaño. También se debe estimar el factor de crecimiento de cada relación.

3.1.3.1.3. Diseñar los mecanismos de seguridad

Los datos constituyen un recurso esencial para la empresa, por lo tanto su seguridad es de vital importancia. Durante el diseño lógico se habrán especificado los requerimientos en cuanto a seguridad que en esta fase se deben implementar. Para llevar a cabo esta implementación, el diseñador debe conocer las posibilidades que ofrece el SGBD que se vaya a utilizar.

1. Diseñar las vistas de los usuarios

El objetivo de este paso es diseñar las vistas de los usuarios correspondientes a los esquemas lógicos locales. Las vistas, además de preservar la seguridad, mejoran la independencia de datos, reducen la complejidad y permiten que los usuarios vean los datos en el formato deseado.

2. Diseñar las reglas de acceso

El administrador de la base de datos asigna a cada usuario un identificador que tendrá una palabra secreta asociada por motivos de seguridad. Para cada usuario o grupo de usuarios se otorgarán permisos para realizar determinadas acciones sobre determinados objetos de la base de datos. Por ejemplo, los usuarios de un determinado grupo pueden tener permiso para consultar los datos de una relación base concreta y no tener permiso para actualizarlos.

3.1.3.1.4. Monitorizar y afinar el sistema

Una vez implementado el esquema físico de la base de datos, se debe poner en marcha para observar sus prestaciones. Si éstas no son las deseadas, el esquema deberá cambiar para intentar satisfacerlas. Una vez afinado el esquema, no permanecerá estático, ya que tendrá que ir cambiando conforme lo requieran los nuevos requisitos de los usuarios. Los SGBD proporcionan herramientas para monitorizar el sistema mientras está en funcionamiento.

RESUMEN

El diseño de bases de datos se descompone en tres etapas: diseño conceptual, diseño lógico y diseño físico. El diseño conceptual es el proceso por el cual se construye un modelo de la información que se utiliza en una empresa u organización, independientemente del SGBD que se vaya a utilizar para implementar el sistema y de los equipos informáticos o cualquier otra consideración física.

Un modelo conceptual es un conjunto de conceptos que permiten describir la realidad mediante representaciones lingüísticas y gráficas. Los modelos conceptuales deben poseer una serie de propiedades: expresividad, simplicidad, minimalidad y formalidad.

El modelo conceptual más utilizado es el modelo entidad-relación, que posee los siguientes conceptos: entidades, relaciones, atributos, dominios de atributos, identificadores y jerarquías de generalización.

En la metodología del diseño conceptual se construye un esquema conceptual local para cada vista de cada usuario o grupo de usuarios. En el diseño lógico se obtiene un esquema lógico local para cada esquema conceptual local. Estos esquemas lógicos se

integran después para formar un esquema lógico global que represente todas las vistas de los distintos usuarios de la empresa. Por último, en el diseño físico, se construye la implementación de la base de datos sobre un SGBD determinado. Ya que este diseño debe adaptarse al SGBD, es posible que haya que introducir cambios en el esquema lógico para mejorar las prestaciones a nivel físico.

Cada vista de usuario comprende los datos que un usuario maneja para llevar a cabo una determinada tarea. Normalmente, estas vistas corresponden a las distintas áreas funcionales de la empresa, y se pueden identificar examinando los diagramas de flujo de datos o entrevistando a los usuarios, examinando los procedimientos, informes y formularios, y observando el funcionamiento de la empresa.

Cada esquema conceptual local está formado por entidades, relaciones, atributos, dominios de atributos, identificadores y puede haber también jerarquías de generalización. Además, estos esquemas se completan documentándolos en el diccionario de datos.

El diseño físico es el proceso de producir una descripción de la implementación de la base de datos en memoria secundaria. Describe las relaciones base y las estructuras de almacenamiento y métodos de acceso que se utilizarán para acceder a los datos de modo eficiente. El diseño de las relaciones base sólo se puede realizar cuando el diseñador conoce perfectamente toda la funcionalidad que presenta el SGBD que se vaya a utilizar.

El primer paso consiste en traducir el esquema lógico global de modo que pueda ser fácilmente implementado por el SGBD específico. A continuación, se escogen las

organizaciones de ficheros más apropiadas para almacenar las relaciones base, y los métodos de acceso, basándose en el análisis de las transacciones que se van a ejecutar sobre la base de datos. Se puede considerar la introducción de redundancias controladas para mejorar las prestaciones. Otra tarea a realizar en este paso es estimar el espacio en disco.

La seguridad de la base de datos es fundamental, por lo que el siguiente paso consiste en diseñar las medidas de seguridad necesarias mediante la creación de vistas y el establecimiento de permisos para los usuarios.

El último paso del diseño físico consiste en monitorizar y afinar el sistema para obtener las mejores prestaciones y satisfacer los cambios que se puedan producir en los requisitos.

CAPÍTULO IV

ESTUDIO DE BASES DE DATOS ORIENTADO A OBJETOS

4.1 INTRODUCCIÓN A LAS BDOO

Las bases de datos orientadas a objetos están diseñadas para simplificar la programación orientada a objetos. Almacenan los objetos directamente en la base de datos, y emplean las mismas estructuras y relaciones que los lenguajes de programación orientados a objetos.

Las bases de datos tradicionales almacenan sólo datos, mientras que las bases de datos orientadas a objetos almacenan objetos, con una estructura arbitraria y un comportamiento definido por el programador. Una simple gráfico nos ayuda a ilustrar la diferencia entre ambos modelos Fig 7. Consideremos el problema de aparcar (almacenar) un coche en un garaje al final del día. En un sistema de objetos el coche es un objeto, el garaje es un objeto, y hay una operación simple que es almacenar_coche_en_garaje. En un sistema relacional, todos los datos deben ser traducidos a tablas; de esta forma, el coche debe ser desarmado, y todos los pistones

almacenados en una tabla, todas las ruedas en otra, etc. Por la mañana, antes de irse a trabajar, hay que componer de nuevo el coche para poder conducir (el problema es que al componer piezas puede salir una moto en vez de un coche).

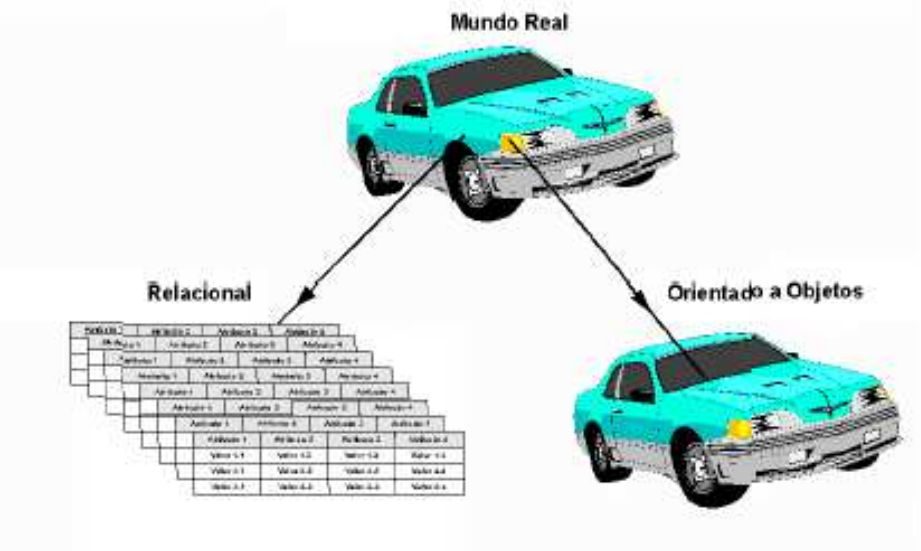


Fig 7 Diferencias conceptuales entre una BD relacional y una BDOO

4.1.1. ¿Qué es una BDOO?

A finales de los 80 aparecieron las primeras bases de datos orientadas a objetos. Se considera una BDOO cuando soporta el paradigma orientado a objetos almacenando datos y métodos (y no sólo datos, como las bases de datos anteriores); está diseñada para ser eficaz a la hora de almacenar objetos completos desde el punto de vista físico; y es más segura ya que no permite tener acceso directo a los datos u objetos, dado que para poder acceder a los mismos se tiene que hacer uso de los métodos proporcionados por el programador.

Las bases de datos orientadas a objetos almacenan y manipulan información digitalizada

(representada) en objetos, y proporcionan una estructura flexible con acceso ágil, rápido y con una gran capacidad de modificación.

Además combina las mejores cualidades de los archivos planos, las bases de datos jerárquicas y las relacionales: las BDOO representan el siguiente paso en la evolución de las bases de datos para poder soportar el análisis, el diseño y la implementación orientados a objetos.

Éstas permiten el desarrollo y el mantenimiento de aplicaciones complejas, ya que se puede utilizar el mismo modelo conceptual durante las fases de análisis, diseño y programación. Esto reduce el problema de realizar una traducción entre los diferentes modelos a través de todo el ciclo de vida, con un coste significativamente menor. El modelo conceptual debe ser la base de las herramientas CASE OO totalmente integradas que ayuden a generar la estructura de los datos y los métodos.

Como cualquier base de datos programable, una base de datos orientada a objetos ofrece un ambiente idóneo para el desarrollo de aplicaciones con un repositorio persistente listo para su explotación.

Además, las BDOO ofrecen un mayor rendimiento que las bases de datos relacionales para aplicaciones o clases con estructuras complejas de datos.

4.1.2. Un Modelo Conceptual Unificado

Las técnicas de orientación a objetos utilizan los mismos modelos conceptuales durante el análisis, el diseño y la implementación. La tecnología de las BDOO da un paso más hacia la unificación: el modelo conceptual de la base de datos orientada a objetos es

igual al del resto del mundo orientado a objetos, en lugar de utilizar tablas independientes (como SQL).

El uso del mismo modelo conceptual para todos los aspectos del desarrollo simplifica éste, particularmente con las herramientas CASE OO, mejora la comunicación entre usuarios, analistas y programadores, y además reduce las posibilidades de error.

4.1.3. Arquitectura de una BDOO

Los primeros productos relacionados con las BDOO se diseñaron como una extensión de los lenguajes de programación como Smalltalk o C++. El LMD (lenguaje para la manipulación de datos; también conocido como DML) y el LDD (lenguaje para la definición de los datos; también conocido como DDL) construían un lenguaje OO común.

El diseño de las BDOO actuales debe aprovechar al máximo las herramientas CASE e incorporar métodos creados con cualquier técnica poderosa, incluyendo enunciados declarativos, generadores de códigos e inferencias con base en reglas.

Algunas características son independientes de la arquitectura fundamental de una BDOO pero son comunes a la mayoría de ellas:

• Versiones: La mayoría de los sistemas de bases de datos relacionales sólo permiten que exista una representación de un ente de la base de datos dentro de esta. Las versiones permiten que las representaciones alternas existan en forma simultánea.

• Transacciones compartidas: Las transacciones compartidas soportan grupos de usuarios en estaciones de trabajo, los cuales desean coordinar sus esfuerzos en tiempo

real. Los usuarios pueden compartir los resultados intermedios de una base de datos. La transacción compartida permite que varias personas intervengan en una sola transacción

4.1.4. Desarrollo con Bases de Datos OO

Las BDOO se desarrollan al describir en primer lugar los tipos de objetos importantes del dominio de la base de datos. Estos tipos de objetos determinan las clases que conformarán la definición de la BDOO.

Por ejemplo: Una base de datos diseñada para almacenar la geometría de ciertas partes mecánicas incluiría clases como CILINDRO, ESFERA Y CUBO. El comportamiento de CILINDRO podría incluir información relativa a sus dimensiones, volumen y área de superficie:

```
Clase de CILINDRO{
        ALTURA                  REAL ();
        RADIO                   REAL ();
        VOLUMEN                 REAL ();
        AREA_DE_SUPERFICIE      REAL ();
        };
```

Se puede llegar a definiciones similares para el cubo y la esfera. En la definición anterior ALTURA, RADIO y ÁREA representan los mensajes que se pueden enviar a un objeto CILINDRO .

La Implantación se lleva a cabo en el mismo lenguaje, escribiendo funciones correspondientes a las solicitudes OO:

```
CILINDRO::ALTURA () { RETORNA altura; }
CILINDRO::VOLUMEN () { RETORNA PI*RADIO ()*ALTURA (); }
```

En este caso, la Altura se almacena como un elemento de los datos, mientras que volumen se calcula mediante la fórmula apropiada. Hay que observar que la implantación interna de volumen utiliza solicitudes para obtener altura y radio. Sin embargo, el aspecto más importante es la sencillez y uniformidad que experimentan los usuarios de CILINDRO. Sólo necesitan conocer la forma de enviar una solicitud y las solicitudes disponibles.

4.1.5. Tres Enfoques de Construcción de Bases de Datos OO

Las BDOO se pueden construir mediante alguno de los tres enfoques siguientes:

• El Primero: se puede utilizar el código actual, altamente complejo, de los sistemas de administración de las bases de datos, de modo que una BDOO se implante más rápido sin tener que partir de cero. Las técnicas orientadas a objetos se pueden utilizar como medios para el diseño sencillo de sistemas complejos. Los sistemas se construyen a partir de componentes ya probados con un formato definido para las solicitudes de las operaciones del componente.

• El Segundo: considerar a la BDOO como una extensión de la tecnología de las bases de datos relacionales. De este modo, las herramientas, técnicas, y vasta experiencia de la tecnología por relación se utilizan para construir un nuevo SGBD.

Se pueden añadir punteros a las tablas de relación para ligarlas con objetos binarios de gran tamaño (BLOB). La base de datos también debe proporcionar a las aplicaciones clientes un acceso aleatorio y por partes a grandes objetos, con el fin de que sólo sea necesario recuperar a través de la red la parte solicitada de los datos.

• El Tercero: reflexiona sobre la arquitectura de los sistemas de bases de datos y produce una nueva arquitectura optimizada, que cumple las necesidades de la tecnología OO. Las compañías como Versant, Objectivity, Itasca, etc. utilizan este enfoque y afirman que la tecnología de relación es un subconjunto de una capacidad más general. Además, las BDOO no relacionales son aproximadamente dos veces más rápidas que las bases de datos por relación para almacenar y recuperar la información compleja. Por lo tanto, son esenciales en aplicaciones como CAD (Diseño Asistido por Computadora) y permitirían que un depósito CASE fuera una facilidad de tiempo real en vez de una facilidad por lotes.

Por ejemplo, la Arquitectura de Versant está orientada al soporte Cliente/Servidor con acercamiento a la computación distribuida. Cualquier aplicación de Cliente es procesada por el servidor, que usa las EDT y las máquinas servidoras que pueden cooperar en una BD distribuida de Versant. Las BD pueden estar diseñadas como un sistema m-Cliente/n-Servidor.

Un servidor en el medioambiente de Versant es una máquina que está corriendo los procesos del servidor. Ésta soporta accesos concurrentes por usuarios múltiples de una o más BD. Un cliente es un proceso de aplicación que tiene acceso a espacios de trabajo de BD persistentes privadas y en adición, puede acceder a diversas BD sobre servidores concurrentes con otras aplicaciones de cliente.

4.1.6. Impacto de la Orientación a Objetos en la Ingeniería del Software.

En las BDOO, la organización ODMG (Grupo Manejador de Datos Objeto) representa el 100% de las BDOO industriales y ha establecido un estándar de definición (ODL, Lenguaje de Definición de datos) y manipulación (OQL, Lenguaje de consulta) de bases de datos equivalente a SQL.

Respecto a las relacionales, todas (Oracle, Informix, etc.) están añadiendo en mayor o menor grado algunos aspectos de la orientación a objetos. ANSI (Instituto Nacional Estadounidense de Estándar), por su parte, está definiendo un SQL-3 que incorpora muchos aspectos de la orientación a objetos. El futuro del SQL-3 es sin embargo incierto, ya que ODMG ha ofrecido a ANSI su estándar para que sirva de base para un nuevo SQL, con lo que solo habría un único estándar de base de datos.

El grupo ODMG nació de un grupo más grande, llamado OMG (Grupo Manejador de Objetos), donde están representadas todas las cosas con alguna influencia en el sector. Este grupo está definiendo un estándar universal por objetos. Este estándar permitirá que un objeto sea programado en cualquier lenguaje y sistema operativo. Esto facilitará enormemente el desarrollo de sistemas abiertos cliente-servidor.

4.1.7. Ventajas en BDOO

Está su flexibilidad, y soporte para el manejo de tipos de datos complejos. Por ejemplo, en una base de datos convencional, si una empresa adquiere varios clientes por referencia de clientes servicio, pero la base de datos existente, que mantiene la información de clientes y sus compras, no tiene un campo para registrar quién proporcionó la referencia, de qué manera fue dicho contacto, o si debe compensarse con una comisión, sería necesario reestructurar la base de datos para añadir este tipo de modificaciones. Por el contrario, en una BDOO, el usuario puede añadir una "subclase" de la clase de clientes para manejar las modificaciones que representan los clientes por referencia.

La subclase heredará todos los atributos, características de la definición original, además se especializará en especificar los nuevos campos que se requieren así como los métodos para manipular solamente estos campos. Naturalmente se generan los espacios para almacenar la información adicional de los nuevos campos. Esto presenta la ventaja adicional que una BDOO puede ajustarse a usar siempre el espacio de los campos que son necesarios, eliminando espacio desperdiciado en registros con campos que nunca usan.

La segunda ventaja de una BDOO es que manipula datos complejos de forma rápida y ágil.

La estructura de la base de datos está dada por referencias (o punteros lógicos) entre objetos

4.1.8. Posibles desventajas

Al considerar la adopción de la tecnología orientada a objetos, la inmadurez del mercado de BDOO constituye una posible fuente de problemas, por lo que debe analizarse con detalle la presencia en el mercado del proveedor para adoptar su producto en una línea de producción sustantiva.

Un aspecto a considerar en los SGBD es que muchos de los sistemas comerciales presentan de orientación a objetos únicamente un envoltorio exterior, pero interiormente se siguen apoyando en la representación tabular del modelo relacional. Estos sistemas de gestión de bases de datos son etiquetados como *objeto-relacionales* y tienen una presencia considerable en el mercado. Tal es el caso por ejemplo de Persistence, que es un producto que proporciona un entorno de orientación a objetos (mediante el lenguaje de programación C++) a bases de datos relacionales como Oracle, Sybase, Informix, etc. Otros productos son Illustra, UniSQL, Omniscience, etc. El problema de estos sistemas es que algunos de ellos soportan sólo algunas características del modelo de objetos y que además se necesita una gran cantidad de trabajo (aunque en este caso no sea realizado por el usuario sino por el propio sistema) y tiempo para el ensamblaje y desensamblaje de objetos en tablas. De lo anterior se deduce que los SGBDOO verdaderos son aquellos en los que se trabaja con objetos a todos los niveles, o lo que es lo mismo, los objetos especificados en las fases de análisis y diseño, son manipulados mediante un lenguaje de programación (C++, Java, Smalltalk, etc.) que es ampliado con

las capacidades de bases de datos y que permite directamente su almacenamiento, recuperación, modificación y consulta.

El tercer problema es la falta de estándares en la industria orientada a objetos. Sin embargo, el OMG (Grupo Manejador de Objetos), es una organización internacional de proveedores de sistemas de información y usuarios dedicada a promover estándares para el desarrollo de aplicaciones y sistemas orientados a objetos en ambientes distribuidos en red. La implantación de una nueva tecnología requiere que los usuarios iniciales acepten cierto riesgo; aquellos que esperen resultados a corto plazo y con un bajo coste se quedarán desilusionados. Sin embargo, para aquellos usuarios que se plantean en un futuro a medio plazo el uso de tecnología avanzada, el uso de tecnología orientada a objetos, verán compensados paulatinamente todos los riesgos iniciales.

4.1.8.1. Desadaptación de impedancias e Interoperabilidad

El problema que se plantea es que el paradigma de orientación a objetos no es adoptado de una forma integral por todos los componentes del sistema lo que provoca:

- Desadaptación de impedancias, o salto semántico, ocasionado cada vez que un elemento del sistema (por ejemplo, el sistema operativo) debe interactuar con otro (una aplicación orientada a objetos). Una posible solución a este problema sería trasladar la orientación a objetos al sistema operativo (como en el proyecto de investigación Oviedo 3, por ejemplo. Aquí, todos los componentes: interfaces de usuario, aplicaciones, lenguajes, compiladores, bases de datos y hasta el propio sistema operativo comparten el mismo paradigma de orientación a objetos).

- Problema de interoperabilidad entre modelos de objetos: es decir, una aplicación desarrollada usando el lenguaje C++, con el modelo de objetos de C++, no tiene ningún problema de comunicación con sus propios objetos, pero cuando se desean usar objetos de otro lenguaje de programación o de una base de datos orientada a objetos aparece un problema de interoperabilidad. Esto es debido a que el modelo de objetos C++ no tiene por qué ser totalmente compatible con el de los otros elementos.

Para solucionar estos problemas los sistemas convencionales recurren a la introducción de capas de software de adaptación adicionales, lo que deteriora el rendimiento global del sistema, provoca una pérdida de portabilidad y de flexibilidad y hace que se aumente la complejidad.

4.1.9. Primer intento de Estandarización: ODMG-93

La mayor limitación de las bases de datos orientadas a objetos es la carencia de un estándar.

ODMG-93 *(Object-Oriented Database Management Grou*p) es un punto de partida muy importante para ello. Adopta una arquitectura que consta de un sistema de gestión que soporta un lenguaje de bases de datos orientado a objetos, con una sintaxis similar a un lenguaje de programación también orientado a objetos como puede ser C++ o Smalltalk. El lenguaje de bases de datos es especificado mediante un lenguaje de definición de datos (ODL), un lenguaje de manipulación de datos (OML), y un lenguaje de consulta (OQL), siendo todos ellos portables a otros sistemas con el fin de conseguir la portabilidad de la aplicación completa.

En definitiva, ODMG-93 intenta definir un SGBDOO que integre las capacidades de las bases de datos con las capacidades de los lenguajes de programación, de forma que los objetos de la base de datos aparezcan como objetos del lenguaje de programación, intentando de esta manera eliminar la falta de correspondencia existente entre los sistemas de tipos de ambos lenguajes. El SGBDOO extiende el lenguaje con persistencia, concurrencia, recuperación de datos, consultas asociativas, etc.

4.1.10. Lenguaje ODL

El lenguaje de definición de datos (ODL) en un SGBDOO es empleado para posibilitar de forma fácil la portabilidad de los esquemas de las bases de datos. Este ODL no es un lenguaje de programación completo: define las propiedades y los prototipos de las operaciones de los tipos, pero no los métodos que implementan dichas operaciones.

El ODL intenta definir los tipos que puedan implementarse en los diversos lenguajes de programación. No está por tanto ligado a la sintaxis concreta de un lenguaje de programación en particular; de esta forma, un esquema especificado en ODL puede ser soportado por cualquier SGBDOO que sea compatible con ODMG-93.

La sintaxis de ODL es una extensión de la del IDL *(Interface Definition Languag*e), desarrollado por OMG como parte de CORBA *(Common Object Request Broker Architectur*e).

La traducción ODL-C++, por ejemplo, se expresará como una librería de clases y una extensión a la gramática estándar de C++. La librería de clases proporcionará las clases y las funciones necesarias para implementar los conceptos definidos en el modelo de

objetos, y la extensión consistirá en un conjunto de palabras reservadas, y su sintaxis asociada, que se añadirán a la declaración de clases de C++ para proporcionar un soporte declarativo para las interrelaciones.

4.1.11. Lenguaje OML

El lenguaje de manipulación (OML) es el que se emplea para la elaboración de programas que permitan crear, modificar y borrar loas datos constituyentes de la BD.

ODMG-93 sugiere que este lenguaje sea la extensión de un lenguaje de programación, de forma que se pueden realizar (entre otras) las siguientes operaciones sobre la base de datos: Creación, Borrado, Modificación e Identificación de un objeto.

4.1.12. Lenguaje OQL

El lenguaje de consulta propuesto por ODMG-93, presenta las siguientes características:

• No es computacionalmente completo. Sin embargo, las consultas pueden invocar métodos, e inversamente los métodos escritos en cualquier lenguaje de programación pueden incluir consultas.

• Tiene una sintaxis abstracta.

• Su semántica formal puede definirse fácilmente.

• Proporciona un acceso declarativo a los objetos.

• Se basa en el modelo de objetos de ODMG-93.

• Tiene una sintaxis concreta al estilo SQL, pero puede cambiarse con facilidad.

• Puede optimizarse fácilmente.

• No proporciona operadores explícitos para la modificación, se basa en las operaciones definidas sobre los objetos para ese fin.

• Proporciona primitivas de alto nivel para tratar con conjuntos de objetos, pero no restringe su utilización con otros constructores de colecciones.

Existen dos posibilidades para asociar un sublenguaje de consulta a un lenguaje de programación: fuerte y débilmente.

• El primer caso consiste en una extensión de la gramática del lenguaje asociado.

• En el segundo caso, las funciones *query* tienen unos argumentos (del tipo *string* o cadena de caracteres) que contienen las preguntas.

4.1.13. Rendimiento

• Las BDOO permiten que un objeto hagan referencia de forma directa a otro mediante punteros. Esto hace que las BDOO pasen más rápido del objeto A al objeto B que las bases de datos relacionales, en las cuales se deben utilizar comandos JOIN para lograr el mismo resultado. Incluso un JOIN optimizado es más lento que un recorrido de los punteros.

• Las BDOO hacen que el agrupamiento sea más eficiente. La mayoría de los sistemas de bases de datos permiten que el operador coloque cerca las estructuras relacionadas entre sí, en el espacio de almacenamiento en disco. Esto reduce de forma radical el tiempo de recuperación de los datos relacionados, puesto que todos los datos se leen con

una única lectura de disco en vez de con varias. Sin embargo, en una base de datos relacional, los objetos de la implantación se traducen en representaciones tabulares que generalmente se dispersan en varias tablas. Así, en una BDR, estos renglones relacionados deben quedar agrupados, de modo que todo el objeto se pueda recuperar mediante una única lectura del disco. Esto es automático en una BDOO. Además, el agrupamiento de los datos relacionados, como todas las subpartes de un ensamble, puede afectar radicalmente el rendimiento general de una aplicación. Esto es relativamente directo en una BDOO, puesto que representa el primer nivel de agrupamiento. Por el contrario, el agrupamiento físico es imposible en una BDR, puesto que esto requiere un segundo nivel de agrupamiento: un nivel para agrupar las hileras que representan a los objetos individuales y un segundo para los grupos de hileras que representan a los objetos relacionados.

4.1.14. El SGBDOO

Un Sistema de Gestión de Bases de Datos Orientadas a Objetos (SGBDOO) se puede decir que es un SGBD que almacena objetos, permitiendo concurrencia, recuperación. Para los usuarios tradicionales de bases de datos, esto quiere decir que pueden tratar directamente con objetos, no teniendo que hacer la traducción a registros o tablas.

Un sistema de BDOO debe satisfacer dos criterios:

- Debe ser un SGBD: un Sistema de Gestión de Bases de Datos es un conjunto de datos relacionados entre sí y un grupo de programas para tener acceso a esos datos.
- Debe ser un sistema OO.

El primer criterio se traduce en cinco características, como son: persistencia, concurrencia, recuperación ante fallos del sistema, gestión del almacenamiento secundario y facilidad de consultas.

El segundo se traduce en ocho características: abstracción, encapsulación, modularidad, jerarquía, control de tipos, concurrencia, persistencia y genericidad como en la Fig 8.

Fig 8 Características de los SGBD y los SGBDOO

Como se puede apreciar en el esquema la persistencia, al igual que la concurrencia son características del SGBDOO heredadas tanto del SGBD como del modelo de objetos. La persistencia en el caso del SGBD hace referencia a la conservación de los datos después de la finalización del proceso que los creó. En el caso del modelo de objetos, se refiere no sólo a la conservación del estado de un objeto, si no también a la

conservación de la clase, que debe trascender a cualquier programa individual, de forma que todos los programas interpreten de la misma manera el estado almacenado. La concurrencia heredada del SGBD se refiere a la capacidad del sistema para gestionar a múltiples usuarios interactuando concurrentemente sobre el mismo, mientras que la concurrencia heredada del modelo de objetos hace referencia a la capacidad de distinguir a un objeto activo de otro que no lo está.

• Persistencia: Es la capacidad que tiene el programador para que sus datos se conserven al finalizar la ejecución de un proceso, de forma que se puedan reutilizar en otros procesos.

• Concurrencia: Se relaciona con la existencia de muchos usuarios interactuando concurrentemente en el sistema. Este debe controlar la interacción entre las transacciones concurrentes para evitar que se destruya la consistencia de la base de datos.

• Recuperación: Proporcionar como mínimo el mismo nivel de recuperación que los sistemas de bases de datos actuales, de forma que, tanto en caso de fallo de hardware como de fallo de software, el sistema pueda retroceder hasta un estado coherente de los datos.

• Gestión del almacenamiento secundario: Es soportada por un conjunto de mecanismos que no son visibles al usuario, tales como gestión de índices, agrupación de datos, selección del camino de acceso, optimización de consultas, etc.

Estos mecanismos evitan que los programadores tengan que escribir programas para mantener índices, asignar el almacenamiento en disco, o trasladar los datos entre el

disco y la memoria principal, creándose de esta forma una independencia entre los niveles lógicos y físicos del sistema.

• Facilidad de Consultas: Permitir al usuario hacer cuestiones sencillas a la base de datos. Este tipo de consultas tienen como misión proporcionar la información solicitada por el usuario de una forma correcta y rápida.

• La abstracción: La abstracción denota las características esenciales de un objeto que lo distinguen de todos los demás tipos de objetos y proporciona así fronteras conceptuales nítidamente definidas respecto a la perspectiva del observador.

• El encapsulamiento: Es el proceso de almacenar en un mismo compartimiento los elementos de una abstracción que constituyen una estructura y su comportamiento; sirve para separar la interfaz contractual de una abstracción y su implantación. (El encapsulamiento oculta los detalles de una implementación de un objeto).

• La modularidad: Cuando se habla de modularidad hay que imaginarse la fragmentación de un programa en componentes individuales para reducir la complejidad esto es como (modulos).

• La jerarquía: Para la jerarquía es “una clasificación u ordenación de abstracciones”.

La herencia es el ejemplo más importante de jerarquía (es un), ya que va ha definir las relaciones entre clases, tanto en el caso de herencia simple como en el caso de herencia múltiple.

Cuando se habla de herencia (es un) normalmente se refiere a generalización / especialización. Sin embargo (parte de) es más bien un caso de agregación. La combinación de la herencia y la agregación puede ser muy potente: La agregación

permite el agrupamiento físico de las estructuras lógicas, y la jerarquía permite a estos grupos su funcionamiento en diferentes niveles de abstracción.

• Los Tipos: Son la puesta en vigor de la clase de los objetos, de modo que los objetos de tipos distintos no pueden intercambiarse o, como mucho, pueden intercambiarse sólo de formas muy restringidas.

4.1.15. Características de un SGBDOO

Atendiendo a importancia de las características de un sistema de BDOO, las podemos clasificar en tres grupos:

• Obligatorias: Son las que el Sistema debe satisfacer a orden de tener un sistema de BDOO y estos son: Objetos complejos, identidad de objetos, encapsulación, tipos o clases, sobre paso combinado con unión retardada, extensibilidad, compleción computacional, persistencia y manejador de almacenamiento secundario, concurrencia, recuperación y facilidad de consultas.

• Opcionales: Son las que pueden ser añadidas para hacer el sistema mejor pero que no son obligatorias. Éstas son: herencia múltiple, comprobación de tipos e inferencia, distribución y diseño de transacciones y versiones.

• Abiertas: Son los puntos en donde el diseñador puede hacer una serie de opciones, como son: el paradigma de la programación, la representación del sistema o el tipo de sistema y su uniformidad.

4.1.15.1. Características obligatorias

Estos son puntos que no deben faltar en una BD.

