UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

Facultad de Ciencias

PANORAMA DE LAS TECNOLOGIAS ACTUALES EN BASES DE DATOS

T E S I S

Que para obtener el título de

A C T U A R I O

p r e s e n t a

JOSE CARLOS MALDONADO
BAEZ

México, D. F. Junio 1990

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

PANORAMA DE LAS TECNOLOGIAS ACTUALES EN BASES DE DATOS.

INDICE.

INTRODUCCION.

La utilización de las computadoras, con el fin de auxiliar en las tareas cotidianas de una organización, se justifica al simplificar y mejorar la forma de llevar a cabo tales tareas de una manera que nunca se había imaginado. En todo tipo de organización, la computadora almacena y facilita el manejo de cantidades muy grandes de información en períodos de tiempo muy cortos, hoy en día, es muy difícil conceptualizar las organizaciones sin las computadoras.

El manejo de la información, ha sido siempre una tarea de cualquier civilización, ya que ante el crecimiento de la industria, la economía y la población, existe la necesidad de manejar adecuadamente cantidades considerables de datos relacionados con diversos objetivos, entre los que destacan, los comerciales y administrativos. A esta información interrelacionada y vista como un todo, se le denomina un ***sistema de información***. La parte más importante del sistema de información son los datos que almacena. Para una organización, las necesidades de información casi siempre rebasan fronteras de diversas divisiones dentro de la misma organización o corporación, lo cual significa que los datos deben ser almacenados de tal forma que se encuentren **disponibles a la organización entera.**

Conforme los años han pasado, los sistemas, equipos y procedimientos usados para el proceso electrónico de datos, han llegado a ser sumamente sofisticados. Sin embargo, los objetivos básicos de dicho proceso, continúan estando vigentes, los cuales consisten en **proveer a la organización de la información correcta, en el tiempo adecuado, a la persona idónea y a un costo razonable.**

El medio ambiente de archivos convencionales, el cual fue utilizado intensamente antes de la llegada de los esquemas de bases de datos, tiene varias desventajas, de las cuales es importante considerar la existencia de redundancia de datos, los problemas con la integridad, las limitaciones para compartir información y las restricciones en el uso de la información.

Una vez mencionados los problemas principales a que se enfrentan las organizaciones con esquemas de manejo de información tradicionales, se puede establecer el objetivo al que fue orientado la ***tecnología de bases de datos***, dicho objetivo es resolver los problemas citados. Una de las tareas fundamentales del sistema de bases de datos es facilitar el uso de la información en un ***esquema abstracto***, es decir, en forma ideal debe permitir al usuario solo tener que especificar **qué debe** hacerse con la información, sin tener que ocuparse de determinar **cómo hay que** realizar las acciones sobre los datos.

Los sistemas de bases de datos son de casos más complejos dentro del software, y por lo tanto sufren actualmente de la denominada ***crisis del software***, crisis caracterizada por el tamaño y complejidad crecientes que el software moderno detenta hoy en día.

El control del desarrollo y mantenimiento de software, siempre involucra el manejo de la complejidad intelectual de programas y sistemas de programas. Los sistemas no solo se crean, deben ser probados, mantenidos y extendidos. Como resultado, mucha gente debe entenderlos y modificarlos varias veces durante su ciclo de vida.

Un tema que domina en la evolución de las metodologías del software, es el desarrollo de herramientas para el manejo de ***abstracciones*** y ***modelos***, situación que se aborda en el **capitulo I** de este trabajo.

La esencia del análisis para bases de datos, es idear o conceptualizar estructuras de datos que puedan sustentar **cualquier** aplicación. Esto incluye aplicaciones futuras que requieran de la información existente. En el análisis, la información en uso o necesaria, debe ser identificada y clasificada de manera que vaya más allá de las políticas, prácticas, prioridades y aplicaciones actuales. El objetivo es documentar y modelar el tipo de cosas con las que la organización trata, esto da un punto común de referencia denominado ***modelo de datos*** para la organización completa, el cual se obtiene como resultado del proceso de abstracción.

Para el estudio adecuado de dichos modelos en el contexto de las bases de datos y su tendencia de desarrollo, es importante la realización de una clasificación de los modelos, a fin de introducirse en los avances de la investigación actual. El análisis se desarrolla en el **capitulo II** y se haya sustentado en la determinación precisa de los conceptos de modelos de datos y su terminología.

Los modelos de datos proveen las bases conceptuales para el proceso de las aplicaciones, y dotan de una base formal para la construcción de herramientas adecuadas para el desarrollo de los sistemas de información.

Actualmente las herramientas se desarrollan abarcando tipos de aplicaciones distintas a las conocidas como ***sistemas de información tradicionales***, en los nuevos tipos de sistemas existe ahora la tendencia de utilizar de forma intensiva la tecnología de las bases de datos.

Para la incorporación de las nuevas aplicaciones al ámbito de las bases de datos, es necesario alcanzar ciertos objetivos, los cuales conforman un reto ya que deben facilitar el desarrollo de las aplicaciones denominadas ***no estándar***, las cuales son mucho más complejas que las denominadas aplicaciones tradicionales, administrativas o de negocios, asimismo es necesario proveer de mecanismos para la manipulación completa de la **información** y el **conocimiento** almacenados en la base de datos.

El alcanzar tales objetivos marca la pauta del desarrollo de las tendencias tecnológicas en cuanto a bases de datos se refiere.

Los nuevos sistemas de bases de datos, necesitan ofrecer conceptos a fin de satisfacer las necesidades que surjan en los tipos de aplicación a incorporar, de manera que renueven la interacción con el usuario y sean más flexibles.

Se han definido algunas tendencias para mejorar la tecnología actual de los sistemas de bases de datos, destacando la incorporación de conceptos desarrollados en otras áreas de la computación, como son la Inteligencia Artificial y los Lenguajes de Programación, en el **capitulo III** de este trabajo se hace una revisión de las necesidades específicas de las nuevas aplicaciones, las tendencias de desarrollo de la tecnología de bases de datos para satisfacer dichas necesidades y se analizan los avances de la investigación actual.

Dentro de los avances obtenidos por la investigación en materia de bases de datos, es importante resaltar algunos de ellos, la razón fundamental de lo anterior obedece a que dichos avances han marcado la pauta que orienta las investigaciones subsecuentes.

De los resultados mencionados, cabe destacar los relativos a las extensiones de los sistemas de bases de datos que han sido propuestas con el fin de soportar estructuras jerárquicas (*objetos complejos*) e implementar dichas extensiones, la forma convencional de realizar lo anterior, ha sido sobre algún sistema de bases de datos existente, lo que conlleva desventajas, no solo desde el punto de vista conceptual, ya que también implica problemas en el aspecto de rendimiento del sistema modificado.

Como ejemplo de las tendencias a minimizar los problemas de las primeras implementaciones en este ámbito, en el **apéndice A** se analiza un prototipo de sistema de bases de datos sustentado en un modelo relacional generalizado denominado **NF² extendido**, cuya característica principal consiste en que maneja de una manera uniforme tanto las relaciones normalizadas (relaciones planas), como las estructuras jerárquicas. Los tópicos de la revisión realizada consideran los aspectos como la caracterización del modelo lógico de datos, el lenguaje para dicho modelo y las estructuras de almacenamiento necesarias que permitan la implementación de las relaciones generalizadas.

En el **apéndice B** se trata un modelo de datos denominado **NST** el cual está basado en secuencias de n-eadas, y pertenece a la categoría de los modelos relacionales no normalizados o ***modelos de relaciones anidadas***, dicho modelo permite la representación de objetos jerárquicos y considera la existencia de un álgebra que dota de formalidad a la manipulación y reestructuración de dichos objetos.

En los últimos tiempos, una serie de investigaciones se ha enfocado al tópico de la información incompleta y los valores nulos, lo anterior responde a la necesidad de manejar símbolos especiales para el manejo de información incompleta en la base de datos, estos símbolos especiales son conocidos como ***valores nulos***. El problema básico con este tipo de valores, es la diversidad de interpretaciones que estos tienen.

En el **apéndice C** se analiza un nuevo enfoque formal para el manejo de información incompleta en las bases de datos a través de ***valores nulos***, la base de este enfoque consiste en las interpretaciones de dichos valores nulos, los cuales requieren de más de un ***tipo de valor nulo***. La validez conceptual del enfoque presentado, se sustenta en la generalización de la estructura formal del modelo relacional al considerar valores nulos. En particular se pone especial énfasis en las propiedades de su conjunto teórico y se generalizan las definiciones de contención, unión y diferencia de conjuntos.

Dadas las limitaciones para la expresión semántica en los modelos tradicionales de datos, la investigación se ha orientado a establecer modelos semánticos de datos, los cuales deben proveer **especificación entendible** al usuario y capturar una porción substancial del **significado de los datos**. Las diferencias principales entre los modelos de datos, son diferencias en el tipo de relaciones que pueden ser representadas y en las restricciones que se imponen a ellas.

En el **apéndice D**, se presenta un modelo de datos orientado a objetos, **OODM**, para el diseño y acceso de una base de datos, este modelo trabaja con tres tipos de relaciones (agregación, generalización y particularización). Además, provee cuatro tipos de operaciones con los datos (definición de un esquema, creación de una base de datos, recuperación de objetos y expansión de objetos), y soporta una interface estructurada basada en menús la cual permite al usuario definir el esquema y la recuperación de la base de datos sin tener que poseer ningún conocimiento del modelo de datos orientado a objetos.

Una cuestión importante, la cual debe considerarse tanto para diseño, como para el proceso de llevar a cabo bases de datos, consiste en definir los esquemas de representación de datos en el ***nivel físico***. El problema básico en la representación física de una base de datos, se trata en el **apéndice E** bajo la panorámica de almacenar un ***archivo*** en diversos tipos de organización, al mismo tiempo que se revisan las operaciones típicas (inserción, eliminación, modificación y búsqueda) que se quiere realizar sobre dicho archivo, lo anterior se debe a que la elección del tipo de organización de almacenamiento de un archivo, depende principalmente de las características que se desea tengan las operaciones mencionadas.

I. *USO DE MODELOS Y ABSTRACCION EN EL DISEÑO DE SOFTWARE.*

Los principales problemas del software moderno son su tamaño creciente y su gran complejidad, los cuales involucran la necesidad de encontrar técnicas y herramientas más efectivas para su desarrollo y mantenimiento.

Las principales controversias del software moderno, derivan del contraste existente entre el alto costo de desarrollo, uso y mantenimiento y la baja calidad resultante de dicho software. Tales controversias cobran especial relevancia por las características (gran tamaño, largo ciclo de vida, alto número de programadores involucrados) del tipo de programas que conforman el software de hoy en día.

El concepto clave de la programación moderna en el proceso de controlar la complejidad, es la abstracción, esto es, el énfasis selectivo en los detalles. Los nuevos desarrollos en lenguajes de programación, proveen formas de soportar y explotar técnicas de abstracción.

De lo anterior se desprende la necesidad de considerar soluciones como el ***impacto del modelado*** y las ***técnicas de abstracción*** al problema del alto costo y la baja calidad.

El modelado no sólo ha influenciado las áreas de programación, una vez que los programas no son el único tipo de sistemas complejos con que se trata, otros sistemas como la economía o el movimiento de los átomos en un cuerpo, son mucho más complejos, es decir, no existe ser humano que entienda dichos sistemas en todos sus detalles, por lo anterior, se trata con ese tipo de sistemas ignorando detalles no relevantes, es decir, se desarrollan modelos que reflejen las propiedades macroscópicas de comportamiento que sean importantes.

El control del desarrollo y mantenimiento de software siempre involucra el manejo de la complejidad intelectual de programas y sistemas de programas. Los sistemas no solo se crean, deben ser probados, mantenidos y extendidos. Como resultado, mucha gente debe entenderlos y modificarlos varias veces durante su ciclo de vida.

Un tema que domina en la evolución de las metodologías del software, es el desarrollo de herramientas para el manejo de abstracciones.

Una ***abstracción*** es una descripción simplificada (especificación) de un sistema que resalta algunos de los detalles o propiedades del sistema, al mismo tiempo que suprime otros. Una buena abstracción es aquella en la cual la información relevante al lector (usuario) es enfatizada y los detalles difusos o irrelevantes son ignorados (por el momento).

El concepto que se maneja como abstracción en el ámbito del software, corresponde cercanamente al concepto de modelado analítico de otros campos.

La construcción de un modelo comienza con observaciones e hipótesis que dan lugar a un conjunto de axiomas o principios que intentan explicar las observaciones. Los axiomas se utilizan para la construcción o derivación de modelos o sistemas observables. Los parámetros o variables de los sistemas se pueden extraer o estimar de los axiomas. El modelo entonces, se puede utilizar para hacer predicciones de las propiedades del sistema observado. El paso final consiste en realizar experimentos (observaciones en un medio ambiente controlado) para determinar la solidez y validez del modelo y sus principios.

En el desarrollo del software, las necesidades o la funcionalidad requerida juegan el papel de las observaciones a ser explicadas, por lo que el proceso de abstracción es muy similar al paradigma de modelado general; el proceso incluye decidir qué características de un sistema son importantes, qué variabilidad (parámetros) debe ser incluida, qué formalismo para la descripción debe utilizarse, cómo debe validarse el modelo, etc.

Como en otros campos, es posible definir jerarquía de modelos en la cual, los modelos de más bajo nivel proveen de explicaciones del fenómeno más detallados que los modelos de mayor nivel.

Se conoce a la descripción abstracta de un modelo como su ***especificación*** y el siguiente nivel en la jerarquía como su ***implementación***, la validación de que la especificación es consistente con la implementación es denominada ***verificación***.

La abstracción que se utiliza para el software tiende a resaltar las propiedades del mismo, al enfatizar qué resultados deben obtenerse, suprimiendo detalles de como se alcanzan dichos resultados.

Muchas técnicas y principios de organización de software se han desarrollado en el principio de la abstracción, dichas técnicas no sólo han servido para el entendimiento y consideración adecuada de los sistemas, además, se han convertido en herramientas para la especificación formal de los mismos.

I.1. PRIMERAS TECNICAS DE ABSTRACCION EN EL SOFTWARE.

En los primeros años de la década de los sesentas, el conjunto de tópicos importantes de programación, fue dominado por la sintaxis de los lenguajes, técnicas de traducción y la solución de ciertos problemas específicos de implementación, como las localidades de almacenamiento y la representación de datos, los procedimientos y las bibliotecas fueron utilizadas y difundidas, dichas bibliotecas se encontraron con diversos problemas, debidos básicamente, al tipo de documentación (especificación informal) que era inadecuada o por que la parametrización de las mismas no soportaban los casos de interés. Las estructuras de datos básicas como las pilas y las listas ligadas, comenzaban a ser entendidas y aplicadas en forma generalizada, sin embargo, continuaban siendo oscuras, lo que ocasionó que fueran separadas de las aplicaciones comunes de aquellos tiempos.

La abstracción fue tratada de manera profunda como una técnica de organización de programas en los últimos años de la década de los sesentas. Los primeros lenguajes soportaban ciertos tipos de datos como enteros, reales, arreglos, y algunas veces booleanos y reales de alta precisión; las estructuras de datos fueron tratadas de manera sistemática solo hasta 1968, y la primera noción de que un programador podría definir sus tipos de datos orientados a una aplicación particular, apareció en 1967.

I.2. LENGUAJES EXTENDIBLES.

A finales de los años sesenta se manifestaron esfuerzos de abstraerse de las notaciones propias de los lenguajes de programación, de forma tal, que cualquier programador pudiera agregar nueva notación y nuevos tipos de datos al lenguaje base. Los objetivos del trabajo de los lenguajes extendibles, incluían permitir a los programadores extender la sintaxis del lenguaje de programación, definir nuevas operaciones para los nuevos tipos de datos o los ya existentes. Los esfuerzos de esos tiempos fracasaron, debido principalmente, a la subestimación de la dificultad para definir tanto extensiones interesantes como encontrar técnicas para describir una extensión de manera adecuada. Sin embargo, este tipo de esfuerzos dejaron influencia para el desarrollo de los tipos de datos abstractos y las definiciones genéricas de los años setentas.

I.3. PROGRAMACION ESTRUCTURADA.

En los últimos años de la década de los setenta, surgió una metodología para la construcción de programas, desde el establecimiento de los objetivos de dichos programas hasta el código final, pasando por una serie de estados intermedios cada vez más precisos, a esta metodología se le denominó ***refinamiento paso a paso*** o ***programación de arriba a abajo***, esta metodología involucra el enfoque de un problema a través de escribir un programa que es libre de asumir la existencia de cualquier estructura de datos u operación.

Bajo el nuevo enfoque, el programa inicial es pequeño, claro y relacionado directamente al problema y "obviamente" correcto.

Aunque las estructuras de datos y las operaciones podían ser especificadas únicamente de manera informal, la intuición de los programadores en relación a ellas, debía hacerse de manera que permitiera concentrarse en la organización general del programa y relegar a segundo término los factores que implicaran tales especificaciones.

Para cada una de las definiciones posteriores se aplica la misma técnica y las implementaciones de las operaciones del nivel superior son sustituidas por las invocaciones correspondientes, el resultado es un programa nuevo y más detallado, los pasos sucesivos del desarrollo de un programa agregan detalles que son más relevantes al lenguaje de programación, hasta que el programa está expresado completamente en términos de operaciones y tipos de datos del lenguaje base, del cual se dispone un compilador o interprete.

La separación de las estructuras y la forma en que estas son utilizadas para resolver un problema, provee de una metodología para descomponer problemas complejos en problemas más pequeños, los cuales son segmentos manipulables e independientes. La clave del éxito de dicha metodología es el grado de abstracción impuesto por la selección a muy alto nivel de las estructuras de datos y las operaciones.

La limitación más importante de la metodología, la cual no se visualizó hasta que no fue utilizada por un largo tiempo, consiste en que el programa final no preserva la serie de abstracciones a través de la cual fue creado y entonces la tarea de modificar dicho programa no es sencilla una vez que está completo. Una limitación adicional es que las descripciones informales de las operaciones no proveen de información precisa, situación que desemboca en malos entendidos que complican el desarrollo y mantenimiento de los programas. El desarrollo de las técnicas para la especificación formal de programas ayuda a solventar tales problemas.

Al mismo tiempo que se instauraba la metodología comentada anteriormente, otro tópico comenzó a tener vital importancia, el cual es relativo a cómo organizar los programas a fin de que sean fáciles de entender y por lo tanto de modificar. Los ***patrones de flujo de control***, hacen que los programas sean entendibles de manera fácil. La primera discusión que tocó el tópico mencionado apareció en 1968 y convergía en un conjunto "ideal" de estructuras de control. Aunque no se ha alcanzado un consenso general en relación a este conjunto de estructuras de control, la validez de la propuesta del mismo, no ha sido descartada.

I.4. ESPECIFICACION Y VERIFICACION.

En paralelo con el desarrollo del ideal de estructuras de control y como parte de su motivación, los científicos de la computación se interesaron en encontrar formas de hacer aseveraciones matemáticas acerca de un programa de manera precisa. La posibilidad de hacer tales aseveraciones, es esencial en el desarrollo de técnicas para el análisis de los programas, particularmente, para las técnicas relacionadas con especificaciones abstractas. Se necesitó de nuevas técnicas debido a que los encabezados de las rutinas o procedimientos aún acompañados de comentarios, proveen de información inadecuada para el análisis preciso de los programas, generando ambigüedades.

La noción de que es posible establecer aseveraciones formales acerca de los valores de las variables (un conjunto de valores para las variables de un programa se denomina ***estado del programa***) y el razonamiento riguroso acerca del efecto de la ejecución de una instrucción sobre el estado del programa, aparecieron a finales de los años sesenta.

Un lenguaje de programación se describe como un conjunto de reglas que definen el efecto que cada instrucción tiene sobre la fórmula lógica que describe el estado del programa. Las reglas para el lenguaje son aplicadas a las afirmaciones del programa, a fin de obtener teoremas cuyas pruebas aseguren que el programa corresponde con la especificación. Cuando se aplican a la ligera, las técnicas de verificación tienden a estar propensas al error y las especificaciones formales lo mismo que las informales son susceptibles de errores de omisión, con el fin de responder a este problema, se han desarrollado sistemas para realizar la verificación de forma automática.

La verificación requiere convertir un programa escrito con proposiciones lógicas en teoremas lógicos con la propiedad de que el programa es correcto si y solo si los teoremas son verdaderos. Este proceso de conversión denominado ***generación de condiciones de verificación***, está muy generalizado, pero existe aún trabajo considerable por realizar en la prueba de los teoremas.

Cuando el énfasis en la metodología de programación se orientó a utilizar estructuras de datos como una base de la organización de programas, surgieron problemas relativos a las técnicas de especificación y verificación. Los esfuerzos iniciales se orientaron a la cuestión de determinar qué información es útil en la especificación, después, la atención se concentró en hacer que las especificaciones fueran más formales y que consideraran los problemas de verificación.

I.5. TIPOS DE DATOS ABSTRACTOS.

En la década de los años setenta se reconoció la importancia de organizar los programas en módulos, de manera que la información acerca de los detalles de implementación fuera localizada de la forma más rápida posible. Esto llevó a diseñar lenguajes para tipos de datos, es decir, para especificaciones que son organizadas usando la misma estructura como datos y para ***definiciones genéricas***. Las facilidades de estos lenguajes se basan en el constructor ***clase*** de **Simula**, en las ideas acerca de la definición de módulos y en los conceptos sobre el impacto de localidad de organización de programas. Las técnicas de especificación correspondientes, incluyen ***verificación fuerte de tipos*** y ***verificación de aseveraciones acerca de la corrección funcional.***

En los mismos años, mucha actividad de investigación en técnicas de abstracción se orientó al lenguaje y los conceptos de especificación fueron ampliamente cubiertos, mucha de esa actividad se identifica con el concepto de ***tipos de datos abstractos***. Como en la programación estructurada, la metodología de tipos de datos abstractos enfatiza la situación de colecciones de información relacionadas. En este caso, la atención se enfoca en los datos más que en el control, y la estrategia es formar módulos consistentes de una estructura de datos y sus operaciones asociadas. El objetivo es tratar estos módulos de la misma manera como son manejados los tipos ordinarios, lo anterior, requiere de soporte para declaraciones, operadores prefijos, especificación de parámetros para las rutinas, etc. El resultado, denominado un tipo de datos abstractos, efectivamente extiende el conjunto de tipos de datos disponibles para un programa, explica las propiedades de un nuevo grupo de variables a través de la especificación de los valores que cada variable puede tener y determina las operaciones que son permitidas sobre las variables del nuevo tipo, al explicar los efectos que estas operaciones tienen sobre los valores de las variables.

En un tipo de datos abstractos, se especifican las propiedades funcionales de una estructura de datos y sus operaciones, implementándose posteriormente en términos de constructores de lenguajes existentes y otros tipos de datos y se muestra que la especificación es adecuada. Como resultado del proceso anterior, cuando se hace uso de la abstracción, se trata con el nuevo tipo solo en términos de su especificación, esta filosofía fue desarrollada en varias investigaciones recientes de lenguajes como **Ada, Alphard, CLU, Pascal Concurrente, Euclid, Gypsy, Mesa y Modula.**

Las técnicas de especificación utilizadas para tipos de datos abstractos, vienen de los predicados utilizados en programas secuenciales simples. El poder expresivo adicional fue incorporado para tratar con la información empacada en módulos y con el problema de abstraer de una implementación a un tipo de datos. Una clase de las técnicas de especificación trabaja sobre las similitudes entre un tipo de datos y la estructura matemática denominada *álgebra*. Otra clase de técnicas modela explícitamente un tipo de datos por medio de la definición de sus propiedades en términos de las propiedades comunes.

En conjunto con el trabajo sobre tipos de datos abstractos y especificación formal, las definiciones genéricas originadas en los lenguajes extendibles, han sido desarrolladas a un nivel mucho más alto de expresividad y precisión que sus antecesores. Estas definiciones no solo están parametrizadas en términos de variables que puedan ser manipuladas durante la ejecución del programa. también, se pueden parametrizar en términos de tipos de datos, asimismo, pueden describir restricciones para los tipos de datos que son aceptables como parámetros.

1.6. INTERACCION ENTRE LA ABSTRACCION Y LAS TECNICAS DE ESPECIFICACION.

Como se ha visto hasta ahora, los lenguajes de programación y las metodologías evolucionan en respuesta a las necesidades de los diseñadores e implementadores de software. Sin embargo, esas necesidades a su vez evolucionan en respuesta a la experiencia ganada de soluciones pasadas. Las primeras técnicas de abstracción de la programación estructurada fueron procedimientos o macros, las cuales han evolucionado a tipos de datos abstractos y definiciones genéricas. Las metodologías para el desarrollo de programas emergen cuando se encuentran patrones útiles y comunes y se intenta utilizarlos como modelos. Los lenguajes evolucionan para soportar esas metodologías cuando los modelos llegan a ser estándares. Como las técnicas de abstracción han sido capaces de orientarse a un intervalo más grande de tipos de organización de programas, las técnicas de especificación formal han llegado a ser más precisas y han jugado un papel crucial en el proceso de la programación.

Para que una abstracción sea efectiva, su especificación debe expresar toda la información necesaria para el programador que la utiliza. Los intentos iniciales de especificación usaban la notación del lenguaje de programación para expresar cosas que pudieran ser verificadas por el compilador, por ejemplo, el nombre de una rutina y el número de parámetros y sus respectivos tipos. Otros hechos como la descripción de las condiciones bajo las cuales la rutina debía ser utilizada, eran expresadas informalmente.

Se ha progresado al punto de poder escribir descripciones precisas de muchas relaciones importantes entre rutinas, incluyendo sus supuestos acerca de los valores de las entradas y el efecto que estas puedan tener sobre el estado del programa. Sin embargo, muchas otras propiedades de las abstracciones se especifican informalmente todavía, como consumo de tiempo y espacio, interacciones con dispositivos de propósito especial, comportamiento agregado complejo y aspectos de procesamiento concurrente. Es razonable esperar que los desarrollos futuros en técnicas de especificación formal y lenguajes de programación respondan a los tópicos mencionados.

La historia de la programación, muestra que existe un balance entre las ideas de los lenguajes y las técnicas formales, en cada metodología las propiedades especificadas son asociadas a la capacidad de validar (verificar) la consistencia de una especificación y su implementación, entonces, puesto que se puede contar con especificaciones formales de las extensiones que se asocien con sus implementaciones, el desarrollo de las técnicas de abstracción, especificación e implementación debe llevarse a cabo concordantemente. En el futuro se debe esperar una mayor diversidad en los programas que se utilicen como base de la modularización, y deben esperarse especificaciones que se vean involucradas con aspectos de los programas, no necesariamente funcionales como los que se consideran hoy en día.

I.7. BASES DE DATOS E INTELIGENCIA ARTIFICIAL.

En el trabajo realizado hasta ahora, el cual es relativo a las técnicas de modelado conceptual en las áreas de lenguajes de programación, bases de datos e inteligencia artificial, ha resaltado la gran cantidad de conceptos comunes entre dichas áreas, asimismo, ha dejado entrever algunas diferencias que dichas técnicas tienen en cuanto la aplicación se realiza.

Muchas de las diferencias filosóficas entre las tres áreas mencionadas, pueden ser diferencias de énfasis. Por ejemplo, el compartir información y el dominio de los datos sobre la computación, recibe mucha más atención en la inteligencia artificial y sobre todo en el ámbito de las bases de datos que en los lenguajes de programación. La computación implícita realizada como un efecto colateral de acceder la información sigue el mismo patrón. Por otro lado, las técnicas formales, en particular la especificación, juegan un papel mucho más extenso en los lenguajes de programación que en las otras dos áreas. También hay diferencias en el grado de regularidad que se asume para la estructura de los datos ordinarios, mientras en la inteligencia artificial aparecen estructuras heterogéneas irregulares, casi no existen en los lenguajes de programación y mucho menos en el contexto de las bases de datos.

Estas diferencias de énfasis entre las áreas, resaltan un número de tópicos que requieren de mayor atención por parte de la comunidad de los lenguajes de programación. Es importante encontrar formas de especificar y controlar aspectos como los siguientes:

. Computación implícita como un efecto colateral del acceso y modificación de datos.

. Información con periodos de vida largos, en particular, datos que sobreviven a los programas que los manipulan.

. Sistemas incompletos, sistemas inconsistentes o parcialmente correctos.

. Compartir datos, particularmente cuando varios procesos actualizan una estructura.

. Tipos de sistemas más ricos, por ejemplo no jerárquicos, sin agregar costo o complejidad excesivos a los lenguajes de programación.

II. *DESARROLLO DE LOS MODELOS DE DATOS.*

El alcance de este capítulo, se centra básicamente en la capacidad de representar modelos reales en un sistema de bases de datos.

Si se considera una organización, la cual tiene una cantidad considerable de información almacenada en una computadora, es importante resaltar los diversos usos que a dicha información se le pueden dar. Asimismo, destacar las relaciones de interés que pueden establecerse con los datos.

A la información que se almacena con cierta permanencia en una computadora se denomina ***base de datos***. El software que permite a los usuarios de la información hacer un uso adecuado de ella se le llama un ***sistema manejador de bases de datos***.

Una tarea fundamental del sistema de bases de datos, es la de facilitar el uso de la información en un *esquema abstracto*, es decir, permitir al usuario desentenderse de la forma en que la computadora almacena los datos. En un sentido estricto, el sistema manejador de bases de datos hace las veces de un interprete, que en forma ideal facilita al usuario especificar ***qué*** debe hacerse con la información, en lugar de establecer ***cómo*** realizar las acciones sobre los datos.

Los sistemas manejadores de bases de datos, son de los casos más complejos del software, una de las mejores formas de establecer sus características más importantes, es considerar los diversos aspectos que el sistema puede ofrecer a los diferentes ***tipos de usuarios*** que existen dentro de las organizaciones.

En el estudio global de los modelos de datos y su desarrollo, sobresale el problema básico del modelado de datos en el contexto de las bases de datos, sustentado en los conceptos de modelos de datos y su terminología. Es de suma importancia la realización de una especie de ***taxonomía*** de los modelos, a fin de introducirse en los avances de la investigación relativa a los mismos.

II.1. MODELOS DE DATOS Y LAS APLICACIONES DE DATOS.

Los modelos de datos son la parte central de los sistemas de información, la amplia gama de sistemas de información se basan en algún modelo de datos.

Una de las tareas más importantes en el desarrollo de una base de datos, es aquella que involucra la definición de un ***modelo conceptual***, también denominado modelo de la organización. Dicho modelo considera las entidades y sus relaciones y es el medio que se utiliza para representar la ***organización conceptual de los datos***.

El modelo conceptual, por lo tanto, es una herramienta que coadyuva a la comunicación entre los diversos usuarios de la información, y debe ser desarrollado sin consideración alguna de representaciones físicas, por último el modelo de datos se utiliza para organizar, visualizar, planear y comunicar. Debe ser independiente de los sistemas manejadores de bases de datos.

El modelo conceptual tiene que ser mapeado al modelo lógico que sustenta el manejador de base de datos que se desea utilizar, el modelo lógico, a su vez, tiene que confrontarse con el modelo interno o físico.

La esencia del análisis para bases de datos, es idear o conceptualizar estructuras de datos que puedan sustentar **cualquier** aplicación. Esto incluye aplicaciones futuras que requieran de la información existente. La estructura debe permitir también, agregar nuevos tipos de datos. En el análisis, la información en uso o necesaria debe ser identificada y clasificada de manera que vaya más allá de las políticas, prácticas, prioridades y aplicaciones actuales. El objetivo es documentar y modelar el tipo de cosas con las que la organización trata, esto da un punto común de referencia para la organización completa denominado ***modelo conceptual de datos***.

Los modelos de datos proveen las bases conceptuales para el proceso de las aplicaciones intensivas de datos, y dotan de una base formal para las herramientas que se usan en el desarrollo y utilización de los sistemas de información. A lo largo de los años, estas herramientas se han desarrollado no sólo en número, también, hay avances significativos en la utilización de ellas en otros tipos de aplicaciones. Tanto en el ámbito de los sistemas de información tradicionales, como en los nuevos tipos de sistemas (p.e. sistemas de ingeniería, sistemas de oficina, etc.) existe ahora un uso intensivo de las bases de datos.

El modelado de datos, con respecto al diseño de bases de datos, puede describirse como sigue:

Dada la información y las necesidades de proceso de una aplicación de datos (p.e. un sistema de información), se construye una representación de la aplicación, la cual captura las propiedades necesarias para soportar el proceso deseado (p.e. consultas, reportes, etc.). Adicionalmente a la consideración de las necesidades de información en el momento del diseño, la representación debe ser capaz de contemplar las necesidades futuras o cambiantes.

Un ***modelo de datos*** es una colección de conceptos matemáticos bien definidos, que auxilian en la consideración y expresión de las propiedades de las aplicaciones de datos.

La descripción anterior del concepto de modelo de datos es muy idealista, muchos de los modelos de datos que se utilizan son desarrollados de manera intuitiva y no han sido definidos formalmente.

En el desarrollo del modelo de datos, las características de los datos que deben considerarse son:

- ***Propiedades estáticas***. Son los objetos (**entidades**), las propiedades de los objetos (**atributos**), y las **relaciones** entre objetos (se pueden considerar estas como un tipo especial de propiedades de los objetos).

- ***Propiedades dinámicas***. Son las **operaciones** sobre los objetos, operaciones sobre las propiedades de los objetos y las operaciones sobre las relaciones entre objetos (llamadas también ***transacciones***).

- ***Reglas de integridad***. Son criterios sobre los objetos (**estados de la base de datos**) y sobre las operaciones (**estados de las transacciones**).

Por lo tanto, todo concepto que constituye un modelo de datos, se ubica en alguna de las categorías mencionadas.

El resultado del modelado de datos, es una representación que tiene dos componentes:

- Un *esquema* que contiene las propiedades estáticas.

- Un *grupo de especificaciones* para transacciones, peticiones y reportes que definen las propiedades dinámicas.

El esquema consiste de una definición de todos los tipos de objetos involucrados en la aplicación, incluyendo sus atributos, relaciones y restricciones estáticas.

Correspondiendo al esquema, habrá un concepto llamado *base de datos*, el cual es una instancia del esquema, es decir, una colección de instancias de los tipos de objetos definidos en el esquema.

Cierto tipo de procesos dentro de una aplicación necesita sólo de un subconjunto predeterminado de los objetos definidos en el esquema, más aún, esos procesos pueden requerir únicamente de un subconjunto de las propiedades estáticas. Por lo anterior, se diseña un subesquema para cada clase de procesos que tengan necesidades similares de información. Un *subesquema*, es algún subconjunto de las propiedades estáticas definidas en el esquema o derivadas del mismo.

Una *transacción* consiste de varias operaciones o acciones sobre los objetos de un esquema o subesquema. Una transacción se usa para definir las necesidades de la aplicación. Una transacción, siempre es **atómica** en el siguiente contexto; si una de las operaciones de la transacción falla, entonces toda la transacción falla y quedará sin efecto sobre la base de datos.

Una *petición* puede quedar determinada como una expresión lógica sobre los objetos y relaciones definidas en el esquema. Los resultados de una petición, se identifican como un *subconjunto lógico* de la base de datos.

Un modelo de datos debe proveer de las bases formales de notación y semántica para las herramientas y técnicas que se usan en el modelado de datos. Las herramientas asociadas con los modelos de datos, son lenguajes para la definición, manipulación, gestión y apoyo para el mantenimiento de la base de datos.

Muchos de los sistemas manejadores de bases de datos de hoy en día, proveen de un *lenguaje de definición* para el establecimiento de esquemas y subesquemas, un *lenguaje de manipulación* para escribir programas que actualicen la base de datos y de un *lenguaje de petición* para realizar peticiones o gestiones a la base de datos. La partición de esas funciones en tres lenguajes no es necesaria, muchos lenguajes de bases de datos combinan manipulación y petición, algunos otros proveen las tres funciones en un solo lenguaje.

De lo anterior se desprende que un *sistema de bases de datos* es un sistema que implementa las herramientas asociadas con el modelo de datos. Generalmente, los modelos de datos proveen *primitivas* para definir, manipular y consultar bases de datos, dichas primitivas pueden usarse como un lenguaje de bajo nivel, las cuales se pueden incluir en algún lenguaje anfitrión. Sin embargo, también pueden diseñarse de manera que conformen un lenguaje de programación de alto nivel.

Con el objeto de proveer formalización al diseño de propiedades dinámicas, los modelos de datos se han extendido e integrado con conceptos de lenguajes de programación e inteligencia artificial; sin embargo, los nuevos conceptos no se integran directamente, puesto que las aplicaciones típicas de bases de datos difieren de las aplicaciones de las disciplinas mencionadas, ya que las bases de datos, generalmente se usan para soportar la consulta y actualización de grandes volumenes de información y sirven a una gran comunidad de usuarios y a un número importante de aplicaciones distintas. Por lo anterior, es necesario contar en este ámbito con lo siguiente:

. Un modelo lógico global de los datos (*esquema lógico*).

. Una interface hombre-máquina de alto nivel.

. Una descripción de las estructuras de datos almacenadas (*esquema físico*).

. Vistas particulares para las necesidades específicas de ciertos usuarios y procesos.

. Mecanismos que faciliten el compartir datos entre tipos de usuarios y procesos.

Debido al tamaño y la longevidad de las bases de datos, estas deben ser capaces de minimizar los efectos dañinos del cambio.

Como resultado de las interrelaciones estáticas de objetos y de que una aplicación puede involucrar el acceso y modificación a muchos objetos, el concepto de *transacción atómica*, es esencial para mantener la integridad de la base de datos con respecto a las operaciones.

Las investigaciones en modelos de datos, se han enfocado tanto a los *aspectos conceptuales* de las bases de datos, como a los conceptos, herramientas y técnicas para el diseño de dichos aspectos. Sin embargo, es válido aseverar, que las investigaciones más recientes relativas a los modelos de datos, se han orientado a los aspectos de *especificación* de bases de datos, dejando en un segundo plano los aspectos de *implementación*, como son; **rendimiento, concurrencia, integridad física y arquitectura de sistemas.**

Las primeras investigaciones, fueron más cercanas a los aspectos de representación física, pero en los últimos años los aspectos lógicos han sido fuertemente considerados. El trabajo técnico ha sido desarrollado básicamente en las siguientes áreas:

- *Definición semántica* de los modelos de datos.

- *Técnicas de diseño* de bases de datos basadas en el modelo relacional.

- *Especificación y verificación* de esquemas y transacciones.

- *Análisis y optimización de lenguajes* de petición y manipulación.

- *Aplicación de la lógica* para la solución de los problemas tradicionales de las bases de datos.

II.2. CONCEPTOS BASICOS DE MODELOS DE DATOS.

Muchos de los modelos de datos contemplan los conceptos básicos de objetos, operaciones y restricciones, sin embargo, difieren en los formalismos con que se representan y en los detalles asociados a estos conceptos básicos, la importancia de dichos conceptos para una discusión estriba que estos forman una estructura con la cual es posible describir y comparar los diversos modelos de datos.

De los conceptos básicos; ***objetos*** y ***operaciones*** son términos indefinidos o sumamente obvios, por el momento es más importante discutir el concepto de ***integridad semántica*** en las restricciones a ser aplicadas, tanto a objetos como a las operaciones, ya que dicho concepto provee intuición para analizar las diferencias de los modelos.

Restricciones. Son reglas que se usan para definir propiedades estáticas y dinámicas de las aplicaciones, las cuales en general, no son expresadas de manera conveniente a través de las utilerías de un sistema manejador de bases de datos.

La definición de un modelo de datos independiente a la representación física, se hace a través de una ***restricción de integridad semántica***, la cual es una representación de cualquier propiedad de una aplicación.

Cada modelo de datos, provee diferentes formas para la representación de restricciones.

Existen tres tipos de restricciones; ***Inherentes***, ***Explícitas*** e ***Implícitas***, las cuales se describen a continuación:

Algunas propiedades pueden representarse directamente y no pueden ser evitadas en un modelo de datos, estas son las relativas al modelo y se denominan ***restricciones inherentes***, constituyen las propiedades semánticas básicas de un modelo de datos, es decir, son reglas que no es posible violar, por ejemplo, las relaciones entre objetos en el modelo jerárquico deben ser jerárquicas, las n-eadas en el modelo relacional deben ser únicas dentro de una relación.

Otro tipo de propiedades, pueden definirse utilizando alguna combinación de los mecanismos que provee el modelo de datos, a estas restricciones se les denomina ***relaciones explícitas***, su uso es básico en el diseño de la base de datos. Ejemplos de restricciones explícitas son; los nombres, estructuras de los objetos, especificación de relaciones y condiciones, es decir, el orden lógico sobre los objetos y sus propiedades.

Finalmente, hay propiedades que son consecuencia lógica de otras restricciones, estas son conocidas como ***restricciones implícitas***. Estas resultan de la interacción de otras restricciones, como ejemplo, las inferencias lógicas se pueden deducir de un conjunto de condiciones y un conjunto completo de dependencias funcionales.

Los modelos de datos difieren principalmente en las diversas clasificaciones que pueden tener sus restricciones en los grupos señalados, es decir, se diferencían a través de la determinación de cuales de sus restricciones son inherentes, explícitas o implícitas. Los modelos son definidos en términos de sus restricciones inherentes y explícitas, sin una ***definición formal de las semánticas*** de un modelo es difícil identificar o analizar las restricciones implícitas. Lamentablemente, muchos de los modelos se han propuesto sin una definición precisa de sus semánticas.

Los conceptos para modelar propiedades estáticas incluyen ***objetos***, ***atributos*** y ***relaciones***.

Los objetos pueden ser simples o compuestos, los ***objetos simples*** son irreductibles, es decir, no es posible descomponerlos en otros objetos y son capaces de existir independientemente, en el modelo relacional se denominan ***relaciones base***, en otros modelos, como son los basados en registros se representan por registros sencillos. La elección de que objetos son simples es una decisión en el proceso de diseño.

Los ***objetos compuestos*** están constituidos de dos o más objetos. Utilizando el modelo relacional, un objeto compuesto puede representarse por una relación derivada de dos o más relaciones. En los modelos jerárquico y de red, un objeto compuesto puede representarse por un registro y una jerarquía o red asociada de registros.

Los objetos también pueden estar restringidos, por ejemplo, pueden restringirse a una existencia única en la base de datos, asegurando que sólo un objeto de un tipo existe y que cada objeto es identificado únicamente por una llave no nula, este tipo de restricción es denominada ***integridad de entidad***.

Un ***atributo*** describe una propiedad estática simple de un objeto y no tiene existencia independiente del objeto, tienen un valor contenido en un rango definido de acuerdo a la naturaleza del mismo atributo.

La selección de los atributos de una entidad, es otro de los puntos cruciales en el diseño del modelo del mundo real.

Algunos modelos avanzados, evitan el concepto de atributo simple, y sólo lo consideran para caracterizar objetos compuestos por medio de relaciones.

Algunos de los modelos de datos más avanzados, utilizan para representar atributos, un concepto denominado ***Tipos de Datos***.

La facilidad para representar ***relaciones*** ha sido objetivo esencial en el desarrollo de los modelos de datos. Los modelos primitivos no soportan relaciones explícitamente, los modelos jerárquico y de red, proveen medios para representar relaciones ***uno a uno***, ***uno a varios*** y ***varios a varios***, estas últimas de manera restringida. El modelo relacional tiene forma de representar relaciones ***varios a varios*** de forma generalizada.

Los modelos de datos más avanzados, influenciados por las ***redes semánticas***, intentan modelar aplicaciones orientadas a relaciones. Algunos modelos de datos, proveen un conjunto fijo de tipos de relación, con el cual, todas las relaciones de la aplicación se ven representadas. Otro grupo de modelos, provee mecanismos generalizados para representar cualquier tipo de relación en las aplicaciones. Finalmente, algunos modelos de datos no forzan una distinción entre objetos y relaciones, algunos otros sí. Cuando la distinción no se forza se dice que el modelo soporta ***relativismo semántico***.

Motivado por el modelo relacional, existe un interés en las relaciones entre atributos, una de las relaciones consideradas como básica es la ***dependencia funcional***.

Las restricciones más importantes entre relaciones, son los ***mapeos***. Las restricciones más avanzadas de relaciones, son las relativas a la existencia de objetos y la dependencia de un objeto sobre otros objetos, por ejemplo, la existencia de un objeto compuesto, se da si existen los objetos que lo constituyen, esta restricción es denominada ***integridad referencial***, la cual inicialmente fue una restricción inherente de los modelos jerárquico y de red, posteriormente se agregó como restricción explícita en el modelo relacional.

Los conceptos de propiedades dinámicas, incluyen ***operaciones simples*** y ***compuestas***, ***relaciones de operaciones*** (estructuras de control), ***restricciones dinámicas*** y ***manejo de excepciones***.

Todos los modelos de datos tienen operaciones simples sobre objetos (inserción, modificación y eliminación). Estas operaciones primitivas, se dice que forman parte del lenguaje ensamblador de un sistema de base de datos.

Muy rara vez, una operación simple es adecuada para ejecutar una operación orientada a aplicaciones. Los primeros modelos no tenían manera de soportar grupos de operaciones llamadas ***transacciones***, las cuales están compuestas de acciones elementales sobre objetos individuales.

Para representar transacciones y acciones, entre los avances recientes se pueden mencionar los conceptos de lenguajes de programación, como son; ***abstracciones de procedimiento***, ***tipos de datos abstractos***, ***módulos*** y ***clases***.

Las ***relaciones entre operaciones***, son las estructuras de control necesarias para representar flujo de control, y están fuertemente influenciadas por la estructura estática de los objetos.

Las ***restricciones dinámicas***, son reglas que aseguran la ejecución apropiada de las operaciones. Los primeros modelos usan condiciones de error y disparadores, pero son ineficientes para bases de datos muy grandes. Las investigaciones recientes, consideran mecanismos de manejo de excepciones originados en las investigaciones de lenguajes de programación e inteligencia artificial.

La manera más generalizada de representar restricciones dinámicas, son las ***precondiciones*** y ***postcondiciones*** que se ponen alrededor de acciones y transacciones.

Precondiciones. Se utilizan para verificar que la base de datos se encuentra en un estado aceptable antes de que la operación proceda.

Postcondiciones. Aseguran que todo haya ocurrido correctamente en la operación.

El objetivo principal de los conceptos arriba mencionados, es soportar comunicación entre las acciones que constituyen una transacción. Cuando hay violaciones a las restricciones dinámicas, es necesario el ***manejo de excepciones***.

Una de las utilerías básicas de un modelo de datos, es la facilidad para establecer peticiones sobre aspectos estáticos de una base de datos. Los primeros modelos dotan de operaciones basadas en registros como la navegación sobre jerarquías y redes. El modelo relacional, introdujo un álgebra y un cálculo de alto nivel para consultar bases de datos relacionales. Muchos de los lenguajes de petición de alto nivel, se basan en la lógica de primer orden y en la teoría de conjuntos.

La siguiente tabla esquematiza los conceptos básicos de los modelos de datos que se han considerado en esta sección.

1. *CONCEPTOS.* a. Propiedades estáticas. Objetos simples Objetos compuestos Restricciones a los objetos Relaciones entre objetos Atributos de un objeto Restricciones a los atributos Relaciones entre atributos Restriciones a las relaciones b. Propiedades dinámicas Operación Simple Operación compuesta Relaciones entre operaciones Manejo de excepciones c. Restricciones y facilidades para las peticiones
2. *HERRAMIENTAS.* Lenguajes de definición Lenguajes de manipulación Lenguajes de petición Tipos de datos Variables Diseño automatizado
3. *TECNICAS.* Diseño de esquemas Diseño de subesquemas Diseño de transacciones Diseño de peticiones Conceptos de objeto y clase

II.3. TAXONOMIA DE LOS MODELOS DE DATOS.

