Resumen Diseño de Bases de Datos

¿Qué es una base de datos?

Es una colección de datos relacionados. Los cuales pueden ser un conjunto de archivos que son diseñados para servir a ciertas aplicaciones

Propiedades de una BD:

Una BD representa algunos aspectos del mundo real, a veces denominado universo de discurso

Una BD es una colección coherente de datos con significados inherentes. Un conjunto aleatorio de datos no puede considerarse una BD. O sea, los datos deben tener cierta lógica

Una BD está sustentada físicamente en archivos en dispositivos de almacenamiento persistente de datos

¿Qué es el DBMS?

Sus siglas significan Data Base Management System o Sistema Gerenciador de Bases de Datos

Es una colección de programas que permiten a los usuarios crear y mantener la BD

Es un sistema de software de propósito general que facilita los procesos de definición, construcción y manipulación de BD

Objetivos del DBMS:

Evitar la redundancia e inconsistencia de datos

Permitir acceso a los datos en todo momento

Evitar anomalías en el acceso concurrente

Restricción a accesos no autorizados

Suministro de almacenamiento persistente de datos(aún ante fallos)

Integridad de datos

Backups

Componentes de un DBMS:

DDL(data definition languaje): especifica el esquema de BD. Resultado: Directorio de archivos

DML(Data manipulation languaje):

Recuperación de información

Agregar información

Quitar información

Modificar información

Características DML:

Procedimentales(SQL): Requieren que el usuario especifique que datos se muestran y cómo obtener esos datos

No procedimentales(QBE): requieren que el usuario especifique qué datos se muestran sin especificar cómo obtener esos datos

Actores involucrados con una BD

DBA o ADB:

Administra el recurso, que es la BD. Autoriza accesos, coordina y vigila la utilización de recursos de hardware y software, responsable ante problemas de violación

Diseñador de BD:

Definen la estructura de la BD de acuerdo al problema del mundo real que es presentado

Analistas de sistemas

Determinan los requerimientos de los usuarios finales, generando la información necesaria para el diseñador

Programadores

Implementan las especificaciones de los analistas utilizando la BD generada por el diseñador

Usuarios(distintos tipos)

Niveles de Visión de los datos

Para que el sistema de software sea útil, éste debe interactuar con una BD y recuperar su información en el menor tiempo posible. Bajo esta circunstancia, la representación de la información de una BD muchas veces utiliza estructuras complejas. Esta complejidad se oculta a los usuarios a través de distintos niveles de abstracción.

Nivel de vista: corresponde al nivel más alto de abstracción. En este nivel se describe parcialmente la BD, solo la parte que se desea ver. Es posible generar diferentes vistas de la BD, cada una correspondiente a la parte a consultar

Nivel conceptual: en este nivel se describe la BD completa, indicando que datos se almacenarán y las relaciones existentes entre esos datos.

Nivel Físico: este es el nivel más bajo de abstracción, en el cual describe cómo se almacenan realmente los datos. Se detallan las estructuras de datos mas complejas a bajo nivel

Modelado

Introducción

Es una colección de herramientas conceptuales para describir datos, relaciones entre ellos, semántica asociada a los datos y restricciones de consistencia

Tipos de modelados:

Basado en objetos(visión conceptual): Estructura flexible, especifican restricciones explícitamente

Modelo Entidad-Relación

Modelo Orientado a Objetos

Basado en registros(conceptual físico): La BD se estructura en registros de formato fijo. Se dispone de lenguaje adecuado para realizar consultas

OO

relacional

Diseño de datos: tres etapas

Conceptual(representación abstracta)

- Integración de vistas
- Lógico(representación en una computadora)
- Físico(determinar estructuras de almacenamiento físico)

Un modelo de datos es un conjunto de herramientas conceptuales que permiten describir la información que es necesario administrar para un SI, las relaciones existentes entre estos datos, la semántica asociada y las restricciones de consistencia

- Se modela para
  - Obtener la perspectiva de cada actor asociado al problema
  - Obtener la naturaleza y necesidad de cada dato
  - Observar como cada actor utiliza cada dato

Modelo Entidad Relación(ER)

Características

- Estándar internacional desde 1988
  - Propuesto por Chen en 1976
  - Ampliado por Codd en 1979
  - Se basa en la concepción del mundo real como un conjunto de objetos llamados entidades y las relaciones que existen entre ellas
  - Permite modelar el nivel conceptual y lógico de una BD
  - Expresividad: disponer de todos los medios necesarios para describir un problema
  - Formalidad: cada elemento representado sea preciso y bien definido, con una sola interpretación posible
  - Minimalidad: cada elemento tiene una única representación posible
  - Simplicidad: el modelo debe ser fácil de entender por el cliente y el desarrollador

Objetivos

- Representar la información de un problema de alto nivel de abstracción
- Captar la necesidad de un cliente respecto del problema que enfrenta

Mejora la interacción cliente/desarrollador disminuyendo la brecha entre la realidad del problema y el sistema a desarrollar

Componentes

Entidades

Representa un elemento u objeto del mundo real con identidad

Se diferencia de cualquier otro objeto o cosa

Conjunto de entidades

Representación que a partir de las características propias de entidad con propiedades comunes, se resume en un núcleo

Relaciones

Representan agregaciones entre dos(binaria) o mas entidades

Ej: el alumno Pérez cursa Matemática 1

Conjunto de relaciones

Es una representación que, a partir de las características de cada relación existente entre dos entidades, las resume en un núcleo

Atributos

Representa una propiedad básica de una entidad o relación

Equivale a un campo de un registro

Cardinalidad de atributos

Monovalente/polivalente

Obligatorio/opcional(nulo)

(1,1):Monovalente obligatorio. La cardinalidad existe y esta presente, pero solamente en este caso no se indica de forma específica

(0,1): Monovalente no obligatorio

(0,N):Polivalente no obligatorio

(1,N): Polivalente obligatorio

Ejemplo:

Tipos de relación

Binaria

Ternaria

N-aria

Recursiva

Cardinalidad de la relación

Define el grado de relación existente en una agregación

Cardinalidad mínima

Cardinalidad máxima

Aclaraciones:

A la hora de hacer un diseño conceptual, no hay que olvidarse la cardinalidad en una relación, es OBLIGATORIO hacerla

Jerarquías de generalización

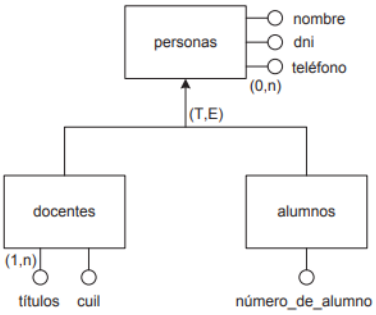
La generalización permite extraer propiedades comunes de varias entidades o relaciones, y generar con ellas una superentidad que las aglutine. Así, las características compartidas son expresadas una única vez en el modelo, y los rasgos específicos de cada entidad quedan definidos en su subentidad