• Extensibilidad: Proporciona los tipos de datos como: carácter, lógico, cadena, etc.

• Concurrencia: Permite que varios usuarios tengan acceso a una misma BD de forma simultánea.

• Recuperación: Permite que se retorne al estado inicial de una transacción cuando ésta se intenta realizar pero resulta imposible completarla.

• Facilidad de “Consultas a Modo”: Es decir, que se ajuste a varios estándares.

4.1.15.2. Características opcionales

Éstas dependen del tipo de producto que se desee realizar.

• Herencia múltiple: Tienen características de padres diferentes y proporcionan mecanismos para saber cuál de dos o más opciones conviene utilizar en cada momento.

• Verificación de tipos de inferencia.

-Distribución: Permite que se puedan tener unas partes de una BD en un servidor y otras partes en otro.

-Sistema de Representación: Forma en la cual se presentan los esquemas.

-Uniformidad: Todo debe ser igual; el diseño de ventanas, etc.

-Asociaciones y cardinalidad de asociaciones: Cardinalidad 1:1 (Uno a Uno), 1:M (Uno a Muchos), M:1(Muchos a Uno), M:M (Muchos a Muchos).

4.1.15.3. Control de concurrencia

Existen varios enfoques para realizar el control:

• Modo pesimista: Tomamos el dato y no dejamos que nadie lo tome para que no accedan al mismo dato.

• Modo optimista: Tomamos una copia del datos y pensamos que nadie lo va a modificar.

• Modo mixto: Combinación del Pesimista y el Optimista.

• Modo semioptimista: Toma las virtudes del Optimista.

4.1.15.4. Bloqueos

• Bloqueos de lectura: Lectura de un objeto e imposibilidad de modificación del objeto por parte de otros usuarios mientras se está utilizando.

• Bloqueos de escritura: Bloqueo del objeto mientras se está escribiendo (nadie más puede escribir).

• Bloqueos de notificación: Sirven para sincronización (por ejemplo, el "*paper out*" de notificación de una impresora).

4.2 Diseño Conceptual bajo el Modelo Entidad Relación Extendido

El diseño de una base de datos se divide en tres pasos, el diseño conceptual, el diseño lógico y el diseño físico. El objetivo del diseño conceptual es capturar los requerimientos del problema en un modelo expresivo y simple, de tal manera que tenga el mismo significado para los diseñadores y los usuarios. Uno de los modelos que cubren esas propiedades es el modelo entidad-relación extendido (MERE).

El modelo entidad relación, originalmente propuesto por Chen, fue enriquecido por el mismo Chen y otros autores, entre los que destaca Teorey. Estas extensiones surgieron de la problemática encontrada en el diseño de bases de datos complejas o de tamaño grande. Las extensiones al modelo original hicieron difícil la aprobación de un modelo entidad- relación con una notación y metodología estándar. En vez de ello, los modelos se han agrupado en familias, como son las de Chen, Teorey y Reiner, las de Everest y las de IDEF1X (desarrollado por la Fuerza Aérea de los E.U.A.). La principal diferencia en las metodologías consiste en que algunas extienden el grado de las relaciones a tres mientras otras sólo permiten dos, diferencias en su notación (aunque se puedan encontrar equivalencias de los objetos visuales entre una familia a otra), etc.

El modelo se define de forma sintáctica en un diagrama entidad-relación, cuya semántica se transfiere posteriormente a un modelo lógico de la base de datos. Pero para iniciar el diseño conceptual se necesitan conocer las definiciones de los elementos básicos: entidad, atributo y relación.

• Entidades: Las entidades denotan los objetos de interés del sistema (personas, facturas, elementos, etc.). A cada ocurrencia particular de una entidad se le denomina instancia de la entidad.

• Atributos: Referente a entidades, un atributo puede definirse como una propiedad o característica de las entidades. Cuando describe una característica atómica se le denomina atributo simple o sin agregados; si describe más de una característica, se le denomina atributo compuesto. Al atributo o conjunto de atributos que distinguen de

manera única una instancia de entidad se le denomina clave. Al resto de atributos se les denomina descriptores.

• Relaciones: Son las declaraciones de las interacciones o vínculos que se tienen entre las entidades. Una instancia de relación es una declaración de los vínculos entre instancias de las entidades. Las relaciones pueden contener atributos descriptivos inherentes (que deben describir las características de la relación y no de las entidades participantes en ella).

4.2.1 Grado de las relaciones

Al número de entidades participantes en una relación se le conoce como grado de la relación.

De este modo, una relación de grado uno es una relación recursiva, ya que sólo participa una entidad. Una relación de grado dos (o binaria) es una relación entre dos unidades.

El modelo entidad-relación no incluía grados mayores a dos. En el modelo ERE, se añade la posibilidad de que existan relaciones de cualquier grado.

4.2.2 Cardinalidad de las relaciones

El comportamiento de las instancias, dentro de una relación, se describe con la cardinalidad de las relaciones. Los valores posibles de cardinalidad de las instancias es uno (1) o muchos (M). Las relaciones de grado uno y dos tienen cardinalidad uno a uno (1:1), uno a muchos (1:M) o muchos a muchos (M:N). Una relación de grado tres, por ejemplo, puede tener cualquiera de las cardinalidades siguientes: de uno a uno a uno

(1:1:1), de uno a muchos a uno (1:M:1), de uno a muchos a muchos (1:M:N) o de muchos a muchos a muchos (M:N).

4.2.3 Relaciones Jerárquicas: Generalización y Especialización

Una entidad E es una generalización de las entidades E1, E2, ..., En, si obtenemos a la entidad E de la unión de los atributos comunes a E1, E2, ..., En.

Dicha generalización resalta la semejanza entre las entidades E1, E2, ..., En, pues es el resultado de su agrupamiento. Una instancia de E es la generalización de una y sólo una instancia de alguna de las entidades E1, E2, ..., En. En la especialización, partiendo de una entidad E tomamos subconjuntos de atributos para formar con ellos entidades especializadas E1, E2, ..., En, existiendo la posibilidad de que alguna instancia de E forme algún subconjunto y de que genere más de un subconjunto.

A la entidad de tipo más alta E de la generalización y de la especialización se le llama entidad genérica o entidad de supertipo. A las entidades E1, E2, ..., En, de niveles más bajos se les denomina entidades de subtipo. La principal diferencia entre la generalización y la especialización es que la primera especifica conjuntos mutuamente excluyentes de entidades de subtipo, mientras que la segunda no lo hace.

La generalización y la especialización, también pueden describirse en términos de herencia: la generalización es la relación jerárquica en las cuales los atributos de un supertipo son heredados (o propagados) a una entidad de subtipo y, adicionalmente, ese subtipo puede tener atributos específicos diferentes a los encontrados en el supertipo. En

la especialización, los atributos de un supertipo son heredados a las entidades de subtipo y, adicionalmente, cada uno de esos subtipos contiene atributos específicos.

4.3 METODOLOGÍA UML (Unified Modelating Languaje)

UML significa "Unified Modeling Language": Lenguaje de Modelado o Modelamiento Unificado.

El Lenguaje de Modelado Unificado es un lenguaje usado para especificar, visualizar y documentar los diferentes aspectos relativos a un sistema de software bajo desarrollo, así como para modelado de negocios y otros sistemas no software.

Puede ser utilizado con cualquier metodología, a lo largo del proceso de desarrollo de software, en cualquier plataforma tecnológica de implementación (Unix, Windows etc.).

Es un sistema notacional (que, entre otras cosas, incluye el significado de sus notaciones) destinado a los sistemas de modelado que utilizan conceptos orientados a objetos.

Los principales factores que motivaron la definición de UML fueron: la necesidad de modelar sistemas, las tendencias en la industria del software, unificar los distintos lenguajes y métodos existentes e innovar los modelos para adaptarse a la arquitectura distribuída.

Es importante resaltar que un modelo UML describe lo que supuestamente hará un sistema, pero no dice como implementar dicho sistema.

4.3.1 DIFERENTES DEFINICIONES DE UML

El Lenguaje Unificado de Modelado prescribe un conjunto de notaciones y diagramas estándar para modelar sistemas orientados a objetos, y describe la semántica esencial de lo que estos diagramas y símbolos significan. Mientras que ha habido muchas notaciones y métodos usados para el diseño orientado a objetos, ahora los modeladores sólo tienen que aprender una única notación.

UML se puede usar para modelar distintos tipos de sistemas: sistemas de software, sistemas de hardware, y organizaciones del mundo real.

El UML es una técnica de modelado de objetos y como tal supone una abstracción de un sistema para llegar a construirlo en términos concretos. El modelado no es más que la construcción de un modelo a partir de una especificación. Un modelo es una abstracción de algo, que se elabora para comprender ese algo antes de construirlo. El modelo omite detalles que no resultan esenciales para la comprensión del original y por lo tanto facilita dicha comprensión.

UML es una consolidación de muchas de las notaciones y conceptos más usadas orientados a objetos. Empezó como una consolidación del trabajo de Grade Booch, James Rumbaugh, e Ivar Jacobson, creadores de tres de las metodologías orientadas a objetos más populares.

4.3.2 BREVE RESEÑA HISTÓRICA

El desarrollo de UML comenzó en octubre de 1994 cuando Grady Booch y Jim Rumbaugh de Rational Software Corporation comenzaron a trabajar en la unificación de los lenguajes de modelado Booch y OMT, desde este momento fueron reconocidos

mundialmente en el desarrollo de metodologías orientadas a objetos. Así, en octubre de 1995, terminaron su trabajo de unificación obteniendo el borrador de la versión 0.8 del denominado *Unified Method*. Hacia fines de este mismo año, Ivar Jacobson (creador de la metodología OOSE - Object Oriented Software Engineer) se unió con Rational Software para obtener finalmente UML 0.9 y 0.91 en junio y octubre de 1996, respectivamente. Igualmente, UML incorpora ideas de otros metodólogos entre los que podemos incluir a Peter Coad, Derek Coleman, Ward Cunningham, David Harel, Richard Helm, Ralph Johnson, Stephen Mellor, Bertrand Meyer, Jim Odell, Kenny Rubin, Sally Shlaer, John Vlissides, Paul Ward, Rebecca Wirfs-Brock y Ed Yourdon.

Luego, muchas organizaciones como Microsoft, Hewlett-Packard, Oracle, Sterling Software MCI Systemhouse, Unisys, ICON Computing, IntelliCorp, i-Logix, IBM, ObjectTime, Platinum Technology, Ptech, Taskon, Reich Technologies, Softeam se asociaron con Rational Software Corporation para dar como resultado UML 1.0 y UML 1.1. Hoy en día llegamos hasta UML 1.4 y UML 2.0 así lo demuestra la Fig 9.

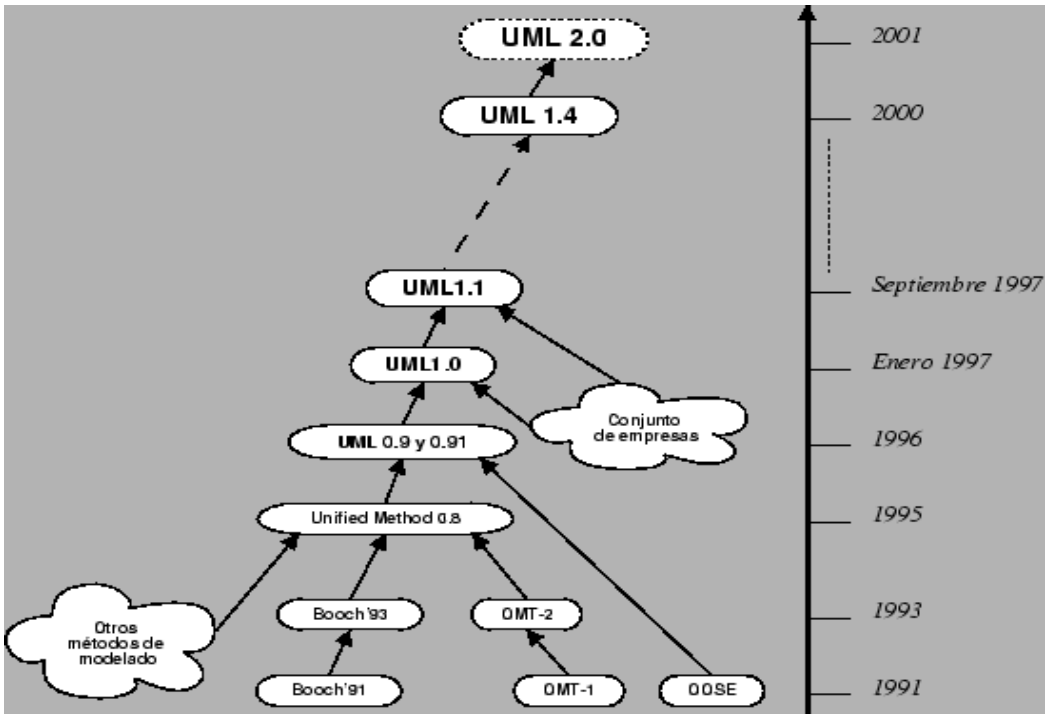


Fig 9 Representación de la evolución de UML

4.3.3 CARACTERÍSTICAS DE UML

UML es una especificación de notación orientada a objetos. Se basa en las anteriores especificaciones BOOCH, RUMBAUGH y COAD-YOURDON. Divide cada proyecto en un número de diagramas que representan las diferentes vistas del proyecto. Estos diagramas juntos son los que representa la arquitectura del proyecto.

UML permite describir un sistema en diferentes niveles de abstracción, simplificando la complejidad sin perder información, para que tanto usuarios, líderes y desarrolladores puedan comprender claramente las características de la aplicación.

UML se quiere convertir en un lenguaje estándar con el que sea posible modelar todos los componentes del proceso de desarrollo de aplicaciones. Sin embargo, hay que tener

en cuenta un aspecto importante del modelo: no pretende definir un modelo estándar de desarrollo, sino únicamente un lenguaje de modelado. Otros métodos de modelaje como OMT (Object Modeling Technique) o Booch sí definen procesos concretos. En UML los procesos de desarrollo son diferentes según los distintos dominios de trabajo; no puede ser el mismo el proceso para crear una aplicación en tiempo real, que el proceso de desarrollo de una aplicación orientada a gestión, por poner un ejemplo.

El método del UML recomienda utilizar los procesos que otras metodologías tienen definidos.

Como objetivos principales de la consecución de un nuevo método que aunara los mejores aspectos de sus predecesores, sus protagonistas se propusieron lo siguiente:

El método debía ser capaz de modelar no sólo sistemas de software sino otro tipo de sistemas reales de la empresa, siempre utilizando los conceptos de la orientación a objetos.

Crear un lenguaje para modelado utilizable a la vez por máquinas y por personas.

Establecer un acoplamiento explícito de los conceptos y los artefactos ejecutables.

Manejar los problemas típicos de los sistemas complejos de misión crítica.

Lo que se intenta es lograr con esto que los lenguajes que se aplican siguiendo los métodos más utilizados sigan evolucionando en conjunto y no por separado. Y además, unificar las perspectivas entre diferentes tipos de sistemas (no sólo software, sino también en el ámbito de los negocios), al aclarar las fases de desarrollo, los requerimientos de análisis, el diseño, la implementación y los conceptos internos de la orientación a objetos.

4.3.4 DESCRIPCION DEL MODELO UML: Nociones Generales

El UML es una técnica de modelado de objetos y como tal supone una abstracción de un sistema para llegar a construirlo en términos concretos. El modelado no es más que la construcción de un modelo a partir de una especificación. Un modelo es una abstracción de algo, que se elabora para comprender ese algo antes de construirlo. El modelo omite detalles que no resultan esenciales para la comprensión del original y por lo tanto facilita dicha comprensión. Los modelos se utilizan en muchas actividades de la vida humana: antes de construir una casa el arquitecto utiliza un plano, los músicos representan la música en forma de notas musicales, los artistas pintan sobre el lienzo con carboncillos antes de empezar a utilizar los óleos, etc. Unos y otros abstraen una realidad compleja sobre unos bocetos, modelos al fin y al cabo. La OMT, por ejemplo, intenta abstraer la realidad utilizando tres clases de modelos OO: el modelo de objetos, que describe la estructura estática; el modelo dinámico, con el que describe las relaciones temporales entre objetos; y el modelo funcional que describe las relaciones funcionales entre valores. Mediante estas tres fases de construcción de modelos, se consigue una abstracción de la realidad que tiene en sí misma información sobre las principales características de ésta.

Los modelos además, al no ser una representación que incluya todos los detalles de los originales, permiten probar más fácilmente los sistemas que modelan y determinar los errores. Según se indica en la Metodología OMT (Rumbaugh), los modelos permiten una mejor comunicación con el cliente por distintas razones:

Es posible enseñar al cliente una posible aproximación de lo que será el producto final.

Proporcionan una primera aproximación al problema que permite visualizar cómo quedará el resultado.

Reducen la complejidad del original en subconjuntos que son fácilmente tratables por separado.

Se consigue un modelo completo de la realidad cuando el modelo captura los aspectos importantes del problema y omite el resto. Los lenguajes de programación que estamos acostumbrados a utilizar no son adecuados para realizar modelos completos de sistemas reales porque necesitan una especificación total con detalles que no son importantes para el algoritmo que están implementando.

Con la creación del UML se persigue obtener un lenguaje que sea capaz de abstraer cualquier tipo de sistema, sea informático o no, mediante los diagramas, es decir, mediante representaciones gráficas que contienen toda la información relevante del sistema. Un diagrama es una representación gráfica de una colección de elementos del modelo, que habitualmente toma forma de grafo donde los arcos que conectan sus vértices son las relaciones entre los objetos y los vértices se corresponden con los elementos del modelo. Los distintos puntos de vista de un sistema real que se quieren representar para obtener el modelo se dibuja dé forma que se resaltan los detalles necesarios para entender el sistema.

4.3.5 DIAGRAMAS.

En todos los ámbitos de la ingeniería se construyen modelos, en realidad, simplificaciones de la realidad, para comprender mejor el sistema que vamos a desarrollar: los arquitectos utilizan y construyen planos (modelos) de los edificios, los grandes diseñadores de coches preparan modelos en sistemas CAD/CAM con todos los detalles y los ingenieros de software deberían igualmente construir modelos de los sistemas software.

Para la construcción de modelos, hay que centrarse en los detalles relevantes mientras se ignoran los demás, por lo cual con un único modelo no tenemos bastante. Varios modelos aportan diferentes vistas de un sistema los cuales nos ayudan a comprenderlo desde varios frentes. Así, UML recomienda la utilización de nueve diagramas para representar las distintas vistas de un sistema.

Los diagramas de UML son los siguientes:

Diagrama de Casos de Uso: modela la funcionalidad del sistema agrupándola en descripciones de acciones ejecutadas por un sistema para obtener un resultado.

Se utiliza para entender el uso del sistema

Muestra el conjunto de casos de uso y actores(Un actor puede ser tanto un sistema como una persona) y sus relaciones: es decir, muestra quien puede hacer qué y las relaciones que existen entre acciones (casos de uso). Son muy importantes para modelar y organizar el comportamiento del sistema.

Diagrama de Clases: muestra las clases (descripciones de objetos que comparten características comunes) que componen el sistema y cómo se relacionan entre sí.

Diagrama de Objetos: muestra una serie de objetos (instancias de las clases) y sus relaciones. A diferencia de los diagramas anteriores, estos diagramas se enfocan en la perspectiva de casos reales o prototipos. Es un diagrama de instancias de las clases mostradas en el diagrama de clases.

Diagrama de Secuencia: enfatiza la interacción entre los objetos y los mensajes que intercambian entre sí junto con el orden temporal de los mismos.

Diagrama de Colaboración: igualmente, muestra la interacción entre los objetos resaltando la organización estructural de los objetos en lugar del orden de los mensajes intercambiados.

El diagrama de secuencia y el diagrama de colaboración: muestran a los diferentes objetos y las relaciones que pueden tener entre ellos, los mensajes que se envían entre ellos. Son dos diagramas diferentes, que se puede pasar de uno a otro sin perdida de información, pero que nos dan puntos de vista diferentes del sistema. En resumen, cualquiera de los dos es un Diagrama de Interacción.

Diagrama de Estados: Se utiliza para analizar los cambios de estado de los objetos. Muestra los estados, eventos, transiciones y actividades de los diferentes objetos. Son útiles en sistemas que reaccionen a eventos.

Diagrama de Actividades: Es un caso especial del diagrama de estados, simplifica el diagrama de estados modelando el comportamiento mediante flujos de actividades. Muestra el flujo entre los objetos. Se utilizan para modelar el funcionamiento del sistema y el flujo de control entre objetos.

Diagrama de Componentes: muestra la organización y las dependencias entre un conjunto de componentes. Se usan para agrupar clases en componentes o módulos.

Diagrama de Despliegue (o implementación): muestra los dispositivos que se encuentran en un sistema y su distribución en el mismo. Se utiliza para identificar Sistemas de Cooperación: Durante el proceso de desarrollo el equipo averiguará de qué sistemas dependerá el nuevo sistema y que otros sistemas dependerán de él.

Clasificación de Diagramas.

Se dispone de dos tipos diferentes de diagramas los que dan una vista estática del sistema Fig 10 y los que dan una visión dinámica Fig 11.

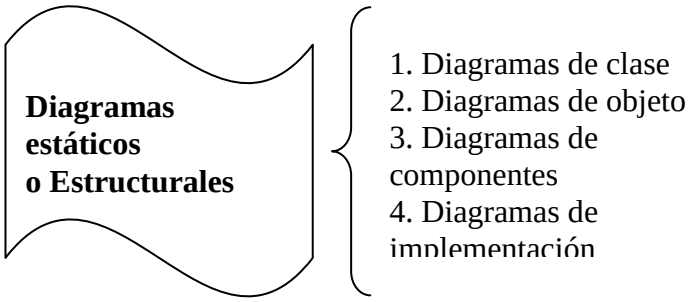
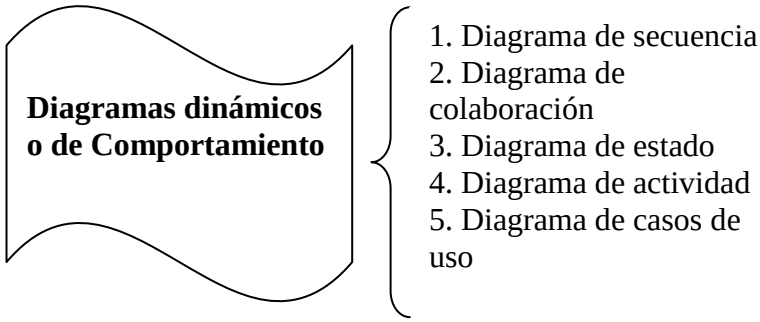


Fig 10 Clasificación de Diagramas Estáticos

La practica de crear diagramas para visualizar sistemas desde perspectivas o vistas diferentes no está limitado a la industria de la construcción. En el contexto del software, existen cinco vistas complementarias que son las más importantes para visualizar, especificar, construir y documentar la arquitectura del software. En el UML las vistas



existentes son:

Fig 11 Clasificación de Diagramas Dinámicos

Vista casos de uso: se forma con los diagramas de casos de uso, colaboración, estados y actividades.

Vista de diseño: se forma con los diagramas de clases, objetos, colaboración, estados y actividades.

Vista de procesos: se forma con los diagramas de la vista de diseño. Recalcando las clases y objetos referentes a procesos.

Vista de implementación: se forma con los diagramas de componentes, colaboración, estados y actividades.

Vista de despliegue: se forma con los diagramas de despliegue, interacción, estados y actividades.

Como podemos ver el número de diagramas es muy alto, en la mayoría de los casos excesivos, y UML permite definir solo los necesarios, ya que no todos son necesarios en todos los proyectos

UML esta pensado para el modelado tanto de pequeños sistemas como de sistemas complejos, y debemos tener en cuenta que los sistemas complejos pueden estar compuestos por millones de líneas de código y ser realizados por equipos de centenares de programadores.

En la práctica todos los diagramas son bidimensionales, pero el UML permite crear diagramas en tres dimensiones como en modelos donde se puede "entrar" al modelo para poderlo visualizar por dentro.

Con UML nos debemos olvidar del protagonismo excesivo que se le da al diagrama de clases, este representa una parte importante del sistema, pero solo representa una vista estática, es decir muestra al sistema parado. Sabemos su estructura pero no sabemos que le sucede a sus diferentes partes cuando el sistema empieza a funcionar.

4.3.5.1 Diagramas Estáticos

4.3.5.1.1 Diagrama de Clases

En el diagrama de clases como ya hemos comentado será donde definiremos las características de cada una de las clases y relaciones de dependencia y generalización. Es decir, es donde daremos rienda suelta a nuestros conocimientos de diseño orientado a objetos, definiendo las clases e implementando las ya típicas relaciones de herencia y agregación.

Este diagrama sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de contenido.

Se utiliza cuando necesitamos realizar un Análisis de Dominio: El analista se entrevista con el cliente con el objetivo de conocer las entidades principales en el dominio del cliente. Durante la conversación entre el cliente y el analista se deben tomar apuntes. Desde estos apuntes, se buscarán las clases para los objetos del modelo buscando los sustantivos (ej: proveedor, pedido, factura, etc.) y convirtiéndolos en clases. Después se verá que algunos de estos sustantivos pueden ser atributos de otros en vez de entidades

por si mismas. También se buscarán los métodos para estas clases buscando los verbos (ej: Calcular, imprimir, Agregar,..)

Elementos

Un diagrama de clases esta compuesto por los siguientes elementos:

Clase: atributos, métodos y visibilidad.

Relaciones: Herencia, Asociación, Ensamblado y Uso.

Clase

Es la unidad básica que encapsula toda la información de un Objeto (un objeto es una instancia de una clase). A través de ella podemos modelar el entorno en estudio (una Casa, un Auto, una Cuenta Corriente, etc.).

En UML, una clase es representada por un rectángulo que posee tres divisiones:

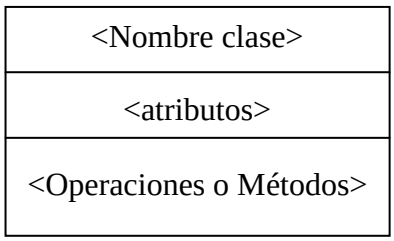


Fig 12 Representación de Clase

En donde:

Superior: Contiene el nombre de la Clase

Intermedio: Contiene los atributos (o variables de instancia) que caracterizan a la Clase (pueden ser private, protected o public).

Inferior: Contiene los métodos u operaciones, los cuales son la forma como interactúa el objeto con su entorno (dependiendo de la *visibilidad*: private, protected o public).

Atributos

Un atributo representa alguna propiedad de la clase que se encuentra en todas las instancias de la clase. Los atributos pueden representarse solo mostrando su nombre, mostrando su nombre y su tipo, e incluso su valor por defecto.

Los atributos o características de una Clase pueden ser de tres tipos, los que definen el grado de comunicación y visibilidad de ellos con el entorno, estos son:

public : Indica que el atributo será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.

private : Indica que el atributo sólo será accesible desde dentro de la clase (sólo sus métodos lo pueden accesar).

protected: Indica que el atributo no será accesible desde fuera de la clase, pero si podrá ser accesado por métodos de la clase además de las subclases que se deriven.

Los atributos definen la estructura de una clase y de sus correspondientes objetos. El atributo define el valor de un dato para todos los objetos pertenecientes a una clase.

Los atributos corresponden a sustantivos y sus valores pueden ser sustantivos o adjetivos.

Se debe definir un valor para cada atributo de una clase. Los valores pueden ser iguales o distintos en los diferentes objetos. No se puede dar un valor en un objeto si no existe un atributo correspondiente en la clase.

Dentro de una clase, los nombre de los atributos deben ser únicos (aunque puede aparecer el mismo nombre de atributo en diferentes clases).

Los atributos no tienen ninguna identidad, al contrario de los objetos.

Un atributo como se ha definido en esta sección se conoce también como atributo básico.

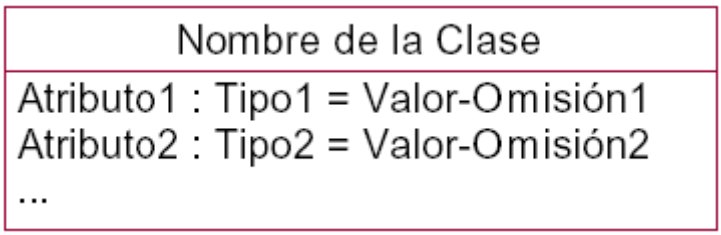


Fig 13 Representación de Notación extendida de clase

Identificadores

En el momento de incluir atributos en la descripción de una clase se debe distinguir entre los atributos los cuales reflejan las características de los objetos en el mundo real, y los identificadores los cuales son utilizados exclusivamente por razones de implementación. Estos identificadores internos del sistema no deben ser incluidos como atributos.

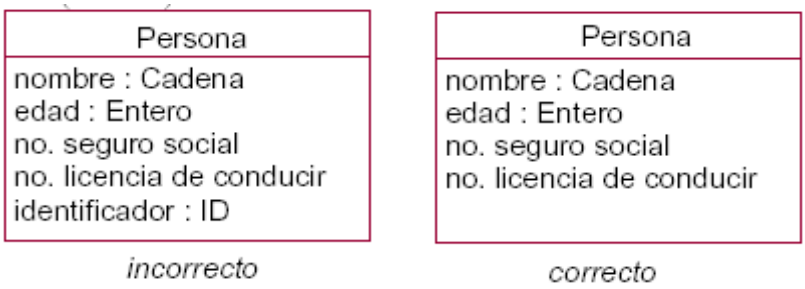


Fig 14 Representación de identificadores

Atributos Derivados

Los atributos básicos son atributos independientes dentro del objeto. En contraste, los atributos derivados son atributos que dependen de otros atributos. Los atributos derivados dependen de otros atributos del objeto, los cuales pueden ser básicos o derivados. La notación es una diagonal como prefijo del atributo, como se muestra en la Fig 15:

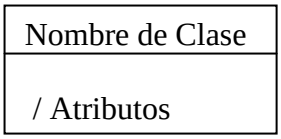


Fig 15 Esquema de Atributos derivados

Restricciones de Atributos

Los valores de los atributos de una clase pueden restringirse. La notación para una restricción es incluir, por debajo de la clase y entre corchetes, la restricción para los valores del atributo, como se muestra en la siguiente Fig 16.

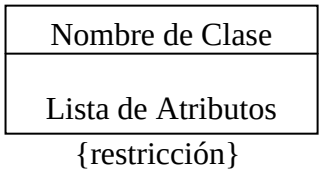


Fig 16 Restriciones de Atributos.

Métodos

Un método u operación es la implementación de un servicio de la clase, que muestra un comportamiento común a todos los objetos. En resumen es una función que le indica a las instancias de la clase que hagan algo.

Las operaciones son funciones o transformaciones que se aplican a todos los objetos de una clase particular. La operación puede ser una acción ejecutada por el objeto o sobre el objeto.

Las operaciones deben ser únicas dentro de una misma clases, aunque no necesariamente para diferentes clases.

No se debe utilizar el mismo nombre para operaciones que tengan un significado totalmente diferente.

Las operaciones pueden tener argumentos, o sea, una lista de parámetros, cada uno con un tipo, y pueden también devolver resultados, cada uno con un tipo.

