UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

NORMALIZACION DE RELACIONES CON TECNICAS DE INTELIGENCIA ARTIFICIAL

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A

LUIS ALBERTO ACUÑA GARCIA

Director de Tesis: Ing. Alejandro Ramos Larios

MEXICO, D. F. 1990

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

I N D I C E

INTRODUCCION

La historia de la computación ha sido breve. Aún cuando se han logrado importantes avances en las áreas que lo componen, ciertos aspectos han quedado rezagados dentro de este desarrollo, tal es el caso del diseño, el cual a la fecha presenta diferencias en las partes que lo componen; por ejemplo, en el campo de la interpretación de información no existe un criterio que unifique metodologías, lo que genera diseños deficientes que requieren de grandes inversiones de tiempo y dinero para su mantenimiento.

Para realizar un diseño útil y eficiente en la elaboración de software, es necesario recopilar, ordenar y verificar la información. En el proceso que se realiza para ordenar ésta, aparecen los mayores problemas, ya que es aquí donde los datos son sometidos a interpretaciones que generalmente no coinciden de un diseñador a otro.

La teoría de Normalización adquiere relevancia en este campo, ya que a través de ella se busca obtener una metodología que genere modelos que minimicen los problemas en cuanto a las anomalías, tanto de búsqueda como de actualización, que se presentan en el momento de la explotación de los datos.

Debido a lo anterior, resulta importante generar paquetes normalizadores que, basados en la teoría de Normalización, proporcionen al diseñador una herramienta que mediante estructuras lógicas y funcionales, agilicen la etapa de diseño y le permitan generar un modelo de datos apegados a sus requerimientos.

Es principio de buena administración que los sistemas se diseñen en función de las necesidades y no que las necesidades se ajusten a los sistemas. Para esto es necesario desarrollarlos con base a una plataforma tecnológica que al mismo tiempo sea vanguardista y permita la utilización de herramientas que faciliten el análisis y diseño de los mismos.

El objetivo de este trabajo se fijo con base en los puntos expuestos anteriormente y se pretende proporcionar una herramienta que facilite una de las etapas del diseño de estructuras de información. La herramienta desarrollada es el paquete normalizador.

La teoría de Normalización involucra el entendimiento de conceptos formales que la fundamentan. Dichos conceptos llegan a ser abstractos y complicados, lo que impide su aplicación practica en el diseño.

Aunque la Normalización surgió con el modelo relacional y en la teoría se aplica directamente a este, puede ser transportada a cualquier modelo de datos.

Las ventajas fundamentales que presenta el paquete normalizador desarrollado en el presente trabajo, radica en su aplicación inmediata, puesto que no requiere de conocimientos formales de la teoría de Normalización y únicamente se necesita conocer el significado de la información y la relación entre los datos (la semántica de los datos).

La estructura del presente trabajo consta de dos partes fundamentales, la primera está constituida por los antecedentes teóricos, que ubican a la teoría de Normalización dentro del proceso de desarrollo de sistemas de información y plantea el por qué se utilizan técnicas de Inteligencia Artificial para el desarrollo de este paquete normalizador. La segunda parte presenta en forma detallada el desarrollo del producto.

La primera parte consta de seis capítulos, dentro de los cuales el primero describe la importancia de perfeccionar el manejo de la información en la vida cotidiana, los requerimientos necesarios para la implantación de un sistema de información y su justificación, los nuevos conceptos que han surgido de la consideración de la información como recurso, las características de los conceptos de las nuevas técnicas de programación y manejo de datos y el surgimiento de una tendencia que apoya cada vez más a los sistemas administrativos en las bases de datos.

El capítulo dos describe los objetivos que persigue un sistema de base de datos, sus elementos básicos y los diversos modelos de datos que comúnmente se manejan.

El capítulo tres describe los pasos del diseño de una base de datos, profundizando en el diseño lógico, en el cual se ubica el proceso de normalización.

En el capítulo cuarto se describen los conceptos que utiliza el modelo relacional así como sus principales características.

En el capítulo cinco se presenta el objetivo de la Normalización así como la descripción detallada de las diferentes Formas Normales y los conceptos asociados a cada uno de ellas.

El capítulo seis presenta un panorama general de la diversas técnicas de Inteligencia Artificial. Se explican las características propias del problema, se describen las más importantes técnicas de búsqueda, se menciona la necesidad de

representar el conocimiento con cierto formalismo, se muestran algunas herramientas con las que se hace posible le implantación de Sistemas Inteligentes, haciendo énfasis en los lenguajes declarativos y como caso específico PROLOG, lenguaje con el que se desarrolló el paquete normalizador.

La segunda parte consta de tiene un solo capítulo, en el cual se analizan las ventajas que brinda el paquete normalizador como parte del análisis y diseño de un modelo de información. Se establece que una forma de normalización automatizada asegura la cobertura de todos los pasos de algoritmos necesarios. También se menciona la relación del lenguaje de programación PROLOG y el enfoque relacional, presentando las ventajas con que éste cuenta para su utilización, además de un panorama general de su sintaxis. Se describe el paquete normalizador en forma detallada, como fue implementado en el presente trabajo explicando los algoritmos utilizados. Se concluye con la presentación completa del código fuente del paquete normalizador y las pruebas de ejecución.

Finalmente se presentan las conclusiones logradas como resultado de la realización de este trabajo.

I ANTECEDENTES TEORICOS

1 EL CONCEPTO DE INFORMACION

1.1 Importancia de la Información.

El avance científico y tecnológico es la característica principal de este siglo. En éste, se ha alcanzado un gran desarrollo tecnológico, así como un notable crecimiento del acervo científico, mucho más del alcanzado a lo largo de siglos anteriores.

Uno de los inventos importantes de este siglo es la computadora, que se ha convertido en una herramienta útil para el manejo de información en áreas tales como la ciencia, la administración, la industria y la educación.

Por manejo de información entendemos la generación, recopilación y procesamiento de datos. Los datos son afirmaciones, hechos o eventos que manipulados y combinados de cierta manera proporcionan información. Actualmente, el manejo de información no tiene precedentes, más aún, crece a un ritmo vertiginoso y continuará así en el futuro.

La computadora permite el registro masivo de datos gracias a la alta tecnología electrónica (Hardware) y permite el manejo de información a través del desarrollo de aplicaciones (Software) que conjuntamente forman un Sistema de Información. Sin embargo, los sistemas de información no se encuentran aislados en algún universo, sino que se encuentran bajo el marco de una organización y son gobernados precisamente por las estrategias y objetivos de las mismas[N,Y].

La infraestructura requerida para la implantación de sistemas de información, implica gastos considerables tales como:

- adquisición e instalación del equipo de computo,
- operativos,
- de desarrollo y mantenimiento de aplicaciones,
- de soporte técnico y
- de mantenimiento del equipo de cómputo (Hardware).

La adquisición de sistemas de información se justifica cuando se obtienen beneficios dirigidos hacia los objetivos de las organizaciones.

Al tener control sobre los datos relativos a alguna organi-

zación y su entorno, podemos obtener información confiable y oportuna, es decir, podemos extraer información veraz justo en el momento que se necesita.

Las organizaciones cuentan con diversos tipos de recursos:

- Humanos (Personal),
- Infraestructura (Edificios, Equipos, etc.) y
- Económicos (Capital).

La información adquiere por sí sola mayor importancia en la medida que proporciona beneficios a la organización; es entonces cuando la información toma el papel de recurso (recurso informático) que puede ser vital para el desarrollo de las diversas organizaciones.

Podemos decir entonces que:

Buen Aprovechamiento de Recursos = Obtención de mayores beneficios

Los objetivos que pretenden alcanzar las organizaciones utilizando el recurso informático son:

1) Contribuir a que las organizaciones ocupen un mejor lugar en el futuro, es decir, el Recurso Informático colabora en la supervivencia y crecimiento de la organización.

2) Proporcionar información oportuna a todos los niveles.

3) Ayudar a la planeación, diseño e implantación de Sistemas de Información ajustados a las estrategias de la organización, y no estrategias ajustadas a los sistemas de información.

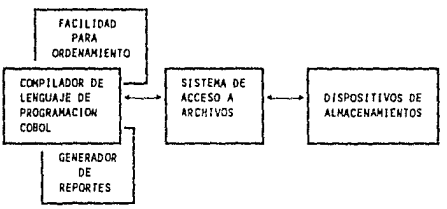
Los sistemas de información actuales son producto de casi 30 años de investigación y desarrollo de la computación, tanto en tecnología de equipo (hardware), como en el desarrollo de técnicas de programación y manipulación de datos (software).

1.2 Evolución de los sistemas de información.

A principios de la década de los años sesenta, el punto más importante fue la introducción por parte de CODASYL (Conference on Data Systems Languages) del compilador del lenguaje COBOL (COmmon Business Oriented Language), acompañado por la evolución de unidades de almacenamiento en cinta y la aparición subsecuente de los dispositivos de almacenamiento directo.

Inicio de la década de los sesenta.

Sin embargo, al comenzar la explotación de los sistemas de información y el surgimiento de aplicaciones más complejas, se observó la necesidad de agregar al compilador de COBOL paquetes (programas generalizados) que facilitan tanto el ordenamiento y clasificación de datos como la generación de reportes.



Mediados de la década de los sesentas.

Conforme se fueron desarrollando sistemas aún más complejos y sofisticados surgieron organizaciones lógicas de alto nivel para datos y las aplicaciones comenzaron a interrelacionarse entre sí para ponerse a disposición de un mayor número de usuarios.

Grupos empresariales, proveedores de equipos, casas de Software y universidades, siguieron de cerca la evolución de los sistemas de información y lograron identificar los problemas que éstos presentaban, visualizando así las características deseables de estos:

- Independencia de los datos.
- Inmunidad de modificación a programas de aplicación.
- Estructuras de almacenamiento.
- Estrategias de acceso.
- Flexibilidad para relacionar diferentes tipos de registros.
- Irredundancia de los datos.
- Rendimiento, eficiencia y seguridad de los sistemas de información.

Como productos comerciales, surgieron los Sistemas Generalizados para Manejo de Archivos: GFMS (Generalized File Management Systems), Sistemas Generalizados para la Administración de Bases de Datos: GDBMS (Generalized Data Base Management Systems) y Sistemas de Bases de Datos/Telecomunicaciones: DB/DC (Data Base/Data Communications).

En 1971, CODASYL presentó un documento acerca de las Bases de Datos, DBTG (Data Base Task Group) en el cual quedaron asentadas las bases para el desarrollo de los sistemas DBMS's (Data Base Management System) subsecuentes.

En el desarrollo de sistemas manejadores de archivos, se presentaron diversas tendencias en cuanto a la tecnología del manejo de datos, como son:

- **GFMS** (Aparece a mediados de los sesentas).

  Se conciben como sistemas que integran tanto las facilidades de los generadores de reportes, como de los paquetes de ordenamiento y clasificación de datos, teniendo como característica principal el poder utilizarlos como una herramienta poderosa en la manipulación de archivos.

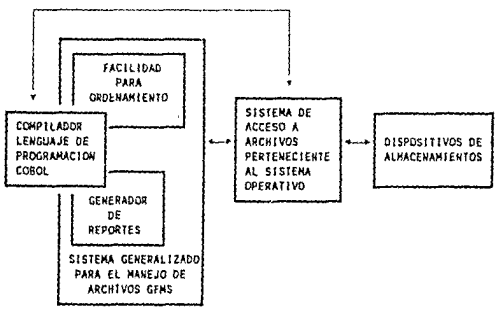
  Funcionalmente, un GFMS es capaz de realizar gran parte de las tareas de un programa COBOL, además de llevar el control del almacenamiento y recopilación de información. Los objetivos principales de un GFMS son la sustitución de lenguajes convencionales de programación en aspectos concernientes a:

  - Almacenamiento y Recopilación de información.

- Comunicación directa con los métodos de acceso básico del Sistema Operativo anfitrión.
- Producción de reportes.

El lenguaje de programación de un GFMS es de alto nivel y contiene operadores capaces de realizar ciertas funciones de manera más sencilla y amigable que en un lenguaje de programación convencional como COBOL.

Generalmente los GFMS se enfocan a aplicaciones que requieren grandes volúmenes de Entradas/Salidas. Los GFMS son parte importante de la tecnología para el manejo de datos.



Fines de la década de los sesenta, principios de los setenta.

- **GDBMS** (Aparece a principios de los setentas)

Simultáneamente aparece otro enfoque diferente, los Sistemas Generalizados de Administración de Bases de Datos(GDBMS). Los primeros esfuerzos en esta área se remontan a los años sesentas y surgieron debido a la necesidad de acceso efectivo a grandes archivos por parte de diversos programas de aplicación que además requerían de la integración e interrelación de la información en una base de datos común.

Se observó que las estructuras convencionales de archivos

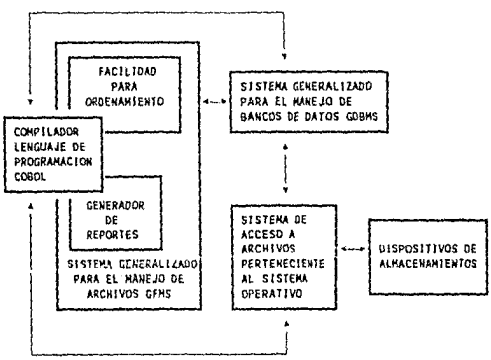
restringían las aplicaciones por sus características limitantes, siendo las principales:

- Una llave única de acceso.
- Un sólo tipo de registros.
- Acceso secuencial.

Estas limitaciones eran inadecuadas para aplicaciones más sofisticadas y específicas. Repercutían directamente en tiempo de procesamiento y en excesivos requerimientos de almacenamiento externo.

A su vez, para estas mismas aplicaciones, los GFMS se veían limitados por los métodos de acceso básico de su Sistema Operativo anfitrión. Es así como surgió el nuevo concepto de organizaciones de alto nivel para archivos (GDBMS) con las siguientes facilidades:

- independencia de datos,
- habilidad de compartir datos,
- flexibilidad de acceso (lenguaje especializado de consulta),
- habilidad de relacionar.



Desde principio de los años 70 al presente.

- **DB/DC**

Los GDBMS se desarrollaron debido a las limitaciones del sistema operativo anfitrion para el manejo de archivos. Análogamente los Sistemas Generalizados para Comunicaciones de Datos (DC) surgieron, más recientemente que los GDBMS debido a las limitaciones para el manejo básico de comunicaciones del sistema operativo anfitrión.

Las facilidades proporcionadas por el sistema DC son:

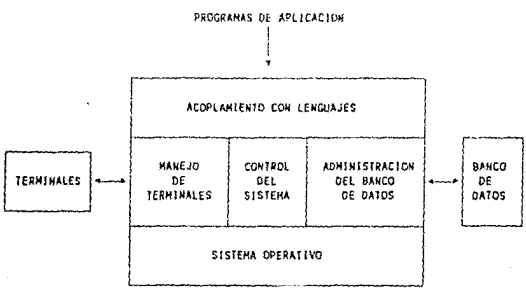
- Independencia de terminales, permitiendo utilizar cualquier tipo de terminal sin modificar el programa de aplicación.

- Eficiencia y control en la transferencia de grandes volúmenes de datos.

- Control de concurrencia. Acceso simultaneo a la Base de Datos a través de diferentes terminales y

- Formateo de mensajes y despliege de información en diferentes terminales.

El acoplamiento de un GDBMS (DBMS) y un DC dan como resultado un sistema DB/DC que ofrece las facilidades de ambos.



Sistema DB/DC

1.3 Alternativas actuales para el manejo de información.

Nos encontramos entonces con diversas opciones para la manipulación de datos, tales como:

- Utilización de lenguajes convencionales de programación (COBOL, PL/1, FORTRAN) para la comunicación con los métodos de acceso del Sistema Operativo anfitrión, a través de instrucciones estándares del lenguaje.

- Acceso a la base de datos a través del GDBMS desde los programas de aplicación, utilizando lenguajes convencionales de programación.

- Utilización de un lenguaje de consultas particular a cada GDBMS, de muy alto nivel para lograr accesos rápidos.

- Los programas de aplicación en un lenguaje de GFMS se comunican con los métodos de acceso del Sistema Operativo anfitrión.

- Los programas de aplicación de un lenguaje del GFMS para tener acceso a la base de datos mediante el GDBMS.

A pesar del desarrollo de estas nuevas técnicas de manipulación de información, aún predomina la utilización de lenguajes de programación convencionales y la utilización de los métodos de acceso básico que proporciona el Sistema Operativo anfitrión. Sin embargo la nueva tendencia se dirige hacia la utilización de los Sistemas Administradores de Bases de Datos (DBMS) y Telecomunicaciones (DB/DC)[f].

2 BASES DE DATOS

2.1 Objetivos de un Sistema Manejador de Bases de Datos.

La evolución de la organización de información ha originado el uso del término "Base de Datos" como el registro masivo de datos en archivos. Sin embargo debido a los adelantos en la tecnología de manipulación de información surge el concepto de Sistema de Bases de Datos o en forma general, Sistema Manejador de Bases de Datos, siendo una de las alternativas actuales mencionadas en el capítulo anterior.

Una Base de Datos es una colección de datos integrada, irredundante y que puede compartirse. Al conjunto de programas que permite tener una Base de Datos con estas características constituyen el Sistema Administrador de la Base de Datos (DBMS)[f,Y,N].

Los Sistemas de Bases de Datos pretenden lograr, entre otros, los objetivos que a continuación se describen.

2.1.1 Independencia de datos.

La independencia de datos se logra cuando aislamos a los usuarios de los programas de aplicación, y a éstos de la Base de Datos de tal manera que los cambios que surjan en la organización específica de la Base de Datos, tanto a nivel lógico como a nivel físico, no los afecten.

La independencia de datos física aisla los programas de aplicación de los cambios en la organización física de los datos, por ejemplo, cambios en la localización de los dispositivos de almacenamiento o modificación de las rutas de acceso.

La independencia de datos lógica aisla los programas de aplicación de los cambios en la organización lógica de los datos, por ejemplo, adición o eliminación de un campo en la definición de un tipo de registro o modificación de una relación lógica entre registros.

Este es un punto importante, ya que los cambios en sistemas de información son muy frecuentes, debido a requerimientos no previstos y al surgimiento de nuevas necesidades. Los cambios que se realizan y no se tienen bajo control,

producen serios problemas en los programas de aplicación. Una prueba contundente de este hecho, es el alto costo del mantenimiento del software, por lo que cada vez se asigna un presupuesto mayor a la modificación de programas de aplicación[U,P].

La tecnología de Bases de Datos pretende lograr tanta independencia de datos como sea posible, sin embargo, en la actualidad no existe un DBMS que proporcione una independencia absoluta de los mismos.

2.1.2 Habilidad de compartir datos e irredundancia de los mismos.

La redundancia de información es el tener una copia total o parcial de los datos que requiere cada aplicación.

El compartir datos significa que varias aplicaciones tengan acceso a una Base de Datos común.

La eliminación de redundancia conduce a la habilidad de compartir datos.

Como consecuencia de la habilidad de compartir datos, el DBMS debe ofrecer la transparencia necesaria para que las aplicaciones puedan operar sin percatarse de la existencia de las demás. Esto ocasiona un mayor tiempo de proceso para el control de concurrencias.

El tiempo de control de concurrencias deteriora el rendimiento (performance) del DBMS, llegando a ser crítico en algunas aplicaciones. En estos casos puede ser necesaria cierta redundancia para mejorar el rendimiento.

2.1.3 Habilidad de relacionar datos.

La habilidad de relacionar permite asociar registros o entidades a nivel lógico para obtener información útil (relacionada) y realista, sin ambigüedades dentro de un Sistema de Bases de Datos.

2.1.4 Integridad.

El término Integridad se refiere a diversas tareas que el DBMS debe realizar para mantener el control y preservar la consistencia de la propia Base de Datos. Estas tareas son las siguientes:

- Coordinación del acceso a los datos por diferentes aplicaciones.
- Propagación de los valores actualizados a otras copias o valores dependientes.
- Preservación de un alto grado de consistencia y validez de los datos.

El usuario es responsable de los datos específicos de su aplicación y la consistencia de datos entre aplicaciones es tarea del DBMS.

2.1.5 Flexibilidad de acceso.

La flexibilidad de acceso es la facilidad que el DBMS proporciona para consultar o actualizar los datos de la Base de Datos, mediante cualquier identificador (llave) eliminando la restricción de llave única que imponen los Sistemas Operativos.

El acceso a los datos puede ser a través de un programa de aplicación escrito en cualquier lenguaje convencional, aprovechando las facilidades de acceso del DBMS o bien, por medio de un lenguaje especializado de consultas similar al lenguaje natural (Ingles, Español, etc.) enfocado a usuarios no programadores.

2.1.6 Seguridad.

Para evitar los accesos accidentales o mal intencionados a la Base de Datos, el DBMS ofrece la facilidad de limitar la vista de la Base de Datos de acuerdo a la función que cada grupo de usuarios desempeña. Este elemento de control es el nivel de seguridad, que determina que tipo de operaciones (solo lectura, inserción, borrado, modificación

o alguna combinación de ellas) y sobre que datos trabajará cada grupo.

2.1.7 Rendimiento y eficiencia.

El gran tamaño de las Bases de Datos y el alto volumen de requerimientos, exigen de un buen funcionamiento de los métodos de almacenamiento, acceso y proceso, que se obtienen de las facilidades del Sistema Operativo de acuerdo a la relación tiempo / costo.

La viabilidad de un DBMS depende directamente de un rendimiento adecuado y de su eficiencia.

2.2 Elementos básicos de un Sistema Manejador de Bases de Datos.

Una vez identificadas la entidades (tipos de registros) y las relaciones que existen entre ellas (las cuales se analizarán en el capítulo 3) se requieren de elementos de traducción del concepto lógico de la Base de Datos a la implantación física de la misma.

Estos elementos que son comunes a cualquier DBMS son los siguientes:

- **DDL** (Data Description Language):

  Es un lenguaje especial de descripción de datos que permite definir la estructura lógica de las entidades y sus relaciones que forman así la Base de Datos. A esta estructura lógica se le conoce como Esquema Lógico de la Base de Datos. A su vez el DDL permite la definición de subconjuntos de la Base de Datos los que se denominan Subesquemas, los cuales dan una vista específica de los datos controlando el acceso a la información de cada programa de aplicación.

- **DML** (Data Manipulation Language):

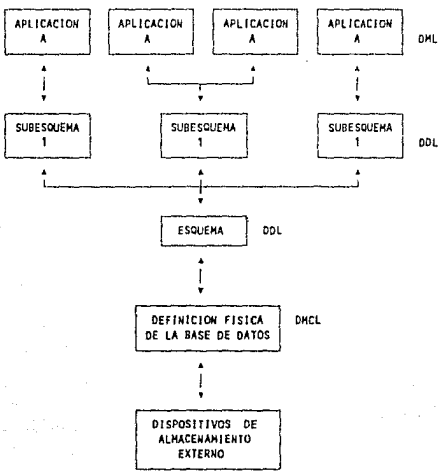
  Es un lenguaje especial de manipulación de datos el cual permite tener acceso a la Base de Datos según los subesquemas definidos.

Las instrucciones del DML se utilizan dentro de los programas de aplicación en lenguajes convencionales o interactivamente a través de lenguajes de consulta.

- **DMCL** (Device Media Control Language):

  Es un lenguaje que define la estructura física organizacional y de almacenamiento de la Base de Datos, partiendo de la definición del esquema.

La siguiente figura representa esquemáticamente una arquitectura general de un DBMS:



Arquitectura general de un DBMS

2.3 Tipos de Bases de Datos.

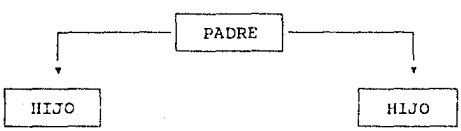
Existen enfoques alternativos para visualizar y manipular datos a un nivel lógico independientemente de cualquier estructura física de soporte en que se basen.

2.3.1 Modelo Jerárquico.

La estructura lógica en la cual se basa la Base de Datos jerárquica es el árbol. Un árbol se compone de un nodo raíz y varios nodos sucesores, ordenados jerárquicamente. Cada nodo representa una entidad (tipo de registro), y las relaciones entre entidades son las conexiones entre nodos.

El nodo colocado en la parte superior es llamado padre u "Owner" y los nodos inferiores a estos son llamados hijos o "Members".

En el sistema jerárquico las conexiones entre archivos no dependen de la información contenida en ellos; las conexiones se definen al inicio y son fijas, entonces, el registro "A" siempre estará ligado al registro "B" no importando cual sea el contenido de estos y obviamente las conexiones entre tipos de registros son jerárquicas como se muestra en la siguiente figura.



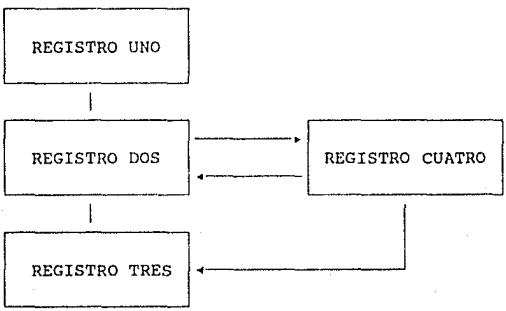
Modelo Jerárquico

La característica sobresaliente de este modelo es el manejo de la conexión UNO-A-MUCHOS, entre un padre y varios hijos, en otras palabras, cada hijo solo tiene un padre.

En este tipo de Base de datos la búsqueda de registros padre puede llevarnos a través de registros hijos y viceversa. En un buen sistema jerárquico, la actualización de un padre automáticamente actualiza los archivos hijos.

2.3.2 Modelo de Redes.

La Base de Datos de Red, a diferencia de las jerárquicas, permite cualquier conexión entre entidades, es decir, se pueden representar relaciones de muchos a muchos. A esta estructura se le conoce como Red. En una Red, un hijo puede tener varios padres y varios hijos a su vez, como se muestra en la siguiente figura:



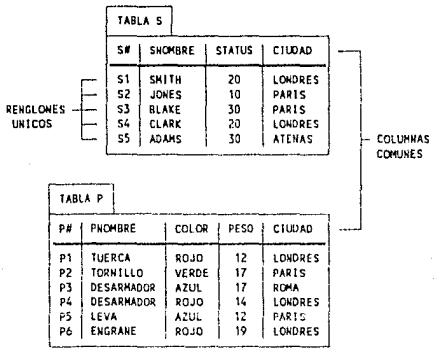
Modelo de Redes

2.3.3 Modelo Relacional.

La estructura lógica de una Base de Datos Relacional está basada en la representación de entidades mediante tablas, las cuales constan de columnas (campos) y renglones (registros). Las relaciones entre tablas se llevan a cabo a través de un conjunto de columnas que se tengan en común, logrando una conexión dinámica entre un número ilimitado de ellas a través del contenido de esas columnas.

La ventaja de los sistemas relacionales es el poder modificar la información sin la preocupación de especificar las combinaciones entre registros.

A continuación se muestra la estructura de una tabla relacional:



Modelo Relacional

2.3.4 Modelo de Formato Libre.

Este modelo permite alimentar información en cualquier formato: texto, tablas o números, y consultarla por medio de llaves de búsqueda. Por ejemplo: Se podría redactar un párrafo en referencia a los contactos de algún negocio y definir las llaves de búsqueda "contacto", "compras" y posteriormente realizar la consulta del párrafo pidiendo información sobre los contactos o compras.

Se puede considerar el modelo de formato libre como relacional, con un formato de registro el cual consiste de dos campos, el párrafo y la llave de búsqueda[5].

3 PROCESO DE DISEÑO DE UNA BASE DE DATOS

En el capítulo 1 se mencionó la importancia que tiene la información para las organizaciones, así como algunas de las alternativas para manipularla.

El diseño de una Base de Datos se encuentra inmerso dentro del proceso de desarrollo de un sistema. Dicho proceso, pretende la creación de sistemas que produzcan información confiable y oportuna para las organizaciones. Se ha identificado una secuencia de pasos general dentro del proceso de desarrollo[U,P], la cual se enuncia a continuación:

1) Especificación de requerimientos
2) Evaluación de alternativas
3) Diseño
4) Implantación

En el primer paso se identifican las necesidades y problemas por resolver, así como las posibles alternativas de solución.

Durante el segundo paso, se analizan las alternativas de solución existentes y en base a ciertos criterios de evaluación, tales como la relación costo/beneficio, se elige la más adecuada. Es aquí donde la Base de Datos aparece como una posible solución. Asumiendo que se elige una Base de Datos entonces en el paso de diseño se identifica y define la estructura de la misma, los procedimientos de operación y las especificaciones de programas de aplicación.

En el último paso, se efectúa la implantación del sistema, que involucra la instalación del Hardware, la codificación y pruebas del Software así como procesos de documentación y entrenamiento a usuarios finales.

Dentro de este proceso, el diseño de la Base de Datos es un paso crítico para alcanzar el éxito en la operación del sistema. El éxito es altamente dependiente de lo preciso de la planeación de la estructura de la Base de Datos. La mayoría de los problemas de funcionalidad y rendimiento de una Base de Datos están asociados a un diseño deficiente[1].

