Índice general

Parte I

Introducción

Capítulo 1

Sistemas de procesamiento de archivos

1.1. Descripción

Los sistemas de bases de datos surgieron en respuesta a los primeros métodos de gestión informatizada de los datos comerciales. Una manera de guardar la información en la computadora es almacenarla en archivos del sistema operativo. Para permitir que los usuarios manipulen la información, el sistema tiene varios programas de aplicación que gestionan los archivos.

Los sistemas operativos convencionales soportan este sistema de procesamiento de archivos típico. El sistema almacena los registros permanentes en varios archivos y necesita diferentes programas de aplicación para extraer y añadir a los archivos correspondientes. Antes de la aparición de los sistemas gestores de bases de datos (DBMS), las organizaciones normalmente almacenaban la información en sistemas de este tipo.

1.2. Inconvenientes

Guardar la información de la organización en un sistema de procesamiento de archivos tiene una serie de inconvenientes importantes:

- REDUNDANCIA E INCONSISTENCIA DE LOS DATOS: Debido a que los archivos y programas de aplicación los crean diferentes programadores en el transcurso de un largo período de tiempo, es probable que los diversos archivos tengan estructuras diferentes y que los programas estén escritos en varios lenguajes de programación diferentes. Además, puede que la información esté duplicada en varios lugares (archivos). Esta redundancia conduce a costes de almacenamiento y de acceso más elevados. Además, puede dar lugar a la inconsistencia de los datos; es decir, puede que las diferentes copias de los mismos datos no coincidan.

- DIFICULTAD EN EL ACCESO A LOS DATOS: Si una solicitud no fue prevista al diseñarse el sistema original, no habrá ningún programa de aplicación que la satisfaga. Para solucionar el inconveniente hay dos alternativas: realizar una extracción manual desde informes ya existentes o bien pedir a un programador de sistemas que escriba el programa de aplicación necesario. Ambas alternativas son obviamente insatisfactorias.

- AISLAMIENTO DE DATOS: Como los datos están dispersos en varios archivos, y los archivos pueden estar en diferentes formatos, es difícil escribir nuevos programas de aplicación para recuperar los datos correspondientes.

- PROBLEMAS DE INTEGRIDAD: Los valores de los datos almacenados en la base de datos deben satisfacer ciertos tipos de restricciones de consistencia. Sin embargo, cuando se añaden nuevas restricciones, es difícil cambiar los programas para hacer que se cumplan.

- PROBLEMAS DE ATOMICIDAD: Los sistemas informáticos, como cualquier otro dispositivo mecánico o eléctrico, está sujeto a fallos. En muchas aplicaciones es crucial asegurar que, si se produce algún fallo, los datos se restauren al estado consistente que existía antes del fallo. Resulta difícil asegurar la atomicidad en los sistemas convencionales de procesamiento de archivos.

- ANOMALÍAS EN EL ACCESO CONCURRENTE: Para aumentar el rendimiento global del sistema y obtener una respuesta más rápida, muchos sistemas permiten que varios usuarios actualicen los datos simultáneamente. Como en un sistema de procesamiento de archivos los datos se acceden por diferentes aplicaciones programadas por diferentes personas, es muy difícil supervisar y coordinar el acceso concurrente.

- PROBLEMAS DE SEGURIDAD: No todos los usuarios de un sistema de bases de datos deben poder acceder a todos los datos. Como los programas de aplicación se añaden al sistema de procesamiento de datos de una forma ad hoc, es difícil hacer cumplir tales restricciones de seguridad.

Estas dificultades, entre otras, motivaron el desarrollo de los sistemas de bases de datos.

Capítulo 2

Sistemas de gestión de bases de datos (DBMS)

2.1. Descripción

Un sistema de bases de datos es una colección de datos interrelacionados y un conjunto de programas que permiten a los usuarios tener acceso a esos datos y modificarlos. Una de las principales finalidades de los sistemas de bases de datos es ofrecer a los usuarios una visión abstracta de los datos. Es decir, el sistema oculta ciertos detalles del modo en que se almacenan y mantienen los datos.

2.2. Abstracción de datos

Para que el sistema sea útil debe recuperar los datos eficientemente. La necesidad de eficiencia ha llevado a los diseñadores a usar estructuras de datos complejas para la representación de los datos en la base de datos. Dado que muchos de los usuarios de sistemas de bases de datos no tienen formación en informática, los desarrolladores ocultan esa complejidad a los usuarios mediante varios niveles de abstracción para simplificar la interacción de los usuarios con el sistema:

- Nivel físico: El nivel más bajo de abstracción describe cómo se almacenan realmente los datos. El nivel físico describe en detalle las estructuras de datos complejas de bajo nivel.

- NIVEL LÓGICO: El nivel inmediatamente superior de abstracción describe qué datos se almacenan en la base de datos y qué relaciones existen entre esos datos. El nivel lógico, por tanto, describe toda la base de datos en términos de un número pequeño de estructuras relativamente simples. Aunque la implementación de esas estructuras simples en el nivel lógico puede involucrar estructuras complejas del nivel físico, los usuarios del nivel lógico no necesitan preocuparse de esta complejidad. Los administradores de bases de datos, que deben decidir la información que se guarda en la base de datos, usan el nivel de abstracción lógico.

- NIVEL DE VISTAS: El nivel más elevado de abstracción solo describe parte de la base de datos. Aunque el nivel lógico usa estructuras más simples, queda algo de complejidad debido a la variedad de información almacenada en las grandes bases de datos. Muchos usuarios del sistema de bases de datos no necesitan toda esta información; en su lugar solo necesitan tener acceso a una parte de la base de datos. El nivel de abstracción de vistas existe para simplificar su interacción con el sistema. El sistema puede proporcionar muchas vistas para la misma base de datos.

2.3. Modelos de datos

Bajo la estructura de las bases de datos se encuentra el modelo de datos: una colección de herramientas conceptuales para describir los datos, sus relaciones, su semántica y las restricciones de consistencia. Los modelos de datos ofrecen un modo de describir el diseño de las bases de datos en los niveles físico, lógico y de vistas.

- El modelo entidad-relación: El modelo de datos entidad-relación (E-R) se basa en una percepción del mundo real que consiste en una colección de objetos básicos, denominados *entidades*, y de las *relaciones* entre ellos. Una entidad es una "cosa" u "objeto" del mundo real que es distinguible de otros objetos. El modelo entidad-relación se usa mucho en el diseño de bases de datos.

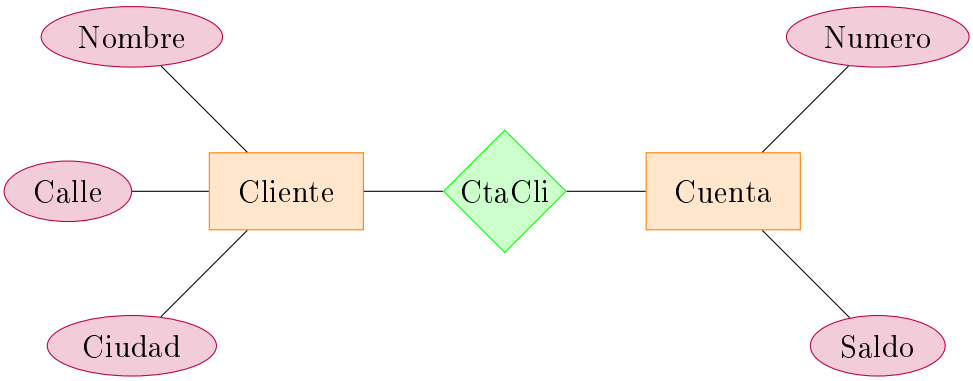


Figura 2.1: Diagrama Entidad Relación

- Modelo relacional: El modelo relacional usa una colección de tablas para representar tanto los datos como sus relaciones. Cada tabla tiene varias columnas, y cada columna tiene un nombre único. El modelo relacional es un ejemplo de un modelo basado en registros. Los modelos basados en registros se denominan así porque la base de datos se estructura en registros de formato fijo de varios tipos. Cada tabla contiene registros de un tipo dado. Cada tipo de registro define un número fijo de campos, o atributos. Las columnas de la tabla se corresponden con los atributos del tipo de registro. El modelo de datos relacional es el modelo de datos más ampliamente usado, y una gran mayoría de sistemas de bases de datos actuales se basan en el modelo relacional.

Nombre	Calle	Ciudad	Numero
Lowery	Maple	Queens	900
Shiver	North	Bronx	556
Shiver	North	Bronx	647
Hodges	Sidehill	Brooklyn	801
Hodges	Sidehill	Brooklyn	647

Cuadro 2.1: Modelo Relacional

2.4. Instancias, esquemas e independencia de datos

Una instancia de una base de datos, es la colección de información almacenada en la base de datos en un determinado momento en el tiempo.

El esquema de la base de datos es el diseño global de la base de datos.

Los DBMS proveen la la independencia de datos, es decir, la capacidad de modificar una definición de un esquema en un nivel sin afectar la definición de un esquema superior siguiente.

- INDEPENDENCIA FÍSICA: Es la capacidad de modificar el esquema físico sin tener que volver a escribir los programas de aplicación.
- INDEPENDENCIA LÓGICA: Es la capacidad de modificar el esquema conceptual (alterar la estructura lógica de la base de datos) sin tener que volver a escribir los programas de aplicación; como por ejemplo añadir un nuevo campo.

2.5. Lenguaje de definición de datos (DDL)

Los esquemas de las bases de datos se especifican mediante un conjunto de definiciones expresadas mediante un lenguaje especial denominado lenguaje de definición de datos (DDL). El DDL también se usa para especificar más propiedades de los datos.

El DDL, al igual que cualquier otro lenguaje de programación, obtiene como entrada algunas instrucciones y genera una salida. La salida del DDL se coloca en el diccionario de datos, que contiene metadatos, es decir, datos sobre datos. El diccionario de datos se considera un tipo especial de tabla, a la que solo puede tener acceso y actualizar el propio sistema de bases de datos (no los usuarios normales). El sistema de bases de datos consulta el diccionario de datos antes de leer o modificar los datos reales.

2.6. Lenguaje de manipulación de datos (DML)

Un lenguaje de manipulación de datos (DML) es un lenguaje que permite a los usuarios tener acceso a los datos organizados mediante el modelo de datos correspondiente o manipularlos. Los tipos de acceso son:

- La recuperación de la información almacenada en la base de datos.
- La inserción de información nueva en la base de datos.
- El borrado de la información de la base de datos.
- La modificación de la información almacenada en la base de datos.

Hay fundamentalmente dos tipos:

- Los DMLs procedimentales necesitan que el usuario especifique qué datos se necesitan y cómo obtener esos datos.
- Los DMLs declarativos (también conocidos como DMLs no procedimentales) necesitan que el usuario especifique qué datos se necesitan sin que haga falta que especifique cómo obtener esos datos.