Cobertura:

Total o parcial

Superpuesta o exclusiva

Ejemplo:

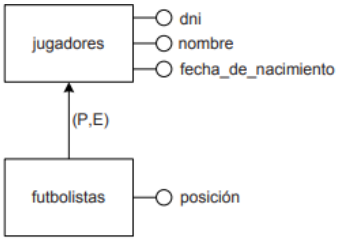


Subconjuntos

Representan un caso especial de las jerarquías de generalización de la que se depende solamente una especialización. Este es el caso de los subconjuntos. La representación, es similar al caso anterior

Solamente hay que considerar que no es necesario indicar la cobertura para los subconjuntos. Esto se debe a que no puede tratarse de una cobertura total; si no, la especialización y la generalización serían lo mismo, representarían los mismos atributos. Además, no puede ser superpuesta, dado que no hay una segunda especialización con la cual superponerse

Ejemplo



Atributos compuestos

Los atributos compuestos representan a un atributo generado a partir de la combinación de varios atributos simples. Un ejemplo para ilustrar esta situación es la dirección de una persona. Se podría optar por modelar la dirección con un solo atributo simple donde se indican la calle, numero, y eventualmente, piso y depto. Así, el dominio podría ser una cadena de 50 caracteres. Sin embargo, es interesante poder mantener la calle, num, piso y departamento por separado.

Un atributo compuesto podría ser polivalente o no obligatorio. Lo mismo podría ocurrir con los atributos simples que lo componen

Identificadores

Es un atributo o conjunto de atributos que permite reconocer a una entidad de manera unívoca dentro del conjunto de entidades

Pueden ser

Simples o compuestos: De acuerdo con la cantidad de atributos que lo conforman, el identificador es simple si está conformado por un solo atributo y es compuesto en el resto de los casos

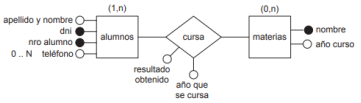


Figura 10.11 a) Identificadores simples

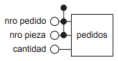


Figura 10.11 b) Identificador Compuesto

Internos o externos: Si todos los atributos que conforman un identificador pertenecen a la identidad que identifica, es interno; en su defecto, es externo

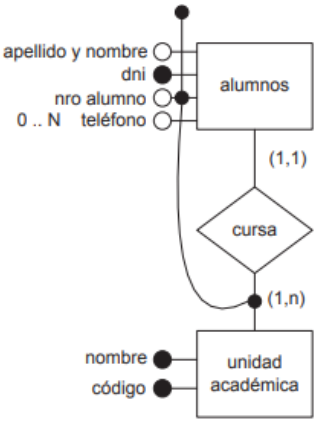


Figura 10.12
Identificador Externo

Revisión del modelo conceptual

Decisiones respecto de entidades, relaciones o atributos

Para construir un modelo conceptual, hay una serie de decisiones que el analista debe tomar. Las decisiones dependen de tres factores fundamentales: el problema, lo que el cliente/usuario espera obtener desde la BD y, por último, la experiencia práctica del analista.

La siguiente pregunta a realizar cuando se genera un modelo conceptual tiene que ver con la ventaja o no de definir generalizaciones. Aquí, la decisión debería tomarse considerando si la representación establece una jerarquía o una clasificación. Por ejemplo, si se quiere definir en el modelo de datos solamente el sexo de una persona, la solución se obtiene agregando un atributo a la entidad persona. Sin embargo, si hubiera características propias para los hombres y para las mujeres, sería conveniente generar una jerarquía. Así, la regla a aplicar debería controlar que no queden definidas entidades sin atributos o sin relaciones con otras entidades. Estas entidades (sin atributos y sin relaciones) se denominan entidades colgadas. La última consideración está ligada, como se explicó anteriormente, a la utilización de atributos compuestos. ¿Conviene un atributo compuesto o directamente se colocan sobre la entidad los atributos simples? Los atributos compuestos se deberían utilizar y luego, cuando se evolucione en el modelo lógico o físico, decidir cómo actuar.

Transformaciones para mejorar el modelo conceptual

Una vez finalizada la construcción del modelo conceptual, este debe ser puesto en consideración y análisis por parte del cliente/usuario. Si bien este no es un proceso sencillo, con la ayuda del analista es posible que alguien no ligado a la actividad informática pueda analizar el modelo resultante para tratar de determinar faltantes errores.

Existe una serie de conceptos a revisar:

- Autoexplicativo: Un modelo se expresa a si mismo si puede sentarse utilizando los elementos definidos, sin necesidad de utilizar aclaraciones en lenguaje natural para expresar características.
- Completitud: un modelo está completo cuando todas las características del problema están contempladas en él. La forma de validar la completitud es

revisar la especificación de requerimientos asociados al problema

- Corrección: un modelo es correcto si cada elemento en su construcción fue utilizado con propiedad. Algunos aspectos de fácil resolución en cuanto a correctitud son observar que todas las cardinalidades y coberturas se hayan expresado, haber tenido en cuenta el concepto de herencia en jerarquías, haber expresado todos los identificadores, etc.
- Expresividad: el modelo conceptual resulta expresivo a partir de su observación es posible darse cuenta de todos los detalles que lo involucran
- Extensible: el modelo conceptual resulta extensible si es fácilmente modificable para incorporar nuevos conceptos en él, resultantes de cambios en los requerimientos del problema
- Legibilidad: un esquema es legible si la representación gráfica es adecuada. Este fue uno de los motivos que llevó a la generación de CASER: disponer de una herramienta para la asistencia en la generación del modelo conceptual, basado en elementos propios del modelo.
- Minimalidad: un esquema es mínimo cuando cada concepto se representa una sola vez en el modelo. Aquí dos factores posibles que pueden afectar la minimalidad. Estos factores son: Atributos derivados y ciclos en relaciones.

Transformaciones para minimalidad

Un esquema es mínimo cuando una idea se expresa en él una sola vez. La redundancia de información puede ser un efecto positivo si es conocida y controlada, o un efecto negativo si aparece en el esquema por deficiencia en la modelización. En estos últimos casos, la redundancia puede llevar a pérdida de consistencia en a información.

Hay dos causas que llevan a un modelo a perder la minimalidad. En este apartado, se analizan las situaciones. Se debe notar que el resultado esperado en este punto es detectar las condiciones que afectan la minimalidad.

Los atributos derivados representan la primera causa que afectan la minimalidad. Un atributo derivado es un atributo que aparece en el modelo, pero cuya información, si no existiera almacenada en él, podría igualmente ser obtenida.

Los ciclos en relaciones presentan la segunda causa que afecta la minimalidad. Se dice que existe un ciclo cuando una entidad A está relacionada con una entidad B, la cual está relacionada con una entidad C, la que a su vez se relaciona con una entidad A.

No todos los ciclos afectan la minimalidad. Un ciclo afecta la minimalidad cuando una de las relaciones existentes puede ser quitada del esquema, y aun en esas condiciones, el modelo sigue representando la misma información.