3.1 Etapas del Diseño.

El diseño de una Base de Datos es un proceso iterativo, para cuya realización no existe un algoritmo, por el contrario, este es guiado por la experiencia e intuición de los diseñadores[7].

En base a los requerimientos de información, se realiza un diseño preliminar que es revisado conjuntamente con los usuarios, modificándose según sea necesario para obtener el diseño final que permita satisfacer las necesidades.

El diseño debe contemplar tanto las necesidades de los usuarios, como las del equipo de cómputo y las de los programas de aplicación. Es por esto que el diseño de una Base de Datos se divide en:

- Diseño Lógico y
- Diseño Físico.

El diseño lógico toma como punto de partida los requerimientos obtenidos en el Análisis, estos son estudiados para poder identificar y especificar formalmente los registros de la Base de Datos, su contenido, sus restricciones y, sobre todo, identificar la relación entre registros, que es la esencia del concepto de Base de Datos.

Como resultado se obtiene el esquema lógico de la Base de Datos que es independiente tanto del equipo de cómputo como del DBMS que se utilicen. El esquema lógico refleja de manera formal la estructura de la Base de Datos.

El diseño físico toma el esquema lógico como entrada y lo define físicamente utilizando las herramientas propias de un DBMS específico (como el DDL mencionado en el capítulo anterior), realizando las adecuaciones necesarias para que la Base de Datos física a través de programas de aplicación o del DML represente fielmente ante los usuarios la estructura lógica.

El diseño físico es totalmente dependiente del DBMS y equipo de cómputo que se utilicen, sin embargo cualquier diseño lógico puede implantarse bajo cualquier DBMS.

3.2 Diseño Lógico.

Existen diversas técnicas para efectuar el diseño lógico de una Base de Datos, que varían desde las que son completamente intuitivas hasta las que involucran procedimientos específicos para el procesamiento del diccionario de datos[1]. Sin embargo en vez de mencionar una técnica específica, creemos de mayor importancia mencionar los puntos críticos que contemplan los elementos necesarios para llegar a obtener un diseño completo.

En el diseño lógico se pretende representar aspectos que son de interés para alguna organización.

Durante el proceso de diseño es necesario tener presente las siguientes consideraciones:

- El nivel de detalle de la representación, que debe ir de acuerdo con los requerimientos y recursos disponibles.

- El marco dinámico bajo el que se encuentran las organizaciones y que por tanto repercute directamente en su representación.

- En que forma afectarán los eventos que ocurren en el mundo real y como se reflejarán en el sistema.

- Los diferentes tipos de usuarios que existen y conllevan a las diferentes interpretaciones que cada uno de ellos da a los datos.

Estas consideraciones son importantes ya que permiten mantener la visión global del sistema, puesto que el diseño lógico es abstracto y se puede perder la objetividad.

Como se mencionó anteriormente, el diseño lógico obtiene como entrada los resultados del Análisis de Requerimientos, los cuales se pueden reportar de diversas maneras:

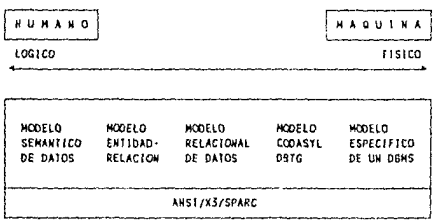
- Diagramas de flujo de información

- Narrativas o

- Diccionario de datos, etc.

El diseñador debe tomar la información contenida en el reporte del análisis y extraer de ahí los conceptos fundamentales para generar el esquema lógico de la Base de Datos. Para lograr esto, existen diversos modelos de datos, los cuales, tienen

diferente orientación, algunos se orientan a la máquina a utilizar y otros a la interpretación humana.



Orientación de seis importantes modelos de datos.

Tomando en cuenta que el objetivo principal del diseño lógico es representar la realidad, y esta toma significado con la interpretación humana, se ha considerado que el modelo semántico de datos es el más adecuado para identificar y explicar los conceptos del diseño (los cuales son aplicables a cualquier modelo pero con nombres diferentes)[6]. Estos conceptos son los siguientes:

- Objetos.

  Un objeto es cualquier cosa, lugar o evento que puede ser representado mediante un sustantivo, por ejemplo, una persona es representada por su nombre.

- Propiedades de los objetos y sus valores.

  Los objetos tienen ciertas características que los diferencian de los demás. Al representar estas propiedades se identifican un conjunto de valores que una propiedad puede tener en un determinado momento, es decir, se identifica el conjunto de valores de las propiedades, por ejemplo, una persona tiene sexo y sus valores son masculino o femenino.

- Clase de objetos.

Los objetos se pueden agrupar al considerar las propiedades que tienen en común y discriminar las diferentes para formar una sola clase de objetos. Al agrupar objetos decimos que generalizamos, por ejemplo, EMPLEADO es toda persona que tiene sexo y empleo.

- Hechos

  Es cuando una propiedad de algún objeto toma un valor específico de su conjunto de valores, por ejemplo, objeto: persona, propiedades: nombre y edad, hecho: Juan tiene 27 años.

- Asociaciones entre objetos.

  Es la relación que existe entre objetos, ya sean de la misma clase o no, por ejemplo, Empleado y Departamento o Empleado y Jefe.

El diseñador analiza los objetos, comunmente conocidos como entidades, sus propiedades o atributos y sus asociaciones o relaciones. Para representar formalmente dichos conceptos se pueden utilizar varias técnicas como el modelo Entidad Relación o el modelo Semántico, etc.. La parte realmente importantes que se identifiquen los objetos, sus propiedades y sus relaciones.

Una vez representadas formalmente las entidades y sus relaciones a nivel conceptual podemos iniciar el diseño, siguiendo el siguiente proceso.

1) Identificación de datos por almacenar.

   Identificación de datos significa identificar las unidades mínimas de información. El diseñador estudia las propiedades de las entidades para determinar cuales son relevantes para la Base de Datos. Estas formarán así los datos elementales que contendrá la base de datos.

2) Definición y estandarización de nombres para los datos elementales.

   Una vez identificados los datos elementales, es necesario

definir los términos que se utilizarán para cada dato elemental.

Como se mencionó, la diferencia entre usuarios provoca una diferencia en la terminología y en la interpretación. Se puede hacer referencia al mismo dato bajo diferentes términos (se identifican sinónimos) o bien, se puede utilizar el mismo término para representar diferentes datos elementales.

Es tarea del diseñador el tratar, en la medida de lo posible, de estandarizar los términos que representarán a los datos elementales para mantener consistencia entre ellos.

La identificación de sinónimos no es una tarea fácil, sobre todo cuando se habla de cientos de datos elementales. Por ejemplo, el dato elemental FECHA, que puede ser FECHA de ingreso de un empleado, FECHA de compra, FECHA de pago, etc.. El diseñador debe definir si se trata del mismo dato elemental o si para efectos del sistema significan eventos diferentes, de ser así, se requerirán de nuevos nombres para referenciar a cada uno de ellos.

3) Desarrollo del Esquema Lógico.

El desarrollo del esquema lógico de una Base de Datos consiste en definir los registros y sus relaciones. Los registros se definen al determinar que datos elementales contendrán. Al identificar entidades, generalmente se traducen de manera intuitiva, en registros. Según Kroenque[7] la identificación de registros es obvia para el 60 ó 75% de los casos. El diseñador infiere del Análisis de Requerimientos algunos registros que deben existir.

Sin embargo, para ciertas entidades es difícil distinguir si deben estar separadas, o bien, deben formar un sólo registro. Esto se debe a que las clases de entidades no son fácilmente identificadas.

La normalización , aunque surgió del concepto relacional de Bases de Datos[6], es una herramienta de diseño importante, para efectos de asignación de datos elementales a registros[C,E].

De las asociaciones entre entidades, que fueron identi-

ficadas, se definen las relaciones potenciales entre registros. El diseñador debe analizar el ambiente deseado del sistema, las necesidades de los diferentes usuarios para poder discriminar entre las relaciones teóricas y las realmente útiles. Una relación teórica existe lógicamente, pero en la práctica no proporciona información relevante.

Cuando es difícil discernir si la relación puede o no ser útil, es recomendable incluirla dentro del esquema de la Base de Datos. Siempre se pueden omitir relaciones en el diseño físico, pero agregarlas puede traer serias complicaciones.

La determinación de registros y tipos de relaciones es un ciclo iterativo, mientras se definen registros, se descubren posibles relaciones y al identificar relaciones se pueden crear nuevos registros. Al analizar conceptualmente entidades, se infieren registros y al encontrar nuevas relaciones de definen nuevas entidades.

Durante este proceso se identifican limitaciones o restricciones que tendrá la Base de Datos. Estas restricciones generalmente se dividen en tres grupos:

a) Restricciones de datos elementales.

b) Restricciones entre datos elementales dentro de un mismo registro.

c) Restricciones entre datos elementales de diferentes registros.

Estas restricciones se consideran para mantener la integridad de la Base de Datos. Una restricción del tipo a) restringe los valores para cierto dato elemental. Las restricciones del tipo b) indican que existe cierta dependencia entre datos elementales o propiedades de una misma entidad o registro. Las restricciones entre elementos de diferentes registros delimitan las relaciones entre estos. Obteniendo con todo esto las reglas de integridad de la Base de Datos.

4) Procesamiento.

La definición del procesamiento establece la forma en que se deberá manipular la Base de Datos para producir los resultados esperados. El diseñador puede definir el

procesamiento describiendo las transacciones y los datos que éstas modifican. Otro método es el realizar diagramas funcionales representando entradas, salidas y funciones específicas de cada programa de aplicación[2].

Para algunos autores la definición del procesamiento de la base de datos no forma parte del diseño. Sin embargo, una descripción lógica del procesamiento ayuda a identificar flujos propios del diseño lógico.

El diseño concurrente tanto de programas como de la Base de Datos mejora el diseño de la base de Datos y la calidad de los programas[1].

5) Revisión del Diseño.

Una vez obtenido el esquema lógico de la Base de Datos, es revisado con el propósito de identificar omisiones o mal interpretaciones de los datos. La revisión involucra los requerimientos, a los diseñadores y a los usuarios conjuntamente, hasta obtener el diseño que prometa responder a las necesidades planteadas.

A lo largo de este capítulo, se ha enfatizado que el diseño lógico es un proceso iterativo, intuitivo y que es difícil de seguir una secuencia rígida de pasos para realizarlo. Esto se debe, en parte, a que al tratar de representar un conjunto de ideas intervienen factores humanos, como el criterio de las personas, su interpretación, los cuales son impredecibles y no se tienen bajo control.

El diseño es una tarea difícil por lo que se han desarrollado una serie de productos para asistirlo como SEGSI "Systéme Expert en Conception de Systémes d'Information"[6] e IDMS/Architect[K].

La normalización de relaciones es una herramienta del diseño por lo que el presente trabajo pretende generar un paquete normalizador que asista a los diseñadores en el proceso de asignación de elementos a las entidades.

Dado que la normalización tiene sus fundamentos en el modelo relacional, consideramos conveniente cubrir los conceptos que giran alrededor del mismo en los siguientes capítulos.

4 MODELO RELACIONAL

4.1 Conceptos Relacionales.

El Modelo Relacional es el punto intermedio entre los modelos orientados al humano y los orientados a la máquina a utilizar, ya que tiene características tanto físicas como lógicas.

El Modelo Relacional es lógico ya que los datos se presentan en un formato familiar a los humanos; así mismo, no contempla la representación física de los datos. Por otro lado, este modelo es más físico que el semántico y que el de entidad relación, ya que el esquema lógico diseñado bajo este modelo no requiere ser transformado en ningún otro formato dado que existen DBMS's relacionales.

El formato utilizado por el modelo relacional es el archivo plano que recibe el nombre de tabla, la cual, tiene renglones que representan registros y columnas que representan los campos del registro.

La verdadera importancia del modelo relacional no radica en el formato utilizado sino en que las relaciones entre registros se dan a través del valor de los datos.

Para poder hablar de la modelo relacional debemos conocer algunos términos que nos servirán para diferenciar las partes de dicha estructura:

- Atributo.

  Se refiere a una columna de una relación, la cual agrupa datos con el mismo significado semántico.

- Tuplo.

  Corresponde a un renglón de una relación el cual puede contener más de un atributo.

- Dominio.

  Para poder explicar el concepto de dominio, hay que partir de la siguiente premisa:

  "La unidad mínima de información en el modelo relacional,

es el valor de un atributo y este valor es atómico, es decir, no se puede dividir"[M].

Entonces, dominio es el conjunto de todos los valores posibles de un atributo. Existen dos tipos de dominios: Simples y Compuestos. Los dominios Simples son aquellos que son indivisibles, por ejemplo, los números enteros. Los dominios Compuestos son aquellos que se forman de dos o más dominios simples, como por ejemplo, las fechas (año,mes, día). Entonces el dominio compuesto se forma al realizar el producto cartesiano de los dominios simples (año, mes, día, en ese orden), aunque no todos los valores sean válidos.

- Relación.

Una Relación es lo que comunmente se conoce como tabla, y ésta consta de dos partes principales, un encabezado y un cuerpo. El encabezado consta de un conjunto finito de atributos y a cada atributo le corresponde un dominio (aunque no todos los dominios necesariamente son distintos). El cuerpo consiste de un conjunto de tuplos, el cual varía con el tiempo. Entonces, cada tuplo está formado por un conjunto finito de atributos apareados con un elemento del Dominio correspondiente. El número de atributos y de tuplos con los que cuenta una relación, nos indican el grado y la cardinalidad respectivamente de dicha relación.La cardinalidad varía con el tiempo y el grado siempre es fijo. Las relaciones tienen ciertas propiedades que son consecuencia directa de la definición, éstas son:

1) No existen tuplos duplicados.

   El cuerpo de la relación es un conjunto y en Teoría de Conjunto, estos, por definición no tienen elementos duplicados, por lo que se obtiene el siguiente corolario:

   "Siempre existe un identificador único (la llave primaria) aunque éste contenga todos los atributos".

2) Los tuplos no están ordenados.
   El cuerpo de una relación es un conjunto de tuplos y dado que en Teoría de Conjuntos el orden no tiene ningún significado, se cumple la propiedad.

3) Los atributos no están ordenados.

Al igual que en el punto anterior el orden de los atributos no tiene ningún significado ya que el encabezado de una relación es un conjunto de atributos.

4) El valor del atributo es atómico.

Esto se deriva del siguiente hecho: un atributo está relacionado directamente con un dominio y éstos tienen valores indivisibles y aunque se piense en Dominios Compuestos, estos últimos únicamente son concatenaciones de los primeros.

- Base de Datos Relacional.

Un usuario puede percibir una Base de datos relacional en Base a lo explicado en los puntos anteriores de la siguiente forma:

"Colección de Relaciones Normalizadas de un grado determinado las cuales varian con el tiempo"[M].

4.2 Caracteristicas de una Base de Datos Relacional.

El modelo Relacional provee un conjunto de operadores algebráicos, para la manipulación de datos en todas sus manifestaciones.

Un sistema relacional, pueden dar la idea de una inmejorable forma de organizar la información, pero el modelo relacional no es una "panacea", dado que un sistema no es necesariamente "bueno" solo por soportar este modelo.

El modelo relacional es una herramienta, no un fin en si mismo, provee fundamentos que pueden ser usados en cualquier sistema, como el lenguaje ensamblador es el fundamento de todo el software de hoy en dia.

Los sistemas relacionales presentan, entre otras caracteristicas, una muy importante en la cual la estructura de los datos proporcionados en un modelo, está dada en la propia relación.

Existen diferencias esenciales entre los distintos modelos de bases de datos y el relacional. Algunas caracteristicas

importantes del modelo relacional, son las siguientes:

- La relación entre tablas, está dada por el valor de los propios datos, eliminando la necesidad del concepto de ligas o apuntadores.

- Simplicidad en el manejo de información, debido a que ésta se procesa con menos operadores y menor procesamiento colateral.

- Es el primer modelo que introduce conceptos matemáticos para la manipulación y representación de información.

Conociendo todos los aspectos que contempla el modelo relacional, podemos decir que un Sistema de Administración de Base de Datos Relacional (Relational DBMS), es aquél que soporta todos las características de éste, aunque algunos aspectos del modelo relacional son cruciales y otros son simplemente deseables.

Siguiendo a Codd, definimos un Sistema como Relacional sólo si soporta lo siguiente[1]:

- Base de Datos Relacional (manejo de Tablas).

- Los operadores select, project, join (natural), funcionan sin necesidad de especificar rutas de acceso físico para soportarlos.

Los sistemas que soporten la definición de Codd, aunque no sea explícitamente, pueden calificarse como sistemas relacionales.

La justificación de la definición de Codd, puede resumirse en los cuatro puntos siguientes:

1) El Algebra Relacional, tiene una serie de operadores, entre los cuales se encuentran select, project y join, mismos que forman un subconjunto que permite solucionar casi cualquier problema práctico dentro de las estructuras relacionales.
2) Un sistema que soporta la estructura de datos relacional, pero no a los operadores, elimina la facilidad del uso y productividad de un sistema relacional genuino.

3) Un sistema que soporta los operadores relacionales, pero requiere una predefinición física de ruta de acceso para soportarlos, no provee la independencia física de datos de un sistema relacional verdadero.

4) Para hacer un buen trabajo de implantación de los operadores relacionales, al menos en un ambiente con sistemas de gran capacidad, se requiere que el sistema haga optimizaciones. Un sistema que únicamente ejecuta las operaciones requeridas por el usuario, podría ocasionar un bajo rendimiento (performance). De esta manera, el implantar un sistema que haga las operaciones del modelo relacional de manera eficiente, no es una tarea sencilla, por lo que en los productos relacionales es muy importante este punto, el cual, es el reto para mantenerse por largo tiempo en el mercado.

Los aspectos más importantes del modelo relacional los podemos englobar en las siguientes divisiones.

4.2.1 Integridad.

El término integridad en el contexto de bases de datos, significa validación y corrección. Con la integridad se evita que los datos dentro de una Base de datos sean actualizados inválidamente.

Una actualización inválida se puede deber a lo siguente: errores en los datos de entrada, fallas en el diseño de la estructura de las relaciones, errores del operador o del programador del sistema, fallas del sistema e incluso por falsificación deliberada.

Las reglas de integridad se pueden dividir en dos categorías: Reglas de Integridad de Dominio y Reglas de Integridad Relacional.

Las reglas de dominio se refieren a la admisión de un valor dado como candidato a un cierto atributo, independientemente de su relación con otro dentro de la Base de Datos.

Las reglas relacionales se refieren a la admisión de un tuplo como candidato a una tabla determinando la relación existente de los atributos de ese tuplo con los atributos de alguna otra relación y consiste de dos reglas:

- La Integridad de Entidades

- La Integridad Referencial

Estas tienen relación con la llave primaria y las llaves extranjeras respectivamente.

- Llave primaria.

  La llave primaria es un caso especial de lo que se conoce como llaves candidatas. Una llave candidata básicamente es un identificador único, pudiendo estar formado por un conjunto de atributos, donde al conocerlos se puede saber el valor de los demás atributos que forman un tuplo, siendo un identificador del tuplo. Por definición, toda relación tiene cuando menos una llave candidata. En una determinada relación se escoge una de las llaves candidatas como llave primaria y las demas, si hubieran, pasan a ser llaves alternas.

  Sea entonces una relación R con un conjunto de atributos A1..An; estos atributos serán llaves candidatas si y sólo si se cumplen las siguientes propiedades:

  a) Unicidad.

     En ningún momento dos tuplos de una misma relación podrán tener el mismo valor para el atributo Ai (i=1..n), o sea, nunca habrá tuplos iguales.

  b) Minimalidad.

     Ningún atributo Ai (i=1..n) puede eliminarse sin violar la propiedad de unicidad.

  Entonces la llave primaria (seleccionanada arbitrariamente de las llaves candidatas) nos permite hacer referencia a los tuplos de dicha relación.

- Llave extranjera.

  La llave extranjera (que no necesariamente es parte de la llave primaria de la relacion que la contiene), es un atributo o un conjunto de ellos en una determinada relación R2, cuyo valor debe de existir en la Relación R1, en donde dicho conjunto es llave primaria, aunque R1 y R2 no necesariamente son diferentes.

Entonces la relación entre las llaves extranjeras y las llaves primarias son el factor que mantiene unida la base de datos mediante la relación entre tuplos.

Es en este momento que podemos plantear las reglas de integridad:

a) Integridad de Entidad.

Ningún atributo que forme parte de la llave primaria de alguna relación, podrá aceptar valores nulos.

b) Integridad Referencial.

Si la Relación R2 incluye una llave extranjera referenciando la llave primaria de una relación R1, entonces cada valor de la llave extranjera tiene que:

1) Ser igual al valor de la llave primaria en un tuplo de R1 ó

2) Ser totalmente nulo (entiéndase por valor nulo alguno fuera del dominio definido).

En el mundo real, cualquier entidad es distinguible, es decir, tiene una identificación única de cualquier tipo, es por eso, que la llave primaria realiza la función de identificador único en el modelo relacional, por lo tanto, no puede tener un valor nulo dado que como llave primaria sería una contradicción, se diría, "Hay una entidad que no tiene identidad", por lo tanto, no existe. De ahí el nombre de integridad de entidad.

El objetivo de la integridad referencial es verificar que el valor de la llave extranjera de alguna relación exista como llave primaria en otra, siempre y cuando la llave extranjera sea no nula.

En un estado dado puede suceder que no se satisfaga alguna de las dos reglas, por lo que ese dato es incorrecto.

Para esto debe existir un subsistema de integridad que se encargue de vefificar que las actualizaciones sean válidas y corregir las que no lo sean.

Consideremos la existencia de un subsistema de integridad como un componente de un DBMS con las siguientes responsabilidades:

- Monitorear transacciones u operaciones de actualización y detectar violaciones a la integridad.

- En caso de violación, tomar la acción apropiada como puede ser: rechazar la operación, reportar la violación, corregir el error, etc.

Para que el subsistema de integridad sea capaz de realizar estas funciones, se le tiene que proveer de un conjunto de reglas que definan qué errores revisar, cuándo realizar la verificación y qué hacer en caso de detectar un error.

Un ejemplo de una regla de este tipo sería:

Regla de integridad del proveedor #1 (RIP1)

```
RIP1 :
       DESPUES DE ACTUALIZAR S.STATUS:  ----------> 1
       S : STATUS > 0                   ----------> 2
       DE LO CONTRARIO
          REGRESA CODIGO "REGLA RIP1 VIOLADA"
          ELIMINA ACTUALIZACION         ----------> 3
       FIN
```

Esta regla consiste de tres partes:

- La primera indica el momento de la validación, que dependiendo del tipo de transacción se efectuará antes o después de actualizar la Base de Datos.

- La segunda parte indica la condición a verificar y
- La tercera parte indica la acción a tomar en caso de que no se cumpliera la condición.

Las reglas de integridad, expresadas en un lenguaje de alto nivel, son compiladas y guardadas en el diccionario del sistema por el compilador de reglas de integridad.

Se puede definir una nueva regla en cualquier momento y será aceptada mientras el estado actual de la base de datos no viole dicha regla.

4.2.2 Manipulación.

4.2.2.1 Algebra relacional.

El álgebra relacional es una opción para la manipulación de datos en el modelo relacional y se puede dividir en dos partes:

- Un conjunto de operadores.
- La operación de asignación.

Cada operador del álgebra relacional utiliza una o dos relaciones como entrada y produce otra nueva como salida. Codd, originalmente, definió ocho operadores que se dividen en dos grupos[J1]:

- Las operaciones tradicionales de conjuntos tales como unión, intersección, diferencia y producto cartesiano extendido (considerando por supuesto, que sus operandos son relaciones y no conjuntos).

- Las operaciones relacionales tales como: selección (select), proyección (project), división (divide) y asociación (join).

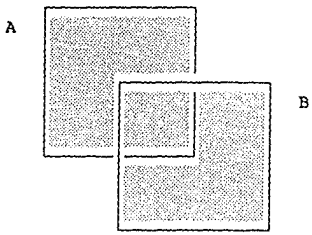
Cabe hacer notar que el resultado de cada operación algebráica, es una relación, gracias a la propiedad de Cerradura que permite generar expresiones anidadas, es decir, hacer la proyección de una unión, o el join de dos selecciones, etc.

Todas estas operaciones utilizan 2 operandos. Los dos operandos deben ser compatibles, (excepto en el producto cartesiano), es decir, deben ser del mismo grado y el iésimo atributo de cada relación debe estar basado en el mismo dominio. Esta compatibilidad es importante para que se cumpla la propiedad de cerradura y el resultado siga siendo una relación.

A continuación explicaremos más detalladamente las ocho operaciones algebráicas.

- UNION.

  La unión de dos relaciones A y B ,A UNION B, es elconjunto de tuplos que pertenecen a la relación A, a la relación B o a ambas.



A UNION B

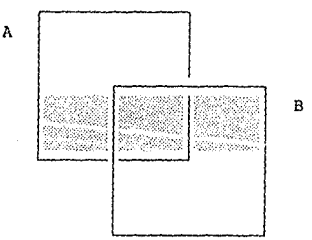
Ej.:
Sea A el conjunto de proveedores en Londres.
Sea B el conjunto de proveedores de la parte P1.

entonces,

A UNION B es el conjunto de proveedores que están en Londres o que proveen la parte P1 o ambas cosas.

- INTERSECCION.

La intersección de dos relaciones A y B, A INTERSECCION B, son todos los tuplos que pertenecentanto a la relación A como a la relación B.



A INTERSECCION B

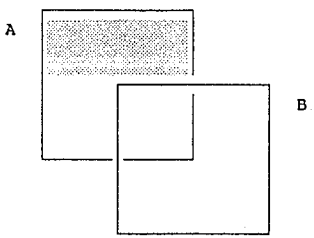
Ej.:
Sea A y B igual que el ejemplo de la unión.

entonces,

A INTERSECCION B, es el conjunto de tuplos de proveedores que se localizan en Londres y proveen la parte P1.

- DIFERENCIA.

La diferencia entre dos relaciones A y B, A MENOS B, es el conjunto de tuplos que pertenecen a la relación A y no a la relación B.



A MENOS B

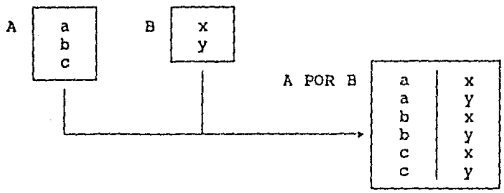
Ej.:
Sea A y B igual que el ejemplo de la unión.

entonces,

A MENOS B es el conjunto de proveedores que se localizan en Londres y no proveen la parte P1.

- PRODUCTO CARTESIANO EXTENDIDO.

El producto cartesiano extendido de dos relaciones A y B, A POR B, es el conjunto de tuplos t, tal que t es la concatenación de tuplo a, que pertenece a la relación A y un tuplo b, que pertenece a la relación B, para cada tuplo de A, con todos los tuplos de B.



PRODUCTO

Ej.:

Sea A el conjunto de Número de proveedor.
Sea B el conjunto de Número de parte.

entonces,

A POR B, es el conjunto de pares, número de proveedor/número de parte, en ese orden y para todo tuplo de A.

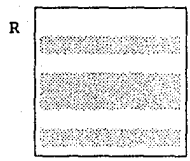
- SELECT.

Sea alfa cualquier operador para comparar valores escalares entonces la selección sobre la relación R y los atributos X y Y será:

R DONDE R.X alfa R.Y

Esto dá como resultado el conjunto de tuplos t que pertenecen a R tal que la comparación X alfa Y evaluada resulta verdadera, es importante notar que tanto X y Y deben ser definidos sobre el mismo dominio y la comparación alfa debe ser coherente con dicho dominio.

Por lo tanto, la operación de comparación alfa genera un subconjunto horizontal de dicha relación, esto es, el subjconjunto de tuplos de la relación dada que satisfacieron la comparación.



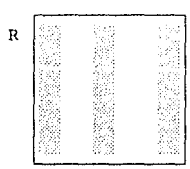
SELECT

- PROJECT.

  La proyección de la relación R sobre los atributos X, Y...Z

  R [X, Y.. Z]

  es el conjunto de tuplos (x, y.. z) tal que el tuplo t aparece en la relación R con X-valor-x, Y-valor-y....Z-valor-z, por lo tanto, la operación de proyección genera un subconjunto vertical de la relación dada, esto es, el subconjunto obtenido de la selección de atributos específicos en un orden específico y de donde se eliminan tuplos redundantes de los atributos seleccionados.



PROJECT

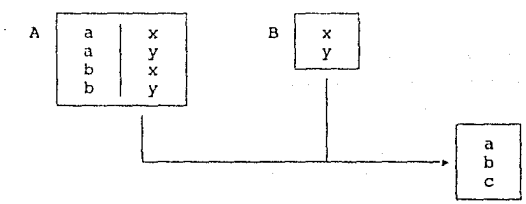
- DIVISION.

