UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
CAMPUS ARAGON

ETAPAS EN EL DISEÑO DE DATOS EN LA TRANSFORMACION DE LOS REQUERIMIENTOS DE INFORMACION DE UNA BASE DE DATOS RELACIONAL

TESIS PROFESIONAL

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A N:

JERONIMO VALENTIN GONZALEZ

MARTIN ACOSTA ZAMUDIO

DIRECTOR DE TESIS:
ING. FERNANDO FLORES ZAVALETA

SAN JUAN DE ARAGON, EDO. DE MEXICO. 1996

TESIS

COMPLETA

DEDICATORIAS

No hay manera de agradecer todos los esfuerzos realizados
por los padres, para sacar adelante a sus hijos...
Por lo cual no tengo con que pagarles el haber dejado su vida
entera a una sola causa, en hacer hombres de provecho...
Por lo cual son un ejemplo a seguir e imitar sus desvelos , sus
esfuerzos, su tenacidad, su esmero en las cosas , su lucha
diaria por arrancar un día más a la vida...
Por lo cual les dedico este trabajo...
Gracias Padre..., Gracias Madre ....

Doy gracias a mi pequeño Erick
el cual es la razón de mi vida.
Espero que perdones las horas de juegos ausente.
te dedico este trabajo a ti y tus hermanos por venir

Doy gracias a mi pareja por tener la paciencia
de aguantar las horas de ausencia,
y le dedico este trabajo.

Doy gracias a mis hermanos por su apoyo,
por todo lo que me han dado
y por su gran amistad...

Doy gracias a la amistad, que existe en la vida,
y que permite que dos amigos hayan podido
llevar a buen término éste trabajo.
Gracias..... Martín

Doy gracias a los amigos y compañeros
que nos ayudaron para poder llevar acabo este trabajo
que sin su ayuda no seria posible este trabajo.
Muchas gracias...

J. V. G.

A Nuestra Máxima Casa de Estudios :

Porque nos dio la gran oportunidad de aprender a través de sus profesores y dentro de sus instalaciones los conocimientos de una carrera universitaria.

A Mis Padres :

Por brindarme el apoyo necesario para estudiar una carrera, en especial a mi madre por soportar todos estos años . . .

A Jeronimo :

Por ser un excelente amigo y por el apoyo brindado durante los últimos dos años deseándole sinceramente un exitoso porvenir . . .

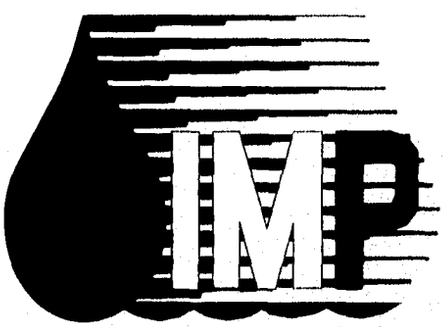
Gracias a todos los seres que se ha compartido conmigo. gracias por su energía, enseñanza y orientación transmitida. gracias por su tiempo y gracias por su espacio. gracias por el asombro y evolución que se ha dado en mi interior.

Martín Acosta Jamudio

AGRADECIMIENTO A LA UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO POR SER PARTE DE NUESTRA FORMACION PROFESIONAL Y POR DARNOS LA OPORTUNIDAD DE SER PARTE DE ELLA.

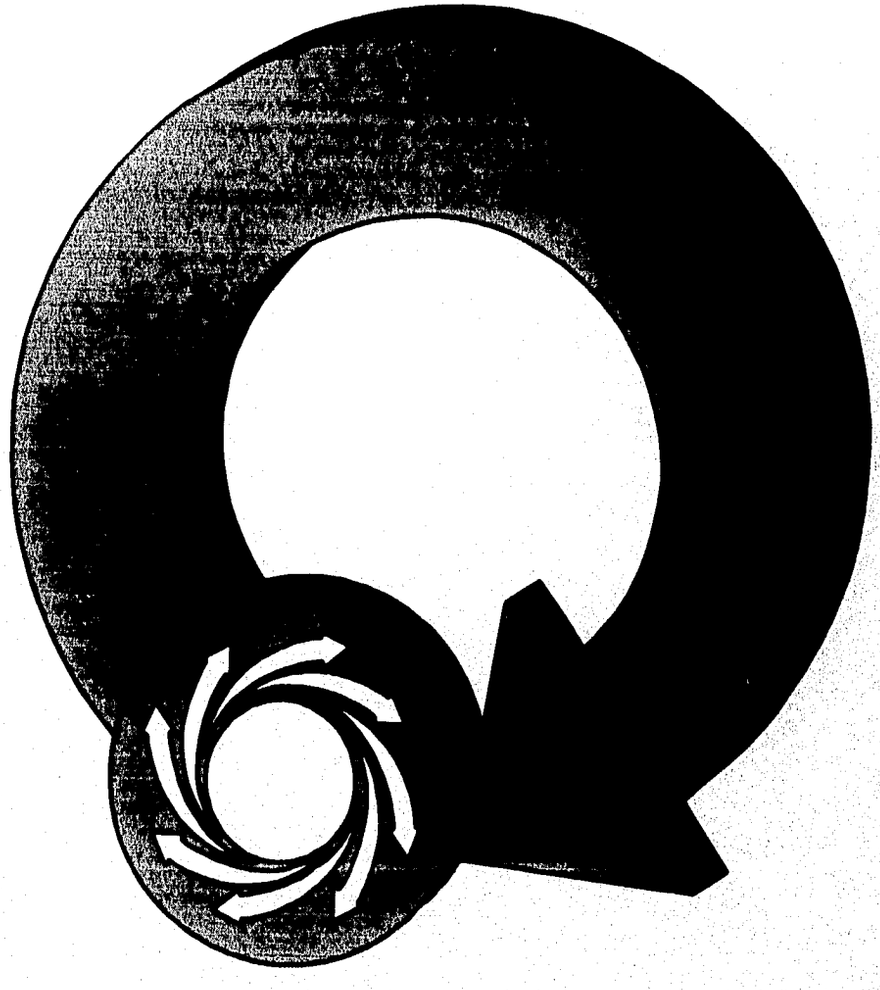
AGRADECIMIENTO A LA ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES CAMPUS-ARAGON POR SER PARTE DE NUESTRA FORMACION PROFESIONAL

AGRADECIMIENTO AL INSTITUTO MEXICANO DEL PETROLEO POR SER PARTE DE MI FORMACION PROFESIONAL Y POR EL APOYO BRINDADO A LA REALIZACION DE ESTE TRABAJO DE TESIS POR MEDIO DE LA DIVISION DE SISTEMAS INFORMATICOS DE LA GERENCIA REPRESENTACION ZONA CENTRO QUE PERTENECE A LA SUBDIRECCION GENERAL DE CAPACITACION Y SERVICIOS TECNICOS.



AGRADECIMIENTO A PETROLEOS MEXICANOS POR EL APOYO BRINDADO A LA REALIZACION DE ESTE TRABAJO DE TESIS A TRAVES DE LA UNIDAD DE APOYO ADMINISTRATIVO DE LA SUBDIRECCION DE SERVICIOS TECNICOS-PEMEX EXPLORACION Y PRODUCCION

INDICE

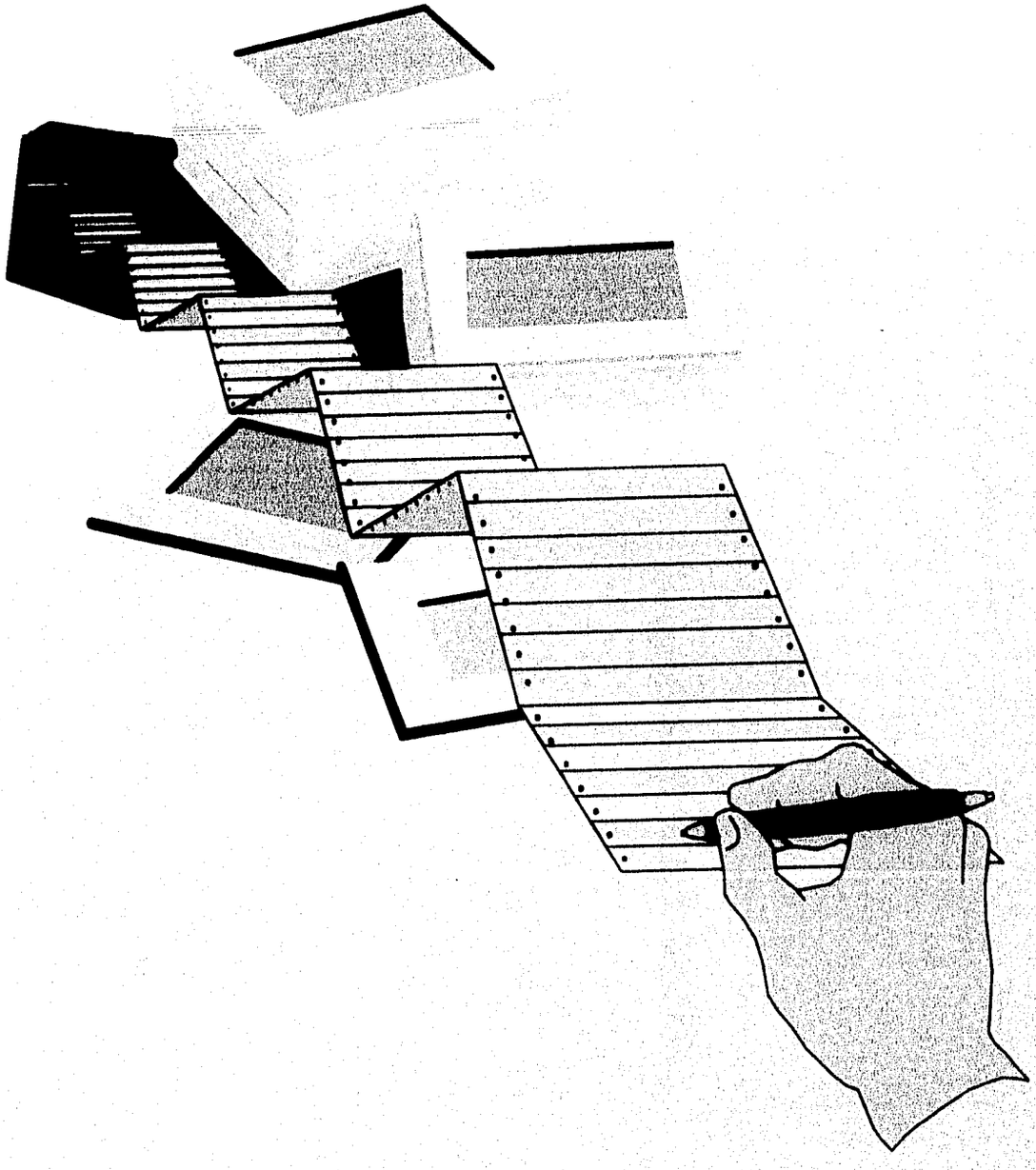


INDICE

CAPITULO 4. CONSTRUCCION DE LA BASE DE DATOS

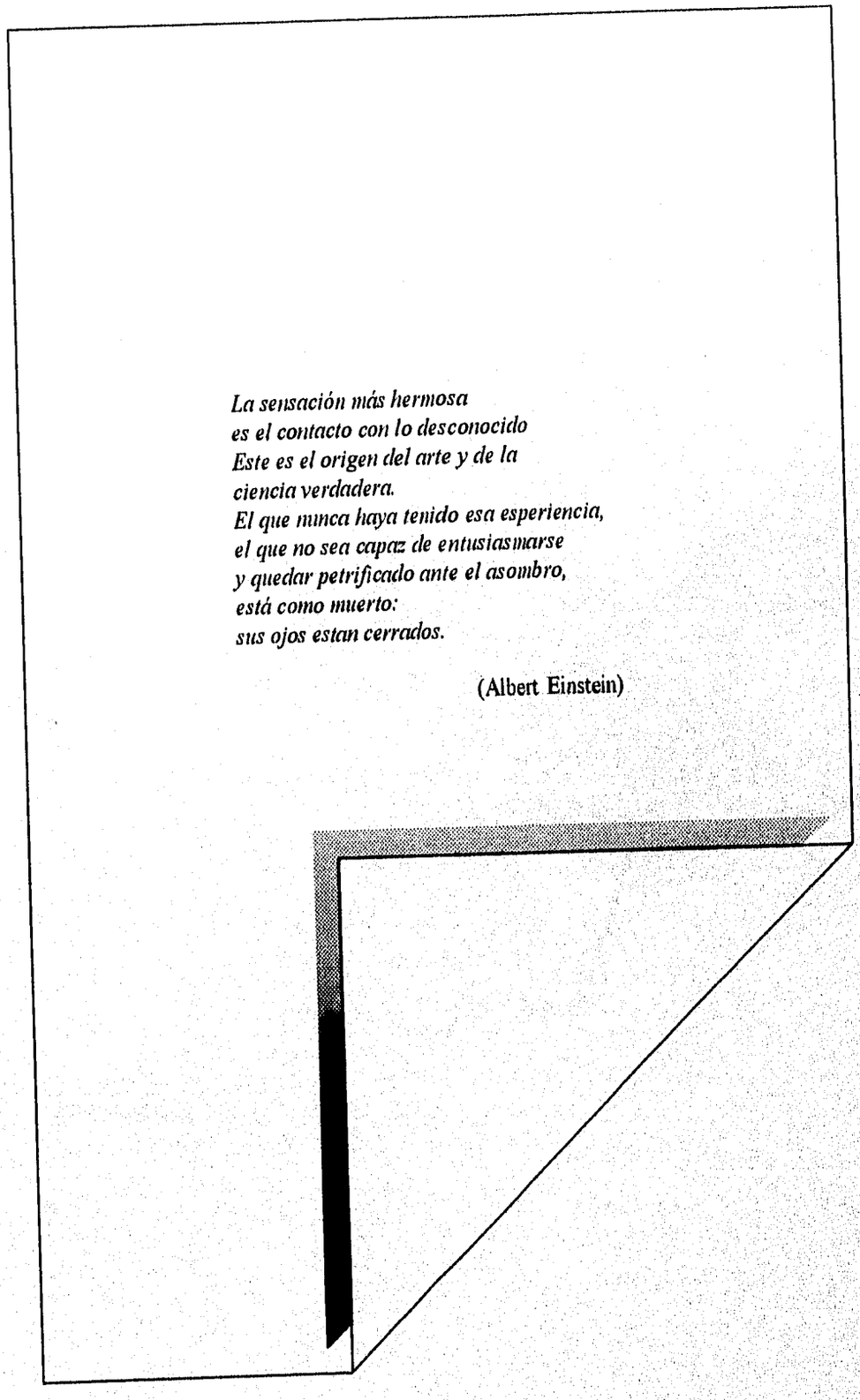
CAPITULO 5. EJERCICIO TEORICO-PRACTICO

INTRODUCCION



La sensación más hermosa
es el contacto con lo desconocido
Este es el origen del arte y de la
ciencia verdadera.
El que nunca haya tenido esa esperiencia,
el que no sea capaz de entusiasmarse
y quedar petrificado ante el asombro,
está como muerto:
sus ojos estan cerrados.

(Albert Einstein)



INTRODUCCION

En 1970, el Dr. E. F. Codd (que trabajaba en IBM), propuso el concepto relacional. Un sistema relacional es aquel que presenta los datos a los usuarios a manera de tablas, donde no existe una relación padre-hijo. Las tablas se forman por renglones (registros) y columnas (campos de datos). Una aplicación accesa los datos a través de estructuras lógicas, llamadas vistas (views). Una vista representa registros de una o varias tablas. El usuario no necesita considerar cómo se almacenan los datos físicamente. Cuando se crea una vista con más de una tabla, debe existir un campo de datos común para relacionar los renglones coincidentes de las tablas participantes. Por ejemplo, los clientes se ligan con las facturas por medio de un número de cliente.

Los primeros productos relacionales para mainframes y minis aparecieron en el mercado a finales de los setenta y principios de los ochenta. Actualmente hay más de cien productos relaciónales que pueden correr sobre cualquier combinación de hardware y sistemas operativos. El concepto relacional no fue aceptado hasta que IBM sacó al mercado el primer DBMS verdaderamente relacional, el DB2 y SQL/DS.

El Lenguaje SQL se introdujo como lenguaje de consulta del sistema "R", el cual fue un proyecto de investigación que se desarrolló en 1974 por IBM. El objetivo del proyecto era demostrar la aplicación práctica del modelo de datos relacional, que en ese entonces se acababa de proponer. El nombre de SQL esta formado por las iniciales en ingles de "Lenguaje de Consulta Estructurado"

(Structured Query Language). Todavía se le conoce con su antiguo nombre, Sequel. El lanzamiento de SQL tuvo un gran impacto en el ambiente. En Mayo de 1986, ANSI (American National Standards Institute) declaro a SQL como el lenguaje estándar para bases de datos relaciónales. SQL por ser un lenguaje no Procedural permite que el usuario solicite a la computadora la información que desea ver y no el como la computadora obtendrá la información. La forma de construir una consulta a la Base de Datos se realiza con base en los comandos de SQL, los cuales son pocos y la facilidad que proporcionan al poder incrementar la complejidad de las consultas que se quieran realizar, hace que la tarea sea sencilla.

El objetivo primordial de este trabajo es el de tratar de presentar el material necesario para poder obtener la definición de las estructuras y así poder almacenar la información en forma clara y eficiente en un Diseño de Datos Relacional, siguiendo para esto el modelo Entidad-Relación como base fundamental en nuestro desarrollo.

Durante las ultimas dos décadas las Bases de Datos sufrieron grandes transformaciones en sus conceptos y en su tecnología. Actualmente; sin embargo parece que en gran parte de la teoría y práctica de los Sistemas de Bases de Datos se han estabilizado; sus conceptos fundamentales al parecer están bien definidos y se conocen más a fondo. Aunque no hay que dudar la posibilidad de que seguirán evolucionando, pero por el momento las Bases de Datos Relaciónales son las que predominan en nuestros días. Debido a la importancia que tiene la información en casi todas las organizaciones, las

Bases de Datos son un recurso sumamente valioso. Esto condujo al desarrollo de un gran número de conceptos y técnicas para manejar los datos en forma eficiente.

Un Sistema de Base de Datos consiste en un conjunto de datos relacionados entre si y un grupo de programas para tener acceso a esos datos; el conjunto de datos se conoce comúnmente como Base de Datos. Esta contiene información acerca de una empresa determinada.

La información siempre ha sido muy importante e indispensable para la toma de decisiones. Por lo cual esta información en nuestros días está cambiando continuamente y su volumen en la actualidad es demasiado alto, por lo tanto es necesario llevar un adecuado control y almacenamiento de esta información que se da en nuestro entorno, y que mejor que tener una Base de Datos bien Diseñada para un control confiable y eficiente de nuestra información requerida. Es por estas razones que en este trabajo de tesis vimos la necesidad de dejar plasmada la importancia de llevar a cabo un buen Diseño de Base de Datos Relacional (Análisis, Diseño e Implantación) que a su vez es muy importante para poder obtener una Base de Datos bien estructurada y confiable para que cumpla con las necesidades y requerimientos de información del Cliente (Usuario final).

El desarrollo de las Bases de Datos es un enfoque Top-Down, que transforma los requerimientos de información en una Base de Datos

Operacional. El desarrollo Top-Down en la Base de Datos comienza con los requerimientos de información del negocio. El siguiente paso es aplicar la metodología del Modelo Entidad-Relación para el Modelo Conceptual de Datos y así poder representar gráficamente el mundo real que nos interesa; y partiendo de este Modelo Conceptual (Modelo Entidad-Relación) podemos realizar el Diseño de Datos, realizando los diferentes procesos de mapeo, es decir, se mapean los requerimientos de información reflejados en el modelo Entidad-Relación a un diseño de Base de Datos Relacional. Por último se aplicarán los comandos de SQL y SQL-PLUS para crear la Base de Datos Relacional físicamente, implementándola de acuerdo al diseño de Base de Datos.

La experiencia que tenemos en el Diseño de Bases de Datos nos hace comprender que es importante llevar acabo un buen Diseño de Datos dentro de las diferentes etapas del desarrollo de los Sistemas de Bases de Datos para que este contenida toda la información requerida por el negocio.

Este trabajo solamente trata lo referente al Modelo Relacional de Bases de Datos sin tocar lo referente a los demás Modelos, como son: el Modelo jerárquico, el Modelo Red, etc, tampoco se hace referencia a las operaciones relaciónales entre tablas.

El primer capítulo describe todo lo referente a los Requerimientos de información, de una forma concisa y breve. Siendo el punto de partida del Diseño de Datos. Esta parte del análisis es importante ya que es esencial de que

exista un alto grado de comunicación entre el diseñador y personal directivo (usuarios) inmersos en la problemática a solucionar, para que posteriormente este análisis de requerimientos permita al ingeniero tener una mayor comprensión del problema y poder representar el dominio de la información en una base de datos. El análisis de requerimientos da al diseñador la representación de la información y las funciones que pueden ser traducidas en datos.

En el capítulo dos analizaremos el Modelo Conceptual de Datos (Modelo Entidad-Relación) de una manera teorica-practica, es decir viendo conceptos y ejemplos de esa teoría. En este capítulo definimos al modelo Entidad-Relación en dos partes Modelo Conceptual Básico y Modelo Conceptual Avanzado. En el primer modelo definimos lo que son Entidades, Relaciones, Atributos y los estándares para su diagramación; en el segundo modelo se analizará la Normalización ¿Qué es? y ¿Como se aplica?, la resolución de relaciones Muchos a Muchos (M:M), Que es un modelo jerárquico de datos, un modelo recursivo, un modelo de roles, un modelo de subtipos, etc.

En el capítulo tres se verá como pasar el modelo Conceptual de Datos (Modelo Entidad-Relación) al Diseño de la Base de Datos. Mostrando paso por paso como se lleva a cabo esta transición del modelo Entidad-Relación a las tablas de la Base de Datos dando ejemplos de cada uno de los pasos siguientes: Mapear las entidades, Mapear Atributos a Columnas, Mapear UIDS a Llaves primarias (PK), Mapear Relaciones para Llaves Foráneas (PF), Opciones de Arco, Subtipos, Normalización y Desnormalización. Definiendo lo que es en

una base de datos una Tupla(Renglón), Columna(atributo), Tabla(Relación), Clave(Llave Primaria (PK)) y Llave Foránea (FK), Vistas, etc.

En el Capítulo cuatro pasaremos del Diseño de la Base de Datos a la Construcción de la Base de Datos. Dando una pequeña semblanza de lo que es SQL (DDL y DML) y SQL*PLUS. Por lo cual es fundamental ver los comandos de SQL y SQL*PLUS para así poder llevar a cabo la Construcción de la Base de Datos y para poder manipular los datos adecuadamente.

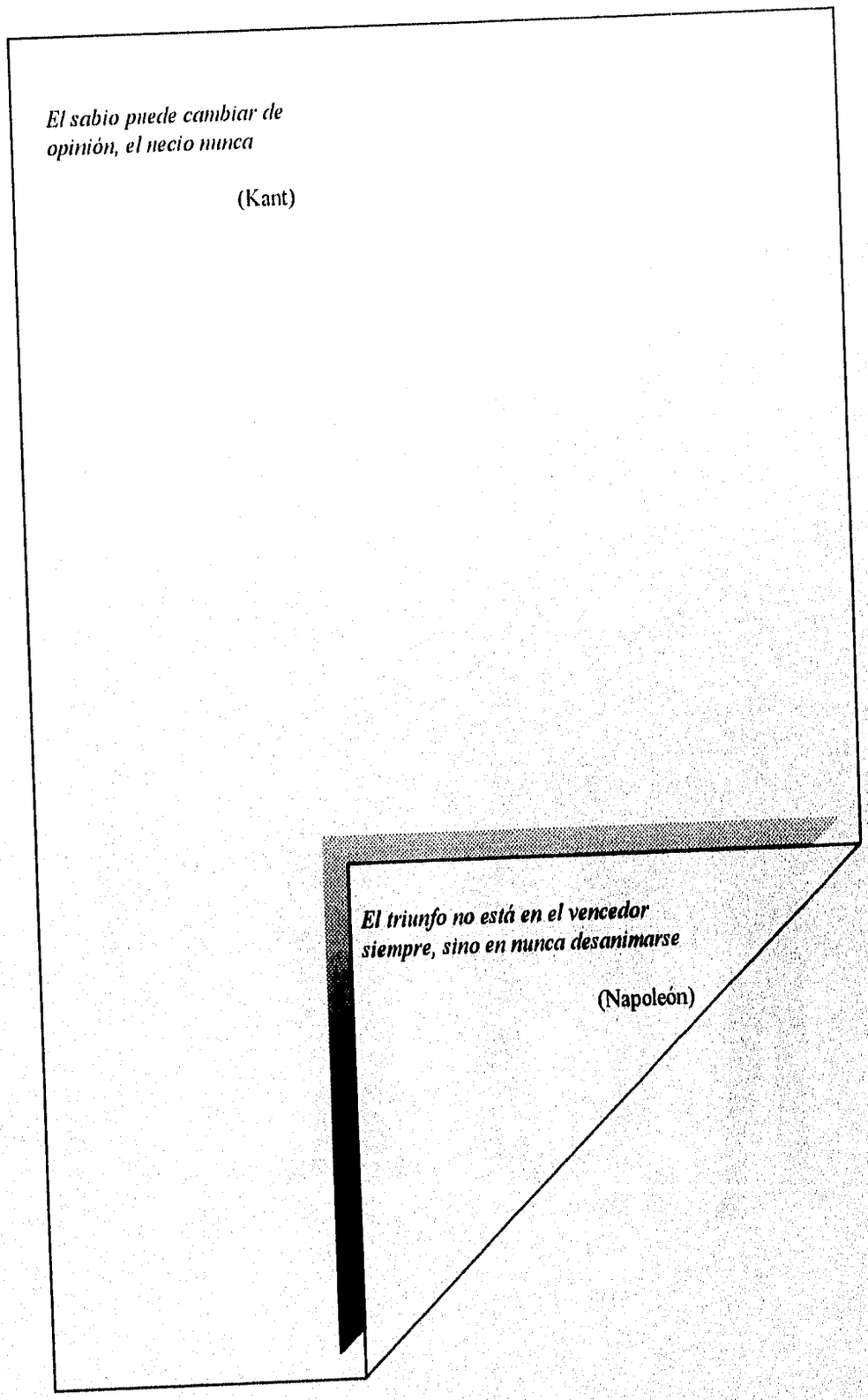
En el Capítulo cinco retomaremos la teoría expuesta en los cuatro capítulos anteriores, iremos paso a paso desde los Requerimientos hasta la Construcción de la Base de Datos, pasando por el Modelo Conceptual y el Diseño de la Base de Datos. Dado que en este capítulo llevaremos a cabo un ejercicio práctico, que comprende un caso real de un Análisis de las Vacaciones de la Subdirección de Servicios Técnicos-Pemex Exploración y Producción.

Por último en este trabajo se tiene un Apéndice el cual esta formado por los anexos "A" ,"B", "C", "D", "E", "F", y "G" en los que se hace mención de algunos puntos importantes del Modelo Entidad-Relación, Tipos de enfoques de Bases de Datos, Operadores Relaciónales e Información breve de Oracle que por alguna u otra razón no se mencionaron en los capítulos desarrollados. También se anexan algunos apoyos que nos sirvieron para la obtención de los requerimientos para realizar el ejercicio practico.

CAPITULO I

El sabio puede cambiar de opinión, el necio nunca

(Kant)



1.1 REQUERIMIENTOS

No todos los sistemas basados en computadora hacen uso de una Base de Datos, pero para aquellos que sí lo hacen este almacenamiento de la información actúa como pivote para todas las funciones del sistema. Muchos grandes sistemas de software necesitan una gran Base de Datos de información. Conforme se ejecuta, el sistema toma información de esta Base de Datos y se la proporciona. En algunos casos, la Base de Datos es independiente del sistema de software (Fig. I.1); en otros, se crea para el sistema en desarrollo. De cualquiér manera, se necesita una definición de la forma lógica de esta Base de Datos.

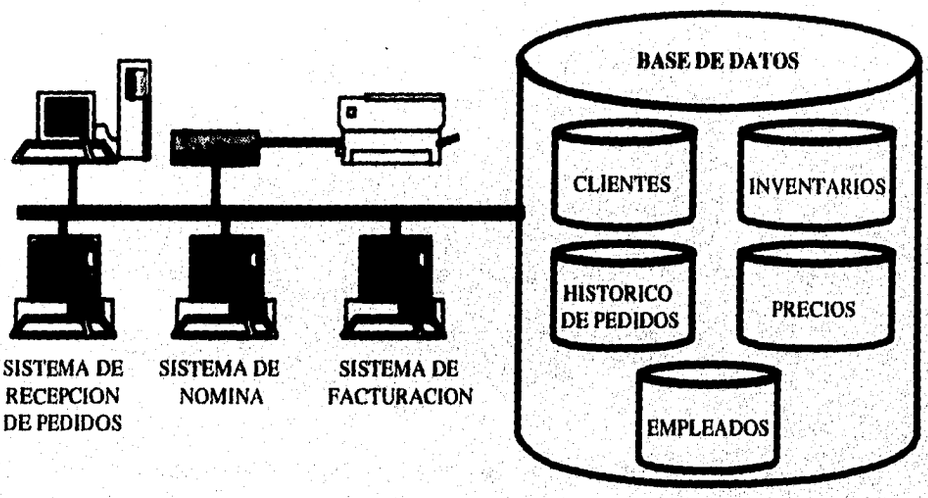


FIG. I.1 Base de Datos Compartida por Diferentes Aplicaciones

Una técnica que se ha empleado para definir la forma lógica de una Base de Datos es utilizar un modelo relacional de datos como el descrito por Codd (1970).

La Ingeniería de Bases de Datos (análisis, diseño e implantación de Bases de Datos) es una disciplina técnica que se aplica una vez que se ha definido el dominio de información. De esta manera, el papel del Ingeniero de Sistemas es: el de analizar la información que va a contener la Base de Datos y que es proporcionada por los usuarios, los tipos de peticiones que se podrán procesar, la manera en que se accesará a los datos y la capacidad que tendra esta.

Estos tópicos de la Ingeniería de las Bases de Datos son llamados en conjunto como **Diseño de Datos** (Fig. I.2).

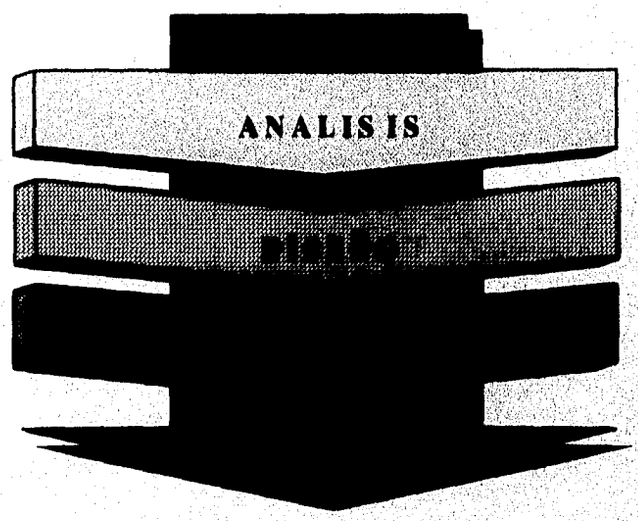


FIG. I.2 Diseño de Datos (Analisis, Diseño e Implantación)

La identificación de las necesidades es el punto de partida en la evolución de la Base de Datos. Pero es importante distinguir entre las necesidades y los requisitos del usuario. Una empresa puede decidir que necesita una Base de

Datos de la información de sus departamentos para apoyar su contabilidad, pero es irreal presentar esta simple necesidad a un ingeniero de software y esperar que desarrolle una Base de Datos aceptable y útil. Más bien, se debe reunir y analizar la información acerca del problema que se va a resolver, y producir una definición completa de éste. A partir de la definición, se puede aplicar el diseño de datos para una posible solución.

También es importante distinguir entre los objetivos y los requisitos de la Base de Datos. En esencia, un requisito es algo que puede probarse, mientras que un objetivo es una característica más general que debe exhibir la Base de Datos.

Para realizar bien el desarrollo del diseño de datos es esencial realizar una especificación completa de los requerimientos de información. Independientemente de lo bien diseñada e implementada que esté, una Base de Datos pobremente especificada no representará todas las necesidades reales de información.

La tarea de análisis de los requerimientos es un proceso de descubrimiento y refinamiento.

Tanto el que desarrolla el diseño de datos como el cliente tienen un papel activo en la especificación de requerimientos. El cliente intenta reformular su concepto, algo nebuloso, de la función y representación de la información en una Base de Datos, en detalles concretos. El que desarrolla el diseño de datos actúa como interrogador, consultor y el que resuelve los problemas.

El análisis y especificación de requerimientos puede parecer una tarea relativamente sencilla, pero las apariencias engañan. Puesto que el contenido de comunicación es muy alto, abundan los cambios por mala interpretación o falta de información (Fig. I.3) .

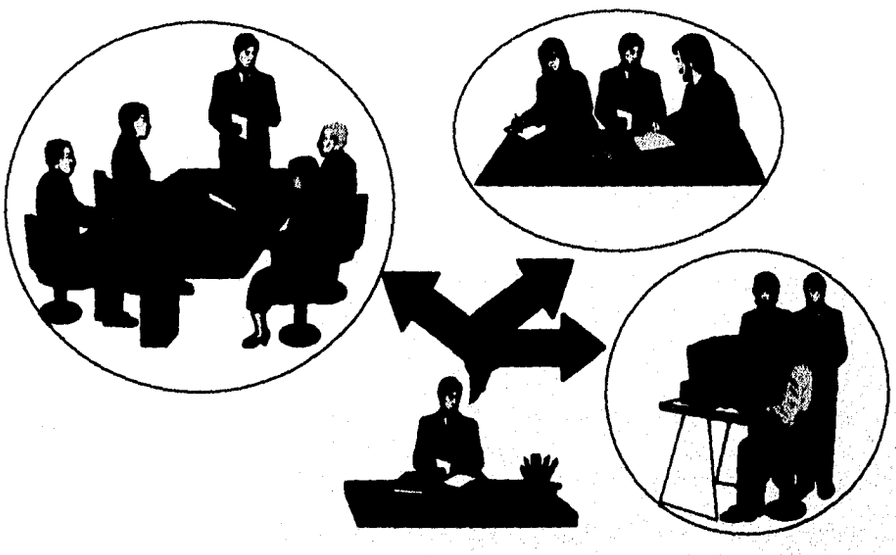


FIG. I.3 La Comunicación es abundante por lo cual no es nada fácil el Análisis de Requerimientos

El análisis de requerimientos permite al ingeniero tener una mayor comprensión del problema y poder representar el dominio de la información que va a contener la Base de Datos, facilita al diseñador la representación de la información y las funciones que pueden ser traducidas en datos. Finalmente, la especificación de requerimientos suministra al técnico y al cliente, los medios para valorar la calidad de la Base de Datos, una vez que se haya construido.

El análisis es una actividad intensiva de comunicación. Cuando existe comunicación, el ruido (mala interpretación) en un camino de comunicación

puede producir dificultad tanto al analista como al cliente. Entre los problemas que pueden encontrarse durante el análisis están: las dificultades asociadas con la adquisición de la información pertinente, el manejo de la complejidad del problema y el acomodar los cambios que puedan ocurrir durante y después del análisis. El reconocimiento del problema, evaluación y síntesis de la solución son necesarias para la adquisición correcta de la información. Frecuentemente la información dada por el cliente no coincide con los requerimientos establecidos anteriormente por otras gentes, Por lo cual es necesario hacerse algunas preguntas tales como:

- ¿ Qué información debe recogerse y cómo debe representarse ?
- ¿ Quién suministra las distintas partes de la información ?
- ¿ Qué herramientas y técnicas están disponibles para facilitar la recolección de información ?

Conforme crece el tamaño del problema, crece también la complejidad de la tarea del análisis. Cada nuevo elemento de información puede tener efecto sobre los otros elementos del diseño de datos. Por esta razón el trabajo del análisis crece geométricamente con la complejidad del problema. Pudiendo surgir entonces interrogantes como las siguientes:

- ¿ Cómo podemos eliminar la inconsistencia cuando se especifican grandes bases de datos ?
- ¿ Es posible detectar las omisiones ?

- ¿ Puede un problema grande ser subdividido con efectividad de forma que se haga más manejable intelectualmente ?

"Independientemente de en qué punto se encuentra el ciclo de vida de la Base de Datos, esta cambiará y el deseo de cambiar persistirá a lo largo del ciclo de vida". Los cambios aludidos en esta sentencia son en los requerimientos y estos ocurrirán. De hecho es frecuente que los cambios se soliciten incluso antes de que se complete la tarea de análisis. Por lo cual es necesario tomar en cuenta lo siguiente:

- ¿ Cómo se coordinan los cambios de otros elementos de la información con los requerimientos de la Base de Datos ?
- ¿ Cómo establecer el impacto de un cambio sobre otras partes, aparentemente no relacionadas, de la Base de Datos ?
- ¿ Cómo corregir los errores en la especificación, de forma que no se generen efectos laterales ?

Existen muchas causas que producen los problemas anotados anteriormente y algunas buenas respuestas para las preguntas establecidas. Los problemas subyacentes al análisis de requerimientos son atribuibles a muchas causas, como por ejemplo:

- Pobre comunicación que hace difícil la adquisición de la información.

- Técnicas y herramientas inadecuadas que producen especificaciones inadecuadas o imprecisas.

- Tendencia a acortar el análisis de requerimientos, conduciendo a un análisis inestable.

- Un fallo en considerar alternativas antes de que se especifique el diseño de datos. Aunque el enfoque de la ingeniería de Base de Datos al análisis de requerimientos no es una panacea, la aplicación de principios fundamentales de análisis y métodos sistemáticos de análisis, reducirá grandemente el impacto de estos problemas.

1.1.1 Principios del Análisis

El análisis de requerimientos es el primer paso técnico en el proceso de la ingeniería del software. Es en este punto cuando se establece de forma general el ámbito del programa y éste se da en una especificación concreta que se convierte en la base de la fase de desarrollo.

Los investigadores han identificado los problemas y sus causas, y desarrollado reglas y procedimientos para resolverlos. Cada método de análisis tiene una única notación y punto de vista. Sin embargo, todos los métodos de análisis están relacionados por un conjunto de principios fundamentales que son:

- El dominio de la información, así como el dominio funcional de un problema debe ser representado y comprendido.

- El problema debe subdividirse de forma que se descubran los detalles de una manera progresiva (o jerárquica).
- Deben desarrollarse las representaciones lógicas y físicas de la información en la Base de Datos.

Aplicando estos principios, el analista enfoca el problema sistemáticamente. Se examina el dominio de la información de tal forma que pueda comprenderse su función completamente. La partición se aplica para reducir la complejidad. La visión lógica y física de la Base de Datos, es necesaria para acomodar las ligaduras lógicas impuestas por los requerimientos de información, y las ligaduras físicas impuestas por otros elementos del software.

1.1.2 Partición

Normalmente los problemas son demasiado grandes y complejos para ser comprendidos como un todo. Por esta razón, tendemos a particionar (dividir) tales problemas en partes que puedan ser fácilmente comprendidas, y establecer interfaces entre las partes, de forma que se realice la función global. Durante el análisis de requerimientos, el dominio funcional y el dominio de la información de la Base de Datos pueden ser particionados.

En esencia la partición descompone un problema en sus partes. Conceptualmente, establecemos una representación jerárquica de la función o información y luego partimos el elemento superior mediante:

- ❑ Incrementando los detalles, moviéndonos verticalmente en la jerarquía.
- ❑ Descomponiendo funcionalmente el problema, moviéndonos horizontalmente en la jerarquía.

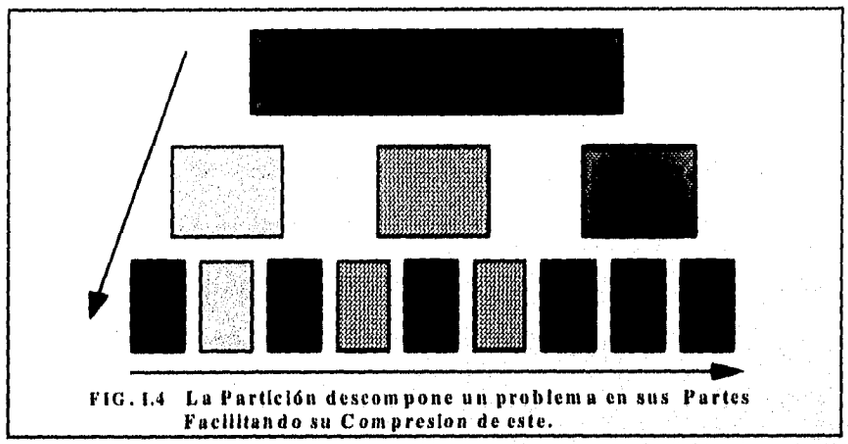


FIG. I.4 La Partición descompone un problema en sus Partes Facilitando su Compresion de este.

Un conjunto de requisitos incompleto o incorrecto puede producir una Base de Datos inconsistente que no satisfaga los requerimientos del cliente. Una especificación inconsistente establece requisitos contradictorios en diferentes partes del documento, pero un requisito ambiguo se presta a distintas interpretaciones de diferentes personas.

Los requisitos deben ser verificables desde dos puntos de vista; primero, debe ser posible comprobar que estos satisfagan las necesidades del cliente y segundo verificar si los productos subsiguientes satisfacen a estos. Debido a la falta de técnica de verificación formal para los requisitos, la herramienta de verificación más importante en la actualidad es el razonamiento lógico y riguroso.

No hay duda de que la forma de especificar tiene mucho que ver con la calidad de la solución. Los ingenieros de software que se han esforzado en trabajar con especificaciones incompletas, inconsistentes o mal establecidas han experimentado la frustración y confusión que invariablemente se produce. Las consecuencias se padecen en la calidad, confiabilidad y completitud de la Base de Datos resultante.

Hemos visto que los requerimientos de la Base de Datos pueden ser analizados de varias formas diferentes. Las técnicas de análisis pueden conducir a una especificación en papel que contenga las descripciones gráficas y el lenguaje natural de los requerimientos de la Base de Datos. Los lenguajes de especificación formal conducen a representaciones formales de los requerimientos que pueden ser verificados o analizados.

1.1.3 Principios de Especificación

La especificación, independientemente del modo en que se realice, puede ser vista como un proceso de representación. Los requerimientos se representan de forma que conduzcan finalmente a una correcta implementación de la Base de Datos.

Las propiedades deseables de una especificación de requisitos para la producción de Base de Datos pueden lograrse y mejorarse por el uso de notaciones formales y herramientas automatizadas.

Hemos visto ya que los requerimientos del diseño de datos pueden especificarse de distintas formas. Sin embargo, cuando estos se llevan al papel (como se hace casi siempre) se deben seguir un conjunto sencillo de criterios:

- El formato de representación y el contenido debe ser adecuado al problema. Puede desarrollarse un perfil general de los contenidos de una Especificación de Requerimientos de la Base de Datos. Sin embargo, las formas de representación contenidas dentro de la especificación, son dependientes normalmente del área de aplicación. Por ejemplo, una especificación para una Base de Datos de una nómina debe usar una simbología diferente al de una especificación para una Base de Datos de facturación.

- Debe anidarse la información contenida dentro de una especificación. Las representaciones deben revelar capas de información, de forma que un lector pueda ir al nivel de detalle requerido. Los esquemas de numeración de párrafos y diagramas deben indicar el nivel de detalle que presentan. Es algunas veces útil presentar la misma información a diferentes niveles de abstracción para ayudar a su compresión.

- Debe restringirse el número de formas notariales y éstas deben ser consistentes en su uso. Una notación confusa o inconsistente, ya sea gráfica o simbólica, degrada la compresión y fomenta los errores.

- Las representaciones deben ser revisables. El contenido de una especificación cambiará. Idealmente, deben estar disponible

herramientas automáticas para actualizar todas la representaciones que puedan ser afectadas por cada cambio. Pragmáticamente, los métodos de papel y lápiz pueden ser eliminados con buenos sistemas de publicación.

Los investigadores han realizado numerosos estudios sobre factores humanos asociados con la especificación (Parecen existir pocas dudas sobre la simbología y la colocación de esta afectan a la compresión). Sin embargo, los ingenieros de software tienen sus propias preferencias sobre determinadas formas simbólicas y diagramáticas específicas. La familiaridad se encuentra frecuentemente en la raíz de las preferencias de las personas, pero otros factores más tangibles, tales como la colocación espacial, patrones fácilmente reconocibles y grado de formalidad, dictan frecuentemente una elección individual.

1.1.4 Revisión de la Especificación

El que desarrolla el diseño de datos y el cliente deben realizar una revisión de la Especificación de Requerimientos de la Base de Datos. Debido a que la especificación constituye el fundamento de la fase de desarrollo, se debe tener un cuidado extremo en realizar esta revisión.

El formato de la revisión puede comprenderse mejor considerando algunas de las preguntas que deben responderse:

- ¿ Los objetivos y fines establecidos para el programa son consistentes con los objetivos y fines de la Base de Datos ?
- ¿ Se ha definido el flujo contenido y estructura de la información de forma adecuada al dominio del problema ?
- ¿ Son los diagramas claros ?
- ¿ Puede cada uno de ellos utilizarse sin el texto suplementario ?
- ¿ Permanecen las funciones principales dentro del ámbito y se describen adecuadamente cada una de ellas
- ¿ Cuál es el riesgo tecnológico del desarrollo ?
- ¿ Se han considerado requerimientos alternativos ?
- ¿ Se han establecido con detalle criterios de validación ?
- ¿ Son adecuados para describir una buena Base de Datos ?
- ¿ Existen inconsistencias, omisiones o redundancias ?
- ¿ Se ha estado en un contacto continuo con el cliente ?

Una vez que se ha completado la revisión, se señala como terminada tanto por el cliente como por el técnico la Especificación de Requerimientos de la Base de Datos. Se producirán cambios en los requerimientos después de que se haya terminado la especificación. Pero el cliente debe tener en cuenta que cada

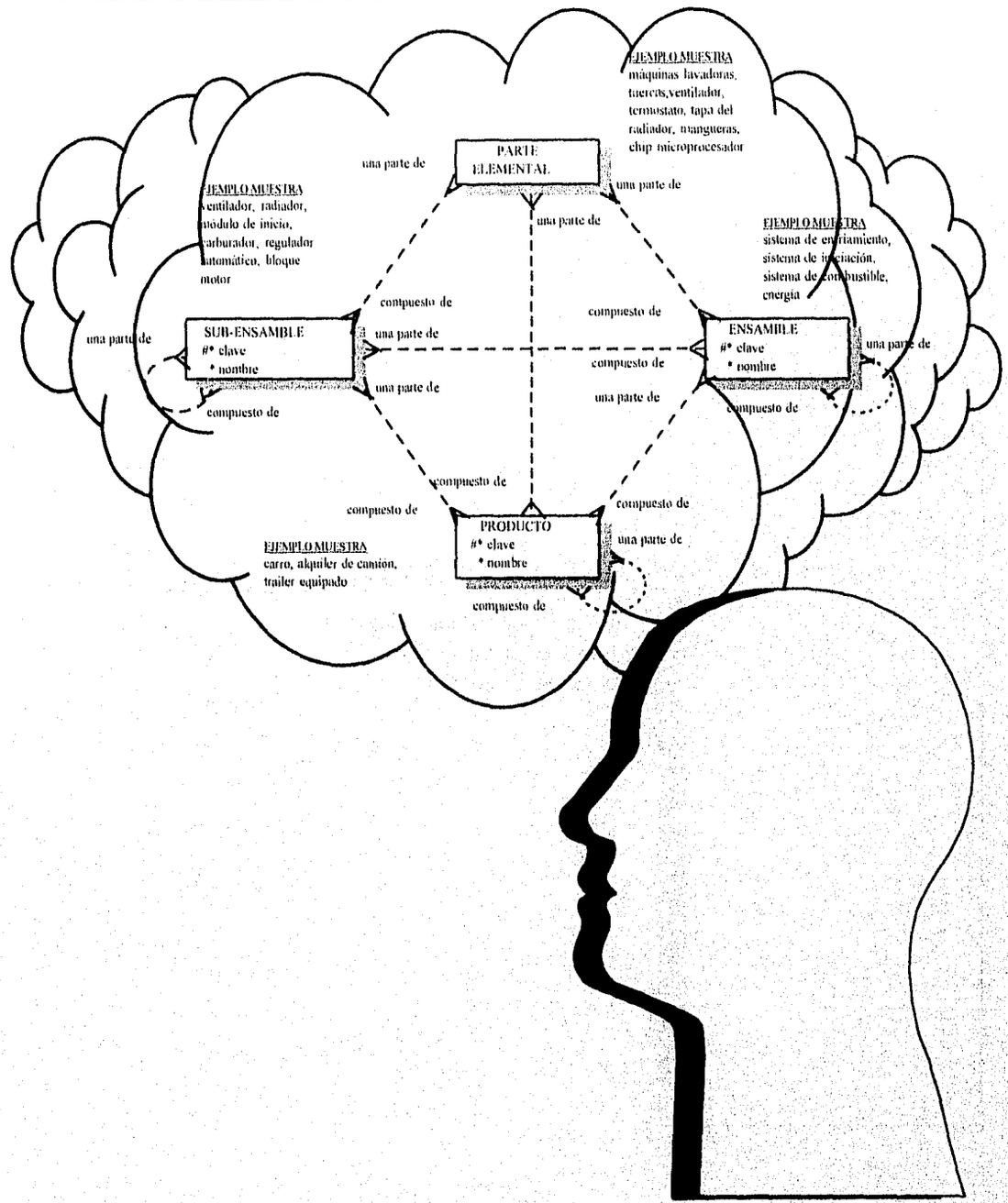
cambio después de dicho hecho, es una extensión del ámbito en el diseño de datos y, por tanto, puede incrementar el costo y/o retrasar la planificación.

Incluso utilizando los mejores procedimientos de revisión, siguen existiendo varios problemas de especificación. Esta es difícil de " probar" de una forma significativa, y por tanto puede que no se noten algunas inconsistencias u omisiones durante la revisión, pueden recomendarse cambios en la especificación. Es extremadamente difícil establecer el impacto global de un cambio; esto es, ¿Cómo afecta un cambio en una función a los requerimientos relativos a otras funciones?; Las herramientas de especificación automática se han desarrollado para ayudar a resolver estos problemas. Esto proporciona flexibilidad máxima a los diseñadores del producto.

En conclusión, los requisitos deben estar anotados en un índice divididos y con referencias cruzadas para facilitar su empleo y su modificación. En teoría cada requisito del producto debe ser rastreable hasta enunciados específicos tanto del cliente, como de la Definición de la Base de Datos; Los cambios ocurrirán, y el éxito de un proyecto con frecuencia depende de la habilidad para incluir cambios sin comenzar de nuevo.

Las propiedades deseables de una Especificación de requisitos para la producción de una Base de Datos pueden lograrse y mejorarse por el uso de notaciones formales y herramientas automatizadas.

CAPITULO II


EJEMPLO MUESTRA
máquinas lavadoras, tuercas,ventilador, termostato, tapa del radiador, mangueras, chip microprocesador
una parte de
PARTE ELEMENTAL
una parte de
una parte de
EJEMPLO MUESTRA
ventilador, radiador, módulo de inicio, carburador, regulador automático, bloque motor
EJEMPLO MUESTRA
sistema de enfriamiento, sistema de ignición, sistema de combustible, energía
compuesto de
compuesto de
una parte de
SUB-ENSAMBLE
#° clave
° nombre
una parte de
ENSAMBLE
#° clave
° nombre
una parte de
una parte de
compuesto de
compuesto de
una parte de
compuesto de
compuesto de
compuesto de
compuesto de
una parte de
EJEMPLO MUESTRA
carro, alquiler de camión, trailer equipado
PRODUCTO
#° clave
° nombre
compuesto de

El saber es para el estudioso y
la riqueza para el cuidadoso,
así como el poder es para el osado
y el cielo para el virtuoso.

(B. Franklin)

Vale más actuar exponiéndose a arrepentirse de ello que arrepentirse de no haber hecho nada.

(Boccaccio)

2.1 PREMISA

Hay que empezar acotando qué parte del mundo exterior nos interesa representar en los datos. Estará formada por todos los objetos y acontecimientos cuyo conocimiento nos permita una mejor gestión de nuestras actividades.

El diseñador, o analista de datos, debe aprender, comprender y conceptualizar este mundo, transformándolo en un conjunto de ideas y definiciones, que firmen una imagen fiel del comportamiento del mundo real. A esta imagen del mundo exterior la llamaremos modelo conceptual. Para construir un buen modelo, el analista debe utilizar una gran dosis de procesos mentales de abstracción, análisis y síntesis. Necesitará además la colaboración de personal directivo, como por ejemplo: Gerentes, Jefes de división, departamento, etc.

El modelo conceptual de datos, también denominado modelo entidad-relación (E-R), es una técnica especial de representación gráfica que incorpora información relativa a los datos y la relación existente entre ellos, para darnos una visión del mundo real. En la actualidad prácticamente todas las metodologías de diseño de sistemas tienen incorporado el modelo entidad-relación (E-R) dentro de su diseño de datos.

Sea una empresa o entidad de cualquier tipo que desea almacenar datos que reflejan información sobre sus actividades puede utilizar este modelo. En la (Fig. II.1) se representan los pasos para conseguirlo.

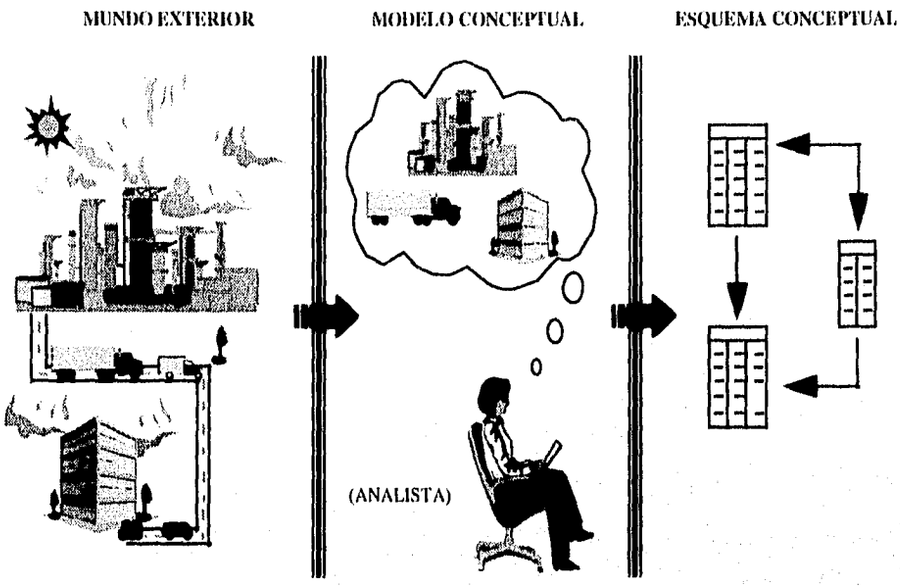


FIG. II.1 Representación Gráfica del Mundo Real

Una vez definido el modelo conceptual, el analista lo transforma en una descripción de datos, atributos y tablas, incluyendo las posibles interrelaciones entre estos elementos y su significado. A esta descripción la llamaremos esquema conceptual de datos.

A la operación de transformar el modelo conceptual en un esquema conceptual la llamaremos diseño lógico de datos (por ello, al esquema conceptual también se le llama a veces esquema de diseño).

Una vez definido el esquema conceptual, hay que traducirlo a estructuras almacenables en soportes físicos controlados por el ordenador,

normalmente discos magnéticos. Esta transformación se suele llamar diseño físico de datos.

Un buen diseño lógico debe producir un esquema conceptual que sea una imagen fiel y completa del modelo conceptual, incluyendo algunas interrelaciones y condiciones semánticas, es decir, aquellas que son consecuencia del significado de los datos.

Es un paso previo al futuro diseño de las bases de datos en el modelo de datos que nosotros designemos: relacional, jerárquico o red. Vale la pena mencionar que éste análisis será realizado sobre el modelo relacional, debido a que en la actualidad este modelo proporciona la mejor herramienta para el diseño de bases de datos.

2.1.1 Características del Modelo Entidad-Relación (E-R)

Estas características se mencionan a continuación :

- Reflejan tan sólo la existencia de los datos, no lo que se hace con ellos.
- Se incluyen todos los datos del sistema en estudio y, por tanto no está orientado a aplicaciones particulares.
- Es independiente de las bases de datos y sistemas operativos concretos.

- ❑ No se tienen en cuenta restricciones de espacio, almacenamiento, ni tiempo de ejecución.

- ❑ Está abierto a la evolución del sistema.

Por tanto, el modelo entidad-relación da una visión del mundo real con la mayor naturalidad mediante los objetos y sus relaciones. De tal forma, que a partir de ahí, su implementación permite mantener las propiedades de las bases de datos.

Como ya se apuntó anteriormente, la representación de datos implica la construcción de un modelo que pueda reflejar el mundo de la realidad dentro del mundo de la máquina a través de nuestras ideas y de los conocimientos del mundo de la información.

El Modelo E-R, por tanto, se basa en la percepción de un mundo real que consiste en un conjunto de objetos básicos denominados Entidades, así como las Relaciones existentes entre ellos.

2.2 MODELO BASICO

El modelo basico comprende lo que son las entidades, relaciones y atributos, a continuación se explicaran cada una de ellas.

2.2.1 Entidades

Una entidad es un objeto concreto o abstracto del cual se necesita tener o conocer información que va a ser representado en un sistema de Bases de

Datos. Cuando se crea el modelo entidad-relación se definen las entidades que forman el mundo de la situación real que tenemos que implementar en una base de datos. Por ejemplo: el objeto DEPARTAMENTO, EQUIPO y FACTURA son entidades. Otras definiciones que podemos mencionar para una entidad son:

- ❑ Un objeto de interés para los negocios
- ❑ Una entidad es una clase o categoría de cosas
- ❑ Una entidad es una cosa con nombre

El objeto CLIENTE y EMPLEADO pueden ser otros aspectos importantes acerca de las necesidades de información de una empresa por lo tanto son entidades.

2.2.1.1 Estándares para Diagramación de Entidades.

Los puntos que a continuación se elistan son los estándares de uso mas común :

- ❑ Cajas de cualquier dimensión.
- ❑ Un nombre único para cada entidad.
- ❑ Nombre de la entidad en mayúsculas y se escribe en la parte superior dentro de la caja
- ❑ Nombre de sinónimo opcional (entre paréntesis)
- ❑ Nombre de los atributos en minúsculas

Un sinónimo es un nombre alternativo para una entidad, por lo cual son útiles cuando dos grupos de usuarios tienen diferentes nombres para el mismo aspecto importante. En la (Fig. II.2) se pueden observar algunos ejemplos:

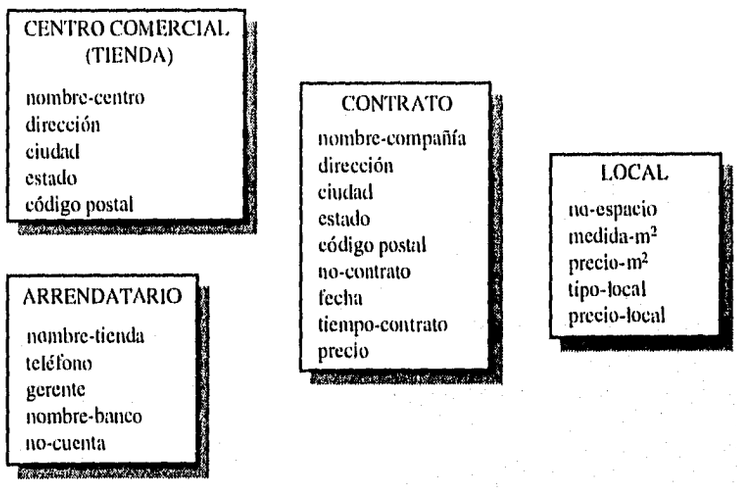


FIG. II.2 Representación de Entidades

2.2.1.2 Instancia de una Entidad

Una instancia o ocurrencia particular o individual es un suceso. Cosa que sucede especialmente cuando es de alguna importancia, por lo cual una entidad debe tener múltiples instancias.

Con lo mencionado anteriormente podemos decir que:

1. La entidad EMPLEADO tiene una ocurrencia para cada EMPLEADO de la compañía.

Javier Hernández, Erika Cervantes, Víctor torres y Antonio Souza son todas la ocurrencias de la entidad EMPLEADO.

2. La entidad DEPARTAMENTO tiene una ocurrencia para cada DEPARTAMENTO en la compañía:

   El departamento de Ventas, el departamento de Desarrollo y el departamento de Finanzas son todas las instancias de la entidad DEPARTAMENTO.

Cada instancia de la entidad tiene valores específicos para cada atributo de la entidad.

Así, podemos decir:

La entidad EMPLEADO tiene los atributos nombre, clave, fecha de ingreso y departamento.

En el caso de Javier Hernández tiene los siguientes valores: nombre *Javier Hernández*, clave *13023*, fecha de nacimiento *16-OCT-60*, y salario de *$5,000.*

En algunas ocasiones las instancias son erróneas para las entidades. Cada instancia debe ser identificada como única de otras instancias de la misma entidad. Un atributo o conjunto de atributos que identifican de manera única a una instancia dentro de una entidad son llamados Identificadores Unicos (UID), si una entidad no puede tener un identificador único (UID) ésta no puede ser una entidad.

Los atributos que identifican de manera única a una entidad y pertenecen a los UID de las entidades son precedidos por (# *). Se verán posteriormente algunos ejemplos.

2.2.1.3 Identificar y Modelar Entidades

Mediante los siguiente pasos se puede llegar a identificar y modelar las entidades de un conjunto de notas de entrevistas :

a) Examinar los sustantivos. ¿Son aspectos importantes?

b) Poner un nombre a cada entidad.

c) ¿Existe información de interés para la empresa acerca de la entidad?

d) ¿Cada instancia de la entidad es identificable de manera única? ¿Cuál o cuáles atributos sirven como UID?

e) Escribir la descripción de la entidad. "Un EMPLEADO tiene como significado ser un empleado pagado por la compañía. Así podemos decir que: Javier Hernández y Verónica Garrido son empleados".

f) Diagramar cada entidad y sus atributos.

Es importante no descalificar tan rápido una entidad candidata ya que posteriormente atributos adicionales para la compañía pueden no ser cubiertos.

2.2.2 Relaciones

El segundo paso para el modelo de datos relacional es identificar las relaciónes entre las entidades. Una relación es bidireccional y es un hecho o una asociación entre dos entidades o entre una entidad consigo misma. Algunas veces es difícil identificar las relaciones, es muy importante entender cuales son las necesidades del usuario.

Poner un verbo o una proposición a la conexión entre dos relaciones implicadas puede ayudar a entender la asociación entre las dos entidades. Para poder obtener una descripción final hay que hacer muchas representaciones.

En la (Fig. II.3) se identifican las relaciones entre las entidades de un Sistema de Administracción de Centros Comerciales. El sujeto se localiza en la parte izquierda y el objeto en la parte derecha. Un simple verbo o una proposición localizada entre las entidades identifica las relaciones, el verbo o proposición localizado en la parte izquierda describe la relación del sujeto con el objeto y a la inversa el objeto con el sujeto.

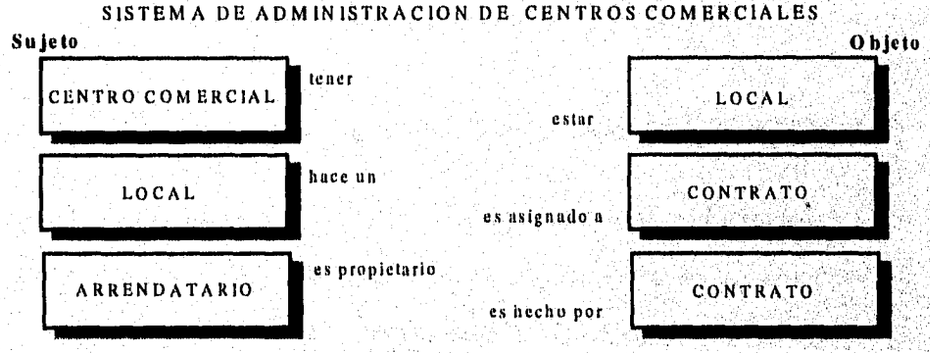


FIG. II.3 Identificación de Relaciones

Sintaxis de una relación.

Con la sintaxis de relación podemos decir que:

La relación entre CENTRO COMERCIAL y LOCAL es:

Cada centro comercial puede tener uno o más locales.

Cada local debe estar en uno y solamente un centro comercial.

Cada dirección (de izquierda a derecha y derecha a izquierda) de una relación tiene:

- ❑ Un nombre *tener* o *estar*
- ❑ Una opción *puede* o *debe*
- ❑ Un grado *uno o más*, o *uno y solamente uno*

2.2.2.1 Estándares de Diagramación de Relaciones.

Existen varias estilos de diagramar relaciones (ver anexo B), el estilo que se seleccione depende del diseñador o analista de datos. En este trabajo se manejaran los estándares que a continuación se listan :

- ❑ Una línea entre dos entidades.
- ❑ Nombre de relaciones en minúsculas.
- ❑ Opcionalidad.
  - ♦ Opcional (*Puede ser*) ----------
  - ♦ Obligatoria (*Debe ser*) ——————
- ❑ Grado
  - ♦ Una o más 
  - ♦ Una y solamente una ——

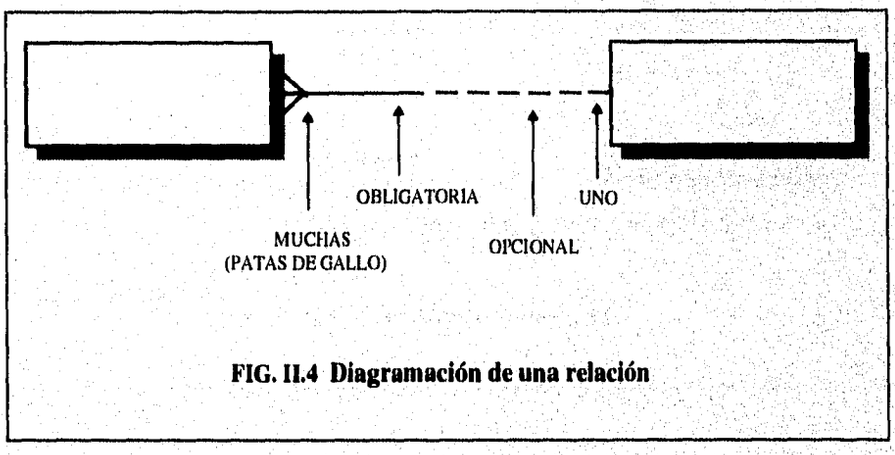


FIG. II.4 Diagramación de una relación

En la (Fig. II.4) se puede observar la representación de una relación en un diagrama. Primero leer la relación en una dirección y después leer la relación en la otra dirección.

En la (Fig. II.5) observamos la relación entre la entidad video y la entidad cliente

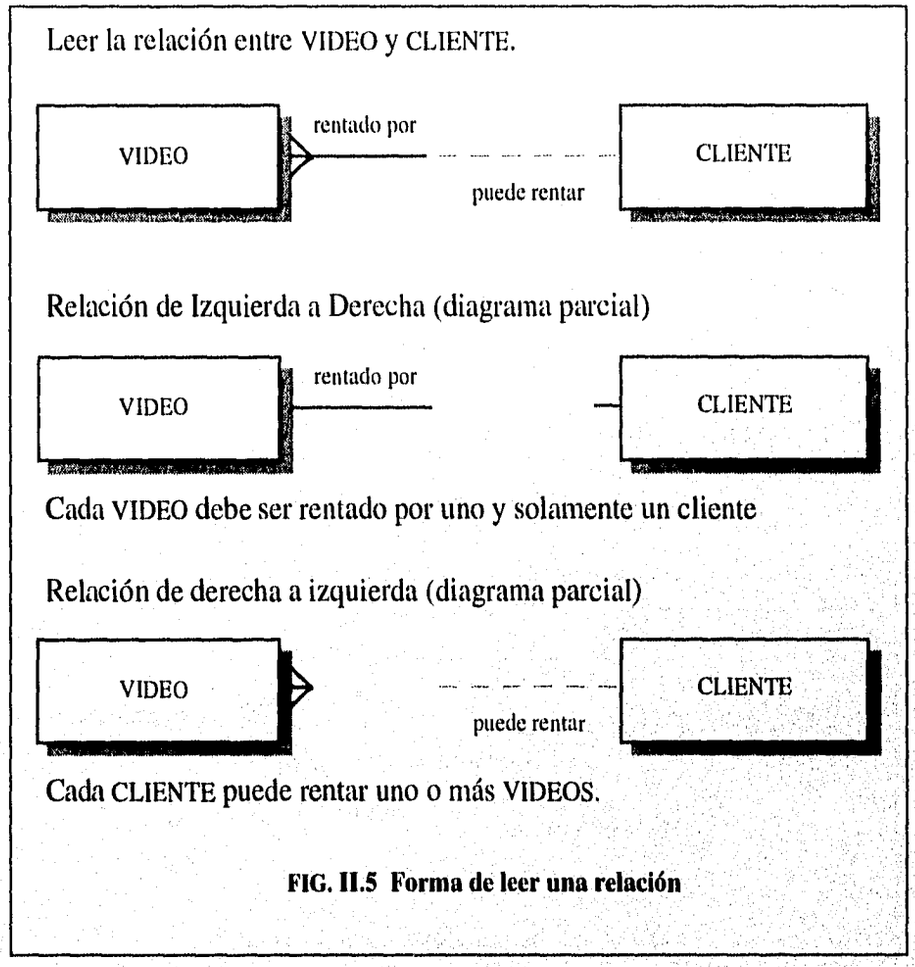


FIG. II.5 Forma de leer una relación

Es importante mencionar que todas las relaciones deben de representar los requerimientos de información y reglas del negocio.

