



**UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO**

**FACULTAD DE ESTUDIOS SUPERIORES
CUAUTITLAN**

**METODOLOGIA PARA EL DESARROLLO
DE BASES DE DATOS**

TESIS

QUE PARA OBTENER EL TITULO DE:

INGENIERO MECANICO ELECTRICISTA
PRESENTA:

MOISES ARQUIMIDES JUAREZ MARTINEZ

ASESOR: ING. MARCO ANTONIO CRUZ MENDOZA



Universidad Nacional
Autónoma de México

Dirección General de Bibliotecas de la UNAM

Biblioteca Central



UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

DEDICATORIAS:

A mis Padres

Porque sin ellos nunca habría hecho nada. Por todo el apoyo brindado a través de toda mi vida, lo único que resta decir es LOS QUIERO porque son una parte súper importante en mí, por toda su sabiduría y Toda su paciencia, por los valores que siempre nos inculcaron. ¡GRACIAS!.

A mi esposa Isabel

Por la paciencia, el tiempo y los consejos que siempre me da, por darnos ambos la Oportunidad de hacer una familia en todos los sentidos y por todo lo de mas TE QUIERO...

A mi Familia

A toda la familia que siempre me apoyo para poder lograr mis metas y seria una gran lista por mencionar. Y no quiero olvidar nombres pues ellos lo saben, GRACIAS.

A la UNAM

Por todo lo que me a brindado espero algún día poder pagárselo con creces.

A los Profesores

No solo de la universidad sino a todos porque sin su ayuda no lo hubiera podido lograr ¡GRACIAS!.

A mi asesor Marco Antonio cruz Mendoza

Todo mi agradecimiento, por el apoyo y consejos para la terminación de este trabajo.

A Dios

Por toda su paciencia para conmigo por enseñarme a que los tropiezos uno solo se los provoca a no depender de él para conseguir las cosas sino a esforzarse hasta el máximo para conseguirlas a enseñarme A amar y lo más difícil a poderlo demostrar sin tapujos, a él le estoy completamente agradecido por Darme una familia como la que tengo, que no podría ser mejor, por darme la oportunidad de conocer a la Gente de la cual me rodeo que es gente leal que me quieren y los quiero, por conocer el amor en su Máxima expresión y por eso es que no se pierden los valores por todo esto y mucho más ¡gracias!

Indice

INTRODUCCION.....	1
OBJETIVOS.....	3
1. Proceso de creación y metodología de desarrollo de bases de datos.	
1.1. Introducción al ciclo de vida de una base de datos.....	4
1.2. Metodología para el desarrollo de bases de datos.....	6
2. Modelado conceptual.	
2.1. Etapas del modelado conceptual.....	7
2.2. Entidad.....	8
2.3. Interrelación.....	9
2.4. Atributo.....	12
2.5. Restricciones.....	14
2.6. Dependencia en existencia y en identificación.....	14
2.7. Jerarquía de generalización/especialización.....	15
3. Diseño lógico.	
3.1. Etapas del diseño lógico.....	18
3.2. Modelo de datos relacional.....	19
3.3. Restricciones.....	20
3.4. Integridad referencial.....	22
3.5. Transformación del esquema conceptual al lógico estándar.....	24
3.6. Diseño lógico específico.....	27
3.7. Implementación del modelo relacional en SQL Server.....	27
4. Diseño físico.	
4.1. Objetivos y actividades del diseño físico.....	43
4.2. Optimización y ajuste ó tuning de SQL Server.....	44
4.3. Indexación de la base de datos en SQL Server.....	48
4.4. Seguridad en bases de datos.....	63
5. Caso de aplicación: Galería View Ridge.	

5.1. Modelado conceptual.....	73
5.2. Diseño lógico.....	80
5.3. Diseño físico.....	83
Conclusiones.....	86
Bibliografía.....	88

INTRODUCCION

El diseño de bases de datos es una parte de un amplio panorama conocido como sistema de información. Dentro del sistema de información no solo se reúnen, guardan y recuperan datos, sino que también se transforman en información útil. Con información oportuna y pertinente, los ejecutivos son capaces de tomar decisiones que les permiten a las organizaciones prosperar y crecer. Probablemente los diseños de base de datos que no reconocen que ésta forma parte de este todo no lleguen a tener éxito. Es decir, los diseñadores de bases de datos deben reconocer que este es un medio crítico para lograr un objetivo, y no un objetivo por si mismo (los gerentes esperan que la base de datos satisfaga sus necesidades de administración, pero aparentemente muchas bases de datos requieren que los gerentes modifiquen sus rutinas para ajustarse a los requerimientos de estas).

Los sistemas de información no surgen de la nada; son el producto de un cuidadoso proceso de desarrollo por etapas. Para determinar la necesidad de un sistema de información y para establecer sus límites, se utiliza un proceso conocido como análisis de sistema. Dentro del análisis del sistema, se crea el sistema de información propiamente dicho mediante un proceso conocido como desarrollo de sistemas.

La existencia de una organización no es estática. Incluso su crecimiento demanda información nueva y ampliada. Por consiguiente, desde el momento en que se completa un sistema de información, se somete a una evaluación continua. ¿Las aplicaciones del sistema aun realizan las funciones para las que fueron creadas? ¿Cómo pueden mejorarse para que satisfagan los requerimientos cambiantes? ¿La base de datos aun realiza las funciones para las que fue creada como deposito de datos con los cuales las aplicaciones producen la información apropiada?

Tarde o temprano aun el sistema de información mas fino alcanza el punto de obsolescencia. Si ya no se ajusta a los requerimientos de la empresa, ¿Cómo se crea un sistema que lo haga? Para citar una frase muy conocida: la única constante en el ámbito del desarrollo de sistemas es el cambio. Este proceso continuo de creación, mantenimiento, mejora y reemplazo constituye el ciclo de vida del desarrollo de sistemas.

Al igual que las aplicaciones basadas en el, la base de datos debe someterse a una constante evaluación. La base de datos primero se crea, después se mantiene y mejora. Pero puede llegar el momento en el que incluso una mejora ya pueda ampliar la utilidad de la base de datos. Cuando la base de datos ya no puedas realizar sus funciones adecuadamente, puede que tenga que reemplazarse. En suma, la base de datos se somete al ciclo de vida de base de datos.

Dentro de un gran sistema de información. La base de datos también esta sometida a un ciclo de vida. El ciclo de vida de base de datos contiene varias fases como: Estudio previo y plan de trabajo; concepción de la base de datos y selección del equipo; y diseño y carga.

La concepción de una base de datos es una tarea larga y costosa. Estas dificultades inherentes al diseño de una base de datos han de afrontarse con procedimientos

ordenados y metódicos en el marco de una metodología de diseño de una base de datos. En el proceso de diseño de una base de datos hemos de distinguir tres grandes fases:

- Diseño conceptual.
- Diseño lógico.
- Diseño físico.

Diseño conceptual.

En la etapa de diseño conceptual, se utiliza el modelo de datos para crear estructuras de base de datos abstractas que representen objetos del mundo real de la manera más real posible. El modelo conceptual debe dar cuerpo a un claro entendimiento de la empresa y a sus funcionales. A este nivel de abstracción, el tipo de hardware o de modelo de base de datos, o ambos a ser utilizados, es posible que aun no puedan ser identificados. Por consiguiente, el diseño debe ser independiente del software y del hardware, de modo que el sistema pueda ser configurado mas adelante con cualquier hardware y plataforma de software que se elija

Diseño logico

El diseño lógico se utiliza para transformar el diseño conceptual en el modelo interno de un sistema de administración de base de datos seleccionado, tal como DB, SQL Server, Oracle, IMS, Informix, Access, etc. Esto incluye la proyección de todos los objetos del modelo en las construcciones específicas utilizadas por el software de base de datos seleccionado. El diseño lógico incluye el diseño de tablas, índices, vistas, transacciones, autoridades de acceso etc.

Diseño fisico

El diseño físico es el proceso de seleccionar las características de almacenamiento y acceso a los datos de la base de datos. Las características de almacenamiento son una función de los tipos de dispositivo soportados por el hardware.

El diseño físico se describe mejor como una tarea técnica, más típica del mundo cliente/servidor y mainframe que del mundo de la PC. No obstante, incluso en los entornos más complejos de la microcomputadora y mainframe, el software de base de datos moderno ha asumido mucha de la carga de la parte física del diseño y su ejecución.

El diseño físico se torna mas competo cuando los datos están distribuidos en diferentes lugares, porque el desempeño se ve afectado por el rendimiento efectivo total de los medios magnéticos de comunicación.

En suma, la creación de una base de datos requiere de una metodología para el diseño de esta, a la que se le seguirá la pista con cuidado.

OBJETIVOS:

- Resumir el ciclo de vida de una base de datos, abordando tanto sus aspectos técnicos como organizativos. Presentar una metodología para el desarrollo de bases de datos relacionales, estudiando sus elementos y sus características.
- Analizar la primera fase de la metodología de desarrollo de bases de datos: el modelado conceptual. Presentar el modelo Entidad/Interrelación que sirve de base para el modelo conceptual.
- Abordar la etapa de diseño lógico estándar, que esta basada en el modelo relacional. Presentar las reglas que permiten transformar un esquema conceptual (E/R) en un esquema lógico (relacional). Exponer de forma resumida, algunos conceptos sobre diseño lógico específico de bases de datos.
- Examinar la última etapa del diseño de bases de datos: el diseño físico. El diseño físico depende del SGBD que se utilice y por tanto, nos limitaremos a ofrecer una visión global de la misma tratando aspectos que son, en general, comunes a muchos productos.
- Crear una aplicación de bases de datos utilizando la metodología propuesta.

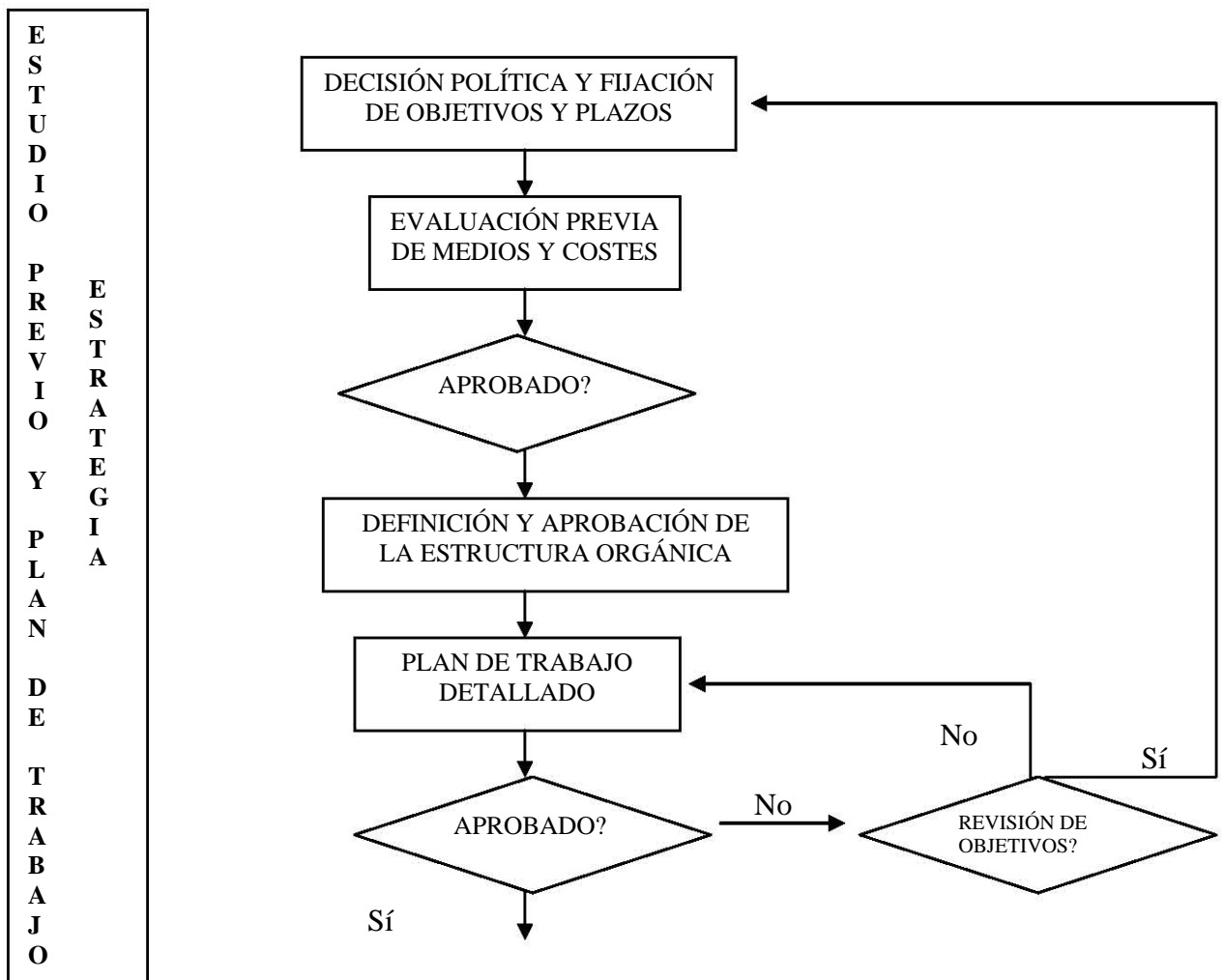
1. PROCESO DE CREACION Y METODOLOGIA DE DESARROLLO DE BASES DE DATOS.

1.1. Introducción al ciclo de vida de una base de datos.

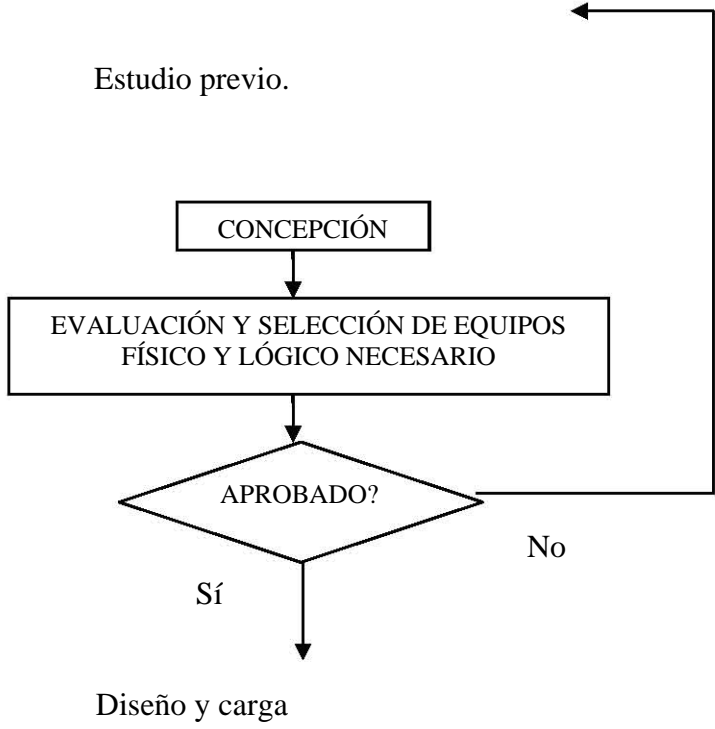
La creación de una base de datos es, generalmente, una operación difícil, larga y costosa, que no puede improvisarse. No se trata solamente de un problema técnico, ya que las repercusiones que esta decisión puede tener en todos los niveles de la empresa hacen de ella una decisión que atañe a la política empresarial, por lo que no debe ser abordada en exclusiva por los técnicos.

Las distintas fases que comprende la puesta en marcha de un sistema de información orientado hacia las bases de datos son:

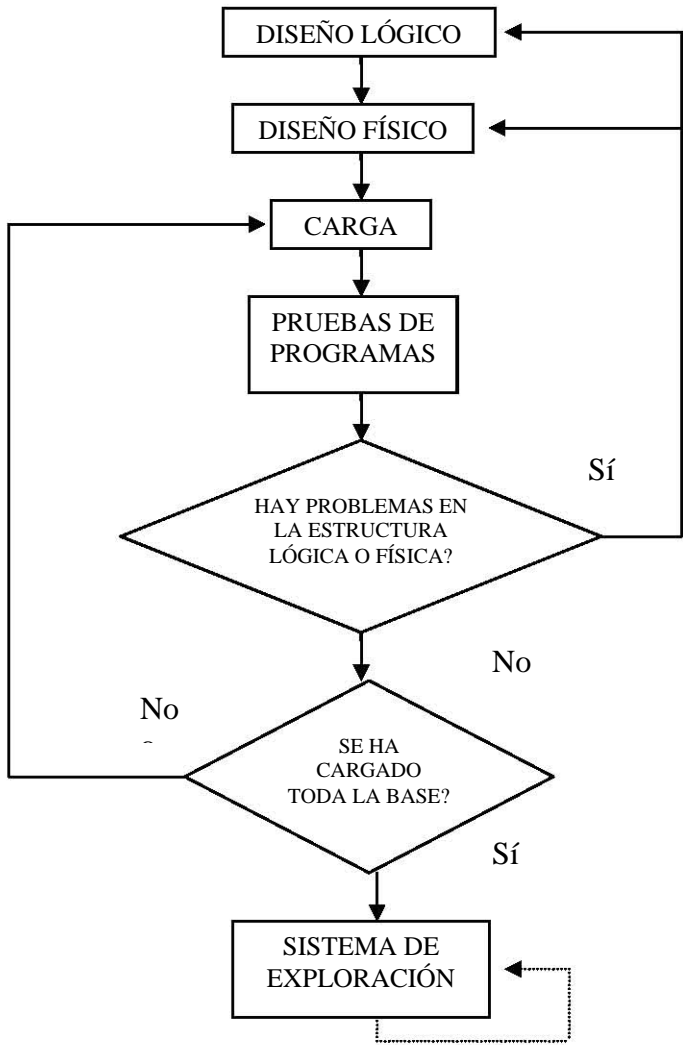
- Estudio previo y plan de trabajo.
- Concepción de la base de datos y selección del equipo.
- Diseño y carga.



C O N C E P C I O N D E L A B D Y S E L E C C I O N D E E Q U I P O



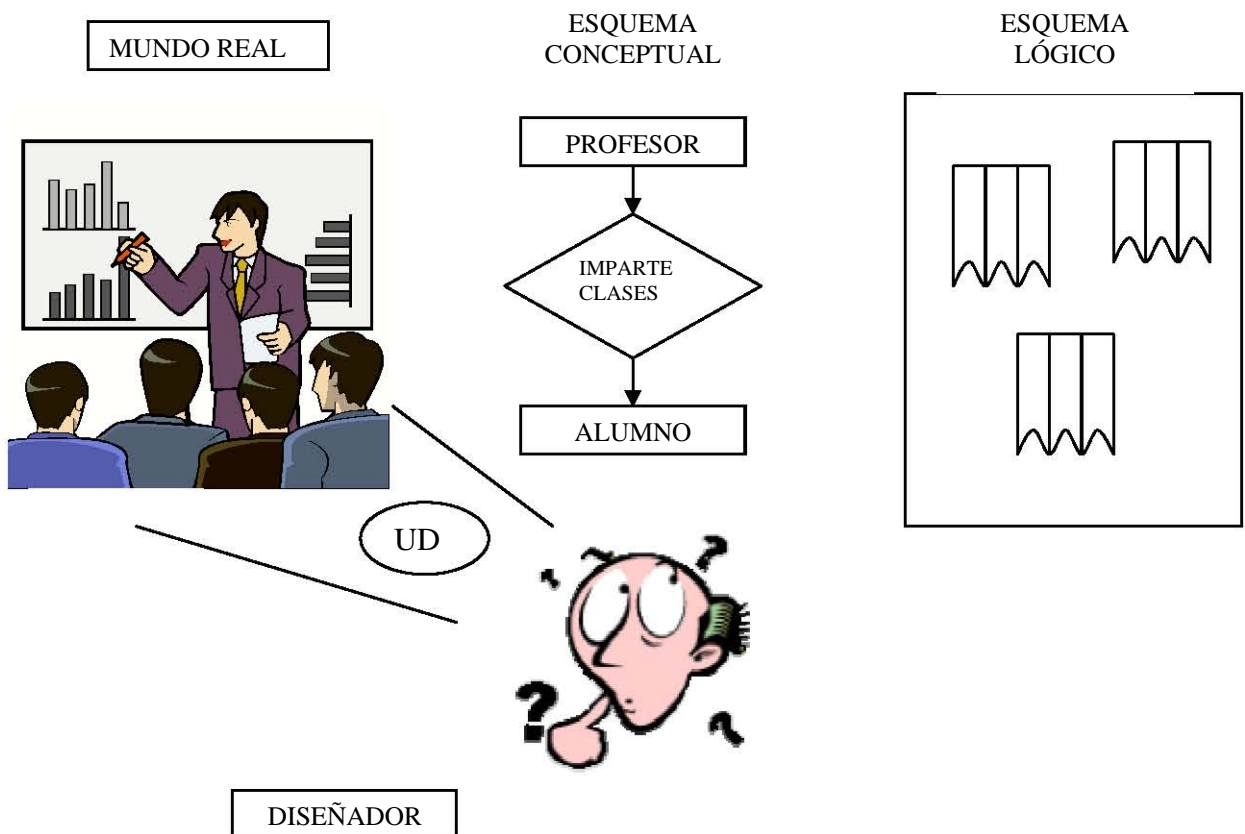
D I S E Ñ O Y C A R G A



1.2. Metodología para el desarrollo de bases de datos.

La concepción de una base de datos es una tarea larga y costosa. Estas dificultades inherentes al diseño de una base de datos han de afrontarse con procedimientos ordenados y metódicos en el marco de una metodología de diseño de bases de datos. En el proceso de diseño de una base de datos hemos de distinguir tres grandes fases:

- Diseño conceptual: cuyo objetivo es obtener una buena representación de los recursos de información de la empresa, con independencia de usuarios o aplicaciones en particular, y fuera de consideraciones sobre eficiencia del ordenador.
- Diseño lógico: cuyo objetivo es transformar el esquema conceptual obtenido en la etapa anterior, adaptándolo al modelo de datos en el que se apoya el SGBD que se va utilizar.
- Diseño físico: cuyo objetivo es conseguir una instrumentación, lo más eficiente posible, del esquema lógico.



2. Modelado conceptual

2.1 Etapas del modelado conceptual.

El modelado conceptual, también denominado diseño conceptual, constituye la primera fase de desarrollo de bases de datos, y puede subdividirse en dos etapas claramente diferenciadas:

- Análisis de requisitos: Esta primera etapa, en general común para datos y procesos, es la etapa de percepción, identificación y descripción de los fenómenos del mundo real a analizar.
- Etapa de conceptualización. En ella se transforma este primer esquema descriptivo, refinándolo y estructurándolo adecuadamente.



Los modelos de datos soportados por los SGBD no suelen ofrecer, dado su bajo nivel de abstracción, los mecanismos suficientes para captar la semántica del mundo real, por lo que surgen modelos conceptuales, más ricos semánticamente, que facilitan la labor del diseñador ayudándole en su comunicación con el usuario.

Entre estos modelos de datos conceptuales, el modelo Entidad/Interrelación, es posiblemente el de más amplia aceptación y al que se le suele dar soporte en la mayoría de las herramientas CASE.

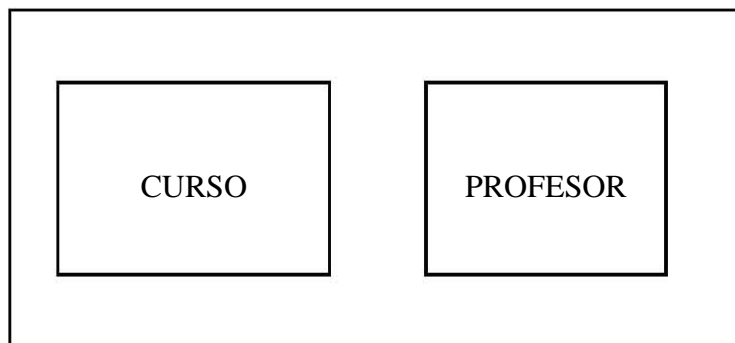
En el modelo E/R, se distinguen los siguientes elementos:

- Entidad.
- Interrelación.
- Atributo.

2.2. Entidad

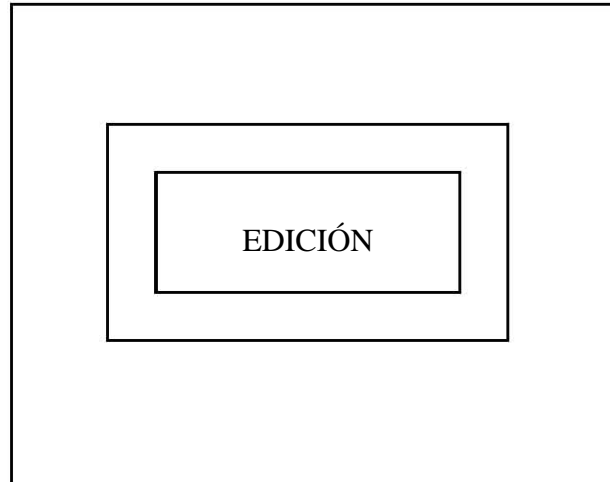
Se puede definir una entidad como cualquier objeto (real o abstracto) que existe en la realidad y acerca del cual queremos almacenar información en la base de datos.

La representación gráfica de una entidad en este modelo es un rectángulo etiquetado en cuyo interior está el nombre de la entidad.



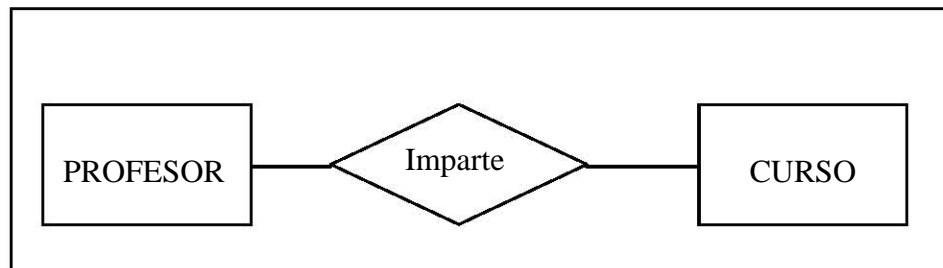
Existen dos clases de entidades:

- Regulares. Las ocurrencias de una entidad regular tienen existencia propia, es decir, existen por sí mismas.
- Débiles. La existencia de cada ocurrencia de una entidad débil depende de la existencia de la ocurrencia de la entidad regular del cual aquella depende. Una entidad débil se representa con dos rectángulos concéntricos con su nombre en el interior.



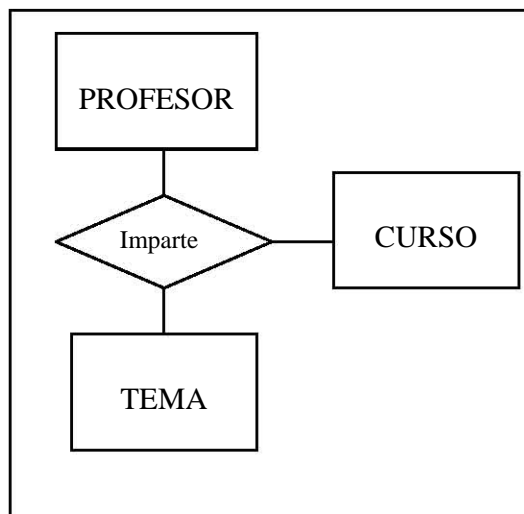
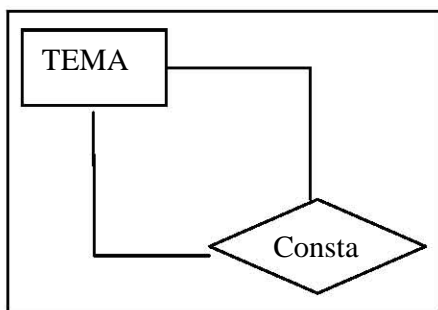
2.3. Interrelación

Se entiende por interrelación una asociación, vinculación o correspondencia entre entidades. Representamos una interrelación mediante un rombo etiquetado con el nombre de la interrelación, unido mediante arcos a las entidades que asocia.