  Esta operación toma como dividendo una relación A de grado (m + n) y como divisor una relación B de grado n y produce una relación como cociente de grado m. Tomando en cuenta que el (m + i)ésimo atributo de A y el iésimo atributo de B (i = 1..n) deben estar definidos en el mismo dominio.

Considerando los primeros m atributos de A como un atributo cualquiera X y el último n como otro Y, tenemos que la relación A se puede considerar como un conjunto de parejas ordenadas de valores (x,y). De la misma forma, la relación B puede ser un conjunto de valores (y). Entonces, el resultado de dividir A entre B, esto es, evaluar la expresión:

A DIVIDE BY B,

es la relación C con un solo atributo X tal que todo valor x de C.X aparece como valor de A.X y el par de valores (x,y) aparecen en A para todo valor "y" que aparece en B. Los m atributos del cociente tienen el mismo nombre que los primeros m atributos de la relación B.



DIVIDE

- JOIN.

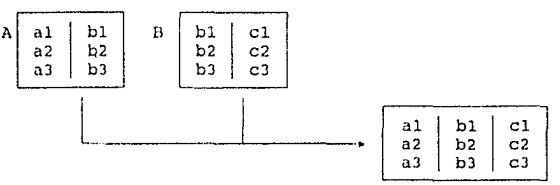
Sea θ (teta) cualquier operador para comparar valores escalares. El Join de la Relación A sobre el atributo X con la relación B sobre el atributo Y aplicando θ es el conjunto de todos los tuplos t, tal que t es la concatenación del tuplo a de la Relación A y el tuplo b de la Relación B; y el predicado "a.x θ b.y" resulta verdadero (los atributos A.X, B.Y, deben pertenecer al mismo dominio y la operación θ debe ser congruente al dominio).

Teta-Join no es una operación primitiva, por lo que es equivalente obtener el producto cartesiano extendido con la condición adecuada.

Si el operador θ es la igualdad teta-join, recibe el

nombre de "Equijoin". Partiendo del postulado que dice: "El resultado de un equijoin debe tener dos atributos idénticos"[M], si uno de esos dos atributos se elimina (mediante una proyección) el resultado recibe el nombre de join natural; entonces el Join natural es la proyección de la restricción de un producto, por lo que el Join natural se convierte en el operador algebráico más importante y más usado.

Entonces A JOIN B se define para cualquier atributo común a la relación A y a la relación B, siempre y cuando el dominio de ambos atributos sea el mismo.



JOIN (natural)

4.2.2.2 Cálculo relacional.

El Cálculo Relacional representa una alternativa para el álgebra relacional como parte manipuladora del modelo relacional. La diferencia entre el álgebra y el cálculo relacional radica en lo siguiente: El álgebra provee un conjunto de operaciones explícitas, tales como el Join, Union, Project, etc. que pueden ser utilizadas para construir determinadas relaciones de las ya existentes en la Base de datos mientras que el cálculo únicamente provee una notación para formular la definición de las relaciones deseadas en términos de las relaciones dadas.

Superficialmente, cuando menos se puede decir que las formulaciones del cálculo relacional son descriptivas y las algebráicas son prescriptivas. El cálculo simplemente plantea cual es el problema y el álgebra provee un procedimiento para resolverlo, en otras palabras, el álgebra es procedural y el cálculo no procedural.

De todas maneras quede claro que las distinciones son superficiales, por lo que es valido decir que el álgebra y el cálculo son equivalentes el uno con el otro, para cada expresión algebraica existe una expresión equivalente en el cálculo relacional y viceversa. Entonces, existe una correspondencia biunivoca entre el cálculo y el álgebra relacional.

El cálculo relacional se fundamenta en una rama de la Lógica Matemática llamada "Cálculo de Predicado". La idea de utilizar el cálculo de predicado como base al lenguaje de manipulación de datos fue originada por Kuhns y el concepto del cálculo relacional fue propuesto por Codd[1].

La notación que definió Codd (sublenguaje Alpha) se muestra a continuación:

Símbolo	Significado
R.D	La relación unaria constituida por el conjunto de valores de datos del dominio D de la relación R.
X(R1.D1,R2.D2...	Una relación llamada X compuesta de los dominios constituidos por los conjuntos de valores R1.D1,R2.D2...
EXIST	Cuantificador existencial: "existe".
FORALL	Cuantificador universal: "para todo".
:	Tal que. La expresión a la izquierda de los dos puntos especifica lo que deberá obtenerse del banco de datos, y la expresión a la derecha es el predicado o calificativo.
AND,OR,NOT	Los operadores lógicos estandar Y, O, NO para formar calificativos.
=,=/,<,<=,>,>=	Los operadores de comparación estandar: igual a, diferente de, menor que, menor o igual que, mayor que y mayor o igual que.

Una consulta bajo la notación del Cálculo Relacional consta de dos partes: Un objeto (relación destino) que

indica los dominios que se desean obtener y una parte calificadora, que selecciona los tuplos especificos de la relación destino, dadas las condiciones que se deben satisfacer. por ejemplo:

```
X(Empleado.nombre,Empleado.edad) :
            Empleado.experto_en = "Contabilidad" AND
            Empleado.localidad  = "Sinaloa"
```

donde

X(Empleado.nombre, Empleado.edad) es la relación destino que otiene los dominios nombre y edad y Empleado.experto_en = "Contabilidad" AND Empleado.localidad = "Sinaloa" es la parte calificadora que indica la selección.

Un aspecto fundamental del cálculo es el concepto de variables de tuplo "Tuple Variable" también conocidas como variables de rango. Este tipo de variables toman su valor a partir de un rango dentro de una relación, es decir, los únicos valores permitidos son tuplos de la relación, en otras palabras si la variable de tuplo T ejerce su rango sobre la relación R entonces, en cualquier momento T representa algún tuplo t de R.

Por la relación que tiene el cálculo relacional con las variables de tuplo, éste ha venido a conocerse como cálculo de tuplos. Lacroix y Pirrote[M] propusieron una alternativa al cálculo relacional la cual llamaron "Cálculo de Dominios".

Dentro del cálculo relacional las variables de tuplo se definen de la siguiente forma:

```
RANGE OF T IS X1;X2;.....;Xn
```

donde T es la variable de tuplo y X1; X2; ...; Xn son expresiones que representan relaciones R1, R2, ...,Rn. Estas relaciones deben ser compatibles para la unión y los correspondientes atributos deben nombrarse igual en cada relación. La variable de tuplo T actúa sobre la unión de dichas relaciones.

Cada ocurrencia de una variable de tuplo con una WFF ("Well formed formula") puede ser libre o limitada. Por "Ocurrencia de una variable de tuplo" queremos decir la

aparición de la variable dentro de la WFF. Una variable de tuplo T ocurre dentro de una WFF, tanto en el contexto de la referencia del atributo (T.A donde A es un atributo de la relación donde T actúa) como en la variable que sigue después de algún cuantificador.

Analizando la siguiente expresion:

```
EXIST x ( x > 3 )
```

donde x actúa sobre el conjunto de los números enteros. La variable limitada x en esta WFF es dummy, únicamente sirve para unir el cuantificador con la expresión del paréntesis. La WFF indica que existe algún número entero mayor a tres. El significado de la expresión permanecerá igual aunque se cambie la variable x por y, entonces la siguiente expresión es semánticamente igual a la anterior:

```
EXIST y ( y > 3 ).
```

Ahora consideremos la siguiente expresión:

```
EXIST x ( x > 3 ) AND x < 0
```

Aquí existen tres ocurrencias de x referenciando dos variables diferentes. Las dos primeras ocurrencias son limitadas y podrán ser cambiadas por otra variable sin alterar el significado de la expresion, la tercera ocurrencia es libre y no puede ser cambiada. Entonces, de las dos WFF's que se muestran a continuación, la primera es equivalente y la segunda no lo es:

```
EXIST y ( y > 3 ) AND x < 0

EXIST y ( y > 3 ) AND y < 0
```

Entonces una expresión del cálculo de tuplos es de la forma:

```
T.A,U.B.....,V.C WHERE F
```

donde T,U,....,V son variables de tuplo, A,B,....,C son los atributos de las relaciones asociadas y F es un WFF conteniendo a T,U,.....,V como variables libres.

El valor de esta expresión es la proyección del subconjunto generado del producto cartesiano extendido T x U x ..... x V, cuando F es verdadero.

4.2.2.3 Algebra relacional vs Cálculo relacional.

El álgebra relacional y el cálculo relacional son equivalentes. Codd probó que el álgebra es tan poderosa como el cálculo, esto lo hizo mediante el algoritmo "Codd Reduction Algorithm" por medio del cual, cualquier expresión del cálculo se puede reducir a su equivalente semántico del álgebra relacional (E.F Codd "Relational Completeness of Data Sublanguaje").

Para probar que el cálculo relacional y el álgebra relacional son semánticamente equivalentes, presentamos a continuación un ejemplo del funcionamiento de dicho algoritmo.

Como Base de datos para este ejemplo utilizaremos las siguientes tablas de datos:

TABLA S proveedores

S#	SNOMBRE	STATUS	CIUDAD
S1	SMITH	20	LONDRES
S2	JONES	10	PARIS
S3	BLAKE	30	PARIS
S4	CLARK	20	LONDRES
S5	ADAMS	30	ATENAS

TABLA P partes

P#	PNOMBRE	COLOR	PESO	CIUDAD
P1	TUERCA	ROJO	12	LONDRES
P2	TORNILLO	VERDE	17	PARIS
P3	DESARMADOR	AZUL	17	ROMA
P4	DESARMADOR	ROJO	14	LONDRES
P5	LEVA	AZUL	12	PARIS
P6	ENGRANE	ROJO	19	LONDRES

TABLA J proyecto

J#	JNOMBRE	CIUDAD
J1	CLASIFICADORA	PARIS
J2	PERFORADORA	ROMA
J3	LECTORA	ATENAS
J4	CONSOLA	ATENAS
J5	INTERCALADORA	LONDRES
J6	TERMINAL	OSLO
J7	UNIDAD DE CINTAS	LONDRES

TABLA SPJ pedido

S#	P#	J#	CANTIDAD
S1	P1	J1	200
S1	P1	J4	700
S2	P3	J1	400
S2	P3	J2	200
S2	P3	J3	200
S2	P3	J4	500
S2	P3	J5	600
S2	P3	J6	400
S2	P3	J7	800
S2	P5	J2	100
S3	P3	J1	200
S3	P4	J2	500
S4	P6	J3	300
S4	P6	J7	300
S5	P2	J2	200
S5	P2	J4	100
S5	P5	J5	500
S5	P5	J7	100
S5	P6	J2	200
S5	P1	J4	100
S5	P3	J4	200
S5	P4	J4	800
S5	P5	J4	400
S5	P6	J4	500

Consideremos la siguiente consulta:

"DAME NOMBRE Y CIUDAD DE LOS PROVEEDORES QUE PROVEEN CUANDO MENOS UN PROYECTO EN ATENAS CON CUANDO MENOS 50 DE CADA PARTE"
La expresión en cálculo relacional para la consulta anterior se expresaría de la siguiente forma:

```
SX.NOMBRE SX.CIUDAD WHERE EXISTS JX FORALL
                            PX EXISTS SPJX
                 (JX.CIUDAD      = "ATENAS" AND
                  JX.J#          = SPJX.J#  AND
                  PX.P#          = SPJX.P#  AND
                  SX.S#          = SPJX.S#  AND
                  SPJX.CANTIDAD > 50)
```

donde SX, PX, JX, SPJX son variables de tuplo que actúan o ejercen su rango sobre S, P, J, SPJ, respectivamente; ahora explicaremos paso a paso como la expresión anterior sería evaluada con operadores del álgebra relacional para obtener el resultado esperado.

Paso 1: Para cada variable de tuplo, consultar el rango (conjunto de posibles valores para esa variable) en forma restringida. Con la frase "en forma restringida" queremos decir que puede haber alguna restricción intrínseca dentro de la cláusula WHERE que pueda utilizarse inmediatamente para eliminar ciertos tuplos; en este caso el conjunto de tuplos se reduce a lo siguiente:

VARIABLE DE TUPLO	RANGO	CANTIDAD
SX	Todos los tuplos de S	5
PX	Todos los tuplos de P	6
JX	Tuplos de J donde ciudad sea igual a "ATENAS"	2
SPJX	Tuplos de SPJ donde cantidad sea mayor que 50	24

Paso 1

Paso 2: Obtenemos el producto cartesiano de los rangos seleccionados del cual obtenemos la siguiente tabla, que se presenta en forma resumida ya que consta de 1440 renglones:

S#	SN	STA	CIU	P#	PN	COLOR	PES	CIU	J#	JN	CIU	S#	P#	J#	CANT
S1	SM	20	LON	P1	TU	ROJO	12	LON	J3	LE	ATE	S1	P1	J1	200
S1	SM	20	LON	P1	TU	ROJO	12	LON	J3	LE	ATE	S1	P1	J4	700
S1	SM	20	LON	P1	TU	ROJO	12	LON	J3	LE	ATE	S1	P3	J1	400
S1	SM	20	LON	P1	TU	ROJO	12	LON	J4	CO	ATE	S1	P1	J4	700
S2	JO	10	PAR	P3	DE	AZUL	17	ROM	J3	LE	ATE	S2	P3	J3	200
S2	JO	10	PAR	P3	DE	AZUL	17	ROM	J4	CO	ATE	S2	P3	J4	500
S4	CL	20	LON	P6	EN	ROJO	19	LON	J3	LE	ATE	S4	P6	J3	300
S5	AD	30	ATE	P2	TO	VERDE	17	PAR	J4	CO	ATE	S5	P2	J4	100
S5	AD	30	ATE	P1	TU	ROJO	12	LON	J4	CO	ATE	S5	P1	J4	100
S5	AD	30	ATE	P3	DE	AZUL	17	ROM	J4	CO	ATE	S5	P3	J4	200
S5	AD	30	ATE	P4	DE	ROJO	14	LON	J4	CO	ATE	S5	P4	J4	800
S5	AD	30	ATE	P5	LE	AZUL	12	PAR	J4	CO	ATE	S5	P5	J4	400
S5	AD	30	ATE	P6	EN	ROJO	19	LON	J4	CO	ATE	S5	P6	J4	500
S5	AD	30	ATE	P6	EN	ROJO	19	LON	J4	CO	ATE	S5	P5	J7	100
S5	AD	30	ATE	P6	EN	ROJO	19	LON	J4	CO	ATE	S5	P6	J2	200
S5	AD	30	ATE	P6	EN	ROJO	19	LON	J4	CO	ATE	S5	P5	J4	500

Paso 2

Paso 3: Restringimos el producto cartesiano recién obtenido acorde a la cláusula WHERE (equivalente a un join), que en este ejemplo es la siguiente:

```
JX.J# = SPJX.J# AND
PX.P# = SPJX.P# AND
SX.S# = SPJX.S#
```

De esta manera eliminamos cualquier tuplo en el cual el valor del proveedor no sea igual al proveedor del pedido y la parte no sea igual a la parte del pedido y el proyecto no sea igual al proyecto del pedido, obteniendo así una tabla con únicamente diez tuplos:

S#	SN	STA	CIU	P#	PN	COLOR	PES	CIU	J#	JN	CIU	S#	P#	J#	CANT
S1	SM	20	LON	P1	TU	ROJO	12	LON	J4	CO	ATE	S1	P1	J4	700
S2	JO	10	PAR	P3	DE	AZUL	17	ROM	J3	LE	ATE	S2	P3	J3	200
S2	JO	10	PAR	P3	DE	AZUL	17	ROM	J4	CO	ATE	S2	P3	J4	500
S4	CL	20	LON	P6	EN	ROJO	19	LON	J3	LE	ATE	S4	P6	J3	300
S5	AD	30	ATE	P2	TO	VERDE	17	PAR	J4	CO	ATE	S5	P2	J4	100
S5	AD	30	ATE	P1	TU	ROJO	12	LON	J4	CO	ATE	S5	P1	J4	100
S5	AD	30	ATE	P3	DE	AZUL	17	ROM	J4	CO	ATE	S5	P3	J4	200
S5	AD	30	ATE	P4	DE	ROJO	14	LON	J4	CO	ATE	S5	P4	J4	800
S5	AD	30	ATE	P5	LE	AZUL	12	PAR	J4	CO	ATE	S5	P5	J4	400
S5	AD	30	ATE	P6	EN	ROJO	19	LON	J4	CO	ATE	S5	P6	J4	500

Paso 3

Paso 4: Aplicamos ahora los cuantificadores de derecha a izquierda, en nuestro ejemplo los cuantificadores son:

EXISTS JX FORALL PX EXISTS SPJX

entonces:

a) Proyectamos los atributos SPJ.S#, SPJ.P#, SPJ.J# y SPJ.CANTIDAD, obteniendo la siguiente tabla:

S#	SN	STA	CIU	P#	PN	COLOR	PES	CIU	J#	JN	CIU
S1	SM	20	LON	P1	TU	ROJO	12	LON	J4	CO	ATE
S2	JO	10	PAR	P3	DE	AZUL	17	ROM	J3	LE	ATE
S2	JO	10	PAR	P3	DE	AZUL	17	ROM	J4	CO	ATE
S4	CL	20	LON	P6	EN	ROJO	19	LON	J3	LE	ATE
S5	AD	30	ATE	P2	TO	VERDE	17	PAR	J4	CO	ATE
S5	AD	30	ATE	P1	TU	ROJO	12	LON	J4	CO	ATE
S5	AD	30	ATE	P3	DE	AZUL	17	ROM	J4	CO	ATE
S5	AD	30	ATE	P4	DE	ROJO	14	LON	J4	CO	ATE
S5	AD	30	ATE	P5	LE	AZUL	12	PAR	J4	CO	ATE
S5	AD	30	ATE	P6	EN	ROJO	19	LON	J4	CO	ATE

Paso 4a

b) Dividimos entre la relación P, y obtenemos:

S#	SNOMBRE	STATUS	CIUDAD	J#	JNOMBRE	CIUDAD
S5	AMAMS	30	ATENAS	J4	CONSOLA	ATENAS

Paso 4b

c) Proyectamos J#, JNOMBRE Y CIUDAD generándose la siguiente tabla:

S#	SNOMBRE	STATUS	CIUDAD
S5	AMAMS	30	ATENAS

Paso 4c

Paso 5: Proyectamos el resultado de la tabla anterior acorde a las especificaciones en la lista de objetivo que en nuestro caso es:

SX.NOMBRE SX.CIUDAD

Obteniendo como resultado la siguiente tabla:

SNOMBRE	CIUDAD
AMAMS	ATENAS

Paso 5

Incidentalmente podemos explicar una de las razones (más no la única) por lo que Codd define precisamente los ocho operadores algebráicos analizados anteriormente: Esos ocho operadores proveen una base conveniente para la posible implementación del cálculo. Pero quizá más importante permiten el análisis de la potencialidad de cualquier lenguaje de Base de Datos.

Se dice que un lenguaje es completamente relacional si es tan poderoso como el cálculo relacional, esto es, si sus expresiones permiten la definición de cualquier relación definible por las expresiones del cálculo relacional. Entonces en base al algoritmo de reducción de Codd el álgebra es completamente relacional.

El cálculo de dominio difiere del cálculo de tuplo en que maneja variables de dominio en vez de variables de tuplo las cuáles actúan sobre dominios y no sobre relaciones.

Lo importante acerca del cálculo de dominio es que soporta una forma adicional de comparación llamada "Membership" que tiene la forma:

R (term1,term2,.....)

En donde R es una relación y cada término es un par de la forma A:v, donde A es un atributo de R y v es una variable de dominio o una constante. Entonces la expresión es verdadera si y solo si existe un tuplo en R que tenga los valores de los atributos especificados.

5 NORMALIZACION

Con el total de datos a ser representados dentro de cualquier organización, surge el problema y la necesidad del diseño de la base de datos, esto es, decidir la estructura lógica para esos datos, decidir qué relaciones son necesarias y qué atributos deben contener.

En el capítulo anterior se explicaron las características del modelo relacional que se orienta tanto al diseño físico como al diseño lógico de la Base de Datos.

La teoría de normalización forma parte importante del modelo relacional ya que toma en cuenta el comportamiento de los datos[N]. Con algunas relaciones al actualizar datos puede tener consecuencias indeseables a las que se les conoce como anomalías de la Base de Datos.

El proceso de normalización tiene como objetivo eliminar dichas anomalías al reorganizar las relaciones en cierta forma normal.

El estudio de las anomalías provee una guía útil para el diseño de una Bases de Datos, pero hay también circunstancias en las cuales es razonable no incluir un proceso de normalización completo, pero el único requerimiento obligado es que la relación se encuentre en primera forma normal (1FN), por ejemplo:

> Cuando se tienen simulaciones financieras que calculan valores de categorías de ingresos y gastos que pueden resultar de la implantación de estrategias financieras o de ventas. Ahora, estas simulaciones contienen muchas dependencias transitivas enforma de columnas de totales, renglones de totales y subtotales, relaciónes, proporciónes e índices financieros, etc. En este momento, no es aún claro ni común que el usuario piense en dividir este modelo en 2 partes, una en donde se contengan los cálculos del detalle de los datos y otra donde se contengan totales y las estadísticas financieras, ésto es, siguiendo una manera común de programación.

Al estucturar los datos bajo el modelo relacional así como al eliminar algunos tipos de anomalías, no se debe perder de vista el efecto que se tendrá en lo que respecta a la eficiencia del procesamiento, los criterios psicológicos y estéticos en la presentación de los datos al usuario.
Por lo anterior es que se considera al modelo relacional como

un punto intermedio en la orientación hombremáquina, ya que facilita la representación lógica de datos así como el análisis de su comportamiento que repercute en su procesamiento.

Sin embargo la teoría de la Normalización no soluciona automáticamente los problemas, pero ayuda la familiaridad que se tenga con ésta en el diseño.

5.1 Anomalías.

Un tipo de anomalías son las anomalías de actualización, es decir, problemas que pueden originarse al añadir, borrar o cambiar tuplos en una relación. Esas anomalías no se originan por la administración relacional de los datos porque los tuplos en un modelo relacional pueden no estar almacenada en forma permanente (store), sino que se originan por la necesidad de responder a consultas formuladas por el usuario.

5.1.1 Anomalías de actualizaciion.

- Anomalías de inserción.

  Para dar de alta un tuplo es necesario especificar todos los atributos que lo identifican, por ejemplo:

  Una relación R={Proveedor, Status, Ciudad, Pieza, Cantidad} tiene como identificador: {Proveedor, Pieza} por lo tanto, no se puede dar de alta un tuplo del cual no se conozca el proveedor o la pieza.

- Anomalías de borrado.

  Esta anomalía se presenta cuando se borra una ocurrencia de una relación, la cual contiene información de un atributo que no queremos borrar, por lo tanto, el tuplo se elimina y adicionalmente perdemos la información de ese atributo; esto realmente es grave cuando el tuplo borrado contenía la última ocurrencia con información del atributo que no se pretendía borrar.

- Anomalías de modificación.

  Esta anomalía se presenta cuando existen instancias de atributos repetidos, por ejemplo:

  Al tener un departamento con 10 empleados existe un

punto crítico al momento de hacer modificaciónes a alguna de esas instancias, dado que se tienen varios tuplos con el mismo atributo, pudiendo caer fácilmente en inconsistencias al modificar sólo parte de esos tuplos.

5.1.2 Anomalías de procesamiento.

Existen otras anomalías llamadas anomalías de procesamiento que son detectadas en la operación del modelo implantado, afectando directamente su rendimiento.

- Anomalías de entrada.

  Se generan cuando el usuario requiere cierta salida de un modelo para la cual deba de proporcionar datos innecesarios a la entrada. La anomalía de entrada es similar a las anomalías que conducen a la 2NF de la administración relacional de los datos.

- Anomalías de búsqueda.

  Ocurre siempre que tenemos dependencias transitivas (dependencias entre atributos que no forman parte de la llave primaria) en una relación. Esta anomalía se elimina al hacer "projects" para obtener 2 relaciónes diferentes, por ejemplo:

  R=(precio, cantidad, costo)

  Si hay algún requerimiento en el que se desea conocer el costo de la producción de acuerdo con una cantidad, es necesario dar diferentes valores de precio hasta hacer coincidir los valores de costo de acuerdo a la cantidad especificada, por lo que es necesario separar la relación anterior en dos diferentes con la siguiente estructura: R1=(Costo, Cantidad) y R2=(Cantidad, Precio).

- Anomalías de salida.

  Una anomalía de salida es una respuesta no determinada a una consulta de usuario.
  Una de las causas de estas anomalías es el tener dos o más atributos idénticos como salida en diferentes modelos o aplicaciones bajo una organización relacional.

5.2 Dependencias funcionales.

Un concepto necesario para entender las formas normales es el de Dependencia Funcional.

La definición de dependencia funcional es la siguiente: "Dada una relación R, un atributo de R llamado Y es dependiente funcionalmente de un atributo X de R, esto es,

$$R.X. \longrightarrow R.Y.$$

R.X = parte determinante
R.Y = parte determinada.

si y solo si, cada valor de X en R es asociado con precisamente un valor Y a la vez[1]. Donde X y Y pueden ser conjuntos de atributos.

Algunos atributos son dependientes funcionalmente de otros porque conociendo un valor especifico de los primeros, se conoce o se tiene acceso al valor especifico de cada uno de los atributos dependientes.

Si un atributo X es un candidato a llave de la relación R, en particular es la llave primaria, entonces, todos los atributos Y de la relación R deben necesariamente ser dependientes funcionales de X (llave primaria).

La dependencia funcional proviene de la SEMANTICA de los datos. El proceso de identificar las dependencias funcionales está formado también, al entender qué significan los datos y las relaciónes entre ellos. Esto es, el analizar semánticamente diferentes situaciones del mundo real que se pueden presentar en la Base de datos.

Para poder comprender la definición de cada una de las formas normales, es necesario aclarar otros conceptos usados para definirlas.

- Atributo no llave: Es un atributo que no participa o no forma parte de la llave primaria de una relación específica.
- Mutuamente independiente: Dos o más atributos son mutuamente independientes si ninguno de éstos dependen funcionalmente entre si.

5.3 Formas Normales.

La teoría de la Normalización, está construída alrededor de un concepto que se conoce como "Formas Normales". De hecho, una relación se dice que está en alguna forma normal, sólo si cumple con un cierto conjunto de condiciónes para cada caso.

En la teoría de Normalización se incluyen numerosas formas normales. Originalmente Codd definió la primera forama normal $1NF_{(1)}$ la segunda forma normal 2NF y la tercera forma normal $3FN^{(1)}$.

La definición original que hizo Codd de la 3NF sufrió algunas modificaciónes debido a fuertes revisiones hechas por Boyce y el propio Codd dado que algunas relaciónes que estaban de acuerdo en la 3NF no lo estaban con la nueva definición. Para evitar confusiones, en la actualidad, la nueva definición es referida como "forma normal de Boyce/Codd" (BCNF) y la original solamente como 3NF.
Después de ésto Fagin definió una cuarta forma normal 4NF (debido a que en ocaciones a la BCNF se le llamaba 3NF). Más recientemente Fagin otra vez definió ontra forma normal que llamo "prjection-join normal form" (PJ/NF) tambien conocida como quinta forma normal $5NF^{(N,T)}$.

En este momento podemos mencionar que la teoria de la Normalización, es decir, el concepto de formas normales e ideas afines no estan concideradas como parte del modelo relacional como tal, pero constituyen un elemento importante en él.

5.3.1 Primera forma normal (1NF).

"UNA RELACION ESTA EN 1NF SI Y SOLO SI TODOS LOS DOMINIOS DELINEADOS PARA CADA UNO DE LOS ATRIBUTOS DE LARELACION CONTIENEN UNICAMENTE VALORES ATOMICOS."

Esta definición implica invariablemente una redundancia en los atributos que así lo requieran, dado que es necesario repetir algunos valores de los atributos para cumplir la condición de valores atomicos sin modificar el contenido original de la información. Estas redundancias en la 1NF pueden ocasionar anomalías de actualización.

Para solucionar ésto, es necesario dividir la relación original en un grupo de relaciónes donde la información contenida en cada una de ellas, además de estar relacionada

entre sí, se pueda accesar con sólo conocer parte de ella.

Para realizar lo anterior, es necesario encontrarse en la segunda forma normal.

5.3.2 Segunda forma normal (2NF).

"UNA RELACION R, ESTA EN 2NF SI Y SOLO SI ESTA EN 1NF Y TODOS LOS ATRIBUTOS NO LLAVE SON DEPENDIENTES TOTALMENTE DE LA LLAVE PRIMARIA".

El proceso de pasar de 1NF a 2NF, consiste en una serie de "Projects" (proyecciónes), sobre la tabla en 1NF. Estos Projects tienen la función de dividir o reducir el número de atributos de la relación, ocasionando con ésto, la generación de una serie de relaciónes, las cuáles, cumplen totalmente con la definición de la 2NF.

