UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CONTADURIA Y ADMINISTRACION

FORTALEZAS DE LAS BASES DE DATOS OBJETO RELACIONALES

TESIS PROFESIONAL
QUE PARA OBTENER EL TITULO DE:
LICENCIADO EN INFORMATICA
PRESENTA:

CARINA GUADALUPE SARABIA DELGADO

ASESOR: M. EN I. GRACIELA BRIBIESCA CORREA

MEXICO, D.F. 2002

Paginación

Discontinua

AGRADECIMIENTOS

❖ A Dios:

Por acompañarme amorosamente a cada instante.

❖ A mi Abuela(†) y a mi Madre:

Porque una buena parte de este logro es de ellas y de los esfuerzos y apoyo que me brindan acompañando todos mis pasos y enseñándome a luchar siempre por lo que me propongo.

❖ A mi padre y hermano:

Por su amor y apoyo cada día. Gracias por su paciencia y compañía en todo momento.

❖ A la DGSCA, especialmente a la Dirección de Sistemas y su programa de Becas:

Gracias por todos estos años en los que han brindado a tantos estudiantes apoyo y conocimientos para mejorar nuestra calidad de profesionistas y universitarios.

❖ A la Lic. Rosario Salinas, al L. I. Hugo Reyes y al L. A. Roberto Viveros:

El segundo hogar, ejemplo y compromiso que me han dado impulsan mi vida en todos los aspectos.

❖ Al Ing. Armando Reyes:

Por su asesoría y recomendaciones profesionales tan valiosas cuando más los necesitaba. Muchas Gracias.

❖ A la M. en I. Graciela Bribiesca:

Agradezco su apoyo y confianza durante el desarrollo de este esfuerzo, sin su paciencia y orientación no habría logrado concluirlo.

❖ A la Universidad Nacional Autónoma de México:

El espíritu universitario y el compromiso que significa son un complemento maravilloso a los conocimientos y habilidades adquiridos bajo su tutoría.

❖ A todas las personas que, de algún modo y por confiar en mí, han colaborado en mi desarrollo espiritual e intelectual, deseando sea de su agrado y cumpla con las expectativas de lo que pueden esperar de una universitaria. Gracias a todos los que caminan a mi lado.

INDICE

ÍNDICE

INTRODUCCIÓN 1

OBJETIVO 4

HIPÓTESIS 4

ALCANCE 4

MARCO TEÓRICO 4

EVOLUCIÓN DE LAS BASES DE DATOS 6
Bases de Datos Orientadas a Objetos 6
Bases de datos relacionales extendidas 7
Las Bases de Datos y la Web 8
Data Warehouse 8
Minería de datos 8
Semántica 9
Otras técnicas 9
La evolución prosigue 9

BASES DE DATOS 9
Objetivos de las bases de datos 12
Costos de las bases de datos 12
Seguridad 12

CAPÍTULO I. BASES DE DATOS RELACIONALES 12

FUNDAMENTOS DE LAS BASES DE DATOS RELACIONALES 13
Algebra Relacional 13
Conceptos Asociados A Bases De Datos Relacionales 20

EL MODELO RELACIONAL 22
Modelo Conceptual 22
Modelo Lógico 23
Modelo Físico 23
Componentes del Modelo Relacional. 23
Componentes del Modelo Relacional. 23
Diagrama Entidad-Relación 25
Normalización 28
Reglas De Codd 33

Reglas de Integridad Referencial 37

TECNOLOGÍAS Y HERRAMIENTAS DE BASES DE DATOS RELACIONALES 38
Definición de Sistema Administrador de Bases de Datos 38

COMPONENTES DEL SUBLENGUAJE EMPLEADO POR EL SABD 40

HERRAMIENTAS PARA BASE DE DATOS RELACIONALES 43

CAPÍTULO II. BASES DE DATOS ORIENTADAS A OBJETOS 38

FUNDAMENTOS DE LAS BASES DE DATOS ORIENTADAS A OBJETOS 41
Modelo Matemático Subyacente 41

Conceptos Asociados A Bases De Datos Orientadas a Objetos 42

EL MODELO ORIENTADO A OBJETOS 45
Primer Enfoque de Construcción de Bases de Datos Orientadas a Objetos. 45

Segundo Enfoque de Construcción de Bases de Datos Orientadas a Objetos. 45

Tercer Enfoque de Construcción de Bases de Datos Orientadas a Objetos. 45

COMPONENTES DEL MODELO ORIENTADO A OBJETOS. 46
Diagrama de Clases 49

Normalización en las Bases de Datos Orientadas a objetos. 50

EL ESTÁNDAR ODMG93 51
LAS "REGLAS DE CODD" DE BASES DE DATOS ORIENTADAS A OBJETOS 51
Primer Manifiesto para los Sistemas Manejadores de BDOO 52

TECNOLOGÍAS ORIENTADAS A OBJETOS 55
Definición de Sistemas Administradores de Bases de Datos Orientadas a Objetos 56

DEFINICIÓN DEL SISTEMA ADMINISTRADOR DE BASES DE DATOS ORIENTADAS A OBJETOS 57
COMPONENTES DEL SUBLENGUAJE EMPLEADO POR EL SISTEMA ADMINISTRADOR DE BASES DE DATOS ORIENTADAS A OBJETOS 58
Lenguaje ODL 58

Lenguaje OML 58

Lenguaje OQL 59

HERRAMIENTAS Y PRODUCTOS PARA BASES DE DATOS ORIENTADAS A OBJETOS 60
Lenguaje Unificado de Modelado (Unified Modeling Language - UML) 60

CAPÍTULO III. BASES DE DATOS OBJETO RELACIONALES 57

LAS BASES DE DATOS OBJETO RELACIONALES 57
FUNDAMENTOS DE LAS BASES DE DATOS OBJETO RELACIONALES. 60
Conceptos asociados a las Bases de Datos Objeto Relacionales. 68

EL MODELO OBJETO RELACIONAL 73
Tercer Manifiesto (Las "Reglas de Codd" del Modelo Objeto Relacional) 73

DIAGRAMAS OBJETO-RELACIÓN 90
Mapeo del modelo 91

MODELADO OBJETO-RELACIONAL 58
Objetos como valores de atributos en una tupla. 59

Atributos objeto como vistas *60*

COMPORTAMIENTO *62*

HERENCIA *62*

TECNOLOGÍAS Y HERRAMIENTAS DEL MODELO OBJETO-RELACIONAL 64

Modelado de Roles de Objetos (Object Role Modeling) *67*

Lenguaje Unificado de Modelado (Unified Modeling Language - UML) *67*

Herramientas de Modelado para Sistemas de Administración de Bases de Datos Objeto –Relacionales *69*

CAPÍTULO IV. COMPARACIÓN DE LOS MODELOS **100**

VENTAJAS DE LAS BASES DE DATOS RELACIONALES 100

DESVENTAJAS DE LAS BASES DE DATOS RELACIONALES: 101

VENTAJAS DE LAS BASES DE DATOS ORIENTADAS A OBJETOS 101

DESVENTAJAS DE LAS BASES DE DATOS ORIENTADAS A OBJETOS 103

VENTAJAS DE LAS BASES DE DATOS OBJETO RELACIONALES. 105

DESVENTAJAS 106

CONCLUSIÓN **106**

ANEXO I. TABLA COMPARATIVA DE MODELOS DE BASES DE DATOS **109**

GLOSARIO DE TÉRMINOS **110**

BIBLIOGRAFÍA **111**

REFERENCIAS EN INTERNET **112**

INTRODUCCION

INTRODUCCIÓN

PLANTEAMIENTO DEL PROBLEMA

En la actualidad, la complejidad de las organizaciones de nuestra sociedad demanda la eficiencia en la utilización de datos para generar información útil para la toma de decisiones. Con el nacimiento de las computadoras ha sido posible manejar grandes volúmenes de información con mayor precisión y rapidez; el concepto de sistema de base de datos surge de este proceso evolutivo.

Los requerimientos de información de las organizaciones son cada vez más demandantes, por ello es sumamente importante el papel que las bases de datos representan en el entorno organizacional hoy día. Debido a ello, se vuelve necesario mantenernos al corriente en cuanto a las tendencias existentes en el modelado y construcción de bases de datos.

La tecnología de administración de bases de datos se halla en una etapa muy madura, tanto que las bases de datos han evolucionado mucho durante los pasados 30 años desde los sistemas de archivos rudimentarios hasta sistemas gestores de complejas estructuras de datos que ofrecen un gran número de posibilidades de administración de datos.

Importancia de las Bases de Datos.

Hoy día, las organizaciones encaran una creciente dificultad en la elección de una tecnología de bases de datos que satisfaga sus actuales y futuras necesidades de administración de datos.

Nada como el papel importantísimo que juegan las Bases de Datos en la vida de las organizaciones en nuestros días, puede justificar de mejor forma la necesidad del conocimiento actualizado y profundo de las tecnologías de bases de datos existentes en el mercado. Es a través de este conocimiento que se facilitará la decisión en el momento de elegir el producto adecuado para las necesidades de información de la organización, e incluso, nada mejor que un buen conocimiento de las posibilidades y tendencias de las

tecnologías para prever requerimientos que será necesario cubrir y las oportunidades que la tecnología elegida nos ofrecerá en dicho caso, lo cual repercutirá en una importante reducción de costos de migración de una Base de Datos a otra, que pueden generarse al requerir cambios radicales en el modelo de nuestra base de datos o la adaptación y nuevo desarrollo de aplicaciones operativas que faciliten estos nuevos requerimientos, o bien, en los costos invertidos en las aplicaciones de filtrado o conversión que forman una capa intermedia entre el modelo actual y el nuevo modelo que cumple con los requerimientos de información de una organización que ha sufrido transformaciones importantes. Además de lo anterior, podemos inferir que se presentan reducciones en el tiempo de desarrollo o adaptación de sistemas adecuados que pueden traducirse a partir de nuevos requerimientos de información.

En la evaluación de tecnologías de almacenamiento de datos, siempre se está buscando incrementar el desempeño, la disponibilidad de plataformas de almacenamiento de fácil administración y la mayor cantidad de características benéficas al desempeño organizacional.

Como se puede observar, el estudio de las tecnologías de Bases de Datos puede brindar múltiples beneficios y ahorrar buena cantidad de dinero mientras brinda a los usuarios una gama de facilidades conforme a sus necesidades de información, garantizando la fuente de información para la toma de decisiones que repercutan en favor de la vida económica y operativa de la organización.

Aunado a lo anterior, se observa la importancia del hecho de que las Bases de Datos Relacionales de amplio uso, se habían visto amenazadas por las Bases de Datos Orientadas a Objetos, que pretendían ser la nueva herramienta estándar para el desarrollo de sistemas de información. Sin embargo, dado su costo y otras características, estas Bases de Datos no han logrado usurpar el lugar preferencial que tienen las Bases de Datos Relacionales, en cambio si, han surgido las Bases de Datos Objeto Relacionales, que tratan de unir las características más importantes de ambos tipos para lograr un mayor desempeño.

En el primer capítulo de esta tesis, se repasarán los conceptos principales sobre Bases de Datos Relacionales y sus fundamentos, lo cual nos permitirá ubicarnos en el punto de referencia adecuado para iniciar nuestro análisis. A lo largo de los otros capítulos iremos haciendo referencia a los diversos puntos tratados en relación a este modelo, lo cuál facilitará el trabajo de conclusión final.

El capítulo dos está dedicado a las Bases de Datos Orientadas a Objetos, su fundamento y principales conceptos, que han dado al paradigma Orientado a Objetos la fama e importancia que hoy reviste. Observaremos de forma rápida como modelar una Base de Datos desde este punto de vista y obtendremos más conceptos que completarán el marco teórico necesario para el estudio de las Bases de Datos Objeto Relacionales.

A este modelo se refiere el tercer capítulo, explicando la forma en que los beneficios de los dos modelos anteriores pueden mezclarse para lograr una Base de Datos fácil de diseñar, manipular y mantener, mediante el aprovechamiento de los recursos ya existentes en el mismo modelo de mayor demanda actualmente en el mercado.

Finalmente, el último capítulo ofrece las conclusiones obtenidas durante el estudio realizado.

El Anexo I, muestra un cuadro comparativo que puede resumir el estudio realizado a través de este trabajo, siendo un punto de consulta rápida a la información general en él contenida.

Ante este panorama, podemos concluir que la solución a la elección de la adecuada tecnología de Bases de Datos se encuentra en el buen conocimiento de las opciones existentes en el mercado y del conocimiento de los requerimientos de información de la organización.

OBJETIVO

El objetivo de este trabajo de investigación, es demostrar las ventajas otorgadas por las Bases de Datos Objeto Relacionales para su aplicación en comparación con las tecnologías de Bases de Datos Relacionales y Bases de Datos Orientadas a Objetos para facilitar a las organizaciones la elección de una adecuada tecnología de Bases de Datos, señalando las fortalezas y debilidades de dichas Bases de Datos y permitiéndoles explotar las bondades y beneficios de tecnologías avanzadas con herramientas más económicas.

HIPÓTESIS

Es posible obtener los beneficios del paradigma Orientado a Objetos con menos recursos que los necesarios para la implementación de una Base de Datos Orientada a Objetos a través de una mejor implementación del modelo relacional que permita explotar las posibilidades de herencia, encapsulamiento y polimorfismo que poseen de forma natural y que no han sido implementadas por SQL, el lenguaje de consulta empleado en las Bases de Datos Relacionales tradicionalmente, y considerado en ocasiones su sinónimo.

ALCANCE

En este trabajo, se señalan las condiciones más representativas, algunas ventajas y desventajas de las tres principales tecnologías de Bases de Datos que se encuentran actualmente en el mercado y de mayor demanda, trataremos de hacer una comparación entre ellas, poniendo en una balanza el provecho y perjuicio que de cada una podemos obtener

MARCO TEÓRICO

MARCO TEÓRICO

Cuando uno no sabe donde quiere ir, cualquier camino es bueno. Los usuarios necesitan entender las diferencias entre los diversos manejadores de bases de datos para poder elegir la mejor opción para ellos.

Tanto para las grandes organizaciones, como para aquellas más pequeñas en vías de crecimiento, existe cada día mayor cantidad de datos que manipular, mismos que comienzan a adquirir una importancia crítica para el negocio. Por ello hoy día, la administración del flujo datos es tan importante.

Tradicionalmente los sistemas empresariales de información no se planeaban o diseñaban correctamente, sino que evolucionaban como sistemas independientes que resuelven problemas aislados de una organización. El problema con este enfoque es que los procedimientos requeridos requerirán adecuaciones de acuerdo a los cambios en el ambiente de la organización

En todas partes, las compañías luchan con un creciente flujo de datos que almacenar y administrar correcta y eficientemente. Lo ideal, es que la compañía sea capaz de hacer uso efectivo de las tecnologías de información (TI) como un arma estratégica, y la administración eficiente de los datos es la clave para lograrlo.

Cuando los datos están diseminados a través de diversos sistemas, los tiempos de respuesta de rastreo y los costos de propiedad de la información y de los sistemas instalados se incrementan considerablemente, y en la era de los negocios electrónicos, donde la buena administración del flujo de datos equivale al éxito del negocio, estos problemas pueden llegar rápidamente a ser puntos de competitividad críticos.

Todo esto crea el reto de diseñar bases de datos compactas, estables y de fácil administración, que sean relativamente independientes de las aplicaciones que se construyan sobre ella. El diseño debe permitir rapidez de consulta. Para obtener mayor beneficio de este enfoque se debe analizar la información de la organización y planear su base de datos con cuidado. Si se toma este enfoque sin una buena planeación, los resultados muy bien podrían ser desastrosos. Estos son los motivos por el que las compañías luchan por asegurarse de tener acceso seguro y confiable a sus datos y aplicaciones.

Los primeros sistemas de Bases de Datos que existieron, en los años sesentas o antes, usaban un arreglo jerárquico, tras ellos, a mediados de la década pasada, aparecieron las bases de datos de red o reticulares, basados en una estructura de punteros que requerían procedimientos a bajo nivel para lograr las consultas y transacciones. Los sistemas relacionales nacieron a fines de los sesentas, simplificando drásticamente los modelos previos. Sin embargo, les tomó una década llegar a productos comerciales que cumplieran con los fundamentos relacionales. A finales de los años ochentas, surgieron los sistemas orientados a objetos, tratando de responder a los requerimientos de aplicaciones como CAD (Diseño Asistido por Computadora), CAM (Fabricación Asistida por Computadora) o CAE (Ingeniería Asistida por Computadora), complejas y con objetos compuestos de otros objetos.

Este campo está aun en desarrollo y a pesar de que la mayoría está de acuerdo en la utilidad del paradigma de objetos, no existe todavía un consenso acerca de lo que esta tendencia pueda significar para la industria de sistemas de Bases de Datos.

Todas las organizaciones deben también evolucionar. En este proceso, viejas y nuevas tecnologías deben trabajar juntas para brindar mayores beneficios a las empresas y para fomentar estas condiciones, en los últimos años se ha dado gran difusión a las bases de datos orientadas a objetos y a la extensión de las relacionales, sin embargo, el desarrollo de las tecnologías continua su camino, y es importante observarlas.

Se ha producido, entre las bases de datos relacionales y las bases de datos orientadas a objetos, una colaboración y una competencia: *colaboración* en el intento aun infructuoso de unificar los modelos en uno solo y de hacer convergir los lenguajes SQL y OQL, llegándose al menos a poder acceder en el futuro tanto a unas bases como a las otras en cualquiera de los dos lenguajes; y *competencia* en los intentos de mantener y ampliar las cuotas respectivas de un mercado tan competitivo y dinámico como el de los sistemas de administración de bases de datos.

EVOLUCIÓN DE LAS BASES DE DATOS

Bases de Datos Orientadas a Objetos

Las bases de datos orientadas a objetos (también conocidas como "bases de datos obje(c)tuales"), no aparecieron por modificaciones a modelos de bases de datos anteriores, como pudieran ser las relacionales, sino por la ruptura respecto a las precedentes, partiendo de lenguajes de programación orientados a objetos y dotándolos de persistencia, más que como una derivación de otro tipo de Bases de Datos.

El esfuerzo llevado a cabo por los fabricantes de Sistemas de Administración de Bases de Datos Orientadas a Objetos estos últimos años ha sido realmente grande. Los más importantes de entre ellos constituyeron el ODMG (Object Data Management Group, antes Object Database Management Group) y desarrollaron un modelo de datos común con su lenguaje OQL.

Las bases de datos orientadas a objetos resultan particularmente ventajosas en aplicaciones de CAD, CAM, CAE, etcétera, para los cuáles parecen haber quedado restringidas. No obstante, el peligro para los fabricantes de Sistemas de Administración de Bases de Datos Orientadas a Objetos al quedar limitados al nicho del mercado constituido por estas aplicaciones CAx parece quedar superado por tres razones, no directamente fruto de la tecnología de bases de datos:

1. Java. La versión 2.0 del modelo ODMG especifica interfaces no sólo a los lenguajes C++ y Smalltalk sino también a Java. Cuando los SABDOO soporten esta interfaz, el ascenso imparable de Java será un tanto a su favor;
2. UML. El uso creciente del Lenguaje Unificado de Modelado (*Unified Modeling Language*) para la especificación y diseño de sistemas de información propicia que sus datos se gestionen en bases de datos orientadas a objetos de una forma más intuitiva que en Sistemas de Administración de Bases de Datos al aplicar tecnología de objetos del principio al final de su ciclo de vida, aun cuando esto no sea el único factor en la elección, y el diseño relacional a partir de UML no sea más difícil que el clásico DER.
3. CORBA. Ante la necesidad de datos/objetos distribuidos, la utilización de *Common Object Request Broker Architecture* resulta muchas veces adecuada, y lo será más cuando los productos implementen todos los servicios especificados en la arquitectura.

Bases de datos relacionales extendidas

Las bases de datos relacionales, y sus Sistemas de Administración de Bases de Datos, por su parte, han ido evolucionando estos últimos años para soportar objetos y reglas, y para ampliar el lenguaje SQL extendiéndolo - nuevos tipos, nuevas operaciones- y computacionalmente completo.

El desarrollo del nuevo estándar de SQL ha sido más largo de lo previsto, y no sólo por lo ambicioso de sus objetivos o como consecuencia directa de los intentos de convergencia con el modelo de ODMG y su lenguaje OQL. Indirectamente, estos intentos han obligado a profundizar en el entendimiento del propio modelo relacional, planteándose y resolviendo preguntas como la de saber de qué tipo es una variable que toma valores sobre tuplas de una relación desde el punto de vista de los lenguajes de programación.

También se ha dado una fuerte competencia, sobre todo entre herramientas para soportar multimedia - gráficos, textos, imagen, sonido, vídeo -. En esta evolución, cabe destacar que el impacto de Java da lugar a JDBC y al SQLJ. Por otra parte, a pesar del soporte de reglas, no ligadas a clases como en las bases de datos orientadas a objetos, el campo de las bases de datos deductivas, tan prometedor hace pocos años, no ha eclosionado todavía.

Las Bases de Datos y la Web

La influencia de la World Wide Web lo impregna todo. En su desarrollo se han ignorado las técnicas de bases de datos y se han repetido errores cometidos en las primeras generaciones de Sistemas de Administración de Bases de Datos (SABD). La Web puede verse como una nueva interfaz de acceso a bases de datos, y muchos SABD ya proporcionan herramientas para generar dinámicamente, a partir de sus datos, páginas virtuales en HTML. Pero la Web puede también ser considerada como una inmensa base de datos, o como una confederación de bases de datos - en los servidores -, o incluso como un sinnúmero de páginas heterogéneas con datos semiestructurados.

Data Warehouse

Aunque los "almacenes de datos" (Data Warehouse o DW) son un desarrollo reciente, ya han demostrado que, implantados convenientemente, pueden ser de gran ayuda en la toma de decisiones y en el "proceso analítico en tiempo real" (On Line Analytical Processing u OLAP). Sus datos, extraídos periódicamente de los sistemas operacionales o de otras fuentes e integrados, son no volátiles, relevantes para la empresa, agregados según diversas granularidades en el tiempo y en otras dimensiones. En la administración de los DW existe una gran competencia entre extensiones de los SABD por un lado y productos específicos por otro. La reciente adquisición de Red Brick por Informix es una muestra de los posicionamientos en este campo.

Minería de datos

Llamamos "minería de datos" a lo que es conocido en inglés por Knowledge Discovery in Databases/Data Mining o KDD. Se trata de descubrir conocimientos útiles y previamente no conocidos a partir de grandes volúmenes de datos. Integra técnicas de bases de datos, estadística y de inteligencia artificial.

Las investigaciones se han plasmado rápidamente en productos comerciales, con un desarrollo reciente espectacular apoyado por diversos eventos internacionales como el Congreso KDD (Knowledge Discovery in Databases), y la creación por la ACM de un grupo de interés específico, el SIGKDD.

Semántica

Un mayor nivel semántico ayuda a solucionar problemas de muchos tipos, incluyendo la ingeniería de sistemas de información federados, la administración de flujos de trabajo (Workflow), la administración de transacciones y el control de concurrencia, la seguridad y confidencialidad, o la evolución de esquemas.

Otras técnicas

Ha habido progresos recientes en bases de datos activas y de tiempo real, en bases de datos espaciales, temporales y espacio-temporales, en bases de datos con información incompleta o incierta, y en bibliotecas digitales, entre otras técnicas.

La evolución prosigue

Esta reciente evolución presagia cambios futuros del máximo interés y una de las mejores maneras de seguir estos cambios es mediante los congresos especializados sobre bases de datos.

BASES DE DATOS.

Pensar en un archivo lineal de registros homogéneos como el arquetipo de una base de datos es una forma muy limitada de visualizarla y equivale a imaginar un patín cuando nos piden pensar en un vehículo de transporte. Una verdadera base de datos es típicamente un repositorio de piezas de información heterogéneas pero relacionadas entre sí y que tienen la cualidad de responder a consultas complejas.

Una base de datos es una colección de datos operacionales utilizados por diversas aplicaciones de una organización. Son, de manera general, un conjunto organizado de información en el que un programa de computadora puede seleccionar rápidamente datos específicos. Las bases de datos tradicionales se organizan en archivos que contienen registros, que a su vez contienen campos que almacenan datos.

Una base de datos es una colección o depósito de datos estructurados de una forma particular que tienen una definición y descripción comunes.[1]

Una base de datos es pues, una colección de información organizada y presentada para servir a un propósito específico[2].

Las bases de datos las encontramos prácticamente en cualquier lado, no sólo en los sistemas bancarios, también en todos los sistemas que administren información, como las utilizadas en el programa "Credencial para Votar con Fotografía" del Instituto Federal Electoral (IFE), los censos de población y vivienda del Instituto Nacional de Estadística, Geografía e Informática (INEGI), en todas las instituciones de crédito, financieras y recaudadoras, empresas que llevan inventario de existencias, cartera de clientes, nómina y los ejemplos llegan al infinito.

Toda base de datos debe ser mantenida con ayuda de un Sistema Administrador de Bases de Datos, que es un conjunto de productos de software que permiten definir, manipular y asegurar los datos dentro de éstas. Los principales objetivos de estos sistemas son los siguientes:

1. *Independencia lógica y física de los datos*: se refiere a la capacidad de modificar una definición de esquema en un nivel de la arquitectura sin que esta modificación afecte al nivel inmediatamente superior. Para ello un registro externo no tiene por qué ser igual a su registro correspondiente en el esquema conceptual.

[1] Conference des Staticiens Europeens, 1977
[2] BYERS, Robert A. Introducción a las Bases de Datos con dBIII Plus.

2. *Redundancia mínima:* se trata de evitar la duplicidad de información.
3. *Acceso concurrente por parte de múltiples usuarios:* control de concurrencia mediante técnicas de bloqueo o cerrado de datos accedidos.
4. *Distribución geográfica de los datos:* la independencia lógica y física facilita la posibilidad de sistemas de bases de datos distribuidas. Los datos pueden encontrarse en otra habitación, otro edificio e incluso otro país. El usuario no tiene por qué preocuparse de la localización geográfica de los datos a los que accede.
5. *Integridad de los datos:* se refiere a las medidas de seguridad que impiden que se introduzcan datos erróneos. Esto puede suceder tanto por motivos físicos (defectos de hardware, actualización incompleta debido a causas externas), como de operación (introducción de datos incoherentes).
6. *Consultas complejas optimizadas:* permite la ejecución rápida de las consultas.
7. *Seguridad de acceso y auditoría:* se refiere al derecho de acceso a los datos contenidos en la base de datos por parte de personas y organismos. El sistema de auditoría mantiene el control de acceso a la base de datos, con el objeto de saber qué o quién realizó una determinada actualización y en qué momento.
8. *Respaldo y recuperación:* se refiere a la capacidad de un sistema de base de datos de recuperar su estado en un momento previo a cualquier tipo de falla.
9. *Acceso a través de lenguajes de programación estándar:* se refiere a la posibilidad de acceder a los datos de una base de datos mediante lenguajes de programación ajenos al sistema de base de datos propiamente dicho.
10. *Gestión del almacenamiento secundario: Es soportada por un conjunto de mecanismos que no son visibles al usuario, tales como gestión de índices, agrupación de datos, selección del camino de acceso, optimización de consultas, etc. Estos mecanismos evitan que los programadores tengan que escribir programas para mantener índices, asignar el almacenamiento en disco, o trasladar los datos entre el disco y la memoria principal, creándose de esta forma una independencia entre los niveles lógicos y físicos del sistema.*

Objetivos de las bases de datos

Entre los principales objetivos de una base de datos tenemos:

- Proteger el valor de los datos
- Hacer que las fuentes de datos sean las responsables de los mismos.
- Permitir a la organización mejor control y seguimiento de planes de negocios y logre sus metas
- Reducir los costos resultantes de los esfuerzos por mejorar el desempeño
- Responder a consultas sobre los datos que contiene, y
- Ejecutar transacciones

Costos de las bases de datos

Los costos de establecer y operar en un ambiente de bases de datos incluyen:

- Costos de la Tecnología del Sistemas Administradores de Bases de Datos.
- Capacitación.
- Costos de mantenimiento.
- Costos de actualización.
- Costos de respaldo.
- Costos de Planeación e Implementación
- Diseño.
- Adquisición de software y equipo.
- Costos de Riesgo.
- Servidor espejo.