2.2.2.2 Definición de Relaciones

Se describen en términos de:

a) Conectividad

b) Cardinalidad

c) Existencia

Una relación entre entidades es descrita en términos de conectividad, cardinalidad y existencia. La representación más común es la conectividad entre entidades. Estos términos auxilian a definir las necesidades del negocio de la empresa.

El modelo E-R aproxima los datos dados por la sintaxis del diagrama para poder representar su conectividad, cardinalidad y existencia.

a) Conectividad

La conectividad describe el número de instancias de una entidad.

Tipos de conectividad

a') Uno a uno (1:1)

b') Muchos a Uno (M:1)

c') Muchos a Muchos (M:M)

El tipo de conectividad más común es Muchos a Uno. Para ayudar a identificar el tipo de conectividad, pensar en la conectividad en términos de instancias de una entidad.

a') Conectividad Uno a Uno.

Una relación uno a uno (1 a 1 o 1:1) tiene un grado de *uno y solamente uno* en ambas direcciones.

En la (Fig. II.6) se representa una relación 1:1 entre COMPUTADORA y TARJETA DE VIDEO.

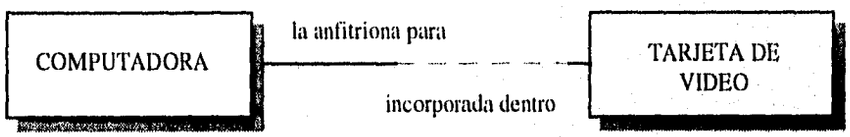


FIG. II.6 Relación Uno a Uno

Cada COMPUTADORA debe de ser la anfitriona para una y solamente una TARJETA DE VIDEO.

Cada TARJETA DE VIDEO puede ser incorporada dentro de una y solamente una COMPUTADORA

Son muy raras las relaciones 1:1. Una relación 1:1 que es obligatoria en ambas direcciones es muy poco común. Las entidades que parecen tener una relación 1:1 pueden ser en realidad la misma entidad.

b') Conectividad Muchos a Uno.

Una relación Muchos a Uno (M a 1 o M:1) tiene el grado de *uno o más* en una dirección y el grado de *uno y solamente uno* en la otra dirección.

En la (Fig. II.7) se representa una relación de M:1 entre CLIENTE y REPRESENTANTE DE VENTAS.

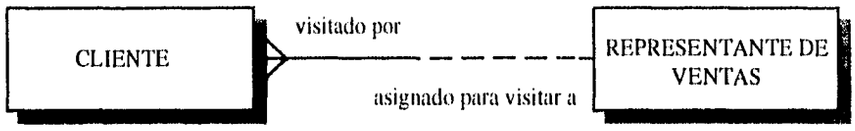


FIG. II.7 Relación Muchos a Uno

Cada CLIENTE debe ser visitado por uno y solamente un REPRESENTANTE DE VENTAS.

Cada REPRESENTANTE DE VENTAS puede estar asignado para visitar uno o más CLIENTES.

Las relaciones M:1 son las más comunes. Las relaciones M:1 que son obligatorias en ambas direcciones son las más raras.

c') Conectividad Muchos a Muchos

Una Relación Muchos a Muchos (M a M o M:M) tiene el grado de *uno o más* en ambas direcciones.

En la (Fig. II.8) se representa una relación M:M entre ESTUDIANTE y CURSOS.

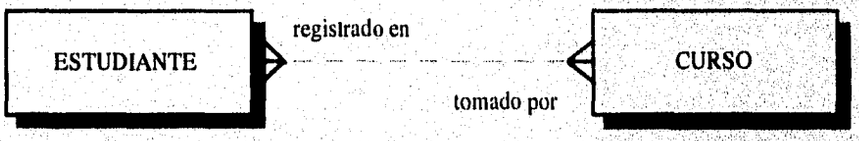


FIG. II.8 Relación Muchos a Muchos

Cada ESTUDIANTE puede estar registrado en uno o más CURSOS.

Cada CURSO puede ser tomado por uno o más ESTUDIANTES.

Las relaciones muchos a muchos son muy comunes y usualmente son opcionales en ambas direcciones o puede ser opcional en una sola dirección.

En la (Fig. II.9) se puede observar la conectividad en un pequeño sistema. Para determinar la conectividad se debe leer de izquierda a derecha y de derecha a izquierda.

Para determinar la conectividad correcta en una combinación de relaciones se pueden aplicar las siguientes reglas:

- ❑ Dos conectividades (1:1) = Una conectividad (1:1)
- ❑ Dos conectividades (M:1) = Una conectividad (M:M)
- ❑ Una conectividad (1:1) y Una conectividad (M:1) = Una conectividad (M:1)

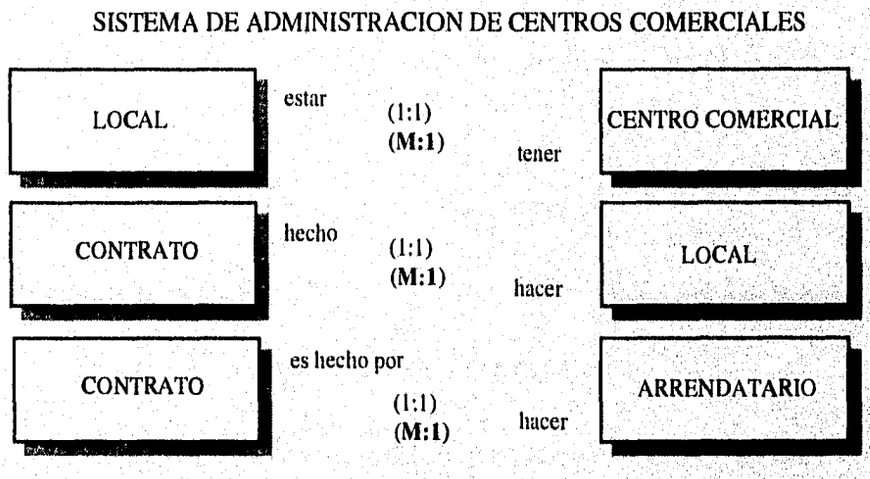


FIG. II.9 Conectividad

b) Cardinalidad o Grado del Término

La cardinalidad define las restricciones sobre el número de instancias en una relación de entidadades.

¿Existe una cardinalidad que restringe la entidad renta con la entidad video?. Una renta consiste de un máximo de 5 videos

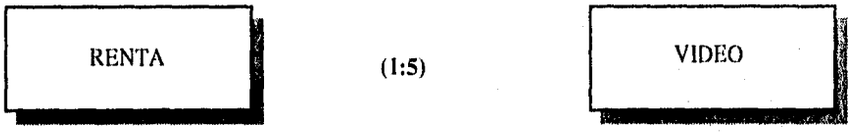


FIG. II.10 Una renta consite de un máximo de 5 videos (Cardinalidad)

Es importante identificar si existe alguna cardinalidad que defina las restricciones en el diseño de una base de datos, estas restricciones deben de ser consideradas en una aplicación.

En el diagrama E-R debe estar bien definido para que sea un instrumento de comunicación para el desarrollo de la aplicación.

c) Existencia

La dependencia de existencia describe si la existencia de una entidad en una relación es obligatoria u opcional. Hay muchas circunstancias donde la existencia de ambas entidades en una relación es requerida para que la relación exista.

El analista debe decidir la conectividad de cada entidad en una relación y determinar si cada relación es obligatoria u opcional. El modelo de información entidad-relación estipula la sintaxis esquemática para indicar la

dependencia de existencia, esta es una consideración muy importante para el diseño físico de la base de datos y el desarrollo de aplicaciones. Existen dos tipos de existencia las cuales son:

- Obligatoria.- Una instancia de la entidad debe existir siempre en la relación.
- Opcional.- Una instancia de la entidad no tiene que existir en la relación.

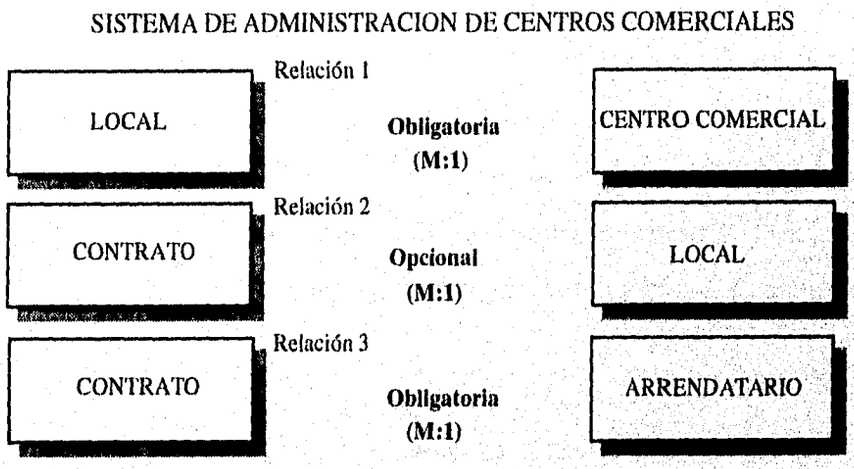


FIG. II.11 Dependencia de existencia

A continuación se explican las razones de el por que se le asigno a cada relación el nombre dependencia de existencia que tienen:

Relación 1.- Un centro comercial no puede existir sin un espacio asociado. La idea es que si un centro comercial existe, tiene que contener locales.

Relación 2.- En un tiempo dado, un local puede existir sin un contrato asociado. El contrato es el que manipula un espacio que no sea ocupado.

Relación 3.- Un arrendatario no puede existir sin un contrato asociado. El usuario no está interesado en inquilinos anteriores.

2.2.2.3 Uso de una Matriz de Relaciones

Se usa una Matriz de Relaciones como una ayuda para la colección inicial de información sobre las relaciones entre una serie de entidades.

Estándares de Matriz de Relaciones :

- Una matriz de relaciones muestra si están relacionadas y en qué forma cada entidad (renglón) con cada entidad (columna) mostrada en la matriz.
- Todas las entidades están listadas en el lado izquierdo y en la parte superior de la matriz.
- Si una entidad-renglón esta relacionada con una entidad columna entonces el nombre de esta relación se muestra en la caja de intersección.

- ❑ Si una entidad-renglón no está relacionada con una entidad-columna,entonces se muestra una línea en la caja de intersección.
- ❑ Cada relación por encima de la diagonal es el inverso o imagen espejo de la relación por debajo de la línea diagonal.
- ❑ Las relaciones recursivas (una entidad consigo misma) son representadas por las cajas en la diagonal.

La siguiente matriz de relaciones muestra una serie de relaciones entre cuatro entidades.

	CLIENTE	ARTICULO	ORDEN	BODEGA
CLIENTE			Generador de	
ARTICULO			Comprado a través de	Guardado en
ORDEN	Generada por	Generada para		
BODEGA		Contenedor de		

FIG. II.12 Matriz de Relación, muestra las relaciones entre cuatro entidades

El CLIENTE está relacionado a una ORDEN y el nombre de la relación es *generador de.*

La ORDEN está relacionada al CLIENTE y el nombre de la relación es *generada por.*

Una vez definida la matriz de relación se debe dibujar el diagrama E-R

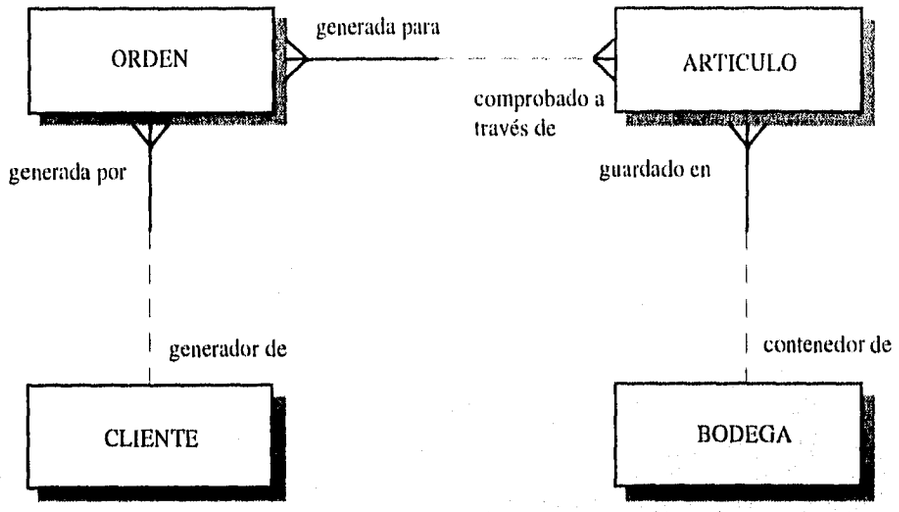


FIG. II.13 Diagrama E-R de la Matriz de Relación de la fig. II.12

2.2.2.4 Analizar y Modelar Relaciones

Existen 5 pasos para analizar y modelar relaciones :

1) Determinar si existe una relación.

2) Nombrar cada dirección de la relación.

3) Determinar la opcionalidad de cada dirección de la relación.

4) Determinar el tipo de cada dirección de la relación.

5) Leer en voz alta las relaciones para validarlas.

Paso 1 Determinar la Existencia de una Relación

Examinar cada par de entidades para determinar si existe una relación. Cuestionar si existe una relación.

- ¿ Existe una relación significativa entre la ENTIDAD A y la ENTIDAD B?

Considerar las entidades DEPARTAMENTO y EMPLEADO.

¿Existe una relación significativa entre DEPARTAMENTO y EMPLEADO?
Sí, existe una relación significativa entre DEPARTAMENTO y EMPLEADO

Considerar las entidades DEPARTAMENTO y ACTIVIDAD.

¿Existe una relación significativa entre DEPARTAMENTO y ACTIVIDAD?
No, no existe una relación significativa entre DEPARTAMENTO y ACTIVIDAD.

Usar una matriz de relación para examinar sistemáticamente cada par de entidades.

En la siguiente matriz de relación se muestra las relaciones entre DEPARTAMENTO, ACTIVIDAD, y EMPLEADO. Las palomas indican las relaciones existentes.

	ACTIVIDAD	DEPARTAMENTO	EMPLEADO
ACTIVIDAD	———	———	✓
DEPARTAMENTO	———	———	✓
EMPLEADO	✓	✓	———

Paso 2 Nombrar Cada Relación

FIG. II.1 Matriz de Relación, muestra la relación entre tres entidades

Nombrar cada dirección de la relación. Cuestionar el nombre adecuado para la relación.

- ¿Cómo está relacionada la ENTIDAD A con la ENTIDAD B?
  Una ENTIDAD A es (*nombre de la relación*) de una ENTIDAD B.

- ¿Cómo está relacionada la ENTIDAD B con la ENTIDAD A?
  Una ENTIDAD B es (*nombre de la relación*) de una ENTIDAD A.

Considerar las relaciones entre DEPARTAMENTO y EMPLEADO.

¿Cómo está relacionado un DEPARTAMENTO con un EMPLEADO?
Cada DEPARTAMENTO es *responsable de* un EMPLEADO.

¿Cómo está relacionado un EMPLEADO a un DEPARTAMENTO?
Cada EMPLEADO es *asignado a* un DEPARTAMENTO.

Opcionalmente, registrar el nombre de las relaciones dentro de una matriz.

En la siguiente matriz de relaciones se muestra el nombre para las relaciones entre DEPARTAMENTO y EMPLEADO.

	ACTIVIDAD	DEPARTAMENTO	EMPLEADO
ACTIVIDAD			✓
DEPARTAMENTO			RESPONSABLE DE
EMPLEADO	✓	ASIGNADO A	

FIG. II.15 Matriz de Relación, muestra el nombre para dos relaciones

Usar una lista de pares de nombres de relaciones para ayudar a ponerle nombre a dichas relaciones.

Pares de Nombres para las Relaciones :

❑ basado en	las bases para
❑ proviene de	el proveedor de
❑ descripción de	para
❑ operado por	el operador de
❑ representado por	la representación de
❑ responsable de	la responsabilidad de

No se puede usar *relacionado a* o *asociado con* como nombre de relaciones

Paso 3 Determinar las Opcionalidades de una Relación

Determinar la opcionalidad de cada dirección de la relación.

Cuestionar acerca de una Relación Opcional

❑ ¿Debe la ENTIDAD A ser (*nombre de la relación*) de la entidad B?

❑ ¿Debe la ENTIDAD B ser (*nombre de la relación*) de la entidad A?

Considerar la relación entre DEPARTAMENTO y EMPLEADO.

¿Debe un EMPLEADO estar asignado a un DEPARTAMENTO? ¿Siempre?

¿Existe una situación en la cual un EMPLEADO no será asignado a un DEPARTAMENTO

No, un EMPLEADO debe estar siempre asignado a un DEPARTAMENTO

¿Debe un DEPARTAMENTO ser responsable de un EMPLEADO?

No, un DEPARTAMENTO no debe ser responsable de un EMPLEADO

En la (Fig. II.16) se observan las líneas de relación con los nombres de las relaciones.

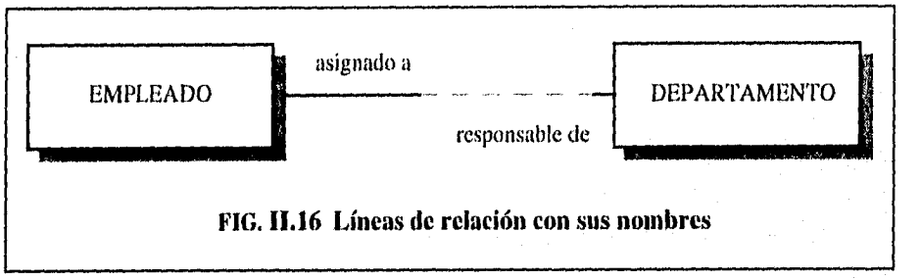


FIG. II.16 Líneas de relación con sus nombres

Paso 4 Determinar el Grado de Relación

Determinar el grado de la relación en ambas direcciones. Cuestionar el grado de la relación.

- ¿Puede la ENTIDAD A ser *nombre de la relación* de más de una de la ENTIDAD B?
- ¿Puede la ENTIDAD B ser *nombre de la relación* de más de una de la ENTIDAD A?

Considerar las relaciones entre DEPARTAMENTO y EMPLEADO.

¿Puede un EMPLEADO ser asignado a más de un DEPARTAMENTO?

No, un EMPLEADO debe ser asignado solamente a un DEPARTAMENTO.

¿Puede un DEPARTAMENTO ser responsable de más de un EMPLEADO?

Si, un DEPARTAMENTO puede ser responsable de más de un EMPLEADO.

En la (Fig. II.17) se agregan los grados de relación al Diagrama E-R.

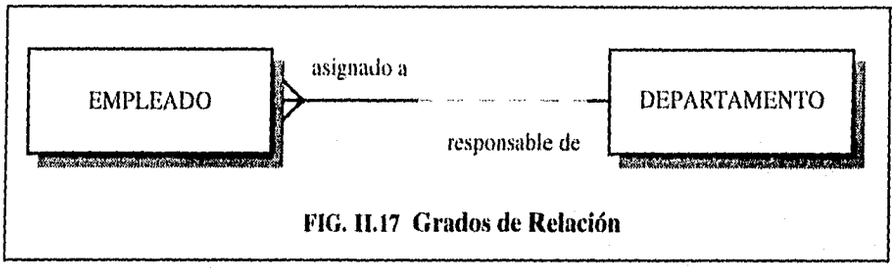


FIG. II.17 Grados de Relación

Paso 5 Validar la Relación

Reexaminar el modelo E-R y validar la relación

Leer en voz alta la relación

❑ Las relaciones deben ser fáciles de leer y tener sentido en el negocio.

Leer la relación representada en la (Fig. II.18) por el siguiente diagrama.

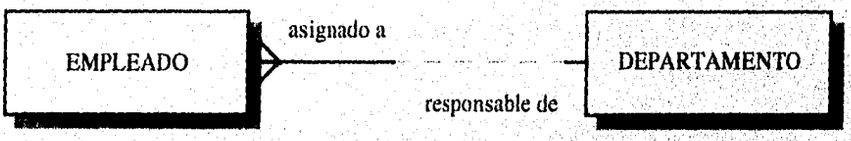


FIG. II.18 Relación entre Empleado y Departamento

Cada EMPLEADO debe estar asignado a uno y solamente un DEPARTAMENTO.

Cada DEPARTAMENTO puede ser responsable de uno o más EMPLEADOS.

2.2.2.5 Representación del Diagrama E-R

Hacer un diagrama E-R fácil de leer y aplicarlo para la gente que necesita trabajar con el, considerando los siguientes puntos :

❑ Limpio y ordenado

- ♦ Alinear las cajas de las entidades.
- ♦ Dibujar las líneas de relación como rectas horizontales o verticales.
- ♦ Usar un ángulo de 30° a 60° el cual facilita seguir las líneas de la relación cuando éstas se cruzan.
- ♦ Evitar el uso de muchas líneas paralelas ya que se dificulta el seguimiento

❑ Texto claro

- ♦ Hacer todo el texto claro.
- ♦ Evitar abreviaciones y modismos.
- ♦ Agregar adjetivos para mejorar entendimiento.
- ♦ Alinear el texto horizontalmente.
- ♦ Poner el nombre de la relación al final de la línea y en lados opuestos de la línea.

❑ Formas fáciles de Recordar

- ♦ Hacer el Diagrama E-R fácil de recordar. Que la gente recuerde las formas y los patrones.

- ♦ No se deben dibujar diagramas E-R en una cuadrícula.
- ♦ Compactar en la medida de lo posible las cajas de las entidades para ayudar a la visualización del diagrama.

❑ Dibujar patas de gallo apuntando hacia arriba o a la izquierda

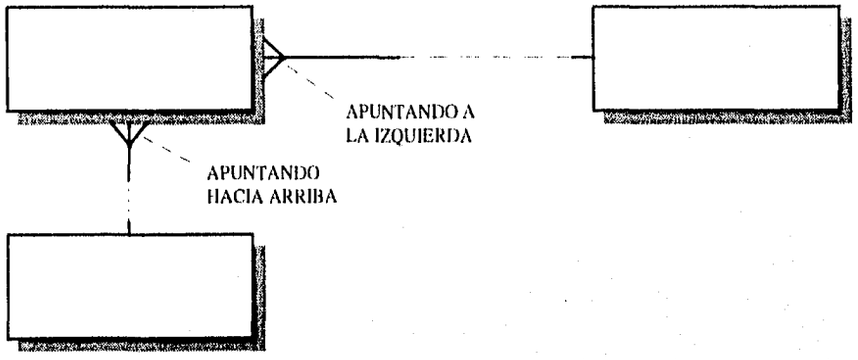


FIG. II.19 Patas de Gallo hacia Arriba ó a la Izquierda

❑ Reglas de Formato

- ♦ Tratar de posicionar la pata de gallo en la línea en la parte izquierda para líneas horizontales y en la parte superior para líneas verticales.
- ♦ Posicionar las entidades más volátiles arriba y a la izquierda del diagrama
- ♦ Posicionar las entidades menos volátiles abajo y a la derecha del diagrama
- ♦ Hasta que una relación M:M sea resuelta, al menos un fin de la relación debe apuntar abajo y a la derecha.

2.2.3 Atributos

Los atributos son información que se necesita conocer o tener acerca de una entidad. Los atributos describen una entidad para calificar, identificar, clasificar, cuantificar o expresar el estado de una entidad.

Un atributo se identifica con un nombre (que puede encontrarse en varias entidades) y todos los posibles valores que puede tener. En los siguientes puntos se pueden observar las características de los atributos :

- Los atributos representan un tipo de descripción o detalle, más no una instancia.
- Los nombres de los atributos deben ser claros para un usuario, más no codificado para el desarrollador.
- El nombre de la entidad es siempre un calificador de atributos del nombre del atributo. -Ejemplo: código de CURSO. Por tanto, los nombres de los atributos no deberían incluir el nombre de la entidad.
- los nombres de los atributos deben ser específicos. -Ejemplo: en el caso de una *cantidad*, es *cantidad regresada* ó *cantidad comprobada*.
- Clarificar siempre la fecha de un atributo con una descripción o una frase. -Ejemplo: fecha de contacto, fecha de orden.
- Un atributo debe estar asignado a una sola entidad.

2.2.3.1 Estándares de Diagramación de Atributos

Para diagramar los atributos tomar encuenta los siguientes puntos:

- ❑ Los nombres de los atributos están en singular y se muestran en minúsculas.
- ❑ Listar los nombres de los atributos en su caja de entidad.

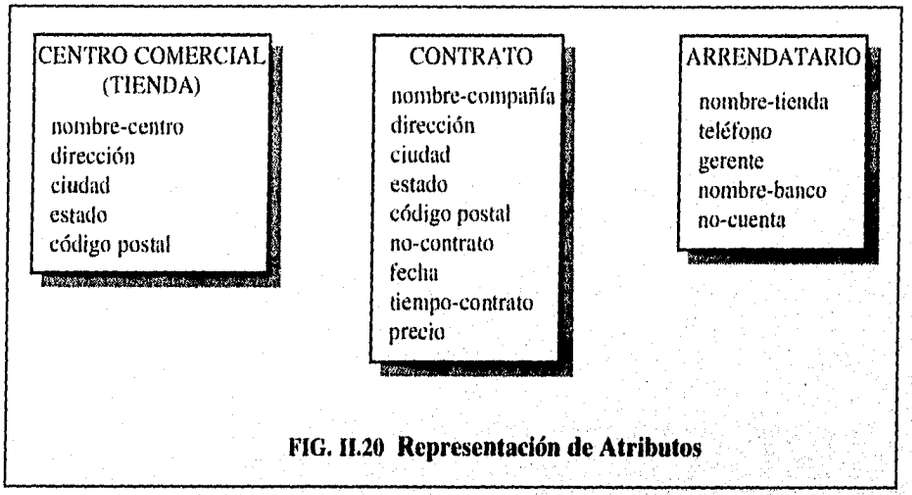


FIG. II.20 Representación de Atributos

Todos los atributos se deben descomponer hasta su mínimo componente con significado.

En la (Fig. II.21) el número de ARTICULO puede ser descompuesto en tipo, vendedor y número de artículo.

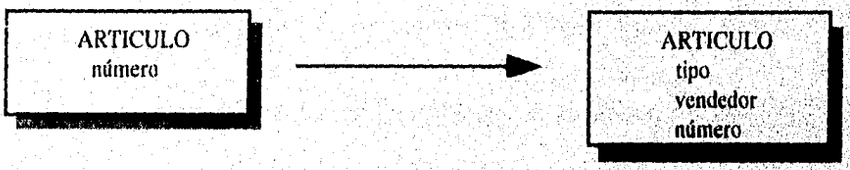


FIG. II.21 Descomposición de un Atributo

Algunas consideraciones que se deben tomar en cuenta sobre los atributos son :

- ❑ Descomponer los atributos agregados y los campos de código en atributos sencillos.
- ❑ Los atributos que contienen fechas, horas, números del seguro social, códigos postales generalmente no se descomponen.
- ❑ Un atributo de dirección frecuentemente se deja como un agregado y se descompone durante el diseño. Alternativamente puede ser descompuesto en múltiples atributos: número/departamento, calle, ciudad, estado y código postal.
- ❑ El nivel de la descomposición de atributos depende de los requerimientos del negocio.

Verificar que cada atributo tenga un sólo valor para cada entidad. Un atributo multivalor o *un grupo de repetición* no es un atributo válido.

En la (Fig. II.22) ¿Los atributos de CLIENTE son valores simples?

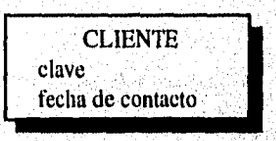


FIG. II.22 Atributos de CLIENTE

No, un cliente puede ser contactado muchas veces, y el negocio necesita guardar todas las fechas de contacto, por lo tanto falta la entidad CONTACTO, ver (Fig. II.23)

FIG. II.23 Creación de la Entidad CONTACTO

En este punto podemos hacer una observación importante: Un atributo repetido indica que falta una entidad.

Verificar que un atributo no sea derivado o calculado de los valores existentes de otros atributos.

Datos Derivados más comunes :

- Contadores (ejemplo: el número de vendedores de una región)
- Totales (ejemplo: el total de ventas mensuales de cada vendedor)
- Max/Min/Promedio (ejemplo: estadísticas de ventas de un grupo de vendedores)
- Otros cálculos (ejemplo: la comisión de un vendedor calculada al 10% de las ventas)

Algunas consideraciones que se deben tomar en cuenta sobre estos datos son :

- ❑ No incluir atributos derivados en un modelo E-R, los atributos derivados son redundantes
- ❑ La redundancia de datos puede ocasionar una inconsistencia de valores de datos. El dato derivado debe ser actualizado cada vez que se modifican los atributos en los que se basa.
- ❑ Decidir utilizar la opción de almacenamiento de datos derivados durante el diseño de la base de datos.

2.2.3.2 Diferencia entre Atributos y Entidades

Si un atributo tiene atributos propios, entonces es realmente una entidad.

En la (Fig. II.24) el usuario identifica el color como un atributo del VEHICULO. Después el usuario define los requerimientos para guardar el color de la pintura, tipo de pintura y el tono para cada color. La combinación del color tendrá sus propios atributos, y llegará a ser una entidad relacionada con VEHICULO.

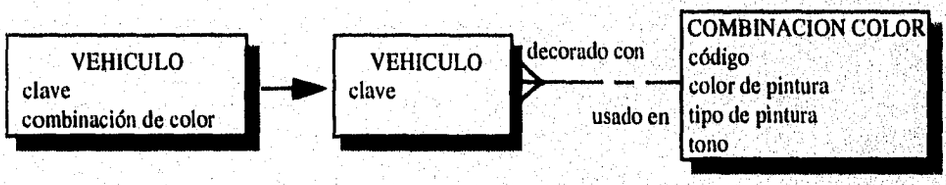


FIG. II.24 Descomposición del atributo COMBINACION DE COLOR

Todas las entidades son sustantivos, pero no todos los sustantivos son entidades.

Características de Entidades	Características de Atributos
Cualquier cosa acerca de la cual podemos tener información	Califican a una entidad
Poseen uno o más atributos	No poseen atributos propios
Si una entidad no tiene atributos, ésta puede ser sólo un atributo	Si un atributo tiene atributos, entonces o es una entidad o no tiene significado
Pueden tener múltiples ocurrencias asociadas con otra entidad a través de una relación	Tiene un sólo valor por cada ocurrencia de la entidad. (No hay repetición en grupos)

FIG. II.25 Diferencias entre Atributos y Entidades

2.2.3.3 Opcionalidad de Atributos

Identificar la opcionalidad de atributo al utilizar una marca de atributo :

- ❑ Atributos obligatorios.
  - ♦ Un valor *debe ser* conocido por cada ocurrencia de la entidad.
  - ♦ Marcarlo con " * ".
- ❑ Atributos opcionales
  - ♦ Un valor *puede ser* conocido por cada ocurrencia de la entidad.
  - ♦ Marcarlo con " o ".

En la (Fig. II.26) se observa los atributos de la entidad PERSONA con sus respectivos atributos obligatorios y opcionales.

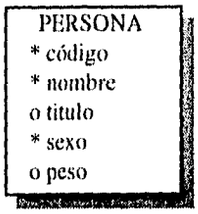


FIG. II.26 Representación de atributos obligatorios (*) y opcionales(o)

Los atributos título y peso son opcionales. Los demás atributos son obligatorios.

Para validar la opcionalidad del atributo se puede utilizar la instancia del atributo de un dato. La (Fig. II.27) es un mapa de instancias que se utiliza para almacenar ejemplos de atributos de datos.

NOMBRE DEL ATRIBUTO	CODIGO	NOMBRE	TITULO	SEXO	PESO
ETIQUETA	*	*	O	*	O
EJEMPLO DE DATOS	510	Gloria	Presidente	F	——
	420	Sebastián	Secretario	M	80
	415	Patricia	——	F	50
	111	Emiliano	Tesorero	M	70
	95	Catalina	——	F	——

2.2.3.4 Identificación de Atributos Mapa de Instancias

Los atributos se pueden identificar a través de examinar las notas de entrevistas y realizando preguntas al usuario.

Los atributos pueden aparecer en notas de entrevistas como:

- ❑ Frases y palabras descriptivas.
- ❑ Sustantivos.
- ❑ Frases preposicionales (salario mensual por cada EMPLEADO)
- ❑ Pronombre y sustantivos posesivos (nombre del EMPLEADO)

Preguntas al usuario tales como:

- ❑ ¿Qué información se necesita para conocer u obtener información acerca de la *entidad x*?
- ❑ ¿Qué información se desea desplegar o imprimir acerca de la *entidad x*?

Examinar la documentación que existe de los procedimientos manuales o sistemas automatizados para descubrir atributos adicionales y omisiones.

Formas de papel	Reportes	Archivos
Encabezados	Campos	Formatos de registros
Prompts	Encabezados	Impresión de los datos de los archivos
	Formas de ordenar	

FIG. II.28 Documentación de un Sistema

Preguntar al usuario de que si el atributo es realmente necesario. Cuidarse de los requerimientos obsoletos acarreados de los sistemas anteriores o de datos derivados.

2.2.3.5 Asignar Identificadores Unicos

Un identificador único (UID) es cualquier identificador de atributos y/o relaciones que sirven para identificar en forma única una ocurrencia o instancia de una entidad. Cada ocurrencia de una entidad debe ser identificada de manera única.

En la (Fig. II.29) en un negocio cada ocurrencia de DEPARTAMENTO se identifica de manera única por el número de departamento.

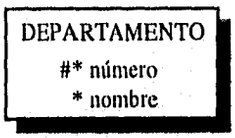


FIG. II.29 Identificación del Identificador único

El UID para la entidad DEPARTAMENTO es el atributo número

Considerar las siguientes observaciones :

- Una entidad debe tener un UID, o ésta no es una entidad, todos los componentes de un UID deben de ser obligatorios (*)
- Marcar cada atributo UID con (# *).

Una entidad puede ser identificada de manera única a través de una relación:

En la (Fig. II.30) para la industria bancaria, a cada banco se le asigna un número único de banco. Dentro de cada banco cada cuenta tiene un número único de cuenta. ¿Cuál es el UID de la entidad CUENTA?

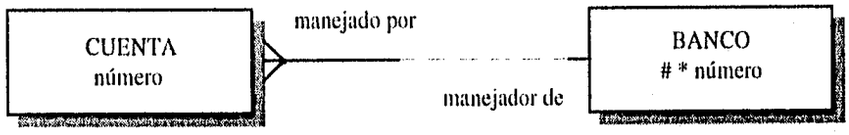


FIG II.30 Relación entre Cuenta y Banco

La CUENTA es identificada como única por su atributo número y el BANCO específico a la que está relacionada

Usar la barra UID para identificar que la relación es parte del identificador único de la entidad. En la (Fig. II.31) la barra UID indica que la relación con el BANCO es parte del UID de CUENTA.

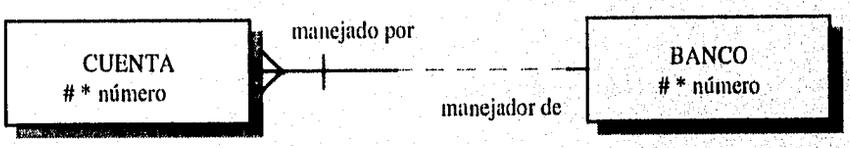


FIG II.31 Representación de la barra UID

Una relación incluida dentro de un UID debe de ser obligatoria y *una y solamente una* en la dirección que participa en el UID.

Una entidad puede ser identificada de manera única a través de múltiples relaciones :

En la (Fig. II.32) un negocio necesita guardar los trabajos asignados a sus empleados. A los empleados se les asignan tareas de los proyectos. Un empleado puede tener múltiples asignaciones en un sólo proyecto, cada uno con diferente fecha de asignación.

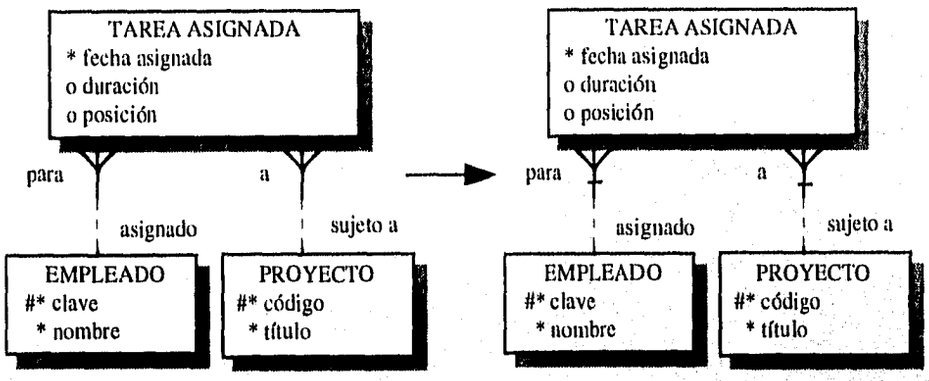


FIG. II.32 Múltiples Relaciones

¿Cuál es el UID de la entidad TAREA ASIGNADA?

Una TAREA ASIGNADA es identificada de manera única para el EMPLEADO a quien se le dió la TAREA ASIGNADA, el PROYECTO que contiene a la TAREA ASIGNADA y por la fecha asignada. Ambas relaciones son obligatorias y una y sólo una en la dirección incluida en el UID.

Una entidad puede tener más de un UID:

¿Qué es lo que identifica de manera única al EMPLEADO?

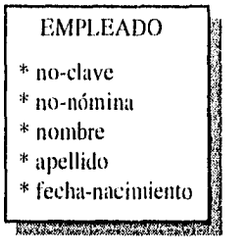


FIG. II.33 Entidad Empleado con sus Atributos

Los candidatos a UID son:

1. Número de clave
2. Número de nómina
3. Nombre/Apellido

¿Todos estos son únicos? La combinación de Nombre y Apellido probablemente no es único

Seleccionar un candidato UID para ser el UID primario, y los otros para ser los UID secundarios

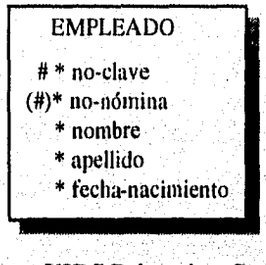


FIG. II. 34 UIDS Primario y Secundario

Cada UID secundario se marca con [(#)] o se deja sin ninguna marca

Considerar la creación de un atributo artificial para ayudar a identificar cada entidad :

¿Cuál es el identificador único de la entidad CLIENTE?

FIG. II.35 Entidad Cliente

Posiblemente el nombre y el apellido del CLIENTE puede ser un UID. Sin embargo, puede haber dos CLIENTES con el mismo nombre.

Crear un atributo artificial llamado código del CLIENTE, el cual sea único para cada instancia del CLIENTE

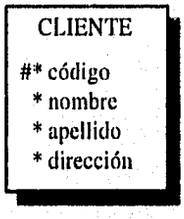


FIG. II.36 Creación de un atributo artificial (#*código)

Observaciones :

- Con frecuencia los atributos artificiales son usados para UIDS.
- Definir un código artificial cuando el negocio no tiene un atributo natural el cual identifique de manera única a una entidad.

Buscar atributos y relaciones para identificar cada entidad :

❑ Evaluar los atributos

- ¿Qué atributos obligatorios identifican a la entidad? Buscar atributos adicionales para ayudar a identificar a la entidad. Considerar el crear atributos artificiales para su identificación.
- ¿Existe algún atributo que identifique de manera única a la entidad?
- ¿Qué combinación de atributos identifica de manera única a la entidad?

❑ Considerar las relaciones

- ¿Qué relaciones ayudan a identificar a la entidad?
- ¿Faltan relaciones que ayuden a identificar a la entidad?
- ¿La relación ayuda a identificar de manera única a la entidad?
- ¿La relación es obligatoria y es una y sólo una en la dirección de la entidad?

❑ Validar el UID

- Examinar datos simples. ¿Podrán las combinaciones seleccionadas de atributos y relaciones identificar de manera única a cada instancia de la unidad?
- ¿Existen todos los atributos y relaciones que están incluidos en el UID obligatorio?

2.3 MODELO AVANZADO

El modelo avanzado abarca lo que es la normalización de los datos y las formas de normalización existentes. A continuación se explican estos puntos.

2.3.1 Normalizar el Modelo de Datos

Normalizar es un concepto de base de datos relacional, pero sus principios se aplican al Modelo Conceptual de Datos.

La normalización es una de las etapas más importantes en el diseño de bases de datos por su aplicación a los problemas con los que nos enfrentamos en el mundo real cuando se pretende incorporarlos y gestionarlos dentro del ordenador mediante sistemas de bases de datos.

Hace no muchos tiempo, por los años 60, los datos que se manejaban dentro del ordenador mediante ficheros fueron diseñados y gestionados por programas (generalmente en COBOL). Realmente no quedaba más remedio, puesto que hasta entonces no existían otros medios y no habían nacido herramientas que ganasen la confianza de los responsables del diseño de aplicaciones.

Entonces, se empezaron a realizar estudios del rendimiento de los datos, orientándolos principalmente en dos áreas:

- Las aplicaciones que los gestionaban
- La evolución de los requerimientos

¿Cuáles fueron los resultados? En la mayoría de los casos, coincidían en mostrar las mismas evidencias:

- ❑ Para nuevas aplicaciones, la información tal y como estaba diseñada no servía y esto hacía que dichas aplicaciones creasen sus propios datos.
- ❑ La información estaba duplicada.
- ❑ Era muy difícil y costoso mantener los datos actualizados.
- ❑ El rendimiento no era el esperado.

Ante esta situación, se empezaron a generar las primeras bases de datos y junto con ellas se estudió una metodología de análisis como ayuda al diseño de los datos, con objeto de optimizar su tratamiento y futuras necesidades. Aparecía el concepto de normalización.

Pues bien, el proceso de normalización se encarga de seguir una serie de pasos y normas que tras aplicar todas ellas, se obtienen los datos agrupados, de tal forma que es la estructura óptima para su implementación, gestión y explotación desde diferentes futuras aplicaciones. Una tabla se dice que está en una forma normal cuando satisface un conjunto de restricciones impuestas por dicha norma.

El proceso de normalización parte de las formas normales definidas por E.F.Codd (1970) creador de las bases de datos relaciónales. Primeramente, Codd formuló las tres primeras formas normales (1FN, 2FN, 3FN);

posteriormente unas anomalías detectadas forzaron a crear una forma normal más completa que la 3FN, es la FNBC (forma normal de Boyce y Codd), después Fagin definió la 4FN y 5FN, estas tres formas normales últimas no son comúnmente utilizadas.

La normalización se basa en que *"los datos son independientes de las aplicaciones que los gestionan"*, y su objetivo es obtener el mayor número de entidades posibles, dejando en cada una de ellas los atributos imprescindibles para representar a la entidad, o a la relación entre entidades.

Las ventajas que se obtienen tras la normalización de datos para su eficaz gestión son:

- *Facilidad de uso.* Los datos están agrupados en entidades que identifican claramente a una relación.
- *Flexibilidad.* La información que necesitan los usuarios se puede obtener de las tablas relaciones o relaciones mediante las operaciones del álgebra relacional. Por ejemplo, uniendo tablas, seleccionando sus valores y proyectándolos, etc.
- *Precisión.* Las interrelaciones entre las tablas consiguen mantener información diferente relacionada con toda exactitud.
- *Seguridad.* Los controles de acceso para consultar o actualizar información (bien a nivel de tablas o atributos) son mucho más sencillos de implementar.

- *Facilidad de implementación.* Las tablas se almacenan físicamente como archivos planos.

- *Independencia de datos.* Los programas no están ligados a las estructuras, con lo que se consigue aumentar la base de datos añadiendo nuevos atributos o nuevas tablas sin que afecten a los programas que las usan.

- *claridad.* La representación de la información es clara y sencilla para el usuario; son tablas simples.

- *Facilidad de gestión.* Los lenguajes manipulan la información de forma sencilla al estar los datos basados en el álgebra y cálculo relacional.

- *Mínima redundancia.* La información no estará duplicada innecesariamente dentro de las estructuras.

- *Máximo rendimiento de las aplicaciones.* Sólo se trata aquella información que va a servir de utilidad a cada aplicación.

El proceso de normalización lo realiza el analista tras sucesivas reuniones que mantiene con el usuario y toda la información que recibe se debe documentar.

También es muy importante plasmar las interrelaciones y las restricciones existentes entre los datos, así como agruparlas para ir formando el modelo entidad-relación.

Regla de Forma Normal	Descripción
Primera Forma Normal	Todos los atributos deben tener un sólo valor para cada instancia
Segunda Forma Normal	Un atributo debe ser dependiente del identificador único completo
Tercera Forma Normal	Ningún atributo no-UID puede ser dependiente de otro atributo no-UID

FIG. II.37 Reglas de Normalización

Un modelo de datos entidad-relación normalizado se traslada automáticamente dentro de un diseño de base datos. La tercera forma normal es un objetivo generalmente aceptado para eliminar redundancia en un diseño de base de datos.

2.3.1.1 Primera Forma Normal (1FN)

Todos los atributos deben tener un sólo valor.

Tomar en cuenta la siguiente consideración:

- Validar que cada atributo tenga un valor único para cada ocurrencia de la entidad. Ningún atributo deberá tener valores repetidos.

En la (Fig. II.38) ¿La entidad CLIENTE cumple con 1FN?, Si no cumple, ¿Cómo se podría convertir a 1FN?

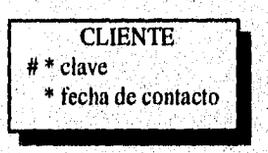


FIG. II.38 Entidad Cliente

El atributo fecha de contacto tiene múltiples valores, por lo tanto la entidad CLIENTE no es considerada como 1FN.

En este caso se crea una entidad adicional CONTACTO con una relación de M:1 hacia CLIENTE, ver (Fig. II.39).

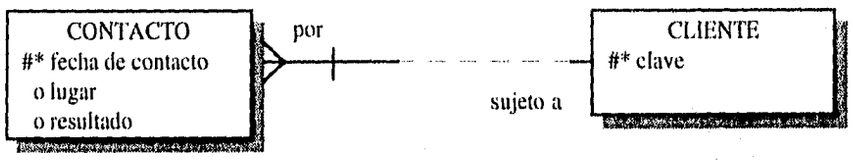


FIG. II.39 Creación de una Entidad Adicional (Contacto)

Observaciones:

- ▲ Si una entidad tiene múltiples valores, se crea una entidad adicional y lo relaciona con la entidad original mediante una relación M:1.

2.3.1.2 Segunda Forma Normal (2FN)

Un atributo debe ser dependiente del identificador único

Tomar en cuenta las siguientes consideraciones:

- ❑ Validar que cada atributo dependa completamente del UID. Cada instancia específica del UID debe determinar una sola instancia de cada atributo
- ❑ Validar que un atributo no dependa de una sola parte del UID de la entidad.

En la (Fig. II.40) validar los atributos asignados a entidad CURSO

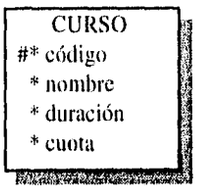


FIG. II.40 Entidad Curso

Cada instancia del código de curso determina un valor específico para el nombre, duración y cuota. Los atributos están ubicados correctamente.

Para la (Fig. II.41) validar que los atributos de las entidades de CUENTA y BANCO estén correctamente ubicados.

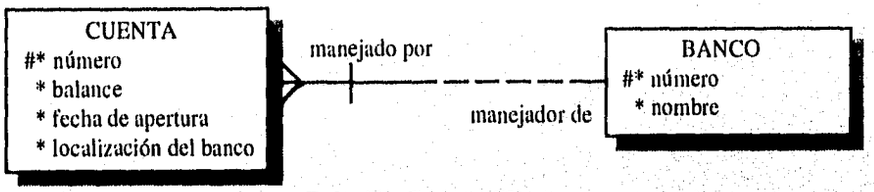


FIG. II.41 Relación Cuenta-Cliente con sus Atributos

Cada instancia de un BANCO y número de cuenta determinan valores específicos del balance y de la fecha de apertura para cada cuenta. El atributo de localización del banco está mal ubicado. Este depende del BANCO, pero no del número de cuenta. Este no debería ser un atributo de CUENTA.

Observaciones:

- ▲ Sí un atributo no es dependiente del UID completo, está fuera de lugar y deberá ser movido

2.3.1.3 Tercera Forma Normal (3FN)

Ningún atributo no-UID puede ser dependiente de otro atributo no-UID.

Tomar en cuenta las siguientes consideraciones:

- ❑ Validar que cada atributo no-UID no dependa de otro atributo no-UID
- ❑ Mover cualquier atributo no-UID que dependa de otro atributo no-UID

Para la (Fig. II.42) ¿Algún atributo no-UID de esta entidad depende de otro atributo no-UID?

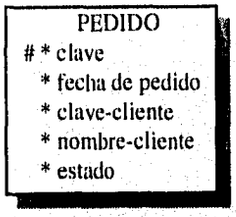


FIG. II.42 Entidad Pedido

Los atributos nombre del cliente y estado dependen de la clave de cliente.

Crear otra entidad llamada CLIENTE con el campo de clave-cliente como su UID y ubicar los atributos correctamente.

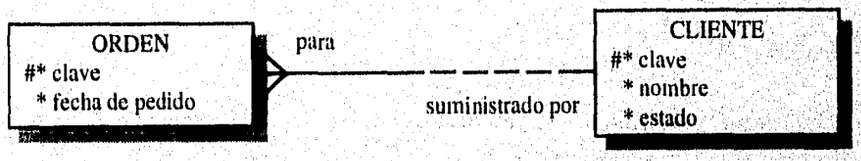


FIG .II.43 Entidades Orden y Cliente

Observaciones:

- ▲ Si un atributo depende de otro atributo no-UID, es necesario mover ambos, el atributo dependiente y el atributo del que depende, a una nueva entidad relacionada con la entidad actual.

2.3.2 Resolución de Relaciones Muchos a Muchos (M:M)

Algunos atributos pueden asociarse con relaciones M:M. Para resolver relaciones M:M se agrega una entidad intersección con esos atributos:

De la (Fig. II.44) considerar la relación M:M entre PRODUCTO y VENDEDOR. ¿Cuál es el precio actual de un PRODUCTO específico para un VENDEDOR específico?

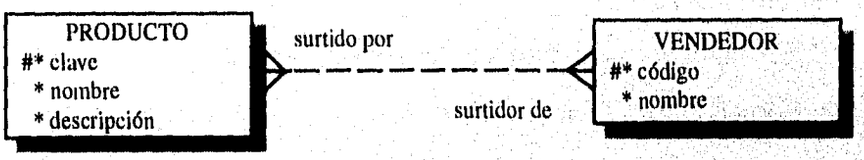


FIG. II.44 Relación Muchos a Muchos

El precio actual parece ser un atributo de la relación entre PRODUCTO y VENDEDOR.

Observaciones:

- ▲ Los atributos únicamente describen entidades. Si los atributos describen relaciones, las relaciones deberán ser resueltas.

Para resolver una relación M:M reemplazarla con una entidad intersección nueva y con dos relaciones M:1:

La relación M:M entre PRODUCTO y VENDEDOR (Fig. II.44) puede ser resuelta agregando una entidad intersección llamada CATALOGO. El precio actual es realmente un atributo de la entidad CATALOGO, ver (Fig. II.45).

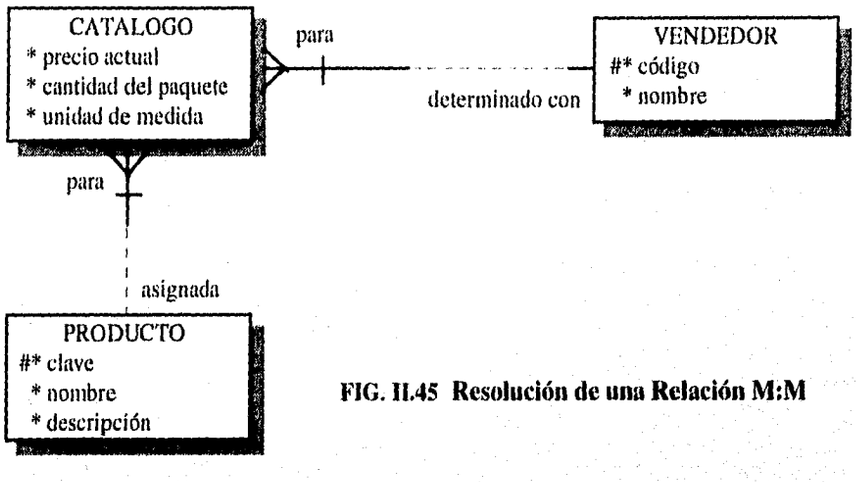


FIG. II.45 Resolución de una Relación M:M

Una vez que está definida la entidad CATALOGO, se analiza la posibilidad de requerimientos para atributos adicionales como: cantidad del paquete y unidad de medida son también atributos del CATALOGO. El UID del CATALOGO es el compuesto de estas dos relaciones.

Observaciones para una entidad intersección:

- ▲ Una entidad intersección es frecuentemente identificada por las dos relaciones que le dieron origen.
- ▲ Las relaciones desde una entidad intersección son siempre obligatorias.

- ▲ Las entidades intersección son muy comunes para representar situaciones de negocios en el mundo real.
- ▲ Las entidades intersección normalmente generan requerimientos para atributos adicionales como el uso de cantidades y fechas. Estas tienden a ser entidades volátiles y de volumen alto.

2.3.2.1 Estándares para Diagramar y Resolver Relaciones M:M

Posicionar las entidades intersección de tal forma que permitan a las patas de gallo apuntar hacia arriba o hacia la izquierda:

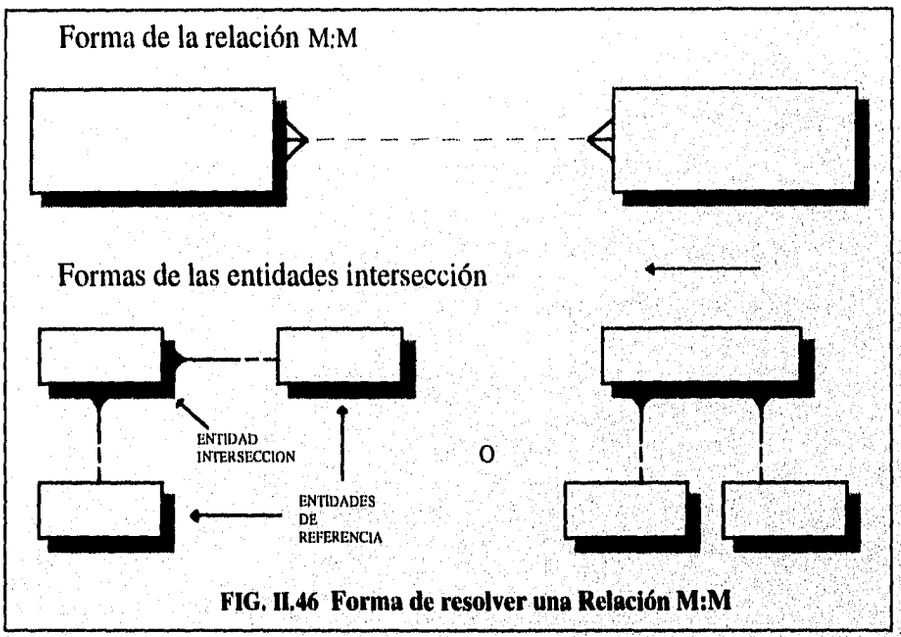


FIG. II.46 Forma de resolver una Relación M:M

Observaciones:

- ▲ Una entidad de referencia es una entidad que no ha tenido relaciones obligatorias conectadas a ésta.

▲ Cuando las relaciones M:M son resueltas, la forma del diagrama puede ser confusa.

El UID de una entidad intersección está frecuentemente compuesta de las relaciones entre las entidades que le dieron origen:

En la (Fig. II.47) se resuelve la relación M:M para acomodar los siguientes requerimientos: "Cada estudiante tiene una fecha de registro en el curso, fecha de terminación del curso y el grado del estudiante."

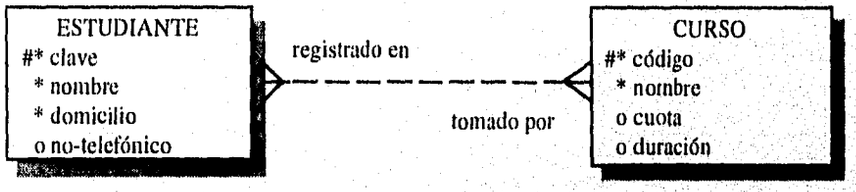


FIG. II.47 Relación M:M entre Estudiante y Cliente

Resolviendo la relación M:M, se agrega la entidad intersección MATRICULA y las dos relaciones M:1, ver (Fig. II.48).

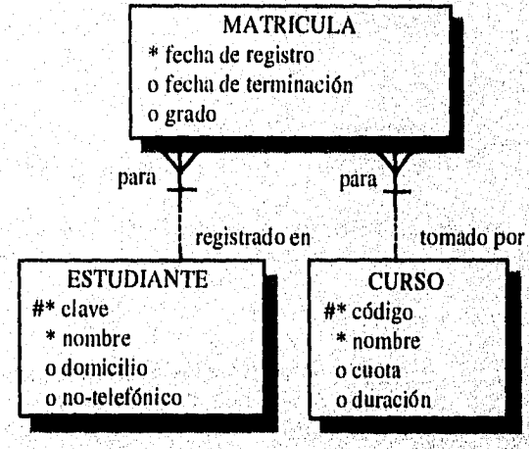


FIG. II.48 Resolución de la Relación M:M entre Estudiante y Cliente

MATRICULA tiene como atributos la fecha de registro, fecha de terminación y el grado. El UID de MATRICULA es creado a partir de las relaciones entre ESTUDIANTE y CURSO.

Observaciones:

- ▲ Este modelo únicamente guarda la última fecha en que un estudiante se inscribió en un curso específico. Si se necesita guardar múltiples fechas de inscripción, incluir el atributo de fecha de inscripción como parte del UID.

Las relaciones de una entidad intersección hacia las dos entidades que le dieron origen puede no ser adecuada para definir de forma única cada ocurrencia de la entidad intersección:

Para resolver la siguiente relación M:M (Fig. II.49) se deben acomodar los requerimientos adicionales siguientes:

"Guardar la fecha en la que cada empleado es asignado a un proyecto, y la duración de cada asignación."

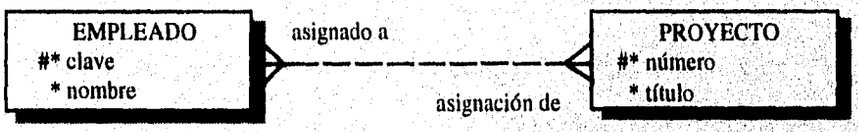


FIG. II.49 Relación M:M entre Empleado y Proyecto

Agregar una entidad intersección llamada TAREA ASIGNADA con los atributos fecha de asignación y duración. (Fig. II.50)

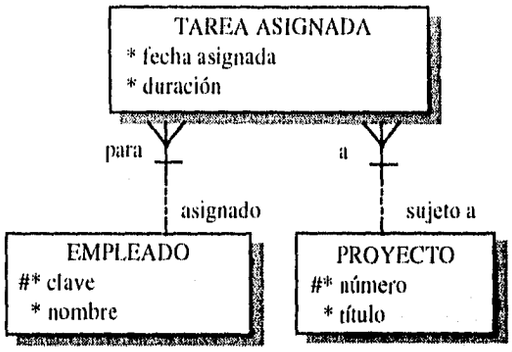


FIG. II.50 Resolución de la Relación M:M entre Empleado y Proyecto

TAREA ASIGNADA se identifica parcialmente por las relaciones a EMPLEADO y a PROYECTO, sin embargo, estas dos relaciones no son suficientes para identificar de manera única a una TAREA ASIGNADA. Un empleado puede tener múltiples asignaciones a proyectos, con diferentes fechas de asignación . Por lo tanto, el UID de TAREA ASIGNADA deberá incluir el EMPLEADO relacionado, el PROYECTO relacionado, y el atributo de fecha de asignación.

Una vez que se identifica una entidad intersección, hay que buscar atributos adicionales para describir la entidad intersección:

la (Fig. II.51) ¿Qué información se necesita conocer sobre la relación entre PRODUCTO y VENDEDOR?

"Necesitamos guardar el precio actual de un PRODUCTO específico para un VENDEDOR específico"

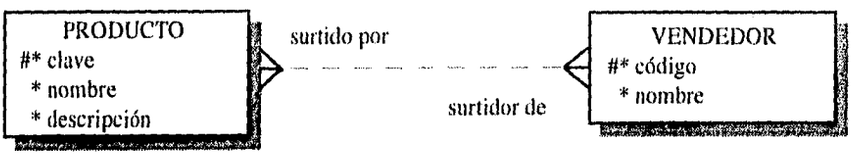


FIG. II.51 Relación M:M entre Producto y Vendedor

Agregar la entidad intersección ARTICULO VENDEDOR con un atributo de precio actual para resolver la relación M:M.

¿Qué otra información se necesita conocer sobre el ARTICULO VENDEDOR?

"También se necesita conocer la cantidad empacada y la unidad de medida de cada ARTICULO VENDEDOR.

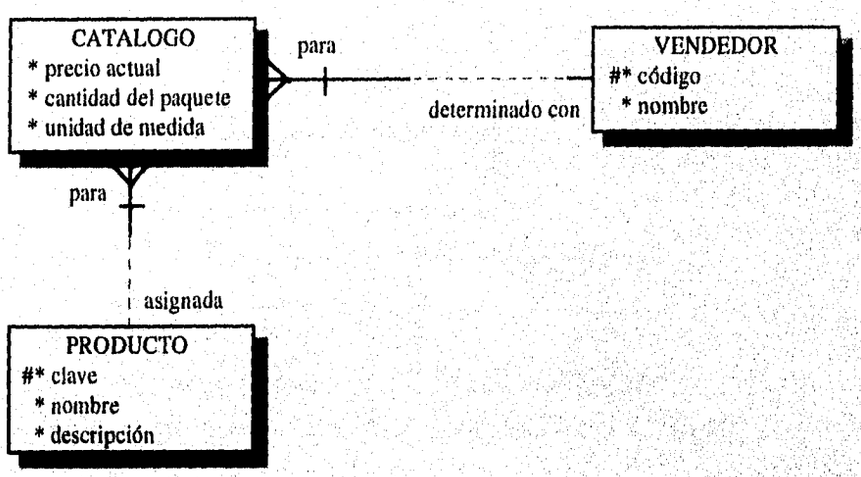


FIG. II.52 Resolución de la Relación M:M entre Producto y Vendedor

Resolver todas las relaciones M:M al final de la fase de Análisis. Esta resolución forzada puede resultar en una Entidad Intersección sin atributos:

Para una tienda de video, se definió la siguiente relación M:M:

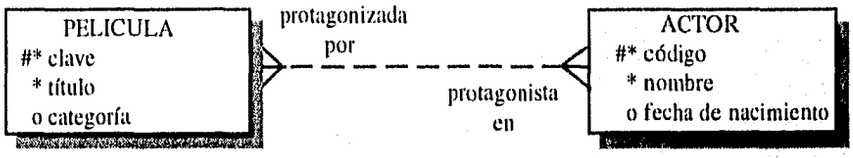


FIG. II.53 Relación M:M entre Pelicula y Actor

Al final de la Etapa de Análisis, el analista no ha identificado algún atributo asociado con la relación M:M. Resolver la relación M:M con un entidad intersección sin ningún atributo:

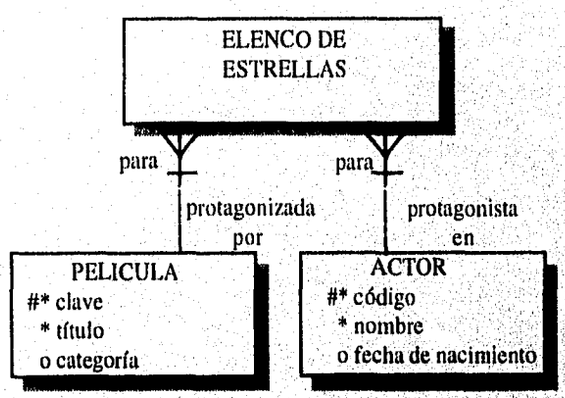


FIG. II.54 Resolución de la Relación M:M entre Pelicula y Actor

Observaciones:

- ▲ Una Entidad intersección sin atributos es justamente una lista de referencia cruzada en doble sentido entre las ocurrencias de las Entidades.

- ▲ Una Entidad intersección sin atributos es la excepción a la regla de que una entidad debe tener atributos para ser una Entidad.
- ▲ El UID para una Entidad intersección vacía, es siempre compuesto de la relación de las dos entidades que lo originaron.

2.3.3 Modelo Jerárquico de Datos

Se puede representar un modelo jerárquica de datos como un conjunto de relaciones muchos a uno:

En la (Fig. II.55) se modela la estructura de organización jerárquica de una compañía como un conjunto de relaciones M:1

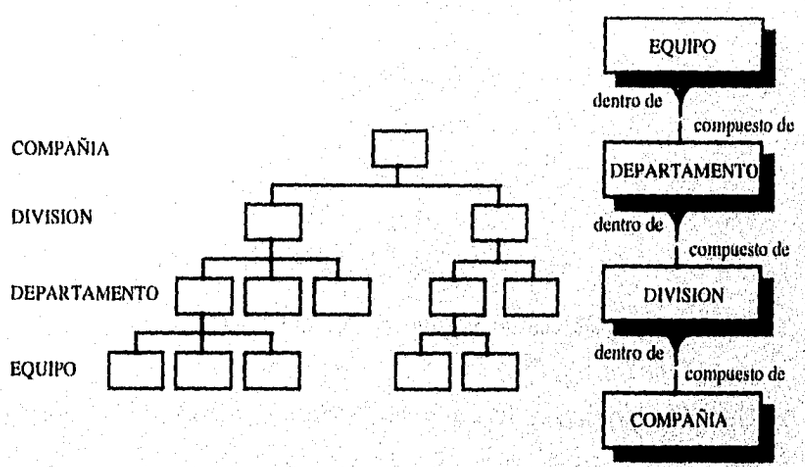


FIG. II.55 Representación Jerárquica como un Conjunto de Relaciones M:1

Observaciones:

- ▲ La forma del diagrama E-R de Oracle de la regla de pata de gallo apuntando hacia el este o hacia el sur causa que las jerarquías sean dibujadas arriba-abajo o a los lados.

Los UID'S para un conjunto de entidades jerárquicas pueden ser propagados a través de relaciones múltiples:

De la (Fig. II.56) ¿Cuales son los UID'S de las entidades PISO, DEPTO. y CUARTO?

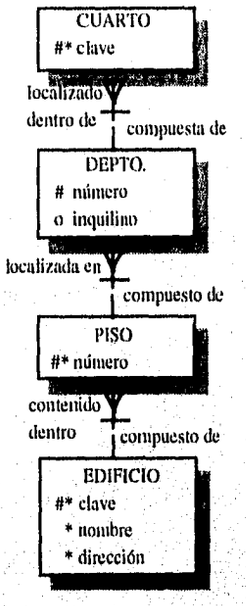


FIG. II.56 Conjunto de Entidades Jerárquicas

El UID del CUARTO es la clave de cuarto y está localizado dentro de un DEPTO.

El UID de el DEPTO. es el número de depto. y está localizado en un PISO

El UID del PISO es el número de piso y está contenido dentro del EDIFICIO

Considerar la creación de atributos artificiales como ayuda para identificar entidades en relaciones jerárquicas:

En la (Fig. II.57) una estructura típica de organización, ¿Qué identifica de manera única instancias de las entidades DIVISION, DEPARTAMENTO, y EQUIPO?

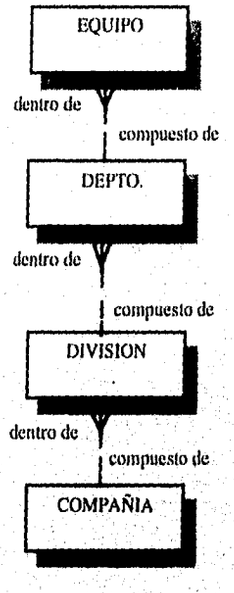


FIG. II.57 Entidades Jerárquicas

Cada EQUIPO puede ser identificado basándose en su DEPARTAMENTO, DIVISION y COMPAÑIA. O cada entidad tiene un código artificial de identificación único e independiente. Si las estructuras jerárquicas cambian ocasionalmente, se recomienda usar identificadores artificiales independientes. Los códigos artificiales de identificación deben ser únicos e independientes.

Recomendaciones:

- ▲ Los códigos artificiales de identificación deben ser únicos e independientes.
- ▲ Si las estructuras jerárquicas cambian ocasionalmente, se recomienda usar identificadores artificiales independientes.

2.3.4 Relaciones del Modelo Recursivo

Una relación recursiva es la relación entre una entidad con ella misma:

En la (Fig. II.58) leer la relación recursiva E-R.

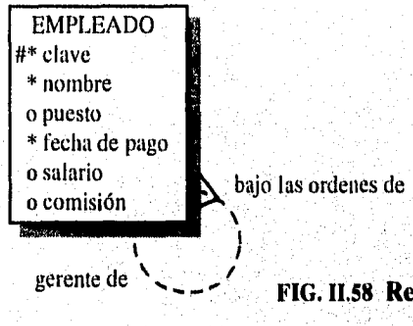


FIG. II.58 Relación Recursiva

Cada EMPLEADO puede estar bajo las ordenes de uno y sólo un EMPLEADO

Cada EMPLEADO puede ser gerente de uno o más EMPLEADOS

Recomendaciones:

- ▲ El ciclo puede aparecer en cualquier lado de la caja de entidad, pero hay que recordar que las patas de gallo se orientan hacia el sur o hacia el este.