Hay que tener en cuenta que la serie de projects obtenidos son equivalentes a la relación original, en el sentido de que siempre se puede recobrar el estado original, aplicando otra serie de "Joins" naturales que contrarresten el efecto de los Projects.

Esto quiere decir que este proceso es reversible. Este proceso es conocido como "descomposición sin pérdida", lo que nos lleva a tener la posibilidad de derivar u obtener cualquier información, tanto de la estructura original como de la nueva estructura, con la diferencia que al tenerla organizada, representa mejor al mundo real, donde ésta tiene sentido.

En ocasiones se puede tener una serie de relaciónes en 2NF que presenten caracteristicas indeseables al momento de hacer actualizaciones o búsquedas de información. Para comprender ésto, es necesario introducir otro concepto. Este concepto es el de Dependencia Funcional Transitiva, la cuál nos dice:

R.A. ⟶ R.B.
(El atributo B es dependiente funcionalmente de A).
y R.B. ⟶ R.C.

se cumple que R.A. ⟶ R.C.

Este comportamiento entre atributos de una misma relación, ocasiona anomalías de actualización, dado que existen atributos que dependen de otros para poder existir o

consultar, sin que éstos sean parte de la llave primaria.

Como sucedió con el cambio de 1NF a 2NF, es necesario aplicar a las relaciónes que presenten características de Dependencia Funcional Transitiva, una serie de Projects para separar atributos y eliminar el efecto mencionado, asegurando con ésto que al actualizar una relación, no se generen anomalías.

Hasta este punto, se tienen relaciónes en 2NF. Al eliminar dependencias funcionales transitivas, nuestras relaciones automáticamente se encontrarán a salvo de este tipo de anomalías.

5.3.3 Tercera forma normal (3NF).

"UNA RELACION R, ESTA EN 3NF SI Y SOLO SI ESTA EN 2NF Y TODOS LOS ATRIBUTOS NO LLAVE, NO SON DEPENDIENTES TRANSITIVAMENTE DE LA LLAVE PRIMARIA".

Esto es, se está en 3NF cuando además de no existir dependencias transitivas, todos los atributos no llave son totalmente dependientes funcionales de la llave primaria.

En esta forma normal se conserva el hecho de existir un proceso reversible, lo cual nos puede llevar a decir que siempre es posible derivar de una serie de relaciónes en 3NF, una serie de relaciónes en 2NF.

El nivel de normalización de una relación es cuestión de semántica, de tal modo que no sólo importa el valor de los datos, sino también el significado que tienen. No es posible decir en qué forma normal se encuentra una relación, con sólo ver la tabulación de los datos, sino que es necesario conocer el significado para emitir un juicio correcto. De esta menera el tener las relaciónes en 3NF no va a asegurar un manejo adecuado de la Base de datos. Para evitar esto, se requiere conocer las dependencias relevantes que existan aún cuando las relaciones se encuentren en 1NF o 2NF.

5.3.4 Forma normal de Boyce-Codd (BCNF).

La 3NF no es válida en el siguente caso:

Tener múltiples llaves candidatas donde esas llaves

candidatas son compuestas y tienen al menos un atributo común.

Por lo tanto, la 3NF fué reemplazada por una definición más fuerte y general que se conoce como forma normal de Boyce/Codd. El caso anterior no sucede comunmente en la realidad por lo que la tercera forma normal es suficiente para la mayoría de los casos.

La definición de la BCNF es la siguiente:

"UNA RELACION R ESTA EN FORMA NORMAL DE BOYCE/CODD (BCNF) SI Y SOLO SI TODOS LOS DETERMINANTES SON UNA LLAVE CANDIDATA".

Es importante notar aquí que se habla no sólo de la llave primaria, sino también de llaves candidatas. Además, que la BCNF es conceptualmente más sencilla que la 3NF y no se hace referencia explícita a la 1NF ni al concepto de dependencia transitiva. A pesar de ésto, es posible obtener, a partir de una relación en estado "nativo", una serie de relaciónes en BCNF por medio de descomposiciónes sin pérdidas, utilizando series de Projects.

5.3.5 Cuarta forma normal (4NF).

Hasta este momento, se ha mencionado el concepto de dependencia funcional y dependencia transitiva, lo cual es aplicable a relaciones de uno a muchos y muchos a uno. Ahora, cuando tenemos una relación entre atributos de muchos a muchos, surge el problema de la representación de las relaciónes, puesto que se llega al límite máximo de una llave primaria en una relación, que es el que esta llave esté formada por todos los atributos de la relación, dado que los atributos entre sí son independientes y están relacionados con un sólo atributo.

Ahora, es necesario introducir un concepto que engloba este tipo de relaciónes entre atributos. Este concepto es el de dependencia multivaluada.

La definición de Dependencia Multivaluada es la siguiente: "Dada una relación R con atributos A,B, y C, la dependencia multivaluada (MVD):

$$R.A. \longrightarrow\!\!\!\!\!\rightarrow R.B$$

(El atributo B depende multivaluadamente de A).

se mantiene o existe, si y sólo si, el conjunto de valores B coinciden con una pareja dada de valores de A y C en R, y ésto depende sólo sobre el valor-de-A y es independiente del valor-de-C. Normalmente donde A, B y C pueden ser compuestos".

Esta definición es aplicable sólo cuando una relación tiene al menos tres atributos. También se hace notar que si la MVD R.A. ⟶ ⟶ R.B. se mantiene, la MVD R.A. ⟶ ⟶R.C. también se mantiene, siguiendo la forma de pares de valores para dos atributos, y un conjunto de valores para el tercer atributo; esto se representa como: R.A. ⟶ ⟶R.B./R.C.

Tratando de relacionar la FD y la MVD, podemos decir que una FD es una MVD en la cuál, el conjunto de valores dependientes coinciden, dado un valor determinado para un par de valores de los otros atributos, contiene un solo elemento. De esta manera, una FD siempre es una MVD, perouna MVD nunca será una FD.

Esta relación original la podemos transformar en sus dos projects, siguiendo el teorema:
"Una relación R con atributos A, B y C, puede ser descompuesta sin pérdidas en sus dos projects R1(A,B) y R2(A,C) si y sólo si existen las MVDS R.A. ⟶ ⟶R.B. / R.C.".

Ahora, tenemos ya los elementos necesarios para definir la 4NF:

"UNA RELACION "R" ESTA EN CUARTA FORMA NORMAL (4NF) SI Y SOLO SI, EXISTE UNA "MVD" EN "R", A ⟶ ⟶ B, DONDE TODOS LOS ATRIBUTOS DE "R" SON TAMBIEN DEPENDIENTES FUNCIONALES DE A."

En otras palabras, sólo las FD's o MVD's en R son de la forma K ⟶ X (o sea, una dependencia funcional de una llave candidata K a muchos otros atributos X). Utilizando otros conceptos para definir la 4NF, podemos definirla como sigue: "R ESTA EN 4NF SI ESTA EN BCNF Y TODAS LAS "MVD's EN"R" SON EN REALIDAD FD'S".

5.3.6 Quinta forma normal (5NF).

El concepto de descomposición sin pérdida tiene validez hasta la 4NF dado que esta etapa de normalización se basa en dicho concepto, ésto es, el proceso de reemplazar una relación por 2 de sus "Projects".

Ahora existen relaciónes que no pueden ser descompuestas sin pérdida con 2 de sus "Projects", ya que al manipular la información puede ocasionar información indeseabl e einexistente, pero haciendo operaciónes sobre los atributos de la relación se puede llegar a eliminar, pero pueden ser descompuestas en 3 o más de sus "Projects".

Para darle formalidad a este concepto, podemos expresarlo así: "Una relación es "n-descomponible" ("n-descomposable") para grado (n) > 2", lo cuál indica que la relación en cuestión puede ser descompuesta sin pérdida en n "projects" pero no en m "projects" para m < n.

Lo anterior puede ejemplificarse de la siguiente manera:

Teniendo una relacion SPJ entonces:

Si el par (S1, P1) aparece en el project SP y
el par (P1, J1) aparece en el project PJ y
el par (J1, S1) aparece en el project JS
entonces el trío (S1, P1, J1) aparece en SPJ, porque el trío (S1, P1, J1) se obtiene del join de SP, PJ, y JS.

Dado que la relación "3-Descomposable" (3D) SPJ es el resultado de una serie de joins de ciertos de sus "projects". Se puede hablar de una dependencia de joins (join dependency (JD)), la cual podemos definir de la siguiente manera:

"La relación R satisface la dependencia de joins:

join (X, Y, ..., Z)

si y sólo si R es igual al Join de sus projects en X, Y,..., Z, donde X, Y, ...Z son subconjuntos del conjunto de atributos de R".

En ocasiones es necesario evaluar si se debe actuar de acuerdo a la "n-descomposable" dado que al realizar los "projects" necesarios se presentan anomalias.

Existe un teorema el cuál nos relaciona la descomposición sin pérdida en una relación, las dependencias multivaluadas y la dependencia de joins. Este teorema es el de Fagin el cual contiene el siguiente postulado:

"R (A, B, C) satisface la dependencia de joins join (AB,AC) sí y sólo si ésta satisface el par de MVD's

A —↠ —↠ B/C".

Esto es, la relación R con atributos A, B y C puede ser descompuesta sin pérdidas en R1 (A, B) y R2 (A, C) si se mantiene una MVDA —↠ —↠ B/C en R.

El teorema puede ser tomado como otra definición de MVD por lo cual podemos decir que una MVD es un caso especial de una dependencia de joins (JD) o que JD es una generalización de MVD así como ésta es una generalizacion de una FD. De ésto se desprende que la JD es la máxima generalización de dependencia posible dentro del área de relaciónes y conceptos de normalización, hasta el momento, aunque pueden surgir otros tipos de dependencia en el futuro.

Ahora, podemos tener el caso en donde una JD no sea una MVD y por lo tanto tampoco una FD, entonces es recomendable y deseable el descomponer cada relación con el menor número de atributos, ésto es en los "projects" que cumplan estrictamente con las caracteristicas de una JD, y esta descomposición puede ser repetida hasta llegar a tener una relación en 5a. forma normal (5NF).

La 5NF es también conocida como projection-join normal form (PJ/NF) y se define de la siguiente manera:

"UNA RELACION R ESTA EN 5NF ó JP/NF SI Y SOLO SI CADA DEPENDENCIA DE JOINS EN R ES UNA CONSECUENCIA DE LAS LLAVES CANDIDATAS DE R"

Desde que la 5NF está basada en el concepto de JD y éste es un caso general de MVD, una relación en 5NF está automáticamente en 4NF.

Para poder entender la definición de la 5NF es necesario observar las llaves candidatas dentro de cada subconjunto de atributos de cada relación. El comportamiento que siguen es que en cada subconjunto de atributos al menos cuenta con un atributo candidato a llave de la relación original, siendo ésta una caracteristica general para poder llegar a tener una relación en 5NF.

Observando de manera contraria a ésta, dada una relación en 5NF podemos llegar a reconocer u obtener las llaves candidatas observando el comportamiento o distribución de atributos en los projects que la forman, aunque puede ser una difícil tarea el reconocer las JD, dado que la inter-

pretación de JD, a diferencia de las MVD o FD, no es obvia, por lo cual se puede llegar sólo a concluir que se está en 4NF pero el llegar a concluir que se está en 5NF no es aún un método claro, aunque el intentar llegar a una 5NF es prácticamente un caso en extremo raro y en pocas ocasiones se tiene la necesidad de llegar a realizar estos análisis.

5.4 Secuencia de normalización.

A continuación se presentan varios ejemplos para identificar los conceptos antes mencionados y mostrar las anomalías que se presentan en cada caso así como su solución a través de la normalización.

Como ejemplo de las facilidades del modelo relacional usaremos una base de datos de personal.
Consideremos la siguiente información a manipular:
- El nombre de cada empleado (EMP).
- Lenguajes que el empleado maneja (LNG).
- Años que el empleado maneja el lenguaje (USO).
- Posición del empleado en la organización (POS).
- Años de experiencia (EXP).
- Sueldo por hora del empleado (SDO).
- Proyecto al que el empleado está asignado (PRY).

Que representada gráficamente se muestra a continuación.

PERSONAL 1

EMP	(LNG, USO)	POS	EXP	SDO	(PRY, JFE)
ALMA	COBOL, 3 FORTRAN, 4	PROGR.	4	25	NOMINA, JOSE CONTAB, JUAN
ARMANDO	COBOL, 2 PL/1, 1 RPG, 3	PROGR.	3	24	INVENT, AGUSTIN
AGUSTIN	BASIC, 4 PASCAL, 3	JEFE	2	32	INVENT, AGUSTIN
LUIS	PASCAL, 1	PROGR.	1	25	CONTAB, JUAN NOMINA, JOSE

En el encabezado de la estructura, los paréntesis encierran a grupos de datos que pueden repetirse.

Con el diseño de PERSONAL 1, cualquier pregunta de la forma:

Dime algo del empleado X.

puede ser respondido fácilmente, pero es mucho más difícil responder a preguntas como:

¿ Qué empleados usan el lenguaje L.?
¿ Dime quién es el jefe del proyecto P.?
¿ Qué empleados están asignados al proyecto P.?
¿ Cambia el jefe del proyecto P por el empleado E.?

En nuestro ejemplo, le damos a cada empleado un nombre único, por eso EMP puede usarse como llave en PERSONAL 1. En la práctica, puede haber más de una manera de construír una llave en una relación: Registro Federal de Causantes, número de empleado en la organización, etc.
Para poder utilizar la estructura anterior en el modelo relacional es necesario convertirla en una relación, duplicando los valores no repetitivos EMP, POS, ESP Y SDO para cada combinación de valores y grupos repetitivos (LEN, USO) Y (PRY, JFE), la relación puede ser representada en primera forma normal, como se muestra en PERSONAL 2.

PERSONAL 2

EMP	LNG	USO	POS	EXP	SDO	PRY	JFE
ALMA	COBOL	3	PROGR.	4	25	NOMINA	JOSE
ALMA	FORTRAN	4	PROGR.	4	25	CONTAB	JUAN
ARMANDO	COBOL	2	PROGR.	3	24	INVENT	AGUSTIN
ARMANDO	PL/1	1	PROGR.	3	24	INVENT	AGUSTIN
ARMANDO	RPG	3	PROGR.	3	24	INVENT	AGUSTIN
AGUSTIN	BASIC	4	JEFE	2	32	INVENT	AGUSTIN
AGUSTIN	PASCAL	5	JEFE	2	32	INVENT	AGUSTIN
LUIS	PASCAL	1	PROGR.	1	25	CONTAB	JUAN
LUIS	PASCAL	1	PROGR.	1	25	NOMINA	JOSE

Nótese que en PERSONAL 2, el atributo EMP no es suficiente para identificar un túplo. Se presentan varios túplos para los empleados que tengan más de un lenguaje o más de un proyecto asignado.

Una posible solución, es considerar la combinación de EMP, LNG Y PRY como llave, ya que estos tres valores juntos son

suficientes para identificar un túplo.

Pudiera parecer de momento, que PERSONAL 2, representa un paso hacia atrás, no sólo porque requiere de más espacio que PERSONAL 1, sino porque también se dificulta la respuesta a peticiónes como:

- Cambia al empleado E el sueldo S (anomalía de modificación)
- Agrega a la asignación del empleado E el proyecto P. (anomalía de inserción)
- Convierte al empleado E en jefe de proyecto P. (anomalía de modificación)
- Eliminar el proyecto Contab sin eliminar uso de fortran de Alma (anomalia de borrado)

Estos problemas se resuelven con los pasos de normalización que aún quedan por hacerse, tomando en cuenta las siguientes consideraciónes semánticas.

- Cada empleado tiene solo una posición,
- Cada empleado es asignado a un conjunto específico de proyectos,
- Cada proyecto tiene un jefe,
- El empleado conoce varios lenguajes.

Para poder eliminar las anomalías que generan las peticiones debemos transformar la relación para que todos los atributos no llave dependan totalmente de ella. En esta caso se identifcia que los atributos POS, EXP y SDO, no son dependientes de la llave entera en PERSONAL 2. Estos atributos son determinados sólo por EMP, así que tenemos una nueva relación que llamaremos PERSONAL 3, que contenga solamente EMP, POS, EXP y SDO, teniendo a EMP como la llave.

Por otro lado JFE se determina por EMP y PRY generándose la relación EMP_JFE que tiene como llave a EMP y PRY.

Asi mismo el valor de USO se determina mediante EMP y LNG, por lo que crea una relación llamada ANTECEDENTES, teniendo como llave compuesta a EMP y LNG.

PERSONAL 3

EMP	POS	EXP	SDO
ALMA	PROGR.	4	25
ARMANDO	PROGR.	3	24
AGUSTIN	JEFE	2	32
LUIS	PROGR.	1	25

ANTECEDENTES

EMP	LNG	USO
ALMA	COBOL	3
ALMA	FORTRAN	4
ARMANDO	COBOL	2
ARMANDO	PL/1	1
ARMANDO	RPG	3
AGUSTIN	BASIC	4
AGUSTIN	PASCAL	3
LUIS	PASCAL	1

EMP_PRY

EMP	PRY	JFE
ALMA	NOMINA	JOSE
ALMA	CONTAB	JUAN
ARMANDO	INVENT	AGUSTIN
AGUSTIN	INVENT	AGUSTIN
LUIS	CONTAB	JUAN
LUIS	NOMINA	JOSE

Ahora es posible modificar el sueldo del empleado sin saber sus antecedentes ni el proyecto en el que trabaja, es posible dar de alta un empleado aunque no tenga asignado un proyecto y al eliminar un proyecto no perderemos información del personal y sus antecedentes.

Separando la relación, se evita tener en el modelo atributos que son dependientes sólo de una parte de la llave. Una relación en primera forma normal que no tiene dependencias parciales sobre la llave, se dice que está en segunda forma normal, cumpliéndose la 2NF para las tres relaciones.

Sin embargo podemos observar que si se cambia al jefe de un poryecto éste tendrá que ser modificado para todos los empleados de dicho proyecto en la relación EMP_JFE (anomalia de modificación), esto se debe a que el nombre del proyecto determina al jefe del proyecto. Se dice entonces, que JFE es dependiente transitivamente de EMP, ya que EMP determina al conjunto de valores de PRY, y a su vez PRY determina funcionalmente a JFE. Esta dependencia transitiva tiene que eliminarse de acuerdo a la definición de la 3NF.

Como cada proyecto tiene sólo un jefe, formaremos la relación PROYECTOS de las columnas PRY y JFE, usando como llave a PRY.

Se puede construir una relación de ASIGNACIONES de EMP y PRY, ya que cada empleado trabaja en uno o más proyectos, tomando como llave a EMP y PRY. Con ésto se forma la asociación entre PERSONAS y PROYECTOS.

PERSONAL 4

EMP	POS	EXP	SDO
ALMA	PROGR.	4	25
ARMANDO	PROGR.	3	24
AGUSTIN	JEFE	2	32
LUIS	PROGR.	1	25

ANTECEDENTES

EMP	LNG	USO
ALMA	COBOL	3
ALMA	FORTRAN	4
ARMANDO	COBOL	2
ARMANDO	PL/1	1
ARMANDO	RPG	3
AGUSTIN	BASIC	4
AGUSTIN	PASCAL	3
LUIS	PASCAL	1

ASIGNACIONES

EMP	PRY
ALMA	NOMINA
ALMA	CONTAB
ARMANDO	INVENT
AGUSTIN	INVENT
LUIS	CONTAB
LUIS	NOMINA

PROYECTOS

PRY	JFE
NOMINA	JOSE
CONTAB	JUAN
INVENT	AGUSTIN

Todas las relaciónes del modelo anterior, están en tercera forma normal, lo que nos permite cambiar al jefe de proyecto una sola vez en la relación proyectos.

Dada la definición de BCNF donde todos los determinantes son llaves candidatas tenemos:

- La relación Personal 4 tiene como identificador único a EMP, sin sin determinantes por lo que cumple la definición.

- La relación ANTECEDENTES, con llave EMP, LNG, tampoco tiene llaves candidatas, estando en BCNF.

- La relación ASIGNACIONES tiene a sus dos atributos EMP y PRY como llave, por tanto también se encuentra en BCNF.

- La relación PROYECTOS tiene al atributo JFE como determinante:

JFE ——→ PRY

y a su vez JFE es llave candidata, cumpliéndose nuevamente la definición.

Asimismo, la definición de 4NF nos dice que las dependencias multivaluadas deben ser dependencias funcionales por lo que concluimos que las cuatro relaciónes obtenidas estan en 4NF. La descomposición de PERSONAL en las cuatro relaciónes es una descomposición sin perdida puesto que a través de varios Joins se obtiene la relación original.

Los casos en los que se aplica la BCNF y la 4FN se presentan rara vez en la vida real. Sin embargo se seleccionaron los siguientes ejemplos para completar la identificación de los conceptos de normalización.

Desafortunadamente las relaciónes en 3NF pueden presentar anomalías, por lo que se hace necesario transformarlas a BCNF. para esto utilizaremos la siguiente relación:

Llave primaria (#EST,MATERIA)

Llave candidata (#EST,MAESTRO)

#EST	MATERIA	MAESTRO
100	MATEMATICAS	ACUÑA
150	PSICOLOGIA	ROSALES
200	MATEMATICAS	URIARTE
250	MATEMATICAS	ACUÑA
300	PSICOLOGIA	RAMOS

Debido a que un maestro está asociado únicamente a una materia pero puede haber varios profesores en cada materia, entonces tenemos la siguiente dependencia funcional:

Dependencias funcionales MAESTRO ——→ MATERIA

La relación se encuentra en 1NF por definición ya que no tiene atributos no llave. Tambien está en 2NF. Y debido a que no

tiene dependencias transitivas esta en 3NF.

Claramente se puede observar que si eliminamos al estudiante 300, se pierde la información referente a que Ramos es maestro de Psicologiía (anomalía de borrado). Similarmente, no podemos almacenar el hecho de que Acuña es maestro de Economía hasta que un alumno tome la materia de Economía (anomalía de inserción).
Situaciones como las anteriores nos orillan a transformar la relación a BCNF en donde cada determinatne es llave candidata; así es que se generan dos nuevas relaciónes que no tendrán anomalías, haciendo los siguientes proyects:

PROYECT RELACION (#EST,MAESTRO)

PROYECT RELACION (MAESTRO,MATERIA)

MATERIA	MAESTRO
MATEMATICAS	ACUÑA
PSICOLOGIA	ROSALES
MATEMATICAS	URIARTE
PSICOLOGIA	RAMOS

Llave MAESTRO

#EST	MAESTRO
100	ACUÑA
150	ROSALES
200	URIARTE
250	ACUÑA
300	RAMOS

Llave #EST

En estas relaciónes tenemos que cada determinante es llave candidata, cumpliéndose la BCNF. Esto nos perite dar de baja al estudiante 300 sin perder el hecho de que Ramos es maestro en Psicologiía, así como nos permite dar de alta a Acuña como maestro en Economía, sin la necesidad de esperar por la inscripción de algún estudiante a esa materia.

Independientemente de que una relación se encuentre en BCNF, puede llegar a presentar anomalías si ésta presenta dependencias multivaluadas, como se explicará en el siguiente ejemplo:

Llave primaria (#EST,MATERIA,ACTIVIDAD)

Dependencias multivaluadas

#EST ——→ MATERIA
#EST ——→ ACTIVIDAD

#EST	MATERIA	ACTIVIDAD
100	MUSICA	NATACION
100	CONTABILIDAD	NATACION
100	MUSICA	TENIS
100	CONTABILIDAD	TENIS
150	MATEMATICAS	ATLETISMO

En esta relación todos los atributos son llave por lo que se encuentra en BCNF, sin embargo, presenta deficiencias si el estudiante 100 toma una materia más, pues deberemos buscar todas sus actividades y agregar un tuplo para cada una de ellas, incluyendo esta nueva materia. En este caso dos tuplos deben de ser agregados; si no se agregan estos dos tuplos, la semántica de la relación no es congruente, ya que el estudiante pierde una actividad al dar de alta una nueva materia (anomalía de inserción).

Ahora, supóngase que el estudiante 100 deja la actividad tenis, por lo que hay que eliminar 2 tuplos ya que si solo se elimina uno se mantiene la información de la actividad tenis para ese estudiante.

La solución a este problema es proyectar la relación en dos relaciónes como se muestra a continuación:

```
PROYECT RELACION (#EST,MATERIA)
PROYECT RELACION (#EST,ACTIVIDAD)
```

#EST	ACTIVIDAD
100	NATACION
100	TENIS
150	ATLETISMO

Llave primaria (#EST,ACTIVIDAD)

#EST	MATERIA
100	MUSICA
100	CONTABILIDAD
150	MATEMATICAS

Llave primaria (#EST,MATERIA)

Dado que las relaciones tienen dos atributos, no tienen dependencias multivaluadas, por lo tanto se encuentran en 4NF porque están en BCNF (todo determinante es parte de la llave) y porque las dependencias son solo funcionales.

6 TECNICAS DE INTELIGENCIA ARTIFICIAL

6.1 Origen de la Inteligencia Artificial.

Desde el orígen de la computación, con el desarrollo de grandes máquinas para efectuar cálculos repetitivos a gran velocidad, surgen una serie de especulaciones concernientes a la posibilidad de crear un nuevo género de máquinas que realizaran actividades "inteligentes".

Aún hoy es difícil dar una definición sobre lo que se entiende por actividad inteligente[9]. Igualmente, no existe una definición exacta para el concepto de Inteligencia Artificial.

Sin embargo, podemos decir que la Inteligencia Artificial estudia la forma en que una computadora llegue a realizar mejor tareas que los humanos realizamos[K]. Por el momento no se pretende definir ni lo que es la inteligencia ni lo que se entiende por Artificial, por el contrario, se describe lo que constituye una rama de la Computación conocida como Inteligencia Artificial.

Algunos de los primeros problemas estudiados por esta rama de la computación fueron la Teoría de Juegos y la demostración de Teoremas Matemáticos. Samuel en 1963, escribió un programa conocido como "Checkers-Playing" que no solo jugaba con sus oponentes sino que utilizaba la experiencia adquirida en juegos anteriores, para mejorar en juegos posteriores[X]. En el mismo año, Newell escribió un programa llamado "The Logic Theorist" que fué un intento para demostrar Teoremas Matemáticos. Aparentemente, la Teoría de Juegos y la demostración de Teoremas Matemáticos, se realizaban adecuadamente a través de una computadora por su rapidez para analizar todas las posibles rutas de solución y seleccionar la mejor.

Por otro lado, Newell estudió el tipo de razonamiento que los humanos utilizamos para resolver problemas en general. Como producto de esta investigación, presentó junto con Shaw y Simon el programa GPS (General Problem Solver), el cual era aplicable a varias tareas, incluyendo la manipulación simbólica de expresiones lógicas.

Hasta este momento, las computadoras ofrecían una adecuada capacidad de memoria y velocidad de respuesta para cierto tipo de problemas, como por ejemplo, las largas cadenas de operaciones aritméticas en un programa de contabilidad. Sin

embargo, el avance que se había alcanzado en materia de investigación, distaba mucho del que se había pronosticado en los inicios de la computación.

Las técnicas hasta ese entonces utilizadas, para la solución de problemas a través de una computadora, presentaban serias limitaciones al aplicarlas a problemas de Inteligencia Artificial. En teoría de Juegos, por ejemplo, se puede escribir un programa para jugar "Gato", almacenando en memoria un vector con todos los posibles movimientos, requiriendo 3^9 posiciones diferentes, para un juego tan sencillo. (No es difícil imaginar el número de posiciónes necesarias para representar todas las posibilidades que presenta un tablero de Ajedrez); por lo que fué necesario dar otro enfoque a la solución de problemas ya que no existe computadora alguna que sea capaz de soportar la explosión de combinaciónes que presentan algunos problemas.

Los primeros esfuerzos realizados, con el estudio de la Teoría de Juegos, la demostración de Teoremas Matemáticos y el estudio del razonamiento humano para resolver problemas, permitieron entender que las actividades "inteligentes" no son triviales, por el contrario, son muy complejas y difíciles de caracterizar. Esta situación propició el desarrollo de nuevas y diversas técnicas para resolver los diferentes problemas estudiados por la Inteligencia Artificial.

Dadas las características que presentaban algunos problemas, se observó que no era suficiente la representación de información a través de números y caracteres. Uno de los avances más relevantes que se ha obtenido con la Inteligencia Artificial es el nuevo concepto en programación conocido como Procesamiento Simbólico.

El Procesamiento Simbólico es la manipulación por alguna computadora, de información y conocimiento representados por símbolos. Los símbolos hacen referencia a objetos reales y sus propiedades,y se pueden enlazar para representar relaciones entre ellos, como dependencias o jerarquías[C]. Este paso de abstracción, en el concepto de información permite representar los objetos involucrados en los problemas de Inteligencia Artificial y manipularlos de alguna manera para llegar a una solución adecuada.