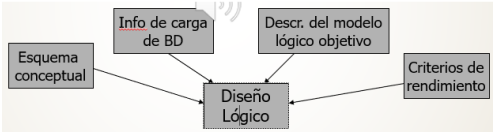
Transformaciones para expresividad y autoexplicación

Las transformaciones para lograr expresividad y autoexplicación tienen varias causas; se analizan algunas de ellas.

Un modelo es más expresivo si las jerarquías que se representan tienen sentido. Una jerarquía se establece para obtener atributos comunes en una generalización, dejando los atributos específicos para la especialización. El problema surge cuando las especializaciones carecen de dichos atributos.

Otras causas que mejoran la expresividad y la autoexplicación están relacionadas con utilizar jerarquías o subconjuntos cuando aún no fueron definidas.

Modelo Lógico



Objetivo

El propósito de la generación de un modelo ER lógico es convertir el esquema conceptual en un modelo más cercano a la representación entendible por el

SGBD. El principal objetivo del diseño conceptual consiste en captar y representar, de la forma más clara posible, las necesidades del usuario definidas en el documento de especificación de requerimientos del problema.

Al iniciar la definición del modelo lógico, se necesita definir el SGBD que se utilizará posteriormente para su implementación física. Esto es, la secuencia de pasos de conversión disponibles tiene estrecha relación con el tipo de SGBD. Recuerde que se mencionarion cuatro tipos diferentes: relacional, OO, jerárquico y de red.

Aclaración: En la materia se ve el tipo RELACIONAL

Características del diseño lógico

- Esquema conceptual: es el resultado tangible de la etapa inmediata anterior. El esquema conceptual, a juico del analista, respecto del problema original. El esquema lógico a obtener debe representar la misma información disponible en el esquema conceptual
- Descripción del modelo lógico a obtener: aquí se deben definir las reglas que se aplicarán en el proceso de conversión. Esas reglas están ligadas al tipo de SGBD seleccionado. Para esta obra se repasarán, el apartado siguiente, aquellas que permiten obtener un modela relacional
- Criterios de rendimiento de la BD: durante la fase de diseño conceptual, se consideraron los requerimientos del usuario. No obstante, hay otro tipo de necesidades que no se pueden definir sobre el modelo conceptual. Estas necesidades tienen que ver con requerimientos, en general, no funcionales del problema, como por ejemplo performance de la BD
- Información de carga de la BD: este concepto aparece, en cierta forma, ligado al concepto anterior. Cuando se genera el esquema lógico, el analista debe observar cada entidad e interrelación definida, y ver la probable evolución de la información contenida en esas estructuras. De este modo, la decisión final sobre el esquema de una relación o entidad dependerá del número probable de elementos que la compondrán, con el propósito de mantener la performance bajo control.

Decisiones sobre el diseño lógico

Las decisiones sobre el diseño lógico están vinculadas, básicamente, con cuestiones generales de rendimiento y con un conjunto de reglas que actúan sobre características del esquema conceptual que no están presentes en los SGBD relacionales

Atributos derivados

Un atributo es derivado si contiene información que puede obtenerse de otra forma desde el modelo. Es importante detectar dichos atributos, y en el diseño lógico se debe tomar la decisión respecto a dejarlos o no

La ventaja de un atributo derivado es, la disponibilidad de información.

La desventaja de un atributo derivado radica en que necesita ser calculado cada vez que se modifica la información que contiene

Por lo tanto, la pauta sobre los atributos derivados es dejar en el modelo a todos aquellos que son muy utilizados, y quitar los que necesitan ser recalculados con frecuencia. Se puede observar que, ante un atributo derivado muy utilizado y con mucho recálculo, no es sencillo establecer una pauta. En dichas situaciones, la decisión queda a criterio del analista.

Ciclos en relaciones

Se deben identificar las relaciones que generan repetición innecesaria de información. Lo que tiene esto, es que si uno elimina esa redundancia, como consecuencia, si se requiere cierto dato que se obtenía rápido acorde a la relación que se eliminó, esto llevará mayor tiempo de espera.

La decisión del analista pasa por tener el modelo mínimo, o por que posteriormente el modelo implique menos tiempo de procesamiento.

Atributos polivalentes

Los SGBD, en general, permiten que sus atributos contengan múltiples valores. Así, es posible que un atributo tenga una dimensión, generando una estructura equivalente a los vectores en memoria. Este tipo de estructura es estática, es decir, la cantidad de lugares previstos para almacenar la información está predeterminada.

Aunque, ningún SGBD relacional permite que un atributo contenga valores múltiples determinados dinámicamente. Es decir, se puede tener un atributo con múltiples valores, pero la cantidad máxima debe ser previamente determinada.

En este caso, la solución debe implementarse con otro criterio. El criterio está establecido y en la bibliografía se lo denomina, básicamente,
primera forma normal.
Un modelo está en Primera Forma Normal (1FN) si todos los atributos de entidades o relaciones son atributos simples.

¿Cuál es la solución propuesta para satisfacer la definición anterior?

Se genera una entidad nueva con ese atributo y se genera una relación entre la nueva entidad, y la que poseía el atributo polivalente.

Ejemplo:

La entidad docentes
tiene definido el atributo títulos como polivalente obligatorio. A fin de cumplir con la 1FN, la solución en este caso consiste en quitar el atributo polivalente de la entidad docentes, generando una nueva entidad denominada títulos, y estableciendo la relación posee entre docentes y títulos

Atributos compuestos

Los SGBD, en general, no soportan esta variante. Por lo tanto, los atributos compuestos resultan de mucho interés para la generación del modelo conceptual, pues permiten evitar tomar decisiones rápidamente con poca información sobre el dominio del problema.

Luego, sobre el modelo lógico es necesario decidir qué hacer con un atributo compuesto.
Existen tres respuestas posibles, cada una de las cuales presenta ventajas y desventajas. El diseñador de la BD es el responsable de optar por una de las tres alternativas.

1. La primera consiste en generar un único atributo que se convierta en la concatenación de todos los atributos simples que contiene el atributo compuesto. Esta solución es simple y sencilla de implantar, pero al unir todos los atributos simples que forman el compuesto, se pierde la identidad de cada atributo simple. Conocer cuánta gente vive en un primer piso no se puede resolver directamente.
2. La segunda plantea definir todos los atributos simples sin un atributo compuesto que los resuma. Con esta solución, el atributo domicilio desaparece y sobre la entidad se definen los atributos calle, num, piso y dpto. La cantidad de atributos aumento, pero esta solución permite al usuario definir cada uno de los datos en forma independiente.
3. La tercera solución presenta una alternativa más radical. Consiste en generar una nueva entidad, la que representa el atributo compuesto, conformada por cada uno de los atributos simples que contiene. Esta nueva entidad debe estar relacionada con la entidad la cual pertenecía el atributo compuesto. Se debe notar que esta solución capta mejor la esencia del atributo compuesto, pero es una opción más compleja.