Considerar representar una relación jerárquica como una relación recursiva, ver la (Fig. II.59):

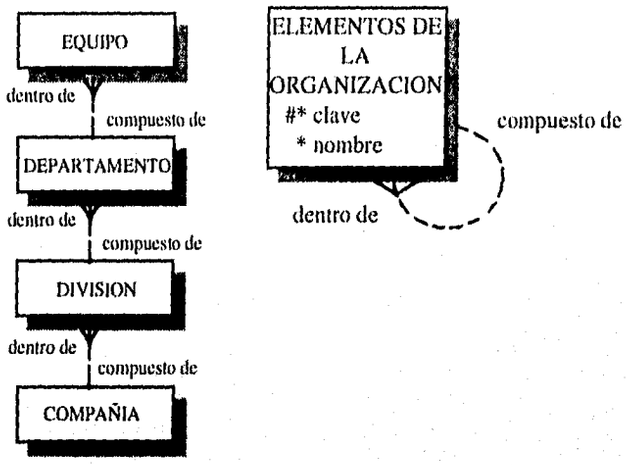


FIG. II.59 Relación Jerárquica con su Relación recursiva

Recomendaciones del modelo recursivo:

- ▲ Una sola entidad recursiva debe incluir todos los atributos de cada entidad individual. Idealmente, las entidades en cada nivel de la jerarquía deben tener los mismos atributos.
- ▲ El modelo de la organización recursiva puede fácilmente manejar la inclusión o eliminación de capas en la organización
- ▲ El modelo de organización recursivo no puede manejar relaciones obligatorias. Si cada ELEMENTO DE LA ORGANIZACION debe estar dentro de otro ELEMENTO DE LA ORGANIZACION, la jerarquía debe ser infinita.

▲ La relación recursiva debe ser opcional en ambas direcciones.

Una lista de materiales puede ser modelada con entidades múltiples para cada categoría de una "parte" y un conjunto de relaciones entre cada una de las entidades:

Una organización de manufactura de automóviles necesita guardar información de partes, subensambles, ensambles, y productos. En la (Fig. II.60) el diagrama E-R modela estos datos considerando cada una de estas categorías de partes como entidad.

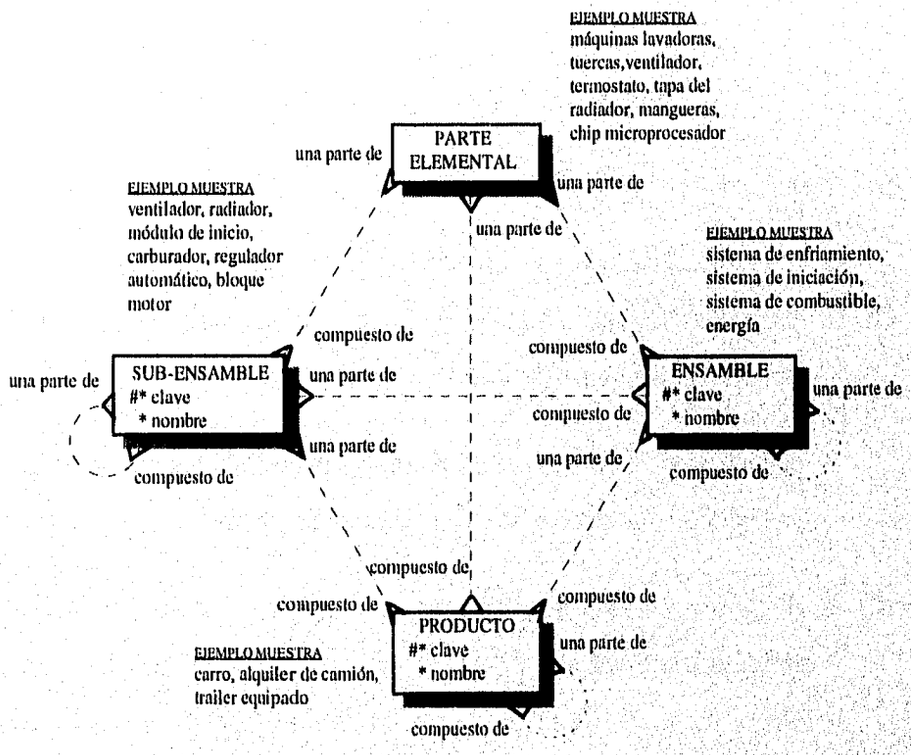


FIG. II.60 Modelo de Entidades Multiples y un Conjunto de Relaciones entre cada una de las Entidades

Modelar la lista de materiales como una relación recursiva de muchos a muchos:

Para la organización de manufactura de automóviles, considerar todas las partes elementales, subensambles, ensambles, y productos como instancias de una entidad llamada COMPONENTE. De esta forma el complejo modelo E-R anterior puede ser remodelado como una simple relación recursiva. (Fig. II.61).

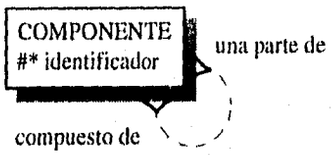


FIG. II.61 Relación Recursiva

Cada COMPONENTE puede ser una parte de uno o más COMPONENTES

Cada COMPONENTE puede estar compuesto de uno o más COMPONENTES

Resolución de la relación recursiva M:M de la (Fig. II.61) con una entidad intersección y dos relaciones M:1 hacia diferentes instancias de la entidad original:

Considerando el modelo recursivo de la lista de materiales. Este modelo guarda información de cuáles componentes son partes de un ventilador. Pero si una rondana es parte de un ventilador, ¿Se guardará información de cuantas rondanas forman parte de un ventilador?

El atributo cantidad parece estar asociado con la relación recursiva. Entonces tomando en cuenta esto podemos resolver esta relación (Fig. II.61) recursiva M:M añadiendo la entidad intersección REGLAS DE ENSAMBLE y dos relaciones M:1 hacia la entidad COMPONENTE. Las reglas de ENSAMBLE tendrán un atributo de cantidad.

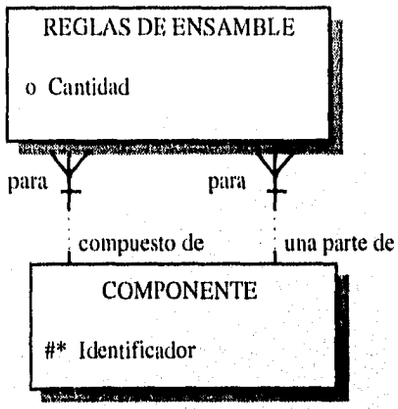


FIG. II.62 Relaciones M:1

Las dos relaciones M:1 de una instancia de REGLAS DE ENSAMBLE se asociarán con diferentes instancias de la entidad COMPONENTE.

Por ejemplo, la instancia de REGLAS DE ENSAMBLE de rondanas para ventiladores tendrá una relación M:1 hacia la instancia de COMPONENTE para rondanas y una segunda relación M:1 hacia la instancia de COMPONENTE para un ventilador. La entidad de REGLAS DE ENSAMBLE guardará la cantidad de rondanas que forman parte de un sólo ventilador

2.3.5 Relaciones que Modelan Roles

Hay que ser precavidos con las entidades que representan roles:

En un modelo Entidad-Relación (E-R) por ejemplo para una compañía de entrenamiento, se define una entidad INSTRUCTOR y una entidad ESTUDIANTE. Este modelo trabaja bien si un INSTRUCTOR nunca es un ESTUDIANTE, y un ESTUDIANTE nunca es un INSTRUCTOR. Pero ¿Qué pasaría si un INSTRUCTOR también fuera ESTUDIANTE?. (Fig. II.63)

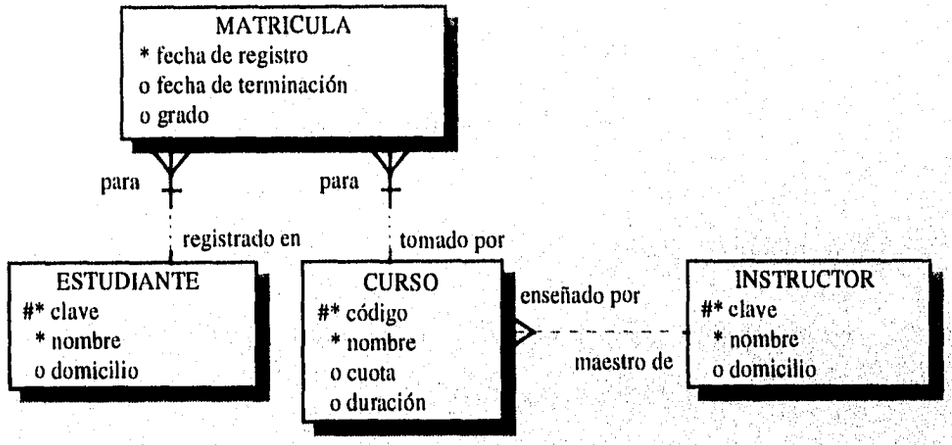


FIG. II.63 Entidades que representan Roles

Recomendaciones:

- ▲ Las entidades que representan roles pueden compartir instancias que se traslapan.

Usar relaciones para modelar roles. Las relaciones permiten que una sola instancia de una entidad asuma múltiples roles:

Para una compañía de entrenamiento, definir una entidad PERSONA que debe tomar los roles de instructor y/o estudiante. (Fig. II.64)

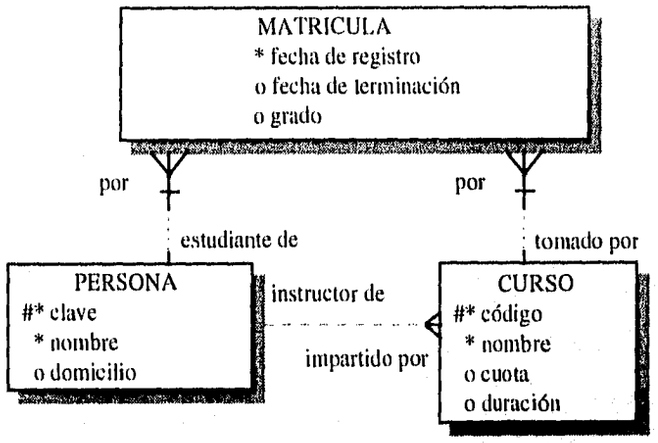


FIG. II.64 Relaciones para Modelar Roles

2.3.6 Modelar Subtipos

Usar subtipos para modelar exclusivamente tipos de entidad que tienen atributos o relaciones comunes:

Para un negocio que tiene definido dos tipos de empleado: Asalariados y por Honorarios, se necesita guardar su clave, nombre y departamento asignado. Pero para los empleados asalariados, se debe guardar el salario del empleado y para los empleados por honorarios, se debe guardar el costo por hora, costo por tiempo, y sindicato

Crear un supertipo EMPLEADO con dos subtipos. Cada EMPLEADO puede ser EMPLEADO ASALARIADO o EMPLEADO POR HONORARIOS. (Fig. II.65)

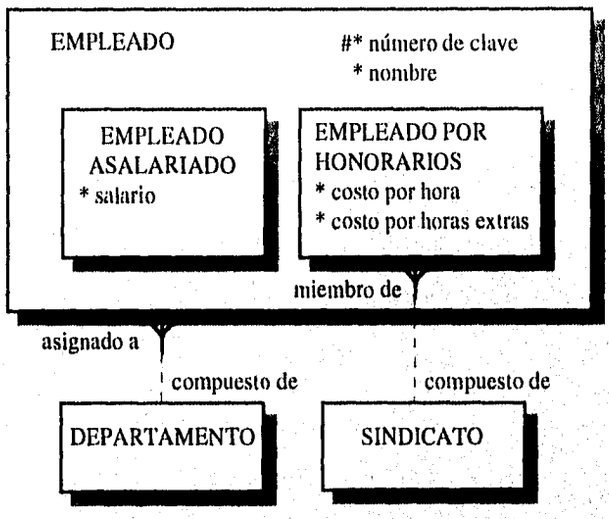


FIG. II.65 Un Supertipo Empleado con dos subtipos

Recomendaciones:

- Se debe tener cuidado con las instancias que pueden estar en ambos subtipos, la construcción del subtipo-supertipo es incorrecta en esas instancias.

Un supertipo es una entidad que tiene subtipos. Un supertipo puede ser separado en dos o más subtipos mutuamente excluyentes:

De la (Fig. II.65), Un EMPLEADO debe ser EMPLEADO ASALARIADO o EMPLEADO POR HONORARIOS, pero no ambos.

El supertipo puede tener atributos y relaciones compartidas por subtipos:

De la (Fig. II.65), a todo EMPLEADO se le clasifica con nombre y clave. Todos los EMPLEADOS deben estar en uno y sólo un DEPARTAMENTO.

Cada subtipo debe tener sus propias relaciones y atributos:

De la (Fig. II.65), el subtipo EMPLEADO ASALARIADO tiene un atributo de salario. El subtipo EMPLEADO POR HONORARIOS tiene como atributos costo por hora y costo por tiempo extra y una relación con la entidad SINDICATO.

Recomendaciones:

- ▲ Un subtipo que no tenga atributos o relaciones propias pueden ser un sinónimo para la entidad prototipo y no un subtipo.

Todas las instancias de una entidad supertipo deben de pertenecer a una y sola una de las entidades del subtipo. Los subtipos deben formar un grupo completo sin traslaparse:

Para ciertos trabajos, los trabajos deben ser MANUALES o DE OFICINA, pero pueden haber excepciones. (Fig. II.66)

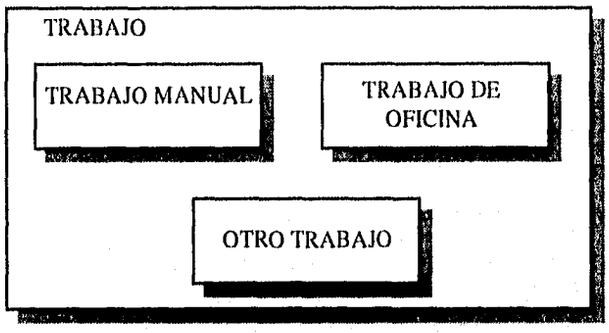


FIG. II.66 Entidad Supertipo con Tres Subtipos

Reglas de lectura de supertipos:

- Cada entidad supertipo debe ser un SUBTIPO 1 o un SUBTIPO 2.

  De la (Fig. II.66) cada trabajo debe ser MANUAL, DE OFICINA, u otro TIPO DE TRABAJO

- ... Subtipo, el cual es un tipo de supertipo,...

  De la (Fig. II.66) "...trabajos de OFICINA, los cuales son un tipo de TRABAJO,..."

Recomendaciones:

- ▲ Siempre se usan subtipos OTROS cuando no se está seguro completamente de los conjuntos

Los subtipos pueden tener además subtipos. Normalmente dos o tres niveles de profundidad son adecuados:

En el diagrama siguiente se definen los subtipos de la entidad subtipo AVION

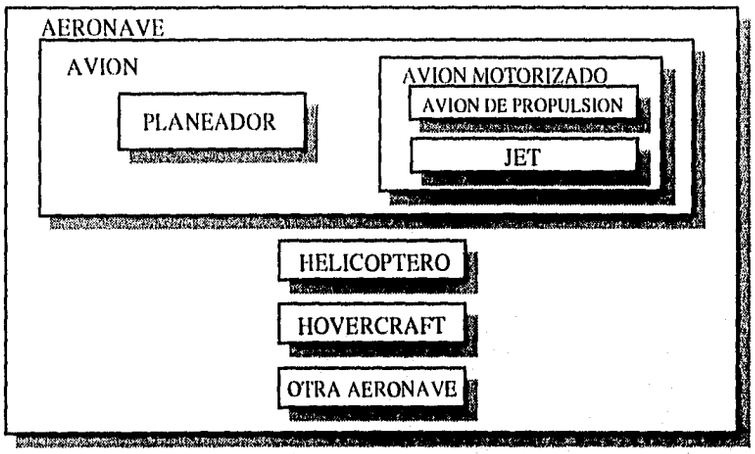


FIG. II.67 Definición de Subtipos de la entidad Subtipo Avión

El AVION es un subtipo de AERONAVE y es un subtipo de AVION MOTORIZADO y PLANEADOR.

El JET hereda los atributos y relaciones de AVION MOTORIZADO, AVION y de AERONAVE

2.3.7 Modelos de Relaciones Exclusivos

Modelar dos o más relaciones mutuamente excluyentes de la misma entidad usando un arco:

Una CUENTA BANCARIA debe pertenecer a una COMPAÑIA o ser PERSONAL. Usar un arco para modelar esta relación. (Fig. II.68)

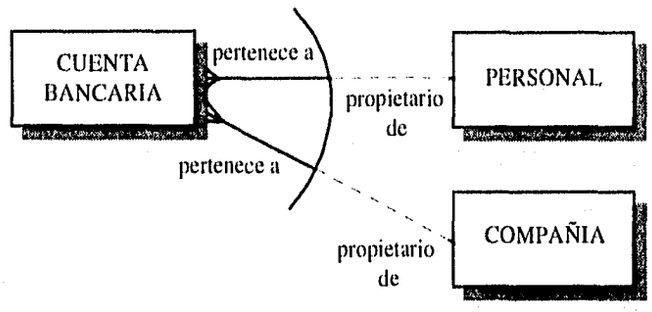


FIG. II.68 Relación Exclusiva

Reglas de lectura para la relación exclusiva:

- Cada entidad A tiene una relación1 con la entidad1.
- Cada entidad A tiene una relación2 con una entidad2

De la figura II.68 cada CUENTA BANCARIA debe pertenecer a una y sólo una PERSONA o pertenecer a una y sólo una COMPAÑIA

Estándares para modelos de arco:

- Las relaciones en arco frecuentemente tienen los mismos nombres de relación
- Las relaciones de arco deben ser o todas obligatorias o todas opcionales.
- Un arco pertenece a una sola entidad, y solamente debe incluir relaciones originales de la entidad.
- Una entidad puede tener múltiples arcos, pero una relación específica solamente puede participar en un sólo arco.

Escoger entre dos convenciones para dibujo de arcos:

Convención Uno. Un arco con puntos opcionales.

Un punto dentro de este arco debe usarse para indicar que una relación pertenece al arco

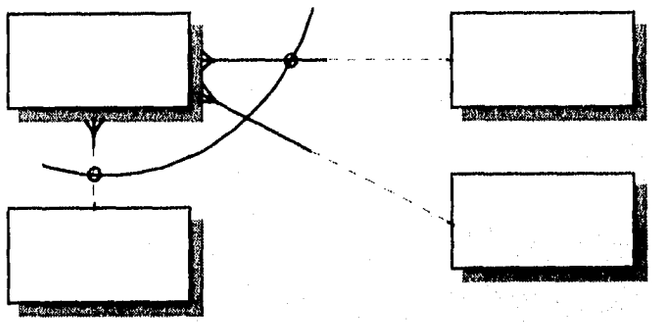


FIG. II.69 Arco con puntos opcionales

Convención Dos. Arco sin puntos

Toda relación cruzada por un arco le pertenece. Un rompimiento en el arco indica una relación que no está incluida en él.

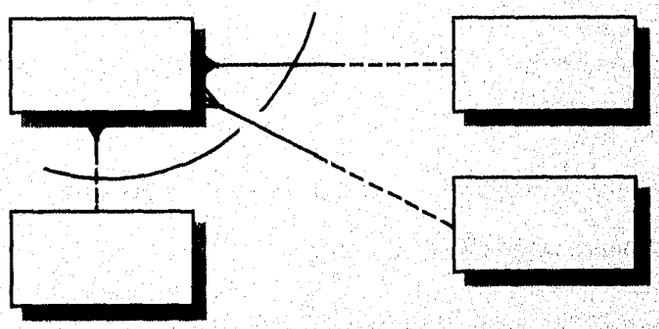


FIG. II.70 Arco sin puntos

2.3.8 Modelos de Datos de Tiempo

Agregar entidades adicionales y relaciones al modelo E-R para acomodar datos históricos:

Preguntas al usuario:

- ❑ ¿Se requiere una auditoria?
- ❑ ¿Los valores de los atributos pueden cambiar en el tiempo?
- ❑ ¿Necesita consultar los valores antiguos?
- ❑ ¿Necesita mantener versiones anteriores?

Recomendaciones:

- ▲ Validar cualquier requerimiento para almacenar datos históricos con el usuario. Almacenar datos históricos innecesarios puede ser costoso

Crear una entidad adicional para mantener los valores de los atributos en el tiempo:

Para una firma de consultoría necesita mantener información acerca de sus contratos. Cada contrato tiene una clave única, y necesitan tener la descripción del contrato, el estatus del contrato (abierto, cerrado o suspendido). Inicialmente fue modelada la siguiente entidad CONTRATO

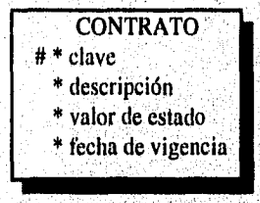


FIG. II.71 Entidad Contrato

La entidad CONTRATO soporta un solo valor de estatus. La firma consultora quiere llevar el registro de las fechas de apertura, cierre o suspensión de cada contrato. Para modelar los valores del estatus en el tiempo agregar la entidad ESTATUS.

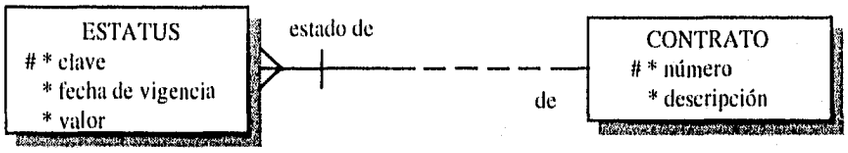


FIG. II.72 Valores de Atributos en el Tiempo

El UID de la entidad ESTATUS es el CONTRATO relacionado y la fecha de vigencia.

Recomendaciones:

- ▲ Usar una sola entidad para grabar los valores en el tiempo de múltiples atributos asociados con una entidad (como CONTRATO)

Agregar una nueva entidad para acomodar la relación que puede variar en el tiempo:

El dueño de unos departamentos quiere saber quienes son los inquilinos de cada uno de los departamentos. (Los departamentos sólo tienen contratos de renta para una sola persona, no con múltiples personas). El siguiente modelo E-R sólo mantendrá al inquilino actual de DEPARTAMENTO.

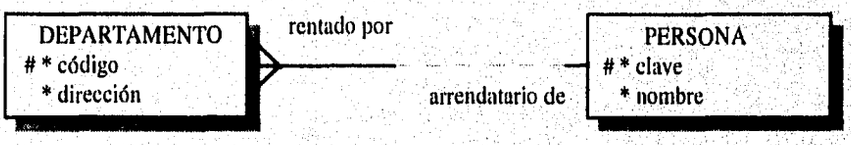


FIG. II.73 Relación entre Departamento y Persona

Agregar la entidad REGISTRO HISTORICO DE RENTA para capturar los valores de las relaciones de renta en el tiempo

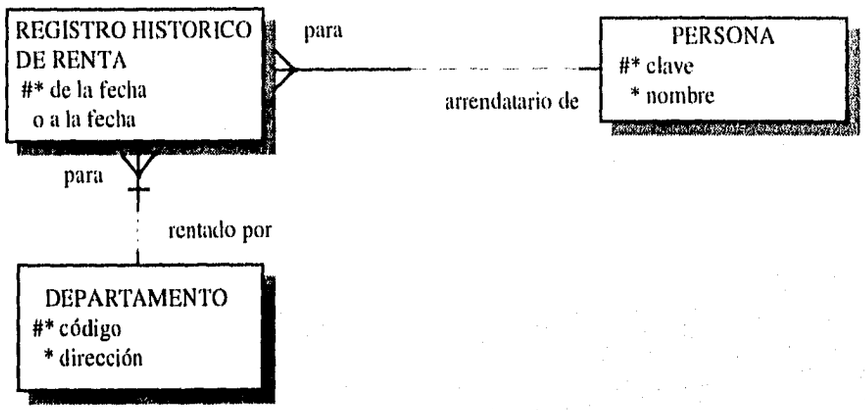


FIG. II.74 Relaciónes de Renta en el Tiempo

Una entidad intersección se usa frecuentemente para mantener información acerca de las relaciones que cambian con el tiempo:

Una sociedad de profesionistas quiere saber las compañías cuyos miembros han sido empleados por tiempo extra y la finalización de cada empleo (por ejemplo de una fecha a otra fecha). Hay una relación M:M entre cada miembro y cada compañía.

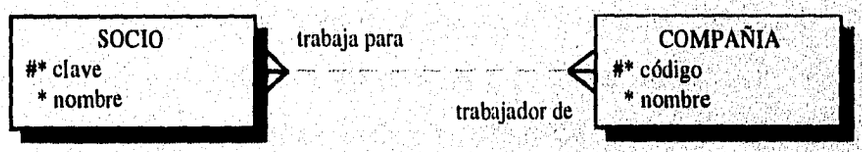


FIG. II.75 Relaciones entre Socio y Compañía

Agregar una entidad intersección, REGISTRO HISTORICO DEL EMPLEADO, para guardar los empleos que cada empleado ha tenido en el tiempo y las fechas de esos empleos.

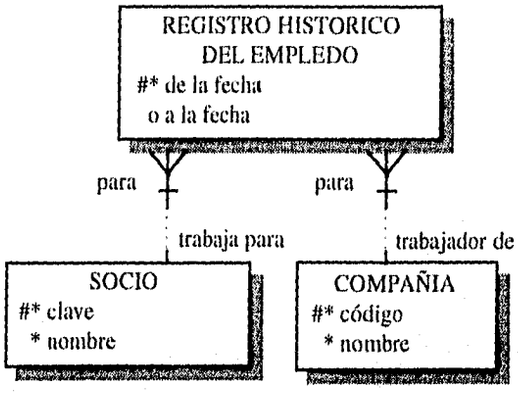


FIG. II.76 Entidad Intersección

Al incluir el atributo de la fecha en el UID del REGISTRO HISTORICO DEL EMPLEADO, este modelo guardará muchas fechas de empleo para una sola compañía y para un sólo empleado.

2.3.9 Modelar Relaciones Complejas

Una relación compleja es una relación entre tres o más entidades. Tener cuidado con las relaciones de anillo M:M:

Desarrollar un modelo E-R para la historia de empleados. Para cada persona, guardar su puesto, la compañía en que trabajó, la fecha de sus puestos obtenidos. Una persona puede tener una posición específica en la

misma compañía múltiples veces durante su carrera. Inicialmente el siguiente modelo E-R fue definido.

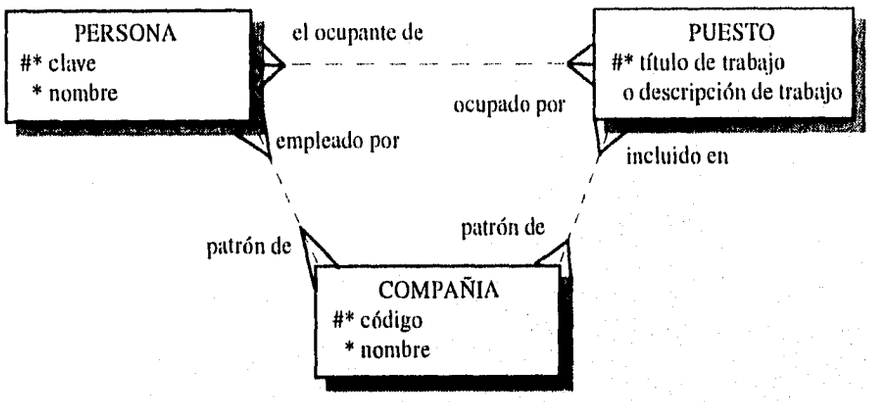


FIG. II.77 Relación de Anillo

Las fechas de la posición parecen ser un atributo de una relación. Con esto, resolver cada relación M:M

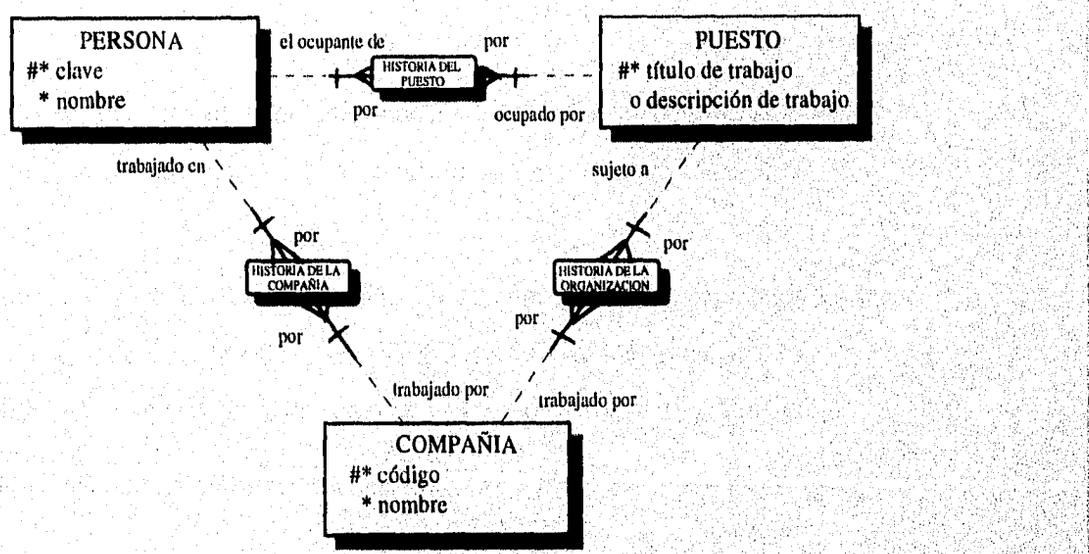


FIG. II.78 Resolución de la relación de Anillo

Modelar una relación entre tres o más entidades como una entidad intersección con relaciones obligatorias con esas entidades:

En un historial de empleos de una persona, existe una relación entre las entidades PERSONA, COMPAÑIA y PUESTO. Usar una entidad intersección llamada HISTORIA DEL EMPLEO para modelar estas relaciones.

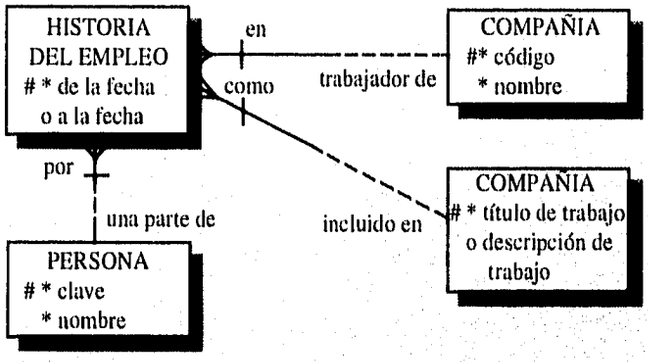


FIG. II.79 Relación entre tres o más Entidades

Recomendaciones:

- ▲ Una entidad intersección para una relación compleja siempre tiene relaciones obligatorias de regreso a las entidades con las cuales se relaciona.
- ▲ Para una representación de una relación compleja de una entidad intersección, seguir las reglas de Modelos E-R básicos para nombrar la entidad, para analizar y modelar sus relaciones, atributos y sus UID.
- ▲ Considerar sus relaciones obligatorias como candidatas para incluirlas.

CAPITULO III

DISEÑO DE LA BASE DE DATOS

Una educación acertada debería ayudar
al individuo a afrontar sus problemas
y a no glorificar el canino de las evaciones.

(Krishnamurti)

Para que un trabajo fácil se le haga muy difícil, basta que lo deje siempre para mañana.

(Edison)

3.1 LA ESTRUCTURA RELACIONAL DE LOS DATOS

Una Base de Datos Relacional es Aquella que es percibida por el usuario como una colección de relaciones o de tablas de dos dimensiones (filas y columnas o lo que es equivalente renglones y atributos), esta es una forma sencilla de estructurar los datos. (Fig. III.1). Es un método muy extendido que lo utilizan los propios usuarios por su simplicidad.

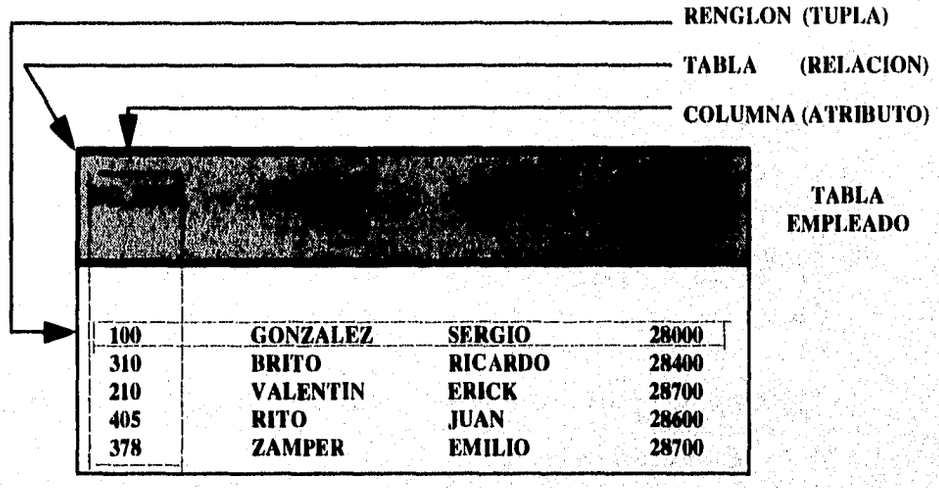


FIG. III.1 En el diagrama se puede observar la tabla compuesta por las Columnas (Atributos) y los Renglones (Tuplas).

Para que una tabla forme parte de una estructura relacional, la tabla debe cumplir las siguientes condiciones:

a) Debe tener un sólo tipo de fila, cuyo formato queda definido por el esquema de la tabla o la relación. Por lo tanto, todas las filas tienen las mismas columnas.

b) Cada columna debe ser única y no pueden existir filas duplicadas.

c) Cada columna debe ser única y no pueden existir columnas duplicadas.

d) Cada columna debe estar identificada por un nombre específico.

e) El valor de una columna para una fila debe ser único. Por lo tanto, no puede existir múltiples valores en una posición de una columna.

f) Los valores de una columna deben pertenecer al dominio que representa, y es posible que un mismo dominio se utilice para definir los valores de varias columnas.

g) Las Tablas de Base de Datos relacional son sencillas pero disciplinadas.

h) Una Base de Datos Relacional debe tener Integridad de Datos, sus datos deben de ser precisos y consistentes.

Una tabla que cumpla estas condiciones se denomina tabla relacional o relación. El concepto relación se utiliza generalmente para indicar que en la tabla relacional se mantiene la asociación de otras tablas y por tanto representa a una entidad asociativa. El concepto tabla relacional se utiliza más bien para expresar que los atributos que contiene no relacionan objetos fuertes y representativos, y la tabla, por tanto representa por sí misma un objeto o entidad propia.

Una tabla que cumpla las condiciones anteriores tiene asociadas las siguientes propiedades:

1) Las filas pueden estar en cualquier orden.

2) A una fila se le hace referencia mediante todos los valores que la forman

3) Las columnas pueden estar en cualquier orden.

4) Se hace referencia a una columna mediante el nombre que la identifica.

En una tabla relacional o relación, a las filas se les denomina **tuplas** y a las columnas **atributos**. Por lo tanto, siempre que se haga referencia al modelo relacional se nombrarán con esta terminología.

Se denomina **Grado** de una tabla relacional al número de atributos que la forma. De ese modo, en la tabla "empleados", G(empleados) = 4

Se denomina **Cardinalidad** de una tabla relacional al número de tuplas que contiene. Por tanto, en la tabla "empleados", C(empleados) = 5

3.2 DOMINIOS

El dominio es el conjunto de todos los posibles valores para una o más columnas de una tabla relacional. Por tanto, los valores contenidos en una

columna de una tabla relacional pertenecen a un dominio que previamente se define.

Puede distinguirse dos tipos de dominios diferente:

a) **Dominio generales o continuos:** aquéllos que contienen todos los posibles valores entre un máximo y un mínimo predefinido. Por ejemplo:

- ❑ DNI: todos los números enteros y positivos de ocho dígitos.
- ❑ Peso del material: todos los numeros reales y positivos.
- ❑ Saldo de la cuenta: todos los números reales, positivos y negativos.
- ❑ Fecha de nacimiento: todas las fechas desde principio de siglo hasta hoy

b) **Dominio restringidos o discretos:** aquéllos que contienen ciertos valores específicos entre un máximo y un mínimo predefinido. Por ejemplo:

- ❑ Estado civil: compuesto por soltero, casado, viudo, divorciado
- ❑ Estado : uno de los Estados de la República Mexicana.

En un sistema de gestión de bases de datos integrado, generalmente los distintos usuarios deben utilizar diferentes subconjuntos del universo total de los datos.

El **Modelo de Dato** denota el universo total de los datos, es decir, el conjunto completo de todas las tablas relacionales almacenadas en la base de datos. **Esquema** es el conjunto de declaraciones que describen el **Modelo**. **Submodelo** es el conjunto de relaciones a las que puede acceder un usuario determinado, y **Subesquema** las declaraciones correspondientes al **Submodelo**.

3.3 CLAVES

En una tabla relacional es necesario poder determinar una tupla concreta, lo cual es posible mediante la clave (Llave Primaria (PK)). La clave (Llave Primaria (PK)) es un atributo(Columna) o conjunto de atributos (Columnas) cuyos valores distinguen de manera única a cada tupla (Renglón) específica de una tabla.

Como una de las condiciones de una tabla relacional es que no pueden existir filas duplicadas, ello implica que siempre tiene que existir al menos una clave(Llave Primaria (PK)), que en el peor de los casos estará formada por todos los atributos.

Para buscar la clave(Llave Primaria (PK)) de una tabla relacional nos debemos mover por los dominios de los atributos(Columnas) junto con los

enlaces conceptuales existentes entre ellos, de tal modo que tomando unos valores determinados se identifique una única tupla (Renglón) de la tabla. No se debe buscar la clave entre los valores de unas tuplas (Renglones) concretas que tengamos de ejemplo, sino a través de todos los posibles valores (dominio) de los atributos(Columnas). En la tabla "Personal" (Fig. III.2), la clave o Llave Primaria (PK) es DNI puesto que cualquier valor del dominio de dicho atributo o columna identifica a un único Renglón. Por tanto, no puede aparecer un mismo valor del DNI en dos Renglones diferentes.

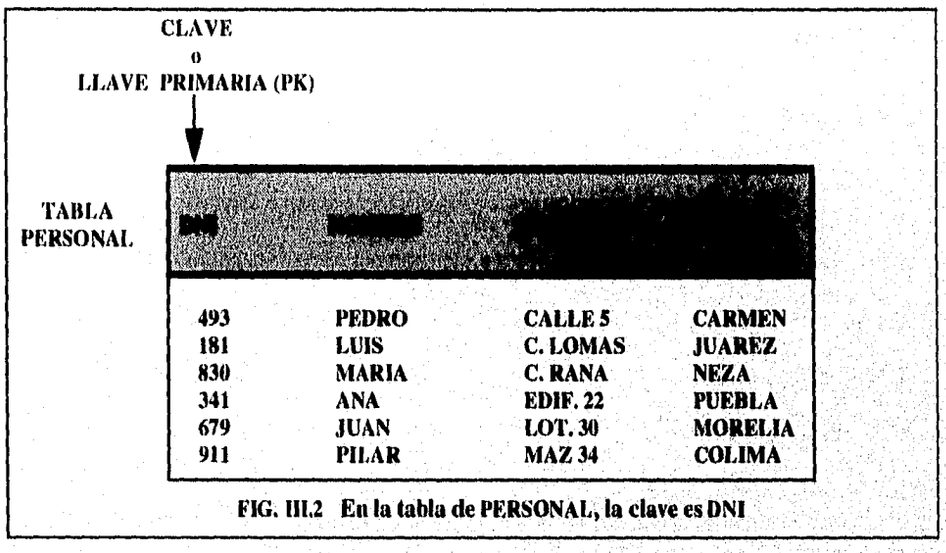
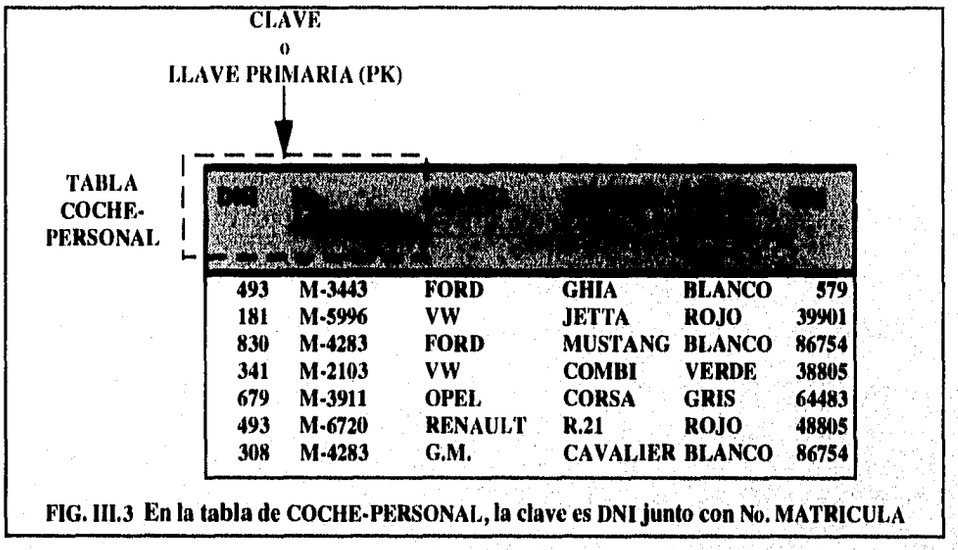


FIG. III.2 En la tabla de PERSONAL, la clave es DNI

En la tabla COCHES-PERSONAL (Fig. III.3) se representan los diferentes coches que puede disponer un individuo determinado. Un coche concreto que se identifica por el "No de matrícula" (sería la clave o llave primaria (PK) de otra tabla" coches") puede estar compartido por varios individuos, por lo tanto, el No de matrícula no identifica a un renglón determinado puesto que

el coche se puede encontrar en distintos individuos. El DNI tampoco nos sirve para localizar un único renglón, puesto que un individuo puede disponer de más de un coche (por ejemplo, el DNI "493"). Lógicamente, la marca, el modelo, el color o los kilómetros tampoco nos sirven para distinguir dos tuplas o renglones puesto que distintos coches pueden tener la misma marca, el mismo modelo, el color o incluso los mismos kilómetros.



493	M-3443	FORD	GHIA	BLANCO	579
181	M-5996	VW	JETTA	ROJO	39901
830	M-4283	FORD	MUSTANG	BLANCO	86754
341	M-2103	VW	COMBI	VERDE	38805
679	M-3911	OPEL	CORSA	GRIS	64483
493	M-6720	RENAULT	R.21	ROJO	48805
308	M-4283	G.M.	CAVALIER	BLANCO	86754

FIG. III.3 En la tabla de COCHE-PERSONAL, la clave es DNI junto con No. MATRICULA

Por lo tanto, la clave o llave primaria debe estar formada por más de un atributo o columna y será DNI junto con No de MATRÍCULA. Esto quiere decir que de todas las tuplas o renglones que tenga la tabla, se distinguirán entre sí mediante los valores concatenados del DNI y No de MATRÍCULA, o lo que es igual, si conocemos el valor de un DNI y de un No de MATRÍCULA asociado identificaremos a una única tupla o renglón. Una Llave Primaria que consta de múltiples columnas se llama Llave Primaria Compuesta. Y las columnas

de una Llave Primaria Compuesta deben de ser únicas en combinación. Las columnas pueden tener duplicados en forma individual, pero en combinación no se permiten duplicados.

Es posible que en una tabla, más de una columna (o combinación de ellas) puedan servir de clave o llave primaria (PK). Por ejemplo, si en la relación PERSONAL sabemos que los valores del nombre no se repiten para distintos individuos, es decir, que no existen dos individuos con el mismo nombre, ello indica que el atributo NOMBRE también es clave.

Si una tabla dispone de varias claves o llaves primarias (PK), a éstas se les denomina claves candidatas o Llaves Candidatas, puesto que más adelante entre todas ellas tendremos que elegir una que será la que por excelencia identifique la tupla o renglón. A esta clave se le denomina clave principal o llave primaria y al resto claves alternas o llaves alternas. Los atributos que pertenecen a la clave primaria se denominan atributos primarios y el resto atributos no primarios o secundarios.

Para elegir la llave primaria nos basaremos fundamentalmente en el dominio, puesto que en la futura gestión de las bases de datos la llave primaria se utiliza para acceder y relacionar otras tablas (como es nuestro caso con la relación "coches-empleados"). Cuando vayamos a analizar la llave primaria, tendremos en cuenta sobre su dominio las siguientes consideraciones:

a) Sus valores siempre debe ser conocidos y únicos (diferentes de nulos).

b) La memoria que ocupen debe ser mínima

c) Las Llaves Primarias generalmente no se pueden cambiar

d) El UID de una entidad irá de acuerdo con la Llave Prímaria en su tabla correspondiente.

e) Su codificación ser sencilla

f) El contenido de sus valores no debe variar.

g) Que se utilice en otras tablas para crear una interrelación (mediante claves ajenas o llaves foráneas).

h) Los nombres de personas normalmente no son Llave Candidata. Porque no se puede garantizar que sean únicas.

De ese modo, en la tabla de PERSONAL las claves o llaves candidatas son:

- ❑ DNI
- ❑ NOMBRE

De ambas, el atributo cuyos valores ocupan menos memoria es el DNI puesto que el dominio del NOMBRE será de tipo carácter de 30 posiciones (por ejemplo). Además, por ese motivo, los accesos serán más rápidos, aparte

de que no se necesita ningún tipo de algoritmo para codificarlo. También hay que tener en cuenta que al dar de alta un individuo podemos equivocarnos en el nombre y éste debe ser posible modificarlo. Por lo tanto:

Clave principal de PERSONAL:

- ❑ DNI

Clave alternativa de PERSONAL:

- ❑ NOMBRE

Por ese motivo, el atributo DNI es el que se utiliza para crear la interrelación entre la tabla de PERSONAL y COCHES-PERSONAL.

Las claves se utilizan para en un futuro definir índices sobre ellas. Los índices son las formas de acceso. El índice primario (generalmente obligatorio) estará formado por la llave primaria (que puede ser **simple** estar formada por un sólo atributo, **compuesta** estar formada por varios atributos, y los índices secundarios (si los hay) lo formarán las claves secundarias u otros atributos que nos interesen. Estos últimos índices sólo se usan si en transacciones on-line (con tiempo de respuesta limitado) se precisa acceder a los renglones por otros atributos. Estos generalmente sólo se aplica para realizar consultas diversas.

Hay numerosas ocasiones en que se necesita acceder a la información o tenerla agrupada mediante unos atributos que no son claves; por ejemplo, en PERSONAL mediante el atributo CIUDAD, para tener al personal agrupado por ciudades y poder consultar todos los que son de "GUADALAJARA"; para ello,

también se generan índices. En este caso, se denominan índices secundarios múltiples por que los atributos que los forman no son clave, y sus valores se repiten dentro de la tabla índice. Si el sistema donde se desea implantar la base de datos no permite el uso de índices múltiples, el truco consiste en generar el índice concatenando al final la clave o llave(principal o alternativa) para de esa forma ser también una clave o llave alternativa y que los valores del índice no se repitan; además, de ese modo, se puede consultar ordenados por ciudad y dentro de este atributo tener algún criterio de clasificación: nombre, por ejemplo.

- ❑ Todas las llaves candidatas deben de ser únicas y no nulas.(Fig. III.4)
- ❑ Los UID secundarios concuerdan con las llaves alternas
- ❑ Los nombres de personas normalmente no son llaves candidatas porque no se puede garantizar que sean únicas.

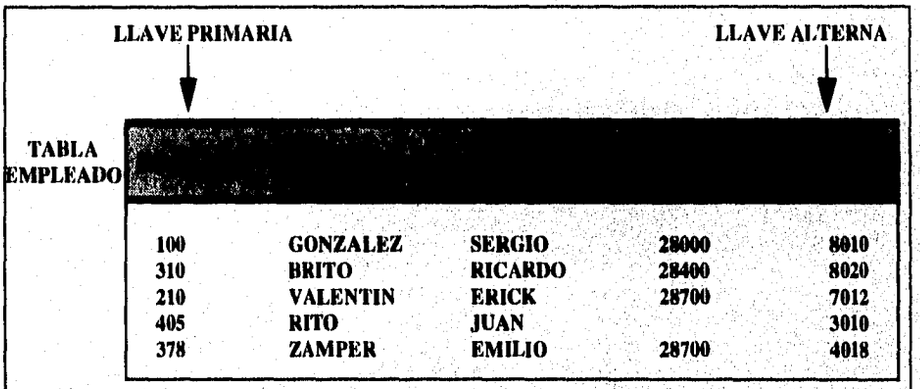


FIG. III.4 En la tabla de EMPLEADOS la llave primaria es la columna NO_CLAVE y la columna NOMINA es una candidata para ser llave primaria

Se denomina clave ajena o llave foránea al atributo o conjunto de atributos que en la tabla donde se encuentran no son Llave Primaria, y sus valores se corresponden con la Llave Primaria de la misma u otra tabla.(Fig. III.5) El dominio de la llave foránea y de la llave primaria de la tabla a la que hace referencia se tiene que corresponder, es decir, ser compatible. Por lo tanto el atributo DNI por ser Llave Primaria en la tabla PERSONAL (Fig. III.2) y estar en una columna de la relación COCHES-PERSONAL (Fig. III.3), es llave foránea en esta última tabla.

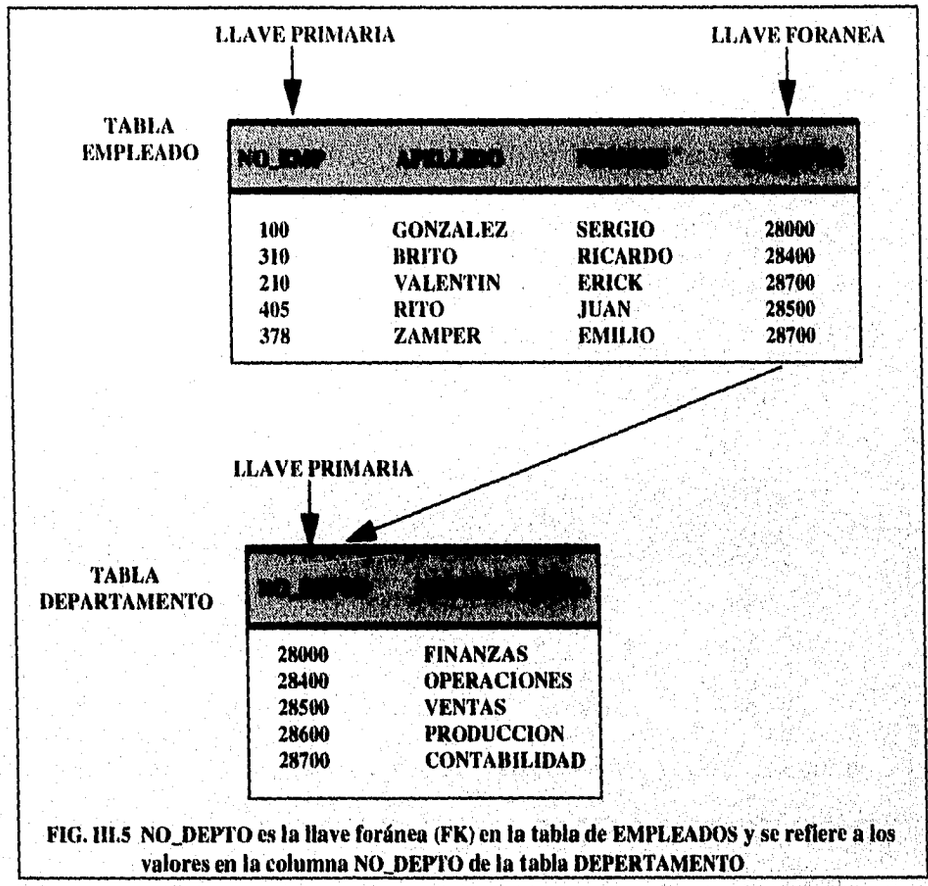


FIG. III.5 NO_DEPTO es la llave foránea (FK) en la tabla de EMPLEADOS y se refiere a los valores en la columna NO_DEPTO de la tabla DEPERTAMENTO

Una llave foránea (FK) es una columna o combinación de columnas en una tabla, que se refieren a una llave primaria en la misma o en otra tabla. (Fig. III.6).

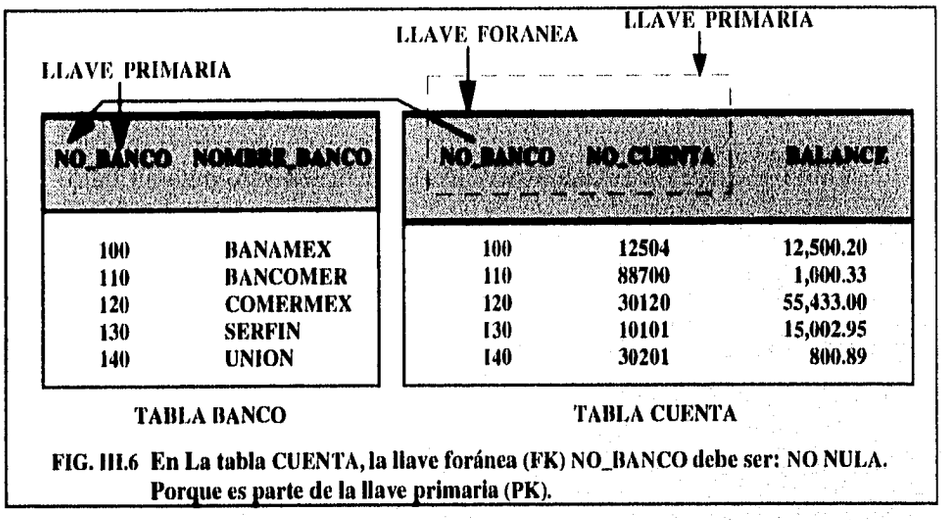


FIG. III.6 En La tabla CUENTA, la llave foránea (FK) NO_BANCO debe ser: NO NULA. Porque es parte de la llave primaria (PK).

Tener en cuenta las siguientes consideraciones para las llaves foráneas:

a) Las claves ajenas o llaves foráneas son un concepto muy importante a tener en cuenta, puesto que la integridad de la base de datos se mantiene o se elimina en una gran parte mediante las transacciones que se realizan sobre dichas claves o llaves.

b) Las llaves foráneas son utilizadas para hacer "join" entre tablas

c) Las llaves foráneas se basan en los valores de los datos y son puramente lógicas.

d) Una llave foránea debe coincidir con un valor de una llave primaria existente.

e) Si una llave foránea es parte de una llave primaria. La FK no puede ser nula

3.4 INTERRELACIONES

Una interrelación es una asociación entre tablas mediante atributos que tienen el mismo dominio (o compatible). Estos atributos generalmente hacen referencia a los mismos conceptos y la interrelación se establece entre la clave ajena o llaves foráneas (FK) de una tabla (tabla hija), y la clave principal o llave primaria (PK) de la otra tabla (tabla padre). Por tanto, en el ejemplo analizado en el punto anterior existe una interrelación entre la tabla "PERSONAL" (tabla padre, puesto que el DNI es clave principal) y la tabla "COCHES PERSONAL" (tabla hija, puesto que el DNI forma parte de la tabla). Esta interrelación se caracteriza por medio de lo siguiente:

- Interrelación: "propiedad-coche"
- Tabla padre: "PERSONAL"
- Tabla hija: "COCHES.PERSONAL"
- Clave ajena o llave foránea (FK) "DNI"

Las interrelaciones se identifican con un nombre, en nuestro caso "propiedad-coche"

3.5 VISTAS

Las tablas que hemos estado viendo hasta ahora, son tablas reales, cuyo esquema o definición, así como sus tuplas o datos están almacenados físicamente en memoria, es decir, son tablas que están implementadas directamente dentro de una base de datos. Estas son las tablas base.

Pues bien, además de estas tablas, el usuario puede crear y manejar (de forma limitada con algunas restricciones según el SGBDR que se use) otro tipo de tablas. Son tablas ficticias cuya definición y tuplas se obtienen a partir de una o más tablas base. Son tablas que obtienen vistas parciales de datos para que usuarios sólo accedan a determinada información.

La tablas vistas, o simplemente vistas, tienen las siguientes características:

- Sus columnas se obtienen a partir de multiples tablas base e incluso pueden estar calculadas a partir de valores de las tablas base. En la (Fig. III.7), la vista V12
- Pueden estar definidas a partir de otras vistas. En la vista V12. (Fig. III.7).
- Sus datos se obtienen como resultado de realizar operaciones de recuperación (lectura) de datos
- Se puede almacenar su definición (esquema o estructura) para una utilización posterior.

Por lo tanto, la vista es una tabla virtual que no existe en realidad como una tabla base en memoria auxiliar (disco, por ejemplo); sólo se almacena, si se desea, su definición, donde se muestra cómo se obtiene a partir de tablas base mediante una sucesión de operaciónes. Es una forma de ver determinados datos de tablas base. Por lo tanto, tal y como se manipulen las tablas base con modificaciónes, inserciones y borrados, así quedarán afectadas las vistas (visiones) que se tengan sobre ellas. Es importante resaltar que los datos que se obtienen de una vista no son datos duplicados.

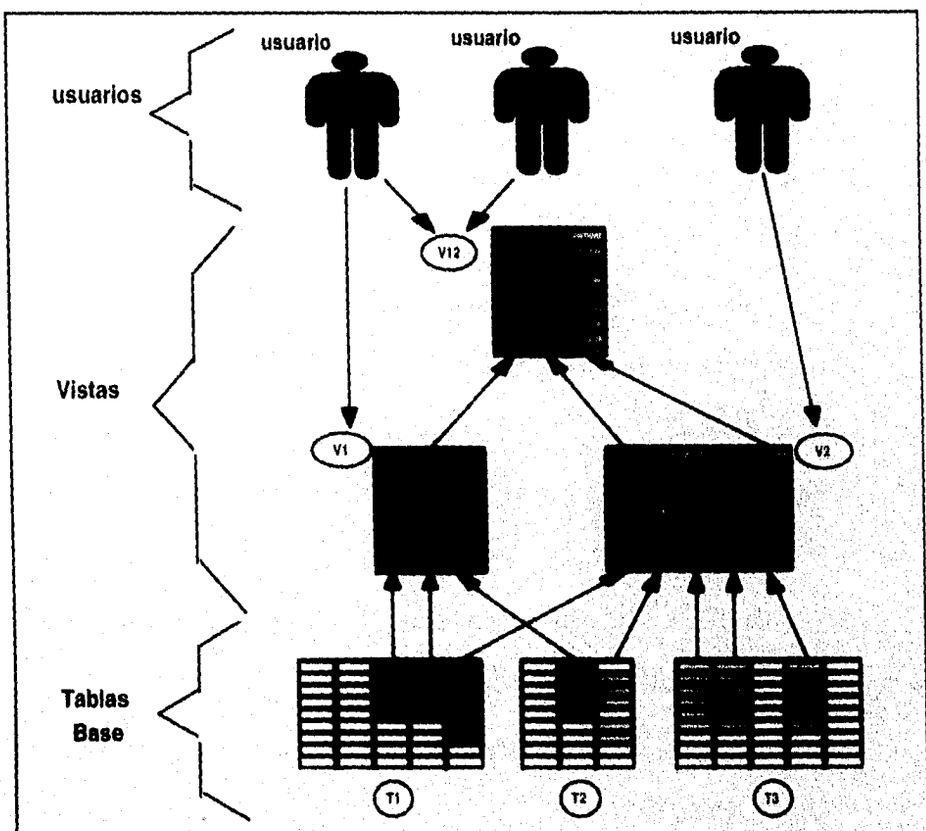


FIG. III.7 Representación de diferentes vistas partiendo de la tabla base para que el usuario final pueda observar solamente una parte de la información

Las operaciones de actualización (MIB) de datos sobre vistas, son:

a) Modificación

b) Inserciòn

c) Borrado

Por las implicaciones tan enormes que pueden tener sobre las tuplas de las tablas base, y el resto de vistas definidas sobre ellas, obligan a que la propia definición de la vista tenga sus propias restricciones para la realización de dichas operaciones. Estas restricciones son diferentes según el sistema de gestión de la base de datos (SGBD) que se use.

Una vista que vaya a admitir las operaciones de actualización (el término actualización indica las tres operaciones fundamentales de variación de datos: modificación, inserción y borrado "MIB") debe estar limitada por las siguientes restricciones:

a) Debe estar derivada de una sóla tabla base

b) Cada fila o tupla distinta de la vista se debe corresponder con una única tupla de la tabla base.

c) Cada columna o atributo distinto de la vista se debe corresponder con un único atributo de la tabla base (no puede tener columnas calculadas).

Diversos diseñadores de SGBD estudian y analizan continuamente situaciones donde las normas a pesar de ser válidas, el sistema no debe admitir determinados tipos de actualización (Inserciones y Borrados). En otros sistemas se deja libertad al administrador de la base de datos de ser el que se responsabilice de las operaciones que se permitan realizar sobre una vista, y el SGBD no tiene unas restricciones determinadas. Algunas de las caracteristicas de las vistas son:

a) Las principales ventajas que ofrecen las vistas son:

- ❑ La visión de los datos está simplificada de cara al usuario.
- ❑ Los mismos datos pueden verse de diferente manera desde distintos usuarios.

b) Las vistas no quedan afectadas tras:

- ❑ Aumento de otras tablas.
- ❑ Aumento de la estructura con nuevos atributos.

c) Reestructuración de la posición de los atributos de una tabla.

d) Se aumenta la seguridad para aquella información que no se desea mostrar.

3.6 INTEGRIDAD DE DATOS

La integridad de datos se refiere a la exactitud y consistencia de los datos.

Todos los constraints de integridad de datos (Fig. III.8) deben ser forzados por el DBMS o el software de aplicación a que cumplan las reglas del negocio, estas determinan el estado correcto de una base de datos. A su vez estas reglas de negocio son llamados Constraints de Integridad de Datos Definidas por el usuario.

Los constraints de integridad de datos definen el estado relacional correcto de la base de datos, aseguran que los usuarios realizarán únicamente operaciones en las cuales dejarán a la base de datos en un estado correcto y consistente.

TIPO DE CONSTRAINT	EXPLICACIÓN
Integridad de Entidades	Ninguna parte de la llave Primaria puede ser NULA
Integridad Referencial	Una llave foránea debe coincidir con un valor de una llave Primaria.
Integridad de Columnas	Una columna debe contener sólo valores consistentes con el formato de datos definido para la columna.
Integridad definida por el Usuario	Los datos almacenados en la base de datos deben cumplir con las reglas del negocio.

FIG. III.8 Tabla de Constraint. de Integridad de Datos

Un dato es inconsistente si existen multiples copias de un registro y no todas las copias han sido actualizadas, una base de datos inconsistente puede proveer información incorrecta o contradictoria a los usuarios.

Por ejemplo un negocio tiene las siguientes reglas de constraints de integridad de datos definida por el usuario.

- A un empleado de confianza no se le paga por las primeras cinco horas de trabajo de tiempo extra.
- Ningún empleado del Departamento de Finanzas no puede llevar el título de "programador".
- La comisión de los vendedores no puede exceder del 50% de su salario.

Los constraints de los datos definidos por el usuario pueden ser administrados por políticas o ser requeridos por las leyes gubernamentales. Y pueden incluir múltiples columnas y tablas. Frecuentemente esas reglas son completamente arbitrarias o al menos parecen ser arbitrarias.

3.7 DISEÑO DE BASE DE DATOS

El Diseño de Base de Datos se lleva a cabo por medio de dos actividades:

1) Pasar el modelo E-R a tablas relacionales para producir el diseño inicial

2) Refinar el diseño inicial para producir un diseño completo de la Base de Datos. Y asi poder Liberar el Diseño de la Base de Datos.

La etapa de Diseño de la base de Datos define especificaciones de diseño para una Base de Datos Relacional, incluyendo definiciones para tablas relacionales, índices, vistas y espacio de almacenamiento.

3.7.1 Introducción al Diseño Inicial de la Base de Datos.

Documentar cada tabla relacional en un mapa de Instancias.(Fig. III.9):

	PK							FK1	FK2
	NN,U	NN	NN		NN				NN
	12504	SERGIO	GONZALEZ	ARCHIVISTA	07-JUN-95	1000		7902	28400
	6006	RICARDO	BRITO	ANALISTA	10-OCT-95	3000		7566	28700
	10020	ERICK	VALENTIN	GERENTE	13-SEP-95	4850	9000	7839	28500
	80005	JUAN	RITO	VENDEDOR	10-NOV-95	1500	5000	7698	28600
	40321	EMILIO	ZAMPER	PRESIDENTE	08-AGO-95	6000	4000		28000

FIG. III.9 Mapa de Instancias de la tabla EMPLEADO

a) Los tipos válidos de llaves son PK para una columna llave primaria, y FK para la columna llave foránea.

b) Usar sufijos para distinguir entre múltiples columnas FK en una tabla, por ejemplo, FK1 y FK2 Etiquetar múltiples columnas con el mismo sufijo.

c) Usar NN para una columna que debe ser definida como NO NULA

d) Usar U para la columna que debe ser única.

e) Si múltiple columnas deben ser únicas en combinación, etiquetarlas con un sufijo, por ejemplo U1.

f) Etiquetar una columna sencilla PK como NN,U

g) Etiquetar múltiples columnas PK como NN,U1 o como NN,U1,U.

Seguir una serie de pasos para dibujar un modelo E-R para una serie de Tablas y producir un diseño inicial de Base de Datos.

Pasos en el diseño inicial de la Base de Datos:

1) Mapear las entidades para las tablas.

2) Mapear atributos para columnas y documentar datos simples.

3) Mapear identificadores únicos a llaves primarias.

4) Mapear relaciones a llaves foráneas.

5) Elegir opciones de arco.

6) Elegir opciones de subtipo.

Utilizaremos el modelo E-R de la Compañia de Entrenamiento, y será usado para ilustrar las actividades del diseño inicial de la Base de Datos.

3.7.2 Mapear las Entidades

Mapear la tabla para cada entidad. Crear un mapa de Instancias para la nueva tabla. Registrar únicamente el nombre de la tabla.

Por ejemplo.(Fig. III.10) Crear un mapa de instancias para la entidad INSTRUCTOR. El nombre de la tabla será INSTRUCTOR.

- El nombre de la tabla debe ser fácil de identificar con el nombre de la entidad. El nombre en plural de una entidad se usa algunas veces porque la tabla debe contener un grupo de renglones.

- Una entidad simple no es un subtipo o supertipo. En el paso 6 (Inciso "F" anterior) el diseñador debe decidir como se hace una construcción supertipo

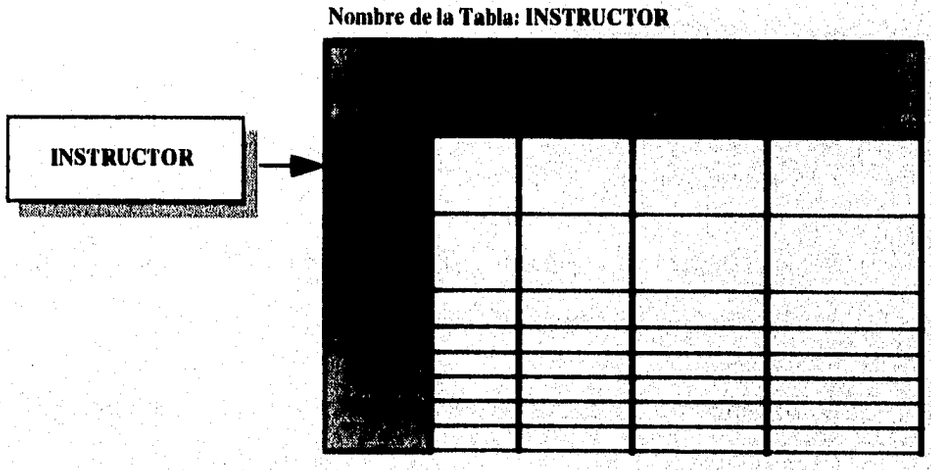


FIG. III.10 Mapear las Entidades (Mapa de Instancia)

3.7.3 Mapear Atributos a Columnas

Mapear cada atributo de la entidad a una columna en su tabla correspondiente. Establecer los atributos obligatorios para columnas NO NULAS(NN).

Por ejemplo (Fig. III.11). Asignar los atributos de la entidad INSTRUCTOR a columnas en la tabla INSTRUCTOR marcar como atributos obligatorios (NO NULAS) las columnas INST_ID, NOMBRE y APELLIDO.

a) Para cada atributo seleccionar un nombre corto pero significativo

b) El nombre de las columnas debe ser fácil de identificar en un modelo E-R

c) Prevenir al usuario de no usar las palabras reservadas de SQL para nombre de columnas. Por ejemplo NUMBER.

d) Usar abreviaciones consistentes que no causen confusión al usuario y al programador. Por ejemplo. ¿Podría ser abreviado NUMERO como NO o NUM y DEPTNO o DEPTNUM?.

e) Los nombres de columnas cortos o pequeños reducirán el tiempo requerido para el comando de SQL "parsing"

f) Documentar renglones de ejemplo de datos en cada mapa de instancias.