Otra de las importantes conlusiones obtenidas, es que cualquier actividad "inteligente" requiere de un buen grado de conocimiento, aunque aparente ser una actividad "trivial". El conocimiento guarda las siguientes características que lo

hacen difícil de representar:

- Es voluminoso
- Es difícil de caracterizar
- Es cambiante

Gran parte de la investigación de Inteligencia Artificial, se ha dedicado al estudio de métodos que permitan explotar y representar el conocimiento de manera que:

- Permita generalizar (no se pretende representar situaciones individuales, por el contrario, se deben agrupar situaciones que compartan propiedades, para efectos de memoria, tiempos de proceso y acceso)

- Sea entendible para las personas que proporcionan el conocimiento.

- Sea fácil de modificar.

- Aplicable a tantas situaciones como sea posible.

El estudio de métodos para la representación y manipulación del conocimiento no ha evolucionado aisladamente. Conjuntamente se han desarrollado una amplia gama de técnicas de búsqueda e inferencia adecuadas a las diversas características que presentan los problemas de Inteligencia Artificial.

Comencemos con las técnicas de búsqueda. Podemos representar un problema a través de una gráfica, donde contamos con un nodo inicial y sabemos cuál es el nodo "meta". Las técnicas de representación del conocimiento se encargan de encontrar la mejor forma de representar cada nodo (punto) y de cómo combinarlos entre sí para que representen un estado completo del problema.

En cuanto a la capacidad de inferir tenemos entendido que al entablar una conversación, no es necesario decir todas las ideas y conceptos que van implícitos en alguna expresión, asumimos que el oyente entiende estas ideas implícitamente y de manera natural. Sin embargo, como podríamos enseñar a una máquina para entender ideas que no se dicen, que por el contario, se asumen o infieren. La capacidad de inferir y deducir no se enseña en ningún lado, forma parte de nuestro comportamiento natural y las conclusiones que el ser humano obtiene, las adquiere de su experiencia cotidiana[2].
La propiedad de inferir hechos a partir de cierto conocimiento previo, es una característica importante de

las actividades inteligentes de los humanos. Los estudios realizados al respecto, por la Inteligencia Artificial, han encontrado que la lógica matemática es una herramienta esencial para el desarrollo de técnicas de inferencia.

Otro aspecto importante en el estudio de esta rama de la computación es la percepción del mundo que nos rodea. El proceso de percepción es difícil, comenzando por el carácter analógico de las señales involucradas en la percepción, ya sean éstas auditivas, visuales o de cualquier otro tipo. La Inteligencia Artificial estudia técnicas de Reconocimiento de Patrones que permiten a una computadora reconocer, clasificar e interpretar imágenes o sonidos.

Aproximadamente 25 años de investigación en Inteligencia Artificial, han permitido entender y desarrollar, entre otras:

- Técnicas para representar y manipular el conocimiento,
- Métodos de búsqueda adecuados a cada problema específico,
- Herramientas de programación que tengan la capacidad de inferencia y abstracción y
- Tecnología que permita la percepción del mundo que nos rodea.

6.2 Problemas a solucionar mediante la Inteligencia Artificial.

El espectro de los problemas estudiados, así como el de técnicas de solución, es muy amplio y continúa extendiéndose cada vez más, sin embargo, todos los diversos problemas, comparten una característica en común, "Son difíciles de resolver". Algunos de los problemas que estudia la Inteligencia Artificial son:

- Resolución de problemas en general.

  Es el estudio de metodologías que permitan construir programas para la solución de problemas, basándose en el desarrollo de técnicas de búsqueda y manipulación de conocimiento.

- Entendimiento del lenguaje natural.

  Desarrollo de técnicas que faciliten la comunicación Hombre-Máquina-Hombre utilizando el Lenguaje Natural.

- Percepción y Reconocimiento de Patrones:

  Hacer computadoras capaces de analizar, extraer, describir e identificar patrones de datos de señales acústicas, visuales o de cualquier otro tipo. Por ejemplo, Análisis/Síntesis de voz.

- Almacenamiento y Recuperación del Conocimiento:

  Debido a la explosión de información, existe la necesidad de realizar sistemas eficientes que permitan manipular gran volumen de conocimientos, por lo cual un area importante de la Inteligencia Artificial se dedica a desarrollar técnicas de Representación del Conocimiento por medio de lenguajes especializados. Los Sistemas Expertos, son ejemplo de la utilización de estas técnicas, y hasta ahora, son los únicos Sistemas Inteligentes del mercado.

  SISTEMAS EXPERTOS

  * Matemática Simbólica,
  * Diagnóstico Médico,
  * Análisis Químico,
  * Ingenierá de Diseño, etc.

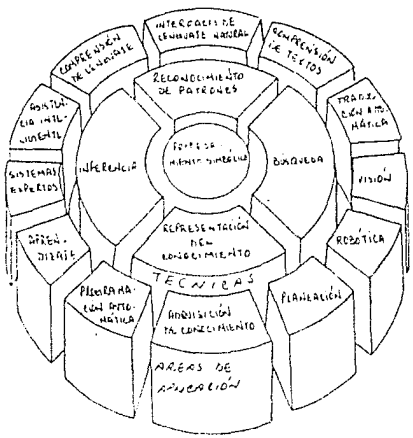
- Simulación y Modelado.

  Un modelo trata de encontrar una imagen o abstracción de algún fenómeno u objeto, reflejando sus rasgos más importantes y tratando de apegarse al mundo real. Su codificación y utilización para resolver algún problema es la Simulación. Por ejemplo, Simulación del Tracto Vocal para la Síntesis de voz.

- Lógica Computacional.

  Trata de probar que ciertos hechos son consecuencia de otros. Por ejemplo, un programa analizado desde el punto de vista lógico y no sintáctico.

El siguiente diagrama muestra un panorama global de la Inteligencia Artificial:



Panorama Global de la Inteligencia Artificial

6.3 Situación actual de la Inteligencia Artificial.

En la actualidad existen dos tendencias dentro de la inteligencia Artificial, la simulación de la experticia y la simulación del sentido común. En el primer caso se obtuvieron resultados alentadores al colocar en el mercado los "Sistemas Expertos", sin embargo, en la simulación del sentido común no se han logrado resultados concretos. Es por esto último que para algunos autores la Inteligencia Artificial ha dado todo lo que tiene que dar: "Sistemas expertos"[V]. Este hecho ha llamado la atención de Fisiólogos, Psicólogos y Computólogos, para estudiar conjuntamente el comportamiento humano, para desarrollar nuevas técnicas que permitan alcanzar los objetivos de la Inteligencia Artificial, así como ampliar y modificar los conceptos que se tienen del ser humano.
Las computadoras de la 5a. generación, ambicioso proyecto Japonés, reflejará de forma integral, los avances alcanzados en materia de Inteligencia Artificial. Este proyecto pretende desarrollar sistemas computacionales rápidos, inteligentes y que tengan la capacidad de:

- Tomar decisiones (comparables a las del ser humano).
- Aprendizaje.
- Interface de entrada / salida a través de voz (Lenguaje Natural)
- Generación automática de programas.
- Procesamiento distribuido.

todo esto, a través de comunidades de Sistemas Expertos, trabajando en paralelo en subproblemas para una tarea especifica.

A continuación se presenta un bosquejo de la forma como se pretende queden constituidas estas computadoras:

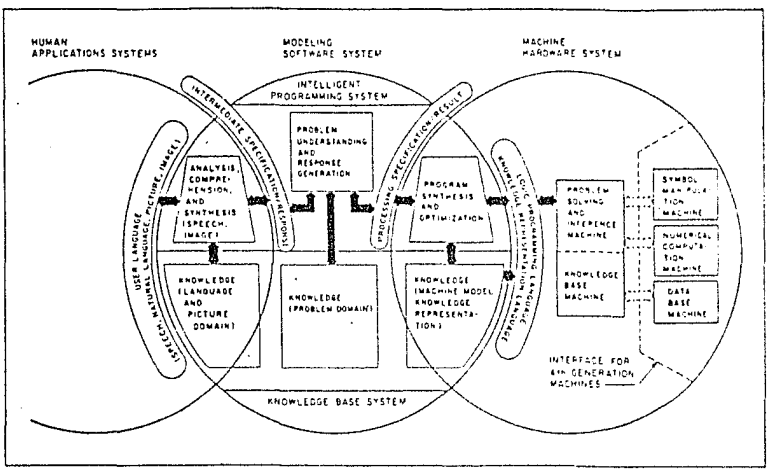


Diagrama conceptual de una computadora de 5a Generación

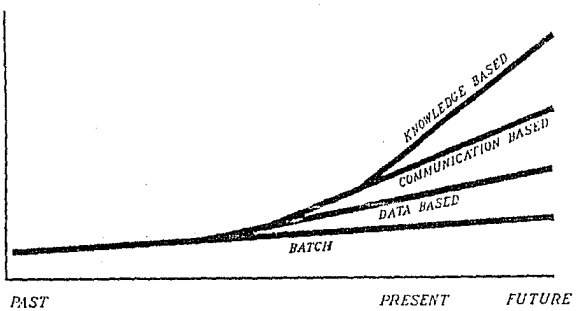
La Inteligencia Artificial, ha tenido repercusiones en todas las demás ramas de la computación. Aunque las técnicas de Inteligencia Artificial deben ser diseñadas para mantener las restricciónes impuestas por cada problema específico, existe cierto grado de independencia entre los problemas y las técnicas para resolverlos. Es posible aplicar técnicas de

Inteligencia Artificial a problemas que no pertencen al área, como también es posible solucionar problemas de Inteligencia Artificial sin las técnicas desarrolladas para dicho propósito, pero la experiencia ha demostrado que sin la utilización de estas técnicas no se obtienen soluciónes eficientes, o bien en algunos casos simplemente no se obtiene solución alguna.

El criterio para determinar si se ha tenido éxito en el desarrollo de programas "inteligentes", radica principalmente en la comparación de los resultados obtenidos con los objetivos para los cuales fué desarrollado el programa. Este punto se presta a discusiones y ambigüedades, pues es tan complejo como definir lo que es la Inteligencia.

La Inteligencia Artificial, escasamente tiene 25 años de edad, por lo que creemos que junto con las demás ramas de la computación, aún hay mucho que investigar y desarrollar vislumbrando un futuro prometedor.

Para algunos autores existe la notable tendencia de implantar sistemas basados en la manipulación del conocimiento enfocados a la toma de desiciónes, proponiendo una evolución de los sistemas de información como se ilustra a continuación (fuente The Knowledge Based Systems Center).



6.4 Tipos de problemas y técnicas de solución.

Aunque la gama de problemas estudiados por la Inteligencia Artificial es mucho más amplia que la gama de técnicas de solución, existen diversas técnicas de solución generales que se explicarán a lo largo de este capítulo.

Existen 3 pasos que se deben seguir para resolver cualquier problema mediante la construcción de un sistema:

- Definir el problema con precisión.
- Analizar el problema. (Obtener características importantes de gran impacto en el momento de escoger una técnica de solución).
- Escoger la(s) mejor(es) técnica(s) y aplicarla(s).

6.4.1 Definición del problema como un espacio de estados.

Un problema se puede definir como un espacio de estados. Por ejemplo, en un tablero de ajedrez, los estados del problema serían las diferentes posiciones que cada pieza puede tener en combinación con todas las demás.

Aparte de definir los posibles estados del problema, necesitamos encontrar reglas que nos permitan movernos de un estado a otro; para el caso del ajedrez, necesitamos de reglas que definan los movimientos legales que cada una de las piezas puede realizar.

Existe una manera fácil de describir un conjunto de reglas, la cual se forma de dos partes:

- Una parte izquierda; que sirva como patrón a cotejar con el estado actual del problema.

- Una parte derecha; que describa las acciónes a realizar para reflejar el movimiento de un estado a otro, por ejemplo:

  Para el ajedrez podríamos definir la siguiente regla:

  Peón blanco en cuadro (renglón2,columnaj) PARTE
  y
  Cuadro (renglón3,columnaj) está vacío IZQUIERDA
  y
  Cuadro (renglón4,columnaj) está vacío (PATRON)

entonces

Mueve peón de Cuadro (renglón2,columnaj) a Cuadro (renglón4,columnaj) — PARTE DERECHA (ACCIONES A TOMAR)

Para el ajedrez, aproximadamente existen 10^{120} posiciones diferentes. Si deseamos escribir reglas específicas para cada estado del problema, sería un número tan grande de reglas y estados que no sería posible definir de esta forma el problema.

El proceso de definición de reglas no es trivial. Se debe buscar la forma de describirlas lo más general posible, pues de otra forma, nos encontraríamos con las siguientes limitaciónes:

- Ninguna persona sería capaz de describir un conjunto completo de reglas, debido a el tiempo que tomaría y a la sensibilidad a errores.

- Ningún programa podría manipularlas fácilmente, debido a las repercusiones en espacio de almacenamiento, así como en tiempo de proceso de búsqueda.

Mientras las reglas se definan de forma concisa y general, tendremos la capacidad de proveerlas y de elaborar un programa que las manipule.

El espacio de estados de un problema forma las bases para casi todos los métodos de Solución de Problemas.

La estructura del espacio de estados del problema corresponde a la estructura de la solución del problema en 2 formas:

- Permite formalizar la definición del problema, como la necesidad de convertir una situación actual en una situación deseada a través de un conjunto de operadores permisibles.

- Permite definir el proceso de solución de un problema específico, como la combinación de reglas para movernos de un estado a otro, y una técnica de búsqueda para explotar el espacio de estados y encontrar una ruta que nos traslade de un estado inicial a otro final o estado solución.

La búsqueda es un proceso importante para encontrar soluciónes a problemas difíciles cuando no podemos utilizar técnicas directas.

Además de la definición del espacio y la descripción de las reglas de movimiento, necesitamos de una estructura de control que permita cotejar entre el estado actual y todas las partes izquierdas de las reglas (patrones), para efectuar la acción descrita en la parte derecha de la regla coincidente, y verificar si el estado resultante de la aplicación de la regla es la solución o no.
Resumiendo, podemos obtener una descripción formal del problema en los siguientes pasos:

a) Definir un espacio de estados que contenga todas las posibles configuraciónes de los objetos relevantes del problema.

b) Especificar uno o más estados del espacio que describan las posibles situaciónes donde el problema puede comenzar. Estos estados se llaman estados iniciales del problema.

c) Especificar uno o más estados que podrían ser soluciones aceptables del problema. Estos estados se conocen como estados meta o estados solución.

d) Especificar un conjunto de reglas que describan las acciónes posibles a realizar (operadores). Para este propósito debemos considerar puntos importantes tales como:

   - Cuáles son los hechos que asumimos implícitamente en la descripción informal del problema.

   - Qué tan generales deben ser las reglas.

   - Cuánto del trabajo necesario para resolver el problema debe ser dedicado para representarlo en reglas.

Bien definido el problema, éste puede ser resuelto utilizando las reglas en combinación con una estrategia de control para movernos de los estados iniciales a través del espacio hasta encontrar una ruta que nos lleve al estado meta o solución.

6.4.2 Análisis del problema.

El proceso de búsqueda es un mecanismo general, fundamental para encontrar soluciónes a problemas, sobre todo cuando no existe un método directo de solución.

La búsqueda es el centro de muchos procesos inteligentes, es por esto que los programas de Inteligencia Artificial deben ser estructurados de manera que faciliten la descripción del proceso de búsqueda, a travéz de estrategias de control.
Las estrategias de control deben presentar 2 características esenciales:

- Deben provocar movimiento
- Deben ser sistemáticas

como por ejemplo:

a) Construir un árbol con el estado inicial como raíz.

b) Generar todos los nodos sucesores de la raíz aplicando cada una de las reglas aplicables al estado.

c) Para todos los nodos sucesores efectuar el punto b).

d) Continuar hasta que alguna regla produzca un estado meta.

A este proceso se le conoce como "Breath-first Search".

Otra estrategia sistemática de control es la llamada "Depth-first-Search", la cuál expande una rama del árbol hasta que encuentra un estado solución, o bien hasta que alcanza un nivel de expansión predeterminado. Y aunque la expansión del árbol puede ser infinita estas dos estrategias permiten el control del movimiento, son sistemáticas y por lo tanto, fácilmente implementables.

Para resolver problemas con una expansión infinita en el árbol, es necesario comprometer los aspectos de movimiento y sistematización para construir una estrategia de control que no garantiza encontrar la mejor solución, pero que almenos garantice obtener alguna. Es así que introducimos el concepto de búsqueda heurística como una técnica que mejora la eficiencia del proceso de búsqueda.

Hay algunas técnicas heurísticas de propósito general que son aplicables en varios problemas. Como ejemplo, tenemos el algoritmo conocido como "the nearest neighbor" que apartir de un nodo inicial, escoge al mejor nodo sucesor, y para cada estado (nodo) escogerá siempre al mejor sucesor. Dependiendo del problema, la función heurística debe evaluar lo que se entiende por "mejor" y dicha evaluación la dan las características propias del problema.

6.4.3 Características de los Problemas.

Para poder escoger el método más apropiado para un problema específico debemos analizar el problema desde los siguientes puntos de vista:

- Analizar si el problema se puede descomponer.

  El ejemplo típico de descomposición de problemas es el de integración matemática

  $$\int (x^2 + 3x + sen^2x * cos^2x)\ dx$$

  fácilmente podemos descomponer en integrales directas.

  $$\int (x^2)\ dx + \int (3x)\ dx + \int (sen^2x * cos^2x)\ dx$$

  Sin embargo, hay otro tipo de problemas que no se pueden descomponer.

  Para los problemas que permiten su descomposición, la técnica "divide y vencerás" es la mejor aplicable:

  dividir el problema en subproblemas hasta obtener un subproblema de solución directa o trivial.

- Analizar si la soluciones son reversibles.

  Dentro del proceso de búsquedas de solución, al dar un paso, es decir, al movernos de un estado del problema a otro, se puede encontrar que ese paso no lleva a la solución que se pretende encontrar.

  Dependiendo del tipo de problema ese paso "mal dado", puede tener repercusiones considerables en la solución. Considérense los siguientes ejemplos:

a) Para probar un teorema matemático, se puede comenzar con algún lema. Eventualmente se descubre que ese lema no ha servido de nada, no obtenemos ninguna repercusión, simplemente se retrocede y se comienza de nuevo con otro lema, y el paso realizado puede ser ignorado.

b) Si tenemos el siguiente problema:

ESTADO INICIAL ESTADO FINAL

Se puede cometer un error al dar un paso, por ejemplo moviendo el número 5 al espacio. Al darse uno cuenta del error, se tiene que retroceder al estado anterior y mover el 6 hacia el espacio.

El paso puede ser recuperable aunque no tan fácilmente como en el ejemplo anterior.

c) En un juego de ajedrez, al dar un paso y darse cuenta hasta dos movimientos después de que ese paso fué erróneo, es un caso en el cuál no se puede regresar y modificar el paso realizado. El paso es no recuperable.

Es entonces que se tienen tres tipos de problemas en cuanto a la recuperación de los pasos:

Ignorables: Donde los pasos de solución pueden ser ignorados.

Recuperables: Donde los pasos de solución permiten regresar y recuperar cierto estado.

No recuperables: Donde los pasos de solución no se pueden modificar.

La recuperación de los pasos de solución de un problema, juega un papel importante en la determinación de la estructura de control. Los problemas ignorables pueden

utilizar una estructura de control simple. Los problemas recuperables pueden utilizar una estrategia de control un poco más compleja. Los problemas no recuperables necesitan una estrategia que dirige su esfuerzo a tomar decisiones, ya que éstas son definitivas (planeación).

- El universo del problema ¿ es predecible ?

  En el ejemplo b) del punto anterior, al resolver el problema se sabe exactamente lo que puede pasar. Se tienen todos los elementos para predecir una solución. Otro tipo de problemas, como por ejemplo el Dominó, no son predecibles, ya que cada paso que se tome dependerá de los pasos que den los oponentes; no existe un estado meta bien definido.

  Entonces se tienen problemas con dos tipos de universo:

  - Predecible
  - Impredecible.

  En el punto anterior, se menciona que los problemas no recuperables necesitan de un proceso de planeación antes de decidir alguna acción. Sin embargo esto es posible si se trata con un universo predecible.

  Los tipos de problemas más difíciles de resolver son los problemas no recuperables de universo impredecible.

- Definir si se desea obtener la mejor solución.

  Hay algunos problemas donde interesa encontrar una solución sin importar la manera en que se obtiene. Sin embargo, hay otros problemas que por sus características requieren de la manera óptima de llegar al estado meta o mejor solución.

- Determinar si la Base de Datos que almacena el conocimiento es consistente.

  Si consideramos que para una demostración matemática la base de conocimientos está formada por lemas, axiomas y teoremas, bien definidos, podemos decir que esta base se encuentra en un mundo consistente.

  Si por el contrario, se habla de un analizador semántico de texto, donde existen palabras que dependiendo del

contexto tienen diferentes significados, se puede decir que se trata de una base que se encuentra en un mundo de inconsistencias.

Hay esquemas de razonamiento adecuados para los dominios consistentes, como por ejemplo, la lógica matemática, sin embargo éstos serianin apropiados para dominios inconsistentes.

- Determinar el papel que juega el conocimiento.

  En un juego de ajedrez el conocimiento requerido, como reglas para realizar movimientos legales de las piezas, permite dar las restricciones en el proceso de búsqueda de la solución.

  Si se trata, por otro lado, de realizar un programa para entender la primera plana de un periódico, se necesitan conocimientos hasta para reconocer lo que sería la solución.

  Hay problemas que utilizan conocimiento únicamente, para restringir y guiar el proceso de búsqueda y problemas donde el conocimiento es tan importante hasta para poder identificar la solución del problema.

- Determinar si se requiere de la interacción de alguna persona en el proceso de solución.

  Hay problemas que no requieren interactuar para dar una solución, sin embargo hay otros que necesitan de la interacción del usuario, o bien para proporcionar algún elemento de guía en el proceso de búsqueda o para proporcionar información adicional.

Al analizar estos siete puntos clave, dentro del problema a resolver, se puede entonces seleccionar una técnica adecuada para solucionar dicho problema.

6.5 Técnicas de búsqueda.

Antes de explicar algunas técnicas especificas de búsqueda es necesario mencionar cinco aspectos importantes de todas ellas.

1) ¿ En que dirección se debe conducir la búsqueda ?

El objeto del proceso de búsqueda es descubrir una ruta a través del espacio del problema que permita llegar al estado meta a partir de una configuración inicial. La búsqueda puede llevarse a cabo en dos direcciónes:

a) Hacia adelante, a partir del estado inicial
Para esto se utiliza el Razonamiento Progresivo cuyos pasos son:

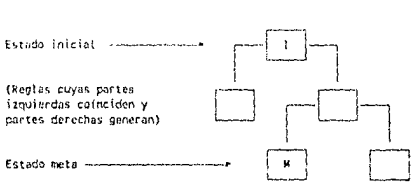
- Generar un árbol cuya raíz es el estado inicial del problema.

- Generar el siguiente nivel del árbol, encontrando todas las reglas cuyos lados izquierdos coincidan con el nodo raíz utilizando las partes derechas para crear nuevos estados (nodos).

- A cada nuevo nodo creado, se le aplicarán reglas cuyas partes izquierdas coincidan, para generar con las partes derechas, un nuevo nivel del árbol.

- Continuar hasta encontrar un estado que coincida con el estado meta.

Como se puede observar gráficamente en el siguiente esquema:



b) Hacia atrás, a partir del estado meta.

Para esto se utiliza el Razonamiento Regresivo, cuyos pasos son:

- Generar un árbol cuya raíz es el estado meta del problema.

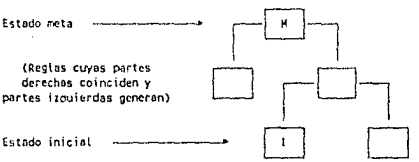
- Generar el siguiente nivel del árbol encontrando todas las reglas cuyas partes derechas coincidan con

el nodo raíz (Estas son las únicas reglas que al aplicarlas darían el estado meta que deseamos obtener), utilizar las partes izquierdas de las reglas para generar el siguiente nivel del árbol.

- Para cada nuevo nodo, encontrar las reglas cuyas partes derechas coincidan y generar el siguiente nivel mediante las partes izquierdas de las reglas.

- Continuar hasta que algún nodo coincida con el estado inicial del problema.

Como se puede observar gráficamente en el siguiente esquema:



Cabe hacer notar que el mismo conjunto de reglas permite razonar tanto progresiva como regresivamente.

Para algunos problemas, es conveniente la dirección hacia adelante y para otros, hacia atrás. Podemos mencionar tres factores para determinar la dirección de la búsqueda:

I Número de estados iniciales vs número de estados meta. Es mas fácil iniciar por el menor número de estados.

II Seleccionar la dirección en la que el"Branching Factor" (promedio del número de nodos que genera un solo nodo) es mejor.

III Si es necesario que el programa justifique el proceso de razonamiento al usuario. Se debe tomar la dirección que tomaría este.

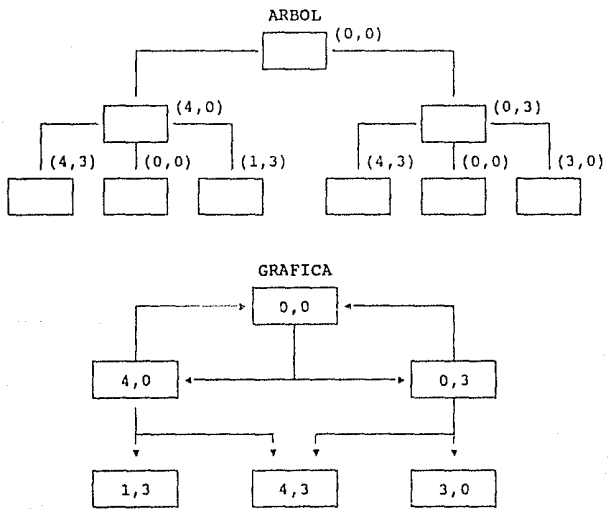
Como ejemplo, la demostración de teoremas matamáticos, se inicia con un teorema por demostrar (estado meta) y

una buena cantidad de lemas y axiomas que se pueden utilizar. Para este ejemplo el razonamiento regresivo es el adecuado ya que existe un solo estado meta bien definido y muchos estados iniciales.

Además para este caso el "Branching Factor" es notablemente mayor razonando hacia adelante. A partir de varios axiomas se pueden generar miles de teoremas, por el contrario, para demostrar un teorema no se necesitan todos los axiomas, sino un número reducido de ellos.

2) Topología del proceso de Búsqueda.

Hasta ahora se ha hablado de árboles como la topología que adquieren los problemas en el proceso de búsqueda. Sin embargo, un árbol es un caso particular de una gráfica, con la diferencia que en un árbol se pueden tener nodos repetidos, mientras que en una gráfica no, por ejemplo:



Al utilizar una topología de gráfica en el proceso de búsqueda, reducimos el esfuerzo requerido para explorar esencialmente la misma ruta varias veces, pero a su vez, necesitamos esfuerzo adicional cada vez que se genera un nodo al verificar si éste existe o no. Dependiendo del problema este esfuerzo adicional se puede justificar o no. Si el mismo nodo se genera de diferentes maneras es más conveniente utilizar gráficas, y si se genera eventualmente, será mas conveniente la utilización de árboles, debido a su simplicidad.

3) Representación de cada nodo en el proceso de búsqueda.

A lo largo de este capítulo se ha descrito el proceso de búsqueda como un mecanismo para viajar alrededor de un árbol o una gráfica donde cada nodo representa un punto en el espacio del problema. Sin embargo, se debe representar a cada nodo individualmente.

En dominios complejos como el mundo que nos rodea, es importante analizar lo siguiente:

- La representación de objetos y hechos individuales.
- La combinación de objetos individuales para obtener una representación completa del espacio del problema.
- La representación eficientemente de secuencias de estados del problema que surgen del proceso de búsqueda.

4) Selección de reglas aplicables ("Matching").

Hasta ahora, no se ha explicado en que forma seleccionar aquellas reglas que pueden ser aplicadas en un cierto estado del problema. Para realizar esta selección se requiere de un tipo de mecanismo que permita cotejar entre el estado actual y las condiciones de las reglas (este punto es crítico en la construcción de sistemas basados en reglas "Rule Based Systems").

Algunos tipos de "Matching" son los siguientes:

- "Matching" indexado.

  En vez de buscar en todo el conjunto de reglas, éstas

se pueden indexar por medio del estado actual del problema.

- "Matching" con variables.

  Cuando se utiliza un "Matching" no literal, es importante no sólo reconocer el "Match" entre el patrón de la regla y el estado del problema, sino también es importante saber que asignaciones de variables fueron hechas justo en el momento en el que ocurrió el "Match".

- "Matching" por aproximación.

  Cuando hay reglas que contienen precondiciones no explícitas en algún estado del problema, se requieren de mecanismos de "Matching" que permitan inferir precondiciones.

  Este "Matching" se vuelve más complejo cuando se tienen aproximaciones a las precondiciones de las reglas, sobre todo cuando se trabaja con descripciones físicas del mundo que nos rodea (la regla más parecida).