La elección dependerá del diseñador de la BD. En general, la segunda alternativa es la más utilizada, pues se adapta mejor a la mayoría de las situaciones de la vida real.

Jerarquías

Las decisiones respecto de las jerarquías constituyen el punto más importante de convertir un modelo conceptual en lógico.

El modelo relacional no soporta el concepto de herencia; por consiguiente, las jerarquías no pueden ser representadas.

El proceso de diseño lógico necesita, entonces, encontrar un mecanismo que represente las jerarquías, captando el dominio de conocimiento de estas, bajo un esquema que no administre herencia.

Hay tres opciones para tratar una jerarquía:

1. Eliminar las especializaciones(subentidades o entidades hijas) dejando solo la generalización(entidad padre), la cual incorpora los atributos de sus hijos. Cada uno de estos atributos deberá ser opcional(no obligatorio)
2. Eliminar la entidad generalización(padre), dejando solo las especializaciones. Con esta solución, los atributos del padre deberán incluirse en cada uno de los hijos.
3. Dejar todas las entidades de la jerarquía, convirtiéndola en relaciones uno a uno entre el padre y cada uno de sus hijos. Esta solución permite que las entidades que conforman la jerarquía mantengan sus atributos originales, generando la relación explícita ES_UN entre padre e hijos.

Las tres soluciones no son aplicables en todos los casos. A continuación, se describen diferentes situaciones y se analizan las soluciones posibles, ventajas y desventajas. La cobertura de la jerarquía es la que determina la solución viable en cada caso.
Si la cobertura de la jerarquía fuese parcial, la segunda solución planteada anteriormente no resultaría aplicable. Una cobertura parcial significa que algunos elementos contenidos en la entidad padre no están cubiertos por las especializaciones. Esto implica que, si se quita la entidad padre, dichos elementos no tendrán más cabida en el modelo. Esta conversión genera un modelo lógico que no es equivalente al modelo conceptual, ya que se pierde información. Por lo tanto, la segunda alternativa no es aplicable en el caso de tratarse de cobertura parcial.
Si se analiza la cobertura superpuesta, la segunda solución, nuevamente, resulta poco práctica. Algunos elementos del padre se repiten en varios hijos; esto significa que se deberá repetir información en las subentidades generadas. Si bien esto no representa un problema en sí mismo,

la repetición innecesaria de información puede ocasionar inconvenientes cuando se utilice la BD generada.

Modelo Físico

El modelo relacional representa a una BD como una colección de archivos denominados tablas, las cuales se conforman por registros.
Cada tabla se denomina relación, y está integrada por filas horizontales y columnas verticales. Cada fila representa un registro del archivo y se denomina tupla, mientras que cada columna representa un atributo del registro.
La definición matemática de las relaciones se desarrolla a partir de la noción de dominio. Un dominio representa un conjunto de posibles valores para un atributo. Como un dominio restringe los valores del atributo, puede considerarse como una restricción.

Un modelo físico (relacional) representa la BD como una colección de relaciones

- Un atributo mantiene su nombre
- Cada tabla de valores resultante se denomina relación
    - Cada relación se obtiene a partir de una entidad o una relación ER

Pasos

- Eliminación de identificadores externos
- Selección de claves
    - Primaria
    - Candidata
- Conversión de entidades
- Relaciones

Eliminación de identificadores externos

El primer paso en la conversión del esquema lógico hacia el esquema físico consiste en la eliminación de los identificadores externos. Cada una de las entidades que conforman el esquema lógico debe poseer sus identificadores definidos en forma interna. Para lograr esto, se deberán incorporar, dentro de la entidad que contenga los identificadores externos, aquellos atributos que permitan la definición del identificador de forma interna a la entidad

Veamos el siguiente ejemplo

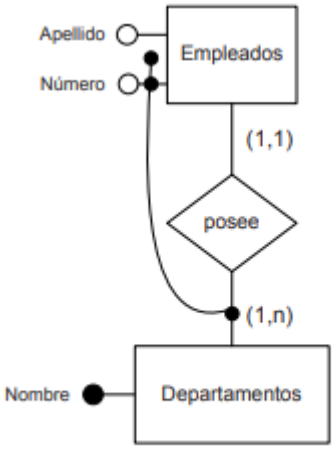
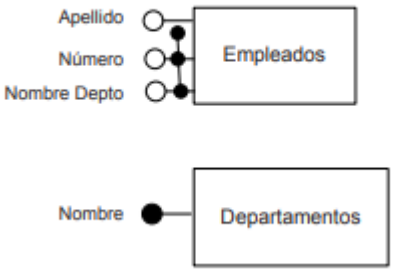


Figura 12.1 a)

Para realizar la conversión es necesario tomar el identificador de Departamento e incorporarlo como atributo de la entidad
Empleados. Quedando de la siguiente forma:



Selección de claves: primaria, candidata y secundara

El administrador de la BD debe decidir cuales de los identificadores se convierten en clave primaria o candidata. Este proceso se realiza al construir el modelo físico.

Cuando se genera el esquema física sobre el modelo relacional, se debe decidir el criterio de definición de clave primaria a partir de los identificadores reconocidos de cada entidad.

Si una entidad solo tiene definido un identificador, ese identificador es clave primaria de la tabla. Si la entidad tuviese definidos varios identificadores, la selección de la Clave Primaria (CP) debería realizarse del siguiente modo:

- Entre un identificador simple y uno compuesto, debería tomarse el simple, dado que así es más fácil de tratar y/o usar.
- Entre dos identificadores simples, se debe optar por aquel de menor tamaño físico
- Entre dos identificadores compuestos, se debería optar por aquel que tenga menor tamaño en bytes. De ese modo, al construir un índice utilizando un árbol B como estructura, es posible almacenar mayor cantidad de claves por nodo

Las consideraciones anteriores definen, en general, el criterio más adecuado para elegir la CP. El resto de los identificadores será definido como Clave Candidata (CC). Sin embargo, los SGBD ofrecen una alternativa que resulta ser la más conveniente. Para presentar esta alternativa se debe recordar la idea que motiva generar una CP.
Un archivo tiene asociada solamente una CP, en tanto que puede tener asociadas varias CC y/o secundarias.
El acceso físico al archivo de datos se logra a partir del uso de la CP.
El resto de las claves pueden utilizarse para generar índices secundarios, los cuales no referencian físicamente al archivo de datos, sino al índice primario.
Por lo tanto, la elección de la CP, que será utilizada para construir el índice primario, debe ser muy cuidadosa.

Cualquier CP elegida que sea directamente tratable por el usuario puede sufrir borrados o modificaciones. Estos posibles cambios influirán negativamente en

la performance final de la BD, dado a que impactarán en los índices primarios y/o secundarios

Los SGBD de la actualidad presentan una alternativa de tratamiento para las CP, a través del uso de tipo de dominio denominado autoincremental. Así, una tabla que tenga un atributo Autoincremental tiene definida una CP que es tratada por el SGBD en forma exclusiva. El usuario solamente tiene permitida la operación de consulta sobre la CP, es decir, no la puede generar, borrar ni modificar.