Una consulta es una instrucción que solicita que se recupere información. La parte de los DMLs implicada en la recuperación de información se denomina lenguaje de consultas. Existen varios lenguajes de consultas de bases de datos en uso, tanto comercial como experimentalmente. El lenguaje de consultas más ampliamente usado se denomina: SQL.

2.7. Gestor de base de datos (DBMS)

Un gestor de bases de datos es un programa que proporciona la interfaz entre los datos de bajo nivel almacenados en la base de datos y los programas de aplicación y consultas hechos al sistema. El gestos de base de datos es responsable de:

- Traducir las distintas sentencias DDL y DML a comandos del sistema de archivos de bajo nivel.
- Almacenar, recuperar y actualizar los datos en la base de datos.
- Mantener la integridad de los datos.
- Asegurar que se cumplan los requisitos de seguridad.
- Detectar fallos y recuperarse ante eventuales fallas.
- Controlar el acceso concurrente a la información.

2.8. Administrador de base de datos (DBA)

Una de las principales razones de usar DBMS es tener un control centralizado tanto de los datos como de los programas que tienen acceso a esos datos. La persona que tiene ese control central sobre el sistema se denomina administrador de bases de datos (DBA). Las funciones del DBA incluyen:

- Definición del esquema conceptual: El DBA crea el esquema original de la base de datos mediante la ejecución de un conjunto de instrucciones de definición de datos en el DDL.
- Definición del esquema interno: El DBA debe decidir cómo se representará la información en la BD almacenada (diseño físico).
- Modificación del esquema y de la organización física: El DBA realiza modificaciones en el esquema y en la organización física para reflejar las necesidades cambiantes de la organización, o para alterar la organización física a fin de mejorar el rendimiento.

- CONCESIÓN DE AUTORIZACIÓN PARA EL ACCESO A LOS DATOS: Mediante la concesión de diferentes tipos de autorización, el administrador de bases de datos puede regular las partes de la base de datos a las que puede tener acceso cada usuario. La información de autorización se guarda en una estructura especial del sistema que el DBMS consulta siempre que alguien intenta tener acceso a los datos del sistema.

- MANTENIMIENTO RUTINARIO: Algunos ejemplos de las actividades de mantenimiento rutinario del administrador de la base de datos son:

  - Copia de seguridad periódica de la base de datos, para impedir la pérdida de datos en caso de desastres.
  - Asegurarse de que se dispone de suficiente espacio libre en disco para las operaciones normales y aumentar el espacio en disco según sea necesario.
  - Supervisar los trabajos que se ejecuten en la base de datos y asegurarse de que el rendimiento no se degrade debido a que algún usuario haya remitido tareas muy costosas.

2.9. Usuarios de base de datos

Hay tres tipos diferentes de usuarios de los sistemas de bases de datos, diferenciados por la forma en que esperan interactuar con el sistema:

- USUARIOS NORMALES: Son usuarios no sofisticados que interactúan con el sistema invocando alguno de los programas de aplicación que se han escrito previamente.

- PROGRAMADORES DE APLICACIONES: Son profesionales informáticos que escriben programas de aplicación que utilizan el DML para interaccionar con el sistema.

- USUARIOS SOFISTICADOS: Interactúan con el sistema sin escribir programas. En su lugar, formulan sus consultas en un lenguaje de consultas de bases de datos.

Parte II

Modelado de datos

Capítulo 3

Modelo relacional

3.1. Descripción

El modelo relacional utiliza un conjunto de tablas para representar tanto los datos como las relaciones entre ellos. Su simplicidad conceptual ha conducido a su adopción generalizada; actualmente, una amplia mayoría de los productos de bases de datos se basan en el modelo relacional.

Una base de datos relacional consiste en un conjunto de tablas, a cada una de las cuales se le asigna un nombre exclusivo. Cada fila de la tabla representa una relación entre un conjunto de valores. De manera informal, cada tabla es un conjunto de entidades, y cada fila es una entidad. Dado que cada tabla es un conjunto de tales relaciones, hay una fuerte correspondencia entre el concepto de tabla y el concepto matemático de relación, del que toma su nombre el modelo de datos relacional. Para cada atributo hay un conjunto de valores permitidos, denominado dominio de ese atributo.

Sucursal	Cuenta	Cliente	Saldo
Downtown	101	Johnson	500
Mianus	215	Smith	700
Perryridge	102	Hayes	400
Round Hill	305	Turner	350

Cuadro 3.1: Tabla *deposito* del modelo relacional

Formalmente: Sean D_1 el conjunto de todos los nombres de sucursales, D_2 el de los números de cuenta, D_3 el de los nombres de clientes y D_4 el de los saldos; entonces $deposito \subseteq D_1 \times D_2 \times D_3 \times D_4$. Cada fila de la tabla sera entonces una cuatro-upla (v_1, v_2, v_3, v_4) donde $v_1 \in D_1$, $v_2 \in D_2$, $v_3 \in D_3$ y $v_3 \in D_4$.

Como las tablas son, esencialmente, relaciones, se usarán los términos matemáticos relación y tupla en correspondencia a los términos tabla y fila. El orden en que aparecen las tuplas en cada relación es irrelevante, dado que una relación es simplemente un conjunto.

3.2. Notación

Sea t una tupla de una relación r, notaremos con $t\,[atributo]$ al valor de t en dicho atributo. Por ejemplo si $t = (Mianus, 215, Smith, 700)$ entonces $t\,[Saldo] = 700$.

Para denotar al esquema de una r escribimos $r = (atributo1, atributo2, \ldots)$.
Por ejemplo para la tabla deposito: $deposito = (Sucursal, Cuenta, Cliente, Saldo)$.

3.3. Claves

Es necesario disponer de un modo de especificar la manera en que las tuplas de una relación dada se distingan entre sí. Esto se expresa en términos de sus atributos. Es decir, los valores de los valores de los atributos de una tupla deben ser tales que puedan identificarla unívocamente.

Una *superclave* es un conjunto de uno o varios atributos que, considerados conjuntamente, permiten identificar de manera unívoca una tupla de la relación. El concepto de superclave no es suficiente para nuestros propósitos, ya que, como se puede ver, las superclaves pueden contener atributos innecesarios. Si C es una superclave, entonces también lo es cualquier superconjunto de C.

A menudo resultan interesantes superclaves para las que ninguno de sus subconjuntos constituya una superclave. Esas superclaves mínimas se denominan *claves candidatas*. Es posible que varios conjuntos diferentes de atributos puedan ejercer como claves candidatas.

Se usará el término *clave primaria* para denotar una clave candidata que ha elegido el diseñador de la base de datos como medio principal para la identificación de las tuplas de una relación.

Formalmente, sea R el esquema de una relación r. Si se dice que un subconjunto C de R es una superclave de r, se restringe la consideración a las relaciones $r(R)$ en las que no hay dos tuplas diferentes que tengan los mismos valores en todos los atributos de C. Es decir:

$$t_1, t_2 \in r \wedge t_1 \neq t_2 \Rightarrow t_1 [C] \neq t_2 [C]$$

Capítulo 4

Modelo entidad relación

4.1. Descripción

El modelo entidad-relación (E-R) es un modelo de datos de alto nivel. En lugar de representar todos los datos en tablas, distingue entre los objetos básicos, denominados *entidades*, y las *relaciones* entre esos objetos. Suele utilizarse como un primer paso en el diseño de los esquemas de las bases de datos.

4.2. Entidades

Una entidad es una "cosa" u "objeto" del mundo real que es distinguible de todos los demás objetos. Las entidades pueden ser concretas, como las personas o los libros, o abstractas, como los préstamos, las vacaciones o los conceptos. Un conjunto de entidades es un conjunto de entidades del mismo tipo que comparten las mismas propiedades, o atributos.

Puede que un conjunto de entidades no tenga suficientes atributos para formar una clave primaria. Ese conjunto de entidades se denomina conjunto de entidades débiles. Los conjuntos de entidades que tienen una clave primaria se denominan conjuntos de entidades fuertes.

Para que un conjunto de entidades débiles tenga sentido, debe estar asociado con otro conjunto de entidades, denominado conjunto de entidades identificadoras o propietarias. Cada entidad débil debe estar asociada con una entidad identificadora; es decir, se dice que el conjunto de entidades débiles depende existencialmente del conjunto de entidades identificadoras.

Aunque los conjuntos de entidades débiles no tienen clave primaria, hace falta un medio para distinguir entre todas las entidades del conjunto de entidades débiles que dependen de una entidad fuerte concreta. El *discriminante* de un conjunto de entidades débiles es un conjunto de atributos que permite que se haga esta distinción.

La clave primaria de un conjunto de entidades débiles se forma con la clave primaria del conjunto de entidades identificadoras y el discriminante del conjunto de entidades débiles.

4.3. Atributos

Cada entidad se representa mediante un conjunto de atributos. Los atributos son propiedades descriptivas que posee cada miembro de un conjunto de entidades. La designación de un atributo para un conjunto de entidades expresa que la base de datos almacena información parecida relativa a cada entidad del conjunto de entidades; sin embargo, cada entidad puede tener su propio valor para cada atributo.

Formalmente, un atributo es una función que asigna al conjunto de entidades, valores en un dominio. Los dominios pueden incluir un valor especial llamado valor nulo, que es el valor que toma una entidad que no puede expresar dicha propiedad. El valor nulo también puede indicar «no aplicable», es decir, que el valor no existe para esa entidad.

Los atributos pueden clasificarse según su tipo:

- SIMPLES O COMPUESTOS: Si un atributo se puede dividir en varias partes (como por ejemplo «direccion») se lo llama compuesto, de lo contrario se trata de un atributo simple.

- MONOVALORADOS O MULTIVALORADOS: Los atributos multivalorados son los que almacenan un conjunto de valores para una entidad, como podría ser el atributo «teléfonos». En general, este tipo de atributos no deberían ser utilizados.

- DERIVADOS: Son atributos que pueden obtenerse a partir de valores de otros atributos. El atributo «edad» es un ejemplo de un atributo derivado de la «fecha de nacimiento». Su valor no se almacena en la base de datos, sino que se calcula cuando es necesario.

4.4. Relaciones

Una relación es una asociación entre varias entidades. Un conjunto de relaciones es un conjunto de relaciones del mismo tipo. Formalmente es una relación matemática con $n \geq 2$ de conjuntos de entidades (posiblemente no distintos). Si $E_1, E_2, \ldots, E_n$ son conjuntos de entidades cuyas claves primarias son $(E_1), \ldots, (E_n)$, entonces un conjunto de relaciones R es un subconjunto de:

$$\{(e_1, \ldots, e_n) / e_1 \in E_1 \wedge \ldots \wedge e_n \in E_n\}$$

donde $(e_1, \ldots, e_n)$ es una instancia de la relación.

Una relación puede también tener atributos denominados *atributos descriptivos*. Sea $A = \{a_1, \ldots, a_m\}$ un conjunto de atributos descriptivos, definimos:

$$atributo\,(R) = (E_1) \cup \ldots \cup (E_n) \cup A$$

Si R no tiene atributos descriptivos, $atributo\,(R)$ es una superclave.