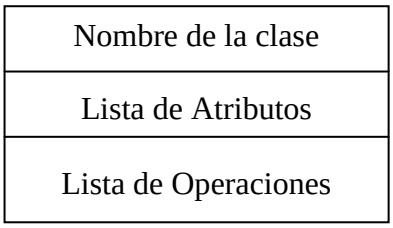


Fig 17 Notación Diagrama de Clases

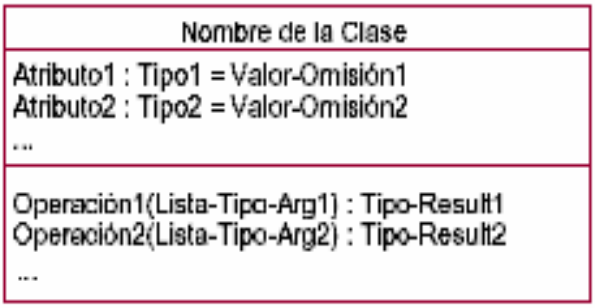


Fig 18 Notación extendida para una clase

Los métodos u operaciones de una clase son la forma en como ésta interactúa con su entorno, éstos pueden tener las características:

public: Indica que el método será visible tanto dentro como fuera de la clase, es decir, es accsesible desde todos lados.

private: Indica que el método sólo será accesible desde dentro de la clase (sólo otros métodos de la clase lo pueden accesar).

protected: Indica que el método no será accesible desde fuera de la clase, pero si podrá ser accesado por métodos de la clase además de métodos de las subclases que se deriven.

Relaciones entre Clases

Existen tres relaciones diferentes entre clases, Dependencias, Generalización y Asociación. En las relaciones se habla de una clase destino y de una clase origen. La

origen es desde la que se realiza la acción de relacionar. Es decir desde la que parte la flecha, la destino es la que recibe la flecha.

Ahora ya definido el concepto de Clase, es necesario explicar como se pueden interrelacionar dos o más clases (cada uno con características y objetivos diferentes).

Antes es necesario explicar el concepto de cardinalidad de relaciones: En UML, la cardinalidad de las relaciones indica el grado y nivel de dependencia, es decir especifica cuantas instancias de una clase se pueden relacionar a una sola instancia de otra clase.

Se anotan en cada extremo de la relación y éstas pueden ser:

uno-uno como en la Fig 19

uno-muchos(1...*) como en la Fig 20

muchos-muchos(* *) ccomo en la Fig 21

opcional (0..1) así como en la Fig 22

número fijo: m(m denota el número).

.

La notación para representar una relación opcional, donde la multiplicidad es "uno" o "cero", escribiendo una relación opcional, 0 o 1. Esto significa que dos objetos pueden o no estar conectados, y si lo están corresponden a una multiplicidad de 1.

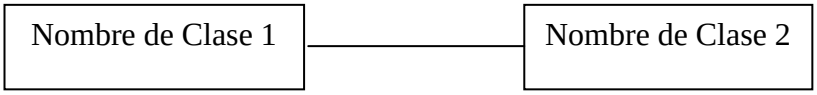


Fig 19 Diagrama de clases “uno a uno”

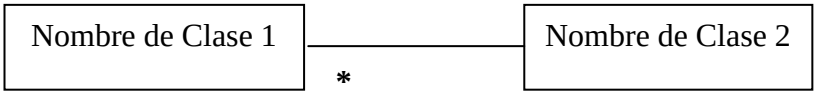


Fig 20 Diagrama de clases “uno a muchos”

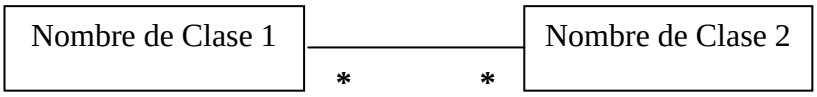


Fig 21 Diagrama de clases “muchos a muchos”

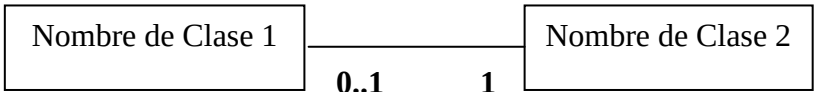


Fig 22 Diagrama de clases “opcional”

Herencia (Especialización/Generalización)

Las clases con atributos y operaciones comunes se pueden organizar de forma jerárquica, mediante la herencia. La herencia es una abstracción importante para compartir similitudes entre clases, donde todos los atributos y operaciones comunes a varias clases se pueden compartir por medio de la superclase, una clase más general. Las clases más refinadas se conocen como las subclases.

La Herencia es útil para el modelo conceptual al igual que para la implementación. Como modelo conceptual da buena estructuración a las clases. Como modelo de implementación es un buen vehículo para no replicar innecesariamente el código. Generalización define una relación entre una clase más generalizada, y una o más versiones refinadas de ella.

Especialización define una relación entre una clase más general, y una o más versiones especializadas de ella.

La superclase generaliza a sus subclases, y las subclases especializan a la superclase. El proceso de especialización es el inverso de generalización. Una instancia de una subclase, o sea un objeto, es también una instancia de su superclase.

La herencia es transitiva a través de un número arbitrario de niveles. Los ancestros de una clase son las superclases de una clase en cualquier nivel superior de la jerarquía, y los descendientes de una clase son las subclases de una clase en cualquier nivel inferior de la jerarquía.

La herencia indica que una subclase hereda los métodos y atributos especificados por una Super Clase, por ende la Subclase además de poseer sus propios métodos y atributos, poseerá las características y atributos visibles de la Super Clase (public y protected).

Los nombres de atributos y operaciones deben ser únicos en la jerarquía de herencia.

La generalización se puede extender a múltiples niveles de jerarquías, donde una clase hereda de su superclase, que a su vez hereda de otra superclase, hacia arriba en la jerarquía. En otras palabras, las relaciones entre subclases y superclases son relativas. La herencia define una jerarquía de clases donde existen ancestros y descendientes, que pueden ser directos o no.

Cada clase tiene sus propios atributos los cuales se van especializando a medida que las clases son cada vez más especializadas. Nótese que no necesariamente todas las clases tienen que incluir atributos.

Asociación

La relación entre clases conocida como Asociación, permite asociar objetos que colaboran entre si.

Una asociación describe la relación entre clases de objetos, y describe posibles ligas, donde una liga es una instancia de una asociación, al igual que un objeto es una instancia de una clase.

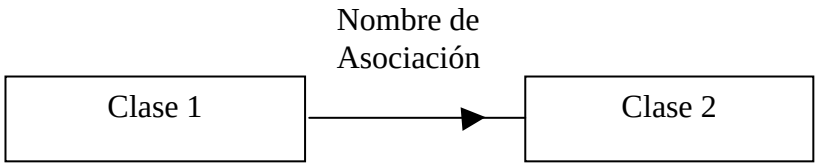


Fig 23 Asociación entre clase1 y clase2

Grado de la Asociación

El grado de una asociación se determina por el número de clases conectadas por la misma asociación. Las asociaciones pueden ser binarias, ternarias, o de mayor grado. Las asociaciones se consideran binarias si relacionan solo dos clases.

Las asociaciones pueden ser de mayor grado si relacionan a la misma vez más de dos clases. Aparte de relaciones binarias, lo más común son relaciones ternarias (entre tres clases) como en la Fig 24, relaciones de más alto nivel son mucho menos comunes. Mientras el grado de una relación aumenta, su comprensión se dificulta, y se debe considerar partir las relaciones en varias relaciones binarias.

El grado de las ligas corresponden al de las asociaciones.

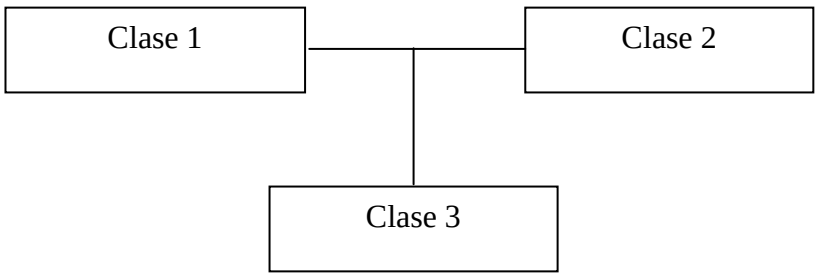


Fig 24 Asociación ternaria

Asociaciones Reflexivas

Las asociaciones pueden ser reflexivas, relacionando distintos objetos de una misma clase.

El grado de una asociación reflexiva puede ser binario, ternario, o de mayor grado, dependiendo del número de objetos involucrados.

Las asociaciones reflexivas relacionan distintos objetos de una misma clase.

Ejemplo: La asociación reflexiva pariente de para la clase Persona se muestra en la Fig25.

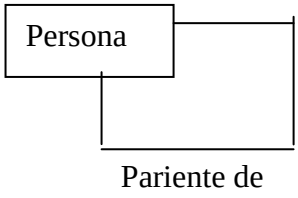


Fig 25 Asociación reflexiva

Atributos de Liga (o Asociación)

Al igual que un atributo de clase es propiedad de la clase, un atributo de asociación (o atributo de liga) es propiedad de una asociación. La notación es similar a la usada para

los atributos de clases, excepto que se añade a la asociación, y no se incorpora un nombre de clase, como se muestra en la Fig 26.

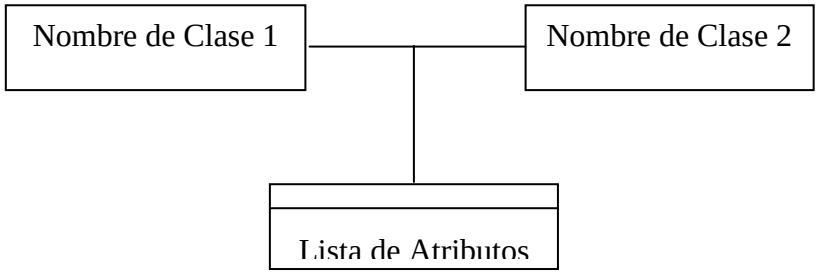


Fig 26 Atributo de asociación

Ensamblados: Agregación y Composición

Los ensamblados, en particular la agregación y composición, son formas especiales de asociación entre un todo y sus partes, en donde el ensamblado está compuesto por sus componentes.

El ensamblado es el objeto central, y la estructura completa se describe como una jerarquía de contenido.

Agregación:

Es un tipo de relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye.

Composición:

Es un tipo de relación estática, en donde el tiempo de vida del objeto incluido esta condicionado por el tiempo de vida del que lo incluye.

Un ensamblado puede componerse de varias partes, donde cada relación se considera una relación separada. En el caso de agregación, las partes del ensamblado pueden aparecer en múltiples ensamblados. En el caso de composición, las partes del ensamblado no pueden ser compartidas entre ensamblados.

Adicionalmente, no tiene mucho sentido que el Motor y la Carrocería existan de manera independiente al Automóvil, por lo cual la composición refleja de manera importante, el concepto de propiedad.

El ensamblado tiene propiedades de transición: Si A es parte de B y B es parte de C; entonces A es parte de C.

El ensamblado es antisimétrico: Si A es parte de B, entonces B no es parte de A. Estas propiedades se propagan entre el ensamblado y sus componentes.

Se considera un ensamblado y no una asociación regular:

Si se puede usar la frase "parte-de" o "consiste-de" o "tiene";

Si algunas operaciones en el todo se pueden propagarse a sus partes;

Si algunos atributos en el todo se pueden propagar a sus partes;

El ensamblado es común en los objetos interfaz. En un sistema de ventanas, por ejemplo, una ventana puede consistir de botones, menús, y barras (“scrollbars”), cada una modelada por su propio objeto interfaz. El resultado es una estructura de interfaz en forma de árbol. La decisión de usar ensamblados es un poco arbitraria, y en la práctica no causa grandes problemas la distinción imprecisa entre agregación, composición y asociación, aunque es bueno ser consistente.

La notación para un ensamblado, en particular para un agregado, es un diamante adherido al lado del objeto correspondiente al ensamblado total, conectado por una línea a sus componentes, como se indica a continuación.

La agregación se destaca por un rombo transparente como en la Fig 27 .

La composición se destaca por un rombo relleno como en la Fig 28.

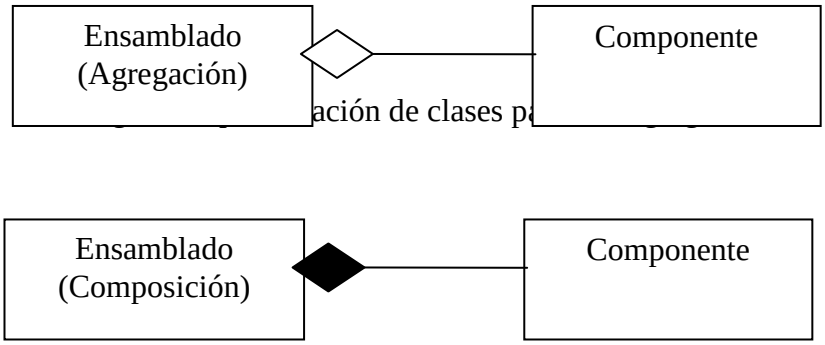


Fig 28 Representación de clases para una composición

La flecha en este tipo de relación indica la navegabilidad del objeto referenciado. Cuando no existe este tipo de particularidad la flecha se elimina.

Dependencia o Instanciación (uso):

Representa un tipo de relación muy particular, en la que una clase es instanciada (su instanciación es dependiente de otro objeto/clase).

Es una relación de uso, es decir una clase usa a otra, que la necesita para su cometido. Se representa con una flecha discontinua va desde la clase utilizadora a la clase

utilizada. Con la dependencia mostramos que un cambio en la clase utilizada puede afectar al funcionamiento de la clase utilizadora, pero no al contrario.

El uso más particular de este tipo de relación es para denotar la dependencia que tiene una clase de otra, como por ejemplo una aplicación gráfica que instancia una ventana Fig 29 (la creación del Objeto Ventana esta condicionado a la instanciación proveniente desde el objeto Aplicación):

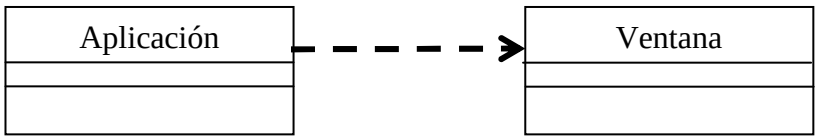


Fig 29 Representación de Instancia

Cabe destacar que el objeto creado (en este caso la Ventana gráfica) no se almacena dentro del objeto que lo crea (en este caso la Aplicación).

4.3.5.1.2 Diagrama de objetos

Pertenece a la clasificación de los diagramas que dan una vista estática del sistema.

Contiene un conjunto de instancias de los elementos encontrados en un Diagrama de Clases. Por lo tanto, expresa la parte estática de una interacción, consistiendo en los objetos que colaboran, pero son ninguno de los mensajes enviados entre ellos.

Para realizarlo primero se debe decidir que situación queremos representar del sistema, en un momento concreto del mismo, permitiendo así mostrar los objetos y sus relaciones

En los diagramas de objetos también se pueden incorporar clases, para mostrar la clase de la que es un objeto representado.

Con los Diagrama de Objetos no se puede especificar completamente la estructura de objetos del sistema. Puede existir una multitud de posibles instancias de una clase particular, y para un conjunto de clases con relaciones entre ellas, pueden existir muchas más configuraciones posibles de esos objetos.

Diagrama

Al momento de construir el Diagrama de Objetos hay que tener bien en claro dos concepto muy importantes.

En primer lugar saber a que nos referimos cuando hablamos de objeto.

Los objetos son las entidades básicas del modelo de objeto. La palabra objeto proviene del latín objectus, donde ob significa hacia, y jacere significa arrojar; o sea, que teóricamente un objeto es cualquier cosa que se pueda arrojar.

Los objetos son más que simples cosas que se puedan arrojar, son conceptos, pudiendo ser abstractos o concretos.

Los objetos corresponden por lo general a sustantivos, pero no a gerundios.

Cualquier cosa que incorpore una estructura y un comportamiento se le puede considerar como un objeto.

Un objeto debe tener una identidad coherente, al que se le puede asignar un nombre razonable y conciso.

La existencia de un objeto depende del contexto del problema. Lo que puede ser un objeto apropiado en una aplicación puede no ser apropiado en otra, y al revés. Por lo general, existen muchos objetos en una aplicación, y parte del desafío es encontrarlos.

Los objetos se definen según el contexto de la aplicación.

Los objetos deben ser entidades que existen de forma independiente. Se debe distinguir entre los objetos, los cuales contienen características o propiedades, y las propias características.

Un grupo de cosas puede ser un objeto si existe como una entidad independiente.

Los objetos deben tener nombres en singular, y no en plural.

Parte de una cosa puede considerarse un objeto.

Los objetos deben tener nombren razonables y concisos para evitar la construcción de objetos que no tengan una identidad coherente.

El objeto integra una estructura de datos (atributos) y un comportamiento (operaciones).

Y segundo lugar, el termino identidad; aclarado a continuación.

Los objetos se distinguen por su propia existencia, su identidad, aunque internamente los valores para todos sus datos sean iguales. Todos los objetos se consideran diferentes.

Los objetos tienen un nombre que puede no ser único.

Los objetos necesitan un identificador interno único cuando son implementados en un sistema de computación para accesar y distinguir entre los objetos. Estos identificadores

no deben incluirse como una propiedad del objeto, ya que solo son importantes en el momento de la implementación.

La notación general para un objeto es una caja rectangular conteniendo el nombre del objeto subrayado, el cual sirve para identificar al objeto, como se muestra en la Fig 30.

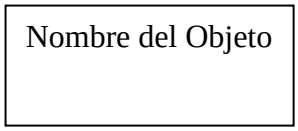


Fig 30 Notación para un objeto

Hay que identificar el mecanismo que se desea modelar. Un mecanismo representa alguna función o comportamiento de la parte del sistema que se está modelando, que resulta de la interacción de una sociedad de clases, interfaces y otros elementos.

Para cada mecanismo hay que identificar las clases, interfaces y otros elementos que participan en esta colaboración; identificar también las relaciones entre estos elementos.

Hay que considerar un escenario en el que intervenga este mecanismo. También hay que congelar este escenario en un momento concreto, y representar cada objeto que participo en el mecanismo.

Hay que mostrar el estado y valores de los atributos de cada uno de esos objetos si son necesarios para comprender el escenario.

Análogamente hay que mostrar los enlaces entre esos objetos, que representan instancias de asociaciones entre ellos.

4.3.5.1.3 Diagrama de Componentes

Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes software, sean éstos componentes de código fuente, binarios o ejecutables. Los elementos de modelado dentro de un diagrama de componentes serán componentes y paquetes. En cuanto a los componentes, sólo aparecen tipos de componentes, ya que las instancias específicas de cada tipo se encuentran en el diagrama de despliegue.

Cada componente en el diagrama debe ser documentado con un diagrama de componentes más detallado, un diagrama de clases, o un diagrama de casos de uso.

Un paquete en un diagrama de componentes representa una división física del sistema. Los paquetes se organizan en una jerarquía de capas donde cada capa tiene una interfaz bien definida. Un diagrama de componentes se representa como un grafo de componentes *software* unidos por medio de relaciones de dependencia (generalmente de compilación).

Normalmente los diagramas de componentes se utilizan para modelar código fuente, versiones ejecutables, bases de datos físicas, entre otros.

Código fuente: En el modelado de código fuentes se suelen utilizar para representar las dependencias entre los ficheros de código fuente, o para modelar las diferentes versiones de estos fichero. Para ello se deben identificar el conjunto de archivos de código fuente de interés y estereotiparlos como archivos file, agruparlos en paquetes, utilizar valores etiquetados para la información de versiones y modelar las dependencias de compilación.

Código ejecutable: Se utiliza para modelar la distribución de una nueva versión a los usuarios. Para tal propósito se identifican el conjunto de componentes ejecutables que intervienen, se utilizan estereotipos para los diferentes tipos de componentes (ejecutables, bibliotecas, tablas, archivos, documentos, etc.), se consideran las relaciones entre dichos componentes que la mayoría de las veces incluirán interfaces que son exportadas (realizadas) por ciertos componentes e importadas (utilizadas) por otros.

Bases de datos física: UML permite el modelado de bases de datos físicas así como de los esquemas lógicos de bases de datos.

Así si tenemos en cuenta las dependencias asociadas al proceso de compilación un componente podría ser:

- Código fuente: que depende de otros componentes (no necesariamente código fuente) que deben estar disponibles durante la compilación del componente.
- Código objeto binario: como por ejemplo una librería, que puede depender de algún código objeto con el que se linkea.
- Código ejecutable: que puede depender de otros programas ejecutables con los que interaccionan en tiempo de ejecución.

El Diagrama de Componentes se usa para modelar la estructura del software, incluyendo las dependencias entre los componentes de software, los componentes de código binario, y los componentes ejecutables. En el Diagrama de Componentes se modela componentes del sistema, a veces agrupados por paquetes, y las dependencias que existen entre componentes (y paquetes de componentes).

4.3.5.1.4 Diagramas de Implementación

Los Diagramas de Implementación se usan para modelar la configuración de los elementos de procesado en tiempo de ejecución (run-time processing elements) y de los componentes, procesos y objetos de software que viven en ellos.

Los Diagramas de Implementación se usan para modelar sólo componentes que existen como entidades en tiempo de ejecución; no se usan para modelar componentes solo de tiempo de compilación o de tiempo de enlazado. Puedes también modelar componentes que migran de nodo a nodo u objetos que migran de componente a componente usando una relación de dependencia con el estereotipo 'becomes' (se transforma)

4.3.5.2 Diagramas dinámicos

4.3.5.2.1 Diagrama de Casos de Usos

Se emplea para visualizar el comportamiento del sistema, una parte de él o de una sola clase; y como se relaciona con su entorno. De ésta forma se puede conocer como responde ésa parte del sistema ante un estímulo del ambiente. El diagrama de uso es muy útil para definir como debería ser el comportamiento de una parte del sistema, ya que solo especifica como deben comportarse y no como están implementadas las partes que define.

Un caso de uso especifica un requerimiento funcional.

Elementos

Un diagrama de casos de uso consta de los siguientes elementos:

Actor

Un actor es un rol que tiene un usuario con respecto al sistema. Es decir, sería un usuario del sistema. Es importante destacar el uso de la palabra “rol”, ya que esto especifica que un actor no necesariamente representa a una persona en particular, si no la labor que realiza frente al sistema.

Caso de Uso

Es una operación o tarea específica que se realiza tras una orden o estímulo de un agente externo, puede ser un actor o desde la invocación desde otro caso de uso.

Asociación

Es el tipo de relación más básica, indica la invocación desde un actor o caso de uso a otra operación (caso de uso). Dicha relación se denota con una flecha simple:

Dependencia o Instanciación

Es una forma muy particular de relación entre clases, en la cual una clase depende de otra, es decir, se instancia (se crea). Dicha relación se denota con una flecha punteada:

Generalización

Este tipo de relación es una de las más utilizadas, cumple una doble función dependiendo de su estereotipo, que puede ser de Uso (<<uses>>) o de Herencia (<<extends>>).

Este tipo de relación esta orientado exclusivamente para casos de uso.

Extends: se recomienda utilizar cuando un caso de uso es similar a otro (en características).

Uses: se recomienda utilizar cuando se tiene un conjunto de características que son similares en más de un caso de uso y no se desea mantener copiada la descripción de la característica.

4.3.5.2.2 Diagrama de Secuencia

Este diagrama (también llamado diagrama de interacción) muestra las interacciones entre un conjunto de objetos (clases, actores), ordenadas según el tiempo en que tienen lugar. Es decir, muestra el orden de las llamadas en el sistema. Se utiliza un diagrama para cada llamada a representar. Es imposible representar en un solo diagrama la secuencia de todas las llamadas posibles del sistema, es por ello que se escoge un punto de partida. El diagrama se compone con los objetos que forman parte de la secuencia, estos se sitúan en la parte superior de la pantalla, normalmente a la izquierda se sitúa el que inicia la acción. De estos objetos sale una línea que indica su vida en el sistema. Esta línea simple se convierte en una línea gruesa cuando representa que el objeto tiene el foco del sistema, es decir cuando él esta activo.

El objeto puede existir sólo durante la ejecución de la interacción, se puede crear o puede ser destruido durante la ejecución de la interacción.

En este tipo de diagramas también intervienen los mensajes, que son la forma en que se comunican los objetos: el objeto origen solicita (llama a) una operación del objeto destino. El diagrama de secuencia puede ser obtenido de dos partes, desde el Diagrama Estático de Clases o desde el de Casos de Usos.

Elementos

Los componentes de un diagrama de interacción son:

Línea de vida

Un objeto (o actor) se representa como una línea vertical punteada con un rectángulo de encabezado y con rectángulos a través de la línea principal que denotan la ejecución de métodos (activación). El rectángulo de encabezado contiene el nombre del objeto y el de su clase como se muestra en la Fig 31.

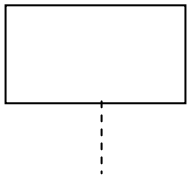


Fig 31 Línea de vida

Activación

Muestra el periodo de tiempo en el cual el objeto se encuentra desarrollando alguna operación, bien sea por sí mismo o por medio de delegación a alguno de sus atributos. Se denota como un rectángulo delgado sobre la línea de vida del objeto.

Mensajes

El envío de mensajes entre objetos se denota mediante una línea sólida dirigida, desde el objeto que emite el mensaje hacia el objeto que lo ejecuta como en la Fig 32.

Representa la llamada de un método (operación) de un objeto en particular.

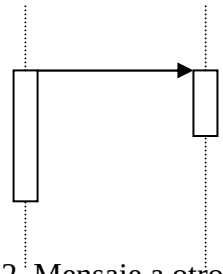


Fig 32 Mensaje a otro objeto

También es posible visualizar llamadas a métodos desde el mismo objeto en estudio como en la Fig 33. El presente diagrama es útil para observar la vida de los objetos en el sistema, identificar llamadas a realizar o posibles errores del modelo estático, que imposibiliten el flujo de información o de llamadas entre los componentes del sistema.

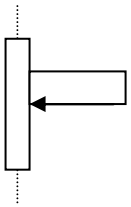


Fig 33 Mensaje al mismo objeto

4.3.5.2.3 Diagrama de Colaboración

Este diagrama muestra la interacción entre varios objetos y los enlaces que existen entre ellos. Representa las interacciones entre objetos organizadas alrededor de los objetos y sus vinculaciones. A diferencia de un diagrama de secuencias, un diagrama de colaboraciones muestra las relaciones entre los objetos, no la secuencia en el tiempo en

que se producen los mensajes. Los diagramas de secuencias y los diagramas de colaboraciones expresan información similar, pero en una forma diferente.

Elementos

Los elementos que intervienen en éstos diagramas son: objetos, enlaces y flujos de mensajes.

Objeto

Un objeto se representa con un rectángulo, que contiene el nombre y la clase del objeto. Un objeto es una instancia de una clase que participa como una interacción, existen objetos simples y complejos. Un objeto es activo si posee un thread o hilo de control y es capaz de iniciar la actividad de control, mientras que un objeto es pasivo si mantiene datos pero no inicia la actividad.

Enlace

Un enlace es una instancia de una asociación en un diagrama de clases. Se representa como una línea continua que une a dos objetos en este diagrama. Esta acompañada por un número que indica el orden dentro de la interacción y por un estereotipo que indica que tipo de objeto recibe el mensaje. El enlace puede ser reflexivo si conecta a un elemento consigo mismo. La existencia de un enlace entre dos objetos indica que puede existir un intercambio de mensajes entre los objetos conectados.

Flujo de mensajes

Expresa el envío de un mensaje. Se representa mediante una flecha dirigida cercana a un enlace. Los mensajes que se envían entre objetos pueden ser de distintos tipos, también según como se producen en el tiempo; existen mensajes simples, sincrónicos, balking, timeout y asíncronos.

Durante la ejecución de un diagrama de colaboración se crean y destruyen objetos y enlaces.

4.3.5.2.4 Diagrama de actividad

Son similares a los diagramas de flujos de otras metodologías OO. En realidad se corresponden con un caso especial de los diagramas de estado donde los estados son estados de acción (estados con una acción interna y una o más transiciones que suceden al finalizar esta acción, o lo que es lo mismo, un paso en la ejecución de lo que será un procedimiento) y las transiciones vienen provocadas por la finalización de las acciones que tienen lugar en los estados de origen. Siempre van unidos a una clase o a la implementación de un caso de uso o de un método (que tiene el mismo significado que en cualquier otra metodología OO). Los diagramas de actividad se utilizan para mostrar el flujo de operaciones que se desencadenan en un procedimiento interno del sistema.

4.3.5.2.5 Diagrama de estado

Representan la secuencia de estados por los que un objeto o una interacción entre objetos pasa durante su tiempo de vida en respuesta a estímulos recibidos. Representa lo

que podemos denominar en conjunto una máquina de estados. Un estado en UML es cuando un objeto o una interacción satisface una condición, desarrolla alguna acción o se encuentra esperando un evento.

Cuando un objeto o una interacción pasa de un estado a otro por la ocurrencia de un estímulo o evento se dice que ha sufrido una transición, existen varios tipos de transiciones entre objetos: simples (normales y reflexivas) y complejas. Además una transición puede ser interna si el estado del que parte el objeto o interacción es el mismo que al que llega, no se provoca un cambio de estado y se representan dentro del estado, no de la transición.

4.3.6 Herramientas Case que soporta UML

El Rational Unified Process describe cómo modelar visualmente aplicaciones para capturar la estructura y el comportamiento de la arquitectura y de los componentes. *Rational Rose* es la mejor herramienta para llevar a cabo el modelado visual. Permite ocultar y mostrar los detalles según el nivel de abstracción requerido y escribir la aplicación mediante bloques de construcción gráficos. Las abstracciones visuales permiten comunicar los diferentes aspectos del software; mostrar cómo los elementos del sistema encajan entre sí; asegurar que los bloques sean consistentes con el código; mantener la consistencia entre el diseño y la implementación; y promover una comunicación clara entre todos los participantes del proyecto.

Rational Rose es la herramienta CASE que comercializan los desarrolladores de UML y que soporta de forma completa la especificación del UML.

Esta herramienta propone la utilización de cuatro tipos de modelo para realizar un diseño del sistema, utilizando una vista estática y otra dinámica de los modelos del sistema, uno lógico y otro físico. Permite crear y refinar estas vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de software.

System Architect 2001

Popkin software ofrece soporte para modelar sistemas con UML en System Architect 2001. Ofrece todas las características descriptas arriba para permitir el modelado eficiente de sistemas.

4.3.7 Conclusión de UML

A nuestro parecer UML es la mejor solución para todos los profesionales relacionados con el Análisis de Sistemas, ya que si nos tocara trabajar en un proyecto de software, en el cual sabemos que el número de integrantes no es para nada reducido (si se trabajase en empresas grandes), sin la aplicación de UML, se hace engorroso ponerse de acuerdo en las metodologías que se utilizarán, en las notaciones que se emplearán para cada modelo (ya sea de análisis o de implementación). Es por ello que este nuevo paradigma de diseño nos posibilita unificar todos nuestros criterios, para un posterior entendimiento, y mejor organización de los proyectos.

Como desventaja podemos destacar (aunque para algunos o muchos no lo sería) que UML permite especificar, visualizar y construir software's, pero orientado a objetos.

Para aquellos que prefieran las metodologías estructuradas deberán esperar que surja un Lenguaje Unificado de Modelado Estructurado.

Pensamos que esto no sucederá, a lo sumo aparecerá alguna extensión de UML para considerar algún aspecto del modelo estructurado, pero a nuestro parecer no sería muy justificable, por que a lo que se apunta en la actualidad es a la aplicación de metodologías orientadas a objetos, pues éstas brindan muchas ventajas y solucionan muchos de los problemas que surgen en las metodologías estructuradas.

CAPITULO V

METODOLOGÍA DE TRANSFORMACIÓN A ESQUEMAS ORIENTADOS A OBJETOS

5.1. Detalle de la Metodología

La metodología propuesta es el resultado de la experiencia adquirida durante el estudio y desarrollo de esta Tesis.

5.2. Requerimientos para la aplicación de la Metodología

El encargado en la transformación del modelado de datos debe tener conocimientos básicos sobre la notación en esquemas relacionales y orientado a objetos para que pueda obtener resultados óptimos con la aplicación de nuestra metodología. Hay que comprender el problema que se debe resolver.

Vale aclarar que esta metodología antepone la agilidad y velocidad en que un modelado relacional pueda ser tratado como un modelado orientado a objetos.