Nota: Según el profesor, es preferible intentar no utilizar autoincrementales sin mucha experiencia. Deberíamos como diseñadores poder con los datos que poseemos, seleccionar la clave primaria mas adecuada

Concepto de superclave

Una superclave es un conjunto de uno o más atributos que permiten identificar de forma única una entidad de un conjunto de entidades.

Conversión de entidades

El proceso de conversión de entidades definidas en el modelo lógico. En general, cada una de las entidades definidas se convierte en una tabla del modelo..

Conversión de relaciones. Cardinalidad

El proceso de conversión continúa con el análisis de las relaciones. Se deben analizar tres situaciones diferentes. Estos tres casos tienen que ver, básicamente, con la cardinalidad existente entre las relaciones definidas: muchos a muchos, uno a muchos y uno a uno.

- **Clave foránea: atributo/s de una tabla que en otra tabla es/son CP y que sirven para establecer un nexo entre ambas estructuras**

- Cobertura total
- Cobertura Parcial
- Relaciones recursivas
- Relaciones ternarias

Cardinalidad muchos a muchos

La resolución de las relaciones con cardinalidad muchos muchos resulta la más sencilla de implementar. La solución propuesta es independiente de la cardinalidad mínima definida, que puede ser en este caso obligatoria u opcional, pero la solución sigue siendo la misma.

Veamos el siguiente ejemplo:

Tablas hasta el momento:

Nota: Las subrayadas son la CP

Alumnos = (id_alumno, nombreyapellido, dni, nroalumno)

Materias= (id_materias, nombre, añocurso)

Cursa es una tabla:

cursa=(id_cursa,id_alumno,id_materia, resultadoobtenido, añoquecursa)

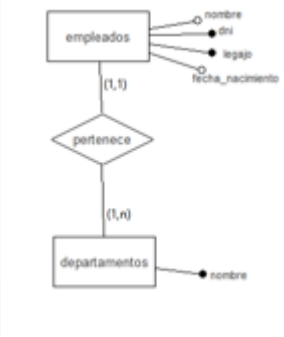
Cardinalidad uno a muchos

La solución para la cardinalidad uno a muchos tiene dos alternativas posibles: puede ocurrir que la relación se transforme o no en una tabla. La decisión deberá ser tomada en función de la cardinalidad mínima definida y de las decisiones que tome el administrador de la BD

Cardinalidad uno a muchos (cobertura total)

La relación no se convierte en una tabla

Veamos el siguiente ejemplo:



Quedando de la siguiente manera:

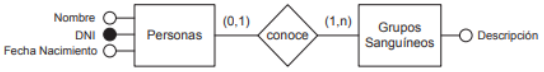
Empleados (id_empleado, dni, legajo, fecha_nacimiento, nombre, id_departamento)

Departamentos (id_departamento, nombre)

Cardinalidad uno a muchos (cobertura parcial del lado de muchos)

La relación no se convierte en una tabla y no modifica el análisis. Quedaría de la misma forma que el ejemplo anterior

Cardinalidad Uno a muchos con participación parcial del lado de uno



PERSONAS = (idpersona, nombre, dni, fecha_nacimiento, idgruposanguineo)
GRUPOS_SANGUINEOS = (idgruposanguineo, descripcion)

Opción 1: La relación no se convierte en tabla y no modifica el análisis

Desventaja: genera un posible valor nulo para id_grupo sanguíneo

Opción 2: La relación se convierte en una tabla

PERSONAS = (idpersona, nombre, dni, fecha_nacimiento)
GRUPOS_SANGUINEOS = (
idgruposanguineo, descripcion)
CONOCE = (
idpersona, idgruposanguineo)

La conclusión que se puede emitir en este punto es que la primera solución analizada genera una tabla menos, la información queda resumida y, por lo tanto, operar con ella es más sencillo. No obstante, se administran atributos con posibles valores nulos, y la situación es no deseada. En consecuencia, se opta por la segunda alternativa.

Uno a muchos con cobertura parcial de ambos lados.

El último caso posible de cardinalidad uno a muchos sucede cuando la cardinalidad es parcial de ambos lados. Aquí se puede proceder igual que en el caso anterior. Hay dos soluciones factibles: aquella que genera atributos con valores nulos (no recomendable) y la variante que lo evita.

Cardinalidad uno a uno

Cuando la cardinalidad es uno a uno con participación total de ambos lados, se debe generar una sola tabla que contenga los atributos de amas entidades.

Quedan para analizar los otros dos casos posibles: con participación parcial de un lado o de ambos.

El caso de cardinalidad parcial de ambos lados no es una situación muy común. Sin embargo, la solución aconsejable en este caso es generar una tabla por cada entidad y otra tabla que involucre la relación. Esta última tabla deberá incluir las CP de las entidades que relaciona.

Participación parcial de un lado. Veamos el ejemplo:

El elemento remito solo existe a partir de una factura

Facturas = (id_factura, tipo, número, fecha, montototal, nro_remito, fecharemito)

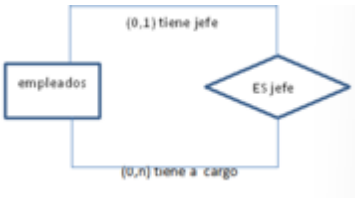
Casos particulares

Si bien lo planteado hasta el momento la conversión al modelo físico es resuelta sin mayores inconvenientes, resulta interesante comentar dos casos particulares: relaciones recursivas y las relaciones n-arias, particularmente las ternarias

Relaciones recursivas:

Se tratan igual que las binarias. La solución depende, nuevamente, de la cardinalidad definida en la
relación

Dado el siguiente ejemplo:



Opción 1: atributo nulo

Empleados=(id_empleado, nombre, dni, legajo, fecha_nacimiento, id_jefe)

Opcion 2: sin atributo nulo

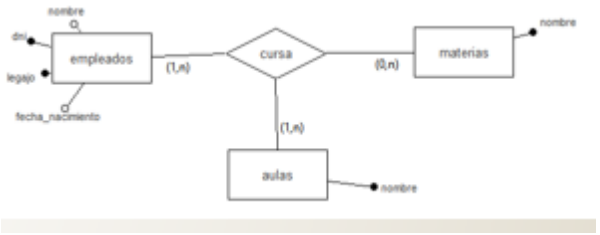
Empleados (id_empleado, nombre, dni, legajo, fecha_nacimiento)

Esjefe=(id_empleado, id_jefe)

Relaciones ternarias:

En general relaciones muchos a muchos

Cada entidad genera una tabla, idem la relación



Empleados (id_empleado, nombre, dni, legajo, fecha_nacimiento)
Aulas (id_aula, nombre)
Materias (id_materia, nombre)
Cursa (id_cursa, id_empleado, id_aula, id_materia)

Integridad referencial

La integridad referencial es una propiedad deseable de las BD relacionales. Esta propiedad asegura que un valor que aparece para un atributo en una tabla aparezca además en otra tabla para el mismo atributo.