En este ámbito se reconoce la existencia de cuatro generaciones de modelos de datos, la última de ellas se encuentra actualmente en desarrollo, la clasificación más generalizada de dichas generaciones es como sigue:

-Modelos primitivos de datos.
-Modelos clásicos de datos.
-Modelos semánticos de datos.
-Modelos de propósito especial de datos (orientados a aplicaciones).

Las primeras generaciones de modelos, resaltaban las propiedades estáticas de las aplicaciones, por lo tanto, los primeros modelos proveen conceptos para restricciones estáticas y las peticiones se caracterizan por contener sólo conceptos primitivos para las restricciones dinámicas.

Los modelos de datos, pueden clasificarse por los conceptos que utilizan para representar objetos y relaciones.

II.3.1. MODELOS PRIMITIVOS DE DATOS.

Los objetos se representan en registros que se agrupan en archivos.

Las relaciones entre los objetos pueden representarse usando directorios. Por ejemplo, índices y listas invertidas.

Las operaciones que se proveen son lecturas y escrituras primitivas sobre los registros. Un caso raro en esta generación, es uno de los primeros modelos jerárquicos (contenía registros maestros y registros detalle), el cual se utilizó sobre cintas antes de que las jerarquías fueran implementadas sobre dispositivos de acceso directo.

II.3.2. MODELOS CLASICOS DE DATOS.

Los modelos clásicos son el ***jerárquico***, el cual es una extensión directa del modelo de primitivo de archivos, la ***red*** constituido por un superconjunto del modelo jerárquico y el ***modelo relacional de datos***, este último, considerado como un despegue importante de los modelos anteriores, en los aspectos conceptual y tecnológico.

La jerarquía y la red son modelos de datos que representan objetos en segmentos de registro organizados, usando relaciones binarias uno a varios, como nodos de árboles y redes, respectivamente. Estos modelos de datos, proveen únicamente operaciones y facilidades primitivas de navegación de un registro cada vez.

A diferencia de los lenguajes relacionales, los lenguajes jerárquicos y de red son más ***cercanos a la representación***, es decir, los usuarios deben tratar más con las utilerías de almacenaje e implementación, y es importante notar que cuentan con un conjunto de restricciones inherentes más amplio.

El modelo relacional de datos, se basa en el concepto matemático de relación. Una ***relación*** es un conjunto de elementos denominados n-eadas. Una ***n-eada*** puede utilizarse para representar tanto objetos como relaciones n-earias varios a varios. El álgebra relacional y el cálculo relacional, proveen de un conjunto de facilidades para acceder la información y pueden utilizarse también, para establecer restricciones.

Muchos de los modelos actuales, usan un modelo de datos primitivo como sistema de archivos, y un sistema manejador de bases de datos se utiliza para soportar algún híbrido de los modelos clásicos. En realidad no hay sistemas manejadores de bases de datos que implementen alguno de los modelos mencionados en ***forma pura***, es decir, se han adoptado de los diferentes modelos, las lógicas más adecuadas a la aplicaciones y al tipo de interacciones con el usuario.

Los sistemas manejadores de bases de datos más difundidos, son básicamente los jerárquicos y de red.

El concepto de modelo de datos, fue desarrollado al mismo tiempo que se propuso el modelo relacional, lo cual ocurrió después de que los modelos jerárquico y de red estaban en uso.

Los modelos jerárquico y de red, se definieron de manera independiente de los lenguajes y sistemas que se usaron para su implementación, por lo que se puede aseverar, que son colecciones de estructuras de datos y lenguajes, sin una teoría definida que los soporte.

Los sistemas manejadores de bases relacionales de datos se han difundido rápidamente, llegando a ser comerciales.

A continuación se realiza una breve descripción de los tres modelos de datos clásicos, los cuales se han usado en la mayoría de los sistemas manejadores de bases de datos, en estos tres casos, es posible de manera relativamente sencilla generar una organización física razonable de bases de datos, a partir de una definición de la descripción de la base de datos que utilice dichos modelos.

II.3.2.1. EL MODELO DE RED.

La influencia dominante en el desarrollo del modelo de datos de red y de los manejadores de bases de datos que usan este modelo, ha sido el conjunto de propuestas establecidas por el Grupo de Tarea de Bases de datos (**DBTG** por sus siglas en ingles) comisión perteneciente a la Conferencia de Lenguajes de Sistemas de Datos (**CODASYL**) **[CODASYL 71], [CODASYL 73]**. En resumen, la propuesta consiste de una notación formal para redes (lenguaje de definición de datos), de un lenguaje de definición de subesquemas de datos y por último de un lenguaje de petición de datos para el desarrollo de aplicaciones que manipulen el esquema conceptual.

La estructura de la información es el modelo de red de datos que está sujeto a relaciones ***varios a uno*** en ambas direcciones, esta restricción permite la utilización de un modelo orientado de datos, lo que hace muy sencillo llevar a cabo bases de red de datos.

En el modelo de red, el término ***registro lógico*** sustituye a n-eada y el término ***formato de registro lógico*** al de esquema de relación. Asimismo, a los nombres de los componentes se les denomina ***campos***.

En el modelo de red, los tipos de registro lógico son usados para representar conjuntos de entidades, para implementar relaciones ***varios a varios*** se utiliza el concepto denominado ***ligas***. Si dibujamos una gráfica para representar tipos de registros y sus ligas, obtenemos una red, en donde los nodos son tipos de registro y los arcos son las ligas.

Hay dos tipos de operaciones en el ámbito del modelo de red, una consistente en realizar ***selecciones*** de tipos de registros, la otra que hace el seguimiento de las ligas en una dirección y en otra, la cual se conoce como ***navegación***, una cuestión importante en este punto, consiste en observar que la navegación en el modelo relacional puede ser llevada a cabo a través del procedimiento que se establecerá más adelante denominado ***unión***, y que tiene la característica de que la navegación en el modelo relacional puede ser realizada de ***varios a varios***, de manera tal que los lenguajes de petición pueden ser más poderosos bajo ese modelo y además tienen la ventaja de ser más cercanos al nivel de abstracción del usuario, sin embargo, el implementar dichos lenguajes de petición es mucho más difícil que establecer uno bajo los lineamentos de la navegación en el modelo de red.

A los registros lógicos también se les denomina en el ámbito de red como ***registros tipo***. Los campos en un formato de registro lógico son llamados ***datos item***,y a los formatos de registro lógico se les conoce como ***registros***.

La base de datos puede contener muchas ocurrencias de registros del mismo tipo. No hay restricción en el sentido de que dichas ocurrencias del mismo tipo sean distinguibles claramente, los tipos de registros sin campos son posibles y suelen ser utilizados para conectar registros de otros tipos.

El concepto de liga se ve claramente representado por un mapeo varios a uno de un registro a otro y se conoce como ***conjunto***.

Cuando se tiene un mapeo varios a uno M de registros del tipo R_2 a registros del tipo R_1, se pueden asociar con cada registro r del tipo R_1 el conjunto S_R, el cual consiste de aquellos registros s del tipo R_2 de manera tal que $M(s) = r$.

Una vez que M es varios a uno, los conjuntos S_{r1} y S_{r2} son ajenos si $r_1 <> r_2$. Si CONJ es el nombre del conjunto que representa la liga M, entonces cada conjunto S_r junto con r es una ocurrencia de CONJ.

El registro r es el *propietario* del conjunto ocurrencia, el tipo R_1 es conocido como el *tipo propietario* de CONJ y R_2 es el *tipo miembro* de CONJ. Una restricción fuerte en este ámbito es la que determina que los tipo miembro y propietario son distintos, esta restricción es de importancia ya que muchas operaciones con el modelo suponen la mencionada diferenciación.

El modelo determina un lenguaje de definición de subesquemas de datos, con el cual, se pueden definir vistas de la información, en dichas vistas es posible usar nombres diferentes para cualquier registro tipo, campo o conjunto.

La definición de vistas no contiene conceptos no definidos con el lenguaje de definición de datos para el esquema conceptual, por lo que las vistas interactúan directamente sobre el mencionado esquema conceptual.

Es posible representar los registros lógicos de un registro tipo dado, por medio de un archivo de manera obvia, es decir, con un campo por cada registro por cada campo del registro lógico. Si un formato de registro lógico consiste únicamente de campos virtuales o de conexión su registro, no necesariamente debe desaparecer, es más, la presencia de esos registros seguramente influirá en la organización de la base de datos, no importando cual es la organización que se elija para la red.

Hay diversas maneras para representar ligas por las cuales se pueda navegar eficientemente de propietarios a miembros y viceversa. Si se supone que se tiene una liga del tipo T_1 al tipo T_2, dado que la liga es varios a uno de T_1 a T_2, es posible representarla con un registro de longitud variable de formato $T_2(T_1)$, es decir, después de cada registro de tipo T_2, hay una lista de todos los tipos T_1 asociados. Si existe una otra liga del registro T_3 a T_2, es posible listar las ocurrencias de T_3 con los registros T_2, correspondientes, por medio de un registro de longitud variable $T_2(T_1)\ (T_3)$.

Por otro lado, si se supone hay otra liga de T_1 a algún tipo T_4, no es posible listar los registros T_1 después de los registros T_2 y listar a continuación los registros T_4. Si se duplican los registros T_1 y se colocan después de los registros T_2 y T_4, se introduce la redundancia y la potencial inconsistencia. Por lo tanto, es necesaria una forma de representar las ligas de manera que no requiera que los registros de un tipo sean adyacentes a los registros del otro. En esta organización a la cual se le denomina ***multilista***, cada registro tiene un apuntador por cada lista en la cual se ve involucrado.

Si se supone que se tiene un liga L de T_1 a T_2, por cada registro r de tipo T_2 se crea una cadena circular comenzando en r y de ahí a todos los registros $r_1,r_2,...,r_k$ de tipo T_1 asociados a r por medio de L, regresando finalmente a r.

Es importante notar, que en una organización multilista cada registro tiene tantos apuntadores como registros tipos tiene asociados. Como los apuntadores son campos en los registros, entonces aparecen en posiciones fijas y es posible seguir la cadena para una liga particular, también es necesario recordar que dado que las ligas son varios a uno, cada cadena circular tiene exactamente un registro del tipo uno y cero o más registros del segundo tipo. Notese que si se intenta representar relaciones varios a varios por medio de multilistas, se desprenden varios problemas, ya que cada registro podría estar en varias cadenas de la misma relación.

No se puede saber a ciencia cierta cuantos apuntadores se requerirían para un registro para cada liga y no se puede determinar qué liga es para cada cadena. Este problema aún no se ha resuelto convenientemente y justifica el porqué el modelo de red restringe a relaciones varios a uno.

Como se ha mencionado, los tipos de registro lógicos se llevan a cabo por archivos de registros, un registro de un archivo por cada registro lógico. Este archivo está organizado en alguna de varias maneras llamadas ***modos de colocación***.

Los modos de colocación que destacan son el denominado **CALC** y el modo de colocación **DIRECT**, este último declara que los registros de un tipo se encuentran sólo por su dirección en el sistema de archivos, las direcciones son denominadas ***llaves de la base de datos***.

Dentro de las ventajas del modelo de red se pueden mencionar las siguientes:

- Existe una gran cantidad de sistemas manejadores de bases de datos exitosos en el mercado, los cuales utilizan el modelo de red como estructura básica.

- Las relaciones varios a varios las cuales ocurren con mucha frecuencia en la vida real, se pueden implementar fácilmente con este modelo.

- El hecho de estar basado en un estándar, influyó en los manejadores de bases de datos subsecuentes.

Por otra parte, la principal desventaja del modelo de red es su complejidad, el usuario debe estar familiarizado con la estructura lógica de la base de datos.

Una complicación adicional, es aquella que se deriva de la reorganización de la base de datos, ya que es posible la pérdida de independencia de datos a causa de que los conjuntos de ocurrencias se eliminan.

II.3.2.2. EL MODELO JERARQUICO.

Es muy fácil entender qué es una jerarquía, porque es común tratar con ellas diariamente. Casi toda organización tiene un jefe con un número de empleados reportándole.

Si se considera un árbol genealógico en el cual los padres tienen ningún, uno o más de un hijo, si tienen hijos los hijos pueden tener sus propios hijos, el árbol familiar tiene una estructura jerárquica si ignoramos un padre en cada uno.

Una estructura de árbol jerárquico se constituye de ***nodos*** y ***ramas***. Un nodo es una colección de atributos de datos describiendo la entidad en dicho nodo. El nodo más alto de una estructura de árbol jerárquico es llamado ***raíz***, los nodos dependientes están en niveles más bajos en el árbol. El nivel de esos nodos depende de la distancia entre ellos y el nodo raíz.

Un modelo jerárquico de datos es un modelo que organiza los datos en una estructura de árbol jerárquico. Cada ocurrencia del nodo raíz, comienza con un registro lógico de la base de datos, esto es, una base jerárquica de datos está hecha de un número determinado de arboles.

En un modelo jerárquico de datos, los nodos del nivel 2 son llamados los hijos del nivel 1, y el nodo del nivel 1 es llamado el padre de los nodos del nivel 2. Los nodos del nivel 3 por su correspondencia a los nodos del nivel 2 son llamados los hijos del nodo al nivel 2, etc.

Una estructura jerárquica de datos tiene que satisfacer las siguientes condiciones:

- Un modelo jerárquico de datos siempre comienza con un nodo raíz.

- Cada nodo consiste de uno o más atributos describiendo la entidad en ese nodo.

- Los nodos dependientes pueden seguir en niveles subsecuentes.

- El nodo en los niveles precedentes se convierte en el nodo padre de los nuevos nodos dependientes. Los nodos dependientes pueden ser agregados horizontal o verticalmente sin limitación.

- Cada nodo ocurrente en el nivel 2, tiene conexión con uno y sólo uno de los que ocurren en el nivel 1, etc.

- Un nodo padre puede tener un nodo o más nodos hijos, si no tiene nodos hijos, no es un nodo padre.

- Cada nodo, excepto por supuesto la raíz, tiene que ser accedido a través de su nodo padre. El nodo representado en una verdadera jerarquía debe ser alcanzado sólo a través de su padre, siendo este el verdadero significado de que la información existe. Entonces la trayectoria a cada nodo dentro del modelo de datos jerárquico es única.

Una jerarquía es una red cuyas ligas forman un ***bosque***; cada liga es una relación varios a uno, es decir, de hijos a padre.

La primera impresión podría ser que ciertas redes son jerarquías, y que estas no tienen el poderío de aquellas para representar información. Una segunda impresión es que con la suficiente duplicación de registros, las redes pueden convertirse en jerarquías, ambas apreciaciones pueden ser correctas, sin embargo, no debe olvidarse el principio de evitar la redundancia. Entonces el objetivo final de la construcción de jerarquías es introducir ***registros de tipo virtual***, los cuales son apuntadores a otros registros tipo que permiten la duplicación aparente de registros sin recurrir a la redundancia.

Una jerarquía, es simplemente un tipo especial de red denominado conjunto de árboles, en el cual todas las ligas apuntan en dirección de ***nodo hijo a nodo padre***, es decir de registro lógico derivado a registro lógico original.

La operación básica en una base jerárquica de datos, es el ***recorrido*** del árbol que la sustenta, esto es dado un nodo de la base de datos, podemos acceder a los nodos descendientes.

Es importante notar que la operación básica es unidireccional, lo que ocasiona que ciertas relaciones sean difíciles de acceder en este modelo.

En un sentido estricto, el implementar una jerarquía es muy similar al procedimiento para las redes. Sin embargo, la forma de las bases jerárquicas de datos tiende a ser diseñada con un enfoque particular muy común y apropiado.

Para cada árbol en la jerarquía se determina un formato de registro de longitud variable con las siguientes reglas:

- El formato para cada hoja es $(A)^*$, en donde A es la lista de campos en el registro lógico para la hoja o nodo.

- Si un nodo tiene k hijos con registros de formato variable $a_1, a_2, ..., a_k$, y la lista de campos en el formato del registro lógico para ese nodo es B, entonces el formato para el registro de longitud variable para el nodo es $(Ba_1, Ba_2, ..., Ba_k)^*$.

Existen varias formas para llevar a cabo los registros de longitud variable, sin embargo, un enfoque especializado es aconsejable, a fin de proveer las facilidades necesarias para acceder a alguna parte de la base de datos sin desperdicio de espacio o tiempo.

Una forma de ver esta organización es tal que cada registro lógico es visto como un conjunto (descrito en el modelo de red) en el cual su padre es el propietario y él es el tipo miembro. Lo anterior, resulta en una organización que dado un nodo, es posible encontrar los descendientes en el árbol en muy pocos bloques en promedio, puesto que ellos siguen al nodo en una secuencia preordenada.

Esta propiedad de preorden junto con la suposición del tipo más frecuente de peticiones de información, justifican la secuencia de preorden como una manera muy importante de como se almacena la información jerárquica.

La mejor ventaja del modelo jerárquico de datos es la existencia de sistemas manejadores de bases de datos probados que utilizan el modelo jerárquico como base.

Una ventaja adicional es la relativa simplicidad de uso del modelo jerárquico y la familiaridad de los usuarios con la jerarquía.

Dentro de las desventajas de este modelo, sobresale la forma compleja en que se implementa la relación varios a varios, ya que puede resultar en redundancia en la información almacenada. Se sabe que la redundancia a nivel lógico no es necesariamente mala, pero redundancia a nivel físico, es indeseable.

Como resultado de una ordenación jerárquica estricta, las operaciones de inserción y eliminación llegan a ser muy complejas. Eliminación de padres involucra la eliminación de hijos: como resultado los usuarios tienen que ser muy cuidadosos con la operación de eliminación.

Las instrucciones en los lenguajes jerárquicos tienden a ser de tipo declarativo debido a la rigidez de la estructura.

II.3.2.3. EL MODELO RELACIONAL.

La publicación pionera sobre sistemas basados en el modelo relacional fue una serie de artículos del **Dr. E.F. Codd, [CODD 70], [CODD 71]**, muchos otros trabajos han redefinido los conceptos originales y han desarrollado varias aproximaciones al ideal relacional. Los sistemas relacionales se han podido instrumentar gracias a trabajos de investigación y herramientas prácticas que ahora existen.

El concepto matemático que fundamenta el modelo relacional es un conjunto teórico llamado ***relación***, el cual se define en este ámbito como un subconjunto del producto cartesiano de una lista de dominios, si por ***dominio*** entendemos simplemente un conjunto de valores.

El producto cartesiano de los dominios $D_1,D_2,...,D_N$, se expresa como D_1 X D_2 X ... X D_N y consiste en la n-eadas $(v_1,v_2,...,v_N)$, de tal manera que v_1 está en D_1, v_2 está en D_2, etc. El valor N se le conoce como el *grado* de la relación.

Una vez que se coloca nombre a las columnas de una relación, el orden de estas deja de ser importante, en términos matemáticos se ven las n-eadas como asociaciones de nombres de atributos a valores en el dominio de los atributos, este nuevo punto de vista puede hacer que ciertas relaciones resulten iguales, pero bajo la forma tradicional de definir una relación no son iguales.

Es útil visualizar una relación como una ***tabla***, donde cada fila corresponde a una n-eada y cada columna a un componente de la relación, a estos últimos se les conoce como ***atributos***, el conjunto de nombres de atributos se le denomina ***esquema de relación***.

Cada entrada de la tabla es una fila que contiene los valores para cada atributo. Las filas se pueden considerar como ***registros*** y las columnas como ***campos del registro***. En matemáticas, una estructura como una tabla se puede tratar como una ***relación***. Para aplicar la teoría relacional matemática, es necesario que se cumplan ciertas condiciones. Como resultado de aplicar esta teoría, obtenemos un método uniforme y consistente, denominado ***método relacional***.

El conjunto de esquemas de relación que se usan para representar información, es denominado un ***esquema relacional de bases de datos***, y a las instancias de las relaciones correspondientes se conoce como ***base relacional de datos*** , a través de este procedimiento es factible establecer relaciones con cualquier conjunto de atributos como esquemas de relación y dar cualquier interpretación a las n-eadas.

Es necesario que un sistema de bases de datos satisfaga una serie de reglas para poder ser calificado como relacional, dichas reglas surgen de la teoría matemática subyacente, la cual provee una sólida fundamentación teórica, permite una mayor integración de los datos y mejores mecanismos para su recuperación.

Las reglas básicas son las siguientes:

- En un modelo relacional, una tabla debe contener sólo un tipo de relación. Cada relación tiene un número fijo de atributos, todos ellos con un nombre explícito. La base de datos generalmente contendrá muchas tablas, de forma que los diferentes tipos de relaciones se encontrarán en tablas diferentes; una tabla por cada tipo de relación.

- Dentro de la tabla cada atributo es distinto, y no se permiten grupos repetitivos.

- Cada relación de una tabla es única, es decir, no hay relaciones duplicadas.

- El orden de las relaciones dentro de la tabla no está determinado. Las relaciones pueden venir en cualquier orden, no hay una secuencia determinada previamente.

- Para cada atributo definido, existe un conjunto de valores posibles que forma el dominio de definición de ese campo.

- Se pueden crear nuevas tablas, relacionando valores de atributos procedentes de dos tablas ya existentes, siempre y cuando en las dos tablas se haya definido el mismo dominio para los valores que pueden tomar los atributos que se relacionan. La formación de nuevas tablas, a partir de las ya existentes, es la esencia del proceso relacional.

Hay que notar que las operaciones que realiza el sistema de bases de datos, se hacen sobre la tabla en su conjunto, no sobre cada relación individual que contiene la tabla. Se considera a esto algo de mucho más poder que el proceso registro a registro característico de otros sistemas.

Los fundamentos teóricos permiten desarrollar operaciones satisfactorias sobre los que se pueden predecir resultados.

La definición de estas operaciones puede ser muy amplia, pero la generalidad de ellas se fundamenta en las tres siguientes:

- Una inquietud que puede surgir es la de conocer una n-ada que cumpla con un criterio de búsqueda. a esta operación se le conoce como ***selección*** (crea un subconjunto de las filas de las tablas).

- El conectar información de dos o más relaciones, se le denomina ***unión***.

- Crear un subconjunto de atributos de las entidades relacionadas (columnas de la tabla).

La ***normalización*** es una técnica que se ha desarrollado para obtener estructuras de datos eficientes. El concepto de normalización fue introducido por **Codd [CODD 74]**, y fue pensado para aplicarse a sistemas relacionales, sin embargo, tiene aplicaciones más amplias. La normalización es la expresión formal del modo de realizar un buen diseño. Provee los medios necesarios para describir la estructura lógica de los datos en un sistema de información.

Antes de definir o describir el proceso de normalización, es de suma importancia abarcar ciertos conceptos que facilitan la comprensión de dicho proceso.

Una relación es ***no reductible*** si es imposible separarla en varias relaciones de grado más bajo por medio de operaciones de proyección, de manera tal, que se pueda reconstituir la relación original al unir las relaciones producto. Una relación no reductible se le conoce también como ***relación elemental.***

Un atributo **B** de una relación es ***funcionalmente dependiente*** del atributo **A**, si en todo momento cada valor de **A** está asociado con uno y sólo uno de los valores de **B**. De otra manera, existe siempre una asociación simple o condicional de **A** a **B**. Lo que significa que **B** esta en función de **A** y se puede representar como una tabla de dos columnas.

La notación que generalmente se utiliza para indicar que **B** es funcionalmente dependiente de **A** en la relación **R** es:

R.A ---> R.B

Por otro lado si **B** no es funcionalmente dependiente de **A** en la relación **R**, se denota como:

$$R.A \text{ -/-> } R.B$$

La definición anterior se puede generalizar a una colección de atributos en **R**, se dice que $B_1,...,B_N$, es funcionalmente dependiente a otra colección de atributos en **R**, $A_1,...,A_M$, si siempre cada lista de atributos $R[A_1],R[A_2],...,R[A_M]$ está asociada a una y sólo una lista de atributos $R[B_1],...,R[B_N]$.

En donde $R[A_1]$ significa un valor del atributo A_1 correspondiente a una fila de la relación.

Una ***dependencia trivial*** (en donde **E** es un subconjunto de **D**) es de la forma:

$$R.D \text{ ---> } R.E$$

El atributo o colección de atributos B, en una relación R, es ***funcionalmente dependiente de forma completa*** de un atributo o colección de atributos A en R; si B es funcionalmente dependiente de A, pero no es funcionalmente dependiente de ningún otro subconjunto de A. Esto es equivalente a tener una asociación simple o condicional. En otras palabras, el atributo o colección de atributos **B** en **R**, es funcionalmente dependiente de forma completa a la colección de atributos de **A** en **R** si existe una asociación simple o condicional de **A** a **B**, en donde toda asociación de cualquier subconjunto propio de A a B es compleja.

La notación usual para indicar que B es funcionalmente dependiente de forma completa de **A** en **R** es:

$$R.A \text{ ===> } R.B$$

Por otro lado

$$R.A \text{ =/=> } R.B$$

significa que **B** no es funcionalmente dependiente de forma completa de **A** en **R**.

La ***dependencia transitiva*** se puede definir como sigue:

Supóngase que **A,B** y **C** son tres distintos atributos de una colección de atributos en una relación **R**, y que las siguientes dependencias ocurren:

$$R.A \text{ ---> } R.B$$

$$R.B \text{ -/-> } R.A$$

$$R.B \text{ —> } R.C$$

Entonces

$$R.A \text{ —> } R.C$$

$$R.C \text{ -/-> } R.A$$

Se dice que **C** es ***transitivamente dependiente en forma estricta*** de A en la relación **R**, en el caso:

R.C ---> R.B

y **B** y **C** son transitivamente dependientes de **A** en la relación **R**.

Al atributo o colección de atributos **C** de una relación **R** para el cual cualquier otro atributo o colección de atributos **D** de **R** es funcionalmente dependiente de **C**, es denominado ***determinante***.

En el proceso de datos los conceptos de ***claves*** o ***llaves*** y ***valores de llaves*** son fundamentales, por lo que el modelo relacional debe incorporar facilidades para el manejo de llaves. De hecho, los sistemas relacionales tienen reglas especiales para éstas.

Un llave puede estar formada por uno o más atributos de las relaciones.

Una ***llave posible o candidata***, es una combinación de atributos de cualquier n-eada de la tabla, que distinga inequívocamente a esa n-eada de cualquier otra de la tabla. En otras palabras, la llave candidata debe ser única. En los sistemas relacionales siempre es posible formar esa llave única, ya que no existen relaciones duplicadas. Otra regla vigente es, que si de la llave candidata se elimina un atributo, se pierde la propiedad de identificación unívoca que ésta nos permite, pues la llave no puede contener atributos redundantes.

Resumiendo, se define como ***llave candidata*** K a un atributo o conjunto de atributos de una relación R, si cumple con las siguientes propiedades:

1. En cada n-eada de **R** el valor de **K** identifica de manera única esa n-eada (**identificación única**). Como corolario se puede establecer que cada atributo de **R** es funcionalmente dependiente de **K**.

2. En el caso en donde **K** es una colección de atributos, ninguno de los atributos de **K** pueden eliminarse sin violar la condición 1 (**No redundancia**). Como corolario cada subconjunto propio de la colección de atributos **K**, es funcionalmente independiente de cualquier otro subconjunto propio de atributos de la colección **K**. En otras palabras, existe un mapeo varios a varios entre cualquier par diferente de subconjuntos propios de la colección de atributos **K**.

Una relación puede poseer diversas llaves candidatas. Una de las llaves candidatas siempre es designada como ***llave primaria***. La forma usual de la designación consiste en que no se permiten valores indefinidos o nulos en los atributos que la constituyen.

Un ***atributo primo*** de una relación **R**, es un atributo que pertenece al menos a una llave candidata de **R**. Análogamente un atributo de una relación **R** que no pertenece a ninguna llave candidata de **R**, es denominado un ***atributo no primo***. A los atributos que conforman una llave primaria se les conoce como ***atributos llave***.

Un atributo de una relación R_1, es una ***llave externa*** si no es llave primaria de R_1, pero sus valores son valores de la llave primaria de alguna otra relación R_2. Las llaves primarias junto con las llaves externas, proveen de un mecanismo de visualización de las asociaciones entre relaciones.

Si una relación **R** contiene varios atributos que son funcionalmente dependientes de otros atributos en **R**, puede ser dividida en varias relaciones de menor grado, de tal manera que esas relaciones producto pueden ser unidas para reconstruir la relación **R** original.

Ahora estamos en posición de describir la normalización de relaciones, dicho proceso es de suma importancia sobre todo en la definición de relaciones que son consistentes con un modelo especificado, de manera que no tengan dependencias indeseables que causen problemas en la manipulación de la base de datos, es decir, cuando se crean, modifican o eliminan los valores de los datos, a dichos problemas se les denomina *anomalías en las operaciones de almacenaje*.

Codd, [CODD 71] originalmente definió los niveles de normalización llamados primera forma normal *(1FN)*, segunda forma normal *(2FN)* y tercera forma normal *(3FN)*. Hablando de manera general cada relación normalizada está en la (1FN), algunas relaciones (1FN) son también (2FN) y estas son un subconjunto de las relaciones (3FN). En resumen, es posible adelantar que las relaciones (3FN) son las mejores porque estas no permiten la mayoría de las inconsistencias de la información de la base de datos.

El proceso de normalización como fue sugerido por Codd, permite convertir cualquier relación en un conjunto semánticamente equivalente de relaciones (3FN). La intención es implementar una base de datos que evite propiedades indeseables como las anomalías en las operaciones sobre la base de datos.

A las relaciones que se caracterizan por tener en cada atributo un valor *atómico*, es decir, que no es posible descomponerlo, se les denomina relaciones en la *primera forma normal (1FN)*. Es posible visualizar este tipo de relaciones en una tabla, de manera tal, que en toda intersección fila y columna de dicha tabla se tiene un solo valor, nunca un conjunto de valores. Los valores indefinidos se permiten sólo en los atributos que no constituyen la llave primaria de la relación.

Por otro lado, una relación que tiene valores múltiples en alguna intersección fila y columna, no está en la forma (1FN) y se les conoce como *relación no normalizada*.

Establecemos primero la definición original de **Codd**; una relación **R** está en *segunda forma normal (2FN)*, si está en primera forma normal (1FN), y cada atributo no primo de **R** es funcionalmente dependiente en forma completa de toda llave candidata de **R**.

Ahora se sabe que la definición anterior no maneja satisfactoriamente las relaciones con llaves candidatas traslapadas, por lo que se propuso **[SCHMID 75]** la siguiente definición revisada:

Una relación está en *segunda forma normal* si está en (1FN), y todo atributo en el complemento de una llave candidata **K** es funcionalmente dependiente en forma completa de **K**.

Una colección de relaciones C está en *segunda forma normal óptima* con respecto a una relación T si se cumple con:

1 Todas las relaciones en C están en (2FN).

2 Las relaciones en C retienen toda la información esencial en la relación T, es decir, T puede recobrarse por la unión de las relaciones en C.

3 No hay una colección más pequeña con las propiedades 1 y 2.

Una relación R está en *tercera forma normal (3FN)* si está en la segunda forma normal y todo atributo no primo de R es dependiente no transitivamente de cada llave candidata de R, una vez más, la definición anterior no maneja satisfactoriamente el caso en que una relación que posee dos llaves candidatas traslapadas, por lo que se propuso una definición alternativa:

Una relación en (2FN) está en (3FN); Si cada atributo en el complemento de una llave candidata es dependiente no transitiva de esa llave candidata.

De manera similar, al caso de las relaciones (2FN), existe una noción para (3FN) óptimas.

Sea C_2 una colección de relaciones en (2FN) óptimas, y C_3 una colección de relaciones en (3FN) obtenidas por proyecciones de las relaciones en C_2. Entonces la colección C_3 es una *forma normal (3FN) óptima* con respecto a C_2 si:

1 Todas las relaciones en C_3 están en (3FN).

2 Las relaciones en C_3 retienen toda la información esencial en C_2, es decir, la colección C_2 puede recobrase por unión de relaciones en C_3.

3 Ninguna relación en C_3 contiene dos atributos que sean transitivamente dependiente de forma estricta en alguna relación en C_2.

4 No hay colección de relaciones más pequeña que tenga las propiedades 1, 2 y 3.

A continuación se mencionan las ventajas más sobresalientes del modelo relacional.

Simplicidad. El usuario final trata con un modelo simple, el mecanismo de acceder es formulado en términos del esquema conceptual, siendo ajeno a las complejidades de los aspectos netamente de sistemas y computación.

Petición de tipo no-procedimiento. Debido a que no existe dependencia posicional entre las relaciones, las peticiones o gestiones a la base de datos no tienen que reflejar ninguna estructura fija.

Independencia de Datos. Este es uno de los objetivos más importantes de cualquier sistema manejador de bases de datos. El modelo relacional, al retirar todos los detalles de estructuras de almacenamiento y estrategias de acceder de la interface del usuario, provee de un alto nivel de independencia de datos, sin embargo, debe señalarse que para lograr esto, es necesario que el diseño de las relaciones debe ser completo y adecuado.

Fundamentos Teóricos. El modelo relacional está basado en teoría matemática relacional, por lo que el diseñar una base de datos rigurosamente apegada al modelo y con la utilización de la normalización, dota de un fundamento sólido al modelo, destacando que esto no necesariamente existe fácilmente en algún otro de los modelos descritos.

Por otra parte, la desventaja principal de un sistema manejador relacional de bases de datos, es su ***rendimiento,*** ya que este no se compara aún con el rendimiento de un sistema manejador de bases de datos basado en un modelo jerárquico o de red. Sin embargo, los avances tecnológicos tienden a solucionar el problema.

Las siguientes tablas permiten visualizar cómo los conceptos básicos de los modelos de datos se han considerado en los modelos primitivos y clásicos.

I.CONCEPTOS.	PRIMITIVOS	MODELOS CLASICOS JERARQUICO	RED	RELACIONAL
a) Propiedades estáticas.				
Objetos simples	registros	segmentos	registros	n-eadas
Objetos compuestos	_____	Jerarquías de registros	redes de registros	n-eadas
Restricciones objetos	_____	existencia del padre	existencia propietario	
Atributos de un objeto	campos	campos	campos (ítems)	atributos
Restricciones a los atributos		llaves no nulas	llaves	llaves, n-eada única
Relaciones entre atributos	_____	Dependencia Funcional	Dependencia Funcional	Dependencia Funcional, Multivaluación
Relaciones entre Objetos	índices, listas	ligas 1:N jerárquicas	redes de ligas 1:N	n-eadas, N:M
Restricciones a las relaciones	_____	eliminación de padres, eliminación de hijos	eliminación de propietario,puede eliminar miembro	Integridad referencial
b) Propiedades dinámicas				
Operación Simple	lectura y escritura registros	orientado a segmentos	orientado a registros	orientado a conjuntos
Operación compuesta (Transacción)	_____	_____	_____	_____
Relaciones entre operaciones (estructuras de Control)	Jerarquía	implícitas, orientado a miembros de conjuntos	implícitas, iteración	implícitas
Manejo de excepciones	_____	vía progr.	vía progr.	_____
c) Restricciones y facilidades para las peticiones	_____	árbol transversal ponderado	navegación ponderada	cálculo relacional y álgebra relacional

	PRIMITIVOS	MODELOS CLASICOS JERARQUICO	RED	RELACIONAL
2. HERRAMIENTAS.				
Lenguajes de definición	definición de archivos	si existe	si existe	si existe
Lenguajes de manipulación	_____	orientado segmentos	orientado registros	orientado conjuntos
Lenguajes de petición	_____	orientado segmentos	orientado registros	basado en álgebra y cálculo
Tipos de datos	tipos registro	campos y segmentos	campos y registros	atributos y n-*adas*
Variables	_____	_____	_____	en álgebra y cálculo
Diseño automatizado	_____	si existe	elemental	si existe
3. TECNICAS.				
Diseño de esquemas	_____	si existe	si existe	si existe
Diseño de subesquemas	_____	si existe	_____	si existe
Diseño de transacciones	_____	_____	_____	_____
Diseño de peticiones	_____	elemental	_____	si existe
Conceptos de objeto y clase	_____	si existes	si existe	si existe

II.3.3. MODELOS SEMANTICOS DE DATOS.

La tendencia importante en el diseño de modelos de datos, es incrementar la capacidad de modelado de los mismos, esto es, los modelos se diseñan para proveer conceptos más expresivos y más ricos, con los cuales se capture más significado (***semántica***) de lo que es posible al utilizar modelos de datos clásicos.

A continuación de los modelos clásicos, una nueva generación de modelos fue desarrollada, a esta generación se le denomina ***modelos semánticos de datos***, el nombre podría ser inapropiado, ya que la siguiente generación intentará incorporar mejoras semánticas.

Los modelos semánticos de datos pueden clasificarse como sigue:

-Extensiones directas a los modelos clásicos.

-Modelos matemáticos de datos.

-Modelos irreductibles de datos.

-Modelos jerárquicos de semántica estática.

-Modelos jerárquicos de semántica dinámica.

II.3.3.1. EXTENSIONES DIRECTAS DE LOS MODELOS CLASICOS.

El ***modelo estructural***, restringe las relaciones en el modelo relacional a cinco tipos o interpretaciones específicas. Por ejemplo, una relación entre dos objetos debe incluir una referencia de cada objeto. Esas restricciones al modelo relacional, guían el diseño de la base de datos y fomentan a utilizar el modelo relacional con precisión. Destaca la característica de que el modelo no requiere de distinguir entre objetos y relaciones; por lo tanto, los esquemas resultantes son relacionales.

El ***modelo objeto-rol***, extiende el modelo de red al agregar la noción de rol. Un objeto puede jugar muchos papeles en una aplicación y puede tener diferentes propiedades para cada rol. El modelo objeto-rol reduce la redundancia al considerar cada rol como un objeto separado, permitiendo un modelado más orientado a las aplicaciones.

El ***modelo entidad-relación***, combina características de los modelos de red y relacional. Hace una clara distinción entre objetos y relaciones. Por lo anterior, las aplicaciones se representan como redes, en las cuales, los objetos o entidades son nodos y las relaciones son ligas. Este modelo provee mecanismos no restringidos para representar relaciones.

La popularidad del modelo entidad-relación en el diseño de bases de datos de alto nivel, se debe a su economía de conceptos y a la utilización extensiva de entidades y relaciones como conceptos naturales de modelado.

II.3.3.2. MODELOS MATEMATICOS DE DATOS.

Mucha de la importancia del modelo relacional, es debida, a su formalidad matemática. Varios investigadores, han utilizado estas bases para formalmente extender el modelo relacional.

Existen tres grupos principales de modelos definidos matemáticamente; ***basados en la teoría de conjuntos, basados en lógica de primer orden*** y ***basados en la consideración de instancias universales***. Estos nuevos modelos, proveen notaciones y definiciones formales para todos los conceptos considerados en otros modelos y lo más importante, dotan de herramientas formales para analizar las propiedades de las aplicaciones y resolver los problemas teóricos relacionados.

Estrictamente hablando, estos modelos no son nuevos, sin embargo, en la definición de otros modelos, por medio de un sistema matemático, como lo es la ***lógica de primer orden***, los modelos de datos se ven extendidos por las herramientas y técnicas asociadas.

Existe una diferencia muy importante entre la panorámica tradicional de las bases de datos, y las bases de datos usadas en modelos que se fundamentan en lógica de primer orden.

En el punto de vista tradicional, los objetos se representan en una base de datos como instancias de los tipos de objetos que son definidos en el esquema correspondiente. Generalmente, hay un gran número de instancias para cada tipo de objeto, y dichas instancias son agregadas y eliminadas de la base de datos para reflejar cambios en el medio ambiente de la aplicación. Desde el punto de vista lógico, los objetos se representan como términos y propiedades (atributos, relaciones, restricciones) en forma de sentencias lógicas, es decir, existe una base de datos de hechos o sucesos contra la cual las sentencias son evaluadas para resolver peticiones, los cambios en la aplicación, se reflejan agregando sentencias lógicas, no necesariamente por medio de la alteración de la base de datos de hechos. La eficiencia de una implementación de un enfoque lógico, continua siendo un objetivo de la investigación actual, particularmente, en el caso de bases de datos muy grandes, que tengan una actividad de actualización significativa.

II.3.3.3. MODELOS IRREDUCTIBLES DE DATOS.

Varios modelos de datos se han orientado a satisfacer la necesidad de representar información como hechos atómicos, más que como grupos complejos de hechos. Esos modelos son denominados ***irreductibles***, ya que los hechos atómicos no pueden ser descompuestos. Una de las ventajas de este enfoque, es que simplifica la actualización, toda vez que, cada hecho es alterado independientemente. Es obvio, que los hechos atómicos incrementan la precisión del modelado, ya que los hechos atómicos se pueden combinar de una forma apropiada para conformar conceptos de nivel superior, en lugar de tener una estructura fija que se imponga a todos lo hechos.

El *modelo de relación binaria*, es un modelo irreductible de datos, el cual es una restricción del modelo relacional, ya que comprende relaciones binarias en lugar de n-arias. Una relación de este tipo es entre un objeto y un atributo simple. Se considera a esta relación como un hecho atómico. Una relación binaria es el hecho representable más pequeño en el modelo relacional.

En el *modelo relacional irreductible*, las relaciones no necesariamente son binarias. Ellas deben tener la propiedad de no poderse descomponer en otras relaciones. Esto es, la información se perderá si una relación irreductible se descompone en dos o más relaciones.

El modelo irreductible más difundido, es el conocido como *modelo funcional de datos*, este modelo de datos combina aspectos del modelo relacional con conceptos de *programación funcional*. Los objetos, en este ámbito se representan directamente con propiedades de objetos y las relaciones se ven representadas como mapeos funcionales entre objetos. Es irreductible una vez que cada atributo está relacionado a su objeto asociado por una función. La propiedad más importante del modelo de datos funcional, es la facilidad de establecer peticiones de manera sencilla a través de la composición funcional.

II.3.3.4. MODELOS JERARQUICOS DE SEMANTICA ESTATICA.

Muchos de los modelos semánticos, han sido influenciados por las *redes semánticas de la inteligencia artificial*. Los modelos jerárquicos de semántica estática, integran conceptos del modelo de datos relacional con cuatro relaciones importantes de la red semántica, denominadas *clasificación*, *agregación*, *generalización* y *asociación*.

Las relaciones mencionadas sirven para soportar abstracción de datos, en la cual, los detalles específicos se suprimen, facilitando resolver el problema de tener a la mano la información importante. La *clasificación* establece una relación entre el esquema y los objetos de la base de datos. La *agregación*, *generalización* y *asociación* establecen relaciones entre los tipos de objeto definidos en el esquema.

Estas relaciones tienen que ser definidas como restricciones inherentes para los modelos de datos semánticos, a continuación se hace una descripción breve de las mismas:

Clasificación. Es una forma simple de abstracción de datos en la cual, un tipo de objeto se define como un *conjunto de instancias*. Esto establece, una relación "**instancia de**" entre un tipo de objeto en un esquema y sus instancias en las bases de datos.

Agregación. Es una forma de abstracción en la cual, una relación entre objetos se considera como un *objeto agregado* de un nivel más alto. Cuando se considera la agregación, los detalles específicos de los objetos que la constituyen se suprimen. Cada instancia de un objeto agregado puede descomponerse en instancias de los objetos componentes. Esto establece una relación "**parte de**" entre objetos. Por ejemplo, una persona se puede representar como un agregado de propiedades de personas.

Generalización. Es una forma de abstracción en la cual, los objetos similares se relacionan a un objeto genérico de más alto nivel. Los objetos constituyentes se consideran *particularidades del objeto genérico*. En el nivel del objeto genérico, las diferencias de las particularidades se suprimen, mientras que las similitudes se resaltan. Cada instancia de un objeto genérico puede encontrarse como una instancia de uno o más de los objetos particulares. Lo anterior establece una relación "**es una**" entre objetos. Por ejemplo, una universidad y una preparatoria son particularidades del concepto genérico institución educativa.

Asociación. Es una forma de abstracción en la cual, una relación entre objetos similares se considera un ***conjunto de objetos*** de nivel más alto. Los detalles de los objetos miembro se suprimen, y las propiedades del conjunto de objetos se resaltan. Una instancia de un conjunto de objetos puede descomponerse en un conjunto de instancias del objeto miembro. Lo anterior establece una relación "***miembro de***" entre un objeto miembro y un conjunto de objetos. Por ejemplo, el conjunto de objetos "canes" se define sobre el objeto miembro "can", notese que el objeto miembro "can" tiene diversos atributos que en el conjunto de objetos no interesan. La asociación también es denominada ***agrupamiento***.

Al utilizar modelos jerárquicos semánticos, los esquemas se representan como agregaciones, generalizaciones y asociaciones jerárquicas de objetos. Cada objeto puede ser simple (definido por sí mismo) o compuesto (definido como una agregación, generalización o asociación de otros objetos).

Cada abstracción puede ser aplicada para formar jerarquías agregadas, jerarquías generalizadas, etc. Un objeto puede tener una descomposición jerárquica de acuerdo a cada agregación, generalización y asociación.

II.3.3.5. MODELOS JERARQUICOS DE SEMANTICA DINAMICA.

Las extensiones a los modelos jerárquicos de semántica estática, fueron desarrolladas para integrar conceptos dinámicos de modelado. Estos modelos integran un número importante de conceptos de lenguajes de programación, con conceptos de bases de datos. Asimismo se incluyen, muchas de las implementaciones descritas en la sección anterior.

Resalta el uso intensivo de conceptos avanzados de tipos de datos como son, los ***tipos de datos abstractos***, ***clases***, y ***tipos polimórficos***.

II.3.4. MODELOS SEMANTICOS DE DATOS DE PROPOSITO ESPECIAL.

La siguiente generación de los modelos semánticos de datos, se orienta a aplicaciones especiales. La teoría de modelado semántico, está siendo aplicada al desarrollo de objetivos particulares. Los modelos anteriormente descritos, proveen conceptos de propósito general para el modelado de bases de datos, esto es, de manera análoga a como lo hacen los lenguajes de programación de propósito general que proveen conceptos generales para la programación. El objetivo es tener lenguajes de alto nivel y modelos que faciliten el desarrollo de modelos de datos, tanto para aplicaciones específicas como procedimientos de oficina, **VLSI, CAD/CAM**, cartografía, etc., como para las aplicaciones tradicionales.

La elección de un modelo de datos para una aplicación específica, es similar al procedimiento que se utiliza para elegir un lenguaje de programación, un esquema de representación de conocimientos o una notación matemática, es decir, es muy subjetiva. Cuando los modelos clásicos se introdujeron, existió discusión sobre el valor de los modelos, ahora se sabe, que los modelos no difieren significativamente en su poder de expresión, no sólo hay lenguajes de alto nivel que se aplican a los tres modelos, sino que la base asociada de datos y sus reglas corresponden a esquemas similares.

La diferencia primordial sigue siendo la clasificación de sus restricciones específicas en inherentes, explícitas o implícitas.

Algunos trabajos han demostrado que los tres modelos clásicos tienen un conjunto común de conceptos. Los modelos jerárquicos y de red, tienen adicionalmente a los conceptos del conjunto común, otros conceptos similares entre ellos. De manera semejante, el modelo relacional de datos, puede expresarse como una serie de conceptos relacionales únicos adicionales al conjunto común de conceptos.

Un argumento análogo se ha propuesto y determina que el modelo relacional está contenido por el modelo jerárquico, el cual a su vez es contenido por el modelo de datos de red.

II.4. TOPICOS A CONSIDERAR PARA EL DESARROLLO DE LOS MODELOS DE DATOS.

Existe una serie de tópicos que han determinado la orientación del desarrollo de los modelos de datos.

A continuación, se hace una breve descripción de los tópicos predominantes, estos tópicos o temas sugieren orientaciones para la investigación, sin embargo, cabe señalar que no todos pueden ser alcanzados paralelamente.

II.4.1. INDEPENDENCIA DE DATOS.