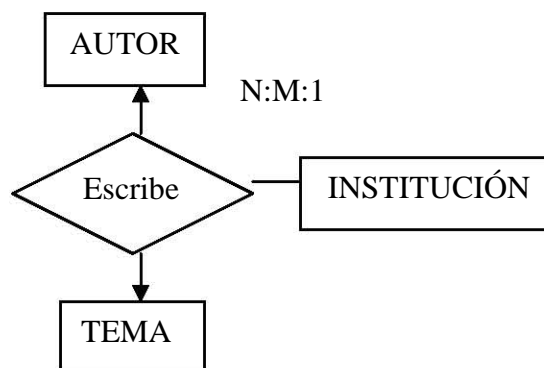
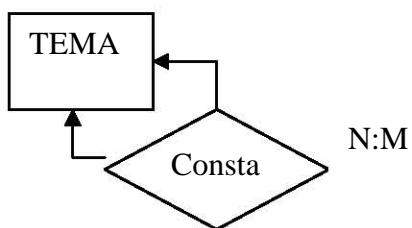
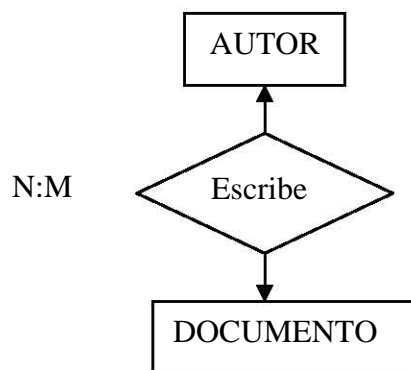
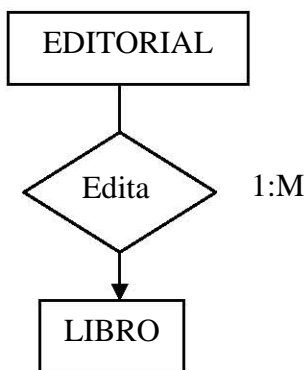


Una interrelación se caracteriza por:

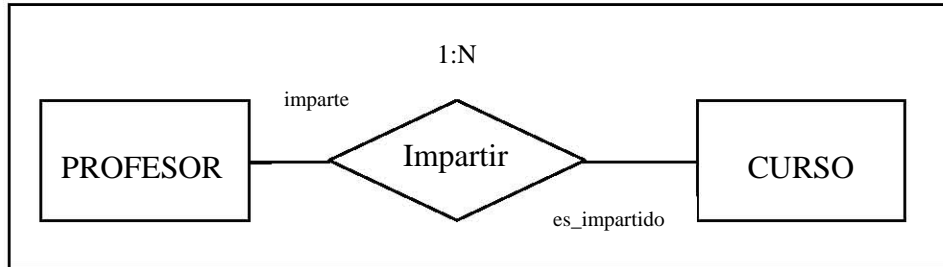
- Nombre: Por el que identificamos de forma única la interrelación y mediante el cual lo referenciamos.
- Grado: Número de entidades que participan en una interrelación. Puede ser de grado 2 (binarias) cuando asocian dos entidades (entre ellas tenemos las reflexivas o recursivas que asocian ocurrencias de un misma entidad); de grado 3 (ternarias) cuando asocian tres entidades; o en general de grado n.



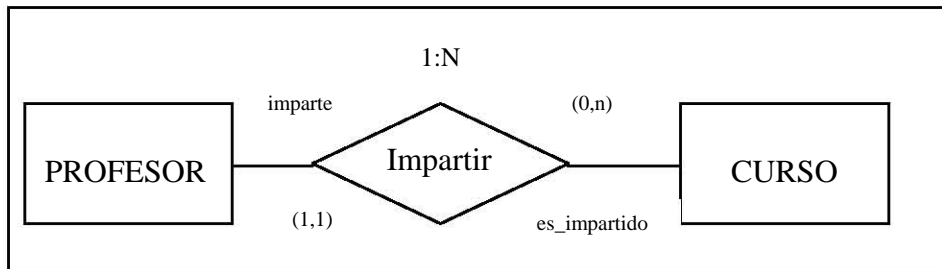
- Tipo de correspondencia: Es el número máximo de ocurrencias de una entidad que puede estar asociados, en una determinada interrelación, con una ocurrencia de otra(s) entidad(es); para representarlo gráficamente, bien se pone una etiqueta con 1:1, 1:N o N:M según corresponda al lado de interrelación, o una punta de flecha hacia la entidad que participa con más de una ocurrencia en la interrelación.



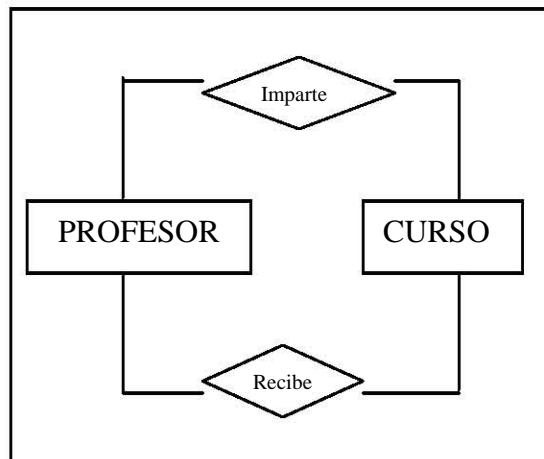
- **Papel (rol):** Es la función que cada una de las entidades realiza en la interrelación; se representa poniendo el nombre del papel en el arco que une cada entidad con la interrelación.



- **Cardinalidad:** Se define como el número máximo y mínimo de ocurrencias de una entidad que participan en la interrelación. Se representa gráficamente mediante una etiqueta del tipo (0,1), (1,1), (0,n) ó (1,n), según corresponda, al lado de las entidades asociadas por la interrelación.



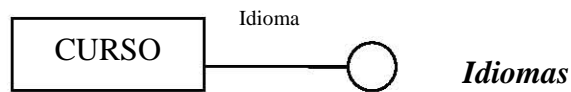
Entre dos entidades puede existir más de una interrelación.



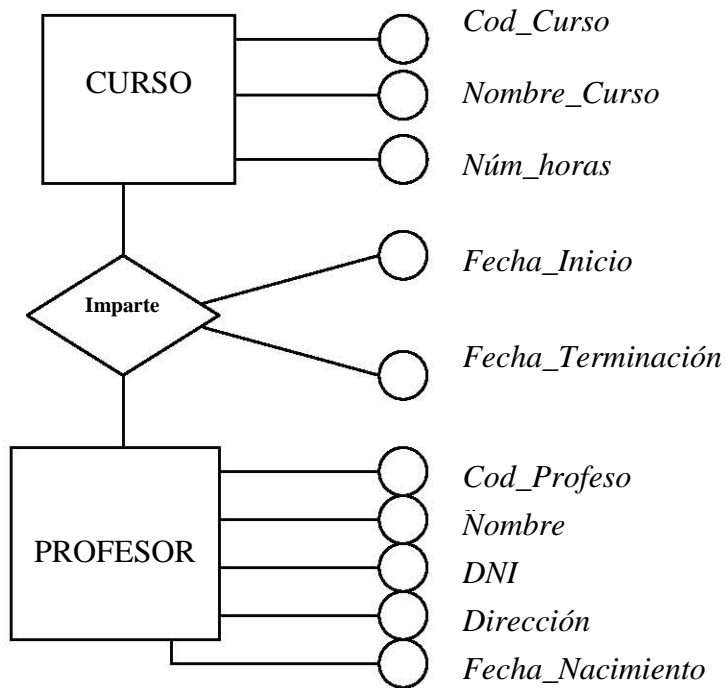
2.4. Atributo

Es cada una de las propiedades o características que tiene una entidad o interrelación. El conjunto de posibles valores que puede tomar un atributo recibe el nombre de dominio.

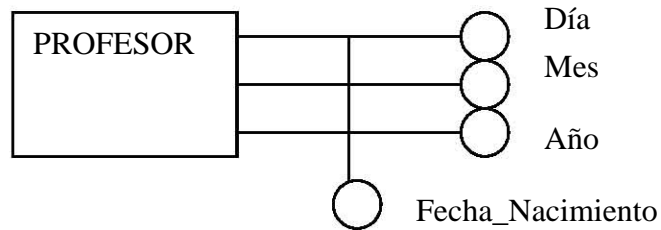
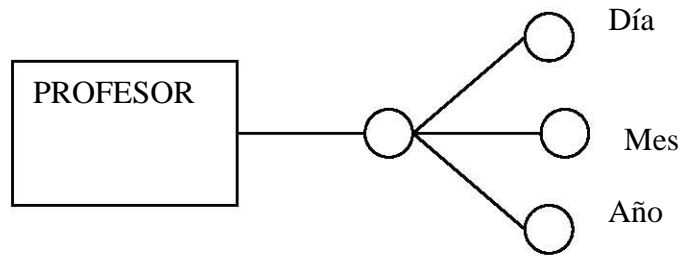
El dominio se representa con un círculo u óvalo en cuyo interior aparece su nombre, mientras que el nombre del atributo se escribe sobre el arco que une el dominio con la entidad o interrelación a la que pertenece dicho atributo.



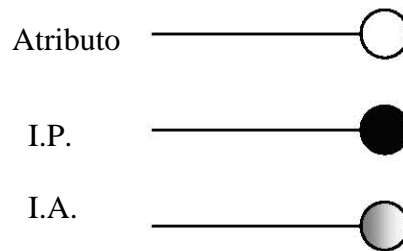
Para simplificar la representación gráfica, en muchos casos será suficiente con el círculo u óvalo con el nombre del atributo.



El modelo E/R admite atributos compuestos, es decir, atributos definidos sobre más de un dominio.

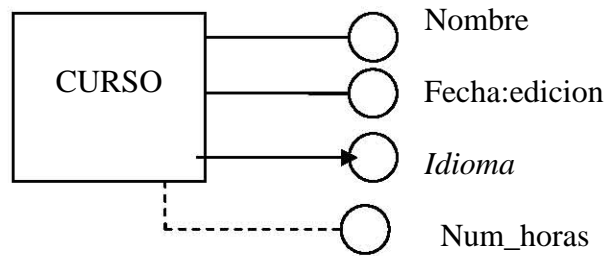


Entre todos los atributos de una entidad debemos elegir uno o varios que identifiquen unívoca y mínimamente cada una de las ocurrencias de esa entidad (atributo identificador principal AIP). Puede que exista más de un atributo que cumple esta condición (atributo identificador candidato), de los cuales se elige uno como principal y los otros son alternativos.



El modelo E/R permite también atributos multivaluados y opcionales (nulos o “faltantes”). En general un atributo toma, para cada ejemplar de entidad, un único valor de cada dominio (o dominios) subyacente(s), pero también existen atributos que pueden tomar más de un valor; estos atributos reciben el nombre de multivaluados frente a los univaluados que toman un solo valor.

Por otro lado, puede obligarse a un atributo de un tipo de entidad a que tome, como mínimo, un valor del (o de los) dominio(s) subyacente(s) para cada ejemplar de entidad; es decir, el valor de este atributo es obligatorio (no puede ser nulo) para todo ejemplar de la entidad.



2.5. Restricciones

El modelo E/R tiene como restricción inherente que sólo permite establecer interrelaciones entre entidades, no estando admitidas entre entidades e interrelaciones ni entre interrelaciones. También obligan al modelo a que todas las entidades tengan un identificador, lo que asimismo podría considerarse una restricción inherente. El no tener apenas restricciones inherentes dota al modelo de una gran flexibilidad para la representación del mundo real.

2.6. Dependencia en existencia y en identificación

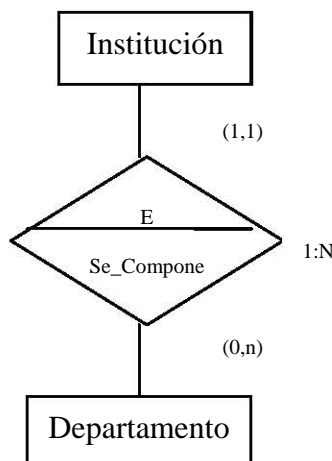
Las interrelaciones se clasifican, según el tipo de entidades que vinculan, en regulares si asocian entidades regulares y débiles si asocian una entidad débil con una entidad regular. Una interrelación débil exige siempre que las cardinalidades de la entidad regular sean (1,1).

Dentro de las interrelaciones débiles podemos distinguir:

- Dependencia en existencia.
- Dependencia en identificación.

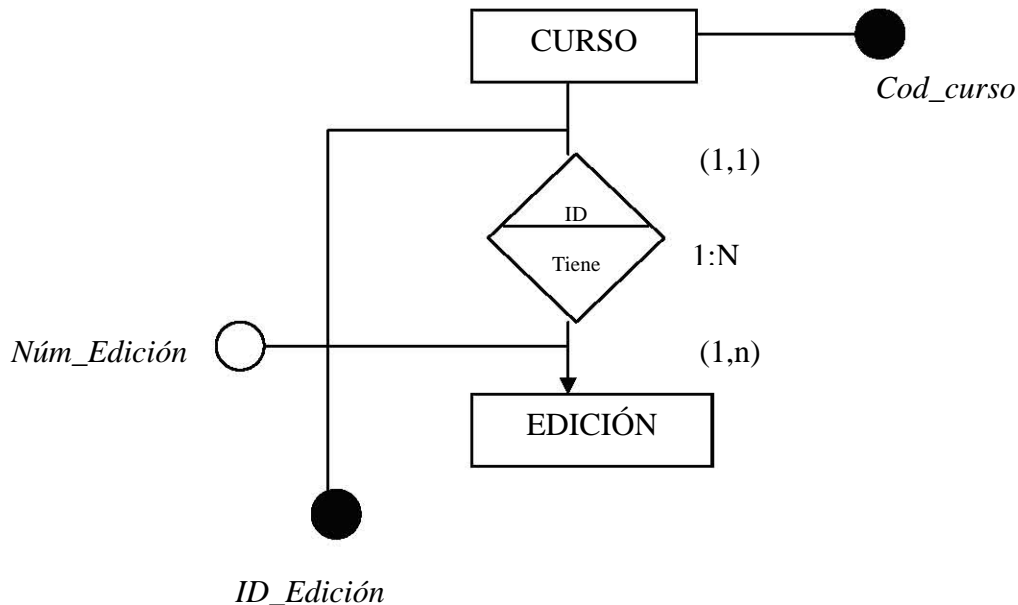
2.6.1. Dependencia en existencia.

Se dice que hay dependencia en existencia cuando los ejemplares de una entidad (entidad débil) no pueden existir si desaparece el ejemplar de la entidad regular del cual dependen. Se indica añadiendo la etiqueta "E" al rombo que representa la interrelación débil con dependencia en existencia.



2.6.2. Dependencia en identificación.

Se dice que existe dependencia en identificación cuando, además de la dependencia en existencia, los ejemplares de la entidad débil no se pueden identificar por sí mismos, es decir, mediante los propios atributos de la entidad, y exigen añadir el identificador principal de la entidad regular del cual dependen. Se indica gráficamente añadiendo la etiqueta ID al rombo que representa la interrelación.

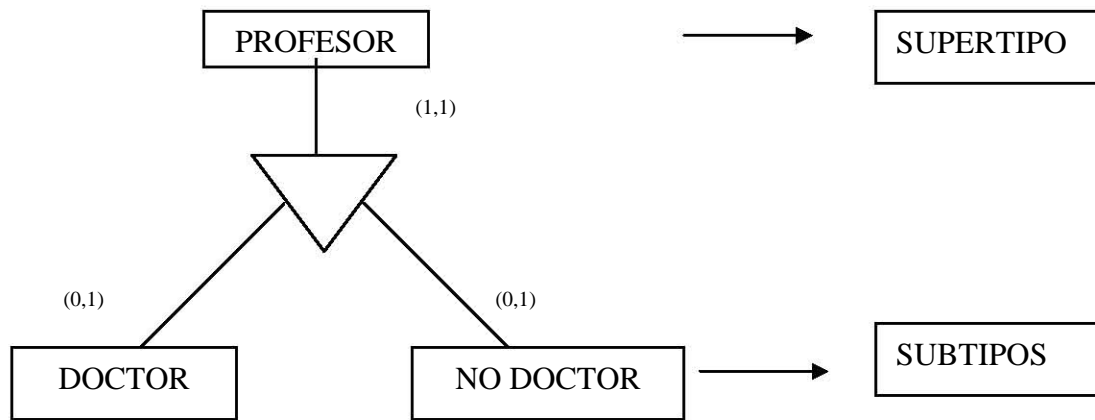


2.7. Jerarquía de generalización/especialización.

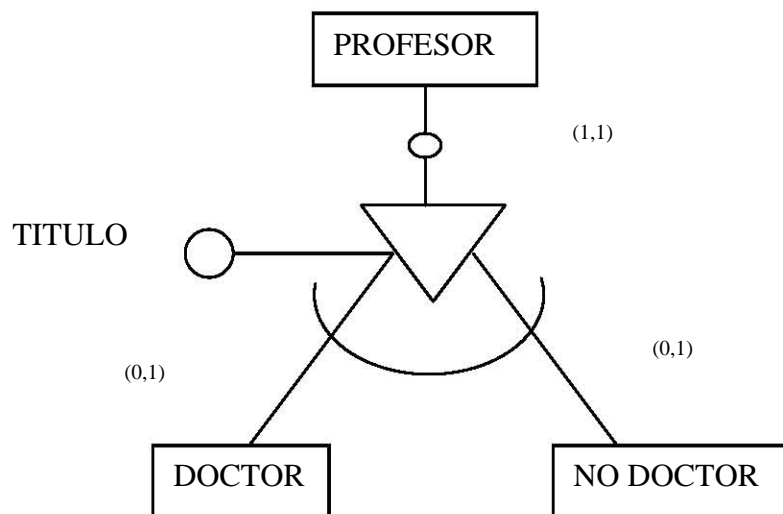
La jerarquía de generalización/especialización en el modelo E/R, se considera como un caso especial de interrelación entre varios tipos de entidad (subtipos o especialización) y un tipo más general (supertipo o generalización) cuyas características son comunes a todos los subtipos. La abstracción correspondiente a este tipo de interrelación entre entidades se denomina ES_UN.

Para la representación de este tipo de interrelación, utilizamos un triángulo invertido, con la base paralela al rectángulo que representa el supertipo y conectado a éste y a los subtipos. Las cardinalidades son siempre (1,1) en el supertipo y (0,1) en los subtipos.

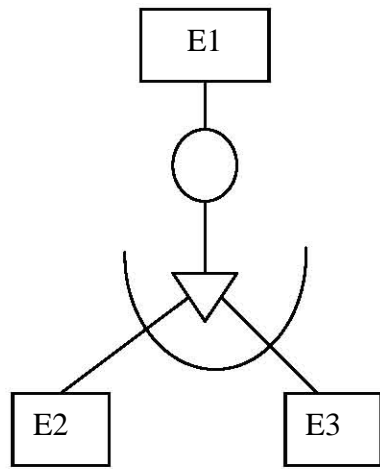
Una de las características más importantes de las jerarquías es la herencia, por la cual, los atributos de un supertipo son heredados por sus subtipos.



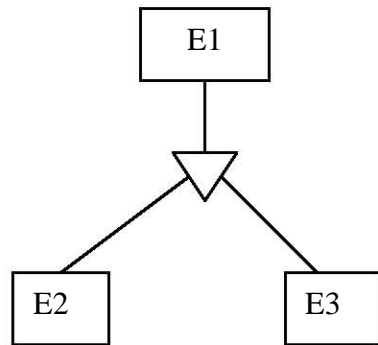
La división en subtipos puede venir determinada por una condición predefinida (por ejemplo, en función de los valores de un atributo) en cuyo caso se representará la condición (o el atributo discriminante) asociada al triángulo que representa la interrelación. Si no interesa considerar ninguna condición predefinida, deberá ser el usuario, en el momento de insertar un ejemplar en la base de datos, quién especifique a cual de los subtipos pertenece.



La abstracción generalización/especialización tiene algunas restricciones semánticas. Si un mismo ejemplar del supertipo puede pertenecer a más de un subtipo habrá solapamiento, y si sólo puede pertenecer a uno de los subtipos existirá exclusividad; por otro lado, si todo ejemplar del supertipo tiene que pertenecer a algún subtipo tendremos totalidad, y si, por el contrario, no tienen obligatoriamente que pertenecer a algún subtipo habrá parcialidad.



Jerarquía de generalización /
especialización (con
restricción de totalidad y
exclusiva)



Jerarquía de
Generalización /
especialización (sin
restricciones)

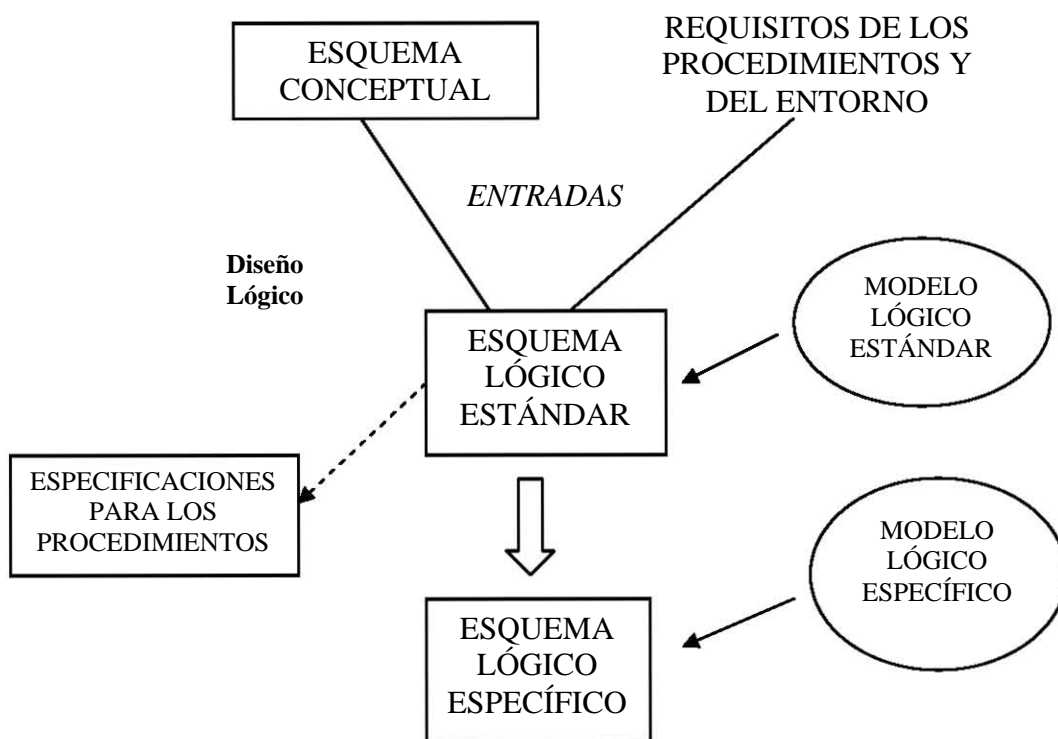
3. Diseño lógico.

3.1 Etapas del diseño lógico.

En la metodología que proponemos, introducimos las características del SGBD lo más tarde posible, ya que, a nuestro juicio, todo producto informático (y una base de datos, es uno más) debe poseer, como ya se ha señalado, la portabilidad como una de sus características más deseables. Esta portabilidad es necesaria para desarrollar productos que pueden ser implementados sobre distintos SGBD, y para facilitar la migración entre versiones de un mismo sistema.

Las etapas que proponemos dentro del diseño lógico son las siguientes:

- Diseño lógico estándar. A partir del esquema conceptual resultante de la etapa anterior, y teniendo en cuenta los requisitos de proceso y de entorno, se elabora un esquema lógico estándar (ELS), que se apoya en un modelo lógico estándar (MLS), el cual será el mismo modelo de datos (Jerárquico, Red o Relacional) soportado por el SGBD que se vaya a utilizar, pero sin las restricciones ligadas a ningún producto comercial.
- Diseño lógico específico. Con el esquema lógico estándar (ELS), y teniendo en cuenta el modelo lógico específico (MLE) propio del SGBD (Ingres, Sybase, DB2, Oracle, Informix, SQLServer, etc.), se elabora el esquema lógico específico (ELE), que será descrito en el lenguaje de definición de datos (LDD) del producto comercial que estemos utilizando.



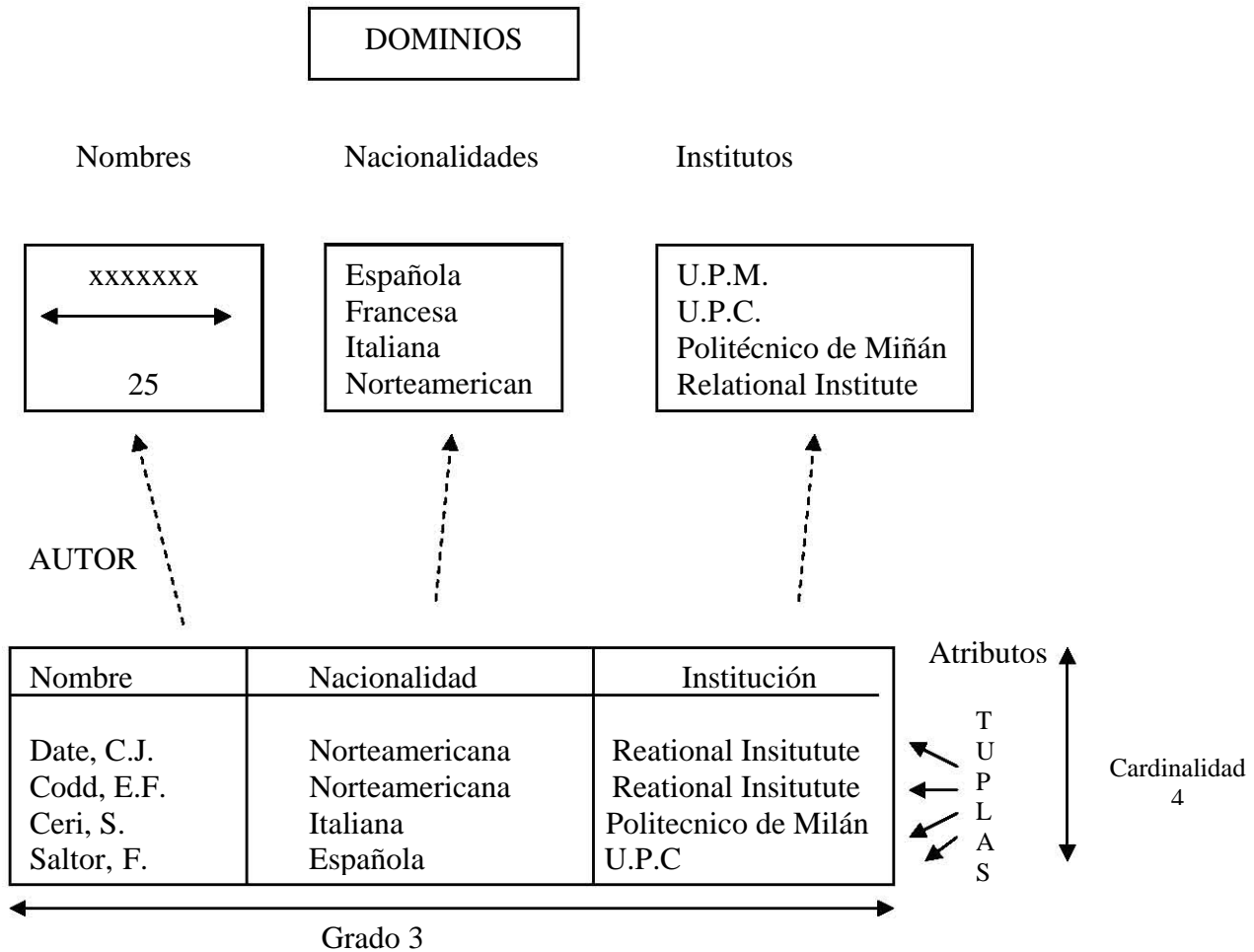
3.2 Modelo de datos relacional.

La estructura básica, y única, del modelo relacional es la relación (también llamada tabla), que sirve para representar tanto los objetos como las asociaciones entre ellos. En ella podemos distinguir su nombre, un conjunto de columnas, denominadas atributos, que representan propiedades de la tabla y que también se caracterizan por su nombre, y un conjunto de filas llamadas tuplas, que contienen los valores que toma cada uno de los atributos para cada elemento de la relación.

Los atributos se definen sobre los dominios (de donde los atributos toman sus valores; varios atributos pueden tomar valores del mismo dominio). El número de atributos se llama grado de la relación. El número de tuplas se llama cardinalidad de la relación. La cardinalidad varía en el transcurso del tiempo. Una clave candidata de una relación es un conjunto no vacío de atributos que identifican unívoca y mínimamente cada tupla de una relación. Una relación puede tener más de una clave candidata, entre ellas cabe distinguir:

- Clave primaria: Es la clave candidata que el usuario elegirá, por consideraciones ajenas al modelo relacional, para identificar las tuplas de la relación. Los atributos que forman parte de la clave primaria no pueden tomar valores nulos.
- Claves alternativas: Son aquellas claves candidatas que no han sido elegidas como claves primarias de la relación.