La IR plantea restricciones entre tablas y sirve para mantener la consistencia entre las tuplas en dichas tablas.

Suponga que se generan las tablas:
FACTURAS = (idfactura, fecha, monto, idcliente)
CLIENTES = (idcliente, nombre, direccion)
El atributo idcliente en la tabla FACTURAS es una CF. Esta CF permite establecer IR entre la tabla FACTURAS y la tabla CLIENTES. Se debe notar que el atributo idcliente es CP de la tabla CLIENTES.
Para que exista IR entre dos tablas, necesariamente debe existir un atributo común. En general, el atributo común es CF en una tabla y CP en la otra tabla, aunque pueden presentarse excepciones a esta regla.
No es condición necesaria que entre dos tablas que tengan un atributo común esté definida la IR. Puede ocurrir que el diseñador de la BD no considere oportuna la definición de la IR entre dos tablas del modelo.

No obstante, en general, es deseable definir la IR, a fin de permitir que el SGBD controle determinadas situaciones.
En el ejemplo anterior, se debe definir la IR entre las tablas FACTURAS y CLIENTES. La IR puede plantearse para dos casos posibles: intento de borrado o intento de modificación.
Cada SGBD tiene escenarios de definición de IR diferentes, pero en general cuando se la define se puede optar por estas situaciones:

- Restringir la operación: es decir, si se intenta borrar o modificar una tupla que tiene IR con otra, la operación se restringe y no se puede llevar a cabo
- Realizar la operación “en cascada”: en este caso, si se intenta borrar o modificar una tupla sobre la tabla donde está definida la CP de la IR, la operación se realiza en cadena sobre todas las tuplas de la tabla que tiene definida la CF.
- Establecer la CF en nulo: si se borra o modifica el valor del atributo que es CP, sobre la CF se establece valor nulo. Esta opción no es utilizada ni está presente en todos los SGBD.
- No hacer nada: en este caso se le indica al SGBD que no es necesario controlar la IR. Esta opción es equivalente a no definir restricciones de IR. Este se recomienda no utilizarlo

Normalización

Concepto

La normalización es un mecanismo que permite que un conjunto de tablas (que integran una BD) cumplan una serie de propiedades deseables. Estas propiedades consisten en evitar:

- Redundancia de datos.
- Anomalías de actualización
- Pérdida de integridad de datos.

La ventaja de utilizar una BD normalizada consiste en disponer de tablas, cuyos datos serán para el usuario de fácil acceso y sencillo mantenimiento.

Algunos autores proponen normalizar el modelo de datos durante el proceso de diseño (conceptual, lógico y físico) de la BD.

El proceso de normalización es un proceso deseado, pero no estrictamente necesario. Durante dicho proceso, se toman decisiones que producen cambios en las tablas, los cuales pueden afectar los tiempos de acceso a los datos almacenados en esas tablas. Por lo tanto, es posible que, en determinadas situaciones, la solución obtenida no sea conveniente para la operatoria sobre la BD. Ante esas situaciones, la normalización puede quedar de lado y priorizarse la performance final de la BD. Si bien esta afirmación puede ser controvertida desde el punto de vista teórico, en la práctica profesional es habitual

Redundancia y anomalías de actualización

El objetivo principal del proceso de normalización es diseñar una BD que minimice la redundancia de información. Para esto, es necesario reagrupar los atributos de cada tabla del modelo. El proceso de diseño genera redundancia de información y en determinadas situaciones esta redundancia es necesaria.

Por el contrario, existen otros casos de redundancia, denominadas no deseadas, que generan anomalías de actualización cuando se opera sobre la BD. Estas anomalías se clasifican en tres grupos: anomalías de inserción, de borrado y de modificación de la BD. En este apartado, se presenta y describe mediante ejemplos cada una de ellas. En general, 1NF es muy restrictiva (se aplica siempre). El resto puede ser opcional, de hecho 2NF y 3NF normalmente se aplican siempre

Dependencias funcionales

Representa una restricción entre atributos de una tabla de la BD. Se dice que un atributo Y depende funcionalmente de un atributo X (denotado por la expresión $X \rightarrow Y$), cuando para un valor dado de X siempre se encuentra el mismo valor para el atributo Y. Se debe notar que X e Y pueden representar, además, un conjunto de atributos.

Más formalmente, dadas dos tuplas cualesquiera de una tabla t1 y t2, si t1[X] = t2[X], entonces t1[Y] = t2[Y].
Generalizando, el atributo X determina al atributo Y.

En una DF X →Y, al atributo X se lo denomina determinante y al atributo, consecuente.

X→ Y

- El atributo Y depende del atributo X
- El atributo X determina el valor único del valor Y
- El valor del atributo Y está determinado por el valor del atributo X

Todos ellos son sinónimos

Dependencia funcional parcial

Una DF X →Y se denomina parcial cuando, además, existe otra dependencia Z →Y, siendo Z un subconjunto de X.

Esta definición contradice la definición de conjunto mínimo de DF. En este caso, la generación de una DF parcial trae aparejada repetición de información y consecuentemente, las anomalías de actualización ya discutidas.

PEDIDOS = (idpedido, idproducto, descripciónproducto, fechapedido, cantidad)
que contiene los pedidos recibidos por un negocio de venta de productos por internet, se pueden observar las siguientes DF a partir de la CP definida:
Idpedido, idproducto → descripcionproducto
Idpedido, idproducto → fechapedido
Idpedido, idproducto → cantidad
Pero además, se puede observar que la descripción del producto vendido es un atributo que depende del código del producto. Asimismo, la fecha del pedido depende funcionalmente del código del pedido.

Entonces se pueden definir:
Idproducto → descripcionproducto
Idpedido → fechapedido

Se puede advertir que, al definir DF con determinante múltiple, es posible generar situaciones de DF parciales. Si el determinante es simple, no es

posible que una DF parcial exista.

Dependencia funcional transitiva

Una condición en la que A, b y C son atributos de una relación tales que A—> B y B ⟶ C entonces C depende transitivamente de A a través de B.

Dada la siguiente tabla:
ALUMNO = (
idalumno, nombre, direccion, idcarrera, nombre_carrera)
se establecen las siguientes DF a partir de la CP:
Idalumno → nombre
Idalumno → direccion
Idalumno → idcarrera
Idalumno → nombre_carrera
Además, se puede notar que el nombre de una carrera depende del código de la carrera en cuestión:
Idcarrera → nombre_carrera
En este ejemplo, X está representado por el atributo idalumno; Y, por el atributo nombre_carrera, y Z, por el atributo idcarrera. En el conjunto inicial definido, la DF Idalumno → nombre_carrera genera una DF transitiva.