Seguridad

La seguridad implica garantizar que los usuarios están autorizados para llevar acabo ciertas tareas sobre la Base de Datos. El problema de la seguridad tiene muchos aspectos, entre ellos los siguientes.

- Aspectos legales, sociales y éticos (*i.e.* ¿tiene la persona que solicita el crédito de un cliente, digamos, derecho legal a obtener la información solicitada?).
- Controles físicos (*i.e.* ¿deberá cerrar o resguardar de alguna otra manera el cuarto de computadoras?).
- Cuestiones de política interna (*i.e.* ¿cómo decide la empresa quiénes pueden tener acceso a qué?).
- Problemas de operación (*i.e.* cómo mantener en secreto las contraseñas, frecuencia de cambio, etcétera).
- Controles de equipo (*i.e.* ¿posee características de seguridad tales como modo de operación privilegiado?).
- Seguridad del sistema operativo (*i.e.* ¿borra el sistema operativo el contenido de las áreas de almacenamiento y los archivos de datos cuando ya no se necesitan?).
- Seguridad en la base de datos: debe considerarse en dos términos, usuario o seguridad de objetos en la Base de Datos concepto que se refiere a permisos de los diferentes usuarios para hacer uso de tablas, procedimientos almacenados, disparadores, etcétera y seguridad de operaciones en la que se manejan permisos para modificar (selección, insertar, borrar, actualizar) la base de datos.

CAPÍTULO I

CAPÍTULO I. BASES DE DATOS RELACIONALES

En 1968 en el IBM Research Laboratory en San José, California, con un modelo abstracto de información se inició el trabajo que dio como resultado el modelo de datos relacional. El objetivo del trabajo era encontrar un fundamento teórico de los diferentes aspectos de un SABD completamente ajeno de los aspectos de un proceso físico dentro de una máquina en particular, este modelo es el que actualmente se conoce como Modelo Relacional o Estructura de Datos Relacionales. Una de las partes más importantes de este modelo es el concepto de una estructura que incluye los operadores lógicos utilizados para toda la manipulación de datos en una estructura relacional. Se diseña un tipo de base de datos que se conecta más fácilmente a la realidad. Un concepto que la teoría relacional pone de relieve, es el concepto de dominio.

El primer nombre que recibió el lenguaje manipulador de datos fue ALPHA y posteriormente cambió a SQL (Structured Query Language). El trabajo en referencia propició que se empezara a diseñar y desarrollar sistemas basados en la teoría del modelo relacional. Cada evento se almacena sólo una vez, dando consistencia, ahorrando recursos y facilitando sus actualizaciones.

Una base de datos relacional es aquella donde todos los datos visibles al usuario están organizados como tablas de valores, y en donde todas las operaciones de la base de datos operan sobre estas tablas. Poseen un modelo y lenguaje independiente de la implementación para operar en ella.

Una base de datos relacional es un conjunto de datos interrelacionados con independencia física y lógica, consistentes, íntegros y con redundancia controlada, almacenados más o menos permanentemente en una computadora, de forma que:

- Los datos son compartidos por diferentes usuarios y programas de aplicación; existe un mecanismo común para inserción, actualización, borrado y consulta de los datos.

- Tanto los usuarios finales como los programas de aplicación no necesitan conocer los detalles de las estructuras de almacenamiento.

FUNDAMENTOS DE LAS BASES DE DATOS RELACIONALES

ALGEBRA RELACIONAL

Definición

El álgebra relacional consiste en una colección de operaciones sobre relaciones donde cada operación toma una o más relaciones como sus operandos y produce otra relación como resultado. Debido a que el resultado de una operación de álgebra relacional es una relación, ésta a su vez puede ser sujeto de posteriores operaciones algebraicas.

En el álgebra relacional se consideran dos tipos de operadores:

- Los operadores tradicionales sobre conjuntos: unión, intersección, diferencia y producto cartesiano.
- Los operadores especiales: proyección, selección, unión y división.

El álgebra relacional es un lenguaje relacionalmente completo. Esto significa que tiene la misma potencia de expresión que el cálculo relacional. Es decir, cualquier fórmula de álgebra relacional es expresable en una sentencia de cálculo relacional, y viceversa. Por tanto, no es necesario inventar más operadores además de los de unión, intersección, diferencia, división, producto cartesiano, proyección, selección y yunción o join, a no ser que queramos dotar al álgebra de más potencia expresiva que el cálculo. Algunos lenguajes relacionales permiten algunas operaciones no expresables en cálculo o álgebra.

Operadores de Conjuntos.

Las bases de datos relacionales están basadas en el concepto matemático de relaciones entre conjuntos. Así las operaciones que se pueden efectuar entre relaciones son tanto las comunes a los conjuntos, unión, intersección, diferencia, producto cartesiano; como las específicas de las relaciones, como selección, proyección, etc.

Si q, r y s son relaciones con todos los dominios iguales, esto es, con el mismo esquema, se les puede aplicar las operaciones típicas de conjuntos.

Unión

Construye una relación formada por todas las tuplas que aparecen en cualquiera de las dos relaciones especificadas o la suma de los elementos de dos conjuntos.

$r \cup s$ Es la relación sobre los mismos dominios que contiene las eneadas que están en r, en s o en ambas. Esto es, son todos los elementos que se encuentran en el conjunto s o en el r. Ejemplo:

Sean r y s relaciones con esquema {A,B,C}

r:

A	B	C
a1	b1	c1
a1	b2	c1
a2	b1	c2

s:

A	B	C
a1	b1	c1
a2	b2	c1
a2	b2	c2

$r \cup s$:

A	B	C
a1	b1	c1
a1	b2	c1
a2	b1	c2
a2	b2	c1
a2	b2	c2

Intersección

Construye una relación formada por aquellas tuplas que aparezcan en las dos relaciones especificadas, es decir, son los elementos contenidos en ambos conjuntos.

r/s ó r ∩ s Es la relación que contiene las eneadas que están en r y en s. Indica todos los elementos que se encuentran tanto en el conjunto r como en el conjunto s, o bien, os elementos comunes a los dos conjuntos. Ejemplo:

Sean r y s relaciones con esquema {A,B,C}

r:

A	B	C
a1	b1	c1
a1	b2	c1
a2	b1	c2

s:

A	B	C
a1	b1	c1
a2	b2	c1
a2	b2	c2

r ∪ s:

A	B	C
a1	b1	c1

Diferencia

Construye una relación formada por todas las tuplas de la primera relación que no aparezcan en la segunda de las dos relaciones especificadas o bien, los elementos que se encuentran en el primer conjunto pero no en el segundo.

r - s Es la relación con las eneadas que están en r pero no en s. Corresponde a obtener los elementos del conjunto r que no se encuentran en el conjunto s. Ejemplo:

Sean r y s relaciones con esquema {A,B,C}

r:

A	B	C
a1	b1	c1
a1	b2	c1
a2	b1	c2

s:

A	B	C
a1	b1	c1
a2	b2	c1
a2	b2	c2

r - s:

A	B	C
a1	b2	c1
a2	b1	c2

Producto Cartesiano

A partir de dos relaciones especificadas, construye una relación que contiene todas las combinaciones posibles de tuplas, una de cada una de las dos, esto es, los pares ordenados.

rXq Obtiene todas las eneadas que se construyen concatenando cada eneada de r con otra de s. En este caso los dominios de r y s no tienen que ser los mismos. Ejemplo:

r:

A	B	C
a1	b1	c1
a2	b1	c2
a2	b1	c2

q:

D	E	F
d1	e1	f1
d2	e2	f1
d2	e2	f2

r X q:

A	B	C	D	E	F
a1	b1	c1	d1	e1	f1
a1	b1	c1	d2	e2	f1
a1	b1	c1	d2	e2	f2
a1	b2	c1	d1	e1	f1
a1	b2	c1	d2	e2	f1
a1	b2	c1	d2	e2	f2
a2	b1	c2	d2	e2	f1
a2	b1	c2	d2	e2	f2

División:

Toma dos relaciones, una binaria y una unaria, y construye una relación formada por todos los valores de un atributo de la relación binaria que concuerda con todos los valores en la relación unaria.

r:

A	B	C
a1	b1	c1
a2	b1	c2
a2	b1	c2

q:

D
c1
c2
c3

r Div q:

A	B	C	D
a1	b1	c1	c1
a1	b1	c2	c2
a1	b1	c3	c3

De igual forma, si r y s son relaciones con todos los dominios iguales, esto es, con el mismo esquema, se les puede aplicar las operaciones relacionales.

Operadores Relacionales

Proyección

Es una operación unaria. El resultado es un subconjunto de dominios, permite obtener subrelaciones de otras más grandes seleccionando algunos atributos. Se eliminan, luego, las eneadas repetidas. Extrae los atributos especificados de una relación dada:

Primero obtendríamos de r:

A	C
a1	c1
a2	c2
a2	c2

Y después eliminamos las eneadas repetidas obteniendo:

A	C
a1	c1
a2	c2

Selección o Restrict

Produce un subconjunto de las eneadas de la relación que cumplen con una condición (simple o compuesta) sobre los valores para uno o varios de los atributos. Extrae las tuplas especificadas de una relación dada. Por ejemplo, seleccionemos los elementos de r donde existe c2:

r:

A	B	C
a2	b1	c2
a2	b1	c2

Unión o Reunión (JOIN)

A partir de dos relaciones especificadas, construye una relación que contiene todas las posibles combinaciones de tuplas de las dos relaciones, tales que, las dos tuplas participantes en una combinación dada satisfagan alguna condición especificada. Construye una relación formada por todas las eneadas que aparecen en cualquiera de las dos relaciones especificadas en que se cumple alguna condición en dominios comunes. Se logra concatenando una eneada de r con una de q, de modo que cumpla una condición en los dominios comunes. Si no hay dominios comunes, es un producto cartesiano. Sean q y r dos conjuntos como siguen:

q:

D	E	F
c1	e1	f1
c2	e2	f1
c2	e2	f2

r:

A	B	C
A1	b1	c1
a2	b1	c2
a2	b1	c2

Luego, en la unión, con la condición de que C y D sean iguales tendríamos:

r∪q:

A	B	C	D	E	F
a1	b1	**c1**	**c1**	e1	f1
a2	b1	**c2**	**c2**	e2	f1
a2	b1	**c2**	**c2**	e2	f2

También es posible efectuar una reunión en atributos que tengan diferentes nombres (como en el ejemplo), pero el mismo dominio. En este caso se llama equireunión. Si la condición es sobre atributos con nombres distintos, mismo dominio pero la condición no es sobre igualdades, se le llama reunión donde indica que tipo de condición se tiene (<, >, etc.).

Operaciones Primitivas

En el álgebra relacional se incluyen las operaciones unión, intersección, diferencia, producto cartesiano, proyección, selección, permutación y reunión. No es necesario tenerlas todas; sino solo las primitivas indispensables de las cuales se pueden obtener las demás: unión, diferencia, producto cartesiano, proyección y selección.

Por ejemplo la intersección se puede expresar en términos de la diferencia como sigue: r/s = (r-(r-s)) y la reunión se obtiene de la selección de los elementos que cumplen con una condición en el producto cartesiano. Esto es importante pues algunos SABD comerciales proporcionan sólo parte de los operadores.

Conceptos Asociados a Bases de Datos Relacionales

Redundancia

Repetición de los mismos datos a través de diferentes registros, aplicaciones o archivos. Esto sucede cuando en la Base cada aplicación tiene sus propios datos, provocando considerable redundancia con el consecuente desperdicio de espacio de almacenamiento.

Puesto que los datos son requeridos por múltiples aplicaciones, con frecuencia se almacenan repetidamente. Esto conduce a problemas de integridad. No es del todo posible o deseable eliminar toda la redundancia, pero sí necesario, controlarla y considerar la propagación de actualizaciones.

Consistencia

El concepto de consistencia nos lleva necesariamente al de integridad referencial. Desde luego, una base de datos que se halle en estado de inconsistencia puede suministrar información incorrecta o contradictoria. Si un cierto dato está representado por dos entradas distintas en la base de datos, y el SABD no está consciente de esta duplicidad (es decir, la redundancia no está controlada), habrá ocasiones en que las dos entradas no coincidan. Es en casos como éste, cuando se dice que la base de datos es inconsistente,

pues no todos los datos almacenados representan hechos reales. Si la redundancia no se elimina, pero sí se controla, el SABD podrá garantizar la consistencia de la base de datos desde el punto de vista del usuario, asegurándose de aplicar en forma inmediata a otras entradas cualquier modificación a cada una de ellas.

Integridad

Integridad implica asegurar que lo que se trata de hacer es correcto. El problema de la integridad radica en asegurar que la información de la base de datos sea correcta. La inconsistencia entre dos entradas que representan al mismo "hecho" es un ejemplo de falta de integridad (que, por supuesto, sólo ocurre si existe redundancia en los datos almacenados). Es conveniente señalar que la integridad de los datos es más importante en un sistema de base de datos que un sistema de archivos privados, precisamente por que el primero se comparte y porque sin procedimientos de validación adecuados es posible que un programa con errores genere datos incorrectos que afecten a otros programas que utilicen esa información.

Así, como se mencionó anteriormente, cuando en una relación de información se modifica algún elemento que se encuentre en varias tablas y no se afecta a las otras teniendo información inconsistente en las tablas, en este momento decimos que hay falta de integridad.

La falta de integridad de los datos también puede deberse a una mala verificación de la vigencia de los datos al hacer cambios. Antes del advenimiento de la tecnología de bases de datos, los intentos por integrar los datos eran más difíciles debido a:

- La insuficiente seguridad proporcionada a los datos almacenados.
- Los inadecuados procedimientos de recuperación en caso de falla.
- La dificultad en el manejo de registros largos.
- La inflexibilidad para hacer cambios.
- Los altos costos de programación y mantenimiento.
- La dificultad para manejar los procedimientos en las operaciones de computación (negligencia y errores humanos).

Seguridad

La seguridad implica asegurar que los usuarios están autorizados para llevar acabo lo que tratan de hacer. El problema de la seguridad tiene muchos aspectos, pero deben considerarse fundamentalmente dos términos:

- Seguridad de Objetos. Se refiere a los permisos de los diferentes usuarios para poder hacer uso o tener acceso a tablas, procedimientos almacenados, disparadores, etc.
- Seguridad de operaciones. Aquí se manejan permisos para poder modificar (insertar, seleccionar, borrar, actualizar) la base de datos.

El Modelo Relacional

Un Sistema Administrador de Bases de Datos utiliza un modelo de datos para definir la estructura fundamental de las mismas. Este modelo es una representación abstracta que define la forma en que los elementos son organizados y relacionados, haciendo uso de varias representaciones o diversos tipos de modelos definidos por el administrador para representar la base de datos.

Modelo Conceptual

En el modelo conceptual se va a realizar un análisis de datos, así mismo se consideran las aplicaciones existentes y potenciales. Define y modela aspectos importantes de la información que el negocio necesita saber o tener y las relaciones entre dicha información. Es el primer proceso del modelo top-down (referirse a la Figura 1) para el desarrollo de bases de datos. Se ejecuta durante las fases de análisis y estrategia del ciclo de desarrollo del sistema y debe basarse en la visión del usuario acerca del proceso a automatizar y los datos implicados.

Modelo Lógico

Implica además de lo enunciado en el modelo anterior, requerimientos y procedimientos impuestos por un SABD. Colección de reglas generales de integridad, que limitan el conjunto de casos de los tipos de objetos que aparecen en forma legal en cualquier base de datos que se ajuste al modelo.

Modelo Físico

Colección de operadores, aplicables a los casos de objetos para obtener información y para otros propósitos. Especifica cómo se almacenarán los datos, el espacio que será ocupado, métodos de acceso rápido a los datos, etcétera.

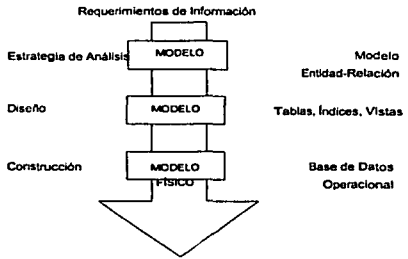


Figura 1. Modelo Top-Down

Componentes del Modelo Relacional.

Tablas

Dentro del enfoque relacional una tabla es conocida como una **Relación**. Es una abstracción del almacenamiento físico de los datos. Una es una tabla tiene dos dimensiones con las siguientes propiedades:

- Cada columna contiene valores relativos al mismo atributo, y cada valor de una columna de la tabla debe ser simple (un solo valor).
- Cada columna tiene un nombre distinto (nombre del atributo), y su orden no es importante.
- Cada renglón es distinto, esto es, un renglón no puede duplicarse en otro para un grupo de columnas seleccionadas como llave.
- Cada atributo no llave debe depender sólo de la llave primaria de la relación.

Tupla

Conjunto de valores que componen un renglón de la relación. Es equivalente a una instancia de un registro.

Grado de una tupla.

Número de atributos que tiene una tupla (n de una n-tupla)

Dominio.

Conjunto de todos los valores posibles para un atributo.

TERMINO RELACIONAL	EQUIVALENTES
Relación	Tabla
Tupla	Fila o registro
Cardinalidad	Número de filas
Atributo	Columna o campo
Grado	Número de columnas
Llave Primaria	Identificador único
Dominio	Fondos de valores legales

Figura 2. Equivalencias de términos relacionales

Tipo de Llaves

Llave primaria.

El atributo que identifica de manera única e inequívoca a un registro. No debe haber dos tuplas que tengan el mismo valor en todos los dominios de la llave primaria. Por lo tanto, con sólo conocer el valor de la llave primaria para un tupla será suficiente para identificarlo de manera única.

Llave extranjera o foránea.

Llave primaria que es la llave primaria en otra relación. Son la materialización de las asociaciones entre las entidades. Son llaves que son compartidas como atributos por dos tablas para lograr una relación entre ellas.

Diagrama Entidad-Relación

El diagrama entidad-relación (DER) se utiliza como una herramienta de comunicación entre los analistas y diseñadores de sistemas y los usuarios finales durante las fases de análisis de requerimientos y de diseño conceptual debido a que es simple y fácil de entender.

Conceptos Básicos del Diagrama Entidad-Relación

El modelo de datos entidad-relación se basa en una percepción de un mundo real que consiste en un conjunto de objetos básicos llamados entidades y relaciones.

Entidad.

Es el objeto principal del cual se tiene que almacenar información, normalmente denotando una persona, lugar, cosa o evento. En un diagrama entidad-relación las entidades se representan con un rectángulo; un sustantivo en español corresponde al nombre de la entidad en el DER. Tienen propiedades o atributos.

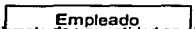


Figura 3. Ejemplo de una entidad en el DER

Relación.

Representa la asociación o correspondencia entre 2 entidades o de una entidad consigo misma; es binaria, es decir, puede leerse en dos direcciones o sentidos. Asocia una entidad de un conjunto a una o varias entidades de otro. En un DER las relaciones se representan con líneas conectando las entidades relacionadas; normalmente un verbo corresponde a la relación. Formalmente, una asociación es un subconjunto del producto cartesiano de una lista de dominios o conjunto de pares ordenados (x,y) de forma tal que el dominio R (dominio o conjunto de valores posibles de la relación) es el conjunto de todos los objetos x tal que (x,y) $\in$ R para alguna y, además de que el rango de R es el conjunto de todos los objetos y tal que (x,y) $\in$ R para alguna x, es decir

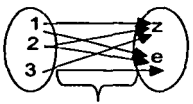


Figura 4. Esquema de una relación en el modelo Entidad Relación

Sintaxis:

	{ Debe }			{ Uno / más }	o	
Cada entidad 1		ser	Nombre de la relación			Entidad 2
	Puede			Uno y sólo uno		

Cardinalidad

La cardinalidad de una relación especifica el tipo de asociación de las ocurrencias de las entidades de la relación. Los valores de la cardinalidad son de "uno" o "muchos". Los tipos básicos de cardinalidad son los siguientes: uno a uno, uno a muchos y muchos a muchos.

Relación de "uno a muchos".

En este caso el identificador de la entidad correspondiente a la cardinalidad "uno" pasa a ser la llave foránea de la tabla correspondiente a la entidad con cardinalidad "muchos".

CLIENTE

Cve_cliente	Nombre

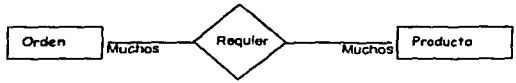
ORDEN

Cve_orden	Cve_cliente	Monto
	Llave Foránea	

Figura 5. Ejemplo de Cardinalidad Uno a Muchos

Relación de "muchos a muchos".

En estos casos es necesario incluir una tabla que corresponda a la relación. Esta tabla contendrá los identificadores de las dos entidades asociadas y los campos propios de la relación.



PRODUCTO

Prod_No	Descripcion	PrecioUnit

ORDEN

Cve_orden	Cve_cliente	Monto
	Llave Foránea	

REQUERIMIENTOS

Cve_orden	Prod_No

Figura 6. Ejemplo de Cardinalidad Muchos a Muchos

Normalización

El proceso de cristalización de las entidades y sus relaciones en formatos de tabla usando los conceptos relacionales se llama proceso de normalización. El proceso de reducción o normalización no es una función de los valores de los datos que aparecen en las relaciones en algún momento determinado, sino que es una función de las relaciones entre los atributos. Por lo tanto el proceso de *normalización* es una disciplina que consiste en *agrupar a los campos de datos en un conjunto de relaciones o tablas que representan a las entidades, sus características y sus relaciones de forma adecuada.*

La teoría de normalización está basada en la observación de que cierto conjunto de relaciones presenta mejores propiedades en un medio de actualización, inserción y supresión, que las que presentan otros conjuntos de relaciones que contienen los mismos datos. Para seguir el proceso de normalización, es absolutamente necesario que el diseñador de la base de datos entienda la semántica de la información, por ello, la insistencia en la importancia de las fuentes de información y la comprensión del contexto alrededor del cual se diseña la base de datos.

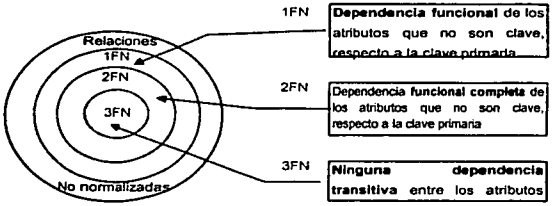
La razón de usar el procedimiento de normalización es asegurar que el modelo conceptual de la base de datos funcionará. Esto no significa que una estructura no normalizada no funcionará, sino que puede causar algunos problemas cuando los programadores de aplicación traten de modificar la base de datos para insertar, actualizar o eliminar datos de formas diversas.

Las *formas normales* son una serie de restricciones que se definen sobre las estructuras relacionales para evitar, como ya se señaló, anomalías al efectuar adiciones, eliminaciones o actualizaciones de tuplas. Con el fin de conseguir que una relación cumpla con una forma normal se efectúa un proceso de descomposición. Ésta implica dividir los atributos de una relación en dos subconjuntos (posiblemente con una intersección no vacía) sin que por ello se pierda alguna información contenida en la relación original.

Las formas de normalización fueron propuestas originalmente por Codd, entre 1971 y 1972. Posteriormente varios investigadores continuaron trabajando en esta teoría y a lo largo del tiempo han surgido varias formas de normalización que complementan y refuerzan a las enunciadas por Codd.

En cada modelo de datos uno o más campos de datos se agrupan para representar entidades y sus relaciones. En los agrupamientos de los campos de datos pueden darse tres tipos generales de problemas, y la eliminación de cada uno de éstos da pie a las tres formas normalizadas básicas de relaciones o tablas (Ver figura 7).

La siguiente figura muestra las tres formas normales:



- Toda relación que está en 1FN es un caso especial de una relación no normalizada. Pero no toda relación no normalizada esta en 1FN.

Figura 7. Formas Normales Básicas

Un concepto que debemos tener claro antes de proseguir en la Normalización es la dependencia funcional.

Dependencia funcional

Una dependencia funcional se representa cuando los valores de un conjunto de atributos de una tupla determinan de manera única los valores de otro conjunto de atributos.

EJEMPLO:

En la siguiente tabla se muestra la entidad ASIGNADOS. Esta entidad indica qué piloto vuela en un día determinado y a qué hora parte el avión. No se permite cualquier combinación de pilotos, vuelos, fechas y salidas ya que se aplican las siguientes restricciones:

- Cada vuelo tiene solamente una hora de salida
- Hay un solo vuelo por cada piloto, fecha y hora de salida
- Para cada vuelo y fecha hay un solo piloto

PILOTO	VUELO	FECHA	SALIDA
Cueto	83	9 Ago	10:15a
Cueto	116	10 Ago	1:25p
Carrasco	281	8 Ago	5:50p
Carrasco	301	12 Ago	6:35p
Carrasco	83	11 Ago	10:15p
Correa	83	13 Ago	10:15p
Correa	116	12 Ago	1:25p
Coz	281	9 Ago	5:50p
Coz	281	13 Ago	5:50p
Coz	412	15 Ago	1:25p

Las restricciones citadas podrían parafrasearse como:

- SALIDA depende funcionalmente de VUELO
- VUELO depende funcionalmente de PILOTO FECHA SALIDA
- PILOTO depende funcionalmente de VUELO FECHA

Primera Forma Normal

Una relación normalizada es una relación que tiene sólo valores elementales (o simples) en la intersección de cada renglón y columna. Así, una relación normalizada no tiene grupos repetitivos.

El primer paso de la normalización consiste en transformar los campos de datos a una tabla de dos dimensiones (filas y columnas) en donde, para cada tupla, exista uno y sólo un valor para cada atributo correspondiente. Normalmente en este paso se requiere eliminar ocurrencias repetidas de campos de datos y buscar campos que puedan ser divididos en "subcampos" para asegurar mayor integridad en la base de datos; i.e.: el nombre puede ser dividido en apellido paterno, apellido materno y nombre (s) para garantizar que serán capturados los tres campos en forma y orden adecuado.

Enunciado:

Una entidad R esta en Primera Forma Norma (1FN) si los valores para cada atributo de A ϵ R, son atómicos. Esto implica que los valores en el dominio no serán listas o conjuntos de valores, ni estarán repetidos innecesariamente.

Una relación está en 1FN si no contiene grupos repetitivos.

Las actualizaciones representan un problema potencial si la entidad no está en 1FN pues el resultado de las actualizaciones es ambiguo.

Segunda Forma Normal

El segundo paso de la normalización es establecer las claves y relacionarlas con los campos de datos. En la primera forma normalizada, el renglón entero de la tabla (tupla) depende de todos los campos de claves. En la segunda forma normalizada, se hace un intento de establecer los campos de datos que están relacionados con alguna parte de la clave completa. Si los campos de datos sólo dependen de una parte de la clave, ésta y los campos conectados a la clave parcial son susceptibles de separarse en registros independientes. La división de la primera tabla normalizada, en una serie de tablas en las que cada campo sólo depende de la clave completa se llama la segunda forma normalizada.

Definiremos al atributo A de la entidad R como primario si forma parte de la llave de R y como no primario en cualquier otro caso.

Se entiende por dependencia parcial cuando los atributos no llave dependen sólo de una parte de una llave compuesta.

Enunciado:

Una entidad R en 1FN, estará en Segunda Forma Normal (2FN) si no existe un atributo no primario (que no forme parte de la llave primaria de la tabla) que dependa parcialmente de la llave de R.

Una relación está en 2FN, si está en primera forma normal y se han eliminado las dependencias parciales.

Con el fin de evitar las anomalías aún presentadas, tendríamos que descomponer la entidad para que cumpla con la 2FN.

Tercera Forma Normal

El tercer paso consiste en separar los campos de las segundas relaciones normales, que aun dependan sólo de una parte de la clave y que por tanto deben tener una existencia independiente en la base de datos. Esto se hace de forma tal que la información sobre estos campos pueda introducirse separadamente a partir de las relaciones en las que se encuentra implicada.

Enunciado:

Decimos que una entidad R está en Tercera Forma Normal (3NF) si está en 1NF y no existe algún atributo no primario en R que depende transitivamente de la llave primaria de R.

Una relación está en tercera forma normal si está en segunda forma normal y no tiene dependencias transitivas.

Una dependencia transitiva ocurre cuando un atributo no llave depende de uno o más atributos no llave.