- Si la cardinalidad es N:N, $atributo\,(R)$ es una clave primaria.
- Si la cardinalidad es N:1 o 1:N, la clave primaria sera un subconjunto de $atributo\,(R)$.

Si R tiene atributos descriptivos, es probable que haya que agregar alguno (o todos) a $atributo\,(R)$ para formar la clave.

4.5. Cardinalidad

La correspondencia de cardinalidades, o razón de cardinalidad, expresa el número de entidades a las que otra entidad se puede asociar mediante un conjunto de relaciones. La correspondencia de cardinalidades resulta muy útil para describir conjuntos de relaciones binarias, aunque pueda contribuir a la descripción de conjuntos de relaciones que impliquen más de dos conjuntos de entidades.

Para un conjunto de relaciones binarias R entre los conjuntos de entidades A y B, la correspondencia de cardinalidades debe ser una de las siguientes:

- UNO A UNO: Cada entidad de A se asocia, a lo sumo, con una entidad de B, y cada entidad en B se asocia, a lo sumo, con una entidad de A (véase la Figura 6.5a).
- UNO A VARIOS: Cada entidad de A se asocia con cualquier número (cero o más) de entidades de B. Cada entidad de B, sin embargo, se puede asociar, a lo sumo, con una entidad de A (véase la Figura 6.5b).
- VARIOS A UNO: Cada entidad de A se asocia, a lo sumo, con una entidad de B. Cada entidad de B, sin embargo, se puede asociar con cualquier número (cero o más) de entidades de A (véase la Figura 6.6a).
- VARIOS A VARIOS: Cada entidad de A se asocia con cualquier número (cero o más) de entidades de B, y cada entidad de B se asocia con cualquier número (cero o más) de entidades de A (véase la Figura 6.6b).

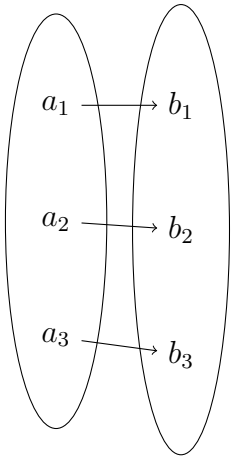


Figura 4.1: Cardinalidad 1:1

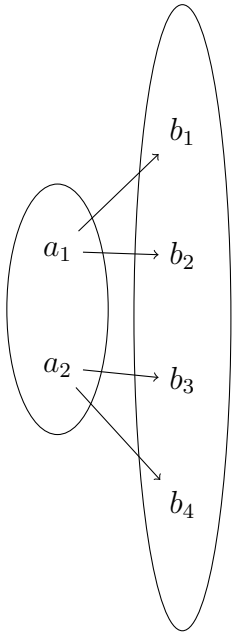


Figura 4.2: Cardinalidad 1:N

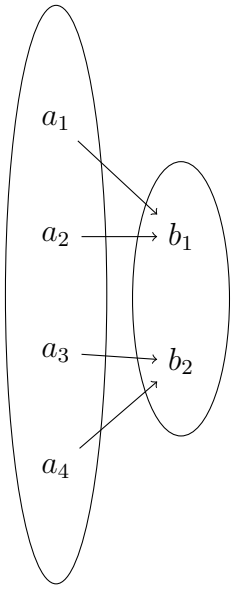


Figura 4.3: Cardinalidad N:1

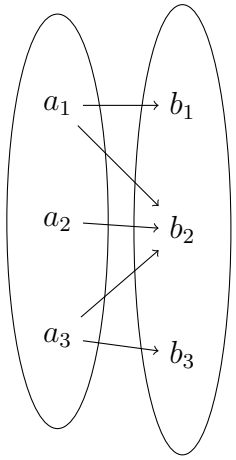


Figura 4.4: Cardinalidad N:N

4.6. Especialización

Los conjuntos de entidades pueden incluir subgrupos de entidades que se diferencian de alguna forma de las demás entidades del conjunto. Por ejemplo, un subconjunto de entidades de un conjunto de entidades puede tener atributos que no sean compartidos por todas las entidades del conjunto de entidades.

Considérese el conjunto de entidades persona con los atributos *dni*, *nombre*, *calle* y *ciudad*. Cada persona puede clasificarse además como cliente o empleado. Cada uno de estos tipos de persona se describen mediante un conjunto de atributos que incluye todos los atributos del conjunto de entidades *persona* más otros posibles atributos adicionales.

Por ejemplo, las entidades *cliente* se pueden describir además mediante el atributo *calificación_ crediticia*, mientras que las entidades empleado se pueden describir además mediante el atributo *sueldo*.

El proceso de establecimiento de subgrupos dentro del conjunto de entidades se denomina *especialización*. La especialización de *persona* permite distinguir entre las personas basándonos en si son empleados o clientes: en general, cada persona puede ser empleado, cliente, las dos cosas o ninguna de ellas.

La especialización parte de un único conjunto de entidades; destaca las diferencias entre las entidades del conjunto mediante la creación de diferentes conjuntos de entidades de nivel inferior. Esos conjuntos de entidades de nivel inferior pueden tener atributos o participar en relaciones que no se aplican a todas las entidades del conjunto de entidades de nivel inferior.

4.7. Generalización

La *generalización*, es una relación de contención que existe entre el conjunto de entidades de nivel superior y uno o varios conjuntos de entidades de nivel inferior. Los conjuntos de entidades de nivel superior e inferior también se pueden denominar con los términos superclase y subclase, respectivamente. El conjunto de entidades persona es la superclase de las subclases *cliente* y *empleado*.

Los conjuntos de entidades de nivel superior e inferior también se pueden denominar con los términos superclase y subclase, respectivamente. El conjunto de entidades persona es la superclase de las subclases cliente y empleado.

La generalización parte del reconocimiento de que varios conjuntos de entidades comparten algunas características comunes (es decir, se describen mediante los mismos atributos y participan en los mismos conjuntos de relaciones). Con base en esas similitudes, la generalización sintetiza esos conjuntos de entidades en un solo conjunto de nivel superior. La generalización se usa para destacar las similitudes entre los conjuntos de entidades de nivel inferior y para ocultar las diferencias; también permite una economía de representación, ya que no se repiten los atributos compartidos.

4.8. Diagrama entidad relación

Los diagramas E-R pueden expresar gráficamente la estructura lógica general de las bases de datos. Estos diagramas constan de los siguientes componentes principales:

- RECTÁNGULOS: que representan conjuntos de entidades.
- ELIPSES: que representan atributos.
- ROMBOS: que representan conjuntos de relaciones.
- LÍNEAS: que unen los atributos con los conjuntos de entidades y los conjuntos de entidades con los conjuntos de relaciones.
- ELIPSES DOBLES: que representan atributos multivalorados.
- ELIPSES DISCONTINUAS: que denotan atributos derivados.
- RECTÁNGULOS DOBLES: que representan conjuntos de entidades débiles.
- TRIÁNGULOS: que representan generalizaciones.

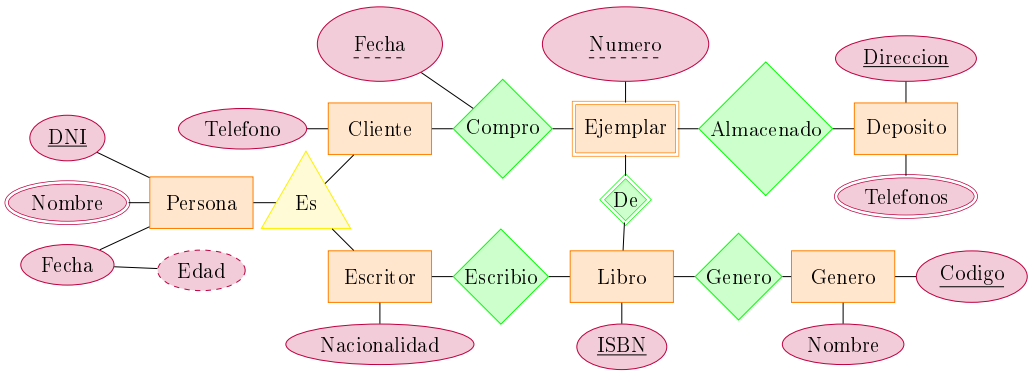


Figura 4.5: Diagrama Entidad Relacion

4.9. Mapa canónico

Las bases de datos que se ajustan a un esquema de bases de datos E-R se pueden representar mediante conjuntos de esquemas de relación. Para cada conjunto de entidades y para cada conjunto de relaciones de la base de datos hay un solo esquema de relación a la que se asigna el nombre del conjunto de entidades o del conjunto de relaciones correspondiente.

Tanto el modelo E-R de bases de datos como el relacional son representaciones abstractas y lógicas de empresas del mundo real. Como los dos modelos usan principios de diseño parecidos, los diseños E-R se pueden convertir en diseños relacionales.

4.9.1. Representación de conjuntos de entidades fuertes

Sea E un conjunto de entidades fuertes con los atributos descriptivos $\{a_1, \ldots, a_n\}$. Esta entidad se representa mediante una tabla denominado E con n columnas distintas. Cada fila de la tabla corresponde a una entidad del conjunto de entidades E.

DNI	Nombre	Fecha
54.321.098	Oliver Harris	24/09/1984
90.123.456	Bill Marsh	11/10/1986
56.789.012	John Pepper	30/02/3075

Cuadro 4.1: Tabla correspondiente a la entidad *persona*

4.9.2. Representación de conjuntos de entidades debiles

Sea A un conjunto de entidades débiles con los atributos $\{a_1, \ldots, a_m\}$. Sea B el conjunto de entidades fuertes del que A depende. La clave primaria de B consiste en los atributos $\{b_1, \ldots, b_n\}$. El conjunto de entidades A se representa mediante una tabla denominada A con una columna por cada miembro del conjunto:

$$\{a_1, \ldots, a_m\} \cup \{b_1, \ldots, b_n\}$$

Para los esquemas derivados de conjuntos de entidades débiles la combinación de la clave primaria del conjunto de entidades fuertes y del discriminador del conjunto de entidades débiles sirve de clave primaria del esquema.

ISBN	Numero
9780385514231	27
9780385541176	12
9780316225908	315

Cuadro 4.2: Tabla correspondiente a la entidad *ejemplar*

4.9.3. Representación de conjuntos de relaciones

Sea R un conjunto de relaciones, sea $\{a_1, \ldots, a_m\}$ el conjunto de atributos formado por la unión de las claves primarias de cada uno de los conjuntos de entidades que participan en R, y sean $\{b_1, \ldots, b_n\}$ los atributos descriptivos de R (si los hay). El conjunto de relaciones se representa mediante la tabla R, con un atributo por cada uno de los miembros del conjunto:

$$\{a_1, \ldots, a_m\} \cup \{b_1, \ldots, b_n\}$$

ISBN	Código
9780385514231	4
9780316225908	2

Cuadro 4.3: Tabla correspondiente a la relación *genero*

Los conjuntos de relaciones que enlazan los conjuntos de entidades débiles con el conjunto correspondiente de entidades fuertes se tratan de manera especial. Estas relaciones son varios a uno y no tienen atributos descriptivos. Además, la clave primaria de los conjuntos de entidades débiles incluye la clave primaria de los conjuntos de entidades fuertes.