El resultado del proceso de "Matching" es una lista de las reglas cuyas partes izquierdas coinciden con el estado actual del problema. El método de búsqueda puede entonces escoger entre todas, la regla específica y el orden en que realmente se aplicará.

La manera en la que interactúan el proceso de "Matching" y el de búsqueda depende de la estructura de ambos.

5) Utilización de una función heurística para guiar la búsqueda.

Se tienen dos formas de incorporar la información heurística en el proceso de búsqueda basado en reglas:

- Incluyéndola dentro de las mismas reglas.

- Como una función heurística que evalúa estados individuales del problema y determina que tan adecuados son.

Una función heurística traduce de una descripción del estado del problema a una medida, generalmente numérica, de que tan aceptable es ese estado en especial.

El propósito de las funciones heurísticas es el de guiar al proceso de búsqueda hacia la dirección más prometedora como solución, sugiriendo qué ruta escoger entre todas las posibles. Entre mas acertada sea la función para estimar los verdaderos méritos de cada nodo en el árbol o gráfica de búsqueda, más directo será el proceso de solución del problema. En el extremo que la función sea perfecta, no se requiriría de un proceso de búsqueda, pues la función heurística proporcionaría una solución directa.

En la mayoría de los problemas no es fácil la definicion de funciónes heurísticas, solo se obtienen aproximaciones adecuadas para ciertas aplicaciones.

A continuación se muestran algunas de las estrategias de control, de propósito general, comunmente conocidas como "Weak Methods" (métodos debiles), llamados asi porque no soportan la explosión de combinaciones que presentan algunos problemas (los métodos de búsqueda son generalmente vulnerables a la explosión de combinaciones).

Aunque se ha observado la limitada eficacia de los métodos para resolver problemas difíciles por si mismos, éste ha sido uno de los resultados más importantes que surgieron de las dos últimas décadas en investigación, sobre Inteligencia Artificial, y continúan formando el centro en la mayoría de los sistemas inteligentes en combinación con otros métodos La correcta elección de alguna técnica en especial, dependerá de las características del problema especifico cuyo analisis se mencionó anteriormente.

6.5.1 "Generate and test"

Esta estrategia de control es la más simple de todas, la cual consiste de los siguientes pasos:

1) Genera una posible solución.

   Ya sea generar un punto particular en el espacio del problema o bien, generar una ruta solución a partir de un estado inicial.

2) Prueba si ésta es realmente una solución, comparando el punto generado con el conjunto de metas o soluciones aceptables.

3) Si es una solución, terminar. De lo contrario regresar al punto 1).

Si la solución existe eventualmente se encontrará con este método. Este algoritmo es un procedimiento de búsqueda "Depth_first", dado que debe darse una solución completa antes de que pueda ser probado. En forma sistemática se reduce a hacer una búsqueda exhaustiva en el espacio de estados. También se tiene la posibilidad de generar soluciones aleatoriamente con su correspondiente prueba, pero esto no garatiza que se llege a una solución.

La forma más directa y segura de implementar este método es con un árbol de búsqueda "depth_first" con evaluación y retroceso (Backtracking).

Como método aislado no presenta ventajas significativas, pero al utilizarlo en combinación con otros métodos se pueden sobre llevar las limitaciones y crear uno más efectivo.

6.5.2 "Breath-first-search".

En este procedimiento todos los nodos en cada nivel del árbol son examinados antes de pasar al siguiente nivel. Un procedimiento de búsqueda breadth-first garantiza el encontrar una solución si ésta existe, considerando que hay un número finito de ramas en el árbol. Si existe solución, entonces existe un ruta de longitud finita y el procedimiento es el siguiente:

- Revisar todas las rutas de longitud uno.

- Revisar todas las rutas de longitud dos, y así sucesivamente hasta encontrar el nodo o estado solución y como consecuencia la solución que tiene la ruta más corta.

Este procedimiento de búsqueda tiene los siguientes inconvenientes:

- Requiere mucha memoria.

- Realiza mucho procesamiento, aún para soluciones cortas.

- Incremento notable del número de nodos si se utilizan operadores redundantes o irrelevantes.

Por estas razones este método esparticularmente inapropiado en la búsqueda de soluciones las cuáles tengan un número excesivo de rutas de longitud superior a dos nodos.

6.5.3 "Best-first-search".

En este método se busca combinar las ventajas de los métodos depth-first y breath-firsrt en uno solo. En cada paso de este procedimiento de búsqueda se selecciona elnodoque sea más adecuado, medido por una función heurística apropiada para cada uno de los nodos analizados.

Después de seleccionar el nodo más adecuado, según la función, se toma como nuevo punto inicial para continuar el proceso. Cuando se concluye que no se llegó a una solución porque se encuentre un nodo menos adecuado en ese nivel, se regresa al nivel superior y se selecciona el siguiente nodo que en orden de prioridad continué del antes seleccionado,iniciando nuevamente el proceso.

Toda la información de los estados analizados, en el transcurso del proceso es almacenadas para obtener la información de la ruta de solución, además de los resultados de las funciones heurísticas para considerar el siguiente nodo de búsqueda. Llegando al final de proceso al encontrar el nodo que satisfaga los requerimientos del problema.

Un método gráfico es muy útil como auxiliar pues facilita el seguimento manual del método.

6.6 Representación del Conocimiento.

Para resolver problemas complejos de Inteligencia Artificial, se requiere de una base amplia de conocimiento y mecanismos de manipulación para encontrar soluciones adecuadas.

Hasta ahora se han visto mecanismo generales de manipulación del conocimiento utilizando búsquedas. Estos métodos son muy generales y por su generalidad son limitados. Existe una variedad de formas de representar el conocimiento (hechos) que han sido explotadas por sistemas de Inteligencia

Artificial. Antes de explicar algunas formas de representación del conocimiento, revisemos dos entidades involucradas en dicha representación:

- Hechos: Verdades en palabras relevantes. (Estas son las cosas que se quieren representar)
- Representación de hechos utilizando algún formalismo elegido. (estas son las cosas que realmente se pueden manipular)

Deben existir entonces funciones que tradiscam de los hechos a su representación y viceversa. Una representación de hechos común para todos y que merece especial atención, es la utilización del lenguaje natural.

6.6.1 Representación de hechos simples en lógica.

Una manera particular de representar hechos es el lenguaje de la lógica. Partiendo del lenguaje natural, a través de la lógica obtenemos una representación formal de hechos. Por ejemplo, tenemos la siguiente proposición lógica:

Rufus es un perro	⟶	Perro (Rufus)
(Lenguaje Natural)		(Formalismo en lógica de proposición)

El formalismo de la lógica es atractivo por la facilidad que presenta para derivar nuevos conocimientos a partir de cierto conocimiento previo: Deducción Matemática. Con este formalismo se puede derivar un nuevo hecho verdadero, que proviene lógicamente de otros hechos previamente conocidos. Esta capacidad de deducción se observa como una manera de obtener respuestas a preguntas y soluciones a problemas. La utilización de la representación del conocimiento en lógica permite solucionar problemas.

- Lógica de predicado.

  Podemos representar hechos en forma de proposiciones lógicas:

Está lloviendo	LLOVIENDO
Cuando llueve no hay sol	LLOVIENDO ⟶ NO SOL

Sin embargo, las proposiciones lógicas se ven limitadas a representar hechos muy simples.

Existe otro formalismo conocido como lógica de predicado, que permite representar una mayor gama de hechos. La lógica de predicado es un buen medio de representación del conocimiento y provee una manera de deducir nuevos hechos.

Los mecanismos de deducción utilizando representación en lógica de predicado se pueden observar como tratar de probar un teorema propuesto (solución del problema) utlizando ciertos axiomas (conocimiento previo). Puede darse el caso que este mecanismo de deducción no llegue a demostrar el teorema (solución) propuesto. A continuación se muestran ejemplos de representación de hechos en forma de lógica de predicado:

Marco fué un hombre

hombre (marco)

Esta representación captura el hecho crítico de "ser hombre" y omite la idea del tiempo (fué). Esta omisión es aceptable o no, dependiendo del uso que se le pretende dar al conocimiento.

Marco era de Pompeya

Pompeya (marco)

Todos los hombres de Pompeya eran Romanos

Todos X Pompeya (X) ⟶ Romano (X)

Todos los Romanos eran leales a Cesar o lo odiaban

Todos X Romano (X) ⟶ leal (X,Cesar) o odia (X,Cesar)

El proceso de convertir oraciones de lenguaje natural a fórmulas en lógica de predicado puede ser un proceso no trivial, por los siguiente:

1) Algunas oraciones pueden ser ambiguas.

2) La elección de los hechos críticos a representar, dependerá del uso que se le dé al conocimiento.

3) En ocasiones, las oraciones no contienen toda la información necesaria para razonar sobre cierto tema, es necesario entonces que contengan la información que es considerada obvia para la gente, pero que es estricamente necesaria para poder deducir mediante los hechos representados.

Uno de los mecanismos de deducción utilizando representación en lógica de predicado es el conocido como "Resolución".
Resolución, es un procedimiento que permite razonar con declaraciones en lógica de predicado. Parte de la idea de demostrar que la negación de un hecho, produce una contradicción con los hechos ya conocidos.

Una de las dificultades que presenta la representación de hechos en lógica de predicado es la existencia de problemas con cierto tipo de información que no es representable por medio de esta técnica. Nuevamente llegamos al punto de dependencia entre el problema especifico y la elección de una técnica adecuada de solución, como se muestra en los siguientes ejemplos no representables en lógica de predicado:

"El dia de hoy, hace mucho calor"

No podemos representar grados relativos de calor, qué tanto es "mucho".

"Si no hay evidencia de lo contrario, asume que cada adulto que conoces sabe leer"

No podemos representar un hecho que debemos inferir a partir de la ausencia de otros.

"Es mejor tener más piezas en el tablero de ajedrez, que el número de piezas que nuestro oponente tiene"

No podemos representar información hurística.

Existen varias técnicas de representación del conocimiento que tratan de enfocar estos problemas como son:

- Razonamiento no monótono (Nonmonotic logic)

- Razonamiento probabilistico.

6.6.2 Razonamiento no monótono.

Los sistema basados en lógica de predicado son monótonos en el sentido que el número de declaraciones (hechos) conocidos como verdaderos estrictamente crece con el tiempo. Nuevos hechos se agregan a la base de conocimientos por los mecanismos de deducción. Cuando no se puede probar la veracidad de un hecho, éste simplemente es descartado.

El trabajar con un sistema "monótono" en este sentido tiene algunas ventajas, como son:

- Cuando agregamos un nuevo hecho no es necesario verificar su consistencia pues proviene lógicamente del conocimiento previo.

- No es necesario recordar la serie de hechos utilizados en la deducción de un nuevo hecho, porque no hay riesgo de que alguno de estos hechos desaparezca de la base de conocimiento.

Sin embargo, estos sistemas limitan la representación en dominios de problemas que presentan:

- Información incompleta

- Situaciones cambiantes

- Generación de hechos que se asumen

Cuando la información que se manipula está incompleta, se pueden adivinar ciertos hechos mientras no exista una evidencia de contradicción. Esta forma de "adivinar" hechos a partir de la ausencia de otros se conoce como Razonamiento por Omisión, por ejemplo si alguien nos invita a cenar, y en el camino pasamos por una florería, pensamos que sería buen detalle obsequiarle un ramo de flores al anfitrión. Nosotros no sabemos precisamente si al anfitrión le gustan las flores, pero intuitivamente utilizamos una regla general:

"Dado que a la mayoría de la gente le gustan las flores, al anfitrión le gustarán, a menos que él haya evidenciado lo contrario".

Este tipo de razonamiento es no monótono porque los hechos que se derivan provienen de la falta de información del resto de los hechos.

Una técnica común de Razonamiento por Omisión es la conocida como "la elección más probable" (the most probable choice). Si sabemos que un hecho, dentro de un conjunto, debe ser verdadero, en la ausencia de información, escogemos el que más se le parezca.

Una descripción precisa del razonamiento por omisión, debe relacionar la falta de información X con una conclusión Y, por ejemplo:

Definición 1. Si X no se conoce, entonces concluye Y.

Definición 2. Si X no se puede probar, entonces concluye Y.

Definición 3. Si X no se puede probar, en cierto tiempo, entonces concluye Y.

6.6.3 Razonamiento probabilístico.

Hasta ahora, se han definido hechos que son ciertos, o bien falsos, o bien hechos que desconocemos. Hay que considerar ahora la posibilidad de tener un hecho que "probablemente es cierto".

Hay tres tipos de situaciones que se prestan para utilizar razonamiento probabilistico:

a) El mundo relevante del problema es realmente aleatorio, por ejemplo:

   La distribución de electrones en un átomo.

b) El mundo relevante del problema no es aleatorio si se cuenta con información suficiente, pero es difícil de obtener.

c) El mundo del problema aparenta ser aleatorio porque no se tiene descrito en un nivel adecuado, por ejemplo:

   El reconocimiento de patrones de voz.

En este caso la teoría matemática de probalbilidad provee una manera de describir y manipular conocimiento o hechos inciertos.

6.7 Herramientas para la implantación de Sistemas Inteligentes.

Como se mencionó el estudio de los problemas de Inteligencia Artificial, ha permitido el desarrollo de nuevas técnicas de programación que se basan en el procesamiento simbólico, mecanismos de deducción, así como representación y manipulación del conocimiento.

Se han discutido algunas de las características que presentan los problemas y de las diversas técnicas de solución que se pueden aplicar. Ahora se mencionarán algunas de las herramientas de programación con que se cuenta en la actualidad para implantar sistemas inteligentes.

Las características deseables en cualquier lenguaje de programación para solucionar problemas, incluyendo aquellos enfocados a la Inteligencia Artificial se resume en los siguientes puntos:

- Una variedad de "data types" para poder describir todos los tipos de datos que un sistema de información complejo necesita.

- Habilidad para descomponer el sistema en unidades más pequeñas y que sea razonablemente fácil el efectuar modificaciones (modularidad).

- Control flexible sobre las estructuras, que facilite recursión y descomposición paralela del sistema.

- Habilidad para comunicarse interactivamente con el sistema, tanto para desarrollo, como para el uso efectivo de sistemas finales.

- Habilidad para producir código eficiente para que el rendimiento (performance) sea aceptable.

Es interesante observar que aunque algunas de estas características se consideran importantes ahora en cualquier sistema de información razonablemente complejo, algunas de ellas fueron requeridas inicialmente para sistemas de Inteligencia Artificial[K].

Se ha mencionado la gran dependencia que exite entre el problema y la estrategia de solución. Los diversos problemas de Inteligencia Artificial determinan las características deseables de las nuevas técnicas de programación, siendo las más generales:

- Facilidad para la manipulación de listas, ya que éstas son la estructura más utilizada en casi todos los programas de Inteligencia Artificial.

- Transparencia referencial, es decir, si para alguna variable no se ha asignado valor en el programa, no hay la necesidad de almacenar el nombre de dicha variable, sino hasta que ésta adquiere un valor.

- Facilidad para evaluar la coincidencia de patrones (Pattern Matching), tanto para identificar tipos de datos como para el control. La coincidencia de datos es una parte importante en la utilización de una base de conocimientos compleja, mientras que la coincidencia para el control forma las bases de la ejecución de un sistema en producción.

- Facilidad para realizar algún tipo de deducción automática y para almacenar en alguna base de datos el resultado de dichas deducciones.

- Facilidad para construir estructuras complejas de conocimiento.

- Mecanismos por medio de los cuáles el programador pueda agregar conocimiento adicional.

- Estructuras de control que faciliten un ambiente orientado a "metas".

No existe ningún lenguaje de progamación que contenga todas las características anteriores. Algunos lenguajes cumplen mejor algunas características que otros. A continuación se presenta una breve descripción de los lenguajes que proporcionan algunas de las características mencionadas.

IPL (INFORMATION PROCESSING LANGUAGE): Desarrollado por Newell en 1960. Fué el primer lenguaje que introdujo el procesamiento de listas. Marca el inicio del desarrollo de lenguajes de programación enfocados a la Inteligencia Artificial.

LISP (LIST PROCESSING): Es uno de los lenguajes más utilizados en Inteligencia Artificial,y su estructura principal es la lista.

INTERLISP Es un dialecto de LISP.

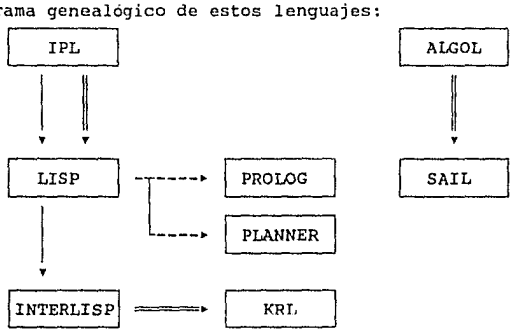
SAIL Es una derivación de ALGOL con algunas características adicionales, incluyendo el soporte primitivo de memoria asociativa (direccionamiento por contenido). De todos los lenguajes de Inteligencia Artificial, SAIL es el más similar a los lenguajes tradicionales de propósito general.

PLANNER Diseñado para representar características tradicionales, así como un ambiente análogo a "metas".

KRL (Knowledge Representation Language): Lenguaje que soporta estructuras complejas del conocimiento.

PROLOG (PROGRAMMING IN LOGIC): Lenguaje basado en hechos y reglas.

Diagrama genealógico de estos lenguajes:



——→ Agregaron construcciones de mayor nivel.

------→ Agregaron mecanismos de deducción automática.

═══→ Agregaron mecanismos para estructurar el conocimiento.

La figura anterior muestra la evolución de estos lenguajes, cada uno con características específicas. Dependiendo del tipo de problema a resolver, algunos son más convenientes que otros.

Las nuevas técnicas de programación han tenido gran impacto sobre la programación tradicional de propósito general, no solo en aplicaciones de Inteligencia Artificial sino en aplicaciones de cualquier tipo.

La programación tradicional desde el lenguaje binario hasta los lenguajes de alto nivel como BASIC o PASCAL, se caracteriza porque el programador debe describir completamente la manera en que se desean obtener resultados en vez de describir lo que se desea obtener como resultado. A esta familia de lenguajes, como BASIC, PASCAL o ADA, se les conoce como lenguajes imperativos.

Los lenguajes imperativos estan constituidos de comandos que especifican paso a paso las acciones a realizar, explícitamente detallan el flujo de control necesario para obtener un resultado, ésto es, están orientados a la máquina en la que se ejecutan.

Los lenguajes declarativos, al contrario de los imperativos, intentan separar la tarea a realizar de la manera en que la computadora la realizará. Los lenguajes declarativos permiten describir un conjunto de relaciones o funciones a evaluar, y la ejecución de un programa declarativo, se reduce al uso de las definiciones para obtener los resultados deseados. La forma en que se obtendrán dichos resultados es tarea del lenguaje que se utilice.

Los lenguajes LISP y PROLOG son ejemplos de lenguajes declarativos. PROLOG tiene características adecuadas a los requerimientos del sistema normalizador, objeto de este trabajo.

PROLOG remonta sus orígenes a principios de los años 70's, desarrollado por Alain Colmerauer y Phillippe Roussel en Marsella, Francia,[14] como una implantacion física de la programación en lógica.
La programación en lógica surge por los esfuerzos realizados alrededor de los años 50's para automatizar la demostración de teoremas matemáticos a través de una computadora.

En PROLOG se representa el conocimiento mediante una variante de lógica de predicado, con oraciones que toman la forma:

CONCLUSION SI CONDICIONES

Este tipo de oraciones son conocidas como cláusulas de Horn, en honor al logista Alfred Horn, quien estudió sus propiedades lógicas, por ejemplo:

te mojarás si está lloviendo y no traes paraguas

(CONCLUSION) SI (CONDICIONES)

Por otro lado, utiliza el mecanismo de deducción de razonamiento regresivo, ya que éste es lo suficientemente mecánico como para implementarse en una computadora.

Los mecanismos de "matching" utilizados por PROLOG están dados por el algoritmo de unificación, una técnica de "matching" con variables.

PROLOG es un caso práctico de la aplicación de varias técnicas de Inteligencia Artificial, dando como resultado una herramienta importante para implantación de Sistemas Inteligentes.

II PAQUETE NORMALIZADOR

1 DESARROLLO

1.1 Utilidad de un paquete normalizador

Aún cuando se reconoce la importancia de la normalización como uno de los pasos dentro del diseño de una Base de dato, son pocas las personas que realmente comprenden y usan los algoritmos de normalización en la práctica[6].

Descuidar este paso del diseño puede traer serias complicaciones mas adelante, cuando el sistema se encuentra en producción, en el momento que se necesiten hacer cambios a la Base de Datos o en el momento de actualizarla.

La utilización de estos algoritmos asegura que no se pasó por alto ninguno de los puntos en el proceso de normalización.

Resulta casi imposible que todos aquellos que se dedican al diseño de Bases de Datos dominen perfectamente los algoritmos. En la medida que una base de datos aumenta de tamaño, su normalización se vuelve mas complicada, a grado tal que el consumo de recursos puede llegar a límites inaceptables. Entiéndase por tamaño de una Base de Datos al número de atributos y de dependencias funcionales y no al volumen de información.

Es por todo esto que resulta de gran utilidad contar con un paquete que le facilite al diseñador la tarea de la normalización.

El usuario de un paquete de esta naturaleza no necesita ser un experto en normalización para obtener los resultados deseados en poco tiempo. Basta con que entienda suficientemente bien el problema, que conozca los atributos a manejar y las dependencias entre ellos.

Por otro lado, se puede realizar este importante paso en el proceso del diseño de una Base de Datos en un equipo diferente del que se utilice para la implementación del sistema.

1.2 PROLOG y el enfoque relacional

El enfoque de Bases de Datos Relacionales se ha caracterizado de los demás, porque introduce terminología matemática. De hecho la teoría de Bases de Datos Relacionales, se fundamenta en la teoría de las Relaciones.

Esta liga matemática entre la teoría Relacional y "PROLOG" hace a este último, una herramienta ideal para implementar los algoritmos de normalización de Relaciones[6].

Los conceptos en los que dichos algoritmos se apoyan, son fácilmente representables. Estos se reducen, a la verificación de ciertas condiciones, como el tipo de dependencias entre conjuntos de atributos, y el razonamiento regresivo hace fácil su implementación.

Por características propia de "PROLOG", la codificación es sencilla, tiene facilidad modular y menor número de lineas de código.

Además de estos aspectos importantes, existen en el mercado una gran variedad de interpretes y compiladores de "PROLOG" lo cual lo hace accesible y versátil. Particularmente, se utilizará Turbo "PROLOG" por la facilidad de utilización en computadoras personales.

A continuación se presenta formalmente la estructura y las características de los componentes de un programa en "PROLOG".

Un programa en "PROLOG" consiste de una descripción de problema que se pretende resolver. Esta descripción está constituida por tres componentes:

1) Nombres y estructuras de objetos involucrados en el problema.

2) Nombres de loas relaciones que existen entre los objetos.

3) Hechos y reglas describiendo estas relaciones.

La descripción dentro de un programa, se utiliza para especificar la relación que se desea obtener entre los datos de entrada y los datos que se generan a la salida. Esta descripción consiste de una lista de oraciones lógicas que se pueden representar en forma de hechos:

ESTA LLOVIENDO

o en forma de reglas:

TE MOJARAS SI ESTA LLOVIENDO Y NO TIENES PARAGUAS

"PROLOG" permite deducir hechos a partir de reglas y hechos previos:

A ELENA LE GUSTA EL TENIS

A JUAN LE GUSTA EL FOOT-BALL

A ELENA LE GUSTA X SI A JUAN LE GUSTA X

deducimos entonces que:

A ELENA LE GUSTA EL FOOT-BALL.

La ejecución de un programa en "PROLOG" es controlada automáticamente. Cuando un programa es ejecutado, "PROLOG" trata de encontrar todos los posibles conjuntos de valores que satisfacen la meta a alcanzar. Esto se logra a través del mecanismo de "BACK TRACKING" que posteriormente se explicará.

Entrando en mayor detalle la estructura básica de un programa en Turbo "PROLOG" es la siguiente.

DOMANIS

PREDICATES

GOAL (opcional)

CLAUSES

Existen algunos conceptos generales que son utilizados en la estructura anterior. Las características de éstos se explican a continuación:

- Variables

  Los nombres de variables deben empezar con mayúscula, aceptando en seguida cualquier número, letra (mayúscula o

minúscula) o el símbolo "subrayado" "_", por ejemplo:

```
Mi_nombre
```

Existen dos tipos de variables libre y limitada.

Las variable libre es aquella de la cual Turbo "PROLOG" no conoces su valor pues no fue asignado directamente, por ejemplo:

```
Gustar(X,leer) and Gustar(X,nadar)
```

donde X es la variable libre ya que su valor dependerá de la consulta que se haga y de las cláusulas que tengan.

La variable limitada es aquella de la cual Turbo "PROLOG" conoce su valor, por ejemplo:

```
Gustar(juan,leer)
```

donde el 1er y 2do argumento son variables limitadas ya que su valor es conocido.

- Objetos y relaciones

Se pueden definir hechos, los cuales se expresan en la sección CLAUSES y que consisten de una relación que afecta a uno o mas objetos, por ejemplo:

```
Relación = gustar
Objetos = Pepe y nadar
```

entonces podemos tener un hecho en donde se tenga la relación Gustar asociada a Pepe y nadar, en otra forma:

```
Gustar(Pepe,nadar)
```

lo cual se va a interpretar como "a pepe le gusta nadar".

- Objetos

El nombre de objetos debe comenzar con alguna letra minúscula seguida por cualquier numero letra (mayúscula o minúscula) o el símbolo subrayado "_", por ejemplo:

```
mi_casa
```

- Relaciones

  El nombre de relaciones puede ser cualquier combinación de letras (mayúsculas o minúsculas), números o símbolos.

- Dominios y Predicados.

  Dentro de un programa se pueden especificar caracteristicas con las cuales se va a trabajar. Estas son:

  - DOMAINS

    Especifica el dominio al cual pertenecen los objetos de una relación. Los dominios estandares que se manejan son:

    - Character (char)
    - Integer
    - Real
    - Symbol
    - File
    - Dbasedom

    Si se utilizan sólo dominios estándar en las declaraciones de predicados, el programa no necesita una sección de dominios.

    Se permite tener varias declaraciones para un mismo predicado (declaraciones múltiples); debe seguir una de otra y todas deben tener el mismo número de argumentos, por ejemplo:

    ```
    add(integer, integer)

    add(real,real)
    ```

  - PREDICATES

    Consiste de una o más cláusulas que indican las relaciones y objetos involucrados en el programa, por ejemplo:

    ```
    Domain    persona,actividad=symbol

    Predicate   Gustar(persona,actividad)
    ```

    lo que se interpreta que se está especificando la

relación Gustar que afecta los objetos persona y actividad, los cuales tienen un dominio simbólico, esto es, tendrán valores asignados por el usuario que pueden ser tomados de un archivo, donde las consultas se harán dando algunos de estos símbolos como entrada, es decir:

Gustar(pepe,beisbol) salida —→ true/false

dependiendo de los datos cargados, o también,

Gustar(x,beisbol) salida —→ Todos los nombres de personas que les gusten el beisbol.

También dependiendo de los datos que se tengan cargados.

- Regla

Dentro de esta misma sección se tiene el concepto de "REGLA", la cual tiene la función de relacionar o agrupar de manera condicional un conjunto de relaciones, por ejemplo:

```
Relación(obj1,obj2..objn) if
Relación(obja,objb..objn) and
Relación(objx,objy..objz).
```

- Objetos Compuestos.

Para simplificar la asignación de valores a las relaciones, (CLAUSES) Turbo "PROLOG" tiene implementado lo que se conoce como objetos compuestos, lo cual significa hacer objetos que contengan otros objetos, de tal manera que al definirlos se manejen exactamente igual que un objeto sencillo.

El primer componente de un objeto compuesto es lo que se conoce como "FUNCTOR", el cual es utilizado para diferenciar entre varios objetos, seguidos por otro conjunto de objetos, por ejemplo:

```
propietario(Juan,libro(nombre,autor))
```

en donde propietario de el "FUNCTOR" que hace la distinción, entre diferentes objetos compuestos.

- Declaración del Dominio de Objetos Compuestos.

  La manera de declarar el dominio a objetos compuestos, es utilizando los dominios alternos:

  ```
  DOMAIN = alter1(d,..,d);alter2(d,..,d);..
  ```

  donde alter1 , alter2, .. son los "FUNCTORS" y d es una lista de diferentes dominios estándar o definidos externamente, como ejemplo:

  ```
  DOMAINS
  articulos = libro(nombre,autor);
  carro(marca;
   bote;
   teléfono

  nombre, autor, bote, teléfono = symbol
  ```

- Recursividad.

  La recursividad es una técnica de programación importante para Turbo "PROLOG", esta es utilizada normalmente en dos situaciones:

  1) Cuando las relaciones son descritas con la ayuda de ellas mismas

  2) Cuando objetos compuestos son parte de otros objetos compuestos, esto es objetos recursivos. El utilizar objetos recursivos permite definir objetos en donde no se sabe con anterioridad el número de elementos que lo formaran.