Por ejemplo (Fig. III.11). Documentar datos del ejemplo para las columnas de la tabla INSTRUCTOR.

Fuentes para Datos del Ejemplo:

a) Notas de entrevistas a usuarios.

b) Mapa de instancias

c) Sistemas actuales de computadoras

d) Otros documentos de análisis

e) Conversaciones adicionales con el usuario.

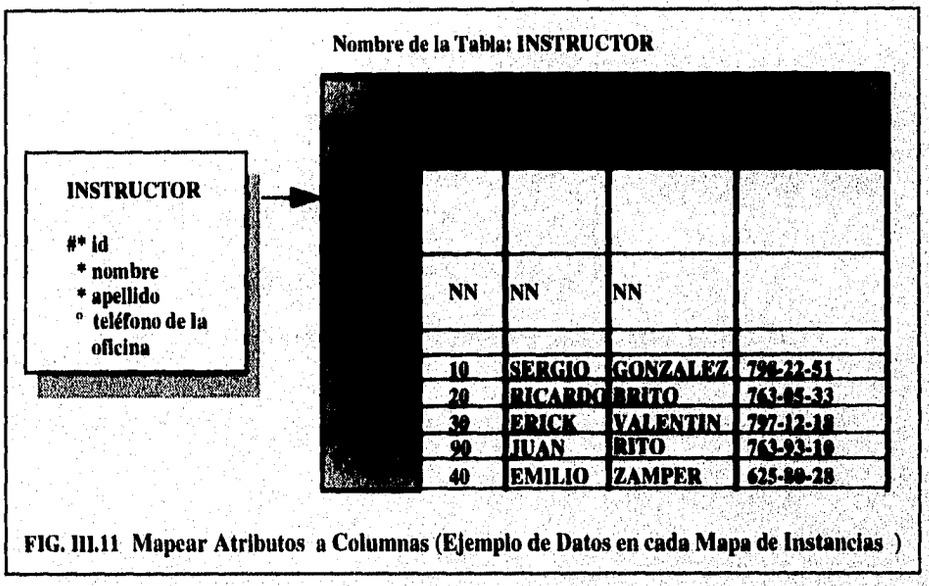


FIG. III.11 Mapear Atributos a Columnas (Ejemplo de Datos en cada Mapa de Instancias)

3.7.4 Mapear UIDS a Llaves Primarias (PK)

Asignar cualquier atributo (S) que sea parte del UID de la ENTIDAD a columnas PK. Etiquetar las columnas con "PK". Un tipo de llave PK indica una columna de llave primaria. Todas las columnas etiquetadas con PK deben de etiquetarse también con NN y U

Por ejemplo (Fig. III.12) . El atributo ID es el UID de la entidad. INSTRUCTOR. Entonces hacer que la columna correspondiente INST_ID sea el PK de la tabla INSTRUCTOR.

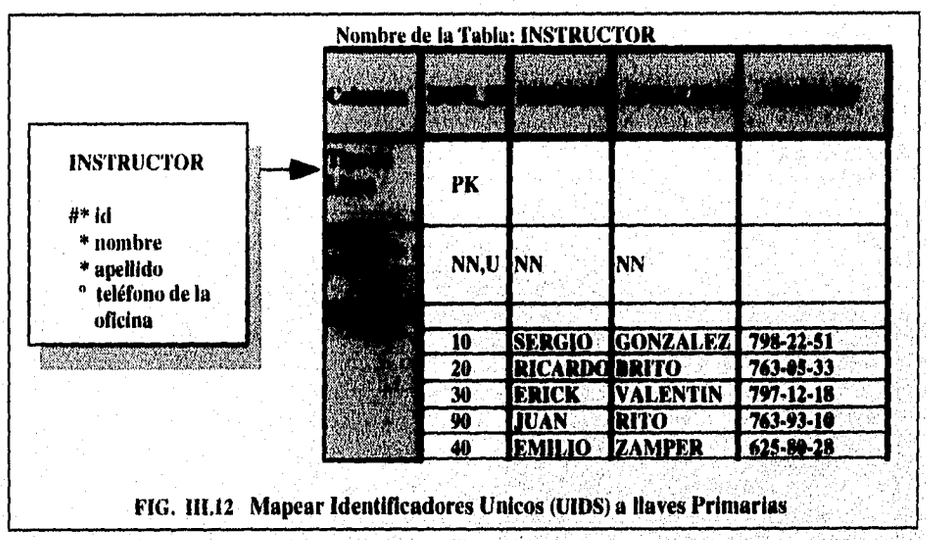


FIG. III.12 Mapear Identificadores Unicos (UIDS) a llaves Primarias

Asignar un UID que incluya multiples atributos, a una PK compuesta. Etiquetar estas columnas como NN y U1.

Si una entidad incluye una relación, agregar columnas de llaves foráneas para la tabla y señalarlas como parte de la llave primaria.

Por ejemplo (Fig. III.13). El UID de la entidad MATRICULA está compuesta de una relación para CURSO y la relación para ESTUDIANTE. Agregar dos columnas FK a la tabla MATRICULA para el PK de la tabla CURSO y para el PK de la tabla ESTUDIANTE.

a) Escoger un nombre único para cada columna FK y etiquetar la (s) columna (s) PK,NN y FK.

b) Las PK compuestas deben de ser únicas en combinación y deben de ser etiquetadas como U1.

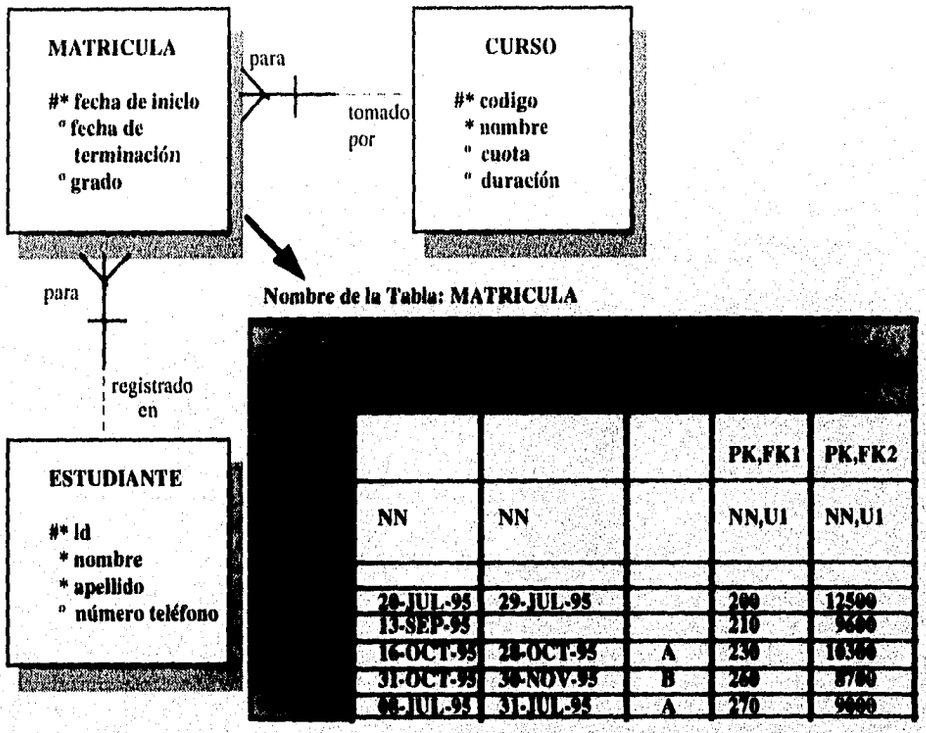


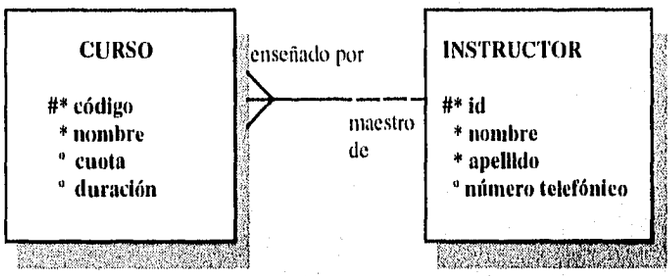
FIG. III.13 El UID de la MATRICULA esta compuesta de una Relación para CURSO y la Relación para ESTUDIANTE.

c) Si existen múltiples columnas FK en una tabla usar sufijos para distinguirlos, por ejemplo FK1 y FK2. Etiquetar múltiples columnas llave con el mismo sufijo

d) Agregar ejemplos de datos para las columnas FK.

3.7.5 Mapear Relaciones para Llaves Foráneas

a) Para una relación de entidades M:1 tomar el PK de la tabla (1) y ponerlo en la tabla (M). Por ejemplo (Fig. III.14). Poner el PK INST_ID de la tabla 1 y ponerlo en la tabla CURSOS (Tabla M).



Nombre de la Tabla: CURSOS

	PK				FK
	NN,U	NN			NN
	200	UNISYS	1500	80	10
	109	SQL-PLUS	2500	40	45
	310	WINDOWS	500	60	19
	120	EXCEL	600	80	89
	410	WORD	500	40	60

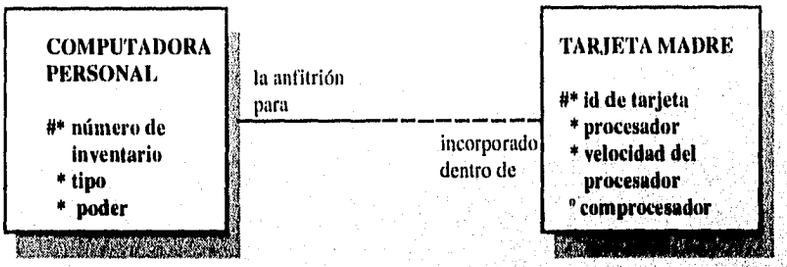
FIG. III.14 Mapear Relaciones para llaves Foraneas

b) Elegir un nombre único para la columna FK y etiquetar la(s) columna(s) FK.

c) Para las relaciones debe ser, etiquetar la columna como NN.

d) Agregar datos de ejemplos.

e) Si el PK de la tabla incluye una llave foránea (FK), las columnas FK que soportan la relación puede ser agregada en el paso 3.7.4 Por ejemplo (Fig. III.13). La PK para la tabla MATRICULA incluye ambas, la llave foránea CODIGO_CURSO y la llave foránea ESTUD_ID. Por consiguiente, estas dos columnas ya existen y no necesitan ser agregadas para soportar la relación



Nombre de la Tabla: COMPUTADORA PERSONAL

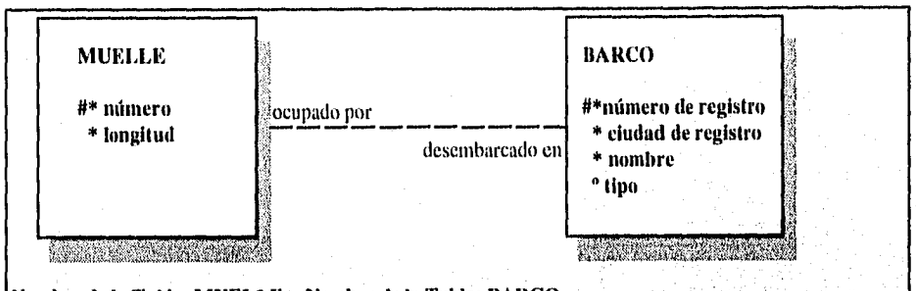
PK			FK
NN,U	NN	NN	NN,U
200	AT	1500	8080
109	AT	2500	4080
310	XT	500	6080
120	XT	600	8081
410	XT	500	4081

Nombre de la Tabla: TARJETA MADRE

PK			
NN,U	NN	NN	NN
8081	386	33	N
8080	486	25	N
4080	486	50	N
6080	386SX	33	Y
4081	386	25	Y

FIG. III.15 Mapear Relaciones para llaves Foráneas

f) Para una relación obligatoria 1:1, colocar las FK como únicas en el mapa de instancias en donde la relación es obligatoria y usar el constraint NO NULO para forzar la condición obligatoria.(La FK para la relación 1:1 debe ser siempre única, pero permite NULOS). Por ejemplo. (Fig. III.15). Ya que la relación de COMPUTADORA PERSONAL es obligatoria, colocar la FK para la relación en la tabla COMPUTADORA personal, y etiquetarla como NO NULA ID_TAR es la columna FK agregada. La FK es etiquetada U para forzar la relación 1:1.



Nombre de la Tabla: MUELLE

	PK	
	NN,U	NN
	200	100
	109	1000
	310	500
	120	600
	570	80
	820	110

Nombre de la Tabla: BARCO

PK	PK				PK
NN,U1	NN,U1	NN			U
128X1S	CUBA	PERLA	CRUCERO		109
M19L2	E.U	DORADO	CRUCERO		120
IV416	VER.	GITANO	CHALAM		200
IV416	VER.	GITANO	CHALAM		570
392SL	S.C.	NEGRO	REMOLQUE		310
392SL	S.C.	NEGRO	REMOLQUE		820

FIG. III.16 Si una relación 1:1 es Opcional en ambas direcciones mover la FK en la tabla en cualquiera de las dos tablas de la relación

g) Si una relación 1:1 es opcional en ambas direcciones mover la FK en la tabla en cualquiera de las dos tablas de la relación. Por ejemplo(Fig. III.16). Para la relación opcional 1:1 entre MUELLE y BARCO la columna FK puede colocarse en la tabla MUELLE o BARCO: La columna M_NUM es agregada a la tabla BARCO, y etiquetada como única para forzar la relación 1:1.

h) Para una relación recursiva 1:M, agregar una columna FK a la tabla. Esta columna FK debe referenciar valores de la columna PK. Por ejemplo. (Fig. III.17). Para esta relación recursiva 1:M, agregar una columna FK a la tabla de EMPLEADOS para cada empleado que sea gerente. Nombrar la columna como MGR_ID para reflejar la relación.

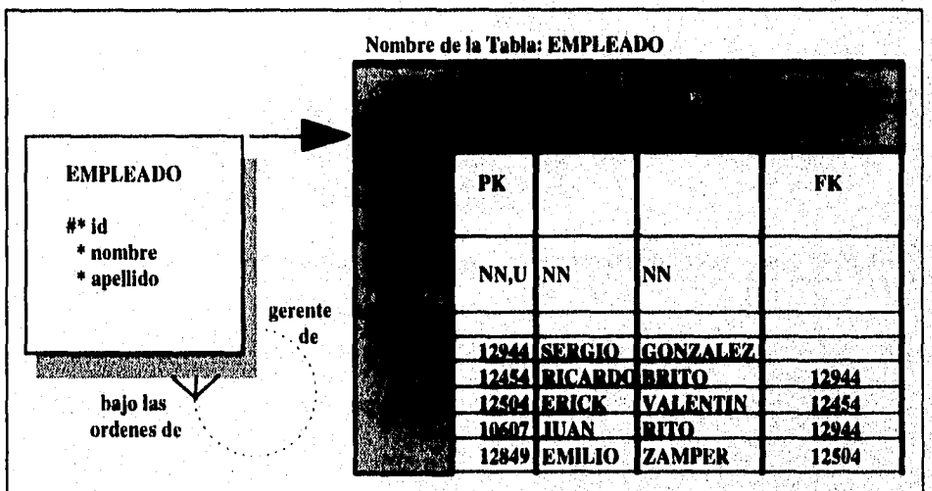


FIG. III.17 Para esta relación recursiva 1:M. Agregar una columna FK a la tabla EMPLEADO

i) Para una relación recursiva 1:1, agregar una FK única a la tabla. Esta columna FK debe referenciar un valor de la columna PK. Por ejemplo (Fig. III.18). Para esta relación recursiva 1:1, agregar una columna única a la tabla PERSONAL.

- La columna FK hace referencia a un renglón en la misma tabla
- El nombre de la columna FK debe reflejar la relación.

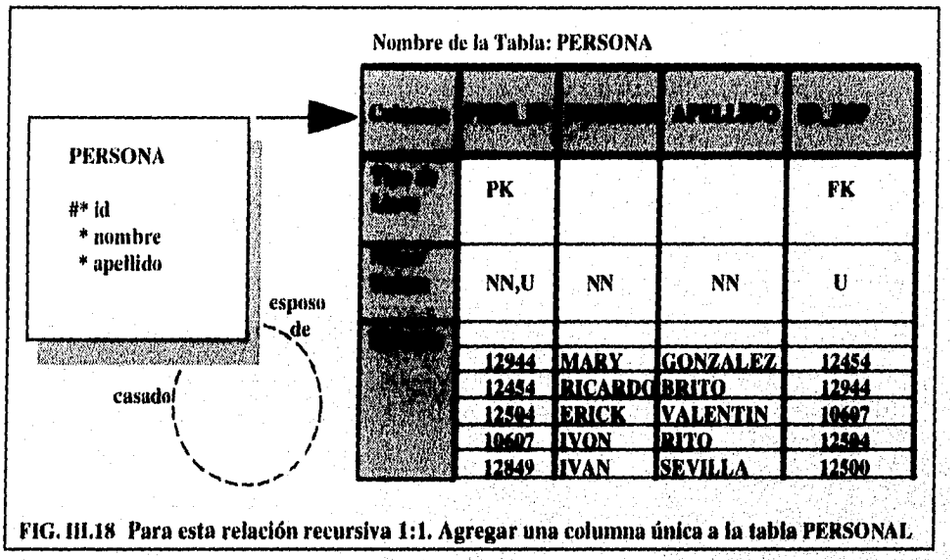


FIG. III.18 Para esta relación recursiva 1:1. Agregar una columna única a la tabla PERSONAL

j) La combinación de columnas PK y FK siempre debe ser única para asegurar la relación 1:1. Poniendo el PK y el FK como único (U) se garantiza que la combinación será única.

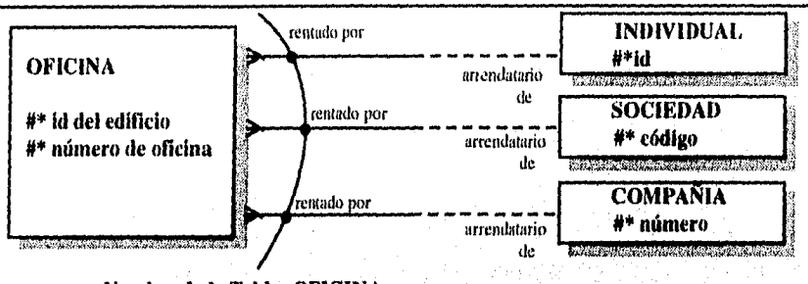
k) Una FK recursiva nunca debe ser NO NULA.

3.7.6 Escoger Opciones de Arco

Los Arcos representan un tipo de llave foránea de alternativa múltiple. Existen dos alternativas de diseño para "mapear" arcos a llaves foráneas que son:

a) Diseño de Arco Explícito

b) Diseño de Arco Genérico



Nombre de la Tabla: OFICINA

PK	PK	FK1	FK2	FK3
NN,U1	NN,U1			
1020	10	30044		
1030	80		XL450	
1070	35	54600		
1060	65			2040
1050	12			3460

FIG. III.19 El diseño de Arco Explícito crea una columna de llave foránea para cada relación que incluye el arco.

Por ejemplo (Fig. III.19).Este Modelo E-R mapeará cuatro tablas. La entidad OFICINA tiene un arco que atraviesa los finales de tres relaciones, y las correspondientes columnas de FK deben ser agregadas a la tabla OFICINA.

Usar un Diseño de Arco Explícito o un Diseño de Arco Genérico para agregar estas múltiples alternativas de llaves foráneas.

También se utiliza el Diseño de Arco Explícito o el Diseño de Arco Genérico para implementar múltiples llaves foráneas cuando un arco atraviesa un conjunto de relaciones 1:1. Los arcos solamente pueden atravesar el final de las relaciones que son o todas obligatorias o todas opcionales.

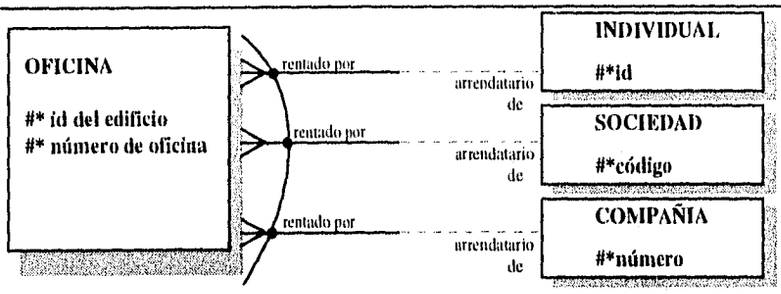
El diseño de Arco Explícito crea una columna de llave foránea para cada relación que incluye el arco.

Por ejemplo. (Fig. III.19). El Modelo E-R contiene cuatro entidades simples, y serán mapeadas para cuatro tablas por separado. El arco atraviesa el final de las tres relaciones. Por lo tanto, las FKS deben de ser agregadas a la tabla OFICINA. Se utiliza un Diseño de Arco Explícito para crear una columna FK para cada relación.

El diseño de Arco Explícito soportará llaves foráneas múltiples con diferentes formatos. Por ejemplo. ID_IN, COD_SOC, y NU_CIA pueden tener diferente formato de columna.

El software de aplicación debe forzar a una relación de exclusividad entre las llaves foráneas.

El diseño de Arco Genérico crea una columna de llave foránea única y una columna para marcar la relación del arco. Como las relaciones son exclusivas, solamente un valor de FK existirá para cada renglón en la tabla.



Nombre de la Tabla: OFICINA

	PK	PK	FK1	
	NN,U1	NN,U1	NN	NN
	1020	10	30044	I
	1030	80	XL450	S
	1070	35	54600	I
	1060	65	2040	C
	1050	12	3460	C

FIG. III.20 El diseño de Arco Genérico crea una columna de llave foránea unica y una columna para marcar la relación del arco. Como las relaciones son exclusivas, solamente un valor de FK existirá para cada renglón en la tabla.

Por ejemplo (Fig. III.20). Otra vez, cuatro tablas por separado para este Modelo E-R (Uno para cada entidad). Como el arco atraviesa el final de las tres relaciones, las FKS. Se deben de agregar a la tabla OFICINA. Utilizar un Diseño de Arco Genérico, y crear una columna de llave foránea simple, y

agregar un tipo de columna para indicar cual de las tres tablas está referenciada por la columna FK en cada renglón. Por ejemplo, "I" para individual, "S" para SOCIEDAD, y "C" para COMPAÑIA.

Si las relaciones debajo del arco son obligatorias, hacer ambas columnas NO NULAS (NOT NULL).

Las llaves foráneas deben de compartir el mismo formato para todas las tablas referenciadas.

La relación de exclusividad se forza automáticamente.

3.7.7 Escoger Opciones de Subtipos

Escoger cualquiera de las tres opciones para el mapeo de subtipos con tablas.

Opciones de Mapeo de Subtipos a Tablas

a) Diseño de una sola Tabla

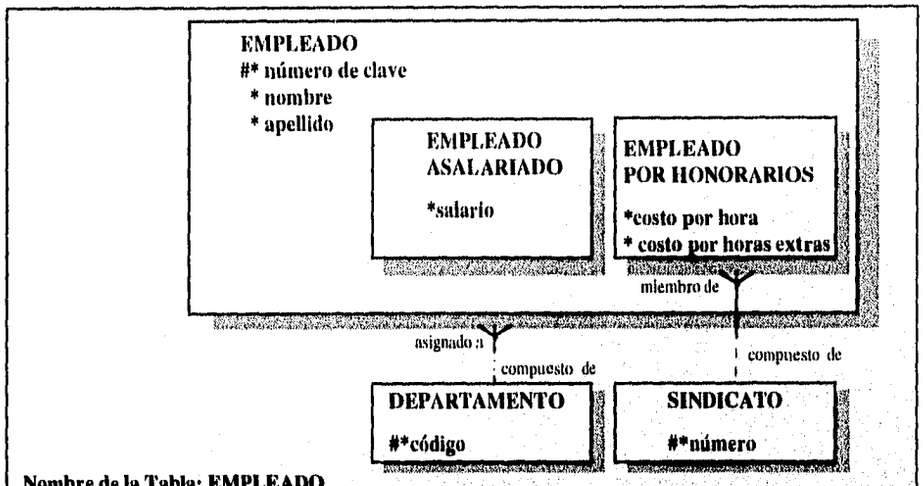
b) Diseño de Tablas Separadas

c) Implementación del Arco

Por ejemplo (Fig. III.21). En el siguiente supertipo/subtipo, las entidades EMPLEADO, EMPLEADO ASALARIADO y EMPLEADO POR HONORARIOS pueden ser mapeadas en una, dos, o tres tablas, dependiendo de la opción seleccionada.

Resumen de Conceptos de Subtipos / Supertipos

- ❑ Los subtipos heredan todos los atributos del supertipo y relaciones
- ❑ Los subtipos pueden tener atributos y relaciones propias
- ❑ Los subtipos deben de ser mutuamente excluyentes.



Nombre de la Tabla: EMPLEADO

PK								
NN,U	NN	NN	NN					NN
6631	MARY	LOPEZ	E	2300				40
1190	JULIA	VEGA	E	2900				30
370	LUIS	SOTO	NE		8.50	12.75	201	40
800	JUAN	RITO	NE		6.75	11.50	150	45
7147	ERICK	VALENTIN	NE		12.0	18.00	201	30
6794	EMILIO	ZAMPER	NE		9.50	16.15	201	45

FIG. III.21 Diseño de Subtipo de una sola tabla

3.7.7.1 Opción 1.- Diseño de Subtipo de una Sola Tabla

Mapear los subtipos dentro de una sola tabla para el supertipo. La tabla sencilla contendrá instancias de todos los subtipos (Fig. III.22).

Se usa el diseño de una sola tabla cuando el subtipo tiene pocas relaciones y atributos propios.

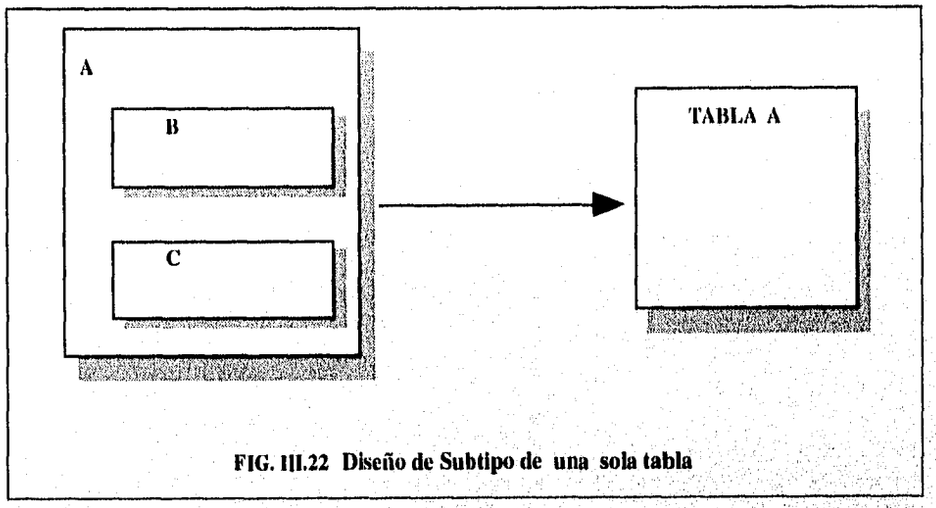


FIG. III.22 Diseño de Subtipo de una sola tabla

Pasos de Diseño

1) Crear una sola tabla para el subtipo.

2) Crear una columna TIPO para identificar a que subtipo pertenece cada renglón.

3) Crear una columna para cada uno de los atributos del supertipo.

4) Crear una columna para cada uno de los atributos del subtipo

5) Crear columnas FK para cada una de las relaciones del supertipo.

6) Crear columnas FK para cada una de las relaciones del subtipo.

Por ejemplo (Fig. III.21). Mapear el supertipo EMPLEADO y sus subtipos dentro de una sola tabla EMPLEADO.

Las columnas de la tabla EMPLEADO son derivadas de los atributos y las relaciones del supertipo y de todos sus subtipos.

El diseño de subtipo de una sola tabla requiere una nueva columna tipo para identificar el subtipo al que pertenece cada renglón. La columna TIP_EMP fué agregada a la tabla EMPLEADO para este propósito.

Usar un Diseño de Subtipo de una sola Tabla cuando hay pocas relaciones y atributos propios (Fig. III.23).

Tipo de Entidad	Columna para Atributos	Columna FK para Relaciones
Supertipo	NUM_CVE NOMBRE APELLIDO	COD_DEPT
Subtipo	EMPEX_SALA EMPNOE EX_CTO_HORA EMPNO_EX_HR_EXTRA	NU_SIN_NE

FIG. III.23 Diseño de Subtipo de una sola tabla.

Ventajas del diseño

- El acceso al supertipo es directo.
- El subtipo puede ser accesado y modificado usando vistas.

Desventajas del Diseño

- Los requerimientos del subtipo NO NULO no se pueden forzar a nivel de base de datos.
- La lógica de las aplicaciones tendrán que manejar diferentes conjuntos de atributos, dependiendo del TIPO.

3.7.7.2 Opción 2.- Diseño de Subtipo de Tablas Separadas

Mapear el subtipo en tablas separadas (Uno para cada subtipo). Cada tabla contendrá solamente instancias de un subtipo (Fig. III.24).

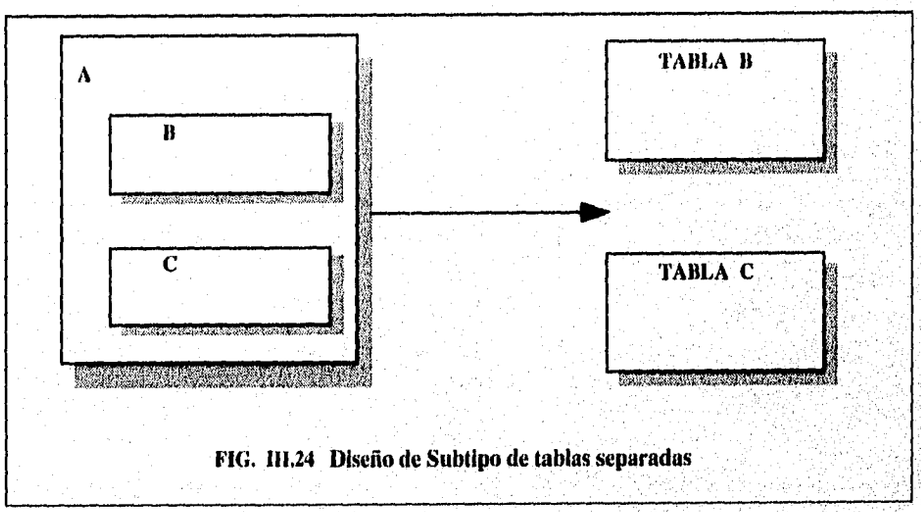


FIG. III.24 Diseño de Subtipo de tablas separadas

Pasos del Diseño

1) Crear una tabla para cada subtipo.

2) En cada tabla subtipo, crear columnas para los atributos del subtipo.

3) En cada tabla subtipo, crear columnas para los atributos del supertipo.

4) En cada tabla subtipo, crear columnas FK para las relaciones del subtipo.

5) En cada tabla subtipo, crear columnas FK para las relaciones del supertipo.

Se usa un Diseño de Subtipo de Tablas Separadas cuando hay muchas relaciones y atributos de subtipos específicos.

Ventajas del Diseño

- La opcionalidad de los atributos del subtipo se forza a nivel de la base de datos
- La lógica de las aplicaciones no requiere de chequeo para los subtipos.

Desventajas del Diseño.

- El acceso al supertipo requiere de un operador de UNION o una vista con un operador de UNION.

- Las vistas que enlazan (JOIN) dos tablas son solamente desplegables.
- El código del programa aplicación debe de ser específico para cada tabla individual.
- El mantenimiento de los UID'S de los subtipos es difícil de implementar.

Nombre de la Tabla: EMPLEADO ASALARIADO

Columnas	NUM_CVE	NOMBRE	APELLIDO	SAL_TE	COD_DEPT
Tipo de llave	PK				FK2
Nulos/ Unico	NN,U	NN	NN	NN	NN
Ejemplo	6631	MARY	LOPEZ	2300	40
	1190	JULIA	VEGA	2900	30
	370	LUIS	SOTO		40
	800	JUAN	RITO		45
	7147	ERICK	VALENTIN		30
	6794	EMILIO	ZAMPER		45

Nombre de la Tabla: EMPLEADO POR HONORARIOS

Columnas	NUM_CVE	NOMBRE	APELLIDO	TIPO_EMP	TARIFA_HR	HRS_TRAB	COD_DEPT
Tipo de llave	PK					FK1	FK2
Nulos/ Unico	NN,U	NN	NN	NN	NN	NN	NN
Ejemplo	6631	MARY	LOPEZ	E			40
	1190	JULIA	VEGA	E			30
	370	LUIS	SOTO	NE	8.50	12.75	40
	800	JUAN	RITO	NE	6.75	11.50	45
	7147	ERICK	VALENTIN	NE	12.0	18.00	30
	6794	EMILIO	ZAMPER	NE	9.50	16.15	45

FIG. III.25 Diseño de Subtipo de tablas separadas

Por ejemplo (Fig. III.25). Hacer el supertipo EMPLEADO en dos tablas (Una para cada subtipo). Primero crear una tabla separada para el subtipo EMPLEADO ASALARIADO. Crear una tabla separada para el subtipo EMPLEADO POR HONORARIOS.

3.7.8 Resúmen del Diseño Inicial de la Base de Datos

Para hacer un mapa del Modelo Entidad-Relación para iniciar un diseño de base de datos se utilizan los siguientes pasos:

1) Mapear las entidades a tablas

2) Mapear los atributos a las columnas y documentar los datos de ejemplo.

3) Mapear UID'S a llaves primarias.

4) Mapear relaciones a llaves foráneas.

5) Escoger opciones de Arco.

6) Escoger opciones de Subtipo.

Documentar el diseño inicial de base de datos en un mapa de Instancias.

3.8 CATEGORIAS DE NORMALIZACIÓN

Categorizar las tablas de acuerdo a su grado de normalización (Fig. III.26).

"Cada valor de una llave no primaria debe ser dependiente solamente de la llave completa y no de ningún otro campo."

¿Por qué hay que normalizar las tablas?

La normalización minimiza la redundancia de los datos. Un dato sin normalizar es redundante. Y la redundancia de datos causa problemas de integridad. Las transacciones de actualización y borrado puede no ser consistentes en todas las copias de los datos causando inconsistencia en la base de datos. Por lo tanto la Normalización ayuda a identificar Entidades, Relaciones y Tablas faltantes. La tercera forma normal es un objetivo normalmente aceptado para un diseño de base de datos para eliminar la redundancia. Y las formas normales mayores no son ampliamente utilizadas.

Regla de la Forma Normal	Descripción
Primera Forma Normal (1FN)	La tabla debe ser expresada como un conjunto desordenado de tabla de dos dimensiones. La tabla no puede contener grupos de repetición.
Segunda Forma Normal (2FN)	La tabla debe de estar en 1FN. Cada columna que no es llave debe de ser dependiente de la llave primaria como un todo.
Tercera Forma Normal (3FN)	La tabla debe de estar en 2FN. Una columna que no es llave primaria no debe ser funcionalmente dependiente de otra columna no llave.

FIG. III.26 Reglas de Normalización

Reconocer datos sin Normalizar

Un dato sin normalizar no cumple con ninguna regla de normalización.

Por ejemplo (Fig. III.27). Considerar el siguiente conjunto de datos. Tres registros de longitud variable son mostrados a continuación uno para cada ID_ORDEN ¿Cuál es ese dato no normalizado?

Tabla: ORDEN

2301	6/23	101	ERICK	SONORA	3786	RED	3	35.00
					4011	RAQUETA	6	65.00
					9132	PAQ-3	8	4.75
2302	6/25	107	VALENTIN	OAXACA	5794	PAQ-6	4	5.00
2303	6/26	110	EMILIO	COLIMA	4011	RAQUETA	2	65.00
					3141	FUNDAS	2	10.00

FIG. III.27 Un dato sin Normalizar no cumple con ninguna regla de Normalizacion

Esta contiene un grupo de repetición para NUM ITEM, DESCRIP ITEM, CANTIDAD y PRECIO. La primera forma normal prohibe los grupos de repetición.

3.8.1 Conversión a la Primera Forma Normal

1) Remover los grupos de repetición de la base de datos

2)-Crear una nueva tabla con la PK de la tabla base y el grupo de repetición

Por ejemplo (Fig. III.28). Convertir el siguiente conjunto de datos sin normalizar a la Primera Forma Normal.Remover el grupo de repetición de NUM ITEM, DESCRIP ITEM, CANTIDAD y PRECIO.

El PK de la tabla que queda es ID ORDEN. Crear una nueva tabla ITEM_ORDEN con ID_ORDEN y el grupo de repetición.

Tabla: ORDEN

ID ORDEN	FECHA	ID CLIENTE	NOMBRE CLIENTE	ESTADO
PK				
2301	6/23	101	ERICK	SONORA
2302	6/25	107	VALENTIN	OAXACA
2303	6/26	110	EMILIO	COLIMA

Tabla: ITEM ORDEN

PK,FK	PK			
2301	3786	RED	3	35.00
2301	4011	RAQUETA	6	65.00
2301	9132	PAQ-3	8	4.75
2302	5794	PAQ-6	4	5.00
2303	4011	RAQUETA	2	65.00
2303	3141	FUNDA	2	10.00

FIG. III.28 Conversión a la Primera Forma Normal

3.8.2 Conversión a la Segunda Forma Normal

1) Determinar cuales columnas que no son llave no dependen de la llave primaria completa de la tabla.. Y remover esas columnas de la tabla base.

- ❑ Si cada columna no depende de la llave primaria completa, la tabla no está en 2FN.
- ❑ Cualquier tabla con una llave primaria de una sola columna está automáticamente en la 2FN.

2) Crear una segunda tabla con esas columnas y la (s) columna(s) de la PK de la cual dependen.

Por ejemplo (Fig. III.29) Poner la siguiente tabla en 2FN.

Tabla: ITEM ORDEN

PK,FK	PK			
2301	3786	RED	3	35.00
2301	4011	RAQUETA	6	65.00
2301	9132	PAQ-3	8	4.75
2302	5794	PAQ-6	4	5.00
2303	4011	RAQUETA	2	65.00
2303	3141	FUNDA	2	10.00

Tabla: ITEM_ORDEN

PK,FK	PK,FK	
2301	3786	3
2301	4011	6
2301	9132	8
2302	5794	4
2303	4011	2
2303	3141	2

Tabla: ITEM

PK		
3786	RED	35.00
4011	RAQUETA	65.00
9132	PAQ-3	4.75
5794	PAQ-6	5.00
3141	FUNDA	10.00

FIG. III.29 Conversión a la Segunda Forma Normal

La tabla ITEM_ORDEN no está en la 2FN ya que PRECIO y DESCRIPCION son dependientes de NUM_ITEM, pero no son dependientes de ID_ORDEN. Para convertir la tabla a la 2FN, remover parcialmente cualquier columna dependiente. Crear una tabla ITEM con estas columnas y la columna de la PK de la que dependen.

3.8.3 Conversión a la Tercera Forma Normal.

1) Determinar que columnas son dependientes de otra columna no llave.

- ❑ Una tabla está en Tercera Forma Normal si una columna no llave no es funcionalmente dependiente de otra columna no llave.
- ❑ Una columna no llave no puede ser funcionalmente dependiente de otra columna no llave.

2) Remover esas columnas de la tabla base.

3) Crear una segunda tabla con esas columnas y con la columna no llave de la cual son dependientes.

Por ejemplo. (Fig. III.30) Poner la tabla ORDEN en Tercera Forma Normal.

Tabla: ORDEN

ID ORDEN	FECHA	ID CLIENTE	NOMBRE CLIENTE	ESTADO
PK				
2301	6/23	101	ERICK	SONORA
2302	6/25	107	VALENTIN	OAXACA
2303	6/26	110	EMILIO	COLIMA

Tabla: ORDEN

ID ORDEN	FECHA	ID CLIENTE
PK		PK
2301	6/23	101
2302	6/25	107
2303	6/26	110

Tabla: CLIENTE

ID CLIENTE	NOMBRE CLIENTE	ESTADO
PK		
101	ERICK	SONORA
107	VALENTIN	OAXACA
110	EMILIO	COLIMA

FIG. III.30 Conversión a la Tercera Forma Normal

NOMBRE CLIENTE y ESTADO son dependientes de ID CLIENTE. ID CLIENTE no es la PK. Por eso, la tabla ORDEN no está en 3FN.

Mover las columnas no llave dependientes con la columna no llave de la cual son dependientes dentro de un nueva tabla CLIENTE.

Por ejemplo (Fig. III.31). Considerar la tabla ITEM_ORDEN. ¿Está en 3FN?

Tabla: ITEM_ORDEN

PK,FK	PK,FK	
2301	3765	3
2301	4011	6
2301	9132	8
2302	5794	4
2303	4011	2
2303	3141	2

FIG. III.31 Tabla ITEM_ ORDEN en 3FN

Todos los atributos no llave son dependientes solamente de la llave. La tabla ITEM_ORDEN está en 3FN.

3.9 LA BASE DE DATOS POSTERIOR

Revisar el diseño de la tabla contra el módulo de requerimientos de aplicación y refinar y extender el diseño inicial para producir un diseño completo de la base de datos.

Actividades:

a) Definir los constraints de integridad referencial

b) Diseñar índices

c) Establecer vistas

d) Desnormalizar el diseño de la base de datos

e) Planear el uso del almacenamiento físico.

3.9.1 Especificar la Integridad Referencial

El valor de una columna de llave foránea debe coincidir con un valor existente en otra columna que sea llave primaria (o ser NULO). Usar los constraint de la integridad referencial para especificar como se debe de mantener la integridad referencial.

Constraint de borrado

¿ Qué sucede si un renglón que contiene una llave primaria referenciada es borrado?

Especificar el Constrait de borrado para definir qué debe suceder si un renglón que contiene a una llave primaria referenciada es borrado.

Opciones: CASCADE, RESTRICTED, o NULLIFY (sólo si se permiten NULOS).

Por ejemplo (Fig. III.32). Considerar las tablas EMPLEADO y DEPARTAMENTO. ¿Qué deberá si el NO_DEPT para el cual el empleado trabaja es borrado de la tabla DEPARTAMENTO?

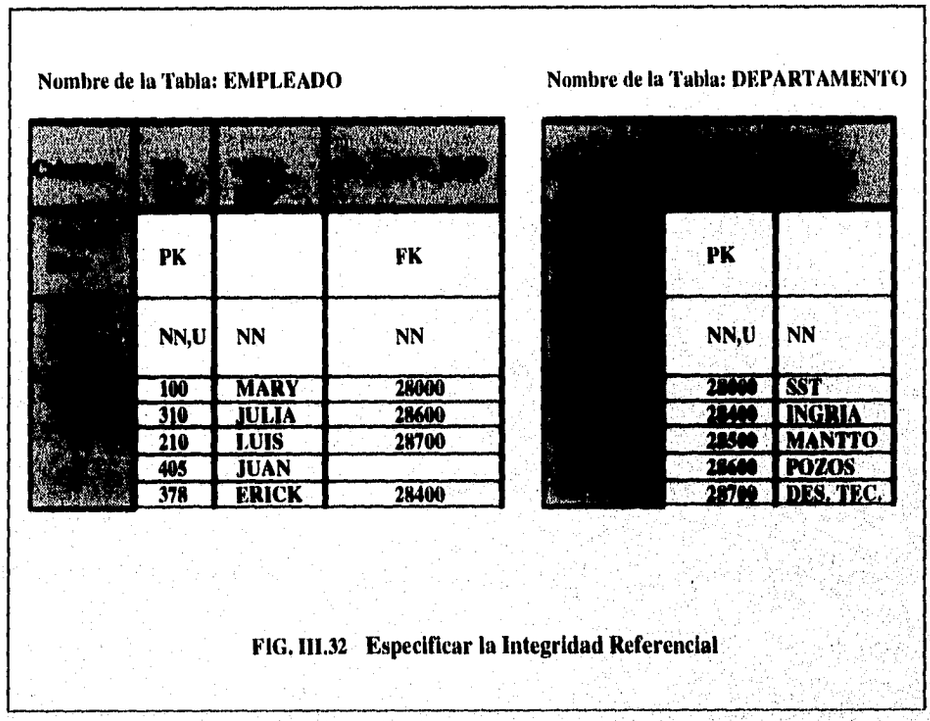


FIG. III.32 Especificar la Integridad Referencial

Opciòn	Explicación del Constraint
CASCADE	El borrado deberá ser en cascada para todos los empleados que coincidan. Los renglones que coinciden de EMPLEADO también deberán ser borrados.
RESTRICTED	El borrado deberá sólo a DEPARTAMENTOS sin empleados
NULLIFY	Se deberá colocar un valor NULO a la llave foránea (Valido sólo para PK's permitiendo NULOS) cuando la PK referenciada es borrada.

FIG. III.33 Tabla de Constraint

Constraint de Actualización

¿Qué sucede si una llave primaria referenciada es actualizada?. Sólo hay consecuencia si la PK es actualizable en primer lugar.

Especificar un Constraint de Actualización para definir qué debería suceder cuando una llave primaria es actualizada. (La regla de ACTUALIZACION) sólo tiene sentido si la PK es actualizable).

Opciones: CASCADE,RESTRICTED o NULLIFY) (sólo si se permiten los NULOS)

Por ejemplo (Fig. III.32). ¿Que debería de pasar cuando el NO_DEPT para el que trabajan actualmente algunos empleados es cambiado?. Ver (Fig. III.34).

Opciòn	Explicación del Constraint
CASCADE	La actualización debería hacerse en cascada con los empleados que coincidan: Los renglones que coincidan de EMPLEADO también deberán ser actualizados para reflejar el nuevo valor de PK.
RESTRICTED	La actualización deberá ser restringida a DEPARTAMENTOS sin empleados.
NULLIFY	Se deberá poner un valor de NULO en la llave foránea (válida sólo para FK's que permiten NULOS) cuando la PK referenciada es actualizada a un nuevo valor de PK..

FIG. III.34 Ejemplo de Constraint

3.9.2 Diseño de Indices

Un índice es asociado con una sóla tabla física y contiene los valores de una o más columnas de esa tabla. Usar índices para mejorar considerablemente el acceso de datos. Un Indice nos ayuda a:

a) Proveen acceso rápido a los renglones de datos y evitan búsquedas en toda la tabla.

b) Facilitan uniones (join) entre tablas.

c) Aseguran que no existe un valor duplicado si se define como único.

d) Son usados automáticamente cuando se referencian en las cláusulas WHERE de las instrucciones de SQL si la columna no es modificada.

Un índice concatenado es un índice creado en un grupo de columnas en una sola tabla. Mapear una llave compuesta a un índice concatenado. (Fig. III.35)

Nombre de la Tabla: CURSO

Columna	COD_CURSO	NOMBRE	DUR	CUOTA	ID_INST
Tipo de llave	PK				FK
Nulos/ Unicos	NN,U	NN	NN	NN	NN
Ejemplo	344	SQL FORMS	5	1000	81
	974	SQL RW	2	400	73
	401	DB	2	400	95
		DESIGN			
	717	DBA	3	900	73
	659	SQL PLUS	2	400	301

Tabla : CURSO

1010	344	SQL FORMS	5	1000	81
1011	974	SQL RW	2	400	73
1012	401	DB DESIGN	2	400	95
1013	717	DBA	3	900	73
1014	659	SQL PLUS	2	400	301

Indice I_CURSOS_PRIN
(Unico)

344	1010
401	1012
659	1014
717	1013
974	1011

Indice I_CURSOS_2
(No Unico)

73	1011
73	1013
81	1010
95	1012
301	1014

FIG. III.35 Diseño de Indices

Por ejemplo (Fig. III.36). La tabla MATRICULA tiene una PK compuesta de COD_CURSO y EST_ID. Crear una llave compuesta llamada I_MATRICULA_PRIN en ambas columnas.

Nombre de la Tabla: MATRICULA

Columna	FECHA_INIC	FECHA_TERMI	GRADO	COD_CURSO	EST_ID
Tipo de Dato			PK,FK1	PK,FK2	
Nulos/Unicos	NN			NN,U1	NN,U2
Ejemplo	20-JUL-95	19-JUL-95	-	344	47592
	05-SEP-95		-	401	15402
	14-JUN-95	28-JUL-95	A	717	51394
	08-MAY-95	28-JUL-95	B	717	94572
	05-MAY-95	21-MAY-95	A	401	51394

Tabla : MATRICULA

1010	20-JUL-95	19-JUL-95	-	344	47592
1011	05-SEP-95		-	401	15402
1012	14-JUN-95	28-JUL-95	A	717	51394
1013	08-MAY-95	28-JUL-95	B	717	94572
1014	05-MAY-95	21-MAY-95	A	401	51394

Indice I_MATRICULA_PRIN (Unica)

344	47592	5011
401	15402	5012
401	51394	5014
717	51394	5015
717	94572	5013

FIG. III.36 Diseño de Indices Concatenados (Diseño de las tablas)

Usar índices para implementar llaves y soportar los requerimientos de acceso de la aplicación.

Crear índices para:

- ❑ Llaves Primarias (Indices únicos)
- ❑ Llaves foráneas (Generalmente índices no-únicos)

Considerar la indexación para

- ❑ Llaves Alternativas (índices únicos)
- ❑ En columnas no-llaves críticas usadas en las cláusulas WHERE.
- ❑ Cualquier llave de búsqueda.

Los índices requieren espacio y generan una sobrecarga al momento de hacer actualizaciones.

- ❑ Un índice único hace referencia a una columna o grupo de columnas que tienen valores únicos en la tabla.
- ❑ Indices no-únicos hacen referencia a columnas o grupos de columnas que no son únicas en la tabla.
- ❑ Tener en cuenta que bajo ciertas condiciones, los índices no son usados por el RDBMS.

3.9.3 Establecer Vistas

Establecer las vistas de la base de datos para cubrir los requerimientos de acceso.

Las vistas se pueden utilizar para:

a) Restringir accesos.

b) Proveer integridad referencial

c) Presentar las tablas a los usuarios en cualquier forma

d) Pre-empacar consultas complejas

e) Producir prototipos rápidos

f) Hacer pre-uniones de tablas base en SQL*Forms

g) Checar la entrada de datos.

Una vista se puede ver como una ventana predefinida en una base de datos.

- Una vista no tiene datos propios y únicamente cuenta con información de tablas fundamentales.
- Una vista es definida por una instrucción SELECT a la cual se pone un nombre y es almacenado en un diccionario de datos ORACLE.

❑ Una vista es consultada como si fuera una tabla.

Una Vista puede restringir lo que ve el usuario, el diseñador o la herramienta.

Por ejemplo (Fig. III.37). Una vista de la tabla EMP puede ser usada para restringir a los usuarios de ver los salarios de los empleados.

Tabla : BASE

EMP_NO	NOMBRE	NUM_EMP	SAL
101	EMILIO	50	55,010
50	ERICK		250,00
			0
210	CLAUDIA	50	45,500
443	ADRIANA	101	25,250
501	ZENON	210	35,250

Tabla : VISTA

101	EMILIO	50
50	ERICK	
210	CLAUDIA	50
443	ADRIANA	101
501	ZENON	210

FIG. III.37 Establecer Vistas

Una vista puede ser usada para presentar datos normalizados en forma desnormalizada.

Una vista definida entre las dos tablas puede ser usada para hacer una pre-unión (pre´join) entre las tablas, asi el usuario sólo podrá ver una sola tabla.

Usar las vistas con precaución. El acceso a través de las vistas es más lento porque se requiere un acceso al diccionario de datos y puede causar que la optimización de la consulta sea más lenta.

Por ejemplo (Fig. III.38). Siguiendo las reglas de normalización, las tablas ORDEN y CLIENTE son separadas.

Tabla : ORDEN

NUM_ORD	FECHA_ORD	ORD_NO_CLIEN
10145	10-JUN-95	111
32157	08-SEP-95	342
43010	30-JUL-95	111
27444	07-AGO-95	550
98710	10-MAY-95	375

Tabla : CLIENTE

NO_CLIEN	NOMBRE_CLIEN	APELLIDO_CLIEN
111	EMILIO	ZAMPER
342	ERICK	VALENTIN
550	CLAUDIA	NAVA
375	ADRIANA	GONZALEZ

Tabla : VISTA

10145	10-JUN-95	111	EMILIO	ZAMPER
32157	08-SEP-95	342	ERICK	VALENTIN
43010	30-JUL-95	111	CLAUDIA	NAVA
27444	07-AGO-95	550	ADRIANA	GONZALEZ
98710	10-MAY-95	375		

FIG. III. 38 Establecer una vista entre dos tablas

Limitaciones de las Vistas

a) Para una vista basada en una sola tabla, los comandos INSERT, UPDATE y DELETE de SQL no tienen limitaciones.

b) Para vistas de múltiples tablas con columnas virtuales, INSERT,UPDATE y DELETE están restringidos.

c) Cuando se accesa la tabla a través de Vistas, es posible agregar renglones no visibles a través de la Vista a menos que la opción WITH CHECK OPTION sea especificada.

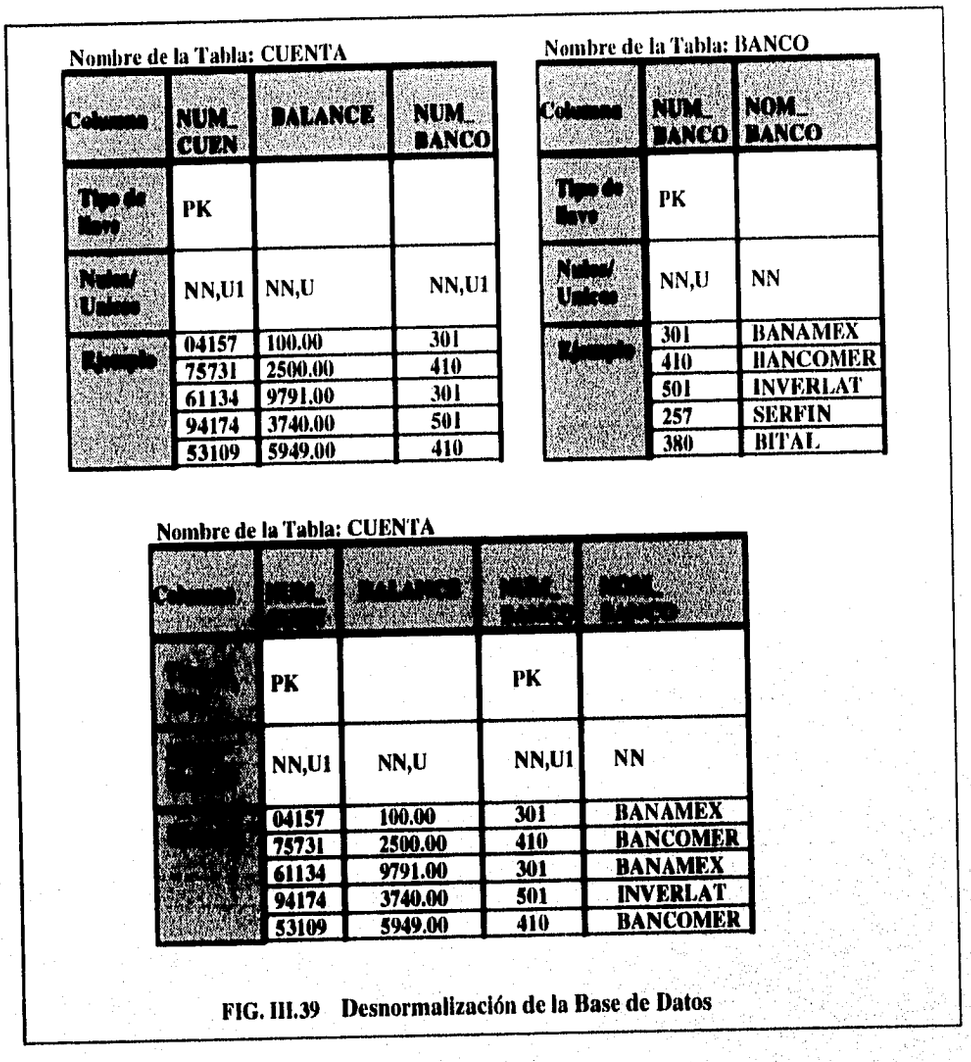
3.9.4 Diseño de Desnormalización de la Base de Datos

Siempre Hay que comenzar con tablas en Tercera Formas Normal. Y hay que ser cuidadosos con la desnormalización:

- ❑ Ser extremadamente renuente a desnormalizar el diseño de default de la tabla
- ❑ La desnormalización puede causar problemas de inconsistencia de datos.
- ❑ La desnormalización puede ser la solución para transacciones con requerimientos de performance como:
  - ❑ High Throughput.
  - ❑ Alta Frecuencia.
  - ❑ Tiempo de respuesta rápido
- ❑ Considerar todas las otras opciones antes de la desnormalización, especialmente el agregar o cambiar la estructura de los índices.

Combinar las tablas es la forma más común de desnormalizar.

Por ejemplo (Fig,III.39). Considerar las tablas CUENTA y BANCO.



Nombre de la Tabla: CUENTA

Columnas	NUM_CUEN	BALANCE	NUM_BANCO
Tipo de llave	PK		
Nulos/ Unicos	NN,U1	NN,U	NN,U1
Ejemplo	04157	100.00	301
	75731	2500.00	410
	61134	9791.00	301
	94174	3740.00	501
	53109	5949.00	410

Nombre de la Tabla: BANCO

Columnas	NUM_BANCO	NOM_BANCO
Tipo de llave	PK	
Nulos/ Unicos	NN,U	NN
Ejemplo	301	BANAMEX
	410	BANCOMER
	501	INVERLAT
	257	SERFIN
	380	BITAL

Nombre de la Tabla: CUENTA

Columnas	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	PK		PK	
[illegible]	NN,U1	NN,U	NN,U1	NN
[illegible]	04157	100.00	301	BANAMEX
	75731	2500.00	410	BANCOMER
	61134	9791.00	301	BANAMEX
	94174	3740.00	501	INVERLAT
	53109	5949.00	410	BANCOMER

FIG. III.39 Desnormalización de la Base de Datos

Si las consultas de las cuentas de gran volúmen siempre accesan el nombre del banco, una tabla combinada puede generar redundancia de datos. La tabla CUENTA y BANCO se combinan en NUM_BANCO.

Las tablas de código individuales pueden ser combinadas en una tabla de referencias para validar y decodificar los valores de los códigos para un sistema de aplicación.

Por ejemplo (Fig. III.40). Las siguientes tablas de código se requieren para un sistema de aplicación . Son usados para proveer la lista de valores de SQL*Forms y para validar los valores de las tablas para INSERT o UPDATE.

Combinar todas las tablas en una sola tabla con una columna adicional, TIPO_COD, que define a que grupo de valores pertenece el código. Crear una vista con cada TIPO_COD.

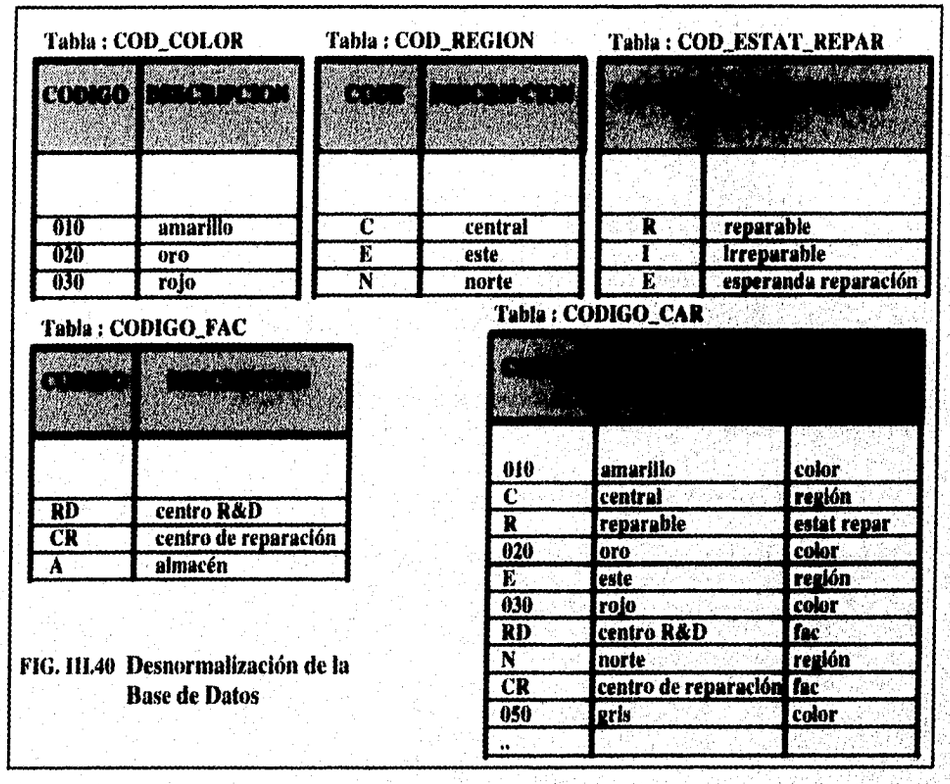


Tabla : COD_COLOR

CODIGO	DESCRIPCION
010	amarillo
020	oro
030	rojo

Tabla : COD_REGION

CODE	DESCRIPTION
C	central
E	este
N	norte

Tabla : COD_ESTAT_REPAR

R	reparable
I	Irreparable
E	esperanda reparación

Tabla : CODIGO_FAC

RD	centro R&D
CR	centro de reparación
A	almacén

Tabla : CODIGO_CAR

010	amarillo	color
C	central	región
R	reparable	estat repar
020	oro	color
E	este	región
030	rojo	color
RD	centro R&D	fac
N	norte	región
CR	centro de reparación	fac
050	gris	color
..		

FIG. III.40 Desnormalización de la Base de Datos

Establecer otra tabla TIPO_COD para validar la longitud de los códigos de descripción.

Por ejemplo. (Fig. III.40). La tabla CODIGO_CAR en las páginas previa incluyen cuatro diferentes tipos de códigos. Cada uno de esos tipos de códigos tienen diferentes longitudes válidas para su descripción de código. Poner una tabla TIPO_COD para validar la longitud de las descripciones (Fig. III.41).

Tabla: CODIGO_CAR

región	20
estat repar	20
colo	10
fac	30

FIG. III.41 Desnormalización de la Base de Datos.

La tabla contiene dos columnas, TIPO_COD y LIMIT, que es la longitud máxima de la descripción para cada TIPO_COD.

Un vector es un arreglo de una dimensión con un número compuesto de valores (un grupo repetido de un tamaño definido). Representar un vector de datos como un conjunto de renglones o un conjunto de columnas (Fig. III.42).

Escoger el diseño de tabla para un vector de datos basados sobre los requerimientos de acceso.

Ventajas de incluir el vector como columnas.

a) Las funciones de grupo de SQL actúan en las columnas, por ejemplo SUM, AVG

b) Los cambios en la longitud del vector pueden ser fácilmente acomodados

Ventajas de incluir el vector como renglón.

a) En la forma de entrada, todos los valores de los datos pueden aparecer en una sola línea.

b) Todos los valores pueden ser insertados con una sola instrucción INSERT.

c) El requerimiento de espacio de almacenamiento es menor.

d) Los reportes de salida con valores horizontales son más fáciles de producir.

Tabla: CODIGO_CAR

MODEL	MES	CANTIDAD
VW SEDAN	ENE	1000
VW SEDAN	FEB	1106
VW SEDAN	MAR	1020
VW SEDAN	ABR	1026
VW SEDAN	MAY	999
VW SEDAN	JUN	1435
VW SEDAN	JUL	1278
VW SEDAN	AGO	1698
VW SEDAN	SEP	920
VW SEDAN	OCT	978
VW SEDAN	NOV	989
VW SEDAN	DIC	886

Tabla: CODIGO_FAC

MODEL												
VW SEDAN	1000	1106	1020	1026	999	1435	1278	1698	920	978	989	886
EXPLORER GT	850	978	1000	104	699	1336	1267	1765	940	995	766	823
MUSTANG	948	674	888	893	1000	1547	1449	1020	950	673	975	345
DERBY	1205	999	1016	1005	1001	1200	1030	1864	999	1040	897	745

FIG. III.42 Desnormalización de la Base de Datos
***Diseño de Tabla de Vectores como Columna (3FN)**
***Diseño de Tabla de Vectores como Renglón**

Reconsiderar el almacenar datos derivados para apoyar a que sean funcionales los requerimientos del acceso y de acuerdo a las características de las herramientas del desarrollo de software.

Por ejemplo (Fig. III.43). Un vendedor regional tiene 200 vendedores a su cargo. El frecuentemente consulta las cuotas de ventas y las ventas actualizadas por cada región. Las cuotas de ventas se establecen

trimestralmente. El dato de ventas es actualizado semanalmente. Es deseable mantener los datos de las cuotas de ventas por región y las ventas actualizadas.

Tabla : PERSON_VENTAS

COLUMNA	ID	APELLIDO	NOMBRE	CUOTA VENTAS	VENTAS	NUM_ REGION
Tipo de Llave	PK					
[illegible]	NN,U	NN	NN	NN	NN	NN
[illegible]	110	VALENTIN	ERICK	500,000	10,000	50
	250	LOPEZ	JUAN	700,000	350,000	35
	310	MATEO	ABEL	1,200,000	500,000	40
	400	FRANCO	LAURA	650,000	250,000	37
	550	BRITO	RICARDO	800,000	420,000	40

Tabla : REGION

COLUMNA	[illegible]	CUOTA VENTAS	VENTAS
[illegible]	PK		
[illegible]	NN,U		
[illegible]	20	2,,000,000	750,500
	50	1,500,000	600,250
	35	700,000	350,000
	40	2,000,000	920,000
	37	1,300,000	250,000

Datos Consolidados, Derivados

FIG. III.43 Diseño de Desnormalización de la Base de Datos

3.9.5 Planeación de Uso del Almacenamiento Físico

Trabajar con el administrador de base de datos para planear el almacenamiento físico de las tablas y los índices.

Consideraciones.

Estimar el monto de espacio requerido en disco para cada tabla e índice

Decidir la localización del lugar de las tablas e índices en espacios de tablespaces lógico y en discos físicos separados.

Definir los parámetros de alojamiento basándose en los patrones esperados de actualización de datos y crecimiento.

3.10 RESUMEN: DISEÑO DE BASE DE DATOS

El diseño de base de datos es el proceso de "MAPEO" de los requerimientos de información reflejados en un Modelo E-R a una base de datos relacional.

Actividad 1: Diseño Inicial de Base de Datos.

- ❑ MAPEAR las entidades simples en tablas
- ❑ MAPEAR los atributos a las columnas y documentar datos de ejemplo.
- ❑ MAPEAR identificadores únicos a llaves primarias
- ❑ MAPEAR relaciones a llaves foráneas.
- ❑ Escoger opciones de arco.
- ❑ Escoger opciones de subtipos.

Actividad 2: Diseño posterior de la base de datos.

- ❑ Definir los constraints de integridad referencial

- ❑ Diseño de índices
- ❑ Establecer vistas.
- ❑ Desnormalizar el diseño de la Base de Datos
- ❑ Agregar las tablas de soporte del sistema.
- ❑ Planear el uso del almacenamiento físico.

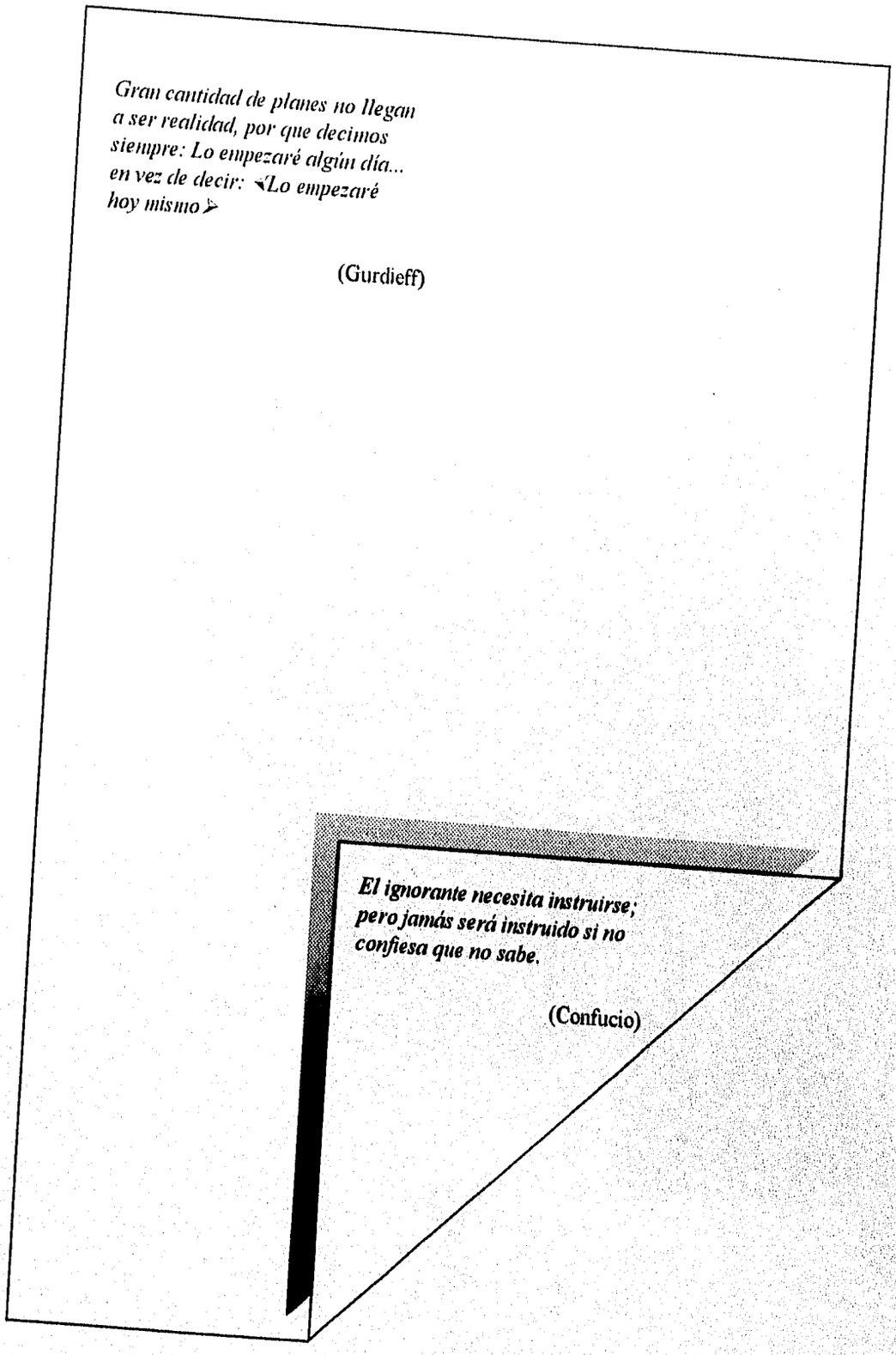
Este Capítulo ha cubierto la segunda parte del proceso de desarrollo de la base de datos top-down. El último paso es la construcción de la base de datos. Que se tocara en el siguiente Capítulo y consiste de los comandos de SQL Y SQL-PLUS para poder construir fisicamente las tablas definidas en este paso. Y asi tener almacenada nuestra Base de Datos fisicamente.