La *independencia de datos*, es uno de los objetivos principales de la tecnología de las bases de datos. Involucra la separación de las propiedades lógicas y físicas, a fin de facilitar la evolución y amortiguar el efecto dañino del cambio. Por ejemplo, involucra considerar la inmunidad de las aplicaciones a los cambios en la estructura de almacenamiento y en las estrategias de acceso.

La *independencia lógica de los datos*, fomenta el separar las necesidades individuales de las necesidades de la organización, involucra el aislamiento de los cambios lógicos de cualquier otro aspecto de una aplicación, de manera que minimice el impacto tanto como sea posible.

La *independencia física de los datos*, fomenta el separar las necesidades de funcionalidad de las necesidades de rendimiento, lo que implica el aislamiento de los cambios físicos con un impacto tan pequeño como sea posible.

La independencia de datos no ha sido alcanzada completamente.

Los modelos primitivos, ofrecían muy poco soporte a la independencia de datos. Los modelos clásicos introducen el concepto de esquema y subesquema, de tal manera, que los cambios pueden ser amortiguados sin depender de la estructura de implementación. Con los modelos jerárquicos y de red, los esquemas y subesquemas son fijos al tiempo de la compilación, destacando, que los subesquemas son subconjuntos de los esquemas. El modelo relacional introduce el concepto de una vista derivada o subesquema. y un lenguaje con el cual definir esquemas y subesquemas dinámicamente.

Las vistas dinámicas ya han sido propuestas recientemente para los modelos jerárquico y de red. El objetivo primordial de algunos modelos semánticos, es proveer mecanismos para la evolución de las propiedades estáticas y dinámicas de las aplicaciones de bases de datos.

II.4.2. RELATIVISMO SEMANTICO.

El *relativismo semántico*, concierne a la posibilidad de ver y manipular información de la manera más apropiada para el usuario. El usuario no es forzado a interpretar información de una manera particular. Una vista específica puede determinarse al mismo tiempo para la información y las necesidades operacionales.

El relativismo semántico, permite que las relaciones y entidades formen parte de futuras entidades y relaciones. De la misma manera, soporta independencia de información lógica, una vez que soporta el diseño de vistas, la evolución de esquemas y subesquemas y el diseño de interfaces para las necesidades específicas del usuario.

El relativismo semántico fue introducido por el modelo relacional de datos, en el cual, todos los objetos y relaciones son representados en relaciones n-arias. Es decir, es permitido interpretar una relación como una entidad con n atributos, o como una relación entre n objetos o como alguna combinación de ellos.

La red, la jerarquía y sus variaciones, incluyendo el modelo entidad-relación, no soportan el relativismo semántico; sin embargo, si es factible encontrarlo en los modelos semánticos basados en el modelo relacional.

II.4.3. ESTRUCTURA DE INTEGRACION Y COMPORTAMIENTO.

La *especificación completa de las semánticas* de una aplicación de bases de datos, incluye las propiedades estáticas y las dinámicas de los objetos de la aplicación.

Los enfoques tradicionales enfatizan las propiedades estáticas, el diseño de bases de datos típicamente involucraba extracción de las propiedades estructurales de todos los programas que accedían la base de datos, diseño de un esquema que representara las propiedades mencionadas, escritura de los programas y las peticiones que se requerían. Lo anterior difiere fuertemente con el enfoque de los lenguajes de programación, el cual destaca las propiedades dinámicas y resalta el diseño de vistas más representativas.

Hoy en día, es muy claro que las propiedades estáticas y dinámicas pueden ser tratadas de forma abstracta, y que ambos tipos de propiedades son necesarios para definir la semántica de un objeto. Por esto, uno de los objetivos de la investigación, es integrar conceptos de modelos de datos con conceptos de lenguajes de programación, asimismo, desarrollar herramientas y técnicas que soporten la especificación completa de bases de datos orientadas a objetos.

II.4.4. ABSTRACCION DE DATOS Y PROCEDIMIENTOS.

La abstracción involucra la capacidad de resaltar los detalles esenciales del problema, al mismo tiempo que se suprime lo irrelevante. La abstracción es esencial en las aplicaciones de bases de datos, una vez que, la complejidad inherente de las mismas es muy difícil de manejar.

Típicamente, una aplicación de bases de datos contiene una gran cantidad de objetos que tienen muchas relaciones entre sí. La abstracción de datos concierne a la abstracción aplicada a las propiedades estáticas. De manera análoga, la abstracción de procedimientos, involucra la abstracción aplicada a las propiedades dinámicas.

El primer enfoque de abstracción de datos, fue la independencia de datos, la cual estableció la separación de los conceptos lógicos de los conceptos de representación física. Como resultado de lo anterior, se tiene que el modelado de datos se trasladó del ámbito de representación orientada a registros al ámbito de objetos con orientación lógica. El modelo de red contribuyó al concepto de esquema, el cual se mueve entre descripciones lógicas y físicas. Por su parte el modelo de datos relacional contribuyó a proveer una vista lógica de alto nivel de la base de datos con muy pocos detalles físicos. Por último, los modelos semánticos contribuyeron ampliamente a la abstracción de datos, a través de una orientación de objetos y conceptos de abstracción como son la clasificación, agregación, generalización y asociación.

El objetivo principal de abstracción es denominado ***encapsulación***, el cual, fomenta el diseño de propiedades de los objetos por medio del aislamiento, al mismo tiempo, que motiva la integración de los diseños en un diseño total de la base de datos.

La encapsulación está inspirada en conceptos de lenguajes de programación, como es la ***modularidad***, proceso recientemente aplicado al diseño de bases de datos. En el pasado, los modelos de datos no incluían mecanismos para modelado. Ahora, las abstracciones de procedimientos y algunos conceptos como los tipos de datos abstractos y las clases se han adoptado para modelar propiedades dinámicas de las bases de datos. Siguiendo a la integración de los mecanismos de abstracción de procedimientos y datos, el modelado de bases de datos se orientará a un nivel conceptual más elevado denominado ***modelado conceptual***.

II.4.5. SOPORTE DE MODELADO.

Otro objetivo de los modelos de datos, es proveer conceptos de modelado, que sean familiares o naturales a los diseñadores, hay varias formas de llevar a cabo esto.

El ***modelado directo*** requiere conceptos de modelos de datos que representen directamente las propiedades de la aplicación. Idealmente, las propiedades inherentes en la aplicación son también restricciones inherentes del modelo de datos. Cabe mencionar que está mucho más sujeto al error la utilización de combinaciones de conceptos para representar una propiedad.

El ***modelado unificado*** se alcanza si hay una y sólo una forma en la cual una propiedad es representada. Esto está orientado a reducir el número de opciones de diseño presentadas por el modelo o la metodología.

El modelado directo y unificado intenta reducir la complejidad del modelo, sin embargo, en la práctica se sabe que también reduce la flexibilidad del modelo.

Un objetivo indiscutible para el modelo de datos es el ser expresivo, de manera tal, que al utilizarlo se pueda representar cualquier propiedad estática o dinámica de interés con un cierto grado de precisión. Como un ejemplo consideremos la expresividad de un modelo de datos para relaciones objeto. Los modelos primitivos no tienen tal concepto, en el caso de los modelos jerárquicos y de red, ofrecen medios para representar relaciones binarias uno a varios en árboles y redes que son determinadas *al tiempo de compilación*. En el caso del modelo relacional, este provee medios directos de representación para relaciones varios a varios, las cuales pueden ser definidas dinámicamente, sin embargo, no hay manera directa de asegurar que tales relaciones sean completas en su significado.

En los modelos semánticos existe el objetivo de proveer los medios necesarios para la representación de relaciones orientadas a aplicaciones, y asegurar su completez de significado, en este ámbito destacan las formas de implementar lo anterior, una de estas formas es proveer de un número pequeño de constructores (agregación, generalización, asociación y partición) con los cuales se pueden representar todas las relaciones, otra es proveer de un medio general para la definición de cualquier relación (lenguaje de primer orden).

II.4.6. DEFINICION PRECISA.

Los conceptos que constituyen un modelo de datos, deben ser definidos en forma precisa, lo que ayuda al usuario a la comprensión del modelo de datos, esto asegura la solidez de los conceptos del modelo de datos y la interacción con el mismo. Generalmente, los modelos de datos no son definidos formalmente, por lo tanto son difíciles de entender, aplicar, comparar y analizar. El principal objetivo del diseño de modelos de datos, incluye el proveer de medios para cumplir las siguientes tareas: dar consistencia a las restricciones de un esquema, dar completez con respecto a otras descripciones y evitar las propiedades no deseadas como son la redundancia y las anomalías de almacenaje.

II.4.7. COMPRENSION.

Existen dos objetivos para la conceptualización, comprensión, documentación y uso de un modelo de bases de datos, estos son la ***simplicidad*** y la ***esquematización***. La simplicidad es posible alcanzarla una vez que se provee de economía e independencia de conceptos a través de una notación sencilla.

El modelo relacional es un claro ejemplo de ese fenómeno. A pesar de la complejidad de los objetos y sus relaciones, es deseable representar de manera tangible y clara los esquemas y subesquemas, de forma tal, que faciliten el diseño y la documentación.

II.4.8. LENGUAJES Y METODOLOGIAS.

Un modelo de datos debe proveer de una base de herramientas (lenguajes) y técnicas (metodologías de modelado) de alto nivel, con las cuales se realicen y utilicen los conceptos de modelos de datos. También, es importante establecer que los lenguajes y metodologías deben interactúar de manera claramente definida.

II.4.9. FACTIBILIDAD.

Un modelo de datos debe ser alcanzable, por medio de la utilización de técnicas de implementación seguras y eficientes. El objetivo de la independencia de datos, ha sido separar dichas técnicas de los aspectos lógicos del modelo.

Muchas investigaciones se han avocado a la implementación de modelos primitivos y clásicos, sin embargo, muy pocas se han dedicado a llevar a cabo modelos semánticos de datos.

II.4.10. REPRESENTACION DE LOS COMPONENTES DEL MODELO DE DATOS.

Uno de los principales tópicos en la investigación de las bases de datos, es la representación de los componentes del modelo de datos. La investigación ha sido inspirada básicamente de los lenguajes de programación y de la inteligencia artificial.

En relación a los tipos de objeto, los modelos primitivos usan tipos de datos simples para definir campos. Los modelos jerárquico y de red usan tipos de datos simples para campos y segmentos de registro para las relaciones. El modelo relacional usa tipos de datos para atributos y relaciones.

Existen intentos por formular todos los conceptos de un modelo de datos semántico en términos de tipos de datos.

Los tipos de datos abstractos y las clases, se usan para la integridad semántica en forma precisa. Un reto en el desarrollo de las bases de datos, es el disminuir la separación entre los tipos de datos en el esquema y un vasto número de instancias. La mayoría de los modelos recientes tienen muchos más tipos de datos en los esquemas.

En relación a las restricciones, los primeros modelos no proveen mecanismos para representar todas las restricciones, es decir, generalmente no abarcan las dinámicas, por lo que es necesario utilizar medios fuera del modelo para representar restricciones, generando falta de uniformidad y dificultades para la definición, análisis y control de las restricciones, es decir, los mecanismos son ineficientes.

La investigación de modelos de datos ha intentando desarrollar medios para expresar restricciones de una manera uniforme que soporte definición, análisis y modificaciones, existe un esfuerzo considerable, el cual está dirigido a hacer la verificación de la integridad semántica a un costo mínimo. Una forma de alcanzar este objetivo, es hacer que la mayoría de las restricciones sean inherentes al modelo de datos, otro enfoque ha sido integrar conceptos de datos y procedimientos en mecanismos uniformes, utilizando resultados de lenguajes de programación e inteligencia artificial.

II.4.11. UTILIZACION DE CONCEPTOS DE LENGUAJES DE PROGRAMACION E INTELIGENCIA ARTIFICIAL.

Las investigaciones en inteligencia artificial, lenguajes de programación y bases de datos, comparten muchos objetivos como son: los conceptos de alto nivel, herramientas y técnicas, abstracción, etc. Sin embargo, existen diferencias significativas en aspectos muy importantes como el alcance de los problemas y la naturaleza de las soluciones.

Muchos de los componentes de los modelos de datos se han desarrollado a partir de los esquemas de representación de conocimiento, particularmente de las redes semánticas, de la misma manera, se ha hecho uso de los lenguajes de alto nivel en el proceso de conformación de interfaces. Por otro lado, cabe destacar que la lógica ha sido utilizada para la definición formal y adecuada de los modelos de datos y sus lenguajes.

Asimismo, la investigación en bases de datos ha hecho un uso intensivo de herramientas desarrolladas en el ámbito de los lenguajes de programación, como son: el concepto de tipo de datos, los tipos de datos abstractos, el concepto de clase, variables, diseño de lenguaje, conceptos de especificación y verificación, estructuras de control, abstracciones de datos y procedimientos, programación funcional y manejo de excepciones, sólo por mencionar algunos.

II.5. TENDENCIAS DEL DESARROLLO DE MODELOS.

Los tópicos mencionados anteriormente, orientan la investigación en modelos de datos, a continuación se describen aquellos que por su importancia innovadora tienen un interés particular.

II.5.1. EXTENSION DEL ALCANCE DE LAS APLICACIONES DE BASES DE DATOS.

Muchas de las aplicaciones de bases de datos involucran las situaciones tradicionales, las cuales son denominadas ***aplicaciones de negocios***, y que se caracterizan porque su información es uniforme y contiene un formato regular. Sin embargo, las bases de datos son útiles para el manejo de otro tipo de información.

Hoy en día, hay una motivación importante que tiende a soportar aplicaciones en las cuales la información está menos estructurada y puede contener tipos de datos mucho más complejos que los que ahora se soportan. Ejemplos de esas aplicaciones son; VLSI, automatización de oficinas y fábricas y toda la ingeniería de bases de datos (específicamente CAD/CAM), cartografía, bases de datos de fotografías y otras imágenes, aplicaciones científicas, aplicaciones económicas, inteligencia, aplicaciones de textos, bases de datos estadísticas, etc. Es decir, cualquier aplicación que soporte una cantidad considerable de información.

Las aplicaciones mencionadas requerirán mecanismos para el modelado de las aplicaciones semánticas asociadas, esto último, es la motivación más importante de la siguiente generación de los modelos de datos, la cual es denominada ***modelo de datos de propósito especial*** o ***modelos de datos orientados a aplicaciones***.

II.5.2. MEDIO AMBIENTE DE BASES DE DATOS.

Generalmente el desarrollo de una aplicación de bases de datos, involucra un ciclo de vida extensivo, el cual podría beneficiarse considerablemente del soporte automatizado. Por ejemplo, sería sumamente útil contar con herramientas que soportaran definición de necesidades, especificación, verificación (chequeo antes de la ejecución), validación (chequeo en la ejecución), manejo de excepciones, diseño lógico, modelado semántico de datos (definición de aplicaciones, transacciones y programación de peticiones), programación de alto nivel (escenarios para usuario final, guías), consideración de funcionalidad de las bases de datos en lenguajes de programación convencionales, interfaces hombre-máquina y múltiples modelos de datos y esquemas.

El objetivo es establecer un medio ambiente de base de datos, que a su vez sea, un medio ambiente de programación o ingeniería de software que soporte el modelado de los datos, utilizando herramientas y técnicas avanzadas a lo largo del ciclo de vida del desarrollo.

II.5.3. EXTENSIONES A LOS MODELOS DE DATOS Y A LOS LENGUAJES.

Muy pocos aspectos semánticos de los modelos de datos son problemas sin resolver, de ellos destacan; el tiempo de modelado, sucesos en el modelado, manejo de anomalías, conocimiento incompleto y el manejo de excepciones. Uno de los retos a los que se enfrentan los lenguajes, es aquel que involucra establecer una integración clara del modelo relacional y los conceptos semánticos jerárquicos. El modelo relacional tiene la propiedad de ***cerradura***, consistente en que al aplicar operaciones relacionales a relaciones siempre se obtienen relaciones. Esta propiedad se pierde si se aplican operaciones semánticas jerárquicas a las relaciones.

Hoy en día, no existe lenguaje de petición que soporte la abstracción de datos y que conserve las principales propiedades del modelo relacional. Otro problema al que se enfrentan los lenguajes, es aquel que involucra la integración de los lenguajes de petición con los conceptos de tipos de datos abstractos y clases, estos últimos son usados para el manejo de la actualización semántica.

Dos retos que se encuentran relacionados en el desarrollo de los modelos de datos son; reducir la distinción entre datos y esquemas, incrementando al mismo tiempo el número de tipos de objeto en el esquema en relación al número de instancias en la base de datos. El primer problema está orientado a permitir que los mismos conceptos del lenguaje se utilicen en las peticiones y actualización, tanto del esquema como de la base de datos. La reducción del intervalo de instancias es una consecuencia de un modelado más preciso, el cual puede ser llevado a cabo, al utilizar jerarquías de tipos, en lugar de tipos simples para representar objetos complejos.

II.5.4. METODOLOGIAS PRACTICAS DE MODELADO.

Las investigaciones en bases de datos se han apoyado de la inteligencia artificial, los lenguajes de programación y la ingeniería del software, por lo que los retos actuales consisten en integrar las ideas de esas disciplinas en forma bien definida y de manera efectiva. La extensión del alcance de las aplicaciones de base de datos, es quizás, la tendencia que más requiere de la integración mencionada. Las áreas de aplicación potenciales involucran; comunicaciones, sistemas expertos, interfaces de propósito especial, semánticas complejas, lenguajes de alto nivel de propósito especial y requieren de la ingeniería de software a lo largo de su ciclo de vida.

II.5.5. MODELADO CONCEPTUAL.

Existe la tendencia de dejar a un lado el modelado de aplicaciones con propiedades específicas, como pueden ser propiedades estáticas y dinámicas en las bases de datos y los lenguajes de programación. Como consecuencia de lo anterior, hoy en día, se tiende al modelado de todas las propiedades de una aplicación. Esto incluye el modelado de comunicaciones, interfaces hombre-máquina y medio ambiente de los sistemas, obviamente incluye las propiedades estáticas y dinámicas.

Los mecanismos de abstracción serán más necesarios que nunca, a fin de facilitar el manejo de una complejidad más amplia. Las observaciones que los investigadores han hecho, concluyen en que existe la necesidad de un modelado de nivel más alto, el cual se le ha denominado ***modelado conceptual***. Muchos de los modelos de datos que se encuentran en desarrollo, tienen el modelado conceptual como uno de sus objetivos.

III. *EXTENSIONES A LA TECNOLOGIA ACTUAL EN BASES DE DATOS.*

Los objetivos a cubrir y que conforman el reto que enfrenta la tecnología de las bases de datos, son los siguientes:

- Facilitar el desarrollo de las aplicaciones denominadas ***no estándar***, las cuales son mucho más complejas que las denominadas ***aplicaciones tradicionales, administrativas o de negocios***, campo de acción al que se ha enfocado la tecnología convencional de los sistemas de bases de datos.

- Proveer de mecanismos para la manipulación completa de la ***información*** y el ***conocimiento*** almacenados en la base de datos.

- Integrar al ámbito de las bases de datos los nuevos tipos de aplicación como son:

  . **Sistemas CAD/CAM.**

  . **Sistemas para el manejo más inteligente de documentos.**

  . **Sistemas de entrenamiento o capacitación.**

  . **Cartografía.**

  . **Planeación robótica**, etc.

Para alcanzar los objetivos mencionados, la comunidad científica ha pensado en extender o rediseñar completamente los sistemas actuales de bases de datos.

III.1. NECESIDADES DE LAS APLICACIONES A INCORPORAR EN EL CONTEXTO DE LAS BASES DE DATOS.

Los nuevos sistemas de bases de datos, necesitan ofrecer conceptos a fin de satisfacer las nuevas necesidades que surjan en las nuevas aplicaciones, de manera que renueven la interacción con el usuario y sean más flexibles.

Dentro de los nuevos conceptos con que deben contar los nuevos sistemas de bases de datos, se pueden mencionar los siguientes:

III.1.1. NUEVOS TIPOS DE DATOS Y OPERACIONES.

Las ***aplicaciones no estándar***, necesitan del manejo de nuevos tipos de datos, hasta ahora no incluidos en las aplicaciones tradicionales. Lo anterior se justifica una vez que cierto tipo de información (mapas, imágenes, dibujos, textos, documentos estructurados, etc.) requiere ser representado dentro del manejador de bases de datos como tipos de datos.

En algunos casos, como el manejo de texto de grandes dimensiones, los datos son almacenados en una cadena, con ninguna otra representación dentro del sistema, en otras situaciones, los nuevos tipos de datos son almacenados en dispositivos separados y se extraen cuando se hace necesaria su utilización, este último proceso, generalmente es arduo.

De lo anterior se desprende, que en este tópico el problema reside en asegurar la integridad de los datos y proveer en el manejo de la información de un tiempo de respuesta razonable.

Por otra parte, la nueva generación de sistemas de bases de datos, debe proveer directamente al usuario de nuevos tipos de datos y debe dotar de facilidades para la definición de los datos que el usuario en el futuro requiera.

La introducción de nuevos tipos de datos, implica además, involucrar nuevos tipos de operaciones con ellos.

Para cada nuevo campo de aplicación, se puede definir un conjunto especial de operaciones o transacciones, las cuales se aplican sobre un tipo específico de datos, ambos conceptos, tipos y operaciones, deben ser sustentados por el sistema de bases de datos. Por ejemplo:

. Operadores especiales, para el manejo de textos grandes, o para la comparación de imágenes.

. Nuevas operaciones como las ***cerraduras transitivas*** para el ***proceso recursivo de reglas***, concepto básico en las aplicaciones basadas en esquemas de conocimiento.

III.1.2. FACILIDADES PARA LA REPRESENTACION DEL CONOCIMIENTO.

Los nuevos sistemas de bases de datos tienen que ***explotar*** el conocimiento almacenado en la base de datos, esto toma relevancia, si hablamos de ***aplicaciones basadas en esquemas de conocimiento***, así mismo, tienen que poseer facilidades para soportar diferentes ***formalismos de representación de conocimientos.***

III.1.3. NUEVOS MECANISMOS DE CONTROL DE CONCURRENCIA Y DE RECUPERACIÓN.

Cada nueva área de aplicación tiene sus propias necesidades en este renglón, las cuales, son diferentes a las necesidades tradicionales.

El manejo de grandes transacciones con objetos complejos, requiere de mecanismos de control adecuados para soportar la manipulación simultanea de varios usuarios, asegurando la integridad de la información.

La recuperación, es una característica tradicional en los sistemas de bases de datos, sin embargo, los mecanismos actuales tienen que ser extendidos, a fin de manejar objetos complejos desde diversos puntos de vista.

III.1.4. NUEVOS METODOS DE ACCESO Y NUEVAS ESTRUCTURAS DE ALMACENAMIENTO.

Las nuevas áreas de aplicación requieren del desarrollo de nuevos métodos de representación física y de nuevos métodos de acceso a la información, de los resultados obtenidos de la investigación más reciente, sobresalen:

. Los *árboles R* [GUTT 84] usados para acceder datos espaciales.

. Los *árboles KDB* [ROBI 81] y los *árboles Grid* [NIEV 84], métodos que permiten tanto el acceso rápido a un objeto en un espacio multidimensional, como la determinación de objetos traslapados.

III.2. TENDENCIAS DE LA INVESTIGACION.

Al considerar las necesidades de las aplicaciones que se desea incorporar al ámbito de las bases de datos, es válido concluir que los sistemas actuales de bases de datos son inadecuados para llevar a cabo dicha incorporación.

Se han definido algunas tendencias para mejorar la tecnología actual de los sistemas de bases de datos, a continuación se hace una descripción rápida de los esfuerzos innovadores que se están realizando.

III.2.1. EXTENSIONES A LOS SISTEMAS ACTUALES DE BASES DE DATOS.

El proceso de extender los sistemas de bases de datos considera principalmente los *sistemas relacionales.*

Este enfoque, básicamente provee varias extensiones a la base teórica del modelo, a fin de mejorar el poder y flexibilidad del mismo, sin embargo, retiene las ventajas originales:

. La *simplicidad.*

. La existencia de un lenguaje *no de procedimiento* para el proceso.

La solución que se le ha dado al problema, puede clasificarse de acuerdo a la forma en que se llevan a cabo las implementaciones:

. En el primer caso, las extensiones se han desarrollado agregando una capa extra al sistema existente.

. En el segundo caso, la base de datos es fortalecida con nuevos componentes, algoritmos y técnicas de almacenamiento, en el interior mismo del sistema.

Los resultados obtenidos se diferencían principalmente en el rendimiento del sistema modificado.

III.2.2. DESARROLLO Y UTILIZACION DE MODELOS SEMANTICOS DE DATOS.

Este enfoque, tiende a definir y utilizar un modelo de datos de mayor nivel, el cual se encuentra más cercano a las necesidades de las aplicaciones complejas que al modelo relacional. Estos modelos, no son comparables con el modelo relacional, ni tampoco con los modelos relacionales modificados, pero pueden considerarse como modelos conceptuales de alto nivel, los cuales hasta ahora, solo es posible implementar apoyándose en el modelo relacional.

Existen ciertos inconvenientes con este tipo de modelos, muchos de ellos sólo han sido definidos teóricamente, muy pocos han logrado implementarse.

III.2.3. SISTEMAS EXTENDIBLES DE BASES DE DATOS.

Este enfoque, pugna por el desarrollo de un sistema básico o central de bases de datos (*KERNEL*), el cual pueda ser extendido fácilmente para satisfacer las necesidades de las nuevas aplicaciones.

Las características de este tipo de sistemas, consisten en el hecho de compartir funciones, así como, el dotar de facilidades para que las estructuras específicas de datos y las operaciones puedan ser definidas, basándose en la base de datos común.

Por otra parte, el desarrollo de la tecnología de las bases de datos, se ha visto influenciada por otras disciplinas de la computación, como la Inteligencia Artificial y los Lenguajes de Programación, por lo tanto, algunas arquitecturas de bases de datos han sido diseñadas con el objetivo de integrar *paradigmas* de dichas disciplinas, encontrandose prototipos desarrollados bajo este esquema.

Los sistemas pertenecientes a este grupo, pueden clasificarse en dos ramas:

. Sistemas de Bases de Datos Orientados a Objetos

Los cuales hacen uso de los resultados obtenidos en el área de orientación de objetos, especialmente, en lenguajes de programación.

. Sistemas Deductivos de Bases de Datos

Los cuales se caracterizan por el uso del trabajo realizado en programación lógica e inteligencia artificial, y corresponden al esfuerzo de incorporar conocimiento o semántica en el ámbito de las bases de datos.

III.3. AVANCES DE LA INVESTIGACION.

III.3.1. LOS MODELOS DE RELACIONES ANIDADAS.

Un enfoque en la definición de modelos de datos, es aquel que tiende a minimizar las extensiones al modelo relacional, a fin de retener la mayoría de sus ventajas. En este contexto caben los trabajos que relajan la *primera forma normal de las relaciones* (*1FN*) [MAKI 77], [SCHE 82], [ABIT 84], [FISH 83].

El relajar la primera forma normal, corresponde a la existencia de un tipo de relaciones no normalizadas denominadas *relaciones anidadas*.

Una *relación anidada* permite como componentes de una *n-eada*, instancias de una relación en lugar de instancias de valores atómicos.

Para el nuevo tipo de relaciones, se han definido operaciones adicionales a las operaciones estándar del álgebra relacional, dos de ellas son las denominadas *anidar* y *desanidar*.

Esencialmente, las operaciones anidar y desanidar permiten pasar de *relaciones planas* a cierto tipo de relaciones jerárquicas y viceversa.

La ventaja básica del modelo relacional anidado, es su simplicidad, una vez que es muy cercano al modelo relacional.

Muchos estudios teóricos se han desarrollado sobre el modelo relacional anidado, y su tendencia es extender los resultados de la teoría relacional al modelo anidado. Además, se han llevado a cabo esfuerzos de implementación, uno de ellos; el NF^2 desarrollado por IBM en Heidelberg, que es utilizado para experimentación en las áreas de robótica, automatización de procesos y manejo de documentos de oficina.

Otros prototipos fundamentados en el principio básico de relaciones anidadas, solo han sido orientados a experimentación.

Los resultados de las primeras investigaciones, indican que el modelo relacional anidado, no es suficiente para capturar todas las características de las aplicaciones no estándar, por lo tanto, el modelo de relaciones anidadas, se ha extendido y modificado para captar las características de las aplicaciones no cubiertas con el modelo original de relaciones anidadas, de las extensiones propuestas se mencionan las siguientes:

. **Nuevos tipos de atributos estáticos** (n-eadas, conjuntos, multiconjuntos y listas).

. **Incorporación de orden entre los componentes de una n-eada.**

. **Consideración de duplicación de multiconjuntos y listas.**

. **Llaves sobre atributos multivaluados.**

. **Nuevos valores nulos.**

. **Definición de operaciones más poderosas** para la reestructuración, en sustitución de anidar y desanidar.

. **Incorporación de una operación de cerradura**, la cual se aplica sobre una expresión algebraica e itera la evaluación de la expresión hasta que el resultado alcanza un punto fijo.

. **Incorporación de álgebras multitipificadas en lugar de álgebras monotipificadas**, un álgebra multitipificada, implica que las operaciones aritméticas y las funciones agregadas puedan ser manejadas por el álgebra de manera uniforme.

. **Definición de extensiones al lenguaje SQL.**

Dentro de los modelos que incorporan algunas de la extensiones mencionadas, se encuentran:

. **NF^2 extendido** desarrollado en IBM Heidelberg **[DADA 86]**.

. **NST** desarrollado en IBM San José **[GUET 87]**.

. **Algres** del Politécnico de Milán **[CERI 88]**.

Todos ellos han servido como base formal para la implementación de prototipos. Los resultados de la experimentación de tales sistemas con aplicaciones de la vida real, se encuentran aún en experimentación.

Desde el punto de vista práctico, el modelo básico no es suficiente para cubrir las necesidades de las nuevas aplicaciones, por lo tanto, se han propuesto extensiones al modelo básico que tienden a obtener un modelo más complejo, sin embargo, el resultado parece ser que no cumple aún con las necesidades mencionadas, entre ellas, el compartir datos y el modelado de información que por su naturaleza no es jerárquica (texto, gráficas, imágenes). Se espera que los resultados con la experimentación de prototipos existentes, darán una respuesta adecuada a tales inquietudes.

III.3.2. TIPOS DE DATOS DEFINIDOS POR EL USUARIO.

Los sistemas relacionales fueron desarrollados para satisfacer las necesidades de las aplicaciones tradicionales, el tipo de valores que son manipulados en esta clase de aplicaciones es muy simple, más aún, se limita al manejo de enteros, reales y cadenas de caracteres, por lo tanto, los límites de los modelos relacionales, se alcanzan cuando dichos modelos se intentan utilizar en una nueva clase de aplicación, puesto que los tipos de los valores que se manipulan en las nuevas aplicaciones son más complejos y específicos.

Al conocer algunas de las necesidades de la nuevas aplicaciones, resulta no práctico tratar de incorporar dentro del sistema relacional de bases de datos los tipos de datos que son necesarios para la nueva aplicación.

Una solución al problema anterior, es dejar al usuario la definición de los tipos de datos necesarios para su aplicación, aumentando el poderío del sistema de bases de datos, mientras éste continúa siendo completamente relacional.

Existen trabajos teóricos que argumentan que los tipos complejos pueden descomponerse en tipos simples por medio de la normalización [CODD 70], sin embargo, el hacer esto, mucha de la semántica de los datos se pierde y los lenguajes de petición no pueden ser utilizados para expresar gestiones completas de información.

ADT-Ingres y **Sabrina**, son dos sistemas que se avocan a la extensión del dominio relacional e incluyen *tipos de datos abstractos* (*ADT* por sus siglas en inglés), los cuales son dominios definidos por los usuarios, los tipos de datos abstractos son almacenados en la base de datos como cadenas de caracteres, y dan un nuevo poder a los sistemas relacionales de bases de datos, sin embargo el costo es muy alto si se utilizan lenguajes convencionales, esto último, se debe a la existencia de un proceso de codificación y decodificación de los tipos de datos abstractos.

La tendencia es reconsiderar algunos aspectos de los manejadores de bases de datos, a fin de facilitar la construcción y operación de tipos abstractos, de la misma manera, se intenta mejorar los lenguajes de alto nivel tipo **SQL** para incluir tipos de datos abstractos, por último se requiere de implementar nuevos métodos para optimizar la recuperación de las n-eadas conteniendo datos complejos.

Si un programa es utilizado para implementar operadores de tipos de datos abstractos, debe ser compilado o interpretado, si los programas con tipos de datos abstractos son compilados, entonces el código que implemente el tipo de datos abstractos, debe ser ligado al código del sistema manejador de bases de datos. Por otra parte, los programas pueden ser interpretados por un medio especialmente diseñado y que se anexa al sistema manejador de bases de datos. Por ejemplo, **ADT-Ingres** usa una versión compilada del lenguaje **C** para implementar tipos de datos abstractos. A pesar de las versiones compiladas, los errores de los programas de los usuarios no pueden manejarse fácilmente, el código es almacenado en archivos secuenciales y debe ser ligado de forma dinámica al programa del sistema manejador de bases de datos.

Debido a la necesidad de especificar el tamaño de la representación física de un tipo abstracto de datos, es muy difícil implementar generadores de tipos como conjuntos y listas los cuales pueden ser de tamaño variable.

III.3.3. SISTEMAS DE BASES DE DATOS FUNDAMENTADOS EN MODELOS SEMANTICOS.

Los críticos del modelo relacional argumentan que parte importante de la semántica de la información se pierde cuando es descompuesta en relaciones normalizadas. En respuesta a lo anterior, los nuevos sistemas de bases de datos que se diseñan se orientan a capturar más del significado de los datos, estos nuevos tipos de sistemas de bases de datos se les conoce como ***sistemas semánticos***.

De los esfuerzos por definir un nuevo modelo de datos que sea más cercano al mundo real, se pueden mencionar los siguientes:

. El modelo binario [ABRI 74].

. El modelo entidad-relación [CHEN 76].

. El modelo de agregación y generalización de datos [SMIT 77].

. El modelo relacional extendido [CODD 79], [LORI82].

. El modelo lógico de datos [KUPE 86].

De los sistemas que se basan en un modelo semántico, destacan:

. DAPLEX [SHIP 81].

. GEM [ZANI 83].

. IFO-fragments [ABIT 87].

Los sistemas semánticos de bases de datos, pueden construirse sobre sistemas existentes, la dificultad de realizar lo anterior, estriba en el grado de diferencia entre la representación base del modelo del sistema manejador de bases de datos y el modelo semántico de datos. Un proceso especial de transformación, debe convertir objetos de su representación en el modelo semántico a una forma compatible con el modelo soportado por el modelo base. La mayoría de los modelos son diferentes, por lo que el proceso de transformación es complicado y el rendimiento del mismo es bajo.

El sistema de bases de datos GEM es un ejemplo de un sistema semántico de bases de datos, el cual se construyó sobre un sistema relacional de bases de datos, el denominado modelo entidad-relación. Las principales ideas de GEM son soportar las nociones de dependencia, agregación, generalización, valores nulos y conjuntos. Un prototipo en función de GEM se ha construido en una máquina IDM500, el principal problema de esta implementación reside en su bajo rendimiento.

III.3.4. SISTEMAS DE BASES DE DATOS ORIENTADOS A OBJETOS.

Los *sistemas de bases de orientados a objetos*, aplican el *paradigma de programación orientado a objetos*, el cual es un desarrollo superior del concepto de *clase* introducido por Simula [BIRT 73]. Este paradigma trata con los conceptos como *clase* e *instancia*, así mismo, se relaciona con los métodos de agregación y generalización.

Los conceptos básicos del paradigma de orientación a objetos son:

. *Clases de objetos*, concepto que permite de forma fácil la declaración de las propiedades y comportamiento de los objetos.

. *Relaciones Parte de*, el cual reune objetos como parte de un objeto en la aplicación.

. *Compartimiento de objetos*, establece que los objetos agregados sólo existen una vez.

. *Relaciones Es una*, que provee un refinamiento jerárquico de los objetos.

. ***Herencia***, herencia simple o múltiple, la cual mantiene una jerarquía *es una*.

. ***Métodos***, los cuales son expresiones típicas del comportamiento de los objetos.

. ***Mensajes***, los cuales se envían para activar los métodos.

La evolución del paradigma orientado a objetos se ha llevado a cabo en las siguientes áreas de la investigación:

. **Representación de conocimiento** en la Inteligencia Artificial, en donde se han desarrollado métodos como *redes semánticas* y *estructuras* tipo **KL-ONE**, de manera que se puede hablar de la *"encarnación computacional de las redes hereditarias estructuradas"* [BRAC 85].

. **Lenguajes de programación**, en donde por medio de la instancia **Smalltalk** [GOLD 83], las partes orientadas a objetos de una herramienta de implementación y de desarrollo de lenguajes como **Hypertext** llevan a cabo el paradigma de orientación a objetos.

. **Desarrollo de sistemas** como el **Apple Macintosh** o el **XEROX Star**, que son implementaciones orientadas a objetos.

. **Tecnología de Bases de Datos**, en donde se soporta el paradigma de orientación a objetos a través de modelos de datos para diversas aplicaciones.

Las bases de datos orientadas a objetos, proveen de todas esas utilerías en diferentes niveles. Por ejemplo, **GemStone** [MAIE 88], **O_2** [BANC 87] o **DAMOKLES** proveen a los usuarios de la oportunidad de construir sus propios objetos, de las siguientes formas:

. **GemStone** soporta Smalltalk.

. **DAMOKLES** provee de un lenguaje de definición de datos específico.

. **O_2** ofrece constructores de conjuntos y n-adas.

Todos los sistemas mencionados, permiten el *uso recursivo de la operación de construcción*, por lo tanto, es válida la existencia de objetos estructurados jerárquicamente.

La *agregación* y el *compartir objetos*, se proveen de diversas maneras, en general, las técnicas utilizadas se fundamentan en apuntadores. En los sistemas existentes, la *herencia* es *simple* en todos los que soportan **Smalltalk**, sin embargo, para soportar tecnología de sistemas basados en conocimiento, es necesario incorporar el concepto de *herencia múltiple*, lo cual permite la *generalización* en redes en lugar de árboles, esta generalización es más realista, ya que la representación del conocimiento en Inteligencia Artificial no se encuentra englobada en la herencia simple.

Los *métodos* son provistos por todos los sistemas mencionados, pero hay controversia en cuanto a la clasificación de los métodos en ***estáticos*** o ***dinámicos***, mientras la clasificación estática provee más libertad y eficiencia, la dinámica parece ser más amistosa al usuario.

Hoy en día existen cuestiones abiertas en los siguientes tópicos:

. *Recuperación.*

. *Control de concurrencia.*

. *Transacciones anidadas.*

. *Objetos dentro del paralelismo.*

En general, no se sabe exactamente como debe definirse el concepto de transacción, sin embargo, es obvio que el control de concurrencia y recuperación deben ser reconsiderados para este enfoque.

III.3.5. SISTEMAS DEDUCTIVOS DE BASES DE DATOS.

Una de las áreas de estudio más importante, consiste en integrar a las bases de datos conceptos provenientes de la Inteligencia Artificial, lo cual significa, que debe proponerse una nueva interface, a fin de incorporar a las bases de datos la definición de un conocimiento común y compartido por las diversas aplicaciones, los esfuerzos en este sentido se ven reflejados en las ***bases de datos deductivas***, que son ***paradigmas fundamentados en reglas.***

El conocimiento incluido en la base de datos, se expresa a través de reglas generales, las cuales definen, deducen o derivan sucesos de los hechos almacenados en la base de datos.

Mientras los hechos almacenados en la base de datos constituyen la ***base de datos extensional***, las reglas definen la ***base de datos intensional.***

El enfoque más común para integrar un lenguaje basado en reglas, es la utilización de un lenguaje lógico de programación como el lenguaje de **Cláusulas de Horn**.

Los diseñadores se han avocado a la construcción de sistemas de bases de datos deductivos prácticos, el primer problema al que se enfrentan dichos diseñadores, consiste en la elección y estructuración de un ***lenguaje de reglas***, después de eso, el problema más importante es el relativo al diseño e implementación de ***algoritmos de inferencia*** más eficientes a los utilizados en el ámbito de los lenguajes lógicos de programación, si se considera la gran cantidad de información que se maneja a través de un sistema de bases de datos, este problema es sumamente importante.

Un sistema deductivo de bases de datos, ante todo es un sistema de bases de datos, por lo que debe soportar todos los conceptos que usualmente soporta un sistema de bases de datos, debido a esto, las nociones de integridad física, integridad lógica, concurrencia, definición de datos, y manejo de lenguajes, deben ofrecerse en un sistema deductivo, al mismo tiempo, que se incorporan las nuevas funciones las cuales son; definición de *relaciones derivadas* y *relaciones virtuales* por medio de un lenguaje de reglas, de lo anterior se establece que el lenguaje puede ser visto como una extensión de la instrucción de definición de vistas de un manejador convencional de bases de datos. Las nuevas funciones de los sistemas de bases de datos deductivos, pueden resumirse como sigue:

. *Capas múltiples de relaciones virtuales*.

. *Recursión* que permita la definición de una regla en función de sí misma.

. *Negación* que facilite la expresión de hechos no existentes.

. *Actualización de relaciones múltiples*.

. *Meta-reglas* y disposición de reglas en módulos sucesivos.

El objetivo es claro, consiste en que el proceso de programación permita el compartir las reglas de conocimiento entre varios usuarios.

Existen varios enfoques para extender y hacer deductivo un sistema convencional de bases de datos, el primer enfoque denominado *acoplamiento débil*, consiste en empezar por un lado con un sistema de base de datos ya existente y por otro lado con un lenguaje lógico de programación también ya existente, se construye un interface que ofrece las facilidades del lenguaje lógico al sistema de base de datos. Los sistemas que se obtienen al utilizar acoplamiento débil, están lejos de satisfacer los objetivos de los manejadores deductivos de bases de datos, concretamente, ciertas funciones no existen, hay dos tipos de sucesos (los hechos de la base de datos y los hechos del lenguaje lógico), las reglas no se integran en una base de conocimientos, los lenguajes de petición son diferentes para las relaciones básicas y las relaciones deducidas, asimismo, el rendimiento de este tipo de implementaciones es generalmente deficiente. La optimización de las peticiones a los sistemas de bases de datos y la optimización del número de n-adas que se utiliza para el proceso de inferencia, se encuentran completamente bajo el control de las reglas del programador. Existen prototipos y algunos productos en el mercado que se han construido bajo el enfoque de acoplamiento débil.

El segundo enfoque, el cual es denominado *acoplamiento fuerte*, consiste en modificar o extender un intérprete de reglas como **PROLOG**, para soportar inferencia sobre predicados de la base de datos. El intérprete se construye como una capa sobre el sistema de base de datos existente, recupera hechos en la base de datos cuando es necesario el proceso de inferencia, más aún, se pueden desarrollar técnicas que optimicen la eficiencia de los lenguajes de petición. Es posible extender el intérprete de reglas con métodos específicos de acceso, a fin de realizar ciertas inferencias de manera más eficiente, por ejemplo las reglas recursivas.

El último enfoque, es denominado ***integración***, en este ámbito el subsistema de inferencia se encuentra completamente integrado al sistema de base de datos. La integración requiere dominar el código fuente del sistema manejador de base de datos existente. El sistema de base de datos se modifica para incluir un manejador de reglas básicas, se agregan al mismo tiempo, un lenguaje de peticiones en lugar de un interprete de reglas y los operadores específicos que traten con reglas calculadas.

El álgebra relacional debe extenderse con funciones u operaciones de punto fijo. Este enfoque permite al usuario obtener interfaces integradas y buenos rendimientos. En general, el lenguaje de definición de reglas es una base de datos orientada y debe soportar peticiones y actualizaciones.

En la actualidad, el enfoque usual es la utilización de un lenguaje lógico de programación como **DATALOG extendido [APT86]** para soportar negación, conjuntos y actualizaciones, sin embargo, hay aún problemas sin solución, por ejemplo, no hay lenguaje completo que provea todas las utilerías mencionadas.

La optimización de los lenguajes de petición, que soporten predicados derivados, sigue siendo un problema complejo. Se han propuesto varios enfoques para tratar con reglas recursivas, clasificandose de acuerdo al tiempo en que se realiza el proceso de optimización. Los métodos como la **Evaluación ingenua** (Naïve), **Evaluación semingenua (Seminaïve) [BANC 86a]** y el **Método Funcional [GARD 86]**, producen como salida un programa de álgebra relacional, el cual se supone eficiente y se ejecuta contra la base de datos. Otros métodos como los **Conjuntos Mágicos [BANC 86b]**, **Alexander** y **Métodos Algebraicos [CERI 86]**, son denominados métodos de reescritura de términos y producen como salida un programa de reglas reescrito, el cual es generalmente más eficiente. Cabe mencionar que no existe todavía una implementación con aplicaciones reales que permita comparar los diferentes métodos.

La traducción de programas de reglas grandes a álgebras relacionales extendidas, debe usarse como técnica de optimización incluyendo los procesos de optimización global y física.

Algunos operadores especializados se han diseñado para tratar eficientemente con algún tipo de peticiones recursivas, como son las cerraduras transitivas y las operaciones restringidas a punto fijo **[PUCH 87]**. Otros problemas importantes siguen vigentes, como son: verificación de consistencias de reglas, manejo de meta-reglas y manejo de restricciones entre predicados derivados. Finalmente, las interfaces involucradas y el medio ambiente de programación, deben estar integrados en un prototipo avanzado.

III.3.6. SISTEMAS EXTENDIBLES DE BASES DE DATOS.

Los sistemas que pertenecen a este tipo, son núcleos extendibles en lugar de sistemas completos de bases de datos. El objetivo de estos sistemas, es soportar el desarrollo de sistemas especializados basados en modelos de datos particulares. Para este propósito, se ofrece un núcleo básico de bajo nivel, que parece ser una caja de herramientas, la cual puede ser ajustada a las necesidades específicas de cada aplicación.

EXODUS [CARE 86], **GENESIS** [BATO 86] y **GEODE** [PUCH 87], son ejemplos de este tipo de sistemas, es importante notar que el uso de núcleos está orientado a desarrolladores de sistemas de bases de datos, en lugar de desarrolladores de aplicaciones.

Como en el caso de **GEMSTONE** y **O_2**, los tipos definidos por el usuario se construyen por medio de la aplicación recursiva de constructores a tipos básicos o a otros tipos de datos definidos por el usuario.

El enfoque seguido por un sistema de datos extendible, es similar al enfoque propuesto por **POSTGRES**, en el cual el programador tiene a su disposición métodos de acceso genéricos ajustables, y una interface de bajo nivel que puede utilizarse para construir su propio método de acceso, dejando únicamente al sistema el manejo de los problemas como el control de concurrencia y la recuperación en caso de siniestro.

En el sistema **GENESIS**, los algoritmos complejos que constituyen el sistema de bases de datos, son sistemáticamente descompuestos en anidamientos de algoritmos primitivos, con su respectiva evaluación de costo. Además, **GENESIS** respeta la distinción entre operación lógica y operación física. La suma de esos conceptos permite la construcción de un optimizador modular y manejable, por ejemplo, la agregación de un método de acceso basado en índices dispersos, implica la agregación de una operación física con su función asociada de costo, la cual corresponde a la operación lógica de selección.