- Listas

  Las listas son parte de la estructura básica de los programas en Turbo "PROLOG" éstas, por su fácil representación se han convertido en estructuras de uso común. Los elementos dentro de ellas se separan por comas "," y encerrados entre corchetes "[y]". Pero es importante hacer notar que todos los elementos deben pertenecer al mismo dominio.

  Al momento de procesamiento la lista se divide en dos partes, estas son: head y tail, encabezado y resto

respectivamente, es decir la primera parte o encabezado contiene al primer elemento de la lista y la segunda parte o resto contiene, como su nombre lo indica el resto de los elementos.

- Proceso de Unificación

  Es el proceso que Turbo "PROLOG" utiliza para encontrar coincidencias de alguna consulta contra una de las metas a través de alguna de las cláusulas definidas.

  Este mecanismo se utiliza en:

  - La asignación de valores a variables
  - Acceso a estructuras de datos mediante patrones de coincidencia
  - Algunos tipos de igualdad.
  - Control de Búsqueda de soluciones.

Al evaluar algún objetivo, Turbo "PROLOG", sabe cuales subobjetivos han sido satisfechos y cuales no. La búsqueda de subobjetivos a satisfacer puede ser representada por un árbol.

Esta búsqueda sigue los siguientes principios básicos:

1) Las submetas deben de satisfacerse de izquierda a derecha.

2) Las cláusulas deben ser probadas en el orden en el que aparecen en el programa.

3) Cuando una submeta coincide con la parte izquierda de alguna regla, la parte derecha de la regla debe de satisfacerse como siguiente punto. La parte derecha de la regla constituye un nuevo conjunto de subobjetivos.

4) Un objetivo ha sido satisfecho cuando se ha encontrado un hecho que coincida con todas las extremidades del árbol de metas. Si todos los subobjetivos han sido satisfechos, hubo una coincidencia para todos dentro del conjunto de cláusulas.

Es entonces así, que turbo "PROLOG" ofrece las caracteristicas necesarias para implementar cualquier programa que trate de dar solución a problemas de Inteligencia Artificial.

1.3 Descripción

El sistema normalizador es una herramienta aplicable a una parte del diseño de Bases de Datos relacionales; que como ya se mencionó persigue la eliminación de anomalías que se dieran al momento de acceso de la Base de Datos y de redundancias que compliquen la actualización de atributos en las diferentes relaciones.

El sistema esta desarrollado con técnicas de Inteligencia Artificial que facilitan el proceso de normalización, mediante el uso de listas y rutinas de búsqueda entre otras.

El lenguaje utilizado es Turbo "PROLOG" el cual presenta las características antes mencionadas.

El mecanismo de operación es muy sencillo, el cual esta dirigido no solo a especialistas en sistemas sino también a usuarios finales.

El usuario que utilice este sistema tiene la facilidad de realizar diversas pruebas, es decir, cambiar la semántica de los atributos a normalizar y con esto lograr mejores resultados y más adecuados a la realidad que los gobierna.

El sistema normalizador se ha elaborado concientemente de las limitaciones teóricas y prácticas que tiene. Estas consideraciones provocaron que algunos conceptos novedosos y útiles no se hayan incluido como característica de este sistema. Existen varias razones por las cuales se hizo de la forma y cobertura que actualmente tiene, estas son:

- La teoría de normalización contempla como nivel máximo la quinta forma normal (5NF), para la cual no existe, en la actualidad, algoritmos que la puedan soportar a nivel automatizado, por lo que el normalizador se limitó a cubrir hasta la tercera forma normal (3NF) que si está respaldada con algoritmos prácticos y la cual consideramos como etapa aceptable dentro del proceso de normalización.

  Para esta determinación se tomaron en cuenta la dificultad de la implantación a una etapa superior (BCNF, 4NF, 5NF) contra los beneficios que brindaba y se concluyó que era demasiado esfuerzo para muy escasos beneficios, este esfuerzo se compone de:

- Programación extensiva y muy elaborada.

- Desarrollo de algoritmos que cubran el normalizador en BCNF, 4NF y 5NF. Aunque existen algoritmos para llegar a BCNF, tampoco brindan los suficientes beneficios para implantarlos. Los algoritmos para 4NF y 5NF son extremadamente más complicados que los ya mencionados, dada la necesidad de realizar un análisis profundo de aspectos de semántica en la relación específica que se quiera normalizar. Los beneficios que obtendrían, suponiendo que se tuvieran los algoritmos adecuados, sería mínima, ya que estas formas normales son muy específicas respecto a la circunstancias en que estas aplican.

La operación del sistema se realiza con la interacción del usuario quien en una pantalla inicial deberá introducir los atributos determinantes y los determinados, es decir las dependencias entre atributos.

Una vez capturada la información el sistema procede al proceso de normalización, el cual genera como resultado en número mínimo de relaciones indicando la llave y los atributos dependientes.

1.4 Diseño mediante Algoritmos de Normalización

La base de datos en la que se apoya el sistema Normalizador, contiene los siguientes hechos:

a) Esquema de la relación.

Esta es almacenada de la siguiente forma:

```
esquema(nombre_de_la_relación,[atrib1..atribn])
```

El predicado "esquema", contiene el nombre de la relación, seguida por un conjunto ordenado de atributos. Cabe hacer notar que es un conjunto ordenado de atributos el cual no tiene elementos duplicados y no un "lista" que podría llegar a tenerlos, al tener lo atributos ordenados disminuimos considerablemente el tiempo de proceso aunque el orden no tiene ningún significado semántico.

b) Dependencias Funcionales.

Representadas por el siguiente predicado:

```
depfun(nombre_de_la_relación,
       [atrib1..atribi],[atribj..atribnbn])
```

La primera posición del predicado contiene el nombre de la relación que posee a la dependencia funcional; el primer conjunto de atributos contiene la parte determinante de la dependencia funcional, mientras que el segundo continúe los atributos determinados. Los algoritmos que se muestran posteriormente requieren que las partes determinadas de las dependencias funcionales contengan un solo atributo. (A la interface de entrada se le han agregado reglas para que realice esta transformación).

c) Llaves.

Representadas de la siguiente manera:

```
Llave(nombre_de_la_relación[atrib1..atribn])
```

La primera posición contiene el nombre de la relación y la segunda presenta el conjunto de atributos que forman parte de la llave de dicha relación.

d) Hechos sobre la descomposición.

Se representan hechos de descomposición, cuando una relación ha sido descompuesta en varias relaciones y se representa:

```
descomp(nombre_de_la_relación_original,
        nombre_de_la_relación_final)
```

La relación original, se descompone en varias relaciones finales, y para cada una de ellas se inserta un nuevo hecho de descomposición.

Otro hecho sobre la descomposición es:

```
en3afn(nombre_de_la_relación)
```

que permite insertar a la base de datos las relaciones producto de la descomposición que se encuentran en tercera forma normal.

Los puntos a) y b) constituyen la entrada inicial de datos proporcionada por el usuario a través de la interface desarrollada. Esta interface verifica que todos los atributos que forman una dependencia funcional estén contenidos en el esquema original, convierte las dependencias funcionales que contienen en su parte determinada más de un atributo, en varias dependencias funcionales con un sólo atributo, por ejemplo:

Si la siguiente dependencia funcional es proporcionada;

A,B,C ⟶ D,E,F

la interface la transforma en las siguientes dependencias;

A,B,C ⟶ D
A,B,C ⟶ E
A,B,C ⟶ F

Una vez verificada la validez de los datos iniciales, estos son cargados en la base de datos para su utilización en el sistema Normalizador.

Los datos iniciales, proporcionados por el usuario, pueden contener información redundante que debe ser eliminada, para hacer eficiente el proceso de normalización. Esta redundancia se presenta en dos formas:

- Dependencias Funcionales Redundantes
- Atributos extraños (superfluos).

Este sistema, contiene un proceso inicial que se encarga de analizar los datos de entrada para eliminar la redundancia de información. A este proceso se le ha llamado Cobertura No redundante del conjunto inicial de datos.

1.4.1 Cobertura no redundante del conjunto inicial de datos.

Este proceso tiene como objetivo transformar el conjunto inicial de datos, en otro conjunto que contenga la misma información "potencial" pero que sea mínimo. Puede darse el caso en que el conjunto inicial de datos no contenga ni dependencias funcionales redundantes ni atributos extraños, en cuyo caso no habria transformación alguna.

El conjunto de datos que se obtiene como resultado de la Cobertura No Redundante tiene las siguientes características:

1) Las partes determinadas de las dependencias funcionales consisten de un sólo atributo.

   Como se mencionó antes, este punto es tarea de la interface del sistema y genera una nueva dependencia funcional por cada atributo contenido el la parte determinada.

   Aparentemente se agregan dependencias funcionales, hecho que puede parecer una contradicción, al pensar en el objetivo de minimalidad del conjunto resultante, sin embargo, para efectos de los algoritmos de normalización, es más fácil manejar la lista de un sólo elemento y un número mayor de dependencias funcionales.

2) Ninguna parte determinante de ninguna de las dependencias funcionales contiene atributos extraños.

   Un atributo que forma parte del determinante, de alguna dependencia funcional, es extraño si no proporciona información para determinar a la parte determinada, y por tanto es necesario eliminarlo.

3) Las dependencias funcionales resultantes no son redundantes, es decir, ninguna dependencia funcional

puede ser deducida a partir de ninguna otra dependencia funcional.

Cabe mencionar que existen axiomas que permiten deducir nuevas dependencias funcionales a partir de un conjunto inicial de dependencias, estos son los Axiomas de Armstrong[A] y son:

REFLEXIVO:

Sean A y B conjuntos de atributos de una relación.

Si B es subconjunto de A, entonces, podemos obtener la dependencia funcional trivial:

$$A \longrightarrow B$$

esto es:

$$\text{Si } B \subset A \longrightarrow A \longrightarrow B$$

TRANSITIVO:

Se A, B y C conjuntos de atributos de una relación, y además sabemos que:

$$A \longrightarrow B$$
$$B \longrightarrow C$$

Entonces podemos deducir la nueva dependencia:

$$A \longrightarrow C$$

AUMENTATIVO:

Sean A, B y C conjuntos de atributos de una relación. Si sabemos que:

$$A \longrightarrow B$$

entonces

$$A, C \longrightarrow B, C$$

Con los Axiomas de Armstrong podemos encontrar TODAS las Dependencias funcionales que se deducen lógicamente del conjunto inicial, obteniendo así el CLOSURE DE UN CONJUNTO DE DEPENDENCIAS FUNCIONALES, que para este caso, no es de utilidad pues se pretende que el conjunto

de dependencias funcionales sea mínimo.

Sin embargo, con estos axiomas es posible saber que alguna dependencia funcional proviene lógicamente de otra y por lo tanto, es redundante.

Estas tres características de la Cobertura No Redundante permiten iniciar el proceso de normalización eficientemente, con un conjunto de dependencias funcionales no redundantes y sin atributos extraños.

1.4.1.1 CLOSURE de un conjunto X de atributos con respecto a un conjunto de dependencias funcionales.

Para poder explicar los algoritmos que permiten obtener la Cobertura No Redundante del conjunto inicial de datos, es esencial la utilización del concepto de CLOSURE DE UN CONJUNTO X DE ATRIBUTOS CON RESPECTO A UN CONJUNTO DE DEPENDENCIAS FUNCIONALES, esto es; Es el conjunto de todos los atributos de A de tal forma que la dependencia X ——▸ A puede ser deducida a través de los Axiomas de Armstrong. Una forma más fácil de entender este concepto es, que dado un conjunto de dependencias y un conjunto X de atributos, con el CLOSURE se obtienen TODOS los atributos que dicho conjunto X determina. Así podemos aplicar el CLOSURE a todas las partes determinantes de las dependencias funcionales y conocer todos los atributos que cada parte determinante realmente determina.

Algoritmo para la obtención del CLOSURE.

Sea X un conjunto de atributos, obtenemos el CLOSURE(X) de la siguiente manera:

Para todas las dependencias funcionales

```
Tomar una dependencia funcional (dfi)
Si la parte determinante de dfi se subconjunto de X:
  Si la parte determinada de dfi no es subconjunto de X:
    X = X U (parte determinada de dfi)
    CLOSURE(X)
  De lo contrario
De lo contrario
CLOSURE(X) = X
```

Este algoritmo recursivo, encuentra las dependencias funcionales en las que la parte determinante es subconjunto de X y la parte determinada no es subconjunto de X, es decir, que contribuye con nuevos atributos. Finalmente el resultado se obtiene en X.

1.4.1.2 Eliminación de Atributos Extraños

Como se mencionó antes, una de las características que posee la Cobertura No Redundante es la ausencia de atributos extraños.

Un atributo es extraño, si al eliminarlo de la parte determinante de alguna dependencia, ésta se conserva, por ejemplo:

sea la dependencia A, B, C ⟶ D

el atributo a será extraño si se cumplo lo siguiente:

(A, B, C) - (A) ⟶ D

quedando B, C ⟶ D

ya que A no es necesario para determinar a D.

El algoritmo para la eliminación de atributos extraños, es aplicable sólo a las dependencias funcionales que contienen más de un atributo en su parte determinante.

Algoritmo para la eliminación de atributos extraños (parte determinante).

```
Si la parte determinante de dfi tiene mas de un atributo
  toma un atributo atri
  eliminarlo temporalmente
  obtener el CLOSURE de la parte determinante reducida
  Si la parte determinada del dfi es subconjunto del
        CLOSURE de la parte determinante reducida
    Elimina el atributo atri definitivamente
    Eliminación de atributos extraños (parte determinante)
  De lo contrario
  Agregar nuevamente el atributo eliminado
De lo contrario
```

Como se puede observar se van eliminando uno a uno los atributos que conforman a la parte determinante de alguna dependencia. Obtenemos el CLOSURE de la parte determinante reducida. si la parte determinada de las dependencias en cuestión está contenida en el CLOSURE, significa que con la parte determinante reducida es suficiente para conocer la parte determinada, por tanto la dependencia no requiere del atributo eliminado, confirmando que es extraño y eliminándolo definitivamente.

1.4.1.3 Eliminación de Dependencias Funcionales Redundantes

Similar al algoritmo de Eliminación de Atributos Extraños el algoritmo de Eliminación de Dependencias Funcionales Redundantes, elimina temporalmente alguna dependencia funcional.

Se obtiene entonces el CLOSURE de la parte determinante de dicha dependencia, pero con respecto al conjunto de dependencias reducido. Si la parte determinada de la dependencia delimitada es subconjunto del CLOSURE, significa que la dependencia eliminada es redundante ya que sin ella fue posible deducir la parte determinada y ésta será eliminada definitivamente.

Algoritmo para la eliminación de dependencias funcionales redundantes.

Elimina temporalmente una dependencia funcional de todo el conjunto

Obtener el CLOSURE de la parte determinante de la dependencia eliminada, con respecto al conjunto de dependencias reducido.

Si la parte determinada de la dependencia eliminada es subconjunto del CLOSURE

Elimina la dependencia definitivamente

De lo contrario

Agrega nuevamente la dependencia funcional eliminada.

Ya con los algoritmos para la eliminación de atributos extraños y dependencias funcionales redundantes, podemos obtener el algoritmo para la Cobertura No Redundante del conjunto inicial de datos.
Para todas las dependencias:

Eliminación de atributos extraños (parte determinante)

Para todas las dependencias:

elimina dependencias funcionales.

1.4.2 Descomposición de la Relación en Tercera Forma Normal

El algoritmo utilizado para descomponer una relación en 3a forma normal, es el presentado por Phillip A. Bernstein[8]. Según S. Ceri y G. Gottlob[6]. El mejor algoritmo de normalización del que se tiene conocimiento es el de Bernstein, ya que éste demuestra que además de obtener relaciones en 3a forma normal, el número de relaciones resultante es el menor posible.

La mayoría de los algoritmos de normalización, construyen relaciones en 3a forma normal directamente sin descomposiciones intermedias en 2a forma normal. Esto se debe a que dadas las características del algoritmo de normalización, se tiene toda la información necesaria para descomponer directamente en 3a forma normal. Si se realiza una descomposición intermedia en 2a forma normal, se tendría que aplicar el algoritmo de descomposición en 3a forma normal a cada una de las nuevas relaciones generadas por la descomposición en 2a forma normal, lo cual no es eficiente.

El algoritmo de Bernstein consiste de los siguientes cinco pasos:

Algoritmo para la descomposición de una relación en tercera forma normal.

PASO 1. Encontrar la cobertura no redundante (H) del conjunto inicial de dependencias funcionales.

PASO 2. División del conjunto de dependencias H en grupos de Hi de tal forma que todas las dependencias en cada grupo tengan partes determinantes iguales.

PASO 3. Mezclar grupos Hi y Hj con partes determinantes X y Y respectivamente, cuando las dependencias X⟶Y y Y⟶X sean válidas. X y Y son llaves equivalentes.

PASO 4. Construir una Cobertura particular de H que incluya todas las dependencias X⟶Y, donde Y y X son llaves equivalentes y todas las demás dependencias no sean redundantes.

PASO 5. Construir relaciones.

El paso 1 se obtiene con el proceso de Cobertura No Redundante expuesto anteriormente. Al observar que nuestras dependencias funcionales contienen sólo un atributo como parte determinada, la eliminación de atributos extraños es comparable a la eliminación de dependencias funcionales parciales que se maneja generalmente en el contexto de normalización, por ejemplo:

Sea la eliminación del atributo extraño:

$$A, B \longrightarrow D$$

y

$$(A, B) - (A) \longrightarrow D$$

esto es

$$B \longrightarrow D$$

significando que D depende parcialmente de A y B.

El paso 2 realiza la agrupación de dependencias funcionales con partes determinantes iguales, por ejemplo:

Sean
$$\begin{aligned} A, B &\longrightarrow C \\ A, B &\longrightarrow D \\ E, F &\longrightarrow G \\ E, F &\longrightarrow H \\ I &\longrightarrow J \end{aligned}$$

obteniendo tres grupos representativos

grupo 1 : (A, B)
grupo 2 : (E, F)
grupo 3 : (I)

El paso 3 encuentra llaves equivalentes y forma un sólo grupo de ambas llaves, por ejemplo:

Sean A,B ⟶ C, D, E, F
C,D ⟶ A, B, H, I

entonces

A, B ⟷ C, D son llaves equivalentes

y de los grupos existentes se forma uno solo grupo (A, B)

grupo ((A, B), (C, D))
grupo (C, D)

El paso 4 elimina las dependencias transitivas y agrega las dependencias resultantes a la obtención de las llaves equivalentes, por ejemplo. (A, B) ⟷ C, D.

Para eliminar las dependencias transitivas se utiliza un proceso similar al de Eliminación de dependencias redundantes, por ejemplo:

Si tenemos A, B ⟶ C, D, E

y D ⟶ E

eliminamos entonces la dependencia funcional temporalmente

A, B ⟶ E

Sin embargo observamos que E se puede conocer a partir de:

D ⟶ E

con lo que eliminamos definitivamente la dependencia transitiva:

A, B ⟶ E

quedando como resultado:

A, B ⟶ C, D

y D ⟶ E

El paso 5 construye nuevas relaciones a partir de cada grupo. Este paso tiene que ver con la creación de nuevos nombres para las nuevas relaciones, crea los esquemas de las nuevas relaciones y obtiene algunas llaves de cada creación.

Las llaves están dadas por la parte determinante que caracteriza a cada grupo, y a través de las dependencias funcionales de la base de datos; se colectan todos los atributos que pertenecen a cada nuevo esquema.

Una vez generadas las nuevas relaciones en tercera forma normal, estas son cargadas a la base de datos del sistema, junto con las llaves y la información acerca de la descomposición.

1.5 CODIFICACION

```
/* tesis                                          */
/* programa que normaliza relaciones              */
/* en base a la 1a, 2a y 3a formas normales       */
/* escrito en Turbo PROLOG                        */

/* directivas del compilador */
nowarnings
code = 3072

/* codigo del programa */
domains
          listas = lista*
          lista = elem*
          depfun = depfun(nombre_relacion,lista,lista)
          elem, nombre_relacion  = symbol


database
          esquema(nombre_relacion,lista)
          depfun(nombre_relacion,lista,lista)
          depfunj(nombre_relacion,lista,lista)
          grupo(nombre_relacion,listas)
          llave(nombre_relacion,lista)
          decomp(nombre_relacion,nombre_relacion)
          en3afn(nombre_relacion)
          tnfdecomp(nombre_relacion,nombre_relacion)
          remember(depfun)
          clo(nombre_relacion,lista,lista)


predicates

          elemento(elem,lista)
          elemento(lista,listas)
          subconjunto(lista,lista)
          atributo(lista,nombre_relacion)
          unions(lista,lista,lista,lista)
          union(lista,lista,lista,nombre_relacion).
          menos1(lista,lista,elem)
          menos1(listas,listas,lista)
          menos(lista,lista,lista)
          concatena(lista,lista,lista)
          concatena(listas,listas,listas)
          closure(nombre_relacion,lista,lista)
          elim_atrib(nombre_relacion)
          reducelhs(nombre_relacion,lista,lista,lista)
```

```
        elim_dep_fun_red(nombre_relacion)
        encuentra_cov_min(nombre_relacion)
        thirdnf(nombre_relacion)
        paso1(nombre_relacion)
        paso2(nombre_relacion)
        paso3(nombre_relacion)
        paso4(nombre_relacion)
        paso5_a(nombre_relacion)
        paso5(nombre_relacion,integer)
        remembercovering(nombre_relacion)
        mezcla(nombre_relacion,lista,lista)
        yaexistegrupo(nombre_relacion,lista,lista)
        elimin(nombre_relacion,lista)
        encuentranombre(nombre_relacion,integer,
                        nombre_relacion)
        guardaalgunallave(nombre_relacion,listas)
        hazesquema(nombre_relacion,nombre_relacion,listas)
        collect1(nombre_relacion,listas,lista)
        collect(nombre_relacion,listas,lista,lista,lista)
        quitamodfds(nombre_relacion)
        reassertrememberedfds.
        esatribvalid(nombre_relacion,listas,elem)

        aux1(nombre_relacion,lista,lista,lista)
        aux2(nombre_relacion,lista,lista,lista)
        borra(nombre_relacion,lista,lista)
        aux3(nombre_relacion,listas,elem,lista,lista)
        aux4(lista,lista,elem)
        inicia
        recorre(integer)
        continua
        imprime


goal

    consult("a:\datos.dat"),esquema(REL,_),inicia,
    paso1(REL),
    thirdnf(REL),
    save("a:\resulta.pro"),
    exit.

clauses
      inicia:- makewindow
             (1,78,10,"N O R M A L I Z A C I O N",0,0,25,80).

      recorre(VN):-shiftwindow(VN),clearwindow.
```

```
/* hechos iniciales: esquema y dependencias funcionales*/


/* utileria */

/* elemento(E,L) = E es elemento de L */

elemento(E,[E|T]).
elemento(E,[_|T]):- elemento(E,T).

/* subconjunto(A,B) = A esta contenido en B */

subconjunto([],X):-!.
subconjunto([H|TA],[H|TB]):-!,subconjunto(TA,TB).
subconjunto(A,[_|TB]):-subconjunto(A,TB).

/* atributo(E,R) = E es una lista de atributos de R */

atributo(E,R):- esquema(R,S), subconjunto(E,S).

/* Unión de dos subconjuntos A y B que pertenecen a L
   en U */

unions(A,A,[],L):-!.
unions(B,[],B,L):-!.
unions([HL|TU],[HL|TA],[HL|TB],[HL|TL]):-!,
                          unions(TU,TA,TB,TL).
unions([HL|TU],[HL|TA],B,[HL|TL]):-!,
                          unions(TU,TA,B,TL).
unions([HL|TU],A,[HL|TB],[HL|TL]):-!,
                          unions(TU,A,TB,TL).
unions(U,A,B,[HL|TL]):-unions(U,A,B,TL).

/* Unión de conjuntos de atributos ordenados de acuerdo
   a la relación */

union(U,A,B,REL):-esquema(REL,L), subconjunto(A,L),
                          subconjunto(B,L), unions(U,A,B,L).

/* Menos1(R,A,B) => substrae el elemento B del conjunto A
   en R */

menos1([],[],B):-!.
menos1(TA,[B|TA],B):-!.
menos1([HA|TX],[HA|TA],B):-menos1(TX,TA,B).

/* Menos(R,A,B) => R = A - B; A y B son conjuntos
   ordenados */
```

```
menos(R,R,[]):-!.
menos(Z,A,[HB|TB]):-menos1(R,A,HB),menos(Z,R,TB).

/* Concatenación de X y Y en Z */

concatena([],L,L).
concatena([X|L1],L2,[X|L3]):-concatena(L1,L2,L3).

/* Calculo del Closure de un conjunto de atributos */

closure(REL,X,RESULTADO):-depfun(REL,LHS,RHS),
                          subconjunto(LHS,X),
                          not (subconjunto(RHS,X)),
                          union(W,X,RHS,REL),!,
                          closure(REL,W,RESULTADO).

closure(REL,X,RESULTADO):-RESULTADO = X.

/* Cobertura no redundante */

/* a) Eliminación de atributos extraños */

elim_atrib(REL):-recorre(1),cursor(5,5),
                 write
                 (" Eliminación de Atributos Extraños"),
                 nl,cursor(7,5),
                 write
                 ("para la cobertura de la relación "
                   ,REL),
                 depfun(REL,LHS,RHS),
                 reducelhs(REL,LHS,RHS,NUEVALHS),
                 not (LHS = NUEVALHS),
                 retract(depfun(REL,LHS,RHS)),
                 asserta(depfun(REL,NUEVALHS,RHS)),
                 cursor(10,5),
                 write
                 ("dependencia : ",LHS," --> ",RHS),
                 nl,cursor(13,5),
                 write
                 ("nueva parte izquierda = ",NUEVALHS),
                 nl,fail.

elim_atrib(REL):-cursor(15,5),
                 write
                 ("todos los atributos extraños
                   eliminados"),
                 nl,continua.
```

```
continua:-cursor(22,5),write
                        ("para continuar digite c "),
                        readchar(_).

reducelhs(REL,LHS,RHS,NUEVALHS):- elemento(A,LHS),
                                  menos1(Z,LHS,A),
                                  not (Z = []),
                                  closure(REL,Z,ZCLO),
                                  subconjunto(RHS,
                                              ZCLO),!,
                                  reducelhs(REL,Z,RHS,
                                            NUEVALHS).

reducelhs(REL,LHS,RHS,NUEVALHS):-NUEVALHS=LHS.

/* b) Eliminación de dependencias funcionales
   redundantes */

elim_dep_fun_red(REL):-recorre(1),cursor(5,5),
                       write
                       ("Eliminación de Dep. Fun.
                         redundantes"),
                       cursor(7,5),nl,
                       write
                       ("dependencias de la cobertura de"
                         ,REL),
                       depfun(REL,LHS,RHS),
                       borra(REL,LHS,RHS),
                       closure(REL,LHS,Z),
                       aux1(REL,LHS,RHS,Z),
                       fail.

elim_dep_fun_red(REL):-cursor(15,5),
                       write
                       ("todas las depend. funcionales
                         redundantes eliminadas"),
                       continua,nl.

aux1(REL,LHS,RHS,Z):-not(subconjunto(RHS,Z)),!,
                     asserta(depfun(REL,LHS,RHS)).

aux1(REL,LHS,RHS,Z):-!,subconjunto(RHS,Z),cursor(10,5),
                       write
                       ("dependencia funcional
                         redundante"),
                       nl,
                       write
```

```
                        (LHS," --> ",RHS," eliminada."),
                        continua,nl.

aux1(REL,LHS,RHS,Z).

/* Combinando a) y b) para encontrar la cobertura minima
    */

encuentra_cov_min(REL):-recorre(1),cursor(5,5),
                        write
                        ("Cobertura minima para la
                          relación ",REL),
                        continua,nl,
                        elim_atrib(REL),
                        elim_dep_fun_red(REL).


/* algoritmo de Berenstein para descomponer una relación
   en 3a forma normal */

thirdnf(REL):- remembercovering(REL),paso2(REL),
               paso3(REL),
               paso4(REL),
               paso5_a(REL).

/* paso 1 : encontrar una cobertura no redundante */

paso1(REL):-recorre(1),cursor(5,5),
            write
            ("Descomposición de ",REL," en 3a forma
              normal: paso 1"),
            continua,nl,encuentra_cov_min(REL),
            remembercovering(REL).


remembercovering(REL):-depfun(REL,LHS,RHS),
                       assertz(remember
                               (depfun(REL,LHS,RHS))),
                       fail.

remembercovering(REL).

/* paso 2 : división de la cobertura en grupos con
   determinantes iguales */

paso2(REL):-recorre(1),
            cursor(5,5),
```

```
            write("Descomposición de ",REL,
                  " en 3a forma normal: paso 2"),
            nl,
            depfun(REL,LHS,_),not (grupo(REL,[LHS])),
            asserta(grupo(REL,[LHS])),
            write("grupo basado en Parte Determ.: ",
                  LHS),nl,
closure(REL,LHS,CLO):-
            asserta(clo(REL,LHS,CLO)),
            fail.

paso2(REL):-nl,write("terminada la división en grupos"),
            continua,nl,nl.

/* paso 3 : mezcla de grupos con llaves equivalentes;
   depfunj son las dependencias de paso del conjunto J
   de [6] */

paso3(REL):-recorre(1),
            cursor(5,5),write("Descomposición de ",REL,
            " en 3a forma normal: paso 3"),nl,
            clo(REL,LHS1,LHS1CLO),
            clo(REL,LHS2,LHS2CLO),
            not (LHS1 = LHS2),subconjunto(LHS2,LHS1CLO),
            subconjunto(LHS1,LHS2CLO),
            not (yaexistegrupo(REL,LHS1,LHS2)),
                cursor(7,5),
            write
            ("llaves equivalentes descubiertas: ",
              LHS1,"<-->",LHS2),
            nl,mezcla(REL,LHS1,LHS2),
            asserta(depfunj(REL,LHS1,LHS2)),
            asserta(depfunj(REL,LHS2,LHS1)),
            fail.

paso3(REL):-clo(REL,LHS,LHSCLO),
            retract(clo(REL,LHS,LHSCLO)),
            fail.

paso3(REL):-depfunj(REL,LHS,RHS),depfun(REL,LHS,A),
            subconjunto(A,RHS),
            retract(depfun(REL,LHS,A)),
            fail.

paso3(REL):-nl,write("todas las llaves equivalentes
               descubiertas"),
            continua,nl,nl.
```

```
mezcla(REL,LHS1,LHS2):- grupo(REL,G1),elemento(LHS1,G1),
                        grupo(REL,G2),elemento(LHS2,G2),
                        retract(grupo(REL,G1)),
                        retract(grupo(REL,G2)),
                        concatena(G1,G2,NVOGPO),
                        asserta(grupo(REL,NVOGPO)).

yaexistegrupo(REL,LHS1,LHS2):- grupo(REL,GRUPO),
                               elemento(LHS1,GRUPO),
                               elemento(LHS2,GRUPO).

/* paso 4 : eliminación de dependencias transitivas */

paso4(REL):-recorre(1),cursor(5,5),
            write("Descomposición de ",REL,
            "en 3a forma normal : paso 4"),nl,
            depfunj(REL,LHS,RHS),
            assertz(depfun(REL,LHS,RHS)),
            fail.

            /*assertz((depfun(REL,LHS,RHS):-
              depfunj(REL,LHS,RHS))),
              fail.*/

/* prueba de redundancia para las dependencias que no
   estan en J */

paso4(REL):-depfun(REL,LHS,RHS),
            not(depfunj(REL,LHS,RHS)),
            borra(REL,LHS,RHS),
            closure(REL,LHS,Z),
            aux2(REL,LHS,RHS,Z),
            fail.

paso4(REL):-depfunj(REL,LHS,RHS),
            retract(depfun(REL,LHS,RHS)),
            fail.


paso4(REL):-cursor(7,5),
            write("dependencias transitivas eliminadas"),
            nl,write("Grupos Definitivos : "),nl,
            grupo(REL,G),write("grupo : ",G),nl,
            fail.

paso4(REL):-continua,nl.

            borra(REL,LHS,RHS):-retract(depfun(REL,
```

```
                                        LHS,RHS)),!.

aux2(REL,LHS,RHS,Z):-not(subconjunto(RHS,Z)),!,
                     asserta(depfun(REL,LHS,RHS)).

aux2(REL,LHS,RHS,Z):-!,subconjunto(RHS,Z),
                     elimin(REL,LHS),
                     nl,
                     write(LHS," --> ",RHS,"
                           eliminada."),
                     nl.

aux2(REL,LHS,RHS,Z).

/* eliminación de un elemento LHS de un grupo G */

elimin(REL,LHS):-not(depfun(REL,LHS,_)),grupo(REL,G),
                 elemento(LHS,G),retract(grupo(REL,G)),
                 menos1(Z,G,LHS),not(Z = []),
                 asserta(grupo(REL,Z)),!.
elimin(REL,LHS).

/* paso 5 : transformación de cada grupo en una relación
   en 3a forma normal */

paso5_a(REL):-recorre(1),cursor(5,5),
              write("Descomposición de ", REL,
                    "en 3a forma normal: paso 5 "),nl,nl,
              paso5(REL,0),imprime,continua.

paso5(REL,NR):-grupo(REL,G),NVONR = NR + 1,
               encuentranombre(REL,NVONR,NVAREL),
               hazesquema(REL,NVAREL,G),
               guardaalgunallave(NVAREL,G),
               assertz(decomp(REL,NVAREL)),
               assertz(en3afn(NVAREL)),
               assertz(tnfdecomp(REL,NVAREL)),
               retract(grupo(REL,G)),!,
               paso5(REL,NVONR).

paso5(REL,NR):-quitamodfds(REL),reassertrememberedfds.

encuentranombre(REL,NR,NVAREL):-
               concat(REL,"_",PASO),
               SUFIJO=NR+96,
               char_int(CHAR,SUFIJO),
               str_char(SCHAR,CHAR),
               concat(PASO,SCHAR,NOMBRE),
```

```
                NVAREL=NOMBRE.

guardaalgunallave(NVAREL,G):-elemento(K,G),
                             assertz(llave(NVAREL,K)),
                             fail.

guardaalgunallave(NVAREL,G).

hazesquema(REL,NVAREL,G):- collect1(REL,G,NVOESQUEMA),
                           assertz(esquema(NVAREL,
                                   NVOESQUEMA)).

/* colectando en RESULTADO el esquema de la relación
   sintetizada asociada al grupo G. Un atributo A
   pertenece al esquema si y solo si pertenese al
   determinante, o determinado de alguna dependencia
   funcional cullo determinante esta en G */

collect1(REL,G,RESULTADO):-esquema(REL,TOTEST),
                           collect(REL,G,TOTEST,[],
                                   RESULTADO).

collect(REL,G,TOTEST,ACCEPT,RES):-elemento(A,TOTEST),
                                  menos1(NEWTOTEST,
                                         TOTEST,A),
                                  aux3(REL,G,A,ACCEPT,
                                       NEWACCEPT),
                                  collect(REL,G,
                                     NEWTOTEST,NEWACCEPT,
                                     RES).

collect(REL,G,TOTEST,ACCEPT,RES):-RES=ACCEPT.

esatribvalid(REL,G,A):-!,elemento(LHS,G),
                       depfun(REL,LHS,RHS),
                       aux4(LHS,RHS,A).


aux3(REL,G,A,ACCEPT,NEWACCEPT):-esatribvalid(REL,G,A),!,
                                union(NEWACCEPT,
                                ACCEPT,[A],REL).

aux3(REL,G,A,ACCEPT,NEWACCEPT):-!,NEWACCEPT = ACCEPT.

aux4(LHS,RHS,A):-elemento(A,LHS),!.

aux4(LHS,RHS,A):-!,elemento(A,RHS).
```

```
quitamodfds(REL):-depfun(REL,LHS,RHS),
                  retract(depfun(REL,LHS,RHS)),
                  fail.

quitamodfds(REL):-depfunj(REL,LHS,RHS),
                  retract(depfunj(REL,LHS,RHS)),
                  fail.

quitamodfds(REL).

reassertrememberedfds:-remember(depfun(REL,LHS,RHS)),
                       assertz(depfun(REL,LHS,RHS)),
                       retract(remember(
                             depfun(REL,LHS,RHS))),
                       fail.

reassertrememberedfds.

imprime:-esquema(NOMBRE,ATRIBS),nl,
         write
         ("Relación : ",NOMBRE,"\natributos : ",ATRIBS),
           fail.

imprime.

/* fin del programa normalizador */
```