Una clave ajena de una relación R2 es un conjunto no vacío de atributos cuyos valores han de coincidir con los valores de la clave primaria de una relación R1 (R1 y R2 pueden ser la misma relación). Se dice que R2 es la relación que referencia, mientras que R1 es la relación referenciada.



3.3 Restricciones

En el modelo relacional, al igual que en otros modelos, existen restricciones, es decir, estructuras u ocurrencias no permitidas, siendo preciso distinguir entre restricciones inherentes y restricciones semánticas (de usuario).

3.3.1 Restricciones inherentes

De la definición de relación se deduce inmediatamente una serie características propias de una relación que se han de cumplir obligatoriamente, por lo que se trata de restricciones inherentes y que, como ya hemos señalado, diferencian una relación de una tabla; estas características son:

- No hay dos tuplas iguales (de donde se deduce la obligatoriedad de la clave primaria).
- El orden de las tuplas no es significativo.
- El orden de los atributos no es significativo.

- Cada atributo sólo puede tomar un único valor del dominio sobre el que está definido, no admitiéndose por tanto los grupos repetitivos.

3.3.2 Integridad de entidad

Además de las anteriores restricciones inherentes, derivadas de la misma definición de relación, existe otra restricción inherente que es la regla de integridad de entidad, la cual impone que: Ningún atributo que forme parte de la clave primaria de una relación puede tomar un valor nulo”; esto es, un valor desconocido o inexistente.

3.3.3 Restricciones semánticas

Las principales restricciones semánticas del modelo relacional son las siguientes:

- Clave primaria (PRIMARY KEY). Permite declarar un atributo o un conjunto de atributos como clave primaria de una relación, por lo que sus valores no se podrán repetir ni se admitirán los nulos.
- Unicidad (UNIQUE). Mediante la cual se indica que los valores de un conjunto de atributos (uno o más) no pueden repetirse en una relación. Esta restricción permite la definición de claves alternativas.
- Obligatoriedad (NOT NULL), de uno o más atributos, con lo que se indica que el conjunto de atributos no admite valores nulos.
- Clave ajena (FOREIGN KEY). Define una clave ajena, que identifica una relación entre dos tablas. El atributo o atributos clave ajena de una relación referencia a una clave primaria (una o varios atributos) de otra relación.

PROGRAMA

Cod_Programa	NOMBRE	DEPARTAMENTO
123FG	Ing.Informática	Leng. y sistemas
123FH	Derechos fundamentales	Derecho Sco.
458TG	Documentación	Biblioteconomía

CURSO_DOCTORADO

Cod_Curso	Nombre	N_Horas	Cod_Programa
DF00012	Sociología en Derec. Fund.	25	123FG
DF00021	Teoría Jurídica	30	123FH
D000034	Evaluación de Revi. Cientificas	20	458GT
II000087	Almacenes de datos OLAP	100	123FG
II000142	Evaluación de procesos	25	123FG
D000487	Automatización de bibliotecas	35	458GT

CLAVE AJENA

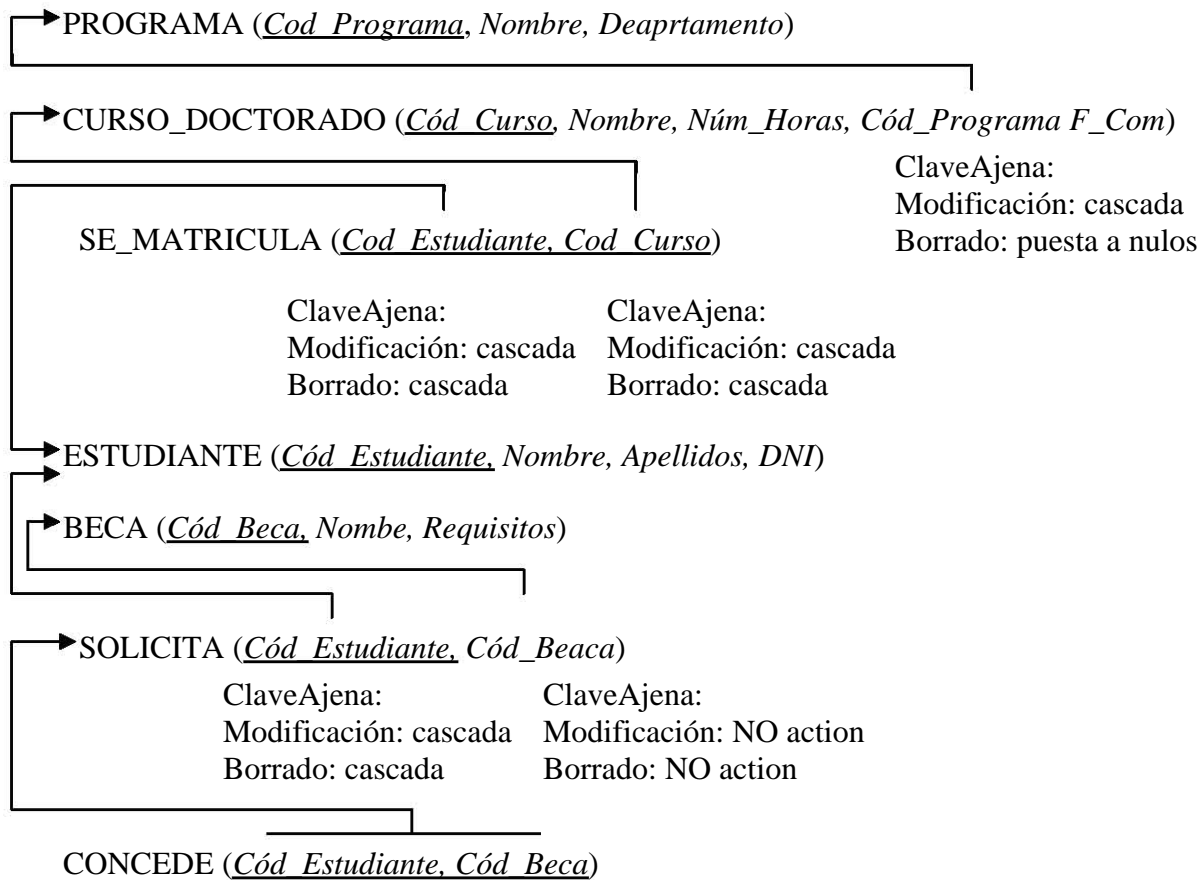


3.4. Integridad referencial

Dentro de las restricciones propias del modelo relacional se encuentra la de integridad referencial, que es una condición y acción específicas y que afirma: “Si una relación R2 tiene un descriptor que es clave ajena que referencia a la clave primaria de la relación R1, todo valor de la clave ajena debe coincidir con un valor de la clave principal, o ser nulo.

Además de definir las claves ajenas, hay que determinar las consecuencias que puede tener ciertas operaciones (borrado y modificación) realizadas sobre tuplas de la relación referenciada; pudiéndose distinguir, las siguientes opciones:

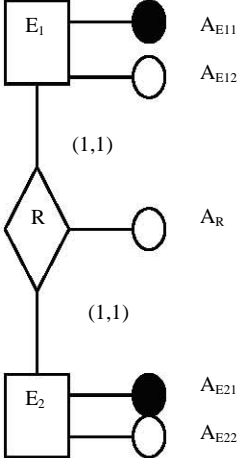
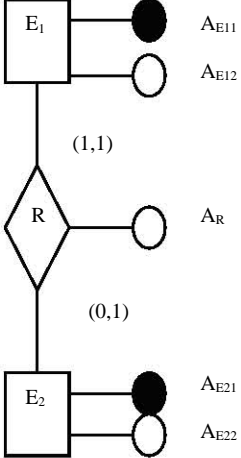
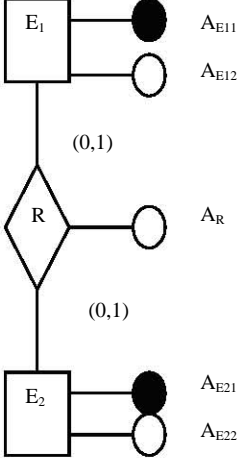
- NO ACTION (rechazar la operación)
- CASCADE (propagar la modificación o borrar las tuplas de la tabla que referencia)
- SET NULL (poner valor nulo en la clave ajena de la tabla que referencia)
- SET DEFAULT (poner valor por defecto en la clave ajena de la tabla que referencia)



Además de las restricciones que acabamos de exponer, existen en el modelo relacional otras restricciones que podríamos llamar de rechazo:

- Verificación (CHECK). Comprueba, en toda operación de actualización, si el predicado es cierto o falso y, en el segundo caso rechaza la operación. La restricción de verificación se define sobre un único elemento (incluyéndose en la definición de dicho elemento) y puede o no tener nombre.

3.5. Transformación del esquema conceptual al lógico estándar

TIPO	ESQUEMA INICIAL	POSIBLE TRADUCCIÓN
<p>UNO A UNO Relación con participación obligatoria para ambas entidades</p>		$E_1 (\underline{A_{E11}}, A_{E12})$ $E_2 (\underline{A_{E21}}, A_{E22}, A_{E11}, A_R)$ <p>ALTERNATIVA:</p> $E_1 (\underline{A_{E11}}, A_{E12}, A_{E21}, A_R)$ $E_2 (\underline{A_{E21}}, A_{E22})$
<p>UNO A UNO Relación con participación opcional para una entidad</p>		$E_1 (\underline{A_{E11}}, A_{E12})$ $E_2 (\underline{A_{E21}}, A_{E22}, A_{E11}, A_R)$
<p>UNO A UNO Relación con participación opcional para ambas entidades</p>		$E_1 (\underline{A_{E11}}, A_{E12})$ $E_2 (\underline{A_{E21}}, A_{E22})$ $R (\underline{A_{E11}}, \underline{A_{E21}}, A_R)$

<p style="text-align: center;">BINARIA Relación muchos a muchos</p>		<p style="text-align: center;"> $E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, A_R)$ </p>
<p style="text-align: center;">TERNA Relación muchos a muchos</p>		<p style="text-align: center;"> $E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22})$ $E_3(\underline{A_{E31}}, A_{E32})$ $R(\underline{A_{E11}}, \underline{A_{E21}}, \underline{A_{E31}}, A_R)$ </p>
<p style="text-align: center;">UNO A MUCHOS Relación con participación obligatoria</p>		<p style="text-align: center;"> $E_1(\underline{A_{E11}}, A_{E12})$ $E_2(\underline{A_{E21}}, A_{E22}, A_{E11}, A_R)$ </p>

<p>UNO A MUCHOS Relación con participación opcional</p>		$E_1 (\underline{A_{E11}}, A_{E12})$ $E_2 (\underline{A_{E21}}, A_{E22})$ $R (A_{E11}, \underline{A_{E21}}, A_R)$ <p>ALTERNATIVA:</p> $E_1 (\underline{A_{E11}}, A_{E12})$ $E_2 (\underline{A_{E21}}, A_{E22}, A_{E11}^*, A_R^*)$
<p>Jerarquía de generalización</p>		$E_1 (\underline{A_{E11}}, A_{E12})$ $E_2 (\underline{A_{E11}}, A_{E21}, A_{E22})$ $E_3 (\underline{A_{E11}}, A_{E31}, A_{E32})$
<p>Relación con identificadores externos</p>		$E_1 (\underline{A_{E11}}, A_{E12})$ $E_2 (\underline{A_{E21}}, A_{E22}, \underline{A_{E11}})$

3.6. Diseño lógico específico

A partir del esquema lógico estándar (ELS) obtenido en la etapa anterior del diseño lógico, y teniendo en cuenta el modelo lógico específico (MLE) en el que se va a instrumentar la base de datos, se elabora el esquema lógico específico (ELE), que será descrito en el lenguaje de definición de datos del producto comercial que estemos utilizando.

La transformación del ELS al ELE lleva consigo un conocimiento del SGBD que va a ser aplicado a fin de:

- Ver en qué grado soporta el modelo relacional y, por tanto, el modelo lógico estándar (MLE).
- Adaptar el ELS a las características propias del producto que se va a utilizar (a su MLE).
- Definir el ELE en la sintaxis propia del SGBD.

3.7. Implementación del modelo relacional en SQL Server

3.7.1. Creación de bases de datos

En el ejemplo siguiente se creará una Base de Datos llamada MiBD que contiene un Archivo de Datos Principal (MiBD_root), un Archivo de Datos Secundario (MiBD_data1) que permanece de manera predeterminada en el grupo de archivos principal y un archivo de registro de transacciones (Log_data1).

Las sentencias SQL para crear la Base de Datos MiBD se muestra a continuación:

```
CREATE DATABASE MiBD

ON
(NAME = MiBD_root,                -- Archivo de datos principal
FILENAME = 'c:\mssql12k\MSSQL\data\mibdroot.mdf',
SIZE = 8MB,
MAXSIZE = 9MB,
FILEGROWTH = 100KB),

(NAME= MiBD_data1,                --Archivo de datos secundario
FILENAME = 'c:\mssql12k\MSSQL\data\mibddata1.ndf',
SIZE = 100MB,
MAXSIZE = 1500MB,
FILEGROWTH = 100MB)

LOG ON
(NAME = Log_data1,                --Archivo de registro
FILENAME: 'e:\log_files\logdata1.ldf',
SIZE = 1000MB,
MAXSIZE = 1500 MB,
```



```
FILEGROWTH = 100MB)
```

Nótese que, en este ejemplo, tanto el archivo de datos principales como el secundario están en la unidad C y que el archivo de registro está en la unidad E. Como ya se ha dicho, siempre se debe separar físicamente los archivos de datos de los archivos de registro para mejorar el rendimiento de E/S. adviértase también que han usado las extensiones de archivo recomendadas: .mdf, .ndf y .ldf. Las opciones SIZE, MAZSIZE y FILEGROWTH pueden especificarse en KB o en MB; de manera predeterminada es en MB.

Se debe recordar que la eliminación de una Base de Datos es una acción permanente. El comando T-SQL que se debe usar para borrar una Base de Datos es: DROP DATABASE. Los comandos para borrar la Base de Datos MiBD y todos sus archivos se muestra aquí:

```
USE master          --Se debe usar la Base de Datos master
GO                  para ejecutar el comando DROP DATABASE
DROP DATABASE MiBD  --El unico parametro es el nombre de la
GO                  base de Datos a eliminar
```

Tras borrar una Base de Datos se deberá hacer una nueva copia de seguridad de la Base de Datos Master, de manera que contenga la información actual de Base de Datos de usuario y no incluya información sobre la Base de Datos que acaba de ser eliminada. Adviértase también que una Base de Datos no puede borrarse mientras haya usuarios accediendo a ella. Todos los usuarios deben desconectarse de una Base de Datos antes de que sea aliminada.

3.7.2. Definición de tabla de una base de datos.

Una *tabla* es un objeto de una Base de Datos que almacena datos en una colección de filas y columnas. Una tabla se define por las columnas que contiene. Así, los datos pueden organizarse de forma similar a una hoja de cálculo, como se ilustra en la Tabla 7.1 que muestra una tabla de la Base de Datos llamada Info__Productos. (En los ejemplos de creará esta tabla en la base de Datos MiBD.)

La tabla Info_Productos se utiliza para almacenar información sobre cada producto que haya a la venta en una tienda. Cuando un producto se halle disponible para la venta, se añaden sus datos como una nueva fila de la tabla. Esta tabla contiene cinco columnas de información: *ID _ producto*, *Nombre _ producto*, *Descripción*, *Precio* e *ID _ marca*. La tabla 7.1 ilustra una muestra de tres filas de datos de la tabla Info_Productos.

Tabla 7.1. Base de Datos *Info_Productos*

ID _ producto	Nombre _ producto	Descripción	Precio	ID _ Marca
1	Tienda cinco pies	Para una o dos personas	80.00	12
2	Miniestufa	Alimentación con queroseno	20.00	33

3.7.3. Utilización de datos del sistema

Como se ha mencionado, se especifica un tipo de datos para cada columna de una tabla. La asignación de un tipo de de datos a una columna fija los siguientes atributos:

- La clase de datos que la columna puede contener como caracteres, enteros o imágenes.
- El tamaño o longitud de los datos de una columna.
- La precisión del número (sólo para tipos numéricos) – esto es, el número de dígitos que puede contener un número.
- La escala del número (sólo para tipos numéricos) – esto es, el número de dígitos que pueden almacenarse a la derecha del punto decimal.

Los tipos de datos también afectan a las columnas en las vistas, a los parámetros en los procedimientos almacenados, a las variables y a las funciones T-SQL que devuelven uno o más valores de datos. Los tipos de datos integrados que ofrece SQL Server se definen en la Tabla 7.2. SQL Server 2000 introduce tres tipos de datos: *bigint*, *sql_variant* y *table*. (Con pocas excepciones, expuestas en esta tabla, se utilizan los mismos tipos de datos de todos los objetos mencionados.)

Tabla 7.2. Tipo de datos del sistema SQL Server 2000

Tipo de datos	Descripción	Tamaño de almacenamiento
<i>bigint</i>	Un número entero de 8 bytes	8 bytes
<i>binary</i> <i>n</i>	Datos binarios de longitud fija de <i>n</i> bytes, donde <i>n</i> es un valor de 1 a 8000. Hay que usar <i>binary</i> cuando las entradas de datos de una columna se supongan próximas al mismo tamaño.	<i>n</i> + 4 bytes
<i>bit</i>	Tipo de datos entero que puede tomar los valores 1, 0 o NULL. Las columnas de bits no pueden tener índices en ellas.	1 byte para una tabla con columnas de hasta 8 bits, 2 bytes para una tabla con columnas de entre 9 y 16 bits, y así sucesivamente.
<i>char</i> <i>n</i>	Datos de caracteres no Unicode de tamaño fijo con longitud de <i>n</i> caracteres, donde <i>n</i> es un valor de 1 a 8000.	<i>n</i> bytes.

<i>cursor</i>	Una referencia a un cursor. Sólo puede usarse para variables y parámetros de procedimientos almacenados.	NO aplicable.
<i>datetime</i>	Datos de fecha y hora desde el 1 de enero de 1753 al 31 de diciembre de 9999 con una precisión de 3.33 milisegundos.	8 bytes.
<i>decimal [(p,[s])] o numeric [(p, [s])]</i>	Número de precisión y escalas finas. (El tipo de datos <i>numeric</i> es sinónimo de <i>decimal</i> .) La precisión (<i>p</i>) especifica el número total de dígitos que pueden almacenarse, tanto a la izquierda como a la derecha del punto decimal. La escala (<i>s</i>) especifica el número máximo de dígitos que pueden almacenarse a la derecha del punto decimal. La escala debe ser menor o igual que la precisión. La precisión mínima es 1, y la precisión máxima es 28 a no ser que se inicie SQL Server con el parámetro <i>-p</i> , en cuyo caso la precisión puede ser hasta 28.	De 5 a 17 bytes, dependiendo de la precisión.
<i>float [(n)]</i>	Datos numéricos de precisión flotante que pueden alcanzar desde $-1.79E+308$ hasta $1.79E+308$. el valor <i>n</i> es en número de bits utilizados para almacenar la mantisa del número de coma flotante y puede variar entre 1 y 53.	De 4 a 8 bytes, dependiendo de la precisión.
<i>image</i>	Utilizado para datos binarios de longitud variable mayor de 8000 bytes, con un máximo de $2^{31}-1$ bytes. Una entrada en una columna <i>image</i> es un puntero a la posición del valor de los datos <i>image</i> . Los datos se almacenan de manera separada de los datos de la tabla.	16 bytes para el puntero.
<i>integer o int</i>	Datos de un número entero desde -2^{31} (-2.147.483.684) hasta 2^{31} (2.147.483.684.)	4 bytes.
<i>money</i>	Valores de datos monetarios entre -2^{63} (-922.337.203.685.477.5808) y $2^{63}-1$ (922.337.203.685.477.5808), con exactitud hasta la diezmilésima de una unidad monetaria.	8 bytes.
<i>nchar [(n)]</i>	Datos de caracteres Unicode de longitud fija de <i>n</i> caracteres donde <i>n</i> es un valor entre 1 y 4000. Los caracteres Unicode usan 2 bytes por carácter y pueden soportar todos los caracteres internacionales.	16 bytes para el puntero y 2 bytes x el número de caracteres introducidos para los datos.

<i>ntext</i>	Datos Unicode de longitud variable con un máximo de $2^{30} - 1$ (1.073.741.823) caracteres. La entrada en la columna <i>ntext</i> es un puntero a la posición de los datos. El dato se almacena de manera separada de los datos de la tabla.	16 bytes para el puntero y 2 bytes x el número de caracteres introducidos para los datos.
<i>nvarchar</i>	Datos Unicode de longitud variable de <i>n</i> caracteres, donde <i>n</i> es un valor entre 1 y 4000. Recuérdese que los caracteres Unicode usan 2 bytes por carácter y pueden soportar todos los caracteres internacionales.	2 bytes x el número de caracteres introducidos.
<i>real</i>	Datos numéricos de precisión flotante que pueden variar entre $-3.40E+38$ y $3.40E+38$. El sinónimo del tipo real es flota (24).	4 bytes.
<i>smalldatetime</i>	Datos de fecha y hora desde el 1 de enero de 1900 hasta el 6 de junio de 2079, con exactitud hasta el minuto (menos preciso que el tipo de datos <i>datetime</i>).	4 bytes.
<i>smallint</i>	Valores de datos monetarios desde -214.748.3648 hasta 214.748.3648, con exactitud de hasta una diezmilésima de unidad monetaria.	4 bytes.
<i>Sql_variant</i>	Permite valores de diferentes tipos de datos. El valor del dato y los datos que describen ese valor – su tipo de datos base, escala, precisión, tamaño máximo e intercalación – se almacenan en esta columna.	El tamaño varía.
<i>sysname</i>	Un tipo de datos definido por el usuario de SQL Server especial y proporcionado por el sistema. El tipo de datos <i>sysname</i> está definido por SQL Server como <i>nvarchar (128)</i> , lo cual significa que puede almacenar 129 caracteres Unicode (o 256 bytes). Hay que usar <i>sysname</i> para columnas que almacenan nombre de objetos.	256 bytes.
<i>table</i>	Similar a utilizar una tabla temporal – la declaración incluye una lista de columnas y tipos de datos puede utilizarse para definir una variable local o para el valor de retorno de una función definida por el usuario.	Varía según la definición de la tabla.
<i>text</i>	Usado para caracteres no Unicode de longitud variable mayor de 8000 bytes. Una entrada en	16 bytes para el puntero.

una columna text puede albergar hasta $2^{31} - 1$ caracteres. Es un puntero a la posición de valor del dato. El dato se almacena de manera separada de los datos de la tabla.

<i>timestam</i>	Una columna <i>timestam</i> se actualiza automáticamente cada vez que se inserta o actualiza una fila. Cada tabla puede tener solo una columna <i>timestam</i> .	8 bytes.
<i>tinyint</i>	Datos enteros entre 1 y 255.	1 byte.
<i>uniqueidentifier</i>	Almacena un valor binario de 16 bytes que es un identificados exclusivo global (Global Unique Identifier, GUID).	16 bytes.
<i>varbinary</i>	Datos binarios de longitud variable de n bytes, donde n es un valor entre 1 y 8000. usar <i>varbinary</i> cuando se espera que las entradas de datos en una columna varíe considerablemente en tamaño.	Longitud real de los datos + 4 bytes.
<i>varchar [(n)]</i>	Datos de caracteres no Unicode de longitud variable de n caracteres donde n es un valor entre 1 y 8000.	Longitud real de los datos introducidos.

Antes de continuar se echará un vistazo al comando T-SQL CREATE TABLE, que se puede utilizar para crear la tabla Info_Productos mostrada en la tabla 7.1 anteriormente en este capítulo. Para este ejemplo se utilizarían tan solo tipos de datos del sistema y columnas de longitud fija.

Cuando se lanzan comandos T-SQL para crear una tabla, dicha tabla se creará en la base de datos que se esté utilizando actualmente. Para usar una base de datos en particular, ejecutar el comando USE nombre_basedatos, como se muestra en el código que sigue. En este ejemplo, la base de datos se llama MiBD. La palabra clave GO indica que cualquier comando previo debe ejecutarse en ese momento.

```
Use MiBD
Go
CREATE TABLE Info_Productos
(
ID_Producto          smallint,
Nombre_Producto     char(20),
Descripcion          char(30),
Precio              smallmoney,
ID_Marca            samllint
)
go
```

Obsérvese lo que ocurre en el código anterior. Después de que se dé el comando CREATE TABLE, se especifica el nombre de la tabla Info_Productos. Entre paréntesis, cada columna se define entonces listado el nombre de la columna seguido de su tipo de datos. Las longitudes de los dos tipos de datos char se fijan a 20 y 30 porque la mayoría de los nombres tendrán 20 caracteres o menos y la mayoría de descripciones tendrá 30 caracteres o menos. ID_Producto e ID_Marca se fijan ambos al tipo de datos smallint, en lugar de tinyint o int, porque anticipamos que habrá más de 255 tipos de productos y marcas pero menos de 32,767 (valor máximo de smallint). Dado que no se necesitarán valores por encima de 32,767, se está desperdiciando espacio si se utilizara el tipo int.

3.7.4. Creación de la tabla *info_productos* utilizando valores NULL

Volviendo al ejemplo de la tabla Info_Productos, se añadirá la opción sobre valores nulos a la definición de cada columna. Si se desea que una columna permita valores nulos, hay que añadir NULL tras el tipo de datos. Si no se quiere permitir valores nulos, hay que añadir NOT NULL tras el tipo de datos. Es buena práctica el especificar siempre si una columna deberá permitir valores nulos, excepto cuando se utiliza un tipo de datos definido por el usuario que ya ha sido definido con NULL o NOT NULL. Hacer esto ayudará a tomar la costumbre de considerar si las columnas deben admitir nulos.

En el ejemplo de la tabla Info_Productos, se permitirá solamente a la columna de descripción del producto aceptar valores nulos. La nueva sentencia CREATE TABLE se parecerá a ésta:

```
USE MiBD
Go
DROP TABLE Info_Productos
Go
CREATE TABLE Info_Productos
(ID_Producto      smallint NOT NULL,
Nombre_Producto  char (20) NOT NULL,
Descripcion      char (30) NULL,
Precio           smallmoney NOT NULL,
ID_Marca         smallint NOT NULL
)
GO
```

Ahora, si no se especifica la descripción de un producto pero los otros cuatro valores sí, al introducir los datos de un producto la nueva fila se insertará en la tabla con NULL para la entrada de la columna Descripción. Se deben introducir valores para las otras cuatro columnas ID_Producto, Nombre_Producto, Precio e ID_Marca, que no aceptan NULL. Si alguna de estas columnas está vacía, el intento de introducción de la nueva fila no tendrá éxito.

3.7.5 Agregar la propiedad IDENTITY a la tabla Info_Productos

Se va a añadir la propiedad IDENTITY a la tabla Info_Productos. En lugar de introducir valores para la columna ID_Productos, se la convertirá en una columna de identidad y se dejará que SQL Server

genere automáticamente los valores de identidad para asegurar su unicidad. El código T-SQL para crear la tabla se muestra aquí:

```
USE MiBD
GO
DROP TABLE INfo_Productos
GO
CREATE TABLE Info_Productos
(
ID_Producto          smallint IDENTITY (1,1) NOT NULL,
Nombre_Producto     char (20) NOT NULL,
Descripcion          char (30) NULL,
Precio               smallmoney NOT NULL,
ID_Marca             smallint NOT NULL
)
GO
```

La columna ID_Producto recibirá valores que comiencen por 1, con un incremento de 1 para cada fila sucesiva que se inserte en la tabla. Agregar la propiedad IDENTITY asegura que a cada producto se le asignará un número de identificación único sin que se requiera que un usuario introduzca uno. Le elección de 1 como incremento es arbitraria. Cualquiera que sea el incremento que se utilice, los valores de identidad serán únicos.