En resumen, los **pasos de la normalización** son los siguientes:

1. Una vez que se tiene el esquema relacional, se revisa que no haya ninguna relación no normalizada, esto es, con grupos repetitivos.
2. Se eliminan todos los grupos repetitivos de esta relación, obteniendo un conjunto de relaciones en *primera forma normal (1FN).*
3. Se eliminan las dependencias funcionales parciales, para obtener relaciones en segunda forma normal (2FN).
4. Finalmente, se eliminan las dependencias transitivas, creando relaciones en *tercera forma normal (3FN).*

Reglas de Codd

Regla 0

Cualquier SABD relacional, deberá manejar, completamente, las Bases de Datos por medio de sus capacidades relacionales (*"For any system that is advertised as, or claimed to be, a relational Data Base management system, that system must be able to manage Data Bases entirely through its relational capabilities."*).

Regla 1 (The Information Rule)

Toda información en una BDR se representa de manera explícita a nivel lógico y exactamente de una sola manera, como valores en una tabla (*"All information in a RDB is represented explicity at the logical level and in exactly one way by values in tables"*).

Regla 2 (Guaranteed Access Rule)

Se garantiza que todos y cada uno de los datos en una BDR pueden ser leídos por una combinación de nombre de la tabla, valor de la llave primaria y nombre de la columna (*"Each and every datum (atomic value) in a relational Data Base is guaranteed to be logically accesible by resorting to a combination of table name, primary key value and column name"*).

Regla 3 (Systematic Treatment of Null Values)

En un SABD totalmente relacional se soportan los valores nulos (que son distintos de una cadena de caracteres vacía o de una cadena con caracteres en blanco o de cero o cualquier otro número). Para representar información faltante o no aplicable de una forma consistente independientemente del tipo de dato (*"Null values (distinct from the empty character string or string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independient of data type"*).

Regla 4 (Dynamic on-line Catalog Based on the Relational Model)

La descripción de la base de datos se representa en el nivel lógico de la misma forma que los datos ordinarios, de tal suerte que los usuarios autorizados puedan aplicar el mismo lenguaje relacional para consultarla, que aquel que emplean con sus datos habituales (*"The DataBase description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data"*).

Regla 5 (Comprehensive Data Sublanguage)

Deberá contener un Sublenguaje de datos completo que permita:
Definición de datos
Definición de vistas
Manipulación de datos
Restricciones de integridad (manejo)

Autorización
Inicio y fin de una transacción

("*Data Definition*
View definition
Data manipulation
Integrity constrains
Authorization
Transaction boundaries").

Regla 6 (View Updating Rule)

Todas las vistas que teóricamente sean actualizables, deben ser actualizadas por medio del sistema ("*All view that are theoretically updatable are also updatable by the system*").

Regla 7 (High-Level Insert, Update and Delete)

La posibilidad de manejar una relación base o una relación derivada como un solo operador se aplica a la lectura, inserción, modificación y eliminación de datos ("*The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update and deletion of data*").

Regla 8 (Physical Data Independence)

Los programas de aplicación y la actividad en terminales no deberán ser afectados por cambios en el almacenamiento físico de los datos o en los métodos de acceso ("*Application program and terminal activity remain logically unimpared whenever any changes are made in either storage representations or access methods.*").

Regla 9 (Logical Data Independence)

Los programas de aplicación y la actividad en terminales no deberán ser afectados por cambios de cualquier tipo, que preserven la información y que teóricamente permitan la afectación, en las tablas base (*"Application program and terminal activities remain logically unimpared when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables"*).

Regla 10 (Integrity Independence)

Las restricciones de integridad de una Base de datos deberán poder definirse con el mismo sublenguaje de datos relacional y deberá almacenarse en el catálogo, no en los programas de aplicación (*"Integrity constraints specific to a particular relational Data Base must be definable the relational data sublanguage and storable in the catalog, not in the applications programs"*).

Regla 11 (Distribution Independence)

Un SABD relacional tiene independencia de distribución (*"A relational DBMS has distribution independence"*).

Regla 12 (Nonsubversion Rule)

Si un sistema relacional tiene un lenguaje de bajo nivel (que opere un registro cada vez), ese lenguaje no deberá poder emplearse para alterar las reglas de integridad y las restricciones expresadas en el lenguaje relacional de alto nivel (*"If a relational system has a low level languaje (single record at a time), that low level cannot be used to subvert or bypass the integrity rules and constrainsts expressed in the higher level relational language"*).

Reglas de Integridad Referencial

Manejo de Valores Nulos

En principio, cualquier atributo en cualquier relación, puede aceptar valores nulos. La interpretación de un valor nulo es: "valor desconocido en este momento" o "valor no aplicable". La necesidad de los valores nulos proviene del hecho de que frecuentemente la información sobre el mundo es incompleta y requerimos de algún medio para manejar esa falta de información en nuestro modelo de datos. Los valores nulos dan la posibilidad de manejar en forma adecuada las siguientes situaciones.

- Cuando se crea una nueva tupla y el usuario no conoce los valores de todos los atributos en ese momento.
- Cuando se agrega un nuevo atributo a un relación ya existente.
- Cuando se desea agregar los valores de algún atributo. Por ejemplo, cuando queremos obtener el valor promedio de todos los valores en los tuplas, para ese atributo. En esta situación es deseable reconocer la presencia de valores nulos e ignorar esas tuplas para fines del cálculo.

Integridad de la Entidad

Ningún componente de la llave primaria puede tener valores nulos, debido a que por definición toda entidad debe distinguirse de cualquier otra, debe poseer alguna propiedad que la identifique de manera única. La llave primaria en una relación desempeña el papel de identificador único para una entidad. La presencia de valores nulos en ella equivaldría a decir que una entidad no puede ser completamente identificada y por lo tanto no puede distinguirse de otras.

Integridad Referencial

Dadas dos relaciones, si el dominio de la llave primaria de la relación r es igual al dominio de un atributo de la relación s, entonces el valor del atributo de cada tupla en la relación s deberá ser nulo o igual a alguno de los valores del dominio de la llave primaria de la relación r, o bien, matemáticamente hablando: Sea D_{PP} un dominio de una relación r_1 y sea r_2 una relación con un dominio D_1, siendo $D_1 = D_{PP}$. Entonces, para cualquier estado de la relación r_1, cada valor del dominio D_1 deberá ser nulo o igual a V donde V es el valor de la llave primaria de alguna tupla en la relación r_2 (r_1 y r_2 no necesariamente son distintas). Al dominio D_1 se le denomina llave foránea y se puede denotar D_{FK}. De hecho, la llave foránea es el mecanismo a través del cual podemos ligar o conectar dos o más entidades (en las estructuras de red o jerárquicas esto se hacía mediante apuntadores físicos).

Tecnologías y Herramientas de Bases de Datos Relacionales

Definición de Sistema Administrador de Bases de Datos

Software que controla la organización, almacenamiento, recuperación, seguridad, integridad y manejo de los datos en una base de datos haciendo uso de algún modelo de datos. Acepta pedidos de datos desde un programa de aplicación o cliente y le ordena al sistema operativo transferir los datos apropiados. Cuando se usa un Sistema Administrador de Bases de Datos (en inglés DBMS) los sistemas de información pueden ser cambiados más fácilmente a medida que cambien los requerimientos de la organización y nuevas categorías de datos pueden agregarse a la base de datos sin dañar el sistema existente.

Así, podemos decir que entre la base de datos física en sí (es decir, el almacenamiento real de los datos) y los usuarios del sistema existe un nivel de software, que a menudo recibe el nombre de Sistema de Administración de Bases de Datos o SABD. Este maneja todas las solicitudes de acceso a la base de datos formuladas por los usuarios, tomando en

Integridad Referencial

Dadas dos relaciones, si el dominio de la llave primaria de la relación r es igual al dominio de un atributo de la relación s, entonces el valor del atributo de cada tupla en la relación s deberá ser nulo o igual a alguno de los valores del dominio de la llave primaria de la relación r, o bien, matemáticamente hablando: Sea D_{PK} un dominio de una relación r_2 y sea r_1 una relación con un dominio D_i, siendo $D_i = D_{PK}$. Entonces, para cualquier estado de la relación r_1, cada valor del dominio D_i deberá ser nulo o igual a V, donde V es el valor de la llave primaria de alguna tupla en la relación r_2 (r_1 y r_2 no necesariamente son distintas). Al dominio D_i se le denomina llave foránea y se pude denotar D_{FK}. De hecho, la llave foránea es el mecanismo a través del cual podemos ligar o conectar dos o más entidades (en las estructuras de red o jerárquicas esto se hacía mediante apuntadores físicos).

TECNOLOGÍAS Y HERRAMIENTAS DE BASES DE DATOS RELACIONALES

Definición de Sistema Administrador de Bases de Datos

Software que controla la organización, almacenamiento, recuperación, seguridad, integridad y manejo de los datos en una base de datos haciendo uso de algún modelo de datos. Acepta pedidos de datos desde un programa de aplicación o cliente y le ordena al sistema operativo transferir los datos apropiados. Cuando se usa un Sistema Administrador de Bases de Datos (en ingles DBMS) los sistemas de información pueden ser cambiados más fácilmente a medida que cambien los requerimientos de la organización y nuevas categorías de datos pueden agregarse a la base de datos sin dañar el sistema existente.

Así, podemos decir que entre la base de datos física en sí (es decir, el almacenamiento real de los datos) y los usuarios del sistema existe un nivel de software, que a menudo recibe el nombre de Sistema de Administración de Bases de Datos o SABD. Este maneja todas las solicitudes de acceso a la base de datos formuladas por los usuarios, tomando en

consideración la seguridad de la información. Una función general del SABD, por tanto, es proteger a los usuarios de la base de datos contra los detalles a nivel de hardware. En otras palabras, el SABD ofrece una vista de la base de datos que está por encima del nivel de hardware y apoya las operaciones del usuario (tales como "obtenga el registro Empleado del empleado X") que se expresan en términos de esa vista de nivel superior.

El Sistema de Administración de Bases de Datos es el software que maneja todos los accesos a la base de datos. En términos conceptuales, lo que sucede es lo siguiente (ver figura 8):

1. Un usuario emite una solicitud de acceso, utilizando algún lenguaje de manipulación de datos específico;
2. El SABD intercepta la solicitud y la interpreta;
3. El SABD inspecciona por turno el esquema externo, la correspondencia externa-conceptual, el esquema conceptual, la correspondencia conceptual-interna y la definición de la estructura de almacenamiento, y
4. El SABD realiza las operaciones necesarias sobre la base de datos almacenada.

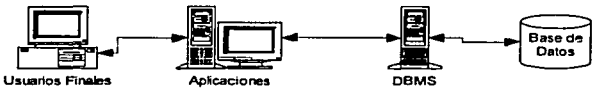


Figura 8. Esquematización del funcionamiento de un SABD

Por ejemplo, considérese lo que interviene en la recuperación de una ocurrencia de un registro externo específico. En general se necesitarán campos de varios registros conceptuales. Cada ocurrencia de un registro conceptual a su vez, puede requerir campos de varias otras. Por tanto al menos desde el punto de vista conceptual, el SABD debe recuperar todas las requeridas, construir las que sean necesarias y luego construir la requerida. En cada etapa pueden necesitarse conversiones de tipos de datos o de otra clase.

La descripción anterior presupone que el proceso completo es interpretativo, lo cual a menudo implica un desempeño bastante deficiente. En la práctica, por supuesto, algunas veces las solicitudes de acceso serán compiladas de antemano, evitándose así los costos de interpretación.

Los Sistemas de Administración de Bases de Datos están divididos en varios componentes de software, cada uno de los cuales tiene asignada una operación específica. Algunas de las funciones de los Sistemas de Administración de Bases de Datos son soportadas por el sistema operativo, sin embargo, éste provee únicamente servicios rudimentarios y el Sistemas de Administración de Bases de Datos debe ser construido a partir de esta consideración, así, su diseño debe tomar en cuenta la interfaz entre él y el sistema operativo.

COMPONENTES DEL SUBLENGUAJE EMPLEADO POR EL SABD

Lenguaje de Definición de Datos (Data Definition Language)

Permite la definición o descripción de los objetos de la base de datos. Puede usarse para crear, alterar o borrar relaciones (tablas), vistas, restricciones de integridad (por ejemplo, llaves primarias y llaves foráneas), tipos de datos, índices, reglas, valores por omisión, vistas, disparadores, procedimientos almacenados etc.

El SABD debe ser capaz de aceptar definiciones de datos (esquemas externos, el esquema conceptual, el esquema interno) en versión fuente y convertirlas en la versión objeto apropiada.

Manipulación de datos (DML)

Apoya el manejo o procesamiento de los objetos de la base de datos. Puede usarse para leer (consultar), modificar, borrar, o agregar tuplas (renglones) a las relaciones existentes. Una de las primeras funciones de los SABD es la de soportar un DML en el cual el usuario pueda formular comandos que permitan manipular los datos. Los DML se distinguen por sus

sublenguajes de recuperación subyacentes; se pueden distinguir dos tipos de DML, el procedural y el no procedural. La principal diferencia entre ambos es que en los primeros se tratan los registros individualmente, mientras que en los segundos se opera sobre un conjunto de registros.

El SABD debe ser capaz de atender las solicitudes del usuario para extraer, y poner al día, datos que ya existen en la base, o para agregar nuevos. Dicho de otro modo, el SABD debe incluir un componente procesador de lenguaje de manipulación de datos.

En general, las solicitudes en el DML pueden ser "planeadas o no planeadas": una solicitud planeada es aquella cuya necesidad se previó mucho tiempo antes de que tuviera que ejecutarse por primera vez; una solicitud no planeada es una consulta *ad hoc*, es decir, una solicitud cuya necesidad no se previó, sino que surgió de improviso.

Lenguaje de Control de Datos (DCL)

Permite la definición de los usuarios de la base de datos. Puede usarse para crear, alterar o eliminar permisos de acceso y manipulación a la base de datos por diferentes usuarios.

Diccionario de datos (DD)

Es una base de datos por propio derecho que contiene "datos acerca de datos" (es decir, descripciones de otros objetos del sistema, y no tan solo "datos en bruto"). En particular, todos los diversos esquemas (externo, conceptual e interno), se almacenan físicamente en el diccionario, tanto en forma fuente como en forma objeto. Un diccionario amplio incluirá también las referencias cruzadas que indican, por ejemplo que partes de datos utiliza cada programa, que informes necesita cada departamento, etc. e hecho, el diccionario puede integrarse a la base de datos que describe, y , por tanto, incluir su propia descripción. Debe ser posible consultar el diccionario de la misma manera que cualquier otra base de datos, de modo que, por ejemplo, el DBA[3] pueda describir con facilidad que programas tienen probabilidad de ser afectados por un cambio propuesto al sistema.

[3] Data Base Administrator, Administrador de la Base de Datos, responsable de la misma.

El diccionario de datos almacena información acerca de la estructura de la base de datos y la información de autorización. Entre los tipos de información que el sistema debe almacenar están:

- Los nombres de las relaciones.
- Los nombres de los atributos de cada relación.
- Los dominios de los atributos.
- Los nombres de las vistas definidas en la base de datos y la definición de esas vistas.
- Las restricciones de integridad de cada relación (por ejemplo, las restricciones de llave primaria o foránea, etc.).

Las principales funciones del Diccionario de Datos son las siguientes:

- Describe todos los elementos en el sistema (flujo de datos, almacenes de datos, procesos).
- Los elementos se centran en los datos y en la forma en que están estructurados.
- Comunica los mismos significados para todos los elementos del sistema.
- Documenta las características del sistema.
- Facilita el análisis de los detalles para evaluar las características y determinar cómo deben realizarse los cambios.
- Localiza errores y omisiones en el sistema

Además de esto es recomendable que en la mayoría de los sistemas se conserven los siguientes datos:

- Nombre de los usuarios autorizados.
- Información contable acerca de los usuarios.

En los sistemas que utilizan estructuras altamente sofisticadas para almacenar relaciones, pueden conservarse datos estadísticos y descriptivos acerca de las relaciones como por ejemplo:

- Número de tuplas de cada relación.
- Método de almacenamiento utilizado para cada relación (por ejemplo, agrupado o sin agrupar).

Es importante además almacenar la información de los índices de cada una de las relaciones:

- Nombre del índice.
- Nombre de la relación que se indiza.
- Atributos sobre los que está el índice.
- Tipo de índice.

HERRAMIENTAS PARA BASE DE DATOS RELACIONALES

Platinum Erwin

Una de las herramientas líder en el mercado, es una herramienta de diseño poderosa y fácil de usar. Su rica variedad de técnicas de diseño incrementa la productividad para implementar sistemas transaccionales. Ayuda a diseñar, generar y mantener aplicaciones de Bases de Datos. Desde el modelo lógico de la información de requerimientos y reglas de negocio que definen la Base de Datos, hasta la optimización de un modelo físico con las características específicas del Sistemas de Administración de Bases de Datos destino, permite visualizar la estructura propia, todos los elementos y permite optimizar el diseño de la Base de Datos. Genera automáticamente tablas y procedimientos almacenados y disparadores. Su tecnología de "comparación completa" permite el desarrollo iterativo de forma que el modelo se encuentra siempre sincronizado con la Base de Datos. Además de ello, puede integrarse con los ambientes de desarrollo líderes en el mercado, de forma que agiliza la creación de aplicaciones. Entre sus beneficios se encuentran:

- Asegura la consistencia, reutilización e integración de los datos de la organización brindando un medio de análisis y comunicación para la estructura de la Base de Datos.

- Mejora la productividad entre desarrolladores cuando los diseños de la Base de Datos se encuentran divididos, compartidos o son reutilizados.
- Su ambiente de uso gráfico facilita la vista de la estructura y su optimización.
- Conserva recursos y mejora la exactitud del diseño a través de la sincronización directa del modelo y la Base de Datos.

CAPÍTULO II

Capítulo II. Bases de Datos Orientadas a Objetos

Las Bases de Datos Relacionales aún no resuelven todos los problemas de información existentes en el mercado. Hay algunas estructuras de datos que no encuadran bien en las relaciones o que cuando son convertidas a relaciones, no permiten su consulta clara y libremente.

Un ejemplo de ello, son los BLOB[4], cuyo problema es la imposibilidad de realizar consultas acerca de su contenido, puesto que una Base de Datos Relacional no puede interpretarlo como lo haría una aplicación nativa, de forma que no pueden realizarse búsquedas sensitivas en estos datos.

Por lo que a representación léxica se refiere, los sistemas de bases de datos relacionales presentan serios problemas reconocidos por diversos autores. En general, las Bases de Datos Relacionales no fueron pensadas para almacenar información compleja, sino grandes cantidades de datos relativamente simples. Como ya hemos mencionado, los datos contenidos en una Base de Datos Relacional han de ser por definición atómicos. Esto es necesario para un correcto tratamiento de los mismos, pero por otra parte entorpece la visión de conjunto, esto es, dificulta el tratamiento "inteligente" de entidades complejas. Este gran inconveniente se hizo evidente una vez superada la fase de dificultades puramente técnicas. Cuando las Bases de Datos comenzaron a ser utilizadas para otras tareas que no fuesen el guardar datos correspondientes a una compañía de seguros o una entidad bancaria. Entonces se planteó la necesidad de mayor atención al diseño lógico de la Base de Datos en lugar de al nivel físico. Esta etapa se encuentra fuertemente influenciada por la investigación en la Inteligencia Artificial.

[4] BLOB Binary Large Objects, cadena de bytes interpretados por otra aplicación. Ver glosario.

Otro ejemplo claro de ello es el problema descrito metamórficamente como "Impedancia mal emparejada" o "*impedance mismatch*" en inglés. Este implica los obstáculos de almacenar en una Base de Datos Relacional la información de circuitos conteniendo otros circuitos de forma natural, debido a que sería imposible averiguar el contenido de uno de estos circuitos sin una aplicación que pudiese interpretar los componentes de cada circuito compuesto como se muestra en la Figura 9.

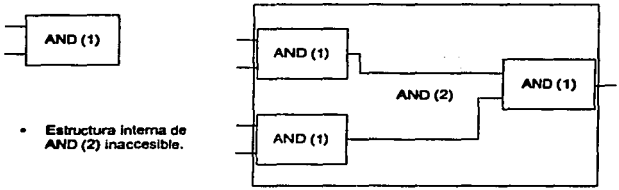


Figura 9. Ejemplo de "mismatch impedance" en Bases de Datos Relacionales.

El problema anterior significa que la base de datos y la aplicación nativa del BLOB no pueden auxiliarse en el proceso de consulta debido a que la aplicación es de tipo procedural – procesa un elemento a la vez - mientras la Bases de Datos es declarativa – procesa múltiples tuplas a la vez – y aunque ambas situaciones tienen sus conveniencias, el tratar de hacer trabajar juntos ambos paradigmas provoca esta "impedancia mal emparejada" o falta de compatibilidad.

La solución propuesta por las Bases de Datos Orientadas a Objetos es acercar lo más posible a dichas aplicaciones y la Base de Datos de forma que el elemento definido por la aplicación pueda ser almacenado en la Base de Datos sin ser desligado e integrando el lenguaje de programación de la aplicación al lenguaje de definición de datos de la Base de Datos.

Los Sistemas Administradores de Bases de Datos Orientadas a Objetos (SABDOO) nacieron en el inicio de los años 80 en respuesta al sentimiento general de que las Bases de Datos Relacionales eran inadecuadas para cierta clase de aplicaciones y han evolucionado hacia un área de investigación importante.

Las primeras áreas donde se emplearon ampliamente las Bases de Datos Orientadas a Objetos fueron CAD, CAE y CAM, aunque incrementalmente fueron extendiéndose a otras áreas como telecomunicaciones, cuidado de la salud, finanzas, multimedia y administración de la calidad.

El primer Sistema Administrador de Bases de Datos Orientado a Objetos fue G-base en 1986, luego GemStone en 1987 y otros, la característica principal de estos fue apoyar lenguajes utilizados en inteligencia artificial como lo es LISP. Aunque lamentablemente fueron Realizados como sistemas independientes con su propio lenguaje e instrucciones.

Luego surgió ONTOS que marcó un cambio radical ya que funcionaba en una plataforma común como C++ y Estaciones de trabajo UNIX.

Por último en los 90 salieron al mercado ITASCA y O2, SABDOO más avanzados con un DDL y DML orientados a objetos.

En otras palabras como todo producto tecnológico este sé ha ido mejorando en el tiempo, pero lo más importante es que a partir de 1986 hasta la fecha los SABDOO han evolucionado con una rapidez extraordinaria.

Aunque los SABDOO carezcan de un modelo común, debido a la falta de estandarización y los productos existentes no tengan el mismo rendimiento que los SABDR hoy por hoy se desea cambiar esta desventaja o defecto, para ello existen grupos estudio para estandarizar un modelo.

Este es el Grupo Manejador de Datos Objeto (ODMG por sus siglas en inglés), que representa a las Bases de Datos Orientadas a Objetos industriales y ha establecido el estándar equivalente al SQL de las Bases de Datos Relacionales.

Una de las principales ventajas que se ve en este modelo, al menos como una ventaja conceptual, es explotar todos los beneficios en el campo de la ingeniería de software, ya que es en dicho campo donde se le ha sacado mayor provecho a la orientación a objetos.

FUNDAMENTOS DE LAS BASES DE DATOS ORIENTADAS A OBJETOS

Modelo Matemático Subyacente

El modelo orientado a objetos no cuenta con un modelo matemático formal aceptado universalmente. Existen diversas propuestas de modelos formales para este paradigma; pero hasta la fecha no se ha publicado oficialmente ninguna de ellas, no obstante, es importante considerar que dichas propuestas están fundamentadas principalmente en la teoría de conjuntos y el cálculo de dominios, igual que el modelo relacional.

Por otro lado, están considerándose como fundamentos, los conceptos básicos de la Programación Orientada a Objetos, que son:

- Tipos objetos, que definen objetos particulares.
- Encapsulamiento, que enlaza o liga los datos (propiedades) con los códigos que los manipulan (métodos) dentro del objeto.
- Polimorfismo, la posibilidad de que cada objeto actúe de modo diferente en respuesta a determinadas acciones.

- Capacidad de extender el código, lo que permite a unidades ya compiladas ser utilizadas como base para la creación y uso de nuevos objetos que eran desconocidos hasta el momento de la compilación.

Conceptos Asociados a Bases de Datos Orientadas a Objetos

Estado

El estado de un objeto se modela a través de un conjunto de valores y variables que representan la abstracción de un objeto y sus características en el tiempo. Existen muchos términos para llamar esta representación del estado aunque las más comunes son propiedades o atributos.

Comportamiento

La modificación del comportamiento de un objeto puede provocar ya sea la ejecución de una acción o el cambio en el estado de un objeto. Este comportamiento se modela a través de un conjunto de funciones que llevan a cabo ciertas acciones o que modifican el estado del objeto. A estas funciones se les conoce como funciones miembro o métodos.

Métodos y Mensajes

Los objetos reciben, interpretan y responden a mensajes de otros objetos, lo cual marca una diferencia entre ellos y las variables y datos pasivos del paradigma tradicional.

El término mensajes se ha empleado para definir el medio de comunicación entre objetos. En la terminología tradicional, un mensaje es una llamada de un procedimiento a otro.

Un método es el código o definición de procedimiento que se invoca, es decir, es una especie de receta a seguir al recibir determinado mensaje. Esta pieza de código asociado con un objeto tiene el privilegio de acceso al estado del objeto. Verificar si un objeto tiene o no una determinada propiedad, la cual es derivada de un estado, puede ser más fácil a través de la invocación de un método escrito especialmente para ello que a través de la ejecución de una consulta sobre los atributos necesarios. Los procedimientos almacenados del modelo relacional, son los precursores de este concepto.

Los métodos y los mensajes ayudan a forzar la división del trabajo: un objeto envía un mensaje a otro, el objeto receptor efectúa un método en respuesta a dicho mensaje.

El pasar mensajes reduce el número de conexiones entre los componentes del sistema. Reducir el número de conexiones incrementa la modularidad de los componentes.

Los lenguajes de programación orientados a objetos requieren que toda la interacción con objetos se realice mediante el envío de mensajes, así un lenguaje para Bases de Datos Orientadas a Objetos debe incluir tanto el modelo de pasar el mensaje de objeto a objeto como el modelo de pasar el mensaje de conjunto en conjunto

Encapsulamiento

Este principio aplicado a los objetos de una base de datos exige la integración de la estructura y el comportamiento de los mismos en torno de una sola entidad: el objeto. Este principio permite ocultar los detalles de implementación (de los métodos aplicables al objeto) y asegurar la integridad de su parte estructural. Sin embargo, es limitativa desde el punto de vista de base de datos, ya que exige la especificación de métodos especializados a cada tipo de consulta, perdiendo con ello la flexibilidad tradicional ofrecida por un SABD.

Herencia

La herencia constituye la principal forma de reutilización del paradigma orientado a objetos. En términos generales, una subclase hereda todos los datos y métodos de la superclase a la que pertenece. La subclase puede adicionar datos y métodos a los que heredó de su superclase aunque también puede modificar los métodos heredados.

El mecanismo de la herencia es de gran utilidad en el modelado de una aplicación. La jerarquía de clases determina una trayectoria de herencia entre las superclases y sus subclases, permitiendo al diseñador reutilizar las definiciones de clases ya existentes, incorporando sus atributos y métodos. La jerarquía de composición (generada al permitir asignar como valor de un atributo el identificador de otro objeto) es la base de la definición recursiva de un objeto complejo en términos de otros objetos. Un objeto complejo es entonces una instancia de la jerarquía de composición de las clases.

Polimorfismo

Este concepto se refiere simplemente a que cada objeto reacciona de manera muy particular a un mensaje que se le envía, lo cual facilita el desarrollo y mantenimiento. El polimorfismo involucra el envío de mensajes a objetos de tipo desconocido.

Seguridad

Una Base de Datos Orientadas a Objetos evita el acceso a los datos en los objetos; esto mediante los métodos almacenados en ella. Es segura ya que no permite tener acceso a los objetos pues ello se tiene que realizar mediante los métodos previamente definidos para ello.

El Modelo Orientado a Objetos

Las Bases de Datos Orientadas a Objetos se pueden construir mediante alguno de los tres enfoques siguientes:

Primer Enfoque de Construcción de Bases de Datos Orientadas a Objetos.