CAPITULO IV

CONSTRUCCION DE LA BASE DE DATOS

Gran cantidad de planes no llegan a ser realidad, por que decimos siempre: Lo empezaré algún día... en vez de decir: «Lo empezaré hoy mismo»

(Gurdieff)



4.1 INTRODUCCIÓN AL SQL

La explosiva popularidad de SQL es una de las tendencias más importantes en la industria informática hoy en día. Durante los últimos años, SQL se ha convertido en el lenguaje de Base de Datos estándar. En la actualidad existen un sin número de productos de gestión de Bases de Datos que soportan ahora SQL, ejecutándose en sistemas informáticos que van desde computadoras personales a mainframes. Se ha adoptado un estándar SQL internacional oficial, y SQL juega un papel clave en el estándar System Application Architecture de IBM. Los artículos en las revistas informáticas celebran a SQL como una nueva *lengua franca* que proporciona compartición de datos sin fisuras entre computadoras en red procedentes de muchos vendedores diferentes. Desde sus oscuros comienzos como un proyecto de investigación en IBM, SQL ha saltado a un lugar prominente como tecnología importante y como fuerza de mercado poderosa.

4.1.1 El lenguaje SQL

SQL es una herramienta para organizar, gestionar y recuperar datos almacenados en una Base de Datos informática. El nombre ‹SQL› es una abreviatura de *Structured Query Languaje (Lenguaje de estructuras de consultas)*. Por razones histórica, SQL se pronuncia generalmente ‹sequel›, pero también se emplea la pronunciación alternativa ‹S.Q.L.›. Como su nombre implica, SQL es un *lenguaje* informático que se puede utilizar para

interaccionar con una Base de Datos. en efecto, SQL trabaja con un tipo específico de Base de Datos, llamada *Base de Datos relacional.*

La (Fig. IV.1) muestra cómo funciona SQL. El sistema informático de la figura tiene una *Base de Datos* que almacena información importante. Si el sistema informático estuviera en una empresa comercial, la Base de Datos podría almacenar datos de inventario, producción, ventas o nómina. En una computadora personal, la Base de Datos podría almacenar datos referentes a los cheques que se han firmado, listas de personas y sus números de teléfono, o datos extraídos de un sistema informático mayor. El programa de software que controla la base datos se denomina *sistema manejador de Base de Datos (database management system)* o DBMS.

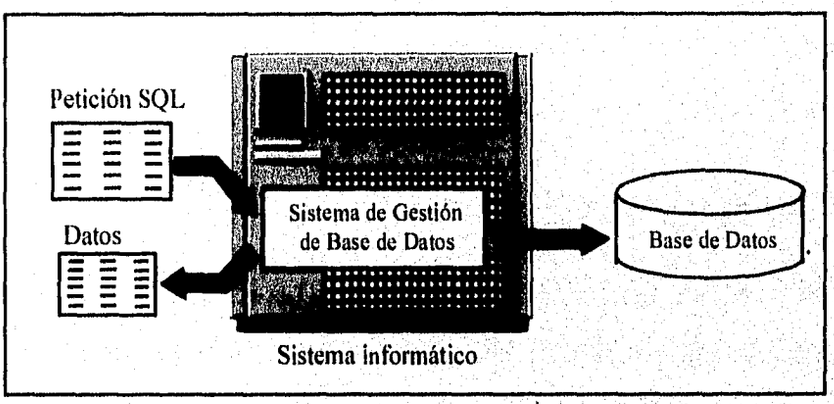


FIG. IV.1 Uso de SQL para Acceso a Base de Datos

Cuando se necesita recuperar datos de la Base de Datos, se utiliza el lenguaje SQL para efectuar la petición. El DBMS procesa la petición SQL, recupera los datos solicitados y los devuelve. Este proceso de solicitar datos

de la Base de Datos y de recibir los resultados se denomina consulta (query) a la Base de Datos; de aquí el nombre Structured Query Language.

El nombre Structured Query Language es realmente y en cierta medida inapropiado. En primer lugar, SQL es mucho más que una herramienta de consulta, aunque ese fue su propósito original y recuperar datos sigue siendo una de sus funciones más importantes. SQL se utiliza para controlar todas las funciones que un DBMS proporciona a sus usuarios, incluyendo:

- *Definición de Datos*. SQL permite a un usuario definir la estructura y organización de los datos almacenados y de las relaciones entre ellos.
- *Recuperación de Datos*. SQL permite a un usuario o a un programa de aplicación recuperar los datos almacenados de la Base de Datos y utilizarlos.
- *Manipulación de datos*. SQL permite a un usuario o a un programa de aplicación actualizar la Base de Datos añadiendo nuevos datos, suprimiendo datos antiguos y modificando datos previamente almacenados.
- *Control de Acceso*. SQL puede ser utilizado para restringir la capacidad de un usuario para recuperar, añadir y modificar datos, protegiendo así los datos almacenados frente a accesos no autorizados.

- *Compartición de Datos.* SQL se utiliza para coordinar la compartición de datos por parte de usuarios concurrentes, asegurando que no interfieren unos con otros.

- *Integridad de datos.* SQL define restricciones de integridad en la Base de Datos, protegiéndola contra corrupciones debida a actualizaciones inconsistentes o a fallos del sistema.

Por tanto SQL es un lenguaje completo de control e interactuación con un sistema manejador de Bases de Datos.

En segundo lugar, SQL no es realmente un lenguaje informático completo tal como C, PASCAL o COBOL. SQL no dispone de la sentencia IF para examinar condiciones, ni de una sentencia GO TO para bifurcaciones, ni de sentencias DO o FOR para iteraciones. En vez de ello, SQL es un *sublenguaje* de Base de Datos, consistente en unas treinta sentencias especializadas para tareas de gestión de Bases de Datos. Estas sentencias SQL se pueden *incorporar* a otro lenguaje, como C, PASCAL o COBOL, para extender ese lenguaje y permitirle utilizar el acceso a la Base de Datos.

Finalmente, SQL no es un lenguaje particularmente estructurado, especialmente cuando se compara con lenguajes altamente estructurados tales como C o PASCAL. En vez de ello, las sentencias SQL se asemejan a frases en inglés, completadas con ‹palabras de relleno› que no añaden nada al significado de la frase pero que hace que se lea más naturalmente. Hay unas cuantas inconsistencias en el lenguaje SQL, y también existen algunas reglas

especiales para impedir la construcción de sentencias SQL que parecen perfectamente legales, pero que no tienen sentido.

A pesar de la imprecisión de su nombre, SQL ha emergido como el lenguaje estándar para la utilización de Bases de Datos relaciónales. SQL es a la vez un potente lenguaje y lenguaje relativamente fácil de aprender.

4.1.2 El Papel de SQL

SQL no es en sí mismo un sistema de gestión de Base de Datos, ni un producto autónomo. Una persona no puede ir a una tienda de informática y ≺comprar SQL≻. En su lugar, SQL es parte integral de un sistema de gestión de Base de Datos, un lenguaje y una herramienta para comunicarse con el DBMS. La (Fig. IV.2) muestra algunos de los componentes de un DBMS típico, y cómo SQL actúa como el ≺adhesivo≻ que los une.

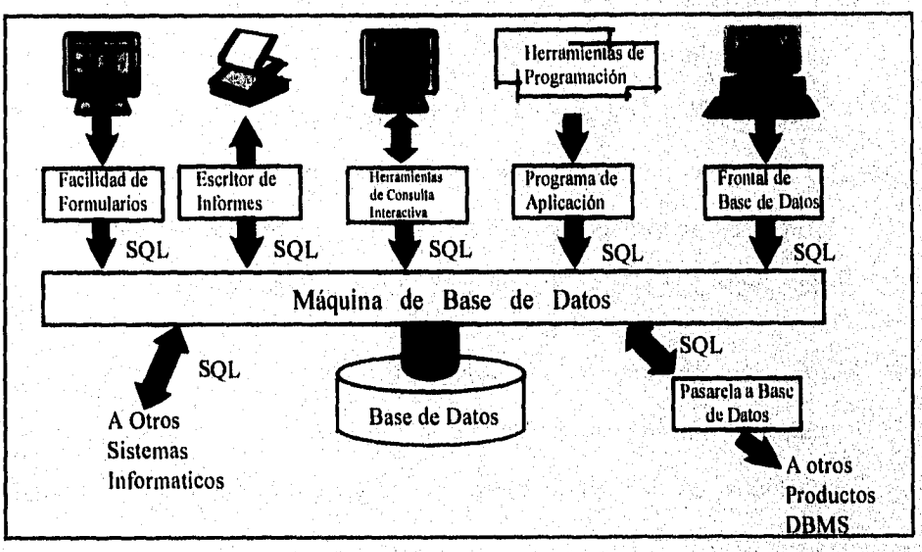


FIG. IV.2 Componentes de un Sistema Típico de Gestión de Base de Datos

La *máquina de Base de Datos* es el corazón del DBMS, responsable de estructurar, almacenar y recuperar realmente los datos en el disco. Acepta peticiones SQL procedentes de otros componentes DBMS, tales como una facilidad de formularios, un escritor de informes o una facilidad de consultas interactivas, desde los programas de aplicación escritos por el usuario e incluso desde otros sistemas informáticos. Como muestra la (Fig. IV.2), SQL juega muchos papeles diferentes:

- SQL es un *lenguaje de consultas interactivas*. Los usuarios escriben órdenes SQL en un programa SQL interactivo para recuperar datos y mostrarlos en la pantalla, proporcionando una herramienta conveniente y fácil de utilizar para consultas ad hoc de la Base de Datos.

- SQL es un *lenguaje de programación de Base de Datos*. Los programadores insertan órdenes SQL en sus programas de aplicación para acceder a los datos de la base. Tanto los programas escritos por el usuario como los programas de utilidad de la Base de Datos (tales como los escritores de informe y las herramientas de entradas de datos) utilizan esta técnica para acceso a la Base de Datos.

- SQL es un *lenguaje de administración de Base de Datos*. El administrador de la Base de Datos responsable de gestionar una Base de Datos en una minicomputadora o en un mainframe utiliza SQL

para definir la estructura de la Base de Datos y para controlar el acceso a los datos almacenados.

- SQL es un *lenguaje cliente/servidor*. Los programas de una computadora personal utilizan SQL para comunicarse sobre una red de área local con servidores de Bases de Datos que almacenan los datos compartidos. Muchas aplicaciones novedosas están utilizando esta arquitectura de cliente/servidor, que minimiza el tráfico por la red y permite que tanto las PCs como los servidores efectúen mejor su trabajo.

- SQL es un *lenguaje de Base de Datos distribuidos*. Los sistemas de gestión de Base de Datos distribuidos utilizan SQL para ayudar a distribuir datos a través de muchos sistemas informáticos conectados. El software DBMS de cada sistema utiliza SQL para comunicarse con los otros sistemas, enviando peticiones para acceso a datos.

- SQL es un *lenguaje de pasarela de Base de Datos*. En una red informática con una mezcla de diferentes productos DBMS, SQL se utiliza a menudo en una *pasarela (gateway)* que permite que nuestro producto de DBMS se comunique con otro producto.

SQL ha emergido por tanto como una herramienta potente y útil para enlazar personas y sistemas informáticos a los datos almacenados en una Base de Datos relacional.

4.1.3 Características y Beneficios de SQL

SQL es a la vez un lenguaje fácil de entender y una herramienta completa para gestionar datos.

He aquí algunas de las principales características de SQL y las fuerzas del mercado que le han hecho tener éxito:

- ❑ Su portabilidad a través de sistemas informáticos.
- ❑ Los estándares SQL.
- ❑ Su independencia de los vendedores.
- ❑ La arquitectura cliente/servidor.
- ❑ Su fundamento relacional.
- ❑ El apoyo de IBM.
- ❑ Su estructura de alto nivel semejante al inglés.
- ❑ Las consultas interactivas ad hoc.
- ❑ Su acceso a la Base de Datos mediante programas.
- ❑ Las vistas múltiples de datos.
- ❑ El ser un lenguaje de Base de Datos.
- ❑ Su definición dinámica de datos.

Estas son las razones por las que SQL ha emergido como la herramienta estándar para gestionar datos en computadoras personales, minicomputadoras y mainframes. Cada una de ellas se describe a continuación:

Portabilidad a través de sistemas informáticos

Los vendedores de DBMS en SQL ofertan sus productos sobre sistemas informáticos que van desde computadoras personales y estaciones de trabajo hasta redes de área local, minicomputadoras y mainframes. Las aplicaciones basadas en SQL que comienzan en sistemas monousuario pueden ser transferidas a sistemas mayores de minicomputadoras o mainframes cuando crecen. Los datos procedentes de Bases de Datos corporativas basadas en SQL pueden ser extraídas y remitidas a Bases de Datos departamentales o personales. Finalmente, Las económicas computadoras personales pueden ser utilizados para construir prototipos de aplicaciones de Bases de Datos basadas en SQL antes de transferirlas a un sistema multiusuario costoso.

Estándares SQL

El American National Standards Institute (ANSI) y la International Standards Organization (ISO) han publicado conjuntamente un estándar oficial para SQL. SQL se ha convertido también en un estándar del U.S. Federal Information Processing Standard (FIPS), lo que le convierte en un requerimiento esencial para los grandes contratos informáticos del gobierno. En Europa, X/OPEN, un estándar para un entorno de aplicación por cable

basado en UNIX, ha añadido SQL como el estándar para acceso a Base de Datos. La Open Software Foundation (OSF), un grupo de vendedores de UNIX, planea basar su estándar de acceso a Base de Datos en SQL. Estos estándares sirven como sello oficial de aprobación para SQL y han acelerado su aceptación en el mercado.

Independencia de los vendedores

SQL es ofertado por todos los principales vendedores de DBMS, y ningún producto nuevo de Base de Datos puede tener éxito sin el soporte de SQL. Una Base de Datos basada en SQL y los programas que la utilizan pueden transferirse de un DBMS al DBMS de otro vendedor con mínimo esfuerzo de conversión y poco reentrenamiento del personal. Herramientas de Bases de Datos basadas en SQL que funcionan con varios productos de DBMS, tales como escritores de informe y generadores de aplicación, están comenzando a aparecer. La independencia del vendedor proporcionada así por SQL es una de las razones más importantes de su popularidad.

Arquitectura cliente/servidor

SQL es un vehículo natural para implementar aplicaciones utilizando una arquitectura cliente/servidor distribuida. En este papel, SQL sirve como enlace entre los sistemas informáticos frontales (front-end) optimizados para interacción con el usuario y los sistemas de apoyo (back-end) especializados para gestión de Bases de Datos, permitiendo que cada sistema rinda lo mejor

posible. SQL También permite que las computadoras personales funcionen como frontales de Bases de Datos mayores dispuestas en minicomputadoras y mainframes, proporcionando acceso a datos corporativos desde aplicaciones informáticas personales.

Fundamento relacional

SQL es un lenguaje para Bases de Datos relaciónales, y se ha popularizado conjuntamente con el modelo de Base de Datos relacional. La estructura tabular, de filas y columnas de una base datos relacional, es intuitiva para los usuarios, y hace que el lenguaje SQL se mantenga simple y fácil de entender. El modelo relacional tiene también un fuerte fundamento teórico que ha guiado la evolución y la implementación de las Bases de Datos relaciónales. Cabalgando sobre una ola de aceptación provocada por el éxito del modelo relacional, SQL se ha convertido en el lenguaje de Base de Datos para las Bases de Datos relaciónales.

Apoyo de IBM

SQL fue inventado originalmente por investigadores de IBM, y desde entonces se ha convertido en un producto estratégico para IBM. SQL es un componente esencial de la System Application Architecture (SAA), la marca de IBM para la compatibilidad de sus diversas líneas de productos. El soporte SQL está disponible en todas las cuatro familias de sistemas incluidas bajo el paraguas SAA: Las computadoras personales PS/2, los sistemas de medio rango AS/400 y los mainframes de IBM que ejecutan los sistemas operativos

MVS y VM. Este extenso soporte de IBM ha acelerado la aceptación del mercado de SQL, y proporcionando una clara señal de las intenciones de IBM para que otros vendedores de sistemas y Bases de Datos las sigan.

Estructura de alto nivel en inglés

Las sentencias SQL parecen sencillas frases en inglés, lo que hace que SQL sea fácil de aprender y entender. Esto es en parte debido a que las sentencias de SQL describen los datos a recuperar, en lugar de especificar cómo hallar los datos. Las tablas y columnas de una Base de Datos SQL pueden tener nombres largos y descriptivos. Como consecuencia, la mayoría de las sentencias SQL dicen lo que significan, y pueden ser leídas como frases claras y naturales.

Consultas interactivas ad hoc

SQL es un lenguaje de consultas interactivas que proporciona a los usuarios acceso ad hoc a los datos almacenados. Utilizando SQL interactivamente, un usuario puede obtener respuesta incluso a cuestiones complejas en minutos o segundos, en fuerte contraste con los días o semanas que le llevaría a un programador escribir un programa de informe a medida. Debido a la potencia de consulta ad hoc de SQL, los datos son más accesibles y pueden ser utilizados para ayudar a una organización a tomar decisiones mejores y más informadas.

Acceso a la Base de Datos mediante programas

SQL es también un lenguaje de Base de Datos utilizado por los programadores para escribir aplicaciones que acceden a una Base de Datos. Las sentencias SQL se utilizan para acceso interactivo y programado, de modo que las partes de acceso a Base de Datos de un programa pueden ser comprobadas primero con SQL interactivo y luego insertadas dentro del programa. En contraste, las Bases de Datos tradicionales proporcionan un conjunto de herramientas para acceso mediante programas y una facilidad de consulta aparte para peticiones ad hoc, sin ninguna sinergía entre los dos modos de acceso.

Vistas múltiples de datos

Utilizando SQL, el creador de una Base de Datos puede dar a diferentes usuarios de la Base de Datos vistas diferentes de su estructura y contenidos. Por ejemplo, la Base de Datos puede ser construida de modo que cada usuario sólo vea datos de su propio departamento o de su propia región de ventas. Además, los datos procedentes de diferentes partes de la Base de Datos pueden combinarse y presentarse al usuario como una simple fila/columna de una tabla. Las vistas de SQL pueden ser utilizadas de este modo para mejorar la seguridad de una Base de Datos y para acomodarla a las necesidades particulares de los usuarios individuales.

Lenguaje completo de Base de Datos

SQL fue inicialmente desarrollado como un lenguaje de consulta ad hoc, pero su potencia va ahora más allá de la recuperación de datos. SQL proporciona un lenguaje complejo y consistente para crear una Base de Datos, gestionar su seguridad, actualizar sus contenidos, recuperar los datos y compartirlos entre muchos usuarios concurrentes. Los conceptos de SQL que son aprendidos en una parte del lenguaje pueden ser aplicados a otras órdenes de SQL, haciendo más productivos a sus usuarios.

4.2 COMANDOS DE SQL

- Los comandos de SQL se utilizan para crear, almacenar, cambiar, recuperar y mantener la información en una Base de Datos
- Un comando de SQL se guarda en una parte de la memoria llamada SQL Buffer, en donde se queda hasta que un nuevo comando es introducido
- tiene 17 Comandos que inician con las siguientes sentencias:

alter	drop	revoke
audit	grant	rollback
commit	insert	select
comment	lock	update
create	no audit	validate
delate	rename	

FIG. IV.3 Comandos de SQL

ALTER .- Se utiliza para modificación de una tabla (ALTER TABLA).

AUDIT.- Para definir una declaración especifica de SQL para seleccionarse en subsecuentes secciones

COMMIT.- Esta sentencia señala el final correcto de una transacción. Informa al DBMS que la transacción está ahora completa; todas las sentencias que forman la transacción han sido ejecutadas, y la Base de Datos es autoconsistente.

Una transacción es una secuencia de una o más sentencias SQL que juntas forman una unidad de trabajo. Las sentencias SQL que forman la transacción suelen estar estrechamente relacionadas y efectuar acciones interdependientes. Cada sentencia de una transacción efectúa una parte de una tarea, pero todas ellas son necesarias para completar la tarea. La agrupación de las sentencias en una sola transacción indica al DBMS que la secuencia de sentencias entera debe ser ejecutada atómicamente; todas las sentencias deben completarse para que la Base de Datos esté en un estado consistente.

COMMENT.- Se utiliza para almacenar una breve descripción de la tabla o el elemento de datos en el catálogo de sistema.

CREATE.- Se usa para crear tablas o bases de datos (CREATE TABLE, CREATE DATABASE)

DELETE.- Esta sentencia elimina filas seleccionadas de datos de una única tabla.

DROP.- Elimina una tabla de una Base de Datos (DROP TABLE CLIENTES)

GRANT.- Se utiliza para conceder privilegios de seguridad sobre objetos de la Base de Datos a usuarios específicos. Normalmente esta sentencia es utilizada por el propietario de una tabla para proporcionar a otros usuarios acceso a los datos.

INSERT.- Se utiliza para insertar filas dentro de una tabla

LOCK.- Esta sentencia se utiliza para cerrar explícitamente una tabla entera.

NO AUDIT.- Para desactivar la declaración definida por AUDIT

RENAME.- Cambia el nombre de una tabla, una vista, etc.

REVOKE.- Los privilegios que se han concedido con la sentencia GRANT pueden ser retirados con esta sentencia

ROLLBACK.- Esta sentencia señala el final sin éxito de una transacción. Informa al DBMS que el usuario no desea completar la transacción; en vez de ello, el DBMS debe deshacer los cambios efectuados a la Base de Datos durante la transacción.

SELECT.- Permite realizar consultas sobre datos de una sola tabla.

UPDATE.- Modifica los valores de una o más columnas en las filas seleccionadas de una tabla única.

4.3 COMANDOS DE SQL*PLUS (ORACLE) NO ENCONTRADOS EN SQL

- Además de los comandos estándar de SQL, el lenguaje SQL*Plus de Oracle incluye comandos adicionales, llamados comandos SQL*Plus
- Estos comandos no se almacenan en el buffer de SQL
- Los comandos SQL*Plus se utilizan para generar reportes sofisticados, editar sentencias de SQL, proveer una facilidad de ayuda y mantener variables del sistema.

@	define	pause
#	del	quit
$	describe	remark
/	disconnect	run
accept	document	save
append	edit	set
break	exit	show
btitle	get	spool
change	help	sqlplus
clear	host	start
column	input	timing
compute	list	ttitle
connect	newpage	undefine
copy		

FIG. IV.4 Comandos de SQL*Plus (Oracle) No Encontrados en SQL

@.- Corre el contenido de un archivo de comandos. Es similar al comando STAR, pero no permite argumentos en la línea de comando

#.- Complementa un bloque de documentación en SQL*PLUS al iniciar un archivo donde el bloque es iniciado por la palabra DOCUMENT. SQL*PLUS ignora todas las líneas desde la línea donde se encuentra la palabra DOCUMENT hasta la línea donde se encuentra (#).

$.- Se usa para correr cualquier comando desde el sistema operativo sin salir de SQL*PLUS. $ es una abreviación corta de HOST

/ .- Corre la instrucción que se encuentre en el Buffer

ACCEPT.- Permite que desde el teclado el usuario pueda definir el nombre de una variable

APPEND.- Agrega una línea al comando de SQL

BREAK.- Suprime repetición de valores

BTITLE.- Pone un encabezado en el pie de página

CHANGE.- Cambia la primera ocurrencia de cierto texto en una línea

CLEAR.- Para borrar y desabilitar comandos

COLUMN.- Especifica el nombre de la columna de la tabla que se va a imprimir

COMPUTE.- Se utiliza para hacer operaciones

CONNECT.- Es un nivel de privilegio de conexión, únicamente para consultas

COPY.- Copia la información una tabla a otra computadora sobre SQL*NET

DEFINE.- Determina que si desde SQL*PLUS se podrán cambiar y sustituir comandos para variables de substitución y cárgalas con sus valores definidos.

DEL.- Borra una línea

DESCRIBE.- Despliega la estructura de una tabla específica

DISCONNECT.- Desconecta algunos privilegios de una Base de Datos

DOCUMENT.- Se utiliza al inicio de un bloque para documentar al archivo. Para finalizar el bloque se pone al final de la línea el signo (#)

EDIT.- Para correr el editor del sistema operativo desde SQL Plus

EXIT.- Sale de la sección de SQL*PLUS y retorna al usuario al sistema operativo, llamando a un programa o menú

GET.- Este comando trae el contenido de un archivo al buffer de SQL y lo despliega en la pantalla

HELP.- Despliega información acerca de un comando en particular que se desee consultar

HOST.- El comando HOST en SQL*PLUS o SQL*FORMS es un pase para regresar al sistema operativo momentáneamente y ejecutar cualquier comando huésped sin salirse de SQL*PLUS o SQL*NET

INPUT.- Agrega una línea al comando de SQL

LIST.- Despliega el contenido del buffer de SQL

NEWPAGE.- Pone el número de líneas en blanco que se imprimirán entre la parte inferior de una página y la parte superior de la siguiente página

PAUSE.- Es similar al PROMPT, excepto que pausa primero despliega una línea en blanco y la siguiente línea contiene el texto y espera hasta que el usuario presione RETURN

QUIT.- Sale de la sección de SQL*PLUS y regresa el usuario al sistema operativo, llamando a un programa o menú

REMARK.- Se usa para poner una sola línea de observaciones, para documentar un archivo

RUN.- Lista y corre las instrucciones que se encuentren en el Buffer

SAVE.- Para salvar a disco un comando de SQL

SET.- Para especificar valores de los parámetros

SHOW.- Despliega los parámetros de la sección corriente de SQL Plus

SPOOL.- Para guardar los resultados de un query en un archivo y desplegarlos en la pantalla.

SQLPLUS.- Para conectarse a SQL*PLUS

START.- Este comando recupera el contenido de un archivo de comandos y lo corre

TIMING.- Guarda el tiempo que tarda una pista en iniciar o parar por área

TTITLE.- Se utiliza para poner el titulo de una página. Al utilizar este comando aparecen automáticamente la fecha y el número de página

UNDEFINE.- Borra la definición de una variable de usuario que ha sido definida por ACCEPT, DEFINE, o como un parámetro por el comando START

4.4 CONSTRUCCIÓN DE LA BASE DE DATOS

Las sentencias SELECT, INSERT, DELETE, UPDATE, COMMIT y ROLLBACK descritas anteriormente se refieren todas a la manipulación de los datos de una Base de Datos. Estas sentencias se denominan colectivamente lenguaje de Manipulación de Datos de SQL, o DML (Data Manipulation Languaje). Las sentencias DML pueden modificar los datos almacenados en una Base de Datos, pero no pueden cambiar su estructura. Ninguna de estas sentencias crea o suprime tablas o columnas, por ejemplo.

Los cambios a la estructura de una Base de Datos son manejados por un conjunto diferente de sentencias SQL, denominadas conjuntamente Lenguaje de definición de Datos de SQL, o DDL (Data Definition Languaje). Utilizando sentencias DDL, se puede:

- ❑ Definir y crear una nueva tabla

- ❑ Suprimir una tabla que ya no se necesita
- ❑ Cambiar la definición de una tabla existente
- ❑ Definir una tabla virtual (o vista) de datos
- ❑ Establecer controles de seguridad para una Base de Datos
- ❑ Construir un índice para hacer más rápido el acceso a la tabla
- ❑ Controlar el almacenamiento físico de los datos por parte del DBMS

En su mayor parte, las sentencias DDL aíslan al usuario de los detalles de bajo nivel referentes a cómo los datos están físicamente almacenados en la Base de Datos. Manipulan objetos abstractos de la Base de Datos, tales como tablas y columnas. Sin embargo, el DDL no puede evitar completamente las cuestiones referentes al almacenamiento físico, y , por necesidad, las sentencias DDL y las cláusulas que controlan el almacenamiento físico varían de un DBMS a otro.

El núcleo del Lenguaje de Definición de Datos está basado en tres verbos SQL:

- ❑ CREATE, que define y crea un objeto de la Base de Datos
- ❑ DROP, que elimina un objeto existente en la Base de Datos
- ❑ ALTER, que modifica la definición de un objeto de la Base de Datos

En todos los principales DBMS basados en SQL, estos tres verbos DDL pueden ser utilizados mientras el DBMS está corriendo. La estructura de la Base de Datos es por tanto dinámica. El DBMS puede crear, eliminar o alterar la definición de las tablas en la Base de Datos, por ejemplo, mientras simultáneamente proporciona acceso a la Base de Datos a sus usuarios. Esta es una ventaja importante de SQL y de las Bases de Datos relaciónales sobre los sistemas anteriores, en los que el DBMS tenía que ser detenido antes de que pudiera modificar la estructura de la Base de Datos. Esto significa que una Base de Datos relacional puede crecer y cambiar fácilmente en el tiempo. El uso de producción de una Base de Datos puede continuar mientras se le añaden nuevas tablas y aplicaciones.

Aunque el DDL y el DML son dos partes distintas del lenguaje SQL, en la mayoría de los productos DBMS basados en SQL la división es solamente conceptual. Generalmente las sentencias DDL y DML se remiten al DBMS exactamente del mismo modo, y pueden ser libremente entremezcladas en aplicaciones de SQL programado en sesiones de SQL interactivo. Si un programa o usuario necesita una tabla para almacenar sus resultados temporales, puede crear una tabla, rellenarla, manipular los datos y luego eliminarla. De nuevo, ésta es una ventaja importante con respecto a los modelos de datos más primitivos, en donde la estructura de la Base de Datos quedaba fijada cuando se creaba la Base de Datos.

Aunque prácticamente todos los productos SQL comerciales soportan el DDL como parte integral del lenguaje SQL, el estándar SQL ANSI/ISO no lo

exige. De hecho el estándar actual ANSI/ISO implica una fuerte separación entre el DML y el DDL.

En una instalación DBMS grande para mainframes, el administrador de la Base de Datos es el único responsable de la creación de nuevas Bases de Datos. En instalaciones DBMS sobre minicomputadoras más pequeñas, los usuarios individuales pueden permitirse crear sus propias Bases de Datos personales, pero es mucho más habitual que las Bases de Datos sean creadas centralizadamente y luego accedidas por los usuarios individuales. Si un usuario esta utilizando un DBMS en una computadora personal, será probablemente a la vez administrador y usuario de la Base de Datos, y tendrá que crear personalmente las Bases de Datos que utilice.

El estándar ANSI/ISO no especifica cómo se crean las Bases de Datos, y cada producto DBMS adopta un planteamiento ligeramente diferente. Las técnicas utilizadas por algunos de los productos SQL más importantes ilustran las diferencias:

- Oracle crea una Base de Datos como parte del proceso de instalación del software Oracle. En su mayor parte, las tablas de usuario se colocan siempre en esta Base de Datos única global.

- SQL Server al igual que SQL informix incluyen una sentencia CREATE DATABASE como parte de su lenguaje de definición de datos. Una sentencia adicional DROP DATABASE destruye Bases de Datos

creadas previamente. Estas sentencias pueden ser utilizadas con SQL interactivo o programado.

- OS/2 Extended Edition proporciona una utilidad CREATE DATABASE para crear Bases de Datos y una utilidad DROP DATABASE para eliminar Bases de Datos. También proporciona llamadas a funciones especiales que pueden ser utilizadas mediante programas de aplicación para crear y suprimir Bases de Datos.

- SQLBase utiliza la orden de MS-DOS COPY para crear una nueva Base de Datos. El usuario simplemente hace una copia duplicada de una plantilla de Base de Datos vacía suministrada con el software de SQLBase. La Base de datos puede ser suprimida posteriormente con la orden de MS-DOS DEL.

- Ingres incluye un programa de utilidad especial, llamado CREATEDB, que será una nueva Base de Datos Ingres. Un programa adicional, DESTROYDB, suprime una Base de Datos que ya no es necesaria.

A causa de estos planteamientos diferentes, es preciso consultar los manuales de cada DBMS particular antes de proceder a crear una nueva Base de Datos. Tras crear una Base de Datos vacía, el siguiente paso es rellenarla con tablas, tal como se describe a continuación.

4.4.1 Definiciones de Tablas

La estructura más importante de una Base de Datos relacional es la tabla. En una Base de Datos multiusuario, las tablas principales son típicamente creadas una vez por el administrador de la Base de Datos y utilizadas luego día tras día. Conforme se utiliza la Base de Datos con frecuencia se encontrará conveniente definir tablas propias para almacenar datos personales o datos extraídos de otras tablas. Estas tablas pueden ser temporales, que duran únicamente una sesión SQL interactiva, o más permanentes, con una duración de semanas o meses. En una Base de Datos sobre una computadora personal, la estructura de las tablas es incluso más fluido. Puesto que uno es a la vez usuario y administrador de la Base de Datos, puede crear o destruir las tablas conforme a sus propias necesidades, sin preocuparse del resto de usuarios.

4.4.2 Creación de una Tabla (CREATE TABLE)

La sentencia CREATE TABLE, mostrada en la (Fig. IV.5), define una nueva tabla en la Base de Datos y la prepara para aceptar datos. Las diferentes cláusulas de la sentencia especifican los elementos de la definición de la tabla. El diagrama sintáctico de la sentencia parase complejo, ya que hay muchas partes de la definición que especificar y muchas opciones para cada elemento. En la práctica, la creación de una nueva tabla es relativamente sencilla.

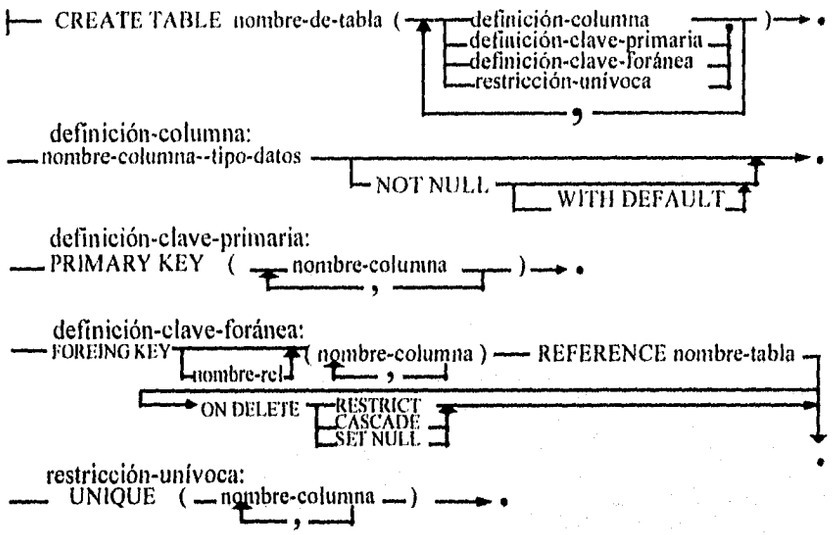


FIG. IV.5 Diagrama sintáctico básico de la sentencia CREATE TABLE

Cuando se ejecuta una sentencia CREATE TABLE, uno se convierte en el propietario de la tabla recién creada, a la cual se le da el nombre especificado en la sentencia. El nombre de la tabla debe ser un nombre SQL legal, y no debe entrar en conflicto con el nombre de alguna otra tabla ya existente. La tabla recién creada está vacía, pero el DBMS la prepara para aceptar datos añadidos con la sentencia INSERT.

4.4.3 Definición de columnas

Las columnas de la tabla recién creada se definen en el cuerpo de la sentencia CREATE TABLE. Las definiciones de columnas aparecen en una lista separada por comas e incluida entre paréntesis. El orden de las definiciones de las columnas determina el orden de izquierda a derecha de las columnas en la tabla. Cada definición especifica:

- El nombre de la columna, que se utiliza para referirse a la columna en sentencias SQL. Cada columna de la tabla debe tener un nombre único, pero los nombres pueden ser iguales a los de las columnas de otras tablas.

- El tipo de datos de la columna, que identifica la clase de datos que la columna almacena. Algunos tipos de datos, tales como VARCHAR y DECIMAL, requieren información adicional, como la longitud o número de lugares decimales de los datos. Esta información adicional se incluye entre paréntesis a continuación de la palabra clave que especifica el tipo de datos.

- Si la columna contiene datos requeridos. La cláusula NOT NULL impide que aparezcan valores NULL en la columna; en caso contrario se permiten los valores NULL.

- Un valor por omisión opcional para la columna. El DBMS utiliza este valor cuando una sentencia INSERT aplica(da) a la tabla no especifica un valor para la columna.

La siguiente sentencia CREATE TABLE es un ejemplo de como crear una tabla:

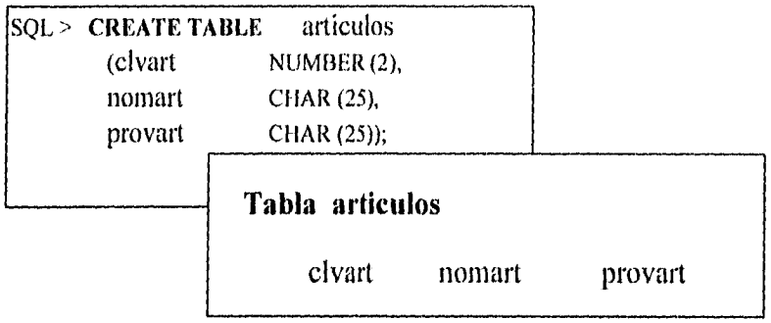


FIG. IV.4 Sentencia CREATE

La sentencia CREATE TABLE varía ligeramente de un producto DBMS a otro, ya que cada DBMS soporta su propio conjunto de tipos de datos y utiliza sus propias palabras clave para identificarlos en las definiciones de columnas. Algunos DBMS como Sybase y SQL Server difieren mucho de otros productos DBMS y del estándar ANSI/ISO en su manejo de valores NULL. El estándar especifica que una columna puede contener valores NULL a menos que específicamente se declare NOT NULL. Sybase y SQL Server utilizan el convenio opuesto, suponiendo que los valores NULL no son permitidos a menos que la columna se declare explícitamente NULL.

- A veces se requiere asegurar de que una columna no tenga valores nulos
- Para prohibir que se inserten valores nulos en una columna, se debe poner la cláusula NOT NULL después de la información de la columna

```
SQL > CREATE TABLE    artículos
  (clvart          NUMBER (2) NOT NULL,
   nomart          CHAR (25),
   provart         CHAR (25));
```

FIG. IV.6 Definición de Valores NO NULOS

4.4.4 Definiciones de Clave Primaria y Foránea

Además de la definición de las columnas de una tabla, la sentencia CREATE TABLE identifica la clave primaria de la tabla y las relaciones de la tabla con otras tablas de la Base de Datos. Las cláusulas PRIMARY KEY y FOREING KEY manejan estas funciones. Estas cláusulas son soportadas por las Bases de Datos IBM (DB2, SQL/DS y OS/2 Extended Edition) y han sido añadidas a la especificación ANSI/ISO original.

La cláusula PRIMARY KEY especifica la columna o combinación de columnas que forman la clave primaria de la tabla, que sirve como identificador único para cada fila de la tabla. El DBMS requiere automáticamente que el valor de la clave primaria sea único en cada fila de la tabla. Además, la definición de columna para todas las columnas que forman la clave primaria debe especificar que la columna es NOT NULL.

La cláusula FOREING KEY especifica una clave foránea en la tabla y la relación que crea con otra tabla (padre) de la Base de Datos. La cláusula especifica:

- La columna o columnas que forman la clave foránea, todas las cuales son columnas de la tabla que está siendo creada
- La tabla que es referenciada por la clave foránea. Esta es la tabla padre en la relación; la tabla que está siendo definida es el hijo.
- Un nombre opcional para la relación. El nombre no se utiliza en ninguna sentencia SQL, pero puede aparecer un mensaje de error y es necesaria si se desea poder suprimir la clave foránea posteriormente.
- Una regla de supresión opcional para la relación (RESTRICT, CASCADE o SET NULL). Si no se especifica regla de supresión, se utiliza la regla RESTRICT.

4.4.5 Restricciones de Unicidad

El estándar ANSI/ISO especifica que las reacciones de unicidad también se definen en la sentencia CREATE TABLE, utilizando la cláusula UNIQUE mostrada en la siguiente tabla.

```
SQL > CREATE TABLE artículos UNIQUE
        (clvart      NUMBER (4) PRIMARY KEY,
        nomart       CHAR (25), REFERENCES art(artículos)
        provart      CHAR (25));
```

FIG. IV.7 Restricción de unicidad

Si una clave primaria, una clave foránea o una restricción de unicidad afecta a una sola columna, el estándar ANSI/ISO permite una forma <abreviada> de la definición. La clave primaria, la clave foránea o la restricción de unicidad se añaden simplemente al final de la definición de la columna, tal como se muestra en la tabla siguiente:

```
SQL > CREATE TABLE artículos UNIQUE
        (clvart     NUMBER (4) NOT NULL PRIMARY KEY,
        nomart      CHAR (25), REFERENCES art( artículos)
        provart     CHAR (25));
```

FIG. IV.8 Forma abreviada de definiciones

Aunque esta abreviatura forma parte del estándar ANSI/ISO, no es aceptada por los productos SQL de IBM, que han sido los pioneros en el soporte de claves primarias y foráneas.

4.4.6 Definición de almacenamiento físico

La sentencia CREATE TABLE incluye típicamente una o más cláusulas opcionales que especifican características de almacenamiento físico para la tabla. Generalmente estas cláusulas sólo las utiliza el administrador de la Base de Datos para optimizar el rendimiento de una Base de Datos de producción. Por su naturaleza estas cláusulas son muy específicas de un DBMS particular. Aunque son de poco interés práctico para la mayoría de los usuarios de SQL, las diferentes estructuras de almacenamiento físico

proporcionadas por los diferentes productos DBMS ilustran sus diferentes aplicaciones y niveles de sofisticación.

La mayoría de las Bases de Datos sobre computadoras personales disponen de mecanismos de almacenamiento físico muy simples. El DBMS dBASE IV, por ejemplo, utiliza un archivo MS-DOS individual para almacenar cada tabla de datos. El uso de archivos MS-DOS limita el tamaño de una tabla o una Base de Datos. También requieren que la tabla o la Base de Datos entera sean almacenadas en un único volumen de disco físico.

Las Bases de Datos multiusuario proporcionan típicamente esquemas de almacenamiento físico más sofisticados para mejorar el rendimiento de las Bases de Datos. Por ejemplo, Ingres permite que el administrador defina múltiples ubicaciones nominadas, que son directorios físicos en donde los datos de la base pueden ser almacenados. Las ubicaciones pueden extenderse a través de múltiples volúmenes para aprovechar las operaciones paralelas de entrada/salida sobre los discos. Opcionalmente se puede especificar una o más ubicaciones para la tabla en la sentencia CREATE TABLE de Ingres:

```
CREATE TABLE EMPLEADOS (definición-de-table)
    WITH LOCATION = (AREA1, AREA2, AREA3)
```

Especificando múltiples ubicaciones, se puede dispensar los contenidos de una tabla a través de varios volúmenes de disco con objeto de ampliar el acceso en paralelo a la tabla.

DB2 ofrece un esquema muy complejo para gestionar el almacenamiento físico. El conjunto completo de tablas gestionado por un subsistema DB2 puede dividirse en dos o más Bases de Datos DB2. Notese que este uso del término <Base de Datos> es confuso, ya que una Base de Datos DB2 es una estructura interna utilizada para gestionar el almacenamiento físico de los datos, y no una estructura visible al usuario. Cada una de estas Bases de Datos se subdivide a su vez en espacios de tablas y espacios de índices en DB2, entidades de almacenamiento lógico que almacenan tablas e índices, respectivamente. Los espacios de tablas y los espacios de índices pueden ser asignados a su vez en un grupo de almacenamiento, una entidad de almacenamiento consta de uno o más volúmenes de memoria de acceso directo sobre el mainframe. Las Bases de Datos, los espacios de tablas y los grupos de almacenamiento se gestionan utilizando las sentencias CREATE DATABASE, CREATE TABLESPACE y CREATE STOPGROUP en el DDL de DB2. Cuando se crea una tabla en DB2, puede asignársele opcionalmente a un espacio de tablas específico:

```
CREATE TABLE OFICINAS (definición-de-table)
  IN BDADMIN.ESPACIO
```

4.4.7 ¿Qué es un JOIN en SQL?

Un JOIN es una de las características más importantes de SQL y consiste en la unión lógica de dos o más tablas. En otros términos, es un *query* en la

cual se accesan datos de tablas diferentes, las cuales se relacionan entre sí por medio de un campo común. En la (Fig. IV.9) se observa un JOIN.

```
SQL > SELECT   clvemp, nomemp, emp.clvdept, nomdept
      FROM     emp, dept
      WHERE    emp.clvdept = dept.clvdept
```

FIG. IV.9 Un JOIN en SQL

4.4.8 Eliminación de una tabla (DROP TABLE)

Con el tiempo la estructura de una Base de Datos crecerá y se modificará. Nuevas tablas serán creadas para representar nuevas entidades, y algunas tablas antiguas ya no serán necesarias. Se puede eliminar de la Base de Datos una tabla que ya no es necesaria con la sentencia DROP TABLE.

El nombre de la tabla en la sentencia identifica la tabla a eliminar. Normalmente se eliminará una de las tablas propias del usuario y se utilizará un nombre de tabla no cualificado. Con el permiso adecuado, también se puede eliminar una tabla propiedad de otro usuario especificando un nombre de tabla cualificado:

```
SQL > DROP TABLE artículos
```

FIG. IV.10 Eliminación de una tabla de la Base de Datos

Cuando la sentencia DROP TABLE suprime una tabla de la Base de Datos, su definición y todos sus contenidos se pierden. No hay manera de recuperar los datos, y habría que utilizar una nueva sentencia CREATE TABLE para volver a crear la sentencia de la tabla. Debido a sus serias consecuencias, debe utilizarse la sentencia DROP TABLE con mucho cuidado.

4.4.9 **Modificación de una definición de tabla** (ALTER TABLE)

Después que una tabla ha sido utilizada durante algún tiempo, los usuarios suelen descubrir que desean almacenar información adicional con respecto a las entidades representadas en la tabla tales como:

- ❑ Añadir una definición de columna a una tabla
- ❑ Definir una clave primaria para una tabla
- ❑ Definir una nueva clave foránea para una tabla
- ❑ Eliminar una clave primaria o foránea existente para una tabla

Cada uno de estos cambios, y algunos otros, pueden ser realizados con la sentencia ALTER TABLE. Al igual que con la sentencia DROP TABLE, ALTER TABLE se utilizará normalmente sobre tablas propias. Con el permiso adecuado, sin embargo, se puede especificar un nombre de tabla cualificado y alterar la definición de la tabla de otro usuario.

Muchos productos DBMS ofrecen cláusulas propias de su DBMS, que alteran otras características de las tablas.

Adicionar el número telefónico a la tabla EMPLEADOS

```
SQL > ALTER TABLE empleados
      ADD notel CHAR (10)
```

FIG. IV.11 Modificación de una

4.4.10 Supresión de una columna

La sentencia ALTER TABLE no puede ser utilizada para eliminar una columna existente, y de hecho SQL no proporciona un método sencillo de eliminar una columna. Naturalmente es posible ignorar sin más una columna que ya no sea necesaria, manteniendo su definición en la tabla. Sin embargo, si realmente se desea eliminar una columna de la tabla, se debe:

a) Descargar todos los datos de la tabla.

b) Utilizar la sentencia DROP TABLE para borrar la definición de la tabla.

c) Utilizar la sentencia CREATE TABLE para definir la tabla sin la columna no deseada.

d) Volver a cargar todos los datos que fueron anteriormente descargados.

La descarga y carga masiva de datos que requieren los Pasos a) y d) se realiza típicamente con programas de utilidad de propósito especial, en lugar de con sentencias SELECT o INSERT.

4.4.11 **Indices** (CREATE/DROP INDEX)

Una de las estructuras de almacenamiento físico proporcionadas por la mayoría de los DBMS basados en SQL es el índice. Un índice es una estructura que proporciona un acceso rápido a las filas de una tabla en base a los valores de una o más columnas.

Para crear un índice, se utiliza la sentencia CREATE INDEX, mostrada en la (Fig. IV.12) La sentencia asigna un nombre al índice y especifica la tabla para cual se crea el índice. La sentencia también especifica la columna o columnas a indexar y si deberían ser indexadas en orden ascendente o descendente. La palabra clave UNIQUE especifica que la columna que está siendo indexada debe contener valores únicos.

```
SQL> CREATE UNIQUE INDEX EMP_EMPNO
     ON EMPLEADOS(EMPNO);
```

FIG.1V.12 Creacción de un índice

Además de las cláusulas mostradas en la figura anterior, la sentencia CREATE INDEX incluye casi siempre cláusulas adicionales específicas de cada DBMS que especifican la ubicación en disco para el índice y parámetros para

mejora del rendimiento tales como el tamaño de las páginas de índice y el porcentaje de espacio libre que el índice debería permitir para filas adicionales.

Si se crea un índice para un tabla y posteriormente se decide que ya no es necesario, la sentencia DROP INDEX suprime el índice de la Base de Datos.

```
SQL > DROP INDEX EMP_EMPNO
```

FIG. IV.13 Borrado de un índice

4.4.12 Recuperación y Concurrencia

Una de las funciones más importantes de un DBMS basado en SQL es que controla de manera transparente el hecho de que los usuarios compartan datos de la Base de Datos. Es importante que el estado actual de la Base de Datos sea siempre el mismo para todos los usuarios. Esta condición no tendría validez si se permitiera que dos usuarios actualizaran el mismo registro de la Base de Datos al mismo tiempo: el usuario que actualizara primero, perdería los datos, ya que la actualización del segundo se sobrepondría.

El DBMS mantiene y controla la concurrencia por medio de bloqueos. Cuando un usuario accesa determinados datos, es "dueño" de ellos durante el tiempo que dure la transacción. Esta es un conjunto de una o más instrucciones SQL relacionadas. Los bloqueos se pueden aplicar a toda la

Base de Datos, a una sola tabla o a un solo registro. En un programa las transacciones se definen con START TRANSACTION y COMMIT.

La operación COMMIT indica el fin exitoso de la transacción y libera a la Base de Datos o registros bloqueados.

Es necesario tener presente que las cosas pueden salir mal; por ejemplo, se puede dañar el sistema a la mitad de una transacción. Si esto sucede, nunca se ejecuta la transacción COMMIT y se genera una serie de errores. El programa puede recuperar la Base de Datos mediante un ROLLBACK. El ROLLBACK regresa la Base de Datos al estado en el que se encontraba antes de iniciar la última transacción. Esto garantiza que los datos no queden incompletas o inconsistentes.

4.4.13 Instrumentación de SQL

El uso de SQL dentro de un programa de aplicación se puede hacer básicamente de dos maneras: SQL integrado (*embedded* SQL) y con un API (*Application Program interface*; Interface para programas de aplicación)

El SQL integrado permite a los programadores incluir instrucciones SQL directamente en el código del programa fuente, junto con las instrucciones del lenguaje de programación. Se utiliza un precompilador SQL para que compile tanto el SQL como el lenguaje y se obtenga un programa ejecutable.

Cuando se utilizan APIs, no se trata de mezclar SQL con el lenguaje de programación. En su lugar, el DBMS debe proporcionar una librería de funciones, conocida como API. Los programas se comunican con el DBMS

por medio del API. para ejecutar un query y obtener resultados. Por otro lado, el compilador del lenguaje de programación se encarga de interpretar sus propias instrucciones. Este método no requiere de un precompilador.

Si se utiliza SQL integrado es mucho más sencillo portar una aplicación de un sistema a otro.

```
(Instrucciones en leguaje 'C')
SELECT    clvemp, nomemp, emp.clvdept, nomdept
FROM      emp, dept
WHERE     emp.clvdept = dept.clvdept
(instrucciones en lenguaje 'C')
```

FIG. IV.14 SQL Integrado

4.4.14 Sintaxis SQL*PLUS y PL/SQL

SELECT, el comando básico

SELECT [DISTINCT] <col1 [alias1],
col2 [alias2],.... | * >
FROM <tabla>;

Selección de Renglones de una tabla

SELECT <col1, col2,....>
FROM <tabla>
WHERE <columna><operador><valor>;

Operadores:

=	Igual
<	Menor que
!=	Diferente
<=	Menor o Igual
>	Mayor que
>=	Mayor o Igual

Búsqueda de Rangos

SELECT <col1, col2,....>
FROM <tabla>
WHERE <columna> [NOT] BETWEEN
<valor> AND <valor2>;

Búsqueda con Patrones de Caracteres

SELECT <col1, col2,....>
FROM <tabla>
WHERE <columna> [NOT] LIKE
<'cadena de caracteres'>;

- % Cualquier número de caracteres
- \- Un solo caracter

Búsqueda de Valores Nulos

SELECT <col1, col2,....>
FROM <tabla>
WHERE <columna> IS [NOT] NULL;

Comparación con un Conjunto de Valores

SELECT <col1, col2,....>
FROM <tabla>
WHERE <columna> [NOT] IN
(<valor1>,valor2>,....);

Condiciones Múltiples de Búsqueda

SELECT <col1, col2,....>
FROM <tabla>
WHERE <condición>
AND <condición>;

Condiciones Alternativas de Búsqueda

SELECT <col1, col2,....>
FROM <tabla>
WHERE <condición>
OR <condición>;

Ordenando Renglones

SELECT [DISTINCT] <col1, col2,.... | * >
FROM <tabla>
ORDER BY <co1, col2,....> [ASC | DESC];

Cálculos

SELECT <col1, col2,....>, <expresión-aritmética>
FROM <tabla>
[WHERE <condición-con-expresión-aritmética>]
[ORDER BY <col1, col2,....>,
<expresión-aritmética>;

Operadores Aritméticos

*	Multiplicación
/	División
+	Suma
-	Resta

Funciones

SELECT función ([DISTINCT] <columna>)
FROM <tabla>
[WHERE <condición>];

Funciones

AVG
SUM
MIN
MAX
COUNT (puede usarse con *)

Agrupación

```
SELECT <col1, col2,....>, <función (argumento)>
FROM <tabla>
[WHERE <condición>]
[GROUP BY <col1, col2,....>]
[HAVING <condición>]
[ORDER BY <col1, col2,....> [ASC | DESC]]
```

Subquerys

```
Query       SELECT <col1, col2,....>
Principal   FROM <tabla>
            WHERE <columna>]
            <operador | IN.>
Subquery    (SELECT <columna>
            FROM <tabla>
              [WHERE <condición>]);
```

Joins

```
SELECT <col1, col2,....>
FROM <tabla1>, <tabla2> [,....,<tabla-n>]
WHERE <tabla1.col>  =  <tabla2.col>
            --condición de join
AND <tabla2.col>  =  <tabla3.col>
            --condición de join
....
AND <tabla-n-1.col>  =  <tabla-n.col>
            --condición de join
[AND | OR <condición>]
[ORDER BY <col1, col2,....> ];
```

Condiciones de join necesarias=Número de tablas - 1

Lenguaje de Manipulació de Datos

```
INSERT INTO <tabla> [(<col1>, col2>,....>)]
VALUES (<valor1>, <valor2>,....);]
INSERT INTO <tabla1> [(col1, col2,....>)]
SELECT <col1, col2,....>
FROM <tabla>
[WHERE <condición>]

UPDATE <tabla | vista>
SET <col1> = <valor | expresión>,
....    ....
  <col-n> = <valor | expresión>
[WHERE <condición>];

DELETE FROM <tabla>
[WHERE <condición>];
```

Proceso de Transacciones

```
COMMIT;
ROLLBACK TO <marca>
SAVEPOINT <marca>;
```

Lenguaje de Definición de Datos

```
CREATE TABLE <tabla>
    (<columna1> <tipo-de-dato> <constraint-de-columna>,
     <columna2> <tipo-de-datos> <constraint-de-columna>,
....  ....  ....
     <columna-n> <tipo-de-datos> <constraint-de-columna>;
       Constraints:
            PRIMARY KEY
            CHECK (<condición>)
            REFERENCES (columna)
            [NOT] NULL
            UNIQUE
```

ALTER TABLE <tabla>
{ADD | MODIFY} <columna> <tipo-de-dato> <constraint-de-columna>;

DROP TABLE <tabla>;

CREATE SYNONYM <sinónimo>
FOR <tabla | vista>;

DROP SYNONYM <sinónimo>;

CREATE [UNIQUE] INDEX <índice>
ON <tabla> (<columna> [, <columna>]....);

DROP INDEX <índice>;

CREATE SEQUENCE <secuencia>
[INCREMENT BY {1|n}]
[START WITH <n>]
[{MAXVALUE <n> | NOMAXVALUE}]
[{MINVALUE <n> | NOMINVALUE}]
[{CYCLE | NOCYCLE}];

DROP SEQUENCE <secuencia>;

CREATE VIEW <vista> [(<col1, col2,....)]
AS <sentencia SAL>
[WITH CHECK OPTION];

DROP VIEW <vista>;

Lenguaje de Control de Datos

GRANT <ALL | SELECT | UPDATE | INSERT | DELETE | INDEX | ALTER>

ON <tabla | vista>
TO <usuario(s) | PUBLIC>
[WITH GRANT OPTION];

REVOKE <ALL | SELECT | UPDATE | INSERT | DELETE |
INDEX | ALTER>
ON <tabla | vista>
FROM <usuario(s)>;

PL/SQL

Estructura de un Bloque PL/SQL

DECLARE
<declaración de objetos>;
BEGIN
<secuencia de instrucciones>;
EXCEPTION
<manejadores de errores>;
END;

Declaración de Variables

Declaración Escalar

<identificador> [CONSTAT]
<tipo-de-dato>
[NOT NULL] [:=<expresión_de_plsql];

Declaración de Tablas

TYPE <nombre-tipo> IS TABLE OF
{<tipo-columna> | <tabla.columna%TYPE>}
[NOT NULL]
INDEX BY BINARY_INTEGER;

<identificador> <nombre-tipo>;

Declaración de Registros

TYPE <nombre-tipo> IS RECORD
(<nom-campo1> {<tipo-reg> | <tabla.columna%TYPE>} [NOT NULL]
<nom-campo1> {<tipo-reg> | <tabla.columna%TYPE>} [NOT NULL],
...);

<identificador> <nombre-tipo>;

Comando IF

IF <condición> THEN
<secuencia de instrucciones>;
[ELSIF <condición> THEN
<secuencia de instrucciones>;]
ELSIF se puede repetir n veces
[ELSE
<secuencia de instrucciones>;]
END IF;

Comando LOOP

LOOP
<secuencia de instrucciones>;
END LOOP;

Para salir del LOOP se usa el comando EXIT:
EXIT [WHEN <condición>];

FOR Numérico

FOR <índice> in [REVERSE] <entero>....
<entero> LOOP
<secuencia de instrucciones<;
END LOOP;

Comando WHILE

WHILE <condición> LOOP
<secuencia de instrucciones>;

END LOOP;

Comando GOTO

<<etiqueta>>
<secuencia de instrucciones>;
GOTO etiqueta;

Declaración de Cursores

CURSOR <nombre-cursor> [(parámetro-tipo)]
IS <sentencia select>;

Abrir Cursores

OPEN <nombre-cursor>;

Hacer Fetch a los Datos del Cursor

FETCH <nombre-cursor> INTO <var1, var2.....>;

Cerrar Cursores

CLOSE <nombre-cursor>;

Comando FOR para Cursores

FOR <nombre-registro> IN <nombre cursor> LOOP
<secuencia de instrucciones>;
END LOOP;

Manejo de Excepciones

WHEN <nombre-excepción> [OR <nombre-excepción>...] THEN
<secuencia de instrucciones>;

o

WHEN OTHERS THEN
<secuencia de instrucciones>;

4.4.15 Arquitectura de Base de Datos Unica

La (Fig. IV.15) muestra una arquitectura de Base de Datos única en donde el DBMS soporta una Base de Datos global al sistema. DB2 y Oracle utilizan ambas este planteamiento. Los datos de procesamiento de pedidos, contabilidad y nómina están todos almacenados en tablas dentro de la Base de Datos. Las principales tablas de esta aplicación están reunidas bajo la propiedad de un único usuario, que probablemente es la persona al cargo de esa aplicación en esta computadora.

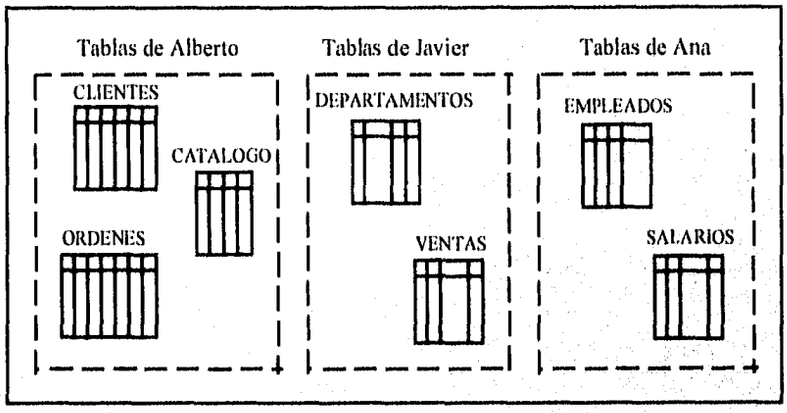


FIG. IV.15 Arquitectura de base de datos única

Una ventaja de esta arquitectura es que las tablas de las diferentes aplicaciones pueden referenciarse fácilmente unas a otras. La tabla PERSONAL de la aplicación de nóminas ejemplo, puede contener una clave foránea que referencie la tabla REPVENTAS, y las aplicaciones pueden utilizar esta relación para calcular comisiones. Con el permiso adecuado, los usuarios pueden ejecutar consultas que combinen datos de las diferentes aplicaciones.

Una desventaja de esta arquitectura es que la Base de Datos crecerá enormemente con el tiempo conforme se le añada más y más complicaciones. Una Base de Datos DB2 u Oracle con varios cientos de tablas no es inhabitual. Los problemas de gestionar una Base de Datos de este tamaño ---- realización de copias de seguridad, recuperación de datos, análisis de rendimiento, etc.---- requieren generalmente un administrador de Base de Datos a tiempo completo.

En la arquitectura de Base de Datos única, obtener acceso a la Base de Datos es muy sencillo --- sólo hay una Base de Datos, por lo que no hay que efectuar decisiones---. Por ejemplo, la sentencia de SQL programado que conecta a un usuario con una Base de Datos Oracle es CONNECT, y los usuarios hablan en términos de ≺conectarse a Oracle≻, en vez de conectarse a una Base de Datos específica.

De hecho las instalaciones Oracle y DB2 ejecutan frecuentemente dos Bases de Datos separadas, una para trabajo de producción y otra para comprobación. Fundamentalmente, sin embargo, todos los datos de producción están recogidos en una única Base de Datos.

4.4.16 Arquitectura de Bases de Datos Múltiples

La (Fig. IV.16) muestra una arquitectura de Base de Datos múltiples en donde cada Base de Datos tiene asignado un nombre único. Ingres y Sybase utilizan todo este esquema. Como se muestra en la figura cada una de las

Bases de Datos de esta arquitectura está dedicada generalmente a una aplicación particular. Cuando se añade una nueva aplicación, lo más normal es crear una nueva Base de Datos.

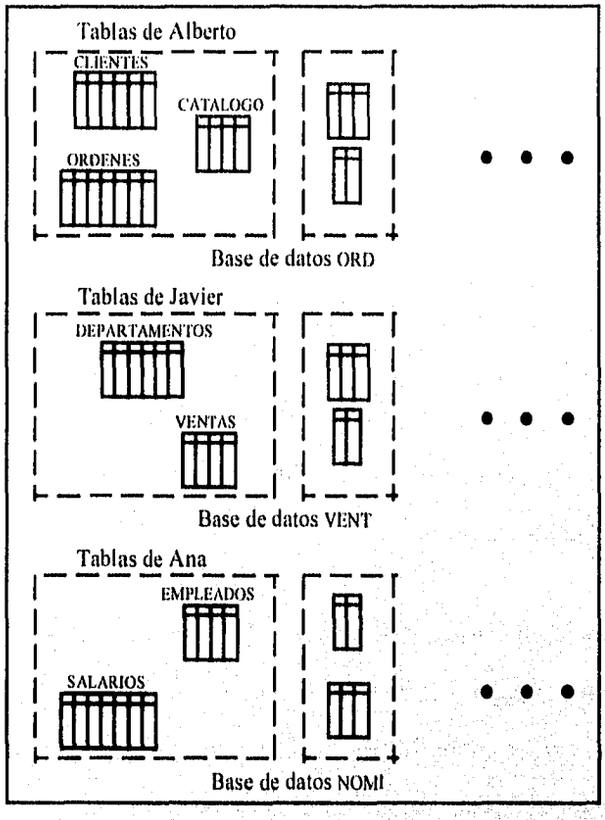


FIG. IV.16 Arquitectura de bases de datos múltiples

La ventaja principal de la arquitectura de Base de Datos múltiples con respecto a la arquitectura de Base de Datos única es que divide a las tareas de gestión de Base de Datos en partes más pequeñas y más manejables. Cada persona responsable de una aplicación puede ahora ser el administrador de su

propia Base de Datos, con menos preocupaciones con respecto a la coordinación global. Cuando llega el momento de añadir una nueva aplicación, puede desarrollarse en su propia Base de Datos sin afectar a la Bases de Datos existentes. También es más probable que los usuarios y los programadores puedan recordar la estructura general de sus propias Bases de Datos.

La desventaja principal de esta arquitectura es que las Bases de Datos individuales pueden convertirse en ≺islas≻ de información, desconectadas unas de otras. Típicamente una tabla de una Base de Datos no puede contener una referencia de clave foránea a una tabla en una Base de Datos diferente. Con frecuencia el DBMS no soporta consultas a través de fronteras de Bases de Datos, haciendo imposible relacionar datos de dos aplicaciones. Si las consultas entre Bases de Datos son soportadas, pueden imponer un recargo sustancial o exigir la compra al vendedor del DBMS de un software de DBMS distribuido adicional.

Si un DBMS utiliza una arquitectura de Bases de Datos múltiples y soporta consultas entre Bases de Datos, debe ampliar las convenciones de designación de tablas y columnas SQL. Un nombre de tabla cualificado debe especificar no solamente el propietario de la tabla, sino también qué Base de Datos contiene la tabla. Típicamente el DBMS extiende la ≺notación punto≻ para nombres de tabla añadiendo como prefijo del nombre de la Base de Datos el nombre del propietario, separado por un punto (.). Por ejemplo, en

una Base de Datos Sybase, esta referencia de tabla: PP.ALBERTO.OFICINAS se refiere a la tabla OFICINAS propiedad del usuario ALBERTO en la Base de Datos de procesamiento de pedidos de nombre PP, y esta otra consulta compone la tabla REPVENTAS en la Base de Datos de nómina con la tabla OFICINA:

```
SELECT PP.ALBERTO.OFICINAS.CIUDAD,NOMINA.ANA.REPVENTAS.NOMBRE
FROM   PP.ALBERTO.OFICINAS, LISTA.ANA.REPVENTAS
WHERE  PP.ALBERTO.DIR = LISTA.ANA.REPVENTAS.NUM_EMPL
```

Afortunadamente, tales consultas entre Bases de Datos son la excepción en vez de la regla, y normalmente pueden utilizarse nombres de usuario de Base de Datos por omisión.

Con esta arquitectura, obtener acceso a una Base de Datos resulta ligeramente más complejo, ya que debe informarse al DBMS de qué Base de Datos se debe utilizar. El programa SQL interactivo de DBMS visualizará con frecuencia una lista de las Bases de Datos disponible o pedirá que se introduzca el nombre de la Base de Datos junto con el nombre del usuario y su contraseña para permitirle el acceso. Para el acceso en programación, el DBMS extiende generalmente el lenguaje SQL incorporado con una sentencia que conecta el programa a una Base de Datos particular. La forma Ingres para conectarse a la Base de Datos de nombre PP es: CONNECT 'PP' y para Sybase es: USE DATABASE 'PP'.