La interpretación de la semántica depende del proceso de cada aplicación y de nuestro propio juicio.

5.3. Descripción de la Metodología

Conociendo las preguntas fundamentales para el proceso de transformación, es decir, donde estoy y ha donde quiero llegar, la metodología recomienda la aproximación gradual a los objetivos propuestos.

5.4. Desarrollo de la Metodología

Para el desarrollo de esta metodología nos hemos basado en conceptos fundamentales como son la Técnica de Agrupación de Clusters, traducción mapeada y traducción estable los cuales los detallamos de una forma rápida más adelante con lo cual hemos podido llevar a cabo nuestra investigación con el objeto de realizar la transformación de bases de datos Relacionales a un esquema orientado a objetos y se aplicarán las reglas de transformación a un caso de estudio.

Reglas de Negocio

Debemos recordar que las bases de datos se establecen para beneficio de los usuarios finales, es así que debemos tener en cuenta las reglas de negocio las cuales son las que determinan la estructura de la información y las políticas de la empresa.

Toda aplicación trata de reflejar parte del funcionamiento del mundo real, para automatizar tareas que de otro modo serían llevadas a cabo de modo ineficiente, o bien no podrían realizarse. Para ello, es necesario que cada aplicación, en principio, refleje

las restricciones que existen en el negocio dado, de modo que nunca sea posible llevar a cabo acciones no válidas. A las reglas que debe seguir la aplicación para garantizar lo anterior, se las llama reglas de negocio. Ejemplos de tales reglas son: no permitir crear facturas pertenecientes a clientes inexistentes, controlar que el saldo negativo de un cliente nunca sobrepase cierta cantidad, etc.

En realidad, la información puede ser manipulada por muchos programas distintos, así una empresa puede tener un departamento de contabilidad que controle todo lo relacionado con compras, cobros, etc., y otro departamento técnico, que esté interesado en relacionar diversos parámetros de producción con los costos. La visión que ambos departamentos tendrán de la información y sus necesidades será distinta, pero en cualquier caso siempre se deberán respetar las reglas de negocio. El hecho de que la información sea manipulada por diversos programas hace más difícil garantizar que todos respetan las reglas, especialmente si las aplicaciones corren en diversas máquinas, bajo distintos sistemas operativos y están desarrolladas con distintos lenguajes y herramientas.

Existen algunos tipos de reglas de negocio de las cuales se destacan las siguientes.

Reglas del modelo de datos

El primer grupo de reglas del negocio engloba todas aquellas reglas que se encargan de controlar que la información básica almacenada para cada atributo o propiedad de una entidad u objeto sea válida: no hay precios de artículos negativos, el sexo de una

persona solo puede ser masculino o femenino, una fecha siempre debe ser una fecha válida, etc.

Reglas de relación

Otro grupo importante de reglas incluye todas aquellas reglas que controlan las relaciones entre los datos. Estas reglas especifican, por ejemplo, que todo pedido debe ser realizado por un cliente, y que el mismo debe atendido. Además, una vez que un cliente haya hecho algún pedido, se deberá garantizar que no es posible eliminarlo, a menos que previamente se eliminen todos sus pedidos.

Reglas de Restricción: especifican políticas o condiciones que restringen la estructura y comportamiento de los objetos y procesos.

Reglas de Derivación: especifican políticas y condiciones para inferir o calcular hechos (información) a partir de otros hechos existentes en el negocio.

Reglas de flujo

Este grupo de reglas del negocio incluye aquellas reglas que determinan y limitan cómo fluye la información a través de un sistema. Por ejemplo, un cliente puede hacer una petición de análisis a un laboratorio, que anota un encargado, hecho esto, se genera una orden para uno o más analistas, estos realizan las mediciones correspondientes y devuelven las partes con la información pertinente, a partir de la cual se genera un

informe de análisis, que será un análisis válido sólo cuando sea firmado por los responsables de garantizar su corrección. Estas reglas indican qué camino recorre la información y obligan a que se sigan solo los caminos válidos.

Reglas de decisión

Hay otro grupo de reglas del negocio, que son las encargadas de representar las reglas para la toma de decisiones, las cuales determinan cuando, donde y quién debe realizar una tarea especifica.

La Técnica de Formación de Clusters(Agrupamiento)

Llamamos cluster a un subsistema completo o a un agrupamiento de clases o entidades.

El procedimiento consiste en realizar iterativa mente de forma arriba-abajo y empezando de entidades atómicas en el esquema entidad relación y construir entidades más complejas (clusters de entidades) hasta que el nivel de abstracción n sea alcanzado, o hasta que no existan más operaciones de clusters por realizar. El nivel n de clusters representa la profundidad deseable de jerarquías de agregación en el esquema de objetos complejos.

Para asegurar una alta cohesión semántica de las entidades complejas, las operaciones de agrupación se realizan con un orden de prioridad (precedencia). Este orden es definido en términos del concepto de cohesión, para representar el grado de importancia de los vínculos entre las entidades en una entidad cluster.

Traducción estable consiste en convertir cada entidad y cada vínculo en un objeto,

Traducción mapeada, consiste en integrar un vínculo en una clase de objeto usando referencias, y creando una clase de objeto para cada vínculo ternario.

Según nuestro estudio hemos propuesto nuestra metodología en una manera más fácil de entender que resulta de forma gráfica y matemática.

Nuestra Metodología la describimos siguiendo un sin número de reglas para llevar a cabo la Conversión:

***** Una **entidad** en el MER generará una clase en el MOO con sus propios atributos y métodos. Así lo demostramos de la siguiente manera:

En el MER representamos una entidad así como en la Fig 34:

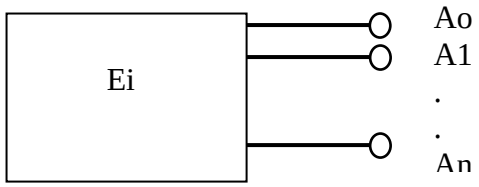


Fig 34 Entidad con atributos

$Ei(A_0:D_0,A_1:D_1,…..An:Dn)$

Donde:

Ai: Di Є Ei

Donde:

A Atributo

D Dominio

E Entidad

Al momento de realizar la conversión tenemos la siguiente clase Fig 35:

Ci={(A_0:D_0,A_1:D_1,…..An:Dn) U (M_0,M_1,.....Mn)}

Donde:

Ci(Ai:Di,Mi)

Ai:Di U Mi Є Ci

Donde:

Mi Método

Ci Clase

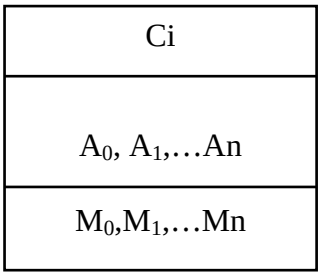


Fig 35 Clase

*** Conversión uno a uno**

Cuando existe una relación uno a uno en el MER Fig 36, La transformación al MOO generan dos clases de objetos, cada clase de objetos contendrá los mismos atributos de

las relaciones en el MER Fig 37, tomando como identificadores a las claves primarias de cada entidad, y cada clase tendrá sus propios métodos.

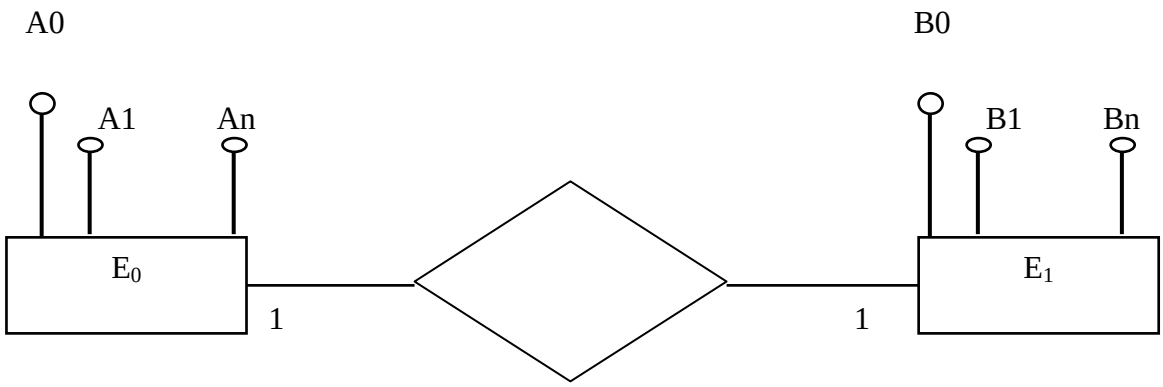


Fig 36 Relación uno a uno.

$E_0 = (A_0:D_0, A_1:D_1, \ldots..An:Dn)$

¥Ai:Di Є E_0

$E_1 = (B_0:D_0, B_1:D_1, \ldots..Bn:Dn)$

¥ Bi:Di Є E_1

Donde ¥ es Todo

A_0 = Cp

B_0 = Cp

Cp clave primaria

En donde al transformar al MOO tenemos:

$C_0 = E_0$ U Mi

$C_0 = (A_0:D_0, A_1:D_1, \ldots..An:Dn)$ U $(M_0, M_1, ...Mn)\}$

C_1= E_1 U Mi

C1={ (B_0:D_0,B_1:D_1,…..Bn:Dn) U (M_0,M_1,.....Mn)}

Donde:

A_0 = Id Є C_0

B_0 = Id Є C_1

Id = Identificador

Quedando la clase como se indica en la Fig 37.

Fig 37 Clase

Con un ejemplo explicaremos mejor esta regla vea Fig 38.

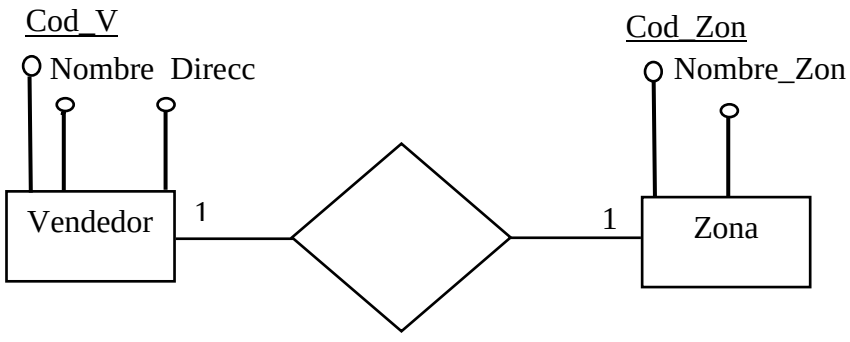


Fig 38 Relación entre un vendedor y una zona

Al transforma esta relación uno a uno al Modelo Orientado a Objetos tendremos que los atributos de las entidades se transforman como atributos de las clase. Fig 39.

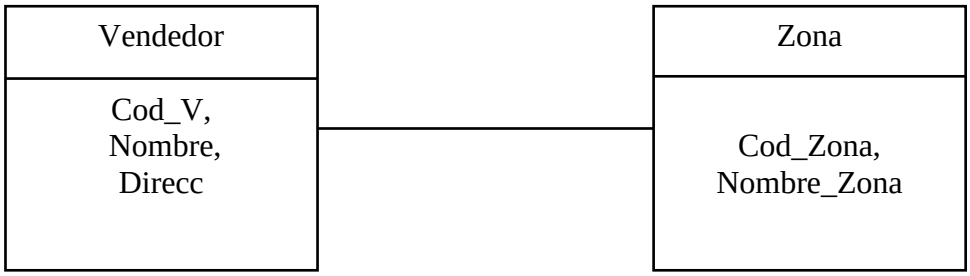


Fig 39 Asociación uno a uno

*** Conversión uno a muchos**

Cuando existe una relación uno a muchos en el MER Fig 40, la transformación al MOO genera dos clases, una clase para cada entidad Fig 41, tomando como identificador de las clases, las correspondientes claves de sus entidades así las dos clases creadas deben tener sus propios métodos.

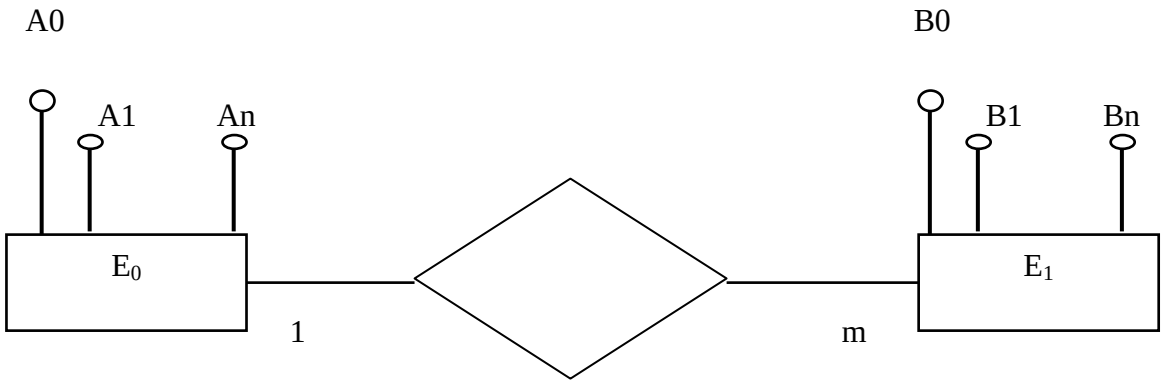


Fig 40 Relación uno a muchos.

$E_0 = (A_0:D_0,A_1:D_1,…..An:Dn)$

¥Ai:Di Є E_0

$A_0 = Cp$

$E_{1=}(B_0:D_0,B_1:D_1,…..Bn:Dn)$

¥ Bi:Di Є E_1

$B_0 = Cp$

En donde al transformar al MOO tenemos:

$C_0= E_0$ U Mi

$C_0= (A_0:D_0,A_1:D_1,…..An:Dn)$ U $(M_0,M_1,.....Mn)\}$

$C_1= E_1$ U Mi U A_0

$C1=\{ (B_0:D_0,B_1:D_1,…..Bn:Dn)$ U $(M_0,M_1,.....Mn)\}$

Donde:

A_0 = Id Є C_0

B_0 = Id Є C_1

Id = Identificador

Fig 41 Notación de clase uno a muchos

A continuación realizaremos un ejemplo para tener más en claro esta regla. Tenemos una relación uno a muchos entre un Administrador administra muchos empleados Fig 42.

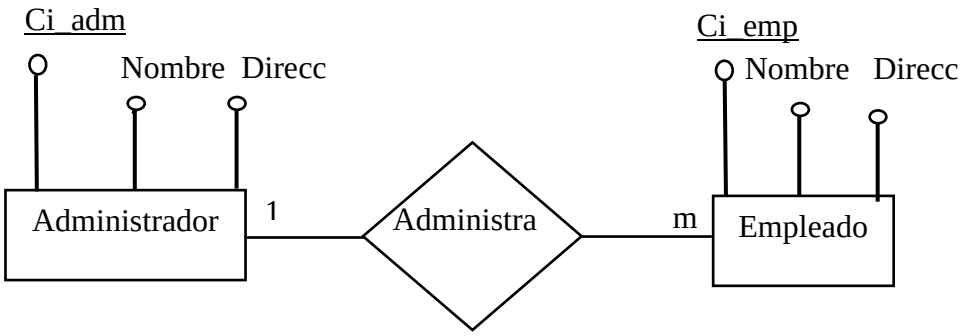


Fig 42 Relación uno a muchos

Al transforma esta relación uno a muchos al Modelo Orientado a Objetos tendremos que los atributos de las entidades se transforman como atributos de las clase. Fig 43.

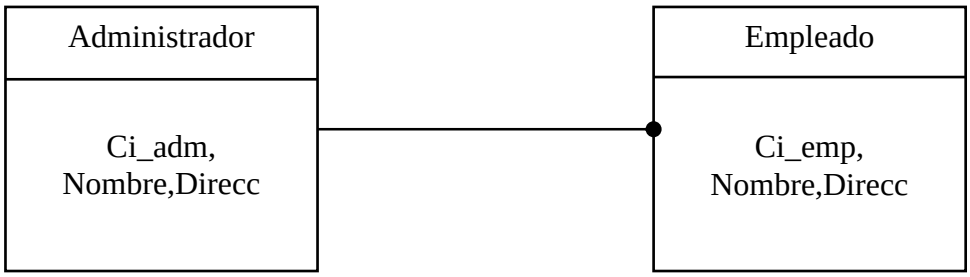


Fig 43 Asociación uno a muchos

*** Conversión muchos a muchos**

Cuando existe una relación muchos a muchos en el MER Fig 44, la transformación al MOO genera dos clases, una clase por cada entidad, cada clase se crea con sus propios atributos y sus métodos Fig 45.

Cuando existen objetos de la misma clase en asociaciones son necesarios los nombres de rol pero no indispensables.

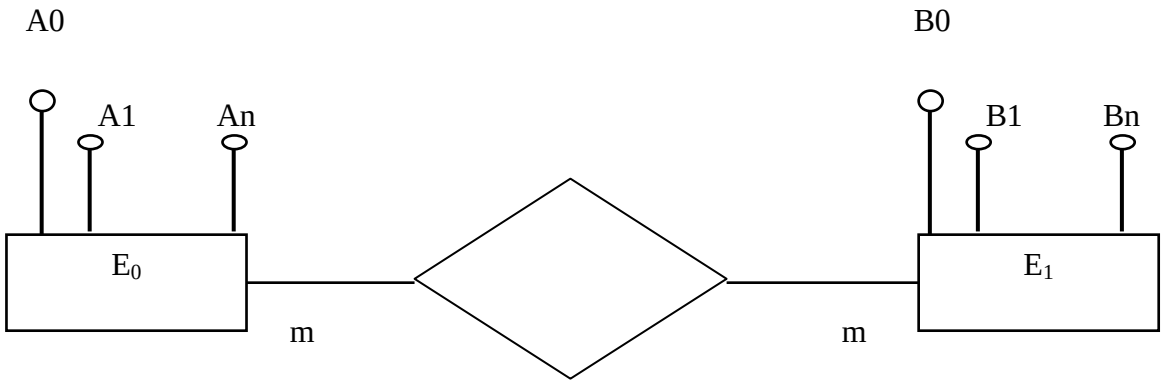


Fig 44 Relación muchos a muchos.

$E_0 = (A_0:D_0, A_1:D_1, \ldots..An:Dn)$

¥Ai:Di Є E_0

$E_1 = (B_0:D_0, B_1:D_1, \ldots..Bn:Dn)$

¥ Bi:Di Є E_1

Donde:

A_0 = Cp Є E_0

B_0 = Cp Є E_1

En donde al transformar al MOO tenemos:

$C_0 = E_0 \cup Mi$

$C_0 = \{(A_0:D_0, A_1:D_1, \ldots..An:Dn) \cup (M_0, M_1, ...Mn)\}$

$C_1 = E_1 \cup Mi$

$C_1 = \{(B_0:D_0, B_1:D_1, \ldots..Bn:Dn) \cup (M_0, M_1, ...Mn)\}$

Donde:

$A_0 = Id \in C_0$

Id = Identificador de C_0

$B_0 = Id \in C_1$

Id = Identificador de C_1

Fig 45 Clases muchos a muchos.

Para comprender mejor esta regla a continuación realizaremos un ejemplo.

Tenemos una relación muchos a muchos entre muchos profesores dan clases a muchos alumnos Fig 46.

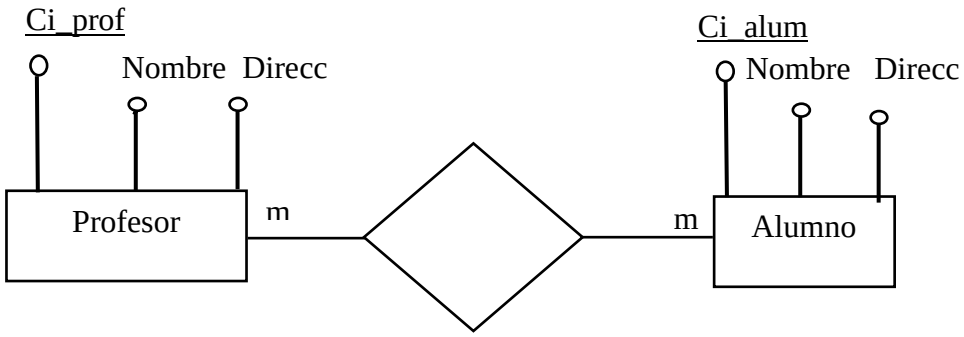


Fig 46 Relación muchos a muchos.

Al transforma esta relación muchos a muchos al Modelo Orientado a Objetos tendremos que los atributos de las entidades se transforman como atributos de las clase, la cual crea dos clases. Fig 47.

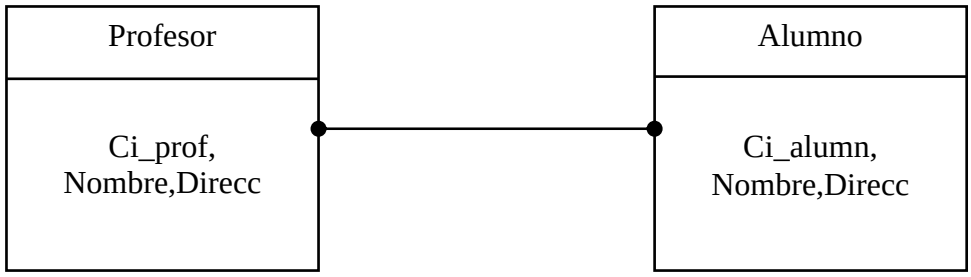


Fig 47 Asociación muchos a muchos

*** Conversión de Atributos de Relación**

Cuando una relación en el MER tiene propiedades propias Fig 48 en el MOO se representa como una clase unida a la línea de la asociación por medio de una línea a trazos Fig 49. Tanto la línea como el rectángulo de clase representan el mismo elemento

conceptual: la asociación. Por tanto ambos tienen el mismo nombre, el de la asociación. Cuando la clase asociación sólo tiene atributos el nombre suele ponerse sobre la línea.

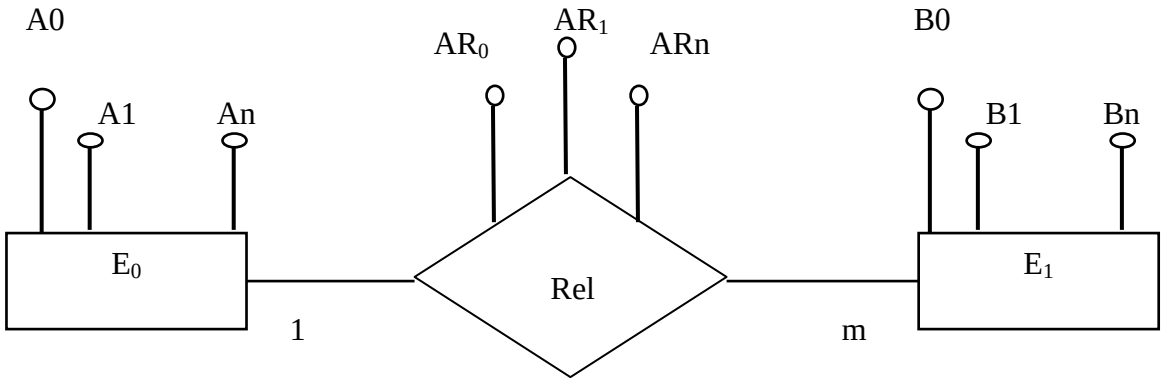


Fig 48 Atributos de relación

$E_0 = (A_0:D_0, A_1:D_1, \ldots..An:Dn)$

¥Ai:Di Є E_0

$E_1 = (B_0:D_0, B_1:D_1, \ldots..Bn:Dn)$

¥ Bi:Di Є E_1

Donde:

A_0 = Cp Є E_0

B_0 = Cp Є E_1

En donde al transformar al MOO tenemos:

$C_0 = E_0$ U Mi

$C_0 = \{(A_0:D_0, A_1:D_1, \ldots..An:Dn)$ U $(M_0, M_1, ...Mn)\}$

$C_1 = E_1$ U Mi

C_1={(B_0:D_0,B_1:D_1,…..Bn:Dn) U (M_0,M_1,.....Mn)}

C_2= (AR_0:D_0, AR_1:D_1,…..ARn:Dn) U Mi

Donde:

AR = Atributos propios de R

M_0,M_1,…..Mn Є C_2

A_0 = Id Є E_0

Id = Identificador de C_0

B_0 = Id Є E_1

Id = Identificador de C_1

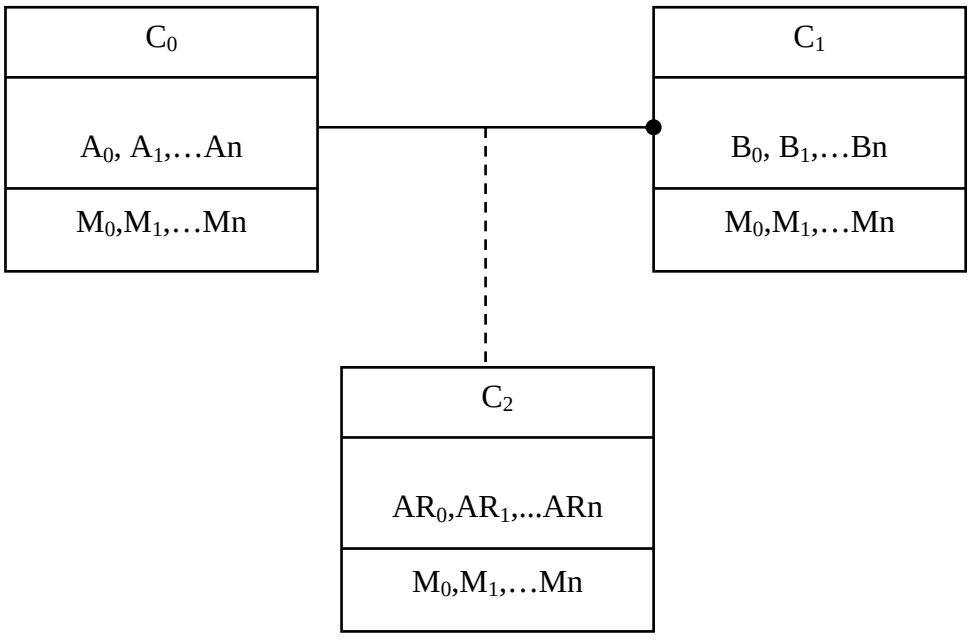


Fig 49 Clase asociación

A continuación veremos unos ejemplos para ver la forma más recomendable de modelar los atributos de enlace, primeramente tenemos una relación uno a muchos en el MER Fig 50, luego al hacer la transformación al modelo objetos se muestra la forma en que es posible plegar los atributos de enlace para asociaciones uno a uno y uno a muchos en la clase que está al otro lado del "uno" Fig 51, pero esto es la forma menos recomendable. En general, los atributos de enlace recomendamos que no deberían plegarse en una clase porque la futura flexibilidad se ve reducida si llega a cambiar la multiplicidad de la asociación la forma más recomendable se lo muestra en la Fig 52.

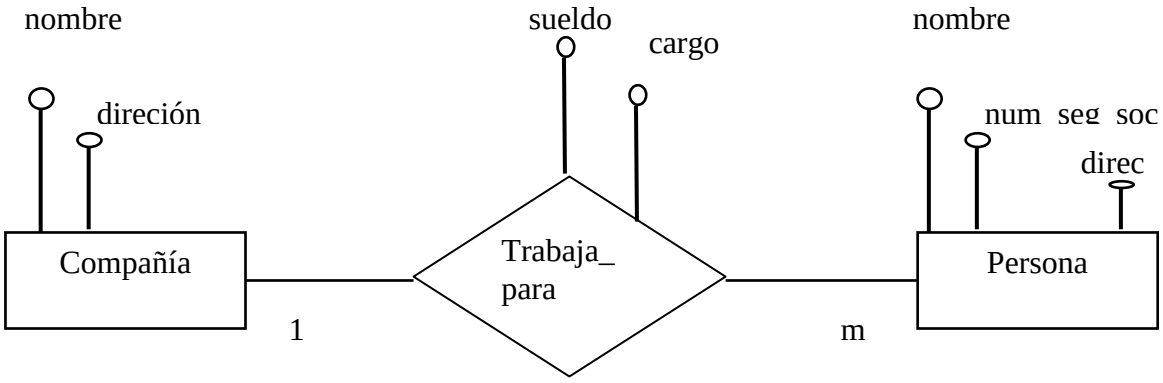


Fig 50 Atributos de relación

Al realizar la transformación al modelo Orientado a Objetos tenemos dos formas de representación:

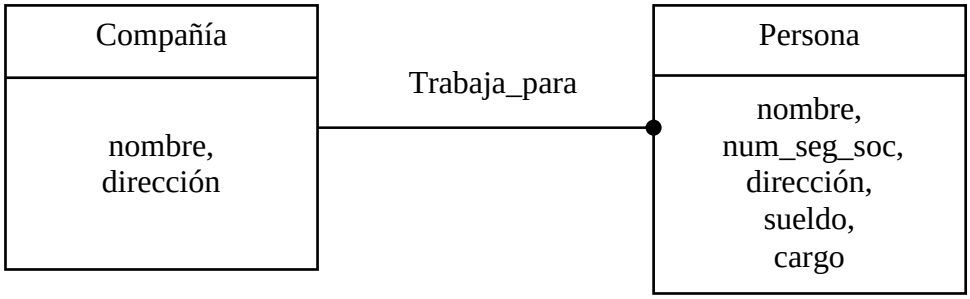


Fig 51 Forma menos recomendable

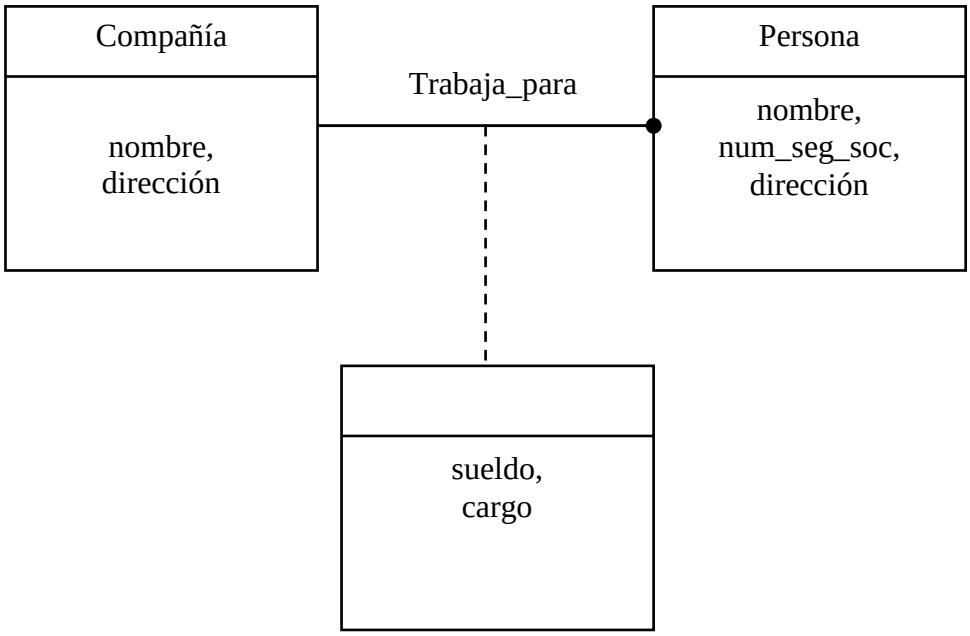


Fig 52 Forma recomendable de modelar los atributos de enlaces

Restricción.

No es posible plegar los atributos de enlace en una clase de la asociación muchos a muchos, para este caso solo se debe crear la clase asociación, esto se explica con el siguiente ejemplo Figuras 53, 54 y 55.