En general, el esquema de los conjuntos de relaciones que enlazan los conjuntos de entidades débiles con su conjunto correspondiente de entidades fuertes es redundante y no hace falta que esté presente en el diseño de la base de datos relacional basado en el diagrama E-R .

4.9.4. Representación de generalizaciones

Existen dos métodos diferentes para designar los esquemas de relación de los diagramas E-R que incluyen generalización:

- Se crea una tabla para el conjunto de entidades de nivel superior. Para cada conjunto de entidades de nivel inferior se crea una tabla que incluye un atributo para cada uno de los atributos de ese conjunto de entidades más un atributo por cada atributo de la clave primaria del conjunto de entidades de nivel superior. Los atributos de clave primaria del conjunto de entidades de nivel superior pasan a ser atributos de clave primaria del conjunto de entidades de nivel superior y de todos los conjuntos de entidades de nivel inferior.

- Es posible una representación alternativa si la generalización es disjunta y completa, es decir, si no hay ninguna entidad miembro de dos conjuntos de entidades de nivel inferior directamente por debajo de un conjunto de entidades de nivel superior, y si todas las entidades del conjunto de entidades de nivel superior también pertenece a uno de los conjuntos de entidades de nivel inferior.

  En este caso no se crea una tabla para el conjunto de entidades de nivel superior. En vez de eso, para cada conjunto de entidades de nivel inferior se crea una tabla que incluye un atributo por cada atributo de ese conjunto de entidades más un atributo por cada atributo del conjunto de entidades de nivel superior.

Parte III

Lenguajes de manejo de datos (DML)

Capítulo 5

Álgebra relacional

5.1. Operaciones fundamentales

5.1.1. Selección

La operación selección selecciona tuplas que satisfacen un predicado dado. Se usa la letra griega sigma minúscula (σ) para denotar la selección. El predicado aparece como subíndice de σ. La relación de argumentos se da entre paréntesis a continuación de σ.

En general, se permiten las comparaciones que usan $=$, $\neq$, $<$, $\leq$, $>$, $\geq$ en el predicado de selección. Además, se pueden combinar varios predicados en uno mayor usando las conectivas *y* ($\wedge$), *o* ($\vee$) y *no* ($\neg$). También es posible comprar atributos en el predicado.

5.1.2. Proyección

La operación proyección es una operación unaria que devuelve su relación de argumentos, excluyendo algunos argumentos. Dado que las relaciones son conjuntos, se eliminan todas las filas duplicadas. La proyección se denota por la letra griega mayúscula *pi* (Π). Se crea una lista de los atributos que se desea que aparezcan en el resultado como subíndice de Π. Su único argumento, una relación, se escribe a continuación entre paréntesis.

5.1.3. Union

La unión de dos relaciones, es decir, las filas que aparecen en alguna de las dos relaciones o en ambas se pueden averiguar mediante la operación binaria unión, denotada, como en la teoría de conjuntos, por $\cup$. Dado que las relaciones son conjuntos, se eliminan los valores duplicados.

Para que la operación unión sea válida hay que exigir que se cumplan dos condiciones:

- Las relaciones que intervienen deben tener la misma aridad, es decir, deben tener el mismo numero de atributos.
- Los dominios de los atributos i-esimos de ambas relaciones deben ser iguales.

5.1.4. Resta

La operación diferencia de conjuntos, denotada por $-$, permite hallar las tuplas que están en una relación pero no en la otra. La expresión $r - s$ da como resultado una relación que contiene las tuplas que están en r pero no en s.

Como en el caso de la operación unión, hay que asegurarse de que las diferencias de conjuntos se realicen entre relaciones *compatibles*.

5.1.5. Producto cartesiano

La operación producto cartesiano, denotada por $\times$, permite combinar información de cualesquiera dos relaciones. El producto cartesiano de las relaciones r_1 y r_2 se escribe $r_1 \times r_2$.

Dado que el mismo nombre de atributo puede aparecer tanto en r_1 como en r_2, es necesario crear un convenio de denominación para distinguir unos atributos de otros. En este caso se realiza adjuntando al atributo el nombre de la relación de la que proviene originalmente. Para los atributos que sólo aparecen en uno de los dos esquemas se suele omitir el prefijo con el nombre de la relación. Esta simplificación no genera ambigüedad alguna.

En general, si se tienen las relaciones $r_1\,(R_1)$ y $r_2\,(R_2)$, la relación $r_1 \times r_2$ es una relación cuyo esquema es la concatenación de R_1 y R_2. La relación resultante contiene todas las tuplas t para las que hay unas tuplas $t_1 \in r_1$ y $t_2 \in r_2$ tales que $t\,[R_1] = t_1\,[R_1]$ y $t\,[R_2] = t_2\,[R_2]$.

5.1.6. Renombre

A diferencia de las relaciones de la base de datos, los resultados de las expresiones de álgebra relacional no tienen un nombre que se pueda usar para referirse a ellas. Resulta útil poder ponerles nombre; la operación renombramiento, denotada por la letra griega *ro* minúscula (ρ), permite hacerlo.

Dada una expresión E del álgebra relacional, la expresión $\rho_x(E)$ devuelve el resultado de la expresión E con el nombre x.

Las relaciones por sí mismas se consideran expresiones (triviales) del álgebra relacional. Por tanto, también se puede aplicar la operación renombramiento a una relación para obtener la misma relación con un nombre nuevo.

5.2. Operaciones derivadas

5.2.1. Intersección

La operación de intersección es una operación binaria que consta de las tuplas t que pertenecen a ambas relaciones. Nosete que:

$$r_1 \cap r_2 = r_1 - (r_1 - r_2)$$

por tanto, la intersección de conjuntos no es una operación fundamental y no añade potencia al álgebra relacional.

5.2.2. Producto natural

Suele resultar deseable simplificar ciertas consultas que exijan un producto cartesiano. Generalmente, las consultas que implican un producto cartesiano incluyen un operador selección sobre el resultado del producto cartesiano.

La reunión natural es una operación binaria que permite combinar ciertas selecciones y un producto cartesiano en una sola operación. Se denota por el símbolo de reunión $\bowtie$. La operación reunión natural forma un producto cartesiano de sus dos argumentos, realiza una selección forzando la igualdad de los atributos que aparecen en ambos esquemas de relación y, finalmente, elimina los atributos duplicados.

Formalmente, sean las relaciones $r(R)$ y $s(S)$, el producto natural de ambas relaciones es una relación con esquema $R \cup S$ definida por:

$$r \bowtie s = \Pi_{R \cup S} \left[\sigma_{r.a_1 = s.a_1 \wedge \ldots \wedge r.a_n = s.a_n} (r \times s) \right]$$

donde $R \cap S = \{a_1, \ldots, a_n\}$.

5.2.3. División

La operación división, denotada por /, resulta adecuada para las consultas que incluyen la expresión «para todos». Formalmente, para las relaciones $r(R)$ y $s(S)$ donde $S \subseteq R$, la relación r/s es una relación con esquema $R - S$ que contara unicamente con todas las tuplas t que satisfacen:

- $t \in \Pi_{R-S}(r)$.
- $\forall t_s \in s \exists t_r \in r / t_r[S] = t_s[S] \wedge t_r[R - S] = t$.

Es decir, estará formada por todas las tuplas t tales que aparece una tupla (x, y) en r para todas las tuplas $y \in s$.

Puede resultar sorprendente descubrir que, dados una operación división y los esquemas de las relaciones, se pueda definir la operación división en términos de las operaciones fundamentales:

$$r/s = \Pi_{R-S}(r) - \Pi_{R-S}\{[\Pi_{R-S}(r) \times s] - r\}$$

5.2.4. Asignación

En ocasiones resulta conveniente escribir una expresión del álgebra relacional mediante la asignación de partes de esa expresión a variables de relación temporal. La operación asignación, denotada por $\leftarrow$, actúa de manera parecida a la asignación de los lenguajes de programación.

La evaluación de una asignación no hace que se muestre ninguna relación al usuario. Por el contrario, el resultado de la expresión situada a la derecha de $\leftarrow$ se asigna a la variable relación situada a la izquierda de $\leftarrow$. Esta variable relación puede usarse en expresiones posteriores.

Capítulo 6

Calculo relacional de tuplas

6.1. Definición formal

Cuando se escribe una expresión del álgebra relacional se proporciona una serie de procedimientos que generan la respuesta a la consulta. El cálculo relacional de tuplas, en cambio, es un lenguaje de consultas no procedimental. Describe la información deseada sin dar un procedimiento concreto para obtenerla.

Las consultas se expresan en el cálculo relacional de tuplas como $\{t/P\,(t)\}$, donde P es una fórmula. En una fórmula pueden aparecer varias variables tupla. Las fórmulas del cálculo relacional de tuplas se construyen con átomos. Los átomos tienen una de las formas siguientes:

- $s \in r$, donde s es una variable tupla y r es una relación.
- $s\,[x]\,\Theta u\,[y]$, donde s y u son variables tuplas, x es un atributo en el que está definida s, y es un atributo en el que está definida u, y Θ es un operador de comparación.
- $s\,[x]\,\Theta c$, donde s es una variable tupla, x es un atributo en el que está definida s, Θ es un operador de comparación y c es una constante en el dominio del atributo x.

Las fórmulas se construyen a partir de los átomos usando las reglas siguientes:

- Cada átomo es una formula.
- Si P_1 es una fórmula, también lo son $\neg P_1$ y (P_1).
- Si P_1 y P_2 son fórmulas, también lo son $P_1 \vee P_2$, $P_1 \wedge P_2$ y $P_1 \Rightarrow P_2$.
- Si $P_1(s)$ es una fórmula que contiene la variable tupla libre s, y r es una relación, también son formulas $\exists s \in r\,(P_1(s))$ y $\forall s \in r\,(P_1(s))$.

6.2. Equivalencia con álgebra relacional

- $r \cup s = \{t/t \in r \vee t \in s\}$.
- $r - s = \{t/t \in r \wedge \neg\,(t \in s)\}$.
- $\Pi_{a_1,\ldots,a_n}(r) = \left\{t^{(n)}/t\,[1] = u\,[a_1] \wedge \ldots \wedge t\,[n] = u\,[a_n]\right\}$.
- COMPLETAR.
- COMPLETAR.
- COMPLETAR.

Capítulo 7

Calculo relacional de dominios

7.1. Definición formal

Una segunda forma de cálculo relacional, denominada cálculo relacional de dominios, usa variables de dominio, que toman sus valores del dominio de un atributo, en vez de hacerlo para una tupla completa. El cálculo relacional de dominios, no obstante, se halla estrechamente relacionado con el cálculo relacional de tuplas.