El manejo de transacciones, es una de las especialidades de estos sistemas. **EXODUS** aplica la periodicidad de actualizaciones de objetos primitivos, los cuales son representados por secuencias variables de bytes. Un objeto estructurado o complejo, puede descomponerse en un conjunto de objetos primitivos, este método permite un incremento periódico similar al encontrado en los sistemas relacionales extendidos. El bloqueo de objetos propuesto para el manejo de grandes transacciones, está inspirado en el modelo de transacciones anidadas. **GEODE** ofrece una estrategia de periodicidad ajustable que toma en consideración la semántica de los operadores definidos por el usuario que manipulan los objetos almacenados. La base de datos implementada puede elegir en actualizar solo bajo ciertas acciones conocidas, de tal manera, que el conjunto de operaciones de actualización puede encontrarse activando las acciones vivas. Un mecanismo similar se utiliza para el manejo de la concurrencia, los objetos primitivos son sistemáticamente bloqueados en cada acceso durante la fase de prototipo, este mecanismo se optimiza por medio de una estrategia selectiva de bloqueo, dicha estrategia evita en ciertos casos el bloqueo de objetos de tipo índice. En realidad los bloqueos están asociados a los identificadores de los objetos. Un bloqueo recursivo adicional puede ser aplicado al bloqueo de cada objeto primitivo, constituyendo un objeto lógico.

Los sistemas extendibles, se diseñan para satisfacer todas las necesidades descritas, aparte de las necesidades que surgen de la experimentación de los sistemas.

El problema principal de los sistemas en este ámbito, consiste en la falta de modelos de datos que faciliten su asociación con un nivel de conocimiento más alto, este tópico de investigación esta aún abierto.

III.3.7. DISEÑO DE HERRAMIENTAS PARA APLICACIONES BASADAS EN CONOCIMIENTO.

El diseño de base de datos, es un proceso largo y tedioso sin la ayuda de la computadora. Muchas herramientas de diseño han sido construidas, pero están lejos de resolver todos los problemas de diseño de las bases de datos, especialmente cuando a esos problemas se añaden los problemas adicionales del diseño fundamentado en conocimiento.

Se pueden distinguir cuatro categorías de herramientas, las cuales están caracterizadas por la metodología y los modelos que pueden soportar, a continuación se describe brevemente cada una de dichas categorías.

. La primera categoría de herramientas, consiste en modelos de datos que son más o menos formalizados, como son, los modelos semánticos de datos. Estos modelos intentan capturar la semántica del mundo real con más precisión y naturalidad. Generalmente son soportados por algunas herramientas de graficación, las cuales algunas veces mantienen un diccionario. Estas herramientas, son muy interesantes para aplicaciones pequeñas, pero son obsoletas con aplicaciones grandes sin ayuda de diseño automatizado.

. La segunda categoría de herramientas, provee un conjunto de algoritmos que permiten la producción de un esquema normalizado a partir de un conjunto de atributos y dependencias, por otro lado, es factible establecer el mapeo de una representación a otra. Los algoritmos de síntesis y descomposición, constituyen la contribución principal en el primer caso, las reglas para el mapeo de esquemas y vistas, constituyen la contribución del segundo. Este enfoque es uno de los más formalizados, desafortunadamente, su aplicación práctica es limitada.

El utilizar el modelo relacional como una herramienta de diseño conceptual está en controversia, una vez que los conceptos relacionales son simples de usar pero no son suficientes para capturar la semántica de las aplicaciones de los usuarios.

. La tercera categoría de herramientas, consiste de un conjunto de programas interactivos que son considerados como una combinación de herramientas manuales y algorítmicas. Casi siempre, se diseñan como herramientas **CAD/CAM**, las interacciones entre los usuarios y el sistema, se orientan al proceso ***pregunta-respuesta*** o por medio de lenguajes gráficos.

Estas herramientas son muy interesantes, pero continúan siendo demasiado estáticas puesto que las ayudas computarizadas se programan para siempre, es difícil modificarlas para agregar nuevas reglas o eliminar algunas obsoletas.

. La cuarta categoría de herramientas, es denominada como categoría de herramientas expertas de diseño. Generalmente son sistemas expertos construidos sobre una base de conocimiento, que a su vez contiene varios modelos de datos, reglas de diseño, algoritmos y conocimientos experimentales.

Las interfaces de alto nivel como los lenguajes naturales y las gráficas, reducen la brecha semántica entre la percepción del mundo real y su representación abstracta en un modelo de datos, su capacidad de generar varias soluciones y explicaciones sobre el proceso de diseño, los hacen atractivos para su uso de manera intensiva, generalmente, enfatizan las características del diseño hasta ahora ignoradas por otras herramientas. Hay dos ejemplos comerciales de este tipo de herramientas, **EXSYS [HOLS 82]** y **SECSI [BOUZ 83]**.

En general, la mayoría de las herramientas han sido utilizadas para el aspecto estático de las aplicaciones (estructura de la base de datos), aunque algunos modelos dinámicos para representar el comportamiento de la base de datos han sido ya propuestos, el diseño de ayudas automatizadas en este campo es muy restringido.

APENDICE A. *NF^2, UN MANEJO INTEGRADO DE TABLAS PLANAS Y TABLAS JERARQUICAS.*

Las extensiones más recientes para los sistemas manejadores de bases de datos, han sido propuestas con el fin de soportar estructuras jerárquicas (***objetos complejos***), y el implementar dichas extensiones se ha realizado sobre algún sistema de bases de datos existente, lo que conlleva a muchas desventajas, no solo desde el punto de vista conceptual, también implica problemas en el aspecto de rendimiento del sistema modificado, en esta parte, se revisa el esfuerzo realizado para implementar un prototipo de sistema de bases de datos que soporte un modelo generalizado de datos denominado ***NF^2 extendido***, el cual trata con relaciones planas, listas y estructuras jerárquicas de una manera uniforme.

Los puntos a tratar son:

. El modelo de datos lógico.

. Un lenguaje para dicho modelo y

. Estructuras de almacenamiento que permitan la implementación de las relaciones generalizadas.

A.1. ANTECEDENTES.

Debido a la creciente demanda de alta calidad de los productos y de la necesidad de incrementar la productividad, el número de sistemas instalados para el diseño computarizado (**CAD**) y la manufactura computarizada (**CAM**), ha crecido en los últimos años, para manejar la información este tipo de sistemas utiliza por lo regular únicamente el sistema de archivos que dota el sistema operativo que los sustenta. Sin embargo, tanto el número de sistemas instalados, como el número de personas que los utilizan crece, entonces, la necesidad de interactuar entre sistemas se ha incrementado también, por lo que este tipo de sistemas comienza a tener problemas como la **falta de independencia de datos, recuperación insuficiente** de información y **falta de controles de concurrencia**, estos problemas fueron encarados hace más de dos décadas por los usuarios de las áreas de datos comerciales o de negocios.

La búsqueda de soluciones para los problemas mencionados arriba, desembocó en el desarrollo de los sistemas manejadores de bases de datos, los cuales hoy están instalados prácticamente en todas las áreas de proceso de datos comerciales o de negocios. Actualmente, existe un fuerte interés en conjuntar en un solo medio ambiente integrado de manufactura (**CIM**), las tareas de CAD, CAM y la administración de negocios, CIM requiere sin embargo, que todos los tipos de datos (al menos los más relevantes) estén accesibles en un mecanismo integrado y controlado, es obvio que este problema solo puede ser solucionado a través de la aplicación de la tecnología de bases de datos.

Para el área de administración de negocios, la tecnología más avanzada se encuentra disponible, o al menos de forma parcial, esta tecnología es denominada ***tecnología de bases relacionales de datos***, la cual provee de un alto grado de independencia de datos para los programas de aplicación, facilidades para la combinación de la información almacenada en forma muy flexible, y ofrece lenguajes de petición que son relativamente fáciles de aprender y utilizar, sin embargo, las facilidades que los sistemas relacionales ofrecen no son suficientes para la estructura lógica de datos y la independencia necesaria en el medio ambiente del CAD, CAM y el CIM. Especialmente, estas tecnologías requieren del manejo de estructuras jerárquicas anidadas y de una gran diversidad de tipos de n-eadas (relaciones), para la representación de toda la información necesaria. Debido a razones semánticas (conceptuales) y de rendimiento los objetos complejos, no pueden ser aplanados y almacenados como relaciones ordinarias, esto significa que el sistema de bases de datos debe proveer de facilidades para soportar directamente estructuras jerárquicas.

Algunos esfuerzos de los investigadores se han orientado a como extender un sistema relacional de bases de datos para soportar las estructuras jerárquicas de manera eficiente, lo cual se ha hecho a través de la inclusión de tipos de datos predefinidos, agregación de llaves generalizadas, a fin de acelerar las uniones y expresar las relaciones jerárquicas, proveer de una interface con los operadores necesarios para insertar, eliminar, buscar y actualizar objetos complejos. La ventaja de este enfoque, consiste en que no es necesario modificar completamente los sistemas manejadores de bases de datos para soportar la manipulación de objetos complejos, la desventaja, sin embargo, es que al utilizar esta técnica los objetos complejos son una "rareza" para el sistema que sustenta la modificación y no son tratados por dicho sistema de una manera uniforme, como lo hace con los objetos planos, esto es, la proyección, selección y unión de objetos complejos no es realizada de la misma manera que para los tipos convencionales de objetos, a fin de evitar lo anterior, es necesario crear una visión integrada de los objetos complejos y los objetos planos, de manera que el modelo común de datos cubra ambos tipos de datos de la forma más adecuada y uniforme.

Una forma de integrar relaciones planas y estructuras jerárquicas dentro de un modelo de datos sin perder la elegancia y expresividad de los lenguajes de petición relacionales, es generalizar el modelo relacional de datos, la idea clave es permitir relaciones como valores de atributos de n-eadas en relaciones, lo anterior, significa perder el requerimiento de la primera forma normal y las relaciones que surgen en este contexto se les denomina ***relaciones No_en_primera_forma_normal*** (NF^2).

El objetivo de esta discusión, consiste en explicar el concepto de las relaciones NF^2 desde el punto de vista del usuario y de los aspectos de implementación.

A.2. LAS TABLAS NF^2 DESDE EL PUNTO DE VISTA DEL USUARIO.

En la presente revisión se tiene el uso de la siguiente terminología:

El término ***tabla*** será utilizado como una generalización de relación (tabla sin orden) y ***lista*** (tabla ordenada).

Se considera un ***objeto complejo*** como un objeto de estructura jerárquica, por lo que no se distingue entre objeto jerárquico, objeto complejo y tablas NF^2, por último el término ***relación plana*** y ***tabla en (1FN)*** son sinónimos.

Si se asume que se desea modelar un objeto con estructura jerárquica, por ejemplo **DEPARTAMENTOS** con un número de departamento (**DNU**), número del jefe (**JEFNU**), proyectos (**PROYECTOS**), presupuesto (**PRESUP**) y equipo (**EQUIPO**) en el nivel superior, y con **EQUIPO** constituido por cantidad (**CN**) y tipo (**TIPO**) para cada ítem. El atributo **PROYECTOS** representa todos los proyectos del departamento con un número de proyecto (**PNU**), nombre del proyecto (**PNOMB**) y los miembros del proyecto (**MIEMBROS**), para cada miembro su número (**EMPNU**) y la función (**FUNCION**), en un manejador de bases de datos convencional es posible modelar esto a través de la definición de segmentos de tipos y relaciones del tipo conocido como ***padre-hijo*** como se muestra en la **figura 1.**

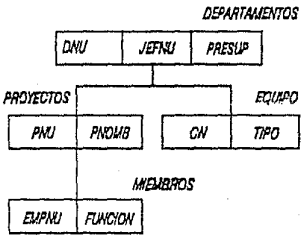

FIGURA 1. Jerarquía DEPARTAMENTOS.

Para la recuperación de la información de un objeto de este tipo, es necesario contar con constructores en un lenguaje de navegación como "***trae el siguiente***" (**GN**) y "***trae el siguiente dentro del padre***" (**GNP**), los cuales son completamente diferentes de los constructores de alto nivel que se usan de un sistema relacional de bases de datos.

Para representar la información de la figura 1 en tablas en **la primera forma normal** (**1FN**) se requiere de al menos cuatro tablas.

DEPARTAMENTOS-1NF		
DNU	JEFNU	PRESUP
314	5619	4320,000
218	71349	440,000
417	91093	360,000

TABLA 1.

PROYECTOS-1NF		
PNU	PNOMB	DNU
17	CGA	314
23	HPAR	314
25	LEXI	218
37	NDBS	417

TABLA 2.

MIEMBROS-1NF			
DNU	PNU	EMPNU	FUNCION
314	17	39582	LIDER
314	17	56019	CONSULTOR
314	17	69011	SECRETARIA
314	23	58912	ASESOR
314	23	90011	LIDER
314	23	78218	SECRETARIA
314	23	98202	ASESOR
218	25	72227	ASESOR
218	25	89211	ASESOR
218	25	92100	LIDER
218	25	89921	CONSULTOR
218	25	99025	SECRETARIA
218	25	44512	CONSULTOR
417	37	87710	SECRETARIA
417	37	81193	LIDER
417	37	75913	ASESOR
417	37	96001	ASESOR

TABLA 3

EQUIPO-1NF		
DNU	**CN**	**TIPO**
314	**2**	**3278**
314	**3**	**PC/AT**
314	**1**	**PC**
218	**2**	**3278**
218	**2**	**PC/AT**
218	**1**	**3179**
218	**1**	**PC/GA**
417	**1**	**4361**
417	**1**	**PC/XT**
417	**1**	**PC/AT**
417	**2**	**3278**
417	**1**	**3270**
417	**1**	**3179**
417	**1**	**PC/GA**

TABLA 4.

Ya se ha mencionado que el modelo NF^2 es una generalización del modelo relacional que permite relaciones que tienen relaciones como valores de atributos, utilizando el modelo NF^2, se puede modelar el objeto de datos **DEPARTAMENTOS** como se muestra en la **tabla 5**, la cual se lee como **DEPARTAMENTOS** es una tabla no ordenada que tiene cinco atributos al nivel más alto denominados **DNU, JEFNU, PROYECTOS, PRESUP** y **EQUIPO**, (en las figuras las tablas no ordenadas (relaciones) se colocan los nombres entre "()", mientras que las tablas ordenadas (listas) se colocan entre "<>"), los atributos DNU, JEFNU y PRESUP son atómicos, mientras que los otros dos con relaciones, no son atómicos. PROYECTOS es una relación NF^2 que tiene tres atributos al nivel superior PNU, PNOMB y MIEMBROS, mientras que PNU y PNOMB son atómicos, MIEMBROS no lo es, etc. Note que los valores de los atributos de PRESUP están relacionados a un DEPARTAMENTO y no a un proyecto, lo mismo ocurre para EQUIPO, el cual es una relación plana describiendo la cantidad (CN) y el tipo (TIPO) del equipo utilizado en el departamento. PROYECTOS, MIEMBROS y EQUIPO son todos tablas sin orden (relaciones). No existe relación implícita o explícita entre un proyecto o uno de sus miembros con el tipo de equipo. Se asume que los números de los empleados en la tabla de DEPARTAMENTOS son siempre únicos, no importando si el número de proyecto lo es (incidentalmente en la **tabla 5** si son únicos).

(DEPARTAMENTOS)								
DNU	JEFNU	(PROYECTOS)				PRESUP	(EQUIPO)	
		PNU	PNOMB	(MIEMBROS)			CN	TIPO
				EMPNU	FUNCION			
314	56194	17	CGA	39582	LIDER	320000	2	3278
				56019	CONSUL		3	PC/AT
				69011	SECRET		1	PC
		23	HPAR	58912	ASESOR			
				90011	LIDER			
				78218	SECRET			
				98902	ASESOR			
218	71349	25	LEXI	72227	ASESOR	440000	2	3278
				89211	ASESOR		2	PC/AT
				92100	LIDER		1	3179
				89921	CONSUL		1	PC/GA
				99025	SECRET			
				44512	CONSUL			
417	91093	37	NDBS	87710	SECRET	360000	1	43621
				81193	LIDER		1	PC/XT
				75913	ASESOR		1	PC/AT
				96001	ASESOR		1	3278
							1	3270
							1	3179
							1	PC/GA

TABLA 5. Ejemplo de una tabla NF^2.

En realidad, cada tabla puede ser definida como ordenada o no ordenada de acuerdo a las necesidades del usuario y es fácil visualizar que las tablas normales, es decir, las tablas en primera forma normal son casos especiales de las tablas NF^2.

A.3. LENGUAJES DE PETICION PARA LAS TABLAS NF^2.

En el diseño de un lenguaje de petición adecuado para tablas NF^2, debe tomarse en cuenta el mismo enfoque que para las relaciones mismas, se tiene que utilizar el concepto de un lenguaje de petición relacional ya existente y generalizarlo de manera análoga a la generalización de las relaciones. En el caso de las tablas NF^2 donde existen atributos que pueden ser tablas (relaciones o listas) se requiere de un mecanismo (En la cláusula **SELECT**) para describir la estructura del resultado de una petición. Otro mecanismo (En la cláusula **FROM**) es necesario para describir de que atributo(s) en que nivel(es) la información debe se extraída.

La falta de espacio no permite hacer una discusión detallada de los lenguajes de petición NF^2, por lo que se tratará de hacer una descripción de ellos por medio de algunos ejemplos seleccionados.

Ejemplo 1:

Este ejemplo muestra como recuperar todas las n-eadas de una tabla NF^2 sin importar la estructura del resultado, all ser la misma que la estructura de la tabla fuente, si se asume que la tabla 5 ha sido almacenada, es posible escribir simplemente::

```
SELECT x.DNU, x.JEFNU, x.PROYECTOS, x.PRESUP, x.EQUIPO
FROM   x IN DEPARTAMENTOS
```

o utilizando la notación corta: `SELECT * FROM DEPARTAMENTOS`

Ejemplo 2::

Aquí se muestra como rescatar todas la n-eadas de una tabla NF^2 y la definición explícita de la estructura resultado, se asume de nuevo que la tabla 5 es una tabla almacenada y que se desean realizar una petición que muestre la misma estructura para la tabla resultado, cuando se define explícitamente la estructura, la petición podría lucir semejante a la petición que se muestra en la figura 2. Un buen ejercicio mental para entender la compaginación de las variables de n-eadas (variables de rango) es asociarlas con un ciclo que se ejecuta sobre todas la n-eadas de las relaciones que las contienen. En la petición de la figura 2 x es término de DEPARTAMENTOS, esto es, x es una vez un término que determina el departamento número 314, otra vez es un término que determina el departamento 218, etc., para un término dado de x, existen variables n-eadas y y w asociadas con los valores de los atributos x.PROYECTOS (el cual es una tabla) y x.EQUIPO.

```
                Atributos de    Atributos de                Atributos de
                Primer nivel    Segundo nivel               Tercer nivel

SELECT      x.DNU,
            x.JEFNU,
            PROYECTOS:= (SELECT y.PNU,
                                y.PNOMB,
                                MIEMBROS:= (SELECT      z.EMPN
U,
                                                        y.FUNCION
                                                        FROM z IN
                                                        y.MIEMBROS
                                                    )
                        FROM    y IN x.PROYECTOS
                        ),
            x.PRESUP,
            EQUIPO:=    (SELECT     v.CN,
                                    v.TIPO
                        FROM        v IN x.EQUIPO
                        )
FROM        x IN DEPARTAMENTOS
```

FIGURA 2. Construcción de la tabla 5 a partir de la tabla 5.

Ejemplo 3:

Aquí se muestra como crear una tabla de estructura compleja basada en tablas planas (ANIDAR), se asume que las tablas 1,2,3 y 4 se han almacenado y se quiere obtener como resultado la tabla 5. La petición resultante es la que se muestra en la **figura 3.**

```
          Atributos de  Atributos de          Atributos de
          Primer nivel  Segundo nivel         Tercer nivel

SELECT    x.DNU,
          x.JEFNU,
          PROYECTOS:= (SELECT y.PNU,
                              y.PNOMB,
                              MIEMBROS:= (SELECT      z.EMPN
U,
                                                      z.FUNCION
                                          FROM z IN MIEMBROS-1FN
                                          WHERE z.PNU = y.PNU
                                          AND  z.DNU = y.DNU
                                          )
                      FROM y IN PROYECTOS-1FN
                      WHERE y.DNU = x.DNU
                      ),
          x.PRESUP,
          EQUIPO:=  (SELECT     v.CN,
                                v.TIPO
                      FROM      v IN EQUIPO-1FN
                      WHERE  v.DNU = x.DNU
                      )
FROM      x IN DEPARTAMENTOS-1FN
```

FIGURA 3. Construcción de la tabla 5 a partir de las tablas 1, 2, 3 y 4. (Operación Anidar).

Ejemplo 4:

Este ejemplo muestra como crear una tabla plana como resultado, al basarse de una tabla fuente compleja (DESANIDAR), también se muestra el uso de la proyección, puesto que los atributos PRESUP y EQUIPO de la tabla fuente no son usados para la construcción de la tabla resultado (tabla 7):

```
SELECT x.DNU, x.JEFNU, y.PNU, y.PNOMB, z.EMPNU, z.FUNCION
FROM  x IN DEPARTAMENTOS, y IN x.PROYECTOS, z IN x.MIEMBROS
```

TABLA RESULTADO					
DNU	JEFNU	PNU	PNAME	EMPNU	FUNCION
314	56194	17	CGA	39582	LIDER
314	56194	17	CGA	56019	CONSULTOR
314	56194	17	CGA	69011	SECRETARIA
314	56194	17	CGA	58912	ASESOR
314	56194	17	CGA	90011	LIDER
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
218	71349	25	LEXI	92100	LIDER
.	.	.	.	.	.
.	.	.	.	.	.
417	91093	37	NDBS	96001	ASESOR

TABLA 7. Tabla resultado del ejemplo 4.

Por comparación la misma petición contra las tablas 1 a 3 podría aparecer como sigue:

```
SELECT x.DNU, x.JEFNU, y.PNU, y.PNOMB, z.EMPNU, z.FUNCION
FROM x IN DEPARTAMENTOS-1FN, y IN PROYECTOS-1FN, z IN MIEMBROS-
1FN
WHERE x.DNU = y.DNU AND y.PNU = z.PNU AND y.DNU = z.DNU
```

Como se puede ver la petición sobre la tabla plana es más difícil de formular que contra la tabla jerárquica, el ejemplo también muestra que las tablas jerárquicas pueden ser utilizadas para almacenar uniones precalculadas.

Ejemplo 5:

Considerese la siguiente petición contra la tabla 5: "liste DNU, JEFNU y PRESUP de todos los departamentos que utilicen una PC/AT".

Esta petición, que muestra el uso de la cláusula **EXIST**, se puede expresar como sigue:

```
SELECT x.DNU, x.JEFNU, x.PRESUP
FROM  x IN DEPARTMENTOS
WHERE EXIST y IN x.EQUIPO: y.TIPO='PC/AT'
```

La salida consiste de una tabla plana con tres valores atómicos.

Ejemplo 6:

Para demostrar el uso de la cláusula **ALL** en una petición, se tiene el siguiente problema "Liste DNU, JEFNU y PRESUP de todos los departamentos que en la tabla 5 solo tienen consultores como empleados", lo cual puede ser expresado como sigue:

```
SELECT    x.DNU, x.JEFNU, x.PRESUP
FROM      x IN DEPARTMENTOS
WHERE     ALL y IN x.PROYECTOS:
      ALL z IN  y.MIEMBROS: FUNCION='Consultor'
```

Puesto que MIEMBROS es una subtabla de PROYECTOS y PROYECTOS en turno es una subtabla de DEPARTAMENTOS, dos cláusulas ALL son necesarias, una para los proyectos y una para los miembros. Para el contenido de la tabla DEPARTAMENTOS que se muestra en la Tabla 5, el conjunto resultado de esta petición es vacío, dado que no existe departamento que cumpla con las condiciones de la cláusula WHERE.

Ejemplo 7:

Se asume que se tiene una tabla plana EMPLEADOS-1FN (**tabla 8**) con número de empleado (EMPNU), apellido (APELL), nombre (NOMBR) y sexo (SEX) como atributos. La tabla EMPLEADOS-1FN contendrá una n-eada por cada miembro o jefe de proyecto almacenado en la tabla 5, al utilizar esta tabla y la tabla EMPLEADOS-1FN como tablas de entrada, la siguiente petición puede ser formulada: "Liste todos los empleados con número de empleado (EMPNU), apellido (APELL), nombre (NOMBR), sexo (SEX) y función (FUNCION) agrupados por departamento, a nivel departamento muestre número de departamento (DNU) y número de jefe (JEFNU)".

EMPLEADOS-1FN			
EMPNU	**APELL**	**NOMB**	**SEX**
56194	SCHIMDT	HORST	MALE
39582	MUELLER	KLAUS	MALE
56019	BAYER	ROLF	MALE
69011	MAIER	ANDREA	FEMALE
.	.	.	.
.	.	.	.
.	.	.	.
96011	PAULSEN	HELEN	FEMALE

TABLA 8. Tabla en Primera Forma Normal (1FN).

Para responder a esta petición, es necesario realizar una unión entre MIEMBROS (DEPARTAMENTOS) y EMPLEADOS-1NF. La petición resultante (**figura 4**) muestra que los atributos de la unión tienen que estar al mismo nivel en la jerarquía.

```
SELECT    x.DNU,
          x.JEFNU,
          EMPLEADOS:=          (SELECT    x.EMPNU,
                                          w.APELL,
                                          w.NOMB,
                                          w.SEX,
                                          z.FUNCION
                               FROM       t IN x.PROYECTOS,
                                          z IN t.MIEMBROS,
                                          w IN EMPLEADOS-1FN
                               WHERE      z.EMPNU = w.EMPNU
                               )
FROM      x IN DEPARTAMENTOS
```

FIGURA 4. Unión entre MIEMBROS (DEPARTAMENTOS) y EMPLEADOS-1FN.

Se puede expresar más de una condición en la unión dentro de la condición de la petición , ya que puede formularse la petición anterior de tal manera que el nombre del jefe y su sexo sean rescatados en lugar de su número (**figura 5**).

```
SELECT   x.DNU,
         y.APELL,
         y.NOMB,
         y.sex,
         EMPLEADOS:=        (SELECT   z.EMPNU,
                                      w.APELL,
                                      w.NOMB,
                                      w.SEX,
                                      z.FUNCION
                            FROM      t IN x.PROYECTOS,
                                      z IN t.MIEMBROS,
                                      w IN EMPLEADOS-1FN
                            WHERE     z.EMPNU = w.EMPNU
                            )
FROM     x IN DEPARTAMENTOS,
         y IN EMPLEADOS-1FN
WHERE    x.JEFNU = y.EMPNU
```

FIGURA 5. Petición con dos uniones.

(REPORTES)				
REPNU	<AUTORES>	TITULO	(DESCRIPCION)	
	NOMBRE		LLAVE	PESO
0179	JONES A.	CONCURRENCY AND	CONTROL	0.6
			RECOVERY	0.3
			DISTRIBUT	0.1
0189	MILLER H	TEXT EDITING	SEARCH	0.7
	ABRAHAM C		FORMAT	0.3
	MEYER F.			
0292	POE A.	BRANCH AND BOUND	OPTIMA	0.6
	TACKER J.		GARBAGE	0.2
			COLLECTI	0.2

TABLA 6. Tabla NF^2 REPORTES no ordenada.

Ejemplo 8:

Considerese la **tabla 6**. Se asume que se desea ver la lista de autores, y los títulos de los reportes en donde "Jones" aparece como el primer autor. Como el atributo AUTORES es una tabla ordenada, decir una lista, la petición puede expresarse como sigue:

```
SELECT x.AUTORES, x.TITULO
FROM  x IN REPORTES
WHERE x.AUTORES[1] = "Jones".
```

Note que la tabla resultado no es plana dado que AUTORES no es un atributo atómico.

Junto con el lenguaje de peticiones, se requiere de facilidades para insertar, modificar y eliminar objetos complejos, lo mismo es necesario un lenguaje que permita la definición de la estructura de las tablas.

El lenguaje de petición que se ha presentado es solo una forma de tratar con el modelo de relaciones extendidas NF^2, no es la intención asegurar que es el mejor lenguaje para este tipo de modelos, especialmente para los usuarios finales se desearía tener un lenguaje más amistoso o fácil, quizá del estilo de petición por ejemplo (**QBE**) [**ZLOO 77**], sin embargo, el tipo de lenguaje presentado, permite la utilización del poder completo del modelo de datos extendido NF^2, especificando por separado la estructura de la tabla resultado de manera independiente de la estructura de la tabla fuente.

Especialmente para el manejo de grandes objetos complejos en el área CAD/CAM/CIM, las interfaces de programación de aplicaciones (**API**) son altamente necesarias, la tendencia en la definición de los lenguajes mencionados es similar al enfoque propuesto por Lorie [**LORIE 84**], que involucra los lenguajes de definición de datos (**DDL**) y los lenguajes de manipulación (**DML**) para el manejo de modelos de datos NF^2 en un lenguaje de programación de alto nivel.

A.4. ESTRUCTURAS DE ALMACENAMIENTO PARA OBJETOS NF^2.

Existe, por supuesto, una gran diversidad de formas de almacenar objetos con estructuras jerárquicas, algunas de ellas muy conocidos desde hace más de una década y utilizadas en sistemas de bases de datos existentes como:

. Estructuras de almacenamiento **IMS [DATE 81]**, el cual provee cuatro formas diferentes de almacenamiento para objetos de bases de datos jerárquicos, conocidos como **HSAM, HSAM, HDAM e HDAM.**

. Estructuras de almacenamiento **CODASYL/DBTG,** dado que cualquier objeto jerárquico puede ser visto como una composición (posiblemente muchas) de relaciones uno a varios, las técnicas de implementación para **COSET's [SHU 84]** puedes utilizarse para objetos NF^2, entonces, las listas, cadenas y arreglos de apuntadores junto con opciones adicionales son candidatos para la implementación de objetos en forma NF^2.

De acuerdo con la propuesta de **Lorie [LORIE 83]**; "Un objeto complejo es implementado como una serie de n-eadas ligadas juntas de manera lógica", las n-eadas son almacenadas como parte de tablas planas con atributos adicionales invisibles para el usuario. Esos atributos (completamente manejados por el sistema) contienen apuntadores, que son usados para encadenar n-eadas dentro del objeto complejo. La ventaja principal de este enfoque consiste en la facilidad de implementar la extensión sobre un manejador de bases de datos existente, sin tener que realizar grandes cambios.

En el manejo de proyectos más recientes, se ha tenido la oportunidad de construir un sistema completamente nuevo, entonces, mucho más énfasis se ha establecido en la integración del manejador de objetos complejos, ganándose claramente más rendimiento en la implementación con respecto a la solución de la extensión sobre un sistema ya existente. Los siguientes objetivos fueron la guía para las implementaciones más avanzadas:

. El **Agrupamiento de datos** debe ser soportado en el nivel de los objetos complejos, dado que un objeto complejo es manejado como un todo, es muy importante que la información sea almacenada en un conjunto de páginas relativamente pequeño y no este distribuido en demasiadas páginas o demasiados segmentos de la base de datos.

. La **información debe ser almacenada por separado de la estructura**, si esta regla es estrictamente observada, la navegación en un objeto complejo (por ejemplo, rescatar un cierto elemento de una lista) puede realizarse por medio de la información de la estructura, sin tener que acceder la totalidad de la información.

. **Procesamiento rápido** (inserción, borrado, rescate y actualización) deben no solo ser soportados para objetos complejos como un todo, sino también para partes arbitrarias de esos objetos. No debe ser necesario por ejemplo, analizar un objeto complejo entero, si solo parte de la información de él es de interés para el usuario.

Antes de entrar en más detalle de los mecanismos de implementación, es necesario determinar cierta terminología utilizada en esta discusión.

En el modelo de datos NF^2 es necesario distinguir entre:

. Tablas NF^2 y tablas 1FN.

. Tablas sin orden : relaciones, tablas ordenadas : listas.

. Objetos complejos y objetos planos.

. Subtablas NF^2 y subtablas 1FN.

. Subobjetos planos y subobjetos complejos.

Si se observa la **tabla 5** una vez más, los ítems listados arriba se pueden explicar como sigue:

. DEPARTAMENTOS como un todo es una tabla NF^2 (o una relación puesto que no hay orden entre la n-eadas).

. Los departamentos 314, 218 y 417 son objetos complejos (proyectos, miembros y equipo incluidos).

. Hay dos subtablas (subrelaciones) en cada uno de esos objetos complejos, PROYECTOS el cual es de estructura jerárquica (N1NF) y EQUIPO el cual es plano (1FN).

. La subtabla PROYECTOS (subrelación) en el departamento 314 contiene dos subobjetos complejos, proyectos 17 y 23. La subtabla EQUIPO en el departamento 314 contiene tres subobjetos planos, ítems 3287, PC/AT, y PC.

. La subtabla MIEMBROS (subrelación) en el proyecto 17 contiene tres subobjetos planos, empleados 39582, 56019 y 69011.

Como se ha mencionado, una premisa esencial para la implementación de objetos complejos ha sido la separación de la información estructural de los datos. Por lo tanto, se ha decidido por implementar un ***minidirectorio*** (***MD***) para cada objeto complejo, el cual es un árbol que contiene toda su información estructural pero nada de sus datos. La estructura del MD corresponde a la estructura jerárquica del objeto complejo, el MD está compuesto por subn-eadas MD (nodos en el árbol MD) que son ligadas por medio de apuntadores, una subn-eada MD es una unidad básica de almacenamiento como una n-eada o registro en los tradicionales sistemas de bases de datos, junto con las subn-eadas MD, se tiene la necesidad de subn-eadas de datos para almacenar la información del objeto complejo. En realidad, todo el primer nivel de atributos valuados como atómicos de un objeto/subobjeto complejo y todos los atributos valuados como atómicos de un objeto/subobjeto plano, son almacenados en una subn-eada de datos. Obviamente si un objeto/subobjeto es plano será guardado en una sola subn-eada de datos, entonces, una tabla plana no tienen minidirectorios para sus objetos. Es importante resaltar que una subn-eada de datos no contiene ninguna información estructural del objeto.

Existen varias alternativas para implementar los minidirectorios para objetos complejos, comunmente utilizan un MD especial llamado ***subn-eada MD raíz***, como raíz del árbol de MD's. Junto con los apuntadores se contiene información adicional acerca del objeto complejo como un todo. Para los otros niveles del árbol de MD's, se debe decidir dentro de las siguientes alternativas:

. Usar subn-eadas MD para las subtablas y para los subobjetos complejos. (**SS1, figura 6a.**)

. Utilizar subn-eadas MD solo para los subobjetos complejos.(**SS2, figura 6b.**)

. Usar subn-eadas MD solo para las subtablas. (**SS3, figura 6c.**)

Notese, que no se mencionan los subobjetos planos en la lista, desde luego, se necesita información estructural acerca de ellos, por ejemplo su longitud, pero esta información es fácil almacenarla en las respectivas subn-eadas de datos.

En las figura 6a, 6b y 6c las subn-eadas MD se esquematizaron como rectángulos, mientras que la subn-eadas de datos son ovalos, para los ejemplos se utiliza el departamento 314 de la tabla 5.

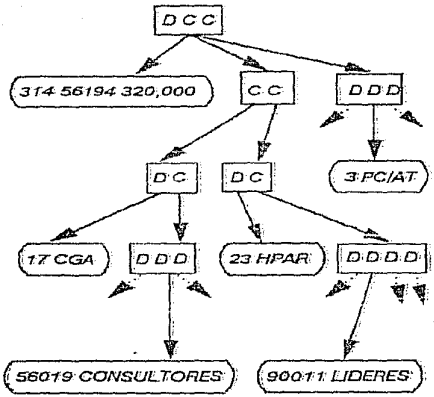

FIGURA 6a.

En la figura 6a. se muestra las alternativas de almacenamiento SSX, junto con la subn-eada raíz MD, existe una subn-eada MD por cada subtabla y una subn-eada MD por cada objeto complejo. Los valores D y C en las subn-eadas MD se sustentan por apuntadores, con D como un apuntador de datos de la subn-eada MD a la subn-eada de datos y C como un apuntador hijo de subn-eada MD a subn-eada MD.

Al nivel de la raíz de este árbol MD, hay un apuntador D refiriéndose a la subn-eada de datos que contiene todos los valores atómicos existentes al primer nivel del departamento 314 (DNU=314, JEFNU=56194 y PRESUP=320,000), y existen dos apuntadores C refiriéndose a las subn-eadas MD de las subtablas PROYECTOS y EQUIPO, respectivamente, DCC es la única entrada en la subn-eada raíz MD. Como consecuencia, la subn-eada raíz MD es de tamaño fijo y mientras no haya cambios estructurales en las tabla NF^2, por ejemplo, la adición de un nuevo atributo no atómico. Las subn-eadas MD para PROYECTOS y EQUIPO al nivel inferior del nivel de la raíz son de tamaño variable, contienen un apuntador por cada proyecto y equipo, respectivamente, el número de proyectos en un departamento lo mismo que la cantidad de equipos puede cambiar. En el siguiente nivel del árbol MD, las subn-eadas MD son de tamaño fijo nuevamente. Finalmente, al nivel más bajo del árbol MD de la figura 6, las subn-eadas MD de longitud variable representan las instancias de la subtabla MIEMBROS la cual está en la primera forma normal.

Las figuras 6b. y 6c. se pueden interpretar como sigue:

. En la figura 6b todas las subn-eadas MD que pertenecen a las subtablas han sido movidas e integradas en las subn-eadas MD de subobjeto complejo. En el nivel más alto del árbol MD, esta integración tiene que ser hecha dentro de la subn-eada raíz del árbol MD. Como consecuencia, SS2 contiene junto con la subn-eada raíz una subn-eada de MD de longitud variable por cada **subobjeto complejo.**

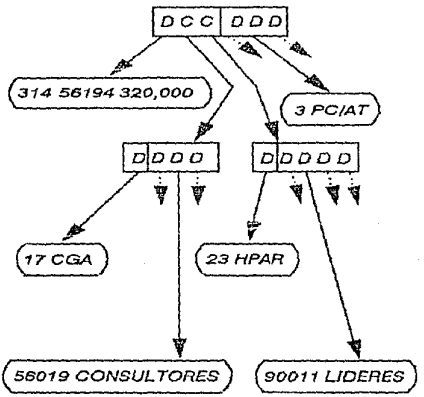

FIGURA 6b.

. En la figura 6c. todas las subn-eadas de MD que pertenecen a subobjetos complejos, tienen que ser movidas e integradas en las subn-eadas MD de las subtablas. Como consecuencia SS3 contiene junto con las subn-eada MD raíz una subn-eada de longitud variable por cada **subtabla.**

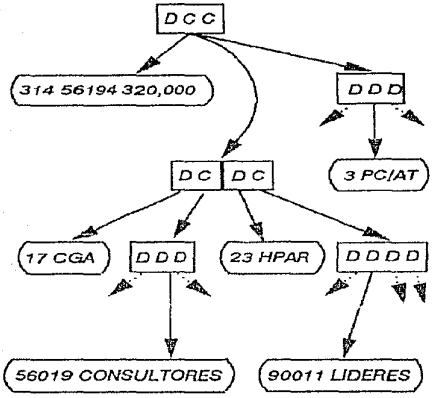

FIGURA 6c.

Se debe notar que los objetivos relativos a la separación de información estructural y de datos y al soporte de procesamiento rápido de partes arbitrarias (subtablas, subobjetos, etc.) de un objeto complejo, han sido alcanzados, sin embargo, el aspecto de agrupamiento de información no ha sido descrito aún.

Cada objeto complejo tiene su propia ***dirección local de espacio***, representada por un lista de páginas almacenada en la subn-eada MD raíz, esta lista de páginas contiene el número de todas la páginas de donde (datos y MD) las subn-eadas de un objeto complejo se encuentran actualmente almacenadas. No importa si nuevas subn-eadas en un objeto complejo tienen que ser almacenadas (o subn-eadas existentes tienen que ser extendidas), la lista de páginas es examinada para encontrar una página con el suficiente espacio para la nueva información, solo si esta operación falla (no existe suficiente espacio disponible en la dirección local de espacio), una página externa del objeto complejo se utiliza para almacenar información, entonces, el número de la página recientemente seleccionada es agregada a la lista. Esta estrategia soporta directamente el agrupamiento de datos en un objeto complejo, puesto que la nueva información es usualmente almacenada en páginas las cuales ya contienen información del objeto complejo.

A.5. VIAS DE ACCESO PARA OBJETOS NF^2.

Obviamente, la optimización de peticiones y la selección de vías de acceso para tablas NF^2, son considerablemente más complejos que para tablas 1FN, debido sobre todo a causas como las siguientes:

. Tienen que ser procesados grandes objetos jerárquicos en lugar de pequeños objetos planos.

. Operaciones no triviales tienen que realizarse para transformar objetos planos en objetos jerárquicos y viceversa, para evaluar cláusulas del tipo EXIST y ALL en las instrucciones de las peticiones.

. Realizar uniones entre diferentes niveles en diferentes tablas NF^2.

En resumen, la complejidad incrementada viene del hecho de que el modelo de datos NF^2 y su lenguaje de peticiones son más poderosos que los modelos relacionales tradicionales y sus lenguajes de petición.

A.6. CONCLUSIONES.

El modelo de datos NF^2 extendido ha sido mostrado en esta discusión como un mecanismo para integrar tablas planas y tablas jerárquicas (conteniendo objetos complejos) de manera natural, la forma de hacer esto es a través de la consideración de las tablas jerárquicas como parte integral del modelo de datos y no como "casos especiales" del mismo, como consecuencia, todas las operaciones sobre tablas planas son aplicables a las tablas jerárquicas también. Se ha mostrado que un lenguaje poderoso de peticiones similar al SQL, es una herramienta para la recuperación eficiente de la información almacenada en la base de datos bajo el enfoque de las relaciones NF^2.

El modelo de datos NF^2 puede ser utilizado como una vía de migración para las bases relacionales y jerárquicas de datos, de manera que puedan ser unidas en un modelo común de datos.

Actualmente se ha completado la primera versión de un sistema manejador de bases de datos, el cual soporta el modelo de datos NF^2 extendido, esto es, puede insertar, recuperar, actualizar y eliminar n-adas como un todo o como partes de ellas, sin embargo, es necesario mencionar que se está trabajando en hacer un prototipo más completo por medio de agregar funciones que no se han considerado hasta ahora, a fin de cubrir conceptos de tipos de datos abstractos en las nuevas áreas de aplicación como puede ser la robótica, se intenta también incluir tópicos de sistemas manejadores de bases extendibles de datos.

APENDICE B. *NST, SECUENCIAS ANIDADAS DE N-EADAS DE DATOS.*

El modelo relacional se ha encontrado inadecuado para diversos tipos de aplicaciones que van desde el CAD/CAM a los sistemas de formas estructuradas de oficina.

Varios modelos semánticos han sido propuestos en la literatura de bases de datos, a fin de capturar más significado en los modelos de datos.

El **modelo binario** [ABRI 74], el modelo **entidad-relación** [CHEN 76], el **modelo relacional extendido** [CODD 79], [LORIE 82], DAPLEX [SHIP 81], GEM [ZANI 83], [ZANI 84], IFO-fragments [AH 87] y el **modelo lógico** [KUPE 86], son intentos para definir un nuevo modelo de datos más cercano al mundo real.

El enfoque al que pertenece el modelo de datos que aquí se revisa, corresponde a minimizar las extensiones al modelo relacional, a fin de retener muchas de sus ventajas. Dentro del contexto anterior, el relajar la ***primera forma normal*** ha sido una tónica propuesta por varios autores [MAKI 77], [SP 82], [ABIT 84].

El relajar la primera forma normal, corresponde a manipular relaciones no normalizadas que permiten a los componentes de n-eadas ser instancias de relaciones o valores atómicos.

Para las relaciones no normalizadas se han propuesto nuevos tipos de operaciones, entre ellas, las operaciones ***anidar*** y ***desanidar***, las cuales han sido estudiadas por diversos autores [SP 82], [FISHER 83], [KTT 83].

Esencialmente, las operaciones anidar y desanidar, permiten pasar de relaciones planas a relaciones jerárquicas y viceversa.

Un gran esfuerzo de implementación ha sido la construcción de un prototipo de sistema de bases de datos basado en el modelo relacional no normalizado denominado NF^2 [DADA 86], [SP 82], el cual ha sido extendido recientemente para capturar ciertas características no cubiertas en el modelo original [PIST 85a], [PIST 85b] (referirse al apendice A).

Hasta ahora se ha presentado una formalización del concepto de orden en relaciones planas, sin embargo, no se conoce de ninguna definición formal de un álgebra que trate con relaciones jerárquicas anidadas.

En esta sección se presenta un modelo de datos basado en secuencias de ***n-eadas*** denominado ***NST***, el cual permite la representación de objetos jerárquicos, al mismo tiempo se trata la existencia de un álgebra para la manipulación y reestructuración de dichos objetos.

La panorámica que se presenta, es un modelo de datos denominado ***modelo de secuencias anidadas de n-eadas de datos*** (***NST*** por sus siglas en ingles), que pertenece a la categoría de los modelos relacionales no normalizados, también denominados modelos de relaciones anidadas, sin embargo, contiene ciertas diferencias que lo caracterizan:

1) **El modelo NST está basado en secuencias anidadas de entidades n-eadas** en lugar de conjuntos anidados, por lo tanto el orden de las n-eadas está representado en el álgebra.

2) **Se permiten los duplicados** y son tratados por medio de operadores explícitos y no son removidos automáticamente.

2) **El álgebra NST es *multitipificada*,** en lugar de ser *monotipificada*, lo que implica que las operaciones aritméticas y las funciones agregadas pueden ser manejadas de manera uniforme por el álgebra, por ejemplo, **un parámetro de una condición en la operación de selección puede ser cualquier expresión del álgebra.**

3) Provee de un nuevo y poderoso operador, denominado **LAMBDA,** para la reestructuración de objetos jerárquicos que permite aplicar cualquier expresión algebraica a cualquier nivel de uno o más objetos jerárquicos.

4) Define las semánticas de los operadores del álgebra en presencia de **atributos valuados como nulos.**

Un subconjunto del álgebra es utilizado actualmente como la base formal de implementación de:

i) Un sistema manejador experimental de documentos en el Centro de Investigación de **IBM Aldamen [CHOY 87].**

ii) Un manejador para la manipulación de objetos geométricos en la **Universidad de Dortmund [GUET 88].**

B.1. EL MODELO DE DATOS NST.

El modelo de datos NST está basado en las nociones de n-eadas y secuencias.

Sea $t=(t_1,...,t_r)$ que denota una ***n-eada***, con $t_1,...,t_r$ como componentes.

Sea $s=<s_1,...,s_r>$ que denota una ***secuencia de n-eadas***, $<>$ denota la secuencia vacía.

El álgebra NST es un *álgebra multitipificada* y agrupa los objetos de datos en clases. La asociación objeto-clase está dada por un esquema que describe la estructura y los valores de todos los objetos en una clase, existen dos tipos de esquemas, los ***atómicos*** y los ***compuestos***.

Considerese una colección de tipos de datos elementales como los enteros, reales, booleanos, etc. Cada tipo de datos está descrito por un identificador. El conjunto de valores posibles de un tipo elemental está dado por la función **DOM.**

Sea el conjunto **ATOMICO** conteniendo los identificadores de los tipos elementales.

Definición: Todo elemento del conjunto **ATOMICO** es un ***esquema atómico.***

El conjunto de valores posibles de objetos descritos por un esquema atómico es:

INSTANCIAS(s)=DOM(s)

Definición: Dados los esquemas $s_1,...,s_r$ la n-eada $(s_1,...,s_r)$ es un *esquema compuesto*.

El conjunto de valores posibles de objetos descritos por un esquema compuesto $S=(s_1,...,s_r)$ es:

INSTANCIAS(S)={$<a_1,...,a_r>$/ n>=0 y para 1<=i<=r: ai=$(a_{i,1},...,a_{i,r})$ y para cada 1<=j<=r: $a_{i,j} \in$ INSTANCIAS(Sj) }.

Cualquier valor asociado con un esquema compuesto, es una secuencia de n-eadas.

Cualquier componente de una n-eada, puede ser una secuencia de n-eadas o un valor atómico.

La definición de un objeto de datos del modelo, viene de una de las definiciones dadas arriba.