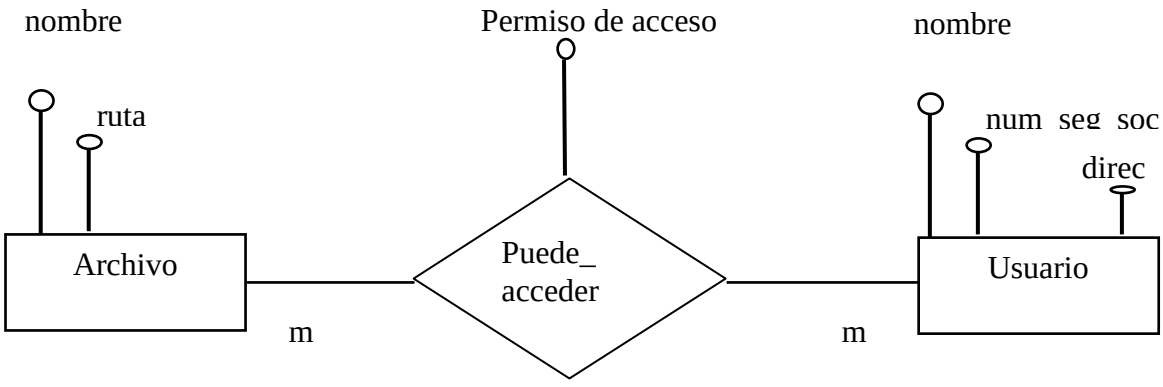


Fig 53 Relación muchos a muchos entre archivo y usuario.

La Fig 54 muestra la forma en que no es posible plegar los atributos de enlace en asociaciones muchos a muchos por lo contrario la Fig 55 muestra la forma correcta de modelar atributos de enlaces.

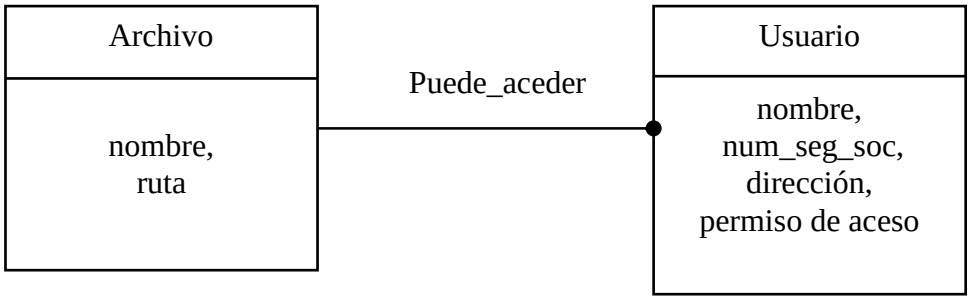


Fig 54 Forma incorrecta de modelar los atributos de enlaces en asociaciones muchos a muchos

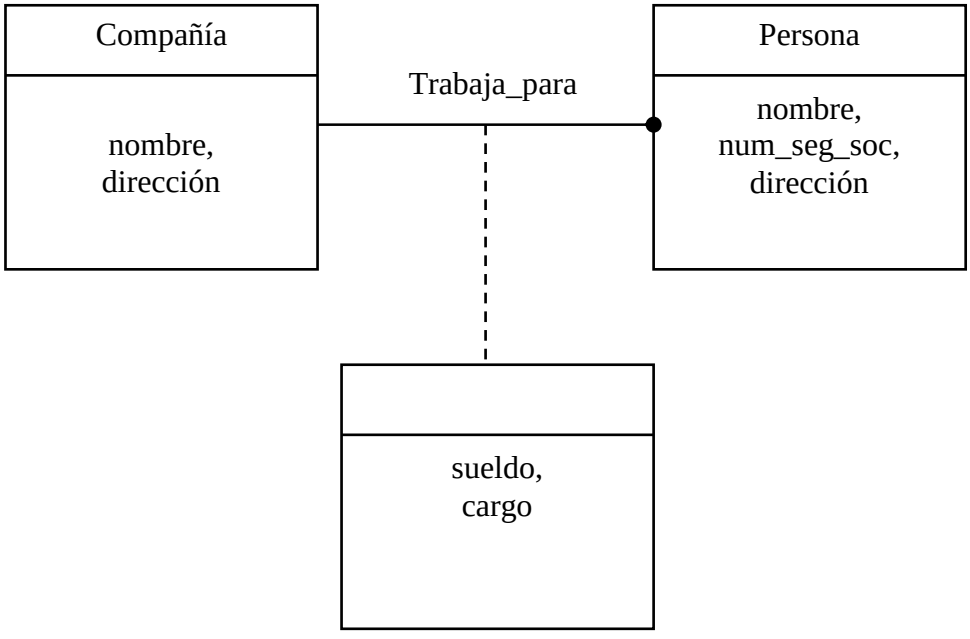


Fig 55 Forma correcta de modelar los atributos de enlaces en asociaciones muchos a muchos

*** Conversión de Generalización/ Especialización**

Cuando existe una generalización en el MER con una o más entidades dentro de la generalización como en la Fig 56; al realizar la transformación al MOO se crea la herencia compuesta por una superclase con sus atributos propios y se crean varias subclases dependiendo del número de entidades con sus atributos propios; estas subclases heredan los atributos de la superclase así como en la Fig 57.

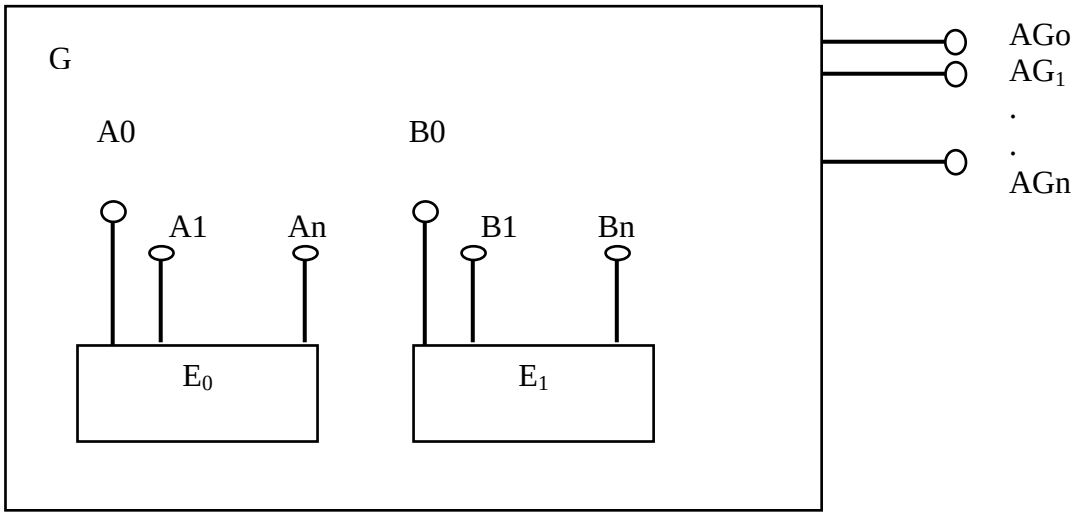


Fig 56 Representación de entidad Generalizada

$G= (AG_0:D_0,AG_1:D_1,…..AGnDn)$

Especialización

$E_0 = (AG_0:D_0,A_0:D_0,A_1:D_1,…..An:Dn)$

Donde:

$AG_0= Cp \in G$

$E_1 = (AG_0{:}D_{0,}B_0{:}D_0,B_1{:}D_1,\ldots..Bn{:}Dn)$

Donde:

AG_0= Cp Є G

En donde al transformar al MOO tenemos:

CG= G U Mi

$CG=\{(AG_0{:}D_0,AG_1{:}D_1,\ldots..AGn{:}Dn)\ U\ (M_0,M_1,.....Mn)\}$

Donde CG = Superclase o clase generalizada

C_0= E_0 U Mi U AG_0

$C_0=\{(A_0{:}D_0,A_1{:}D_1,\ldots..An{:}Dn)\ U\ (M_0,M_1,.....Mn)\ U\ AG_0\ \}$

Donde:

AG_0 = Identificador de G

$M_0,M_1,\ldots..Mn$ Є C_1

C_1= E_1 U Mi U AG_0

$C_1=\{(B_0{:}D_0,B_1{:}D_1,\ldots..BAn{:}Dn)\ U\ (M_0,M_1,.....Mn)\ U\ AG_0\}$

Donde:

AG_0 = Identificador de G

$M_0,M_1,\ldots..Mn$ Є C_2

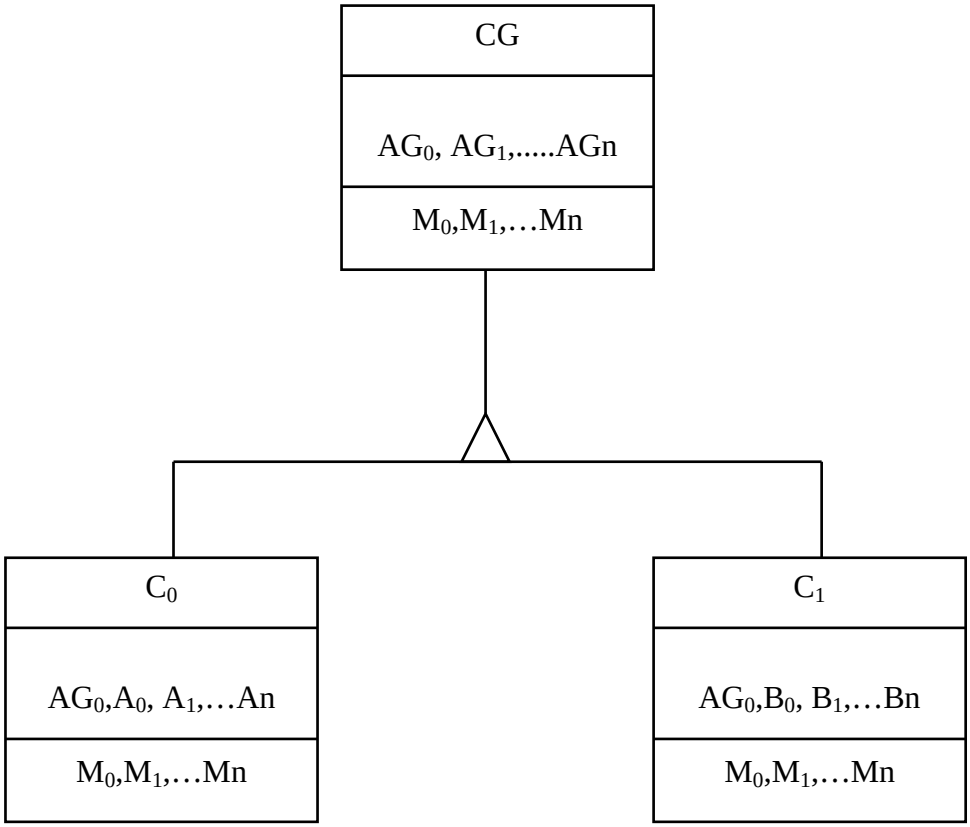


Fig 57 Representación de Herencia

*** Conversión de Agregación**

En el caso de la agregación en el MER con una entidades y relaciones dentro de la agregación como en la Fig 58; al realizar la transformación al MOO se crea la clase agregada compuesta por sus atributos, sus métodos y las claves primarias de las entidades de la relación, las elaciones se los realiza según el caso que sea debiendo utilizar las reglas descritas anteriormente, la clase agregada esta formada por componentes, los componentes forman parte del agregado, y están enlazados por un rombo que indica el extremo de ensamblaje de la relación como se muestra en la Fig 59.

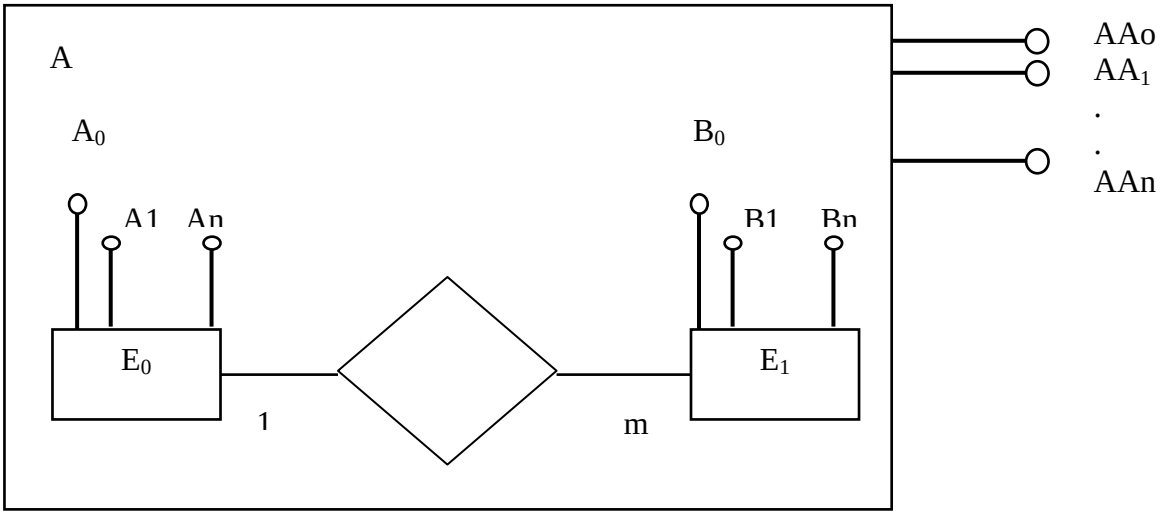


Fig 58 Representación de Agregación

A={ (AA_0:D_0,AA_1:D_1,…..AAnDn) U A_0 U B_0}

Donde:

A = Agregación

AAi = Atributos propios de agregación

A_0 = Cp Є E_0

B_0 = Cp Є E_1

E_0 = (A_0:D_0,A_1:D_1,…..An:Dn)

¥Ai:Di Є E_0

A_0 = Cp

E_1=(B_0:D_0,B_1:D_1,…..Bn:Dn)

¥ Bi:Di Є E_1

B_0 = Cp Є E_1

En donde al transformar al MOO tenemos:

CA= A U Mi

$CA=\{(AA_0:D_0,AA_1:D_1,\ldots..AAnDn,\ A_0:D_0\ U\ B_0:D_0\ )\ U\ (M_0,M_1,.....Mn)\}$

Donde CA = Clase agregada

$A_0 = Cp\ \epsilon\ C_0\ B_0 = Cp\ \epsilon\ C_0$

$C_0= E_0\ U\ Mi$

$C_0= (A_0:D_0,A_1:D_1,\ldots..An:Dn)\ U\ (M_0,M_1,.....Mn)\}$

$C_1= E_1\ U\ Mi\ U\ A_0$

$C1=\{\ (B_0:D_0,B_1:D_1,\ldots..Bn:Dn)\ U\ (A_0:D_0)\ U\ (M_0,M_1,.....Mn)\}$

Donde:

$A_0 = Id\ \epsilon\ E_0$

Id = Identificador de E_0

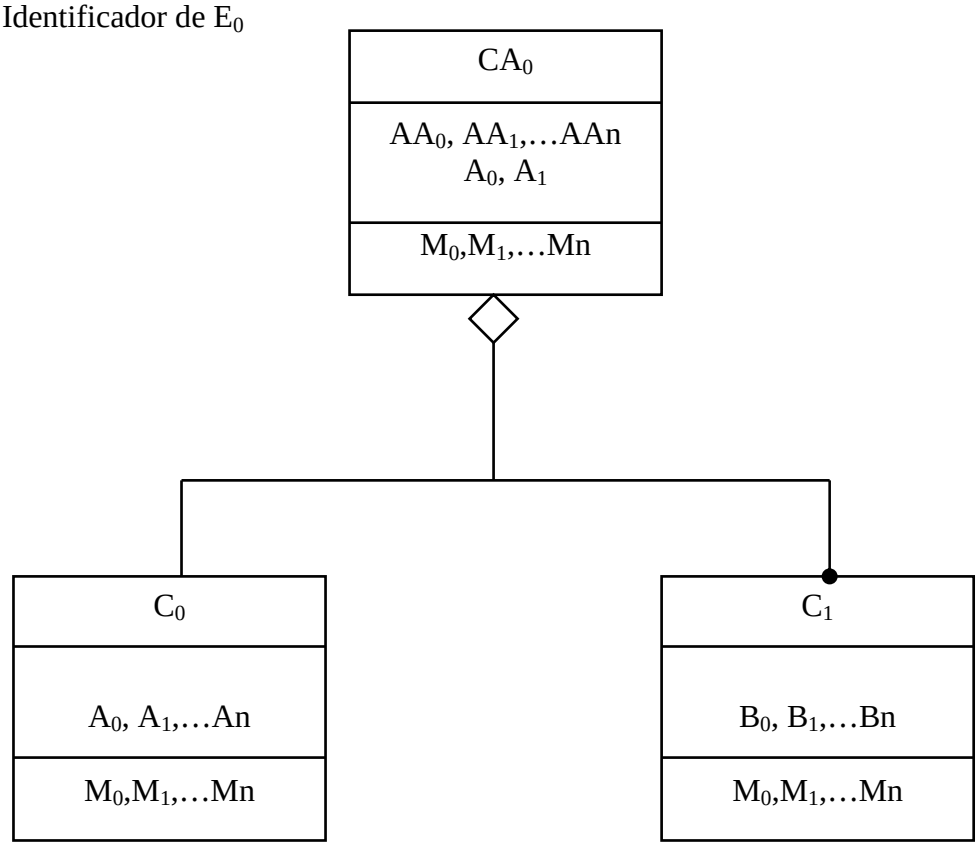


Fig 59 Representación de Agregación en el MOO

*** Conversión de entidad débil y agrupación dominante.**

Para la conversión de un entidad débil Fig 60, creamos un tipo complejo como una clase agregación de la entidad fuerte/dominante y las entidades débiles/relacionadas. La clase fuerte tiene sus propios atributos y sus métodos, en tanto que la clase débil tendrá sus atributos más la clave de la entidad dominante como identificador de la clase mas sus propios métodos como se indica en la Fig 61.

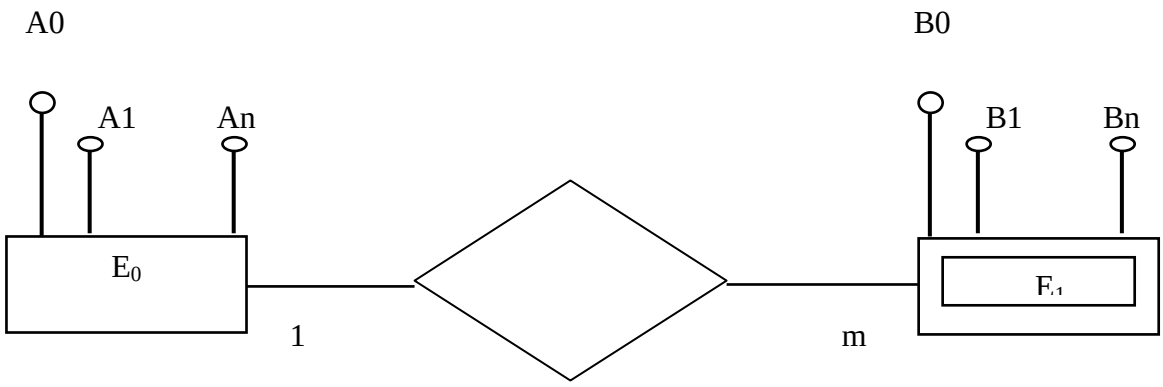


Fig 60 Relación con entidad débil.

$E_0 = (A_0:D_0, A_1:D_1, \ldots..An:Dn)$

¥Ai:Di Є E_0

$A_0 = Cp$

$E_{1=}(A_0:D_0, B_0:D_0, B_1:D_1, \ldots..Bn:Dn)$

¥ Bi:Di Є E_1

Donde:

A_0 = Cp Є E_0

En donde al transformar al MOO tenemos:

C_0= E_0 U Mi

C_0= (A_0:D_0,A_1:D_1,…..An:Dn) U (M_0,M_1,.....Mn)}

C_1= E_1 U Mi U A_0

C1={ (B_0:D_0,B_1:D_1,…..Bn:Dn) U (A_0:D_0) U (M_0,M_1,.....Mn)}

Donde:

A_0 = Cp Є E_0

Cp = identificador de E_0 y E_1

Fig 61 Asociación con clase débil.

*** Relaciones de grado mayor con cardinalidad 1 a 1**

En las relaciones que tengan un grado mayor de 2 son bastante más difíciles de manejar que las binarias al transformarles al Modelo Orientado a Objetos.

Casi nunca se utilizan relaciones de grado igual o superior a 4, pero con la regla de transformación que le daremos le será muy fácil aplicar la transformación al Modelo Orientado a Objetos.

Al tener una relación con 3 entidades como se muestra en la Fig. 62 la transformación al MOO genera cuatro clases, una clase por cada entidad y adicional se crea otra clase de la relación entre las tres entidades formadas por las claves primarias de cada entidad y sus atributos de relación si los tiene, cada clase a excepción de la cuarta se crea con sus propios atributos y sus métodos; tomando como clave primaria las correspondientes claves de sus entidades, las clases están asociadas con la clase creada en este caso la cuarta clase con cardinalidades 1 a mucho Fig. 63.

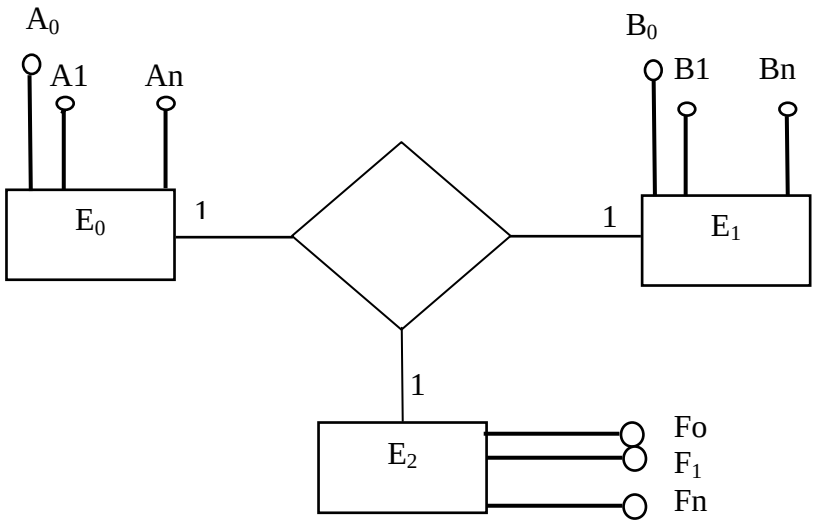


Fig 62 Relación Ternaria.

$E_0 = (A_0:D_0, A_1:D_1, \ldots..An:Dn)$

¥Ai:Di Є E_0

$E_1 = (B_0:D_0,B_1:D_1,…..Bn:Dn)$

¥ Bi:Di Є E_1

$E_2= (F_0:D_0,F_1:D_1,…..Fn:Dn)$

¥ Fi:Di Є E_1

Donde:

A,B,F = Atributos

En donde al transformar al MOO tenemos:

$C_0= E_0$ U Mi

$C_0=\{(A_0:D_0,A_1:D_1,…..An:Dn)$ U $(M_0,M_1,.....Mn)\}$

$C_1= E_1$ U Mi

$C_1=\{(B_0:D_0,B_1:D_1,…..Bn:Dn)$ U $(M_0,M_1,.....Mn)\}$

$C_1= E_2$ U Mi

$C_2=\{(F_0:D_0,F_1:D_1,…..Fn:Dn)$ U $(M_0,M_1,.....Mn)\}$

$C_3= A_0$ U B_0 U F_0 U Mi

$C_2=\{ (A_0:D_0$ U $B_0:D_0$ U $F_0:D_0)$ U $(M_0,M_1,.....Mn) \}$

Donde:

A_0 = Id Є C_0

Id = Identificador de C_0

B_0 = Id Є C_1

Id = Identificador de C_1

F_0 = Id Є C_2

Id = Identificador de C_2

$M_0,M_1,\ldots..Mn \in C_3$

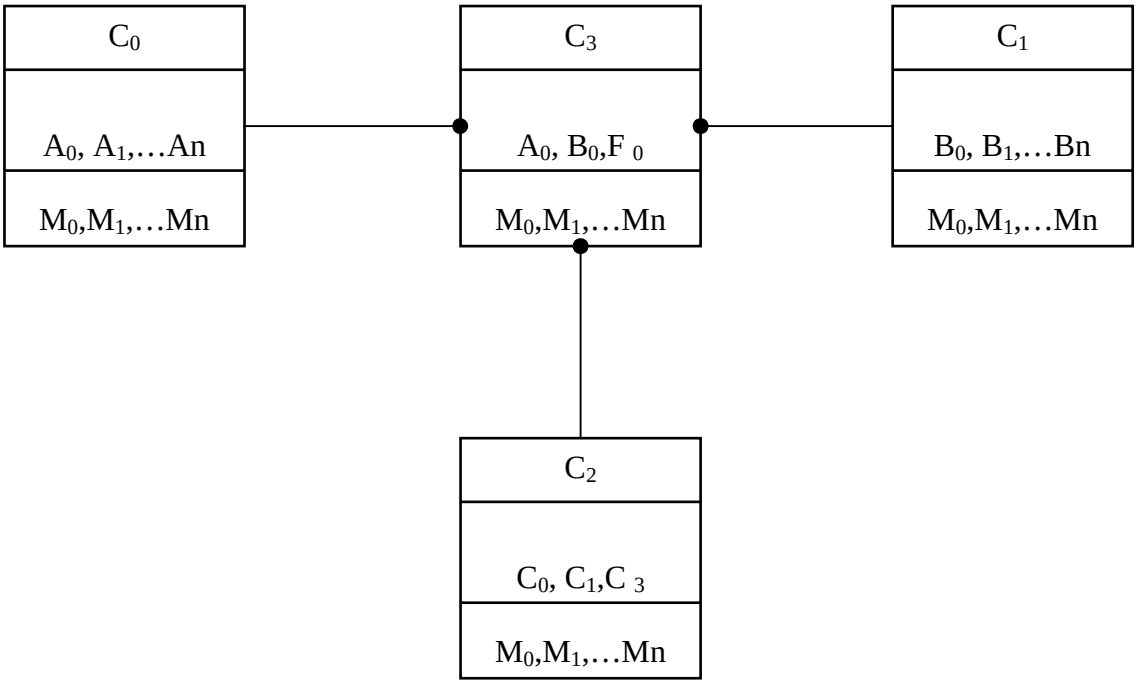


Fig 63 Clase transformada de Relación Ternaria.

Con este ejemplo daremos a entender mejor esta regla, visualizamos en las Fig 64 y Fig 65.

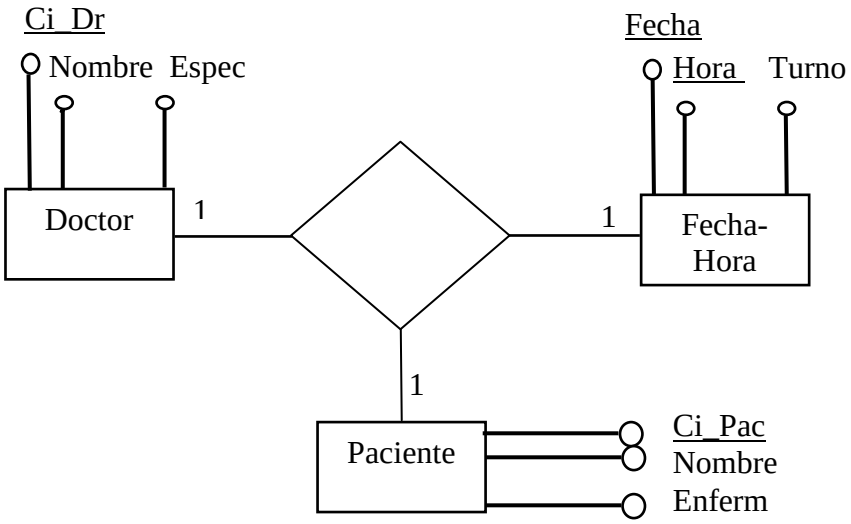


Fig 64 Relación ternaria de un doctor, un paciente y una fecha-hora

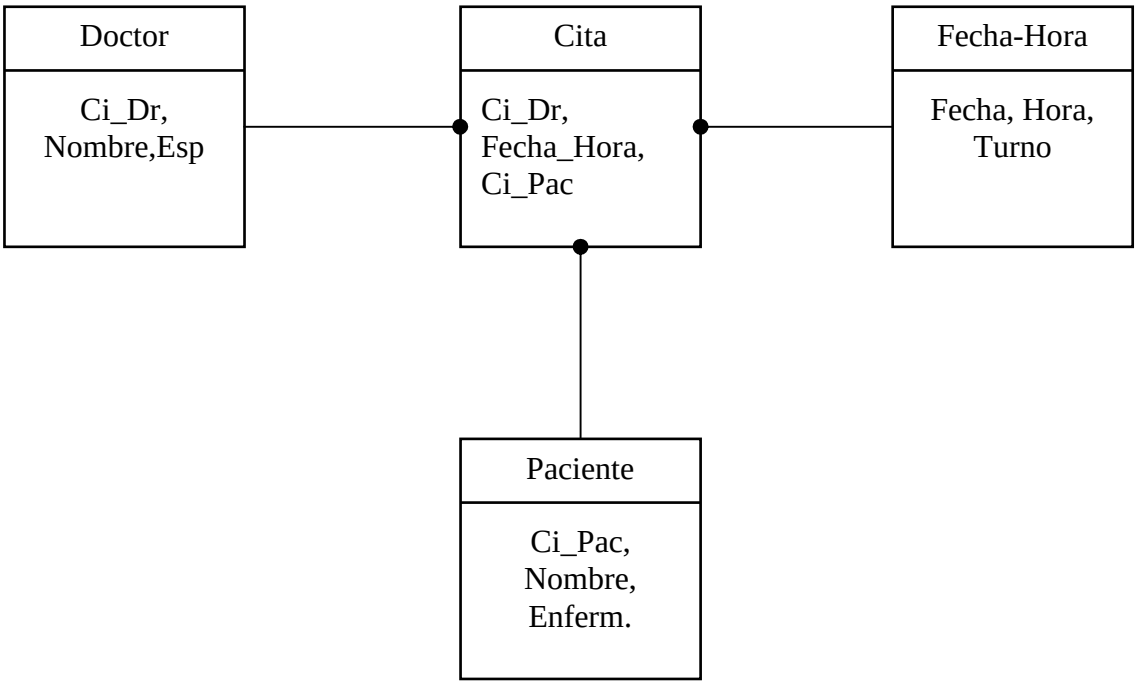


Fig 65 Clase transformada de Relación Ternaria.

*** Relaciones de grado mayor con distintas cardinalidades**

En las relaciones de grado mayor de 2 son bastante más difíciles de manejar que las binarias al transformarles al Modelo Orientado a Objetos.

Casi nunca se utilizan relaciones de grado igual o superior a 4.

Lo aconsejable es transformar una relación ternaria Fig 66 a varias binarias (lo mismo para n= 4, 5, ...) Fig 67. que recogen la misma semántica, luego para realizar las transformaciones al Modelo Orientado a Objetos se utilizarán las reglas antes explicadas según el caso que corresponda Fig 68.

La determinación de las cardinalidades de cada tipo de entidad participante debe realizarse con cuidado.

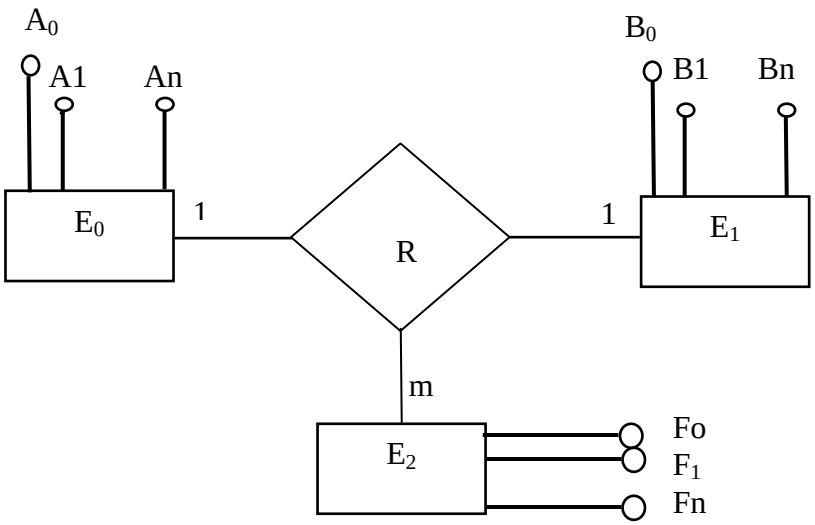


Fig 66 Relación Ternaria con diferentes cardinalidades .