Se puede utilizar el código actual altamente complejo de los sistemas de administración de Bases de Datos, de modo que una Base de Datos Orientadas a Objetos se implante más rápido sin tener que iniciar de cero. Las técnicas orientadas a objetos se pueden utilizar como medios para el diseño sencillo de sistemas complejos. Los sistemas se construyen a partir de componentes ya probados con un formato definido para las solicitudes de las operaciones del componente.

Segundo Enfoque de Construcción de Bases de Datos Orientadas a Objetos.

Considera a las Bases de Datos Orientadas a Objetos como una extensión de la tecnología de las Bases de Datos Relacionales, usando las herramientas y técnicas usadas en esta tecnología, así como la vasta experiencia que la gente tiene en ellas para construir un nuevo Sistema Administrador de Bases de Datos en el que se pueden añadir apuntadores a las tablas relacionales para ligarlas con objetos binarios de gran tamaño (BLOB) simulando un entorno Orientado a Objetos.

Tercer Enfoque de Construcción de Bases de Datos Orientadas a Objetos.

Se basa en la producción de una nueva arquitectura optimizada, que cumple las necesidades y requerimientos de la tecnología orientada a objetos. Las compañías como Versant, Objectivity, Itasca, entre otras, utilizan este enfoque y afirman que la tecnología relacional es un subconjunto de una capacidad más general. Además que las Bases de Datos Orientadas a Objetos no relacionales

son aproximadamente dos veces más rápidas que las Bases de Datos Relacionales al almacenar y recuperar datos de tipo complejo. Por lo tanto son esenciales en aplicaciones como CAD y permitirían que un depósito CASE fuera una facilidad de tiempo real en vez de funcionar por lotes.

COMPONENTES DEL MODELO ORIENTADO A OBJETOS.

Clases

Una clase es la definición abstracta de un grupo de objetos con datos y métodos comunes. Las clases son plantillas que actúan agrupando objetos relacionados entre sí con propiedades y comportamiento similares. Cada clase define las propiedades y métodos que comparten todas sus instancias u objetos derivados, pero cada objeto tiene sus propios valores únicos.

Equivalen al concepto de tipos de datos en los lenguajes de programación, con la diferencia de que los usuarios pueden definir siempre nuevos tipos incluso a partir de tipos existentes.

Además de lo anterior, es posible definir subclases, o bien, clases a partir de otras clases (superclases), lo que indica que puede existir una jerarquía entre clases: conforme descendo la jerarquía, es posible definir objetos más especializados con las subclases. Las superclases expresan características comunes entre clases.

Objetos

Los objetos son las unidades estructurales primarias de la programación orientada a objetos. Se distinguen de los procedimientos porque tienen un ciclo de vida. Es posible crearlos y destruirlos. Siempre y cuando un objeto esté vivo y activo, se puede realizar el acceso a cualquiera de sus elementos públicos. Una vez destruido, se vuelve irrecuperable y es imposible comunicarse con él.

Un objeto es la abstracción de una entidad tangible del mundo real o de un concepto teórico. Un objeto tiene estado, comportamiento e identidad. Típicamente se definen como instancia de una clase, puesto que se usa una clase para crear un objeto.

Identidad de los objetos

En todo sistema Orientado a Objetos, los objetos son reconocidos, diferenciados y referenciados gracias a su identificador (nombre único). Este hecho permite su uso como valores de atributos de otros objetos, ofreciendo gran riqueza en la construcción de estructuras de datos de gran complejidad, que representan objetos del mundo real de manera directa. La noción de identidad ha sido también utilizada como base en la implementación de extensiones para la persistencia de los objetos en algunos lenguajes de programación Orientada a Objetos.

Relación

Representa la asociación o correspondencia entre dos entidades o de una entidad consigo misma; es binaria, es decir, puede leerse en dos direcciones o sentidos. Asocia una entidad de un conjunto a una o varias entidades de otro. En un DER las relaciones se representan con líneas conectando las entidades relacionadas; normalmente un verbo corresponde a la relación.

En el modelo orientado a objetos existen diversos tipos de relaciones, las cuales se listan a continuación:

- Herencia ("Inheritance)"
- Asociación ("Association")
- Agregación ("Agregation")
- Origen ("Parent" o "Inverse")

Relación de Herencia.

Indica cuando un objeto es del tipo de otro objeto. Por ejemplo, supongamos que tenemos la clase de objeto insecto, y la clase mariposa, la cual tiene una relación de herencia respecto a la clase insecto; sabemos que los insectos tienen determinadas características como exoesqueleto o seis patas, estas mismas características serán heredadas por los objetos de la clase mariposa, de forma que aparte de sus características propias, tendrán seis patas y un exoesqueleto.

Relación de Asociación.

Indica cuando un objeto tiene una conexión con otro objeto. Es decir, supongamos que tenemos la clase mariposa, y además la clase flor, ya que la fuente de alimento de la mariposa es la flor, entonces decimos que la mariposa y la flor tienen una relación de asociación: la mariposa se alimenta de la flor.

Relación de Agregación.

Indica cuando un objeto está hecho o formado de otros objetos. Para ejemplificar este tipo de relación, debemos considerar ahora que tenemos un objeto de la clase mariposa, el cual consta de alas, antenas, cabeza, patas, etc. Cada uno de estos objetos conforma a una mariposa de modo que tenemos una relación de agregación de estos objetos para constituir una mariposa. En esta relación tenemos la vista usual: una mariposa "sabe" de que partes está formada.

Relación de Origen.

Indica cuando un objeto es parte de otro, suele ser confundida con la relación de agregación. Consideremos esta ocasión el ala derecha de la mariposa, ésta es parte de un objeto mariposa, de modo que tiene una relación de origen con el objeto mariposa. En este caso, el ala "sabe" que es parte de una mariposa,

aunque no forzosamente "sepa" que está formada entre otras cosas por un ala derecha.

Diagrama de Clases[5]

La técnica del diagrama de clase se ha vuelto medular en los métodos orientados a objetos. Describe los tipos de objetos que hay en el sistema y las diversas clases de relaciones estáticas que existen entre ellos. Hay dos tipos principales de relaciones estáticas:

- Asociaciones
- Subtipos

Los diagramas de clase también muestran los atributos y operaciones de una clase y las restricciones a las que se ven sujetos, según la forma en que se conecten.

Dependiendo del detalle del diagrama, la notación de un atributo puede mostrar el nombre, tipo y valor predeterminado de un atributo.

Las operaciones son los procesos que una clase sabe llevar a cabo. Corresponden a los métodos sobre una clase

Los diagramas de clase son la columna vertebral de casi todos los métodos de desarrollo orientados a objetos, por lo cual se emplean todo el tiempo, y es lógico que no se usará otro tipo de desarrollo al emplear una base de datos orientada a objetos.

[5] **FOWLER** Martin y **KENDALL** Scott. *UML Gota a gota*

Por lo que respecta a la notación empleada en ellos, al igual que cuando hablamos de los fundamentos del modelo Orientado a Objetos, no se cuenta con un estándar oficial, sin embargo, el Lenguaje unificado de Modelado (UML por sus siglas en inglés) proporciona un buen acercamiento a dicho estándar, estableciendo la siguiente notación:

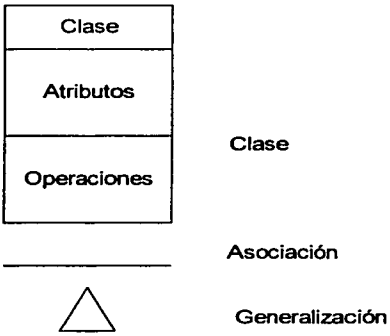


Figura 10. Notación UML del Diagrama de Clases en Bases de Datos Orientadas a Objetos.

Es importante considerar que éstos son solo los símbolos básicos, pero es posible ir indicando cada tipo de asociación de las listadas como parte de los conceptos asociados a las Base de Datos Orientadas a Objetos, así como diversos tipos de restricciones correspondientes a las de cardinalidad en un diagrama entidad-relación.

Normalización en las Bases de Datos Orientadas a objetos.

Las Bases de Datos Orientadas a Objetos poseen lenguajes de acceso a datos esencialmente procedurales (en un lenguaje declarativo, el programador da la especificación matemática de lo que se debe calcular, dejando el como se calcula al compilador), debido a que la consulta se almacena directamente mediante la implementación de apuntadores para unir las clases y los objetos que contienen.

Otra característica importante de estas Bases de Datos es el hecho de que no deben estar necesariamente normalizadas. Además de ello, son eficientes para realizar consultas orientadas a conjuntos de datos, para ello acceden rápidamente un objeto a través de apuntadores.

Las Bases de Datos Orientadas a Objetos almacenan y manipulan información que puede ser representada por objetos, proporcionando una estructura flexible con acceso ágil, rápido y con gran capacidad de modificación.

Adicionalmente, estas Bases de Datos permiten definir arbitrariamente tipos de datos complejos de la misma forma que su contraparte en lenguajes de programación orientada a objetos, posiblemente anidados en jerarquías del tipo "es un..." y "tiene un...". Conjuntos, listas y otros contenedores son usados entre otras cosas, para representar el resultado de una consulta que devuelve diversos objetos.

El Estándar ODMG93

Las "Reglas de Codd" de Bases de Datos Orientadas a Objetos

De forma similar al estándar ANSI SQL, la ODMG, formada en 1991, es un consorcio de un grupo de vendedores expertos cuyo objetivo es producir un estándar abierto para los Sistemas Manejadores de Bases de Datos Orientadas a Objetos. Este estándar tiene un mandato parecido a OOPEA.

Pese a que ODMG no es derivada de la OMG, se asoció con ella, y el estándar ODMG93 fue adoptado como una interfaz estándar para un servidor persistente en el que se encuentran el Lenguaje de definición de objetos Object Definition Language o ODL una extensión de IDL de OOPEA muy similar a C++, el Lenguaje de consulta de objetos Object Query Language o OQL con una sintaxis basada en SQL y el Lenguaje de manipulación de objetos Object Manipulation Language o OML que es un lenguaje ligado a C++ que provee acceso a los

Otra característica importante de estas Bases de Datos es el hecho de que no deben estar necesariamente normalizadas. Además de ello, son eficientes para realizar consultas orientadas a conjuntos de datos, para ello acceden rápidamente un objeto a través de apuntadores.

Las Bases de Datos Orientadas a Objetos almacenan y manipulan información que puede ser representada por objetos, proporcionando una estructura flexible con acceso ágil, rápido y con gran capacidad de modificación.

Adicionalmente, estas Bases de Datos permiten definir arbitrariamente tipos de datos complejos de la misma forma que su contraparte en lenguajes de programación orientada a objetos, posiblemente anidados en jerarquías del tipo "es un..." y "tiene un...". Conjuntos, listas y otros contenedores son usados entre otras cosas, para representar el resultado de una consulta que devuelve diversos objetos.

El Estándar ODMG93

Las "Reglas de Codd" de Bases de Datos Orientadas a Objetos

De forma similar al estándar ANSI SQL, la ODMG, formada en 1991, es un consorcio de un grupo de vendedores expertos cuyo objetivo es producir un estándar abierto para los Sistemas Manejadores de Bases de Datos Orientadas a Objetos. Este estándar tiene un marcado parecido a CORBA.

Pese a que ODMG no es derivada de la OMG, se asoció con ella, y el estándar ODMG93 fue aceptado como una interfaz estándar para un servicio persistente en el que se encuentran el Lenguaje de definición de objetos (Object Definition Language u ODL, una extensión del IDL de CORBA muy similar a C++), el Lenguaje de consulta de objetos (Object Query Language u OQL, con una sintaxis basada en SQL) y el Lenguaje de manipulación de objetos (Object Manipulation Language u OML, que es un lenguaje ligado a C++ que provee acceso a las

funciones de la base de datos y la persistencia de los objetos definidos con el ODL, y entre otras cosas define un contenedor de clases). Este estándar ha sido formalizado en el libro The Object Databse Standard: ODMG93, Release 1.1 de R. G. G. Catell, pero hasta 1995 no existían implementaciones completas de este estándar.

De forma similar a las reglas establecidas por el Dr. E. Codd, en el modelo orientado a objetos existen tres documentos hasta hoy día, donde se señalan importantes datos de su definición.

El primer manifiesto, publicado por Atkinston en 1989, indica diversas reglas, de forma que si un Sistema Administrador de Bases de Datos las cumple, puede proclamarse Orientado a Objetos.

Primer Manifiesto para los Sistemas Manejadores de BDOO

El Primer Manifiesto de las Bases de Datos Orientadas a Objetos, intenta definir un sistema de BDOO y describe las principales características que deben cumplir. Conforme a dicho documento, y a semejanza con las Bases de Datos Relacionales, existen algunas Reglas de Oro que debe cumplir un Sistema Administrador de Bases de Datos Orientado a Objetos para poder ser considerado como tal. Según ello, un sistema de BDOO debe satisfacer dos criterios:

1. Debe tener un Sistema Administrador de Bases de Datos, es decir, debe proporcionar:
   - Persistencia
   - Administrador de almacenamiento secundario
   - Concurrencia
   - Recuperación
   - Facilidad de Consulta

2. Debe ser un sistema Orientado a Objetos, lo que significa que cumple con:
   - Objetos Complejos
   - Identidad del objeto
   - Encapsulamiento
   - Tipos ó Clases
   - Sobre carga con combinación retrasada
   - Extensibilidad
   - Completez Computacional

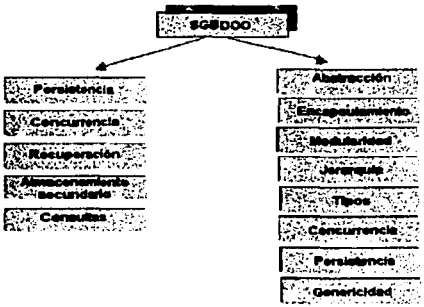


Figura 11. Reglas de Oro de las Bases de Datos Orientadas a Objetos

Estas características han sido separadas en 3 grupos:

3. Obligatorias o Reglas de Oro. Son las que el Sistema Administrador de Bases de Datos debe satisfacer para proclamarse como un Sistema Administrador de Bases de Datos Orientado a Objetos y éstas son:
   - Objetos complejos
   - Identidad de objetos
   - Encapsulamiento
   - Tipos ó Clases
   - Sobre paso combinado con unión retardada

- Extensibilidad
- Completez Computacional
- Persistencia: se refiere no sólo a la conservación del estado de un objeto, si no también a la conservación de la clase, que debe trascender a cualquier programa individual, de forma que todos los programas interpreten de la misma manera el estado almacenado.
- Administrador de almacenamiento secundario
- Concurrencia: Permite a varios usuarios tener acceso a una Base de Datos al mismo tiempo, controlando la interacción entre transacciones concurrentes para evitar que se destruya la consistencia de la base de datos.
- Recuperación: Al intentar realizar una transacción, si ésta no puede completarse por fallo de hardware o fallo de software, el sistema debe poder regresar al estado coherente anterior al inicio de dicha transacción
- Facilidad de realizar consultas ad hoc: se debe permitir realizar consultas personalizadas conforme las necesidades del usuario con la misión de proporcionar la información solicitada por el usuario de forma correcta y rápida.

4. Opcionales. Son las que pueden ser añadidas para hacer el sistema mejor pero que no son requeridas para calificar como Orientado a Objetos al Sistema Administrador de Bases de Datos; éstas son:
   - Herencia múltiple: Tienen características de padres diferentes y proporcionan mecanismos para saber cual es la más conveniente entre dos o más opciones.
   - Verificación de tipos e inferencia
   - Distribución: Que se puede tener parte de una BD en un servidor y otra parte en otro.
   - Diseño de transacciones y versiones

5. Abiertas. Son los puntos donde el diseñador puede hacer un número de operaciones variadas. Es como si fuera una especialización con cierta marca de software y son:
   - ❖ Paradigma de la programación
   - ❖ Representación del sistema ó el tipo de sistema: Forma en como se presentan los esquemas.
   - ❖ Uniformidad del sistema.

TECNOLOGÍAS ORIENTADAS A OBJETOS

Utilizan los mismos modelos conceptuales para el análisis, diseño y construcción de la Base de Datos y dan un paso más hacia la unificación, el modelo conceptual de la base de datos Orientado a Objetos es igual al del resto del paradigma Orientado a Objetos, en lugar de utilizar tablas por relación independientes como opera el modelo relacional.

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

Las primeras Bases de Datos Orientadas a Objetos se diseñaron como una extensión de los lenguajes de programación como Smalltalk ó C++. El LMD (Lenguaje de Manipulación de Datos; también conocido como DML) y el LDD (Lenguaje de Definición de Datos; también conocido como DDL) construían un lenguaje Orientado a Objetos común.

El diseño de las Bases de Datos Orientadas a Objetos actuales debe aprovechar al máximo el CASE e incorporar métodos creados con cualquier técnica poderosa, incluyendo enunciados declarativos, generadores de código e inferencias con base en reglas.

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

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

Transacciones compartidas.- Las transacciones compartidas soportan grupos de usuarios en estaciones de trabajo, los cuales desean coordinar sus esfuerzos en tiempo real, los usuarios pueden compartir los resultados intermedios de una base de datos y permite que varias personas intervengan en una sola transacción

Definición de Sistemas Administradores de Bases de Datos Orientadas a Objetos

Tienen la característica común de contar con clases de colecciones, estructuras de datos indizadas como arreglos, y no indizadas como diccionarios, conjuntos, bolsas, colecciones ordenadas, facilitando la construcción de objetos complejos con algoritmos de búsqueda muy eficientes, y de manera transparente.

Los SABDOO, almacenan objetos no sólo datos. Objetos que representan entidades u objetos del mundo real que está siendo modelado en la aplicación, objetos en el sentido de combinaciones encapsuladas de estructuras de datos (atributos, propiedades) y procedimientos asociados (métodos) que describen su comportamiento. Este mapeo uno a uno reduce la distancia semántica entre el mundo real y el modelo utilizado para representarlo dentro de la computadora. Más aún, asociado a un estilo de programación Orientado a Objetos, el SABDOO reduce la diferencia semántica entre el programa y la BD que lo soporta, ofreciendo al usuario la posibilidad de manipular objetos con estructuras de datos y operaciones definidas por él mismo, de manera flexible

DEFINICIÓN DEL SISTEMA ADMINISTRADOR DE BASES DE DATOS ORIENTADAS A OBJETOS

La mayor limitación de las bases de datos orientadas a objetos es la carencia de un estándar. ODMG-93 es un punto de partida importante para ello

Adopta una arquitectura que consta de un sistema de gestión que soporta un lenguaje de bases de datos orientado a objetos, con una sintaxis similar a un lenguaje de programación también orientado a objetos.

El lenguaje de bases de datos es especificado mediante un lenguaje de definición de datos (ODL), un lenguaje de manipulación de datos (OML), y un lenguaje de consulta (OQL), siendo todos ellos portables a otros sistemas con el fin de conseguir la portabilidad de la aplicación completa.

ODMG-93 trata de definir un Sistema de Administración de Bases de Datos Orientadas a Objetos que integre tanto las capacidades de una base de datos como las capacidades de los lenguajes de programación del paradigma Orientado a Objetos, de forma que los objetos dentro de la base de datos aparezcan como objetos en el lenguaje de programación, eliminando de esta manera la falta de correspondencia existente hoy en día entre estos sistemas y los tipos de datos definidos en ellos. El Sistema Administrador de Bases de Datos Orientado a Objetos extiende el lenguaje con persistencia, concurrencia, recuperación de datos, consultas asociativas, etc.

COMPONENTES DEL SUBLENGUAJE EMPLEADO POR EL SISTEMA ADMINISTRADOR DE BASES DE DATOS ORIENTADAS A OBJETOS

Lenguaje ODL

El lenguaje de definición de datos (ODL) en un Sistema Administrador de Bases de Datos Orientado a Objetos es empleado para facilitar la portabilidad de los esquemas de las bases de datos. Este ODL no es un lenguaje de programación completo, define las propiedades y los prototipos de las operaciones de los tipos, pero no los métodos que implementan esas operaciones.

El ODL intenta definir tipos que puedan implementarse en diversos lenguajes de programación; no está por tanto ligado a la sintaxis concreta de un lenguaje de programación particular. De esta forma un esquema especificado en ODL puede ser soportado por cualquier Sistema Administrador de Bases de Datos Orientado a Objetos que sea compatible con el estándar ODMG-93.

La sintaxis de ODL es una extensión de la del IDL (Interface Definition Language) desarrollado por la OMG como parte de CORBA.

Lenguaje OML

El lenguaje de manipulación es empleado para la elaboración de programas que permitan crear, modificar y borrar datos que constituyen la base de datos.

ODMG-93 sugiere que este lenguaje sea la extensión de un lenguaje de programación, de forma que se puede realizar la creación, borrado, modificación e identificación de un objeto entre otras operaciones sobre la base de datos.

Lenguaje OQL

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

- No es computacionalmente completo. Sin embargo, las consultas pueden invocar métodos, e inversamente los métodos escritos en cualquier lenguaje de programación pueden incluir consultas.
- Tiene una sintaxis abstracta.
- Su semántica formal puede definirse fácilmente.
- Proporciona un acceso declarativo a los objetos.
- Se basa en el modelo de objetos de ODMG-93.
- Tiene una sintaxis concreta.
- Puede optimizarse fácilmente.
- No proporciona operadores explícitos para la modificación, se basa en las operaciones definidas sobre los objetos para ese fin.
- Proporciona instrucciones primitivas de alto nivel para tratar con conjuntos de objetos, pero no restringe su utilización con otros constructores de colecciones.

Existen dos posibilidades para asociar un sublenguaje de consulta a un lenguaje de programación:

- Fuerte: consiste en una extensión de la gramática del lenguaje asociado.
- Débilmente: las funciones *query* tienen unos argumentos *String* que contienen las preguntas.

HERRAMIENTAS Y PRODUCTOS PARA BASES DE DATOS ORIENTADAS A OBJETOS

Lenguaje Unificado de Modelado (Unified Modeling Language - UML)

Existe otra alternativa de modelado de Bases de Datos: el UML del Object Managemente Group. Su desarrollo es impulsado por Rational Software Corporation basado en el Modelo Unificado para objetos de software, y el llamado Modelado de Componentes Empresariales (*Enterprise Component Modeling - ECM*), el cuál es una compleja propuesta que incluye la conceptualización y requerimientos de análisis que cubren el modelado conceptual con su respectivo mapeo en clases, componentes y distribución en el sistema y fases detalladas de diseño. Las clases y métodos de UML, sin embargo, son muy equivalentes a los tipos y métodos en un Sistema de Administración de Bases de Datos Objeto - Relacionales.

Con el advenimiento de sistemas cada vez más complejos, la importancia de encontrar una forma visual de representarlos se incrementó. El UML fue desarrollado por Grady Booch, Jim Rumbaugh e Ivar Jacobson como respuesta a esta necesidad, en el intento de crear un sistema simple para modelar y documentar los sistemas de información y los procesos de negocio, con la filosofía del análisis y diseño Orientado a Objetos. Para hacer sistemas con éxito, es esencial un modelo y por ello es importante comunicar el plan completo al equipo de desarrollo, y ahí es donde apoya UML. Como cualquier otro lenguaje, UML tiene sus propios elementos y guías de uso para ser empleadas en el modelado. Los principales objetivos de UML son:

- Proporcionar a los usuarios un lenguaje visual de modelado fácil de leer, de forma que puedan desarrollar e intercambiar modelos de forma significativamente sencilla.

- Brindar mecanismos de especialización para extender los conceptos básicos.
- Ser independiente de lenguajes de programación y procesos de desarrollo específicos.
- Dar bases formales para entender el lenguaje de modelado.
- Impulsar el crecimiento del mercado de herramientas Orientadas a Objetos.
- Soportar conceptos de desarrollo de alto nivel como la colaboración, los marcos de trabajo (*frameworks*), patrones y componentes.
- Integrar las mejores prácticas.

El UML no es la respuesta a todas las necesidades de modelado, pues pueden encontrarse aun reglas de negocio relacionadas con datos que no pueden ser representadas por este modelo. Sin embargo, brinda mayor flexibilidad y permite representar un porcentaje mayor de éstas reglas.

UML no hace el diseño de las Bases de Datos Orientadas a Objetos de forma mágica como el modelo Entidad-Relación no hace diseños en Tercera Forma Normal tampoco, pues es tan fácil hacer malos diseños con o sin UML. Sin embargo, representa un paso adelante en la notación de modelado. No sólo nos permite representar mas reglas de negocio, si no que además favorece la visión de objetos en el diseño.

Una de las herramientas de UML para Bases de Datos en el mercado en es E/R Studio de Embarcadero, que se integra fácilmente con herramientas de modelado UML para el resto del ciclo de desarrollo de sistemas conforme al RUP.

CAPÍTULO III

CAPÍTULO III. BASES DE DATOS OBJETO RELACIONALES

LAS BASES DE DATOS OBJETO RELACIONALES

Hace algunos años parecía que los días de las Bases de Datos Relacionales estaban contados. La migración a las Bases de Datos Orientadas a Objetos parecía inevitable pues todos comenzaban a aceptar el paradigma Orientado a Objetos como la nueva forma de desarrollar aplicaciones.

La gran cantidad de información en las empresas hoy en día, no es solamente de los tipos de datos almacenados tradicionalmente en las bases de datos relacionales. Suele ser almacenada en archivos, hojas de cálculo o incluso, en forma no digital. Por ahora, las Bases de Datos relacionales han sido empleadas para automatizar la mayoría de las tareas de oficina, sin embargo, para encontrar otras tecnologías de información para contar con ventajas competitivas, las organizaciones están considerando Internet y las intranets, así como un conjunto más vasto de tipos de datos. Para mantenerse acordes con las necesidades de sus clientes, casi todos los vendedores de Sistemas de Administración de Bases de Datos Relacionales están extendiendo las capacidades de sus líneas de productos para soportar aplicaciones habilitadas para Internet y los tipos de datos multimedia encontrados a diario en la web la cual da acceso global a clientes universales.

Estamos en un periodo de intensos cambios e innovación respecto a las Bases de Datos y los productos relacionados con ellas. El año de 1996 terminó con una amplia gama de anuncios de productos de los vendedores de Bases de Datos que implicaban el soporte extendido para tipos de datos adicionales o complejos, tradicionalmente no considerados en los productos relacionales puestos a disposición por los vendedores.

Muchos de los recientes anuncios de productos, usan el término de "universales" en los nombres de sus productos o mensajes de mercadotecnia. Algunos están orientados a la arquitectura de *middleware* reflejando la dirección futura en el desarrollo de sus productos. El principal punto es la promoción de una nueva versión "extendida" del modelo relacional llamada "Modelo Objeto Relacional" a través de la cuál las Bases de Datos Relacionales se mantendrán en el mercado en plena competencia con las Bases de Datos Orientadas a Objetos.

SISTEMAS MANEJADORES DE BASES DE DATOS POR VENDEDOR			
VENDEDOR	PRODUCTO		
	SISTEMAS DE ADMINISTRACIÓN DE BASES DE DATOS RELACIONALES	SISTEMAS DE ADMINISTRACIÓN DE BASES DE DATOS OBJETO RELACIONALES	SISTEMAS DE ADMINISTRACIÓN DE BASES DE DATOS ORIENTADAS A OBJETOS
ORACLE	ORACLE 7.X	ORACLE 8.X, ORACLE 9.X	
SYBASE	SYSTEM 10/11		
INFORMIX	DYNAMIC SERVER	UNIVERSAL SERVER (ILUSTRA)	
IBM	DB/2	UNIVERSAL DATABSE (DB/2 EXTENDERS)	
UNISQL		UNISQL/X	
UNISYS		OSMOS	
COMPUTER ASSOCIATES	OPENINGRES		JASMINE
GEMSTONE			GEMSTONE
O2			O2
OBJECT DESIGN			OBJECT STORE
OBJECTIVITY			OBJECTIVITY/DB
VERSANT			VERSANT ODBMS

Michael Stonbraker ha escrito un libro llamado "Objet-Relational DBMSs, The Next Great Wave", en el cual se discute ampliamente la definición de las Bases de Datos Objeto Relacionales a través del uso de Informix.

Las BDOR emplean un modelo de datos que "agrega" características de la Orientación a Objetos a la BD. Toda la información persistente está aun en las tablas, pero algunas de las entradas pueden tener una estructura variada o tipos de datos abstractos. El soporte a estos tipos de datos es atractivo porque las funciones y operaciones asociadas con ellos pueden ser usadas para indizar, almacenar y recuperar registros basados en su contenido (i.e. multimedios). Estas ventajas facilitan estructuras para modelos más complejos.