Dependencia funcional de Boyce-Codd

La DF de Boyce-Codd está basada en DF que tienen en cuenta las CC de una tabla. Su definición formal es la siguiente:

> Una DF X⟶Y se denomina Dependencia Funcional de Boyce-Codd (DFBC) cuando X no es una CP (Clave primaria) o CC (Clave candidata), e Y es una CP o CC, o parte de ella

Formas normales

La normalización es una técnica que permite analizar el esquema físico de datos buscando la mejor representación de cada una de las tablas, evitando

la repetición de información y las anomalías de actualización que puedan surgir.

Para realizar el proceso de normalización, se parte de un esquema no normalizado, y se aplica un conjunto de reglas que permiten convertirlo en un esquema normalizado. Estas reglas son acumulativas, y el diseñador de la BD puede decidir hasta qué punto su modelo cumple lo planteado en cada una. Es decir, el proceso de normalización en la vida real, se pueden aplicar las reglas definidas hasta un determinado nivel.

Los niveles de normalización propuestos inicialmente fueron tres y se denominaron primera forma normal, segunda forma normal y tercera forma normal. La forma normal siguiente fue la planteada por Boyce Codd. Existen dos niveles más, la cuarta forma normal y la quinta forma normal, que fueron introducidos posteriormente.

Primera forma normal

La primera forma normal presenta una regla que ya fue definida cuando se explicó la conversión del esquema conceptual al esquema lógico. La primera forma normal plantea que todos los atributos que conforman una tabla deben ser monovalentes, es decir, la cardinalidad de atributos deber ser 0 o 1.

Los atributos polivalentes generan inconvenientes para la definición de una tabla. Como fue establecido, la solución de llevar el modelo a 1FN consistió en quitar el atributo polivalente, creando una nueva entidad.

Segunda forma normal

La segunda forma normal involucra el tratamiento de las DF parciales

Un modelo esta en Segunda forma normal (2FN) si está en 1FN (Primera forma normal) y, además, no existe en ninguna tabla del modelo una DF parcial.

Como se analizó anteriormente, las DF parciales generan anomalías de actualización. El proceso de normalización que permite llegar a un modelo en 2FN debe tratar las DF parciales, eliminándolas de dicho modelo.

La manera de tratar una DF parcial es quitar de la tabla el atributo o los atributos que generan esta DF, y situarlos en una nueva tabla. Esto significa que la DF planteada debe mantenerse en el modelo, pero en una ubicación donde ya no sea una DF parcial.

Tercera forma normal

Un modelo está en Tercera forma Normal si está en 2FN y , además, no existe ninguna tabla del modelo con una DF transitiva.

Las DF transitivas generan anomalías de actualización. En este caso, para el proceso de normalización se trabaja de manera similar, es decir, se deben quitar las DF transitivas de las tablas que las originan, colocando esos atributos en una nueva tabla. De este modo, se convierte a cada DF transitiva en solamente DF.

Forma normal de Boyce-Codd

La forma normal de Boyce-Codd fue propuesta con la posterioridad a la 3FN como una simplificación de esta, pero en definitiva generó una forma normal más restrictiva.

Un modelo está en Forma Normal de Boyce-Codd (FNBC O BCNF) si está en 3FN y, además, no existe en ninguna tabla del modelo una DF de BC.

Para generar un modelo BCNF, se debe garantizar que los determnantes de cada DF sean siempre CC.

La regla de normalización que plantea las DF de BC deben ser quitadas de la tabla donde se generan, y llevadas a una nueva tabla.

Dependencias Multivaluadas

La posible existencia de DM en una relación se debe a 1NF, que impide que una tupla tenga un conjunto de valores diferentes.

Así, si una tabla tiene dos atributos multivaluados, es necesario repetir cada valor de uno de los atributos con cada uno de los valores del otro. Así se garantiza la coherencia de la BD.

En general, una DM se da entre atributos A, B y C en una relación de modo que para cada valor de A hay un conjunto de valores de B y un conjunto de valores de C, sin embargo los conjuntos B y C no tienen nada que ver entre sí.

Se dice que A $\rightarrow\!\!\!\rightarrow$ B y que A $\rightarrow\!\!\!\rightarrow$ C

Se lee A multidetermina B y A multidetermina C
No es un problema el hecho que un atributo esté multideterminado

Cuarta forma normal

Un modelo se encuentra en BCNF y para toda relacion r del mismo solo existen dependencias multivaluadas triviales

Cuales DM son triviales?
A $\rightarrow\!\!\!\rightarrow$ B
A $\rightarrow\!\!\!\rightarrow$C
Las anteriores son triviales.

Quinta forma normal

Un proceso de normalización no finaliza con la 4FN. Las formas normales posteriores a la 4FN representan situaciones muy particulares y específicas.

Un modelo está en 5FN si está en 4FN y no existen relaciones con dependencias de combinación.
Una dependencia de combinación es una propiedad de la descomposición que garantiza que no se generen tuplas espurias al volver a combinar las relaciones mediante una operación del álgebra relacional.

Lenguajes de procesamiento de datos. Álgebra y cálculos

Lenguaje de procesamiento de datos

Los lenguajes de procesamiento de datos permiten operar con la información contenida en una BD. A partir de los requerimientos del usuario, el diseñador de la BD construye un modelo conceptual, el cual, a través de sucesivas transformaciones, se convierte en un esquema lógico primeramente, para luego llegar a un esquema físico. Este esquema físico se impacta sobre un SGBD y de esta manera se define el modelo de datos para el problema en cuestión.

Lenguajes de consulta: utilizados para operar con la BD.

- Procedurales: (instrucciones para realizar secuencia de operaciones) (que y cómo)
- No procedurales: (solicita directamente la información deseada) (que).

A priori, para realizar consultas, los lenguajes no procedurales parecen más fáciles de utilizar, pero la operatoria se complica rápidamente a medida que se intentan plantear consultas más complejas. Los lenguajes procedurales permiten al analista generar la operación e incidir
en cómo resolver las consultas.

Estudios estadísticos realizados sobre BD indican que aproximadamente 80% de las operaciones sobre una BD son operaciones de consulta. Por este motivo, en primera instancia se detallan las operaciones de consulta para cada uno de los lenguajes. Luego, solamente para álgebra relacional se describen, además, las otras operaciones (altas, bajas y modificaciones).

Álgebra Relacional

La manipulación de datos de una BD implantada sobre el modelo relacional mantiene al álgebra relacional como componente principal. Básicamente, el

Álgebra Relacional (AR) representa un conjunto de operadores que toman las tablas (relaciones) como operandos, y regresan otra tabla como resultado.

Operadores básicos

Los operadores básicos de AR son seis y se dividen en dos tipos:

1. Unarios: operan sobre una tabla o relación
2. Binarios: operan sobre dos tablas o relaciones

Los operadores unarios son tres: selección, proyección y renombre; y los binarios también son 3: producto cartesiano, unión y diferencia.

Selección

La selección es una operación que permite filtrar tuplas de una tabla.