Las expresiones del cálculo relacional de dominios son de la forma:

$$\{\langle x_1, \ldots, x_n \rangle / P(x_1, \ldots, x_n)\}$$

donde $x_1, \ldots, x_n$ representan variables de dominio y P representa una fórmula compuesta por átomos, como era el caso en el cálculo relacional de tuplas. Los átomos del cálculo relacional de dominios tienen una de las formas siguientes:

- $\langle x_1, \ldots, x_n \rangle \in r$, donde r es una relación con n atributos y $x_1, \ldots, x_n$ son variables de dominio o constantes de dominio.
- $x \Theta y$, donde x e y son variables de dominio y Θ es un operador de comparación.
- $x \Theta c$, donde x es una variable de dominio, Θ es un operador de comparación y c es una constante del dominio del atributo para el que x es una variable de dominio.

Las fórmulas se construyen a partir de los átomos usando las reglas siguientes:

- Los átomos son fórmulas.
- Si P_1 es una fórmula, también lo son $\neg P_1$ y (P_1).
- Si P_1 y P_2 son fórmulas, también lo son $P_1 \vee P_2$, $P_1 \wedge P_2$ y $P_1 \Rightarrow P_2$.
- Si $P_1(x)$ es una fórmula en x, donde x es una variable de dominio, también son formulas $\exists x (P_1(x))$ y $\forall x (P_1(s))$.

7.2. Equivalencia con calculo de tuplas

- COMPLETAR.
- COMPLETAR.
- COMPLETAR.
- COMPLETAR.
- COMPLETAR.
- COMPLETAR.

Parte IV

Normalización

Capítulo 8

Dependencias funcionales

COMPLETAR.

Capítulo 9

Formas normales

COMPLETAR.

Parte V

SQL

Capítulo 10

DML

10.1. Consultas

10.1.1. Consultas básicas

La estructura básica de una expresión SQL consta de tres cláusulas: *select*, *from* y *where*.

- La cláusula *select* se corresponde con la operación proyección del álgebra relacional. Se usa para obtener una relación de los atributos deseados en el resultado de una consulta.
- La cláusula *from* se corresponde con la operación producto cartesiano del álgebra relacional. Genera una lista de las relaciones que deben ser analizadas en la evaluación de la expresión.
- La cláusula *where* se corresponde con el predicado selección del álgebra relacional. Es un predicado que engloba a los atributos de las relaciones que aparecen en la cláusula *from*.

Las consultas habituales de SQL tienen la forma:

```
SELECT [DISTINCT] a1,...,an FROM r1,...,rm WHERE P [ORDER BY aj1,...,ajk]
```

donde cada a_i representa un atributo, cada r_i es una tabla y P es un predicado. Si se pretenden todos los atributos puede utilizarse el símbolo «*». Al omitirse la cláusula *where*, el predicado P es cierto.

A diferencia de la expresión del álgebra relacional, el resultado de la consulta SQL puede contener varias copias de algunas tuplas. Para evitar este comportamiento se debe agregar la clausula *distinct* luego de *select*.

La cláusula *from* define por sí misma un producto cartesiano de las relaciones que aparecen en la cláusula.

SQL proporciona un mecanismo para renombrar tanto relaciones como atributos. Utiliza la cláusula *as* de la forma siguiente:

```
nombre_viejo AS nombre_nuevo
```

La cláusula as puede aparecer tanto en la cláusula *select* como en la cláusula *from*.

La cláusula *order by* hace que las tuplas del resultado de una consulta se presenten en un cierto orden. De manera predeterminada, la cláusula order by coloca los elementos en orden ascendente. Para especificar el tipo de ordenación se puede especificar *desc* para orden descendente o *asc* para orden ascendente. Además, se puede ordenar con respecto a más de un atributo.

En la práctica, SQL puede convertir la expresión en una forma equivalente que puede ser procesada de manera más eficiente.

10.1.2. Funciones de agregado

Las funciones de agregación son funciones que toman una colección (un conjunto o multiconjunto) de valores como entrada y devuelven un solo valor. SQL ofrece cinco funciones de agregación incorporadas:

- Promedio: *avg*.
- Mínimo: *min*.
- Máximo: *max*.
- Total: *sum*.
- Recuento: *count*.

Los datos de entrada para *sum* y *avg* deben ser una colección de números, pero los otros operadores también pueden operar sobre colecciones de datos de tipo no numérico como las cadenas de caracteres.

A modo de ejemplo, considérese la consulta «Determinar el saldo total de las cuentas de las sucursales de Rosario». Esta consulta se puede formular del modo siguiente:

```
SELECT SUM(saldo) FROM cuenta WHERE ciudad = "Rosario"
```

10.1.3. Agrupamiento

Existen circunstancias en las cuales sería deseable aplicar las funciones de agregación no sólo a un único conjunto de tuplas sino también a un grupo de conjuntos de tuplas; esto se especifica en SQL mediante la cláusula *group by*. El atributo o atributos especificados en la cláusula *group by* se usan para formar grupos. Las tuplas con el mismo valor en todos los atributos de la cláusula group by constituyen un mismo grupo.

Como ejemplo, considérese la consulta «Determinar los saldo medios de las sucursales de cada ciudad». Esta consulta se formula del modo siguiente:

```
SELECT nombre, AVG(saldo) FROM cuenta GROUP BY ciudad
```

A veces resulta útil establecer una condición que se aplique a los grupos en vez de a las tuplas. Por ejemplo, puede que solo estemos interesados en las sucursales de Rosario en las que el saldo medio de las cuentas sea superior a 1000. Esta condición no se aplica a una sola tupla, sino a cada grupo creado por la cláusula *group by*. Para expresar este tipo de consultas se utiliza la cláusula *having* de SQL. SQL aplica los predicados de la cláusula *having* una vez formados los grupos, de modo que se puedan usar las funciones de agregación. Esta consulta se expresa en SQL del modo siguiente:

```
SELECT nombre, AVG(saldo) FROM cuenta GROUP BY ciudad HAVING AVG(saldo) > 1000
```

10.1.4. Subconsultas

SQL proporciona un mecanismo para anidar subconsultas. Las subconsultas son expresiones *select from where* que están anidadas dentro de otra consulta. Una finalidad habitual de las subconsultas es llevar a cabo comprobaciones de pertenencia a conjuntos.

La conectiva *in* comprueba la pertenencia a un conjunto, donde el conjunto es la colección de valores resultado de una cláusula *select*. La conectiva *not in* comprueba la no pertenencia a un conjunto.

La forma de este tipo de consultas es:

```
SELECT a1,...,an FROM r1,...,rm WHERE ai [NOT] IN (subconsulta)
```

10.1.5. Clausula exists

SQL incluye la posibilidad de comprobar si las subconsultas tienen alguna tupla en su resultado. El constructor *exists* devuelve el valor true si su argumento subconsulta no resulta vacía. Mediante el constructor *not exists* se puede comprobar la inexistencia de tuplas en el resultado de las subconsultas.

La forma de este tipo de consultas es:

```
SELECT a1,...,an FROM r1,...,rm WHERE [NOT] EXISTS (subconsulta)
```

10.2. Altas, bajas y modificaciones

Las tablas recién creadas están inicialmente vacías. Se puede utilizar el comando *insert* para añadir datos a la tabla.

```
INSERT INTO nombre VALUES (v1,...,vn)
```

Los valores se especifican en el orden en que se relacionan los atributos correspondientes en el esquema de la relación.

Las solicitudes de borrado se expresan casi igual que las consultas. Solo se pueden borrar tuplas completas; no se pueden borrar solo valores de atributos concretos. SQL expresa los borrados mediante:

```
DELETE FROM nombre WHERE P
```

donde P es un predicado.

En determinadas situaciones puede ser deseable modificar un valor dentro de una tupla sin cambiar todos los valores de la misma. Para este tipo de situaciones se puede utilizar la instrucción *update*. Al igual que ocurre con *insert* y *delete*, se pueden elegir las tuplas que se van a actualizar mediante una consulta. La sintaxis general es la siguiente:

```
UPDATE nombre SET campo = valor WHERE P
```

Capítulo 11

DDL

11.1. Tipos básicos de dominios

La norma SQL soporta gran variedad de tipos de dominio predefinidos, entre ellos:

- *char(n)*: Una cadena de caracteres de longitud fija, con una longitud n especificada por el usuario.
- *varchar(n)*: Una cadena de caracteres de longitud variable con una longitud máxima n especificada por el usuario.
- *int*: Un entero (un subconjunto finito de los enteros dependiente de la máquina).
- *smallint*: Un entero pequeño (un subconjunto dependiente de la máquina del tipo de dominio entero).
- *numeric(p, d)*: Un número de coma fija, cuya precisión la especifica el usuario. El número está formado por p dígitos (más el signo), y de esos p dígitos, d pertenecen a la parte decimal. Así, *numeric(3,1)* permite que el número *44.5* se almacene exactamente, pero ni *444.5* ni *0.32* se pueden almacenar exactamente en un campo de este tipo.
- *real*, *double precision*: Números de coma flotante y números de coma flotante de doble precisión, con precisión dependiente de la máquina.
- *float(n)*: Un número de coma flotante cuya precisión es, al menos, de n dígitos.

- *date*: Una fecha de calendario que contiene el año (de cuatro cifras), el mes y el día del mes.
- *time*: La hora del día, en horas, minutos y segundos. Se puede usar una variante, *time(p)*, para especificar el número de cifras decimales para los segundos (el valor predeterminado es 0).
- *timestamp*: Una combinación de *date* y *time*. Se puede usar una variante, *timestamp(p)*, para especificar el número de cifras decimales para los segundos (el valor predeterminado es 6).

11.2. Altas, bajas y modificaciones

Las tablas se definen mediante el comando *create table*:

```
CREATE TABLE nombre(a1 d1,...,an dn)
```

donde cada a_i es el nombre de un atributo de la relación y d_i es el dominio de los valores de dicho atributo.

Para eliminar una relación de una base de datos SQL se utiliza el comando drop table. Este comando elimina de la base de datos toda la información de la relación:

```
DROP TABLE nombre
```

Una vez eliminada, no se puede insertar ninguna tupla en dicha relación, a menos que se vuelva a crear con la instrucción *create table*.

El comando *alter table* se utiliza para añadir atributos a una relación existente. Se asigna a todas las tuplas de la relación un valor nulo como valor del atributo nuevo. La forma del comando alter table es:

```
ALTER TABLE nombre ADD a d [BEFORE b]
```

donde a es el nombre del atributo que se desea añadir y d es el dominio del atributo. Se pueden eliminar atributos de una relación utilizando el comando:

```
ALTER TABLE nombre DROP a
```

Para crear un indice y acelerar la búsqueda sobre un campo en una tabla se utiliza el comando *create index*:

```
CREATE [UNIQUE] INDEX nombre ON (a1 [ASC|DESC],...)
```

Si lo que se desea es borrar el indice, se debe utilizar la siguiente instrucción:

```
DROP INDEX nombre
```

11.3. Integridad

Las restricciones de integridad garantizan que las modificaciones realizadas en la base de datos por los usuarios autorizados no den lugar a una pérdida de la consistencia de los datos.