Este tipo de Bases de Datos soporta un lenguaje SQL extendido, algunas veces llamado Object SQL que generalmente incluye consultas que devuelven objetos anidados, atributos que contienen conjuntos de valores, inclusión de métodos y funciones en los predicados de búsqueda y consultas incluyendo tipos de datos abstractos. El modelo continúa siendo relacional pues el almacenamiento sigue realizándose en tablas de columnas y renglones, además de que SQL, con las respectivas extensiones, continúa siendo el lenguaje para definición, manipulación y consulta. El origen y resultado de las consultas siguen siendo tablas o tuplas, sin embargo, la nueva versión de SQL generalizará las colecciones de objetos, incorporando OQL[6] o limitándolo para realizar consultas *ad hoc* de sólo lectura. Lo que se pretende es la posibilidad de realizar búsquedas en el contenido absolutamente de toda la base de datos y los nuevos tipos de datos incluidos.

Date y Hugh Darwen, observando de forma preocupada lo que la industria había estado haciendo los últimos años, finalmente, comenzaron a escribir el "Tercer Manifiesto", que indicara lo que pensaban debía ser en adelante la industria de las Bases de Datos. La mayor parte de este documento, discute como casar la tecnología de objetos y la relacional. Este manifiesto siguió a dos importantes documentos, el primero de ellos, llamado "El manifiesto de los sistemas de Bases de Datos orientadas a objetos" escrito por Malcom Atkinston y otros autores; y el segundo, el manifiesto de la tercera generación de Sistemas de Bases de Datos de Mike Stonebraker entre otros.

[6] El cual hemos tratado más a detalle en el tercer capítulo.

Si Date decidió escribir este documento, se debe a que consideró que el Primer Manifiesto ignoraba el modelo relacional por completo, lo cual considera un grave error. El Segundo Manifiesto, ciertamente considera que no se debe descartar el modelo relacional, pero considera que eso significa usar SQL, lo cuál es un error.

Una sección del Tercer Manifiesto llamada "Volver al futuro" dice que la mejor manera de hacer algo correctamente, es volviendo a las raíces y tomar lo mejor de ellas. Debemos hacer esto con el modelo relacional. El documento habla sobre prescripciones y proscripciones así como sugerencias importantes para las BDOR. Este documento ha pasado por muchas versiones y foros internacionales de los que se han recibido comentarios que han ido incorporándose. Es este documento lo que se considera lo más cercano a la definición del modelo objeto-relacional.

FUNDAMENTOS DE LAS BASES DE DATOS OBJETO RELACIONALES.

Antes de responder técnicamente en que forma soporta el modelo relacional los conceptos que llamamos orientados a objetos, se necesita tener claro lo que significa orientado a objetos. Aunque existen la programación, el análisis, el diseño, interfaces y BDOO, no necesariamente se encuentra cosas comunes entre estas disciplinas, lo que significa que cualquier atributo positivo o negativo acerca de ellas, no aplica sin embargo, al resto de las disciplinas de este paradigma.

La cuestión principal en el campo de estas Bases de Datos es como integrar sus buenas ideas en el modelo relacional. Lo maravilloso de este asunto, es que no es necesario hacer nada al modelo relacional para lograr este objetivo[7]. Es tan sólido y robusto, que el Tercer Manifiesto señala: *"el modelo relacional no necesita extensiones, correcciones, inclusiones ni cualquier otra modificación"* para aplicar las mejores ventajas del modelo OO. Es decir, no necesitamos destruir el modelo relacional para obtener los beneficios aportados por el paradigma Orientado a Objetos.

[7] **DATE**, Chris J. *Relational Database Writtings 1991-1994*

El punto clave al tratar de unir estas tecnologías es encontrar el análogo de las clases del modelo Orientado a Objetos en el modelo relacional. La respuesta está en el dominio o tipo de dato. La noción clave que subraya el manifiesto es la ecuación: dominio = clase de objeto, el cuál (o una clase objeto), es un tipo de dato que está encapsulado, lo que significa que la única forma de operar los valores de ese tipo es a través de los operadores definidos para ese tipo. No se puede observar realmente la forma en que el dato es representado, pues esto no es relevante, sólo sabemos que existen ciertas funciones que pueden realizarse. Éste puede ser lo mismo un tipo de dato primitivo definido por el sistema o bien un tipo de dato definido por el usuario, y sus valores pueden ser arbitrariamente datos complejos.

La razón por la que el Tercer Manifiesto considera las mejores ideas del paradigma Orientado a Objetos como ideas ortogonales al modelo relacional es que de ninguna forma el modelo relacional prescribe los tipos de datos que puede tenerse. Este simplemente tiene un mecanismo que permite definir tipos de datos.

Las clases objeto y los dominios son realmente la misma cosa. Entonces, un sistema relacional que soporte dominios propiamente es capaz de hacer todas las cosas que los defensores del paradigma de objetos proclaman que los sistemas orientados a objetos pueden hacer y los sistemas relacionales no. Es posible tener sistemas relacionales para CAD/CAM, automatización de oficinas y diagnóstico médico manteniéndonos en el marco relacional. Lo que es necesario es que los vendedores soporten el modelo relacional correctamente, incluyendo la extensión de dominios que realizados correctamente permiten agregar tipos de datos de complejidad arbitraria– la cual no es realmente una extensión del todo –. Los valores en los renglones y las columnas pueden ser entonces todo lo que deseemos. Pueden ser simples enteros, cadenas, arreglos, libros, ilustraciones de ingeniería, videos, cualquier cosa que se requiera tanto como uno defina los tipos de datos necesarios.

De hecho, puede considerarse que el surgimiento de las Bases de Datos Orientadas a Objetos es el incorrecto uso de los vendedores del modelo relacional. Además, es necesario ser claros acerca de la diferencia entre el tipo y la representación del tipo. Si se tiene un sistema de Bases de Datos que soporte correctamente los dominios definidos por el usuario (por ejemplo, definir un dominio de número de empleado).

La representación puede ser un carácter cadena, pero esto no obliga a que sólo se puedan realizar operaciones para cadenas sobre los números de empleado. Es posible realizar todas las operaciones definidas para el dominio número de empleado definidas previamente, es decir, si se requieren construcciones de objetos en un modelo relacional, es posible hacerlas, únicamente debemos de considerar que el diseño se vuelve mucho más complicado, más no por ello imposible. Incluso pueden tenerse tablas dentro de tablas la única restricción es no intentar ver en ambas tablas al mismo tiempo, pues estaríamos hablando de dos niveles distintos de abstracción. El hecho de que según la Primera Forma Normal debe haber valores atómicos en una tabla, no es violado al guardar un arreglo o un tipo de dato complejo, pues siempre hemos visto a los datos como el nivel atómico necesario en el momento y contexto determinado, esto es como el concepto de átomo, en ocasiones puede verse como algo compuesto por toda una estructura de protones, neutrones y electrones, sin embargo, en otras ocasiones lo vemos como un elemento indivisible.

De acuerdo a Date, hay especialmente dos únicas ventajas verdaderas del modelo orientado a objetos: el concepto de tipo de dato – tipos de datos definidos por el usuario arbitrariamente, complejos con encapsulamiento y funciones definidas por el usuario - y la herencia, sin embargo en ésta última, hay que tomar algunas consideraciones.

Los SABDOR carecen también de un modelo común, debido a la falta de estándares, por ello existen grupos que trabajan en ella. Con este objetivo, ANSI (Instituto Nacional Estadounidense de Estándares por sus siglas en inglés) está definiendo el Estándar SQL-3 que incorpora muchos de los aspectos del paradigma Orientado a Objetos para las Bases de Datos Objeto Relacionales, su futuro, sin embargo, es incierto, ya que la ODMG ha ofrecido a ANSI su estándar para que sirva de base para un nuevo SQL, con lo que sólo habría un único estándar de Bases de Datos.

En lo que respecta al modelo objeto-relacional, trata de unificar aspectos tanto del modelo relacional como del modelo Orientado a Objetos (anotando claro, que no existe un estándar definido para las Bases de Datos Orientadas a Objetos ni una definición clara aún de una Base de Datos Objeto-Relacional).

El modelado de objetos describe un sistema a través de los objetos que tienen identidad, comportamiento y un estado encapsulado. El modelo relacional describe a un sistema por su información. Luego entonces, ¿cómo podría un modelo relacional soportar el modelado de objetos?

El modelo relacional parece no tener forma de representar cualquiera de las propiedades del modelo de objetos apropiadamente. Aparentemente, las tuplas nunca permitirían la identidad o el encapsulamiento. Los valores en una tupla están encapsulados pero son valores puros, de forma que nunca han tenido ni identidad ni estado. Afortunadamente, esto no es cierto. La lógica de predicados es bastante buena para describir el estado del mundo (o de un modelo del mundo), por lo que las Bases de Datos Relacionales pueden describir muy bien el estado de un modelo de objetos.

Algunos reclamos en la eficiencia del modelo relacional son justos, mientras otros son un completo fraude. Como consecuencia, algunas de las extensiones propuestas al modelo son genuinas, significando que hacen un verdadero servicio agregando funcionalidades muy útiles; otras sin embargo, son falsas, lo que quiere

decir que no agregan ninguna funcionalidad realmente nueva o que la funcionalidad que agregan no es útil. Nótese en este punto que el que una extensión no sea auténtica, no significa que sea algo "malo" sino que no es realmente una extensión al modelo. Ejemplos de extensiones genuinas tenemos a los operadores EXTEND y SUMARIZE, los operadores relacionales de comparación y la teoría de las vistas actualizables. Como ejemplos de extensiones inútiles tenemos a las consultas cuantitativas (*"quota queries"*) que en realidad son esencialmente una forma sintácticamente corta de una funcionalidad que ya existe, soporte de fecha y tiempo, que ya existen funcionalmente, y el tipo de dato REF (ref=referencia) que supone un retroceso a los modelos que hacían uso de apuntadores y sus correspondientes complicaciones, eliminados racionalmente por Codd por amplias razones de peso.

De acuerdo a lo escrito por Date y Darwen en el Tercer Manifiesto, el modelo relacional no necesita extensión, ni corrección, ni inclusiones, mucho menos modificaciones para soportar funcionalidad de objetos. Todo lo que necesita, es soportar dominios relacionales propiamente (lo que SQL nunca hizo), reconociendo estos dominios básicamente como tipos de datos abstractos (ADT). Luego entonces, el modelo O/R no es una genuina extensión al modelo relacional.

Existen algunas extensiones genuinas al modelo relacional, publicadas por el mismo Code en 1979 bajo el nombre de "Extending the Relational Model to Capture more Meaning", mejor conocido como el documento RM/T. Lo que realmente proponía este documento, era un conjunto de extensiones "semánticas" al modelo original. Como lo más destacable en este documento podemos señalar:

- Que el modelo extendido hace innecesaria la distinción entre entidades y relaciones – viendo estas últimas como un tipo especial de entidad.
- Los aspectos estructurales y de integridad del RM/T están más extensa y precisamente definidos que los del modelo entidad – relación.

- El RM/T incluye sus propios operadores especiales, aparte de los operadores del modelo relacional básico (aunque hace falta mas trabajo en esta área).

En resumen el modelo RM/T trabaja de la siguiente forma:

1. Las entidades, incluyendo las relaciones, son representadas por relaciones-e y relaciones-p, ambas formas especiales de la relación n-aria general. Las relaciones-e son usadas para registrar el hecho de que ciertas entidades existen, las relaciones-p son usadas para registrar ciertas propiedades de estas entidades (las relaciones-e son de grado exactamente uno, las relaciones-p son de grado al menos dos).
2. Una variedad de relaciones pueden existir entre las entidades, por ejemplo, entidades tipo A y B pueden ser relacionadas juntas en una asociación (el término usado por RM/T para las relaciones muchos a muchos), o las entidades tipo Y pueden ser un subtipo de las entidades tipo X. RM/T incluye una estructura formal de catálogo por el cual cada relación puede ser conocida por el sistema, que es entonces capaz de forzar las restricciones de integridad implicadas en cada relación.
3. Como ya se mencionó un vasto número de operadores de alto nivel son agregados para facilitar la manipulación de los variados objetos RM/T (relaciones-e, relaciones-p, catálogo de relaciones, y otras más).

Además, provee un esquema de clasificación de entidades, lo que constituye el aspecto más significativo, o al menos el más visible del modelo entero. Las entidades están clasificadas (de manera informal) en tres categorías:

1. Núcleos (*Kernels*): entidades de existencia independiente; aquello acerca de lo que es la Base de Datos realmente. Los núcleos, son entidades que nunca serán características o asociaciones, como por ejemplo los proveedores, las partes pero no los embarques de un sistema de proveedores y partes.

2. Características (*Characteristics*): es una entidad cuyo propósito primordial es describir o "calificar" alguna otra entidad. Un ejemplo puede ser una línea individual de artículos de una orden de un cliente. La existencia de las características depende de la entidad que describen. La entidad descrita puede ser un núcleo, otra característica o una asociación
3. Asociaciones: una entidad asociativa es una entidad cuya función es representar una relación muchos a muchos entre dos o más entidades distintas. Los embarques en una base de datos de proveedores y partes es un ejemplo de este tipo de entidades. Las entidades asociadas pueden ser de cualquier tipo (núcleo, características e incluso asociaciones).

Las entidades, (sin importar su clasificación) pueden tener propiedades, por ejemplo, las partes tienen colores, las líneas de artículos tienen costo, los embarques tienen cantidad.

En particular, cada entidad (nuevamente sin importar su clasificación) pueden tener una propiedad cuya función es designar algunas otras entidades relacionadas, por ejemplo, las órdenes estás designadas por un cliente. Una designación representa una relación de uno a muchos entre dos entidades. Nótese que la idea de designación fue agregada posteriormente, pues no fue incluida en el documento RM/T original.

Las entidades supertipo y subtipo son soportadas. Si B es un subtipo de A, entonces B es un núcleo, una característica o una asociación dependiendo de que A sea un núcleo, una característica o una asociación. Sin embargo, el documento RM/T no hace referencia alguna al concepto de herencia, por el contrario, tiene más respecto a los conceptos de supertablas y subtablas expresado en el SQL3, como se discute en el Tercer Manifiesto.

Cada uno de estos puntos encuentra un equivalente en el modelo relacional:

Una entidad núcleo corresponde a una entidad independiente, una característica corresponde a una entidad dependiente y una asociación corresponde a una relación.

Los lenguajes de consulta y procedurales, así como las llamadas interfaces de los Sistemas de Administración de Bases de Datos Orientadas a Objetos son familiares: SQL-3, lenguajes procedurales, OBDC, JDBC e interfaces propietarias son extensiones de Sistemas de Administración de Bases de Datos Relacionales y los líderes de venta son bien conocidos: IBM, Informix y Oracle.

Los Sistemas Administradores de Bases de Datos Objeto Relacionales agregan nuevas capacidades de almacenamiento de objetos a los sistemas relacionales sobre la esencia de los modernos sistemas de información. Estas nuevas facilidades integran la administración de datos tradicionales organizados en campos, objetos complejos como series de tiempo y datos geoespaciales y diversos medios binarios como audio, video, imágenes o *applets.* A través del encapsulamiento de métodos con las estructuras de datos, un SABDOR puede ejecutar complejas operaciones analíticas y de manipulación de datos para buscar y transformar multimedios y otros objetos complejos.

Como una tecnología en evolución, las ventajas de las Bases de Datos Objeto Relacionales heredan la administración de transacciones robustas y el rendimiento de sus "antecesores", los Sistemas de Administración de Bases de Datos Relacionales y la flexibilidad de sus "primos" los Sistemas de Administración de Bases de Datos Orientadas a Objetos. Los diseñadores de Bases de Datos pueden trabajar con las estructuras tabulares y los Lenguajes de Definición de Datos (DDL) que les son familiares mientras asimilan las nuevas posibilidades de la tecnología de objetos.

Las más importantes características de las Bases de Datos Objeto Relacionales son los tipos de datos definidos por el usuario (UDT, por sus siglas en inglés *User Defined Types*, funciones definidas por el usuario (UDF) y las infraestructuras – métodos de acceso indizado y el optimizador mejorado - que los soportan.

Los tipos de datos definidos por el usuario pueden ser *distincts*, *opaque*(base) y *row* (*composite*). Los tipos *distinct*, también conocidos como valores tipo son derivados de otros tipos pero tienen sus propios dominios (permiten valores de conjunto), operaciones, funciones y alcances: las aplicaciones de los Sistemas de Administración de Bases de Datos Objeto - Relacionales pueden ser fuertemente tipificadas, ayudando a asegurar la integridad de las aplicaciones. Los tipos definidos por el sistema (y los *not user-distinct*) son familiares: la mayoría de los sistemas relacionales implementan el tipo *money* como un tipo numérico con un número predefinido de decimales. Es lógico sumar y restar valores monetarios y multiplicarlos por escalares, pero no por otros valores monetarios. Los tipos *money* pueden tener funciones como una de crecimiento (interés) y alcance como los reales y los tipos cadena, si los métodos asociados con este tipo los necesitan.

Conceptos asociados a las Bases de Datos Objeto Relacionales.

Extensibilidad

El uso tradicional de los Sistemas de Administración de Bases de Datos estaba limitado a los tipos de datos primitivos manejados por la mayoría de los fabricantes. En los sistemas de Bases de Datos Objeto Relacionales, la "extensibilidad" significa que tiene ventajas para los usuarios (principalmente programadores habituados al paradigma orientado a objetos) para definir y soportar nuevos tipos de datos.

Algunos patrones comunes han emergido de las aplicaciones objeto-relacionales. Los profesionales del área que han usado esta tecnología repetidamente han comenzado a entender y reconocer ciertas estructuras y comportamientos que las aplicaciones de Bases de Datos Objeto Relacionales han exhibido exitosamente. Estas estructuras y comportamientos han sido denominados Servicios objeto-relacionales comunes y les daremos un vistazo en seguida.

Persistencia.

Este es un término usado para describir como los objetos utilizan un medio de almacenamiento secundario para mantener su estado a través de sesiones discretas. La persistencia provee al usuario la habilidad de salvar objetos en una sesión y acceder a ellos en una sesión posterior. Cuando son subsecuentemente accedidos, su estado (por ejemplo, sus atributos) serán exactamente los mismos que se tenían en la sesión previa. Debido a que en sistemas multiusuario esto puede no ser completamente cierto pues múltiples usuarios pueden modificar los objetos entre una sesión y otra de determinado usuario, la persistencia se interrelaciona con otros servicios los cuáles describiremos a continuación:

- Administración de la conexión al origen de los datos (*Data Source*)
- Recuperación de objetos.
- Almacenamiento de objetos.
- Eliminación de objetos.

Consulta.

El almacenamiento persistente de los datos es de poca utilidad sin un mecanismo de búsqueda para recuperar objetos específicos. Las facilidades de consulta permiten a las aplicaciones interrogar a la Base de Datos y recuperar objetos en base a una amplia variedad de criterios. Las operaciones básicas de consulta son FIND y FIND UNIQUE devolviendo correspondientemente varios, un solo objeto o ninguno que cumpla el criterio de consulta.

Herencia:[8]

La herencia basada en clases es usada para la fácil creación de clases que tengan subclases que heredan los atributos y el comportamiento de la superclase.

¿Qué es heredar entre tablas? No es herencia propiamente dicha, sino sólo una multirelación comprimida y administrada. Las tablas están relacionadas a través de atributos comunes y por predicados comunes. La herencia de tablas no afecta en cambio, el hecho de si las tablas están relacionadas o no, sólo es un simple camino para implementar posibles tablas relacionadas. En este sentido, es similar a la herencia de clases, pero no hay razón para sobrecargar el término herencia. Según los escritos de Date en el Tercer Manifiesto:

1. La herencia se aplica a los conceptos de subdominio, superdominio y tipos. Se aplica herencia a dominios, no a relvars y a operadores, no a atributos. Además, sólo ha sido considerada únicamente la herencia simple, la herencia compuesta no ha sido considerada por faltar estudios alrededor de la misma.
2. Asumiendo que el dominio V' ha sido definido como un subdominio del superdominio V entonces:

   - V y V' no son necesariamente distintos, esto es, cada dominio es un subdominio y un superdominio del mismo.
   - Cada subdominio de V' es considerado como un subdominio del dominio V. Cada superdominio de V es considerado como un superdominio de V'.
   - Si V y V' son distintos, se dice que V' es un subdominio propio de V, y se dice que V es un superdominio propio de V'.
   - Debe existir al menos un dominio raíz, es decir un dominio sin un superdominio propio. No se asume que existe únicamente un dominio raíz.

[8] *Idem*

- ❖ Si V' es un subdominio propio de V y V'' no hay un dominio V'' distinto se dice que V es un superdominio inmediato de V'.
- ❖ Sigue el principio de la herencia de operadores: en cualquier lugar donde un valor del dominio V sea permitido, un valor de cualquier subdominio en V' de V será también permitido – principio de substitución.

3. Sea θ un operador polimórfico, y sea x alguna expresión que hace uso de θ. La implementación específica de θ para invocarlo en conexión con la evaluación de x será determinada (en general) en base a la consideración de los dominios de los operandos de θ'. Será posible (en general) para cada operando, participar en este proceso.
4. La definición de una variable escalar necesita algunas extensiones si se desea soportar la herencia de tipos. Sea S una variable escalar declarada del tipo V. Por sustitución, el valor asignado a S en cualquier momento dado puede ser cualquier valor del subdominio V' de V; este valor actual es simultáneamente considerado como del tipo V' y del tipo V'' para todos los superdominios V'' (incluyendo el dominio V) del dominio V'. V' es conocida como el dominio actual (algunas veces, el dominio actual más específico) para S.
5. La definición de una variable tupla necesita algunas extensiones si se permite la herencia de tipos. Sea T una variable tupla con un encabezado dado {<A1, DV1>, <A2, DV2>, ..., <An, DVn>}. Entonces los valores permitido para T son las tuplas de forma {<A1, CV1>, <A2, CV2>, ..., <An, CVn>} donde:

   - ❖ Sea X una variable escalar y Y denote el valor de una expresión escalar. Nótese que Y puede ser considerado como otra variable escalar

Si se considera objetos puros – objetos encapsulados – esto significa que sólo podemos operar en ellos a través de sus operadores predefinidos (o métodos). Cuando consideramos el hecho de las subclases y superclases y su herencia, sólo estamos heredando operadores, no la estructura. No se debe confundir el tipo con su representación. La única estructura cercana es la representación, y ésta es parte de la implementación, no del modelo. Si bajo este nivel, la estructura cambia, por ejemplo, un polígono es una serie de puntos que generan los vértices,

y un rectángulo (subclase del polígono) es representado sólo por los puntos del vértice superior derecho e inferior izquierdo, el código para implementar los operadores tiene que cambiar también, pero nuevamente, esta es la implementación, y no el modelo.

Desde el punto de vista del modelo, si tenemos una función llamada área que regresa el área de un polígono, esto significa que puedes invocar la función área de un rectángulo y obtener la respuesta correcta, aun cuando tengas que reimplementarla. Un punto muy en contra es que al no encontrarse bien definido el modelo, pueden encontrarse diversas definiciones de la herencia.

Algunos de los problemas encontrados en la herencia son los siguientes: supongamos que tenemos una superclase de elipses, y una subclase de círculos. La primera cuestión es, que si debemos particionar la superclase en los elementos que son círculos, de aquellos que no lo son. Una elipse tiene dos ejes, a y b, un círculo es una elipse donde a es igual a b, el punto aquí es averiguar si se actualizaría automáticamente el sistema diciendo que ahora se trata de un círculo y no de una elipse al modificar la elipse para que su eje a fuese igual al eje b. Este es el tipo de cuestiones que deben responderse en cualquier modelo formal y el problema es que diferentes lenguajes y sistemas hacen cosas diferentes.

El problema es que las bases de datos han ido en contra de los avances logrados en las últimas décadas con tal de facilitar una vista que parece más de programador que de diseñador de bases de datos.

El modelo Objeto Relacional

Tercer Manifiesto

(Las "Reglas de Codd" del Modelo Objeto Relacional)

Date y Darwen difieren de la opinión expuesta en el primer manifiesto por Atkinston , así como de Stonebraker respecto a la dirección del modelo relacional y el valor de SQL en dicha dirección. A partir de ello, se deciden a escribir este manifiesto en el que incluyen prescripciones, proscripciones y sugerencias muy importantes, con la completa intención de soportar ciertas características como aspectos del paradigma Orientado a Objetos, consideradas ortogonales al modelo relacional, además de considerar que el modelo relacional no requiere de extensiones, correcciones, inclusiones o sobre todo, desnaturalizaciones realizadas para acomodarlo en lenguajes de Bases de Datos, sobre todo porque ellos tienen la consideración de que el lenguaje SQL no es un fundamento sólido del futuro de las bases de datos, por el contrario, creen firmemente que cualquiera de estos fundamentos debe tener su raíz en el modelo relacional definido por Codd en 1969.

Como podemos apreciar, es posible realizar una integración de los objetos en el modelo relacional. Estos objetos tienen identidad, la cual es el fundamento de la verdadera integración del modelado de objetos. Además, estos objetos tienen atributos y comportamiento, pero estas propiedades deben estar definidas en términos de las sentencias y predicados que son los fundamentos del modelo relacional. Los atributos son leídos de expresiones relacionales y los estados modificados por el comportamiento deben alterar el estado de las variables relacionales en la Base de Datos. Los dos modelos son combinados y nos permiten tener la conveniencia de la notación de objetos con el poder expresivo de la lógica de predicados.

A continuación hacemos una breve presentación de este documento.

Prescripciones del Modelo Relacional[9]

1. Dominios

Un dominio – usado por Date y Darwen como sinónimo indistinto de tipo de dato - es un conjunto determinado de valores. Estos valores son de complejidad arbitraria, manejados únicamente por los operadores definidos por dicho dominio (lo cuál permite implementar el encapsulamiento). Para cada dominio debe disponerse de una notación explícita para la especificación o construcción de un valor arbitrario de ese dominio.

2. Tipos Escalares

Los valores de un dominio son valores escalares que pueden ser arbitrariamente complejos – un arreglo de listas de listas, por ejemplo-. Todos los valores deberán aparecer, al menos conceptualmente, acompañados de un identificador del dominio al que el valor en cuestión pertenece, es decir, del tipo de dato al que pertenecen.

3. Operadores escalares

Para cada lista ordenada de n dominios, cada dominio deberá permitir la definición de n operadores (o funciones) adicionales que se apliquen a la correspondiente lista ordenada de cada uno de sus n valores. La definición de estos operadores deberá incluir una especificación del dominio del resultado de cada operador. Dicha definición de un operador deberá ser lógicamente distinta de la definición del mismo dominio al que se refieren en lugar de estar inmersos en ella. Una función que asigna directamente a alguno de sus argumentos, se conoce como mutador o función con efectos laterales. Una función puede ser desaprobada pero no prohibida y están necesariamente conectadas con la herencia.

[9] *Idem*

4. Representación actual

Siendo V un dominio, los operadores definidos por V deben incluir necesariamente los operadores que muestren la representación actual de los valores de V. La intención es que estos operadores sean usados únicamente en la implementación de otros operadores y que este efecto se logre a través de mecanismos de autorización del sistema. En otras palabras, la representación actual de los valores del dominio deben estar ocultos a la mayoría de los usuarios. Es deseable que se defina un conjunto de operadores que muestren una posible representación (no necesariamente la actual) de los valores del dominio; dados estos operadores, el usuario será capaz de operar los valores en V como si fuese expuesta la actual representación.

5. Valores de Verdad

El lenguaje deberá incluir ciertos dominios preconstruidos, incluyendo en particular el dominio de valores de verdad (verdadero y falso). Los operadores usuales (NOT, AND, OR, IF ..., THEN ..., IFF, etc.) deberán ser soportados en este dominio.

6. Constructor de tipo tupla

Si H es el encabezado de una tupla, entonces, es posible definir un dominio cuyos valores son tuplas con el encabezado H – en otras palabras, una tupla deberá ser un constructor de tipo válido. Los operadores definidos por cada dominio serán precisamente el conjunto de operadores de tuplas soportado por el lenguaje. Estos operadores incluirán uno para la construcción de una tupla a partir de valores escalares específicos y otra para extraer valores específicos de la misma. Además incluirán capacidades análogas para anidar y desanidar tuplas.