3.7.6 CREATE TABLE con DEFAULT

La creación de valor predeterminado en una columna usando el comando CREATE TABLE es la técnica predeterminada y estándar. La siguiente crea una tabla en MiBD que tiene un valor predeterminado definido para *columnA*, una columna de tipo *char*, y para *columnB*, una columna de tipo *int*.

```
USE MiBD
CREATE TABLE MyTable
(
columnA char (15) null
constraint DF_MyTable_columnA DEFAULT 'n/a',
columB int null
constraint DF_MyTable_columnB DEFAULT 0
)
Go
```

El valor predeterminado *n/a* para *columnA* es compatible con el tipo de datos *char* de dicha columna., y el valor predeterminado *0* para *columnB* es compatible con el tipo de datos *int*. Si no se especifica ningún valor para alguna de las columnas cuando se añade una fila a la tabla, se usará el valor predeterminado. Por lo tanto la única manera de que una de estas columnas contenga el valor NULL es si se inserta el NULL explícitamente. Los valores nulos se permiten ya que ambas columnas admiten nulos. Si las columnas se hubieran definido NOT NULL, no se podría insertar explícitamente el valor NULL.

Se va a cambiar el valor predeterminado para *columnA* de *n/a* a *not aplicable*. Hay que recordar que primero se debe eliminar el valor predeterminado existente y después añadir el nuevo. Para eliminar este valor predeterminado se usa el siguiente código:

```
ALTER TABLE MyTable
DROP CONSTRAINT DF_MyTable_ColumnA
```

Ahora se puede añadir el nuevo valor predeterminado, esta vez dándole un nombre, usando el comando siguiente:

```
ALTER TABLE MyTable
ADD CONSTRAINT DF_MyTable_columnA
DEFAULT 'not applicable' FOR columnA
Go
```

Cuando se cambia un valor predeterminado de una columna, todas las filas que había mantienen sus valores originales para esa columna, incluso si su valor es NULL. Sólo las filas nuevas que se inserten usan el nuevo valor predeterminado.

De nuevo, como se explicó, también se puede añadir una columna completa entera a una tabla existente y asignar a dicha columna un valor predeterminado usando el comando ALTER TABLE, como se muestra a continuación:

```
ALTER TABLE MyTable
ADD columnC tinyint NOT NULL
CONSTRAINT DF_MyTable_columnC DEFAULT 13
GO
```

Ahora la tabla de ejemplo *MyTable* tiene otra columna, *columnC*, que tiene un valor predeterminado de 13. Debido a que *columnC* es una columna nueva y a que está definida como NOT NULL, a cualquier fila de datos perteneciente a la tabla se le asignará el valor predeterminado 13 en la nueva columna.

3.7.7. Restricciones

Las restricciones automáticamente fuerzan la integridad de datos. Las restricciones definen reglas que determinan los valores de datos que se permiten en una columna. Esto permite restringir los valores que se pueden insertar en una columna, de forma que no se acepten los valores no válidos. Por ejemplo, se puede restringir los valores de una columna de tipo integer a valores entre 1 y 100 usando una restricción. De esta forma cualquier valor que este fuera de ese rango no se podrá insertar en la columna (Se deberá usar una restricción CHECK para crear esta restricción como ya se verá). Una restricción en una sola columna se denomina *restricción de columna*; solo restringe los valores de esa columna, una restricción que afecte a dos o más columnas se denomina *restricción de tabla*; asegura que la combinación de valores para las columnas de las restricciones cumplen los requerimientos de la restricción. Los cinco tipos de restricciones son NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY y CHECK.

NOT NULL

La restricción NOT NULL es bastante simple; de hecho, ya se ha usado en los ejemplos de este capítulo. La restricción NOT NULL se coloca en una columna para impedir que se inserten valores nulos en esa columna (lo opuesto a la restricción NULL, que permite los valores nulos.)

Usando T-SQL, se puede especificar NOT NULL tanto en el momento de la creación de la tabla o más tarde, cuando se está modificando una columna. Se deberá usar NOT NULL en vez de NULL siempre que sea posible debido a que las operaciones que tratan con valores nulos, como las comparaciones, necesitan más costo de procesamiento. Como se ha mencionado anteriormente en este capítulo, es mejor usar un valor predeterminado, cuando sea posible, que permitir valores nulos.

UNIQUE

La restricción UNIQUE asegura que una columna o un grupo de columnas no permitirán valores repetidos – en otras palabras, se impone la unicidad de los valores en la columna o el conjunto. Para imponer la unicidad, SQL crea de forma predeterminada un índice único y no agrupado en la columna o columnas que tiene la restricción UNIQUE. Sin embargo, se puede especificar si el índice debería estar agrupado o no agrupado. Hay que considerar que una tabla sólo puede tener un índice agrupado.

Una restricción UNIQUE se puede usar en cualquier columna que no sea parte de una restricción PRIMARY KEY (que se cubre en la siguiente sección), que también impone valores únicos.

La restricción UNIQUE se puede usar en columnas que permitan valores nulos, mientras que las restricciones PRIMARY KEY no pueden. Los valores nulos son ignorados por las restricciones UNIQUE. Una columna con una restricción UNIQUE puede referenciarse con una restricción FOREIGN KEY (descrita más adelante). Se permiten varias restricciones UNIQUE en una tabla con tal de que el número total de índices de esta tabla no exceda el 249 no agrupados y 1 agrupado. Para crear una restricción UNIQUE en una tabla T-SQL, se usa el comando CREATE TABLE o el comando ALTER TABLE. Por ejemplo, la siguiente sentencia crea la tabla *customer* con una restricción UNIQUE en la columna SSN como un índice no agrupado:

```
CREATE TABLE customer
(
  first_name      char (20) not null,
  mid_init        char (1) null,
  last_name       char (20) not null,
  SSN             char (11) not null
  CONSTRAINT UQ_ssn UNIQUE,
  cust_phone      char(10) null
)
go
```

La sentencia CREATE anterior usa una restricción de columna. El siguiente ejemplo vuelve a crear la tabla *customer*, esta vez con una restricción UNIQUE de tabla añadida denominada *UQ_full_name* en las columnas *first_name*, *mid_init* y *last_name*.

```
CREATE TABLE customer
(
first_name          char (20) not null,
mid_init           char (1) null,
last_name          char (20) not null,
SSN                char (11) not null
CONSTRAINT UQ_ssn UNIQUE,
cust_phone         char(10) null,
CONSTRAINT        UQ_full_name UNIQUE (first_name,
mid_init, last_name)
)
go
```

Una restricción de tabla UNIQUE (una restricción en más de una columna) asegura que la combinación de valores en las columnas es única. En este caso, no se puede introducir en la base de datos dos nombres de clientes que consistan en el mismo nombre, apellido e inicial. Una o dos de estas columnas pueden ser iguales pero no las tres.

PRIMARY KEY

Una restricción PRIMARY KEY se usa para especificar la *clave primaria* de una tabla, la columna o conjunto de columnas que identifican unívocamente a una fila. Debido a que identifican a una fila, la clave primaria nunca puede ser NULL. Esta es la diferencia entre la restricción PRIMARY KEY y una restricción UNIQUE, la cual permite valores nulos. Cuando se define una restricción PRIMARY KEY en un conjunto de columnas, la restricción indica que la combinación de los valores de dichas columnas deberá ser única para cada fila, que es parecido a la restricción UNIQUE en un conjunto de columnas. Y, como en la restricción UNIQUE, la restricción PRIMARY KEY no permite valores repetidos. Cuando se asigna una restricción PRIMARY KEY a una columna o a un conjunto de columnas, se crea automáticamente un índice único en la columna o columnas de la clave primaria. También se debe especificar si es un índice para clave primaria agrupado o no agrupado; de forma predeterminada es agrupado cuando no se especifica ninguno, siempre y cuando la tabla no tenga ya un índice agrupado.

Una tabla solo puede tener una restricción PRIMARY KEY. Una columna IDENTITY es una buena candidata para una clave primaria, como lo son cualquier otra columna o conjunto de columnas que sean únicas para cada fila. Por ejemplo, en la tabla de ejemplo *customer*, se puede crear la columna *SSN* como clave primaria en vez de crear una restricción UNIQUE en ella. La restricción PRIMARY KEY no permite los valores nulos e impondrá valores únicos en la columna *SSN*, y se creará automáticamente un índice agrupado en la columna de la clave primaria. Los siguientes comandos T-SQL muestran una forma de especificar la columna *SSN* como clave primaria cuando se está definiendo la tabla.

```

CREATE TABLE customer
(
first_name      char(20) not null,
mid_init        char(1)  null,
last_name       char(20) not null,
SSN             char(11) not null
constraint pk_SSN primary key,
cust_phone      char(10) null
)
go

```

Para añadir una restricción PRIMARY KEY a una tabla que no tiene una restricción PRIMARY KEY se usa el comando ALTER TABLE. El siguiente comando añade una restricción PRIMARY KEY a la tabla *customer*:

```

ALTER TABLE customer
ADD CONSTRAINT PK_SSN PRIMARY KEY (SSN)
GO

```

Para eliminar una restricción PRIMARY KEY se usa el comando ALTER TABLE con la sentencia DROP CONSTRAINT. A continuación se borra la restricción de la columna SSN:

```

ALTER TABLE customer
DROP CONSTRAINT PK_SSN
GO

```

Se debe observar que solo es necesario el nombre de la restricción para la sentencia DROP CONSTRAINT. Para cambiar una restricción PRIMARY KEY existe en una tabla usando los comandos de T-SQL, primero se debe eliminar la restricción existente y después modificar la tabla para añadir la nueva restricción. Esto se realiza usando las sentencias ALTER TABLE... DROP CONSTRAINT y ALTER TABLE... ADD CONSTRAINT.

FOREIGN KEY

Una restricción FOREIGN KEY define una *clave externa* que identifica una relación entre dos tablas. La columna o columnas clave externa de una tabla referencia a una *clave candidata* – *una o varias columnas* – de otra tabla. Cuando se inserta una fila en la tabla con la restricción FOREIGN KEY, los valores que se van a introducir en la columna o columnas que se han definido como clave externa se comprueban frente a los valores de la clave candidata de la tabla referenciada. Si ninguna fila de la tabla referenciada se ajusta a los valores de la clave externa, la nueva fila no se puede insertar. Pero si los valores de la clave externa que se van a insertar en la tabla existen en la clave candidata de la otra tabla, se insertará la nueva fila. También se permite insertar la fila si el valor que se va a insertar en la tabla con la restricción FOREIGN KEY es NULL.

La restricción FOREIGN KEY también se comprueba cuando se va a actualizar una fila ya sea en la tabla referenciada o en la tabla de la clave externa. No se puede actualizar el valor de una clave candidata ni el valor de una clave externa si esto provoca que no se cumpla la restricción. Hay una

excepción a esta regla cuando se actualiza la tabla referenciada: se puede actualizar la tabla usando la opción `ON UPDATE CASCADE` de la sentencia T-SQL `CREATE TABLE`.

Además, las restricciones `FOREIGN KEY` se comprueban cuando se quiere borrar una fila de la tabla referenciada. No se puede borrar una fila de una tabla referenciada si el valor de la columna de la clave externa está referenciado por una fila de la tabla de la clave externa (la tabla tiene la restricción `FOREIGN KEY`). En otras palabras, para cada fila de la tabla de la clave externa, debe existir la correspondiente fila en la tabla referenciada, y esta fila no puede borrarse mientras siga estando referenciada. También existe una excepción a esta regla: se puede borrar una fila de la tabla referenciada usando la opción `ON DELETE CASCADE` de la sentencia T-SQL `CREATE TABLE`.

Una clave externa sólo puede referenciar columnas que tengan las restricciones `PRIMARY KEY` o `UNIQUE` en la tabla referenciada. Si se intenta crear una clave que referencie una columna que no es parte de una de estas restricciones, SQL Server devolverá un mensaje de error. El tipo de datos y el tamaño de la columna o columnas de la clave externa, también debe coincidir con el de la columna o columnas referenciadas.

Para entender mejor las claves externas, se verán varios ejemplos. Primero, se crea una tabla llamada *ítems* que tiene una restricción `PRIMARY KEY` en la columna *item_id*, como se puede ver a continuación:

```
CREATE TABLE ítems
(
item_name      char(15)          not null,
item_id        smallint         not null identity (1,1),
price          smallmoney       null,
item_desc      varchar (30)     not null default 'none',
CONSTRAINT    pk_item_id       primary key (item_id)
)
go
```

Después se crea una tabla denominada *inventory* con una restricción `FOREIGN KEY` llamada *FK_item_id* que referencia a la columna *item_id* de la tabla *ítems*, como se muestra aquí:

```
CREATE TABLE inventory
(
store_id       tinyint          not null,
item_id        smallint         not null,
item_quantity  tinyint         not null,
CONSTRAINT     FK_item_id       FOREIGN KEY (item_id)
REFERENCES     ítems(item_id)
)
go
```

Para modificar una restricción `FOREIGN KEY` usando comandos T-SQL, primero se debe borrar la restricción antigua y después crear la nueva usando el comando `ALTER TABLE`. Este método es parecido al que se usó para modificar la restricción `PRIMARY KEY`. A continuación estarán los

comandos para primero eliminar la restricción original de la tabla *inventory* y después añadir la nueva restricción:

```
ALTER TABLE inventory
DROP CONSTRAINT FK_item_id
Go
```

```
ALTER TABLE inventory
ADD CONSTRAINT FK_item_id FOREIGN KEY (item_id)
REFERENCES items (item_id)
Go
```

También se puede habilitar o deshabilitar el uso de la restricción FOREIGN KEY si se quiere insertar una fila que no cumple con la restricción existente, se puede deshabilitar temporalmente la restricción, insertar la fila y entonces habilitar la restricción.

La palabra NOCHECK indica que la restricción deberá ignorarse (deshabilitarse) y la palabra CHECK indica que la restricción deberá habilitarse. Los siguientes comandos deshabilitan y vuelven a habilitar una restricción FOREIGN KEY usando las palabras clave NOCHECK y CHECK.

```
ALTER TABLE inventory
NOCHECK CONSTRAINT FK_item_id          -- Deshabilita la restricción
Go
```

-- la sentencia INSERT va aquí

```
GO
ALTER TABLE inventory
CHECK CONSTRAINT FK_item_id          -- vuelve a habilitar la
                                     restricción
go
```

ATENCIÓN: no se debe insertar una fila de datos que no cumpla con la restricción FOREIGN KEY a menos que sea absolutamente necesario. Si se hace, las futuras actualizaciones de la tabla podrían fallar.

La creación de una clave externa no crea un índice en la tabla; no obstante que sea un buen candidato para ser índice. Por lo tanto, por lo general necesita utilizar instrucciones CREATE INDEX después de crear tablas con claves externas. Puede hacerse referencia a las tablas de la misma base de datos sólo al crear restricciones de clave externa. Debe tener el permiso apropiado (SELECT o REFERENCES) en la tabla a la que hace referencia, y cualquier tabla individual puede tener un máximo de 253 claves externas apuntando a ella. No puede extenderse de este límite.

El siguiente código se crea una tabla de empleados y una de pedidos (los pedidos son introducidos por un empleado). Para verificar que un empleado válido haya introducido el pedido, puede programar la funcionalidad o declararla con claves externas. Luego, cuando alguien trate de eliminar un empleado, no le será permitido mientras existan pedidos para ese empleado.

```

CREATE TABLE emp
(emp_id int not null
CONSTRAINT pk_emp PRIMARY KEY,
emp_nombre char(30) not null)
go

```

```

CREATE TABLE pedidos
(pedido_id int not null
CONSTRAINT pk_pedido PRIMARY KEY,
emp_id int not null
CONSTRAINT fk_pedido foreign key (emp_id)
REFERENCES emp (emp_id)
ON DELETE NO ACTION
ON UPDATE NO ACTION
)
GO

```

El código del listado agrega a la ecuación una tabla de detalles de pedidos que utiliza las nuevas capacidades de SQL Server 2000 para realizar eliminaciones y actualizaciones en cascada

```

CREATE TABLE pedidos_detalle
(pedido_id int not null,
num_linea int not null,
CONSTRAINT pk_pedido_detalle primary key (pedido_id, num_linea),
Num_pieza int not null,
Precio money not null,
Constraint fk_pedido_detalle foreign key (pedido_id)
references pedidos(pedido_id)
on delete cascade
on update cascade
)

```

CHECK

La restricción CHECK se usa para restringir los valores permitidos de una columna a unos valores específicos. Los valores de una columna que se van a introducir o actualizar se confirman. Si devuelven TRUE de la condición lógica de búsqueda especificada en la restricción. Por ejemplo, si se quiere restringir el rango de valores que se permiten en la columna *price* de la tabla *items* entre \$0.01 y \$500.00, se usa la siguiente sentencia:

```

CREATE TABLE items
(
item_name char(15) not null,
item_id smallint not null identity (1,1),
price smallmoney null,
item_desc varchar(30) not null default 'none',
constraint pk_item_id primary key (item_id),
constraint ck_price check (price >= .01 and price <= 500.00)
)

```

```
)  
go
```

Obsérvese que se permite NULL en la columna *price* y que también tenemos una restricción CHECK en la columna. NULL se permite en la columna *price* a pesar de la restricción CHECK debido a que SQL Server puede distinguir un valor nulo de cualquier otro tipo de valor. También se puede observar que se ha dado el nombre *ck_price* a esta restricción. Como se ha visto, asignar un nombre a una restricción hace más fácil eliminar y volver a crear la restricción más tarde por el nombre, usando T-SQL. Por ejemplo, para cambiar el rango de valores de 1.00 hasta 1000.00, se usa la siguiente sentencia:

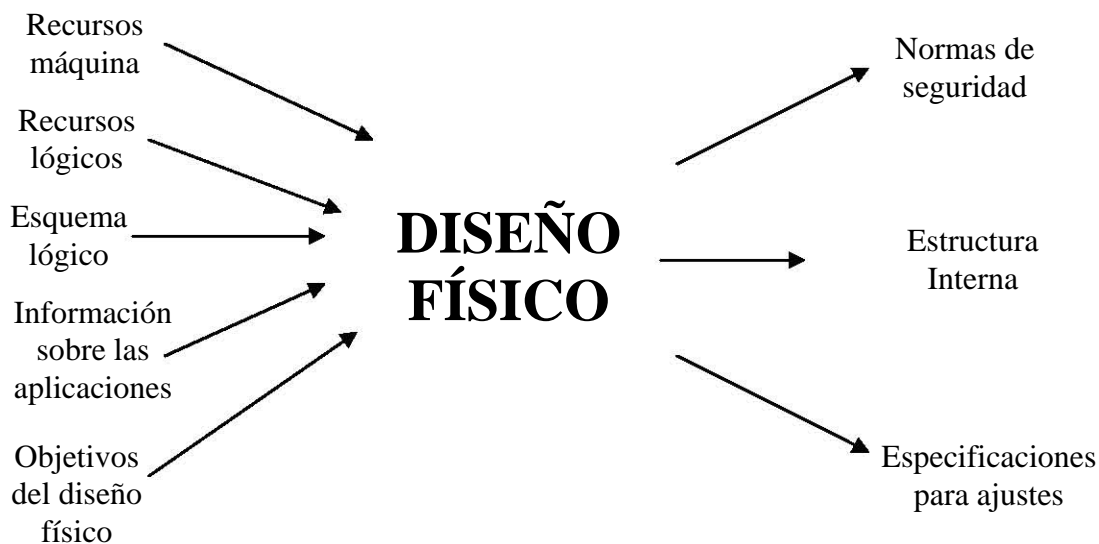
```
ALTER TABLE items  
DROP CONSTRAINT CK_price  
Go  
ALTER TABLE items  
ADD CONSTRAINT CK_price check (price >= 1.00 and price <= 1000.00)  
Go
```

4. Diseño físico

4.1 Objetivos y actividades del diseño físico

Las entradas y salidas del diseño físico se representan en la figura. En ella, como entradas, además de los objetivos de diseño físico (con sus correspondientes prioridades y cuantificados en lo posible), aparece el resultado de la etapa de diseño lógico (es decir, el esquema lógico), así como los recursos de máquina y de software (como el sistema operativo) disponibles, además de información sobre las aplicaciones que es preciso tener en cuenta a fin de optimizar aspectos como el tiempo de respuesta y definir ciertas políticas como la de seguridad.

A partir de estas entradas, en la etapa de diseño físico, se producirán, como salidas, una estructura interna junto con especificaciones que sirvan para realizar el ajuste o tuning de la base, así como las normas relativas a la seguridad de la misma.



El SGBD proporciona una estructura interna a partir de algunos parámetros que le proporciona el diseñador, el cual, posteriormente, puede irlos modificando a fin de realizar su ajuste (tuning) y optimizar, así, el rendimiento de la base de datos. El administrador ha de elegir entre todas las opciones disponibles, aquellas técnicas de estructuración física que permitan un acceso más eficiente, dados los requisitos concretos del correspondiente sistema de información.

Una vez diseñadas las aplicaciones se conocerá cuáles son las consultas más frecuentes y/o prioritarias a la base de datos, por lo que será conveniente crear las estructuras físicas que ayuden a localizar las filas seleccionadas en dichas consultas y a reducir así los accesos a disco. Entre los instrumentos más importantes (y generales) del diseño físico se encuentra la selección de los índices secundarios, que es uno de los problemas clave en la implementación física de una base de datos.

El aspecto global de seguridad de los datos está muy vinculado al propio concepto de lo que es la base de datos: "Un conjunto de datos integrados, adecuado a varios usuarios y a diferentes usos". Es el propio uso concurrente de los datos el que, en muchos casos, plantea problemas de seguridad que el administrador de la base de datos debe paliar, en la medida de lo posible, con las facilidades que le proporciona el SGBD.

4.2. Optimización y ajuste ó tuning de SQL Server.

La configuración de SQL Server 2000 es un proceso relativamente simple. Hay varias opciones disponibles para optimizar, pero lo bueno es que casi nunca hay que cambiar estas opciones. Sin embargo, es útil por lo menos ver lo que se tiene disponible y cuándo podría necesitar cambiar estas opciones.

```
EXEC sp_configure 'Show Advanced Options',1
GO
RECONFIGURE with Override
GO
EXEC sp_configure
```

A continuación se desplegará algo como lo siguiente:

name	minimum	maximum	config_value	run_value
-				
affinity mask	0	2147483647	0	0
allow updates	0	1	0	0
awe enabled	0	1	0	0
c2 audit mode	0	1	0	0
cost threshold for parallelism	0	32767	5	5
cursor threshold	-1	2147483647	-1	-1
default full-text language	0	2147483647	3082	3082
default language	0	9999	5	5
fill factor (%)	0	100	0	0
index create memory (KB)	704	2147483647	0	0
lightweight pooling	0	1	0	0
locks	5000	2147483647	0	0
max degree of parallelism	0	32	0	0
max server memory (MB)	4	2147483647	2147483647	2147483647
max text repl size (B)	0	2147483647	65536	65536
max worker threads	32	32767	255	255
media retention	0	365	0	0
min memory per query (KB)	512	2147483647	1024	1024
min server memory (MB)	0	2147483647	0	0
nested triggers	0	1	1	1
network packet size (B)	512	65536	4096	4096
open objects	0	2147483647	0	0
priority boost	0	1	0	0
query governor cost limit	0	2147483647	0	0
query wait (s)	-1	2147483647	-1	-1
recovery interval (min)	0	32767	0	0
remote access	0	1	1	1
remote login timeout (s)	0	2147483647	20	20
remote proc trans	0	1	0	0
remote query timeout (s)	0	2147483647	600	600

scan for startup procs	0	1	0	0
set working set size	0	1	0	0
show advanced options	0	1	1	1
two digit year cutoff	1753	9999	2049	2049
user connections	0	32767	0	0
user options	0	32767	0	0

A pesar de su enorme incremento en funcionalidad, SQL Server 2000 tiene, al igual que SQL Server 7.0 menos opciones que las versiones anteriores. Como lo mencionamos anteriormente, nunca tendrá que optimizar la mayoría de estas opciones. Muchas tienen el valor de 0 en las opciones `config_value` y `run_value`, lo que significa que SQL Server optimiza automáticamente esa opción por usted. A menos que exista una razón muy fuerte para modificar estas configuraciones automáticas, no debe hacerlo. Observe que el valor de `run_value` se aplica actualmente; el valor de `config_value` sería distinto si se solicitara un cambio pero no se hubiera reiniciado SQL Server ni ejecutado el comando `RECONFIGURE`.

Desde una perspectiva de rendimiento y optimización, tal vez sea conveniente modificar unos cuantos parámetros de configuración. Como lo dijimos anteriormente, y no está de más volverlo a repetir: lo más probable es que para todas las configuraciones de SQL Server, excepto las más extensas, no se necesitará modificar estos parámetros. No obstante, si administra un servidor SQL Server de ocho procesadores o más, tal vez necesite considerar estas opciones de configuración. Para facilitar el análisis de estas opciones, las agruparemos en cuatro categorías: opciones de procesador, opciones de configuración de memoria, opciones de E/S y opciones de consulta/índice.

4.2.1. Opciones de procesador

El primer conjunto de opciones que vamos a analizar se relaciona con el uso que da SQL Server a los procesadores en su equipo. Puede configurar la prioridad relativa del proceso SQL Server, cuáles CPUs en su equipo van a ser utilizadas por SQL Server, y el número total de subprocesos del sistema operativo que puede utilizar SQL Server en su equipo. También puede configurar el número máximo de CPUs que pueden utilizarse durante las operaciones de consultas en paralelo. Cada opción se muestra en un orden lógico que debe considerarse al realizar modificaciones en su configuración de SQL Server. En otras palabras, probablemente sea mejor cambiar la opción de aumento de prioridad (la cual analizaremos primero) antes de considerar cambiar el umbral de costo para la opción de paralelismo (la cual se analiza al último).

Aumento de prioridad (priority boost)

La opción de aumento de prioridad determina si los subprocesos de SQL Server se ejecutan con una prioridad normal (nivel de prioridad 8 en Windows 2000) o con una alta (nivel de prioridad 13 en Windows 2000). Cuando esta opción se establece a 0, se ejecutan con una prioridad normal; cuando se establece a 1, se ejecutan con una prioridad alta. Estas configuraciones son para un equipo con un solo procesador. En un equipo con varios procesadores, la prioridad normal de SQL Server a nivel del sistema operativo es de 15 y la prioridad alta es de 24.

Cuando cambia esta opción, tiene que reiniciar el servicio SQL Server para que el cambio tenga efecto.

Vigile cuidadosamente su equipo después de configurar esta opción, ya que podría afectar a cualquier otro proceso que se ejecute en su equipo SQL Server. No obstante, en un equipo SQL Server dedicado, se observa una mejora en el rendimiento al configurar esta opción.

Máscara de afinidad (affinity mask)

Esta opción es un mapa de bits que representa las CPUs que van a utilizarse para SQL Server en un sistema con varias CPUs. Por lo tanto, para asignar los procesadores 0, 1 y 3, debe establecer el patrón de mapa de bits a 00001011 (como se configuran de derecha a izquierda, siendo la posición 1 igual a la CPU 0, la posición 2 igual a la CPU 1, y así sucesivamente). Para simplificar esto, escriba el equivalente en base 10 (número normal), que viene siendo $11(8 \times 1) + (4 \times 0) + (2 \times 1) + (1 \times 1)$, empezando desde el primer bit encendido. De manera predeterminada, SQL aprovecha todos los procesadores de su sistema; cambie esta opción sólo cuando desee que algunos procesadores estén reservados para Windows 2000 o cualquier otro proceso que se ejecute en el servidor. La mayor parte del tiempo es conveniente permitir que SQL Server utilice todos los procesadores disponibles.

Opciones de consultas en paralelo

SQL Server 2000 tiene dos opciones de consultas en paralelo. En realidad, el paralelismo toma dos formas en SQL Server. La primera forma es la capacidad de SQL Server de ejecutar algunas partes de una consulta en paralelo y luego volver a combinar el procesamiento para producir un solo resultado. Este tipo de paralelismo sucede todo el tiempo en SQL Server.

El segundo tipo de consulta en paralelo -el que usted puede configurar- es la capacidad de SQL Server de dedicar procesadores a una sola consulta en paralelo. Este tipo de consulta en paralelo está disponible sólo en sistemas con varios procesadores y por lo general se ejecuta sólo cuando se tienen más procesadores que usuarios.

Por ejemplo, supongamos que en la noche del sábado a las 9 p.m. no hay usuarios en el sistema, y usted inicia un trabajo para ejecutar un informe extenso. Si tiene un sistema con cuatro procesadores, SQL Server puede optar por utilizar los cuatro procesadores para completar este informe con mucha más rapidez que si lo hiciera de otra forma. No obstante, una vez que inicia la consulta en paralelo se ejecuta hasta completarse, utilizando todas las CPUs configuradas. Si otro usuario se conecta y trata de ejecutar otra consulta, el rendimiento de la consulta del nuevo usuario se ve afectado debido a esta consulta en paralelo. Por esta razón, debe configurar el paralelismo cuidadosamente en caso de que tenga varios procesadores.