Definición: Un objeto de datos es una pareja (s,o) donde s es un *esquema* y o es una *ocurrencia* de S, una ocurrencia de S es un elemento del conjunto de instancias.

Una representación gráfica de un esquema compuesto, se puede ver en la **figura 1**, donde el esquema describe la estructura de una colección de "**Productos**"; para cada producto se han dado el código, la descripción y los elementos que constituyen el producto.

Los elementos componentes del producto son objetos compuestos denominados *colección de elementos*, cada elemento tienen un código y un componente de cantidad.

Dado que las colecciones son ordenadas, un nodo interno en el esquema de árbol siempre representa una secuencia de elementos (n-eadas).

Aunque en la figura 1 se hayan usado etiquetas para denotar atributos, el modelo **NST** no contiene etiquetas y un esquema forma correspondiente a "productos" es como sigue:

(CADENA,CADENA,(CADENA,CADENA))

En lo subsecuente, algunas veces se utilizan etiquetas para denotar atributos, esto es únicamente para mejor comprensión y claridad de los ejemplos. Al nivel de usuario, el uso de etiquetas es útil, ya que conlleva al usuario algo de "significado" de los objetos de datos involucrados. Sin embargo, las etiquetas utilizadas, son irrelevantes en el nivel formal, en donde se utilizan las etiquetas posicionales (enteros) para referirse a algún componente de un objeto de datos.

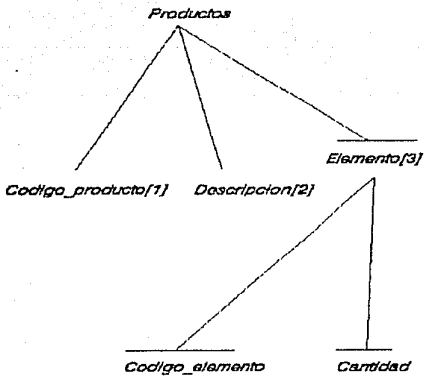

FIGURA 1.

Una ocurrencia del esquema "Productos" puede ser:

< (P1,banca,<(P2,4),(P3,8)>),
(P2,silla,<(P5,4)>),
(P3,tabla,<(P4,1)>)>

B.2. OPERADORES DEL MODELO NST.

Algunos operadores del modelo **NST** son extensiones del álgebra relacional, mientras que otros son nuevos. Cuando se definió el álgebra para el modelo, no se orientó a encontrar un conjunto mínimo de operadores, por lo tanto, algunos de los operadores del álgebra no son primitivos pero pueden ser derivados de otros.

En primer lugar, se presenta una descripción informal de las operaciones, determinando especial interés en los que son nuevos.

Las operaciones del álgebra, son mapeos que toman uno o más objetos de datos como operandos y crean un nuevo objeto de datos como resultado. Algunos operadores toman parámetros que modifican el comportamiento del operador. Otros operadores pueden tomar como parámetros expresiones arbitrarias de álgebra, este tipo de operadores es llamado *operadores no simples*, mientras que al resto se le denomina *operadores simples:*

El álgebra del **NST** es multitipificada y por lo tanto no todos los operadores son aplicables a todos los objetos.

CLASE	ARGUMENTOS	NOTACION
1	NUM X NUM : NUM	+, -, *, /
2.	BOOL X BOOL : BOOL	AND, OR
3	BOOL : BOOL	NOT
4	COMP X COMP : COMP	unión, intersección, diferencia (-), producto cartesiano (X), BD, CONCAT, PAIR
5	COMP : COMP	proyección (*P*), remover (*R*), ordena (*OR*), remueve duplicados (*RDUP*), agrupa (*A*), distribución(*D*), empaca (*E*) desempaca (*U*), selección (*S*) LAMBDA, HEAD, TAIL, PORTION, REVERSE
6	NUM X NUM : BOOL TEXT X TEXT : BOOL BOOL X BOOL : BOOL	<, <>, C, [
7	COMP X COMP : BOOL	=, <>, C, [
8	BOOL X NUM X NUM : NUM BOOL X TEXT X TEXT : TEXT BOOL X BOOL X BOOL : BOOL BOOL X COMP X COMP : COMP	IF THEN ELSE FI
9	COMP : NUM	COUNT, AVG, SUM, MIN, MAX
10	COMP : BOOL	ISEMPTY, EXIST, FORALL

TABLA I. Tablas de operadores NST.

La **tabla I** resume los operadores agrupados por su funcionamiento. En la tabla, **NUM, TEXT, BOOL** y **COMP**, son las especies o clases del álgebra y los portadores asociados son los conjuntos respectivos de objetos de datos atómicos para **NUM, TEXT**, y **BOOL** y el conjunto de todos los objetos de datos compuestos para **COMP**.

Para los operadores que toman parámetros, estos aparecen después del símbolo operador, cuando se aplican operadores a objetos se utiliza una notación posttija.

B.3. OPERADORES SIMPLES.

Todos los operadores simples del álgebra, permiten referirse al primer nivel de componente de un objeto (hijos del nodo raíz), para referirse a niveles más profundos de un objeto de datos jerárquico tiene que utilizarse el operador **LAMBDA**.

Las **clases 1,2,3 y 6** corresponden a los operadores matemáticos estándar para objetos atómicos cuyas semánticas son bien conocidas.

La **clase 7** contienen la igualdad (=), subconjunto (⊆) y subsecuencia ([) operadores prueba para objetos compuestos.

En particular, el operador = se define para objetos compuestos como sigue:

$s_1=(x_1,...,x_r)$ y $o_1=<a_1,...,a_n>$

$s_2=(y_1,...,y_s)$ y $o_2=<b_1,...,b_m>$

se tiene que

$(s_1,o_1)=(s_2,o_2)$ <====> $s_1=s_2$ y $o_1=o_2$

donde

$s_1=s_2$ <====> r=s y para $1<=i<=r$ $x_i=y_i$

$o_1=o_2$ <====> n=m y para $1<=i<=n$ $a_i=b_i$

La **clase 4** contiene la teoría de conjuntos unión, intersección, diferencia de conjuntos, producto cartesiano, concatenación y operaciones de paridad. Los primeros cinco operadores usan secuencias para simular conjuntos y son extensiones de los operadores clásicos relacionales. En particular, la unión y la intersección de dos secuencias de n-eadas puede ser formado si los dos esquemas de los operandos son iguales. El producto cartesiano puede ser formado para cualquiera dos secuencias y el esquema resultado es obtenido por la concatenación de los dos esquemas de las n-eadas operadas. Las últimas dos, son operaciones puras de secuencias. Dos secuencias pueden ser concatenadas si sus respectivos esquemas son iguales. El operador de paridad es nuevo y acepta dos secuencias como operandos, el esquema resultante es el mismo que para el producto cartesiano, la ocurrencia resultado está formada por la combinación de la primera n-eada a partir de la primera secuencia operada con la primera n-eada a partir de la segunda secuencia operada, después las dos segundas n-eadas, etc.

La **clase 5** contiene varios operadores dentro de los que destacan; Proyección *P*, que es una extensión del muy conocido operador del álgebra relacional. El orden de los componentes especificados de la proyección puede cambiar con respecto al esquema del operando, por lo que la proyección puede ser utilizada para reordenar componentes.

Remover *R*, es similar a la proyección pero permite especificar los componentes de una n-eada a ser removidos, el orden de los componentes no se altera, por lo que sigue siendo el mismo del esquema del operando.

Ordenación *ORD*, permite clasificar una secuencia basado en los valores de uno o más componentes de la n-eada. En particular el operador de ordenación, permite clasificar una secuencia de n-eadas por el valor de cualquier componente atómico de forma ascendente o descendente, o por los valores de algunos componentes atómicos.

Remueve duplicados, *RDUP*, es una operador explícito para remover duplicados, en el álgebra los duplicados no son removidos de manera estándar.

Agrupar *A*, distribución *D*, Empacar *E*, y Desempacar *U*, son operadores para agrupar n-eadas con valores de componentes idénticos y realizan también la operación inversa. Estos operadores juntos son casi equivalentes a los operadores anidar y desanidar del **modelo relacional no normalizado en la primera forma normal [THF 86]**. Existen algunas diferencias, en particular la operación distribuir baja los componentes de su nivel más alto a componentes compuestos, la función distribución es también, más flexible que el operador desanidar, ya que permite distribuir ciertos componentes seleccionados en un subárbol, mientras que la función desanidar separa el árbol completo forzando la duplicación de todos los atributos más altos de cada n-eada en el subárbol.

HEAD, **TAIL**, y **PORTION** son secuencias de operadores para seleccionar un número específico de elementos del frente, final y dentro de una secuencia. **REVERSE** invierte el orden de los elementos de una secuencia.

La **clase 9** contiene las bien conocidas funciones agregadas **COUNT, AVG, SUM, MIN** y **MAX**, estas son formalizadas en la estructura del álgebra **NST**.

La función **ISEMPTY** de la **clase 10**, simplifica la prueba de un objeto compuesto, que permite conocer si este es una secuencia vacía o no.

A continuación se hará la definición formal de algunos de los operadores mencionados, a fin de tener más claridad se describen los mapeos de los esquemas y de las ocurrencias por separado.

Diferencia de conjuntos (-).

Restricciones: s = t.

a) s t - = s

b) $<a_1,...,a_n>$ $<b_1,...b_m>$ - =

$<>$ si n=0

$<oa'_1,...,oa'_n>$ en otro caso

donde a'_i =

$<a_i>$ si a_i no pertenece a $\{b_1,...,b_m\}$

$<>$ en otro caso

Producto Cartesiano (X)

a) s t X = s#t ('#' es el operador concatenación de n-eadas)

b) $<a_1,...,a_n> <b_1,...b_m>$ X =

$<>$ si n=0 o m=0

$<a_1\#b_1,a_1\#b_2,...,a_1\#b_m,a_2\#b_1,...,a_n\#b_m>$ en otro caso.

Proyección: *P* $[i_1,...,i_p]$

Parámetros: p>=1, $i_j \in [1,...,r]$, j<>k => $i_j <> i_k$

Dado el esquema de entrada $(s_1,...,s_r)$ se tiene que

a) $(s_1,...,s_r)$ *P* $[i_1,...,i_p] = (s_{i1},...,s_{ip})$.

Dada la ocurrencia de entrada $<a_1,...,a_n>$ se tiene que

b) $<a_1,...,a_n>$ *P* $[i_1,...,i_p]$=

$<(a_{1,i1},...a_{1,ip}),...,(a_{n,i1},...,a_{n,ip})>$

La proyección selecciona de un esquema de todas las n-eadas de la ocurrencia los componentes especificados por los números $i_1,...i_p$. Cada componente (número) puede ocurrir una sola vez, sin embargo, el orden de los componentes especificados puede cambiar con respecto al esquema del operando.

Ejemplo: Considerese el objeto de datos "Productos" con el esquema mostrado en la figura 1 y siguiente ocurrencia.

<(P1,banca,<(P2,4),(P3,8)>),
(P2,silla,<(P5,4)>),
(P3,tabla,<(P4,1)>)>

La evaluación de la expresión:

Productos *P* [3,1]

resulta en el siguiente esquema:

((cadena,cadena),cadena)

y la siguiente ocurrencia

<(<(P2,4),(P3,8)>,P1),
(<(P5,4)>,P2),
(<(P4,1)>,P3>

Remueve $R\ [i_1,...,i_p]$

Parámetros: $p>= 1$, $[i_1,...i_p]$ no contenido propiamente en $\{1,..,r\}$, $j<>k ==> j <>i_k$.

$R\ [i_1,...i_p] = P\ [j_1,...j_q]$ en donde

$[j_1,...j_q] = \{1,...,r\} - \{i_1,...,i_p\}$ y para $1<=t<q: j_t<j_{t+1}$

La última condición asegura que el orden de los componentes conserva el orden del esquema operando.

Distribución $D\ [z,i_1,...,i_p]$

El operador distribución permite "bajar" los componentes del nivel más alto a un componente compuesto (subárbol).

Los parámetros son z, el índice de un subárbol e $i_1,...,i_p$, el número de componentes a ser "bajados".

Parámetros: $p>=1, \{z,i_1,...,i_p\}, \{1,...,r\}$, s_z es un componente compuesto $s_{i1},...,s_{ip}$ son esquemas atómicos.

a) $(s_1,...,s_z,...,s_r)\ D\ [z,i_1,...,i_p] =$

$(s_1,...,s_z',...,s_r)\ R\ [i_1,...,i_p]$

donde $s_z' = (s_{i1},...,s_{ip})$ o s_z ("o" indica concatenación)

Los componentes $i_1,...,i_p$ son removidos del nivel más alto del esquema y agregados al subesquema s_z.

b) $<a_1,...,a_n>\ D\ [z,i_1,...,i_p] = <a_j',...,a_n'>\ R\ [i_1,...,i_p]$.

donde $a_j=(a_{j,1},...,a_{j,z},...,a_{j,r})$,

$a_j'=(a_{j,1},...,a_{j,z}',...,a_{i,r})$

y

$a_{j,z}'$ es el producto cartesiano de $<(a_{j,i1},...,a_{j,ip})>$ con $a_{j,z}$.

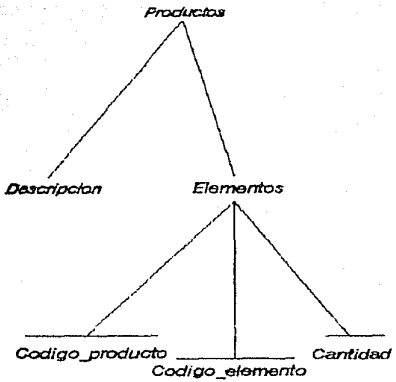

FIGURA 2.

Ejemplo: Considere el objeto de datos "Productos", la expresión del álgebra:

Productos D **[3,1]**

Resulta en el esquema de la figura 2 y en la ocurrencia:

< (banca,<(P1,P2,4),(P1,P3,8)>),
(silla,<(P2,P5,4)>),
(tabla,<(P3,P4,1)>) >

Agrupamiento: A $[i_1,...,i_p]$

Parámetros: $p>=1$, $\{i_1,...,i_p\}$ no contenido propiamente en $\{1,...r\}$, s_{ij} es un esquema atómico.

a) $(s_1,...,s_r)$ A $(s_{i,1},....,s_{i,p},(s_1,...,s_r)$ P $[i_1,...i_p])$

El propósito del operador de agrupamiento, es particionar un colección de n-eadas en subconjuntos donde todas las n-eadas tengan el mismo valor para uno o más componentes especificados $(i_1,...,i_p)$, y entonces construir una n-eada jerárquica para cada uno de los subconjuntos.

b) Para definir la ocurrencia resultado, primero debe considerarse la relación de equivalencia "=[i_1,...i_p]" sobre la n-eadas, se define como sigue:

$$r=[i_1,...,i_p]\ t <==> ((r_{;1}=t_{;1}) \text{ y } ... \text{ y } (r_{;i_p}=t_{;i_p})$$

La relación de equivalencia agrupa las n-eadas que tengan valores iguales en los componentes i_1,...,i_p.

Considerese la función auxiliar *f* la cual transforma una secuencia de n-eadas en una n-eada jerárquica:

Para s=<s_1,...,s_z>

$f(s,i_1,...,i_p)=(s_{1;i1},...,s_{1;ip}s\ P\ [i_1,...,i_p])$

Sea <G_1,...,G_m> =0

<a_1,...,a_n> **ORD** [i_1,...,i_p] **MODULO** [=[i_1,...i_p]]

El operador modulo particiona una secuencia en subsecuencias maximales de n-eadas que son equivalentes con respecto a la relación de equivalencia especificada.

Se tiene que:

<a_1,...,a_n> *A* [i_1,...,i_p] =

< f(G_1,i_1,...,i_p),...,f(G_m,i_1,...,i_p)>

Ejemplo: Supóngase un objeto de datos plano dado en el siguiente esquema:

(Código,Descripción,Código_de_elemento, Cantidad)

y la ocurrencia:

< (P1,Banca,P2,4),
(P1,Banca,P3,8),
(P2,silla,P5,4),
(P3,tabla,P4,1) >

El nombre de este objeto de datos es "lista_de_productos", supóngase que se usa el operador de agrupamiento como sigue:

lista_de_productos *A* [1,2]

El procedimiento formal de la reestructuración de la ocurrencia trabaja como sigue:

1) La secuencia es clasificada por los "componentes del agrupamiento" (utilizando *ORD*[1,2]).

2) Todas las n-eadas con los mismos valores para esos componentes son adyacentes, por lo que la secuencia es dividida en subsecuencias maximales que tengan el mismo valor dentro de los "componentes del agrupamiento" (utilizando la primitiva **MODULO** sobre secuencias).

El resultado es:

$$<G_1,G_2,G_3> \text{ con}$$

$$G_1 = <(P1,banca,P2,4),(P1,banca,P3,8)>$$

$$G_2 = <(P2,silla,P5,4)>$$

$$G_3 = <P3,tabla,P4,1)>$$

Cada grupo se transforma en una n-eada jerárquica conteniendo los valores de los "componentes del agrupamiento" solo una vez (se toma de la primera n-eada del grupo). El grupo completo de hecho está construido de los últimos componentes de la n-eada jerárquica.

El resultado final es:

$$<(P1,banca,<(P2,4),(P3,8)>)$$

$$(P2,silla,<(P5,4)>)$$

$$(P3,tabla,<(P4,1)>)>.$$

El esquema resultante es el de la figura 1.

Las operaciones de anidar y desanidar, permiten pasar de relaciones planas a relaciones anidadas (jerárquicas) y viceversa, informalmente anidar sobre un conjunto de objetos **Y** recoge en conjuntos aquellas subn-eadas sobre **Y**, que concuerdan con todos los valores diferentes a los atributos del anidamiento, desanidar invierte este proceso.

En particular, el operador anidar tiene las siguientes características:

. atributos simples o compuestos.

. niveles simples o compuestos.

En donde se tiene que:

Los atributos simples o compuestos se refieren al número permitido de atributos **Y** sobre los cuales el operador anidar se aplica, simple permite solo un elemento para **Y**, mientras que compuesto permite un conjunto de atributos para **Y**.

Los niveles simples o compuestos implican la posibilidad de que anidar se aplique o no a cualquier porción de un objeto de datos o solo al primer nivel de componentes.

El operador NST, de agrupamiento es tanto *multi-nivel* como *multi-atributo*. Los efectos de dicho operador son casi equivalentes a los de un anidamiento.

El operador NST de distribución (*D*), retiene las relaciones jerárquicas de los atributos que no toman parte en la distribución, también ofrece un poco de flexibilidad adicional al operador de desanidar, dado que permite "bajar" ciertos componentes seleccionados (atributos) en un subárbol, mientras que el operador anidar "tira" el subárbol completo y forza a la repetición de todos los atributos del nivel más alto para cada n-eada del subárbol.

B.4. OPERADORES NO SIMPLES.

El álgebra multitipificada del NST, integra los objetos atómicos y las operaciones sobre dichos objetos en un modelo formal, entonces, se pueden permitir expresiones arbitrarias del álgebra como parámetros de las operaciones selección, unión, **EXIST**, **FORALL** y del operador **LAMBDA**.

B.5. EXPRESIONES NST.

Una expresión del álgebra es ante todo un objeto sintáctico, que consiste de etiquetas posicionales, refiriéndose a objetos de datos y símbolos de operación puestos de acuerdo con las funciones y sintaxis de las operaciones. Obviamente en la ausencia de una colección de objetos de datos con etiquetas asociadas, la cual se le denomina una ***base NST de datos***, una expresión del álgebra no tiene sentido. En el contexto de una base de datos NST, sin embargo, la semántica de una expresión del álgebra es un objeto de datos específico consistente de algún esquema y una ocurrencia del esquema. La conexión de la expresión del álgebra con el objeto de datos es por medio de dos funciones semánticas ***sch*** y ***occ***. La función sch toma como parámetros la expresión y parte del esquema de la base de datos **NST**, este último consistente del conjunto de esquemas con sus etiquetas asociadas. La evaluación de sch genera el esquema de los objetos de datos, el cual es la semántica de la expresión del álgebra dada. Análogamente, occ toma una expresión y la parte de la ocurrencia de la base de datos NST y genera la ocurrencia del objeto de datos, se le denomina al segundo parámetro de sch y occ el ***medio ambiente*** en el cual la operación se interpreta.

El medio ambiente en que una operación se interpreta (la base de datos **NST**) es un objeto dinámico, el cual cambia cuando una expresión del álgebra ocurre anidada en otra expresión.

Considere por ejemplo, una selección, el operador *S* se aplica a una secuencia de n-eadas y el parámetro de la expresión se refiere a objetos de datos que son componentes de n-eadas. La semántica deseada es tal que para cada n-eada la expresión es evaluada una vez con respecto a los componentes de la n-eada.

Si el objeto resultante de la evaluación de la expresión es el objeto (BOOL, true) entonces la n-eada es "seleccionada", esto es, se convierte en un elemento de la secuencia resultante, en otro caso, se omite.

Similarmente, el operador **LAMBDA** se aplica a una secuencia de n-eadas y su expresión se refiere a los componentes de una n-eada. La semántica deseada, es otra vez que la expresión parámetro sea evaluada en cada n-eada, el objeto resultante es entonces agregado a la n-eada como un nuevo componente o reemplaza un componente existente, dependiendo de la sintaxis de la aplicación de **LAMBDA**.

El mecanismo formal de alcanzar lo mencionado anteriormente, consiste en que las funciones sch y occ interpretan una expresión parámetro **E** de *S* o de **LAMBDA**, una vez para cada una de las n-eadas **t** en la secuencia operando por medio de la construcción de un medio ambiente para **t**. El nuevo medio ambiente consiste de todos los objetos de datos que se conocían antes (en el medio ambiente en cual *S* o **LAMBDA** fue interpretado), más los objetos de datos que son componentes de la n-eada **t**.

B.6. EL USO DE OPERADORES NO SIMPLES.

Los operadores **LAMBDA**, selección (*S*), unión ($|X|$) pertenecientes a la **clase 5** y los operadores EXIST y FORALL (clase **10**), toman como parámetros expresiones del álgebra. El operador **LAMBDA** y el operador selección son operadores fundamentales, los otros pueden ser definidos en términos de **LAMBDA**, selección y otros operadores simples, por lo que la atención será concentrada en los dos primeros.

La selección no cambia el esquema del objeto de datos al que es aplicado. La transformación de la ocurrencia es como sigue:

El operador selección se aplica a una secuencia de n-eadas (la ocurrencia de los objetos de datos), y la expresión parámetro se refiere a los objetos que son componentes de las n-eadas.

Para cada n-eada de la secuencia de ocurrencia, la expresión parámetro es evaluada una vez con respecto a los componentes de esa n-eada. Si el objeto resultante de la evaluación de la expresión es el objeto de datos (**BOOL**, **true**), entonces, la n-eada se convierte en un elemento de la ocurrencia resultante, en otro caso la n-eada no pertenece a la secuencia resultado.

Considerese el objeto de datos "Productos" y la siguiente expresión del álgebra:

Productos *S* **[(Elementos** *S* **[Cantidad=3] COUNT) >= 2]**

Esta expresión selecciona aquellos productos cuya existencia de elementos componentes es de al menos dos en la cantidad dada (tres unidades).

LAMBDA es quizá la más importante operación en el álgebra **NST**, tiene dos funciones sobresalientes:

1) Es un operador general y poderoso para extender y reestructurar un objeto compuesto.

2) Permite la aplicación de cualquier expresión del álgebra a cualquier nivel de un objeto jerárquico.

LAMBDA acepta como parámetro una expresión arbitraria del álgebra, considera cada n-eada de su secuencia operando y evalúa su expresión parámetro en dicha n-eada. El objeto resultante puede ser o agregado a la n-eada como un componente nuevo o como reemplazo de alguno ya existente.

En general, **LAMBDA** puede tener una lista de expresiones como parámetro y agregar (reemplazar) varios componentes a (de) una n-eada de forma simultanea.

Un ejemplo de como extender un objeto con LAMBDA es:

Considerese el objeto de datos "Productos" y la expresión:

Productos LAMBDA [(Elementos COUNT) {número de elementos}]

La cual, agrega un nuevo componente al objeto "Productos", que contiene el número de elementos de cada producto, el nuevo componente es denominado "No_de_elementos" y el esquema resultante se muestra en la figura 3.

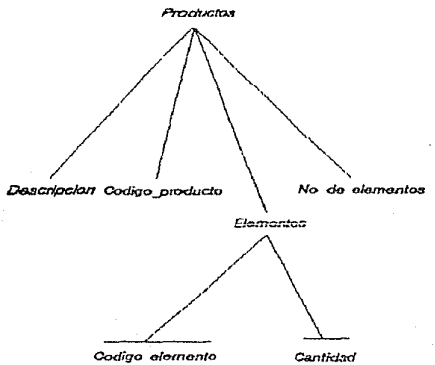

FIGURA 3.

La otra faceta importante de **LAMBDA**, es que permite la aplicación de una expresión del álgebra (sin limitación) a cualquier nivel del esquema del objeto, y por otra parte hace disponible cualquier componente del subárbol especificado para la aplicación de operaciones posteriores.

Lo anterior se basa en la capacidad de **LAMBDA** de reemplazar dentro de cada n-eada de la secuencia operando, un elemento por el resultado de una expresión. La sintaxis para ellos se puede ver como sigue:

<operando> LAMBDA [<atributo> : <expresión>]

Ejemplo: Para clasificar dentro del objeto "Productos" cada secuencia de elementos de los productos por la cantidad, se puede escribir:

Productos **LAMBDA** [[Elementos: Elementos **ORD** [Cantidad+]]

De forma más precisa, el resultado es una copia de la secuencia de productos donde para cada n-eada la secuencia elementos ha sido reemplazada por la secuencia de elementos ordenada.

Implícitamente, se aplica aquí el operador **ORD** al subárbol de elementos. Dado que aplicar un operador a un subárbol es una operación importante y general, esto se puede esquematizar como sigue:

Productos ORD [Elementos : Cantidad +]

cuyas semánticas están definidas por la expresión de arriba.

La facilidad mencionada anteriormente, está disponible para todos los operadores de la **clase 5 (COMP : COMP)**, sintácticamente, el subárbol al cual el operador se aplica se menciona en la lista de parámetros en primera instancia, seguido de un "punto y coma", obteniendo la siguiente sintaxis:

<operando> <operador> [<subárbol>;<lista>]

En donde <lista> es la lista normal de parámetros del operador.

El subárbol puede estar a cualquier profundidad del esquema del operando, por lo consiguiente varios **LAMBDAS** pueden estar anidados.

LAMBDA también puede hacer un subárbol a cualquier profundidad disponible para un operador binario, por ejemplo la unión.

Considere los dos esquemas de la **figura 4** con las siguientes ocurrencias:

<(LOPEZ,<((P1,2),(P2),(P3,2)>,1987),
(<(P2,4)>,1986) >),
(SANCHEZ,<(<(P3,1),(P2,4)>,1980>)>.

y

<(P1,banca,80),
(P2,silla,10),
(P3,tabla,40)>

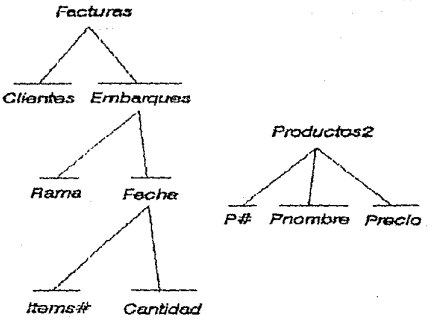

FIGURA 4.

Se asume que se están construyendo las facturas, para continuar con este proceso es necesario determinar el precio de cada ítem, esto puede ser hecho por medio de unir cada secuencia de ítems con la secuencia de productos:

FACTURAS LAMBDA [Embarques:Embarques LAMBDA [ítems:ítems productos union [ítem#=p#]]].

o la expresión corta

FACTURAS LAMBDA [Embarques ; ítems: ítems productos union [ítem#=p#]].

Lo que se puede leer como: "Tomar la secuencia de Facturas, recorrer a través de los embarques; dentro de cada embarque reemplazar los componentes ítems por la unión de la secuencia ítems con la secuencia productos".

Sin embargo, **LAMBDA** puede también percibirse desde un punto de vista puramente funcional como "un operador de modificación de n-eadas".

El esquema resultante se muestra en la **figura 5** y la ocurrencia resultante es:

< (LOPEZ,< (< (P1,2,P1,Banca,80),
(P2,2,P2,Silla,10),
(P3,2,P3,Tabla,40)>,1987),
(< (P2,4,P2,Silla,10)>,1986) >),
(SANCHEZ,< (<(P3,1,P3,Tabla,40),
(P2,4,P2,Silla,10) > 1980) >)>.

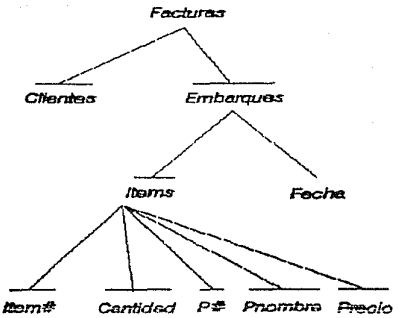

FIGURA 5

Por otra parte, **LAMBDA** puede ser utilizado para realizar operaciones binarias involucrando partes de la misma jerarquía.

B.7. EL MANEJO DE VALORES NULOS EN EL MODELO NST.

Algunas de las definiciones formales de los modelos de "*relaciones anidadas*" no orientan el problema de atributos valuados como *nulos* **[SP 82]**, **[SCS 86]**, sin embargo, existen enfoques que permiten conjuntos vacíos para atributos valuados por conjuntos y son asociados con la **interpretación inexistente [MAKI 77]**, sin embargo, el tópico del tratamiento de nulos no ha sido completamente desarrollado.

En el modelo **VERSO [ABIT 84]**, los conjuntos vacíos son permitidos para atributos valuados por conjuntos, pero no se consideran valores nulos para atributos valuados de forma sencilla.

Roth, Korth y Silbertchatz [RKS 85] proponen un manejo formal de nulos en el modelo de "relaciones anidadas", asumen un *mundo abierto* **[REIT 78]**, permiten tres interpretaciones de nulos, **desconocido, inexistente y no hay información [ZANI 84b]** como valores posibles para atributos valuados en forma sencilla y el conjunto vacío como el valor nulo para atributos compuestos.

Un conjunto vacío se interpreta como "*no hay información*", en este enfoque la interpretación "*inexistente*" para conjuntos vacíos está prohibida, extendiendo el concepto de **Zaniolo** de "*n-eada más informativa*" al "*modelo de relaciones anidadas*".

Güting, Zicari y Choy [GZC 87] proponen un tratamiento alternativo de los valores nulos para el modelo NST, el enfoque nulo en el modelo extiende dicho esquema con tres valores nulos para atómicos, **desconocido, inexistente y no hay información**, en cuanto a los componentes no atómicos, se maneja la **secuencia vacía** y la **secuencia no hay información**, la secuencia vacía está asociada con la interpretación inexistente, mientras que la secuencia no hay información denota una secuencia de la que nada es conocido. Los autores también, asumen un "mundo abierto" y el concepto de "n-eada más informativa" es extendido a secuencias anidadas, sin embargo, las "n-eadas menos informativas" no tienen que ser forzosamente eliminadas **[RKS 85]**.

APENDICE C. *MANEJO DE RELACIONES CON VALORES NULOS.*

En esta parte se analiza un nuevo enfoque formal para el manejo de información incompleta en las bases de datos, a través de ***valores nulos***, la base de este enfoque consiste en la interpretaciones de dichos valores nulos, los cuales requieren de más de un ***tipo de valor nulo***.

La validez conceptual de este enfoque, se sustenta en la generalización de la estructura formal del modelo relacional al considerar valores nulos. En particular se pone especial énfasis en las propiedades de su conjunto teórico y se generalizan las definiciones de contención, unión y diferencia de conjuntos.

La gente involucrada en el proceso de diseño, implantación y mantenimiento de bases de datos, ha reconocido la conveniencia de manejar símbolos especiales para el manejo de información incompleta en la base de datos, estos símbolos especiales son conocidos como *valores nulos*.

En los últimos tiempos, una serie de investigaciones se ha enfocado al tópico de la información incompleta y los valores nulos **[BISK 81]**, **[CODD 79]**, **[GRANT 77]**, **[IMIEL 81]**, **[LACROIX 76]**, **[LIEN 79]**, **[LIPSKI 79]**, **[REITER 83]**, **[VASS 79]**, **[WONG 82]**, **[ZANI 77]**. Los resultados obtenidos de estos trabajos determinan que ciertas áreas dentro de este tópico continúan abiertas.

Un problema básico con los valores nulos, es las diversas interpretaciones que estos tienen. Por ejemplo, existen autores que asientan catorce diferentes interpretaciones de nulos, sin embargo, la mayoría coincide en que la diversidad de los nulos puede ser reducida a dos interpretaciones básicas:

- La ***interpretación desconocido***, en donde el valor existe pero no es conocido.

- La ***interpretación inexistente***, en donde el valor no existe.

E.F. Codd [CODD 79], propuso un tratamiento formal bajo la ***interpretación desconocido***, el cual usa una lógica trivaluada, que junto con los usuales ***VERDADERO*** y ***FALSO***, incrementa un valor adicional ***QUIZÁ***. Las expresiones relacionales tradicionales como $X > Y$, siempre son evaluadas a verdadero o falso, si **X** y **Y** son no nulos, sin embargo, bajo la óptica de Codd si **X** o **Y** son nulos, la expresión se evalúa como quizá.

Al mismo tiempo, Codd propuso una extensión al álgebra relacional, donde las operaciones de selección, unión y proyección vienen en dos diferentes versiones:

. ***La versión verdadera*** y

. ***La versión quizá***.

Por ejemplo, el resultado de una selección verdadera contiene sólo n-eadas en donde la expresión de selección se evalúa a verdadero. La versión quizá contiene aquellas n-eadas en donde la expresión de selección se evalúa a quizá. Codd sugiere que los sistemas relacionales de peticiones se extiendan para permitir a los usuarios recobrar, no sólo las n-eadas que satisfagan la petición en verdadero, sino que también cumplan el sentido quizá.

Hay problemas con el tratamiento de Codd a los valores nulos, una primera área de problemas es la relativa a factores prácticos y el principal de ellos es el alto costo que se involucra para una pequeña información adicional, al caracterizar las versiones quizá de las peticiones.

Una segunda área de problemas es la relativa a factores lógicos, se puede ver cuando la lógica trivaluada no responde correctamente a la *interpretación desconocido* de los nulos, esto es, ciertas expresiones que deberían evaluarse siempre como verdadero (**tautologías**), son algunas veces evaluadas a quizá.

Hay un segundo problema lógico que no se ha mencionado, este tiene que ver con el conjunto de propiedades de las relaciones y su comportamiento en la actualización.

Por ejemplo, supóngase las siguientes dos instancias de la relación **partes-proveedores** (en donde **w** significa nulo).

PS'(P#,S#)
w s1
p1 s2
PS"(P#,S#)
w s1
p1 s2
p2 s2

Bajo el enfoque de **Codd**, una expresión como:

$$PS' \subseteq PS''$$

Es evaluada utilizando el ***principio de sustitución nula***, el cual reemplaza cada ocurrencia de **w** por un distinguible y posible de valor no nulo. Por lo tanto, cualquier expresión que evalúa todas las sustituciones como verdadero, será evaluada como verdadera, sin embargo, si para una expresión se evalúan las sustituciones como verdadero y falso dependiendo de los valores utilizados para la sustitución, la expresión será evaluada como quizá. Si el valor **w** para **PS'** y el valor **w** para **PS"** se sustituyen por el mismo valor por decir, p_1, la expresión genera verdadero, pero si la expresión en **PS'** es substituida por p_2 y por p_1 en **PS"**, la expresión será evaluada como falso.

Notese que PS" puede obtenerse de PS' al agregar la n-eada (p_2,s_2), luego entonces, el usuario común después de agregar nueva información, espera que su base de datos **contenga** su información anterior como una realidad (verdadero) y no como una especulación (quizá).

Por lo tanto, la definición de contención de conjuntos en la lógica trivaluada de **Codd**, no es un modelo que permita el entendimiento intuitivo del comportamiento dinámico de un sistema de información. Más aún, las operaciones con conjuntos propuestas por él, no poseen aún muchas de las propiedades del álgebra de conjuntos, por ejemplo:

$$PS' = PS'$$

y

$$PS' = PS''$$

Son evaluadas a quizá y no a verdadero y falso como es esperado.

Entonces, la generalización del conjunto teórico de propiedades de las relaciones en presencia de valores nulos, representa un problema abierto, el cual es muy importante toda vez que la teoría de conjuntos conforma los cimientos en los que descansan tanto el modelo relacional como la construcción de un álgebra completa que incluye las operaciones de unión y diferencia.

Por otra parte, el problema de la ***interpretación inexistente*** de los nulos, fue estudiada por varios autores, de los cuales sobresale el trabajo realizado por **Lien [LIEN 79]**, quien propuso operaciones de unión y selección similares a las versiones verdadero de **Codd**, procediendo a la formalización del concepto de dependencias multivaluadas con nulos, para las cuales derivó un conjunto completo de reglas de inferencia.

La tercera faceta de los valores nulos, se relaciona con el manejo simultaneo de las dos interpretaciones de valores nulos, este problema fue revisado por **Vassiliou [VASS 79]**, quien encontró problemas muy serios de semántica al intentar extender la lógica trivaluada a una lógica tetravaluada. Vassiliou, también mostró que la interpretación de peticiones en presencia de valores nulos puede solucionarse en la estructura denotacional semántica de Scott, sin embargo, la desventaja de tal concepto consiste en el alto costo computacional para la evaluación de las peticiones.

La faceta final del escenario de los valores nulos, no se orienta a la generalización de conceptos viejos, por el contrario, explora y propone herramientas conceptuales para el desarrollo de aplicaciones que tienen que manejar valores nulos. De dichas aplicaciones se pueden mencionar las siguientes:

- La utilización de valores nulos para el traslado de esquemas de red a esquemas relacionales **[LIEN 82]**.

- Las bases de datos distribuidas.

- La consideración del concepto relación universal **[FAGIN 82]**.

Por lo tanto, existe la necesidad de integrar esos nuevos conceptos y las aplicaciones adecuadas en una estructura completa y consistente.

El enfoque que presenta este estudio, evita el dilema de la ***interpretación desconocido*** contra la ***interpretación inexistente***, y provee una extensión que preserva dos de las ventajas del modelo relacional:

- Fundamentos de la teoría de conjuntos.

- Algoritmos eficientes, basados principalmente en la correspondencia ampliamente conocida entre el cálculo relacional y el álgebra relacional.

C.1. UN NUEVO ENFOQUE PARA EL TRATAMIENTO DE LOS NULOS.

El enfoque, está fundamentado en el principio que la interpretación inexistente y la interpretación desconocido, no constituyen la más básica y elemental interpretación de los valores nulos, por lo que hay una interpretación más primitiva y menos pretenciosa sustentando las otras dos mencionadas.

Para ilustrar este punto basta considerar una aplicación típica de los valores nulos:

Sea una base de datos que contiene la relación EMP con los siguiente atributos:

EMP(E#, Nombre, Sexo, Jefe#) (2.1)

EMP (E#	Nombre	Sexo	Jefe#)
1120	PEREZ	M	2235
4335	LOPEZ	F	2235
8799	ROBLES	M	1255

TABLA I. La relación empleado.

Se dice que la instancia de la relación es la que representa la **tabla I**, sin embargo, anticipándose a las futuras necesidades, el diseñador decide cambiar el esquema e incluir un nuevo atributo a la relación, denominado **Tel#**.

EMP(E#, Nombre, Sexo, Jefe#, Tel#) (2.2).

Que debe contener el número telefónico del empleado, sin embargo, por el momento no se piensa solicitar a todos los empleados su número telefónico, se irá incorporando este dato cuando se disponga de él. De lo anterior, se desprende que el administrador se enfrenta con el problema de operar por el momento con un esquema expandido, mientras que no ha habido cambio en la información de la base de datos.

Una solución, es ver la información de acuerdo a la **tabla II**, a la cual se le ha agregado el símbolo '-' para llenar el campo Tel#, es decir, el símbolo se utiliza para denotar un valor nulo. La **tabla II** muestra claramente la utilidad de los valores nulos, se puede apreciar que en este caso ninguna de las dos interpretaciones de nulos (desconocido e inexistente) es aplicable directamente.

EMP (E#	Nombre	Sexo	Jefe#	Tel#)
1120	PEREZ	M	2235 -	
4335	LOPEZ	F	2235 -	
8799	ROBLES	M	1255 -	

TABLA II. La relación empleado después de la adición del nuevo atributo TEL#.

En el caso que nos ocupa, el valor nulo únicamente denota el concepto de ***no hay información*** en el campo Tel#, por lo que puede interpretarse como un espacio para almacenar un **valor desconocido** o un **valor inexistente.**

Más aún, si '-' fuese interpretado como desconocido o como inexistente, entonces la **tabla II** debería **"contener más"** información que la **tabla I**, esto contravendría la suposición de que no hay información adicional al realizarse el cambio de esquema.

Bajo el enfoque de interpretación ***no hay información*** de los nulos, es correcto aseverar que las **tablas I** y **II** son equivalentes en ***información inteligente***. La noción de equivalencia en información inteligente, es un punto central en el nuevo enfoque de tratamiento de los nulos y será discutido formalmente más adelante.

Obviamente las relaciones mencionadas, son equivalentes en términos de contenido, pero la información intencional representada en los dos esquemas es diferente. Por ejemplo si E# es una llave, entonces la dependencia funcional de Tel# sobre E# es considerada en (2.2) pero no en (2.1).

En resumen, no hay solución disponible actualmente para el manejo simultaneo de las interpretaciones desconocido e inexistente de los nulos, más aún, si existiera la solución quedaría pendiente el problema de aquellas situaciones en donde cualquiera de las interpretaciones puede ser asignada al nulo, ya que una asignación arbitraria no es factible de ser llevada a cabo en la representación física de la base de datos.

Por otra parte, si un valor nulo bajo la interpretación de no hay información se usa para modelar cualquier tipo de información incompleta, entonces la información almacenada en la base de datos es factible y correcta, sin embargo, debe señalarse que parte del conocimiento que pudiera estar disponible al usuario de la información puede perderse, lo anterior es el precio que debe pagarse en muchas situaciones.

El argumento básico de solidez del enfoque no hay información, es el principio consistente en que la base de datos sólo puede proveer de una aproximación del mundo real.

Muchos tipos de nulos pueden establecerse para mejorar la aproximación que dota la base de datos, por ejemplo, considérese el esquema de base de datos (2.2). Si no se permitiera el valor nulo en el atributo Tel#, no podría almacenarse información de los empleados que no tienen teléfono, o de aquellos que por alguna razón su número telefónico no estuviera disponible.

El uso del enfoque no hay información, mejora de manera radical la exactitud y completez de la base de datos, ya que la información disponible (E#,nombre,sexo y Jefe#) puede almacenarse en registros con un nulo en el atributo Tel#, debe admitirse que esta aproximación puede mejorarse utilizando dos tipos de nulos, uno para denotar la inexistencia de teléfono y otro para establecer que el número es desconocido, sin embargo, la complejidad que se agrega no establece un solución determinante, únicamente mejora la aproximación, veamos esto último con un ejemplo:

Un caso de información incompleta que ocurre muy a menudo en la vida real, es por ejemplo la que determina la sentencia "El jefe de Ernesto Pérez es una mujer", la cual establece que aunque la identidad del jefe de Ernesto Pérez no se conoce si se sabe su sexo. Para preservar este tipo de información, se tienen que utilizar nulos marcados para asociar diferentes n-eadas, en el caso de la relación EMP, como ejemplo, debe existir una n-eada con un valor nulo único E# en donde el valor de Sexo es F (femenino) y entonces el mismo valor nulo único tiene que aparecer en Jefe# del registro de Ernesto Pérez. Entonces, mientras este valor nulo marcado será tratado como un valor desconocido común cuando se realice una operación de selección, el

mismo valor será considerado como un valor no nulo bajo la operación de unión sobre Jefe#.

En la práctica, sin embargo se puede conoce mucho acerca de un valor desconocido, por ejemplo, aunque no se conoce exactamente el color de un objeto, se puede saber que los únicos colores posibles son rojo o azul, de manera más general, se puede asociar una distribución de probabilidad para un valor desconocido en un dominio. La información adicional, puede ser asociada a tipos muy sofisticados de nulos, sin embargo la calidad de información que se obtiene no justifica la complejidad agregada.

En realidad, cuando el conocimiento adicional es muy importante, generalmente es más sencillo conservarlo por medio de modificaciones al esquema que a través de valores nulos complejos. Por ejemplo, si fuese necesario registrar que un empleado no tenga teléfono, podría agregarse un atributo NUM_TEL el cual contendría un cero en el registro de tal empleado.

Concluyendo, un sistema de bases de datos puede aproximar el mundo real solo hasta cierto grado de precisión. El grado elegido para un sistema debe haber considerado los aspectos prácticos. En la elección de utilizar la interpretación no hay información para los nulos, se acepta que la aproximación mencionada es muy burda, sin embargo, se obtienen varios beneficios a cambio, la primera ventaja obvia es la generalidad que se obtiene al determinar que un tipo de nulo puede contener cualquier manifestación de información incompleta, la segunda ventaja es la simplicidad conceptual, la cual orienta a una generalización simple del modelo relacional. Por último, una ventaja final, será la eficiencia computacional en el proceso de evaluación de peticiones.

C.2. CONCEPTOS BASICOS DEL NUEVO ENFOQUE.

A efecto de llevar a cabo un análisis del nuevo enfoque, es necesario revisar una serie de notaciones y convenciones.

Una relación con valores nulos se denotará simplemente como ***relación***.

Una relación sin valores nulos, se denotará como ***relación definida completamente*** o como ***relación total***.

Una relación R definida sobre el conjunto de atributos $W=\{A_1,...,A_n\}$, se denota como $R(W)$.

Cada atributo A_i contenido en W, tiene un dominio denotado como $\mathbf{DOM}(A_i)$ el cual se extiende para incluir un símbolo especial **ni** que denota el ***valor nulo bajo la interpretación no hay información***.

Para cada $A \in W$, un *A-valor* es una asignación del *A-dominio extendido*, generalizando esta noción, el concepto *X-valor* en donde X está contenido propiamente en W ($X \subseteq W$), es una asignación de valores a los atributos en X de sus respectivos dominios extendidos.

Una relación R(W) es un conjunto de *W-valores*, los elementos de este conjunto son denominados ***renglones o n-eadas*** de R.

Una relación puede representarse como una tabla en donde los renglones son las n-eadas de la relación y las columnas corresponden a los atributos de la misma.

En las tablas se puede representar ni por el símbolo '-' (guión).

Sea r un W-valor, es decir, una n-eada de R(W), sea A $\underline{E}$ W y

X $\underline{C}$ W.

Entonces, r[A] y r[X], denotan el *A-valor de* y el *X-valor* de r respectivamente.

Se asume sin pérdida de generalidad, que los atributos de las relaciones están contenidos en un universo finito de atributos, el cual se denota como *U*.

En aspectos de notación, se utilizan las primeras letras del alfabeto como A,B, y C para denotar atributos simples en U y las últimas letras como W,X,Y, y Z para denotar subconjuntos de U.

Un X-valor r se dice que es *más informativo* que un Y-valor t, cuando para toda B $\underline{E}$ Y, si t[B] es *no nulo*, entonces, B $\underline{E}$ X y r[B]=t[B]. Esto es, r debe coincidir en cada valor no nulo de t, se denota $r>=t$ si r es más informativa que t, por analogía si $t<=r$ se dice que t es menos informativa que r.

Si $r>=t$ y $t>=r$, entonces se dice que t y r son *equivalentes (información inteligente)* y se denota como *rPt*, la relación es >= es reflexiva y transitiva, entonces P es de equivalencia, una vez que también es simétrica.