7. Constructor de tipo Relación

Si H es el encabezado de una relación, entonces será posible definir un dominio cuyos valores sean relaciones con el encabezado H, en otras palabras, una relación, será un constructor de tipo válido. Los operadores definidos para cada dominio serán, precisamente, el conjunto de operadores relacionales soportados por el lenguaje. Estos operadores incluirán uno para la construcción de una relación a partir de tuplas específicas y para la extracción de tuplas de una relación. Además incluirán la capacidad de incluir anidados y desanidados relacionales.

8. Operador de igualdad

El operador de comparación = será definido para cada dominio. Sean v1 y v2 cada uno denotando un valor de algún dominio V. Entonces v1 = v2 es verdadero si y sólo si v1 y v2 son ambos, miembros de V.

9. Tuplas

Una tupla, t, es un conjunto ordenado de tríos de la forma <A,V,v> donde:

- A es el nombre de un atributo de t. Dos tríos distintos en t no pueden tener el mismo nombre de atributo.
- V es el nombre del dominio único correspondiente a A.
- v es un valor del dominio V, llamado valor del atributo para el atributo A en la tupla t.
- El conjunto de pares ordenados <A,V> que se obtiene de la eliminación del componente v (el valor) del trío en t es el encabezado de t. Dado el encabezado de una tupla, deberá disponerse de la notación para especificar explícitamente (o construir) una tupla arbitraria con este encabezado.

10. Relaciones

Una relación R consiste en un encabezado y un cuerpo. El encabezado de R es una tupla de encabezado H. El cuerpo de R es un conjunto B de tuplas, todas con el mismo encabezado H. Los atributos y sus correspondientes dominios identificados en H son los atributos y correspondientes dominios de R. Dado el encabezado de una relación, debe haber disponible una notación para la especificación explícita (o construcción) de una relación arbitraria con este encabezado. Notemos que cada tupla en R contiene exactamente un valor v para cada atributo A en H, en otras palabras, R está en Primera Forma Normal, 1FN se debe considerar además que existe una diferencia entre las relaciones *per se* y las variables relación. Una distinción análoga aplica a las Bases de Datos. Aún cuando esta distinción terminológica es poco familiar para la mayoría de los lectores, es importante aplicarla para incrementar la precisión del texto.

11. Variables Escalares

Una variable escalar de tipo V es una variable cuyos valores permitidos son valores escalares del dominio específico V, dominio declarado para esta variable escalar. Crear una variable escalar S tendrá el efecto de inicializar S con un valor escalar – aun un valor especificado explícitamente como parte de la operación que crea S, o algún valor dependiente de la implementación si no se especifica ningún otro valor.

12. Variables Tupla

Una variable tupla de tipo H es una variable cuyos valores permitidos son tuplas con un encabezado específico de tupla H, encabezado declarado para dicha variable tupla. Crear una variable tupla T tendrá el efecto de inicializar T con algún valor de tupla - incluso un valor especificado explícitamente como parte de la operación que construye T, o algún valor dependiente de la implementación si no se especifica un valor explícitamente.

13. Variables relación (relvars)

Una variable relación de tipo H es una variable cuyos valores permitidos son relaciones con un encabezado de relación específico H, el encabezado declarado para esa variable relación.

14. Variables Relación Base vs. Derivadas

Las variables relación además pueden ser base o derivadas. Una variable relación derivada es una variable relación cuyos valores en cualquier momento dado son una relación que está definida por el significado de una expresión relacional específica. Una variable relación base es aquella que se crea a partir de una relación vacía. Las relaciones base y derivadas equivalen a los términos conocidos como relaciones base y vistas actualizables respectivamente. Nótese sin embargo, que se consideran aquí muchas más vistas actualizables de las que se consideran tradicionalmente.

15. Variables Base de Datos (dbvars)

Una variable base de datos es un conjunto definido de variables relación. Cada una es sujeto de un conjunto de restricciones de integridad. El valor de una variable Base de Datos dada es un conjunto de pares ordenados <R,r> (donde R es el nombre de una variable relación y r es el valor actual de R) de forma que (a)hay un par ordenado para cada variable relación en la variable Base de Datos, y (b) además, estos valores de las variables relación satisfacen las restricciones aplicables. Cada valor de la variable Base de Datos es llamado una Base de Datos.

16. Transacciones y dbvars

Cada transacción interactúa con una sola variable Base de Datos. Sin embargo, distintas transacciones pueden interactuar con distintas variables Base de Datos, y distintas variables Base de Datos no están necesariamente separadas. Además, una transacción puede cambiar dinámicamente su variable Base de Datos asociada agregando y/o removiendo variables relación. Un propósito del concepto de variable Base de Datos es definir alcance para las operaciones relacionales. Esto es, todas las variables relación mencionadas en cualquier expresión relacional dada serán contenidas en la misma variable Base de Datos. El conjunto de todas las variables relación base puede ser denominado variable Base de Datos base. Las transacciones individuales, sin embargo, interactúan con una variable Base de Datos derivada o de usuario que consiste una mezcla de variables relación base y derivadas. El mecanismo para hacer y terminar una conexión entre la transacción y su Base de Datos no se establece aquí.

17. Operaciones de creación y destrucción

El lenguaje brindará operadores para crear y destruir dominios, variables y restricciones de integridad. Cada dominio, variable o restricción de integridad creada deberá estar bien nombrada. Cada variable relación tendrá al menos una llave candidata, especificada explícitamente como parte de la operación que la crea. La creación y destrucción de variables Base de Datos (asumidas persistentes) son realizadas fuera del ambiente del lenguaje.

18. Álgebra Relacional

El álgebra relacional será expresable sin demasiados circunloquios. Esto implica, entre otras cosas, que la cuantificación universal y existencial serán igualmente fáciles de expresar. Por ejemplo, si el lenguaje incluye un operador específico para la proyección relacional, entonces deberá además incluir un operador específico para la forma general de la división relacional descrita como DIVIDED PER. Implica también, que la proyección sobre atributos especificados será igualmente fácil de expresar.

19. Nombres de relvas y valores relación explícitos

Los nombres de las variables relación y de los valores explícitos de las relaciones serán expresiones relacionales válidas.

20. Funciones Relación

El lenguaje proveerá operadores para crear y destruir funciones cuyo valor en cualquier momento dado es una relación que está definida por determinada expresión relacional. Serán permitidas las llamadas a funciones sin expresiones relacionales aún cuando se permitan valores relación específicos. Podemos considerar que estas funciones corresponden al concepto de vistas de sólo lectura exceptuando que se permiten expresiones relacionales definiendo cada vista de forma parametrizable. Cada parámetro representa valores escalares que son permitidos sin la definición relacional aun cuando se permiten valores escalares explícitos.

21. Asignación de relación y tupla

Deberá permitir

- Asignar la expresión del valor de una tupla a una variable tupla y
- Asignar el valor de una expresión relacional a una variable relación
- Sin prohibir el uso de opciones más sencillas como INSERT, DELETE y UPDATE.

22. Comparaciones

Soportará los siguientes operadores de comparación:

Comparación de:	Operadores de comparación
Tuplas	=, ≠
Relaciones	=, ≠, "is a subset o / subconjunto de"
Miembros de una tupla en una relación	∈

23. Restricciones de integridad

Cualquier expresión que evalúe un valor verdadero es llamada una expresión condicional. Si equivale lógicamente, o es una WFF cerrada del cálculo relacional, será permitida como las especificaciones de una restricción de integridad, las cuáles serán clasificadas en las de dominios, de atributos, de relaciones y de Bases de Datos soportando herencia de restricciones como lo requiera el esquema.

24. Predicados relación y Base de Datos

Cada variable relación tiene un predicado relación correspondiente y cada variable Base de Datos tiene un predicado Base de Datos correspondiente. Estos conceptos, que Date y Darwen creen cruciales y fundamentales tienen lamentablemente demasiado que ver con el pasado y lo amplifican aquí. Básicamente un predicado relación es el AND lógico de todas las restricciones de integridad que aplican a las variables relación en cuestión, y un predicado Base de Datos es el AND lógico de todas las restricciones que aplican a la variable Base de Datos en cuestión. Debe enfatizarse fuertemente el punto de que son los predicados, no los nombres los que representan la semántica de los datos. Decir que un predicado relación satisface los límites de su sentencia, significa precisamente, que no hay asignaciones relacionales que dejen alguna variable relación en algún estado en el que su relación predicado sea violado. Correspondientemente, el decir que un predicado base de datos satisface su sentencia, significa que es necesario que ninguna transacción le deje en algún estado que viole su predicado Base de Datos. Esta prescripción indica además, que no será posible actualizar una "vista actualizable" en la que se viole de cualquier forma la definición de la misma vista. En otras palabras, las vistas actualizables, siempre estarán sujetas a lo que en SQL se llama CASCADED CHECK OPTION (Actualización en Cascada).

25. Catálogo

Cada variable Base de Datos incluirá un conjunto de variables relación que constituyen el catálogo para cada variable Base de Datos. Será posible asignar variables relación al catálogo. Esto implica que el catálogo deberá ser lo que es comúnmente conocido como auto descriptivo.

26. Diseño de Lenguaje

El lenguaje será construido de acuerdo a principios bien establecidos de diseño de un buen lenguaje.

Proscripciones del Modelo Relacional[10]

1. No ordenamiento de atributos

No incluirá construcciones que dependan de la definición de algún orden para los atributos de una relación. En lugar de ello, para cada relación R expresable en D, los atributos de R serán distinguibles por nombre. Esto implica que no pueden existir columnas anónimas ni duplicadas

2. No ordenamiento de tuplas

No incluirá construcciones que dependan de la definición de algún orden de las tuplas de una relación. Esto no implica que el ordenamiento no puede ser impuesto con propósitos como los de presentación, sin embargo, implica que el efecto de imponer un ordenamiento es convertir la relación en algo que no sea una relación.

[10] *Idem*

3. No tuplas duplicadas

Para cada relación R, si t1 y t2 son dos tuplas distintas en R, entonces debe existir un atributo A de R para el cual el valor del atributo A en t1 sea distinto del valor del atributo A en t2. En otras palabras, los registros duplicados están absoluta, categórica e inequívocamente prohibidos.

4. No nulos

Cada atributo de cada tupla de cada relación tendrá un valor del dominio aplicable. Es decir, no se permiten mas valores nulos ni más lógica multivaluada.

5. No errores de nulo lógico

El lenguaje considerará la importancia de todas las relaciones, incluso de aquellas que no contienen atributos.

6. No construcciones de nivel interno

No contendrá construcciones relacionadas con o lógicamente afectadas por el nivel físico, de almacenamiento o interno del sistema. Si al implementar se desea o necesita introducir cualquier tipo de lenguaje de definición de estructura de almacenamiento, las sentencias de dicho lenguaje y el mapeo de las variables Base de Datos al almacenamiento físico serán claramente separables de todo lo expresable en el lenguaje.

7. No operaciones a nivel de tupla

No habrá operaciones de una tupla a la vez en las relaciones. Las sentencias INSERT, DELETE y UPDATE, de haberlas, trabajarán con un conjunto de tuplas, una operación sobre el conjunto de una única tupla será tratado como un caso especial para el cual se definirán métodos cortos de sintaxis. De requerirse una operación de una tupla a la vez, podrá realizarse convirtiendo la relación en una secuencia o lista ordenada con la que se trabajará mediante una iteración.

8. No columnas compuestas

No incluirá ningún soporte específico para "dominios compuestos" o "columnas compuestas" pues esto puede lograrse a través de los dominios soportados en las prescripciones.

9. No sobrecarga de dominios

No se soportará la "sobrecarga de dominios (*domain check override*)"

10. No SQL

El lenguaje no podrá ser llamado SQL.

Otras Prescripciones Ortogonales.[11]

1. Revisión en tiempo de compilación

D permitirá compilación de escritura en tiempo real. Esto no impide la posibilidad de "compila y ejecuta" o la implementación de un intérprete.

2. Herencia simple (condicional)

(Herencia simple) Si el lenguaje permite a algún dominio V' ser definido como un subdominio de un superdominio V, cada capacidad será acorde a un modelo claramente definido y generalmente aceptado, con la esperanza de que este modelo sea encontrado en algún momento.

3. Herencia Múltiple (condicional)

(Herencia Múltiple) Si D permite a algún dominio V' ser definido como un subdominio de un superdominio V, entonces V' no estará a salvo de ser definido adicionalmente como un subdominio de otro dominio W que no sea V o de algún superdominio V.

[11] *Idem*

4. Completez computacional

Será computacionalmente completo, esto es, podrá soportar pero requerirá, invocaciones de los llamados "*hosts programs*" escritos en lenguajes distintos. Similarmente, podrá soportar, pero no requerir el uso de otros lenguajes de programación para la implementación operadores definidos por el usuario.

5. Transacciones explícitas.

El inicio de una transacción será realizado únicamente por un operador de inicio explícito. La terminación puede ser correcta o incorrecta, y ésta última será explícita o implícita (debida a una falla por ejemplo).

6. Transacciones anidadas

Soportará transacciones anidadas.

7. Agregados y conjuntos vacíos

Sea A un operador agregado que es esencialmente un acceso rápido de algún operador iterativo θ. Si algún argumento está vacío, sucederá que :

- Si existe un valor de identidad para θ, entonces el resultado de la invocación será el valor de identidad.
- En otro caso, el resultado será indefinido.

Otras Proscripciones Ortogonales.[12]

1. Relvars y no dominios

Las variables relación no son dominios, es decir, se niega la expresión: relación = clase objeto .

[12] *Idem*

2. No identificadores de objetos

Si el atributo A de la relación R es definido en el dominio V, entonces el atributo A de la relación R contendrá calores del dominio V y no apuntadores a dichos valores. En otras palabras, se niega la idea de que las relaciones pueden tener identificadores de objeto en lugar de valores. De forma más general, de hecho, se niega la idea de que variables escalares de cualquier tipo pueden contener identificadores de objetos en lugar de valores, y por lo tanto, la idea de que los objetos pueden hacer uso de apuntadores para compartir "subobjetos". De hecho se niega también la idea de que los objetos puedan tener cualquier tipo de identificador distinto al valor del objeto.

3. No "variables de instancia pública"

Cualquier notación variable de instancia pública provista para operar valores en los dominios deberá ser mero acceso rápido para ciertas invocaciones especiales a funciones. No será necesaria ninguna correlación directa entre las variables de instancia y la representación actual de los valores del dominio en cuestión.

4. No "variables de instancia protegidas" o "amigas"

No se incluirá el concepto de variables de instancia "protegidas" o el de "amigas", pues se considera mejor el mecanismo de autorización del sistema que los intentos de direccionamiento.

Sugerencias Importantes Sobre el Modelo Relacional.[13]

1. Llaves candidatas para relvars derivadas

Será posible especificar una o más llaves candidatas para cada variable relación derivada. Para aquéllas para las que han sido definidas estas llaves candidatas, será posible nominar exactamente una de ellas como llave primaria.

[13] *Idem*

2. Llaves generadas por el sistema

D incluirá soporte para llaves generadas por el sistema.

3. Integridad referencial

Incluirá algún acceso rápido declarativo conveniente para expresar las restricciones de integridad referencial y las acciones referenciales como el borrado en cascada.

4. Herencia de llaves candidatas

Es deseable para el sistema, que sea capaz de procesar las llaves candidatas de cada relación R expresable en D de forma que:

- Las llaves candidatas de R sean heredadas por R' cuando R sea asignada a R' y
- Las llaves candidatas de R puedan ser incluidas en la información sobre R que está disponible para el usuario del lenguaje.
- Las implementaciones del lenguaje podrán competir por el grado de deducción de llaves candidatas.

5. Quota queries

Brindará un mecanismo conveniente no procedural para expresar las "*quota queries*" sin confundirlas con la conversión de relaciones a listas ordenadas.

6. Cerradura transitiva

Brindará además un mecanismo conveniente no procedural para expresar las "*generalized transitive closure*" de una relación gráfica, incluyendo la habilidad de realizar operaciones de concatenación y agregación.

7. Parámetros tupla y relación

Permitirá los parámetros para funciones de relaciones valuadas para representar tuplas y relaciones así como escalares.

8. Valores por omisión

Proveerá un mecanismo para trabajar con información desconocida del esquema por omisión basado en dominios en lugar de atributos.

9. Migración de SQL

SQL podrá ser implementado en el lenguaje D – no porque sea deseable per se, sino con el ánimo de facilitar la migración para los usuarios actuales de SQL y para que algunos programas existentes hoy en SQL puedan pasar de forma transparente al lenguaje D propuesto.

Otras Sugerencias Ortogonales Importantes

1. Herencia de tipos

Algunas formas de tipo de herencia serán soportadas, por lo que D no incluirá la conversión implícita de tipos.

2. Constructores de tipos colección

Los constructores del tipo colección como LIST, ARRAY y SET deberán ser soportados.

3. Conversión a/desde relaciones

Si se cuenta con una colección de tipo constructor, C, y uno más de tipo RELACION, entonces deberá existir un a función de conversión, digamos C2R, para convertir valores de tipo C a relaciones y lo inverso.

4. Nivel de almacenamiento simple

D estará basado en el modelo de nivel simple de almacenamiento, sin importar pues, la ubicación física de los datos.

Otra importante característica del modelo objeto-relacional es la de permitir Tablas Anidadas (*"Nested tables"*), que son un tipo de dato de colección no ordenada de elementos del mismo tipo. Pueden emplearse como un tipo de dato o como variable, parámetro o valor de retorno de una función. Un ejemplo es el siguiente:

CREATE TABLE *tabla_pedidos* OF *pedidos* NESTED TABLE *linea_pedidos* STORE AS *tabla_linea_pedidos*;

Donde:

tabla_linea_pedidos es la tabla anidada que almacena los valores de todos los atributos *lineas_pedidos* para todos los objetos *pedidos* de la tabla *tabla_pedidos*

DIAGRAMAS OBJETO-RELACIÓN

Para hacer uso de una Base de Datos Objeto Relacional es necesario contar con herramientas de modelado adecuadas. Algunas de las que se encuentran en el mercado son las herramientas de Logic Works, Inc., Computer Systems Advisers Inc., y la infoModelers Inc.

Logic Woks está trabajando en una versión de su herramienta para modelado universal conocida como Universal Modeling Arquitecture (UMA). Esta herramienta tiene un ambiente de modelado visual que permite cambiar de un modelo relacional a uno objeto-relacional fácilmente. La idea es modelar la base de datos con cualquiera de estos modelos y poder cambiar entre ellos de acuerdo a los requerimientos. Brinda además un explorador de modelos y un inspector de propiedades de objetos.

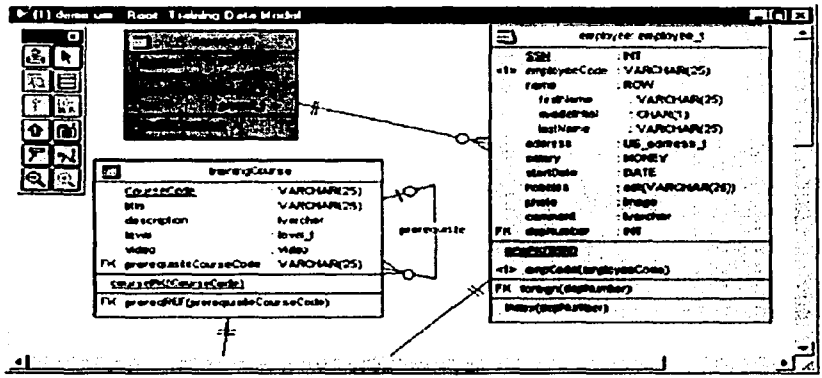


Figura 12. Modelado con Universal Modeler

Computer Systems Advisers desarrolló la herramienta gráfica Universal Modeler que usa el estándar UML. Esta herramienta permite tomar componentes de distintos elementos de datos para construir un solo modelo objeto-relacional.

Mapeo del modelo

Los sistemas de información modernos están construidos alrededor de arquitecturas cliente/servidor para tomar ventaja de muchas configuraciones diferentes y los negocios están notando los beneficios de tener computadoras personales poderosas cooperando en ambientes de red.

Las empresas en evolución, necesitan una forma de integrar su gran inversión en Bases de Datos Relacionales con las aplicaciones Orientadas a Objetos para optimizar el uso de sus redes y mantener centralizados los datos de forma segura y accesible a un gran número de usuarios concurrentes. Retener la riqueza de las aplicaciones del paradigma Orientado a Objetos y un repositorio centralizado demanda una arquitectura más sofisticada.

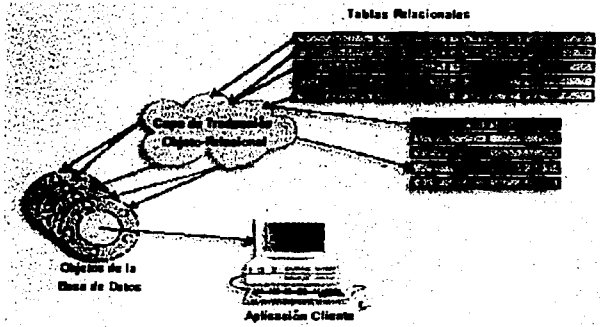


Figura 13. Mapeo Objeto Relacional

Integrar la lógica de las aplicaciones Orientadas a Objetos con las Bases de Datos Relacionales requiere conocimientos especializados. Puede realizarse un "mapeo" de relaciones a un modelo Orientado a Objetos que permita a las aplicaciones tratar como objetos a cada tabla y relación de la Base de Datos.

Un mapeo adecuado debe servir como puente a las diferencias existentes entre los dos modelos. La clave de la tecnología Orientada a Objetos es que encapsula los datos y procesos relacionados de cada objeto, mientras las Bases de Datos Relacionales sólo almacenan datos en forma de tablas interrelacionadas.

Este mapeo de datos puede llegar a ser realmente complejo, por lo que es recomendable el uso de herramientas, ya existentes en el mercado, que realizan este trabajo y reducen el número de defectos en el mapeo.

Relaciones de datos complejos.

Mientras los vendedores de Bases de Datos Relaciones y Objeto – Relacionales gustan de hablar de los tipos de datos complejos, no hablan tanto de las "relaciones complejas", como los casos de relaciones muchos a muchos o de composición (jerárquicas). Con el uso de identificadores de objetos, el uso de este tipo de relaciones se vuelve mucho más sencillo. Esto se debe a la habilidad de navegar a través de referencias, relaciones bidireccionales muchos a muchos y propagación de operaciones a través de las relaciones en forma de objetos compuestos. Los Sistemas de Administración de Bases de Datos Objeto – Relacionales poseen aún una parte muy limitada de estas características bien desarrolladas en las Bases de Datos Orientadas a Objetos y nulas en los Sistemas de Administración de Bases de Datos Relacionales.

Una integración adecuada de las tecnologías de objetos y relacional, requiere una estrategia para realizar un "mapeo" para pasar el modelo de objetos al modelo relacional para su almacenamiento persistente dentro de la Base de Datos.

La raíz del problema es que los objetos no pueden ser directamente almacenados y recuperados en una Base de Datos Relacional. Mientras un objeto tiene identidad, estado y comportamiento además de los datos, una Base de Datos Relacional almacena únicamente datos. Además de ello, un objeto puede ser referenciado directamente a través de su identificador, mientras los datos en una Base de Datos Relacional deben ser referenciados con ayuda de llaves primarias y foráneas y no puede aplicar directamente el concepto de herencia empleado en el paradigma orientado a objetos.

El mapeo es el proceso de transformar las características comunes del modelo de objetos y el modelo relacional y los sistemas que soportan dichas características. Hacer un buen trabajo de mapeo requiere un entendimiento sólido de ambos modelos, sus similitudes y diferencias. De hecho, idealmente debemos iniciar con un modelo integrado que describa ambos modelos, lo cual aseguraría nuestro concepto de cada elemento y de cada relación.

En ocasiones, los productos existentes en el mercado complican un poco este trabajo, ello debido a que no todos cumplen en la práctica con la teoría de cada uno de los modelos, pues las Bases de Datos Relacionales no siempre implementan adecuadamente el modelo y los productos orientados a objetos no tienen un estándar, todo ello, complica el mapeo objeto-relacional.

El mapeo más simple se realiza convirtiendo una clase persistente en una tabla del modelo relacional. En este caso, todos los atributos de la clase persistente son representados como las columnas de la nueva tabla. En algunas ocasiones, este mapeo puede tener conflictos con los modelos de objetos o relacional ya existentes debido a que, mientras el modelo orientado a objetos modela un proceso de negocio usando objetos del mundo real, el objeto del modelo relacional es la normalización y el rápido acceso a los datos. Hay otros dos tipos de mapeo que pueden ayudar a sobrellevar estos problemas: el mapeo de subconjunto (subset) o el de superconjunto (superset).

Mapeo de subconjunto (subset).

En el mapeo subset los atributos de una clase persistente representan sólo una porción de las columnas de una tabla. Esto es útil cuando una clase persistente no está relacionada con algunas de las columnas de la tabla correspondiente en la Base de Datos debido a que no son parte del modelo de negocio. También puede ser usado este tipo de mapeo para crear clases de "proyección" para tablas con un gran número de columnas. Una clase de proyección tiene únicamente información para permitir al usuario seleccionar un registro completo de la Base de Datos que

pueda ser mapeado a otra clase persistente. Este tipo de diseño reduce la cantidad de información transmitida en la red. También puede usarse este mapeo para las clases de árboles de herencia para tablas que usan filtros.

Mapeo de superconjunto (superset).

Una clase persistente con un mapeo de este tipo contiene atributos derivados de columnas de múltiples tablas. También es conocido como mapeo de tablas emparejadas y es usado para crear "clases vista" que ocultan el modelo de datos diseñado, o para mapear un árbol de clase de herencia usando un mapeo vertical lo cual puede mantenerse con inserción, eliminación y actualización que trabaja automáticamente con las llaves foráneas correspondientes.

Mapeo de clases de árbol de herencia.

Las tres estrategias más comunes para representar una clase de árbol de herencia en una Base de Datos Relacional son el mapeo vertical, el mapeo horizontal y el mapeo filtrado.

Supongamos que se tiene el siguiente modelo de una clase abstracta persona de la que heredan dos clases concretas: empleado y cliente:

Mapeo Vertical.

En este mapeo cada clase en el árbol, sea abstracta o concreta, es mapeada a una tabla diferente. Toda rama u hoja en el árbol debe estar conectada con su tabla origen. Esto puede lograrse a través de llaves foráneas que hacen referencia a la llave primaria de la tabla padre. Para instanciar una clase concreta usando este método es necesario realizar una consulta de tipo join de la clase concreta y todas las clases abstractas relacionadas.

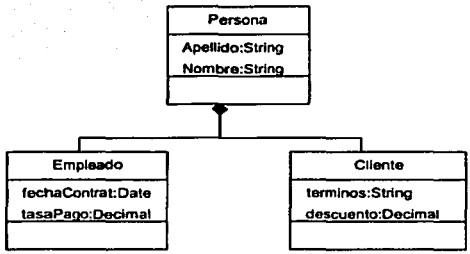


Figura 14. Ejemplo de modelado de una clase

Como podemos observar a continuación, nuestro modelo de ejemplo se mapea verticalmente en tres tablas:

Tabla PERSONA	
Columna	**Tipo de Dato**
PersonaId	Autonumber
Apellido	Text(25)
Nombre	Text(25)

Tabla EMPLEADO	
Columna	**Tipo de Dato**
Id	Autonumber
PersonaId	LongInteger
FechaContrat	Date
TasaPago	Float

Tabla CLIENTE	
Columna	**Tipo de Dato**
Id	Autonumber
PersonaId	LongInteger
Terminos	Text(25)
Descuento	Float

La desventaja de este método de mapeo es que puede resultar en consultas complejas y de consumo de mucho tiempo en jerarquías de herencia de tamaño moderado.

Mapeo Horizontal.

Bajo este tipo de mapeo, cada clase concreta es mapeada en una tabla diferente que contiene columnas para cada atributo en la clase concreta además de cada atributo de la clase padre abstracta. En otras palabras, la clase abstracta no es mapeada a su propia tabla, sino que se incluye en cada clase concreta que herede sus atributos. Este mapeo brinda un desempeño realmente muy rápido y es sencillo de diseñar. Sin embargo, si un atributo de una clase padre abstracta cambia, entonces hay muchas tablas que deben ser modificadas. Consecuentemente, este método es más útil si el árbol de herencia es más orientado a métodos que a atributos. Para ser más específicos, si un número sustancial de clases heredan un gran número de atributos de una clase padre abstracta, entonces los métodos vertical o filtrado son mejor opción.