1.6 PRUEBAS

Como elemento práctico del presente trabajo, se presenta el proceso completo de la ejecución del normalizador considerando el siguiente ejemplo:

La relación ejemplo está formada por los siguientes atributos:

Nombre: conciertos

Atributos:

- Instrumento
- Obra
- Intérprete
- Sala
- Estilo
- Autor

Esta relación contiene información de información existente en determinada Asociación de Música referente a conciertos, tipos de instrumentos, intérprete, sala, estilo, obra y autor, considerando las siguientes condiciones de dependencia:

a) Intérprete depende del instrumento y la obra. Esto es, para hacer referencia a un intérprete específico se necesita saber qué instrumento domina y qué obra puede o tiene asignada para interpretar.

b) El instrumento depende del intérprete que lo toque. Esto es, para saber que instrumento se domina dentro de la Asociación es necesario saber el nombre del intérprete que lo puede dominar.

c) El autor depende del estilo y la sala. Esto limita el alcance de la difusión del autor, ya que el autor está limitado a ser interpretado en aquellas salas que estén consideradas de algún estilo definido.

d) La sala depende del estilo. Las salas tocan únicamente cierto estilo. Además el estilo a su vez depende de sala pues determinado estilo es interpretado solo en salas selectas.

e) El autor depende de la sala, pues se tiene clasificada la información de los autores por el estilo en que componen.

Ahora , la forma en que se le debe representa la información

anterior de tal manera que el normalizador pueda hacer uso de ella para su proceso de normalización es la siguiente:

```
esquema("conciertos",["instrumento","obra","interprete",
        "sala","estilo","autor"])
depfun("conciertos",["instrumento","obra"],["interprete"])
depfun("conciertos",["interprete"],["instrumento"])
depfun("conciertos",["sala","estilo"],["autor"])
depfun("conciertos",["estilo"],["sala"])
depfun("conciertos",["estilo"],["autor"])
depfun("conciertos",["sala"],["estilo"])
```

Cabe aclarar que la relación inicial se le debe de especificar un nombre, el cuál en este ejemplo es "conciertos".

Se presenta a continuación la información completa de la "corrida" del normalizador conteniendo la descripción de cada acción que se realiza y los datos que se requieren como entrada, además de resultados parciales:

```
Descomposición de conciertos en 3a forma normal: paso 1

para continuar digite c


Cobertura No Redundante para la relación : conciertos

para continuar digite c


Eliminación de Atributos Extraños

para la cobertura de la relación : conciertos

dependencia : {"sala","estilo"}-->["autor"] modificada

nueva parte izquierda = ["estilo"]

todos los atributos extraños eliminados

para continuar digite c
```

```
Eliminación de Dep. Fun. redundantes
dependencias de la cobertura de conciertos
dependencia funcional redundante
["estilo"] --> ["autor"] eliminada.
para continuar digite c

todas las depend. funcionales redundantes eliminadas
para continuar digite c

Descomposición de conciertos en 3a forma normal: paso 2
grupo basado en Parte Determ.: ["sala"]
grupo basado en Parte Determ.: ["estilo"]
grupo basado en Parte Determ.: ["interprete"]
grupo basado en Parte Determ.: ["instrumento","obra"]
terminada la división en grupos
para continuar digite c

Descomposición de conciertos en 3a forma normal: paso 3
llaves equivalentes descubiertas:["estilo"]<-->["sala"]
todas las llaves equivalentes descubiertas
para continuar digite c

Descomposición de conciertos en 3a forma normal: paso 4
dependencias transitivas eliminadas
Grupos Definitivos :
grupo : [["estilo"],["sala"]]
```

```
grupo : [["instrumento","obra"]]

grupo : [["interprete"]]

para continuar digite c


Descomposición de conciertos en 3a forma normal: paso 5

Relación : conciertos
atributos : ["instrumento","obra","interprete",
             "sala","estilo","autor"]

Relación : conciertos_a
atributos : ["estilo","autor"]
Llave : ["estilo"]
Llave : ["sala"]

Relación : conciertos_b
atributos : ["instrumento","obra","interprete"]
Llave : ["instrumento","obra"]

Relación : conciertos_c
atributos : ["instrumento","interprete"]
Llave : ["interprete"]

para continuar digite c
```

En este punto contamos con la información y el apoyo suficiente para poder presentar la nueva estructura de nuestra relación ejemplo "conciertos" y normalizada en tercera forma normal evitando, como ya sabemos, las anomalías en la información.

A continuación se presenta la nueva estructura de nuestra Base de de Datos, en forma de esquema, que fue el resultado de la corrida anterior y que respeta totalmente las consideraciones semánticas de entrada:

```
esquema("conciertos",["instrumento","obra","interprete","
         sala","estilo","autor"])
esquema("conciertos_a",["estilo","autor"])
esquema("conciertos_b",["instrumento","obra",
        "interprete"])
esquema("conciertos_c",["instrumento","interprete"])
depfun("conciertos",["sala"],["estilo"])
```

```
depfun("conciertos",["estilo"],["autor"])
depfun("conciertos",["estilo"],["sala"])
depfun("conciertos",["interprete"],["instrumento"])
depfun("conciertos",["instrumento","obra"],
       ["interprete"])
llave("conciertos_a",["estilo"])
llave("conciertos_a",["sala"])
llave("conciertos_b",["instrumento","obra"])
llave("conciertos_c",["interprete"])
decomp("conciertos","conciertos_a")
decomp("conciertos","conciertos_b")
decomp("conciertos","conciertos_c")
en3afn("conciertos_a")
en3afn("conciertos_b")
en3afn("conciertos_c")
tnfdecomp("conciertos","conciertos_a")
tnfdecomp("conciertos","conciertos_b")
tnfdecomp("conciertos","conciertos_c")
```

Podemos concluir que se generaron 3 relaciones que tienen las siguientes características:

- Relación # 1: conciertos_a:

  atributos: estilo
  autor

  llave : estilo o sala (son llaves equivalentes o candidatas)

- Relación # 2: conciertos_b:

  atributos: instrumento
  obra
  intérprete

  llave : instrumento y obra

- Relación # 3: concierto_c:

  atributos: instrumento
  intérprete

  llave : intérprete

CONCLUSIONES

A lo largo del presente trabajo encontramos que:

Primera: Existen grandes deficiencias en cuanto al proceso de diseño de sistemas de cómputo.

Segunda: Cada vez más, se reconoce a la información como un recurso importante para la toma de decisiones.

Tercera: El proceso de diseño es empírico, no existe una metodología que garantice un buen diseño; por el contrario, observamos divergencias entre los diferentes autores, donde los únicos puntos en común son la intuición y la iterativídad de dicho proceso.

Cuarta: El proceso de normalización utiliza conceptos abstractos que dificultan su comprensión.

Quinta: La normalización depende en gran medida de la manera en que el diseñador interpreta los datos y sus relaciones.

Sexta: El gran impacto teórico que ha tenido la Teoría de Normalización se debe a que ésta introdujo por primera vez conceptos matemáticos en lo que se refiere al manejo de datos.

Séptima: Resulta difícil llevar a la práctica los conceptos contenidos en la Teoría de Normalización debido a su nivel de abstracción.

Octava: Existe confusión en cuanto a los alcances de la Inteligencia Artificial.

Novena: PROLOG es un lenguaje que facilita la programación de procesos altamente recursivos.

En cuanto al paquete normalizador encontramos que:

Décima: Permite utilizar la Teoría de Normalización sin conocerla a fondo.

Décimo Primera: Minimiza las relaciones generadas ya que se identifican atributos irrelevantes.

Décimo Segunda: Garantiza que se lleven a cabo todos los pasos requeridos en el proceso de normalización evitando errores.

Décimo Tercera: Contribuye a esclarecer las relaciones entre

datos.

Adicionalmente encontramos que:

Décimo Cuarta: La profundidad misma de cada tema impide una cobertura total en un solo trabajo; esto nos llevó a reestructurar el planteamiento original que pretendía cubrir tópicos fuera de contexto, lo que sugirió extraer los temas necesarios para abarcar solamente los conceptos fundamentales en torno al paquete normalizador.

Décimo Quinta: La rápida evolución de los sistemas de información nos permite reconocer la importancia que estos tienen hoy en día puesto que al obtener información veraz y oportuna, se tienen los elementos para una mejor toma de decisiones en beneficio de las organizaciones.

Décimo Sexta: No es fácil mecanizar el pensamiento humano pero si es posible mecanizar el proceso de normalización.

Décimo Séptima: Al utilizar el paquete normalizador confirmamos la importancia de la semántica de los datos. Aún entendiendo el significado de los mismos, nos enfrentamos a resultados inesperados. Descubrimos que en ocasiones se eliminaban dependencias aparentemente redundantes debido a una identificación incorrecta de las dependencias funcionales.

Décimo Octava: Las limitaciones más importantes para la mecanización del proceso de diseño de sistemas de información en general, es la falta del conocimiento para poder implantar herramientas que analicen semánticamente los datos.

Décimo Novena: La Inteligencia Artificial ha hecho aportaciones importantes. Como ejemplo de ello tenemos la programación orientada a objetos, que abrió el espectro de la programación tradicional al obtener un enfoque diferente que pretende acercarse al razonamiento humano.

Vigésima: Observamos que la teoría siempre va un paso adelante de la tecnología; por esto, en un futuro no muy lejano, se tendrán sistemas que utilicen el conocimiento, razonamiento y procesamiento simbólico conjunta y eficientemente para modelar el pensamiento humano.

Vigésimo Primera: Finalmente, creemos que hemos desarrollado una herramienta práctica basada en la teoría formal de normalización de relaciones, que permite mostrar el lado útil de la abstracción teórica.

CITAS

A Armstrong W.W. DEPENDENCY STRUCTURES OF DATABASE RELATIONS. IN INFORMATION PROCESSING. North-Holland, Amsterdam, (1974), pp. 580-583.

B Bernstein P.A.SYNTHESIZING THIRD-NORMAL-FORM RELATION FROM FUNCTIONAL DEPENDENCIES. ACM Trans. Database Syst. 1, 1, (Dic. 1976). pp. 277-298.

C Blanning R.W. ISSUES IN THE DESIGN OF RELATIONAL MODEL MANAGEMENT SYSTEMS. In Proceedings of the National Computer Conference. AFIPS Press, (Jun. 1983). pp. 395-401.

D Borland International Inc. TURBO PROLOG USER MANUAL, 2th ed.

E Bouzeghoub M., Gardain G. and Metais E. DATABASE DESIGN TOOLS: AN EXPERT SYSTEM APROACH. In Proceedings of the 11th VLDB (Stockholm, Sweden, Agu. 21 1985). pp. 82-95

F Cárdenas Alfonso F., SISTEMAS DE ADMINISTRACION DE BANCO DE DATOS. Ed. Limusa, 1982.

G Ceri S. and Gottlob G. NORMALIZATION OF RELATIONS AND PROLOG. Communications of the ACM. 29, 6, (Nov. 1986). pp. 524-544.

H Clark K.L., Mc Cabe F.G., MICRO-PROLOG: PROGRAMMING IN LOGIC. Prentice-Hall International Series, 1984.

I Cood E.F. FURTHER NORMALIZATION OF THE DATABASE RELATIONAL MODEL. In Data Base Systems Courant Computer Science Symposia. Vol. 6 Ed. Prentice_hall (1972).

J Codd E.F. A RELATIONAL MODEL OF DATA FOR LARGE SHARED DATA BANKS. Communications of the ACM 13, 6, (Jun. 1970). pp. 377-387.

K Cullinet Software Inc., IDMS ARCHITECT USER MANUAL.

L Cullinet Software Inc., DATABASE DESIGN IN DEFINITION GUIDE.

M Date C. J., AN INTRODUCTION TO DATABASE SYSTEMS. Vol 2, Addison Wesly Publishing Co.

N Date C. J., AN INTRODUCTION TO DATABASE SYSTEMS. 2a. Ed., Addison Wesly Publishing Co., 1977.

O Fagin R., FUNCTIONAL DEPENDENCIES IN A RELATIONAL DATABASE AND PROPOSITIONAL LOGIC, IBM Res. Dev. 21, 6, noviembre 1977.

P Fairley, Richard, INGENIERIA DE SOFTWARE, McGraw-Hill de México, 1987.

Q Floyd L. Rutch y Phillipe G. Simbarlow, PSICOLOGIA Y VIDA, Ed. Trillas, México, 1979.

R James O. Wittaker, PSICOLOGIA, Ed. Interamericana, 1971.

S Krajewski R., DATABASE TYPES, Byte (Oct. 1984), pp. 137-142.

T Kroenke David M., DATABASE PROCESSING FUNDAMENTALS DESIGN IMPLEMENTATION, Science Research Associates Inc.,Estados Unidos de América, 1983.

U Pressman Roger S., SOFTWARE ENGEENERING: A PRACTITIONERS APROACH, McGraw-Hill International Book Co., 1982.

V Jiménez Alejandro. POR QUE PROLOG. Revista 010, Vol 7, # 9, Mayo 1985.

W Rich Elaine, ARTIFICIAL INTELLIGENCE, McGraw-Hill, 1983.

X Samuel A.L., SOME STUDIES IN MACHINE LEARNING USING THE GAME OF CHECKERS, McGraw-Hill, 1963.

Y Wiederhold Gio, DISEÑO DE BASES DE DATOS, McGraw-Hill, 1985.

Z Yourton Edward, Constantin Larry, STRUCTURED DESIGN, Prentice-Hall, 1979.

BIBLIOGRAFÍA

Aho A.V., Beeri C. and Ullman J.D. THE THEORY OF JOINS IN RELATIONAL DATABASES. ACM Trans. Database Sys. 4, 3. September, 1979, pp. 297-314.

Armstrong W.W. DEPENDENCY STRUCTURES OF DATABASE RELATIONS IN INFORMATION PROCESSING. North-Holland, Amsterdam, 1974, pp. 580-583.

Beeri C. and Bernstein P.A. COMPUTATIONAL PROBLEMS RELATED TO THE DESIGN OF NORMAL FORM RELATIONAL SCHEMATA. ACM Trans. Database Syst. 4, 1. March, 1979. pp. 30-59

Beeri C. ON THE MEMBERSHIP PROBLEM FOR FUNCTIONAL AND MULTIVALUED DEPENDENCIES IN RELATIONAL DATABASES. ACM Trans. Database Syst. 5, 3. September, 1980. pp. 241-259.

Beeri C.,Bernstein,P.A. and Goodman N.A SOPHISTICATE'S INTRODUCTION TO DATABASE NORMALIZATION THEORY. In Proceedings of the 4th International Conference On Very Large Data Bases West Berlin. 1978. pp. 113-124.

Bernstein P.A.SYNTHESIZING THIRD-NORMAL-FORM RELATION FROM FUNCTIONAL DEPENDENCIES. ACM Trans. Database Syst. 1, 1. December, 1976. pp. 277-298.

Blanning R.W. ISSUES IN THE DESIGN OF RELATIONAL MODEL MANAGEMENT SYSTEMS. In Proceedings of the National Computer Conference. AFIPS Press. June, 1983. pp. 395-401.

Bobrow Daniel G., Mittal Sanjoy y Stefik Mark J. EXPERT SYSTEMS: PERILS AND PROMISE, Communications of the ACM, Vol 29 No 9. September, 1986.

Borland International Inc. TURBO PROLOG USER MANUAL, 2th ed.

Bouzeghoub M., Gardain G. and Metais E. DATABASE DESIGN TOOLS: AN EXPERT SYSTEM APROACH. In Proceedings of the 11th VLDB. Stockholm, Sweden. August, 1985). pp. 82-95

Cárdenas Alfonso F., SISTEMAS DE ADMINISTRACION DE BANCO DE DATOS. Ed. Limusa, 1982.

Ceri S. and Gottlob G. NORMALIZATION OF RELATIONS AND PROLOG. Communications of the ACM. 29, 6. Nov. 1986. pp. 524-544.

Clark K.L., Mc Cabe F.G., MICRO-PROLOG: PROGRAMMING IN LOGIC.

Prentice-Hall International Series, 1984.

Cood E.F. FURTHER NORMALIZATION OF THE DATABASE RELATIONAL MODEL. In Data Base Systems Courant Computer Science Symposia. Vol. 6 Ed. Prentice-hall, 1972.

Codd E.F. A RELATIONAL MODEL OF DATA FOR LARGE SHARED DATA BANKS. Communications of the ACM 13, 6. June, 1970).

Cuadrado Clara, Cuadrado John, PROLOG GOES TO WORK. Byte. August, 1985.

Cullinet Software Inc., DATABASE DESIGN IN DEFINITION GUIDE.

Cullinet Software Inc., IDMS ARCHITECT USER MANUAL.

Cullinet Software Inc., CULLINET SYSTEM SOFTWARE GLOSARY.

Date C. J., AN INTRODUCTION TO DATABASE SYSTEMS. Vol 2, Addison Wesly Publishing Co.

Date C. J., AN INTRODUCTION TO DATABASE SYSTEMS. 2a. Ed., Addison Wesly Publishing Co., 1977.

Eisenbach Susan, Sadler Chris, DECLARATIVE LANGUAJES: AN OVERVIEW, Byte. August, 1985.

Fagin R., FUNCTIONAL DEPENDENCIES IN A RELATIONAL DATABASE AND PROPOSITIONAL LOGIC, IBM Res. Dev. 21, 6. Noviembre, 1977.

Fairley, Richard, INGENIERIA DE SOFTWARE, McGraw-Hill de México, 1987.

Ferguson Ron, PROLOG: A STEP FORWARD, THE ULTIMATE COMPUTER LANGUAGE, Byte, November, 1981.

Fisher Edward M, BUILDING A. I. BEHIND CLOSED DOORS, Datamation, Vol 32, No 15. August, 1986.

Floyd L. Rutch y Phillipe G. Simbarlow, PSICOLOGIA Y VIDA, Ed. Trillas, México, 1979.

Gagle M., Koehler G. J. and Winston A. DATA-BASE MANAGEMENT SYSTEMS: POWERFUL NEWCOMERS TO MICROCOMPUTERS. Byte. Nov. 1981.

Gottlob. G COMPUTING COVERS FOR EMBEDDED FUNCTIONAL DEPENDENCIES. Intern. Rep. *6-006. Dipartimento di Elettronica, Politecnico di Milano, Italy.

James O. Wittaker, PSICOLOGIA, Ed. Interamericana, 1971.

Jiménez Alejandro. POR QUE PROLOG. Revista 010, Vol 7, # 9, Mayo, 1985.

Kershberg L. Ed. EXPERT DATABASE SYSTEMS. In Proceedings of the 1st International Conference On Expert Database Systems, Kiawah Island, S. C. 1984.

Konopasek Leilos, Jayaraman Sundareson, EXPERT SYSTEMS FOR PERSONAL COMPUTERS, Byte. July, 1984.

Kowalski Robert, LOGIC PROGRAMMING, Byte. August, 1985.

Krajewski R., DATABASE TYPES, Byte, 1984).

Kroenke David M., DATABASE PROCESSING FUNDAMENTALS DESIGN IMPLEMENTATION, Science Research Associates Inc.,Estados Unidos de América, 1983.

Lee R. M. DATABASE INFERENCING FOR DECISION SUPPORT. Decis. Support syst. 1, 1, 1985.

Loizou G.and Thanish P.TESTING A DEPENDENCY-PRESERVING DECOMPOSITION FOR LOSSLESSNESS. Inf. Syst. 8, 1, 1983.

Lucchesi C. L. and Osborn S. L. CANDIDATE KEYS FOR RELATIONS. J. Comput. Syst. Sci. 17, 2, 1978.

Marcot Bruce, TESTING YOUR KNOWLEDGE BASE, A. I. Expert, 1987.

Odette Lou, HOW TO COMPILE PROLOG, A. I. Expert, 1987.

Potter W. D. DESIGN-PRO: A MULTI-MODEL SCHEME DESIGN TOOL IN PROLOG. In proceedings of the 1st International Conference on Expert Database Systems. (Kiawah Island, S. C. Oct. 1984).
Pressman Roger S., SOFTWARE ENGEENERING: A PRACTITIONERS APROACH, McGraw-Hill International Book Co., 1982.

Rich Elaine, ARTIFICIAL INTELLIGENCE, McGraw-Hill, 1983.

Salzberg Steven, KNOWLEDGE REPRESENTATION IN THE REAL WORLD, A.I. Expert, (jul 1987).

Samuel A.L., SOME STUDIES IN MACHINE LEARNING USING THE GAME OF CHECKERS, McGraw-Hill, 1963.

Simons Gary F. DATA ABSTRACTION, Byte (oct 1984).

Tsou D. M. and Fischer P. C. DECOMPOSITION OF A RELATION SHEME INTO BOYCE-CODD NORMAL FORM.ACM-SIGACT 14, 3, (Summer 1982).

Wiederhold Gio, DISEÑO DE BASES DE DATOS, McGraw-Hill, 1985.

Yourton Edward, Constantin Larry, STRUCTURED DESIGN, Prentice-Hall, 1979.

Yu C. T. and Johnson D. T. ON THE COMPLEXITY OF FINDING THE SET OF CANDIDATE KEYS FOR A GIVEN SET OF FUNCTIONAL DEPENDENCIES. Inf. Process. Let 5, 4, (Oct. 1976).

Zaniolo C. and Melkanoff M. A. ON THE DESIGN OF RELATIONAL DATABASE SHEMATA. ACM Trans. Database Syst 6, 1,(Mar. 1981).