Sean X $\underline{C}$ Y $\underline{C}$ U, r un X-valor dado, entonces, un Y-valor que cumpla con tPr puede construirse a partir de r llenando los *(Y-X)-valores* con nulos, entonces puede establecerse la siguiente convención, si r es un X-valor y el atributo A no pertenece a X, entonces r[A] P ni.

Por lo tanto, cualquiera dos n-eadas consistentes solo de valores nulos son equivalentes y cualquier n-eada es equivalente a la n-eada consistente de ni. Esas n-eadas son denominadas *n-eadas nulas*. Una n-eada sin nulos es denominada **n-eada total** y una n-eada con un X-valor total se conoce como *n-eada X-total*.

Si U^* denota el conjunto de todas las n-eadas posibles, es decir, el conjunto que contiene todo X-valor para cada X $\underline{C}$ U, entonces, la noción de más informativa al ser transitiva y reflexiva, establece un *cuasi-orden de* U^*.

Una n-eada t será denominado un *encuentro* de dos n-eadas r_1 y r_2 denotado como t P (r_1 y r_2), cuando para cada A $\underline{E}$ U:

$$t[A]=r_1[A] \quad \text{si } r_1[A] = r_2[A]$$

$$t[A]=ni \quad \text{si } r_1[A] <> r_2[A]$$

Notese que en esta definición, no es importante si se asume que ni=ni o ni<>ni.

Si r_1'P r_1 entonces (r_1 y r_2) P (r_1' y r_2), por lo tanto si no se distingue entre n-eadas equivalentes, **siempre existe el encuentro de cualquiera dos n-eadas en U^* y este es único.**

El **encuentro** de r_1 y r_2 es más informativo que cualquier otra n-eada que sea menos informativa que r_1 y r_2.

Mientras existe un encuentro para cualesquiera dos n-eadas en U^*, la unión puede no existir, dos n-eadas r_1 y r_2, se dice que son *unibles* cuando lo siguiente es verdadero para cada atributo $A \in U$:

$$\text{Si } r_1[A] <> r_2[A] \text{ entonces } r_1[A]=ni \text{ o } r_2[A]=ni.$$

Una n-eada t se denominará la *unión* de dos n-eadas η y r_2 denotado como $t \mathrel{P} (\eta \text{ y } r_2)$, cuando r_1 y r_2 son **unibles** y para cada A que pertenece a U, se cumple que:

$$t[A]=r_1[A] \quad \text{si } r_1[A] >= r_2[A]$$

$$t[A]=r_2[A] \quad \text{si } r_2[A] >= r_1[A]$$

Si $r_1' \mathrel{P} r_1$, entonces, r_1' y r_2 son **unibles** si y solo si (*ssi*) r_1 y r_2 **existen** y se cumple que:

$$(r_1' \circ r_2) \mathrel{P} (r_1 \circ r_2) \text{ y}$$

$$(r_1' \text{y } r_2) \mathrel{P} (r_1 \text{ y } r_2).$$

Regularmente, se evitan distinciones entre n-eadas equivalentes y se habla de la unión o encuentro de dos n-eadas, si no se distingue entre n-eadas equivalentes, la relación >= define un ordenamiento parcial.

En este contexto, se dice que la unión de r_1 y r_2 es la menos informativa de las n-eadas que son más informativas que r_1 y r_2, por otro lado el encuentro de r_1 y r_2 es la más informativa entre las n-eadas que son menos informativas que r_1 y r_2.

C.3. RELACIONES EXTENDIDAS.

La noción de ser más informativo se puede extender a relaciones, de las cuales se puede decir que son más informativas que otras relaciones o que *subsuman* a otras relaciones.

Definición 4.1.

Una relación R_1 *subsume* a una relación R_2, lo cual se denota como $R_1 >= R_2$ si para toda n-eada no nula $r_2 \in R_2$ existe una n-eada $r_1 \in R_1$ que $r_1 >= r_2$.

Ya que >= es una relación transitiva y reflexiva, se puede definir la noción de equivalencia en información inteligente como sigue:

Definición 4.2.

Las relaciones R_1 y R_2 son equivalentes en información inteligente, lo cual se denota como $R_1 \mathrel{P} R_2$, cuando $R_1 >= R_2$ y $R_2 >= R_1$.

La relación de equivalencia ^{P} (reflexiva, simétrica y transitiva) particiona el universo de relaciones en clases ajenas, por lo que se utilizará básicamente el mismo mecanismo de extensión de los números enteros a los números reales para introducir la noción de *relaciones extendidas* las cuales se les denomina *x-relaciones*.

Definición 4.3.

Una *x-relación* es una clase de equivalencia bajo ^{P} y la clase de relaciones equivalentes de R se denota como R^0, se dice que R es una representación de R^0.

Entonces, $R^0_1 = R^0_2$ ssi $R_1 {}^{P}R_2$. Si $R_1' {}^{P}R_1$ y $R_2' {}^{P}R_2$, entonces $R_1' >= R_2'$ssi $R_1 >= R_2$, por lo tanto se puede definir las nociones de inclusión o contención de conjuntos para las X-relaciones.

Definición 4.4.

Si R^0_1 contiene a R^0_2 cuando $R_1 >= R_2$.

Claramente si $R^0_1 {}^{P}R^0_1$, $R_2' {}^{P}R_2$ y $R^0_1 >= R^0_2$ entonces $R^0_1' {}^{P}R^0_2'$ como se esperaba, surgiendo así la siguiente proposición:

Proposición 4.1. $R^0_1 = R^0_2$ ssi $R^0_1 >= R^0_2$ y $R^0_2 >= R^0_1$.

También se puede decir que R^0_1 contiene propiamente a R^0_2, cuando R^0_1 contiene a R^0_2 pero $R^0_1 = R^0_2$. Es conveniente generalizar la noción de una n-eada, como un elemento o miembro de una x-relación como sigue:

Definición 4.5.

Una n-eada t se dice que x-pertenece a, o es un x-elemento de R^0 ($t {}^{E} R^0$) cuando para algún R'en R^0, $t {}_{E} R'$.

Proposición 4.2. $t {}^{E} R^0$ ssi existe una n-eada $r {}_{E} R$, tal que r>=t. Esto es una n-eada t pertenece a una x-relación ssi su representación contiene una n-eada que es más informativa que t, también, se dice que una n-eada t x-pertenece a una relación R para denotar que para alguna $r {}_{E} R$, r>=t. Aunque ahora la x-pertenencia se usa en dos diferentes contextos no existe confusión, ya que $t {}^{E} R^0$ ssi $t {}^{E} R$.

Dado un conjunto de n-eadas, $\{t_1,t_2,...,t_n\}$ se pueden eliminar todas las n-eadas que sean menos informativas que algunas otras n-eadas, y completar las otras a sus U-valores equivalentes.

La x-relación representada por el conjunto de U-valores obtenidos de la manera anterior se denota como sigue:

$$\{t_1,t_2,...,t_n\}^0.$$

Ahora se puede definir la unión, la x-intersección y la diferencia utilizando los conceptos vistos como sigue:

(4.1)

Unión:

$$R^{o}_{1} \text{ unión } R^{o}_{2} = \{r/r \in R^{o}_{1} \text{ o } r \in R^{o}_{2}\}^{o}.$$

(4.2)

X-intersección :

$$R^{o}_{1} \text{ x-intersección } R^{o}_{2} = \{r/r \in a\ R^{o}_{1} \text{ y } r \in R^{o}_{2}\}^{o}.$$

(4.3)

Diferencia:

$$R^{o}_{1} - R^{o}_{2} = \{r/\ r \in R^{o}_{1} \text{ y r No x-pertenece a } R^{o}_{2}\}.$$

Proposición 4.3. Si $R^{o}_{1}' = R^{o}_{1}$ y $R^{o}_{2}' = R^{o}_{2}$, entonces:

1) R^{o}_{1}' unión $R^{o}_{2}' = R^{o}_{1}$ unión R^{o}_{2}.

2) R^{o}_{1}' x-intersección $R^{o}_{2}' = R^{o}_{1}$ x-intersección R^{o}_{2}.

3) $R^{o}_{1}' - R^{o}_{2}' = R^{o}_{1} - R^{o}_{2}'$.

La unión y la x-intersección respectivamente, definen la mínima cota superior y la máxima cota inferior con respecto al ordenamiento parcial $>=$.

Proposición 4.4. Si $R^{o} >= R^{o}_{1}$ y $R^{o} >= R^{o}_{2}$, entonces

$$R^{o} >= R^{o}_{1} \text{ unión } R^{o}_{2}.$$

Prueba: Si $r \in (R^{o}_{1} \text{ unión } R^{o}_{2})$, entonces se tiene que $r \in R^{o}_{1}$ o $r \in R^{o}_{2}$, si suponemos que $r \in R^{o}_{1}$, entonces $r \in R^{o} >= R^{o}_{1}$, entonces $R^{o} >= (R^{o}_{1} \text{ unión } R^{o}_{2})$.

Proposición 4.5. Si $R^{o} >= R^{o}_{1}$ y $R^{o} >= R^{o}_{2}$, entonces

$$R^{o} >= (R^{o}_{1} \text{ x-intersección } R^{o}_{2}).$$

De los resultados anteriores se concluye que se tiene una latíz de x-relaciones con las propiedades bien conocidas asociadas a ella, esta latíz es también distributiva:

(4.4)

R^{o}_{1} x-intersección (R^{o}_{2} unión R^{o}_{3}) =

(R^{o}_{1} x-intersección R^{o}_{2}) unión (R^{o}_{1} x-intersección R^{o}_{3})

(4.5)

R^{o}_{1} unión (R^{o}_{2} x-intersección R^{o}_{3})=

(R^{o}_{1} unión R^{o}_{2}) x-intersección (R^{o}_{1} unión R^{o}_{3})

La latíz tiene un elemento mínimo, denotado como O^0, el cual se caracteriza por la propiedad de que para toda x-relación R^0:

$$(R^0 \text{ x-intersección } O^0) = O^0.$$

O^0 puede ser representado por una relación vacía.

El elemento máximo de nuestra latíz denotado como TOP_U, está caracterizado por la propiedad de que:

$$(R^0 \text{ unión } TOP_U) = TOP_U \text{ para toda } R^0.$$

Si $U=\{A_1,...,A_P\}$ entonces TOP_U puede ser representado por:

$$TOP_U = DOM(A_1) X ... X DOM(A_P).$$

Notese que TOP_U es un subconjunto propio de U^*.

En general una x-relación R^0 no tiene un complemento, es decir, no hay una relación $R^{0'}$ para la cual:

$$R^0 \text{ intersección } R^{0'} = O^0 \text{ y } R^0 \text{ unión } R^{0'} = TOP_U.$$

Lo anterior se puede ver en el siguiente ejemplo:

$$U=\{A,B\}, DOM(A)=\{a1\}, DOM(B)=\{b1,b2\}$$

Las siguientes 2 n-eadas son x-elementos de TOP_U:

$$r1:(a1,b1),\ r2:(a1,b2)$$

Si se toma una relación R x-conteniendo r_1 pero no a r_2, entonces una x-relación $R^{0'}$ para generar:

$$R^0 \text{ unión } R^{0'} = TOP_U$$

Debe tener r_2 como un x-elemento, r_2 x-pertenece a $R^{0'}$ pero entonces la n-eada $(a_1,-)$ x-pertenece a ambos R^0 y $R^{0'}$, por lo tanto, pertenece a $(R^0$ x-intersección $R^{0'}) <> O^0$.

De manera similar, la intersección:

$(R^0_1 - R^0_2)$ x-intersección R^0_2, no puede ser vacía, sin embargo, las siguientes dos propiedades se mantienen:

Proposición 4.6. Para cualesquiera dos x-relaciones R^0_1 y R^0_2 donde $R^0_1 >= R^0_2$

$$(R^0_1 - R^0_2) \text{ unión } R^0_2 = R^0_1.$$

Proposición 4.7. Si $(R^0$ unión $R^0_2) = R^0_1$ entonces:

$$R^0 >= (R^0_1 - R^0_2).$$

Entonces, (R^o_1 - R^o_2) es la más pequeña x-relación cuya unión con R^o_2 genera R^o_1.

La noción de representación mínima es conveniente para la representación y manejo de x-relaciones.

Definición 4.6. Una relación R constituye una *mínima representación* para R^o, cuando ningún subconjunto propio de R es representación de R^o.

La representación mínima de una x-relación sobre un conjunto de atributos es única, sin embargo, puede tener dos representaciones mínimas distintas sobre dos conjuntos diferentes de atributos.

Definición 4.7. El conjunto de atributos W se dice que es el *alcance* de R^o, cuando R^o puede ser representado por cualquier relación en W pero no puede ser representado por una relación con un conjunto de atributos más pequeño que W.

Las definiciones enunciadas como tales de (4.1) a (4.3), no conducen a una implementación eficiente, en realidad, la definición de x-pertenencia sugiere una explosión combinatoria en la cual una gran cantidad de n-eadas menos informativas son evaluadas y posiblemente incluidas en la relación resultante. Este problema puede resolverse derivando formulaciones equivalentes, las cuales no utilizan la operación de x-pertenencia, por ejemplo, las siguientes tres pueden derivarse de (4.1) a (4.3):

(4.6)

R^o_1 unión R^o_2=

$$\{r/r \in R_1 \text{ o } r \in R_2\}^o$$

(4.7)

R^o_1 x-intersección R^o_2=

$$\{r_1 \text{ y } r_2 / r_1 \in R_1 \text{ y } r_2 \in R_2\}^o$$

(4.8)

R^o_1 - R^o_2=

$$\{r/r \in R_1 \text{ y para toda } t \in R_2\text{: no } (t>=r)\}^o.$$

Entonces el alcance de la unión, es la unión de los alcances de sus operandos: el alcance de la x-intersección no es más grande que la intersección de los alcances de los operandos; el alcance de la diferencia no es más grande que el alcance del minuendo.

Una implementación pensada a la ligera de (4.6), genera un tiempo de corrida del orden de $|R_1| + |R_2|$, mientras que (4.7) y (4.8) sugieren una cota superior del orden de $|R_1| \times |R_2|$.

Sin embargo, técnicas más sofisticadas como una dispersión combinatoria, puede proveer de soluciones más eficientes. Esas técnicas pueden ser útiles para la reducción de las relaciones a una forma mínima. Por ejemplo, si R_1 y R_2 son representaciones mínimas de R^o_1 y R^o_2, entonces, el resultado de (4.8) provee de una representación mínima para R^o_1-R^o_2.

Sin embargo, aunque R_1 y R_2 sean mínimas, la aplicación de (4.6) y (4.7) puede introducir n-eadas menos informativas, las cuales tienen que ser eliminadas para reducir las relaciones a su forma mínima.

C.4. EVALUACION DE PETICIONES.

Como asienta **Lipski** [LIPSKI 79], si una petición **Q** es formulada en una base de datos con información incompleta, existen dos cotas o fronteras de interés:

(1) Una cota inferior $||Q||_*$: conjunto de objetos los cuales sobre la base de *información disponible*, se puede concluir que satisfacen **Q** de **manera segura.**

(2) Una cota superior $||Q||^*$: conjunto de objetos los cuales pueden satisfacer posiblemente **Q**, es decir, sobre las bases de la información disponible no pueden ser constadas de manera segura.

En este estudio, el interés está en el problema de evaluar la cota inferior $||Q||_*$ para un lenguaje basado en cálculo relacional o álgebra relacional, este es el tópico de más interés en las situaciones del mundo real.

La evaluación de la cota superior $||Q||^*$ es de menos interés práctico.

La solución propuesta, es similar a la solución de **Codd**, puesto que emplea una lógica trivaluada, sin embargo, utiliza una interpretación diferente de esta lógica y un tratamiento de conjuntos.

Los lenguajes basados en cálculo de predicados contienen expresiones relacionales como:

t.A OPE m.B

t.A OPE k

En donde **t** y **m** son n-eadas. **A** y **B** son atributos, **k** es una constante (no nula), **OPE** es uno de los operadores de comparación (<,>,<>,>=,<=).

Si los A-valores de **t** es nulo, entonces, las dos expresiones relacionales evalúan a **ni**, también si **m.B** evalúa a nulo, entonces **t.A OPE m.B** evalúa a **ni**, en cualquier otro caso, las expresiones evalúan a verdadero o falso como es usual, las expresiones booleanas combinando expresiones relacionales se evalúan de acuerdo con la siguiente tabla:

OR	T	F	ni
T	T	T	T
F	T	F	ni
ni	T	ni	ni

AND	T	F	ni
T	T	F	ni
F	F	F	F
ni	ni	F	ni

	NOT
T	F
F	T
ni	ni

TABLA III. Tablas de Lógica trivaluada.

La cota inferior $||Q||_*$ bajo la ***interpretación ni*** (interpretación no hay información) es evaluada por medio de la selección de aquellas n-eadas que evalúan a verdadero, es decir, las n-eadas que evalúan a falso o a **ni**, son descartadas.

La Lógica trivaluada y el método de evaluación de peticiones descritos, son equivalentes a la estrategia de trivaluación de **Codd** como se mostró, esta estrategia no produce la cota inferior correcta para la interpretación desconocido. Para las peticiones que corresponden a tautologías, afortunadamente la interpretación **ni** evita el problema.

Si se toma la siguiente petición:

Q_A: **range de** e is EMP.

retrieve (e.NOMBRE, e.E#)

where (e.SEXO="F" **and** e.TEL# > 2634000)

and (e.TEL# <= 2634000)

Dado que el valor nulo puede contener cualquier valor, la estrategia correcta para decidir que n-eadas satisfacen una expresión **where** consiste en sustituir cada nulo en todas las n-eadas, de manera tal que **no se violen las restricciones de integridad del esquema**, si para cualquier sustitución posible la cláusula **where** evalúa a verdadero, entonces debe ser incluida cuando se construya $||Q||_*$, en cualquier otro caso debe descartarse.

Al considerar la petición Q_A y la segunda n-eada de la **tabla II**:

(4335,LOPEZ,F,2235,-)

Si el valor nulo '-' se interpreta como un TEL# existente pero desconocido, entonces, es claro que para cualquier número que sustituya a '-' la cláusula **where** de Q_A evalúa a verdadero y la n-eada debe se incluida en $||Q_A||_*$.

Por lo tanto, bajo la interpretación desconocido, la n-eada debe ser considerada en $||Q_A||_*$ bajo la interpretación ni, sin embargo, se suplen los valores desconocidos o inexistentes.

Se asume que un valor no existente no satisface cualquier expresión relacional, es decir, alguna que involucre un operador de comparación, por ejemplo, un número telefónico que no existe, ni es más grande, más pequeño o igual que 2634000.

Entonces las n-eadas de EMP que tengan TEL# nulo, no pueden ser incluidas en la cota inferior $||Q_A||_*$. Por lo tanto, la interpretación **ni** evita la necesidad de determinar tautologías en las peticiones, problema que acomete la interpretación desconocido y significa desde el punto de vista práctico una ventaja importante.

Se puede definir la operación de selección de acuerdo con la interpretación de peticiones que se ha discutido. La operación de selección viene en dos sentidos:

$$R^0[A\ OPE\ B] \text{ y } R^0[A\ OPE\ k]$$

En donde:

A y **B** son atributos de **U** y del mismo dominio.

k es una constante del **DOM**(A) no nulo y

OPE es un operador relacional como (>,=,etc).

La definición de estas operaciones es:

(5.1)

$$R[A\ OPE\ B] = \{r/r \in R \text{ es A-total y B-total y } r[A]\ OPE\ r[B]\}^0.$$

(5.2)

$$R[A\ OPE\ k] = \{r/r \in R \text{ es A-total y } r[A]\ OPE\ k\}^0.$$

El producto cartesiano de dos relaciones R^0_1 y R^0_2 se define como:

(5.3)

$$R_1 \times R_2 = \{r_1 \circ r_2 / r_1 \in R_1 \text{ y } r_2 \in R_2 \text{ no son nulos}\}^0.$$

En el caso de las relaciones totales, las diversas operaciones de unión pueden definirse como selecciones sobre el producto cartesiano:

(5.4)

$$R^0_1\ [A\ OPE\ B]\ R^0_2 = (R^0_1 \times R^0_2)\ [A\ OPE\ B].$$

En el caso de la ***equiuniones***, no es necesario repetir la columnas unidas y se orienta a la definición de la unión sobre X de R_1 y R_2, como:

$$R^0_1 <.X> R^0_2 = \{r_1 \circ r_2 / r_1 \in R_1, r_2 \in R_2 \text{ son X-total}\}^0$$

Cuando ambos operandos de la selección o de la unión están en su forma mínima, entonces, los resultados calculados de acuerdo a (5.1) a (5.4) están en forma mínima también.

La proyección de una relación R^O sobre un conjunto de atributos X, denotada como **R[X]** se define como:

$$R[X] = \{r[X] / r \in R\}^O$$

El uso de valores nulos permite la definición de nueva información preservando las uniones, estas son llamadas ***uniones extendidas*** o ***uniones externas***.

La ***unión-asociación*** sobre X de R^O_1 y R^O_2 se denota como:

$$R^O_1 <*X> R^O_2 = R^O_1 <.X> R^O_2 \text{ unión } R^O_1 \text{ unión } R^O_2.$$

Esto es, la unión-asociación contiene aquellas n-eadas de las relaciones unidas que no pertenecen a la unión.

Se nota que el concepto de unión natural no encuentra una extensión obvia en esta estructura, el hecho que las relaciones (x-relaciones) no son explícitamente asociadas a un conjunto de atributos, representa una fuente seria de dificultades. Las equiuniones y uniones-asociaciones son las candidatas para ser la base de una extensión a la unión natural, sin embargo no contienen todas las propiedades deseables.

Como se observa en las últimas definiciones, se ha utilizado el operador ε en lugar de E, las inconsistencia es aparente, una vez que la sustitución en todas las fórmulas genera relaciones que son equivalentes en información inteligente a las originales, es fácil ver que las x-relaciones tienen la propiedad de igualdad de sustitución con respecto a los operadores.

C.5. CUANTIFICADORES UNIVERSALES Y NEGACION.

La operación de división no necesita ser considerada para obtener un álgebra relacional completa, ya que esta es derivable del producto, diferencia y la proyección cartesiana, sin embargo, se requiere de su consideración explícita una vez que es el puente para el tratamiento correcto de los cuantificadores universales en un mundo de información incompleta.

Sean R^O y S^O x-relaciones y R_y denota el conjunto de n-eadas Y-total de R, entonces, el cociente de R^O dividido por S^O se define como:

(6.1)

$$R^O </Y> S^O = R^O[Y] - (R^O[Y] \times S^O - R^O_y)[Y]$$

Para las definiciones totales esto reduce la definición usual de la división. De esta definición, se obtiene que aquellas n-eadas que no son Y-total no contribuyen al cociente y por lo tanto se puede escribir:

(6.2)

$$R^0 \text{ </Y> } S^0 = R^0{}_y[Y] - (R^0{}_y[Y] \text{ X } S^0 - R^0{}_y)[Y]$$

El único caso de interés práctico es cuando los alcances de $R^0[Y]$ y S^0 son ajenos, en dicho caso, la siguiente definición equivalente de división se puede obtener de (6.2)

(6.3)

$$R^0 \text{ </Y> } S^0 = \{y/y \text{ es Y-total y para todo } z \in S^0, y_{yz} \in R^0\}^0$$

Una tercera caracterización equivalente de división, puede derivarse de (6.3) por medio de $Z^0{}_r(Y)$ una z-imagen de un cierto Y-valor y, bajo R^0.

(6.4)

$$Z^0{}_r(Y) = \{z/ \text{ existe } r \in R^0: r[Y] = y_y\, r[Z] = z\}^0$$

entonces se obtiene que:

$$R^0 \text{ </Y> } S^0 = \{y/y \text{ es Y-total y } S^0 <= Z^0{}_r(Y)\}^0$$

Es decir, la operación definida constituye una extensión natural de la operación de división para relaciones totales, para entender mejor las propiedades de esta operación, se considera la relación **partes-proveedores**, a fin de facilitar una comparación más sencilla con el enfoque de Codd, cabe mencionar que no se han eliminado las n-eadas menos informativas.

PS	(S#,	P#)
	s_1	p_1
	s_1	p_2
	s_1	-
	s_2	p_1
	s_2	-
	s_3	-
	s_4	p_4

Considérese:

Q: Encuentre cada proveedor que surte todas las partes que surte el proveedor s_2.

La respuesta de esta petición puede ser calculada como sigue:

(6.7)

$$A^0 = PS^0 \text{ </S#> } P^0{}_{s2}$$

Donde P_{s2} denota la P# imagen de s_2 construida por una selección seguida de una proyección como sigue:

(6.8)

$$P^0{}_{s2} = PS^0\,[S\# = s_2]\,[P\#]$$

Se pueden comparar los resultados bajo la definición de división previa con los resultados obtenidos bajo la versión verdadero de **Codd**. La aplicación de (6.8) genera el siguiente resultado bajo la versión verdadera de selección:

(6.9)

$$P_{S2}=\{p_1,-\}$$

La versión quizá regresa el conjunto vacío.

La definición establecida produce el resultado correspondiente P^{O}_{S2}, sin embargo, la evaluación verdadera de **Codd** a (6.7) regresa:

$$A_1 = O^{O} \text{ (es decir, no hay proveedor)}$$

La evaluación quizá de **Codd** produce $A_2 = \{S_1, S_2, S_3\}$.

Por el contrario, usando la definición de división se obtiene:

$$A^{O}_3=\{S_1,S_2\}^{O}$$

Entonces, la evaluación verdadera de **Codd**, plantea la siguiente reformulación de Q:

Q_1: Encontrar todo proveedor quien de ***forma segura*** provee todas las partes que ***pueden ser*** surtidas por s_2.

Por su parte, la versión quizá de **Codd** corresponde a la siguiente reformulación de Q:

Q_2: Encontrar todo proveedor quien ***puede*** surtir todas las partes que en ***forma segura*** surte s_2.

Por último, la evaluación propuesta en este artículo corresponde a la siguiente versión:

Q_3: Encontrar todo proveedor quien ***de manera segura*** surte todas las partes que son proveidas en ***forma segura*** por s_2.

Estos ejemplos ponen en juego una consecuencia abierta e importante que considera el significado del cuantificador universal, y el proceso de formación de conjuntos en presencia de valores nulos.

El concenso unánime de los investigadores en este tópico, es que las peticiones como:

"Encontrar todos los empleados que ganan más de 20"

"Encontrar cada proveedor que surte partes rojas"

son ambiguas cuando tratan con información incompleta, se debe acompañar a las palabras todos y cada por cuantificadores como; "***de forma segura***" y de "***manera posible***" o por una especificación como; "***con más de 50% de probabilidad***".

En las peticiones como Q y en general en las que se involucran divisiones y cuantificadores universales, el proceso de formación de conjuntos especificados por las palabras "todos" y "cada" ocurren más de una vez, se ha elegido ser consistente e interpretar todas las ocurrencias de las palabras "todos" y "cada" en el sentido "de

forma segura", esta política consistente es simple de entender para el usuario y fácil de soportar para el sistema, una vez que se elimina el problema de computar cotas superiores.

También, se evita la siguiente paradoja, la cual acomete el tratamiento de la división de **Codd** y los cuantificadores universales:

Puesto que $A_1(S\#)=O^0$ uno debe concluir que "De manera segura, s_2 no surte todas las partes que s_2 surte".

El operador diferencia también implica una cuantificación universal y por ejemplo, una petición como:

Q_4: Encontrar todas las partes surtida por s_1 pero no por s_2.

Puede evaluarse como:

$$R^0{}_4=PS^0[S\#=s_1][P\#] - PS^0[S\#=s_2][P\#]$$

Claramente, el resultado es $R^0{}_4=\{p_2\}^0$, esto corresponde al conjunto de partes que son suministradas de *forma segura* por s_1, pero no estén en el conjunto de las que suministra de *forma segura* s_2.

C.6. GENERALIZACION DEL MODELO RELACIONAL PARA EL MANEJO DE NULOS.

En el principio de la década de los 70, **Codd** estableció los fundamentos de la teoría relacional de bases de datos, su principal contribución fue la introducción del tipo de datos ***relación***, con un conjunto completo de operadores relacionales (álgebra relacional), a fin de modelar bases de datos, peticiones y operaciones de actualización.

La noción de **Codd** de completez, estaba basada en la equivalencia entre el poder expresivo del álgebra relacional y el cálculo relacional.

Un álgebra relacional completa consiste de las siguientes operaciones:

- **unión**
- **diferencia**
- **selección.**
- **producto cartesiano y**
- **proyección.**

El cálculo relacional es una versión generalizada del cálculo de predicados, el cual se encuentra en muchos de los lenguajes de manipulación de datos relacionales.

La actualización de las bases de datos, también encuentra una definición precisa en términos del álgebra relacional. El resultado de agregar un conjunto a una relación, es definido como la unión del conjunto con la relación, de igual manera, la eliminación está definida como la diferencia de conjuntos, una modificación se puede ver como una eliminación seguida de una agregación.

En las secciones anteriores, se han extendido las relaciones tradicionales, a fin de modelar información incompleta por medio del uso de valores nulos.

La idea de esta sección, es probar que la extensión es correcta y completa, para tal propósito se recurre al problema análogo de extender los números naturales a los números reales, un paso de esta generalización es la definición de los números racionales a partir de los enteros.

Los números racionales, se definen como clases de equivalencia de pares de enteros a_1/b_1, siendo equivalentes al par a_2/b_2 si y solo si $a_1b_2=a_2b_1$. Se dice que **Z** denota el conjunto de los enteros y **K** el conjunto de los racionales.

Para probar correctez, solo es necesario mostrar que a un subconjunto $K_2 \subseteq K$, existe una correspondencia uno a uno entre **Z** y K_2, la cual preserva las operaciones en **Z**, es decir, preserva la adición, sustracción, multiplicación y división, al mismo tiempo que preserva el orden, lo que hace completamente transparente si se opera sobre **Z** o sobre K_2, esto asegurando la correctez de la extensión.

La importancia de la generalización de los números, parte del hecho que completan el sistema numérico con respecto a los cuatro operadores numéricos. En realidad, mientras los números racionales tienen la propiedad de cerradura con respecto a los cuatro operadores, los enteros no la tienen con respecto a la división.

En una sección anterior, se ha definido una x-relación R^O como una clase de equivalencia bajo ^{P} la cual contiene a R como un elemento, se dice que R(W) es una relación tradicional sin nulos, entonces, $R^O(W)$ es una x-relación total con alcance W, más aún, dos relaciones de **Codd** distintas mapéan a dos x-relaciones totales distintas, entonces existe una correspondencia uno a uno entre las relaciones de **Codd** y las x-relaciones. Esta correspondencia preserva todos los operadores del álgebra relacional:

- **unión.**
- **diferencia**
- **producto cartesiano.**
- **selección y**
- **proyección.**

A fin de verificar esto, solo basta recordar las condiciones bajo las cuales los operadores relacionales son definidos para las relaciones de **Codd**, y aplicar las definiciones (4.1), (4.3), (5.1), (5.2) y (5.5), así como, la definición de producto cartesiano para concluir que:

(1) Si R_1 y R_2 son relaciones de **Codd** compatibles-bajo-unión y si (R_1 unión R_2)=R_3, entonces (R^O_1 unión R^O_2) = R^O_3, si $R_1 - R_2=R_4$, entonces $R^O_1 - R^O_2 = R^O_4$ y si $R_2 \subseteq R_1$, entonces $R^O_2 \subseteq R^O_1$.

(2) Si R_1 y R_2 son relaciones de **Codd** y $R_1 \times R_2=R_3$, entonces $R^O_1 \times R^O_2=R^O_3$.

(3) Si A es un atributo de una relación de **Codd** R y si R[A OPE k]=R_1, entonces R^O[A OPE k]=R^O_1.

(4) Si A y B son atributos de una relación de **Codd** R y si R[A OPE B]=R_1, entonces R^O[A OPE B]=R^O_1.

(5) Si W es un subconjunto de atributos de una relación R de **Codd** y si $R[W]=R_1$, entonces $R^{\Omega}[W]=R^{\Omega}{}_1$.

En conclusión, se puede operar en el ámbito de las denominadas x-relaciones totales, en lugar de operar sobre las relaciones de **Codd**, para todas las situaciones en que se involucran las operaciones definidas, se puede además verificar que las relaciones (x-relaciones) tienen la propiedad de cerradura respecto a la unión, diferencia, producto cartesiano, selección y proyección, por lo tanto la extensión de las relaciones de **Codd** a x-relaciones, completa el tipo de datos relación con respecto al álgebra relacional.

A partir de las razones mencionadas, las x-relaciones son de gran interés para los sistemas manejadores de bases de datos, es más, proveen de un ejemplo interesante de generalización en la teoría de conjuntos, a diferencia de los conjuntos, no constituyen un álgebra booleana, en lugar de ello establecen una ***latíz seudo-complementada distributiva*** **[BIRK 67]**, en donde el seudo-complemento de R^{Ω}, denotado como R^{Ω^*}, está definido como:

$$R^{\Omega^*}=TOP_U - R^{\Omega}$$

Esto es, R^{Ω^*} es la más pequeña x-relación que cuando es unida con R^{Ω} genera TOP_U.

Las latices seudo-complementadas y distributivas se conocen como ***latices de Brouwer*** **[FRIK 62]**, después que **Brouwer** y **Heyting** caracterizaron una generalización importante del álgebra booleana en 1962.

C.7. COMENTARIOS FINALES.

Los sistemas de bases de datos se diseñan para almacenar grandes cantidades de información o conocimiento del mundo real y para contestar preguntas en relación a esta información. Sin embargo, a diferencia de los sistemas de la Inteligencia Artificial basados en el conocimiento, los sistemas de bases de datos no intentan, por el momento, preservar el tesoro sin fronteras del conocimiento del mundo real, el cual está estructurado en las mentes humanas y es comunicado por medio de los matices del lenguaje natural. Las bases de datos pueden aproximar este tesoro complejo y sin fronteras solo de manera limitada e imperfecta, una vez entendido esto, los usuarios de bases de datos tienen que aceptar los límites razonables que existen, el alcance y sofisticación con el cual sus bases de datos modelan el mundo real, a cambio, dichos usuarios demandan sistemas que realicen de manera correcta y eficiente las tareas encomendadas, al mismo tiempo que sean simples de entender y utilizar.

La solución propuesta en este estudio al problema de la manipulación de los valores nulos, asegura la simplicidad lógica y la correctez combinadas con la eficiencia computacional, en particular, se extienden los fundamentos teóricos del modelo relacional y se garantiza la eficiente evaluación de la peticiones por medio de la bien conocida correspondencia entre el cálculo y el álgebra relacionales.

Lo anterior, se ha sustentado en el uso de la interpretación **ni** de los nulos, la cual es capaz, aunque de manera imperfecta de modelar y retener el conocimiento incompleto del mundo real.

La interpretación **ni** evita los problemas computacionales serios que ocurren cuando una intepretación "*más adecuada*" se utiliza, como es el caso de la interpretación desconocido.

Se ha probado, la solidez teórica del enfoque **ni** de manera que es posible la definición de equivalencia en información inteligente sobre relaciones con conjuntos arbitrarios de atributos.

Se ha introducido el concepto de relaciones como clases de relaciones equivalentes en información inteligente y se generalizaron los operadores del álgebra relacional para aplicarse a las relaciones extendidas.

Por último, se ha mostrado la correctez y completez de la generalización, por lo tanto, el enfoque es teóricamente sólido y práctico, al mismo tiempo que evita muchas de las complejidades e inconsistencias presentadas por otros enfoques.

Sin embargo, existen aún problemas que requieren de una mayor investigación, por ejemplo, un área de interés práctico es el derivar una taxonomía completa de los valores nulos y sus relativos cambios. En particular, se desea conocer que extensión fundamentada en la interpretación **ni** es más adecuada en las aplicaciones del mundo real, y estudiar aquellas que requieran de interpretaciones más informativas.

APENDICE D. *UN MODELO DE DATOS ORIENTADO A OBJETOS PARA EL MODELADO, IMPLEMENTACION Y ACCESO A UNA BASE DE DATOS.*

Ha sido definido un modelo de datos orientado a objetos (**OODM**) para el diseño y acceso de una base de datos. El **OODM** trabaja con tres tipos de relaciones:

. Agregación.
. Generalización y
. Particularización.

Ademas provee cuatro tipos de operaciones con los datos:

. Definición de un esquema.
. Creación de una base de datos.
. Recuperación de objetos y
. Expansión de objetos.

La operación de expansión de los objetos se utiliza en el contexto de la recuperación previa de objetos o tipos de objetos y permite la navegación automática a través de la base de datos.

El modelo de datos soporta una interface estructurada basada en menús la cual permite al usuario definir el esquema y la recuperación de la base de datos sin tener que poseer ningún conocimiento del modelo de datos orientado a objetos. Una implementación interesante de esta interface es que la información del menú es modelada como un conjunto de objetos interrelacionados que se alojan en la base de datos. Por lo tanto, los menús se crean, se acceden y se actualizan en la misma forma que los datos. Más aún, un conjunto básico de menús es generado del esquema de la base de datos y puede hacerse disponible al usuario con el propósito de contestar peticiones sencillas. Un prototipo del modelo de datos orientado a objetos se ha implementado y se utiliza para modelar una aplicación para la descripción y clasificación de orquídeas.

D.1. INTRODUCCION.

Esta parte se presenta un modelo de datos orientado a objetos (**OODM**) para el diseño, implementación y uso de una base de datos. El propósito del modelo de datos es una dualidad:

. Proveer al diseñador con una herramienta expresiva para representar la información.

. Auxiliar al usuario para acceder la información almacenada en la base de datos.

De lo anterior, se desprende la orientación que rige la definición del modelo de datos:

. **Habilidad expresiva.**
. **Comprensión.**
. **Simplicidad y**
. **Accesibilidad.**

En general el modelo de datos consiste de dos partes:

. **El esquema conceptual y**
. **Las operaciones sobre los datos.**

El esquema conceptual representa al mundo real como objetos y relaciones entre esos objetos. Se identifican tres tipos de relaciones :

. **Agregación.**
. **Generalización y**
. **Particularización.**

Las operaciones con los datos incluyen las necesarias para la definición del esquema, la creación de la base de datos y la manipulación de los objetos.

El concepto de agregación y generalización fueron introducidos por **Smith** y **Smith [SMITH 77], [SMITH 78].** La ***generalización*** se refiere a una abstracción que permite a una clase de objetos individuales ser considerada generalmente como un objeto que se nombra de manera sencilla y la ***agregación*** se refiere a una abstracción en la cual la relación entre objetos es considerada como un objeto de nivel más alto. Estas dos abstracciones de datos han sido ampliamente aceptadas en el diseño lógico de bases de datos. En el modelo que se trata se ha considerado otro tipo de relación, la ***particularización***, la cual es muy importante cuando un objeto tiene una lista de propiedades que de forma natural pueden ser agrupadas en otro objeto.

A fin de incrementar su accesibilidad y alcanzar la simplicidad, el modelo de datos orientado a objetos provee de una interface basada en menús. Esta interface puede ser vista como un modelo externo del modelo orientado a objetos, el cual auxilia al usuario a acceder la base de datos y definir un esquema. Una característica importante de esta interface es que información del menú es almacenada parcialmente en un menú de la base de datos que puede ser manipulado en la misma forma que los objetos y generado automáticamente de la base de datos y del esquema de la base de datos durante el tiempo de la petición. Por lo tanto, una interface sencilla y sin cambios puede soportar menús variables.

Un prototipo del modelo de datos orientado a objetos se ha implementado utilizando una **POP-11** bajo sistema operativo **UNIX**. Este prototipo actualmente provee cuatro operaciones de datos:

. **Definición de esquema.**
. **Creación de la base de datos.**
. **Recuperación de datos.**
. **Expansión de datos.**

La interface basada en menús, la cual se encuentra en desarrollo, soporta cinco operaciones:

. **Definición de esquemas.**
. **Creación de la base de datos.**
. **Creación de vistas.**
. **Acceso a la base de datos entera.**
. **Recuperación de información.**

En este estudio, se utiliza para ilustrar las características del modelo de datos orientado a objetos una base de datos de prueba que representa el **dominio de la descripción y clasificación de las orquídeas**, el cual muestra la factibilidad de uso del prototipo.

En general, el dominio ejemplo de la clasificación y descripción de la orquídeas incluye **especies, variación y forma** de las mismas. Cada orquídea es identificada por un nombre y su clasificación dentro de cierto tipo de especie, la variación o forma es de acuerdo a sus **caracteres de campo**, los caracteres de campo de cada orquídea son caracterizados por sus **hojas, fragancia y periodo de floración**, etc. Las peticiones típicas en este ámbito pueden ser del tipo:

. Recuperar todas o ciertas especies de orquídeas.

. Buscar por todos o ciertos caracteres de campo para determinada especie.

. Encontrar todas las variaciones de una especie.

. De acuerdo con los caracteres de campo de una orquídea, encontrar sus especies.

D.2. ANTECEDENTES.

Dadas las limitaciones para la expresión semántica en los modelos tradicionales de datos. La investigación se ha orientado a establecer modelos semánticos de datos. Representantes típicos de los modelos semánticos de datos son:

. **El modelo entidad-relación (E-R) [CHEN 76].**
. **El modelo relacional de Tasmania (RM/T) [CODD79].**
. **El modelo semántico de datos (SDM) [MCLEOD 76], [HAMMER 78].**
. **El modelo evento de datos [KING 82], [KING 85].**

Un modelo semántico de datos está orientado para proveer **especificación entendible** al usuario y capturar una porción substancial del **significado de los datos**. Las diferencias principales entre los modelos de datos son, diferencias en el tipo de relaciones que pueden ser representadas y en las restricciones que se imponen a ellas. Por ejemplo, el modelo **entidad-relación** representa la información como entidades y relaciones entre entidades. Sin embargo, en el modelo mencionado los tipos de relaciones no se distinguen. En el **modelo RM/T**, un tipo se representa como una entidad relación la cual contiene una columna simple que especifica el "subordinado" para cada instancia del tipo. Los atributos son representados por propiedades de la relación las cuales asocian valores subordinados con propiedades de los valores.

Tres tipos de relaciones se identifican en el **modelo RM/T**:

. **Asociación (relación varios a varios).**
. **Designación (relación varios a uno).**
. **Generalización (jerarquías tipo/subtipo).**

El **modelo SDM** estructura una base de datos a través de clases de objetos. Establece relaciones del tipo "**instancia**", "**atributo**" y "**subtipo**". Una de las características especiales del modelo SDM es que cuenta con una interface para el usuario. El **modelo EVENTO** representa una base de datos como objetos y eventos de la aplicación (transacciones), los cuales son clasificados en tipos de objetos. Este modelo soporta dos tipos de relaciones:

. **Generalización (relación tipo y subtipo).**
. **Agregación (formada vía atributos).**

Las ideas de esos modelos han orientado el desarrollo del modelo de datos orientado a objetos que se presenta en esta discusión.

D.3. ESQUEMA CONCEPTUAL DEL MODELO DE DATOS ORIENTADO A OBJETOS.

Los constructores básicos del esquema conceptual del **OODM** son:

. **Objetos.**
. **Tipos de objeto.**
. **Atributos.**
. **Relaciones.**

OBJETOS, TIPOS DE OBJETOS Y ATRIBUTOS.

Las definiciones de objeto, tipo de objeto y atributo han sido dadas por Chen [**CHEN 76**].

Un ***objeto*** es un elemento o concepto del mundo real que puede ser identificado claramente. Una persona es identificada por su nombre (David Manzano) y un curso por su título (Diseño de bases de datos), los anteriores son ejemplos de objetos. Un objeto puede ser caracterizado por sus ***propiedades***, las cuales tienen un identificador, por ejemplo, David Manzano vive en Mina No. 62 o Diseño de base de datos tiene un número de módulo D24. En esos casos las cadenas "David Manzano" y "Diseño de bases de datos" son identificadores de los objetos y las cadenas "Mina No. 62" y "D24" son propiedades de los objetos identificados por David Manzano y Diseño de bases de datos.

Es interesante que mientras "Mina No. 62" ha sido descrito como una propiedad del objeto "David Manzano", a su vez es el identificador de una calle. Por lo tanto, todo modelo semántico debe ser capaz de cubrir este tipo de papeles duales.

Los objetos se clasifican en diferentes *tipos de objetos* en términos de sus propiedades: objetos con las misma propiedades, es decir, el mismo significado, son agrupadas en un tipo de objeto el cual es identificado por un *nombre de tipo*.

La información acerca de un objeto, está expresada por un conjunto de parejas *atributo-valor*, David Manzano y D24 son *valores*. Los valores son clasificados en diferente *tipos de valor*, como son nombre y número.

Un *atributo* puede definirse formalmente como una función que tiene como dominio un tipo de objeto y como codominio un tipo de valor o un producto cartesiano de tipos de valor.

En el **OODM** hay tres clases de atributos:

. **Atributo valor.**
. **Atributo grupo.**
. **Atributo agregado.**

Los *atributos valor*, son atributos propios cuyos valores son arrastrados de los valores tipos. Los atributos **nombre_chino** y **habitat_distribución** de orquídeas, son ejemplos de atributos valor.

Un *atributo grupo* puede verse como una representación de un grupo de atributos, por ejemplo, el atributo **caracteres_campo** en el tipo de objetos de **especies**, es un atributo grupo,para representar un grupo de atributos como el siguiente:

[número_de_flores, diámetro_de_la_flor, periódo_floración, hoja, fragancia,...,fruto].

El valor de un atributo grupo puede ser una descripción resumida o un apuntador a un conjunto de valores de un grupo de atributos, dependiendo del contexto de recuperación y de la extensión de la base de datos.

Un tipo de objeto, puede ser referido a otro tipo de objeto a través de *atributos agregados*. El valor de un atributo agregado puede ser un identificador de un objeto o un conjunto de identificadores de objetos de algún otro tipo de objetos, dependiendo del grado de la relación.

Un *atributo* podría definirse como un identificador de un objeto que puede ser monovaluado o multivaluado, sin embargo, no debe ser nulo.

Un identificador de un objeto debe ser no nulo, un atributo monovaluado define un mapeo uno a uno de un tipo de objeto a un tipo de valor. Un atributo multivaluado define un mapeo varios a varios (incluyendo uno a varios) de un tipo de objeto a un tipo de valor.

D.4. RELACIONES: AGREGACION, GENERALIZACION Y PARTICULARIZACION.

Los objetos pueden relacionarse a otro objeto, por medio de relaciones binarias. Un tipo de objeto puede ser considerado como un tipo de relación, la cual agrupa los objetos. Este tipo de relación es denominada relación "instancia de" y es muy esencial, ya que existe herencia.

Las relaciones ha ser discutidas en este ámbito, pertenecen a otra categoría, ellas relacionan objetos de diferentes tipos de objetos u objetos del mismo tipo de objeto pero en diferentes roles.

Agregación. Define relaciones uno a varios o varios a varios entre tipos de objetos. En el **OODM**, la agregación es una relación bidireccional que permite a dos tipos de objetos referirse entre sí. La "liga" entre dos tipos de objetos, es un par de atributos agregados (cada uno asociado con un tipo de objeto, por ejemplo, si se desea modelar estudiantes y cursos, un atributo agregado de curso en el tipo de objeto estudiante, podría usarse para referirse al tipo de objetos curso, mientras un atributo agregado estudiante en el tipo de objeto curso, podría referirse al tipo de objeto estudiante. Para el propósito de recuperación, los objetos de un tipo de objetos pueden ser rastreados de un objeto de otro tipo de objeto.

Generalización. Define relaciones tipo-subtipo entre tipos de objetos. Cada subtipo es una especialización de su tipo correspondiente. En el **OODM**, una generalización se obtiene por:

. Agrupamiento de objetos de diferentes tipos de objetos para formar otro tipos de objetos.