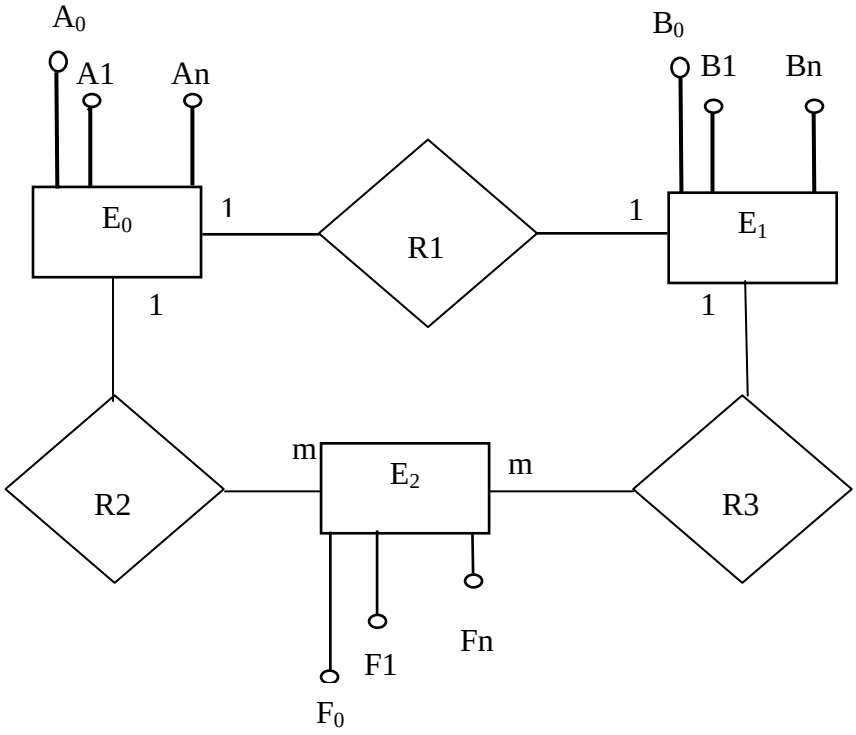


Fig 67 Transformación de Relación Ternaria a Binarias.

Teniendo relaciones binarias nos resulta más fácil la transformación a Objetos ahí ya podemos ocupar una de las reglas anteriores según sea el caso.

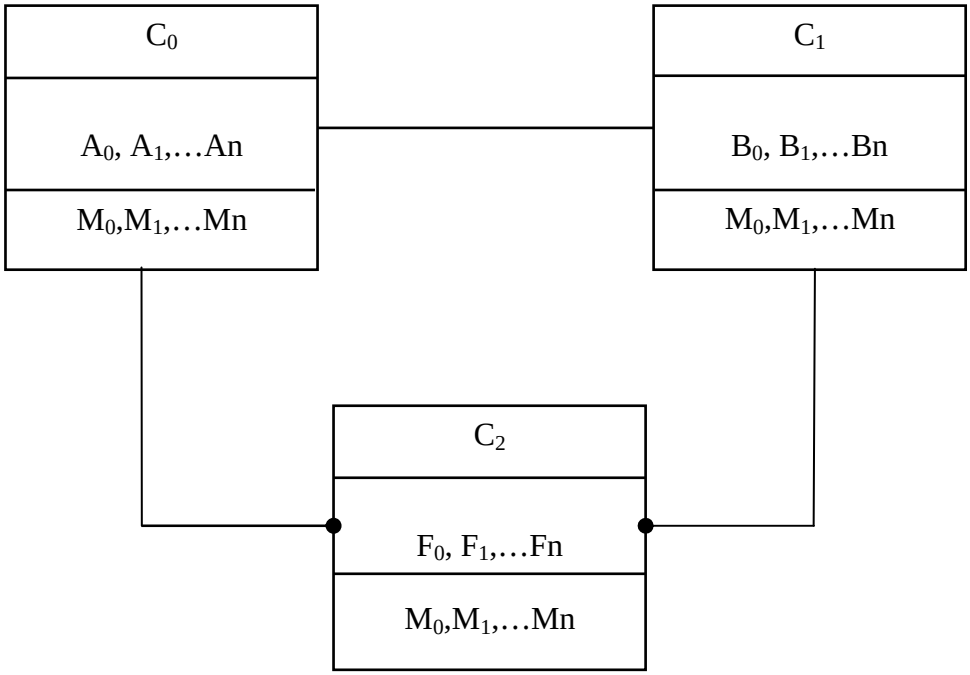


Fig 68 Transformación de Relación Ternaria a Clases.

Con el siguiente ejemplo daremos a entender mejor esta regla, visualizamos en las Fig 69, 70 y Fig 71.

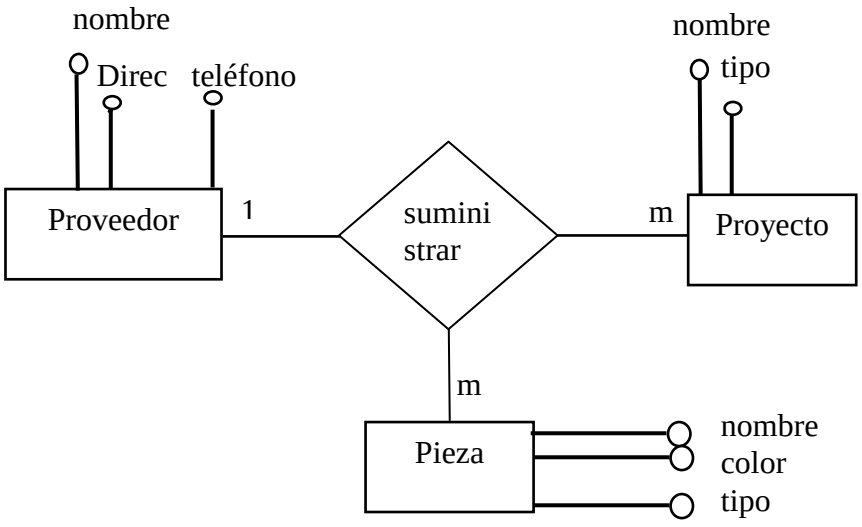


Fig 69 Relación Ternaria entre proveedor, proyecto y pieza.

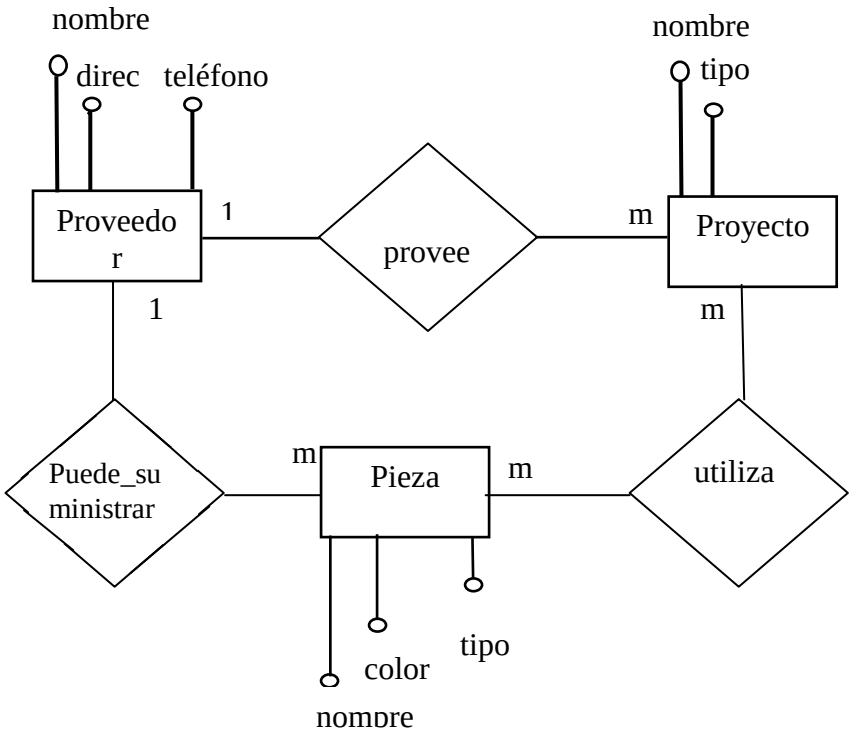


Fig 70 Transformación de Relación Ternaria a Binarias.

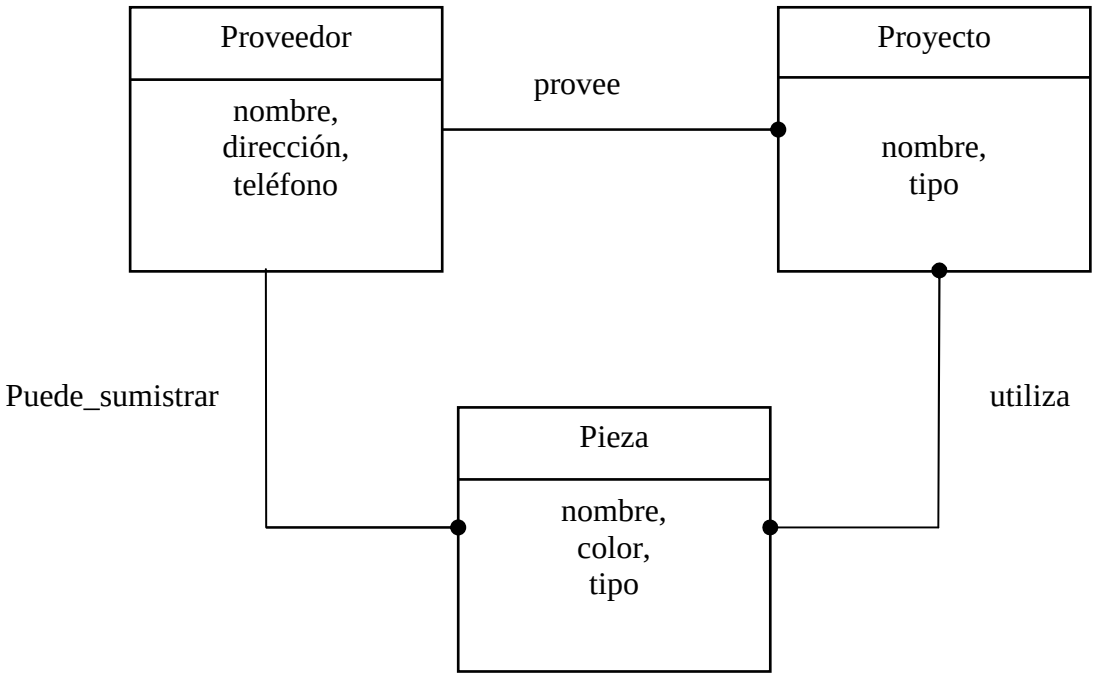


Fig 71 Clase de Relación Ternaria

*** Relaciones de grado mayor con cardinalidades muchos a muchos**

Cuando tenemos un relación ternaria con cardinalidad muchos a muchos Fig 72 , que casi nunca se utilizan estas relaciones, pero si es el caso nosotros recomendamos realizar una agregación entre dos entidades teniendo como clave primaria la unión de las dos claves de las entidades las cuales son escogidas para ir en la agregación Fig 73 y luego tener una relación muchos a muchos entre la agregación y la tercera entidad sobrante y en este paso podemos realizar la transformación orientadas a objetos según el caso correspondiente para tener una asociación entre la tercera clase y la agregación con cardinalidad muchos a muchos.

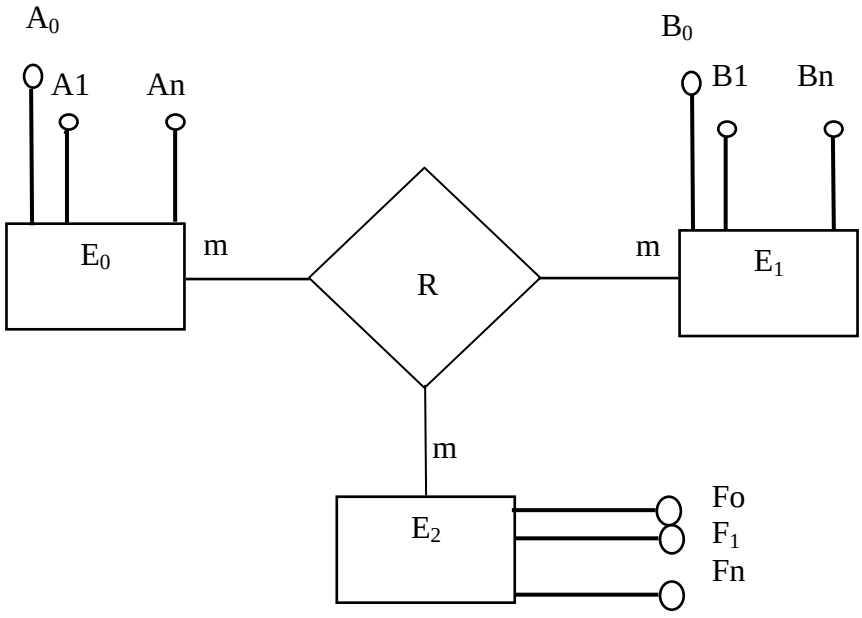


Fig 72 Relación Ternaria con cardinalidades muchos a muchos.

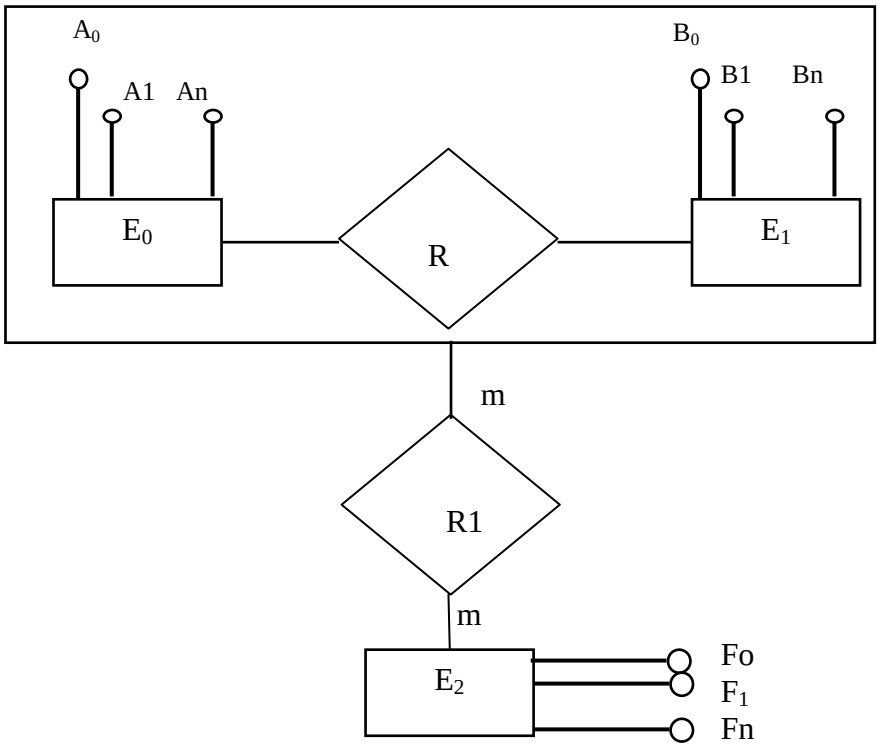


Fig 73 Relación entre entidad y agregación.

Al transformar al MOO tenemos la Fig 74

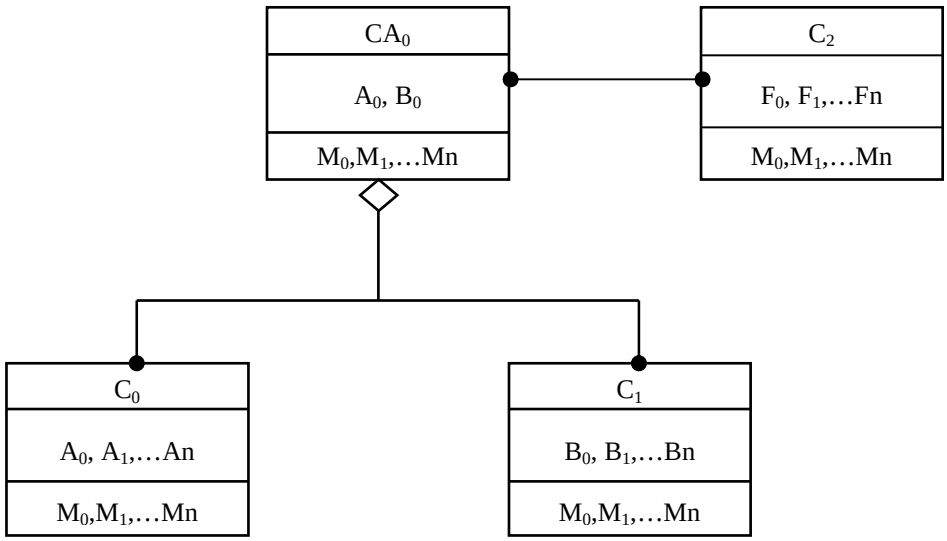


Fig 74 Asociación entre entidad y agregación.

5.5 Caso De Estudio

A continuación se realizará la transformación del caso de estudio de una base de datos analizada con el modelo entidad relación al MOO. Para esto partiremos del siguiente:

Cada persona tiene cédula de identidad, nombre, dirección.

Las personas pueden hacer cargos de tiempos a distintos proyectos y ganar un sueldo.

Las compañías tienen nombre, dirección número de teléfono y producto principal.

Las compañías contratan y despiden a personas.

Persona y compañía tienen una relación muchos a muchos el cargo depende tanto de la persona como de la compañía.

Hay dos tipos de personas trabajadores y administrativos.

Todo trabajador trabaja en muchos proyectos.

Cada administrador es responsable de muchos proyectos.

Los proyectos tienen en plantilla muchos trabajadores y exactamente un administrador.

Todo proyecto tiene nombre presupuesto y prioridad interna para conseguir recursos.

Una compañía consta de múltiples departamentos cada departamento de la compañía esta identificado de forma única por su nombre.

Los departamentos suelen tener un administrador, pero no siempre es así. La mayoría de los administradores dirige un departamento; hay unos pocos administradores que no están asociados a ningún departamento.

Cada departamento fabrica muchos productos; mientras que cada producto es fabricado por cada departamento únicamente. Cada producto tiene un nombre, un coste y un peso.

5.5.1 Desarrollo Del Diagrama Entidad Relación

A continuación tenemos el diagrama en el MER Fig 75

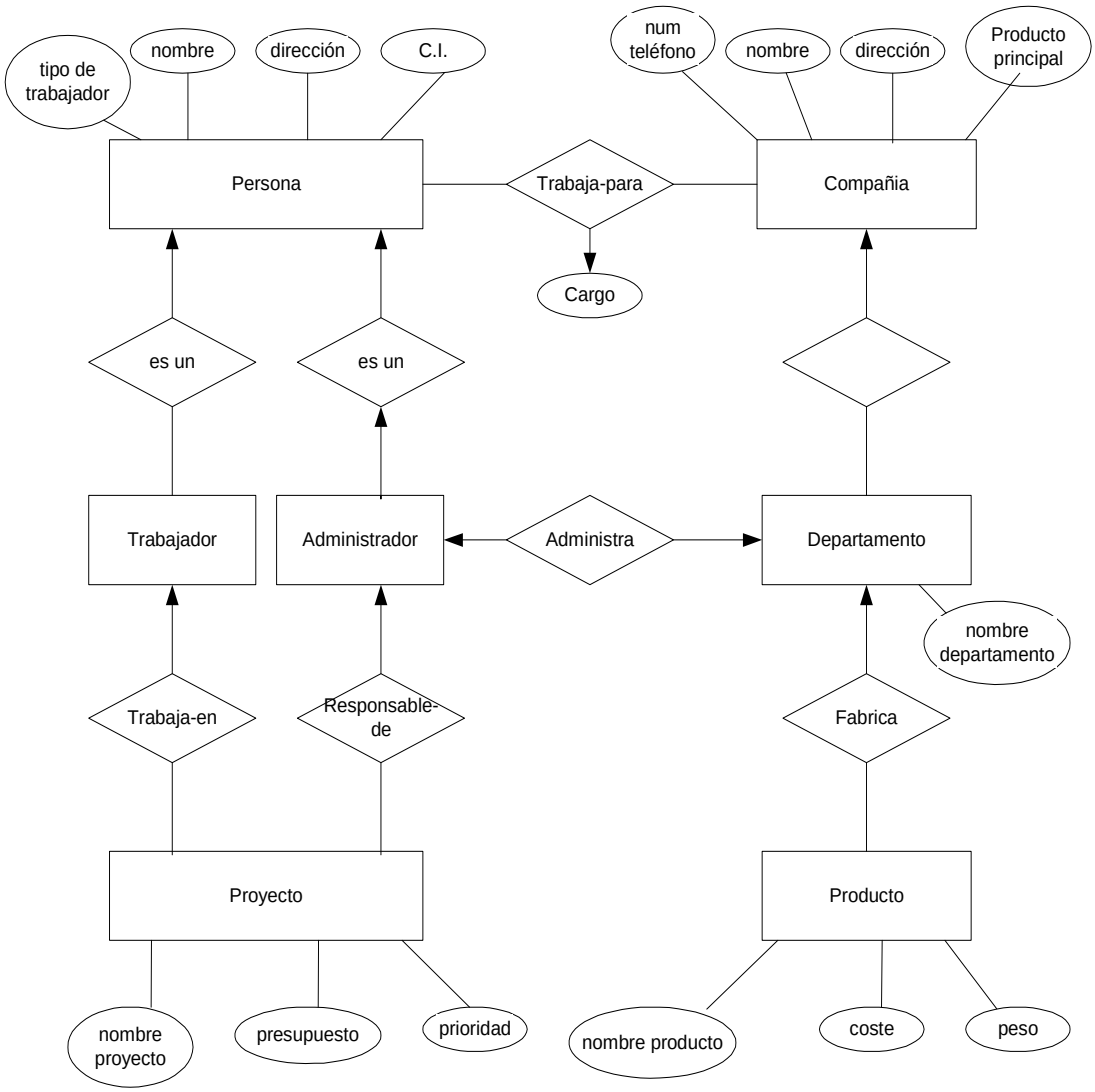


Fig. 75 Representación de una BD en el Modelo Entidad Relación

En primer lugar siguiendo la metodología realizamos la traducción estable que convierte cada entidad y cada vínculo en un objeto se demuestra en las Figuras 76 y 77.

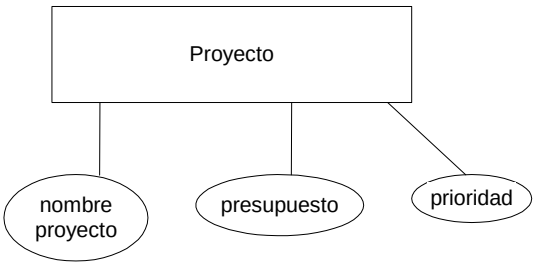


Fig 76 Representación de una Entidad en el MER

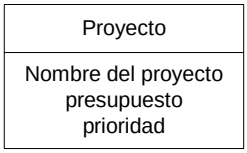


Fig 77 Representación de Clase en el MOO

Cuando hay una relación de dos entidades muchos a muchos Fig 78 se crea una clase de asociación con los atributos de la relación Fig 79.

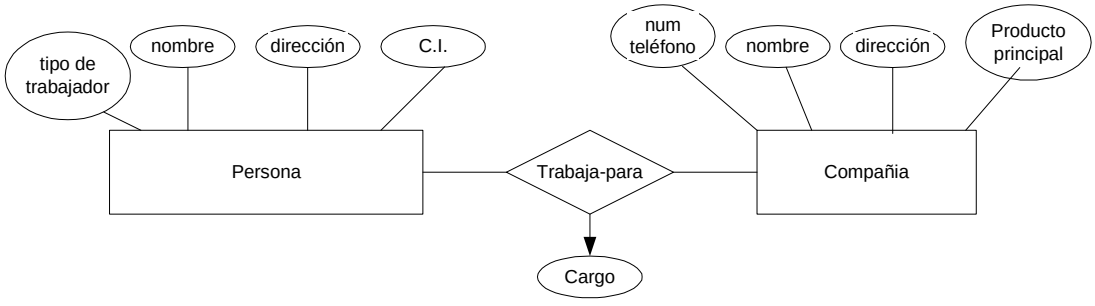


Fig 78 Representación de una Relación muchos a muchos en el MER

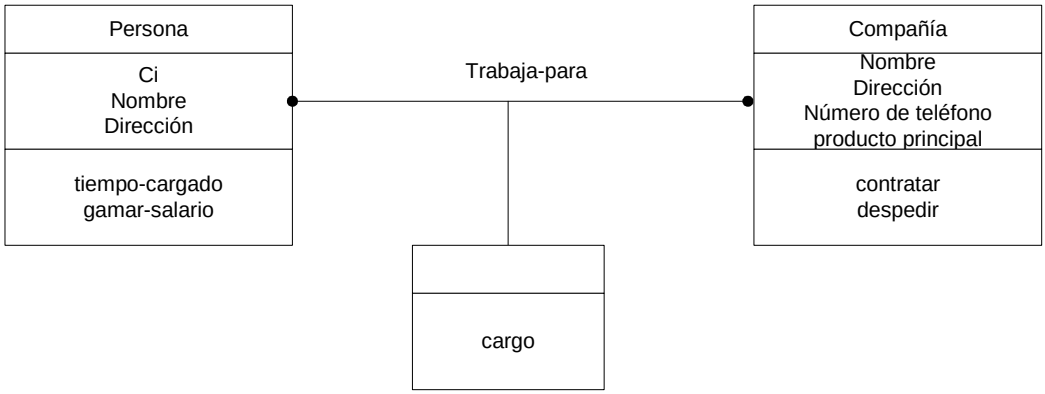


Fig 79 Representación de una Clase asociación en el MOO

Cuando existen objetos de la misma clase en asociaciones Fig 80 son necesarios los nombres de rol pero no indispensables Fig 81.

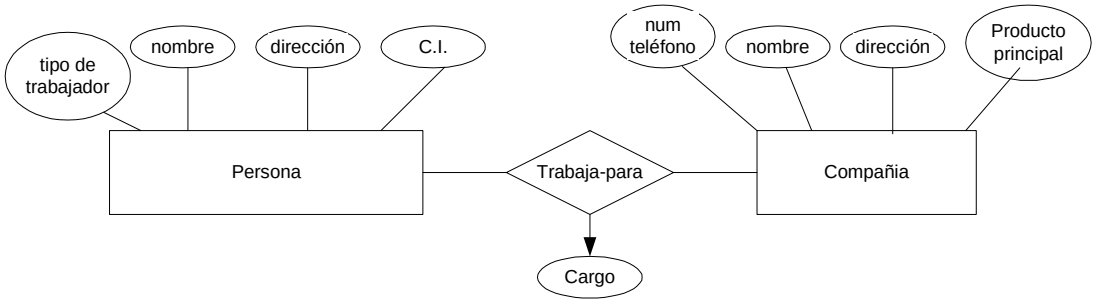


Fig 80 Representación de Entidades y Relación en el MER

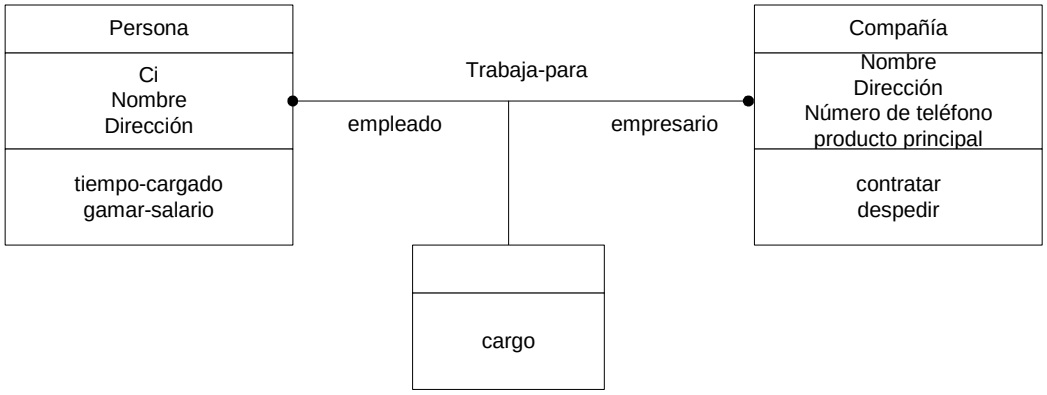


Fig 81 Representación de Esquema de roles en el MOO

Cuando existe una relación sin nombre o no existe la relación de dos entidades muchos a muchos en el MER esto crea una asociación cualificada en el MOO así Fig 82 y 83 .

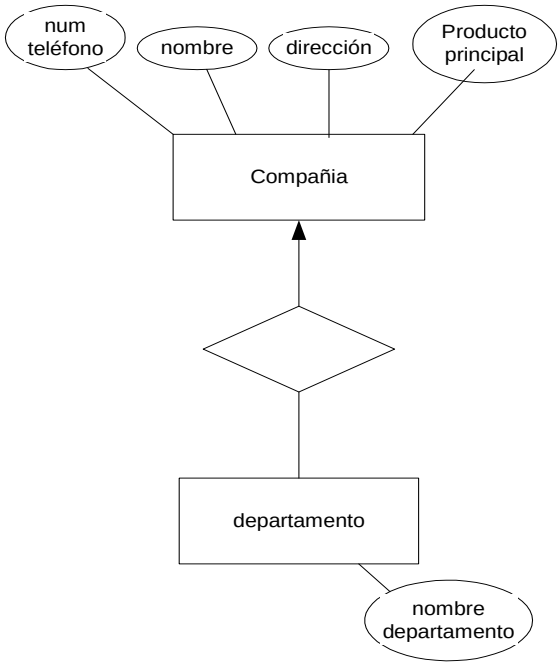


Fig 82 Representación de Relación sin nombre ni atributos en el MER

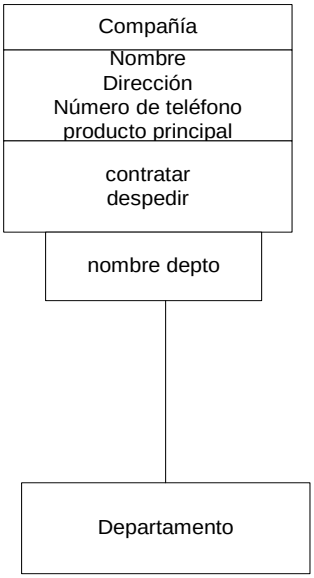


Fig 83 Representación de Asociación cualificada en el MOO

Todos los enlaces de cada asociación deben conectar a objetos procedentes de la misma clase. Una entidad generalizada y sus entidades especializadas puede ser percibida como una traducción de vínculos 1:1 entre la entidad generalizada y cada uno de sus subtipos como demuestran las figuras 84 y 85.