Grado máximo de paralelismo (max degree of parallelism). Esta opción especifica el número de subprocesos disponibles para ser utilizados en consultas en paralelo (entre 1 y 32). Un valor de 0 indica que pueden utilizarse todos los procesadores configurados (de la configuración de máscara de afinidad), con un subproceso por procesador. Un valor de 1 desactiva la consulta en paralelo. Cualquier otro valor especifica el número máximo de CPUs utilizadas para una consulta en paralelo. De nuevo, las consultas en paralelo se consideran sólo cuando su servidor SQL Server está relativamente libre de cualquier otra actividad.

En equipos con un solo procesador, esta opción se ignora. Los cambios a este parámetro de configuración tienen efecto inmediatamente, sin necesidad de reiniciar SQL Server.

Umbral de costo para paralelismo (cost threshold for parallelism). Esta opción especifica el número de segundos que dura una consulta si se ejecuta sin paralelismo, antes de considerar un plan de ejecución en paralelo. El valor predeterminado es de 5 segundos, y la opción puede establecerse hasta 32,767. En otras palabras, si una consulta tarda por lo menos 5 segundos en ejecutarse sin paralelismo, SQL Server examinará la opción para usar las facilidades de consulta en paralelo.

Esta opción se aplica sólo en equipos con varios procesadores cuando se establecen las opciones de máscara de afinidad y grados máximos de paralelismo de SQL Server para permitir que ocurran las consultas en paralelo. Los cambios a este parámetro tienen efecto inmediatamente, sin necesidad de reiniciar SQL Server.

4.2.2. Opciones de configuración de memoria

Analizaremos dos opciones de configuración de memoria: min server memory y max server memory. De manera predeterminada, SQL Server establece la opción min server memory en 0 y max server memory en 2,147,483,647 MB (un número realmente grande). Esta configuración significa que SQL Server utiliza una administración automática de memoria. A medida que SQL Server va necesitando más memoria, ocupa más. No obstante, para prevenir la paginación, el valor predeterminado de SQL Server evita que se expanda demasiado de manera que la carga total de memoria en el servidor ocupe más que la cantidad de RAM física -5 MB. Por lo tanto, en mi sistema de 128 MB, SQL Server nunca confirma más RAM de 123 MB (incluyendo Windows 2000 y cualquier otro proceso en ejecución). No obstante, puede establecer también estas opciones manualmente.

Por lo general, debe dejar estas configuraciones sin cambios; SQL Server 2000 realiza un excelente trabajo al asignar y quitar memoria en forma dinámica, según sea necesario. No obstante, si ejecuta en el mismo equipo otro programa que tenga un uso intensivo de memoria, tal como Microsoft Exchange Server, tal vez sea conveniente restringir la cantidad de RAM que se asigna a SQL Server. Siempre es conveniente tener un equipo SQL Server dedicado.

Memoria mínima del servidor (min server memory)

Si establece la opción min server memory, está indicando a SQL Server que nunca debe utilizar menos de n megabytes de memoria. Por ejemplo, si configura la opción min server memory en 50 (megabytes), al iniciar SQL Server, siempre pedirá 50 MB de memoria para iniciar.

Si también establece la opción max server memory a 50 MB, está configurando una cantidad "fija" de memoria que SQL Server siempre utilizará.

Memoria máxima del servidor (max server memory)

Si configura la opción max server memory, restringe la cantidad máxima de memoria que puede utilizar SQL Server en cualquier momento dado. De nuevo, el valor se establece en megabytes.

Memoria mínima por consulta (min memory per query)

La opción min memory per query especifica la mínima cantidad de memoria asignada para cada

usuario en SQL Server, por cada consulta que ejecute cada servidor. El valor predeterminado es de 1,024 KB, o 1 MB. Para la mayoría de los equipos SQL Server, este valor predeterminado es suficiente memoria. No obstante, si utiliza SQL Server 2000 como una base de datos de soporte de decisiones o de almacén de datos, tal vez sea conveniente incrementar este valor. Puede incrementar el rendimiento de las consultas realizando ordenamientos de datos o combinaciones grandes, al asignar más memoria antes de que las consultas la necesiten realmente (en lugar de buscar la memoria en tiempo de ejecución).

4.2.3. Opciones de E/S

No existen parámetros de configuración de E/S para optimizar en SQL Server 2000. En SQL Server 7.0 y versiones anteriores, se optimizaba el parámetro max async I/O, pero éste se eliminó en SQL Server 2000. Ahora SQL Server siempre inicia los subprocesos necesarios para realizar la E/S con el sistema operativo. Por lo tanto, lo mejor que puede hacer para el rendimiento de su disco y E/S de SQL Server es esparcir sus datos a través de varios discos y varios controladores.

4.2.4. Opciones de consulta/índice

Hay dos opciones disponibles para optimizar o limitar la ejecución de consultas, o para optimizar índices. Una de ellas es el factor de relleno. La otra opción es el límite de costo del regulador de consultas integrado (query governor cost limit).

El valor predeterminado para el límite de costo del regulador de consultas es de 0, lo que significa que se permite la ejecución de todas las consultas. No obstante, si establece esta opción en un número distinto de cero, está indicando a SQL Server que se evitará la ejecución de cualquier consulta que tenga la probabilidad de ejecutarse durante un número mayor de n segundos. Tenga cuidado al configurar esta opción, ya que fácilmente puede evitar que los usuarios ejecuten consultas que necesitan para realizar sus trabajos. Por otro lado, puede evitar que se ejecuten esas consultas descontroladas que duran una hora.

4.3. Indexación de la base de datos en SQL Server

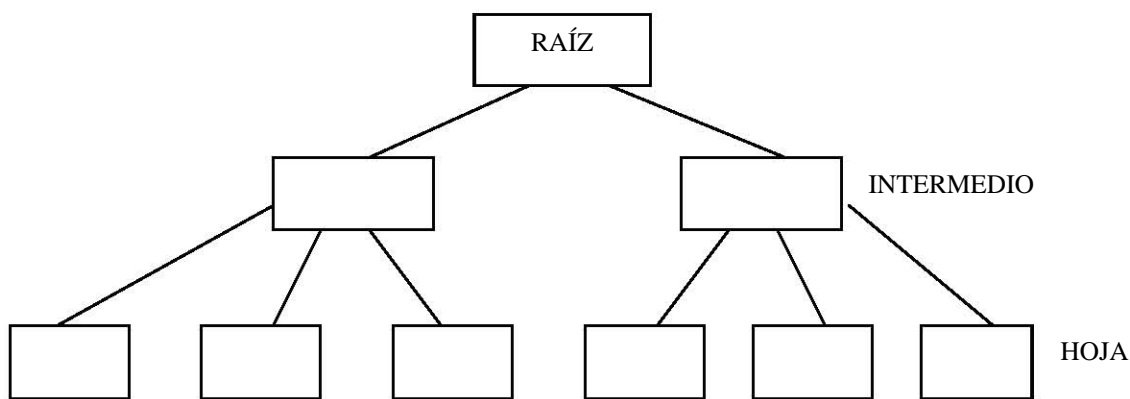
Los índices proporcionan un conjunto de apuntadores lógicos a sus datos, en forma muy parecida al índice que se encuentra en la parte posterior de un libro, y que le ayuda a encontrar cosas. Aunque todas las consultas que se examinaron anteriormente en esta semana funcionan sin índices (SELECT, INSERT, UPDATE, DELETE), por lo general se ejecutan con más rapidez si se utilizan índices.

Tal vez quiera utilizar los índices por muchas razones, siendo la más obvia la que acabo de mencionar –rapidez. Sin los índices, SQL Server accede los datos leyendo cada página de datos de cada tabla que haya especificado en su instrucción de SQL. Esta exploración de tabla (leer cada página de los datos) puede ser un excelente método de recuperación de datos. Por ejemplo, si una tabla es pequeña, o si está accediendo una porción extensa de la tabla, una exploración de tabla podría ser el mejor plan para tener acceso a los datos. No obstante, con mucha frecuencia el acceso a los datos es mucho más rápido con un índice. También puede agilizar la velocidad de las combinaciones de tablas.

Otra razón para crear un índice es para hacer valer la unicidad. Tener dos filas idénticas en una tabla no es una condición de error. No obstante, probablemente ésa no sea la manera en que las personas desean guardar datos. Imagine un sistema que lleva el registro de los clientes. Si no puede distinguir sus clientes, podría tener dificultades tratando de mantenerlos si les cobra en forma incorrecta. Existen varias opciones para identificar únicamente a sus clientes. Podría darles números, usar sus nombres y fechas de cumpleaños en conjunto, utilizar sus números de tarjeta de crédito o algún otro valor o conjunto de valores. Sin importar la elección que haga, la manera de indicarla a SQL Server es utilizando un índice único.

4.3.1. Arquitectura de índices

Un índice por lo general consiste en un conjunto de páginas conocidas como árbol binario. La apariencia de un árbol binario es como se muestra en la figura.

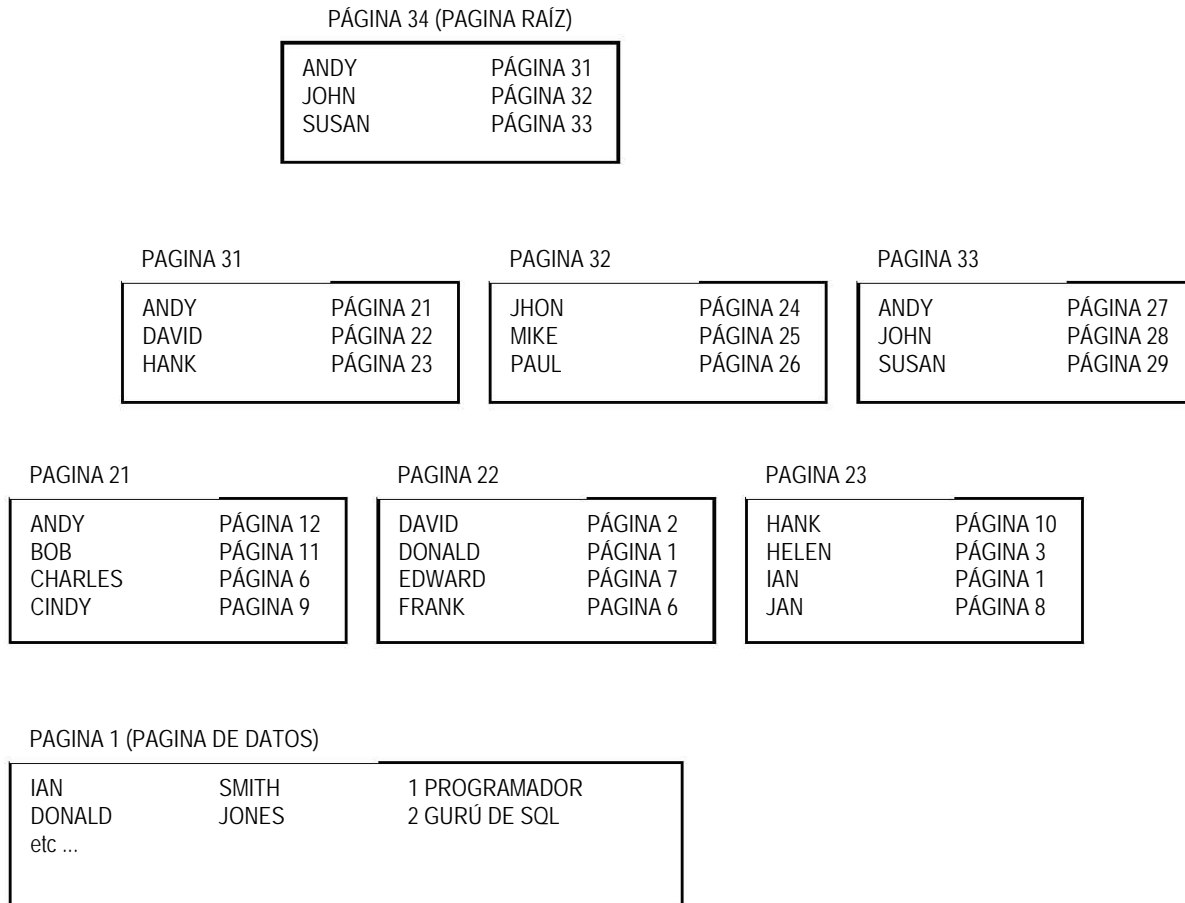


Como se mencionó anteriormente, un índice ayuda a encontrar datos rápidamente. Para encontrar una fila individual de datos, debe navegar por el árbol binario para buscar esa fila y a continuación llegará a la fila de datos individual. Se empieza con la página raíz. Un apuntador a la página raíz está ubicado en la tabla de sistema sysindexes (aunque parezca extraño, en la columna llamada root para los índices no agrupados). La página raíz contiene entradas de índices (los datos para la columna o columnas que haya indizado), así como apuntadores a cada página por debajo de la página raíz. Cada índice podría tener uno o más niveles intermedios. De nuevo, cada entrada tiene un valor de índice y un apuntador a la siguiente página inferior.

En las páginas de hoja (el nivel más bajo en el árbol), lo que encuentre depende de si una tabla tiene o no un índice agrupado. Lógicamente hablando, debe haber una entrada para cada fila de la tabla que se está indicando. Así como un apuntador a la página de datos y número de fila que tiene la fila actual de datos. Si la tabla también tiene un índice agrupado, cualquier índice no agrupado contiene los valores de clave de índice agrupado en vez de la información acerca de la página de datos y el número de fila. Más adelante describiremos los índices agrupados y no agrupados, pero por ahora piense en un índice agrupado como una manera de ordenar anticipadamente los datos entre sí. Las tablas que no tienen un índice agrupado se llaman datos apilados, y los índices no agrupados son estructuras de índice separadas que no ordenan directamente los datos.

Los datos en sí se guardan en páginas llamadas páginas de datos (no tiene caso hacer esto difícil). El tamaño de cada página es de 8,192, con un encabezado de 96 bytes. Por lo tanto, cada página tiene 8,096 bytes disponibles para almacenamiento. Cada página tiene la misma estructura básica.

La figura siguiente muestra un ejemplo de lo que podría ser un índice. Este índice se encuentra en la columna de nombre.



Cada nivel del índice es una lista con vínculos dobles. Cada página sabe acerca de la página anterior y de la página que va lógicamente después de ella. En los niveles raíz e intermedio del índice, cada valor de índice es el primer valor en la página del siguiente nivel inferior. El nivel de hoja del índice contiene una entrada para cada fila de la tabla. Observe que el índice se ordena con base en la columna (o columnas) elegida como su clave de índice. Este ordenamiento no cambia el orden físico de los datos.

Cuando modifica datos en la tabla, también se modifica cada índice de esa tabla. SQL Server garantiza la consistencia entre los datos de sus tablas e índices. Esto es bueno, en el sentido de que es conveniente tener una excelente integridad en los datos. No obstante, también significa que las operaciones INSERT, UPDATE y DELETE que podrían haber sido bastante veloces antes, ahora podrían tener un poco más de trabajo por hacer y podrían ejecutarse un poco más lentas. Para agregar una nueva fila con la instrucción ISERT por lo general se necesitan dos E/S (entradas/salidas) –una para los datos y una para el registro. Si tiene dos índices en la tabla, agregar una fila requiere de al menos dos E/S más, y tal vez más que eso. Debe equilibrar sus necesidades entre las modificaciones de datos y unas consultas más veloces.

¿Cómo se utiliza un índice? Busque una fila de datos en la figura anterior. Pretenda que el índice está en el nombre. Para encontrar la fila que tenga Donald como nombre, debe buscar en la página raíz. Como Donald es “menor” que John, se sigue la entrada para Andy hacia la página 6. En la página 6 descubre que Donald es “mayor” que David pero “menor” que Hank, por lo que avanza a la página 22. En la página 22 descubre que la entrada para Donald apunta a la página 1 (que es una página de datos, ya que éste es el nivel de hoja del índice). En SQL Server también se busca el número de la fila que contiene a Donald como nombre, pero se omitió de la figura por razones de simplicidad. Ahora puede leer la fila de la página. SQL Server utiliza un índice para buscar datos exactamente de la misma manera.

Opciones de índice

Se tienen disponibles varias opciones para los índices. Debe especificar estas opciones y comprenderlas antes de poder crear sus índices.

Índices agrupados

Existen dos opciones para el almacenamiento físico de sus índices. El primer tipo se conoce como agrupado. Un índice *agrupado* vuelve a ordenar físicamente los datos. En lugar de tener una estructura de índice completamente separada (como la que se describió anteriormente), el nivel de hoja del índice está compuesto por los datos. El acceso a los datos por medio de un índice agrupado es casi siempre más rápido que utilizar un índice no agrupado, ya que no es necesaria la búsqueda adicional de la página/fila de datos desde el nivel de hoja del índice.

Como los datos están ordenados físicamente en el orden de la clave de índice, sólo se puede tener un índice agrupado en una tabla (no es conveniente mantener varias copias de los datos). Ya que sólo está disponible un índice agrupado, debe elegirlo con cuidado. Esta elección puede ser bastante complicada, pero aquí se incluye algunos lineamientos básicos.

Una importante cuestión que surge con los índices agrupados es el espacio libre. La creación de un índice agrupado requiere que por lo menos esté disponible un 120 por ciento del tamaño de la tabla como espacio de trabajo temporal. Este espacio libre debe existir en la base de datos en donde se va a crear el índice. Para crearlo, SQL Server copia la tabla, ordena la copia según el orden de los valores del índice (en orden ascendente), crea las estructuras de índice (la página raíz y cualquier página intermedia que se necesite) y luego quita la tabla original. Al completarse esta operación, el índice agrupado ocupa aproximadamente un 5 por ciento más de espacio que la misma tabla. La sobrecarga (las páginas que no son de datos) de este tipo de índice es relativamente pequeña, pero depende del tamaño de los valores que se van a indicar.

Tal vez esta sobrecarga no sea tan mala, ya que se necesita el espacio libre sólo durante la creación del índice, pero puede ser difícil justificarlo. Si tiene una base de datos de 500 MB, pero una tabla de la base de datos es de 100 MB, y se desea crear un índice agrupado en esa tabla, necesitaría por lo menos 120 MB de espacio libre en la base de datos.

Otra cuestión de mucha importancia en SQL Server 2000 es que los valores de clave (columnas indizadas) que elija para su índice agrupado se “llevan” junto con los índices no agrupados. Por lo tanto, si elige un índice agrupado amplio –por ejemplo, un char(30)– no sólo le lleva más tiempo buscar en el índice agrupado, sino que también todos sus índices no agrupados tienen que llevar el

valor char(30) de su clave de índice agrupado en cada fila de los índices no agrupados. Ésa es una cantidad considerable de sobrecarga, por lo que debe mantener sus claves de índices agrupados lo más pequeñas que sea posible. También debe asegurarse de que la clave agrupada que seleccione no se actualice frecuentemente, ya que todos sus índices no agrupados necesitan actualizarse al cambiar los valores de índice agrupado.

La otra cuestión a considerar es el orden en el que debe crear sus índices. Como la clave de índice agrupado forma parte de los valores de clave para cada uno de los índices no agrupados, cada uno de éstos necesita volver a generarse. Por lo tanto, siempre debe crear primero sus índices agrupados.

La figura muestra un ejemplo de un índice agrupado. Los datos se ordenan según el orden de la clave de índice (que es el nombre), y los datos en el nivel de hoja del índice (la página de datos) se ordenan según el nombre.

PAGINA 34 (PAGINA RAÍZ)

ANDY	PÁGINA 31
JOHN	PÁGINA 32
SUSAN	PÁGINA 33

PAGINA 31 (INTERMEDIA)

ANDY	PÁGINA 21
DAVID	PÁGINA 22
HANK	PAGINA 23

PAGINA 32

JOHN	PÁGINA 24
MIKE	PÁGINA 25
PAUL	PAGINA 26

PAGINA 21 (PAGINA DE DATOS)

ANDY	JOHNSON	14 PROGRAMADOR
BILL	SMITH	22 GURÚ DE SQL
BOB	GREENE	9 GERENTE
CAL	ANDERSON	12 DBA
CHARLES	ALLEN	35 PROGRAMADOR
CINDY	SHELLHOR	28 GERENTE

PAGINA 22 (PAGINA DE DATOS)

DAVID	JONES	8 PROGRAMADOR
DON	JACKSON	16 PROGRAMADOR
DONALD	ANDRES	3 CAPTURA DE DATOS
ELLIS	WASHINGTON	11 GERENTE
ETHAN	ALLEN	9 RECEPCIONISTA
FRANK	THOMAS	32 VICEPRESIDENTE

Índices no agrupados

Un índice no agrupado es básicamente igual a un índice de árbol binario estándar. Cada índice tiene una página raíz, uno o más niveles de páginas intermedias y un nivel de hoja, el cual contiene una fila por cada fila de la tabla. Los índices no agrupados requieren en general de un mayor espacio que los índices agrupados, pero ocupan mucho menos espacio durante el proceso de creación.

Puede tener hasta 249 índices no agrupados en una sola tabla. El orden en que los cree no es importante. Un índice no agrupado no cambia el orden de los datos, como lo hace un índice agrupado. Las filas del nivel de hoja del índice se ordenan según el orden de las columnas seleccionadas como parte del índice. Cada fila contiene un apuntador a la combinación número de página/número de fila de los datos en la tabla, en caso de que no exista un índice agrupado, o al valor de la clave del índice

agrupado, si la tabla también tiene un índice agrupado. En la figura se muestra un ejemplo de un índice no agrupado.

PÁGINA 34 (PAGINA RAÍZ)

ANDY	PÁGINA 31
JOHN	PÁGINA 32
SUSAN	PÁGINA 33

PAGINA 31

ANDY	PÁGINA 21
DAVID	PÁGINA 22
HANK	PÁGINA 23

PAGINA 32

JHON	PÁGINA 24
MIKE	PÁGINA 25
PAUL	PÁGINA 26

PAGINA 33

ANDY	PÁGINA 27
JOHN	PÁGINA 28
SUSAN	PÁGINA 29

PAGINA 21

ANDY	PÁGINA 12
BOB	PÁGINA 11
CHARLES	PÁGINA 6
CINDY	PAGINA 9

PAGINA 22

DAVID	PÁGINA 2
DONALD	PÁGINA 1
EDWARD	PÁGINA 7
FRANK	PAGINA 6

PAGINA 23

HANK	PÁGINA 10
HELEN	PÁGINA 3
IAN	PÁGINA 1
JAN	PÁGINA 8

PAGINA 1 (PAGINA DE DATOS)

IAN	SMITH	1 PROGRAMADOR
DONALD	JONES	2 GURÚ DE SQL
etc ...		

Índices únicos/no únicos

La unicidad determina si se permiten valores duplicados en su índice. Por ejemplo, en el índice de nombre que vio anteriormente, no se permitiría que dos personas tuvieran el mismo nombre si el índice fuera único. Los índices de SQL Server son no únicos de manera predeterminada, lo que significa que se permiten valores duplicados.

Si sus datos lo soportan, hacer que un índice sea único puede mejorar considerablemente el rendimiento al utilizarlo. Cuando el valor que busca no se encuentra, no son necesarias más búsquedas (si sabe que sólo existe una entrada, al encontrarla puede dejar de buscar más).

Los índices agrupados son muy buenos candidatos para ser índices únicos, ya que SQL Server siempre obliga internamente a que los índices agrupados sean únicos. Si usted no crea un índice agrupado único, SQL Server genera un valor de clave adicional oculto para forzar la unicidad del índice. Así que, ¿para qué hacer que SQL Server genere esta clave si usted tiene una clave que también es única y que es un buen candidato?

Índices de una sola columna/de varias columnas

Muchos índices tienen sólo una columna; sin embargo, puede crear fácilmente un índice de varias columnas. Este tipo de índices pueden ser bastante útiles, ya que se puede reducir el número de índices utilizados por SQL Server y así obtener un rendimiento más veloz. Si especifica ambas columnas en conjunto frecuentemente durante las consultas, las columnas son un candidato excelente para convertirse en un *índice compuesto* (es sólo otro nombre para un índice con varias columnas). Los índices compuestos pueden ser agrupados o no agrupados, pueden contener de 2 a 16 columnas y una anchura de hasta 900 bytes.

La desventaja aquí es que si crea un índice demasiado amplio, no será útil ya que explorar la tabla podría llevar menos que tiempo utilizar el índice. Lamentablemente, la indización implica muchas concesiones y raras veces presenta elecciones obvias para varias aplicaciones.

Índices ascendentes/descendentes

En todas las versiones anteriores de SQL Server, siempre se supuso que los índices eran ascendentes (una manera muy natural de pensar acerca de los índices). Por lo tanto, para ordenar el alfabeto, la A sería primera y la Z vendría al final. No obstante, algunas veces es necesaria una lista ordenada en forma descendente (es decir, ordenada de la Z a la A). SQL Server 2000 ofrece un soporte completo para los índices descendentes.

La figura muestra un índice no agrupado en el nombre, pero ahora el índice se guarda en orden descendente. Las filas se encuentran de la misma manera, simplemente navegando a través del índice, empezando por la página raíz. La ventaja aquí es que si desea que su consulta devuelva los datos en orden descendente, puede hacerlo sin necesidad de un orden adicional.

PÁGINA 34 (PAGINA RAÍZ)

SUSAN PAGINA 33

SUSAN	PÁGINA 31
JOHN	PÁGINA 31
ANDY	PAGINA 33

PAGINA 31

WILLIAN	PÁGINA 21
TUCKER	PÁGINA 22
SUSAN	PÁGINA 23

PAGINA 32

PAUL	PÁGINA 24
MIKE	PÁGINA 25
JOHN	PÁGINA 26

PAGINA 33

HANK	PÁGINA 27
DAVID	PÁGINA 28
ANDY	PÁGINA 29

PAGINA 21

WILLIAM	PÁGINA 2
WALLY	PÁGINA 1
VICTOR	PÁGINA 7
VALERIE	PAGINA 6

PAGINA 22

TUCKER	PÁGINA 2
THOMAS	PÁGINA 1
FERRY	PÁGINA 7
TARA	PAGINA 6

PAGINA 23

SUSAN	PÁGINA 10
STEVE	PÁGINA 3
SCOTT	PÁGINA 1
RICHARD	PAGINA 8

PAGINA 1 (PAGINA DE DATOS)

SCOTT	SMITH	1 PROGRAMADOR
THOMAS	JHONES	2 GURÚ DE SQL
etc ...		