. Definición de subtipos de un tipo de objetos, utilizando predicados que restrinjan los valores de los atributos.

Particularización. Define relaciones uno a uno entre tipos de objetos, es una relación unidireccional que permite a un objeto (objeto padre) de un tipo de objeto (tipo padre) referirse a un objeto (objeto hijo) de otro tipo (tipo hijo) a través de un atributo grupo en el tipo padre, pero no es válida la viceversa. Un objeto hijo es un conjunto de valores de un grupo de atributos. Este grupo de atributos puede representarse por un atributo grupo en el tipo padre. Por lo tanto, un objeto hijo representa propiedades de su objeto padre y su identificador es el mismo que el del objeto padre.

La existencia de un objeto hijo, se determina por el valor del atributo grupo. Si el valor de un atributo grupo corresponde a "ver abajo", existe un objeto hijo, en cualquier otro caso el objeto hijo no existe. En el primer caso el valor de un atributo grupo puede incluir una descripción resumida y entonces indicar "ver abajo" permitiendo al usuario expander la descripción con los detalles contenidos en el objeto hijo. Mientras en el último caso el valor de un atributo grupo incluirá simplemente la descripción y los detalles no pueden ser captados. Entonces, los objetos hijo son objetos dependientes, debido a que su existencia depende de sus objetos padres.

D.5. EJEMPLO: DEFINICIONES DE LOS TIPOS DE OBJETOS EN UNA BASE DE DATOS DE ORQUIDEAS.

La sintaxis para definir tipos de objetos en el dominio de la descripción y clasificación se muestra en el **cuadro 1**. La descripción y clasificación en este ejemplo, está basada en el trabajo realizado por **Pen** y **Liu** [PEN 84]. En este contexto un ejemplo de tipo de objeto es **especies**, con atributos valor **nombre_chino**, **nombre_científico**, **nombre_inglés**, **otros_nombres** y **habitat_distribución**, atributo grupo es **caracteres_campo** y los atributos agregados son **variación** y **forma**. En el tipo de objetos **especies**, los atributos valor **nombre_chino** sirven como un identificador de objeto, el atributo grupo **caracteres_campo**, tiene un grupo de atributos que son asociados con un tipo de objetos, el atributo agregado **variación** sirve como un apuntador al tipo de objetos **variación**.

Puede notarse por ejemplo, que el atributo grupo "**caracteres_campo**" tiene un nombre idéntico al tipo de objetos "**caracteres_campo**" y el atributo agregado "**variación**" tiene un nombre idéntico al tipo de objeto "**variación**". El propósito de hacer esto es proveer una vista multinivel de los objetos y los tipos de objetos. El punto es que no existen fronteras rígidas entre un tipo y un objeto, esta es una de las distinciones entre los modelos semánticos de datos y los modelos tradicionales de datos.

Un ejemplo de agregación es la relación entre los tipos de objetos, **especies** y **variación**, estos dos tipos pueden referirse uno a otro por medio de un par de atributos agregados, "**variación**" en **especies** y "**especies**" en **variación**. El valor del atributo agregado "**variación**" es el identificador del objeto **variación**, el valor del atributo agregado **especies** es el identificador del objeto **especies**. Por lo tanto, los objetos de **variación** pueden ser rastreados de un objeto **especies** y viceversa. La agregación entre **especies** y **variación**, es una relación uno a varios, una **especie** puede tener más de una **variación** y una **variación** pertenece a una **especie**. La relación entre **especies** y **forma** y la relación entre **variación** y **forma**, son también relaciones de agregación.

Es interesante notar que en el dominio que se está manejando, se ilustra una diferencia fundamental entre "***la red semántica***" utilizada para representación del conocimiento y el **modelo semántico de datos**. Si se deseara almacenar la información en una red semántica, cada **variación** sería un tipo de **especie**. Aunque esto pudiera parecer natural, impone restricciones sobre las vías de acceso a los datos, situación indeseable en el contexto de bases de datos.

La relación de ***particularización*** se introduce para capturar las semánticas entre un tipo de objeto y sus atributos directos e indirectos.

Por ejemplo, el atributo grupo **caracteres_campo** puede caracterizarse por un grupo de atributos como sigue:

[número_de_flores, diámetro_de_la_flor, periódo_floración, hoja, fragancia, ... ,fruto].

Como este grupo de atributos caracteriza **caracteres_campo**, ellos no pueden ser agregados directamente a otros atributos de **especies** sin la pérdida de semántica, por otro lado, este grupo de atributos tiene valores que son propiedades de especies. El significado de la relación entre **especies** y su atributo (directo) **caracteres_campo** y sus atributos (indirectos) **número_de_flores, diámetro_de_la_flor**, etc. está representado por medio de la definición de otro tipo de objeto **caracteres_campo** el cual tiene un grupo de atributos y un identificador **nombre_chino** de especies. Un objeto de **especies** corresponde a lo más a un objeto en **caracteres_campo**.

La relación entre **especies** y **fruto**, la relación entre **especies** y **fragancia** y la relación entre **caracteres_campo** y **hoja** son todos ejemplos de particularización.

Un ejemplo de generalización, es la relación entre **especies, variación** y **forma**. Dado que las **especies, variación** y **forma** son especializaciones de orquídeas, ellos pueden ser agrupados en un tipo de objeto general, el **tipo orquídea**. Obviamente **especie, variación** y **forma** son especializaciones de **orquídea**, mientras **orquídea** es una generalización de **especies, variación** y **forma**.

D.6. OPERACIONES CON LOS DATOS.

Las operaciones con los datos para el **OODM** incluyen definición de esquema, creación de la base de datos, recuperación de datos, expansión y actualización. Estas operaciones han sido implementadas y su sintaxis se describe en las siguientes secciones.

D.7. DEFINICION DE ESQUEMA Y CREACION DE LA BASE DE DATOS.

DEFINICION DE ESQUEMA.

La operación para la definición de esquema es de la siguiente forma:

CREAOBJETO ([nombre_tipo_objeto,[definición atributo, definición de relación]]) --> nombre_tipo_objeto.

CREACION DE BASE DE DATOS.

La operación para la creación de la base de datos tiene la siguiente forma:

AGREGAOBJETO ([nombre_tipo_objeto, definición_objeto]).

D.8. RECUPERACION DE DATOS.

La variedad de las operaciones de recuperación de datos depende del esquema del modelo de datos. En el **OODM** se dota de dos operaciones rudimentarias para realizar la recuperación mencionada.

RECUPERACION, PROYECCION Y SELECCION DE UN TIPO DE OBJETO.

La sintaxis para la recuperación es:

RECUPERA (nombre_tipo_objeto, lista_atributos, lista_condiciones);

En donde lista_atributos tiene tres patrones posibles:

1) [todo] proyecta todos los atributos de un tipo de objetos.

2) [todo sin atr1 atr2...] proyecta todos los atributos con excepción de los atributos especificados atr1, atr2, etc.

3) [atr1 atr2 atr3 ...] proyecta solo los atributos especificados atr1, atr2, atr3, etc.

y la lista_condiciones puede tener las siguientes formas:

1) Estándar.

En este caso, la recuperación es de la forma:

RECUPERA (nombre_tipo_objeto, lista_atributos);

2) [atr1 COMP val1 UNION atr2 COMP val2.....].

En donde atr1, atr2, etc. representan atributos que pueden ser alguno de los siguientes:

PALABRA: (Por ejemplo, Xian_ye_chun_lan, yes, no);

NUMERO : (Por ejemplo, 60, -30, 12, 5);

LISTA : (Por ejemplo, [Encontrado en Dali], [lian_ban_lan, xian_ye_chun_lan]).

COMP puede ser uno de los siguientes:

= (igual a), <> (no igual a), > (mayor que), < (menor que), >= (mayor o igual que), <= (menor o igual que) e **INCLUYE**.

UNION puede ser alguna de:

AND (conjunción), **OR** (disyunción).

La unión **INCLUYE** se utiliza para hacer corresponder un valor suministrado con un elemento de una lista, por ejemplo:

RECUPERA (especies, [todo], [variación **INCLUYE** fen_hong_duo_xiang);

El ejemplo anterior recupera la especies que incluyen 'fen_hong_duo_xiang' entre sus variaciones.

A continuación se dan ejemplos de operaciones de recuperación:

RECUPERA (especies, [todo sin variación forma], [nombre_chino=chun_lan]);

RECUPERA (variación, [todo], [forma=jin_huang_duo_xiang, ya_huang_duo_xiang, huang_hua_duo_xiang]]);

El último ejemplo es equivalente a:

RECUPERA (variación, [todo], [forma INCLUYE ya_huang_duo_xiang]);

EXPANSION.

La operación de expansión toma la siguiente forma:

EXPANDER (nombre_tipo_objeto, lista_atributos, lista_condiciones);

En donde lista_atributos y lista_condiciones tienen las mismas formas a las discutidas en la operación de **RECUPERA**.

La operación **EXPANDER** está diseñada como un pariente de la operación **RECUPERA** y se utiliza para sustentar las relaciones de agregación y particularización. Las operaciones mencionadas se aplican conjuntamente, por ejemplo, después de recuperar un objeto o conjunto de objetos de un tipo de objeto, el usuario puede usar **EXPANDER** para presentar todos los detalles de un grupo de atributos en otro tipo de objetos o arrastrar objetos de otro tipo de objetos por medio de un atributo agregado. Para el primer caso, existe una relación de particularización entre dos tipos de objetos, el segundo caso indica que existe una relación de agregación entre dos tipos de objetos.

Como un ejemplo de la utilización de esta operación, considerense las siguientes peticiones:

1) Imprimir nombre_inglés, caracteres_campo, habitat_distribución de las especies Chun_lan;

2) Imprimir toda la información relacionada a las especies Chun_lan;

Para la primera petición se utiliza **RECUPERA**:

RECUPERA (especies, [nombre_inglés, caracteres_campo, habitat_distribución], [nombre_chino=chun_lan]);

La segunda petición incluirá requerir información relativa a caracteres_campo,hoja,fruto,variación y forma de Chun_lan. Se utiliza EXPANDER para la satisfacción de las peticiones.

EXPANDER (caracteres_campo,[todo]);

EXPANDER (hoja,[todo]);

EXPANDER (fruto,[todo]);

EXPANDER (variación,[todo]);

EXPANDER (variación, [todo], [nombre_chino=fen_hong_duo_xiang]);

EXPANDER (forma,[todo]);

Los ejemplos anteriores, muestran que **EXPANDER** es una operación inteligente, ya que recuerda que fue lo que realizó **RECUPERA** y navega entre el tipode objeto de manera automática. Al utilizar **EXPANDER**, el usuario no requiere de especificar conexión alguna entre tipos de objetos y por lo tanto, evita la "trampa de conexión" como fue identificada por **Codd [CODD 70]**. Aparte de la navegación entre tipos de objetos, **EXPANDER** puede navegar dentro del mismo tipo de objetos, en otras palabras, **EXPANDER** tiene habilidad recursiva (o reflexiva) la cual puede relacionar objetos del mismo tipo. Esta característica recursiva es muy útil, ya que hace la definición de esquemas de manera más natural y sencilla.

D.9. UNA INTERFACE BASADA EN MENUS PARA EL MODELO DE DATOS ORIENTADO A OBJETOS.

La interface es soportada por una base de datos de menús y un sistema manejador de menús y actualmente provee cinco operaciones para la definición de esquemas, captura de datos, generación de vistas y recuperación de información.

EL MENU, BASE DE DATOS DE MENUS Y ESPACIOS DE MENUS.

Desde el punto de vista del usuario, un menú es una pantalla de información que consiste de tres secciones:

- **Título del menú.**
- **Opciones del menú.**
- **Diálogo interactivo.**

El Título del menú provee de una descripción completa del propósito del menú. Las opciones del menú representan las funciones de interface. El diálogo interactivo aparece en cada menú para guiar al usuario en la navegación a traves de la red de menús. En la interface basada en menús del **OODM**, los menús están estructurados como una red, lo que permite que un menú sea alcanzable por más de una vía de acceso y puede ser repetido por movimientos hacia atras.

Existen tres tipos de opciones de menús:

- **Predefinidos.**
- **Definidos por el usuario.**
- **Generados por el sistema.**

Las opciones de menús predefinidos, proveen un conjunto primario de funciones las cuales son de interface e independientes del dominio, por ejemplo, en la interface las funciones primarias pueden incluir:

. **Recupera datos.**
. **Define esquema.**
. **Consulta información.**
. **Genera vista.**
. **Abandona menú.**

Las opciones predefinidas de los menús pueden aplicarse a cualquier dominio de aplicación, representado por el **OODM.**

Las opciones definidas por el usuario están relacionadas al dominio particular de la aplicación, a fin de contestar las peticiones del usuario. Cada opción definida por el usuario corresponde a la traducción de una petición del usuario a operaciones del **OODM.**

Estas dos primeras clases de opciones de menús están almacenadas en las bases de datos de menús, cada una de ellas está representada como un objeto con un identificador y una vía de acceso. Todos los objetos en la base de datos de menús están agrupados en un tipo de objeto denominado tipo menú. Una via de acceso puede estar relacionada a un conjunto de opciones de menú en la base de datos de menús o a un procedimiento que generará opciones de menú a partir del esquema de la base de datos y la base de datos, estas últimas opciones son denominadas opciones de menú generadas por el sistema, y son utilizadas para contestar peticiones muy simples.

Por lo tanto, un menú puede ser definido como un tipo menú instanciado e identificado por un nombre de menú. Todos los menús son generados automáticamente de la base de datos de menús, el esquema de la base de datos y la base de datos y estructurados como una red. La red de menús puede ser definida como un espacio de menús, a traves del cual el usuario puede navegar. Dentro del espacio de menús algunos espacios de submenús (o redes de submenús) pueden ser formados dinámicamente durante el tiempo de la petición para responder a diferentes peticiones.

EL SISTEMA MANEJADOR DE MENUS.

Los objetivos del sistema manejador de menús son:

. **Generar menús.**
. **Desplegar menús.**
. **Ejecutar peticiones.**

Para alcanzar esos objetivos el sistema manejador de menús debe tener las siguientes funciones:

. Controlar el movimiento de los menús.

. Generar la red de menús de forma dinámica y de acuerdo a la elección del usuario.

. Generar un conjunto básico de opciones de menú a partir del esquema y la base de datos.

. Acceder a otras opciones de menú a partir de la base de datos de los menús.

. Presentar menús, cada uno con títulos, un conjunto de opciones y un diálogo interactivo con el usuario.

. Traducir una petición del usuario a operaciones del **OODM**.

. Ejecutar las operaciones y desplegar los resultados.

D.10. IMPLEMENTACION DEL PROTOTIPO OODM.

El lenguaje de programación utilizado para construir el sistema prototipo **OODM**, es **POP-11** [BARRETT 85], [BURTON 86], un lenguaje práctico para inteligencia artificial, como lenguaje de programación de propósito general, ha sido utilizado en investigación de inteligencia artificial, diseño VLSI, gráficas, proceso de textos, diseño de compiladores, proceso de imagenes, diseño de sistemas expertos y desarrollo de programas interactivos. Este lenguaje puede también ser usado en el área de proceso de datos, ya que ofrece una amplia variedad de estructuras de datos como son listas, procedimientos y propiedades. Las listas son muy generales y son estructuras básicas muy útiles para almacenar objetos del mundo real y relaciones entre esos objetos. También provee un conjunto amplio de operaciones para manipulación de listas, incluyendo acceso a listas, construcción de listas y mapeo de patrones. El mapeo de patrones es una técnica poderosa de programación que puede ser utilizada para verificar la correspondencia de una lista con un patrón, por ejemplo, para la definición de un programa interactivo que responda dependiendo del patrón de palabras que se han tecleado es muy útil. Una propiedad de **POP-11** es una tabla de asociaciones de valores, la cual funciona como un tipo de memoria y almacena asociaciones de valores de acuerdo a su código disperso.

En el diseño que nos ocupa todas las operaciones de datos del **OODM** están definidas como procedimientos **POP-11** y soportadas por listas y operaciones de listas, la base de datos está almacenada como propiedades **POP-11**. Se hace uso de la pila **POP-11** del usuario y de variables globales para almacenar la información que permita la interacción de las operaciones y procedimientos, también se hace uso extensivo del mapeo de patrones sobre las listas para el análisis del lexico y la recuperación. Como un ejemplo, se analiza como trabaja la operación **RECUPERA**. La operación recupera toma tres argumentos y su formato es:

RECUPERA (nombre_tipo_objeto, lista_atributos, lista_condiciones);

En donde lista_atributos es una lista de atributos a ser proyectada la cual tiene tres patrones, [todo], [todo sin atr1 atr2...], [atr1 atr2 atr3 ...]. Donde un patrón es mapeado y será instanciado a la operación correspondiente.

Lista_condiciones es un conjunto de condiciones utilizado para seleccionar objetos. Al utilizar accesos a listas y construcción de listas, este conjunto de condiciones es descompuesto en cuatro sublistas:

. **Una lista de atributos.**
. **Una lista de valores.**
. **Una lista de comparaciones y**
. **Una lista de uniones.**

De acuerdo a esas cuatro listas, el sistema puede trabajar sobre la base de datos utilizando comparaciones aritméticas, conjunciones lógicas y disyunciones lógicas. La implementación de las otras operaciones se basa en la misma idea. El prototipo que nos ocupa es utilizado sobre bases experimentales para probar el **OODM**. El dominio de clasificación y descripción de orquídeas ha sido modelado utilizando este prototipo con resultados adecuados.

APENDICE E. *ORGANIZACION FISICA DE LOS DATOS.*

En esta parte se analiza una cuestión de importancia, que debe considerarse tanto para diseño, como para el proceso de llevar a cabo bases de datos, el tópico consiste en definir los esquemas de representación de datos en el ***nivel físico***.

El problema básico en la representación física de una base de datos, se trata bajo la panorámica de almacenar un ***archivo***, el cual se conforma de ***registros***, los cuales a su vez, tienen un formato idéntico.

Un ***formato de registro***, esta conformado de una lista de ***nombres de campos***, en donde cada campo tiene tamaño y tipo fijos. Un registro, entonces, se compone de un ***juego de valores para cada campo***.

También se define que las operaciones típicas que se desean realizar en un archivo son:

1. ***Inserción*** de un registro.

2. ***Eliminación*** de un registro.

3. ***Modificación*** de un registro.

4. ***Búsqueda*** de un registro con un cierto valor en determinado campo o con una cierta combinación de valores en una determinada combinación de campos.

La decisión del tipo de organización de almacenamiento de un archivo, depende principalmente de las características que se desea tengan las mencionadas operaciones, las cuales actuarán sobre el archivo.

E.1. CARACTERISTICAS DEL ALMACENAMIENTO.

Se establece la convención que al hablar de ***almacenamiento externo o secundario***, se trata de almacenar datos en disco.

Un sistema de archivos usualmente divide el disco en bloques de igual tamaño, a estos se les denomina como ***bloques físicos***, cada uno de estos bloques tienen una ***dirección***, la cual es una dirección absoluta en todo el disco.

Un archivo puede estar almacenado en uno o más bloques, con uno o más registros en cada bloque, si se supone un sistema operativo, que relaciona el nombre de los archivos y las direcciones de los bloques que ellos ocupan, se encontran bloques que contienen espacio que no es ocupado por registros del archivo, sin embargo, este espacio puede ser dedicado a otros usos como el almacenar ***encabezados***, que son colecciones de ***bytes*** que contienen información para relacionar los bloques que son ocupados por un mismo archivo o determinan como el bloque representa el archivo.

Los registros a su vez contienen una dirección, la cual puede ser la dirección absoluta del comienzo del registro o la dirección del bloque en el que se encuentra el registro, junto con un dato, comúnmente el número de bytes que en el bloque preceden el inicio del registro.

En este ámbito se habla de apuntadores a un registro o a un bloque, un ***apuntador a un bloque*** puede ser su dirección absoluta. Algunas veces el almacenamiento secundario esta organizado como una ***memoria virtual***, y los bloques o registros se ven señalados por una dirección dada a partir de un área base de la memoria virtual, todo ello con la ayuda de un sistema que traduce de entre estas direcciones y las direcciones absolutas. Una forma de apuntar a un registro, consiste en direccionar a su bloque. Si se decide por esta representación, se puede encontrar el registro deseado dentro del bloque, una vez que el bloque se haya encontrado, para esto último, se debe saber lo suficiente acerca del registro para identificarlo, la ventaja de utilizar apuntadores a los bloques consiste en que los registros pueden moverse dentro de los bloques, sin causar que los apuntadores se vean ***desbalanceados***, es decir, continuaran siendo correctos en su asignación.

La operación básica del almacenamiento externo, es la transferencia de un bloque del dispositivo secundario a la memoria principal y viceversa, la cual se denomina operación de ***acceder un bloque***, una situación muy importante que debe notarse, consiste en el tiempo que los sistemas se toman para realizar las operaciones de acceder los bloques, siendo relevante destacar, que una vez en la memoria principal es posible usar la información de manera ágil, por lo anterior, es justificable referirse a la ***rapidez*** de varios de los algoritmos para acceder la información, en función del número de bloques que son leidos para cargar la memoria principal en una unidad de tiempo, siendo este número la velocidad estimada del algoritmo.

En la organización más simple, y que consiste en suponer que cada registro tiene un formato fijo y los mismos campos en el mismo orden, los cuales a su vez tienen un tipo asociado con el mismo numero de bytes, si los registros son empacados desde el principio de los bloques, a partir del encabezado, se pueden recobrar de manera única los registros del bloque, así el único problema vigente, consiste en distinguir entre los registros y el espacio en blanco del bloque, una solución común, es la de colocar un contador del número de registros en el encabezado del bloque respectivo.

Si se considera el modelo de datos teórico conocido como ***entidad-relación***, se nota, que los archivos pueden almacenar información de dos tipos:

1. Archivos que representan ***conjuntos de entidades***, cada registro será una ***entidad*** y cada campo un ***atributo***.

2. Archivos que representan ***relaciones***. Para el caso de una relación ***varios a varios***, se puede representar a través de un archivo con registros con dos campos (o **k** campos en caso que la relación sea de **k** conjuntos de entidades). Cada campo es del tipo apuntador, asumiendo que la relación es entre dos conjuntos de entidades, cada registro consiste de un par de apuntadores (p_1,p_2), en donde p_1 apunta al registro de una entidad del primer conjunto y p_2 apunta a la entidad relacionada en el segundo conjunto de entidades. Por el contrario, si la relación es de tipo ***uno a uno o varios a uno***, existen otro tipo de representaciones.

Una diferencia importante entre estos dos tipos de archivos, consiste en que en el primer caso existe una *llave no trivial*, ya que uno o más campos juntos identifican, de manera única al registro, mientras en el segundo caso no hay garantía de que exista una llave diferente a la unión de todos los campos, para la generalidad de la situaciones, una consulta que resulta típica es aquella consistente en encontrar un registro que corresponda a determinados valores de la(s) llave(s), este tipo de cuestionamiento es mucho más sencillo de manejar, que ciertas peticiones de información más generales, por lo tanto es con este enfoque más general como debe realizarse el estudio de la definición física de las bases de datos.

Otra consideración importante en el diseño físico, corresponde a determinar si se manejan registros sujetos a una localización física fija, los registros llegan a estar sujetos porque existen apuntadores a ellos desde diversos lugares de la base de datos. Por ejemplo, se ha mencionado que existen posibles diseños, en donde las relaciones son representadas por pares de apuntadores a los registros relacionados, también, se conoce que algunas veces una entidad se puede identificar de manera única, solo por medio de su liga a otra entidad, por lo que se necesita, que un apuntador del registro de la primera entidad se direccione al registro de la segunda entidad, es así, como se deduce que no es posible mover los registros correspondientes, o los apuntadores asociados se verán desbalanceados, es decir, no apuntarán hacia la información para la cual fueron creados.

Es por la razón anterior, que en la mayoría de los casos, no es posible utilizar una organización de archivos en que los registros se muevan (por ejemplo, al momento de insertar algún registro), dado que no se conoce generalmente la organización actual de los apuntadores en la base de datos.

El borrado de registros por lo general, también es peligroso, puesto que cualquier apuntador a los registros puede perderse, afortunadamente si hay apuntadores asociados a un archivo en particular, es posible establecer algunas organizaciones más flexibles si se parte del principio de que los registros están sujetos a su localización original.

E.2. TECNICAS CLASICAS DE ALMACENAMIENTO.

El más sencillo y obvio esquema de almacenar un archivo de registros, consiste en listarlos en tantos bloques como registros se tengan. Esta organización es conocida como ***pila*** cuando es necesario denominarla de alguna manera, los bloques usados por una pila pueden ligarse por apuntadores, o por una tabla de direcciones que se almacena en algunos bloques adicionales. Para insertar un registro, este es colocado en el último bloque si existe espacio, o se utiliza un nuevo bloque cuando no lo haya. El borrado consiste en establecer un ***bit de borrado*** en el registro. La reutilización del espacio de los registros borrados es peligrosa si existen apuntadores a los registros. Sin embargo, si se asegura que el archivo no esta sujeto por apuntadores, existe una gran variedad de estrategias de reutilización de espacio.

Dado el valor de una llave, la búsqueda del registro requiere de un recorrido completo del archivo, o en el mejor de los casos, del recorrido en promedio de la mitad del archivo. Es esta operación lo que hace prohibitivo la utilización de demasiados bloques para el archivo. En adelante, el tópico de más importancia en esta discusión es el considerar alternativas de organización de archivos, los cuales permitan búsquedas arbitrarias, sin la necesidad de recorrer más que una pequeña fracción del archivo, en el diseño físico, se debe evitar al máximo utilizar demasiado espacio, y establecer las operaciones de inserción y borrado lo más fáciles posible, a continuación se describen algunos mecanismos que han sido utilizados para alcanzar estos objetivos.

E.2.1. ARCHIVOS DISPERSOS (HASH).

La idea básica en la organización de archivos por medio del método ***hash***, es la de manejar los registros del archivo en ***segmentos*** que consisten de uno o más bloques. Para cada archivo almacenado de esta manera, existe una función **H**, denominada ***función de dispersión (hash)***, la cual tiene como argumento un valor **v** de una llave del archivo y produce un entero que va de cero a un valor máximo determinado, es decir, si **v** es el valor de una llave, **H(v)** es el número de segmento en el cual se encuentra un registro con el valor de la llave, si es que este existe.

Es deseable que **H** disperse homogéneamente a **v**, esto es, que **H(v)** tome todos los posibles valores con una probabilidad igual. Un procedimiento usual para la realización del método de dispersión es el siguiente:

1. Tratar al valor de la llave como una secuencia de ***bits***, formada por la concatenación de los valores de cada campo de la llave, esta secuencia de bits es de tamaño fijo, una vez que cada campo es de tamaño fijo.

2. Dividir la secuencia de bits en grupos de número fijo, por ejemplo **16**, rellenando con ceros el último grupo si es necesario.

3. Sumar los grupos de los bits como enteros.

4. Dividir la suma entre el número de grupos y el residuo determina el número de segmento resultante.

Se considera una organización con **B** segmentos y la existencia de un directorio de dichos segmentos, el cual consiste de **B** apuntadores, uno por cada segmento, y a su vez cada apuntador se encuentra direccionado al primer bloque del segmento.

En el caso de un segmento que consiste de un solo bloque, el encabezado de dicho bloque se direcciona al *valor nulo*, este valor no puede ser la dirección de un bloque, por el contrario, si el segmento se conforma de más de un bloque, el apuntador del primero se direcciona al encabezado del segundo y así sucesivamente hasta que el encabezado del último de los bloques apunta al valor nulo.

Si el número **B** es pequeño, el directorio de segmentos puede alojarse en la memoria principal, en otro caso, este es almacenado en tantos bloques como sea necesario en el dispositivo secundario, y el bloque del directorio de segmentos con el apuntador al primer bloque del segmento **i**, será transportado a la memoria principal cuando el valor **i** es el resultado de la función **H**.

Los bloques tienen una capacidad para un número fijo de registros, si un registro requiere de r bytes, entonces es posible suponer que cada registro comienza en un múltiplo de r bytes a partir del primer byte después del encabezado.

El espacio necesario para contener un registro se conoce como *sub-bloque*. En ciertas circunstancias el primer intervalo de r bytes se encuentra vacío y el siguiente contiene un registro, existen formas de distinguir entre sub-bloques vacíos y llenos, uno de ellos es rellenar los sub-bloques vacíos con una secuencia de bits que ningún registro pueda tomar como valor, sin embargo, un método más seguro consiste en colocar en el encabezado un bit por cada sub-bloque, el cual toma el valor 0 si este se encuentra vacío y 1 en caso contrario, si al mismo tiempo se agrega un bit al registro, el cual indique su situación de borrado, es posible entonces evitar la reutilización de sub-bloques y por lo tanto eliminar la posibilidad de apuntadores desbalanceados.

E.2.1.1. OPERACIONES EN ARCHIVOS DISPERSOS.

Búsqueda. Si se supone un valor v, el cual es el valor del campo llave, o si la llave de más de un campo, es la lista de valores de los campos llave en un orden fijo y se calcula **H(v)** generando así un número de segmento, por decir **i**, el sistema consulta el directorio de segmentos y localiza el primer bloque para dicho segmento, a continuación cada sub-bloque no vacío es consultado para encontrar el registro con valor de la llave igual a v. Si el registro no se localiza, se accede el bloque siguiente a partir del apuntador del encabezado, repitiendo este proceso hasta que se ubique el registro o se analice el último bloque del segmento.

Modificación. Si se supone que se desea modificar uno o más registros cuya llave es el valor v, es necesario determinar si alguno de los campos a modificar es parte de la llave, de ser este el caso, el tratamiento de la modificación consiste en realizar una eliminación del registro actual seguido de una inserción con los nuevos valores, esto se debe a que es posible que el registro modificado pertenezca a un segmento diferente al cambiar el valor de la llave, esta precaución cobra mayor importancia cuando los registros se encuentra en una estructura *sujeta*. Por el contrario, si los campos a modificar no son parte de la llave, el procedimiento consiste en encontrar el registro bajo el proceso de búsqueda ya descrito, si el registro existe se modifica como se desea.

Inserción. Primero, debe aplicarse el procedimiento de búsqueda, si existe ya un registro con valor de llave v se debe generar una condición de error, contrariamente a esta situación, si no existe registro con llave igual a v, se localiza el primer sub-bloque vacío en los bloques que conforman el segmento **H(v)**, la localización de éste se puede realizar de acuerdo al procedimiento de búsqueda ya descrito y colocar el registro en el sub-bloque encontrado, si por el contrario, no existen sub-bloques vacíos en los bloques del segmento correspondiente, deberá solicitarse al sistema que provea de un bloque nuevo al segmento **H(v)**, colocar el apuntador del encabezado del último bloque a este nuevo bloque y su correspondiente apuntador direccionarlo al valor nulo, finalmente colocar el registro a insertar en el primer sub-bloque del nuevo bloque.

Eliminación. Para cancelar un registro con valor de la llave v, debe ser primero aplicado el procedimiento de búsqueda, una de las formas más simples de eliminación, consiste en hacer que el sub-bloque que contiene al registro sea marcado como vacío en el encabezado, haciendo así que el sub-bloque pueda ser reutilizado, sin embargo, si existen apuntadores a los registros, no es permisible que los sub-bloques sean reutilizados, porque puede ser causa de que un apuntador se direccione de manera errónea a un registro nuevo. Para evitar esta situación es aconsejable la utilización de un bit de borrado, que asegure la consistencia de los apuntadores.

E.2.1.2. TIEMPOS BAJO EL ESQUEMA DE ARCHIVOS DISPERSOS.

Como se puede deducir, cada una de las operaciones requiere de acceder una vez el dispositivo secundario para obtener el bloque que interesa, a partir del directorio de segmentos, no se necesitarán *más* accesos para examinar el segmento. Si el registro buscado se encuentra presente, se debe analizar en la mitad de los bloques en promedio, para una operación diferente a la búsqueda, se debe escribir el bloque modificado de regreso al dispositivo de almacenamiento. En el mejor de los casos, un segmento consiste de un bloque en promedio, en cuyo caso, toma dos accesos para la búsqueda y tres para cualquier otra operación, independientemente del tamaño del archivo. Para lograr que los segmentos cuenten con un número pequeño de bloques, el número de segmentos debe ser aproximadamente el mismo que el de los registros en el archivo dividido entre el número de registros que puede caber en un bloque. Si el número de registros crece radicalmente, es necesario reorganizar el archivo, modificando la función hash e incrementando el tamaño del directorio de segmentos, esto no es necesariamente caótico, si se toman los siguientes lineamientos:

1. Considerar que la función **H** es calculada al tomar un valor v de una llave, generando un entero (muchas veces más grande del necesario), dividiendo ese entero entre el número de segmentos y tomando el residuo.

2. Reorganizar el número de segmentos, multiplicando por un entero c fijo (comúnmente c=2).

Si se decide por un número de segmentos de **n** a **2n**, por la consideración (1), los registros del segmento original i se agruparán en dos segmentos, uno, el nuevo i y otro el **i+n**, generando una partición de los segmentos originales. La misma idea es posible aplicarla, si el número de segmentos se multiplica por algún entero **c>2**, entonces cada segmento original se divide en c segmentos nuevos independientes.

E.2.2. ARCHIVOS INDEXADOS.

En esta parte, se considera otra representación de archivos que son accedidos por medio de una llave, este método se conoce como *ISAM* (por las siglas en ingles de método de acceso secuencial de índices), en este método, se parte de una clasificación de los registros de un archivo a través del valor de sus llaves, se observa que no importa el dominio de dichos valores, la justificación es que una vez almacenados, los valores de los campos pueden ser tratados como cadenas de bits y por lo tanto ser ordenados como enteros, caracteres o reales. Para el caso en que la llave se componga de más de un campo, se puede ordenar seleccionando arbitrariamente un orden en los campos que conforman la llave.

Se puede obtener ventaja de conocer el orden en que se encuentran los registros, sobretodo para la búsqueda rápida de uno de ellos en función de la llave, la representación de este tipo de archivos consiste en tener el archivo clasificado, el cual se denomina *principal* y la existencia de un segundo archivo conocido como *índice*, este último consistente de pares (**valores de llave, bloques**), esto es, en el archivo índice el par (**v,b**), indica el primer registro en el bloque **b** cuyo valor de llave es **v**, el primer campo es una llave para el archivo índice y los registros que lo conforman están clasificados por esta llave. Estrictamente hablando, un archivo de índices es otro archivo con una llave.

Sin embargo, hay una diferencia importante entre los archivos índices y los archivos comunes, lo cual se puede observar si al realizar inserciones, eliminaciones y modificaciones al archivo índice, también se desea obtener respuestas del tipo; dado valor de la llave v_1, encontrar el registro (v_2,**b**) en el índice, que cumple con que $v_2 <= v_1$ y ademas, (v_2,b) es el último registro en el índice o el siguiente registro es (v_3,b) con $v_1 < v_3$, lo cual se dice que v_2 ***cubre a*** v_1, esto es, como encontrar el bloque **b** que contiene un registro con valor de la llave igual a v_1, ya que el archivo índice garantiza que se encuentra clasificado. Cuestionamientos como el anterior, descartan ciertas organizaciones para aplicarse a los archivos de índices, por ejemplo, no es conveniente la utilización del esquema hash, puesto que no existe forma de encontrar el valor v_2 que cubra a v_1, sin tener que recorrer el archivo completo.

E.2.2.1. BUSQUEDA DE INDICES.

Suponiendo que existe un archivo de índices, que clasifica una colección conocida de bloques, y se desea encontrar un registro (v_2,b) de forma tal que v_2 cubre un valor dado de llave v_1. Una estrategia puede ser la conocida como ***búsqueda lineal***, que consiste en iniciar la inspección del índice desde su comienzo hasta que un registro de los analizados cumpla con que cubra a v_1, la desventaja de este proceso consiste en tener que cargar a memoria principal todo el archivo de índices, y en promedio tener que buscar en la mitad de los registros de éste, sin embargo, este esquema es mejor que un proceso de búsqueda lineal en el archivo principal; ya que si el archivo principal contiene c registros por bloque, entonces el índice tiene $1/^{cesimo}$ de registros. Aunado a lo anterior, el tamaño de los registros en el archivo de índices pueden ser más cortos que los registros del archivo maestro, facilitando su almacenamiento en menos bloques.

Una mejor estrategia es la denominada *búsqueda binaria*, la que funciona de acuerdo con lo siguiente; Dado un valor de la llave v_1 y un índice en los bloques $B_1,B_2,...,B_3$, buscar en el bloque de enmedio $B_{[n/2]}$ y comparar v_1 con el valor de la llave v_2 en el primer registro de dicho bloque, si $v_1 < v_2$, repetir la operación, pero en el conjunto de bloques $B_1,...,B_{[n/2]-1}$. Por el contrario, si $v_1 >= v_2$, el proceso se repite para $B_{[n/2]},...,B_n$, en todo caso, solo un bloque queda pendiente de ser inspeccionado, y debe usarse búsqueda lineal para encontrar la llave que cubre a v_1. Como se divide el número de bloques en dos en cada paso, en $[\log_2(n+1)]$ pasos a lo más se limita la búsqueda de un bloque. De lo anterior, se deduce que la búsqueda binaria requiere de que $\log_2(n)$ bloques sean cargados a la memoria principal, una vez inspeccionado el índice, se sabe exactamente que bloque del archivo principal debe ser examinado y quizás reescrito, a fin de realizar la operación deseada. El número total de bloques accedidos puede calcularse en $3+\log_2(n)$, lo que no es prohibitivo.

Un método mejor a la búsqueda binaria, puede considerarse en el denominado ***interpolación*** o ***búsqueda de cálculo de direcciones***, en este método se presupone el conocimiento de la distribución estadística de los valores de la llave. En general, si se supone que existe un algoritmo que para un cierto valor de la llave v_1 determina una fracción del intervalo entre v_2 y v_3 en donde es posible encontrar v_1, esta fracción puede expresarse como $f(v_1,v_2,v_3)$. Si un índice o parte de un índice se ubica en los bloques $B_1,...,B_n$, sea v_2 el primer valor de la llave en B_1 y v_3 el último valor de B_n, al analizar el bloque B_i donde $i=[n*f(v_1,v_2,v_3)]$ y al comparar el primer valor de llave con v_1, entonces, como en el proceso de búsqueda binaria, se repite el proceso en $B_1,...,B_{i-1}$ o $B_i,...,B_n$. Es posible mostrar que si se conoce la distribución de las llaves, entonces se espera solo examinar aproximadamente $1+\log_2(\log_2(n))$ bloques del archivo índice. Si a lo anterior se agregan dos accesos de lectura y escritura de un bloque del archivo principal, se tiene que el número total de operaciones consiste en $3+\log_2(\log_2(n))$, sensiblemente menor que el requerido en la búsqueda binaria.

E.2.2.2. OPERACIONES CON ARCHIVOS INDEXADOS DE REGISTROS NO SUJETOS.

Si se consideran las formas de realizar las operaciones de búsqueda, inserción, eliminación y modificación de registros en el esquema de archivos indexados, bajo la suposición de tener registros no sujetos por apuntadores a localidades físicas. Estas operaciones, tendrán una transacción similar que debe hacerse al archivo de índices, es importante considerar que el archivo de índices se encuentra ordenado por el valor de las llaves y tiene una estructura no sujeta de registros.

El archivo original se encuentra alojado en una secuencia de bloques $B_1,B_2,...,B_k$, con los registros contenidos cada bloque en cierto orden, los registros del bloque B_i preceden a aquellos contenidos en el B_{i+1}, para $i=1,2,...,k-1$. Al mismo tiempo, en el encabezado de cada bloque se cuenta con la información referente a que sub-bloques contienen información y cuales se encuentran vacíos, de manera similar a la mencionada en la organización hash. A continuación se describen las operaciones básicas, ignorando las condiciones de error, ya que al presentarse estas, se les da un tratamiento similar al mencionado ya en el esquema hash.

Búsqueda. Suponiendo que se desea encontrar en el archivo principal el registro con valor de llave v_1, se examina el archivo de índices, a fin de encontrar el bloque cuyo primer registro contiene un valor de llave v_2, de tal manera que v_2 cubra a v_1. A continuación se busca en ese bloque el registro con valor de llave v_1. La búsqueda del bloque puede ser lineal, una vez que, la lectura de dicho bloque toma más tiempo que la búsqueda propiamente dicha, sin embargo, es factible utilizar una búsqueda binaria. Una cuestión importante es la de asegurarse que el sub-bloque localizado no se encuentre vacío, situación que se puede resolver utilizando la información del encabezado del bloque.

Modificación. Para modificar un registro con valor de llave v_1, se utiliza el procedimiento de búsqueda a fin de localizar el registro, si la modificación es sobre la llave, es necesario tratar dicha transacción como una eliminación y una inserción, si este no es el caso, se realiza la modificación y se reescribe el registro.

Inserción. Para insertar un registro con valor de llave igual a v_1, se utiliza el procedimiento de búsqueda, para localizar el bloque B_i en el cual, el registro con valor de llave v_1 se pudiera encontrar. Para el caso especial en que v_1 preceda el valor de la llave del primer registro del bloque B_1, debe hacerse $i=1$. Se coloca el nuevo registro en el lugar correcto del bloque B_i haciendo los registros con valores de llave mayores a v_1 hacia la derecha, a fin lograr espacio para el mencionado nuevo registro, situación posible de resolver, si el bloque B_i tiene al menos un sub-bloque vacío. Si v_1 precede al valor de la llave v_2 del primer registro de B_i debe realizarse la modificación en el índice asociado al bloque B_i, para lo cual se utiliza el procedimiento de modificación ya descrito. Finalmente es necesario cambiar la información del encabezado del bloque B_i para reflejar la existencia del nuevo registro.

En caso de que el bloque B_i se encuentre lleno al momento de recorrer los registros, a fin de ganar espacio para el registro a insertar, para el último registro no hay lugar, para solucionar esta situación existen diversas técnicas, una de ellas consiste en dividir B_i en dos bloques no llenos, sin embargo, otra solución alternativa es la de examinar el bloque B_{i+1} en caso de que exista, si este tiene un sub-bloque vacío, se mueve el registro excedente del bloque B_i al primer sub-bloque de B_{i+1}, recorriendo los otros registros a la derecha hasta llenar el primer sub-bloque vacío, cambiando adecuadamente la información del encabezado correspondiente y modificando el registro del archivo de índices asociado a B_{i+1} para reflejar el nuevo valor de la llave del primer registro.

Si B_{i+1} no existe, porque $i=k$ o bien B_{i+1} se encuentra lleno, se debe solicitar al sistema un nuevo bloque el cual seguirá en orden a B_i. Colocar el registro excedente de B_i al nuevo bloque e insertar un registro para este en el archivo de índices, utilizando el mismo procedimiento descrito como si se tratara de insertar un registro en el archivo principal.

Eliminación. Similar al caso de la inserción, existen una diversidad de estrategias para la eliminación, se presenta el esquema más simple para este objetivo, el cual es apropiado si el número de eliminaciones es reducido. Para el borrado de un registro con valor de llave igual a v_1, se utiliza el procedimiento de búsqueda para ubicarlo, moviendo todo registro existente a la derecha de él un lugar a la izquierda y ajustando la información del encabezado del bloque correspondiente. Si el bloque resulta completamente vacío tras de la operación, es necesario ajustar también el registro del archivo de índices correspondiente al bloque modificado.

Como se observa en la operación de inserción, algunas veces es necesario encontrar el siguiente bloque de un archivo principal, en lugar de hacer un recorrido del archivo índice es posible colocar un apuntador en el encabezado de cada bloque, direccionado al siguiente bloque en el archivo. Similarmente, es posible colocar un apuntador al bloque siguiente en cada bloque del archivo de índices. Este procedimiento elimina la necesidad de tener una tabla de bloques del índice si la búsqueda lineal se utiliza ya que sólo se requiere, bajo este proceso, el conocer la dirección del primer bloque del archivo de índices.

E.2.2.3. OPERACIONES CON ARCHIVOS INDEXADOS DE REGISTROS SUJETOS.

Si los registros están sujetos al lugar en que fueron almacenados inicialmente, no es posible mantener registros clasificados dentro de un bloque. De los inconvenientes más fuertes, resalta el no poder asegurar siempre que los registros de un bloque preceden en orden a los registros del siguiente bloque. Una solución consiste en tratar el archivo de manera inicial, como si tuviera una organización no sujeta, sin embargo, cada bloque es visto como el primer bloque de un segmento, así cuando los registros se insertan, se agregan bloques al segmento y estos nuevos bloques son encadenados por apuntadores desde los bloques originales del segmento, también se crea un bloque vacío al principio del segmento, cuya función consiste en almacenar cualquier registro que preceda al primer registro del primer bloque en el archivo inicial. El índice en esta organización no cambia, y el primer registro de cada bloque del archivo inicial determina la distribución de los registros, dentro de los segmentos en forma permanente o al menos hasta que se hace necesario reorganizarlo en un esquema que tenga un número mayor de segmentos. A continuación se describe cómo se realizan las operaciones básicas en un archivo con una organización como la mencionada.

Iniciación. Se clasifica el archivo, los registros se distribuyen en bloques considerando no llenar completamente dichos bloques, esto último es para tener espacio en el caso de crecimiento del archivo y evitar tener grandes cadenas de bloques en los segmentos. Debe mantenerse un bloque adicional al inicio del segmento, para insertar los registros que preceden a los registros en el archivo inicial, se crea el índice con un registro para cada bloque, incluyendo desde luego el bloque vacío al inicio. El registro del índice para el bloque siguiente no tiene llave, sólo un apuntador.

Búsqueda. Encontrar el registro del índice, cuyo valor de la llave V_2 cubre al valor de la llave V_1 y es menor que el primer valor de la llave del archivo de índices, entonces el registro de índices buscado es el primer registro, siguiendo el apuntador en el registro índice seleccionando el primer bloque del segmento deseado, se examina el bloque y cualquier otro bloque encadenado a este, a fin de encontrar el registro con valor de llave igual a V_1.

Modificación. La estrategia a seguir en esta parte se sujeta a la descrita en la organización indexada con registros no sujetos.

Inserción. Se utiliza el procedimiento de búsqueda para localizar el segmento de interés, examinando los bloques del segmento y ubicar el primer lugar vacío. Si no hay sub-bloques vacíos, se obtiene un nuevo bloque y se establece un apuntador a partir del encabezado del último bloque del segmento, se inserta el nuevo registro en el nuevo bloque.

Eliminación. Se utiliza el procedimiento de búsqueda para ubicar el registro deseado, estableciendo el bit de indicación de vacío o lleno para el sub-bloque en cero, sin embargo, puede haber apuntadores a los registros ya eliminados, por lo que una estrategia alternativa más confiable debe ser usada, como la de indicar si el registro está eliminado marcándolo en su interior con una indicación pertinente.

En forma similar a la descrita en la organización indexada con registros no sujetos, se puede encontrar útil establecer ligas entre los bloques de los índices y entre los segmentos ordenados, una forma de hacer esto consiste en dejar espacio para un apuntador adicional en cada encabezado de bloque y ligar el primer bloque de los segmentos sucesivos. Una forma de optimizar el espacio es reemplazar el valor nulo del apuntador al final de cadena de cada segmento por un apuntador al primer bloque del siguiente segmento, indicando con un bit si el apuntador se direcciona al siguiente segmento o al siguiente bloque del mismo segmento. Si los registros no se alojan en un segmento en orden después de la inicialización, será muy complicado el proceso de examinar los registros en los bloques. Para auxiliar a este proceso es aconsejable agregar un apuntador en cada registro direccionado al siguiente en el orden, estos apuntadores son algo diferentes a los apuntadores convencionales, ya que no solo indican un bloque, ademas ellos señalan una imagen dentro del mismo, dicha imagen corresponde al número de byte en que comienza el registro almacenado en relación al inicio del bloque.

E.2.3. ARBOLES B (ARBOLES BALANCEADOS).