4.4.17 Arquitectura de Ubicación Múltiple

La (Fig. IV.17) muestra una arquitectura de ubicación múltiple que soporta múltiples Bases de Datos y utiliza la estructura de directorio del sistema informático para organizarlas. Informix utiliza este esquema para soportar múltiples Bases de Datos. Como con la arquitectura de Base de Datos múltiples, cada aplicación tiene asignada típicamente su propia Base de Datos. Como muestra la figura, cada Base de Datos tiene un nombre, pero es posible que dos Bases de Datos diferentes en dos directorios diferentes tengan el mismo nombre.

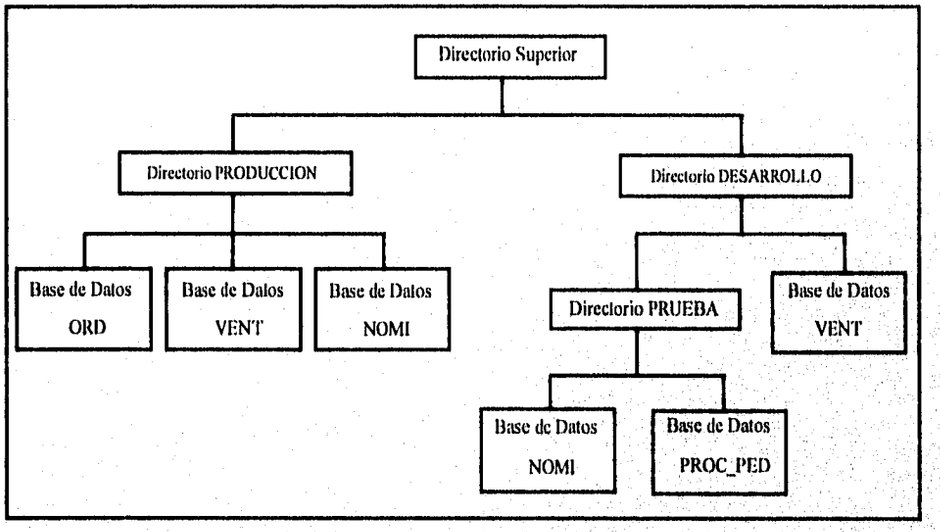


FIG. IV.17 Arquitectura de Ubicación Múltiple

La principal ventaja de esta arquitectura es la flexibilidad. Es especialmente adecuada en aplicaciones tales como la ingeniería y diseño, en

donde hay muchos usuarios sofisticados del sistema informático, que pueden todos desear utilizar varias Bases de Datos para estructurar su propia información. Las desventajas de esta arquitectura son las mismas que las de la arquitectura de Bases de Datos múltiples. Además el DBMS no suele conocer todas las Bases de Datos que han sido creadas, las cuales pueden extenderse a través de toda la estructura de directorios del sistema. No existe una ⋖Base de Datos maestra⋗ que lleve la cuenta de todas las Bases de Datos, y esto hace que la administración centralizada sea muy difícil.

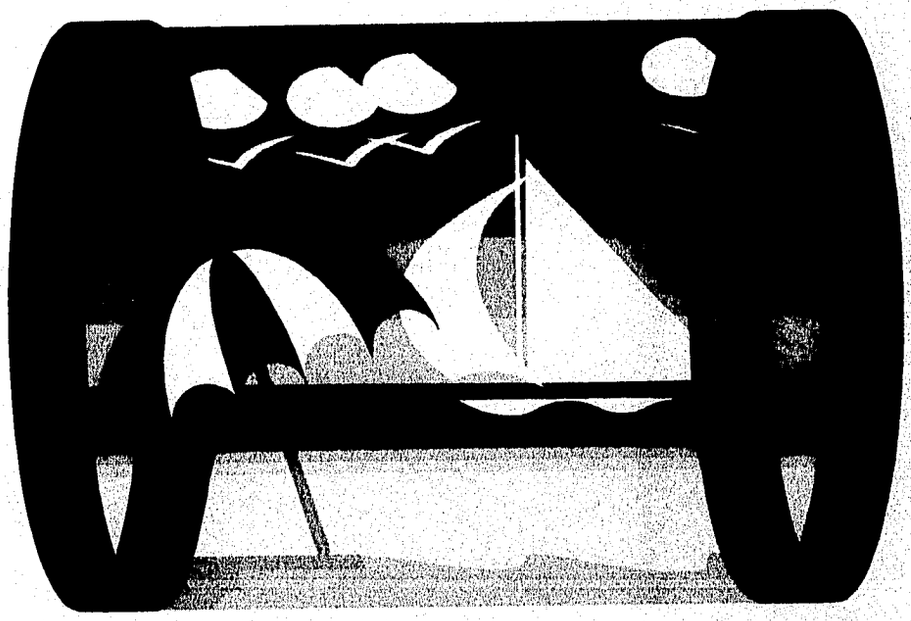
CAPITULO V

EJERCICIO TEORICO - PRACTICO

PEMEX

BASE DE DATOS DE LAS VACACIONES DE LA SUBDIRECCION DE SERVICIOS TECNICOS

PEMEX EXPLORACION Y PRODUCCION



De todos los talentos, el de hacer las cosas oportunamente es quizá el más útil y menos común

(Joseph Jouy)

Nadie debe avergonzarse de preguntar lo que no sabe.

(Prov. Oriental)

5.1 INTRODUCCIÓN

En este último capítulo pasaremos de lo teórico a lo práctico elaborando un ejercicio real; es decir retomaremos toda la parte teórica de los cuatro capítulos anteriores para ponerlos en práctica en un caso específico, del cual se hará el análisis de una Base de Datos requerida para el control de los ciclos vacacionales de la Subdirección de Servicios Técnicos-Pemex Exploración y Producción.

En primer lugar se verán las necesidades, los requerimientos y especificaciones del problema real, con lo cual nos apoyaremos en la teoría del capítulo uno. En segundo lugar haremos el modelo conceptual de datos del problema en cuestión retomando la teoría de modelo Entidad-Relación del capitulo dos. En tercer lugar haremos el diseño de la Base de Datos a partir del modelo conceptual apoyándonos en el capítulo tres. Y por último utilizaremos el cuarto capítulo para la construcción de la Base de Datos. Después de haber seguido rigurosamente los pasos anteriores podremos obtener la Base de Datos de vacaciones de la Subdirección de Servicios Técnicos-Pemex Exploración y Producción. Con lo cual se pretende tener un mayor control sobre los ciclos de vacaciones del personal y así poder ser mas eficiente en la toma de decisiones por parte de la Unidad de Apoyo Administractivo-de la Subdirección de Servicios Técnicos.

En resumen este capítulo trata de ir viendo paso por paso desde los requerimientos, hasta llegar a la construcción de la Base de Datos pasando por el modelo conceptual de Datos y el Diseño de la Base de Datos para ir comprendiendo como es que se puede llegar a un buen Diseño de Datos lo cual nos debe de dar finalmente una Base de Datos bien estructurada y que posteriormente ésta a su vez pasará a formar parte de un Sistema de Información confiable, eficiente, e integral.

5.2 MARCO HISTÓRICO

El 16 de julio de 1992, por medio de un Decreto Presidencial, se establecieron las bases de una nueva estructura administrativa de Petróleos Mexicanos, creando cuatro organismos descentralizados con personalidad jurídica y patrimonio propios:

- Pemex Exploración y Producción
- Pemex Refinación
- Pemex Gas y Petroquímica Básica.
- Pemex Petroquímica

Cada organismo descentralizado tiene un Consejo de Administración presidido por el Director General de Petróleos Mexicanos, e integrado por los directores generales de los otros organismos, cuatro representantes gubernamentales y un comisario. La función del Consejo de Administración

es la de supervisar a los organismos, con el fin de asegurar su orientación económica, exigir responsabilidad por los resultados obtenidos y vigilar el cumplimiento de normas de seguridad y ambientales.

Este nuevo modelo de estructura se implanta con el propósito de que Pemex compita con las grandes empresas internacionales, como lo exigen las condiciones de un nuevo mercado mundial caracterizado por su globalización, además de ofrecer productos y servicios en el mercado interno, a precios y de una calidad competitiva con los ofrecidos por otros países. ver anexo "G"

Pemex-Exploración y Producción esta formada de la siguiente manera: Una dirección General, y las siguientes Subdirecciones que son: Exploración, Producción, Perforación, Administración y Finanzas, Servicios Técnicos, Planificación, las siguientes regiones: Región-Marina, Región-Sur, Región-Norte, Seguridad y Protección Ambiental y una Contraloría Interna. ver anexo "G"

El caso que nos compete será con referente a la Subdirección de Servicios Técnicos que esta formada de la siguiente manera: Por una Unidad de Apoyo Administrativo, y las Gerencias de: Inspección y Mantenimiento, Ingeniería en Construcción, Servicios a pozos e Instalaciones, Desarrollo Tecnológico y Por sus tres gerencias regionales región Marina, Región Sur y Región Norte. ver anexo "G"

En la actualidad cada área de la Subdirección de Servicios Técnicos lleva su control de los ciclos vacacionales de su personal, encargándose únicamente de enviar su programa anual de vacaciones a la Unidad de Apoyo Administrativo. Esta a su vez concentra en uno solo los diferentes programas de vacaciones y los tramita como el programa anual de vacaciones de la Subdirección de Servicios Técnicos ante la Gerencia de Recursos Humanos regresando a su vez el programa ya registrado y en el transcurso del año envía las boletas de vacaciones a la Unidad de Apoyo Administrativo una semana antes de que salga cada trabajador. el trabajador pasa por la copia de su boleta a la unidad con lo cual se le confirma su período de vacaciones y puede pasar a cobrar sus vacaciones a la caja. Archivando su boleta de vacaciones en el expediente del trabajador en la Unidad de Apoyo Administrativo. Con ésto se finaliza el trámite de vacaciones. Siempre y cuando no exista alguna reprogramación de las vacaciones del trabajador requerida por el área; no dejando de cobrar por esto sus vacaciones en la fecha programada, quedando pendiente sus vacaciones en tiempo hasta nuevo aviso por parte de su área. Otras de las posibles causas para interrumpir el trámite de vacaciones es la cancelación de éstas requeridas por su área con lo cual queda pendiente parcialmente sus vacaciones en tiempo hasta nuevo aviso por parte de su área.

5.3 NECESIDADES

Es necesario tener un control adecuado de los ciclos vacacionales del personal de confianza y Sindicalizado de la Subdirección de Servicios

técnicos en sus diferentes áreas en sede (Oficinas de la Subdirección, Unidad de Apoyo Administrativo, Gerencia de Inspección y Mantenimiento, Gerencia de Servicios a pozos, Gerencia de Ingeniería y Construcción y Gerencia de Desarrollo Tecnológico), para una mejor administración y para que las funciones de las áreas no se vean interrumpidas por la disminución temporal de su personal en sus períodos vacacionales y tener a su vez un control de los períodos que se le deben a los trabajadores para tratar de que se regularice su situación vacacional de éstos y no vengan arrastrando muchos períodos rezagados. Por lo cual es necesario llevar un registro histórico de los ciclos vacacionales reprogramados y cancelados para que después puedan ser disfrutados por los trabajadores sin tener ningún problema por no estar registrados sus ciclos pendientes.

5.4 REQUERIMIENTOS

Para poder cumplir las necesidades mencionadas con anterioridad de la Subdirección de Servicios Técnicos es necesario diseñar una Base de Datos que satisfaga los requerimientos de información para llevar un mejor control de los ciclos vacacionales del personal de confianza y Sindicalizado adscrito a esta Subdirección y así poder tener una mejor administración de éstos. Por lo cual es necesario que la Base de Datos contenga toda la información que tienen las boletas de vacaciones (Ficha del Trabajador, Fecha del programa de Vacaciones, No. de Boleta de vacaciones, Fecha de Salida, Días de Incentivos, Económicos, Observaciones) y Datos Personales del Trabajador

(Ficha, Nombre, Nivel, Departamento, Fecha del Ciclo de Vacaciones y Antigüedad). La periodicidad con la que se requiere manejar esta información es semanalmente de acuerdo a como son enviadas las boletas de vacaciones de los trabajadores por parte de la Gerencia de Recursos Humanos y Capturándose solamente una vez el Programa Anual de Vacaciones. Por lo cual es necesario tener actualizado la información conforme se va recibiendo ésta, a su vez también se requiere sacar un reporte anual del programa de vacaciones, para entregarlo para la anuencia de la Gerencia de Recursos Humanos; sacar reportes mensuales del personal que va a salir en el mes en curso y se requiere sacar reportes del personal que esta pendiente de disfrutar su ciclo vacacional.

Es necesario tomar en cuenta las cláusulas del contrato colectivo de trabajo celebrado entre Petróleos Mexicanos y el Sindicato de Trabajadores (1995-1997) que hacen referencia a las vacaciones para poder hacer algunas validaciones de la información. ver anexo "F"

Después de haber analizado todos los requerimientos anteriores los siguientes aspectos, a nuestro juicio consideramos que pueden ser los más importantes acerca de las necesidades para construir el sistema de Bases de Datos Relacional para nuestro control de vacaciones : EMPLEADO, CICLO VACACIONAL, CICLO VACACIONAL CANCELADO, CICLO VACACIONAL SUSPENDIDO, CICLO VACACIONAL PROGRAMADO, CICLO VACACIONAL

LIQUIDADO, DIAS ECONOMICOS, CENTRO DE TRABAJO, CLASIFICACION DEL EMPLEADO, DEPARTAMENTO.

5.5 DEFINICIÓN DE LAS ENTIDADES DE LA BASE DE DATOS DE VACACIONES

Así podemos formar nuestras Entidades con sus respectivos Atributos :

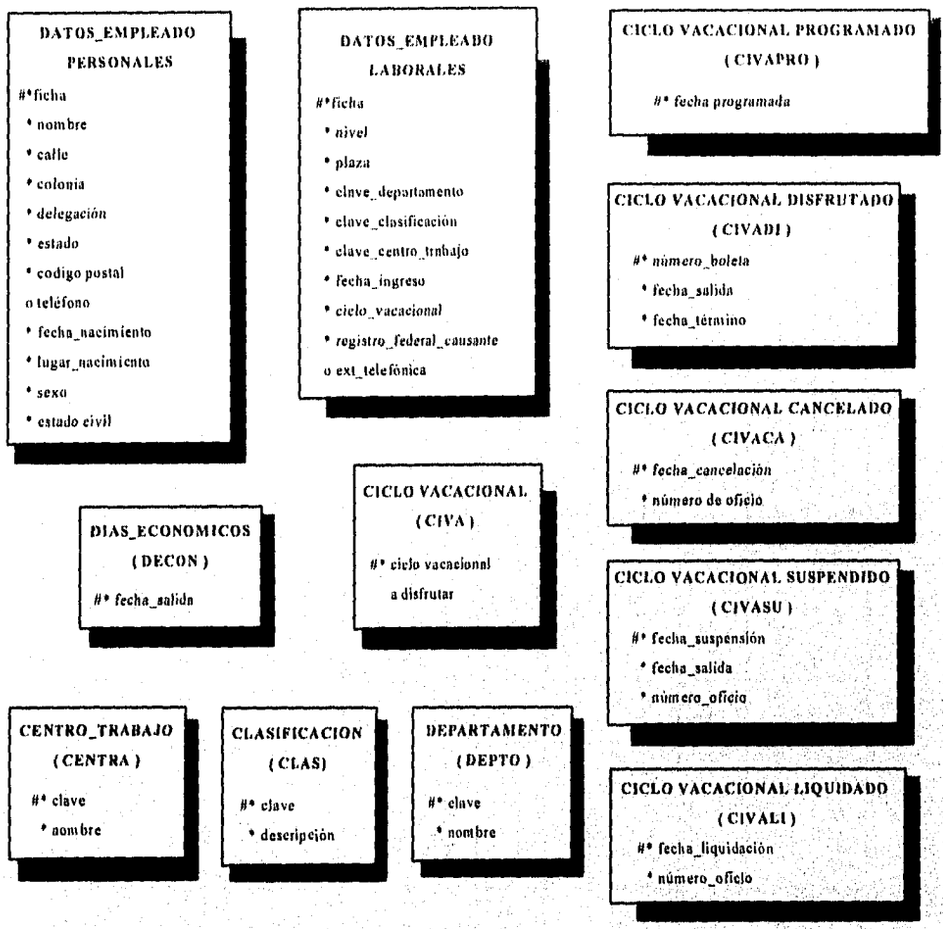
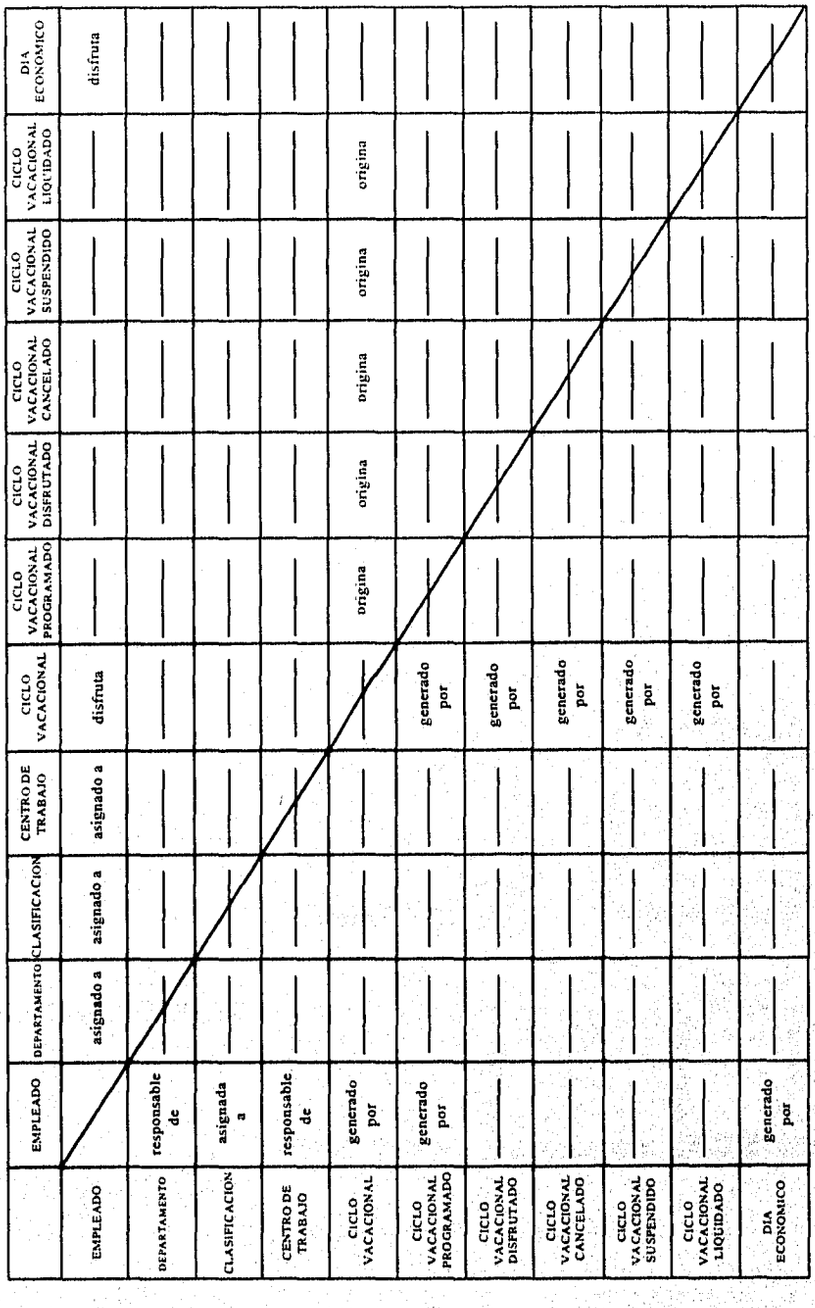


FIG. V.1 Entidades de la Base de Datos de Vacaciones

5.6 MATRIZ DE RELACIONES PARA LA COLECCION INICIAL DE INFORMACION



	EMPLEADO	DEPARTAMENTO	CLASIFICACION	CENTRO DE TRABAJO	CICLO VACACIONAL	CICLO VACACIONAL PROGRAMADO	CICLO VACACIONAL DISFRUTADO	CICLO VACACIONAL CANCELADO	CICLO VACACIONAL SUSPENDIDO	CICLO VACACIONAL LIQUIDADO	DIA ECONOMICO
EMPLEADO		responsable de	asignada a	responsable de	generado por	generado por	———	———	———	———	———
DEPARTAMENTO	asignado a		———	———	———	———	———	———	———	———	———
CLASIFICACION	asignado a	———		———	———	———	———	———	———	———	———
CENTRO DE TRABAJO	asignado a	———	———		———	———	———	———	———	———	———
CICLO VACACIONAL	disfruta	———	———	———		origina	origina	origina	origina	origina	———
CICLO VACACIONAL PROGRAMADO	———	———	———	———	origina		———	———	———	———	———
CICLO VACACIONAL DISFRUTADO	———	———	———	———	generado por	generado por		———	———	———	———
CICLO VACACIONAL CANCELADO	———	———	———	———	generado por	generado por	generado por		———	———	———
CICLO VACACIONAL SUSPENDIDO	———	———	———	———	generado por	generado por	generado por	generado por		———	———
CICLO VACACIONAL LIQUIDADO	———	———	———	———	generado por	———	———	———	———		———
DIA ECONOMICO	disfruta	———	———	———	———	———	———	———	———	———	

FIG. V.2 Matriz de Relaciones de la Base de Datos de Vacaciones

5.7 RELACIONES ENTRE LAS ENTIDADES DE LA BASE DE DATOS DE VACACIONES

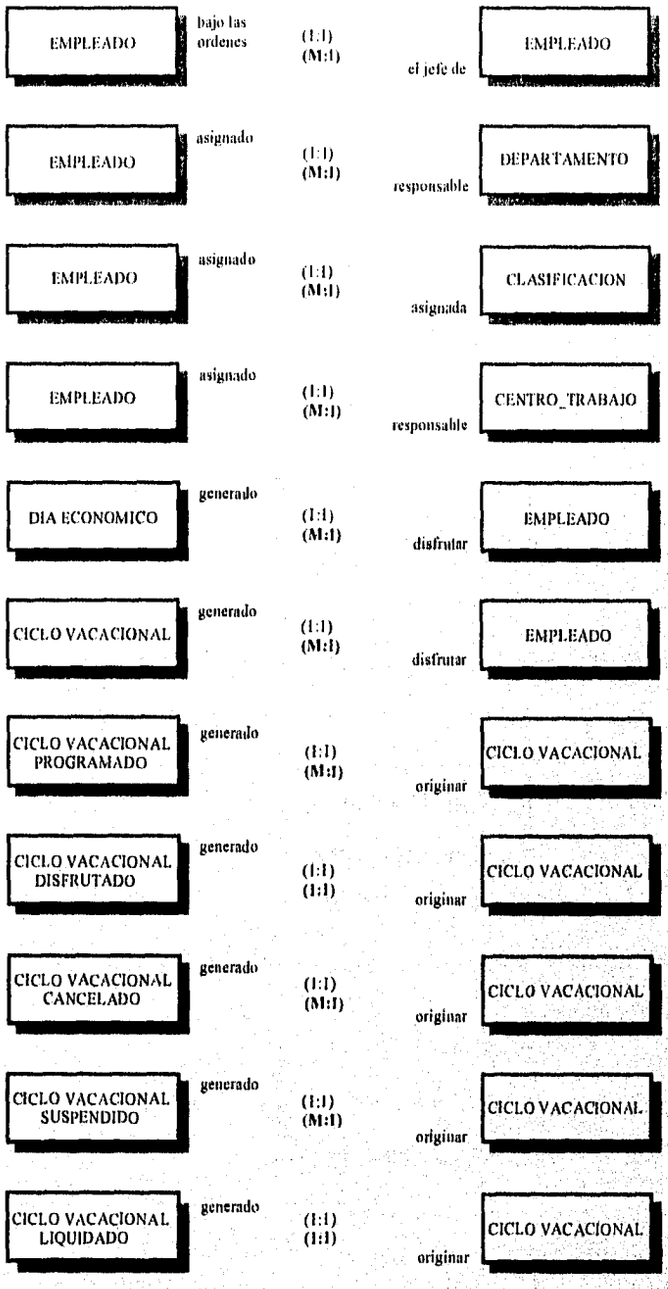


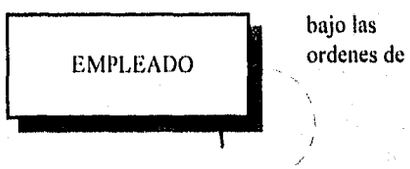
FIG. V.3 Relaciones entre las Entedidades de la Base de Datos deVacaciones

5.8 DEFINICIÓN DE LA OPCIONALIDAD Y EL GRADO DE LAS RELACIONES

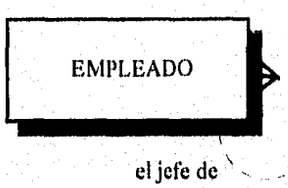
El paso siguiente es determinar la opcionalidad y el grado de cada dirección de cada una de las relaciones.

Relación 1

EMPLEADO y EMPLEADO (Relación Recursiva)



Cada EMPLEADO puede estar bajo las ordenes de otro empleado



Cada EMPLEADO puede ser el jefe de uno o más empleados

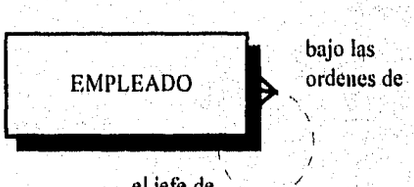


FIG. V.4 Relación entre Empleado y Empleado (Relación Recursiva)

Relación 2

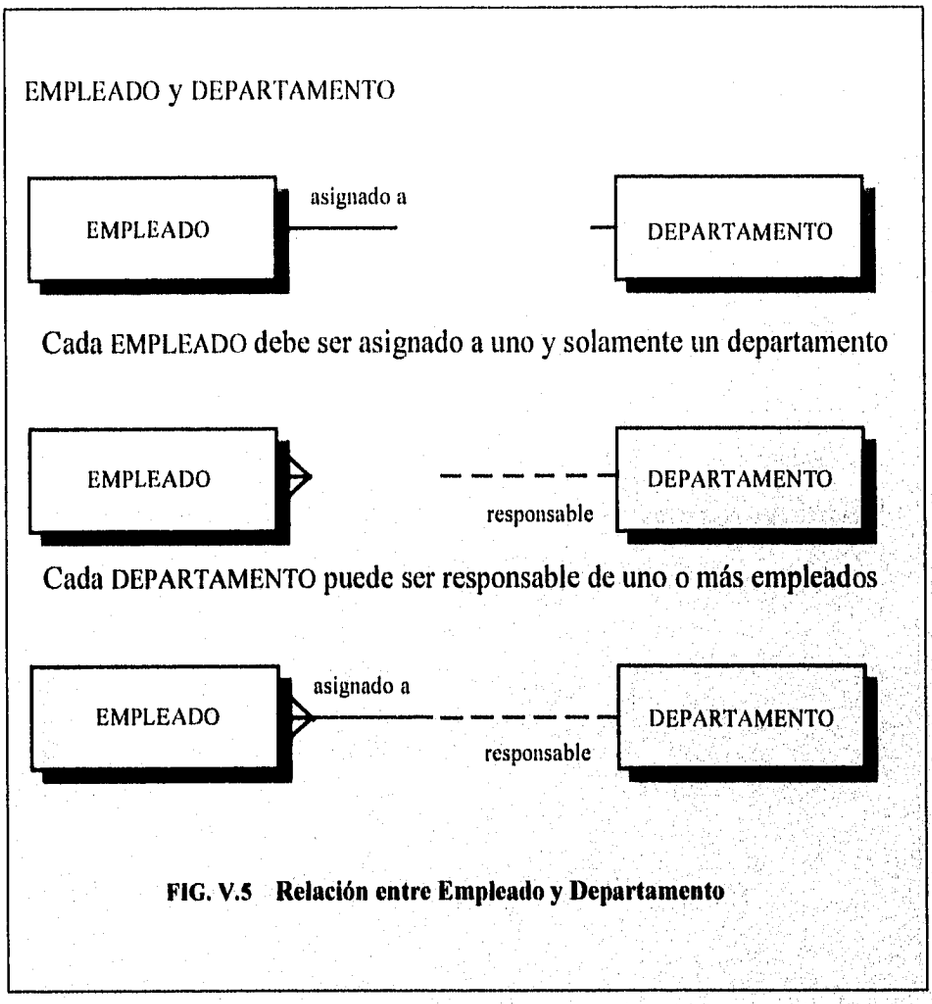


FIG. V.5 Relación entre Empleado y Departamento

Relación 3

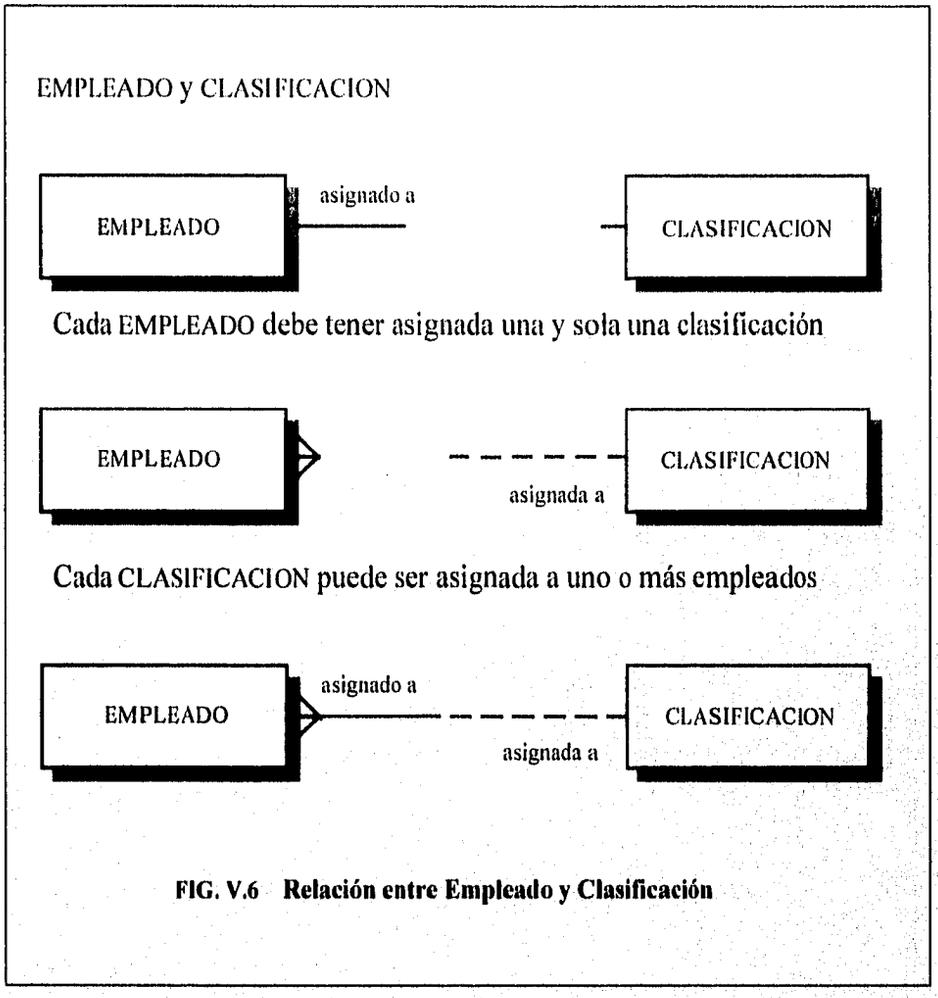
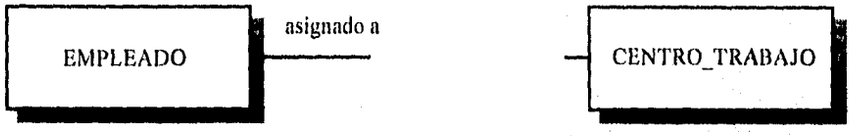


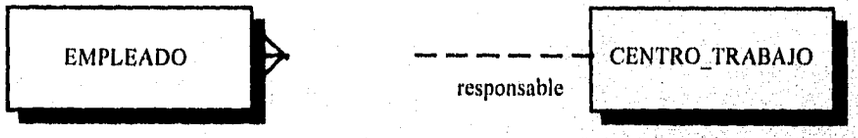
FIG. V.6 Relación entre Empleado y Clasificación

Relación 4

EMPLEADO y CENTRO DE TRABAJO



Cada EMPLEADO debe ser asignado a uno y solamente un centro de trabajo



Cada CENTRO DE TRABAJO puede ser responsable uno o más empleados

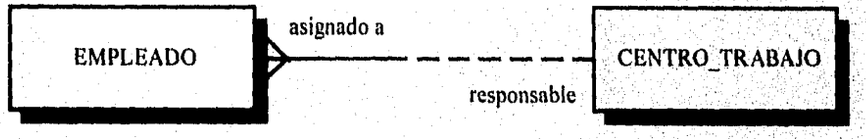
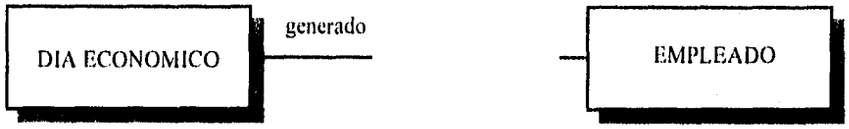


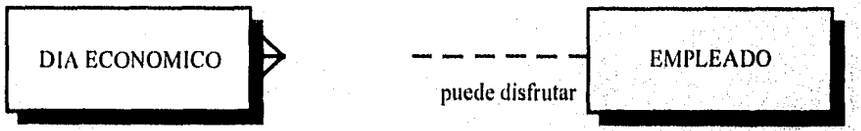
FIG. V.7 Relación entre Empleado y Centro de Trabajo

Relación 5

DIA ECONOMICO y EMPLEADO



Cada DIA ECONOMICO debe ser generado por uno y solamente un empleado



Cada EMPLEADO puede disfrutar de uno o más días económicos

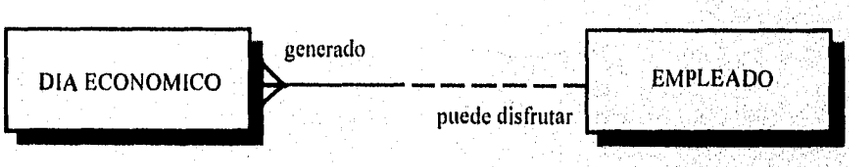
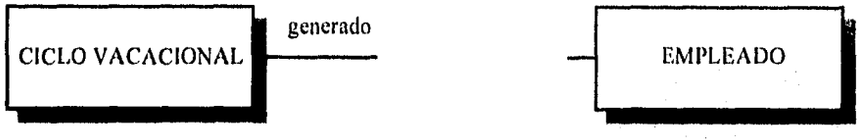


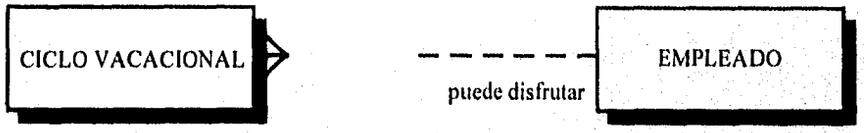
FIG. V.8 Relación entre Día Económico y Empleado

Relación 6

CICLO VACACIONAL y EMPLEADO



Cada CICLO VACACIONAL debe ser generado por uno y solamente un empleado



Cada EMPLEADO puede disfrutar de uno o más ciclos vacacionales en un tiempo

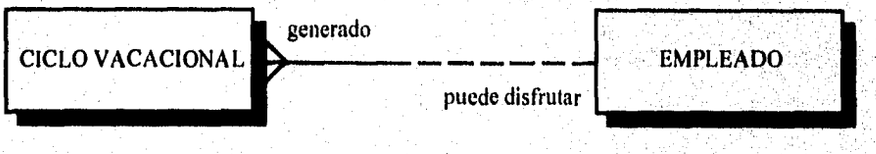


FIG. V.9 Relación entre Ciclo Vacacional y Empleado

Relación 7

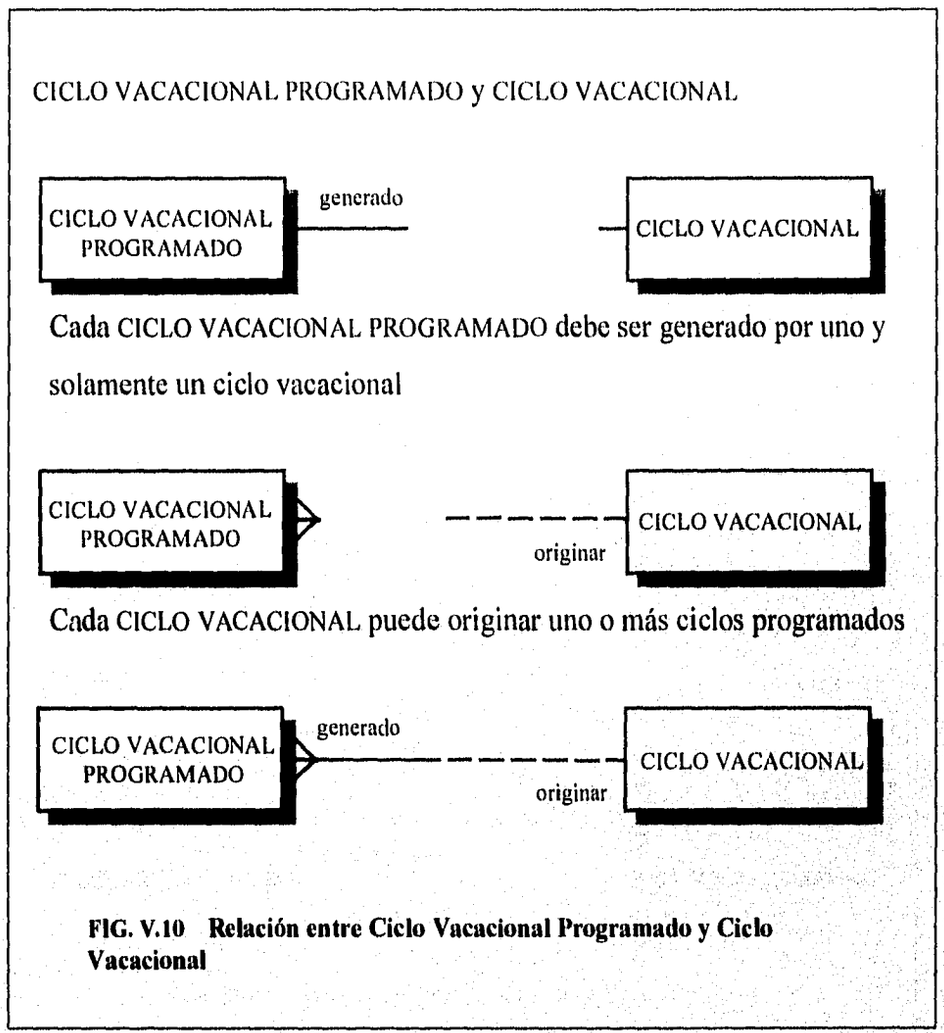
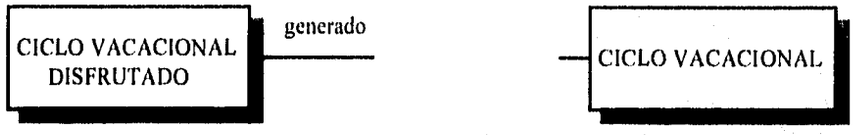


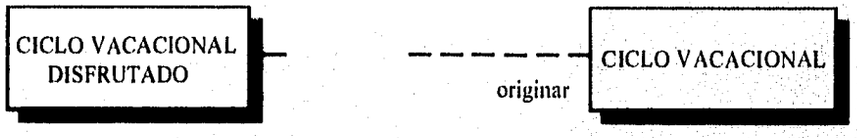
FIG. V.10 Relación entre Ciclo Vacacional Programado y Ciclo Vacacional

Relación 8

CICLO VACACIONAL DISFRUTADO y CICLO VACACIONAL



Cada CICLO VACACIONAL DISFRUTADO debe ser generado por uno y solamente un ciclo vacacional



Cada CICLO VACACIONAL puede originar uno y solamente un ciclo vacacional disfrutado

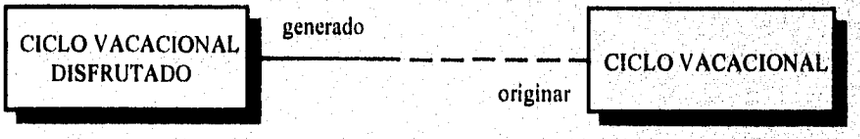
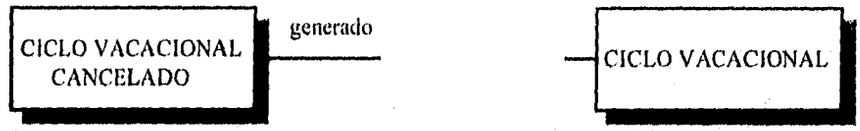


FIG. V.11 Relación entre Ciclo Vacacional Disfrutado y Ciclo Vacacional

Relación 9

CICLO VACACIONAL CANCELADO y CICLO VACACIONAL



Cada CICLO VACACIONAL CANCELADO debe ser generado por uno y solamente un ciclo vacacional

Cada CICLO VACACIONAL puede originar uno o más ciclos cancelados

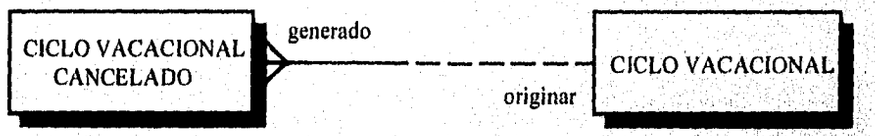


FIG. V.12 Relación entre Ciclo Vacacional Cancelado y Ciclo Vacacional

Relación 10

CICLO VACACIONAL SUSPENDIDO y CICLO VACACIONAL

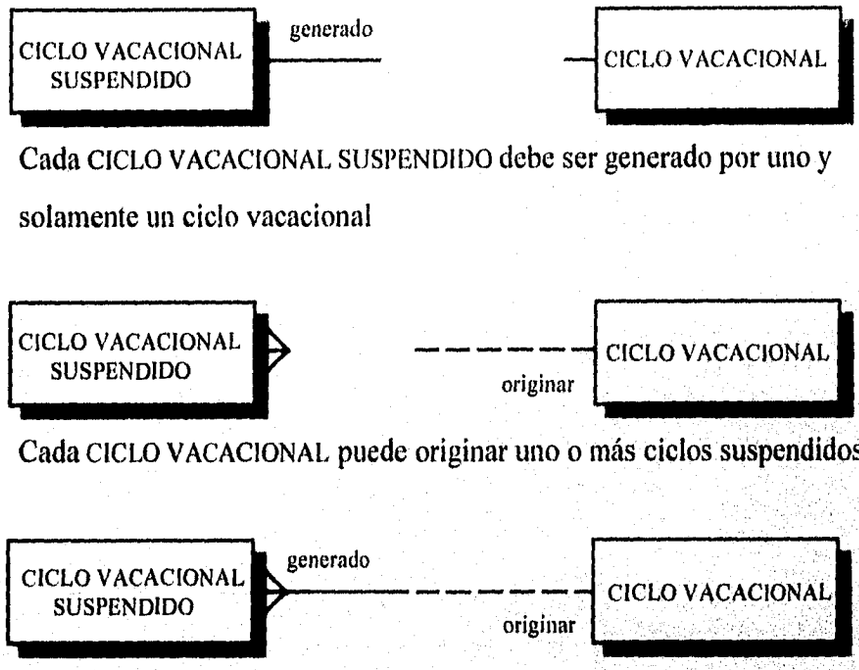
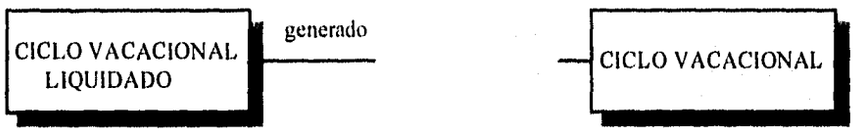


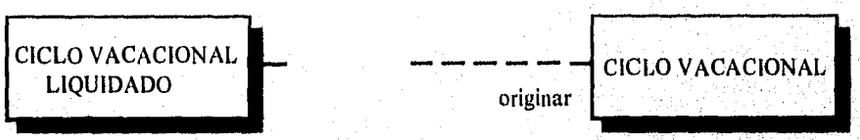
FIG. V.13 Relación entre Ciclo Vacacional Suspendido y Ciclo Vacacional

Relación 11

CICLO VACACIONAL LIQUIDADO y CICLO VACACIONAL



Cada CICLO VACACIONAL LIQUIDADO debe ser generado por uno y solamente un ciclo vacacional



Cada CICLO VACACIONAL puede originar uno y solamente un ciclo liquidado

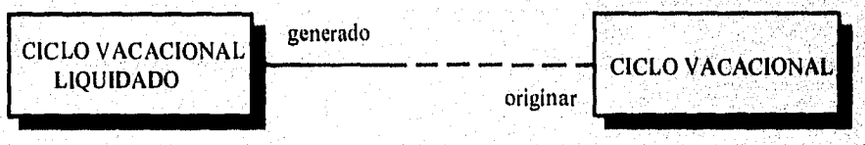


FIG. V.14 Relación entre Ciclo Vacacional Liquidado y Ciclo Vacacional

5.9 DIAGRAMAS ENTIDAD-RELACIÓN GENERAL DE LA BD DE VACACIONES

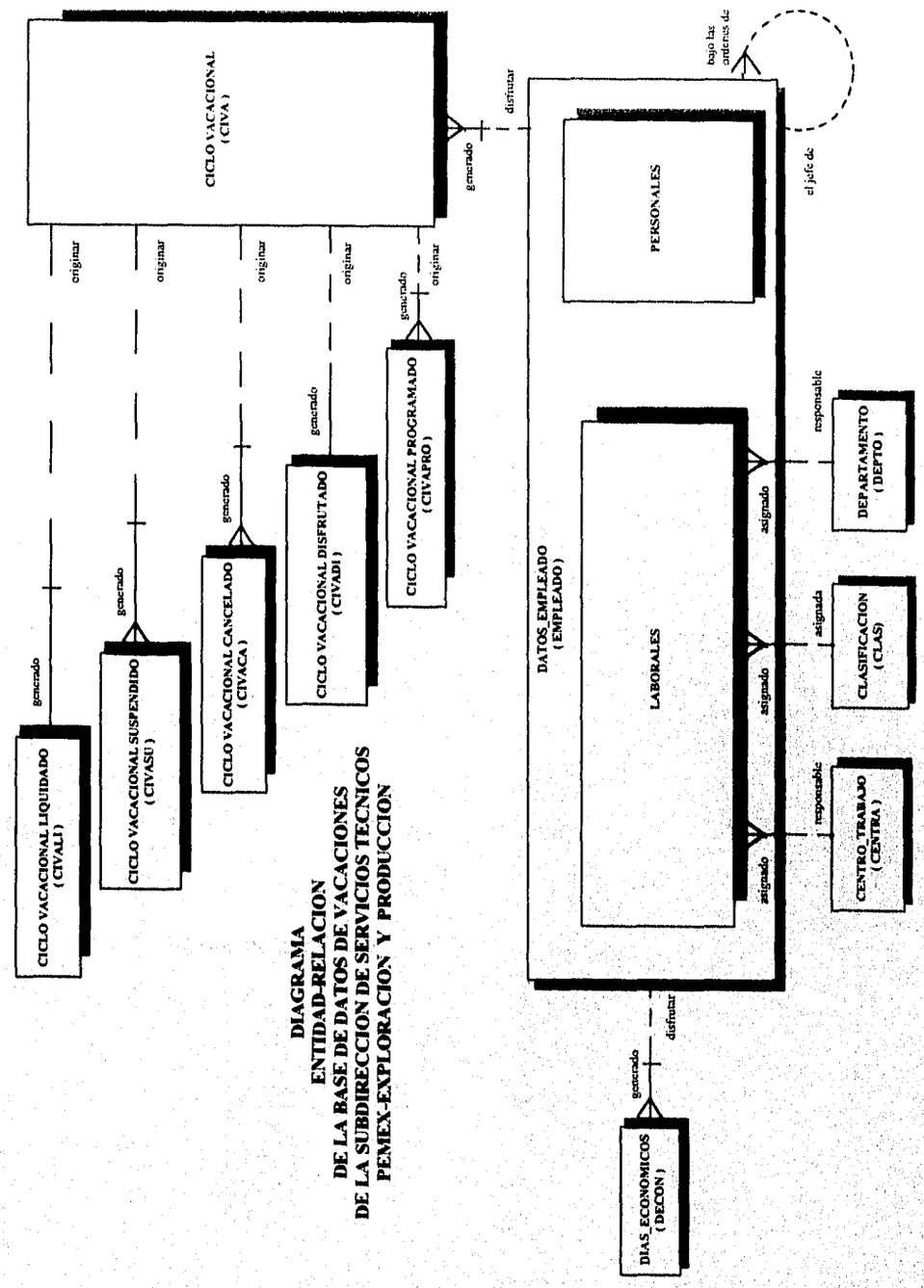


FIG. V.15 Diagrama E-R General del Sistema de Vacaciones

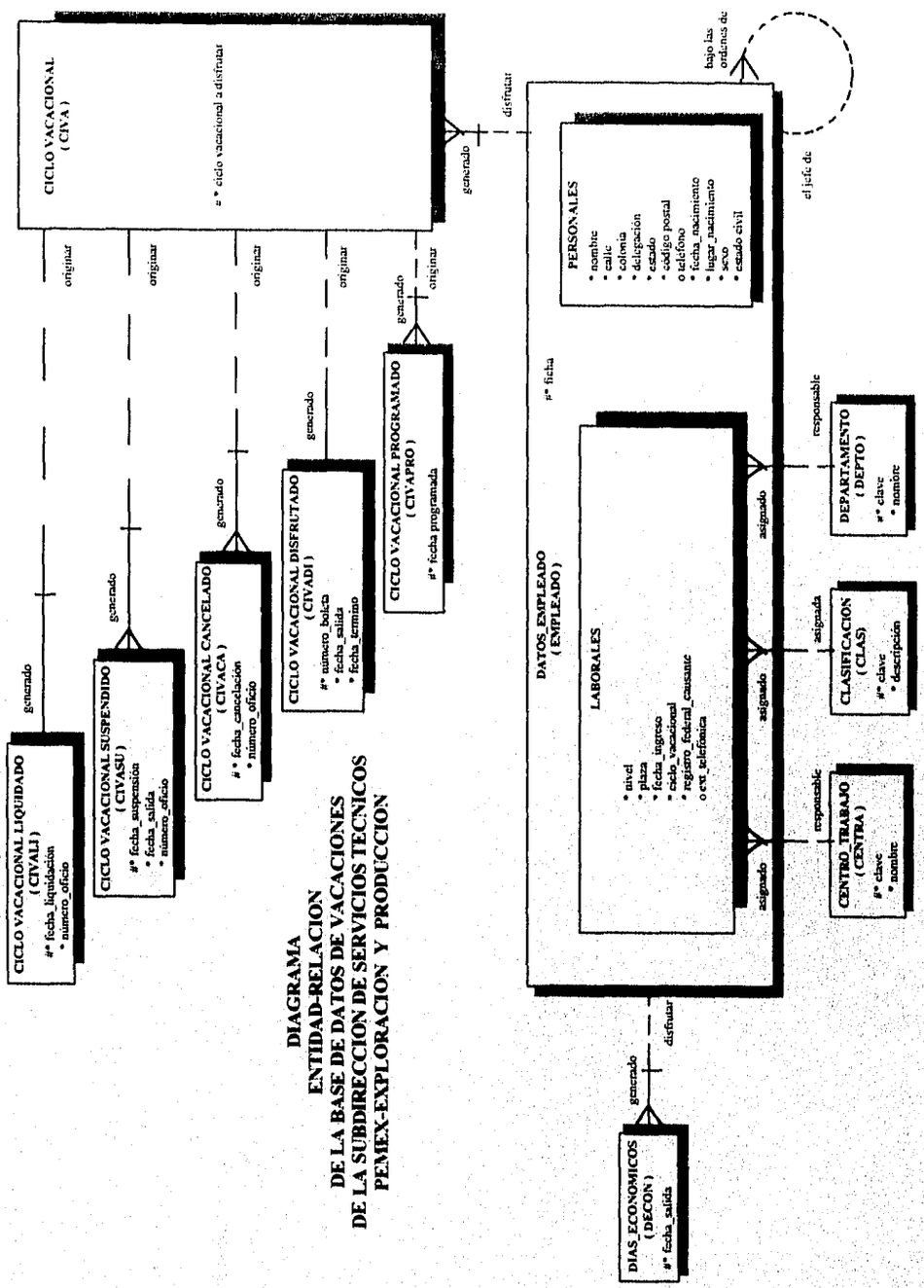


FIG. V.16 Diagrama E-R con sus Respectivos Atributos General del Sistema de Vacaciones

5.10 DEFINICIÓN DEL MAPA DE INSTANCIAS PARA CADA UNA DE LAS ENTIDADES

El paso siguiente es determinar el Mapa de Instancia de cada una de las entidades de la Fig. (V.16) Con la finalidad de ver los datos que contendrán cada una de las entidades.

Mapa de Instancias 1

NOMBRE DEL ATRIBUTO	CLAVE	NOMBRE
ETIQUETA	*	*
EJEMPLO DE DATOS	28000	SUBDIRECCION DE SERVICIOS TECNICOS
	28400	G. INGENIERIA Y CONSTRUCCION
	28600	G. SERVICIOS A POZOS
	28700	G. DESARROLLO TECNOLOGICO
	28500	G. INSPECCION Y MANTENIMIENTO

FIG. V.17 Mapa de Instancias de la Entidad DEPARTAMENTO

Mapa de Instancias 2

NOMBRE DEL ATRIBUTO	CLAVE	DESCRIPCION
ETIQUETA	*	*
EJEMPLO DE DATOS	118502	CHOFER
	044406	MENSAJERO
	084406	OFICINISTA DE SEXTA
	244413	SECRETARIA "A"
	252908	PROGRAMADOR "A"

FIG. V.18 Mapa de Instancias de la Entidad CLASIFICACION

Mapa de Instancias 3

NOMBRE DEL ATRIBUTO	CLAVE	NOMBRE
ETIQUETA	*	*
EJEMPLO DE DATOS	200	VILLAHERMOSA
	800	MEXICO
	300	ALTAMIRA
	100	C. CARMEN
	400	VERACRUZ

FIG. V.19 Mapa de Instancias de la Entidad CENTRO DE TRABAJO

Mapa de Instancias 4

NOMBRE DEL ATRIBUTO	FICHA	NOMBRE	CALLE	COLONIA	DELEGACION	ESTADO	C.P.	TEL	FECHA_NACIMIENTO	LUGAR_NACIMIENTO	ESTADO_CIVIL	SEXO
ETIQUETA	*	*	*	*	*	*	*	0	*	*	*	*
EJEMPLO DE DATOS	125690	ALMA LOPEZ M.	ROSA	FLORES	JUAREZ	DF.	5080	5-30-12-60	16-OCT-63	MEXICO D.F.	SOLTERO	F
	204069	MARY RITO LOPEZ	12	SUR 2	IZTACALCO	DF	5020	5-08-05-09	02-OCT-60	VERACRUZ	SOLTERO	F
	132020	ANGELES ERAZO H	SALTO	CENTRO	CUAUHTEMOC	DF.	4900	5-06-30-45	09-ENE-50	MEXICO D.F.	CASADO	F
	941090	ALEJANDRA BRITO	AGUA	CENTRO	CUAUHTEMOC	DF	4700	5-23-02-92	15-MAY-80	MEXICO D.F.	SOLTERO	F
	101204	PEDRO SOTO G.	EVA	MARES	ECATEPEC	MEXICO	5800	7-12-09-77	30-DIC-77	EDO. DE MEX.	CASADO	M

FIG. V.20 Mapa De Instancias De La Entidad PERSONALES

Mapa de Instancias 5

NOMBRE DEL ATRIBUTO	FICHA	NIVEL	PLAZA	FECHA_ INGRESO	CICLO_ VACACIONAL	REGISTRO_ FEDERAL_CAUSANTE	EXT. TELEFONICA
ETIQUETA	*	*	*	*	*	*	#
EJEMPLO DE DATOS	125690	08	800-28400- -00100	10-12-60	05-FEB	LOMA631016	20660
	204069	12	200-28000- -02103	08-05-9	08-OCT	RILM601002	32060
	132020	24	800-28000- -00109	05-06-90	05-MAR	ERBA500109	32028
	941080	30	800-28500-ME-01000	03-02-0-92	03-ABR	BRLA800515	32051
	101204	34	200-28000-VH-00089	12-09-77	12-SEP	SOGP771230	20474

FIG. V.21 Mapa De Instancias De La Entidad LABORALES

Mapa de Instancias 6

NOMBRE DEL ATRIBUTO	FECHA_ SALIDA
ETIQUETA	*
EJEMPLO DE DATOS	10-05-94
	12-05-94
	12-04-95
	03-10-93
	02-02-96

FIG. V.22 Mapa de Instancias de la Entidad DIAS ECONOMICOS

Mapa de Instancias 7

NOMBRE DEL ATRIBUTO	CICLO VACACIONAL
ETIQUETA	*
EJEMPLO DE DATOS	JUN/96
	AGOSTO/93
	OCTUBRE/95
	JULIO/92
	ENERO/94

FIG. V.23 Mapa de Instancias de la Entidad CICLO VACACIONAL

Mapa de Instancias 8

NOMBRE DEL ATRIBUTO	FECHA_PROGRAMADA
ETIQUETA	*
EJEMPLO DE DATOS	280/UAA/96
	03/GIM/95
	20/GIC/96
	27/SST/95
	08/GDT/96

FIG. V.24 Mapa de Instancias de la Entidad CICLO VACACIONAL PROGRAMADO

Mapa de Instancias 9

NOMBRE DEL ATRIBUTO	NUMERO_ BOLETA	FECHA_SALIDA	FECHA_TERMINO
ETIQUETA	*	*	*
EJEMPLO DE DATOS	96101234	23-03-96	23-04-96
	96013338	03-06-95	03-07-95
	95101234	02-06-94	02-07-94
	94108720	10-10-96	10-11-96
	93101204	08-08-96	08-09-96

FIG. V.25 Mapa de Instancias de la Entidad CICLO VACACIONAL DISFRUTADO

Mapa de Instancias 10

NOMBRE DEL ATRIBUTO	FECHA_CANCELACION	NUMERO_OFICIO
ETIQUETA	*	*
EJEMPLO DE DATOS	10-02-96	280/SST/96
	10-12-95	03/SST/95
	12-10-95	280/UAA/96
	29-06-95	27/GIC/94
	05-08-94	280/GIM/95

FIG. V.26 Mapa de Instancias de la Entidad CICLO VACACIONAL CANCELADO

Mapa de Instancias 11

NOMBRE DEL ATRIBUTO	FECHA_ SUSPENCION	FECHA_ SALIDA	NUMERO_OFICIO
ETIQUETA	*	*	*
EJEMPLO DE DATOS	10-02-96	1-02-96	280/GDT/96
	10-12-95	25-11-95	03/GSP/95
	12-10-95	20-09-95	20/UAA/96
	29-06-95	15-06-95	27/GIC/95
	05-08-94	28-07-94	08/GIM/95

FIG. V.27 Mapa de Instancias de la Entidad CICLO VACACIONAL SUSPENDIDO

Mapa de Instancias 12

NOMBRE DEL ATRIBUTO	FECHA_LIQUIDACION	NUMERO_OFICIO
ETIQUETA	*	*
EJEMPLO DE DATOS	10-02-96	280/UAA/96
	10-12-95	03/GIM/95
	12-10-95	20/GIC/96
	29-06-95	27/SST/95
	05-08-94	08/GDT/96

FIG. V.28 Mapa de Instancias de la Entidad CICLO VACACIONAL LIQUIDADO

5.11 MAPEO DE LAS ENTIDADES Y SUS RELACIONES A TABLAS

En el siguiente paso Mapearemos las Entidades y sus relaciones a tablas con la finalidad de obtener todas las tablas del Ejercicio-Propuesto

Mapeo 1

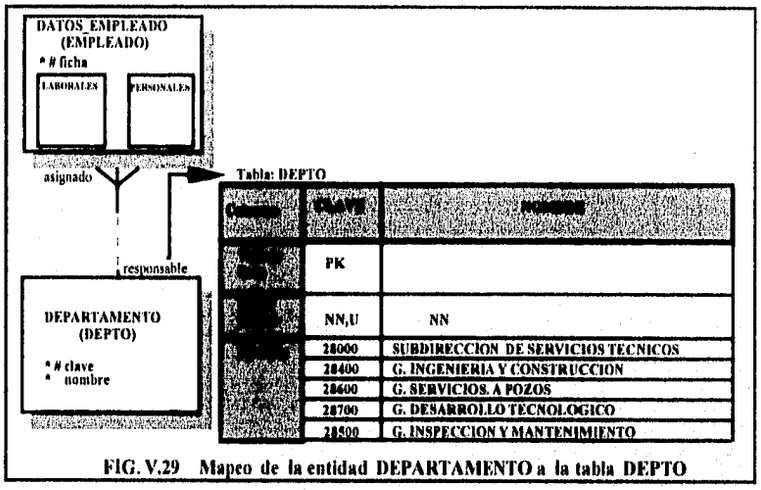


FIG. V.29 Mapeo de la entidad DEPARTAMENTO a la tabla DEPTO

Mapeo 2

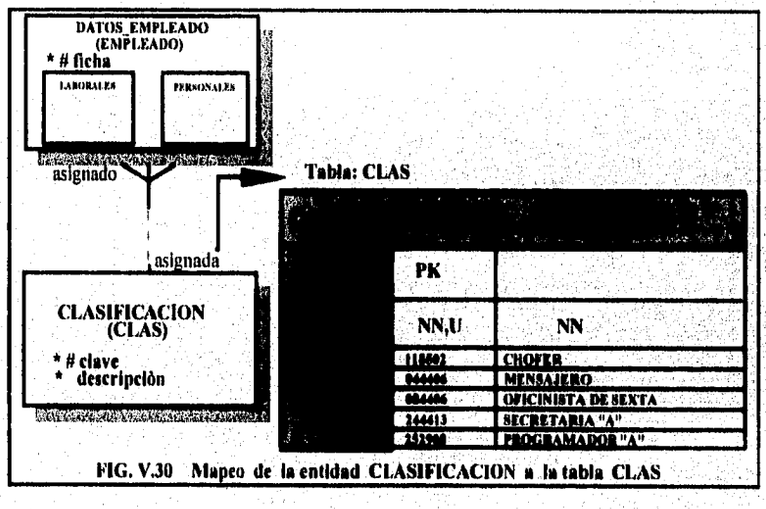


FIG. V.30 Mapeo de la entidad CLASIFICACION a la tabla CLAS

Mapeo 3

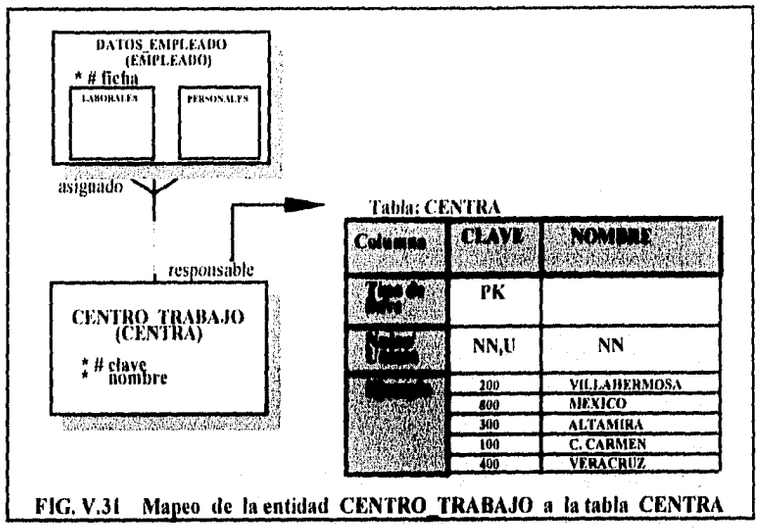


FIG. V.31 Mapeo de la entidad CENTRO_TRABAJO a la tabla CENTRA

Mapeo 4

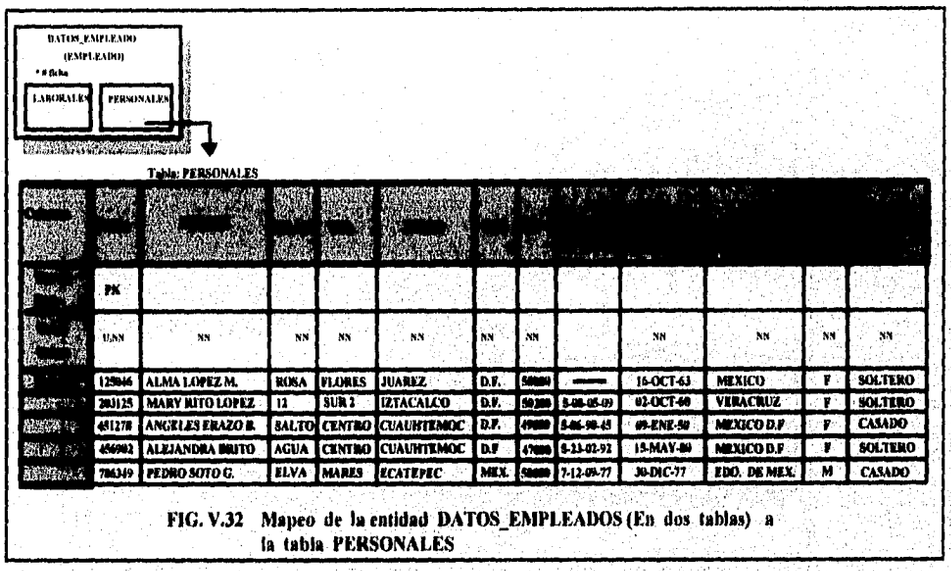


FIG. V.32 Mapeo de la entidad DATOS_EMPLEADOS (En dos tablas) a la tabla PERSONALES

Mapeo 5

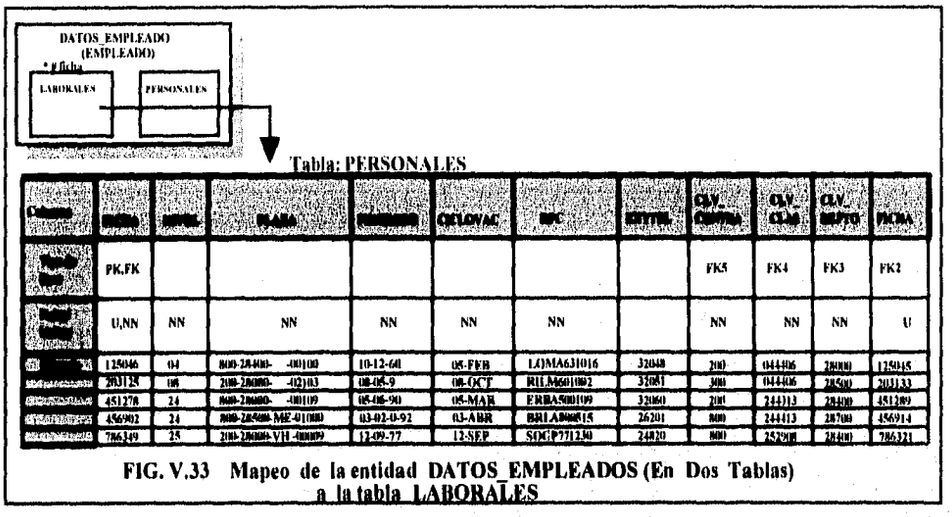


FIG. V.33 Mapeo de la entidad DATOS_EMPLEADOS (En Dos Tablas) a la tabla LABORALES