4.3.2. Índices de SQL Server

SQL Server implementa árboles binarios para generar sus índices. Utilice la instrucción **CREATE INDEX** para crear los índices que necesite:

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX nombre_indice
ON [propietario.][nombre_tabla | nombre_vista]
(nombre_columna [ASC|DESC] [...n])
[WITH
[PAD_INDEX] [[,] FILLFACTOR = X]
[[,] IGNORE_DUP_KEY]
[[,] DROP_EXISTING]
[[,] STATISTICS_NORECOMPUTE]
[[,] SORT_IN_TEMPDB]]
[ON grupoarchivos]
```

En esta sintaxis:

- **UNIQUE** especifica que no se permiten valores duplicados para este índice. El valor predeterminado es no único y se permiten entradas duplicadas en el índice.
- **CLUSTERED** especifica que los datos en sí se ordenarán físicamente y se convertirán en el nivel de hoja del índice. Los valores del índice agrupado deben ser únicos. Si crea un índice agrupado **UNIQUE**, no hay problema. No obstante, si crea un índice agrupado no único, se agrega un “unificador” de 4 bytes a cada clave de índice agrupado, para garantizar la unicidad.

- NONCLUSTERED especifica que un índice binario normal se va a crear como un objeto completamente separado. Éste es el tipo de índice predeterminado.
- nombre_índice es el nombre único de SQL Server para este objeto.
- nombre_tabla es el nombre de la tabla que contiene las columnas que desea indizar.
- nombre_vista es el nombre de la vista que contiene las columnas que desea indizar.
- nombre_columna es el nombre de la columna (o columnas) que van a indizarse. Puede crear un índice que tenga hasta 16 columnas y hasta 900 bytes de anchura. Las columnas no pueden ser de tipo text, image, bit ni ntext. Sin embargo, la columna puede ser una columna calculada, característica nueva en SQL Server 2000.
- ASC | DES indica si la columna se va a ordenar en forma ascendente (ASC) (por ejemplo, A-Z) o descendente (DES) (por ejemplo, Z-A). La opción predeterminada es ascendente, lo cual es consistente con versiones anteriores que no tenían la opción descendente.
- grupoarchivos es el nombre del grupo de archivos en el que debe crearse el índice. Si no se especifica grupoarchivos, el índice se crea en el grupo de archivos predeterminado.

Por ejemplo, la siguiente línea de código crea una tabla llamada misautores de la base de datos pubs y luego copia todos los datos de la tabla authors. A continuación, el código crea un índice en la columna au_id de la tabla misautores. El índice agrupado resultante hace valer la unicidad.

USE pubs

```
--CREA LA TABLA
CREATE TABLE dbo.misautores (
au_id int NOT NULL,
au_apellido varchar (40) NOT NULL,
au_nombre varchar (20) NOT NULL,
telefono char (12) NOT NULL,
direccion varchar (40) NULL,
ciudad varchar (20) NULL,
estado char (2) NULL,
codpostal char (5) NULL,
contrato bit NOT NULL)

--Copia los datos de la tabla authors a misautores
INSERT misautores SELECT * FROM authors

--Crea en índice agrupado único en la tabla misautores
CREATE UNIQUE CLUSTERED INDEX miauind ON misautores (au_id)
```

El siguiente código crea un índice descendente no agrupado y no único en la columna au_nombre de la misma tabla:

USE pubs

```
CREATE INDEX minombindice ON misautores (au_nombre DESC)
```

Observe que el siguiente comando de Transact-SQL es funcionalmente idéntico y podría ser un poco más obvio:

USE pubs

CREATE NONCLUSTERED INDEX minombindice ON misautores (au_nombre)

Las opciones FILLFACTOR y PAD_INDEX

La opción FILLFACTOR (factor de relleno) especifica qué tan llena debe estar cada página en el nivel de hoja de un índice. El factor de relleno predeterminado es de 0. Como fillfactor es un parámetro de configuración, debe asegurarse de verificar que no haya sido modificado. Recuerde que puede verificarlo ya sea desde el Administrador corporativo de SQL Server, o puede ejecutar sp_configure sin especificar parámetros. La salida que obtenga debe ser similar a la siguiente:

```
EXEC sp_configure
go
```

name	minimum	maximum	config_value	run_value
-				
affinity mask	0	2147483647	0	0
allow updates	0	1	0	0
awe enabled	0	1	0	0
c2 audit mode	0	1	0	0
cost threshold for parallelism	0	32767	5	5
cursor threshold	-1	2147483647	-1	-1
default full-text language	0	2147483647	3082	3082
default language	0	9999	5	5
fill factor (%)	0	100	0	0
index create memory (KB)	704	2147483647	0	0
lightweight pooling	0	1	0	0
locks	5000	2147483647	0	0
max degree of parallelism	0	32	0	0
max server memory (MB)	4	2147483647	2147483647	2147483647
max text repl size (B)	0	2147483647	65536	65536
max worker threads	32	32767	255	255
media retention	0	365	0	0
min memory per query (KB)	512	2147483647	1024	1024
min server memory (MB)	0	2147483647	0	0
nested triggers	0	1	1	1
network packet size (B)	512	65536	4096	4096
open objects	0	2147483647	0	0
priority boost	0	1	0	0
query governor cost limit	0	2147483647	0	0
query wait (s)	-1	2147483647	-1	-1
recovery interval (min)	0	32767	0	0
remote access	0	1	1	1
remote login timeout (s)	0	2147483647	20	20
remote proc trans	0	1	0	0
remote query timeout (s)	0	2147483647	600	600
scan for startup procs	0	1	0	0
set working set size	0	1	0	0
show advanced options	0	1	1	1
two digit year cutoff	1753	9999	2049	2049

```

user connections          0          32767          0          0
user options              0          32767          0          0

```

Busque el nombre fillfactor en la salida, y compruebe que las columnas config_value y run_value muestren una configuración en fillfactor de 0.

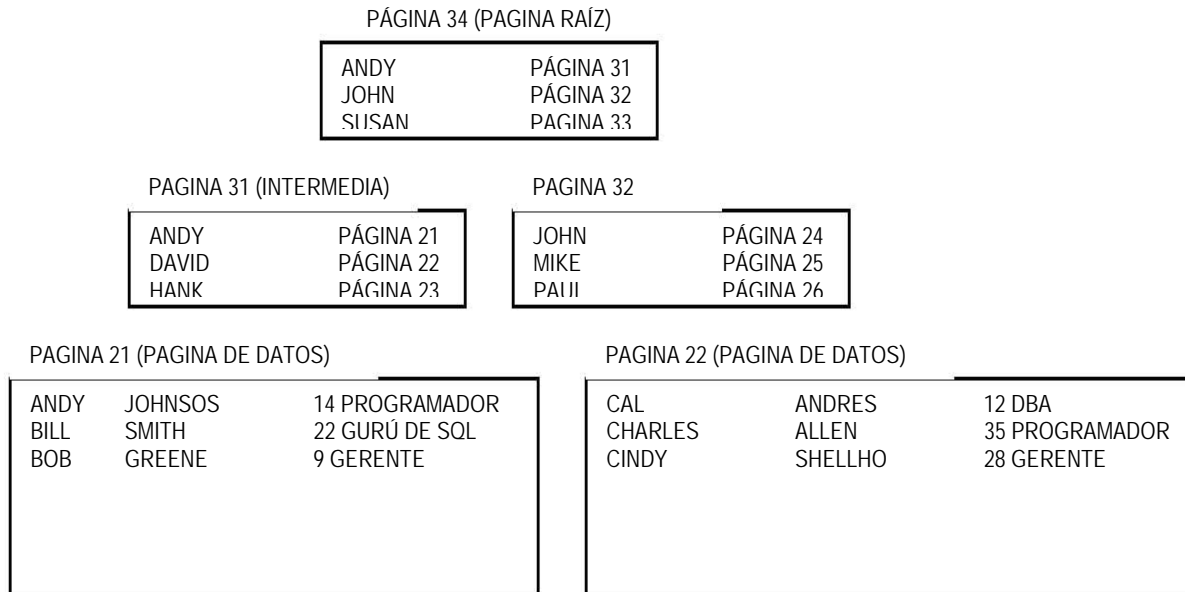
Si no especifica FILLFACTOR en su instrucción CREATE INDEX, se utiliza el valor predeterminado (generalmente 0). Un valor de 0 significa que las páginas de hoja de su índice están casi llenas, pero que las páginas que no son de hoja (las páginas intermedias y la página raíz) aún tienen espacio para cuando menos dos filas. Si FILLFACTOR es de 100, todas las páginas de hoja están llenas un 100 por ciento, sin espacio para filas adicionales. De nuevo, las páginas raíz e intermedias aún tienen espacio para dos filas adicionales. Cualquier otro valor es el porcentaje de cada página de hoja que debe llenarse con filas. SQL Server redondea el porcentaje al tamaño de fila más cercano, por lo que raras veces se obtiene el porcentaje que se pidió, pero el resultado es tan cercano como SQL Server pueda lograrlo.

Si crea un índice agrupado con un FILLFACTOR de 50, cada página estaría un 50 por ciento llena. En la figura puede ver que la página de hoja del índice agrupado está llena sólo a la mitad. El código podría verse de la siguiente manera:

```

CREATE INDEX aunombreindice ON authors (au_fname)
WITH FILLFACTOR = 50
GO

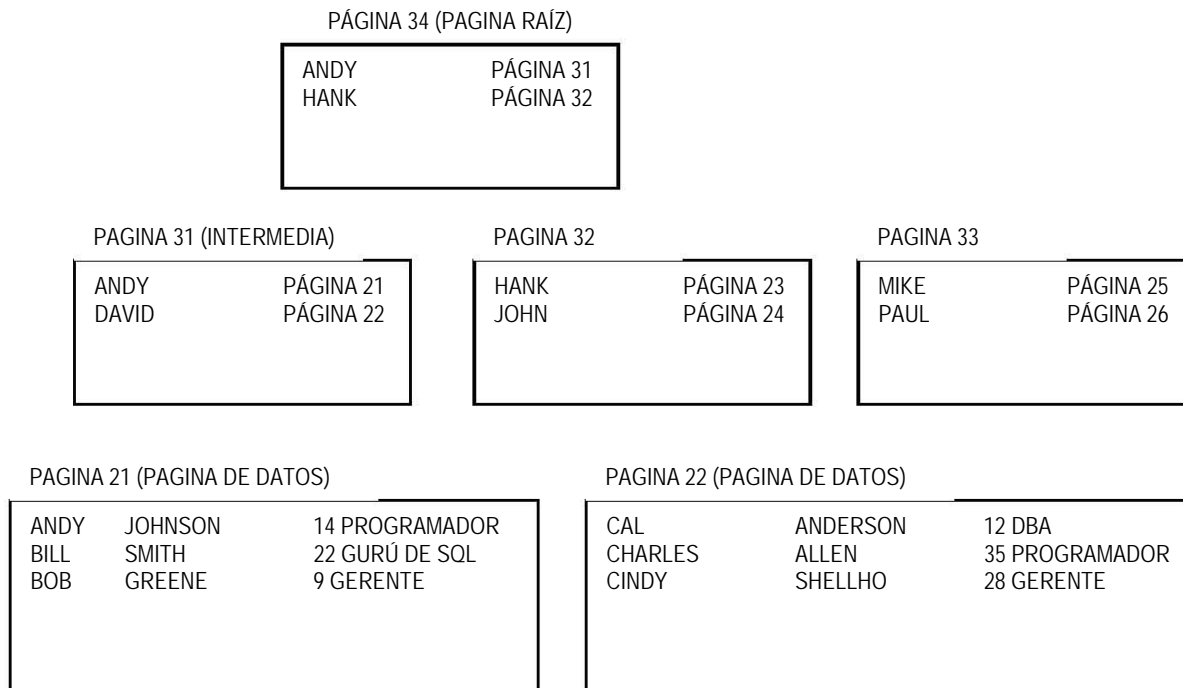
```



Al utilizar la opción PAD_INDEX con un factor de relleno, especifica que se va a aplicar el factor de relleno en las páginas que no sean hojas del índice, junto con las páginas de hoja. La manera más sencilla de comprender las opción PAD_INDEX es viendo un ejemplo.

Si crea el mismo índice de la figura anterior pero agrega la opción PAD_INDEX, también se aplicará ahora el factor de relleno a las páginas que no sean de hoja. Observe que si sus filas no encajan perfectamente, SQL Server se acerca lo más posible al factor de relleno que usted haya solicitado. Por ejemplo, el siguiente código podría crear un índice como el de la figura:

```
CREATE INDEX aunombreindice ON authors (au_fname)
WITH FILLFACTOR = 50, PAD_INDEX
GO
```



Hasta ahora hemos visto cómo funcionan FILLFACTOR y PAD_INDEX, pero no hemos visto por qué es conveniente utilizarlos. A menudo, puede ser bastante útil especificar un factor de relleno. Para comprender el motivo, debe conocer un par de términos. El primero es división de página. Cuando una página de SQL Server está llena y debe colocarse otra fila en esa página, ocurre una división de página. Se asigna una nueva página al índice o tabla, y se mueve el 50 por ciento de las filas a la nueva página. Entonces se agrega la nueva fila en la ubicación apropiada. Como podría imaginarse, esta operación puede ser costosa si ocurre con frecuencia. Si utiliza un factor de relleno al crear su índice, puede permitir que se agreguen nuevas filas sin ocasionar divisiones de página. Configurar el factor de relleno apropiado muy probablemente mejorará el rendimiento. Sin embargo, se requiere de experiencias para obtener el factor de relleno apropiado.

Esta opción puede ser especialmente útil con los índices agrupados, ya que el nivel de hoja del índice está compuesto por las páginas de datos en sí. No obstante, debe tener en cuenta que si aplica un factor de relleno tal como 50 al índice agrupado, el tamaño de la tabla se duplica aproximadamente. Este método puede tener un impacto muy dramático en el uso del espacio, y debe llevarse a cabo con cierta precaución.

La división de páginas en datos apilados (tablas sin un índice agrupado) puede ser especialmente costosa en términos de su impacto, debido a la manera en que se guardan los índices no agrupados. Cada fila del índice no agrupado tiene un apuntador hacia la combinación número de página/número de fila de la fila de datos. Si ocurre una división de página en la pila, se mueve aproximadamente la mitad de las filas. Esto significa que la mitad de las filas tienen que modificar cada entrada para cada índice no agrupado. Todas estas modificaciones ocurren mientras usted espera que ocurra la inserción o actualización de los datos, lo que produce tiempos de respuesta más lentos. Por lo tanto, probablemente sea conveniente tener un índice agrupado en la mayoría de las tablas. Como se vio anteriormente, los índices no agrupados no apuntan a la ubicación física de las filas si existe un índice agrupado, así que las divisiones de página en el índice agrupado no afectan a los índices no agrupados.

Al agregar `PAD_INDEX`, también puede evitar las divisiones de páginas que no sean hojas. De esta manera, se incrementará el uso del espacio –pero no tanto como si se utilizara `FILLFACTOR`.

La opción `DROP_EXISTING`

Esta opción especifica que debe quitarse y volver a crearse un índice (en esencia, ésta es una “reorganización” del índice). Si va a quitar y a volver a crear el índice agrupado, tal vez piense que esto afecta a los índices no agrupados. Sin embargo, SQL Server 2000 es lo suficientemente inteligente como para simplemente quitar y volver a crear el índice agrupado. Como los índices no agrupados tienen como identificador de fila al valor de clave del índice agrupado, y como ese valor no cambia durante la instrucción `CREATE INDEX` con la cláusula `DROP_EXISTING`, no necesitan modificarse los índices no agrupados.

Esta opción es útil principalmente para los índices agrupados, pero también puede utilizarse en índices no agrupados. La opción `DBCC DBREINDEX`, tiene una funcionalidad similar, pero la opción `DROP_EXISTING` probablemente sea una mejor opción a largo plazo en general. No obstante, en SQL Server 2000 Microsoft ha introducido una herramienta de reorganización de índices en línea llamada `DBCC INDEXDEFRAG`.

La opción `STATISTICS_NORECOMPUTE`

SQL Server 2000 vuelve a calcular automáticamente las estadísticas de índices, según sea necesario. Puede deshabilitar estas características si lo desea, utilizando la opción `STATISTICS_NORECOMPUTE`. Para volver a habilitar la característica de recopilación automática de estadísticas de SQL Server, ejecute `UPDATE STATISTICS` sin `NORECOMPUTE`. Por lo general es mala idea desactivar la recopilación automática de estadísticas de sus índices. El optimizador de SQL Server 2000 depende mucho de las estadísticas precisas para optimizar sus consultas.

La opción `SORT_IN_TEMPDB`

SQL Server 2000 introduce la habilidad de utilizar `tempdb` para guardar estructuras de datos en forma temporal, al crear un índice. Esto supone que el índice que se va a crear es más grande que la cantidad de memoria físicamente disponible para SQL Server 2000. Si éste es el caso, SQL Server necesita copiar al disco algunas estructuras de datos temporales utilizadas para crear los índices. Sin la opción `SORT_IN_TEMPDB`, esas estructuras van dentro de la base de datos en la que se está creando el índice. Si se solicita esta opción, se crean estas estructuras temporales en la base de datos `tempdb`. Si `tempdb` se encuentra en un disco separado, este proceso puede producir un mejor rendimiento en la

creación de índices, pero probablemente requiera de más espacio en disco del que se requeriría si se creara el índice sin utilizar tempdb.

La opción IGNORE_DUP_KEY

La opción IGNORE_DUP_KEY especifica que al ejecutar una actualización de varias filas en una tabla con un índice agrupado único, las filas duplicadas de esa inserción se descartan silenciosamente y la instrucción se completa correctamente, pero SQL Server emite una advertencia. El siguiente ejemplo de código muestra cómo funciona esto:

```
CREATE TABLE t1 (col1 int not null, col2 char(5) not null)
Go
CREATE UNIQUE CLUSTERED INDEX mi_ind ON t1 (col1) WITH IGNORE_DUP_KEY
Go
CREATE TABLE t2 (col1 int not null, col2 char(5) not null)
Go
INSERT t2 VALUES (1, 'abcde')
INSERT t2 VALUES (2, 'abcde')
INSERT t2 VALUES (2, 'abcde')
INSERT t2 VALUES (3, 'abcde')

INSERT t1 SELECT * FROM t2
```

Después de ejecutar este código, recibe el siguiente mensaje:

```
    Servidor: mensaje 3604, nivel 16, estado 1, línea 1
    Clave duplicada omitida.
```

Si después realizara una selección en la tabla t1, vería tres filas, como es de esperarse.

4.3.3. Selección de índices

Ahora que ha visto las opciones disponibles y la manera de crear índices, la siguiente pregunta lógica podría ser, “¿qué columnas debo indizar y como debo hacerlo?” Algunos buenos candidatos a ser índices son:

- Columnas de clave primaria.
- Columnas de clave externa.
- Columnas en las que se utiliza la cláusula ORDER BY.
- Columnas en las que se utiliza la cláusula GROUP BY.
- Columnas que se especifican exactamente en la cláusula WHERE.

En general, debe indizar columnas que se incluyan en su cláusula WHERE, usando condiciones que no incluyan funciones ni cálculos. En otras palabras, la siguiente instrucción indica probablemente que la columna emp_id es buen candidato para ser índice:

WHERE emp_id = 5

No obstante, en este ejemplo probablemente no se pueda utilizar un índice, por lo que no tiene caso crear uno (para esta consulta):

WHERE SUBSTRING (emp_id, 1, 5) = 'Annet'

Las instrucciones como la primera cláusula WHERE que aparece en el párrafo anterior se conoce como argumentos de búsqueda. Los argumentos de búsqueda (también conocidos como SARGs) son los tipos de argumentos que pueden ser utilizados por SQL Server. Además de las coincidencias exactas, algunas veces se pueden utilizar coincidencias aproximadas en columnas de caracteres. Por ejemplo,

WHERE emp_apellido LIKE 'w%'

Se puede utilizar para realizar una búsqueda, pero:

WHERE emp_apellido LIKE '%w%'

no. Observe que el símbolo % especifica un carácter comodín. En el primer ejemplo, se realiza una búsqueda en la columna emp_apellido para los valores que empiecen con w. En el segundo ejemplo se buscan en la columna emp_apellido todos los empleados que tengan una w en cualquier lugar del apellido. Imagine si tratara de realizar una búsqueda como ésta usted mismo en el directorio telefónico. ¿Podría encontrar fácilmente la segunda condición? SQL Server tendría el mismo problema que usted; siempre y cuando se especifique la primera letra, usted tiene algo por dónde empezar.

Otro punto a considerar es que no todas las columnas son buenas candidatas a ser índices. No se debe indizar las siguientes columnas:

- Columnas que utilicen los tipos de datos text, image o bit
- Columnas que no sean demasiado únicas (como femenino o masculino)
- Columnas que sean demasiado amplias como para ser índices útiles

La última opción depende de su aplicación y de la tabla, pero es muy probable que un char(200) sea mal candidato para ser índice.

No obstante, debe ser cuidadoso ya que cada índice implica mantenimiento, uso de espacio y cuestiones relacionadas con el rendimiento. Por lo general, yo le recomiendo no tener más de tres o cuatro índices en una tabla. Para cada regla existen algunas excepciones, pero éste es un buen lineamiento a seguir. El siguiente paso después de decidir que columnas va a indizar es averiguar el tipo de índice que se necesita: agrupado o no agrupado.

Cómo elegir índices agrupados

Sólo puede tener un índice agrupado, por lo que es muy conveniente elegirlo primero. Algunos buenos candidatos a ser índices agrupados son:

- Consultas muy específicas (por ejemplo, en donde coll = 5).

- Consultas con un intervalo de datos (por ejemplo, WHERE col1 BETWEEN 5 AND 30 y WHERE col1 > 20).
- Consultas en columnas en las que se utilicen cláusulas ORDER BY o GROUP BY con frecuencia.

Una cuestión muy importante al seleccionar su índice agrupado en SQL Server 2000 es que los valores clave (columnas indizadas) que elija como índice agrupado se “llevan” hacia los índices no agrupados. Por lo tanto, si elige un índice agrupado amplio, como un char(30), no sólo toma más tiempo realizar búsquedas en el mismo índice agrupado, si no que todos sus índices no agrupados también tienen que llevar el valor Char(30) de su clave de índice agrupado en cada fila de los índices no agrupados. Ésta es una cantidad considerable de sobrecarga, por lo que debe mantener el tamaño de sus claves de índice agrupado lo más pequeño que sea posible. Debe asegurarse también de que la clave agrupada que se seleccione no se actualice con frecuencia, debido a que todos sus índices no agrupados necesitan actualizarse cada vez que cambian los valores del índice agrupado.

Otra opción podría ser indizar columnas de clave externas.

Cómo elegir índices no agrupados

Después de seleccionar su índice agrupado, el resto de los índices que vaya a crear deben ser no agrupados. Algunos buenos candidatos son:

- Consultas específicas (así es, son buenos candidatos para ambos tipos de índices).
- Consultas que puedan contestarse completamente con el uso de un índice (conocidas como consultas cubiertas por un índice). Por ejemplo, si se tiene un índice en las columnas au_lname y au_fname de la tabla authors la base de datos pubs, la siguiente consulta puede ser contestada completamente desde el índice, sin tener que acceder los datos:
SELECT au_fname FROM pubs..authors WHERE au_lname = “White”
- Columnas en las que se aplican las cláusulas ORDER BY o GROUP BY.
- Columnas en las que se utilizan funciones (tales como MIN, MAX o COUNT).

Lo importante aquí es que sólo se deben indizar columnas que se utilicen en la cláusula WHERE de sus consultas, y que usted compruebe que los índices elegidos sean utilizados por SQL Server mediante las opciones SHOWPLAN o mediante el plan de ejecución mostrado en forma gráfica en el Analizador de consultas de SQL Server.

4.4 Seguridad en bases de datos.

4.4.1. ¿Por qué crear inicios de sesión de usuario?

Los inicios de sesión de usuarios permiten proteger los datos contra modificaciones no autorizadas, sean intencionadas o no. Gracias a los inicios de sesión de usuario, SQL Server permite identificar usuarios individuales autorizados. Cada inicio de sesión de usuario tiene asignado un nombre único y una contraseña. Cada usuario tiene asignado su propia cuenta de inicio de sesión. Sin los inicios de sesión de usuario, todas las conexiones a SQL Server usarían el mismo identificador, lo que imposibilitaría crear diferentes niveles de seguridad basados en quién accede a la base de datos.

Con los inicios de sesión de usuario, puede crear varios niveles de seguridad permitiendo permisos diferentes a diferentes cuentas de inicio de sesión con el fin de acceder a objetos y realizar funciones.

También se pueden cifrar ciertos objetos de base de datos, tales como procedimientos almacenados y vistas, para ocultar sus definiciones a usuarios no autorizados. Y gracias a los inicios de sesión de usuario, se puede permitir solamente a algunos usuarios insertar y actualizar información en ciertas tablas de la base de datos, y conceder accesos de sólo lectura a las tablas al conjunto de usuarios en general.

Para ver cómo se puede limitar el acceso a datos con inicios de sesión de usuario, se vuelve al ejemplo, sobre el uso de una vista para restringir el acceso a datos sensibles. Se supone que se tiene una tabla Employee que contiene información sobre empleados, con su nombre, número de teléfono, número de oficina, nivel académico, salario, bonos, etc. Para evitar que ciertos usuarios accedan a información confidencial en la tabla, se debería crear primero una vista que contuviera solamente información no sensible, tal como los nombres de los empleados, números de teléfono y números de oficina.

Después, implementando inicios de sesión de usuario, se puede restringir el acceso a la tabla subyacente mientras se permite a todos los usuarios acceder a la vista. Sin embargo, si no se aprovecharan las posibilidades de los inicios de sesión de usuario, cualquier usuario podría acceder a la vista o a la tabla, perdiendo así el uso de la vista su intencionalidad.

4.4.2. Modos de autenticación

Para acceder a SQL Server se dispone de dos modos de autenticación: Autenticación con Windows de Microsoft y Autenticación en modo mixto. En Autenticación con Windows, el sistema operativo es el responsable de identificar al usuario.

SQL Server usa después esta identificación del sistema operativo para determinar los permisos de usuario que hay que aplicar. Con Autenticación en modo mixto, los dos Windows NT/2000 y SQL Server son responsables de identificar al usuario. Para acceder a SQL Server, hay siempre que acceder primero mediante un inicio de sesión a una cuenta Windows NT/2000, de forma que cuando se selecciona un modo de autenticación, hay que decidir simplemente si se quiere utilizar la Autenticación SQL Server además de la Autenticación de Windows. Se va a tratar con más detalle cada uno de estos modos de autenticación. Se pretenderá cómo implementar estos modos más adelante en esta sección.

Autenticación de Windows

Según se ha indicado, con Autenticación de Windows, SQL Server se basa en Windows NT/2000 para proporcionar seguridad al inicio de sesión. Cuando un usuario inicia sesión de Windows NT/2000, se comprueba la identificación de la cuenta de usuario. SQL Server comprueba que el usuario fue validado por Windows NT/2000 y permite el acceso basado en esa identificación.

SQL Server integra su proceso de seguridad en el inicio de sesión con el de Windows para proporcionar estos servicios. Los atributos de seguridad en la red se validan por medio de un sofisticado proceso de cifrado proporcionado por Windows NT/2000.

Debido a que los procesos de seguridad en el inicio de sesión de SQL Server y Windows se integran cuando se utiliza este modo, una vez que se es identificado por el sistema operativo, no es necesario utilizar ningún otro método de identificación al acceder a SQL Server. La única contraseña que hay que suministrar en el inicio de sesión con SQL Server es la contraseña de Windows NT/2000.

La Autenticación de Windows se considera un método mejor que la Autenticación en modo mixto debido a las características de seguridad adicionales que proporciona. Estas características comprenden la validación y cifrado de forma segura de las contraseñas, auditorías, caducidad de contraseñas, longitud mínima de la contraseña, y bloqueo automático de la cuenta tras un determinado número de intentos fallidos en el inicio de sesión.

Autenticación en modo mixto

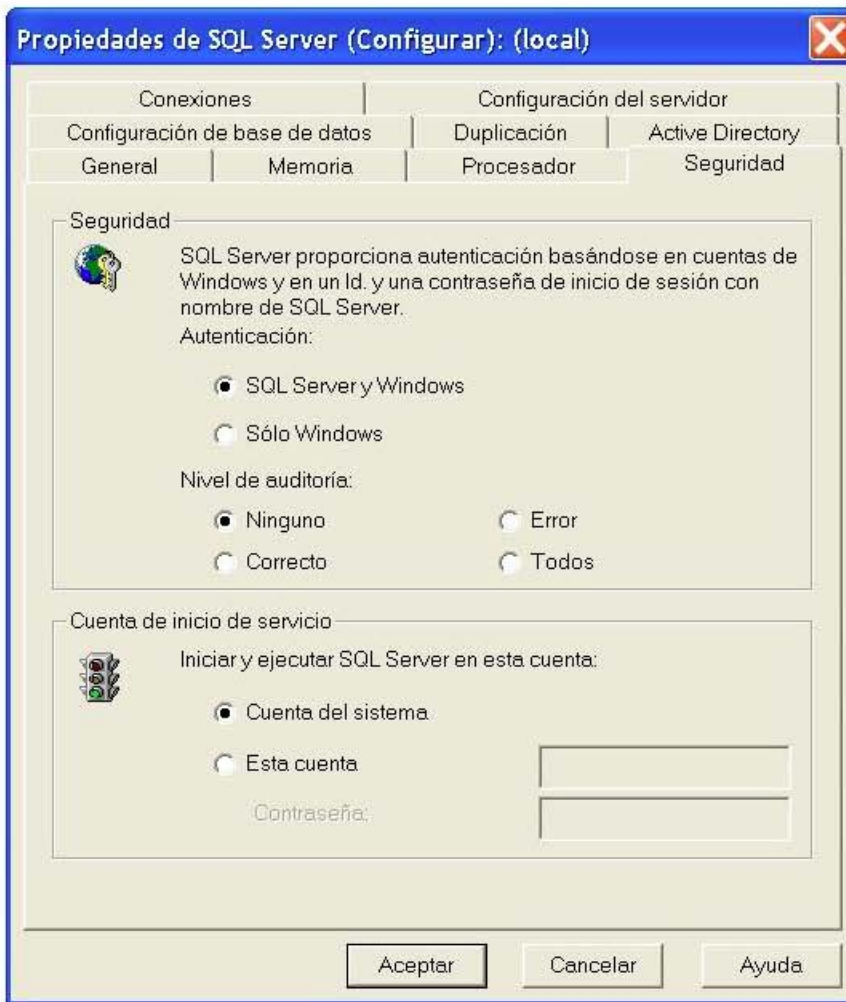
En la Autenticación en modo mixto, los usuarios pueden acceder a SQL Server utilizando la Autenticación de Windows o la Autenticación de SQL Server. Al usar la Autenticación en modo mixto, si se produce una conexión desde un sistema inseguro, SQL Server identifica el inicio de sesión comprobando si la cuenta de inicio de sesión se ha configurado con el acceso solicitado por el usuario. SQL Server realiza la identificación de la cuenta comparando el nombre del código y contraseña que el usuario proporciona al conectarse a SQL Server con la información de la cuenta almacenada en la base de datos. Si la cuenta para el inicio de sesión no ha sido configurada por el usuario o si el usuario no proporciona el nombre correcto y su contraseña, se rechaza el acceso a SQL Server.

El modo de Autenticación de Windows no está disponible cuando se ejecuta SQL Server en Windows 95/98, de manera que hay que usar la Autenticación de SQL Server (usando la Autenticación en modo mixto) en tales plataformas. Además, las aplicaciones Web requieren la Autenticación de SQL Server (a través de Microsoft Internet Information Server) porque los usuarios de estas aplicaciones no estarán probablemente dentro del mismo dominio que el servidor y por tanto no pueden trabajar basados en los mecanismos de seguridad de Windows. Otras aplicaciones que necesitan acceso a base de datos podrían requerir igualmente la autenticación de SQL Server: algunos desarrolladores de aplicaciones prefieren usar la seguridad de SQL Server para sus aplicaciones porque simplifica la seguridad de las mismas. Cuando las aplicaciones utilizan la seguridad de SQL Server (dentro de una red de confianza), los desarrolladores de aplicaciones no tienen que proporcionar autenticación de seguridad dentro de la propia aplicación, lo que simplifica el trabajo.

Configuración del modo de autenticación

En la configuración del modo de autenticación se siguen estos pasos:

1. Abrir la ventana del Administrador corporativo. En el panel izquierdo, pulsar con el botón secundario el nombre del servidor que contiene la base de datos para la que se desea establecer el modo de autenticación y seleccionar Propiedades del menú emergente para mostrar la ventana de SQL Server. Pulsar en la ficha Seguridad, que aparece en la Figura.



- En esta ficha se puede seleccionar tanto el método de seguridad como la cuenta de inicio del servicio. En el área de Seguridad, se especifica si debería usar Autenticación SQL Server y Windows NT/2000 (Modo mixto) o si se debería usar sólo Autenticación Windows NT/2000. También se puede especificar el nivel de auditoría del inicio de sesión. Esta configuración determina qué tipo, en caso de existir, de auditoría de inicio de sesión se realiza. El nivel de auditoría seleccionado dependerá de los requisitos de seguridad. Hay cuatro niveles disponibles que son los siguientes:

- Ninguno No realiza ninguna auditoría de inicio de sesión. Es la configuración predeterminada.
- Correcto Registra todos los intentos válidos de inicios de sesión.
- Error Registra todos los intentos fallidos de inicio de sesión.
- Todos Registra todos los intentos de inicio de sesión.

Nota: El nivel de auditoría es una propiedad de la base de datos. De esta manera, se aplica el mismo nivel de auditoría a todos los inicios de sesión.

- En el área de la cuenta de inicio de servicio, hay que especificar la cuenta de Windows que se debería usar al iniciar y ejecutar SQL Server. Se puede usar la cuenta del sistema o especificar

una cuenta, por ejemplo la de Administrador, con su contraseña. Pulsar en Aceptar para confirmar la configuración establecida.

4.4.3. Inicios de sesión y usuarios

En las dos secciones siguientes se aprenderá cómo crear sesiones de inicio de sesión y usuarios. Pero antes de iniciar este proceso es necesario comprender lo que son los inicios de sesión y los usuarios. Esta sección proporciona una breve definición de los dos términos.

Según se ha visto, una cuenta de usuario de Windows puede ser necesaria para conectarse a la base de datos. También podría ser necesaria la Autenticación de SQL Server. Tanto si se va a usar Autenticación Windows o Autenticación en modo mixto, la cuenta que se usa para la conexión a SQL Server se conoce como inicio de sesión de SQL Server.

Además del inicio de sesión a SQL Server, cada base de datos tiene un conjunto de cuentas ficticias asignadas a ella. Estas pseudocuentas proporcionan alias a cuentas de inicio de sesión de SQL Server.

Por ejemplo, en la base de datos Northwind, podría haber un usuario llamado administrador que se asocia con el inicio de sesión invitado, y la base de datos pubs podría tener un usuario llamado administrador que se asocia con la cuenta de SQL Server de inicio de sesión sa.

En forma predeterminada, una cuenta de inicio de sesión de SQL Server no tiene un ID de usuario de base de datos asociado con ella; por tanto, carece de permisos.

4.4.4. Creación de inicios de sesión de SQL Server

Para crear inicios de sesión mediante el uso de T-SQL, se usa el procedimiento almacenado `sp_addlogin` o el procedimiento almacenado `sp_grantlogin`. El procedimiento almacenado `sp_addlogin` puede agregar solamente un usuario autenticado- SQL Server a una base de datos SQL Server. El procedimiento almacenado `sp_grantlogin` puede agregar un usuario autenticado de Windows.

El procedimiento almacenado `sp_addlogin` tiene la sintaxis siguiente:

```
sp_addlogin [ @loginame = ] 'inicio_de_sesión'  
[ , [ @passwd = ] 'contraseña'  
[ , [ @defdb = ] 'base_de_datos'  
[ , [ @deflanguage = ] 'lenguaje'  
[ , [ @sid = ] 'ds '  
[ , [ @encryptopt = ] 'opción_de_cifrado']
```

Los parámetros opcionales son los siguientes:

- `contraseña` Especifica la contraseña del inicio de sesión de SQL Server. El valor predeterminado es NULL.
- `Base_de_datos` Especifica la base de datos predeterminada del inicio de sesión. El valor por defecto es master.

- lenguaje Especifica el lenguaje predeterminado del inicio de sesión. El valor predeterminado es la configuración del lenguaje actual de SQL Server.
- ds Especifica el dígito de seguridad, que es un número único. Si no se especifica un valor, se generará uno en su lugar. El parámetro ds no se utiliza generalmente por los usuarios, pero los administradores usan ds en cierto número de situaciones. Cuando el administrador de bases de datos realiza tareas de solución de problemas, podría ser necesario el ds para determinar el inicio de sesión que se está comprobando. El parámetro ds es un identificador interno para el inicio de sesión.
- Opción_de_cifrado Especifica si la contraseña será cifrada en las tablas del sistema. El valor predeterminado es NULL, lo que significa que la contraseña se cifrará. La contraseña no se cifrará si se especifica skip_encryption. Si se especifica skip_encryption_old, la contraseña que ya se cifró en una versión anterior de SQL Server no se cifrará de nuevo. Esta configuración solo se cambiaría a partir de su valor predeterminado si necesita evitar cifrar la contraseña en las tablas del sistema.

Un ejemplo sencillo de agregar un inicio de sesión se muestra a continuación:

```
EXEC sp_addlogin 'PatB'
```

Hay que recordar el usar la palabra reservada EXEC antes del nombre del procedimiento almacenado.

Un ejemplo más complicado sobre la agregación de un inicio de sesión aparece a continuación:

```
EXEC sp_addlogin 'SharonR', 'micontraseña', 'Northwind', 'us_english'
```

Este comando crea un usuario llamado SharonR con la contraseña «micontraseña». La base de datos predeterminada es Northwind y el lenguaje predeterminado es el inglés americano. En general, se deberá permitir a SQL Server crear por sí mismo un dígito de seguridad en lugar de dejarlo al propio interesado.

El procedimiento almacenado sp_grantlogin tiene la sintaxis siguiente:

```
sp_grantlogin 'nombre_inicio_de_sesión'
```

Un ejemplo de uso del procedimiento almacenado sp_grantlogin aparece a continuación:

```
EXEC sp_grantlogin 'MOUNTAIN_DEW \DickB'
```

«DickB» es el nombre Windows de la cuenta. «MOUNTAIN_DEW» es el nombre del sistema. Después de agregar estos inicios de sesión el Administrador Corporativo puede verlos. Para ello se pulsa la carpeta Inicios de sesión del panel izquierdo.

4.4.5. Creación de usuarios de SQL Server

Para usar T-SQL para crear usuarios de base de datos se puede ejecutar el procedimiento almacenado sp_adduser.

El procedimiento almacenado puede ejecutarse a partir de ISQL o OSQL y tiene la sintaxis siguiente:

```
sp_adduser[ @loginame = ] 'inicio_de_sesión'  
[ , [ @name_in_db = ] 'usuario' ]  
[ , [ @grpname = ] 'grupo' ]
```

El parámetro inicio_de_sesión es el nombre de la cuenta de inicio de sesión de SQL Server y tiene que suministrarse.

La variable usuario es el nuevo nombre de usuario, y grupo es el grupo o función al que pertenece el nuevo usuario. Si no se especifica ningún valor para usuario, tomará el valor del parámetro inicio_de_sesión.

El comando siguiente crea un nuevo usuario de base de datos con el nombre de JackR y la cuenta de Windows FORT_WORTH \DB_User:

```
sp_adduser 'FORT_WORTH \DB_User', 'JackR'
```

«FORT- WORTH »es el nombre del sistema o del dominio. «DB_User» es nombre de la cuenta de Windows.

4.4.6. Administración de permisos de bases de datos

Los permisos se utilizan para controlar el acceso a los objetos de bases de datos y para especificar qué usuarios pueden realizar ciertas acciones en las bases de datos. Se pueden establecer tanto permisos del servidor como de la base de datos. Los permisos de servidor permiten al administrador de la base de datos realizar las tareas de administración de la base de datos. Los permisos de base de datos se usan para permitir o denegar el acceso a objetos de base de datos y a instrucciones. En esta sección se tratarán todos los tipos de permisos y cómo asignarlos.

4.4.7. Permisos de servidor

Como ya se mencionó anteriormente, los permisos de servidor se asignan a los administradores de la base de datos para permitirles desarrollar tareas administrativas. Los permisos se definen de acuerdo a determinadas funciones del servidor. Los inicios de sesión de usuario pueden tener asignadas determinadas funciones del servidor, pero las funciones no se pueden modificar. Los permisos de servidor comprenden los permisos SHUTDOWN, CREATE DATA BASE, BACKUP DATABASE y CHECKPOINT. Los permisos de servidor se usan sólo para autorizar al administrador de la base de datos a realizar tareas administrativas y no tienen por qué ser modificados ni concedidos a usuarios individuales.

4.4.8. Permisos de objetos de base de datos

Los permisos de objetos de base de datos son una clase de permisos que se conceden para permitir el acceso a objetos de base de datos. Los permisos de objetos son necesarios para acceder a una tabla o vista utilizando instrucciones SQL tales como SELECT, INSER UPDATE y DELETE.

Un permiso de objeto también se necesita para usar la instrucción EXECUTE para la ejecución de un procedimiento almacenado.

Utilización de T-SQL para asignar permisos de objeto

Para usar T-SQL en la asignación de permisos de objeto a un usuario, se ejecuta la instrucción GRANT. La instrucción GRANT tiene la sintaxis siguiente:

```
GRANT{ ALL | permisos }  
[ columna ON { tabla | vista } ] |  
[ ON tabla (columna) ] |  
[ ON vista (columna) ] |  
[ ON { procedimiento_almacenado | procedimiento_extendido } ]  
TO cuenta_de_seguridad  
[ WITH GRANT OPTION ]  
[ AS { grupo / función } ]
```

El parámetro *cuenta_de_seguridad* tiene que tener el tipo de cuenta siguiente:

- Usuario SQL Server.
- Función SQL Server.
- Usuario Windows.
- Grupo Windows.

La utilización de la palabra clave GRANT OPTION permite al usuario o usuarios especificados en la instrucción conceder el permiso especificado a otros usuarios. Esto puede ser útil cuando se conceden permisos a otros administradores de la base de datos. Sin embargo, la opción GRANT se deberá usar con cuidado.

La opción AS especifica cuál es la autoridad con la que se ejecuta la instrucción GRANT. Para ejecutar la instrucción GRANT, el usuario o la función tienen que tener una autoridad especial concedida a tal efecto.

He aquí un ejemplo de uso de la instrucción GRANT:

```
GRANT SELECT, INSERT, UPDATE  
ON Customers  
TO MaríaW  
WITH GRANT OPTION  
AS Accounting
```

La opción AS Accounting se usa porque la función Accounting tiene derecho a conceder permisos en la tabla Customers. La palabra clave GRANT OPTION permite a MaríaW conceder permisos a otros usuarios.

Uso de T-SQL para revocar permisos de objeto

Se puede usar el comando REVOKE de T-SQL para revocar los permisos de objeto a un usuario. La instrucción REVOKE tiene la sintaxis siguiente:

```
REVOKE [GRANT OPTION FOR ]
    { ALL [ PRIVILEGES ] | permisos }
    [ columna ON { tabla | vista } ] |
    [ ON tabla (columna) ] |
    [ ON vista (columna) ] |
    [ ON { procedimiento_almacenado | procedimiento_extendido } ]
    { TO | FROM } cuenta_de_seguridad
    [ CASCADE ]
    [ AS { grupo | función } ]
```

El parámetro cuenta_de_seguridad tiene que pertenecer a uno de los tipos siguientes:

- Usuario SQL Server.
- Función SQL Server.
- Usuario Windows.
- Grupo Windows.

La opción GRANT OPTION FOR permite revocar permisos que se concedieron con anterioridad usando la palabra clave GRANT OPTION, así como revocar permisos. La opción AS especifica con qué autoridad se ejecuta la instrucción REVOKE.

Un ejemplo de uso de la instrucción REVOKE es el siguiente:

```
REVOKE ALL
ON Customers
FROM MaríaW
```

La instrucción REVOKE ALL elimina todos los permisos que el usuario tiene en la tabla Customers.

4.4.9. Permisos de instrucción de base de datos

Además de asignar permisos de objeto de base de datos, se pueden asignar permisos de instrucción. Los permisos de objeto capacitan a los usuarios para acceder a los objetos dentro de la base de datos, mientras que los permisos de instrucción les autorizan a crear objetos de base de datos, inclusive bases de datos y tablas. Los permisos de instrucción se listan a continuación:

- BACKUP DATABASE Permite al usuario ejecutar el comando BACKUP DATABASE.
- BACKUP LOG Permite al usuario ejecutar el comando BACKUP LOG.
- CREATE DATABASE Permite al usuario crear nuevas bases de datos.
- CREATE DEFAULT Permite al usuario crear valores predeterminados que pueden ser vinculados a las columnas.
- CREATE PROCEDURE Permite al usuario crear procedimientos almacenados.

- CREATE RULE Permite al usuario crear funciones.
- CREATE TABLE Permite al usuario crear nuevas tablas.
- CREATE VIEW Permite al usuario crear nuevas vistas. Se pueden asignar permisos de instrucción por medio del Administrador corporativo o por T-SQL.

Utilización de T-SQL para asignar permisos de instrucciones

Para asignar permisos de instrucción a un usuario utilizando T-SQL, se usa la instrucción GRANT. La instrucción GRANT tiene la sintaxis siguiente:

```
GRANT { ALL | instrucción }
To cuenta_de_seguridad
```

Los permisos de instrucción que se pueden asignar a un usuario son CREATE DATA BASE, CREATE DEFAULT, CREATE PROCEDURE, CREATE RULE, CREATE TABLE, CREATE VIEW, DROP TABLE, DROP VIEW, BACKUP DATABASE y BACKUP LOG, según se dijo anteriormente. Por ejemplo, para agregar los permisos de instrucción CREATE DATA BASE Y CREATE TABLE a la cuenta de usuario JackR, se usa el comando siguiente:

```
GRANT { ALL | instrucción }
TO 'JackR'
```

Como se puede ver, agregar permisos de instrucción a una cuenta de usuario no es un proceso nada complicado.

Utilización de T-SQL para revocar permisos de instrucción

Puede usar la instrucción T-SQL, REVOKE para eliminar permisos de instrucción de una cuenta de usuario. La instrucción REVOKE tiene la sintaxis siguiente:

```
REVOKE { ALL | instrucción }
FROM cuenta_de_seguridad.
```

Por ejemplo, para eliminar sólo el permiso de instrucción CREATE DATABASE de la cuenta de usuario JackR, se usa el comando siguiente:

```
REVOKE CREATE DATABASE
FROM 'JackR'
```

Como se puede ver, eliminar sólo el permiso de una cuenta de usuario no es un proceso nada complicado.

5. CASO DE APLICACIÓN: GALERÍA VIEW RIDGE

La galería View Ridge es una pequeña empresa que vende arte contemporáneo, incluyendo litografías, pinturas originales y fotografías. Todas las litografías y fotografías están firmadas y numeradas y la mayoría de las piezas cuestan entre \$1000 y \$25000. View Ridge ha estado en el negocio durante 27 años; tiene un solo propietario que le dedica tiempo completo, así como tres vendedores y dos trabajadores que elaboran marcos, cuelgan los cuadros en la galería y preparan las obras de arte para su envío.

View Ridge realiza exposiciones y otros eventos con el fin de atraer clientes. Las obras de arte también se exhiben en compañías locales, restaurantes y otros lugares públicos. View Ridge es dueña de todo el material que vende; no tiene obras de arte a consignación.

5.1. Modelado conceptual.

Para desarrollar un buen diseño conceptual, se debe ser capaz de reunir información que permita identificar con precisión las entidades y describir sus atributos e interrelaciones. Las interrelaciones entre entidades deben reflejar con precisión las relaciones en el mundo real.

5.1.1. Análisis de requisitos.

El estudio inicial de una base de datos, básicamente es una descripción muy detallada de los ambientes de sistema de base de datos actuales y propuestos de una organización. Por consiguiente, el estudio inicial de una base de datos debe incluir un cuidadoso recuento del análisis de la situación de la empresa, los problemas y restricciones del sistema, la definición de los objetivos del sistema, y el alcance y límites de este.

La información contenida en el estudio inicial de la base de datos es, en gran medida, el producto de entrevistas con usuarios finales clave. Estas personas son los beneficiarios principales del sistema y deben identificarse con cuidado. Los usuarios clave de la aplicación de la galería View Ridge desarrollada en este capítulo son: el personal de ventas, el administrativo y los administradores del sistema.

- Al personal de ventas se le permite ingresar un nuevo cliente y los datos de la transacción, cambiar la información del cliente y solicitar cualquier información. No se les permite meter nuevos artistas o la información del trabajo. Tampoco pueden eliminar cualquier dato.
- El personal administrativo tiene acceso a todos los permisos del personal de ventas, además pueden introducir nuevos artistas y datos de las obras de arte, así como modificar los datos de las transacciones. Aunque el personal administrativo está autorizado para eliminar datos, no tienen ese permiso en esta aplicación. Esta restricción tiene como fin evitar la posibilidad de perder información accidentalmente.
- Al administrador del sistema se le permite el acceso irrestricto a los datos. Puede crear, actualizar, leer y eliminar cualquier información de la base de datos. También puede otorgar derechos de procesamiento a otros usuarios y cambiar la estructura de los elementos de la base de datos tales como tablas, índices, procedimientos almacenados y cosas por el estilo.

Un diseño debe tomar en cuenta los requerimientos pertinentes del usuario. Los requerimientos pertinentes se basan en el nivel propuesto de eficiencia de generación de información. El resumen de todos los requerimientos pertinentes de los usuarios de la galería View Ridge genera una descripción general de los requerimientos del sistema:

En primer lugar, tanto el dueño como los vendedores desean dar seguimiento a sus clientes, así como a los intereses de éstos en cuanto a la compra de arte. Los vendedores necesitan saber a quién contactar cuando llega nuevo material, y tener toda esa información para crear cartas personalizadas y comunicarse verbalmente con sus clientes.

Además, la base de datos debe registrar las compras de arte de los clientes para que los vendedores dediquen más tiempo a los compradores más activos. En ocasiones también necesitan usar los registros de adquisiciones para ubicar las obras de arte, porque a veces la galería compra nuevamente algunas obras que son difíciles de encontrar y las revende. La aplicación de la base de datos debe tener un formato para agregar las nuevas obras que compra la galería.

View Ridge quiere que su aplicación de base de datos proporcione una lista de los artistas y los trabajos que se han exhibido en la galería. Al dueño también le gustaría poder determinar con qué rapidez se vende el trabajo de un artista y cuáles son los márgenes de utilidad, y la aplicación de base de datos debe mostrar un inventario actual en una página Web a la que los clientes puedan tener acceso a través de Internet. Por último, la galería View Ridge quiere que la base de datos produzca reportes que aminoren la carga de trabajo del contador de tiempo parcial que está contratado.

5.1.2. Etapa de conceptualización

Desde el análisis de requisitos de la base de datos, se identifica un conjunto inicial de entidades. Estas entidades representan los objetos del sistema de información más importantes, desde el punto de vista del usuario final y del diseñador. La galería View Ridge utilizará las entidades mostradas en la tabla.

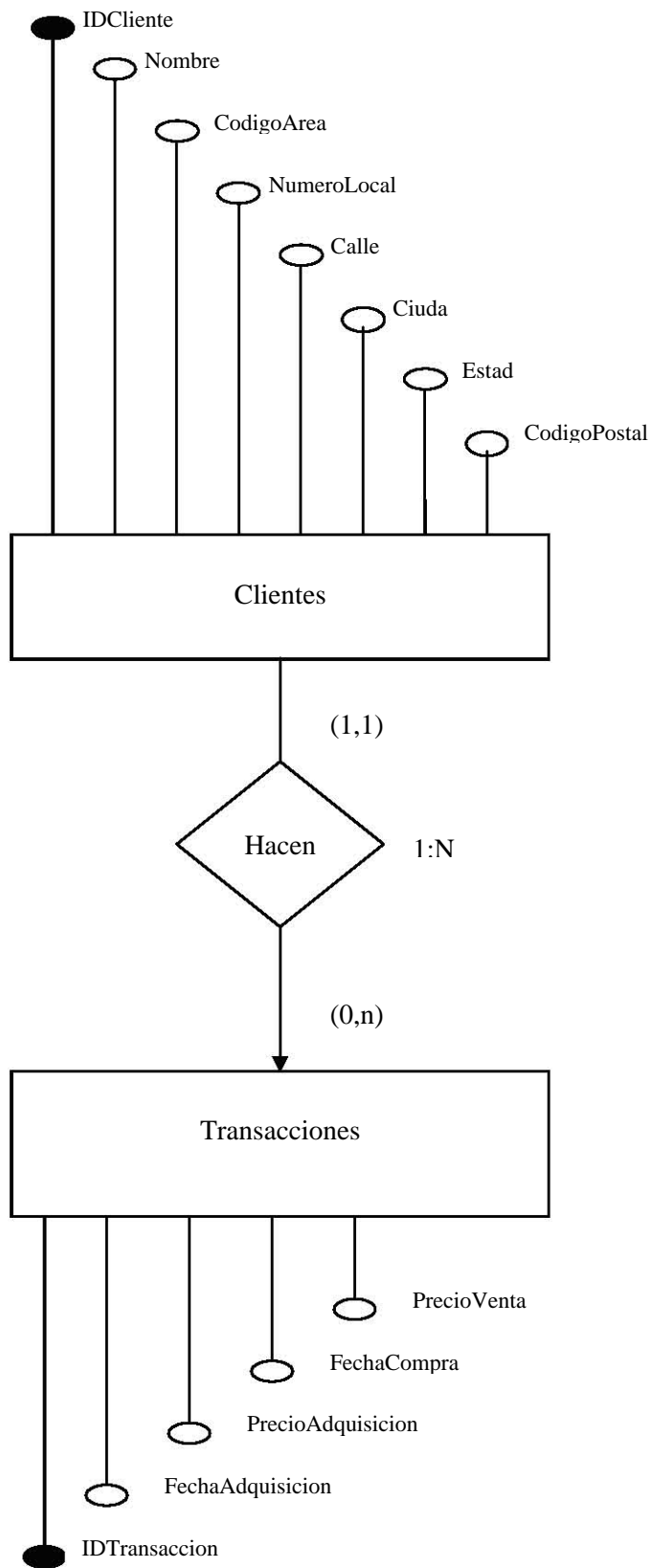
Nombre de entidad	Descripción de entidad	Atributos de la entidad
Clientes	Datos de los clientes de la galería View Ridge.	<ul style="list-style-type: none"> • IDCliente • Nombre • CodigoArea • NumeroLocal • Calle • Ciudad • Estado • CodigoPostal
Artistas	Información de los artistas que han exhibido sus obras en la galería.	<ul style="list-style-type: none"> • IDArtista • NombreArtista • Nacionalidad • FechaNacimiento • FechaFallecimiento
Trabajos	Lista de los trabajos de los artistas.	<ul style="list-style-type: none"> • IDTrabajo • Titulo • Copia • Descripción

Transacciones	Representa las adquisiciones y las ventas.	<ul style="list-style-type: none"> • IDTransaccion • FechaAdquisicion • PrecioAdquisicion • FechaCompra • PrecioVenta
---------------	--	--

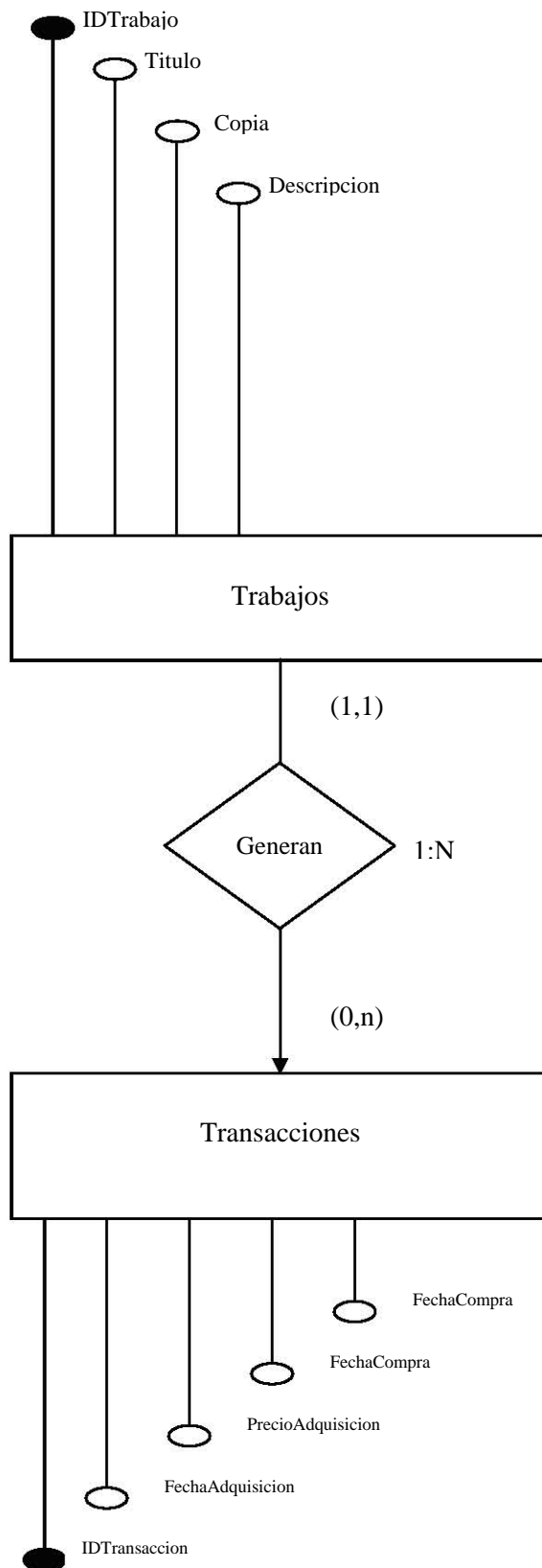
El diseñador y el usuario final deben ponerse de acuerdo en las entidades. Después, el diseñador debe definir las interrelaciones entre las entidades, en general, con base en la descripción de operaciones. Más específicamente, las interrelaciones entre entidades se basan en reglas de negocios derivadas de la descripción pormenorizada de los procedimientos operativos.

El proceso de modelado E/R para la galería View Ridge produce el siguiente resumen de reglas de negocio, entidades e interrelaciones:

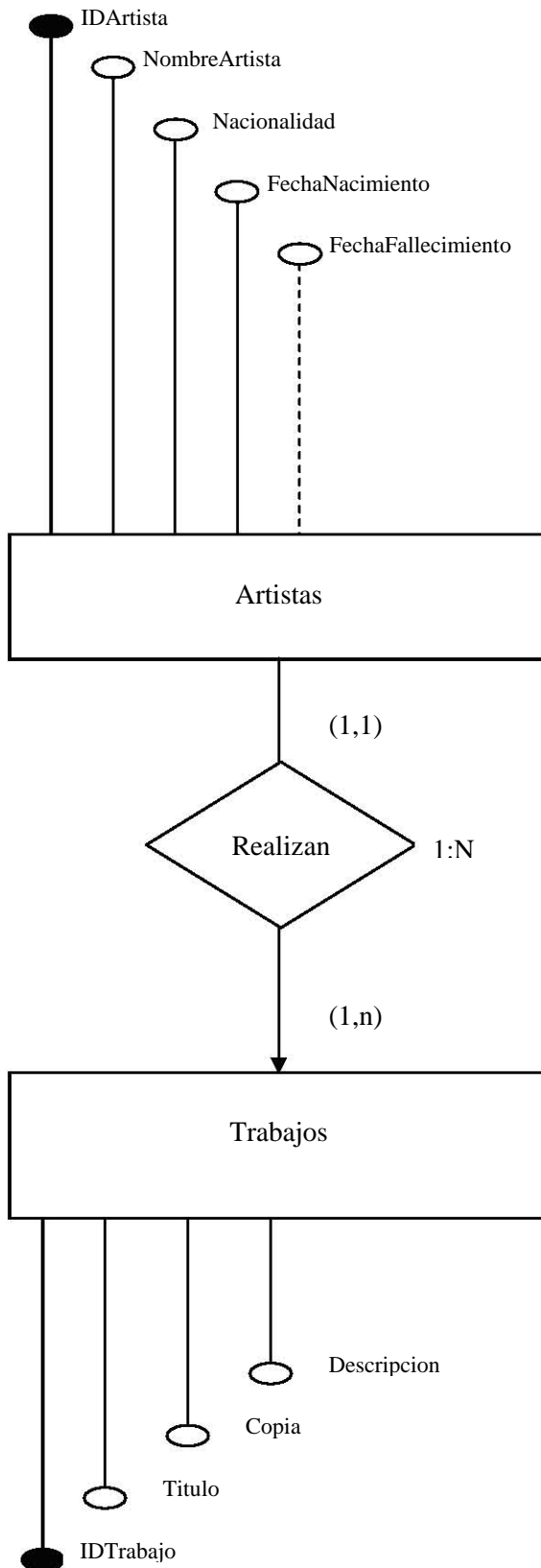
- Registrar las compras de material artístico de los clientes.



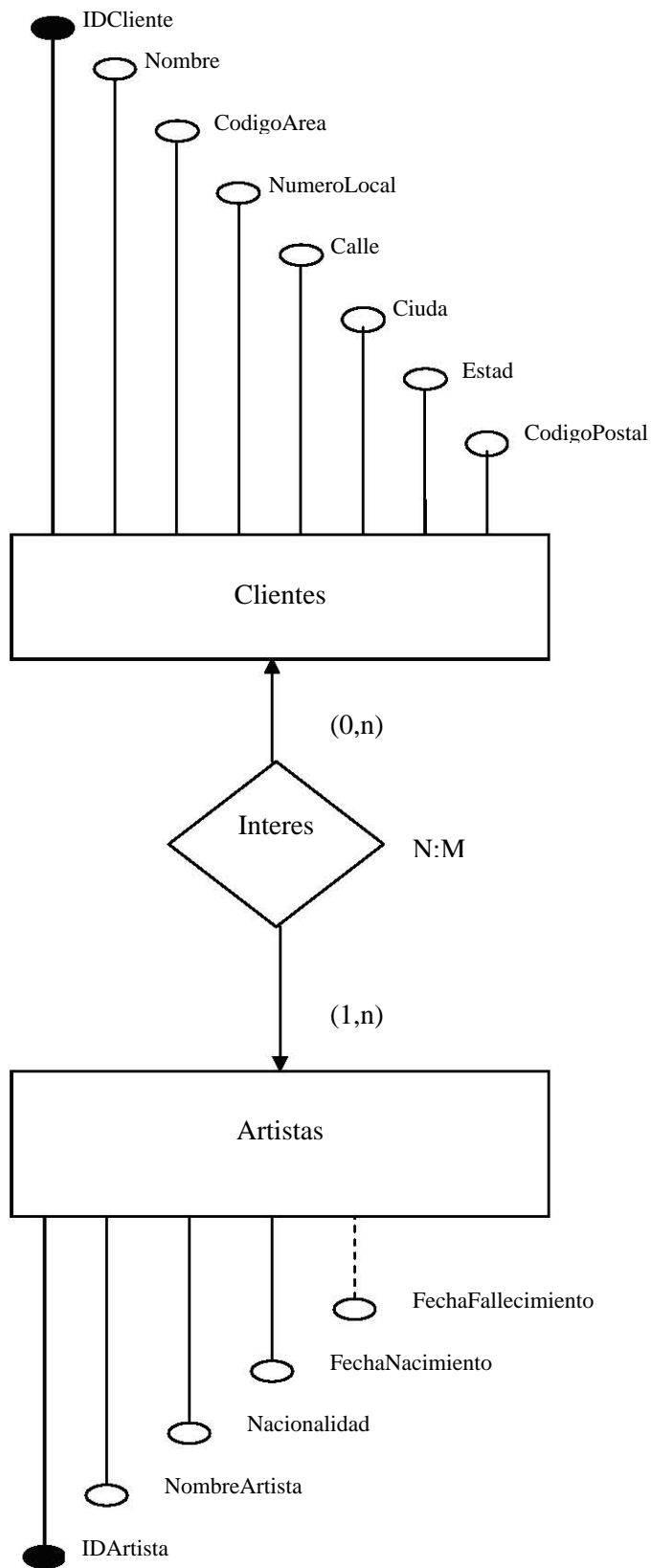
- Reportar con qué rapidez se vende el trabajo de un artista y con qué margen de utilidad.



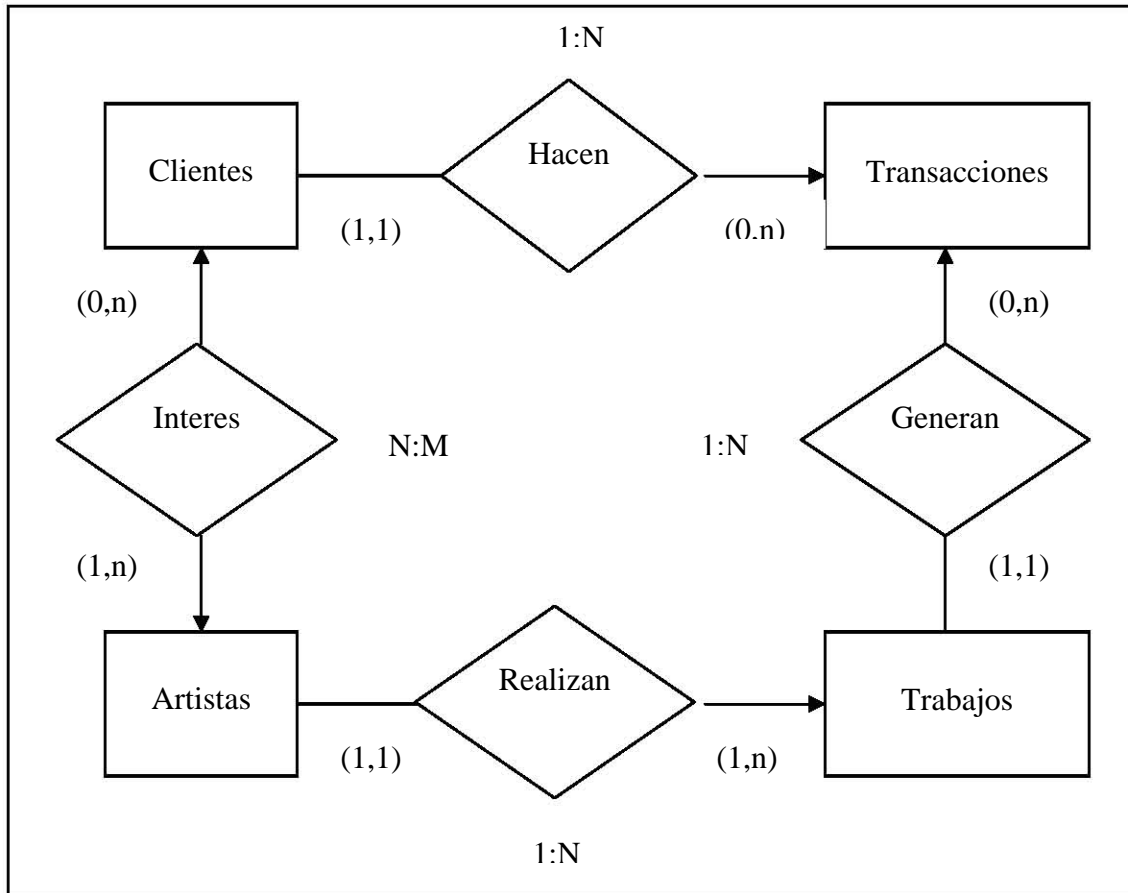
- Listar los artistas y las obras que se han exhibido en la galería.



- View Ridge quiere registrar los intereses de sus clientes. En particular, desea conocer los artistas en los cuales un cliente en particular está interesado, y aquellos a los que les interesa un artista en especial.



La figura siguiente representa el diagrama E/R visualizada por los usuarios finales y el diseñador a estas alturas de la galería View Ridge.



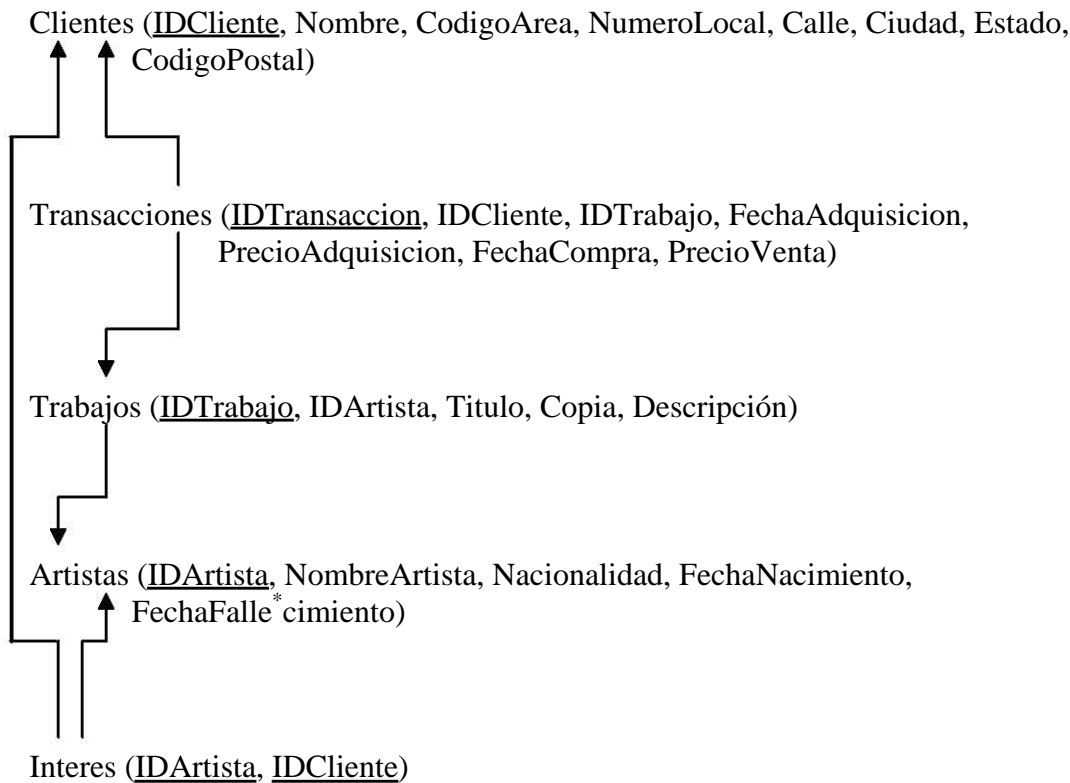
5.2. Diseño lógico

5.2.1. Diseño lógico estándar

Cuando se completa la fase de diseño conceptual, el diagrama E/R refleja –a nivel conceptual- las reglas de negocios que, a su vez, definen las entidades, interrelaciones, restricciones, identificadores externos y jerarquía de generalización (recuerde que algunos elementos del diseño no pueden modelarse y, por consiguiente, se ponen en vigor a nivel de la aplicación.). Además, el modelo conceptual incluye la definición de los atributos que describen cada una de las entidades y qué se requiere para satisfacer requerimientos de información.

Cabe repetir que el diseño lógico estándar transforma el modelo conceptual de conformidad con el formato esperado del SGBD utilizado para ejecutar el sistema. Como estaremos utilizando un modelo de bases de datos relacional, la fase del diseño lógico estándar monta el escenario para aplicar las reglas de transformación de esquema E/R a esquemas relacionales, junto con las restricciones semánticas que hay que contemplar en un esquema relacional.

Sin importar cómo se transforme el diseño mostrado en el modelado conceptual anterior en la estructura de base de datos correspondiente, el esquema relacional de la base de datos debe concordar con el diseño. La figura siguiente muestra el esquema relacional de la galería View Ridge.



5.2.2. Diseño lógico específico

La transformación del esquema lógico estándar al esquema lógico específico es prácticamente directa, sólo se ha de describir (simplemente transcribir) el esquema lógico en la sintaxis propia del SGBD utilizado, que en nuestro caso es Microsoft SQL Server.

Los listados siguientes ilustran el diseño del modelo lógico, mediante SQL (hay que asegurarse de que las tablas se ajusten a la estructura del modelo relacional y que obedezcan las reglas de clave foránea).

Con SQL, pueden crearse estructuras de tabla dentro de la base de datos diseñada. Las tablas de la galería View Ridge se crearían con:

```
CREATE TABLE Clientes
(IDCliente smallint NOT NULL
CONSTRAINT PK_Clientes PRIMARY KEY,
Nombre char(50) NOT NULL,
CodigoArea char(5) NOT NULL
CONSTRAINT CK_CodigoArea CHECK (CodigoArea IN ('206', '425', '212'))
CONSTRAINT DF_CodigoArea DEFAULT '206',
NumeroLocal char(8) NOT NULL,
Calle char(60) NOT NULL,
Ciudad char(15) NOT NULL,
```

```
Estado char(15) NOT NULL,  
CodigoPostal char(5) NOT NULL )
```

```
CREATE TABLE Transacciones  
( IDTransaccion int IDENTITY (1,1) NOT NULL  
  CONSTRAINT PK_Transacciones PRIMARY KEY,  
FechaAdquisicion datetime NOT NULL,  
PrecioAdquisicion money NOT NULL,  
FechaCompra datetime NOT NULL,  
PrecioVenta money NOT NULL,  
IDCliente smallint NOT NULL,  
IDTrabajo smallint NOT NULL )
```

```
CREATE TABLE Trabajos  
( IDTrabajo smallint NOT NULL  
  CONSTRAINT PK_Trabajos PRIMARY KEY,  
Titulo char(40) NOT NULL,  
Copia char(7) NOT NULL,  
Descripción char(60) NOT NULL,  
IDArtista smallint NOT NULL )
```

```
CREATE TABLE Artistas  
( IDArtista smallint NOT NULL  
  CONSTRAINT PK_Artistas PRIMARY KEY,  
NombreArtista char(50) NOT NULL,  
Nacionalidad char(15) NOT NULL,  
FechaNacimiento smallint NOT NULL  
  CONSTRAINT CK_FechaNacimiento CHECK (FechaNacimiento BETWEEN 1400 AND 2100),  
FechaFallecimiento smallint NULL  
  CONSTRAINT CK_FechaNacimiento CHECK (FechaNacimiento BETWEEN 1400 AND 2100) )
```

```
CREATE TABLE Interes  
( IDArtista smallint NOT NULL,  
IDCliente smallint NOT NULL,  
CONSTRAINT PK_Interes PRIMARY KEY (IDArtista, IDCliente) )
```

Quando se crean todas las tables requeridas por el diseño, se establecen las relaciones especificadas en él.

```
ALTER TABLE Transacciones  
ADD CONSTRAINT FK_Transacciones_Clientes FOREIGN KEY (IDCliente)  
REFERENCES Clientes (IDCliente)
```

```
ALTER TABLE Transacciones  
ADD CONSTRAINT FK_Transacciones_Trabajos FOREIGN KEY (IDTrabajo)  
REFERENCES Trabajos (IDTrabajo)
```

```
ALTER TABLE Trabajos
ADD CONSTRAINT FK_Trabajos_Artistas FOREIGN KEY (IDArtista)
REFERENCES Artistas (IDArtista)
```

```
ALTER TABLE Interes
ADD CONSTRAINT FK_Interes_Artistas FOREIGN KEY (IDArtista)
REFERENCES Artistas (IDArtista)
```

```
ALTER TABLE Interes
ADD CONSTRAINT FK_Interes_Clientes FOREIGN KEY (IDCliente)
REFERENCES Clientes (IDCliente)
```

5.3. Diseño físico

5.3.1. Optimización y ajuste ó tuning

El diseño físico requiere que, el diseñador deba afinar la base de datos y el sistema mismo para lograr un desempeño aceptable y asegurarse de distinguir entre desempeño aceptable y óptimo. El desempeño no es el único factor importante en las bases de datos, ni es necesariamente el más importante.

Para la versión Personal de SQL Server que es la que utilizamos, desperdiciaríamos nuestro tiempo tratando de poner mucho esfuerzo en optimizar SQL Server para la mayoría de las implementaciones. Las opciones de optimización automática se encargan de las opciones de configuración más importantes por usted.

5.3.2. Indexación de la base de datos

En la fase de diseño, el diseñador puede especificar índices apropiados para mejorar el funcionamiento de la base de datos. SQL Server automáticamente crea un índice sobre todas las claves primarias y alternativas. El programador debe hacer que SQL Server cree un índice sobre las columnas que se usan como clave externa.

```
CREATE INDEX IX_IDCliente
ON Transacciones (IDCliente)
WITH FILLFACTOR = 80, PAD_INDEX
GO
```

```
CREATE INDEX IX_IDTrabajo
ON Transacciones (IDTrabajo)
WITH FILLFACTOR = 80, PAD_INDEX
GO
```

```
CREATE INDEX IX_IDArtista
ON Trabajos (IDArtista)
WITH FILLFACTOR = 80, PAD_INDEX
GO
```

5.3.3. Seguridad

El objetivo de la seguridad de la base de datos es asegurar que sólo los usuarios autorizados puedan desempeñar actividades permitidas en momentos también establecidos. Este objetivo es difícil de alcanzar y para lograrlo el equipo programador de la base de datos debe, durante la fase de especificación de requerimientos del proyecto, determinar los derechos y responsabilidades de procedimiento de todos los usuarios. Estos requerimientos de seguridad se pueden imponer usando las

características de seguridad del SGBD y las adiciones a las características que se describen en los programas de aplicación.

Considere las necesidades de la galería View Ridge que analizamos en este capítulo. Existen tres tipos de usuarios: el personal de ventas, el administrativo y los administradores del sistema. La figura resume estos requerimientos.

	Cientes	Transacciones	Trabajos	Artistas
Personal de Ventas	Insertar, cambiar, consultar	Insertar, Cambiar	Consultar	Consultar
Personal Administrativo	Insertar, cambiar, consultar	Insertar, cambiar, consultar	Insertar, cambiar, consultar	Insertar, cambiar, consultar
Administrador del sistema	Insertar, cambiar, consultar, Eliminar, Otorgar derechos, Modificar, Estructurar	Insertar, cambiar, consultar, Eliminar, Otorgar derechos, Modificar, Estructurar	Insertar, cambiar, consultar, Eliminar, Otorgar derechos, Modificar, Estructurar	Insertar, cambiar, consultar, Eliminar, Otorgar derechos, Modificar, Estructurar

Los permisos en esta tabla están por tipos de usuarios, o grupos de usuarios y no por personas. Esto es normal pero no es obligatorio. Sería posible decir, por ejemplo, que el usuario identificado como “Benjamín Franklin” tiene ciertos derechos de procedimiento. También observe que cuando se usan los grupos es necesario tener un medio para asignar usuarios a los grupos. Cuando “Mary Smith” se registra en la computadora, se necesitan algunos medios para determinar a qué grupo o grupos pertenece.

De acuerdo con la figura anterior, el administrador de la base de datos tiene la tarea de administrar los derechos y responsabilidades del procedimiento. Como es obvio, esos derechos y responsabilidades cambiarán con el tiempo. Conforme se use la base de datos, y se realicen cambios a las aplicaciones y la estructura del SGBD, surgirá la necesidad de derechos y responsabilidades nuevas o diferentes. El administrador de la base de datos es un punto focal para el análisis de estos cambios y su implementación.

Una vez que los derechos del procedimiento han sido definidos, se pueden implementar en el SGBD.

Creación de inicios de sesión:

```
sp_addlogin 'ventas', 'contraseña_ventas', 'ViewRidge'
```

```
sp_addlogin 'administrativos', 'contraseña_administrativos', 'ViewRidge'
```

```
sp_addlogin 'administradores', 'contraseña_administradores', 'ViewRidge'
```

Creación de usuarios:

```
sp_adduser 'ventas', 'ventas'
```

```
sp_adduser 'administrativos', 'administrativos'
```

```
sp_adduser 'administradores', 'administradores'
```

Asignar permisos de objetos:

```
GRANT SELECT, INSERT, UPDATE  
ON Clientes  
TO ventas
```

```
GRANT INSERT, UPDATE  
ON Transacciones  
TO ventas
```

```
GRANT SELECT  
ON Trabajos  
TO ventas
```

```
GRANT SELECT  
ON Artistas  
TO ventas
```

```
GRANT SELECT, INSERT, UPDATE  
ON Clientes  
TO administrativos
```

```
GRANT SELECT, INSERT, UPDATE  
ON Transacciones  
TO administrativos
```

```
GRANT SELECT, INSERT, UPDATE  
ON Trabajo  
TO administrativos
```

```
GRANT SELECT, INSERT, UPDATE  
ON Artistas  
TO administrativos
```

```
GRANT SELECT, INSERT, UPDATE  
ON Clientes  
TO administradores  
WITH GRANT OPTION
```

```
GRANT SELECT, INSERT, UPDATE, DELETE, EXEC  
ON Transacciones  
TO administradores  
WITH GRANT OPTION
```

```
GRANT SELECT, INSERT, UPDATE, DELETE, EXEC  
ON Trabajos  
TO administradores  
WITH GRANT OPTION
```

```
GRANT SELECT, INSERT, UPDATE, DELETE, EXEC  
ON Artistas  
TO administradores  
WITH GRANT OPTION
```

Conclusiones

En el proceso de diseño de una base de datos, hay que concentrarse en las características de los datos requeridas para construir el modelo de dicha base. En suma, se tiene dos visualizaciones de los datos dentro del sistema: la visualización de los datos de la empresa con fuente de información y la del diseñador de la estructura de los datos, su acceso y las actividades requeridas para transformarlos en información.

Como la base de datos es el componente más básico del sistema de información, el proceso de su diseño es de particular interés. El proceso de diseño pasa por una serie de etapas bien definidas:

I. Modelado conceptual.

El diseño conceptual inicial de la base de datos se basa en los resultados de definir la estructura organizacional y la descripción de las operaciones, la definición de los problemas y restricciones, los objetivos del sistema propuesto, y el alcance y los límites del mismo. La definición del alcance y los límites permite que el diseñador se enfoque en los procesos requeridos para el sistema y sus interacciones.

Después de este estudio inicial, el diseñador de la base de datos se prepara para la fase de diseño conceptual estableciendo con cuidado las fuentes de información y a los usuarios de la organización. Un buen diseño conceptual debe tomar en cuenta los requerimientos del usuario final, tanto para consultar ad hoc como para reportes formales. El contenido y los formatos del reporte se analizan con cuidado para obtener indicios sobre las entidades que han de estar contenidas en el modelado conceptual.

La cuidadosa descripción de las operaciones ayuda a definir las reglas de negocios en las cuales se basan los componentes del diagrama E/R. Las reglas de negocio ayudan a confirmar o establecer tanto las entidades como sus interrelaciones con otras entidades.

II. Diseño lógico.

El diseño lógico se desprende de la decisión de utilizar un modelo de base de datos específico (jerárquico, red o relacional). Una vez que se identifica el modelo de base de datos, puede proyectarse el diseño conceptual sobre el diseño lógico hecho a la medida del modelo de bases de datos seleccionado. En esta etapa, el diseño lógico depende del software.

Actualmente, los sistemas relacionales son un estándar en el mercado, especialmente en operaciones comerciales. Por lo que nos centramos en las reglas de transformación del esquema E/R al relacional en esta etapa del diseño de bases de datos. El proceso de conversión garantizará automáticamente la normalización de los resultados del modelo conceptual.

El diseño lógico se utiliza para transformar el diseño conceptual en el modelo interno de un sistema de administración de bases de datos seleccionado (SGBD), tal como DB2, SQL Server, Oracle, Informix, Access, Ingress, etc. Esto incluye la proyección de todos los objetos del modelo en las construcciones específicas utilizadas por el software de bases de datos seleccionados. En el caso de un SGBD relacional, el diseño lógico incluye el diseño de tablas, la definición de los dominios de atributo y formatos de restricciones de acceso apropiados. El escenario ahora está montado para definir los

requerimientos físicos que permitan que el sistema funcione dentro del ambiente de hardware seleccionado.

III. Diseño físico.

El diseño físico es el proceso de seleccionar las características del desempeño de una base de datos. Sin embargo, no todos los SGBD disponen de herramientas de monitoreo y afinación del desempeño incluidas en su software, con lo que hace difícil evaluar dicho desempeño.

La evaluación del desempeño también se dificulta por el hecho de que no existe una medida estándar para el desempeño de una base de datos. El desempeño varía de acuerdo con el ambiente de hardware o software utilizado.

Algunos factores importantes en el desempeño de una base de datos también incluyen los parámetros de configuración del sistema y la base de datos, tales como las opciones de procesador, de configuración de memoria, de E/S y de consulta/índice.

El derecho de utilizar la base de datos también se especifica durante la fase de diseño físico. ¿Quién permitirá utilizar las tablas y qué parte(s) de la(s) tabla(s) estarán disponibles para qué usuarios? Dentro de un marco de referencia relacional las respuestas a tales preguntas requieren la definición de derechos de acceso.

En la fase de diseño físico, el diseñador puede especificar índices apropiados para mejorar la velocidad de operación. Los índices también permiten producir secuencias de salida lógicamente ordenadas.

La terminación del diseño físico permite que el diseñador inicie el proceso de ejecución de la base de datos. Los datos se ingresan en las estructuras de tablas creadas durante la fase de diseño lógico, por medio de utilerías de carga y conversión, o de ambas. El orden en el cual se cargan los datos depende de los requerimientos de clave foránea.

Cuando todas las fases de la metodología del desarrollo de la base de datos hayan sido abordadas y ejecutadas, la base de datos finalmente está en condiciones de operar. La existencia de la base de datos no solo garantiza la disponibilidad de la información en la que se basa la toma de decisiones, también prepara el escenario para la evaluación de los datos como recurso corporativo y estrecha la estructura operativa en toda la organización.

Bibliografía

Fundamentos de Bases de Datos
Korth / Silbershatz
Mc Graw Hill

Aprendiendo Microsoft SQL Server 2000 en 21 Días
Waymire / Sawtell
Prentice Hall

Fundamentos y Modelos de Bases de Datos
De Miguel / Piattini
Compu-tec ra-ma

Aprendiendo Programación de Bases de Datos con Visual Basic 6 en 21 Días
Smith / Curtis
Prentice Hall

Bases de Datos con Visual Basic 6
Macmanus
Prentice Hall

Microsoft Visual Basic 6.0 Manual del Programador
Microsoft
Mc Graw-Hill

Database Systems
Atzeni / Ceri / Paraboseni / Torlone
Mc Graw Hill

Database Management Systems
Ramakrishnan / Gehrke
Mc Graw Hill

Introducción a los Sistemas de Bases de Datos
C. J Date
Addison Wesley Iberoamericana

Diseño de Bases de Datos Relacionados con Access y SQL Server
Rebeca. M. Riodas
Mc- Graw-Hill

Fundamentos y Modelos de Bases de Datos
Miguel Castaño / Mario G. Piattini Velthuis
Alfaomega