Como observamos a continuación, el modelo de ejemplo que venimos trabajando se reduce en el mapeo vertical a sólo dos tablas que involucran una sola consulta sencilla para recuperar el registro completo de un empleado o un cliente:

Tabla EMPLEADO	
Columna	**Tipo de Dato**
Id	Autonumber
Apellido	Text(25)
Nombre	Text(25)
FechaContrat	Date
TasaPago	Float

Tabla CUENTE	
Columna	**Tipo de Dato**
Id	Autonumber
Apellido	Text(25)
Nombre	Text(25)
Terminos	Text(25)
Descuento	Float

Mapeo filtrado.

Todas las clases concretas del árbol son mapeadas a la misma tabla, la cuál debe contener columnas para cada atributo de todas las clases concretas y abstractas del árbol de herencia (o de la parte del árbol donde se usa este mapeo). Además, se crea una columna "filtro" en la tabla, cuyo valor es usado para distinguir entre subclases. Las clases abstractas no son mapeadas a esta tabla. Esta ventaja implica un desempeño adecuado pero no cumple las reglas de normalización. Más específicamente, esto podría llevar a un gran número de columnas con valor nulo en la tabla desperdiciando espacio. Consecuentemente, este método es más útil si la mayoría de los atributos son heredados de la clase abstracta padre. Como se muestra a continuación, sólo se requiere una tabla para mapear el modelo que estamos usando de ejemplo:

Tabla EMPLEADO	
Columna	**Tipo de Dato**
Id	Autonumber
Tipo	Integer
Apellido	Text(25)
Nombre	Text(25)
FechaContrat	Date
TasaPago	Float
Terminos	Text(25)
Descuento	Float

La columna *tipo* es usada para filtrar o distinguir el tipo de objeto que se está almacenando. Podría simplificarse las columnas de tasa de pago y descuento en una sola ya que corresponden al mismo tipo de dato.

Tabla CLIENTE	
Columna	**Tipo de Dato**
Id	Autonumber
Nombre	Text(75)

Relaciones Uno a Uno

Estas relaciones suelen representarse, en el paradigma Orientado a Objetos como una referencia directa a un objeto. Cuando un objeto posee una referencia a otro objeto, entonces puede crearlo y eliminarlo, e incluso, cuando es actualizado, todas las referencias en él contenidas son actualizadas automáticamente. En el ejemplo que hemos definido, la clase FACTURA tiene una referencia contenida o una relación Uno a Uno con la clase CLIENTE, si esta referencia no estuviese contenida en la FACTURA, entonces una actualización a la misma no significaría una actualización al CLIENTE. En el modelo relacional, esta referencia suele mantenerse por llaves foráneas inmersas en la tabla. También es posible implementar esto a través de una tabla de tipo "join" o transitiva.

Relaciones Uno a Muchos

En el modelo de objetos hay dos tipos de relaciones uno a muchos, la de agregación y la de asociación. La de agregación requiere que se defina un atributo como una colección contenida en la tabla y la de asociación se representa como un atributo que hace referencia a una colección. La diferencia entre ellos es que en la relación contenida en la tabla, cuando se realiza una actualización todos los objetos en dichas colecciones son actualizados automáticamente.

Mapeo de Relaciones Objeto.

Para el estudio de este tipo de mapeo, usaremos el siguiente ejemplo de un diagrama de clases empleando UML de la simplificación de clases de negocio de una aplicación de un punto de venta (mejor conocido como POS, por sus siglas en inglés *Point of Sale*).

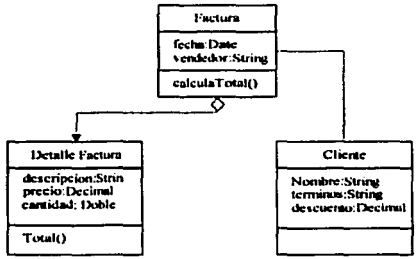


Figura 15. Diagrama de clases de un POS

Este modelo puede mapearse a una Base de Datos Relacional en una forma como esta:

Tabla FACTURA	
Columna	**Tipo de Dato**
Id	Autonumber
ClienteId	LongInteger
Fecha	Date
Vendedor	Text(30)

Tabla DETALLE_FACTURA	
Columna	**Tipo de Dato**
Id	Autonumber
InvoiceId	LongInteger
Descripción	Text(30)
Precio	Currency
Impuesto	Doble

Agregación.

En una relación de agregación la clase propietaria tiene una referencia a su colección mediante un atributo de colección y las clases componentes tienen una referencia directa a la propietaria usando un atributo de referencia. Un objeto propietario maneja automáticamente todas las operaciones de sus objetos componentes en la Base de Datos y este comportamiento es recursivo, de forma que cualquier actualización a un propietario se transmite a sus componentes. En el ejemplo, la tabla DETALLE_FACTURA tiene una llave foránea que hace referencia a su propietaria, la tabla FACTURA.

Asociación.

En una asociación, la clase del lado Uno no es propietaria absoluta de la otra clase, sino que hace referencia. Esto significa que un objeto en el lado Uno de la relación no puede crear, catalizar o eliminar objetos en el Muchos de la relación, sino únicamente recuperarlos. En este caso, se usa un atributo de colección por referencia que nada más puede ser usada por objetos de la misma clase.

Relaciones Muchos a Muchos.

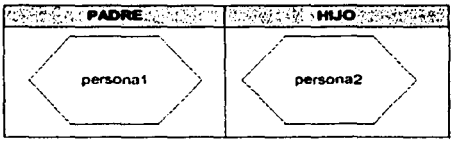
Una relación Muchos a Muchos puede ser pensada como una relación Uno a Muchos bidireccional. Para crear este tipo de relación se debe definir simplemente un atributo de colección referenciada en cada una de las clases envueltas en la relación. En el modelo relacional esto se logra usando una tabla "join" o transitiva.

MODELADO OBJETO-RELACIONAL

Para ver como se puede modelar fácilmente el estado de un modelo de objetos, primero expandiremos el modelo relacional ligeramente.

Objetos como valores de atributos en una tupla.

¿Que tal si permitimos objetos (con identidad y estado) como valores de atributos en una tabla?

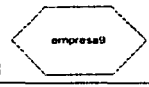


En lugar de simplemente tener un número de seguro social o una CURP (o incluso un valor complejo como una imagen gráfica o un rectángulo) como el valor de un atributo de una tupla, podemos tener una "persona" como valor de un atributo de una tupla. Esto nos permitiría tener un predicado como: "Esta persona es padre de ésta otra" en lugar de decir: "La persona con el número de seguro social 215631254 es el padre de la persona con el número de seguro social 502487566 (el cuál podría cambiar), pues ahora hemos hecho representaciones directas de la gente en si misma.

Esta unión provee mucha flexibilidad. Donde soliamos usar un tipo de dato primitivo o abstracto podemos ahora tener un objeto. Para lograr esto necesitamos incrementar los dominios para ser capaces de tomar sus valores de una lista de objetos existentes o ser capaces de crear un nuevo objeto cuando sea necesario.

Este problema es especialmente obvio para las tablas básicas con atributos (o entidades, en el modelo relacional) donde cada fila lista los atributos de un objeto.

¿Debemos ahora pedir a un objeto que cambie su atributo o hacer cambios en una fila en la tabla?

OBJETO	NSS	NOMBRE	APELLIDO	EMPRESA
	123-45-6789	Luis	Beltrán	
	234-56-7890	Arturo	Beltrán	

Debido a que este es un modelo relacional, las características de este modelo toman precedencia, pues sólo deseamos extender el modelo relacional lo suficiente para representar objetos más fácilmente en éste. Como el modelo relacional tiene una serie de ventajas completas para cambiar el estado de la Base de Datos, no necesitamos agregar algo más. Entonces, no cambiaremos el estado de un objeto a través de los métodos, pero por el contrario debemos modificar las variables de relación apropiadas (agregando, eliminando o reemplazando y cambiando tuplas) para lograr los cambios deseados.

Atributos objeto como vistas

¿Podríamos permitirle a un objeto responder a una pregunta acerca de sus atributos? Esto sería muy conveniente. En lugar de tener que buscar en la tabla PARENTESCO por los padres de una persona determinada, y en la tabla PERSONA por los atributos básicos de los mismos, podríamos preguntar a la persona1 por su primer apellido y por sus padres. Desde la perspectiva relacional, el objeto persona podría proveer una vista centralizada de todas las tablas que pueden referirse a una persona.

Asumiendo la notación de <persona 1> representando a la persona actual, entonces la consulta:

SELECT Persona.nss FROM persona WHERE Persona.Objeto = persona1

equivale a: SELECT persona1.nss

Esta consulta asume que es obvio que la relación controla el atributo nss para la persona y los objetos de las clases relacionadas. En general, estas consideraciones son verdaderas, por lo que necesitamos definir explícitamente como los atributos de un objeto son una vista de las relaciones apropiadas. Algo como:

```
CREATE DOMAIN PERSONA CLASS {
nss AS SELECT nss FROM Persona WHERE objeto = THIS
nombrePersona AS SELECT nombre FROM Persona WHERE objeto= THIS
paterno AS SELECT padre FROM PARENTESCO WHERE hijo = THIS
hijos AS SELECT hijo FROM PARENTESCO WHERE padre = THIS
}
```

Donde agregamos la habilidad de declarar un dominio el cual tendrá objetos con identidad como sus valores (una clase), la habilidad de declarar atributos de esta clase como vistas en la Base de Datos, y la nueva palabra clave THIS para referirnos a cualquier objeto con el que estamos tratando actualmente. Bajo la cubierta de estas vistas podría optimizarse significantemente lo cual podría darnos ventajas de desempeño como un objeto en la Base de Datos.

```
SELECT hijos.nombrePersona
FROM Persona.objeto AS padre, padre.hijos AS hijos
WHERE padre.nombrePersona="Arturo"
```

COMPORTAMIENTO

Para agregar comportamiento a los objetos, requerimos la habilidad de especificar métodos de implementación para cualquier objeto dado. Algo así:

```
CREATE DOMAIN PERSONA CLASS {
        nombra(nuevoNombre : String) AS UPDATE ... WHERE objeto = THIS
}
```

Con todo lo anterior tenemos suficiente para modelar un sistema de información que puede almacenar comportamiento que es fácilmente agregado a objetos, tablas o a la Base de Datos total. Efectivamente, estos son tres tipos de objetos que podrían tener comportamiento en nuestro sistema relacional.

HERENCIA

La herencia basada en tipos puede ser usada como parte de las restricciones de integridad referencial. Cuando especificamos que un dominio es de un tipo particular, estamos restringiéndolo a objetos que implementan ese tipo así como cualquier subtipo que esté conforme a los valores del dominio. Esto permite flexibilidad e integridad similar para un tipo basado en un lenguaje de programación.

OBJETO	NSS	NOMBRE	APELLIDO	EMPRESA
persona1	123-45-6789	Luis	Beltrán	empresa3
persona1	234-56-7890	Arturo	Beltrán	empresa9

Por ejemplo, si tenemos las siguientes dos clases:

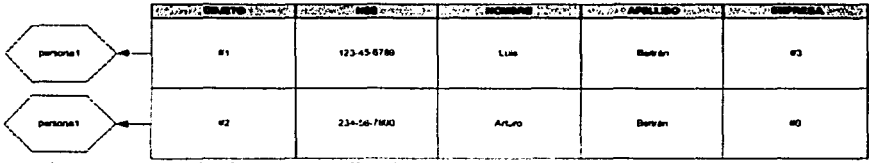
CREATE DOMAIN Persona CLASS {...}
CREATE DOMAIN Empleado CLASS EXTENDS Persona {...}

Entonces un objeto de cualquiera de las dos clases puede ser el valor de un atributo del tipo Persona. Sabemos que el objeto soportará al menos la interfaz Persona además de algún soporte adicional de la interfaz Empleado.

Las clases basadas en herencia pueden ser usadas para crear fácilmente clases con subclases con sus mismos atributos y comportamiento.

Llaves de identidad.

Como en la tecnología actual no siempre se puede almacenar objetos con identidad de manera directa. En lugar de ello, almacenamos una llave primaria del objeto como una "sombra" o "referencia" del objeto (*"objectShadow"*). La existencia de dicha llave indica que existe un objeto pero que la Base de Datos no



puede representarlo de manera directa.

Distintivos

En el ejemplo anterior, el tipo del objeto es obvio, siempre se trata de una persona correspondiente a la columna OBJETO y siempre es una compañía correspondiente a la columna EMPRESA. Pero, ¿qué tal si permitimos que los dominios varíen sobre múltiples clases?, ¿cómo identificaríamos el tipo de objeto al que hace referencia la llave?. Necesitaríamos alguna manera de poder

Por ejemplo, si tenemos las siguientes dos clases:

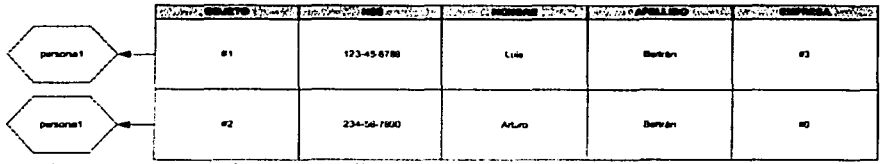
CREATE DOMAIN Persona CLASS {...}
CREATE DOMAIN Empleado CLASS EXTENDS Persona {...}

Entonces un objeto de cualquiera de las dos clases puede ser el valor de un atributo del tipo Persona. Sabemos que el objeto soportará al menos la interfaz Persona además de algún soporte adicional de la interfaz Empleado.

Las clases basadas en herencia pueden ser usadas para crear fácilmente clases con subclases con sus mismos atributos y comportamiento.

Llaves de identidad.

Como en la tecnología actual no siempre se puede almacenar objetos con identidad de manera directa. En lugar de ello, almacenamos una llave primaria del objeto como una "sombra" o "referencia" del objeto (*"objectShadow"*). La existencia de dicha llave indica que existe un objeto pero que la Base de Datos no

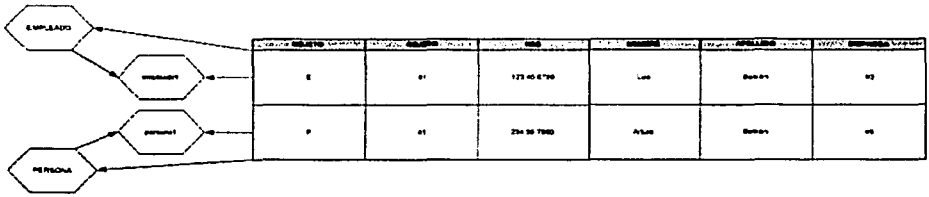


puede representarlo de manera directa.

Distintivos

En el ejemplo anterior, el tipo del objeto es obvio, siempre se trata de una persona correspondiente a la columna OBJETO y siempre es una compañía correspondiente a la columna EMPRESA. Pero, ¿qué tal si permitimos que los dominios varíen sobre múltiples clases?, ¿cómo identificaríamos el tipo de objeto al que hace referencia la llave?. Necesitaríamos alguna manera de poder

diferenciar la clase del objeto. Para ello, podemos agregar un distintivo a la llave. En su caso, primero encontraremos la clase a la que pertenece el objeto y después el objeto adecuado.



TECNOLOGÍAS Y HERRAMIENTAS DEL MODELO OBJETO-RELACIONAL

Actualmente, los Sistemas de Administración de Bases de Datos Orientadas a Objetos, como ObjectStore o Poet, no tienen la madurez suficiente para soportar aplicaciones con grandes volúmenes de datos que requieren tiempos de respuesta rápidos. Para este tipo de aplicaciones se utilizan las nuevas versiones de los productos objeto-relacionales.

Además de ello, no todos los productos que realizan un mapeo objeto-relacional son capaces de soportar el modelo completamente integrado y todas las capacidades cliente servidor correspondientes. Algunos de ellos eligen sólo ciertas características. La siguiente tabla describe los diferentes niveles de mapeo objeto-relacional posibles:

Mapeo ligero a objetos	Tiene algún uso de objetos a nivel cliente y trata de aislar la mayoría del código de las aplicaciones de las especificaciones SQL. Convierte tuplas de la Base de Datos en objetos básicos en el cliente. Encapsula llamadas a código SQL en algunas clases/métodos por lo que la aplicación no requiere tanta atención en ello.
Mapeo medio a objetos	Uso de objetos en todo el cliente, se escribe todas las aplicaciones en términos de objetos. Las aplicaciones pueden necesitar administrar las transacciones de éstos. Flexiblemente, convierte tuplas de la Base de Datos en los tipos de objetos apropiados y mantiene la identidad de dichos objetos. Administra las asociaciones entre objetos de tipo variable. Permite recuperar consultas para ser especificadas en términos del modelo de objetos y las convierte en las llamadas SQL necesarias. También permite que las consultas sean respondidas de manera local (sin la Base de Datos).
Mapeo completo a objetos	Uso completo de objetos en el código de la aplicación, de forma similar al nivel anterior, pero permite además formato de almacenamiento de clases a nivel general (para herencia y composición), administra la replicación y las diferencias respecto al servidor, permite que las consultas se ejecuten de manera local siempre que esto sea posible y combina muchas consultas sobre objetos simples en una sola consulta a la Base de Datos.

Elegir una técnica para usar en estos niveles, puede impactar el costo, la escalabilidad, el comportamiento de la aplicación, el tiempo de desarrollo y el mantenimiento. No existe una técnica mejor para todos los criterios. Las técnicas mas sofisticadas no siempre son apropiadas para todas las aplicaciones y muchas aplicaciones trabajan muy bien con Bases de Datos Relacionales puras. Otras aplicaciones crecerán en tamaño y complejidad de forma que será mas fácil administrarlas con modelos Orientados a Objetos puros, y tendremos que decidir que productos y técnicas necesitaremos.

Entre las herramientas con las que contamos con capacidad de modelado objeto-relacional, se pueden encontrar ciertos ejemplos que son independientes de cualquier Sistema de Administración de Bases de Datos, algunos de los cuales se describen a continuación.

El modelo objeto-relacional incluye tanto datos como procesos (la información que se tiene y lo que se va a hacer con ella), en contraste, la Bases de Datos Relacionales soportan únicamente un limitado encapsulamiento de operaciones y procesos. Es claro que las herramientas y metodologías de diseño deben ahora modelar tanto datos como operaciones, un requerimiento que clama por características que cubran tanto las tradicionales Bases de Datos Relacionales como las aplicaciones Orientadas a Objetos y permitan a los diseñadores encapsular óptimamente funciones con datos y generar clases o códigos de estructuras externos a la Base de Datos. Finalmente, las herramientas deberían ofrecer la capacidad de trabajar con tipos de datos y métodos predefinidos por el usuario así como extender el modelo con tipos de datos y funciones definidos por el usuario.

Modelado de Roles de Objetos (Object Role Modeling)

Las metodologías tradicionales de modelado de Sistemas de Administración de Bases de Datos Relacionales, niegan típicamente el nivel conceptual. El modelado de roles de objeto (ORM) es una metodología que sí incluye ese nivel. Ha sido formado a través de algunos años durante los cuales la metodología no había atraído la atención hasta recientemente con el trabajo de Terry Halpin y su implementación en la herramienta de diseño InfoModeler. El lenguaje formal de modelado de roles de objeto (FORML) encapsula el ORM, con la ventaja de capturar los conceptos y reglas de negocio. Los modeladores conceptuales de FORML generan sentencias de datos en forma de verbos, símbolos o diagramas, restringiendo y validando el modelo.

Lenguaje Unificado de Modelado (Unified Modeling Language - UML)

Como hemos comentado en el Capítulo "Bases de Datos Orientadas a Objetos", otra alternativa de modelado de Bases de Datos es UML, una compleja propuesta que incluye la conceptualización y requerimientos de análisis que cubren el modelado conceptual con su respectivo mapeo en clases, componentes y distribución en el sistema y fases detalladas de diseño. Las clases y métodos de UML, sin embargo, son muy equivalentes a los tipos y métodos en un Sistema de Administración de Bases de Datos Objeto Relacionales.

Usando UML

Lo que es importante reconocer, es que el cambio a UML no es difícil para los diseñadores que han trabajado con el modelo Entidad-Relación, e incluso, aún hay aspectos en los que dicho modelo es más robusto que UML. Debemos tener en mente además, que la noción de "Entidad" en el modelo Entidad-Relación está siendo remplazada por la noción de "Clase Objeto" en UML, y notar que la definición de una Clase Objeto es prácticamente la misma que la de una entidad.

El punto crucial en el uso de UML, es que una Clase Objeto o entidad siempre representan algo del mundo real. Esto es un mecanismo clave que puede usarse para verificar la validez del modelo de datos, cuestionando si es posible identificar en el mundo real alguna "cosa" a la que corresponda dicha Clase Objeto.

Un punto también muy importante, es la descripción de cada objeto o Clase Objeto. Una buena descripción no deberá contener nunca frases como "este objeto contiene información acerca de...". es necesario tener en cuenta que un objeto debe ser, en sí mismo, una generalización de algo de nuestro interés.

Desventajas de los diagramas de UML

UML no es una panacea. No resuelve todos los problemas asociados con el modelado Entidad-Relación pero si hace la tarea más sencilla. Existen algunas desventajas al usar UML:

1. No existe el concepto de arco, de modo que si se tienen relaciones "OR", para representarlas en UML es necesario crear una estructura de agregación. Por ejemplo, si deseamos decir que un contrato puede realizarse con una persona física o con una moral o compañía, es necesario crear una Clase Objeto artificial llamada "entidad contratante" y relacionarla con el contrato como se muestra a continuación:
2. Aunado a ello, esta implementación es incompleta en algunos Sistemas Administradores de Bases de Datos, pues suelen faltar dos estructuras:
3. Tablas anidadas: en ellas, el tipo de datos es una tabla en si misma.
4. Arreglos de referencias: funcionan de forma similar a apuntadores que pueden ser usados como referencias que facilitan el almacenamiento de relaciones maestro/detalle en la tabla maestro. Esta construcción preserva el detalle de la tabla.

Herramientas de Modelado para Sistemas de Administración de Bases de Datos Objeto -Relacionales

Cada una de las herramientas que se describen, permiten la especificación de las propiedades físicas (espacio, modo de bloqueo, y demás características) de la Base de Datos, de las tablas y de otros objetos de la Base de Datos. Generan además esquemas, ya sea conforme a la Base de Datos específica o en Lenguaje de definición de datos estándar (DDL).

OR-Compass

Es un nuevo producto de Logic Works y no simplemente una extensión de su ya tradicional Erwin/ERX; sin embargo, puede importarse modelos de ERX. Usa una metodología de modelado entidad relación con notación IDEF1X.

Incluye dos asistentes que ayudan a modelar, un asistente que apoya los tipos de renglones (*Row Type Wizard - RTW*) y uno para los índices funcionales (*Functional Index Wizard - FIW*). El RTW apoya el trabajo de los diseñadores que están migrando una Base de Datos Relacional a una Bases de Datos Objeto Relacionales, lo que permite seleccionar determinadas columnas de una tabla como un tipo de renglón. El FIW guía a través de un índice específico en base a los valores de un campo determinado. Soporta la migración automática de llaves al crear relaciones. Se puede arrastrar y soltar una columna desde una tabla hasta otra para establecer las relaciones. Es una interfaz muy fácil de usar.

InfoModeler

Permite aplicar el modelado conceptual con FORML, así como los modelos lógicos relacionales de ER e IDEF1X. Tiene una colección de tipos predefinidos muy limitada que sólo puede ampliarse mediante una conexión directa a un Sistema de Administración de Bases de Datos Objeto – Relacionales y de igual forma, no es posible trabajar ingeniería inversa para obtener un diseño a partir de un script sin dicha conexión directa.

Universal Modeler

Es un conjunto de herramientas interoperables que incluye PBM para modelado de procesos de negocio, ERX para modelado conceptual relacional y RDM para el modelado físico de datos. Tiene conexión a muchas Bases de Datos Relacionales y gener código para Delphi y Power Builder. Puede ser usado como un repositorio de modelos. Soporta además el empleo de UML, IE y tipos de dato definidos por el usuario.

Power Designer

Es una herramienta de Sybase que soporta tipos de dato abstractos con soporte total para objetos y modelado UML desde el año de 1998.

CAPÍTULO IV

CAPÍTULO IV. COMPARACIÓN DE LOS MODELOS

VENTAJAS DE LAS BASES DE DATOS RELACIONALES

- Permite el acceso eficiente de datos en línea y/o en *batch* (por lotes).
- Portabilidad
- Confiabilidad de los datos
- Puede reducirse la redundancia, así como el desperdicio resultante del espacio de almacenamiento.
- Puede evitarse la inconsistencia (al menos en cierta medida).
- Los datos pueden compartirse. No sólo significa que las aplicaciones existentes pueden compartir los datos de la base de datos, sino también que es factible desarrollar nuevas aplicaciones que operen con los mismos datos almacenados.
- Pueden hacerse cumplir las normas establecidas. Con un control central de la base de datos, el DBA puede garantizar que se cumplan todas las formas aplicables a la representación de los datos. Es muy deseable unificar los formatos de los datos almacenados como ayuda para el intercambio o migración de datos entre sistemas.
- Pueden aplicarse restricciones de seguridad. Al tener jurisdicción completa sobre los datos de operación, el DBA puede, a) asegurar que el único medio de acceder la base de datos sea a través de los canales establecidos y , por tanto, b) definir controles de autorización para que se apliquen cada vez que se intente el acceso a datos sensibles.
- Puede conservarse la integridad.
- Es rápido. La máquina puede obtener y modificar datos con mucha mayor velocidad que un ser humano, así es posible satisfacer con rapidez consultas de casos particulares, sin necesidad de búsquedas visuales o manuales que requieren mucho tiempo.

- El álgebra relacional tiene muchas ventajas. Dadas dos expresiones equivalentes, un optimizador de consultas puede elegir automáticamente la más eficiente.
- El álgebra es un lenguaje de alto nivel que habla en término de propiedades de conjuntos de tuplas y no en término de ciclos. Esto permite especificar que hacer sin tener que dar detalles de cómo hacerlo.

DESVENTAJAS DE LAS BASES DE DATOS RELACIONALES:

- Las Bases de Datos Relacionales son muy vulnerables.
- Se pierde el sentido de propiedad de los datos
- La percepción de pequeños errores se torna difícil.
- Las Bases de Datos Orientadas a Objetos brindan en general mayor rendimiento que las Bases de Datos Relacionales, esto debido a la referencia directa entre objetos a través de apuntadores suaves, haciendo que las Bases de Datos Orientadas a Objetos pasen más rápidamente de un objeto a otro, a diferencia de las Bases de Datos Relacionales, que requieren un join para lograrlo.

VENTAJAS DE LAS BASES DE DATOS ORIENTADAS A OBJETOS

- Una de las principales fortalezas del paradigma Orientado a Objetos aplicado a las bases de datos es su enfoque para el modelado del esquema conceptual, reduciendo la distancia semántica entre los objetos reales, los objetos representados y los objetos almacenados.
- El modelo objeto representa de manera más cercana y fiel el dominio del problema, mapeando las abstracciones de las entidades del mundo.

- Una de sus ventajas más importantes es su flexibilidad, pues pueden añadirse subclases a clases ya existentes para adaptar un sistema a nuevos requerimientos sin necesitar grandes cambios de estructura. Una subclase heredará todos los atributos de la definición de clase original y se especializará en los nuevos campos que se requieren así como los métodos para manipular estos campos, además de generar los espacios para almacenar la información adicional en ellos.
- Otra ventaja es que manipula datos complejos en forma rápida y ágil. La estructura de la base de datos está dada por referencias (o apuntadores lógicos) entre objetos. No se requieren búsquedas en tablas o uniones para crear relaciones. Esta capacidad resulta atractiva en aplicaciones de la ingeniería, donde las relaciones entre componentes dependen de factores diversos. Por ejemplo, consideremos una aplicación en el diseño de vehículos automotores. El fabricante quiere determinar una lista de partes necesarias para un modelo particular de auto, para lo cuál requiere de diferentes decisiones subsecuentes como decidir si el modelo es automático o estándar, de lo que depende si se necesita de un chasis particular o de una caja de velocidades determinada. Escoger un tipo de motor obliga a decidir sobre otras partes requeridas, todo esto hasta el nivel de componentes y piezas individuales. Armar esta lista de componentes resulta más ágil en una BDOO que en una base de datos relacional. En un modelo relacional las tablas deben ser barridas, buscadas cada vez que se indica una condición, resultando, posiblemente, en miles de direccionamientos, a diferencia de estas bases de datos en las que se tiene referencias directas a cada componente requerido.
- Una ventaja más, es que las Bases de Datos se vuelven una extensión persistente del lenguaje de las aplicaciones orientadas a objetos que las emplean, y no son sólo un servicio externo con una interfaz limitada.