Mapeo 6

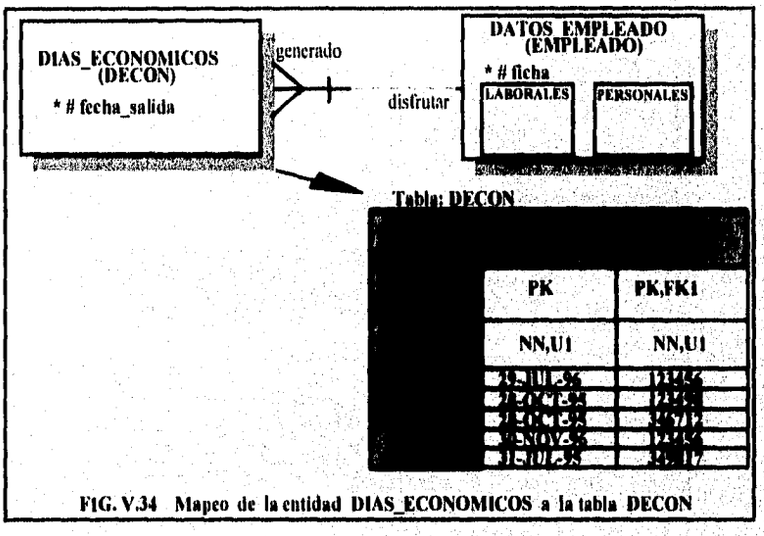


FIG. V.34 Mapeo de la entidad DIAS_ECONOMICOS a la tabla DECON

Mapeo 7

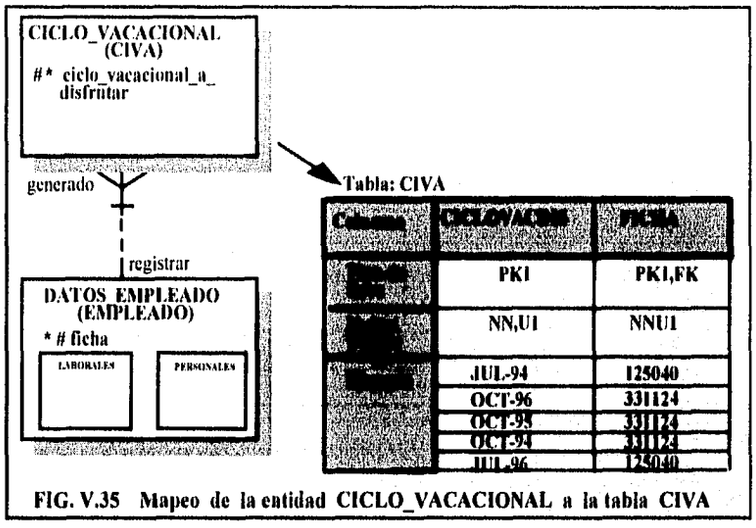


FIG. V.35 Mapeo de la entidad CICLO_VACACIONAL a la tabla CIVA

Mapeo 8

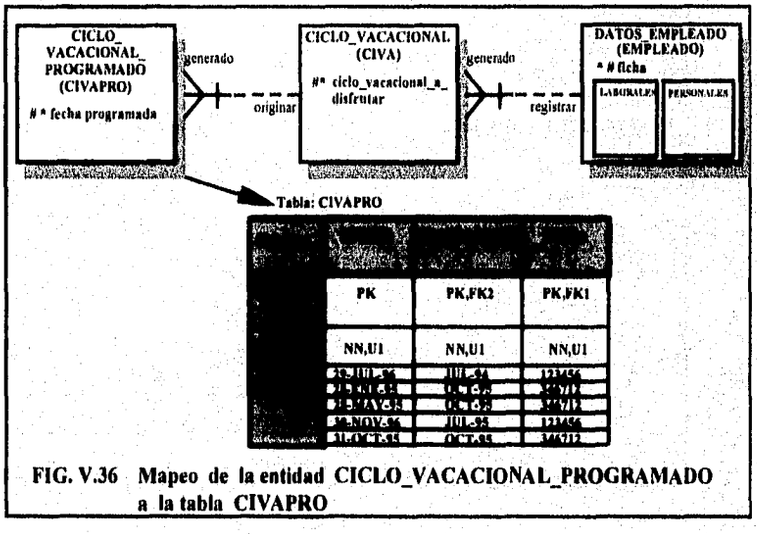


FIG. V.36 Mapeo de la entidad CICLO_VACACIONAL_PROGRAMADO a la tabla CIVAPRO

Mapeo 9

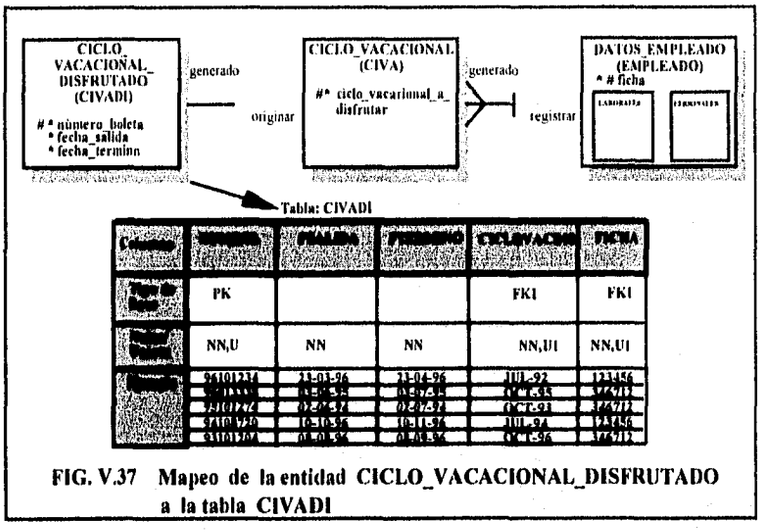


FIG. V.37 Mapeo de la entidad CICLO_VACACIONAL_DISFRUTADO a la tabla CIVADI

Mapeo 10

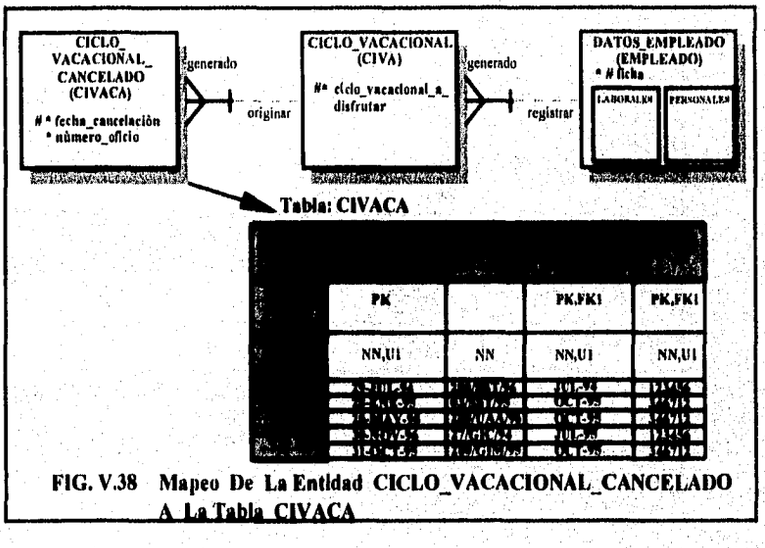


FIG. V.38 Mapeo De La Entidad CICLO_VACACIONAL_CANCELADO A La Tabla CIVACA

Mapeo 11

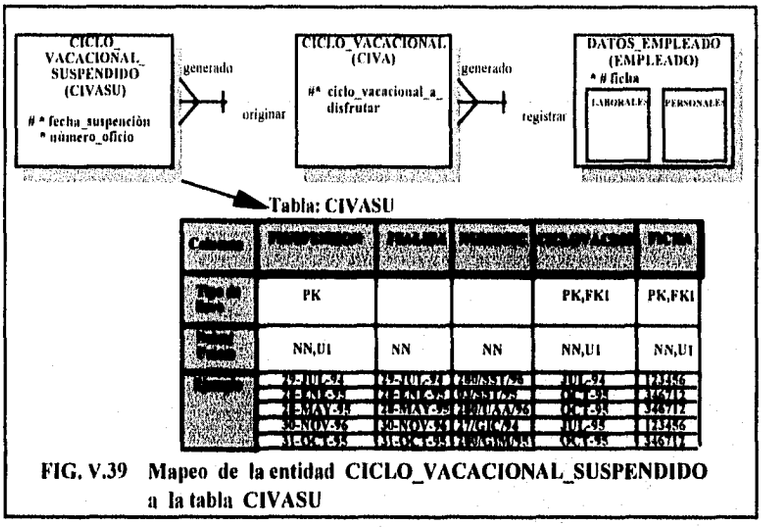


FIG. V.39 Mapeo de la entidad CICLO_VACACIONAL_SUSPENDIDO a la tabla CIVASU

Mapeo 12

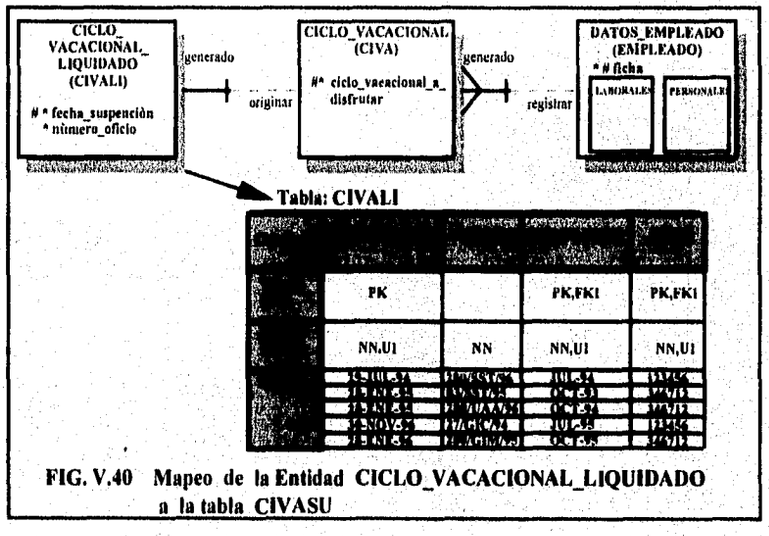


FIG. V.40 Mapeo de la Entidad CICLO_VACACIONAL_LIQUIDADO a la tabla CIVASU

5.12 DEFINICIÓN DE LAS TABLAS DE LA BASE DE DATOS DE VACACIONES

```
CREATE DATABASE VACACIONES


CREATE TABLE DEPTO
(clave        NUMBER(5)       CONSTRAINT pk_depto PRIMARY KEY,
nombre        VARCHAR2(38)    CONSTRAINT nn_depto_nombre NOT NULL);

CREATE UNIQUE INDEX i_depto_clave ON depto (clave ASC );

CREATE TABLE CLAS
(clave        NUMBER(6)       CONSTRAINT pk_clas PRIMARY KEY,
descripcion   VARCHAR2(25)    CONSTRAINT nn_clas_descripcion NOT NULL);

CREATE UNIQUE INDEX i_clas_clave ON clas (clave ASC);

CREATE TABLE CENTRA
(clave        NUMBER(3)       CONSTRAINT pk_centra PRIMARY KEY,
nombre        VARCHAR2(25)    CONSTRAINT nn_centra_nombre NOT NULL);

CREATE UNIQUE INDEX i_centra_clave ON centra (clave ASC);

CREATE TABLE PERSONALES
(ficha        NUMBER(6)       CONSTRAINT pk_personales PRIMARY KEY,
 nombre       VARCHAR2(38)    CONSTRAINT nn_ personales_nombre NOT NULL,
 calle        VARCHAR2(20)    CONSTRAINT nn_ personales_calle NOT NULL,
 col          VARCHAR2(20)    CONSTRAINT nn_ personales_col NOT NULL,
 deleg        VARCHAR2(20)    CONSTRAINT nn_ personales_deleg NOT NULL,
 edo          VARCHAR2(20)    CONSTRAINT nn_ personales_edo NOT NULL,
 tel          VARCHAR2(10)
 f_nac        DATE            CONSTRAINT nn_ personales_f_nac NOT NULL,
 l_nac        VARCHAR2(20)    CONSTRAINT nn_ personales_l_nace NOT NULL,
 sexo         VARCHAR2(1)     CONSTRAINT nn_ personales_sexo NOT NULL,
                              CONSTRAINT ck_ personales_sexo CHECK(sexo = UPPER (sexo)),
 e_civil      VARCHAR2(1)     CONSTRAINT nn_ personales_e_civil NOT NULL,
                              CONSTRAINT ck_ personales_e_civil CHECK(e_civil = UPPER
                                   (e_civil));

CREATE UNIQUE INDEX i_personales_ficha ON personales (ficha ASC);

CREATE TABLE LABORALES
(ficha        NUMBER(6)       CONSTRAINT pk_laborales  PRIMARY KEY,
                              CONSTRAINT fk_ laborales_ficha REFERENCES personales (ficha),
 nivel        VARCHAR2(2)     CONSTRAINT nn_ laborales_nivel NOT NULL,
 plaza        VARCHAR2(18)    CONSTRAINT nn_ laborales_plaza NOT NULL,
```

```
fingreso        DATE            CONSTRAINT nn_ laborales_fingreso NOT NULL,
ciclovac        VARCHAR2(6)     CONSTRAINT nn_ laborales_ciclovac NOT NULL,
rfc             VARCHAR2(10)    CONSTRAINT nn_ laborales_rfc NOT NULL,
exttel          VARCHAR2(5),
clv_centra      NUMBER(3)       CONSTRAINT nn_ laborales_clv_centra NOT NULL,
                                CONSTRAINT fk_ laborales_clv_centra FOREIGN KEY
                                        (clv_centra) REFERENCES centra (clave),
clv_clas        NUMBER(6)       CONSTRAINT nn_ laborales_clv_clas NOT NULL,
                                CONSTRAINT fk_ laborales_clv_clas FOREIGN KEY
                                        (clv_clas) REFERENCES clas (clave),
clv_depto       NUMBER(5)       CONSTRAINT nn_ laborales_clv_depto NOT NULL,
                                        CONSTRAINT fk_ laborales_clv_depto FOREIGN
KEY                                             (clv_depto) REFERENCES depto (clave),
clv_ficha       NUMBER(6)       CONSTRAINT nn_ laborales_clv_ficha NOT NULL,
                                        CONSTRAINT fk_ laborales_clv_ficha FOREIGN
KEY                                             (clv_ficha) REFERENCES  laborales
(ficha));
```

CREATE UNIQUE INDEX i_laborales_ficha ON laborales (ficha ASC);

CREATE TABLE DECON

```
(fsalida        DATE            CONSTRAINT nn_ decon_fsalida NOT NULL,
ficha           NUMBER(6)       CONSTRAINT nn_ decon_ficha NOT NULL,
                                CONSTRAINT pk_decon PRIMARY KEY (ficha,fsalida),
                                        CONSTRAINT fk_ decon_ficha FOREIGN KEY
(ficha)                                     REFERENCES laborales (ficha));
```

CREATE UNIQUE INDEX i_decon_ficha_fsalida ON decon (ficha,fsalida ASC);

CREATE TABLE CIVA

```
(ciclovacdis    VARCHAR2(6)     CONSTRAINT nn_ civa_ciclovacdis NOT NULL,
 ficha          NUMBER(6)       CONSTRAINT nn_ civa_ficha NOT NULL,
                                CONSTRAINT pk_civa PRIMARY KEY (ficha,ciclovacdis),
                                CONSTRAINT fk_ civa_ficha FOREIGN KEY (ficha) REFERENCES
                                        laborales (ficha));
```

CREATE UNIQUE INDEX i_civa_ficha_ciclovacdis ON civa (ficha,ciclovacdis ASC);

CREATE TABLE CIVAPRO

```
(fprog          DATE            CONSTRAINT nn_ civapro_fprog NOT NULL,
 ciclovacdis    VARCHAR2(6)     CONSTRAINT nn_ civapro_ciclovacdis NOT NULL,
 ficha          NUMBER(6)       CONSTRAINT nn_ civapro_ficha NOT NULL,
                                CONSTRAINT pk_civapro PRIMARY KEY (ficha,ciclovacdis,fprog),
                                 CONSTRAINT fk_ civapro_ficha_ciclo FOREIGN KEY
                                        (ficha,ciclovacdis) REFERENCES civa (ficha,ciclovacdis));
```

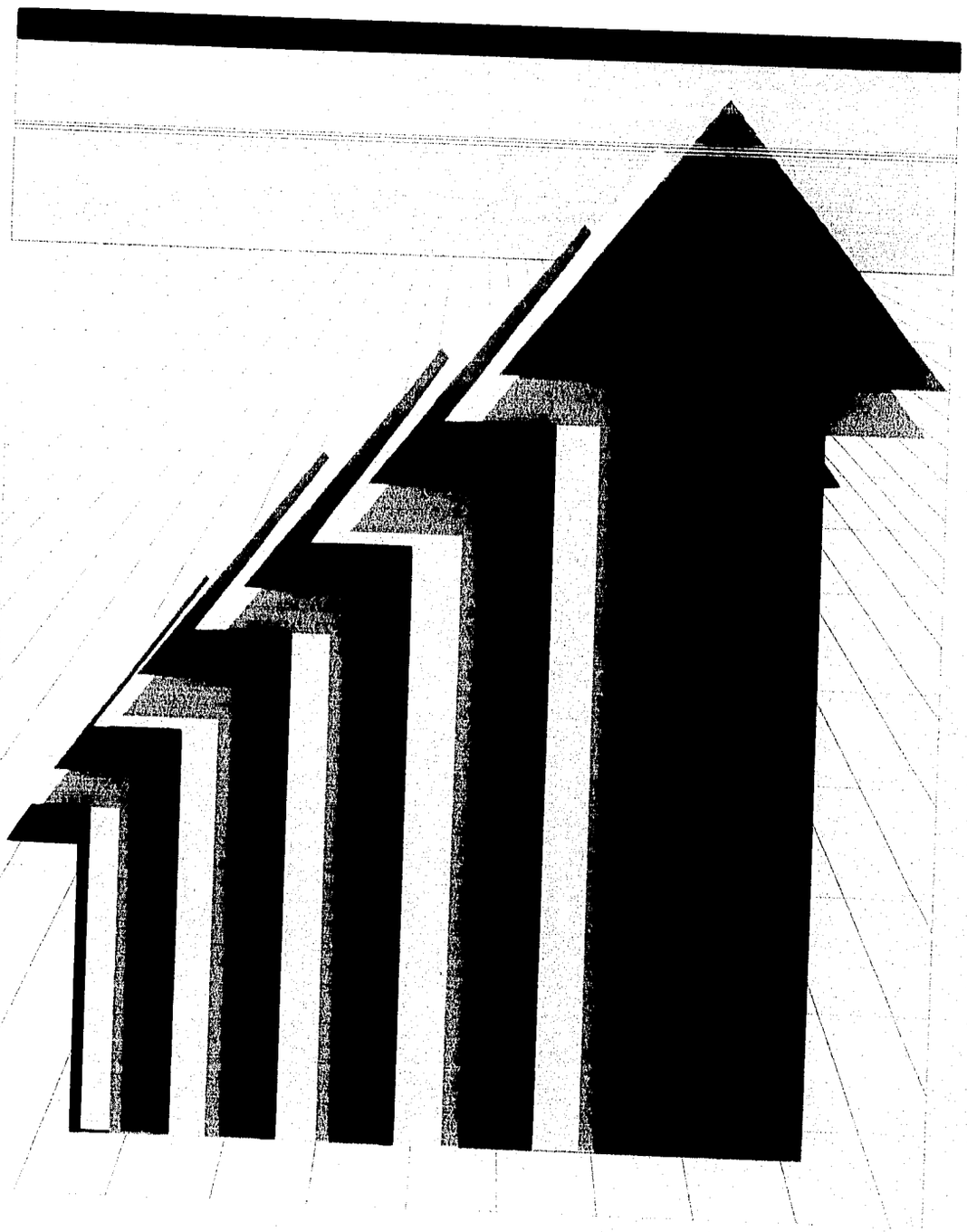
CREATE UNIQUE INDEX i_civapro_ficha_ciclovacdis_fprog ON civapro (ficha, ciclovacdis,fprog ASC);

```
CREATE TABLE CIVADI
(numbol       VARCHAR2(8)     CONSTRAINT nn_ civadi_numbol NOT NULL,
 fsalida      DATE            CONSTRAINT nn_ civadi_fsalida NOT NULL,
 ftermino     DATE            CONSTRAINT nn_ civadi_ftermino NOT NULL,
 ciclovacdis  VARCHAR2(6)     CONSTRAINT nn_ civadi_ciclovacdis NOT NULL,
 ficha        NUMBER(6)       CONSTRAINT nn_ civadi_ficha NOT NULL,
                              CONSTRAINT pk_civadi PRIMARY KEY (numbol),
                              CONSTRAINT fk_ civadi_ficha_ciclo FOREIGN KEY
(ficha,ciclovacdis)
                                        REFERENCES civa (ficha,ciclovacdis));

CREATE UNIQUE INDEX i_civadi_numbol ON civadi (numbol ASC);

CREATE TABLE CIVACA
(fcancelacion DATE            CONSTRAINT nn_ civaca_fcancelacion NOT NULL,
 numofic      VARCHAR2(10)    CONSTRAINT nn_ civaca_numofic NOT NULL,
 ciclovacdis  VARCHAR2(6)     CONSTRAINT nn_ civaca_ciclovacdis NOT NULL,
 ficha        NUMBER(6)       CONSTRAINT nn_ civaca_ficha NOT NULL,
                              CONSTRAINT pk_civaca PRIMARY KEY (ficha,ciclovacdis,
                                        fcancelacion),
                              CONSTRAINT fk_ civaca_ficha_ciclo FOREIGN KEY
(ficha,ciclovacdis)
                                        REFERENCES civa (ficha,ciclovacdis));

CREATE UNIQUE INDEX i_civaca_ficha_ciclovacdis_fcancelacion ON civaca (ficha,
ciclovacdis, fcancelacion ASC);

CREATE TABLE CIVASU
(fsuspensionl DATE            CONSTRAINT nn_ civasu_fsuspension NOT NULL,
 fsalida      DATE            CONSTRAINT nn_ civasu_fsalida NOT NULL,
 numofic      VARCHAR2(10)    CONSTRAINT nn_ civasu_numofic NOT NULL,
 ciclovacdis  VARCHAR2(6)     CONSTRAINT nn_ civasu_ciclovacdis NOT NULL,
 ficha        NUMBER(6)       CONSTRAINT nn_ civasu_ficha NOT NULL,
                              CONSTRAINT pk_civasu PRIMARY KEY (ficha,ciclovacdis,
                                        fsuspension),
                              CONSTRAINT fk_ civasu_ficha_ciclo FOREIGN KEY
(ficha,ciclovacdis)
                                        REFERENCES civa (ficha,ciclovacdis));

CREATE UNIQUE INDEX i_civasu_ficha_ciclovacdis_fsuspension ON civasu (ficha,
ciclovacdis, fsuspension ASC);

CREATE TABLE CIVALI
(fliquidacion DATE            CONSTRAINT nn_ civali_fliquidacion NOT NULL,
 numofic      VARCHAR2(10)    CONSTRAINT nn_ civali_numofic NOT NULL,
ciclovacdis   VARCHAR2(6)     CONSTRAINT nn_ civali_ciclovacdis NOT NULL,
 ficha        NUMBER(6)       CONSTRAINT nn_ civali_ficha NOT NULL,
                              CONSTRAINT pk_civali PRIMARY KEY (ficha,ciclovacdis,
                                        fliquidacion),
```

```
                                CONSTRAINT fk_civali_ficha_ciclo FOREIGN KEY
(ficha,ciclovacdis)
                                REFERENCES civa (ficha,ciclovacdis));
CREATE UNIQUE INDEX i_civali_ficha_ciclovacdis_fliquidacion ON civali (ficha,
ciclovacdis, fliquidacion ASC);
CREATE TRIGGER tb_centra
AFTER
DELETE
ON centra
BEGIN
    UPDATE laborales
    SET
    laborales.clv_centra = NULL
    WHERE
    laborales.clv_centra = :old.clv_centra
END;

CREATE TRIGGER tb_clas
AFTER
DELETE
ON clas
BEGIN
    UPDATE laborales
    SET
    laborales.clv_clas = NULL
    WHERE
    laborales.clv_clas = :old.clv_clas
END;

CREATE TRIGGER tb_depto
AFTER
DELETE
ON depto
BEGIN
    UPDATE laborales
    SET
    laborales.clv_depto = NULL
    WHERE
    laborales.clv_depto = :old.clv_depto
END;

CREATE TRIGGER ta__centra
AFTER
```

```
UPDATE
ON centra
BEGIN
    IF : old.clv_centra <> : new.clv_centra
    THEN
      UPDATE laborales
      SET
      laborales.clv_centra = NULL
    WHERE
      laborales.clv_centra = :old.clv_centra
    END IF;
END;

CREATE TRIGGER ta__clas
AFTER
UPDATE
ON clas
BEGIN
    IF : old.clv_clas <> : new.clv_clas
    THEN
      UPDATE laborales
      SET
      laborales.clv_clas = NULL
    WHERE
      laborales.clv_clas = :old.clv_clas
    END IF;
END;

CREATE TRIGGER ta__depto
AFTER
UPDATE
ON depto
BEGIN
    IF : old.clv_depto <> : new.clv_depto
    THEN
      UPDATE laborales
      SET
      laborales.clv_depto = NULL
    WHERE
      laborales.clv_depto = :old.clv_depto
    END IF;
END;
```

CONCLUSIONES



CONCLUSIONES

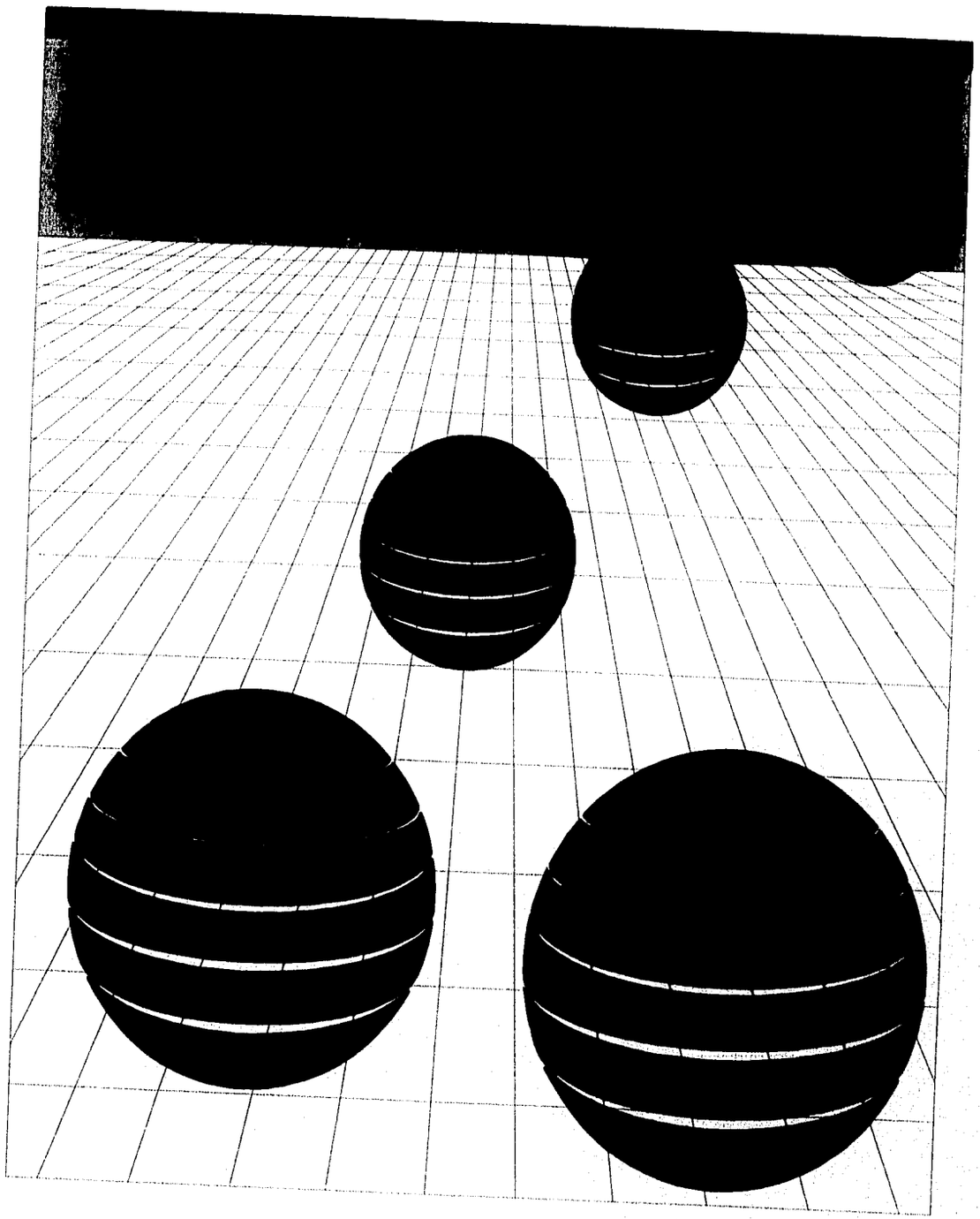
Las conclusiones a las que llegamos de acuerdo al trabajo y a las investigaciones realizadas, es que las Bases de Datos Relacionales tienen una gran aceptación y son las que actualmente están predominando en el mercado. Estas Bases de Datos Relacionales son las que cuentan con las mejores características para: controlar la redundancia; mantener la consistencia; lograr la integración de los datos; compartir los datos entre las diferentes aplicaciones; cumplir con los estándares; tener facilidad en el desarrollo de aplicaciones; uniformar los controles de seguridad, privacia e integridad; independencia entre los datos y los programas y reducir el mantenimiento a los programas. En resumen el enfoque de Base de Datos Relacional provee una solución sólida y simple al procesamiento de información evitando muchos de los problemas que representa el enfoque tradicional, como son: que cada aplicación tiene sus propias estructuras de datos, se generan arquitectura caóticas y redundantes. Además resulta una labor titánica tratar de obtener información proveniente de estructuras de datos de diversas aplicaciones. Adicionalmente, otra característica de este tipo de instalaciones es que emplean a la computadora para almacenar, procesar y producir información, sin preocuparse por controlar y vigilar su integridad.

A través de este trabajo podemos concluir que es necesario seguir paso a paso el Diseño de Datos Relacional para la obtención de una Base de Datos

Relacional la cual cumpla con los requerimientos de información especificados por el cliente, con lo cual nos dará una Base de Datos consistente, confiable y que nos facilite la administración de la misma e integración de las aplicaciones.

En estos días surgen otros enfoques de Bases de Datos (Bases Orientadas a Objetos) con respecto a las nuevas necesidades de información requerida por los usuarios voz, imagen, gráficos complejos, etc. en estos momentos queda la interrogante de que si las Bases de Datos Relacionales seguirán predominando como hasta hoy lo han hecho o será el comienzo de darle paso al nuevo concepto de Bases de Datos Orientadas a Objetos. Aún los grandes Desarrolladores de Manejadores de Bases de Datos no se ponen de acuerdo y divergen aún en este tema. Mientras Oracle se prepara para arrancar con un nuevo producto de Bases de Datos Orientadas a Objetos, con lo cual apuesta a esta nuevo concepto de Bases de Datos. Por otro lado Informix propone seguir trabajando con el concepto de Bases de Datos Relacionales. Nosotros lo que si podemos decir es que va a pasar algún tiempo en que seguiran las Bases de Datos Relacionales en el mercado, y si algún día se cambia a esta nueva tecnología los fabricantes de Bases de Datos no podrán dejar desprotegidos a sus usuarios y tendrán que ver la convertibilidad de las Bases de Datos Relacionales en Bases de Datos Orientadas objetos.

GLOSARIO



GLOSARIO

ANSI.- (American National Standards Institute) Organismo americano y miembro de la ISO que participa en el desarrollo de normas para la comunicación de datos.

API.- (Application Program Interface) Interface para programas de aplicación. Un conjunto de llamadas y rutinas formalizadas de software a las que puede hacer referencia un programa de aplicación para tener acceso a los servicios básicos de la red.

ATRIBUTO AGREGADO.- Objeto que puede ser representado como un conjunto de cosas homogéneas.

BASE DE DATOS.- Una Base de Datos es un conjunto de datos relacionados entre si, almacenados, estructurados, no redundante, (normalizada) y de fácil acceso

CLIENTE.- Es un dispositivo o entidad en una arquitectura de computación distribuida, que solicita servicios e información.

CONSTRAINTS.- Restricciones de integridad de la Base de Datos.

DBA.- El administrador de la Base de Datos (DBA), es la persona o grupo responsable del control del DMBS.

DBMS.- (DataBase Management System), es un sistema de manejo de Base de Datos que consiste en un conjunto de datos relacionados entre si y un conjunto de programas para tener acceso a esos datos. el conjunto de datos se conoce comúnmente como Base de Datos. Esta contiene información acerca de una organización determinada. El objetivo primordial de un DBMS es crear un ambiente en que sea posible guardar y recuperar información de la Base de Datos en forma conveniente y eficiente.

DDL.- (Date Definition Languaje) lenguaje de definición de datos. El Administrador de la Base de Datos (DBA), utiliza este lenguaje para

crear columnas, para definir sus características y para indicar a qué tabla pertenecen estas columnas.

DICCIONARIO DE DATOS.- Un diccionario de datos es una Base de Datos que contiene datos acerca de los datos de la Base de Datos; es una herramienta para identificar y clasificar los datos almacenados en la Base de Datos. Es una librería central para definir el significado, uso, características y otros datos relevantes de todas las entidades, sinónimos, referencias cruzadas y relaciones que existen entre ellas; consiste de archivos, registros y campos que contienen información descriptiva de los datos de la Base de Datos. Por ejemplo, nos dice cuantas y cuales son las columnas de la tabla de empleados, además mencionar que tipo de datos son válidos para cada columna.

DML.- (Date Manipulation Languaje) lenguaje de manipulación de datos. Este lenguaje se utiliza para actualizar los Datos de la Base de Datos así como insertar nuevos registros.

ENTIDAD.- Objeto tangible que puede describirse con palabras, código numérico o no numérico.

ESQUEMA.- Describe un modelo de base de Datos conceptual por medio de la definición no sólo de los campos y registros, sino también de las relaciones entre los datos dentro de los diferentes registros.

HOST.- El computador central (o uno de un grupo de computadores) en un sistema de comunicación de datos, que prove las funciones primarias de procesamiento de datos tales como computación acceso a Base de Datos o programas especiales o lenguajes de programación; con frecuencia abreviado como "principal".

INTERFAZ.- Es un límite compartido; un punto físico de desmarcación entre dos dispositivos, donde las señales eléctricas, conectores, temporización y control de flujo están definidos; los procedimientos, códigos y protocolos que permiten a dos entidades interactuar para un intercambio de información.

LLAVE PRIMARIA.- Campo cuyo contenido puede identificar de manera única cada registro del archivo.

LLAVE SECUNDARIA ó FORANEA.- Cualquier campo a excepción de la llave primaria, puede designarse como secundaria y se utiliza como llave de consulta. Campo de conexión en terminología relacional.

MAIN-FRAME.- Es una computadora en gran escala que puede alojar Software completo y varios equipos periféricos, y también manejar muchos usuarios, generalmente cientos de ellos.

MAPEO.- Transformación de datos de una forma y un contexto, en otra forma y otro contexto. Por ejemplo, registros lógicos se mapean en registros físicos para su almacenamiento.

MICROCOMPUTADOR.- Computador de escritorio. Computador que utiliza un microprocesador para su CPU. Computador personal (PC).

MODELO ENTIDAD-RELACION.- Método empleado en el diseño de base de datos basado en el análisis de tres modelos semánticos claves: entidades, relaciones y atributos.

ORACLE.- Oracle es una compañía dedicada a ofrecer un rango completo de software y servicios basados en el sistema manejador de Bases de Datos relaciónales Oracle (RDBMS), Oracle rdbms es líder en el mercado con miles de instalaciones alrededor de todo el mundo y de diversos sectores en el mercado, todos ellos con el propósito de dar la mejor solución en estrategia de información dentro de diferentes negocios.

RDBMS.- Es un sistema de manejo de Base de Datos relacional que tiene las siguientes propiedades : representación de datos en forma de tabla, lenguaje de 4a generación (sintaxis no procedural), capacidades relaciónales completas (navegación automática y todos los operadores relaciónales), flexibilidad (fácil de modificar datos y cambiar la estructura de los datos) y diccionario de datos integrado.

SERVIDOR.- Una computadora que mantiene programas, archivos o memoria, compartidos por los usuarios de una red.

SOFTWARE.- Un programa de computadora o conjunto de programas mantenidos en algunos medios de almacenamiento y cargados en memoria de lectura/escritura (RAM) para su ejecución.

SOFTWARE DE APLICACION.- Programas que realizan funciones útiles en el procesamiento o manipulación de datos; incluye administradores de Bases de Datos, procesadores de palabras, hojas de cálculo y otros programas que permiten la manipulación útil de datos.

SQL.- (Structured Query Languaje) lenguaje de estructura de consultas. Es un lenguaje Estándar para Bases de Datos Relaciónales.

SQL*FORMS.- Es el programa que trae Oracle para crear los formatos de captura de los datos, facilitando con esto el diseño de pantallas en los sistemas

SQL*REPORTS.- Es el programa que trae Oracle para generar los reportes que el usuario requiera, facilitando la creación de éstos para el sistema.

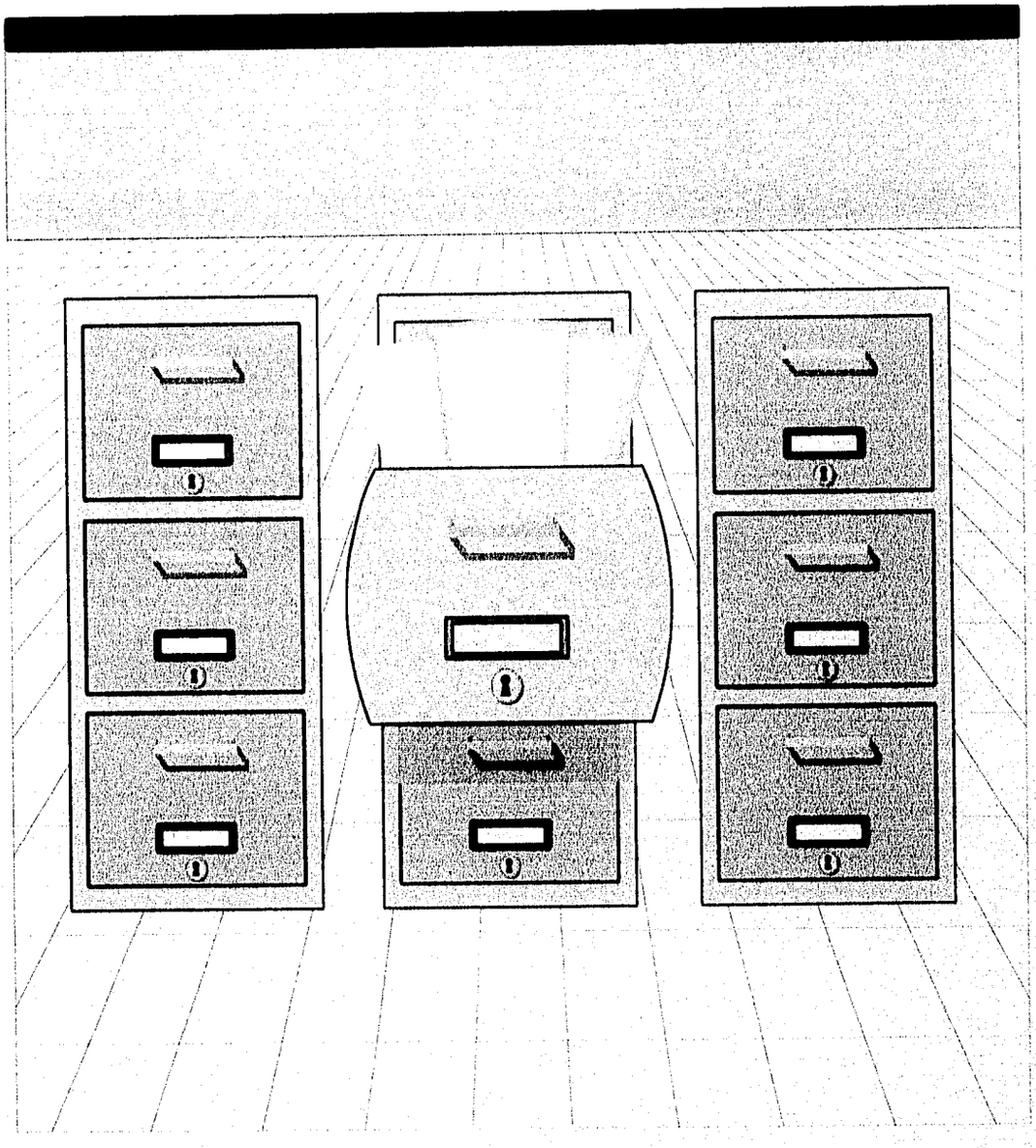
TRIGGER (DISPARADOR).- Los disparadores son acciones que ejecuta automáticamente el sistema gestor de la Base de Datos cuando se produce un cambio (MIB) en los valores de una tabla. Los tipos de cambios, la tabla afectada, los atributos, así como las acciones a llevar a cabo, lo define el usuario según sus necesidades.

Un trigger define la acción a tomar cuando se realiza alguna actualización sobre una tabla.

Los disparadores sólo se han incorporado en algunas extensiones de SAL y son capaces de tratar sentencias condicionales. Las ventajas de los trigger son: La disminución de la complejidad de las aplicaciones y la facilidad para diseñar reglas semánticas según se decida. Las desventajas son que el SGBD es más complejo y los programadores, usuarios desconocen las operaciones que se realizan internamente en la Base de Datos y esto lleva a que existan transacciones no tan rápidas como se esperaba.

UID.- Son las iniciales para representar a un "Identificador Unico". Es un atributo o conjunto de atributos que identifican de manera única a una instancia dentro de una

APENDICE



ANEXO A

TIPOS DE ENFOQUE DE BASES DE DATOS

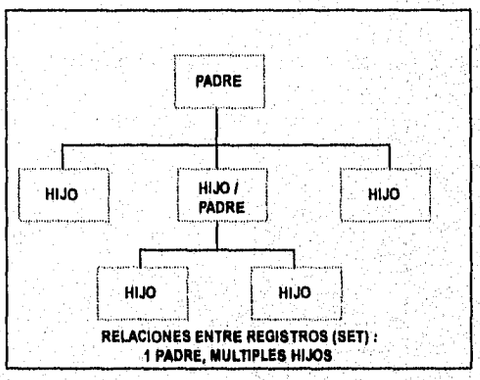
Existen enfoques alternativos para visualizar y manejar datos a un nivel lógico independientemente de cualquier estructura física de soporte en que se basen.

Los modelos de Base de Datos que existen son:

- **Modelo Jerárquico**
- **Modelo de Red**
- **Modelo Relacional**
- **Modelo Orientado a Objetos**

En el Enfoque Jerárquico:

- Los datos se presentan como estructuras de árbol.
- El árbol representa una jerarquía
- El procesamiento es Top-Down, navegacional



La estructura lógica en la cual se sustenta la Base de Datos Jerárquica es el árbol. Un árbol se compone de un nodo raíz y varios nodos sucesores, ordenados

jerárquicamente. Cada nodo representa una entidad (tipo de registro) y las relaciones entre entidades son las conexiones entre los nodos.

El nodo colocado en la parte superior es llamado padre y los nodos inferiores son los hijos.

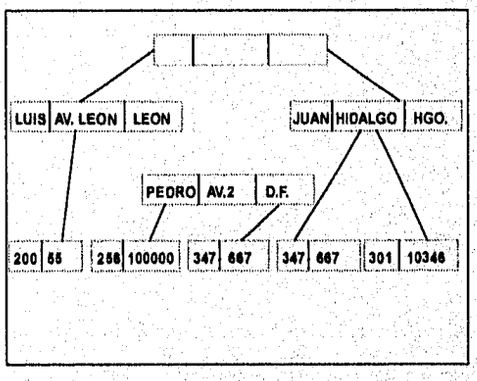
En el sistema jerárquico, las conexiones entre archivos no dependen de la información contenida en ellos, se definen al inicio y son fijos.

La característica sobresaliente de este modelo es el manejo de la conexión uno a muchos, entre un padre y varios hijos, en otras palabras, cada hijo solo tiene un padre

Tenemos 2 entidades: cuentahabiente y cuenta, relacionadas entre si con una relación muchos a muchos es decir, un cuentahabiente puede tener varias cuentas, y una cuenta puede pertenecer a varios cuentahabientes.

El tipo de registro cuentahabiente consiste de 3 campos: nombre, calle y ciudad.

El tipo de registro cuenta tiene 2 campos: número y saldo.

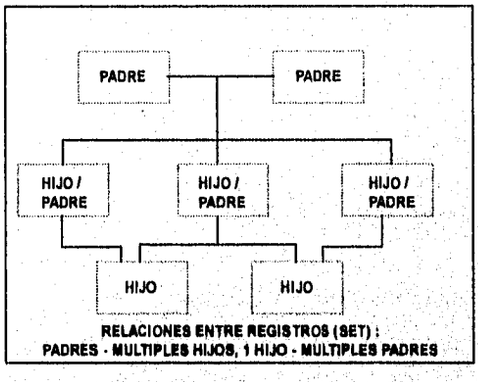


Desventajas en el enfoque jerárquico:

- No modela sencillamente las relaciones M a M
- Anomalías de inserción
- Anomalías de borrado
- Anomalías de Actualización
- Se pueden dar consultas inconsistentes

En el Enfoque de Red:

- Representación de los datos similar al Modelo Jerárquico.
- Se pueden tener relaciones de un hijo-varios padres.
- En una Red, un hijo puede tener varios hijos y varios padres a la vez.



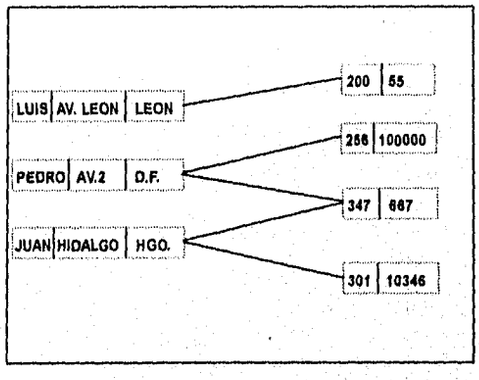
Los datos se representan como registros ligados formando un conjunto de datos interceptados.

La Base de Datos de Red, a diferencia de las jerárquicas, permite cualquier conexión entre entidades, es decir, se pueden representar relaciones de muchos a muchos. En una red, un hijo puede tener varios padres y varios hijos a la vez.

Tenemos 2 entidades: cuentahabiente y cuenta, relacionadas entre si con una relación muchos a muchos, es decir, un cuentahabiente puede tener varias cuentas, y una cuenta puede pertenecer a varios cuentahabientes.

El tipo de registro cuentahabiente consiste de 3 campos: nombre, calle y ciudad.

El tipo de registro cuenta tiene 2 campos: número y saldo.



Desventajas en el enfoque de red:

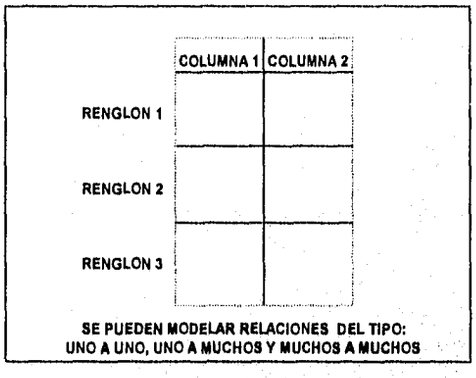
- Resulta difícil definir nuevas relaciones.
- Es complicado darle mantenimiento ya que cualquier cambio en la estructura requiere una descarga en los datos.
- Representa desperdicio de recursos.
- Anomalías de inserción.
- Anomalías de borrado.

En el Enfoque Relacional:

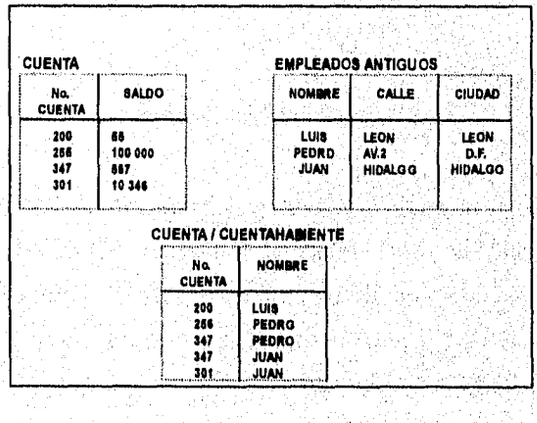
La estructura lógica de una Base de Datos relacional está basada en la representación de entidades mediante tablas, las cuales constan de columnas

(campos) y renglones (registros). Las relaciones entre tablas se llevan a cabo a través de un conjunto de columnas que se tengan en común logrando una conexión dinámica entre un número ilimitado de ellas a través del contenido de esas columnas.

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



Tenemos 2 entidades: cuentahabiente y cuenta relacionadas entre si con una relación muchos a muchos, es decir, un cuentahabiente puede tener varias cuentas, y una cuenta puede pertenecer a varios cuentahabientes.



La tabla cuentahabiente consiste de 3 columnas: nombre, calle y ciudad.

La tabla cuenta tiene 2 columnas: número y saldo.

Características:

- Representación de datos a través de tablas.
- Desarrollo de aplicaciones a través de herramientas de alta productividad
- Flexibilidad en el mantenimiento de las estructuras y de los datos, en el tipo de consultas.
- Diccionario de Datos integrado.
- Soporta a todos los operadores relaciónales.

Ventajas:

- Simplicidad:
  - Fácil de usar.
  - Fácil obtener respuestas.
  - Fácil insertar y actualizar datos.
  - Fácil cambiar la estructura de los datos.
  - La navegación es responsabilidad del DBMS. NO del programador.
- Poder:
  - Todas las consultas son posibles.

Existen en el mercado sistemas manejadores de Bases de Datos relaciónales que originalmente no eran relaciónales, como: IDMS de Cullinet que fue originalmente una Base de Datos en Red y ahora se conoce con el nombre de IDMS/R; TOTAL de Cincom originalmente era una Base de Datos en Red y ahora se conoce como SUPRA (relacional).

Sin embargo existen productos que desde su origen fueron relaciónales como: ORACLE, INGRES, DB2 e INFORMIX

- Sistemas Manejadores de Bases de Datos Originales
  - IDMS (RED) CULLINET
  - IMS (JERARQUICO) IBM
  - TOTAL (RED) CINCOM
- Sistemas Manejadores de Bases de Datos RE-BORNS
  - IDMS/R CULLINET
  - SUPRA CINCOM
- Sistemas Manejadores de Bases de Datos de Origen Relacional
  - ORACLE
  - INGRES
  - INFORMIX
  - IDB2

Modelo Orientado a Objeto

El propósito de los sistemas de Bases de Datos es la gestión de grandes cantidades de información. Las primeras Bases de Datos surgieron del desarrollo de los sistemas de gestión de archivos. Estos sistemas primero evolucionaron en Base de Datos de red o en Bases de Datos jerárquicas y, más tarde, en Bases de Datos relaciónales. Entre las características comunes de estas aplicaciones están:

- Uniformidad
- Orientación en registros
- Datos pequeños
- Campos atómicos
- Transacciones cortas
- Esquemas de concepto estático

Sin embargo, en años recientes, la tecnología de Bases de Datos se ha adaptado a aplicaciones fuera del ámbito del procesamiento de datos que, en general, les falta al menos una de las características anteriores. Estas nuevas aplicaciones incluyen:

- Diseño asistido por computadora(Computer-aided design (CAD)).
- Ingeniería de software asistida por computadora (Computer-aided software engineering (CASE)).
- Bases de Datos de Multimedios.
- Sistemas de información de oficina (Office information Systems (OIS)).
- Sistemas expertos de Bases de Datos.

Estas nuevas aplicaciones de las Bases de Datos no se consideraron en la década de los setenta, que fue cuando se diseñaron la mayor parte de los sistemas de

Base de Datos actuales. En la actualidad son prácticas debido al aumento en el tamaño de memoria principal disponible, la velocidad de las unidades centrales de procesamiento, la reducción del coste del hardware y la mejora en el entendimiento de la gestión de la Base de Datos que se ha logrado en los últimos años.

Estas nuevas aplicaciones requieren nuevos modelos de datos, nuevos lenguajes de consulta y nuevos modelos de transiciones. Entre los requisitos de estas nuevas aplicaciones están:

- Objetos Complejos. Un objeto complejo es un dato que es visto como un simple objeto en el mundo real, pero que contiene otros objetos.

- Datos de Comportamiento. Puede que distintos objetos necesiten responder de diferentes formas a la mismo orden.

- Meta conocimiento. A menudo los datos más importantes sobre aplicaciones son reglas generales acerca de la aplicación más que de las tuplas específicas.

- Transacciones de larga duración. Las aplicaciones CAD y CASE implican interacción humana con los datos.

El modelo orientado a objetos se basa en encapsular código y datos en una única unidad, llamada objeto. El interfaz entre un objeto y el resto del sistema se define mediante un conjunto de mensajes. En general, un objeto tiene asociado:

- Un conjunto de variables que contienen los datos del objeto. El valor de cada variable es un objeto.

- Un conjunto de mensajes a los que el objeto responde.

- Un método, que es un trozo de código para implementar cada mensaje. Un método devuelve un valor como respuesta al mensaje.

El término mensaje en un contexto orientado a objeto no implica el uso de un mensaje físico en una red de computadoras, sino que se refiere al paso de solicitudes entre objetos sin tener en cuenta detalles específicos de implementación.

Puesto que el único interfaz externo que presenta un objeto es el conjunto de mensajes al que responde, es posible modificar la definición de métodos y variables sin afectar a otros objetos. También es posible sustituir una variable por un método que calcule un valor.

La capacidad de modificar la definición de un objeto sin afectar al resto del sistema está considerada como una de las mayores ventajas del modelo de programación orientado a objetos.

Jerarquía de Clases

Normalmente en una Base de Datos existen muchos objetos similares. Por similar queremos decir que responden a los mismos mensajes, utilizan los mismos métodos y tienen variables del mismo nombre y tipo. Sería un trabajo inútil definir cada uno de estos objetos por separado. Por tanto, agrupamos los objetos similares para que formen una clase. A cada uno de estos objetos se le llama instancia de su clase. Todos los objetos de una clase comparten una definición común, aunque difieran en los valores asignados a las variables.

El concepto de clases es similar al concepto de tipos abstractos de datos. Sin embargo, en el concepto de clase existen varios aspectos adicionales más allá de los de los tipos abstractos de datos. Para representar estas propiedades adicionales, tratamos cada clase como si fuera un objeto. Un objeto clase incluye:

- Una variable con valores en un conjunto cuyo valor es el conjunto de todos los objetos que son instancias de la clase.

- Implementación de un método para el nuevo mensaje, el cual crea una nueva instancia de la clase.

Un esquema de Base de Datos orientada a objetos normalmente requiere un gran número de Clase. Sin embargo, a menudo se da el caso de que varias clases son similares. caso de que varias clases son similares. Sería deseable definir una representación para las variables comunes en un sito. Esto puede hacerse sólo si los empleados y los clientes están combinados en una clase. Las variables asociadas a ceda clase en el ejemplo son:

- Persona; número-seguridad-social, nombre, dirección, número-teléfono-particular, fecha de -nacimiento.

- Cliente: tasa-crédito, estado-retención-impuestos, número-teléfono-trabajo.Empleado: fecha-de-contrato, salario, número-de-dependientes.

- Empleado:fecha-de-contrato, salario, número-de-dependientes.

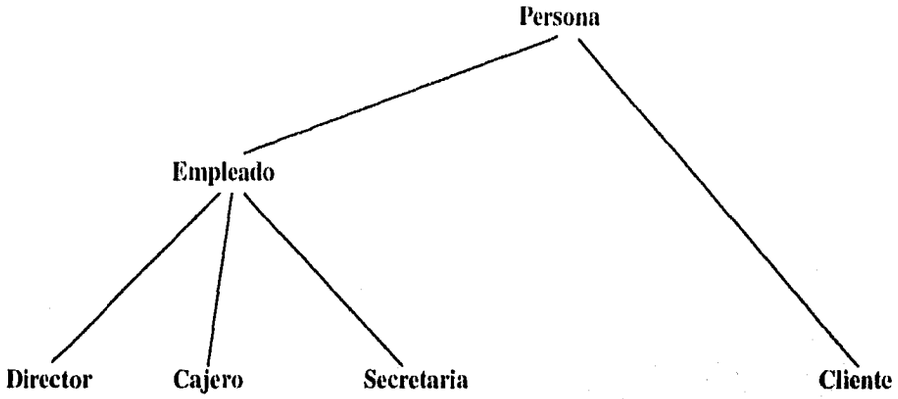
- Director: título, número-despacho, número-cuenta-de-gastos.

- Cajero:horas-por-semana, número-estación

- Secretaria: horas-por-semana, supervisor.

Normalmente, en sistemas orientados a objetos se elige la última forma. Es posible determinar el conjunto de todos los objetos de empleado en este caso

tomando la unión de aquellos objetos de empleado en este caso tomando la unión de aquellos objetos asociados a todas las clases en el subárbol con raíz en Empleado.



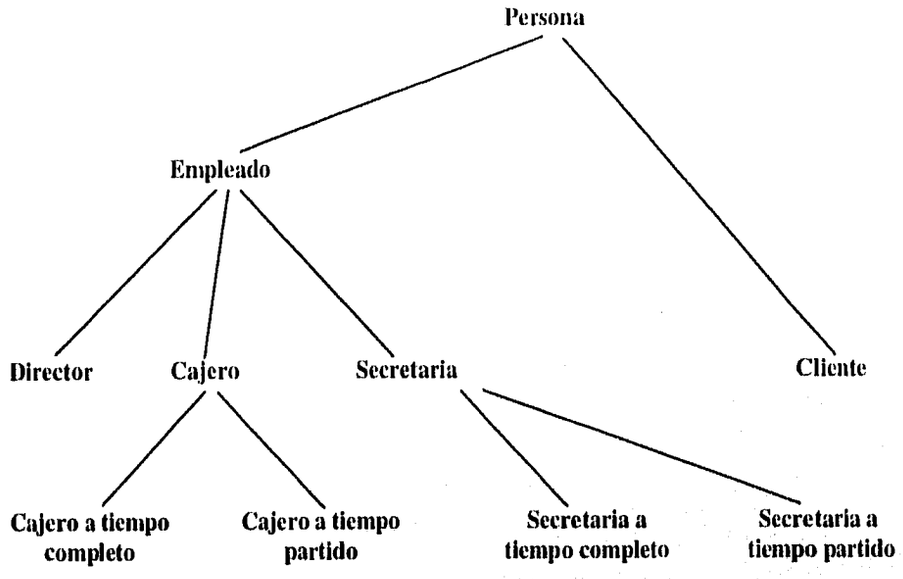
Jerarquìa de clases

Como observamos anteriormente, la jerarquía de clase/subclase es similar al concepto de especialización (la relación "ISA") en el modelo entidad-relación. Decimos que Cajero es una especialización de Empleado porque el conjunto de todos los cajeros es un subconjunto del conjunto de todos los empleados. Es decir, cada cajero es un empleado.

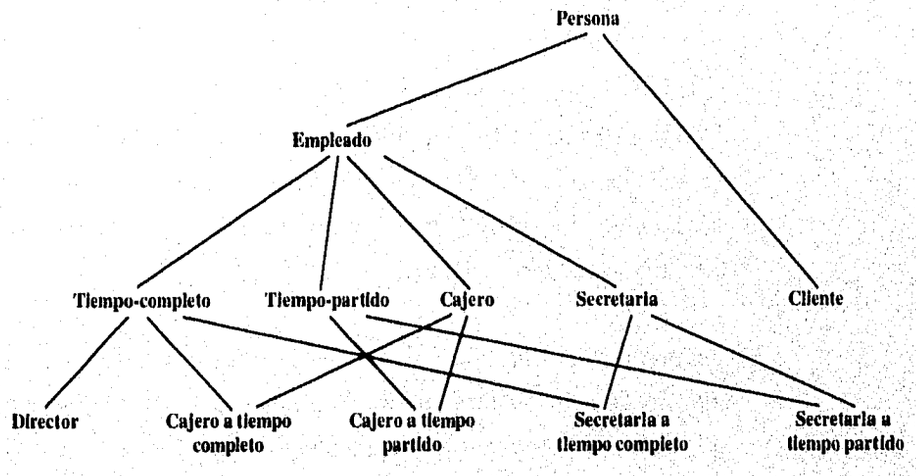
En la mayoría de lo sistemas orientados a objetos suelen tener implícita la especialización en vez de la generalización. Así, en lo que sigue, supondremos que la jerarquía de clase/subclase está basada en la especialización.

Herencia Múltiple

En la mayoría de los casos, una organización jerárquica de clases es adecuada para describir aplicaciones. En tales casos, todas las superclases de una clase son antepasados de otra en la jerarquía. Sin embargo, existen situaciones que no pueden representarse bien en una jerarquía de clases.



Jerarquìa de clases para empleados a tiempo completo y partido



Clase DAG

Identidad de Objetos

Los objetos en una Base de Datos orientada a objetos, normalmente corresponde a una entidad en la empresa que está modelando la Base de Datos. Una entidad conserva su identidad aun cuando algunas de sus propiedades cambien con el tiempo. Igualmente, un objeto conserva su identidad aun cuando algunos o todos los valores de las variables o las definiciones de los métodos cambien con el tiempo. Este concepto de identidad no se aplica a las tuplas de una Base de Datos relacional. En los sistemas relaciónales, las tuplas de una relación se distinguen únicamente por los valores que contienen.

La identidad de objeto es una noción más fuerte que la que se encuentra normalmente en los lenguajes de programación o en los modelos de datos que no están basados en la orientación a objetos. A continuación ilustramos varias formas de identidad.

- Valor. Se utiliza un valor de dato por identidad. Esta es la forma de identidad que se usa en los sistemas relaciónales.

- Nombre. Se utiliza un nombre facilitado por el usuario por identidad. Esta es la forma de identidad que normalmente se usa para las variables en los procedimientos. A cada variable se le da un nombre que identifica de manera única a la variable sin importar el valor que contenga.

- Incorporación. Una noción de identidad es incorporar en el modelo de datos el lenguaje de programación, y no se requiere que el usuario proporcione ningún identificador. Esta es la forma de identidad que se usa en los sistemas orientados a objetos.

Un tema relacionado con el tipo de identidad es la permanencia de identidad. una forma sencilla de lograr incorporar la identidad es promedio de punteros a localizaciones físicas en memoria. Sin embargo, la asociación de un objeto a una localización física en memoria puede cambiar con el tiempo. A continuación listamos varios grados de permanencia de identidad.

- Intraprograma. La identidad permanece solamente durante la ejecución de un único programa o consulta. Ejemplo de identidad de intraprograma son los nombres de variables en los lenguajes de programación son los nombres de variables en los lenguajes de programación y los identificadores de tuplas en sql.

- Interprograma. la identidad permanece de una ejecución del programa a otra. Ejemplo de identidad de interprograma son los nombres de relaciones en un lenguaje de consultas relacional del tipo del SQL.

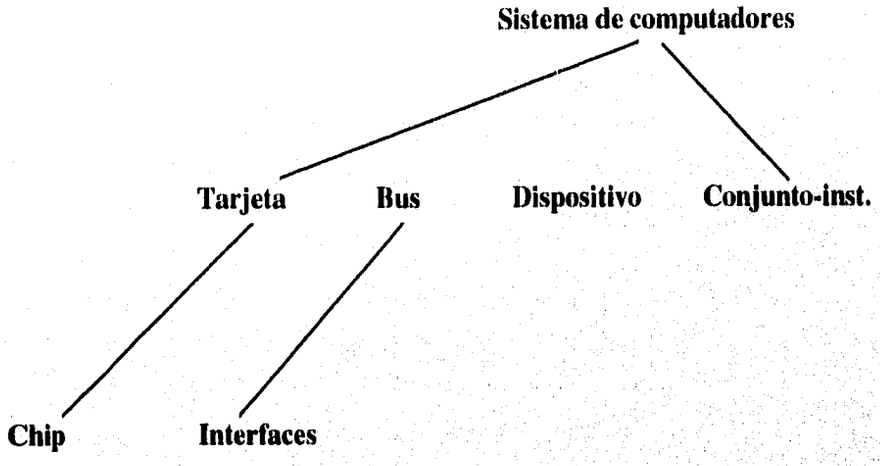
- Persistente. La identidad permanece no sólo entre las ejecuciones del programa sino también entre las reorganizaciones estructurales de los datos. las relaciones en SQL no tienen identidad persistente, ya que una reorganización de la Base de Datos puede resultar en un nuevo esquema de Base de Datos con relaciones con nuevos nombres. La forma persistente de identidad es la que se requiere en los sistemas orientados a objetos.

Estas formas de identidad sirven para distinguir entre identidad en sistemas orientados a objetos y punteros en organización física de datos. los punteros de la memoria principal o de la memoria virtual solamente ofrecen identidad de interprograma. Los punteros a datos de un sistema de archivos en disco solamente ofrecen identidad de interprograma. Así, la identidad de objetos es decir, la

identidad persistente es una noción más fuerte que la que proporcionan los punteros.

Contenido de Objetos

Anteriormente observamos que el valor de una variable de un objeto es ella misma un objeto. Esto crea una jerarquía de contenido entre los objetos. Un objeto 0_2 es el hijo de un objeto 0_1 SI 0_1 contiene a 0_2 ; es decir, 0_2 es decir 0_1.



Jerarquía de contenido para la base de datos de diseño de sistemas de computadoras

ANEXO B

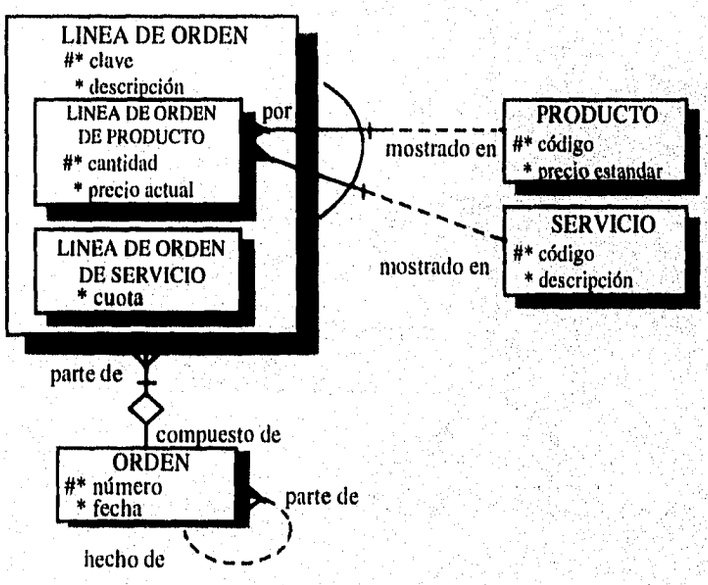
Esta sección presenta las convenciones del modelo conceptual de datos usados por muchas metodologías de ingeniería de información.

Tópicos

- ❑ Convenciones del Método *CASE
- ❑ Convenciones de Diagramas de Peter Chen
- ❑ Convenciones de la Metodología de Ingeniería de Información de James Martin

Convenciones del Método *CASE

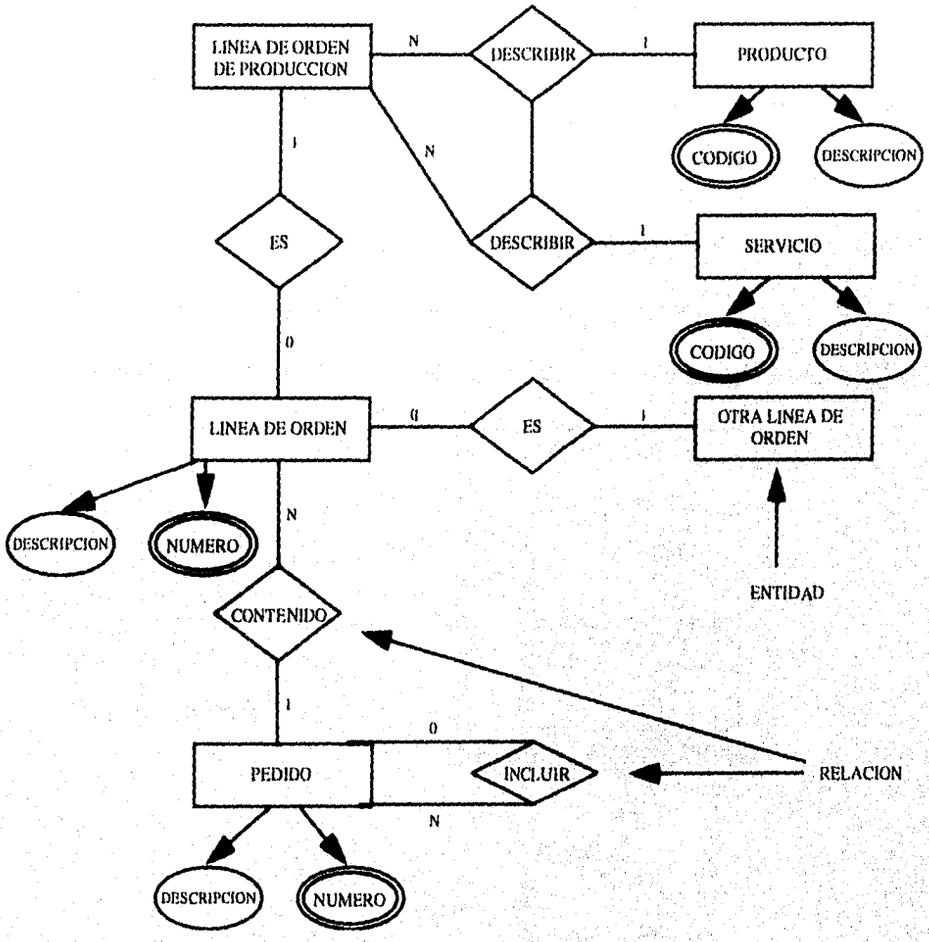
Este modelo E-R de una Orden y de una Línea de Orden de Producto, refleja las convenciones del modelo de datos presentadas en el capítulo II.



El diamante indica una relación no transferible, la transferencia es discutida en el apéndice "C".

Convenciones de diagramas de Peter Chen

Este diagrama muestra las convenciones del modelo de datos de Peter Chen, quien "inventó" el Modelo de Entidad-Relación en 1977.

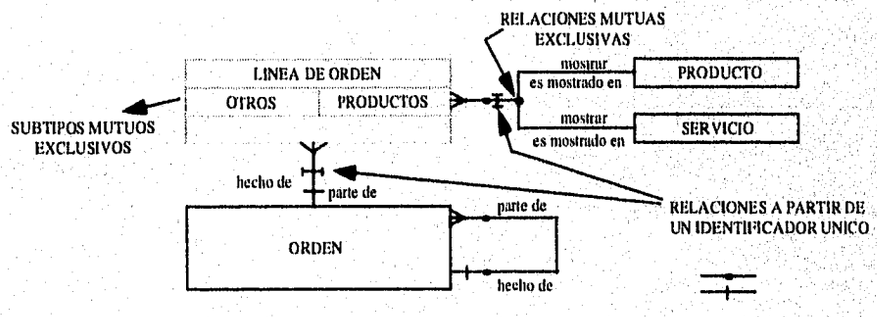


Convenciones del Modelo

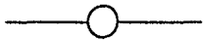
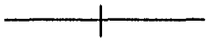
- ❑ Este método permite una relación entre dos, tres, o más entidades
- ❑ A una relación le pueden pertenecer muchos atributos. No todos los atributos son mostrados aquí.
- ❑ Los atributos son mostrados en elipses para la relación o para la entidad relevante.
- ❑ Los identificadores únicos son indicados por una elipse doble.
- ❑ Los subtipos son mostrados en una relación binaria de uno-a-uno.
- ❑ La exclusividad y la no-transferencia de las relaciones no son mostradas generalmente.