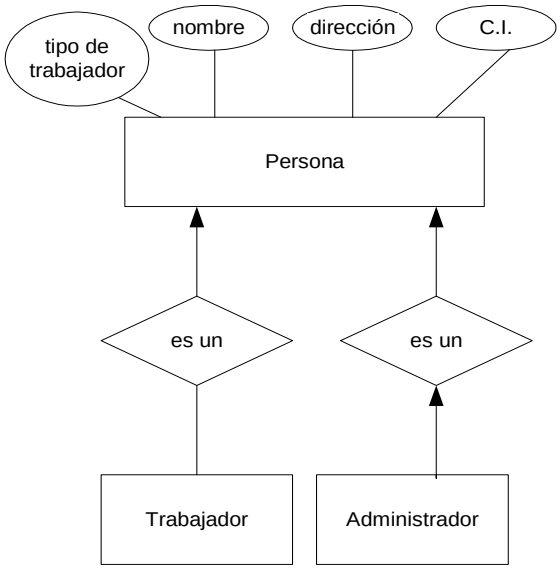


Fig 84 Representación de entidades de mismo tipo en el MER

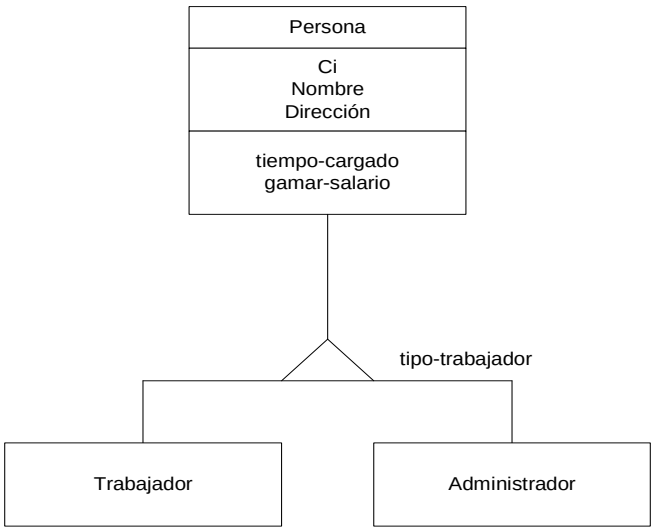


Fig 85 Representación de Herencia y discriminador de clases en el MOO.

5.5.2 Diagrama Orientado a Objetos

Realizando el seguimiento de estas reglas llegamos al siguiente esquema final Fig 86.