Existen tres tipos de restricciones de integridad:

- De dominio.
- De las entidades
- Referencial.

Como se estudio previamente, el comando *create table* permite especificar reglas de integridad al indicar los dominios de los atributos. También es posible exigir que un campo no acepte valores nulos, que sea único o que satisfaga algún predicado. La sintaxis en estos casos es:

```
CREATE TABLE nombre [UNIQUE](a1 d1 [NOT NULL],...,an dn [NOT NULL]) CHECK P
```

donde P es un predicado que deben cumplir los atributos.

Las restricciones de dominio son específicas porque se refieren a una tabla en particular, sin embargo el modelo relacional incluye dos reglas generales de integridad que se aplican a toda la base de datos: las claves primarias y las claves ajenas. Para definir una clave primaria se utiliza la sentencia:

```
CREATE TABLE nombre (a1 d1,...,an dn, PRIMARY KEY (aj1,...,ajm))
```

La especificación de clave primaria determina que los atributos $a_{j1}, \ldots, a_{jm}$ forman la clave primaria de la relación. Los atributos de la clave primaria tienen que ser no nulos y únicos; es decir, ninguna tupla puede tener un valor nulo para un atributo de la clave primaria y ningún par de tuplas de la relación puede ser igual en todos los atributos de la clave primaria

A menudo se desea garantizar que el valor que aparece en una relación para un conjunto dado de atributos aparezca también para un conjunto determinado de atributos en otra relación. Esta condición se denomina integridad referencial. Las claves externas pueden especificarse como parte de la instrucción de SQL *create table* mediante la cláusula *foreign key*.

Para cada clave ajena el diseñador de la base de datos debe responder tres preguntas:

- ¿Puede aceptar nulos esa clave ajena?
- ¿Qué deberá suceder si hay un intento de eliminar el objetivo de una referencia de clave ajena?
- ¿Qué deberá suceder si hay un intento de modificar la clave primaria del objetivo de una referencia de clave ajena?

Formalmente, sean $r_1\,(R_1)$ y $r_2\,(R_2)$ relaciones con las claves primarias C_1 y C_2 respectivamente, se dice que un subconjunto α de R_2 es una clave externa que hace referencia a C_1 de la relación r_1 si se exige que, para cada tupla t_2 de r_2, deba haber una tupla t_1 de r_1 tal que $t_1\,[C_1] = t_2\,[\alpha]$.

La sintaxis para crear una restricción de integridad referencial es:

```
CREATE TABLE nombre (a1 d1,...,an dn,
                     FOREIGN KEY ai REFERENCES tabla(campo)
                     [ON UPDATE RESTRICT|CASCADE|SET NULL)]
                     [ON DELETE RESTRICT|CASCADE|SET NULL)]
                    )
```

Capítulo 12

Otras características

12.1. Vistas

Hasta este momento los ejemplos se han limitado a operar en el nivel de los modelos lógicos. Es decir, se ha dado por supuesto que las relaciones facilitadas son las relaciones reales almacenadas en la base de datos.

No resulta deseable que todos los usuarios vean el modelo lógico completo. Las consideraciones de seguridad pueden exigir que se oculten ciertos datos a los usuarios. Aparte de las consideraciones de seguridad, puede que se desee crear un conjunto personalizado de relaciones que se adapte mejor a la intuición de un usuario determinado que el modelo lógico.

Las relaciones que no forman parte del modelo lógico pero se hacen visibles a los usuarios como relaciones virtuales se denominan vistas. Es posible definir un gran número de vistas de cualquier conjunto dado de relaciones reales.

Las vistas en SQL se definen mediante la instrucción *create view*. Para definir una vista hay que darle un nombre e indicar la consulta que la va a calcular. La forma de la instrucción *create view* es:

```
CREATE VIEW nombre AS expresion [WITH CHECK OPTION]
```

donde *expresión* es cualquier expresión legal de consulta.

Para borrar una vista se utiliza:

```
DROP VIEW nombre
```

Los usuarios pueden trabajar en las vistas como si fueran una tabla real, pues el sistema convierte esta operación en una equivalente realizada sobre la tabla (o tablas) subyacente. La conversión se hace combinando la instrucción del usuario con la expresión de la vista.

12.2. Seguridad

Se pueden asignar a los usuarios varios tipos de autorización para diferentes partes de la base de datos. Por ejemplo:

- La autorización de lectura.
- La autorización de inserción.
- La autorización de actualización.
- La autorización de borrado.

Cada uno de estos tipos de autorización se denomina privilegio. Se puede conceder a cada usuario todos estos tipos de privilegios, ninguno de ellos o una combinación de los mismos sobre partes concretas de la base de datos, como puede ser una relación o una vista.

La norma de SQL incluye los privilegios *select*, *insert*, *update* y *delete*. El privilegio *select* autoriza al usuario a leer los datos. Además de estas formas de privilegio para el acceso a los datos, SQL soporta otros privilegios, como el de crear, borrar o modificar relaciones y ejecutar procedimientos.

El usuario que crea una relación nueva recibe de manera automática todos los privilegios sobre esa relación. Se puede permitir al usuario al que se le ha concedido alguna forma de autorización que transmita (conceda) esa autorización a otros usuarios o que retire (revoque) una autorización concedida previamente.

La instrucción *grant* se utiliza para conceder autorizaciones. La forma básica de esta instrucción es:

```
GRANT SELECT|INSERT|UPDATE|DELETE ON tabla TO usuario
```

La autorización *update* puede concederse sobre todos los atributos de la relación o solo sobre algunos. Si se incluye la autorización *update* en una instrucción grant, la lista de atributos sobre los que se concede la autorización *update* puede aparecer entre paréntesis justo después de la palabra clave *update*. Si se omite la lista de atributos, el privilegio *update* se concede sobre todos los atributos de la relación.

El privilegio *insert* también puede especificar una lista de atributos; cualquier inserción en la relación debe especificar solo esos atributos y el sistema asigna al resto de los atributos valores predeterminados (si hay alguno definido para ellos) o los define como nulos.

El nombre de usuario *public* hace referencia a todos los usuarios actuales y futuros del sistema. Por tanto, los privilegios concedidos a *public* se conceden de manera implícita a todos los usuarios actuales y futuros.

Para retirar una autorización se emplea la instrucción *revoke*. Su forma es casi idéntica a la de *grant*:

```
REVOKE privilegios ON tabla FROM usuarios
```

Parte VI

Otros temas

Capítulo 13

Catalogo y optimizaciones

13.1. Catalogo

Un sistema de bases de datos relacionales necesita tener datos sobre las tablas, como puede ser su esquema. Esta información se denomina diccionario de datos o catálogo del sistema. Entre los tipos de información que debe guardar el sistema figuran los siguientes:

- El nombre de las relaciones.
- El nombre de los atributos de cada relación.
- El dominio y la longitud de los atributos.
- El nombre de las vistas definidas en la base de datos, y la definición de esas vistas.
- Las restricciones de integridad (por ejemplo, las restricciones de las claves).

Además, muchos sistemas guardan los siguientes datos de los usuarios del sistema:

- El nombre de los usuarios autorizados.
- La autorización y la información sobre las cuentas de los usuarios.
- Las contraseñas u otra información utilizada para autentificar a los usuarios.

Por ultimo, la base de datos puede guardar información estadística y descriptiva sobre las relaciones, como:

- El número de filas de cada tabla.
- El método de almacenamiento utilizado para cada relación.

El catalogo esta almacenado en forma de tablas dinámicas que mantiene el DBMS, y utiliza esta información para chequear autorizaciones y optimizar las consultas entre otras cosas. Algunas de las tablas del catalogo son:

- SYSTABLES: Esta tabla contiene una fila por cada tabla del sistema. Proporciona el nombre de la tabla, usuario, cantidad de columnas y otros datos.
- SYSCOLUMNS: Esta table contiene una fila por cada columna de cada tabla mencionada en la tabla anterior. Proporciona el nombre de la columna, de la tabla a la cual pertenece, dominio, etc.
- SYSINDEXES: Contiene una fila por cada indice en el sistema. Proporciona nombre del indice, nombre de la tabla indexada, creador, etc.
- SYSUSER: Contiene una fila por cada usuario en el sistema. Proporciona el nombre del usuario, clave de acceso, nivel de privilegio, etc.
- SYSTABAUTH: Contiene información sobre privilegios de acceso a nivel tabla. Proporciona propietario de la tabla, beneficiario del privilegio, nombre de la tabla, código de privilegio, etc.
- SYSCOLAUTH: Contiene una fila por cada privilegio columna atribuido a un usuario. Proporciona: propietario de la tabla, beneficiario del privilegio, nombre de la columna, etc.
- SYSVIEWS: Contiene sentencias de definición de vistas. Proporciona nombre de la vista, creador, instrucción, etc.

Como la información del catalogo esta almacenada en forma de tablas, estas pueden consultarse normalmente mediante SQL. Por ejemplo, puede consultarse cuales tablas tienen una columna *sCod* con la instrucción:

```
SELECT tbname FROM syscolumns WHERE name = "sCod"
```

El catalogo no se puede actualizar utilizando *update*, *delete* o *insert*.

13.2. Optimizaciones

La optimización de consultas es el proceso de selección del plan de evaluación de las consultas más eficiente de entre las muchas estrategias generalmente disponibles para el procesamiento de una consulta dada, especialmente si la consulta es compleja. No se espera que los usuarios escriban las consultas de modo que puedan procesarse de manera eficiente. Por el contrario, se espera que el sistema cree un plan de evaluación que minimice el coste de la evaluación de las consultas. Ahí es donde entra en acción la optimización de consultas.

La diferencia en coste (en términos de tiempo de evaluación) entre una estrategia buena y una mala suele ser sustancial, y puede resultar de varios órdenes de magnitud. Por tanto, merece la pena que el sistema invierta una cantidad importante de tiempo en la selección de una buena estrategia para el procesamiento de la consulta, aunque esa consulta solo se ejecute una vez.

A modo de ejemplo, consideremos la siguiente consulta, donde la tabla autores tiene 100 filas y la tabla libros tiene 10.000 de los cuales solo 50 son de terror:

$$\Pi_{nombre}\left[\sigma_{autor=id \wedge genero=rerror}\left(libros \times autores\right)\right]$$

Para calcular el producto cartesiano es necesario leer 100 veces cada uno de los 10.000 libros, es decir 1.000.000 lecturas. Dicha cantidad seguramente no quepa en la memoria principal por lo que habrá que realizar 1.000.000 grabaciones de tuplas al disco. Sobre ese resultado se deben filtrar el genero terror por lo que hay que volver a leer 1.000.000 de filas del disco. Todo esto nos ha costado 3.000.000 de accesos al disco, para una tabla final de 50 filas.