- El uso del mismo modelo conceptual para todos los aspectos del desarrollo simplifica al mismo, particularmente con las herramientas CASE Orientadas a Objetos, mejora la comunicación entre usuarios, analistas y programadores, además de que reduce las posibilidades de error.
- Permite el manejo de tipos de datos complejos y más tipos de relaciones i.e.: agregación, especialización.
- Mejor control de versiones.
- Incluye todas las ventajas del paradigma Orientado a Objetos.
- La navegación a través de la Base de Datos es fácil y más natural, con objetos capaces de contener punteros a objetos en la Base de Datos.
- La reutilización reduce los costos de desarrollo.
- El control de concurrencia se simplifica por la capacidad de poner un bloqueo simple en una jerarquía entera (mientras se conserva la opción de hacer bloqueos individuales a los objetos).

DESVENTAJAS DE LAS BASES DE DATOS ORIENTADAS A OBJETOS

- El modelo orientado a objetos no cuenta con un modelo matemático formal aceptado universalmente, el cual sería una poderosa herramienta de análisis y deducción. Esto lo pone en franca desventaja con el modelo relacional, el cual tiene como fundamento la teoría de conjuntos y los conceptos matemáticos de relación; además de tener perfectamente definidos un conjunto de operaciones (álgebra relacional).
- En sí, el modelo orientado a objetos simplifica una estructura evitando elementos o variables repetidas en diversas entidades, sin embargo esto implica dedicarle un minucioso cuidado a las relaciones entre clases cuando un modelo es complejo, la dificultad del manejo de objetos radica en la complejidad de las modificaciones y eliminaciones de clases, ya que de tener variables que heredan otros objetos se tiene que realizar una reestructuración que involucra una serie de pasos complejos.

- La mayor limitación de las Bases de Datos Orientadas a Objetos es la carencia de un estándar, incluyendo la ausencia de un lenguaje de consulta común como el SQL (el mejor intento es el OQL); no existe tampoco una semántica formal para ellas, sin embargo, ODMG-93 es un punto de partida importante para ello.
- El proceso de navegación, está basado fuertemente en apuntadores, de los cuáles una gran proporción deben ser persistentes.
- Se pierde la simplicidad de las tablas relacionales.
- Los cambios en el paradigma orientado a objetos pueden dificultar el desarrollo de los Sistemas de Administración de Bases de Datos Orientadas a Objetos.
- Debido a que no existe un consenso acerca de cómo debiera ser un Sistema Administrador de Bases de Datos Orientadas a Objetos, éstos no han alcanzado un mismo nivel de madurez deseable. Nuevos productos son refinados constantemente tanto en el ámbito académico como en el comercial pero no hay un benchmarking universal como en el caso de las Bases de Datos Relacionales (Reglas de Codd) contra el cuál comparar estos productos. En el mercado actual existen algunos manejadores de Bases de Datos que pretenden ser Orientados a Objetos como: Oracle, Informix, Sybase, Progress, y DB2, sin embargo la falta de una base matemática para este modelo pone en duda esta aseveración.
- La inmadurez del mercado constituye una posible fuente de problemas al considerar la adopción de la tecnología orientada a objetos, por lo que debe analizarse la presencia en el mercado del proveedor a elegir. Además de que la falta de estándares en la industria orientada a objetos representa un riesgo que no cualquiera está dispuesto a adoptar.
- La propiedad de clausura del álgebra relacional que dicta que un operador toma relaciones como operandos y devuelve como resultado otra relación, es una propiedad de mucha utilidad que comúnmente se pierde en los Sistemas Manejadores de Bases de Datos Orientadas a Objetos.

- Mientras el encapsulamiento (la idea de que el estado interno de un objeto sólo es accesible a través de métodos diseñados específicamente para ello) es una técnica valiosa para la modularidad, presenta problemas por lo que respecta a las consultas que realizan preguntas sencillas para las cuáles debemos ir generando métodos específicos mientras el modelo relacional permite su rápida ejecución.
- Los Sistemas Manejadores de Bases de Datos Orientadas a Objetos introducen el manejo de punteros relacionando a los objetos entre sí, lo cual puede ser visto como un retroceso después de los sistemas declarativos del enfoque relacional.

VENTAJAS DE LAS BASES DE DATOS OBJETO RELACIONALES.

- Las Bases de Datos Objeto Relacionales pueden tomar ventaja de la gran cantidad de herramientas maduras en el mercado orientadas a las Bases de Datos Relacionales y que tienen la posibilidad de emplear gracias a que se trata de Bases de Datos Relacionales con pequeñas extensiones.
- Otra ventaja importante es la gran cantidad de expertos orientados a SQL que pueden explotar las características de las Bases de Datos Objeto Relacionales sin la necesidad de grandes y novedosos conocimientos, situación a la que las Bases de Datos Orientadas a Objetos están tardando en llegar.
- Una ventaja adicional para este tipo de Bases de Datos es la permanencia e historia de sus proveedores en el mercado. Pese a cualquier ventaja que pueda representarles, la mayoría de las organizaciones prefieren seleccionar un sistema de Bases de Datos Relacional u Objeto – Relacional que uno Orientado a Objetos debido al tamaño y reconocimiento de los fabricantes, aunque esta ventaja puede irse perdiendo con el tiempo.
- Las Bases de Datos Relacionales son normalmente modelos altamente normalizados con poca abstracción. Cada elemento de nuestro interés (cada entidad), se convierte en una tabla.

- Otra ventaja importante es que con la inclusión de las ventajas del paradigma Orientado a Objetos, el número de entidades en los modelos se reduce considerablemente; los diseñadores están incorporando el paradigma Orientado a Objetos al diseño de sus sistemas. El diseño con este paradigma resulta en modelos de datos más claros incrementando la flexibilidad y facilidad de mantenimiento.
- Son más robustas, pues soportan no sólo los requerimientos especificados durante el análisis, sino que además soportan requerimientos futuros pues son fácilmente adaptables.

DESVENTAJAS

- En todos los casos de extensibilidad, existe un número de restricciones que los desarrolladores deben de conocer para poder considerar los tipos de dato definidos por el usuario. Por ejemplo, los tipos por ellos definidos no pueden ser tipos abstractos por si mismos; el tipo de dato de "referencia" puede ser usado para hacer referencias a renglones en una tabla, pero debe mencionarse el tipo de dato del valor.

CONCLUSIÓN

CONCLUSIÓN

Las Bases de Datos Orientadas a Objetos permiten el desarrollo y mantenimiento de aplicaciones complejas con un costo significativamente menor a las Bases de Datos tradicionales. Permiten que el mismo modelo conceptual se aplique al análisis, diseño, programación, definición y acceso a la base de datos. Esto reduce el problema de traducción entre los diferentes modelos a través de todo el ciclo de vida. El modelo conceptual es la base de las herramientas CASE Orientadas a Objetos totalmente integradas, las cuales ayudan a generar la estructura de datos y los métodos.

También ofrecen un mejor rendimiento de la máquina que las bases de datos por relación, para aplicaciones o clases con estructuras complejas de datos.

La situación actual para implementar Bases de Datos Orientadas a Objetos es análoga a la sufrida por las Bases de Datos Relacionales a mediados de los 70s, cuando había pequeñas discusiones, que finalmente fueron en su mayoría, casos superficiales, con la diferencia de que entonces se estableció desde su surgimiento un modelo común.

Actualmente, se está desarrollando las especificaciones de este tipo de Base de Datos y además se experimenta implementando la tecnología para soportarlo. Para ello, se ha confiado en que después de construir diversos prototipos, surgirá un modelo adecuado y en paralelo una implementación de la tecnología para dicho modelo, puesto que es importante llegar a un acuerdo en la definición de un modelo de Sistema Administrador de Bases de Datos Orientado a Objetos.

Sin embargo, aunque existen diversas propuestas de modelos formales para el paradigma Orientado a Objetos; hasta la fecha no se ha publicado oficialmente ninguna; no obstante, es importante considerar que estas propuestas están suficientemente fundamentadas matemáticamente; pero han surgido como resultado de la necesidad del modelo formal cuando el paradigma surgió de problemas de implementación. Esto es, las características del modelo orientado a objetos se han definido y se han ido refinando a través del tiempo, resolviendo problemas prácticos sin que hubiera existido la preocupación de establecer una definición formal.

Aunque para al menos los autores de los libros relacionados los Sistemas de Administración de Bases de Datos Orientadas a Objetos, éstos poseen grandes ventajas, como el mayor poder semántico y una representación de la realidad más "simple", no podemos aseverar que la simplicidad del modelo sea tal, ya que para una comprensión cabal del modelo es necesario un conocimiento medio de programación orientada a objetos, de lo contrario lograr la abstracción necesaria para el modelo se hace difícil.

Además, aun cuando las Bases de Datos Orientadas a Objetos ofrecen de acuerdo a la bibliografía y referencias, un mejor rendimiento de la máquina que las Bases de Datos Relacionales para aplicaciones o clases con estructuras complejas de datos, estas Bases de Datos existirán aun durante un buen periodo con las Bases de Datos Relacionales como una forma de estructura de datos dentro de una Base de Datos Orientadas a Objetos.

Queda además, el hecho de los costos de adquisición actuales de las Bases de Datos Orientadas a Objetos, aun son muy elevados en comparación a los productos más comunes de mercado del tipo Bases de Datos Objeto Relacionales.

Por ultimo, debido a la ausencia de estandarización de los Sistemas de Administración de Bases de Datos Orientadas a Objetos y que éstas todavía están en una etapa de estudio, es recomendable el uso de Bases de Datos Orientadas a Objetos sólo en casos especiales, estos podrían ser, cuando no es posible modelar la realidad o se pierde gran significado semántico con el modelo de Base de Datos Relacionales o cuando tenemos grandes volúmenes de información a manejar y el tiempo de respuesta es uno de los factores críticos del sistema.

Las Bases de Datos Relacionales han madurado al punto de lograr trabajar en muchas de las limitantes que se han señalado en sus implementaciones con SQL. Considerando esto, el valor del uso de las Bases de Datos Orientadas a Objetos disminuye considerablemente, de forma especial si se toma en cuenta los avances que representan las Bases de Datos Objeto Relacionales.

Un punto realmente importante es el uso práctico y la aplicación, pues mientras puede forzarse la implementación de una Base de Datos Orientada a Objetos en cualquier modelado, se corren los riesgos de no usar una herramienta relacional debidos a factores como la falta de herramientas de modelado de amplia capacidad en comparación con la amplia variedad existente en el mercado para Base de Datos Objeto Relacionales o Bases de Datos Relacionales.

Al usar una Base de Datos Objeto Relacional, se puede aprovechar los mejores aspectos de ambos modelos mientras se puede acceder a los datos en forma de relaciones u objetos, aunque aquí se vuelve difícil para los diseñadores realizar un modelo que salvaguarde los datos en ambos puntos de vista (no olvidemos que es difícil crear una Base de Datos Orientada a Objetos con un nivel adecuado de normalización desde el punto de vista relacional). Además debemos tomar en cuenta que actualmente una Base de Datos Orientada a Objetos no puede denominarse escalable, pues se ha trabajado sobre todo en implementaciones para aplicaciones de gran volumen.

Por lo que respecta a este último punto, debemos tomar en cuenta que aun cuando una buena parte de los mercados de Bases de Datos lo representan empresas de tamaño considerable, los datos que éstas concentran se encuentran diseminados en Bases de Datos de menor tamaño, o incluso provienen de las Bases de Datos o sistemas de información de organizaciones mas pequeñas, quienes normalmente tienen Sistemas de Administración de Bases de Datos de menores capacidades. Esto considerando la gran cantidad de PYMES que no tienen en nuestro país la oportunidad de pagar grandes desarrollos de sistemas con Bases de Datos de gran capacidad.

Durante varios años, se ha estado previendo una disminución en el uso de Bases de Datos Relacionales y un incremento en las Bases de Datos Orientadas a Objetos, sin embargo, lo que ha sucedido realmente en este último lustro, es el incremento de productos objeto-relacionales, su mejora y difusión, y ante la situación en la industria, que sigue favoreciendo a las Bases de Datos Relacionales, es muy probable que las Bases de Datos Objeto Relacionales serán las más usadas en la siguiente década y que irán mejorando día con día al punto de convivir con lenguajes Orientados a Objetos para brindar las mismas facilidades que las Bases de Datos de dicho paradigma y permitir su trabajo sin modificaciones importantes al desarrollo de los sistemas de información.

Por todo esto se puede concluir que las Bases de Datos Objeto Relacionales son una herramienta para la cual se vislumbra un amplio futuro, siendo empleadas en diversos ámbitos de los sistemas de información gracias a sus ventajas e indudables beneficios en relación costo-beneficio comparadas con Bases de Datos Orientadas a Objetos, permitiéndonos además, como se menciona en la hipótesis, explotar las mismas características del paradigma de la Orientación a Objetos.

ANEXO I

ANEXO I. TABLA COMPARATIVA DE MODELOS DE BASES DE DATOS[14]

CARACTERÍSTICAS	BASES DE DATOS RELACIONALES	BASES DE DATOS ORIENTADAS A OBJETOS	BASES DE DATOS OBJETO RELACIONALES
Implementación de relaciones	Representadas a través de claves primarias.	Las referencias se representan por identificadores, dados por el SABD.	Representadas por claves primarias
Lenguaje de Manipulación	No puede expresar muchos tipos de operaciones.	Se pueden definir múltiples variables colección, ampliando su potencialidad.	Mismas ventajas de las Bases de Datos Orientadas a Objetos
Lenguaje de interacción.	No procedurales.	Basados en procedimientos.	Pueden usarse ambos tipos.
Estándar de definición	SQL2.	ODMG-2.0 (En proceso de estandarización).	SQL3 (En desarrollo).
Paradigma Orientado a Objetos	No lo soporta; es difícil mapear programas de objetos a éstas BD.	Amplio soporte.	Soporte limitado; principalmente orientado a nuevos tipos de datos.
Facilidad de Uso	Fácil de usar.	Correcto para los programadores; algún acceso a SQL para los usuarios finales.	Fácil de usar exceptuando algunas extensiones.
Soporte relaciones complejas	No soporta tipos de datos abstractos.	Soporta una amplia variedad de tipos de datos con relaciones complejas.	Soporta tipos de datos abstractos y relaciones complejas.
Desempeño	Muy buen desempeño.	Relativamente menor desempeño.	Se espera constante mejora.
Madurez de los productos	Relativamente antiguos y por tanto muy maduros.	Relativamente maduro por su poco tiempo en el mercado.	Está en desarrollo, aun no tiene mucha madurez.
Uso de SQL	Extenso soporte.	OQL es similar a SQL, pero con características adicionales como datos complejos y la Orientación a Objetos.	SQL3 con características orientadas a objetos ya incorporadas.
Ventajas	Gracias a SQL, realizar optimización de consultas es relativamente simple dando un buen desempeño.	Puede manipular todo tipo de aplicaciones complejas y reutilización de código, lo que disminuye las líneas de código necesarias.	Habilidad para realizar consultas a aplicaciones simples y complejas.
Desventajas	Inhabilidad para manipular aplicaciones complejas.	Bajo desempeño por poca optimización de consultas y la inhabilidad de soportar sistemas de gran escala.	Bajo desempeño en aplicaciones web.
Soporte de los vendedores	Actualmente el más exitoso, por lo que el mercado es muy grande aunque muchos vendedores se están orientando a BDOR.	Poca presencia en comparación con el mercado relacional.	La mayoría de los vendedores de tecnología relacional están invirtiendo mucho a esta tecnología.

[14] International Data Corporation

GLOSARIO

GLOSARIO DE TÉRMINOS

applet	Programa diseñado para ser ejecutado por otra aplicación.
BDOO	Base de Datos Orientada a Objetos
BDOR	Base de Datos Objeto Relacional
BDR	Base de Datos Relacional
BLOB	Tipo de dato consistente en una cadena de bytes u objetos binarios de gran tamaño almacenados en una base de datos que es reconocida por una aplicación distinta a la Base de Datos.
CAD	Computer Aided Design, Diseño Asistido por Computadora
CAE	Computer Aided Engineering, Ingeniería Asistida por Computadora
CAM	Computer Arded Manufacturing, Manufactura Asistida por Computadora
CASE	Computer Aided System Engineering, Ingeniería de Sistemas Asistida por Computadora
CORBA	Common Object Request Broker Architecture
DW	Data Warehouse, Base de Datos diseñada para soportar Sistemas de Soporte a Decisiones (DSS)
DBA	Database Administrador, Administrador de la Base de Datos
DD	Diccionario de Datos
DER	Diagrama Entidad Relación
JDBC	Java Database Connectivity, API que permite a los programas de Java interactuar con Bases de Datos
LDD ó DDL	Lenguaje de Definición de Datos
LMD ó DML	Lenguaje de Manipulación de Datos
middleware	Software que permite la conexión de dos aplicaciones distintas y separadas, como una Base de Datos y una aplicación en web.
ODL	Lenguaje de Definición de Objetos
ODMG	Object Data Management Group antes Object Database Management Group
OLAP	On Line Analitical Processing, categoría de herramientas de software que permite analizar en tiempo real diversas dimensiones de datos multidimensionales almacenados en una Base de Datos.

OMG	Grupo Manejador de Objetos
OML	Lenguaje de Manipulación de Objetos
OQL	Lenguaje de Consulta de Objetos
RUP	Rational Unified Process
SABD	Sistema Administrador de Bases de Datos
SABDOO	Sistema Administrador de Bases de Datos Orientadas a Objetos
SABDOR	Sistema Administrador de Bases de Datos Objeto Relacionales
SABDR	Sistema Administrador de Bases de Datos Relacionales
SMALLTALK	Lenguaje de Programación Orientado a Objetos desarrollado por Xerox Corporation's
SQL	Lenguaje Estructurado de Consulta
UML	Lenguaje Unificado de Modelado, Unified Modeling Language

BIBLIOGRAFÍA Y REFERENCIAS EN INTERNET

BIBLIOGRAFÍA

DATE, Chris J. *Relational Database Writtings 1991-1994*. Ed. Addison Wesley L.; EUA; 1995; ISBN 0-201-82459-0

FOWLER Martin y **KENDALL** Scott. *UML Gota a gota*. Ed. Addison Wesley Longman; 1999; ISBN 968-444-364-1

GOOS G. y **HARTMANIS** J. Lecture Notes in Computer Science. Ed. ICDT'90 *International Conference on Databases Theory. Third International Conference on Databases Theory*. Paris, Francia; Dic. 1990; ISBN 3-540-53507-1

HUGHES, John G. *Object Oriented Databases*. Ed. Prentice Hall; EUA; 1991; ISBN 0-13-629882-6

PIATTINI, Mario et al. *Diseño de las Bases de Datos Relacionales*. Ed. AlfaOmega; México, 2000; ISBN 970-15-0526-3

SRINIVASAN, B. y **ZELEZNIKOW** J. *Proceedings of the Second Australian Databases-Information Systems Conference*; Ed. World Scientific; Sidney, Australia; 1991; ISBN 981-02-0603-8

STONEBRAKER, Michael. *Readings in Databases Systems*. Ed. Michael Stonebraker, University of California, Berkeley;1988; ISBN 0-934613-65-6

ZDONIK, Stanley B. y **MAIER** David. *Readings in Object-Oriented Database Systems*. Ed. Morgan Kaufmann Publishers, Inc. California, EUA; 1990; ISBN 0-55860-000-0

REFERENCIAS EN INTERNET

AUTOR	TITULO	EDITORIAL	FECHA DE PUBLICACIÓN	UBICACIÓN	FECHA DE LA ÚLTIMA CONSULTA
CERVERA J. y MARCOS E.	Tendencias de la Tecnología de Objetos	NOVATICA 145 Ed. Digital	May-Jun 2000	http://www.ati.es/novatica/2000/145/jcerve-145.pdf	24 Ago 2002
DATE, Chris J.	Moving forward with relational:looking for objects in the relational model. Chris Date finds they were there all the time	DBMS on Line DBMS Interview by David M. Kalman	Oct 1994	http://dbmsmag.com/int9410.html	24 Ago 2002
DATE, Chris J.	Thirty Years of Relational: Relational forever! The Relational model will stand the test of time	DBMS Magazine on Line	Jun 1999	http://dbmsmag.com/int9907.html	24 Ago 2002
DATE, Chris J.	Thirty Years of Relational: Extending the Relational Model When's an extension not an extension?	DBMS Magazine on Line	Jun 1999	http://dbmsmag.com/int9907.html	24 Ago 2002
DEVARAKONDA, Ramakanth S.	Object-Relational Database Systems, The Road Ahead	ACM Crossroads Student Magazine	2 Mzo 2001	http://www.acm.org/crossroads/zrds7-3/ordbms.html	19 Ene 2002
DIMATE, J.	Bases de datos orientadas a objetos	Universidad de Colombia		http://atenea.udistrital.edu.co/profesores/jdimate/basesdedatos1/tema7.html	24 Jun 2001
DORSEY, Paul Dr.	Object-Relational Databases: Their time has come... Almost	Select Magazine	Jul 1998	http://www.dulcian.com/magazine/articles/ObjectRelationalDatabases:Theirtimehascome.htm	19 Ene 2002
DORSEY, Paul Dr.	Taking Advantage of the Object-Relational Paradigm, Today!	Dulcian Inc		http://www.dulcian.com/magazine/articles/TakingAdvantageoftheObject-RelationalParadigm.htm	19 Ene 2002
ESTIVILL Castro Vladimir	Un panorama de las bases de datos orientadas a objetos	Revista Informarte Num. 27 de la SEMARNAP	Abr/May 2000	http://beta.semarnap.gob.mx/informarte/revista/archivo/bdoo.shtml	7 Jun 2001
FUSSELL, Mark L.	Foundations of Object Relational Mapping	ChiMu Corporation	15 Jul 1997	http://www.chimu.com/publications/objectRelational	17 Oct 2001

AUTOR	TÍTULO	EDITORIAL	FECHA DE PUBLICACIÓN	UBICACIÓN	FECHA DE LA ÚLTIMA CONSULTA
GARCÍA, Dr. Alejandro J.	Lógica de Predicados	Universidad Nacional del Sur, Argentina		http://cs.uns.edu.ar/~ajg/SFCC/clase6sfcc.pdf	17 Oct 2001
HAND, Steve y **CHANDLER**, Jane	Introduction to Object-Oriented Databases		3 Sep 1998	http://www.dis.port.ac.uk/~chandler/OOLetures/database/database.htm	17 Oct 2001
HANEY, Clare	PC EXPO: Panelist predict resurgence of object/relational databases	ComputerWorld	23 Jun 1999	http://198.112.59.30/home/news.nsf/all/990623obj	19 Ene 2002
JEPSON, Brian	Object-Oriented Apps in a Relational Database	Web Techniques	Mzo 2000	http://www.webtechniques.com/archives/2000/03/jepson/	19 Ene 2002
KLEIN, Lauren	Stonebraker pitches Object-Relational DBMS	Dr. Dobb's Journal	2001	http://www.ddj.com/print/documentID=12589	19 Ene 2002
LINTHICUM, David S.	Mixing Tuples and Objects Object/Relational Databases are all the rage, but do they really fulfill a need?	DBMS Magazine on Line	Dic 1997	http://dbmsmag.com	30 Jun 2002
LINTHICUM, David S.	Objects meet Data	DBMS Magazine on Line	Sep 1996	http://dbmsmag.com/9609d16.html	18 May 2002
MACIASZECK Leszek A.	Relational versus Object Databases Contention or Coexistence?	Macquarie University Sydney, Australia	Jul 1997	http://www.comp.mq.edu.au/courses/comp866/oovsrel.html	19 Ene 2002
McCLURE, Steve	Object Databases vs. Object-Relational Databases	IDC Bulletin #14821E	Ago 1997	http://www.cai.com/products/jasmine/analyst/idc/14821E.htm	19 Ene 2002
MEDINA, J.M. y **PONS**, M.A. Vila	Prácticas de Modelos Avanzados de Bases de Datos: Oracle como BD Relacional Orientada a Objetos		15 Dic 2001	http://www-etsi2.ugr.es/depar/ccia/mabd/material/practicas/transparonclas/OracleBDOO.pdf	15 Nov 2001
STAJANO, Frank	A gentle introduction to Relational and Object Oriented Databases	ORL Technical Report TR-98-2	May 1998	http://www.orl.co.uk/~fms/	24 Ago 2002
STAJANO, Frank	Object Oriented Databases: An Overview	Olivetti Research Limted	Nov 1995	http://www-ice.eng.cam.ac.uk/~fms27/	24 Ago 2002

AUTOR	TÍTULO	EDITORIAL	FECHA DE PUBLICACIÓN	UBICACIÓN	FECHA DE LA ÚLTIMA CONSULTA
STEVENS, Larry	The Object/Relational Datbase Takes Shape	Uniforum	Nov 1996	http://www.uniforum.org/news/html/publications/ufm/nov96/ordb.html	19 Ene 2002
STODDER, David y KESTELYN Justin	What's next for the Database?	Intelligent Enterprise	9 May 2002	http://www.itelligententerprise.com/020509/508feat1_1.shtml	18 May 2002
	Bases de Datos	Monografias.com		http://www.monografias.com/trabajos5/tipbases.shtml	24 Jun 2001
	Bases de Datos	OSMOSIS Latina	Sep 2000	http://www.osmosislatina.com/aplicaciones/bases_de_datos.htm	7 Ago 2001
	Caracteristicas y Objetivos de las Bases de Datos	Estudios de lingüistica española Vol 19	Ene 2002	http://elies.rediris.es/elies9/4-1-2.htm	24 Jun 2001
	Integrating Object and Relational Technologies	Rational Software	Ene 2002	http://www.rational.com/products/whitepapers/296.jsp	19 Ene 2002
	Integrating Objects with RDBMS	GemStone White Paper	27 Abr 2002	http://www.gemstone.com/products/s/papers_integrate.html	17 Oct 2001
	Lógica de Predicados	ITESM Campus Morelos	Ago 2001	http://www.mor.itesm.mx/~logica/log9808/log_pred.html	11 May 2002
	Los modelos de datos y el modelo objeto relacional	Universidad de Los Andes, Mérida Venezuela		http://sistemas.ing.ula.ve/sistemas/bd/#8	17 Oct 2001
	Next generation UML Design and Development Tool	Embarcadero Technologies	Ene 2001	http://www.embarcadero.co.uk/products/describe/dedatasheet.asp	19 Ene 2002
	Object-Orientes Language: Databases: Mapping Objects to Relations	Cetus Links	3 Ene 2002	http://www.cetus-links.org/oo_db_systems_3.html	19 Ene 2002
	Object Relational Databases	CERN DB Group	Ene 2000	http://wwwinfo.cern.ch/db/aboutdbs/classification/or_model.html	19 Ene 2002

Autor	Título	Editorial	Fecha de Publicación	Ubicación	Fecha de la última consulta
	Object-Relational Mapping	ODBMS Facts	Ene 2000	http://www.odbmsfacts.com/articles/object-relational_mapping.html 19 Ene 2002	17 Oct 2001
	Object Relational Mapping Strategies	Object Matter	Ene 1998	http://www.objectmatter.com/vbsf/docs/maplool/ormapping.html	12 Ene 2002
	Platinum ERwin	Platinum Technology	Ago 1998	http://www.microway.com.au/platinum/erwin.stm	11 May 2002
	Sobre la evolución reciente de las bases de datos	NOVATICA 140 Ed. Digital	Jul-Ago 1999	http://www.ati.es/novatica/1999/149/nv140sum.html	24 Jun 2001
	The Object/Relational Model	Lazy Software	Ene 2001	http://www.lazysoft.com/associativemodel/object_relational_model.htm	19 Ene 2002