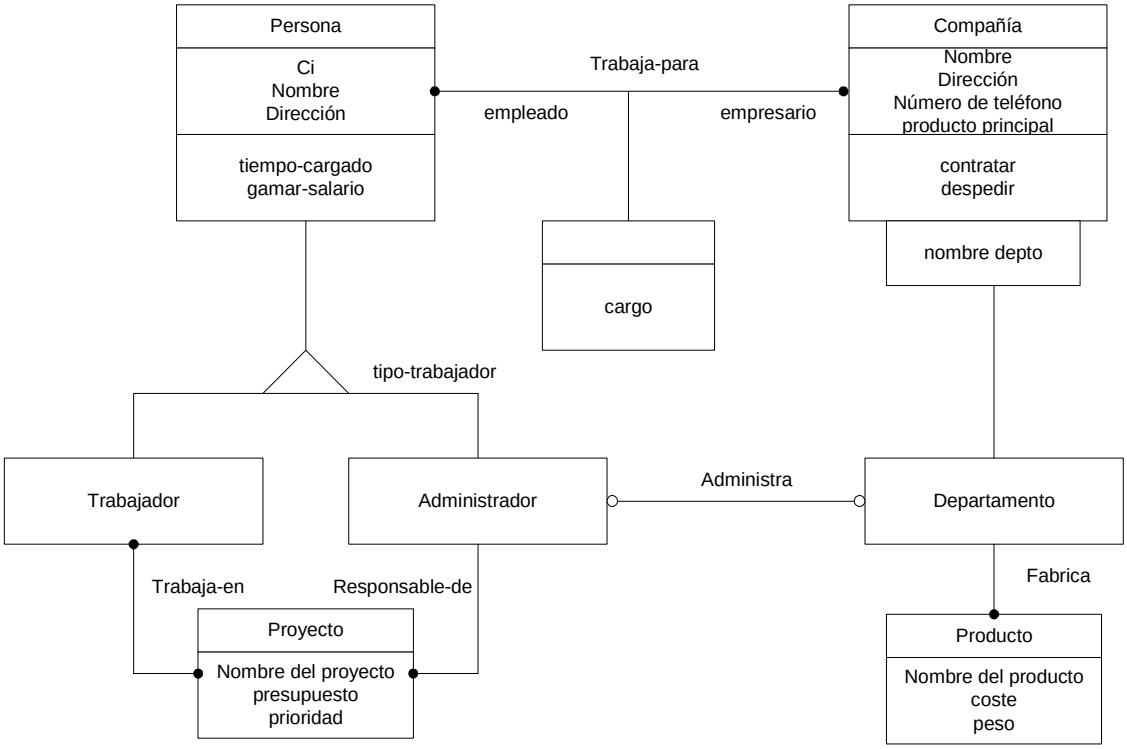


Fig 86 Esquema final de la conversión al MOO

5.5.3 Diagrama de Flujo de Datos

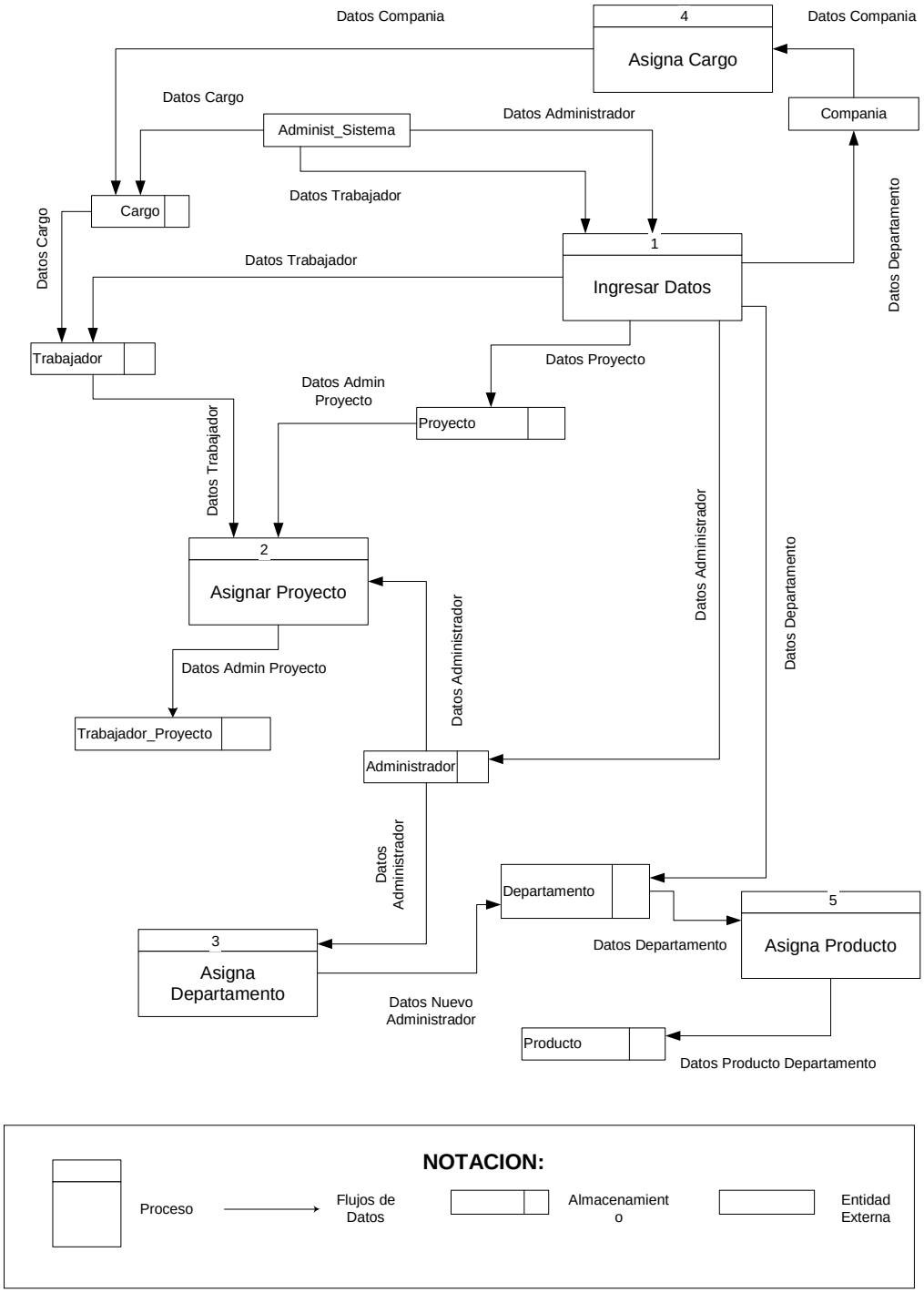


Fig 87 Diagrama de Flujo de Datos

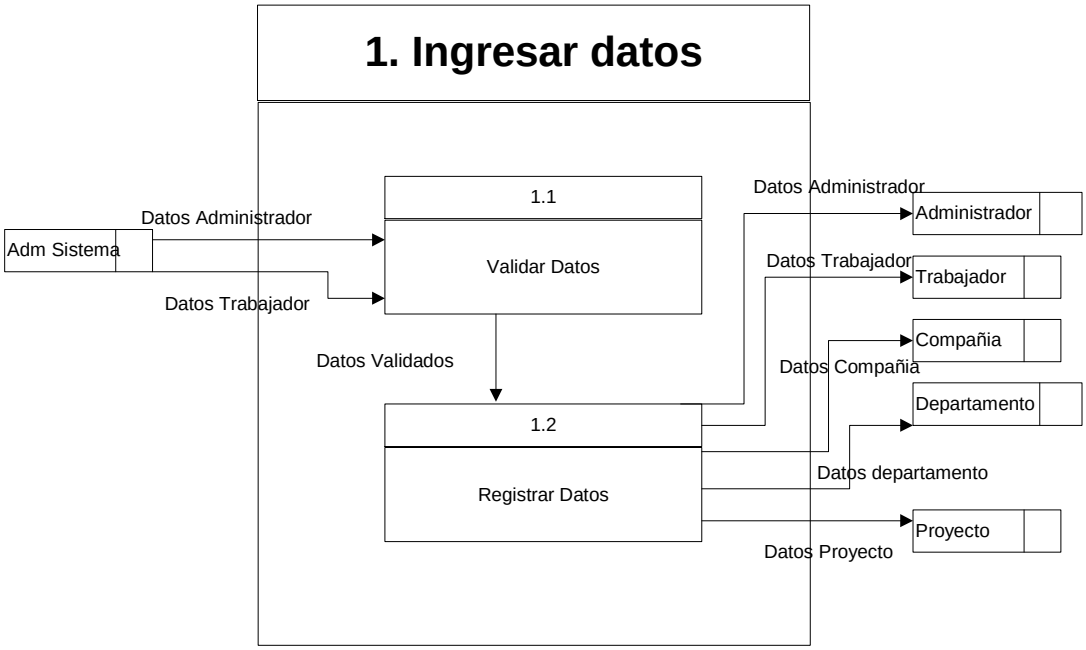


Fig 88 Representación del proceso Ingresar Datos

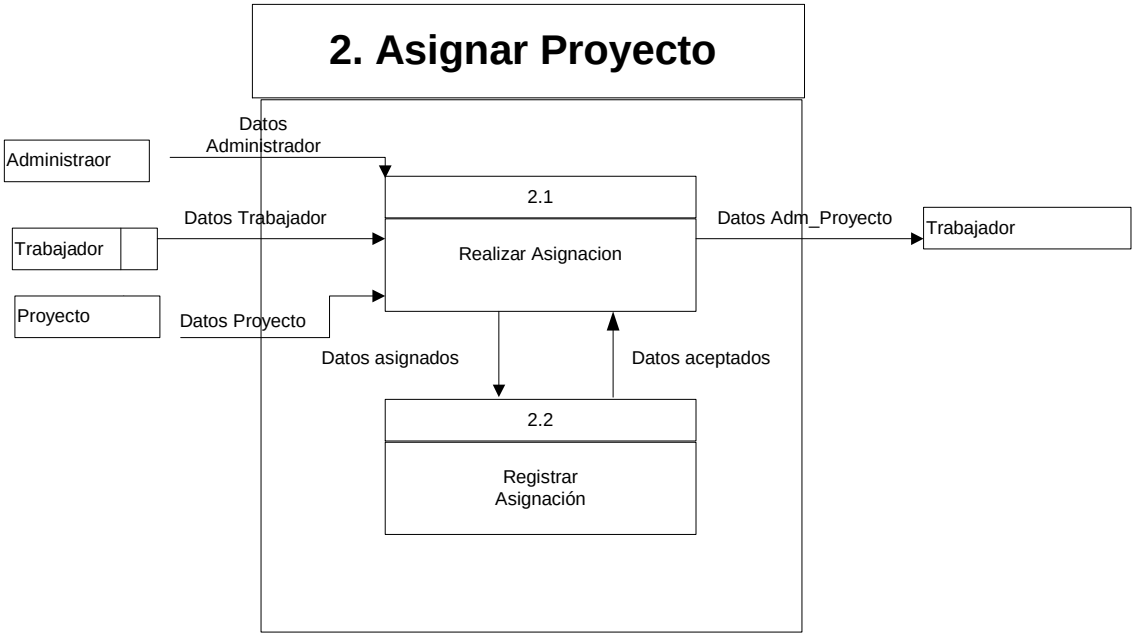


Fig 89 Representación del proceso Asignar Proyecto

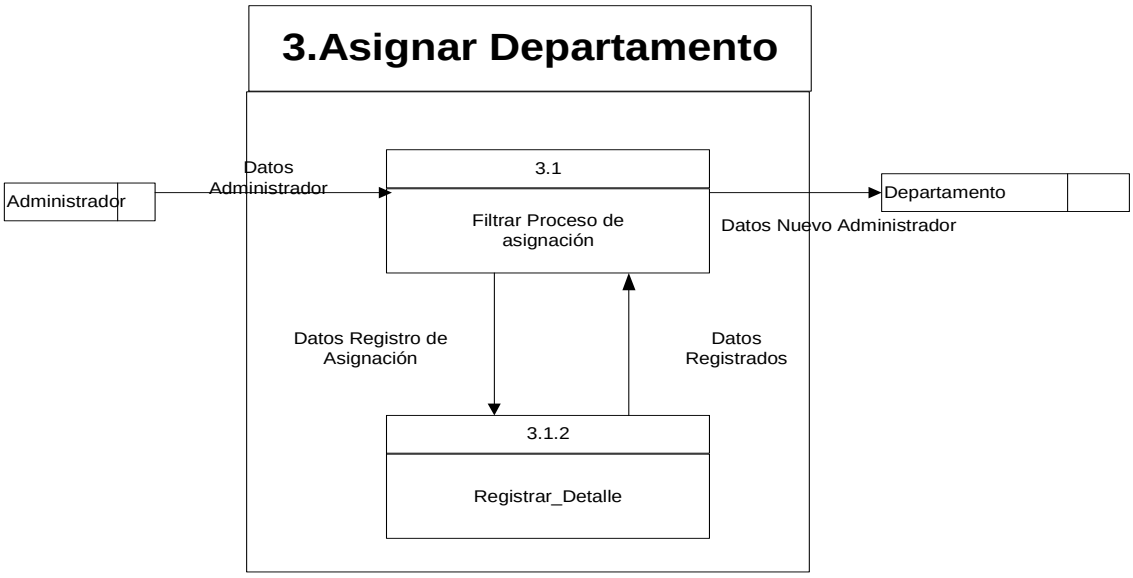


Fig 90 Representación del Proceso Asignar Departamento

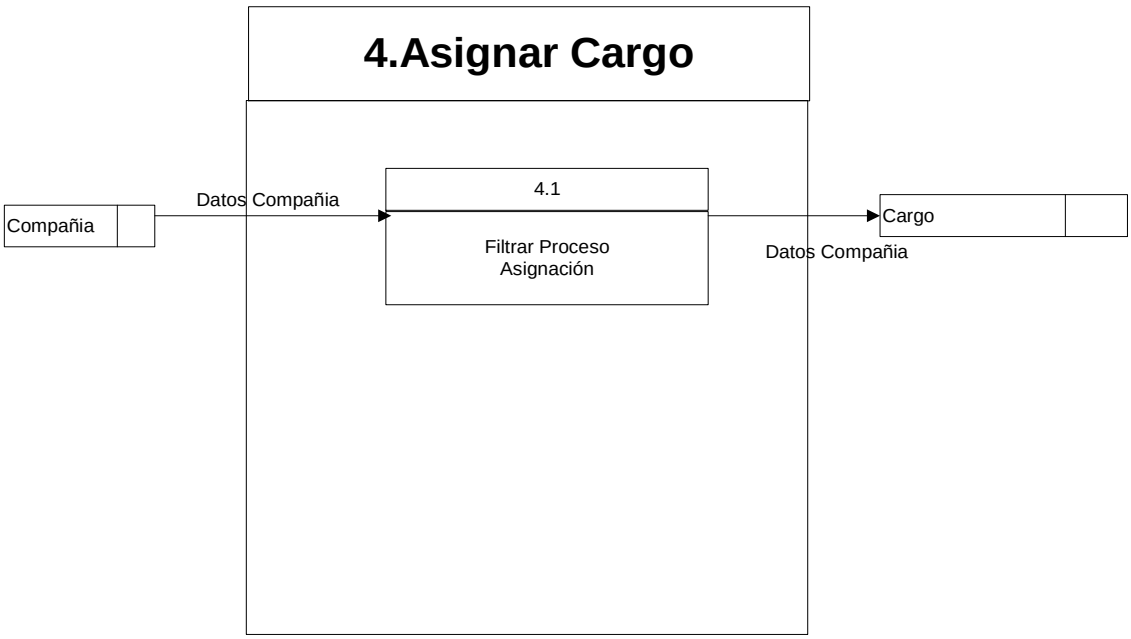


Fig 91 Representación del Proceso Asignar cargo

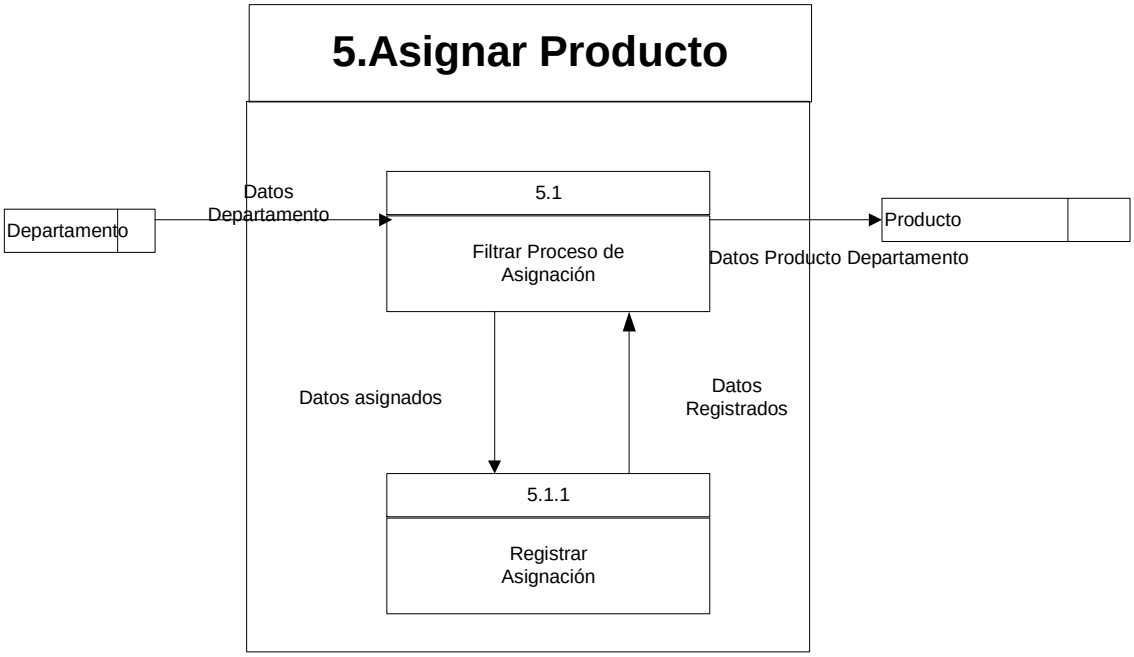


Fig 92 Representación del Proceso Asignar Producto

5.5.4 Diccionario de Flujo de Datos

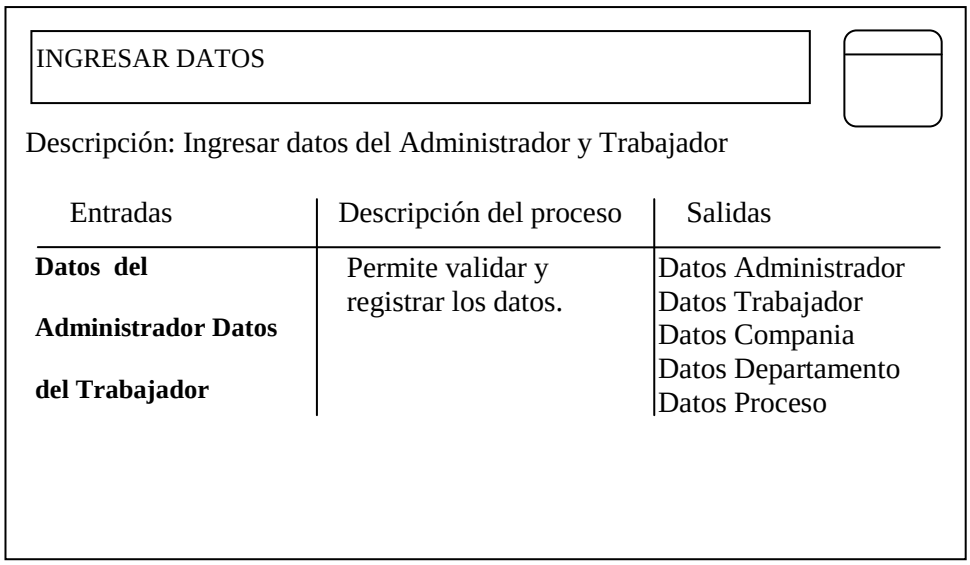


Fig 93 Esquema de Proceso Ingresar Datos

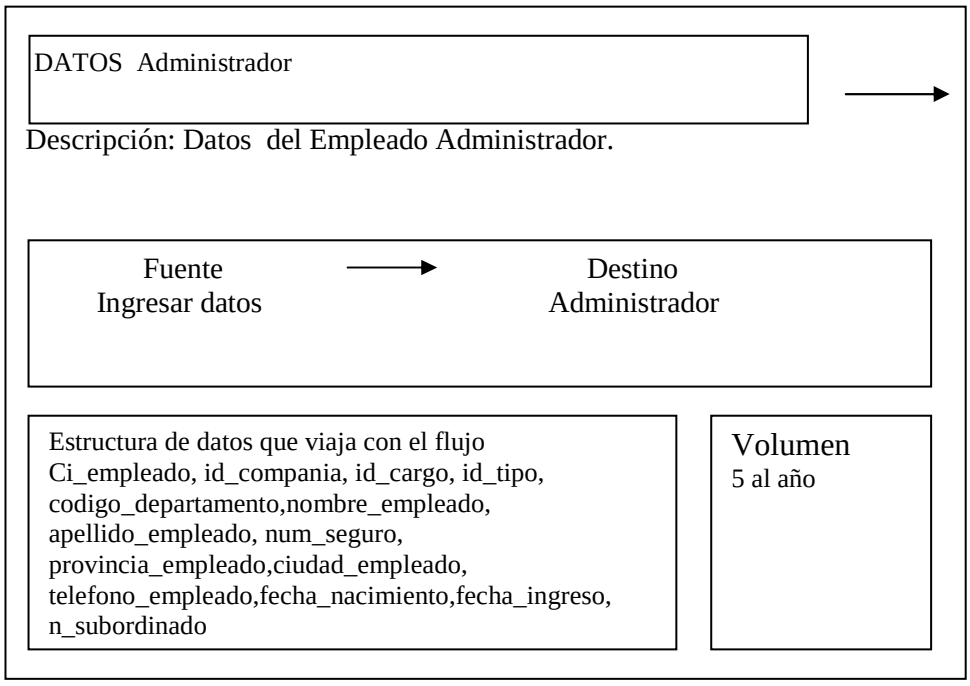


Fig 94 Esquema del Flujo de datos de Administrador

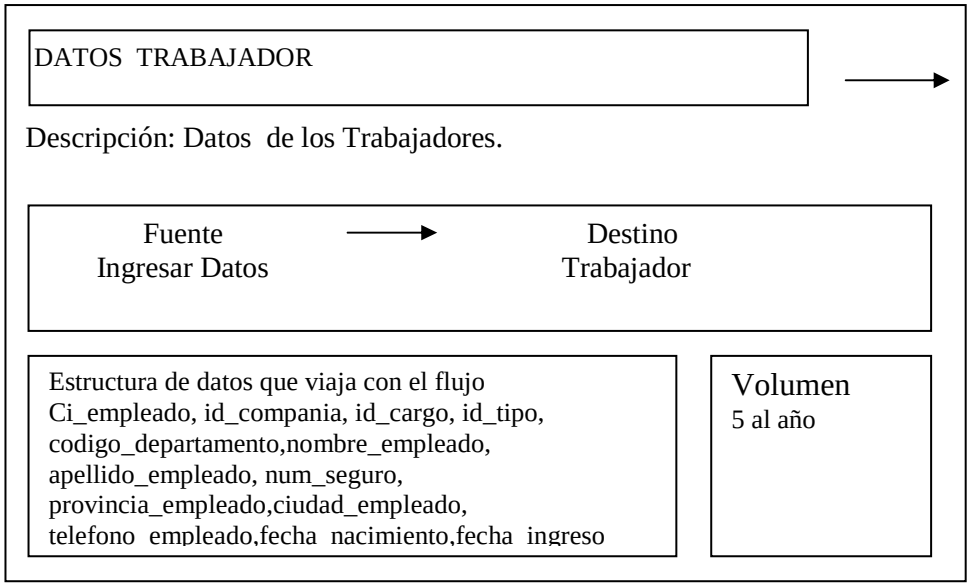


Fig 95 Esquema del Flujo de datos de Trabajador

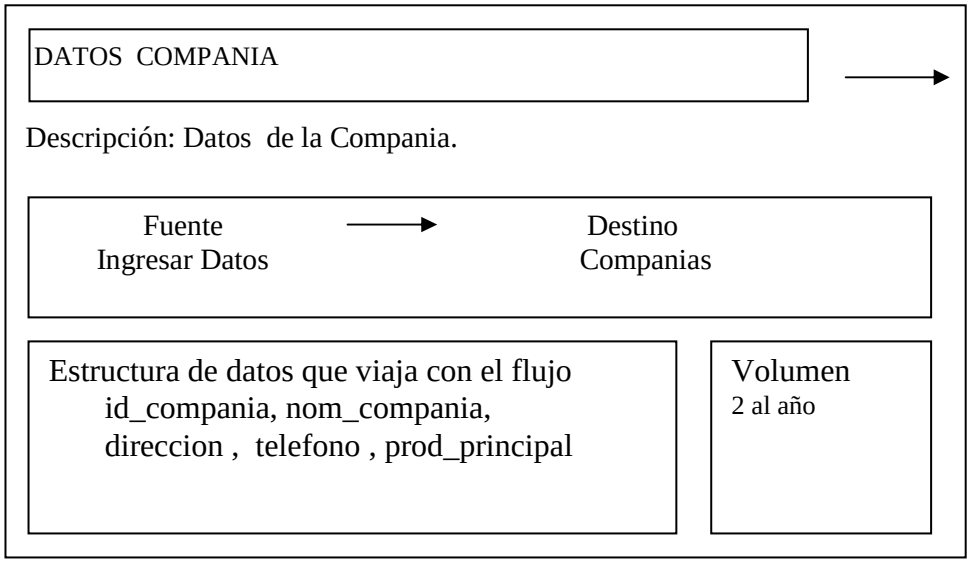


Fig 96 Esquema del Flujo de datos de Compania

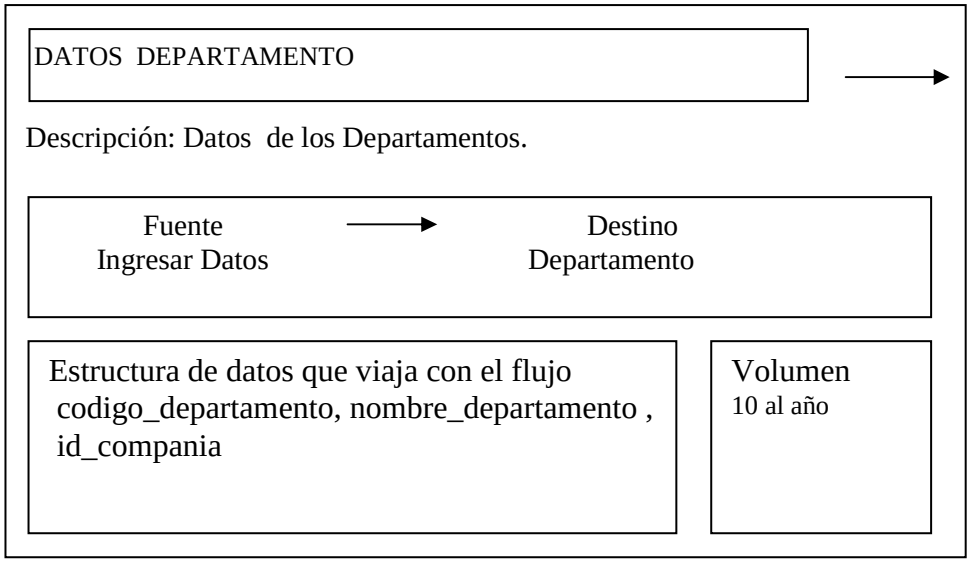


Fig 97 Esquema del Flujo de datos de Departamento

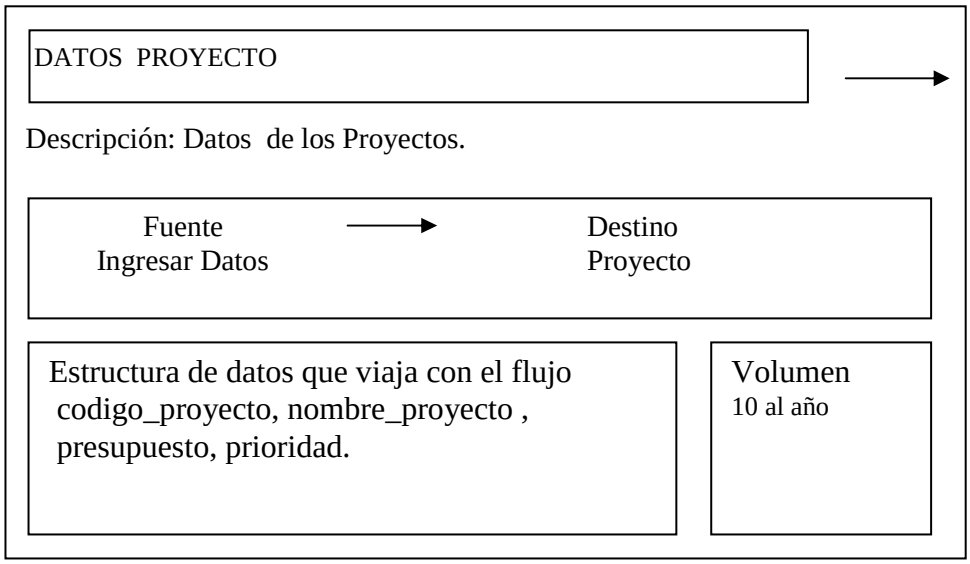


Fig 98 Esquema del Flujo de datos de Proyucto

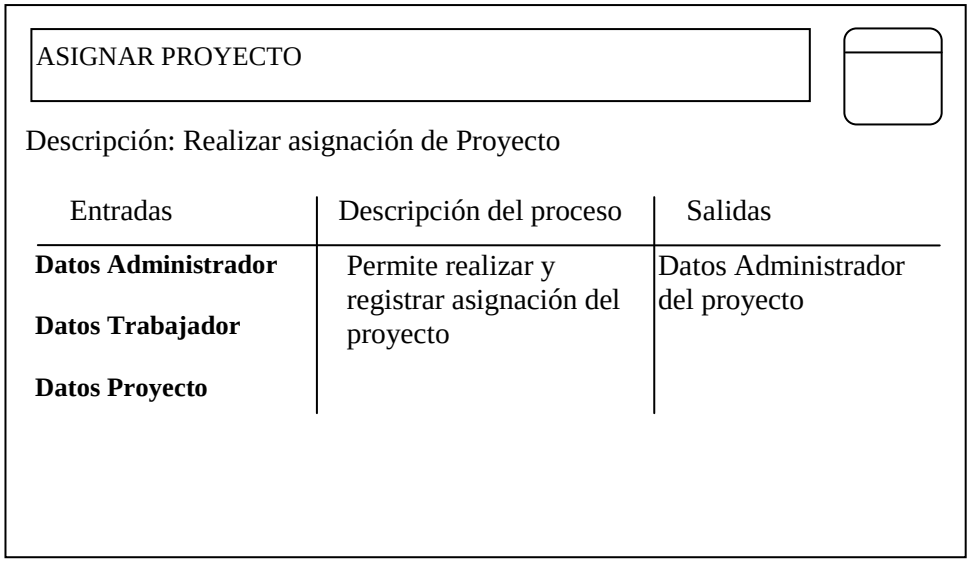


Fig 99 Esquema del Proceso Asignar Proyecto

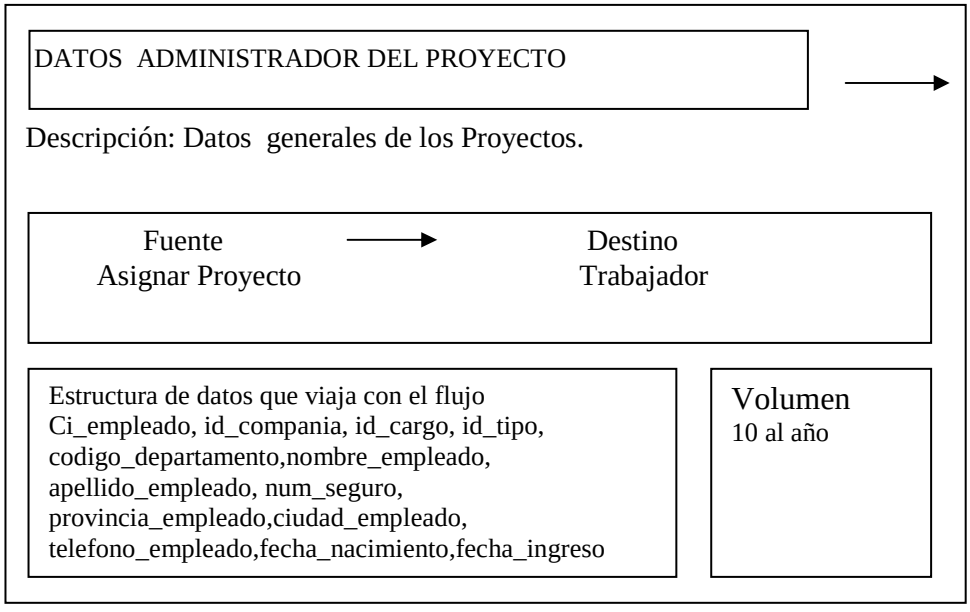


Fig 100 Esquema del Flujo de datos de Administrador del Proyecto

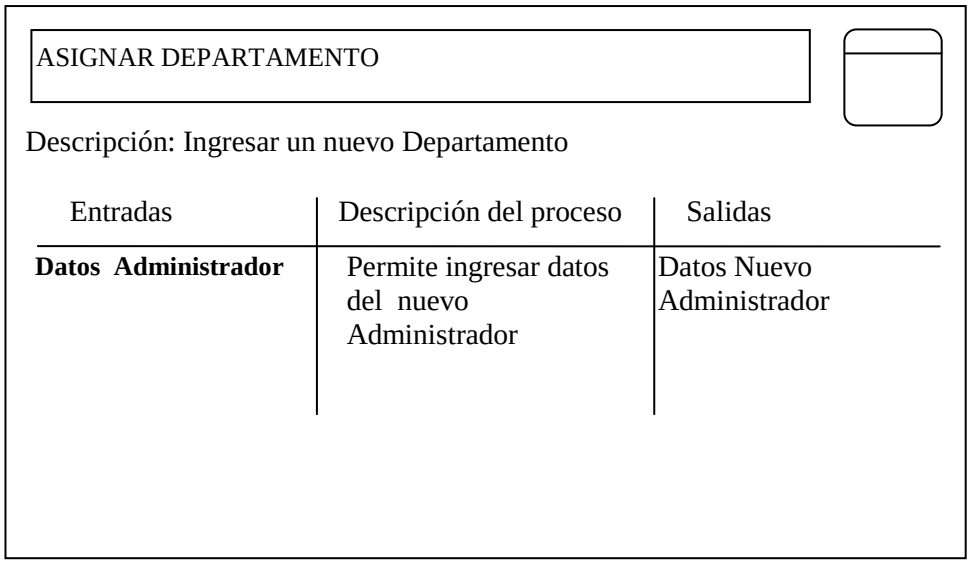


Fig 101 Esquema del Flujo de datos de Departamento

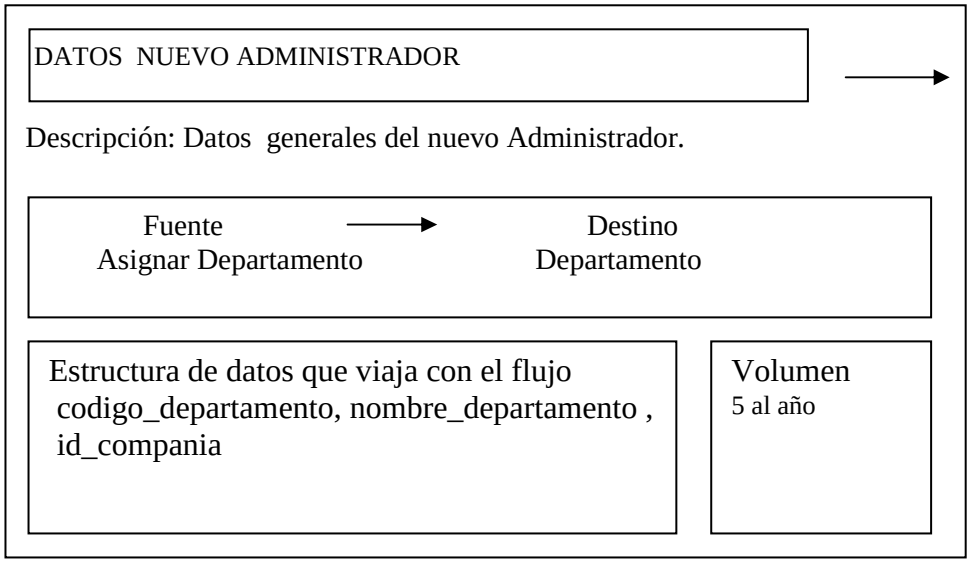


Fig 102 Esquema del Flujo de datos de Nuevo Administrador

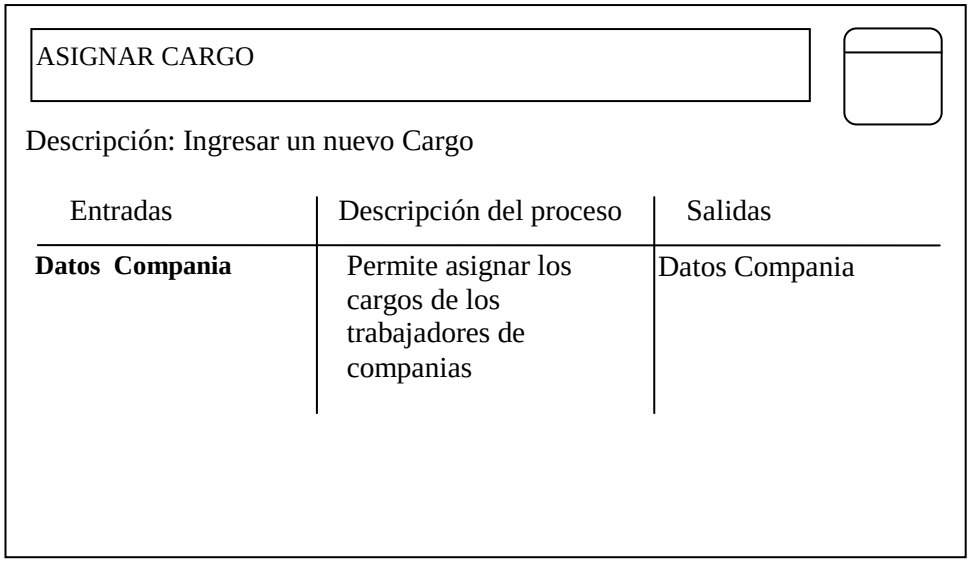


Fig 103 Esquema del Proceso Asignar Cargo

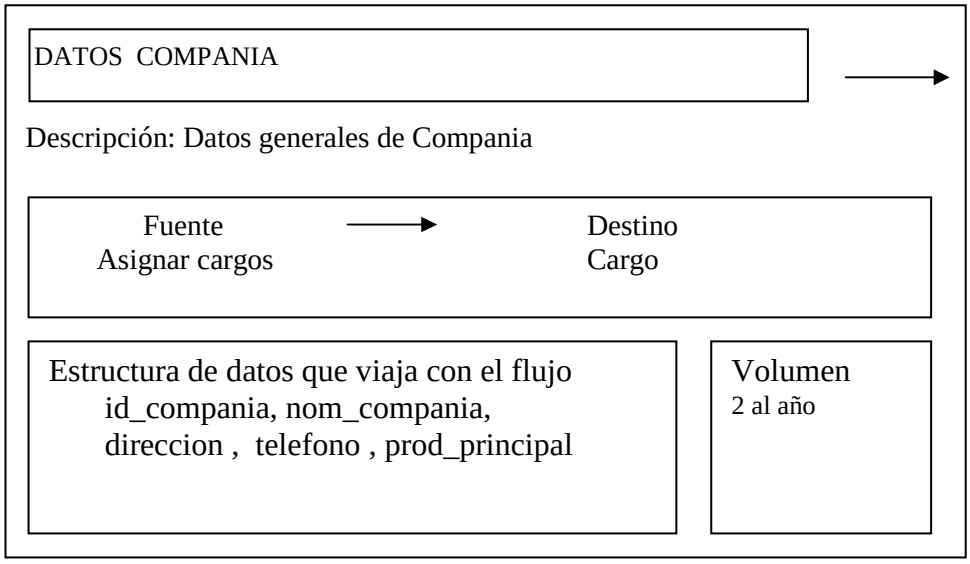


Fig 104 Esquema del Flujo de datos de Compania

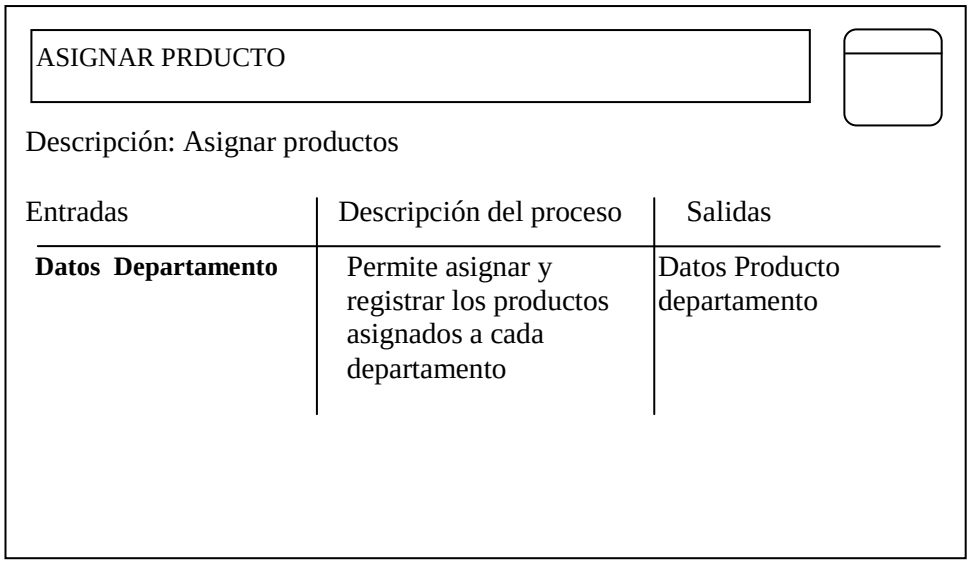


Fig 105 Esquema del Proceso Asignar Producto

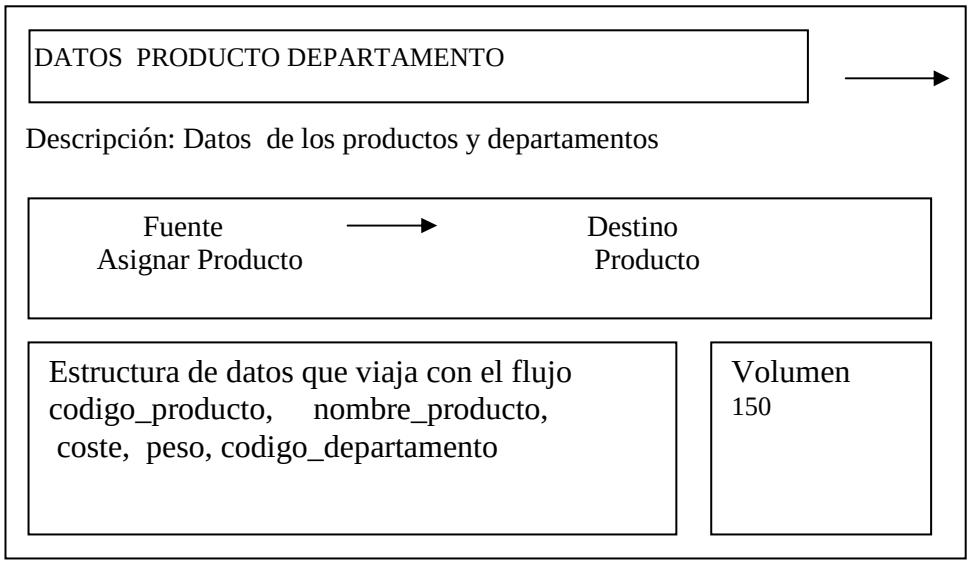


Fig 106 Esquema del Flujo de datos de Producto Departamento

5.5.5 Modelo Entidad Relación Extendido

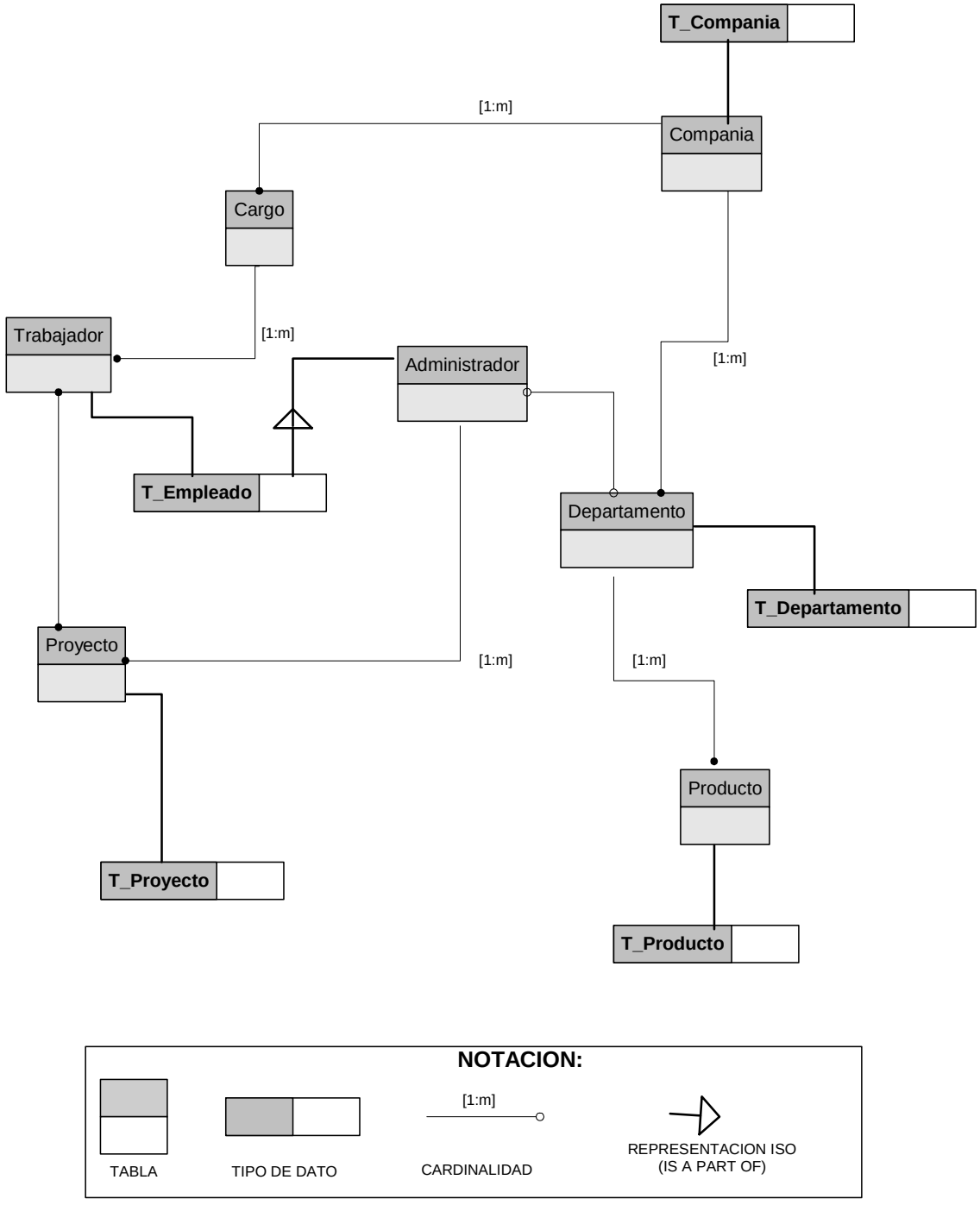


Fig 107 Diagrama modelo Entidad Extendido

5.5.6 Modelo Relacional Extendido Formal

TABLA COMPANIA DEL TIPO DE DATO T_COMPANIA

T_COMPANIA = (id_compania,
nom_compania,
direccion,
telefono ,
prod_principal)

COMPANIA **ref** T_COMPANIA
CP = ID_COMPANIA

TABLA DEPARTAMENTO DEL TIPO DE DATO T_DEPARTAMENTO

T_DEPARTAMENTO (codigo_departamento,
nombre_departamento,
id_compania)

DEPARTAMENTO **ref** T_DEPARTAMENTO
CP = CODIGO_DEPARTAMENTO
CF = ID_COMPANIA

TABLA PRODUCTO DEL TIPO DE DATO T_PRODUCTO

T_PRODUCTO = (codigo_producto,
nombre_producto,
coste,
peso,
codigo_departamento)

PRODUCTO **ref** T_PRODUCTO
CP = CODIGO_PRODUCTO
CF = CODIGO_DEPARTAMENTO

TABLA TRABAJADOR DEL TIPO DE DATO T_EMPLEADO

T_EMPLEADO = (ci_empleado,
id_compania,
id_cargo,
id_tipo,
codigo_departamento,
nombre_empleado,

apellido_empleado,
num_seguro,
provincia_empleado,
ciudad_empleado,
telefono_empleado,
fecha_nacimiento,
fecha_ingreso)

EMPLEADO **ref** T_EMPLEADO
CP = CI_EMPLEADO, ID_COMPANIA
CF = CODIGO_DEPARTAMENTO
CF = ID_TIPO,
CF = ID_CARGO
CF = ID_COMPANIA

TABLA ADMINISTRADOR DEL TIPO DE DATO T_EMPLEADO

T_EMPLEADO = (ci_empleado,
id_compania,
id_cargo,
id_tipo,
codigo_departamento,
nombre_empleado,
apellido_empleado,
num_seguro,
provincia_empleado,
ciudad_empleado,
telefono_empleado,
fecha_nacimiento,
fecha_ingreso)

EMPLEADO **ref** T_EMPLEADO
CP = CI_EMPLEADO, NSUBORDINADO
CF = CODIGO_DEPARTAMENTO

TABLA PROYECTO DEL TIPO DE DATO T_PROYECTO

T_TIPO_PROYECTO = (codigo_proyecto,
nombre_proyecto,
presupuesto,
prioridad)

PROYECTO **ref** T_TIPO_PROYECTO
CP = CODIGO_PROYECTO
CF = CI_EMPLEADO
CF = NSUBORDINADO

TABLA CARGO DEL TIPO DE DATO T_CARGO

T_CARGO = (id_cargo,
cargo,
descripcion)

CARGO **ref** T_CARGO
CP = ID_CARGO

5.5.7 Identificación De Objetos

TABLA: COMPANIA

DESCRIPCIÓN: Almacena y detalla características del Objeto COMPANIA					
CAMPO	**TIPO**	**TAMAÑO**	**DECIMALES**	**DOMINIO**	**DESCRIPCIÓN**
Id_compania	VARCHAR2	5	0	ABCDE ABCDE son letras	Identificador que representa en forma única a Compania
Nom_compania	VARCHAR2	40	-	Caracteres alfanuméricos	Nombres del Cliente
Dirección	VARCHAR2	60	-	Caracteres alfanuméricos	Dirección actual de la Compania
Teléfono	VARCHAR2	9	0	Dígitos ABCDEFGH AB=01,02,03,04,05,09,06	Número Telefónico del Funcionario
Prod_principal	VARCHAR2	40	-	Caracteres alfanuméricos	Nombres de producto principal

Tabla 1 Características de la Tabla Compania

TABLA: DEPARTAMENTO

DESCRIPCIÓN: Almacena y detalla características del Objeto DEPARTAMENTO					
CAMPO	**TIPO**	**TAMAÑO**	**DECIMALES**	**DOMINIO**	**DESCRIPCIÓN**
codigo_departamento	VARCHAR2	5	0	Caracteres alfanuméricos	Identificador que representa en forma única a cada departamento
nombre_departamento	VARCHAR2	30	-	Caracteres alfanuméricos	Nombres del departamento
Id_compania	VARCHAR2	5	0	ABCDE ABCDE son letras	Identificador que representa en forma única a Compania

Tabla 2 Características de la Tabla Departamento

TABLA: PRODUCTO

DESCRIPCIÓN: Almacena y detalla características del Objeto PRODUCTO					
CAMPO	**TIPO**	**TAMAÑO**	**DECIMALES**	**DOMINIO**	**DESCRIPCIÓN**
codigo_producto	VARCHAR2	5	0	ABCDE ABCDE son letras	Identificador que represente en forma única a cada Producto
nombre_producto	VARCHAR2	40	0	Caracteres alfanuméricos	Nombre con el que se le conoce al producto
Coste	NUMBER	6	2	Dígitos 8 ABCDEF,GH ABCDEF>100 GH<100	Valor en dólares del costo del producto

Peso	NUMBER	3	2	Dígitos 5 ABC,GH ABC>0 GH<100	Valor del peso del producto en Kg
codigo_departamento	VARCHAR2	5	0	Caracteres alfanuméricos	Identificador que representa en forma única a cada departamento

Tabla 3 Características de la Tabla Producto

TABLA TRABAJADOR

DESCRIPCIÓN: Almacena y detalla características del Objeto TRABAJADOR					
CAMPO	**TIPO**	**TAMAÑO**	**DECIMALES**	**DOMINIO**	**DESCRIPCIÓN**
ci_empleado	VARCHAR2	10	0	Dígitos ABCDEFGHIJ 01<=AB<=22 0<CDEFGHIJ	Identificador que representa en forma única a cada Empleado, y será el número de cédula de ciudadanía
Id_compania	VARCHAR2	5	0	ABCDE ABCDE son letras	Identificador que representa en forma única a Compania
id_cargo	INTEGER	-	-	Caracteres alfanuméricos	Identificador que representa en forma única id_departamento

Id_tipo	SMALLINT	-	-	Dígitos	Identificador que representa en forma única un tipo de empleado
codigo_departamento	VARCHAR2	5	0	Caracteres alfanuméricos	Identificador que representa en forma única a cada departamento
nombres_empleado	VARCHAR2	30	-	Caracteres alfanuméricos	Nombres del Empleado
apellido_emple	VARCHAR2	30	-	Caracteres alfanuméricos	Apellidos del Empleado

num_seguro	VARCHAR2	13	0	Dígitos ABCDEFGHIJKLM 01<=AB<=22 0<CDEFGHIJKLM	Identificador que representa en forma única a cada empleado, y será el número del seguro
provincia_empleado	VARCHAR2	20	-	22 Provincias del País	Provincia en que vive el empleado
ciudad_empleado	VARCHAR2	20		Caracteres alfanuméricos	Ciudad donde vive
teléfono	VARCHAR2	8	0	Dígitos ABCDEFGH AB=01,02,03,04,05,09,06	Número telefónico del empleado
fecha_nacimiento	DATE	8	-	Fecha válida	Fecha de nacimiento del empleado
Fecha_Inicio	DATE	8	-	Fecha válida	Fecha de ingreso del empleado

Tabla 4 Características de la Tabla Trabajador

TABLA ADMINISTRADOR

DESCRIPCIÓN: Almacena y detalla características del Objeto ADMINISTRADOR					
CAMPO	**TIPO**	**TAMAÑO**	**DECIMALES**	**DOMINIO**	**DESCRIPCIÓN**
ci_empleado	VARCHAR2	10	0	Dígitos ABCDEFGHIJ 01<=AB<=22 0<CDEFGHIJ	Identificador que representa en forma única a cada Empleado, y será el número de cédula de ciudadanía
Nsubordinado	NUMBER	3	0	Dígitos 3 ABC 0<ABC<100	Numero de subordinados que tiene a su cargo
Id_compania	VARCHAR2	5	0	ABCDE ABCDE son letras	Identificador que representa en forma única a Compania

id_cargo	INTEGER	-	-	Caracteres alfanuméricos	Identificador que representa en forma única id_departamento
Id_tipo	SMALLINT	-	-	Dígitos	Identificador que representa en forma única un tipo de empleado
codigo_departamento	VARCHAR2	5	0	Caracteres alfanuméricos	Identificador que representa en forma única a cada departamento
nombres_empleado	VARCHAR2	30	-	Caracteres alfanuméricos	Nombres del Empleado
apellido_emple	VARCHAR2	30	-	Caracteres alfanuméricos	Apellidos del Empleado

num_seguro	VARCHAR2	13	0	Dígitos ABCDEFGHIJKLM 01<=AB<=22 0<CDEFGHIJKLM	Identificador que representa en forma única a cada empleado, y será el número del seguro
provincia_empleado	VARCHAR2	20	-	22 Provincias del País	Provincia en que vive el empleado
ciudad_empleado	VARCHAR2	20		Caracteres alfanuméricos	Ciudad donde vive
teléfono	VARCHAR2	8	0	Dígitos ABCDEFGH AB=01,02,03,04,05,09,06	Número telefónico del empleado
fecha_nacimiento	DATE	8	-	Fecha válida	Fecha de nacimiento del empleado
Fecha_Inicio	DATE	8	-	Fecha válida	Fecha de ingreso del empleado

Tabla 5 Características de la Tabla Administrador

TABLA: PROYECTO

DESCRIPCIÓN: Almacena y detalla características del Objeto PROYECTO					
CAMPO	**TIPO**	**TAMAÑO**	**DECIMALES**	**DOMINIO**	**DESCRIPCIÓN**
Codigo_proyecto	VARCHAR2	5	0	ABCDE ABCDE son letras	Identificador que representa en forma al Proyecto
Nombre_proyecto	VARCHAR2	40	-	Caracteres alfanuméricos	Nombres del Cliente
Presupuesto	NUMBER	6	2	Dígitos 8 ABCDEF,GH ABCDEF>100 GH<100	Valor en dólares del presupuesto para el proyecto
Prioridad	VARCHAR2	10	0	Caracteres alfanuméricos	Prioridad de Proyecto

Ci_empleado	VARCHAR2	10	0	Dígitos ABCDEFGHIJ 01<=AB<=22 0<CDEFGHIJ	Identificador que representa en forma única a cada Empleado, y será el número de cédula de ciudadanía
Nsubordinado	NUMBER	3	0	Dígitos 3 ABC 0<ABC<100	Numero de subordinados que tiene a su cargo

Tabla 6 Características de la Tabla Proyecto

TABLA: CARGO

DESCRIPCIÓN: Almacena y detalla características del Objeto CARGO					
CAMPO	**TIPO**	**TAMAÑO**	**DECIMALES**	**DOMINIO**	**DESCRIPCIÓN**
id_cargo	INTEGER	-	-	Caracteres alfanuméricos	Identificador que representa en forma única id_departamento
cargo	VARCHAR2	30	-	Caracteres alfanuméricos	Nombres del cargo
descripcion	VARCHAR2	60	-	Caracteres alfanuméricos	Nombres de la descripción del cargo

Tabla 7 Características de la Tabla Cargo

CAPITULO VI

CONCLUSIONES Y RECOMENDACIONES

6.1 CONCLUSIONES

- La utilización de diagramas nos ayudó en una forma explicita y visual el entendimiento de la Metodología
- Se hizo necesario la utilización de la notación de conjuntos para la realización del proceso matemático en el desarrollo de las reglas de transformación
- Se aprovecho la documentación existente para guiarnos en diferentes conceptos y mecanismos durante el desarrollo del tema.

6.2 RECOMENDACIONES

- Es importante que el lector de este trabajo tenga conocimiento de los conceptos fundamentales de la tecnología Orientado a Objeto, tales como Clases, instancias, interface, polimorfismo, encapsulación y herencia.
- Para nuestros lectores, llamémoslos novicios en el modelado Orientado a Objetos, es necesario conocer el camino a recorrer cuando se trabaja con Objetos; esta tesis presenta un bien definido plan de actividades, etapa por etapa, para lograr este objetivo partiendo del Modelo Requerimiento hasta el Modelo Orientado a Objetos.
- Utilizar herramientas potentes de modelamiento de objetos que nos permita facilitar el análisis, diseño y desarrollo de la aplicación, como por ejemplo el Oracle Designer, Racional Rose.

GLOSARIO

Almacén de Datos: Objeto pasivo que almacena datos para un acceso posterior.

Asociación Cualificada: Asociación que relaciona dos clases y un cualificador.

BD: Bases de Datos

BDOO: Bases de Datos Orientadas a Objetos

BDR: Bases de Datos por Relación

CAD: Diseño Asistido por Computadora

Coherencia: Propiedad de una entidad como una clase, una operación o un modulo.

Cualificador: Atributo de un objeto que se distingue entre el conjunto de objetos.

DFD: Diagrama de Flujo de Datos.

Metaclase: Clase que describe otras clases.

OO: Orientación a Objetos.

ODL: Estándar de Definición de Lenguaje de Datos

OMG: Grupo Manejador de Objetos

OML: Lenguaje de Manipulación de Datos

OMT: Técnica de Modelado de Objetos.

Proceso: Algo que transforma valores de datos.

Rol: Un terminal de una Asociación.

Tipo: Conjunto de objetos o valores con un comportamiento similar.

ODMG: Gestión Manejadora de datos Objeto

OQL: Equivalente al SQL(Lenguaje de Consulta)

SQL: Lenguaje de Consulta

SGBD: Sistema de Gestión de Bases de Datos

SGBDOO: Sistema de Gestión de Bases de Datos Orientada a Objeto

SO: Sistema Operativo