En forma alternativa, podríamos haber planteado la siguiente consulta equivalente:

$$\Pi_{nombre}\left\{\sigma_{id=autor}\left[\sigma_{genero=terror}\left(libros\right) \times autores\right]\right\}$$

De esta manera necesitamos 10.000 accesos al disco para producir las primeras 50 filas y luego leemos 50 veces los 100 autores para juntar los resultados, que ahora si entran en memoria. En total hemos necesitado 15.000 accesos al disco para generar la misma consulta. Con solo cambiar el orden de las operaciones, hemos obtenido una consulta 200 veces mas rápida.

Ademas, si la tabla de libros estuviera indexada por genero, solo necesitaríamos 50 accesos a disco en vez de 10.000. Esto incrementa notablemente a una perfomance 600 veces mas rápida.

El proceso de optimización consiste en transformar la consulta al álgebra relacionar y tratar de encontrar una forma equivalente mas eficiente que la original.

En general se intenta efectuar las selecciones antes que las reuniones, pues reducen el tamaño de entrada a la reunión y el tamaño de salida por lo que quizás se pueda conservar dicho resultado en la memoria.

Otra estrategia de optimización es convertir cada condición de restricción a una condición en forma normal conjuntiva. Es decir, a un conjunto de disyunciones enlazadas por conjunciones. Por ejemplo: $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$. Esto tiene la ventaja de que si una de las disyunciones es falsa, no es necesario seguir evaluando el resto.

En caso de proyecciones sobre una misma tabla, se puede hacer caso omiso a todas, a excepción de la ultima. También es posible intercambiar el orden de una restricción seguida de una proyección para poder aprovechar mas optimizaciones.

Luego de convertir la consulta, el optimizador deberá evaluarla, considerando:

- La distribución de los datos almacenados.
- La existencia de índices u otras rutas de acceso.
- El agrupamiento físico de los registros, etc.
- Nro. de tuplas de cada relación.
- Nro. de valores distintos de un atributo.
- Tamaño del registro de una relación.

En base a ello, construye planes de consulta candidatos y elige el de menor costo. El costo es en esencia, una estimación de las operaciones de entrada/salida de disco requeridas.

Capítulo 14

Recuperación y concurrencia

14.1. Recuperación

El término transacción hace referencia a un conjunto de operaciones que forman una única unidad lógica de trabajo. Por ejemplo, una transferencia de fondos desde una cuenta corriente a otra es una operación simple desde el punto de vista del cliente; sin embargo, en el sistema de base de datos, está compuesta internamente por varias operaciones.

Consideremos la secuencia de operaciones para efectuar una transferencia bancaria:

```
EXEC SQL WHENEVER SQLERROR GO TO ANULAR;
EXEC SQL UPDATE cuentas SET saldo = saldo + 100 WHERE cliente = "Ariel";
EXEC SQL UPDATE cuentas SET saldo = saldo - 100 WHERE cliente = "Damian";
EXEC SQL COMMIT;
GO TO TERMINAR;
ANULAR:
        EXEC SQL ROLLBACK;
TERMINAR:
        RETURN;
```

Evidentemente es esencial que tengan lugar todas las operaciones o que, en caso de fallo, ninguna de ellas se produzca. Sería inaceptable efectuar el deposito en la cuenta de destino y que no se abonase en la cuenta de origen.

El DBMS tiene un componente llamado manejador de transacciones que se encargara de ejecutar alguna de las modificaciones y si se presenta una falla antes de que llegue a término normal de la transacción, se anularán esas modificaciones. Es decir, la transacción se lleva a cabo en su totalidad, o se cancela en su totalidad (como sí jamás se hubiera ejecutado).

De eso precisamente se encargan las instrucciones *commit* y *rollback*:

- Commit le dice al manejador de transacciones que ha finalizado con éxito una unidad lógica, que la BD está de nuevo en un estado consistente y que se pueden comprometer o hacer permanentes todas las modificaciones efectuadas.
- Rollback señala el término no exitoso de la transacción. Le dice al manejador de transacciones que algo salió mal, que la BD podría estar en un estado inconsistente y que todas las modificaciones deben retroceder o anularse.

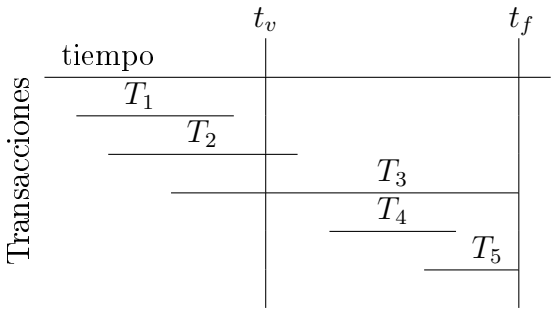
Para que esto sea posible, el sistema mantiene un registro o diario donde se registran los detalles de todas las operaciones de actualización, en particular los valores inicial y final del objeto modificado. Si resulta necesario anular alguna modificación, el sistema puede utilizar la entrada correspondiente del registro para restaurar el valor original del objeto modificado. El registro se deberá grabar físicamente antes de poderse completar el procesamiento de una instrucción *commit*. Así el procedimiento de reinicio recuperará todas las transacciones completadas con éxito pero cuyas modificaciones no lograron grabarse físicamente antes de la caída.

Se le llama punto de sincronización al límite entre dos transacciones consecutivas, de modo que corresponde al final de una unidad de trabajo, y por tanto al punto en el cual la BD está (o debería estar) en un estado de consistencia. Las únicas operaciones que establecen un punto de sincronización son *commit*, *rollback* y el inicio de un programa.

Una sola ejecución de programa puede incluir varias transacciones. Si una transacción se compromete con éxito, el sistema deberá garantizar el establecimiento permanente de sus modificaciones en la BD, aún si el sistema cae en el instante siguiente.

Para poder saber cuales transacciones anular y cuales volver a realizar luego de una caída; cada cierto intervalo previamente establecido, el sistema «establece un punto de revisión» de manera automática. El registro de punto de revisión incluye una lista de todas las transacciones que se estaban realizando en el momento de establecerse el punto de revisión.

Veamos un ejemplo de como se usa la información. Se presento una falla en un momento t_f y el ultimo punto de verificación fue tomando en el momento t_v. El siguiente cuadro modela la situación:



Cuadro 14.1: Linea de tiempo de una falla

Al reiniciarse el sistema, deberán:

- Anularse las transacciones del tipo T_3 y T_5, pues no llegaron a hacer *commit*.
- Deberán realizarse de nuevo las transacciones del tipo T_2 y T_4, pues si lograron hacerlo.
- Las del tipo T_1 no entran porque sus modificaciones ya fueron grabadas físicamente en la BD en el momento t_v como parte del proceso de punto de revisión.

14.2. Concurrencia

14.2.1. Descripción

En los sistemas de bases de datos en los que se ejecutan de manera concurrente varias transacciones, si no se controlan las actualizaciones de los datos compartidos, existe la posibilidad de que las transacciones operen sobre estados intermedios inconsistentes creados por las actualizaciones de otras transacciones. Esta situación puede dar lugar a actualizaciones erróneas de los datos almacenados en la base de datos. Por tanto, los sistemas de bases de datos deben proporcionar los mecanismos para aislar las transacciones de otras transacciones que se ejecuten de manera concurrente. Esta propiedad se denomina aislamiento.

14.2.2. Problemas

Los problemas que se pueden presentar en los cuales una transacción correcta puede producir un resultado incorrecto debido a una interferencia de otra transacción son:

- El problema de la modificación perdida.
- El problema de la dependencia no comprometida.
- El problema del análisis inconsistente.

Estudiaremos a continuación, en que consisten cada uno de estos problemas y como podrían solucionarse.

14.2.2.1. Problema de la modificación perdida

Para estudiar este problema, consideremos la situación en la que el cliente Damian recibe dos transferencias de dinero simultaneas por $100 y $200. El siguiente cuadro modela la situación:

Transacción A	Tiempo	Transacción B
`x = consultarSaldo("Damian")`	t_1	
	t_2	`y = consultarSaldo("Damian")`
`x = x + 100`	t_3	
	t_4	`y = y + 200`
`actualizarSaldo("Damian", x)`	t_5	
	t_6	`actualizarSaldo("Damian", y)`

Cuadro 14.2: Problema de la modificación perdida

Si el saldo inicial de Damian era $0, el saldo final será de $200 cuando en realidad debería ser de $300, puesto que la segunda transacción a leído el saldo antes de que la primera lo actualice.

14.2.2.2. Problema de la dependencia no comprometida

Se presenta cuando se permite a una transacción leer (o modificar) un registro que ha sido actualizado por otra transacción y esta última todavía no lo ha comprometido.

Transacción A	Tiempo	Transacción B
`incrementarSaldo("Damian", 100)`	t_1	
	t_2	`extraerDinero("Damian", 100)`
`ROLLBACK`	t_3	

Cuadro 14.3: Problema de la dependencia no comprometida

Si el saldo inicial de Damian era $0, este ha podido retirar un dinero que en realidad nunca fue ingresado, ya que por alguna razón la transacción no se llego a comprometer.

14.2.2.3. Problema del análisis inconsistente

Transacción A	Tiempo	Transacción B
`total = consultarSaldo("Damian")`	t_1	
	t_2	`x = consultarSaldo("Marotte")`
	t_3	`actualizarSaldo("Marotte", x-100)`
	t_4	`y = consultarSaldo("Damian")`
	t_5	`actualizarSaldo("Damian", y-100)`
	t_6	`COMMIT`
`total += consultarSaldo("Marotte")`	t_7	

Cuadro 14.4: Problema del análisis inconsistente

Si ambos saldos iniciales eran de $100, la transacción A esta realizando un análisis inconsistente pues ha determinado que el saldo total es de $100, cuando en realidad es de $0.

A diferencia del ejemplo anterior, en este caso A no depende en absoluto de una modificación no comprometida, pues B compromete todas sus modificaciones.

14.2.3. Soluciones

Como ejemplo trivial de esquema de control de concurrencia considérese éste: una transacción realiza un bloqueo en la base de datos completa antes de comenzar y lo libera después de haberse comprometido. Mientras una transacción mantiene el bloqueo, no se permite que ninguna otra lo obtenga, y todas ellas deben esperar hasta que se libere el bloqueo.

Como resultado de esta política de bloqueo, solo se puede ejecutar una transacción cada vez. Un esquema de control de concurrencia como éste produce un rendimiento pobre, ya que fuerza a que las transacciones esperen a que finalicen las precedentes para poder comenzar. En otras palabras, produce un grado de concurrencia pobre.