Dado que un índice no es más que un archivo con registros no sujetos, no hay razón por la cual no pueda existir un índice de un índice y a la vez un índice del nuevo índice, así sucesivamente hasta llegar a un índice que pueda ser almacenado en un bloque. Un esquema con estas características puede ser mucho más eficiente que un archivo con un nivel sencillo de indexado. Una configuración muy común para archivos extremadamente grandes, es inducir una jerarquía de índices que se ajuste a la jerarquía natural de los dispositivos de almacenamiento en los cuales el archivo reside.

Como ejemplo, si un archivo abarca varias unidades de disco, es posible establecer que todos los registros alojados en el primer disco tengan valores en las llaves de manera tal, que precedan a los almacenados en al unidad número dos y estos en orden preceden a los de la unidad tres, etc. El primer nivel de índices da el primer valor de llaves en cada unidad, en el interior de la unidad los cilindros son ordenados y el segundo nivel de índices indica el primer valor de llave en cada cilindro, y así quizá, dentro del cilindro se puedan ordenar las pistas, proceso que determina un tercer nivel de índices que da el primer valor de llave de cada pista, cada pista a su vez puede ser particionado en bloques y establecer un nivel cuatro de índices. Si esta organización se utiliza, es muy útil al momento de iniciar el archivo, dejar bloques libres en cada pista, pistas libres en los cilindros, etc., de manera tal que si un cilindro se llena, es factible encontrar un cilindro libre en el mismo disco para captar excesos al momento de insertar, así de esta manera, la base de datos puede existir por períodos muy largos de tiempo sin tener que reorganizarse físicamente.

Con un sistema de archivos adecuado se puede utilizar un esquema más general, en el cual los bloques se tratan de manera uniforme, sin tomar en cuenta el tipo de almacenamiento que tengan.

Definitivamente, se puede esquematizar la jerarquía de índices como un árbol, los métodos para realizar las operaciones básicas con un índice de varios niveles son generalizaciones claras de las técnicas ya mencionadas en otros esquemas de organización física. El único problema a considerarse, aparece cuando el primer nivel de índices excede la capacidad de un bloque, la idea básica de índices en varios niveles es independiente del número de dichos niveles, por lo que es válido agregar siempre un nivel de índices al que formalmente fue el primer nivel, facilitando que este pueda crecer más allá de la capacidad de un bloque.

La idea de un ***árbol balanceado*** (***árbol B***), es la configuración del árbol en donde cada trayectoria de la ***raíz*** (el bloque del primer nivel de índices) a cualquier ***hoja*** (bloques del archivo principal), es del mismo tamaño.

E.2.3.1. OPERACIONES EN ARBOLES B.

Al asumir que los registros del archivo principal son no sujetos y que los bloques de dicho archivo son parte del árbol balanceado, no se asegura que el espacio sea utilizado en forma eficiente, sin embargo, se facilita de gran manera su conceptualización.

Para la inserción y eliminación en un árbol balanceado, se puede utilizar la misma estrategia mencionada en la sección anterior (archivos indexados), aplicando las operaciones de inserción y eliminación a los nodos (bloques) del árbol en todos los niveles. Esta estrategia podría resultar en nodos que tuvieran desde un registro hasta el número máximo posible, más aún, los **árboles B** son definidos para utilizar una estrategia particular de inserción y eliminación que asegure que ninguno de los nodos a excepción del nodo raíz se encuentren llenos, como convención se asume que el número máximo de registros índice que puede contener es un entero impar de la siguiente forma **2d-1>=3**, y el número de registros que un bloque del archivo principal puede alojar también es un impar entero de la forma **2e-1>=3**.

Búsqueda. Para ubicar un registro con valor **v** en la llave, se busca la trayectoria de la raíz a alguno de los nodos, en donde el registro deseado se encuentra si es que existe, supóngase que en cierta parte de la búsqueda se ha alcanzado el nodo (bloque) **B**. Si **B** es una hoja, es decir, si se encuentra en un bloque del archivo principal, simplemente se examina el bloque **B** para encontrar el registro con valor de llave **v**.

Si **B** no es una hoja, entonces se trata de un bloque de índices, por lo que se procede a determinar el valor de llave en **B** que cubre a **v**, recordando que el primer registro de **B** no tiene valor de llave y que el valor de llave buscado está cubierto por el valor de llave del segundo registro. En el registro en **B** que cubre a **v** existe un apuntador a otro bloque, este bloque continua a **B** en el trayecto de considerado y es factible repetir el proceso de inspección con el nuevo bloque en lugar del bloque **B**.

Modificación. Como en los otros tipos de organización física ya descritas, una modificación que involucra un campo llave es en realidad una eliminación seguida de una inserción, mientras que una modificación que mantenga los campos llave fijos, es una búsqueda seguida de la reescritura del registro involucrado.

Inserción. Insertar un registro con valor de llave **v** implica aplicar el procedimiento de búsqueda para ubicar el bloque al cual pertenece el nuevo registro. Si existen menos de **2e-1** registros en **B**, simplemente se inserta el nuevo registro en el orden correspondiente en el bloque.

Por lo contrario, si existen ya 2e-1 registros en el bloque **B**, se crea un bloque nuevo B_1 y se dividen los registros de **B** junto con el nuevo registro en grupos de e registros cada uno, el primer grupo de e registros se acomoda en **B** y el restante en B_1.

Ahora si **P** es el bloque padre de **B**, el cual es conocido por la forma en que se realiza el procedimiento de búsqueda. Se aplica el procedimiento de inserción recursivamente, con una constante **d** en lugar de **e**, se inserta un registro para B_1 a la derecha del registro para **B** en el bloque de índices **P**. Nótese que si muchos de los antecesores de **B** tienen ya el número 2d-1 de registros, el efecto de insertar un registro en **B** puede impactar al árbol en varios niveles. Sin embargo, solo los antecesores de **B** se verán afectados. Si los efectos impactan a la raíz, esta se puede dividir y crear un nueva raíz con dos hijos, siendo esta la situación en la cual un bloque de índices puede tener menos de **d** registros.

Eliminación. Si se desea eliminar un registro con valor de llave v, se utiliza el procedimiento de búsqueda, a fin de ubicar el trayecto de la raíz al bloque **B** que contiene el registro. Si después de la eliminación el bloque **B** tiene **e** o más registros, la eliminación esta completa. Si por el contrario, el registro eliminado fue el primer registro del bloque **B**, debe actualizarse el nodo padre de **B**, cambiando el valor de llave en el registro existente para **B**, de acuerdo con el nuevo primer valor de llave de **B**. Si **B** es el primer nodo hijo, el padre no tiene valor de llave para **B** y debe acudirse al nodo padre del padre de **B**, etc., hasta encontrar un antecesor A_1 de **B**, de tal manera que A_1 no es el primer hijo de su nodo padre A_2. De esta manera, cada registro (v_1,p_1) en cada bloque de índices, tiene valor de llave v_1 igual a la más pequeña de todas las llaves del archivo original que se hayan encontrado entre las hojas que son descendientes del bloque apuntado por p_1. Esto es, el **árbol B** después de la eliminación, continua comportándose como un índice multinivel.

Si después de la eliminación, el bloque **B** tiene **e-1** registros, se puede analizar el bloque B_1 que tiene el mismo padre que **B** y reside inmediatamente a la izquierda o derecha de **B**. Si B_1 tiene más de e registros, se distribuyen los registros de **B** y B_1 como sea posible, manteniendo el orden, se modifica también el valor de las llaves para **B** y/o B_1 en el padre de **B** y si es necesario impactar a los demás antecesores de **B** como sus llaves se hayan afectado.

Si B_1 solo tiene e registros, entonces se combina **B** y B_1, el cual tendrá 2e-1 registros exactamente, y en el padre de **B** se modifica el registro para B_1 (proceso que podría requerir modificación de algunos antecesores de **B**) y se elimina el registro para **B**, la eliminación anterior requerirá de aplicar recursivamente el proceso de eliminación.

Si la eliminación impacta toda la trayectoria desde la hoja hasta la raíz, se podría dar el caso en que es necesario combinar los dos únicos hijos de la raíz. En ese caso el nodo formado por la combinación de los hijos se convierte en la raíz, y el nodo raíz original se elimina, siendo esta una situación en la cual el número de niveles disminuye.

E.2.3.2. TIEMPOS BAJO EL ESQUEMA DE ARBOLES B.

Si se supone que se tiene un archivo con n registros, organizado en un árbol B con parámetros d y e. El árbol no tendrá más de n/e hojas, y no más de n/de padres de hojas, n/d^2e padres de padres de hojas, etc.

Si hay i nodos en las trayectorías de la raíz a las hojas, entonces $n >= d^{i-1}e$ o por el contrario habría menos de un nodo en el nivel de la raíz, lo cual es imposible por definición.

De lo anterior se concluye que $i <= 1+\log_d(n/e)$, por lo que realizar una búsqueda, i operaciones de lectura sobre los bloques serán suficientes. Para una inserción, modificación o eliminación, normalmente solo el bloque que contiene el registro involucrado requiere de ser escrito. Aunque en los casos patológicos cerca de i lectura e i escrituras son necesarias. La probabilidad de encontrar bloques con muchos registros en una inserción o de encontrar bloques con muy pocos registros en una eliminación es baja. Sin embargo, no es difícil mostrar que para e=d=2 el número extra de lecturas y escrituras es una proporción muy aceptable y se puede hacer caso omiso de ella, estimando el número de lecturas y escrituras en $2+\log_d(n/e)$.

Por ejemplo, si n=1,000,000, e=5 y d=50, el número esperado de lecturas y escrituras de bloques en una operación es de $2+\log_{50}(20{,}000) <= 6$, siendo esta expresión mayor que la encontrada para el acceso de archivos dispersos (cerca de 3 lecturas/escrituras), pero mejor que al utilizar un nivel sencillo de indexado exceptuando desde luego, aquellas situaciones donde se puede realizar una interpolación para la búsqueda.

E.2.4. ARCHIVOS CON INDICE DENSO.

Si se supone que no se desea mantener el archivo clasificado permitiendo que los registros aparezcan aleatoriamente, bajo el esquema de archivos con índices densos es posible evitar que se tengan muchos bloques que se encuentren parcialmente vacíos en el archivo principal, más aún, las inserciones se realizan fácilmente ya que solo se debe mantener la localización del último bloque del archivo e insertar ahí el registro, cuando el último bloque se encuentra lleno, simplemente se coloca un nuevo bloque. Si las eliminaciones son frecuentes, aparecen "hoyos" en el archivo, siendo válido ignorar que ciertos sub-bloques se encuentran vacíos por eliminaciones o establecer un archivo separado con registros conformados por un campo, el cual es un apuntador al bloque que contiene sub-bloques vacíos.

El problema al usar un archivo sin clasificar, aparece cuando se debe tener un mecanismo para encontrar un registro dado su valor de llave. Para hacer esto eficientemente, se requiere de un archivo adicional denominado un *índice denso*, el cual consiste de registros (v,p) para cada valor de llave v en el archivo principal, donde p es un apuntador al registro del archivo principal que tiene un valor de llave p.

Para buscar, modificar o eliminar un registro del archivo principal, se debe realizar una búsqueda en el archivo de índices densos, el cual arroja el bloque en el que se debe buscar el archivo deseado. Entonces es necesario leer el bloque del archivo principal, si el registro requiere ser modificado se debe cambiar el registro y reescribirlo en su bloque. De lo anterior se desprende que se requieren de dos acciones sobre el bloque, adicionales a la búsqueda sobre el archivo de índices.

Si lo deseado es eliminar el registro, se requiere reescribir su bloque y también eliminar su registro con el valor de llave del archivo de índices densos. Esta operación toma dos acciones más que una búsqueda y eliminación en el archivo de índices densos.

Para realizar una inserción se debe agregar el registro al final del archivo principal e insertar un apuntador al registro en el archivo de índices densos, una vez más, esta operación toma dos acciones adicionales.

Hasta ahora se ha visto, que el uso de un archivo de índices densos requiere de dos acciones más sobre el archivo principal, que si se utilizara cualquier otro tipo de organización. Sin embargo, hay dos factores que justifican el uso de índices densos en algunas situaciones:

1. Los registros del archivo principal podrían ser sujetos, pero los registros del archivo de índices densos no necesitan serlo, facilitando el uso de una organización más eficiente sobre el archivo del índice denso.

2. Si los registros del archivo principal son muy grandes, el número total de bloques usados en el archivo de índice denso puede ser mucho menor que los necesarios para alojar un índice poco denso o un árbol B sobre el archivo principal.

Otro factor que apoya el uso de un archivo de índice denso, es la posibilidad de compactar completamente los bloques del archivo principal, mientras que en la organización de un árbol B, los bloques denominados hojas que contienen al archivo principal, se encuentran semivacíos, desperdiciando casi el 25% del espacio de almacenamiento, más aún, si el archivo principal tiene registros sujetos, no es posible utilizar el esquema del árbol B.

E.2.4.1. METODOS PARA MANEJAR ARCHIVOS CON INDICE DENSO DE REGISTROS NO SUJETOS.

Un uso adicional para un esquema de índice denso, es jugar el papel de receptor de apuntadores a los registros, esto es, un apuntador al registro r en lugar de ir directamente a él, se posiciona al índice denso que apunta a r. La desventaja de este procedimiento consiste en seguir un apuntador extra del índice denso al archivo principal, la compensación consiste en que los registros del archivo principal ahora son no sujetos (aunque los registros del archivo índice lo son). Cuando se desea mover un registro del archivo principal, se tiene solo que cambiar el apuntador en el índice denso que apunta al registro movilizado. A través de este esquema es posible usar una organización de almacenamiento más compacta y el ahorro de espacio puede cubrir el costo del uso del archivo de índice denso, por ejemplo, si el archivo es de estructura no sujeta, se pueden reusar los sub-bloques de registros eliminados.

Una idea similar, es aquella que evita un índice denso para el archivo principal completo, en su lugar posiciona apuntadores en cada encabezado del bloque a los registros de ese bloque. Bajo este esquema es posible movilizar los registros dentro del bloque con solo mover los apuntadores del encabezado, aunque no es posible mover registros entre bloques como puede hacerse en el esquema clásico de índice denso, es decir, los registros son no sujetos dentro de su bloque, pero siguen siendo sujetos a su bloque. Una generalización a esta idea, consiste en apuntar al segmento de un registro (asumiendo desde luego una organización dispersa). Sin embargo, puesto que los segmentos pueden crecer sin cota, debe cuidarse de proveer almacenamiento dentro del segmento para los apuntadores a todos los registros y estos apuntadores pueden ocupar varios bloques.

El costo de esta flexibilidad básicamente es el espacio extra necesario para alojar los apuntadores en los encabezados, ya que el tiempo para el seguimiento de estos apuntadores no es importante, una vez que no es necesario un acceso extra al bloque. Una situación en donde este esquema cobra mucha importancia, es aquella que consistente en tener registros de longitud variable y por lo tanto los ajustes dentro de los bloques son frecuentes (aunque algunas veces es necesario mover un registro que crece a otro bloque, una vez que no hay espacio suficiente en el bloque original).

Otra técnica para conformar archivos de registros no sujetos, es usar los valores de las llaves de los registros en lugar de apuntadores. Esto es, en lugar de almacenar la dirección de un registro r, se almacena el valor de la llave para r y su búsqueda se hace por medio del procedimiento estandar.

E.3. ARCHIVOS CON REGISTROS DE LONGITUD VARIABLE.

Hay muchas situaciones en las cuales es necesario almacenar archivos cuyos registros tienen una estructura más general que la planteada en la sección anterior. En este esquema, es necesario permitir que los campos como hasta ahora se conocen, sean reemplazados por grupos repetitivos de campos, en donde cada campo en el grupo representa el mismo tipo de objeto. Por ejemplo, una forma de almacenar un mapeo *varios a varios* del conjunto de entidades E_1 al conjunto de entidades E_2, es crear un archivo con registros de longitud variable por cada entidad de tipo E_1. Los registros consisten de campos representando una entidad e de tipo E_1 y un grupo repetitivo de valores, cada uno de ellos del tipo E_2 relacionados a e.

Se debe remarcar, que un archivo con las características descritas, es un ***archivo lógico***, esto es, un archivo con un nivel de abstracción más elevado que los archivos de registros de longitud fija.

Esta parte de la discusión, se centra en describir como se lleva a cabo un archivo lógico de registros de longitud variable a partir de archivos convencionales de registros de longitud fija.

Si la relación entre conjuntos de entidades E_1 y E_2 es *varios a uno de* E_2 *a* E_1, se podrían mantener los atributos de una entidad de tipo E_2 en el registro único para la entidad de tipo E_1 a la cual pertenecen.

Se puede establecer que existe la posibilidad de que los elementos de los grupos consistan de más de un campo. A ciencia cierta los campos de un grupo pueden ser también grupos, de esta manera se podría representar en un archivo una relación varios a uno de un conjunto de entidades E_2 a un conjunto de entidades E_1, una relación varios a uno de un conjunto de entidades E_3 a E_2, etc.

La definición más aceptada que existe en relación a los registros de formato de longitud variable y que se asume en este documento, consiste de dos puntos.

1. Un registro de longitud variable es una lista de elementos.

2. Un elemento puede ser sencillamente un nombre de campo o un registro de formato de longitud variable, los cuales representan un grupo de cero o más registros de longitud variable.

La notación en este ámbito para el manejo adecuado de registros de longitud variable, se puede mencionar como sigue.

Un elemento, es un nombre de campo que es representado por su nombre, si el elemento es un grupo con formato A se representa como $(A)^*$, en donde * implica " repetición cero o más veces ".

A la secuencia de valores que toman los campos de un registro se le denomina la ***ocurrencia*** del registro.

Un registro de longitud variable, es representado como $A_1,A_2,...,A_n$.

E.3.1. ALMACENAMIENTO DE REGISTROS DE LONGITUD VARIABLE.

Un registro con grupos repetitivos no puede ser alojado indiscriminadamente en un bloque o bloques de dispositivo, porque no se tiene forma de saber el comportamiento de dicho grupo, entre ese comportamiento esta la variabilidad del tipo de campos que pueden aparecer en él. Una posibilidad es almacenar cada campo en secuencia, alojando tantos campos como quepan en un bloque; aunque no es usual que los campos se extiendan más allá de un bloque, un problema surge cuando se establece que no es fácil determinar que campos contiene un bloque, dado que no se conoce en donde se inicia un campo y donde termina, puesto que se ignora el tipo de dato que representa los campos, en función de lo anterior, es necesario establecer una indicación del nombre del campo o el tipo de dato alojado en cada bloque.

Esta información puede mantenerse en un encabezado de bloque o en los mismos campos, el primer enfoque requiere de dejar espacio en el encabezado para el número máximo posible de campos en un bloque, el segundo punto de vista requiere de examinar el bloque entero desde el principio, a fin de determinar todo campo existente en él.

Un enfoque más aceptado que los anteriores, es aquel que consiste en representar cada registro de longitud variable por uno o más registros de longitud fija, siendo las estrategias más comunes de llevar a cabo esto, las mencionadas a continuación.

1) METODO DEL ESPACIO RESERVADO.

Se supone que el número de ocurrencias de un grupo, esta limitado por un número c, por lo que se reemplaza el grupo por c grupos de nombres de campo.

Si el grupo ocurre menos de c veces, es necesario contar con un mecanismo que detecte campos vacíos, la forma más común de este proceso consiste en usar valores nulos, otra alternativa es agregar un campo que indique el número actual de ocurrencias del grupo.

2) METODO DEL APUNTADOR.

Se reemplaza un grupo por un apuntador al primer bloque de una cadena de bloques usados para alojar las ocurrencias del grupo.

3) METODOS COMBINADOS.

Hay varios mecanismos surgidos de la combinación de 1) y 2) que pueden establecerse, por ejemplo, se podría usar apuntadores para un grupo y una repetición fija para otro.

En general, se decide por una repetición fija si se conoce el número máximo de ocurrencias de un grupo, por regla casi siempre se decide utilizar el promedio de ocurrencias como el máximo, por lo que si el promedio y el máximo difieren mucho, el desperdicio de espacio será demasiado. Los métodos basados en apuntadores tienden a utilizar menos espacio, pero requieren más accesos a bloques para encontrar un campo.

Otro método de combinación de estrategias, es reemplazar un grupo por un número pequeño de ocurrencias fijas y un apuntador a la cadena de bloques en donde se alojan las ocurrencias adicionales que puedan surgir. Esta última estrategia cobra importancia cuando el número de ocurrencias tiende a estar cercano a su número promedio. Por lo tanto, si se elige dejar fijo un número de ocurrencias ligeramente mayor al promedio, el número de accesos a bloques por medio de apuntadores será mínimo y se reducirá el desperdicio de espacio.

E.3.2. OPERACIONES SOBRE REGISTROS DE LONGITUD VARIABLE.

Si se supone que se desea llevar a cabo un archivo lógico de registros de longitud variable por medio de un archivo principal con registros de longitud fija, con apuntadores adicionales y bloques que no formen parte del archivo principal. Una vez revisados los mecanismos de llevar a cabo esta situación, se puede asumir que un subconjunto de los campos en el registro de longitud fija (excluyendo campos que son apuntadores a bloques que contienen ocurrencias de grupos repetitivos) sirve como llave para los registros de longitud variable del archivo lógico, y por lo tanto, se puede determinar que la implementación de búsquedas, inserción, eliminación y modificación se puede hacer utilizando cualquier tipo de organización de las revisadas anteriormente.

El único detalle adicional, consiste en la creación de bloques o cadenas de bloques para los grupos repetitivos durante una inserción, si el esquema utilizado es el basado en apuntadores, análogamente debe preveerse un mecanismo para el regreso de bloques al sistema de archivos, si una eliminación ocurre.

También, es necesario considerar una nueva clase de operaciones, las cuales no tienen semejante en el ámbito de los archivos de longitud fija, como ejemplo, la inserción y eliminación de ocurrencias dentro de un grupo repetitivo, las cuales corresponden a modificaciones de una relación, si el archivo representa relaciones. No importa el método que se use para la representación física del archivo lógico, es posible ver cada grupo repetitivo como un archivo pequeño, y por lo tanto, de la misma manera que un archivo principal, existe una llave para el grupo repetitivo; esta llave se forma de los campos del grupo repetitivo.

Cualquier tipo de organización se puede utilizar para mantener el archivo para un grupo repetitivo, pero estos archivos son tan cortos que normalmente no se usa una estructura especial, dado un valor de la llave para el grupo repetitivo, simplemente se hace una búsqueda a través de las ocurrencias de un grupo. En este caso, dado un valor, la búsqueda dentro del grupo es directa; se puede eliminar una ocurrencia una vez que esta se ha ubicado y se puede insertar ya sea al principio o al final o en orden clasificado, una vez que se ha determinado que no existe una ocurrencia con el valor de llave dado.

E.4. ESTRUCTURAS PARA BUSQUEDAS SOBRE CAMPOS NO LLAVE.

Una base de datos versátil, debe dar respuesta a la petición de información sobre campos que no forman una llave, en esta parte se describen los mecanismos para organizar un archivo, de manera tal, que dado el valor para un conjunto particular de campos, que no son llaves, se puedan obtener los registros eficientemente. El método más generalizado consiste en crear ***índices secundarios***, los cuales relacionan valores de campo o campos con los registros que contienen esos valores.

Considerese un archivo cuyos registros tienen un campo **F**, cuyos valores posibles se comprenden en un conjunto de valores **D**. El campo **F** puede ser parte de una llave o puede estar fuera de ella, es posible que el valor de **F** pueda determinar de manera única un registro pero en este ámbito se considera que esto último no ocurre. Un ***índice secundario*** para un campo **F**, es una relación entre el dominio **D** y el conjunto de registros del archivo en cuestión. Un archivo con un índice secundario sobre el campo **F** se dice que es un archivo ***invertido sobre F***.

Se puede representar un índice secundario como un archivo lógico con formato:

VALOR(REGISTRO)*

Una instancia de **VALOR** es un valor de **D** y una instancia de registro puede ser:

(1). Un apuntador a un registro con un valor asociado en el campo **F**.

(2). Un valor llave para un registro con el valor deseado en el campo **F**.

Si el caso (1) se utiliza, el apuntador señala al sub-bloque que contiene al registro, o puede apuntar al bloque que contiene al registro, en tal caso una búsqueda de bloque es necesaria para encontrar el registro o registros deseados. En el caso (2), los registros del archivo son por consecuencia sujetos al menos dentro de su bloque. El apuntador podría direccionarse al segmento que contiene el registro deseado (para organizaciones de registros fijos que usan segmentos), en este caso la búsqueda de un segmento es necesaria para encontrar el registro, pero los registros son no sujetos, a excepción de su bloque.

Usando el caso (2) los registros del archivo principal son no sujetos por apuntadores del índice secundario, sin embargo, este método requiere de varios accesos adicionales para realizar una búsqueda de un registro dado su valor de llave, por otro lado el método (2) va directamente al registro, o al menos a su bloque o segmento.

CONCLUSIONES.

Las metodologías para el desarrollo de software, evolucionan en respuesta a las necesidades de los diseñadores y desarrolladores de software, sin embargo, esas necesidades a su vez evolucionan en respuesta a la experiencia ganada de soluciones pasadas. Las primeras técnicas de abstracción de la programación estructurada fueron procedimientos o macros, las cuales han evolucionado a tipos de datos abstractos y definiciones genéricas. Las metodologías para el desarrollo de sistemas de información emergen cuando se encuentran patrones útiles y comunes y se intenta utilizarlos como modelos, al mismo tiempo, evolucionan para que dichos modelos lleguen a ser estándares.

Como las técnicas de abstracción han sido capaces de orientarse a un intervalo más grande de tipos de organización de sistemas, las técnicas de especificación formal han llegado a ser más precisas y han jugado un papel crucial en el proceso del desarrollo y mantenimiento del software, de manera que se llegue a asegurar un balance adecuado en el costo-beneficio del mismo.

En el futuro se debe esperar una mayor diversidad en los programas que se utilicen como base de la modularización, y deben esperarse especificaciones que se vean involucradas con aspectos de los programas no considerados hasta ahora.

La elección de un modelo de datos para una aplicación específica, es similar al procedimiento que se utiliza para elegir un lenguaje de programación, un esquema de representación de conocimientos o una notación matemática, es decir, es una elección subjetiva. En el tiempo que se introdujeron los modelos clásicos, existió discusión sobre el valor de los modelos, ahora se sabe, que los modelos no difieren significativamente en su poder de expresión ya que sus reglas corresponden a esquemas similares.

La diferencia primordial entre los diversos modelos de datos es la clasificación de sus restricciones específicas en inherentes, explícitas o implícitas.

La investigación de modelos de datos ha intentando desarrollar medios para expresar restricciones de una manera uniforme que soporte definición, análisis y modificaciones, también existe un esfuerzo considerable que está dirigido a hacer la verificación de la integridad semantica a un costo mínimo. Una forma de alcanzar este objetivo, es hacer que la mayoría de las restricciones sean inherentes al modelo de datos, otro enfoque ha sido integrar conceptos de datos y procedimientos en mecanismos uniformes, utilizando resultados de lenguajes de programación e inteligencia artificial.

Hoy en día, hay una motivación importante que tiende a soportar en el ámbito de las bases de datos las aplicaciones en las cuales la información está menos estructurada, y puede contener tipos de datos mucho más complejos que los que ahora se soportan. Ejemplos de esas aplicaciones son; **VLSI**, automatización de oficinas y fábricas y toda la ingeniería de bases de datos (específicamente **CAD/CAM**), cartografía, bases de datos de fotografías y otras imágenes, aplicaciones científicas, aplicaciones económicas, inteligencia, aplicaciones de textos, bases de datos estadísticas, etc. Es decir, cualquier aplicación que soporte una cantidad considerable de información.

Las aplicaciones mencionadas requerirán mecanismos para el modelado de las aplicaciones semánticas asociadas, esto último, es la motivación más importante de la siguiente generación de los modelos de datos, la cual es denominada ***modelo de datos de propósito especial*** o ***modelos de datos orientados a aplicaciones***.

Los conceptos, técnicas y herramientas necesarias para el diseño y la implementación de los futuros sistemas de información, se espera que surjan como un resultado combinado de varias áreas de la Ciencia de la Computación. Una área clave es la Inteligencia Artificial, que provee técnicas básicas de conocimiento para el razonamiento, solución de problemas y contestación a preguntas. Otra área es la de los Lenguajes de Programación, especialmente los lenguajes orientados a objetos, que proveen herramientas poderosas para el manejo de aplicaciones complejas, esto es, con diferentes y numerosos tipos de datos. Finalmente, el desarrollo de la tecnología de datos se encuentra relacionada con el manejo eficiente y preciso de grandes cantidades de información permanente y compartida. Los primeros resultados de la integración parcial de los campos mencionados, han recibido la atención generalizada de las comunidades científicas y técnicas.

Desde el punto de vista de las bases de datos, se ha invertido una gran cantidad de tiempo en intentar extender la actual tecnología, a fin de soportar ***modelos semánticos*** más ricos.

El área de las relaciones anidadas, está dedicada a extender la noción de ***relación plana*** a conceptos como los ***objetos jerárquicos*** o las ***relaciones anidadas***, en general, esto se asocia con la extensión de los conceptos relacionales como son; los lenguajes de petición, el diseño de esquemas y la evaluación de peticiones, en este ámbito, algunos prototipos se han diseñado e implementado de manera satisfactoria, sin embargo, quedan aún problemas por resolver.

En el escenario de los ***valores nulos*** para el manejo de información incompleta en las bases de datos, la investigación propone herramientas conceptuales nuevas para el desarrollo de aplicaciones que tienen que manejar valores nulos, como la utilización de valores nulos para el traslado de esquemas de red a esquemas relacionales, las bases de datos distribuidas y la consideración del concepto de relación universal.

Por lo tanto, existe la necesidad de integrar esos nuevos conceptos y las aplicaciones adecuadas en una estructura completa y consistente, sustentada tal vez en enfoques debidamente formalizados como la interpretación "***no hay información***" de los nulos ya que esta interpretación provee una extensión que preserva las ventajas del modelo relacional al estar fundamentada en la teoría de conjuntos y tener correspondencia amplia con el cálculo relacional y el álgebra relacional.

Otro enfoque, es dejar al usuario la definición de los tipos de datos que son necesarios para su aplicación. El concepto denominado ***tipos de datos abstractos***, es quizás, el más importante en este ámbito, dado que extiende la noción de dominio en teoría relacional.

Los ***sistemas orientados a objetos*** tratan de integrar conceptos como clases o tipos, encapsulación y herencia, desembocando en un modelo formal de datos soportado por un sistema de base de datos, entonces, los conceptos estándar que provienen de los lenguajes orientados a objetos, son integrados con los conceptos usuales del área de base de datos.

Un campo importante de estudio es el mejoramiento de los lenguajes de manipulación de bases de datos, a fin de soportar capacidad de inferencia sobre un modelo relacional estándar. Lo anterior significa que las técnicas provenientes de la Inteligencia Artificial, deben ser integradas en un sistema relacional. El punto de enfoque de acoplamiento implica la existencia de un mecanismo de comunicación entre un lenguaje de programación y un sistema manejador de bases de datos. Esta estrategia puede dividirse dependiendo del grado de acoplamiento, correspondiendo respectivamente al uso estático o dinámico del mecanismo de comunicación.

Otra estrategia en la investigación, la cuál es más ambiciosa, consiste en crear una nueva clase de sistemas que integran ambas tecnologías, lo cuál significan que nuevos lenguajes y mecanismos de inferencia tienen que ser diseñados e implementados.

Finalmente, una nueva área de investigación se centra alrededor del diseño e implementación de bases de datos, que puedan extenderse fácilmente para soportar sistemas del usuario final. El objetivo de estos sistemas, es soportar los prototipos especializados para modelos de datos particulares, esto es, dichos sistemas permitirán extensiones para el modelado de datos, proceso de peticiones, métodos de acceso y estructuras de almacenamiento.

REFERENCIAS BIBLIOGRAFICAS.

[ABIT 84] **S. Abiteboul, N. Bidoit:**

"Non First Normal Form For Relations to represent Hierarchically Organized Data", Proc. of the 3rd. ACM PODS, Waterloo, Ontario, Canada, 1984, and also in the Journal of Computer en System Sciencies 33, 1986.

[ABIT 87] **S. Abiteboul, S. Grumbach:**

"Bases de Données et Objets Structurés", Techniques et Sciencies de l'Informatique 1987.

[ABRI 74] **Abrial J.R.**

"Data Semantics", in Database Management, Klimbie & Coffman eds., North Holland Publ. Co., 1974.

[AH 87] **S. Abiteboul, R. Hull:**

"IFO: A Formal Semantic Database Model", ACM TODS, 12, 4, December 1987.

[APT 86] **C. Apt, H. Blair, A. Walker:**

"Towards a Theory of Declarative Knowledge", IBM Research Report RC11681, April 86.

[ATRE 80] **Atre, S.:**

"Structured Techniques for Design, Performance and Management", John Wiley & Sons. 1980.

[BANC 86a] **F. Bancilhon, S. Khoshafian:**

"A Calculus of Complex Objects", Proc of the 5th ACM SIGACT SIGMOD Symp on PODS, Cambridge, Massachussets, March 1986, pp 53-59.

[BANC 86b] **F. Bancilhon, D. Maier, J.D.Ullman:**

"Magic sets and other strange ways to implement logic programs", 5th ACM PODS, Cambridge, 1986.

[BANC 87] **F. Bancilhon, & all:**

"O2, an Object Oriented Data Model", Série de rapports techniques, GIP Altaïr, INRIA, 1987.

[BARRETT 85] R. Barrett, A. Ramsay and A. Sloman:

"POP-11, a Practical Language for Artificial Intelligence", Ellis Horwood Ltd., Chichester U.K. (1985).

[BATO 86] D. Batory, J. Garnett, J. Garza, K. Smith, K. Tsukuda, C. Txichell, T. Wise:

"GENESIS: A reconfigurable Database Management System", Technical Report TR-86-07, departament of Computer Science, University of Texas at Austin, March 1986.

[BIRK 67] G. Birkoff:

"Lattice Theory", Amer. Math. Soc., Providence, R.I., 1967.

[BIRT 73] G. Birtwistle, O.J. Dahl, B. Myhrhaug, K. Nygaard:

"Simula Begin, Auerbach Publ.", Philadelphia, Pa., 1973.

[BISK 81] J. Biskup:

"A Formal Approach to null values in database realtions", in "Advances in Database Theory", (H. Gallaire, J. Minker, and J.M. Nicolas, eds.) Vol. 1, pp. 299-341, New York 1981.

[BOUZ 83] Bouzeghoud M. and Gardarin G.:

"The design of an expert system for database design.", Intl. Workshop on New Applications of Databases, Cmbridge (UK), sept. 83. Published in New Applications of Databases, Academic Press, Gardarin & Gelenbe eds. 1984.

[BRAC 83] Bragger R.P., Dudler A., Rebsamen J., Zehnder C.A.:

"GAMBIT: An Interactive Database Design Tool for Data Structure, Integrity Constrains and Transactions", in Database Techniques for Professional Workstations, ETH Zurich 1983.

[BRODIE 80] Brodie Michael L.:

"On the Development of Data Models", Proceedings of the IEEE, Vol. 68, No. 9, September 1980.

[BURTON 86] M. Burton and N. Shadbolt:

"POP-11, Programming for Artificial Intelligence", Addison Wesley, New York, 1986.

[CARE 86] M. Carey, D. De Witt:

"The Architecture of the EXODUS extensible DBMS", Proc. of the International Workshop on Object Oriented Database Systems, Pacific Group, Sept 1986.

[CERI 86] S. Ceri, G. Gottlob, L. Lavazza:

"Traslation and Optimization of Logic Queries: The Algebraic Approach", Proc. of VLDB, Kyoto, Japan, 1986.

[CERI 88] S. Ceri, S. Crespi-Reghizi, G. Gottlob,F. Lamperti, L. Lavazza, L. Tanca, R. Zicari:

"The ALGRES Project", in advances in Database Technology-EDBT 88, Lectures Notes in computer Science, no. 303.

[CHEN 76] P.P. Chen:

"The Entity Relationship Model: Toward a Unified View of Data", ACM-TODS (1976) 1, pp. 9-37.

[CHOY 78] Choy D.M., F. Barbic, R.H. Gueting, R. Ruland, R. Zicari:

"Document Management and Handling", Proc. IEEE Office Automation Symposium, Gaithersburg, April, 1987.

[CODASYL 71] CODASYL Data Base Task Group:

"April 1971 Report", (Available from IFIP Administrative Data Processing Group, 40 Paulus Potterstraat, Amsterdam).

[CODASYL 73] CODASYL Data Base Language Task Group:

"Proposal for a data base facility in COBOL", Jan. 1973 (Available from Technical Services Branch, Dept. of Supply and Services, Ottawa, Canada).

[CODD 70] Codd E.F.:

"A Relational model of data for large shared data banks", Communications of the ACM 13 (6), 377-387 (1970).

[CODD 71] Codd, E.F.:

"Further Normalization of the Relational Model", in "Data Base Systems", Courant Computer Science Symposium 6, 1971, rustin, R.: Ed. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1972, pages 33-64.

[CODD 74] Codd, E.F.:

"Recent Investigations in Relational Data Base Systems", Proc. IFIP Congress 74, August 5-10, Stockholm Sweden, pages 1017-1021.

[CODD 79] Codd, E.F.:

"Extending the database relational model to capture more meaning", ACM Transaction on Database systems 4 (1979).

[DADA 86] P. Dadam, K. Kuesper. F. Andersen, H. Blanken, R. Erbe, J. Guenauer, V. Lum, P.Pistor, G. Walch:

"A DBMS Prototype to support extended NF^2 relations: An integrated view on flat tables and Hierarchies", Proc. of ACM-SIGMOD '86 Int. Conf. on Management of Data, Washington, D.C., 5/86, ACM-SIGMOD Record (1986) 2, pp. 356-367.

[DATE 81] Date, C.J.:

"An Introduction to Database Systems", (3rd. Ed.) Addison-Wesley Publ. Comp., Reading, Mass., march 1981.

[FAGIN 82] R. Fagin, A. Mendelzon, and J. Ullman:

"A Simplified Universal Relation Asumption and Properties", ACM Trans. Database Systems 7(3)(1982), 343-360.

[FISH 83] P. C. Fisher, S.J. Thomas:

"Operators for Non First Normal Form Relations", Proc. IEEE Computer Software and Applications Conference, 1983.

[FRINK 62] O.Frink:

"Pseudo-complements in semilattices", Duke Math. J. 29 (1962), 505-514.

[GARD 86] G. Gardarin, C. de Maindreville:

"Evaluation of Recursive Logic Programs as Recurrent Function Series", Proc of ACM SIGMOD 1986.

[GOLD 83] A. Goldberg, D. Robson:

"Smalltalk-80: The Language and its Implementation",Addison-Wesley Publishing Company, Reading, Massachussets, 1083.

[GRANT 77] J. Grant.:

"*Null values in a relational data base*", Inform. Process Lett. 6 (5) (1977). 156-157.

[GUET 87] R. Gueting, R. Zicari, D.M. Choy:

"*And Algebra for Structured Office Documents*", IBM amaden, Report R.J. 5559 (56648), March 1987, submitted for Publication.

[GUET 88] Güting R.H.:

"*A Geo-Relational Algebra: A model and a Query Language for Geometric Database System*", in Proc. Advances in Database Technolog-EDBT'88, Lectures Notes in Computer Sciencie, no. 3030 (Schmidt J. W., CeriS., Missikoff M. eds.), Springer Verlag, New York, 1988.

[GUTT 84] Guttman, A.:

"*R-Trees: A Dynamic Index Structure for Spatial Searching*", Proceedings of the 1984 SIGMOD Conference, Boston, Ma., 1984.

[HAMMER 78] M. Hammer and D. McLeod:

"*The Semantic Data model: A modelling mechanism for data base application*" in Proceedings of The ACM SIGMOD, International Conference on Management of Data (1978).

[HOLS 82] Holsapple C., Shen S., Whinston A.:

"*A Consulting System for Database Design*", Information Systems Vol 7, Nb 3, 1982.

[IMIEL 81] Imielinski and W. Lipski:

"*On representing incomplete information in a relational database*", in "Proceedings, 7h. Int. Conf. on Very Large Data Bases, Cannes, France". pp. 388-397, 1981.

[KING 82] King and D. McLeod:

"*The Event Database specification model*", in Improving Database Usability and Responsiveness, edited P. Scheuermann (1982).

[KING 85] R. King and D. McLeod:

"*Semantic Database models*", in Principles of Database Design" Vol. 1, Logical Organization, edited S.B. Yao (1985).

[KUPE 86] G.M. Kuper, M.Y. Vardi:

"The Logical Data Model", IBM Almaden, 1986.

[LACROIX 76] M. Lacroix and A. Pirotte:

"Generalized Joins", SIGMOD Record, Vol. 8. No. 3. Assoc. Comput. March. pp. 14-15 1976.

[LIEN 79] Y.E. Lien.:

"Multivalued Dependencies with Null Values in Relational Databases", in "Proceedings 5th. Int. Conf. on Very Large Data Bases. Rio de Janeiro." pp. 155-168. 1979.

[LIEN 82] Y.E. Lien.:

"On The Equivalence of Database models", J. Assoc. Comput. March. 29 (2) (1982) 333-362.

[LIPSKI 79] W. Lipski.:

"On Semantic issues connected with incomplete information databases", ACM Trans. Database Systems, 4 (3) (1979). 262-296.

[LORI 82] R.A. Lorie, R Haskin:

"On extending the Functions of a Relational Database System", Proc. ACM SIGMOD, 1982.

[LORIE 83] Lorie, R.A., Plouffe, W.:

"Complex Objects and Their Use in Design Transactions", Proc. Annual Meeting Database Week: Engineering Design Aplications (IEEE), San Jose, Cal., May 1983, PP. 115-121.

[LORIE 84] Lorie, R.A. et all:

"User interfaces and access techniques for Engineering Databases", Research Report RJ4155, IBM Research Lab., San Jose, Cal., Jan. 1984.

[MAIE 88] D. Maier, J. Stein:

"Development and Implementation of an Object-Oriented DBMS", B. Shriver, P. Wegner (Eds.) Research Directions in Object-Oriented Programming, MIT Press, 1988.

[MAKI 77] A. Makinouchi:

"*A Consideration on Normal Form of Not-necessarily Normalized Relations in the Relational Data Model*", Proc. VLDB, Japan 1977.

[MCLEOD 78] D. McLeod:

"*A Semantic Database model and its associated structured user interface*", Technical Report, MIT laboratory for Computer Science, Cambridge, Mass. (1978).

[NIEV 84] J. Nievergelt, H. Hintenberg, K.C. Sevcik:

"*The Grid File. An adaptable, Symetric Multikey File Structure*", ACM Transaction on Database Systems, Nol 9/1, March, 1984.

[PEN 84] Z. S. Pen and B. Liu:

"*Distribution of Cymbidium in Yunan*", Agricultural University of Yunan, Kunming, People's Republic of China [in Chinese] (1984).

[PIST 85a] Pistor P., T. Traunmueller:

"*A Database Language for Sets, Lists, and Tables*", IBM Heidelberg, Report TR 85.10.004, 1985, also in Information Systems.

[PIST 85b] Pistor P., F. Anderson:

"*Designing a Generalized NF^2 Data Model with SQL-type Language Interface*", in Proc. VLDB, Kyoto, 1986.

[PUCH 87] P. Pucheral:

"*Traitement efficace des Régles par un Opérateur Basé sur des Parcours de Graphe*", IIIéme Journées Base de Données Avancées, Port Camargue, 1987.

[REIT 78] Reiter R.:

"*On Closed World Database*", in Logic and Database, (Gallaire H. Nicolas J.M., Minker J. eds.), Plenum Press, New York, 1978.

[REITER 88] R. Reiter:

"*Towards a Logical Reconstruction of Relational Database Theory*", in "Perspectives on Conceptual Modelling", (M.L. Brodie, J.M. Mylopoulos. and J. W. Schmidt eds.). Springer Verlag. Berlin/New York. 1988.

[RKS 85] Roth M.A., H.K. Korth, A. Silberschatz:

"*Null Values in the Non-First-Normal-Form Relational Databases*", University of Texas in Austin, Report TR-85-32, 1985.

[ROBI 81] J.T. Robinson:

"*The k-d-b tree: A Search Structure for Large Multidimensional Dynamic Indexes*", Proc. of the SIGMOD Conference, June 1981.

[SCHMID 75] Schmid, H.A.; Swenson, J.R.:

"*On the Semantics of the Relational Data Model*", Proc. ACM SIGMOD, San Jose, California, May 1975, King, W.F., Ed., pages 211-223.

[SCS 86] Schek H.J., M.H. Scholl:

"*An algebra for the Relational Model with Relation-Valued Attributes*", Information Systems, 11, 1986.

[SHAW 80] Shaw Mary:

"*The Impact of Abstraction Concerns on Modern Programing Languages*", Proceedings of the IEEE, Vol. 68, No. 9, September 1980, pages 1119-1130.

[SHIP 81] D.W. Shipman:

"*The Functional Data Model and the Data Language DAPLEX*", ACM TODS, 6,1,1981.

[SMITH 77] J. M. Smith and D. C. P. Smith:

"*Agreggation and Generalization*", ACM Transactions on Database Systems 2 (2), (1977) pp. 105-133.

[SMITH 78] J. M. Smith and D. C. P. Smith:

"*Principles of Conceptual Database Design*", in Proceedings of NYU Symposium on Database Design, New York, 18-19, may 1978.

[SP 82] Schek H.J., P. Pistor:

"*Data Structures for an Integrated Data Base Management and Information Retrieval System*", Proc. VLDB, Mexico City, 1982.

[THF 86] Thomas S.J., P.C. Fisher:

"*Nested Relational Structures*", in Advances in Computing Research, (Kannelakis eds.), JAI, Press, 1986.

[ULLMAN 82] Ullman Jeffrey D.:

"*Principles of Database Systems, Second Edition*", Computer Science Press, Inc. 1982.

[VASS 79] Y. Vassiliou:

"*Null Values in Data Base Management: a Denotational Semantics Approach*", in "Proceeding. ACM SIGMOD Int. Conf. Management of Data. Boston." pp. 260-269. May 30 -June 1. 1979.

[VETTER 81] Vetter, Max:

"*Database Design Methodology*", Prentice-Hall International, Inc. 1981.

[WONG 82] E. Wong:

"*A statiscal approach to incomplete information in Database Systems*". ACM Trans. Database Systems 7 (3). 470-488. Sept. 1982.

[ZANI 77] C. Zaniolo:

"*Relational Views in a Database System: Support for Queries*", in "Proceeding IEEE Computer Applications and Software Conf. Chicago." pp. 267-275. Nov. 8-11. 1977.

[ZANI 83] C. Zaniolo:

"*The Database Language GEM*", SIGMOD '83, Database Week Proceedings of Anual Meeting, San Jose 5/83, ACM-SIGMOD Record (1983) 4, pp. 207-218.

[ZANI 84] C. Zaniolo et all:

"*An Implementation of GEM: Supporting a Semantic Data Model on a Relational Back-end*", in Proc. ACM SIGMOD, 1984.

[ZANI 84b] C. Zaniolo:

"*Database Relations with Null Values*", JCSS, 28, 1984.

[ZHAO 87] L.P. Zhao:

"*A menu-based inteface to the OODM*", Technical Memorandum No. 6, Department of Computer Studies, University of Leeds, 1987.

[ZHAO 88] Lipping Zhao and S.A. Roberts:

"*An Object-oriented data model for Database modelling, implementation and access*", The Computer Journal Vol. 31, No. 2, (1988).

[ZLOO 77] Sloof, M. M.:

"*Query-By-Example: A Data base language*", IBM Systems Journal, Vol. 16, No. 4, 1977, pp. 324-343.