Convenciones de la Metodología de Ingeniería de Información de James Martin

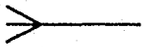
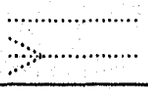
Las convenciones del modelo de datos de James Martin soportan muchos de los mismos conceptos de la Convención del Método*CASE.



Convenciones del Modelo

símbolo	Descripción
	Indica una relación opcional y está al lado contrario de lo que se mostró en el Método*CASE
	Indica un final de una relación

ESTILOS DIFERENTES DE REPRESENTAR LA CONECTIVIDAD EN UNDIAGRAMA ENTIDAD-RELACION

Conectividad	Crows Foot	Reiner	Chen
1:1			
M:1			
M:M			
Obligatoria			
Opcional			

ANEXO C

Este apéndice cubre adicionalmente Consideraciones del Modelo Conceptual de Datos que no son incluidos en al capitulo II.

Tópicos

- ❑ Reconocer Patrones Genéricos
- ❑ Asegurar un Modelo Perfecto y con Calidad
- ❑ Modelo de Ralación de Transferencia

Reconocer Patrones Genéricos

Producir un modelo simple, es el modelo más poderoso para reconocer entidades que tienen relaciones y/o atributos similares. Un modelo genérico combina patrones genéricos.

En un modelo E-R para una organización de una comunidad recreacional, puede incluir las siguientes entidades separadas (Fig. B.1).

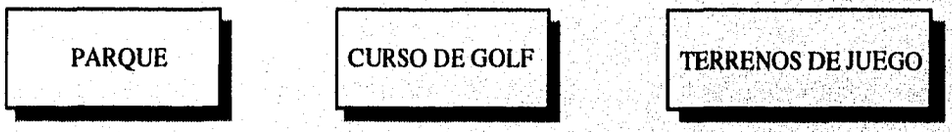


FIG. B.1 Entidades de una Comunidad Recreacional

Estas tres entidades podrían ser combinadas en una sola entidad genérica (Fig. B.2) o en una construcción subtipo/supertipo (Fig. B.3).

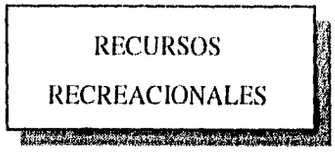


FIG. B.2 Entidad Genérica de una Comunidad Recreacional

o

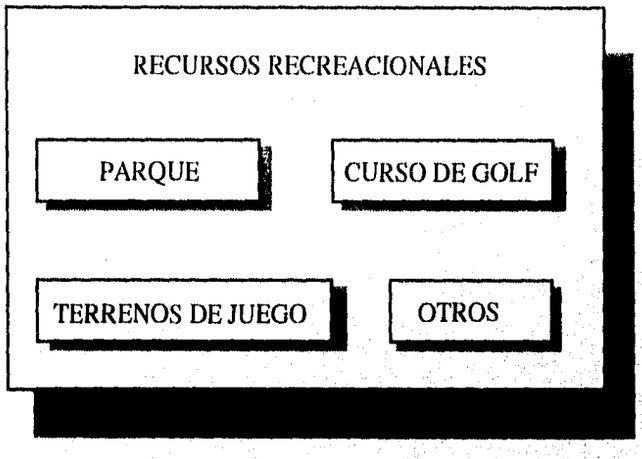


FIG. B.3 Construcción Subtipo/Supertipo de una Comunidad Recreacional

Un modelo genérico combina o conjunta relaciones y entidades similares.

Por ejemplo en la (Fig. B.4), ¿Se puede identificar un patrón genérico en este modelo? ¿Se ven atributos similares? ¿Se ve una relación similar?

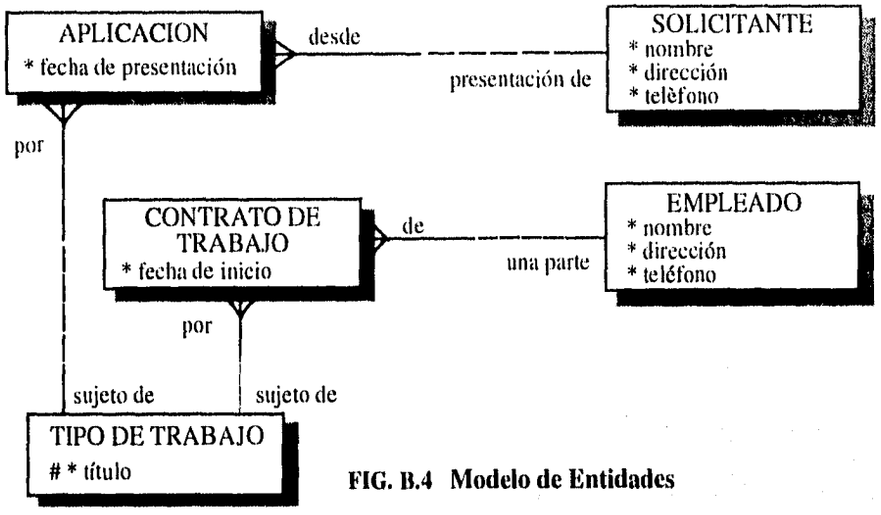


FIG. B.4 Modelo de Entidades

La APLICACION y el CONTRATO DE TRABAJO tienen fechas.

El SOLICITANTE y el EMPLEADO tienen atributos similares

Los siguientes modelos (Fig. B.5) resultan cuando se usan los patrones genéricos.

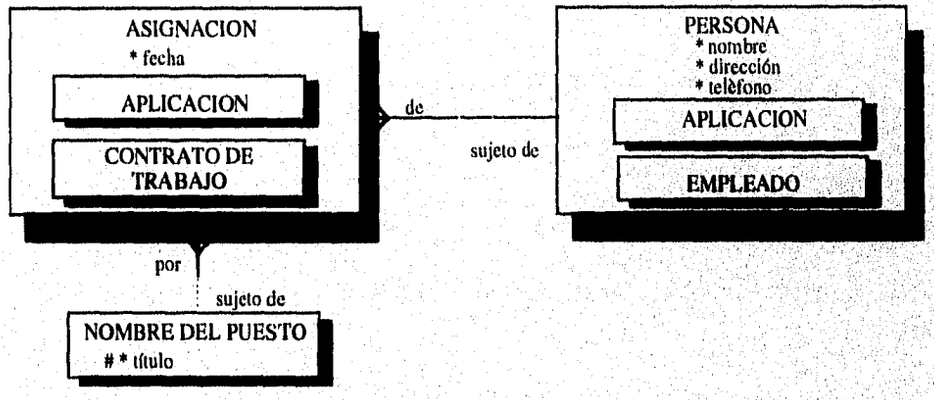


FIG. B.5 Patrones Genericos

Las APLICACIONES pueden ser hechas por los EMPLEADOS

Examinar de cerca un Modelo Genérico

- ¿Alguna capacidad puede ser perdida?
- ¿Alguna capacidad puede ser agregada?
- ¿Todas las relaciones y atributos existirían todavía?

FIG. B.5 Modelo Resultante Usando Patrones Genéricos

¡ Ser cuidadoso de no hacer un modelo excesivamente genérico !

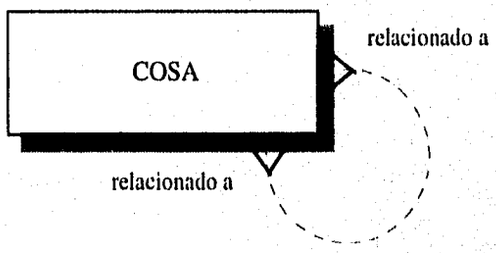


FIG. B.6 Modelo Excesivamente Genérico

¡ A la Vida, al Universo y a Todo

Recomendaciones:

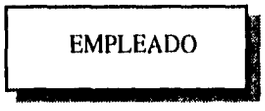
- Probar un nuevo modelo genérico en otro
- Estar dispuesto a regresar a un modelo genérico menor
- Mantener el entendimiento para cada cosa involucrada

Asegurar un Modelo Perfecto y con Calidad

Checar el modelo E-R para llegar a la perfección y a la calidad antes de mover el escenario del diseño de la base de datos.

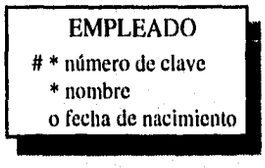
Dibujar Convenciones

Entidades



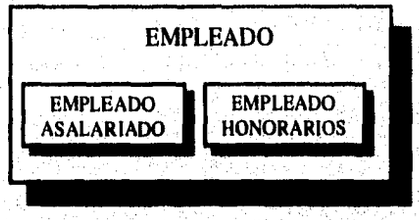
Dibujar una caja
Nombre en MAYUSCULAS
Nombre sinónimo opcional
Debe ser un UID

Atributos



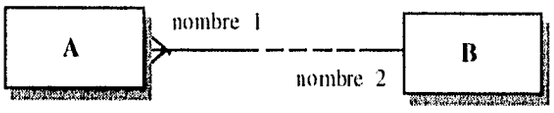
Nombre en minúsculas (el nombre de la entidad no)
Todos los atributos deben estar marcados
* parte del UID
* obligatorios
o opcional

Subtipos

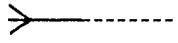
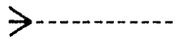
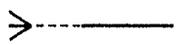
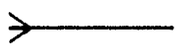
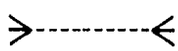
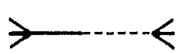
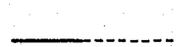
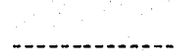


Descripción completa de las entidades
Una razón debe existir para subtipos

Relaciones

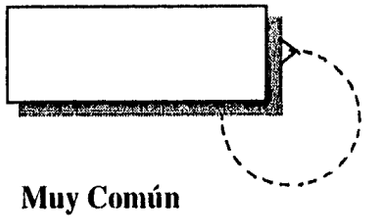


Cada relación debe tener:
nombre en minúsculas
grado
opcionalidad

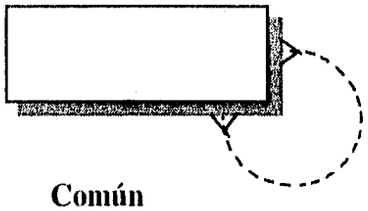
Entidad A y B	Relación	frecuencia
	A debe ser para una y sólo una B B puede tener una o más A	Muy común
	A puede ser para una y sólo una B B puede tener una o más A	Común
	A puede ser para una y sólo una B B debe tener una o más A	No común
	A debe ser para una y sólo una B B puede tener una o más A	Raro
	M:M ¿Esto se necesita resolver ?	Inicialmente
	M:M ¿Esto se necesita resolver ?	Inicialmente
	M:M y completamente obligatoria	Muy improbable
	1:1 ¿Esta es realmente una entidad	Rara
	1:1 ¿Esta es realmente una entidad	Rara
	1:1 y completamente obligatoria	Muy improbable

Checar si existe una relación recursiva válida

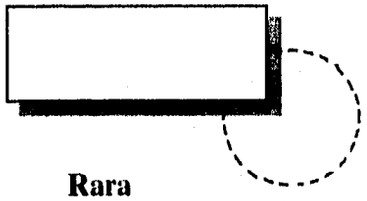
Relaciones Válidas



Muy Común

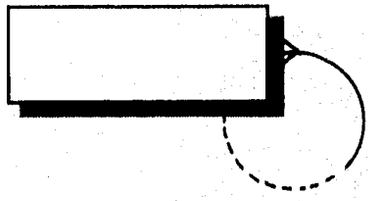


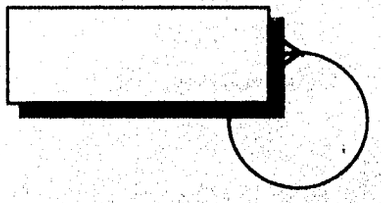
Común

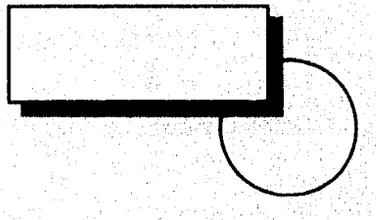


Rara

Relaciones Inválidas

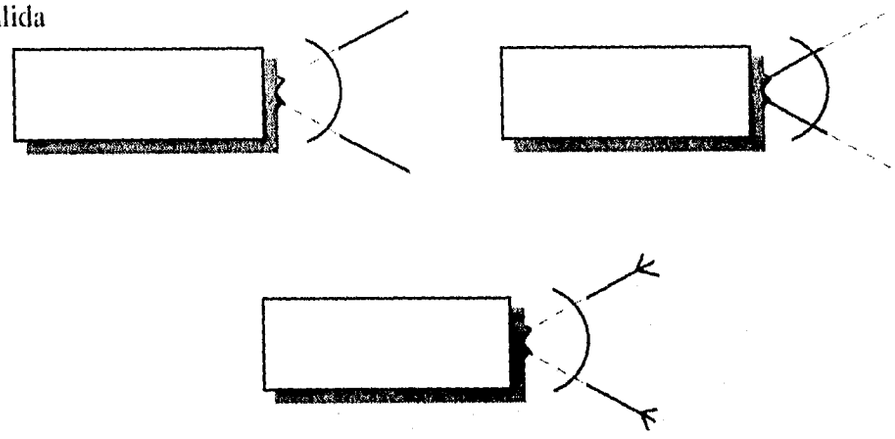




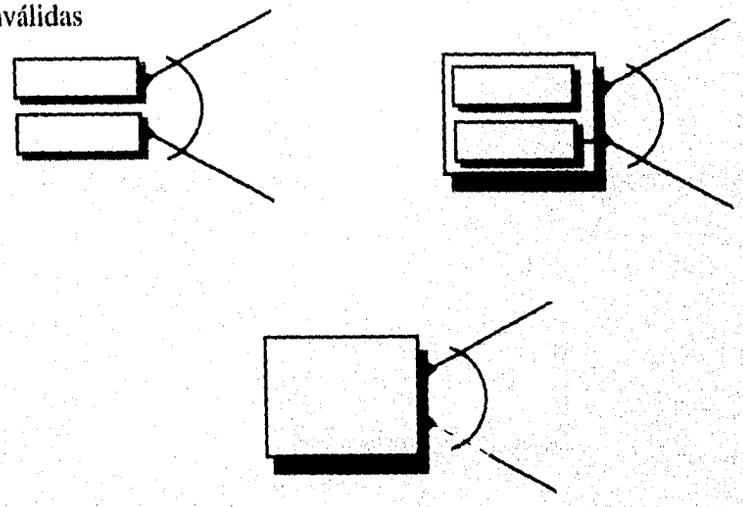


Checar si existe una relación exclusiva válida

Válida



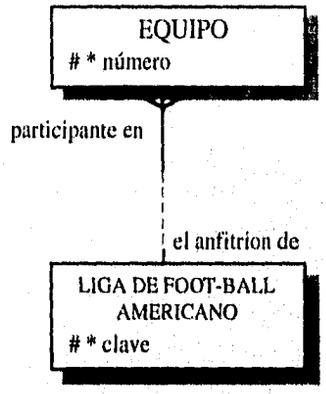
Inválidas



Modelo de Relación de transferencia

Una relación obligatoria normal es transferible, esta puede ser desconectada y reconectada

Un EQUIPO debe ser participante en una LIGA DE FOOT-BALL AMERICANO, pero un EQUIPO puede cambiar de una LIGA a otra.



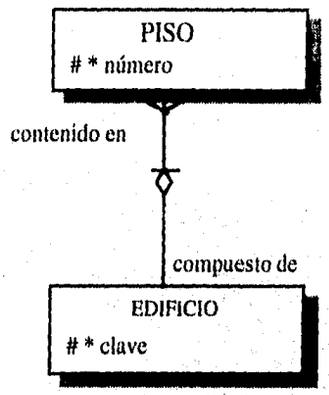
Recomendaciones:

- Si una relación es transferible, la entidad puede existir momentáneamente sin la entidad relacionada y puede ser relacionada con diferentes instancias de la entidad.
- Las relaciones son transferibles por default.
- Si el UID de la entidad está compuesto de una relación transferible, el UID tiene el potencial de cambiar cuando la instancia relacionada cambie. En tales casos, la relación debe ser desconectada y reconectada en la misma transacción.

El final de una relación es no-transferible, sí la instancia de la entidad no puede ser desconectada y reconectada en diferentes instancias de la entidad relacionada.

Un PISO no puede existir sin el EDIFICIO.

Un PISO no puede cambiar de EDIFICIO.



◊ - Significa que es una relación no-transferible.

Recomendaciones:

- Si la relación es no transferible, la entidad no puede existir sin la entidad relacionada y debe ser siempre relacionada a la misma instancia de la entidad relacionada.

ANEXO D

OPERADORES RELACIONALES

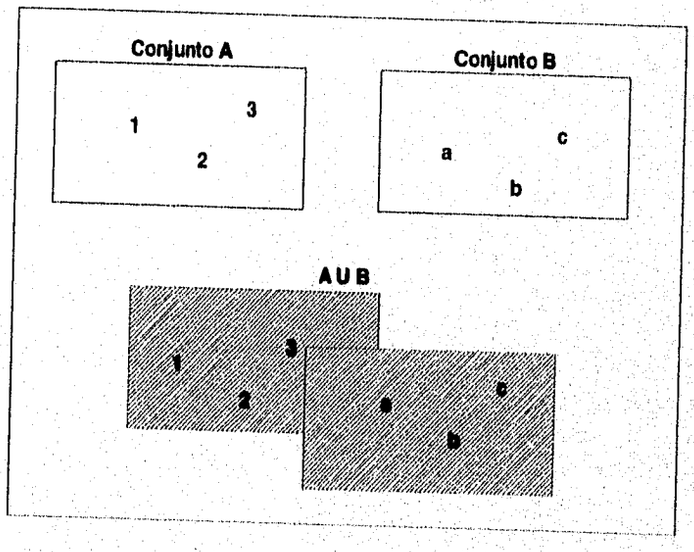
- **UNION**
- **INTERSECCION**
- **DIFERENCIA**
- **PROYECCION**
- **SELECCION**
- **JOIN**

El interés actual en el enfoque relacional se debe en gran medida al trabajo del Dr. E.F. Codd, quien en junio de 1970 publicó el articulo: "A Relational Model of Data for Large Shared Data Banks " (CACM 13, Num. 6).

A continuación presentaremos un resumen de la teoría matemática involucrada en el modelo relacional del Dr. Codd.

• UNION

El operador de UNION acepta como entrada dos tablas con las mismas columnas en el mismo orden, y produce como resultado todas las columnas y todos los renglones de ambas tablas. Si existe algún renglón con la misma información en ambas tablas, en la tabla que se genera de aplicar el operador de UNION ese renglón sólo aparece una vez.



EJEMPLO: Tenemos las siguientes tablas:

EMPLEADOS ANTIGUOS

NUMERO DE EMPLEADO	NOMBRE DEL EMPLEADO	SALARIO
1	PEDRO	12,000
2	LUIS	
3	FRANCISCO	36,000

EMPLEADOS ANTIGUOS

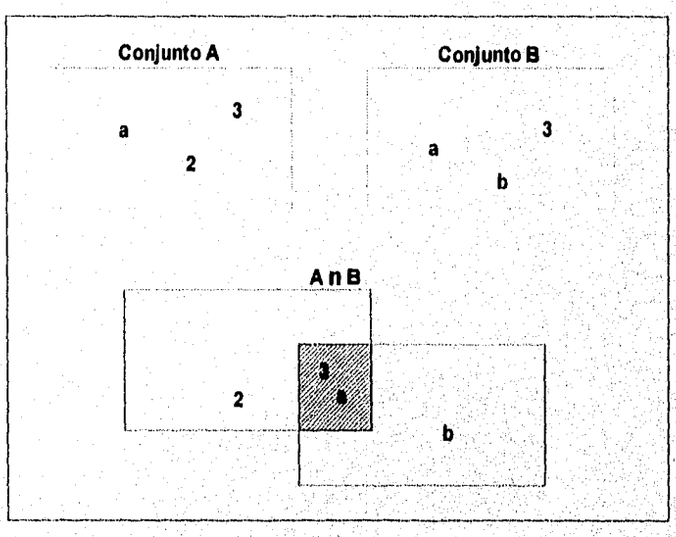
NUMERO DE EMPLEADO	NOMBRE OEL EMPLEADO	SALARIO
3	FRANCISCO	36,000
4	LORENA	24,000
5	GABRIELA	24,000

La UNION dará como resultado

NUMERO DE EMPLEADO	NOMBRE DEL EMPLEADO	SALARIO
1	PEORO	12,000
2	LUIS	
3	FRANCISCO	36,000
4	LORENA	24,000
5	GABRIELA	24,000

- **INTERSECCION**

El operador de INTERSECCION selecciona de ambas tablas los renglones que tengan exactamente la misma información en todas las columnas.



EJEMPLO: Tenemos las siguientes tablas:

EMPLEADOS ANTIGUOS

NUMERO DE EMPLEADO	NOMBRE DEL EMPLEADO	SALARIO
1	PEDRO	12,000
2	LUIS	
3	FRANCISCO	36,000

EMPLEADOS ANTIGUOS

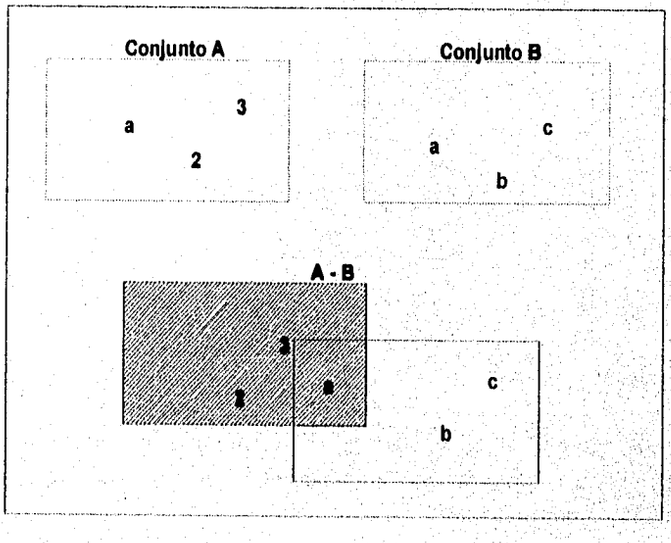
NUMERO DE EMPLEADO	NOMBRE DEL EMPLEADO	SALARIO
3	FRANCISCO	36,000
4	LORENA	24,000
5	GABRIELA	24,000

La INTERSECCION dará como resultado

NUMERO DE EMPLEADO	NOMBRE DEL EMPLEADO	SALARIO
3	FRANCISCO	36,000

- **DIFERENCIA**

El operador de DIFERENCIA acepta como entrada dos tablas que tengan al menos una columna en común, en donde la tabla resultante tendrá todas las columnas de la primer tabla y los renglones que no aparezcan en la segunda tabla



EJEMPLO: Tenemos las siguientes tablas:

DEPARTAMENTO

CODIGO DEPART.	NOMBRE DEL DEPARTAMENTO
VE	VENTAS
NO	NOMINA
IN	INVESTIGACION
ME	MERCADOTECNIA
RE	RESULTADOS

EMPLEADOS

NUMERO DE EMPLEADO	NOMBRE DEL EMPLEADO	SALARIO	CODIGO DEPART.
1	PEDRO	12,000	VE
2	LUIS		NO
3	FRANCISCO	36,000	
4	LORENA	24,000	
5	GABRIELA	24,000	NO

La DIFERENCIA dará como resultado

CODIGO DEPART.	NOMBRE DEL DEPARTAMENTO
IN	INVESTIGACION
ME	MERCADOTECNIA
RE	RESULTADOS

- **PROYECCION**

El operador de PROYECCION tiene como entrada una tabla, y produce como resultado sólo aquellas columna: especificadas por el usuario.

El orden en el cual aparecen las columnas, es el que se indica cuando se hace la proyección.

El número de columnas que se pueden proyectar es como máximo el mismo número de columnas de la tabla y como mínimo una sola columna.

EJEMPLO: Tenemos las siguiente tabla:

EMPLEADOS

NUMERO DE EMPLEADO	NOMBRE DEL EMPLEADO	SALARIO	CODIGO DEPART.
1	PEDRO	12,000	VE
2	LUIS		NO
3	FRANCISCO	36,000	
4	LORENA	24,000	
5	GABRIELA	24,000	NO

La PROYECCION de las columnas Nombre y Número empleado, *muestran el siguiente resultado*

NOMBRE DEL EMPLEADO	NUMERO DE EMPLEADO
PEDRO	1
LUIS	2
FRANCISCO	3
LORENA	4
GABRIELA	5

- **SELECCION**

El operador de SELECCION acepta una sola tabla como entrada, y produce como resultado las mismas columnas que contiene la tabla de entrada y los renglones que sean especificados por el usuario.

EJEMPLO: Tenemos las siguiente tabla:

EMPLEADOS

NUMERO DE EMPLEADO	NOMBRE DEL EMPLEADO	SALARIO	CODIGO DEPART.
1	PEDRO	12,000	VE
2	LUIS		NO
3	FRANCISCO	36,000	
4	LORENA	24,000	
5	GABRIELA	24,000	NO

La SELECCION de todos los empleados que trabajan en el departamento de Codigo igual a "NO" muestra el siguiente resultado

NUMERO DE EMPLEADO	NOMBRE DEL EMPLEADO	SALARIO	CODIDO DEPART.
2	LUIS		NO
5	GABRIELA	24,000	NO

Las condiciones de selección de renglones pueden ser de varios grados de complejidad y pueden incluir a los operadores booleanos AND, OR y NOT (se pueden utilizar paréntesis para indicar precedencia de operación).

Las comparaciones pueden realizarse con valores literales, valores contenidos en las columnas, o expresiones matemáticas que involucren valores literales de las columnas.

- **JOIN**

El operador de JOIN acepta como entrada dos o más tablas, teniendo cada una al menos una columna en común con las otras tablas, y produce como resultado a todas las columnas de las tablas de entrada, y los renglones se concatenan con aquellos renglones cuyos valores en las tablas de entrada cumplen la condición que indica el usuario para hacer el join.

Los operadores relaciónales para indicar las condiciones de join pueden ser >,<, =,!=. Las columnas en común sólo se muestran una vez.

EJEMPLO: Tenemos las siguientes tablas:

DEPARTAMENTO

CODIGO DEPART.	NOMBRE DEL DEPARTAMENTO
VE	VENTAS
NO	NOMINA
IN	INVESTIGACION
ME	MERCADOTECNIA
RE	RESULTADOS

EMPLEADOS

NUMERO DE EMPLEADO	NOMBRE DEL EMPLEADO	SALARIO	CODIGO DEPART.
1	PEDRO	12,000	VE
2	LUIS		NO
3	FRANCISCO	36,000	
4	LORENA	24,000	
5	GABRIELA	24,000	NO

El JOIN con la columna en común del Código de Departamento muestra el siguiente resultado

CODIGO DEPART.	NOMBRE DEL DEPARTAMENTO	NUMERO DE EMPLEADO	NOMBRE DEL EMPLEADO	SALARIO
VE	VENTAS	1	PEDRO	12,000
NO	NOMINA	2	LUIS	
NO	NOMINA	5	GABRIELA	24,000

ANEXO E

ORACLE

ORACLE es una compañía dedicada a ofrecer un rango completo de software y servicios basados en el Sistema Manejador de Bases de Datos Relaciónales ORACLE (RDBMS) ORACLE (RDBMS). ORACLE RDBMS es líder en el mercado con miles de instalaciones alrededor de todo el mundo y de diversos sectores en el mercado, todos ellos con el propósito de dar la mejor solución en estrategias de información dentro de diferentes negocios.

HISTORIA

La compañía empieza en 1977. En 1979 Lany Ellison y Bob Miner (dos de las personas más importantes dentro de la compañía actualmente) implementaron la idea del reporte técnico de IBM, el cual hablaba sobre el SQLR que fue el sistema de recurso humanos del cual surgió el lenguaje SQL. Ellos escribieron el primer SQL basado en una base de datos relacional. otra importante decisión fue tomada también en 1979, escribir el código en lenguaje C. Ellos tomaron la idea de IBM y la hicieron portable a través del lenguaje SQL, de hecho la máquina que probó dicho desarrollo fue una computadora DEC llamada PDF11. ORACLE. toma su nombre de un primer contrato, un proyecto confidencial para el gobierno, el password de ese proyecto de seguridad era precisamente ORACLE. Larry y Bo decidieron llamar a la compañía como dicho concurso. La compañía inicialmente se encontraba situada en Menlo Park. Actualmente se encuentra en San Francisco.

Hasta ahora la compañía se encuentra en 95 ciudades alrededor del mundo, cuenta con 7,500 empleados aproximadamente, además ocupa el primer lugar dentro del mercado de los RDBMS´s y el tercer lugar en ventas de software (un billón de dólares en ventas).

AQUITECTURA

La arquitectura cliente servidor explota al máximo la tecnología más avanzada, incluyendo aquellos sistemas que incluyen multiprocesadores (SMP´s). La base de datos distribuida ORACLE permite accesar en una sola consulta, datos que se encuentran en servidores remotos.

La arquitectura abierta de ORACLE provee una transparencia de acceso a los datos de otras bases de datos relaciónales como DB2 y SQL/DS de IBM, e incluso aquellas que no son relaciónales, como DABLE de ASHTON TATE.

Las aplicaciones del mundo real demandan el acceso concurrente de los datos críticos, con la mayoría de los sistemas de bases de datos los sistemas pueden convertirse en una barrera de contención con el desempeño limitado, no precisamente por la capacidad del CPU o lecturas y escrituras sobre disco, sino por los usuarios que tienen que esperar el acceso a los datos ORACLE utiliza un bloqueo a nivel registro irrestringido y consultas libres de contención , que minimizan y en algunos casos eliminan completamente tiempos de espera.

CARACTERISTICAS

La filosofía de ORACLE involucra diversos conceptos, como son: COMPATIBILIDAD, PORTABILIDAD, CONECTIVIDAD, CAPACIDAD Y PRODUCTIVIDAD. La compatibilidad es hablar de una gama de productos, los cuales se rigen bajo los estándares de la industria de las bases de datos relaciónales, lo que produce que una base de datos ORACLE sea compatible con otras Bases de

Datos existentes en el mercado. Una aplicación de ORACLE puede ser transportada dé una máquina a otra, con diferentes sistemas operativos y esto no provoca ningún cambio o modificación a dicha aplicación, esto habla de la portabilidad que ofrece ORACLE. La arquitectura de ORACLE permite que los datos y las aplicaciones residan en diferentes computadoras, plataformas, sistemas operativos y ambientes de red, la idea es tener una conectividad completa. La capacidad de manejo de grandes volúmenes de información facilita el uso de aplicaciones grandes. Además ORACLE cuenta con diversas herramientas y productos que apoyan en la toma de decisiones para lo cual se cumple la productividad.

ORACLE7

Con la tecnología de servidores cooperativos de ORACLE7, se puede aprovechar al máximo los beneficios de los sistemas relaciónales abiertos para todas sus aplicaciones. El hecho que ORACLE7 hace posible compartir los datos residentes en diferentes servidores en forma transparente. La confiabilidad, la escalabilidad, el bajo costo y la flexibilidad de un sistema abierto se logran mediante operaciones eficientes, aplicaciones activas y una integración total.

Con ORACLE7 tenemos implementada la arquitectura de servidores multiconectados, control de concurrencia, alta disponibilidad, restricciones de integridad, lenguaje procedural, procedimientos guardados, triggers a nivel Base de Datos, seguridad más controlada, itegración a través de redes, Bases de Datos distribuidas.

ANEXO F

CAPITULO XVII

DESCANSO, VACACIONES, Y PERMISOS

CLAUSULA 138. Serán considerados como días de descanso obligatorio con goce de salario, los siguientes: 1o. de enero, 5 de febrero, 21 de marzo, 1o.de mayo, 16 de Septiembre, 20 de noviembre, 25 de diciembre, 1o. de diciembre cuando corresponda a la transmisión del Poder Ejecutivo Federal, y el que determinen las leyes federales y locales electorales en el caso de elecciones ordinarias, para efectuar la jornada electoral.

CLAUSULA 139. Se consideran días festivos con goce de salario, los siguientes: 18 marzo, jueves, viernes y sábado de la semana de primavera, 5 de mayo, 12 de octubre y 1o. y 2 de noviembre. Estos días festivos podrán variarse cuando las partes convengan en ello.

TRABAJADORES DE PLANTA SINDICALIZADOS

CLAUSULA 140. Los trabajadores de planta sindicalizados que tengan de 1 a 9 años y 364 días de servicios tendrán derecho a 21 días laborables por concepto de vacaciones y los que hayan cumplido 10 años o más, tendrán derecho a 30 días laborables por el mismo concepto.

Para este efecto el día siguiente a los cinco de trabajo consecutivo, será considerado como descanso contractual, y tendrá el carácter de hábil para efectuar el cómputo de éstas.

CLAUSULA 141. Los trabajadores transitorios sindicalizados los mismos puntos que los trabajadores de planta sindicalizados.

Cuando el trabajador temporal haya prestado servicios durante 275 días o más en un lapso de 1 año, tendrá derecho a que se le otorguen vacaciones de 21 o 30 días laborables. Si son menos días a lo estipulado, se concederán en forma proporcional al tiempo en que hubiere disfrutado de salario durante el ciclo en cumplimiento al Artículo 77 de la ley Federal del trabajador.

- El perìodo de vacaciones que se conceda a los trabajadores transitorios, se considerará como tiempo trabajado para los efectos de cómputo del nuevo ciclo.

- Cuando el trabajador transitorio adquiera el carácter de planta, los períodos a que se refiere el párrafo anterior, se computarán dentro de la antigüedad general de empresa.

CLAUSULA 142. Las vacaciones se contarán por días laborables y el período para disfrutarlas deberá iniciarse precisamente en el primer día hábil que siga al de descanso semanal, debiendo boletinarse la fecha en que tome sus vacaciones cada trabajador de planta mediante acuerdo entre el sindicato y el patrón.

- Los trabajadores de turno, sus vacaciones se iniciarán precisamente el día posterior al de su descanso semanal, aún cuando dicho día posterior coincida con un día festivo o de descanso obligatorio, días éstos que no se estimarán como laborables para establecer el período vacacional.

- Los programas de vacaciones serán formulados anualmente; en la inteligencia de que, para fijar la fechas en que los trabajadores deban disfrutarlas, se tomarán como base las de ingreso en aquellos casos en que durante los años anteriores las hayan disfrutado en tales aniversarios, o en su defecto, las fechas con que aparezcan mencionadas en los programas de vacaciones de los años de vigencia del contrato anterior.

- Las fechas fijadas en el boletín podrán anticiparse o posponerse hasta por 90 días, siempre que haya mutuo acuerdo entre el patrón y el sindicato, y dado el caso de que se anticipen o pospongan las vacaciones, este hecho no significará que los períodos de vacaciones que correspondan a los años subsecuentes, deban ser anteriores o posteriores a las fechas fijadas en el boletín sino que seguirán señalándose en la misma forma.

- Las vacaciones no serán acomulables ni permutables por gratificación o salario, salvo el caso de terminación o rescisión del contrato.

- El boletín o programa de vacaciones deberá darse a conocer a los trabajadores precisamente con 30 días de anticipación a la fecha en que entre en vigor.

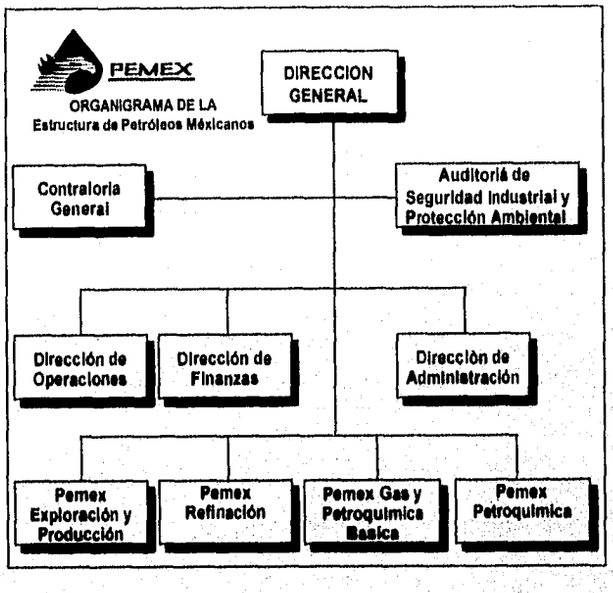
CLAUSULA 143. Las vacaciones se concederán aún cuando los servicios del trabajador de planta sindicalizado hayan tenido interrupciones con tal de que no excedan éstas de 90 días en el año, salvo en casos de enfermedades o accidentes, los trabajadores que conforme a este contrato desempeñen comisiones sindicales con goce de salario, y todos aquellos en que los trabajadores hayan faltado sin su culpa.

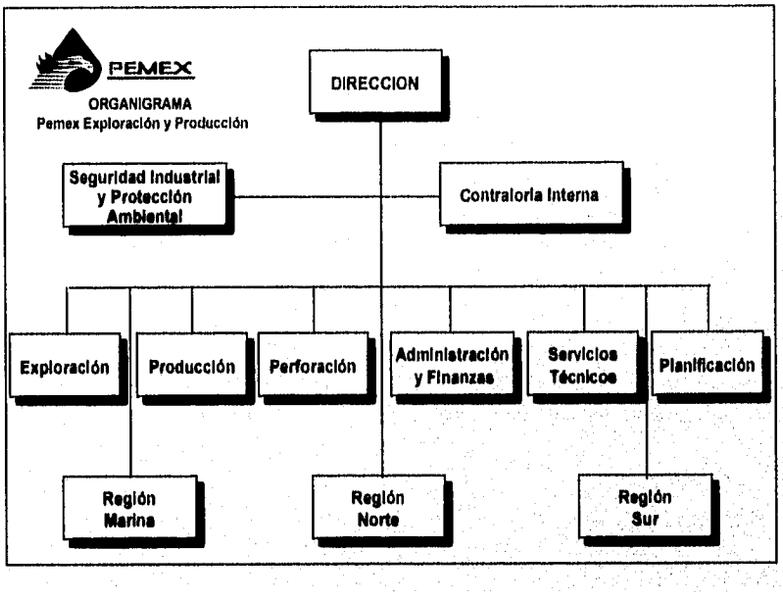
- Cuando las interrupciones excedan en el caso de que se trata, de 90 días en el año, las vacaciones se concederán en forma proporcional a los días en que el trabajador hubiera percibido salario, sin que el procedimiento modifique el ciclo original, el cual se tendrá por inamovible, salvo que el trabajador opte porque se le posponga, a efecto de que pueda disfrutar vacaciones completas en los términos del párrafo anterior. En estos casos y para el efecto de las vacaciones subsecuentes, se tendrá por modificado el ciclo original, considerándose como nuevo aniversario de vacaciones del trabajador, la fecha en que las tome.

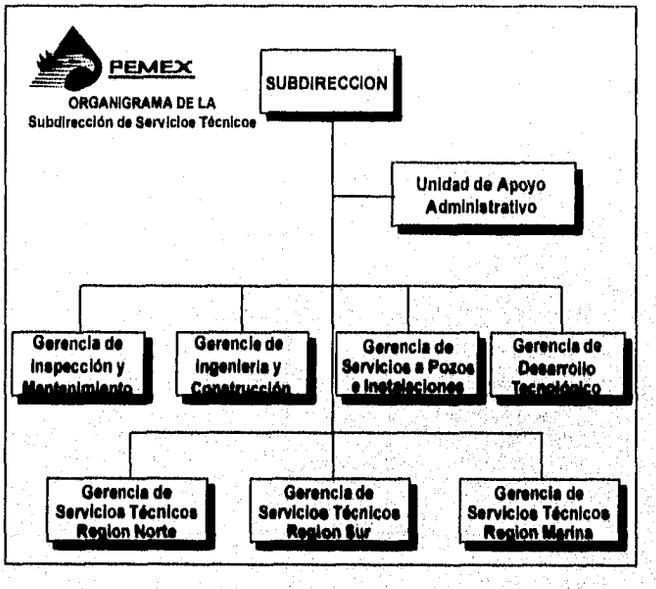
CLAUSULA 144 Cuando un trabajador sindicalizado se encuentre en vacaciones y en el transcurso de ellas sufra algún accidente o enfermedad no profesional que lo incapacite para seguir gozando de las vacaciones, quedarán suspendidas éstas, reanudándose cuando el trabajador se encuentre en condiciones de salud para seguirlas disfrutando, a juicio del médico o del patrón.

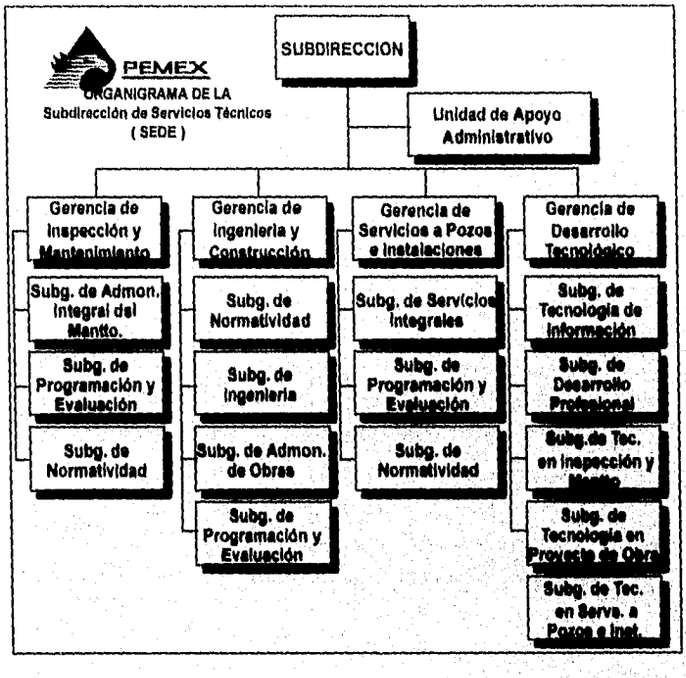
- En concordancia con lo establecido anteriormente en los casos de rescisión de contrato de trabajo, se pagarán las vacaciones por la parte proporcional correspondiente al tiempo laborado o que haya disfrutado de salarios. Tratándose de trabajadores de planta, en los casos de fallecimiento, jubilación o separación del servicio por enfermedad, las vacaciones se concederán completas si dentro del ciclo correspondiente el trabajador ha percibido salarios durante seis meses o más. Si en dichos casos el trabajador de planta hubiera percibido salarios por menos de 6 meses, se le liquidará el importe del 50% de las vacaciones completas.

ANEXO G


PEMEX
ORGANIGRAMA DE LA
Estructura de Petróleos Mexicanos
DIRECCION GENERAL
Contraloria General
Auditoriá de Seguridad Industrial y Protección Ambiental
Dirección de Operaciones
Dirección de Finanzas
Dirección de Administración
Pemex Exploración y Producción
Pemex Refinación
Pemex Gas y Petroquimica Basica
Pemex Petroquimica


PEMEX
ORGANIGRAMA
Pemex Exploración y Producción
DIRECCION
Seguridad Industrial y Protección Ambiental
Contraloría Interna
Exploración
Producción
Perforación
Administración y Finanzas
Servicios Técnicos
Planificación
Región Marina
Región Norte
Región Sur


PEMEX
ORGANIGRAMA DE LA
Subdirección de Servicios Técnicos
SUBDIRECCION
Unidad de Apoyo Administrativo
Gerencia de Inspección y Mantenimiento
Gerencia de Ingeniería y Construcción
Gerencia de Servicios a Pozos e Instalaciones
Gerencia de Desarrollo Tecnológico
Gerencia de Servicios Técnicos Region Norte
Gerencia de Servicios Técnicos Region Sur
Gerencia de Servicios Técnicos Region Marina

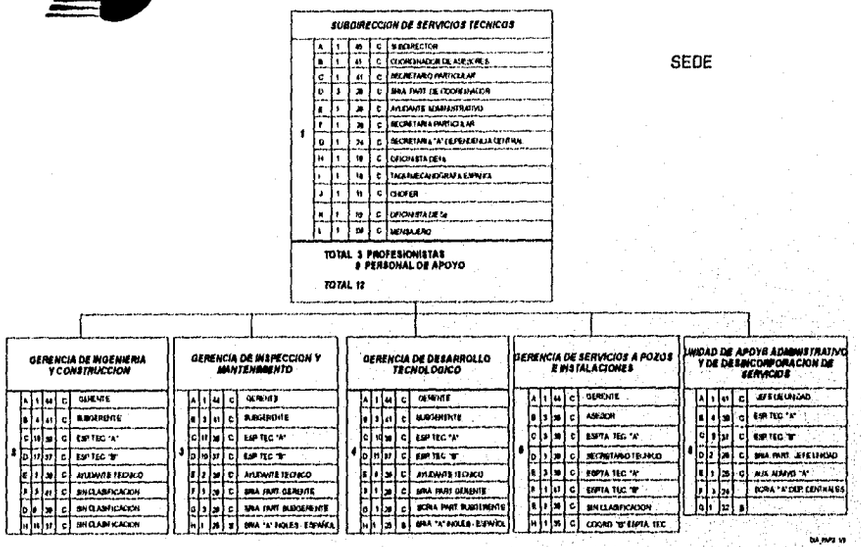

PEMEX
ORGANIGRAMA DE LA
Subdirección de Servicios Técnicos
(SEDE)
SUBDIRECCION
Unidad de Apoyo Administrativo
Gerencia de Inspección y Mantenimiento
Subg. de Admon. Integral del Mantto.
Subg. de Programación y Evaluación
Subg. de Normatividad
Gerencia de Ingeniería y Construcción
Subg. de Normatividad
Subg. de Ingeniería
Subg. de Admon. de Obras
Subg. de Programación y Evaluación
Gerencia de Servicios a Pozos e Instalaciones
Subg. de Servicios Integrales
Subg. de Programación y Evaluación
Subg. de Normatividad
Gerencia de Desarrollo Tecnológico
Subg. de Tecnología de Información
Subg. de Desarrollo Profesional
Subg. de Tec. en Inspección y Mantto.
Subg. de Tecnología en Proyectos de Obra
Subg. de Tec. en Servs. a Pozos e Inst.

PEMEX-EXPLORACION Y PRODUCCION

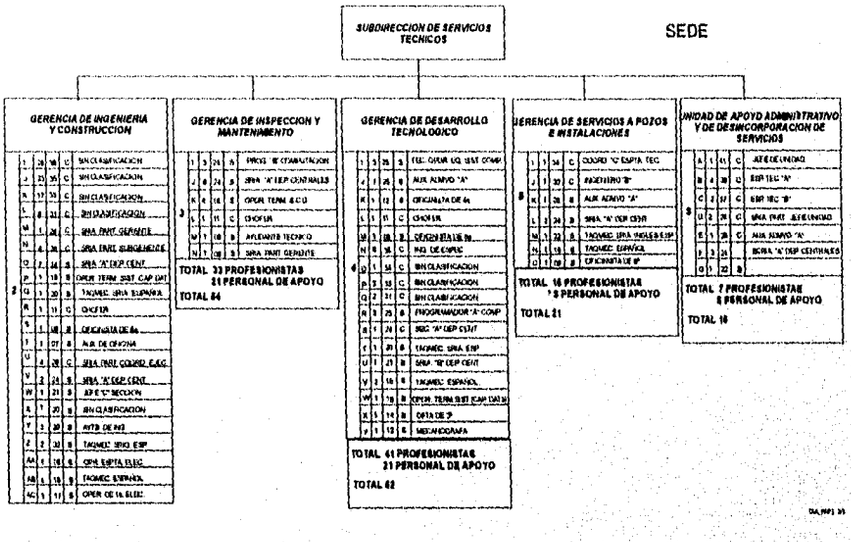
SUBDIRECCION DE SERVICIOS TECNICOS

ORGANIZACION ACTUAL REAL

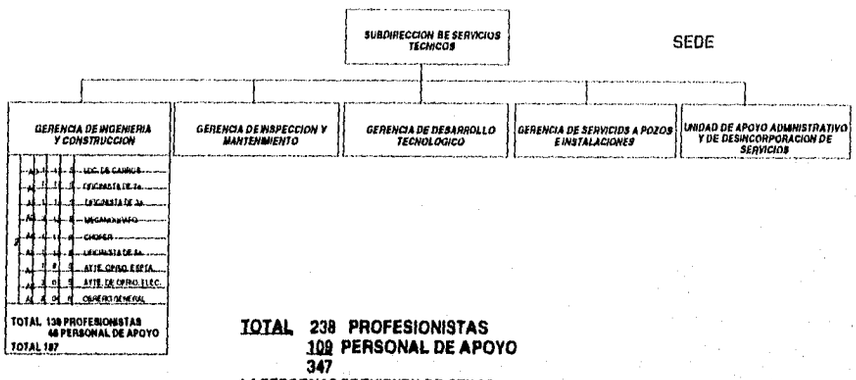
SEDE



PEMEX-EXPLORACION Y PRODUCCION
SUBDIRECCION DE SERVICIOS TECNICOS
ORGANIZACION ACTUAL REAL



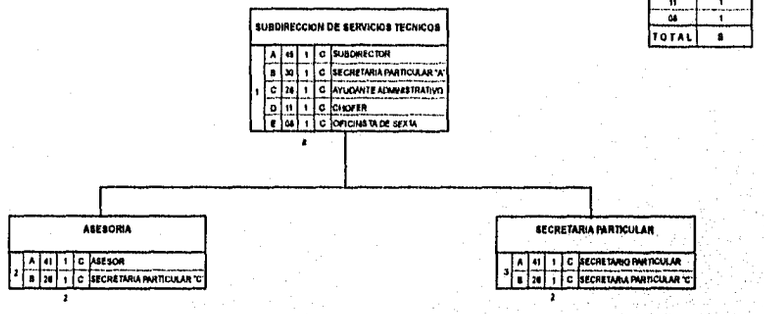
PEMEX-EXPLORACION Y PRODUCCION
SUBDIRECCION DE SERVICIOS TECNICOS
ORGANIZACION ACTUAL REAL



TOTAL 238 PROFESIONISTAS
109 PERSONAL DE APOYO
347
*** 4 PERSONAS PROVIENEN DE OTRAS SUBDIRECCIONES (NO CONTABILIZADOS)**

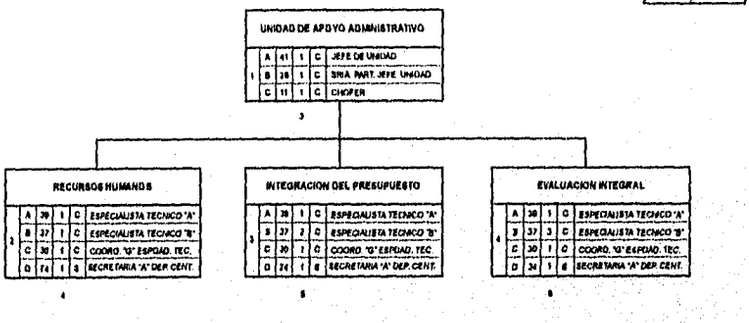
PEMEX EXPLORACION Y PRODUCCION
SUBDIRECCION DE SERVICIOS TECNICOS
ORDANOGRAMA ESTRUCTURAL Y DE PUESTOS
OFICINA DE LA SUBDIRECCION

NIVEL	CANTIDAD
46	1
41	2
30	1
28	1
26	2
11	1
08	1
TOTAL	8



PEMEX EXPLORACION Y PRODUCCION
SUBDIRECCION DE SERVICIOS TECNICOS
ORGANOGRAMA ESTRUCTURAL Y DE PUESTOS
UNIDAD DE APOYO ADMINISTRATIVO

NIVEL	CANTIDAD
41	1
38	3
37	8
30	3
28	1
24	3
11	1
TOTAL	18



PEMEX EXPLORACION Y PRODUCCION
SUBDIRECCION DE SERVICIOS TECNICOS
ORGANOGRAMA ESTRUCTURAL Y DE PUESTOS
GERENCIA DE SERVICIOS A POZOS E INSTALACIONES

NIVEL	CANTIDAD
44	1
41	3
38	10
37	8
36	1
32	2
28	2
24	3
24	1
11	1
08	4
TOTAL	36

GERENCIA DE SERVICIOS A POZOS E INSTALACIONES

1	A	44	1	C	GERENTE
	B	38	4	C	AYUDANTE TECNICO
	C	28	1	C	SRIA. PART. GERENTE
	D	24	1	C	SECRETARIA "A" DEP. CENT.
	E	11	1	C	CHOFER
	F	08	1	C	OFICINISTA DE SEXTA

6

SUBGERENCIA DE SERVICIOS INTEGRALES

3	A	41	1	C	SUBGERENTE
	B	38	3	C	ESPECIALISTA TECNICO "A"
	C	37	3	C	ESPECIALISTA TECNICO "B"
	D	33	1	C	COORD. "E" ESPDAD. TEC.
	E	28	1	C	SRIA. PART. SUBGERENTE
	F	08	1	S	OFICINISTA DE SEXTA

10

SUBGERENCIA DE PROGRAMACION Y EVALUACION

3	A	41	1	C	SUBGERENTE
	B	38	4	C	ESPECIALISTA TECNICO "A"
	C	37	2	C	ESPECIALISTA TECNICO "B"
	D	28	1	C	TECNICO EN INFORMATICA
	E	28	1	C	SRIA. PART. SUBGERENTE
	F	08	1	S	OFICINISTA DE SEXTA

10

SUBGERENCIA DE NORMATIVIDAD

4	A	41	1	C	SUBGERENTE
	B	38	3	C	ESPECIALISTA TECNICO "A"
	C	37	3	C	ESPECIALISTA TECNICO "B"
	D	32	1	C	COORD. "E" ESPDAD. TEC.
	E	28	1	C	SRIA. PART. SUBGERENTE
	F	08	1	S	OFICINISTA DE SEXTA

10

1

PEMEX EXPLORACION Y PRODUCCION
SUBDIRECCION DE SERVICIOS TECNICOS
ORGANOGRAMA ESTRUCTURAL Y DE PUESTOS
GERENCIA DE INSPECCION Y MANTENIMIENTO

NIVEL	CANTIDAD
44	1
41	3
38	15
37	6
36	1
32	2
24	1
26	3
24	7
11	1
08	1
TOTAL	41

GERENCIA DE INSPECCION Y MANTENIMIENTO

1	A	44	1	C	GERENTE
	B	38	1	C	AYUDANTE TECNICO
	C	28	1	C	SRIA. PART. GERENTE
	D	24	1	C	SECRETARIA "A" DEP. CENT.
	E	11	1	C	CHOFER
	F	08	1	C	OFICINISTA DE SEXTA

6

SUBGERENCIA DE ADMINISTRACION INTEGRAL DEL MANTENIMIENTO

2	A	41	1	C	SUBGERENTE
	B	38	6	C	ESPECIALISTA TECNICO "A"
	C	37	2	C	ESPECIALISTA TECNICO "B"
	D	28	1	D	SRIA. PART. SUBGERENTE
	E	24	3	B	SECRETARIA "A" DEP. CENT.

11

SUBGERENCIA DE PROGRAMACION Y EVALUACION

3	A	41	1	C	SUBGERENTE
	B	38	8	C	ESPECIALISTA TECNICO "A"
	C	37	3	C	ESPECIALISTA TECNICO "B"
	G	32	1	C	COORD. "E" ESPDAD. TEC.
	H	28	1	D	SRIA. PART. SUBGERENTE
	F	24	2	B	SECRETARIA "A" DEP. CENT.

13

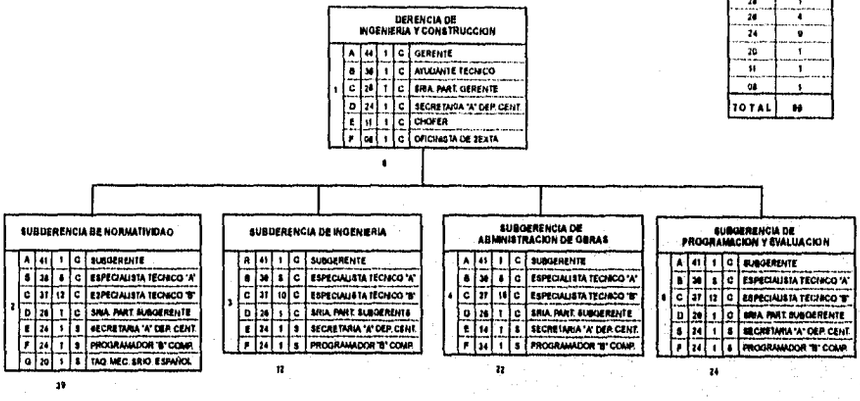
SUBGERENCIA DE NORMATIVIDAD

4	A	41	1	C	SUBGERENTE
	B	38	8	C	ESPECIALISTA TECNICO "A"
	C	37	3	C	ESPECIALISTA TECNICO "B"
	D	32	1	C	COORD. "E" ESPDAD. TEC.
	E	28	1	C	SRIA. PART. SUBGERENTE
	F	24	2	B	SECRETARIA "A" DEP. CENT.

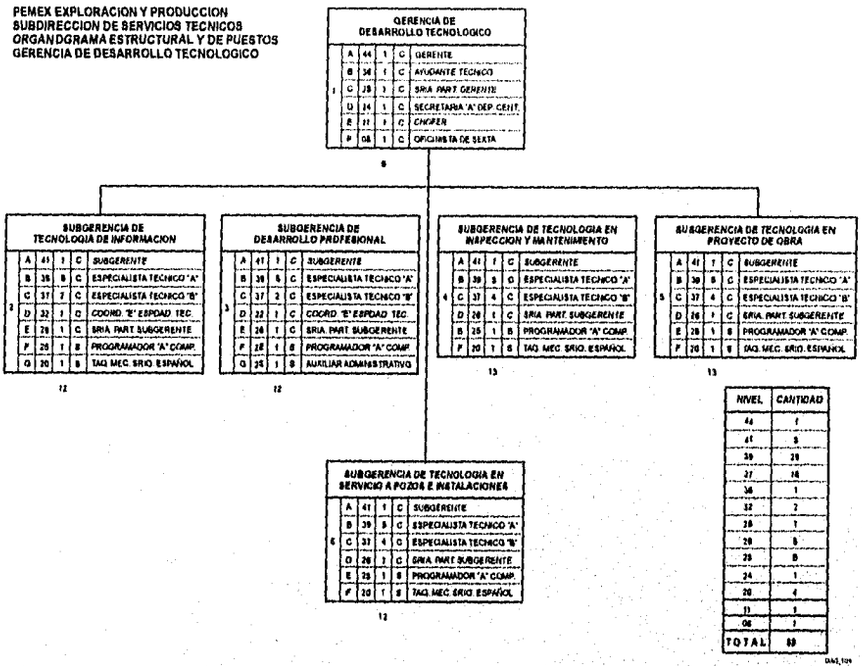
12

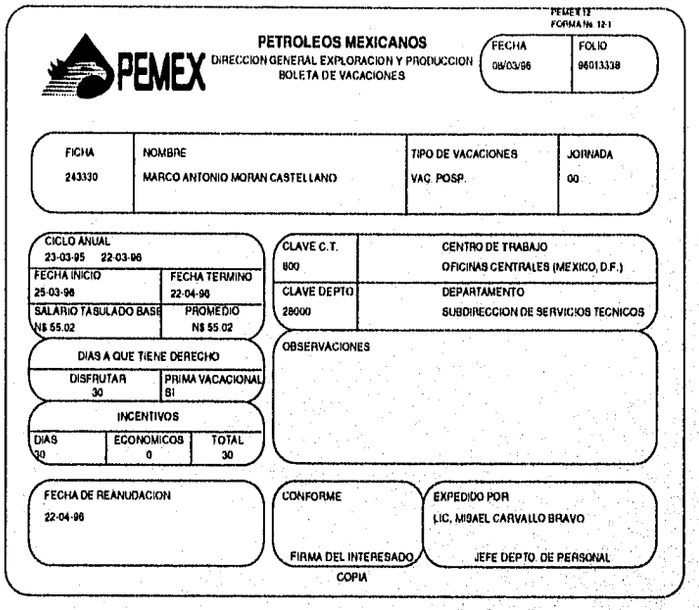
PEMEX EXPLORACION Y PRODUCCION
SUBDIRECCION DE SERVICIOS TECNICOS
ORGANOGRAMA ESTRUCTURAL Y DE PUESTOS
GERENCIA DE INGENIERIA Y CONSTRUCCION

NIVEL	CANTIDAD
44	1
41	4
39	32
37	44
36	1
28	1
26	4
24	9
20	1
11	1
08	1
TOTAL	99



PEMEX EXPLORACION Y PRODUCCION
SUBDIRECCION DE SERVICIOS TECNICOS
ORGANIGRAMA ESTRUCTURAL Y DE PUESTOS
GERENCIA DE DESARROLLO TECNOLOGICO





PEMEX 12
FORMA No. 12-1

PETROLEOS MEXICANOS
DIRECCION GENERAL EXPLORACION Y PRODUCCION
BOLETA DE VACACIONES

FECHA	FOLIO
08/03/96	96013338

FICHA	NOMBRE	TIPO DE VACACIONES	JORNADA
243330	MARCO ANTONIO MORAN CASTELLANO	VAC. POSP.	00

CICLO ANUAL
23-03-95 22-03-96

FECHA INICIO	FECHA TERMINO
25-03-96	22-04-96

SALARIO TABULADO BASE	PROMEDIO
N$ 55.02	N$ 55.02

DIAS A QUE TIENE DERECHO

DISFRUTAR	PRIMA VACACIONAL
30	SI

INCENTIVOS

DIAS	ECONOMICOS	TOTAL
30	0	30

CLAVE C.T.	CENTRO DE TRABAJO
800	OFICINAS CENTRALES (MEXICO, D.F.)
CLAVE DEPTO	DEPARTAMENTO
28000	SUBDIRECCION DE SERVICIOS TECNICOS

OBSERVACIONES

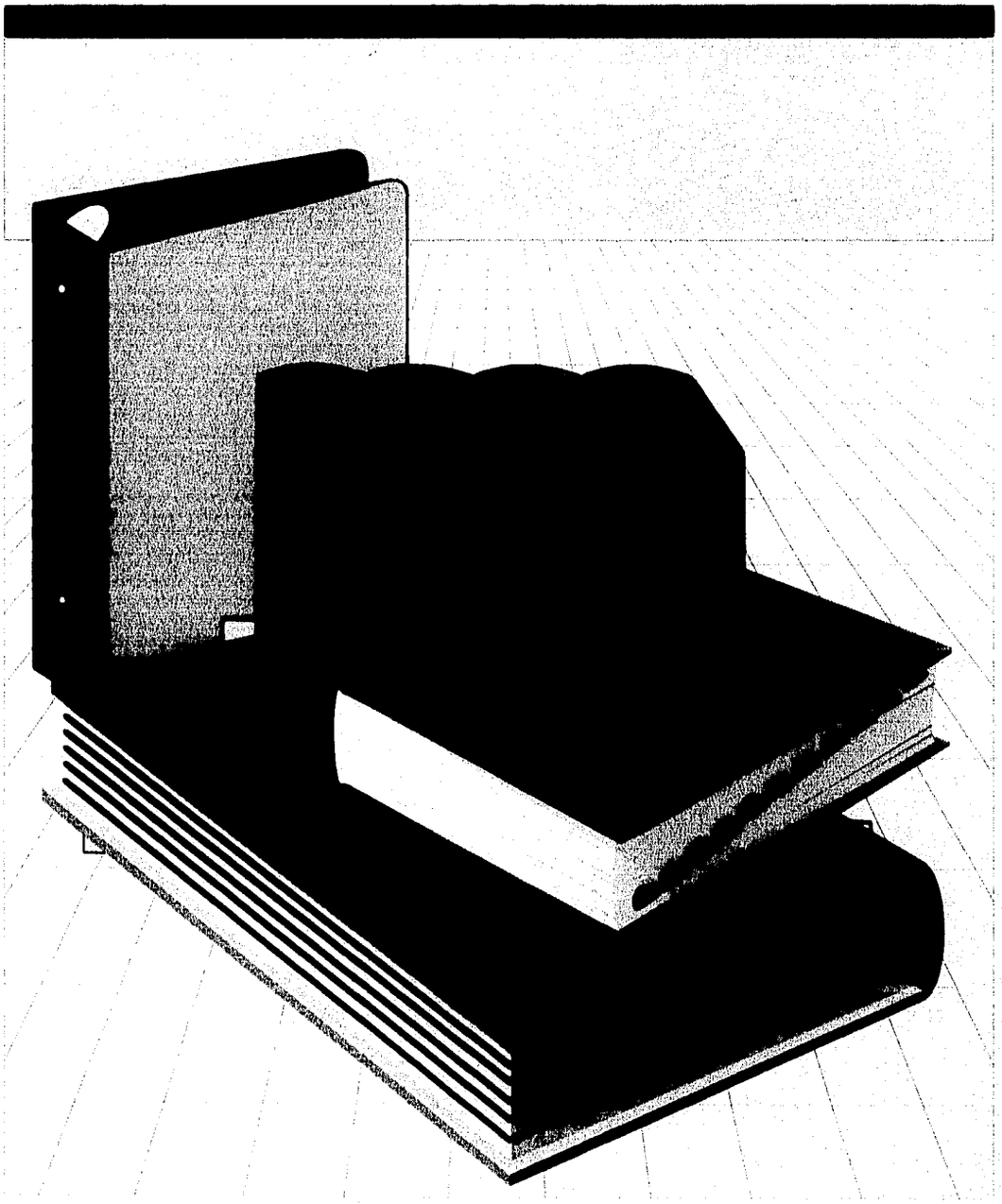
FECHA DE REANUDACION
22-04-96

CONFORME
FIRMA DEL INTERESADO

EXPEDIDO POR
LIC. MIGAEL CARVALLO BRAVO
JEFE DEPTO. DE PERSONAL

COPIA

BIBLIOGRAFIA



BIBLIOGRAFIA

1. BENAVIDES ABAJO J., OLAIZOLA BARTOLOME J.M., RIVERO CORNELIO E.
   SQL (PARA USUARIOS Y PROGRAMADORES)
   SEGUNDA EDICION 1992
   EDITORIAL PARANINFO

2. DINERSTEIN NELSON T. (COMPUTER SCIENCE DEPARTMENT UTAH STATE UNIVERSITY)
   SISTEMA DE MANEJO DE ARCHIVOS Y BASES DE DATOS PARA MICROCOMPUTADORAS
   SEGUNDA IMPRESION MAYO DE 1990
   CECSA(COMPAÑIA EDITORIAL CONTINENTAL, S.A. DE C.V. MEXICO)

3. FAIRLEY RICHARD
   INGENIERIA DE SOFTWARE
   PRIMERA EDICION 1987
   MCGRAW HILL

4. GROFF JAMES R., WEINBERG PAUL N.
   APLIQUE SQL
   MARZO 1992
   MCGRAW HILL

5. HORTON ANN, BENBROOK HOWARD, DAMERON DEAN, HETHERINGTON ART, JACOBS JEFF, STRICKLAND STEVE, ANDRONICA KATHY, CASSIDY PETE, HERZOG CLAUDIA, HOPKINS BILL, LONGMAN CLIFF, TRAVER TOM
   DISEÑO DE BASE DE DATOS Y MODELO DE DATOS
   DICIEMBRE 1993

6. KOCH GEORGE
   ORACLE (THE COMPLETA REFERENCE)
   OSBORNE MCGRAW HILL

7. KORTH HENRY F., GILBERSCHATZ ABRAHAM
   FUNDAMENTOS DE BASES DE DATOS
   MCGRAW HILL

8. LUCAS GOMEZ ANGEL, ROMERA GARCIA PALOMA, FRAILE DOTES MARIA VICTORIA, ARGANTE DEL CASTILLO FCO. JOSE, ALFARO PESA ANTONIO
   DISEÑO Y GESTION DE SISTEMAS DE BASES DE DATOS
   PRIMERA EDICION 1993
   EDITORIAL PARANINFO, S. A.

9. PRESSMAN ROGER S.
   INGENIERIA DE SOFTWARE (UN ENFOQUE PRACTICO)

MCGRAW HILL

10. RIVERO CORNELIO E.
BASES DE DATOS RELACIONALES
SEGUNDA EDICION 1992
EDITORIAL PARANINFO

11. SENN JAMES A.
ANALISIS Y DISEÑO DE SISTEMAS DE INFORMACION
SEGUNDA EDICION
MCGRAW HILL

12. SOMMERVILLE IAN
INGENIERIA DE SOFTWARE
SEGUNDA EDICION 1988
ADDISON WESLEY IBEROAMERICANA

13. **INTRODUCCION A ORACLE PARA DESARROLLADORES (PARTE 1: SQL*PLUS)**
SEPTIEMBRE 1991

14. **BASE DE DATOS RELACIONAL**
1992

15. INFORMIX SOFTWARE, INC.
RELATIONAL DATABASE DESIGN
1993

REVISTA:

1. **RED**
CONSEJO ED. ING. MARCELINO GOMEZ VELASCO, DR. ROBERT A. YATES S.
MENSUAL
MEXICO, D.F.
AÑO III, NUMERO 27. NOVIEMBRE 1992