Existen muchos modos mediante los cuales se puede bloquear un elemento de datos. En este apartado se centra la atención en dos de dichos modos:

- COMPARTIDO: Si una transacción T_i obtiene un *bloqueo en modo compartido* (denotado por C) sobre el elemento Q, entonces T_i puede leer Q pero no lo puede escribir.
- EXCLUSIVO. Si una transacción T_i obtiene un *bloqueo en modo exclusivo* (denotado por X) sobre el elemento Q, entonces T_i puede tanto leer como escribir Q.

Es necesario que toda transacción solicite un bloqueo del modo apropiado sobre el elemento de datos Q dependiendo de los tipos de operaciones que se vayan a realizar sobre Q. La petición se hace al gestor de control de concurrencia. La transacción puede realizar la operación solo después de que el gestor de control de concurrencia conceda el bloqueo a la transacción.

Dado un conjunto de modos de bloqueo, se puede definir sobre ellos una función de compatibilidad como se muestra en la siguiente tabla:

	C	X	$-$
C	✓	×	✓
X	×	×	✓
$-$	✓	✓	✓

Cuadro 14.5: Matriz de compatibilidad de bloqueo

Si a una transacción se le ha denegado un bloqueo, está entrara en modo de espera hasta que la otra transacción libere el bloqueo.

Las solicitudes de bloqueos sobre registros por parte de las transacciones son implícitas en condiciones normales. Es decir, cuando una transacción lee un registro, adquiere automáticamente un bloqueo del tipo C sobre él y cuando una transacción actualiza un registro, adquiere automáticamente un bloqueo del tipo X sobre él.

Observemos ahora como se ha solucionado el problema del análisis inconsistente:

Transacción A	Tiempo	Transacción B
`incrementarSaldo("Damian", 100)` $(X \equiv \checkmark)$	t_1	
	t_2	`extraerDinero("Damian", 100)` $(X \equiv \times)$
`ROLLBACK`	t_3	⋮

Cuadro 14.6: Solución del problema de la dependencia no comprometida

14.2.4. Bloqueo mutuo

Desafortunadamente, el uso de bloqueos puede conducir a una situación no deseada. El bloqueo mutuo es una situación en la cual dos o más transacciones están en un estado de espera simultáneo y cada uno espera la liberación del otro para poder continuar.

Transacción A	Tiempo	Transacción B
`x = consultarSaldo("Damian")` $(C \equiv \checkmark)$	t_1	
	t_2	`y = consultarSaldo("Damian")` $(C \equiv \checkmark)$
`x = x + 100`	t_3	
	t_4	`y = y + 200`
`actualizarSaldo("Damian", x)` $(X \equiv \times)$	t_5	
⋮	t_6	`actualizarSaldo("Damian", y)` $(X \equiv \times)$
⋮		⋮
⋮	t_7	

Cuadro 14.7: Bloqueo mutuo en problema de la modificación perdida

Transacción A	Tiempo	Transacción B
`total = consultarSaldo("Damian")` $(C \equiv \checkmark)$	t_1	
	t_2	`x = consultarSaldo("Marotte")` $(C \equiv \checkmark)$
	t_3	`actualizarSaldo("Marotte", x-100)` $(X \equiv \checkmark)$
	t_4	`y = consultarSaldo("Damian")` $(C \equiv \checkmark)$
	t_5	`actualizarSaldo("Damian", y-100)` $(X \equiv \times)$
`total += consultarSaldo("Marotte")` $(C \equiv \times)$	t_6	⋮
⋮		⋮
	t_7	⋮

Cuadro 14.8: Bloqueo mutuo en problema del análisis inconsistente

Cuando aparece un bloqueo mutuo, el sistema debe retroceder una de las dos transacciones. Una vez que se ha provocado el retroceso de una de ellas, se desbloquean los elementos de datos que estuvieran bloqueados por la transacción. Estos elementos de datos están disponibles entonces para otra transacción, la cual puede continuar su ejecución.

Capítulo 15

Sistemas distribuidos

15.1. Descripción

En un sistema distribuido de bases de datos se almacena la base de datos en varias computadoras. Los medios de comunicación como las redes de alta velocidad o las líneas telefónicas pueden poner en contacto las distintas computadoras de un sistema distribuido. No comparten ni memoria ni discos. Las computadoras de un sistema distribuido pueden variar en tamaño y función pudiendo abarcar desde las estaciones de trabajo a los grandes sistemas.

Las principales diferencias entre las bases de datos centralizadas y las bases de datos distribuidas son que las bases de datos distribuidas normalmente se encuentran en varios lugares geográficos distintos, se administran de forma separada y poseen una interconexión más lenta. Otra gran diferencia es que en un sistema distribuido existen dos tipos de transacciones: locales y globales. Una transacción local es aquella que accede a los datos del único sitio en el cual se inició la transacción. Por otra parte, una transacción global es la que, o bien accede a los datos situados en un sitio diferente de aquél en el que se inició la transacción, o bien accede a datos de varios sitios distintos.

Un sistema de base de datos distribuida (BDD) deberá ser capaz de trabajar en forma transparente, es decir que la aplicación trabajaría desde un punto de vista lógico, como si un solo DBMS ejecutado en una sola máquina administrara todos los datos.

Se compone de un conjunto de sitios conectados entre sí mediante algún tipo de red de comunicaciones, en el cual cada sitio es un sistema de BD en sí mismo, es decir, cada sitio tiene sus BD reales locales, sus propios usuarios locales, sus propios DBMS y programas para la administración de transacciones.

15.2. Ventajas

Existen varias razones para construir sistemas distribuidos de bases de datos, incluyendo el compartimiento de los datos, la autonomía y la disponibilidad:

- DATOS COMPARTIDOS: La principal ventaja de construir un sistema distribuido de bases de datos es poder disponer de un entorno donde los usuarios puedan acceder desde una única ubicación a los datos que residen en otras ubicaciones.

- AUTONOMÍA: La principal ventaja de compartir datos por medio de distribución de datos es que cada ubicación es capaz de mantener un grado de control sobre los datos que se almacenan localmente. En un sistema centralizado, el administrador de bases de datos de la ubicación central controla la base de datos. En un sistema distribuido, existe un administrador de bases de datos global responsable de todo el sistema. Una parte de estas responsabilidades se delegan al administrador de bases de datos local de cada sitio. Dependiendo del diseño del sistema distribuido de bases de datos, cada administrador puede tener un grado diferente de autonomía local. La posibilidad de autonomía local es a menudo una de las grandes ventajas de las bases de datos distribuidas.

- DISPONIBILIDAD: Si un sitio de un sistema distribuido falla, los sitios restantes pueden seguir trabajando. En particular, si los elementos de datos están replicados en varios sitios, una transacción que necesite un elemento de datos en particular puede encontrarlo en varios sitios. De este modo, el fallo de un sitio no implica necesariamente la caída del sistema. El sistema puede detectar el fallo de un sitio y es posible que sea necesario aplicar acciones apropiadas para la recuperación del fallo. El sistema no debe seguir utilizando los servicios del sitio que falló. Finalmente, cuando el sitio que falló se recupera o se repara, debe haber

mecanismos disponibles para integrarlo sin problemas de nuevo en el sistema. Aunque la recuperación ante un fallo es más compleja en los sistemas distribuidos que en los sistemas centralizados, la capacidad que tienen muchos sistemas de continuar trabajando a pesar del fallo en uno de los sitios produce una mayor disponibilidad. La disponibilidad es crucial para los sistemas de bases de datos que se utilizan en aplicaciones de tiempo real.

15.3. Desventajas

La complejidad de estos sistemas, también los dota de una serie de desventajas. A saber:

- Falta de experiencia generalizada.
- Si no hay un buen diseño y organización trae mayor complejidad: problemas del centralizado y problemas del distribuido.
- Puede aumentar costos iniciales: Hardware y software de comunicación y distribución.
- Se debe aumentar los controles de seguridad respecto a BD centralizadas.
- Complejidad de los sistemas distribuidos (desde el punto de vista técnico).

15.4. Principios fundamentales

Desde el punto de vista del usuario, un sistema distribuido deberá ser idéntico a un sistema no distribuido. Es decir que las operaciones de DML no deberán sufrir cambios, aunque las operaciones de DDL requerirán cierta ampliación.

Para lograr este cometido, es recomendable respetar las siguientes doce reglas:

1. AUTONOMÍA LOCAL: Los sitios deben ser autónomos. Todas las operaciones en un sitio dado se controlan de ese sitio. Ningún sitio debe depender de otro para su correcto funcionamiento.

2. NO DEPENDENCIA CENTRAL: Todos los sitios deben tratarse por igual. No debe haber dependencia de un sitio central para obtener un servicio. Si el sitio central sufriera un desperfecto todo el sistema dejaría de funcionar.

3. OPERACIÓN CONTINUA: Nunca debería haber necesidad de apagar el sistema para realizar alguna función.

4. INDEPENDENCIA DE LOCALIZACIÓN: Los usuarios no deberían necesitar saber dónde están almacenados físicamente los datos. Debe comportarse desde el punto de vista lógico como si todos los datos estuvieran almacenados en su propio sitio local.

5. INDEPENDENCIA DE FRAGMENTACIÓN: Los usuarios tendrán una vista de los datos con fragmentos combinados lógicamente mediante reuniones y uniones apropiadas. El optimizador determina a cuáles fragmentos físicos es necesario tener acceso para satisfacer cualquier solicitud del usuario.

6. INDEPENDENCIA DE REPLICA: Los usuarios deberán comportarse lógicamente como si existiera una sola copia aunque en realidad existan varias. Debe ser transparente para el usuario. Las réplicas son deseable por dos razones: puede producir un mejor desempeño (las aplicaciones pueden operar sobre copias locales en vez de tener que comunicarse con sitios remotos) y brindan mayor disponibilidad (un objeto está disponible para su procesamiento en tanto esté disponible por lo menos una copia , al menos para propósitos de recuperación).

7. PROCESAMIENTO DISTRIBUIDO: La optimización es más importante en un sistema distribuido que en el centralizado. Una solicitud de unión de una relación R_x almacenada en el sitio X y una R_y en Y, podría llevarse a cabo: trasladando R_x a Y, ó trasladando R_y a X ó trasladando las dos a un tercer sitio. El optimizador distribuido deberá encontrar la estrategia más eficiente.

8. Transacciones distribuidas: En las BDD el sistema debe asegurarse que todos los agentes correspondientes a la transacción se comprometan o retrocedan al unísono.

9. Independencia de equipo: Es conveniente ejecutar el mismo DBMS en diferentes equipos y presentar al usuario una sola imagen del sistema.

10. Independencia de sistema operativo: Se debe poder ejecutar el mismo DBMS en diferentes equipos y sistemas operativos.

11. Independencia de red: Se debe poder manejar varias redes de comunicación distintas.

12. Independencia de DBMS: Los DBMS en los distintos sitios deben manejar la misma interfaz.

Bibliografía

[1] Abraham Silberschatz & Henry Korth - Fundamentos de Bases de Datos (*5ta edición*).

[2] Claudia Deco - Cátedra de Teoría de Bases de Datos (*FCEIA*).