El formato genérico de una selección es:

σ p (tabla)

donde σ indica la operación de selección, tabla es una relación definida en la BD y P es un predicado o condición lógica que deben cumplir las tuplas de la tabla, para ser presentadas como resultado.

Proyección

La proyección es la operación que permite presentar en el resultado algunos atributos de una tabla. El formato genérico de una selección es:

π l (tabla)

donde π indica la operación de proyección, tabla es una relación definida en la BD y L es una lista, separada por comas, de atributos definidos en tabla.

Producto cartesiano

El producto cartesiano es equivalente al producto cartesiano entre conjuntos. Se aplica a las dos tablas o relaciones de la BD, y vincula cada

tupla de una relación con cada una de las tuplas de la otra relación. El formato genérico de un producto cartesiano es:

tabla1 × tabla2

donde × indica la operación de producto cartesiano, de la misma forma que se indica esta operación de producto en matemáticas, y tabla1 y tabla2 son dos relaciones definidas en la BD.

Ejemplo:

Nombre	Id_localidad
Juan	1
Pedro	2
Hector	1

Id localidad	nombre
1	La Plata
2	junin

Aso.nombre	Aso.idloc	Loc.idloc	Loc.nombre
Juan	1		La plata
Juan	1	2	Junin
Pedro	2	1	La Plata
Pedro	2	2	Junin
Hector	1	1	La Plata
Hector	1	2	Junin

Renombre

El renombre es la operación que permite utilizar la misma tabla dos veces en una misma consulta. El formato genérico es:

ρ nombre_nuevo (tabla)

donde ρ indica la operación de renombre, tabla es una relación definida en la BD y nombre_nuevo es el alias (nuevo nombre) que tiene la tabla.

Unión

El operando unión es equivalente a la unión matemática de conjuntos. Se aplica a dos tablas o relaciones de la BD, generando una nueva tabla cuyo contenido es el contenido de cada una de las tuplas de las tablas originales involucradas. El formato genérico de una unión es:

tabla1 U tabla2

Donde U indica la operación de unión, de la misma forma que se indica esta operación en matemáticas, y tabla1 y tabla 2 son dos relaciones definidas en la BD.

Cuando se utiliza el operando de unión en AR, debe tenerse en cuenta que los dos conjuntos a unir deben ser unión-compatibles. Esto es, unir dos conjuntos que tengan la misma cantidad de atributos, y además el i-ésimo atributo de la primera tabla y el i-ésimo atributo de la segunda tabla deben tener el mismo dominio (i: 1..n). Caso contrario, el resultado obtenido no representa ninguna información útil.

Diferencia

El operando diferencia es equivalente a la diferencia matemática de conjuntos. Se aplica a dos tablas o relaciones de la BD, generando una nueva tabla cuyo contenido es el contenido de cada una de las tuplas de la primera tabla que no están en la segunda. El formato genérico de una diferencia es:

tabla1 – tabla2

Donde - indica la operación de diferencia, de la misma forma que se indica esta operación en matemáticas, y tabla1 y tabla2 son dos relaciones definidas en la BD.

Cuando se utiliza el operando de diferencia en AR, debe tenerse en cuenta que las dos tablas deben ser unión-compatibles.

Operadores adicionales

Además de los seis operadores básicos definidos, existen otros operadores que otorgan mayor expresividad al AR, facilitando en muchos casos la construcción de las operaciones de consulta.

Producto natural

La definición del operando producto natural surge a partir de la necesidad de realizar una operación de selección posterior a un producto cartesiano de dos tablas, para dar como resultado las tuplas con sentido. Además y por definición, el producto cartesiano relaciona cada tupla de la primera tabla con cada una de las tuplas de la segunda tabla. Así, si la tabla tiene 1000 registros y la segunda, 100, el producto cartesiano genera una tabla intermedia de 100000 elementos. Luego, la selección deja solamente las tuplas deseadas.

El producto natural directamente reúne las tuplas de la primera tabla que se relacionan con la segunda tabla, descartando las tuplas no relacionadas. Esto conlleva una gran ventaja en cuanto a performance, dado que no se
generan tuplas que luego se descartan, sino que solo se generan las tuplas necesarias.

El operador que denota producto natural es |×|

tabla1 |×| tabla2

Se debe notar que entre las tablas sobre las que se realiza el producto natural debe existir un atributo común, o sea, una de las tablas debe tener definida una CF de la otra. En su defecto, de no existir un atributo común, el producto natural y el producto cartesiano actúan de la misma forma.

Intersección

El operando de intersección es equivalente a la intersección matemática de conjuntos. Se aplica a dos tablas o relaciones de la BD, generando una nueva tabla con las tuplas comunes a ambas tablas. El formato genérico de una intersección es:

tabla1 ∩ tabla2

donde ∩ indica la operación de intersección, de la misma forma que se indica esta operación en matemáticas, y tabla1 y tabla2 son dos relaciones definidas en la BD

Nuevamente, cuando se utiliza el operando de intersección en AR, debe tenerse en cuenta que las dos tablas deben ser unión-compatibles; en su defecto, el resultado será una tabla (relación) vacía.
El operador de intersección se define como un operador adicional debido a que es posible obtener la intersección de conjuntos a partir de su diferencia, o de combinar la diferencia con la unión:
$A \cap B = A - (A - B)$
$A \cap B = A \cup B - ((A - B) \cup (B - A))$

Asignación

El operador asignación no aporta una funcionalidad extra al AR, sino que solamente tiene como objetivo otorgar mayor legibilidad a las consultas. Con el operador de asignación se puede generar una subconsulta y

volcar su resultado en una variable temporal de tabla, la cual puede ser utilizada posteriormente. El operador asignación es ←.

División

Dadas dos tablas R1 y R2, R1 ÷ R2 tiene como resultado los valores de atributos de R1, que se relacionan con todas las tuplas de R2.
R1 % R2 si, y solo si, el esquema de R2 está incluido en el esquema de R1, es decir que todos los atributos de R2 son atributos de R1.
El esquema o "estructura" de la tabla resultante es el esquema de R1 − el esquema de R2.

Actualizaciones utilizando AR

El lenguaje original de AR permite realizar, además, las operaciones básicas de actualización: alta, baja y modificación de datos contenidos en la BD.

Altas

La operación de alta permite agregar una nueva tupla a una tabla existente. Para ello, se debe realizar una operación de unión entre la tabla y una tupla escrita de manera literal

Alumnos ← Alumnos U {"Delia", "30777987", 2, 1}

Bajas

La operación de baja permite quitar una tupla de una tabla. Para ello, se debe realizar una operación de diferencia entre la tabla y una tupla escrita de manera literal.

Ejemplo 14: Se debe quitar de la tabla alumnos al alumno López.
Alumnos ← Alumnos − σ nombre = 'Lopez' (alumnos)

Modificaciones

A diferencia de las dos operaciones de actualización anteriores, que se resolvían a partir de la utilización de operadores básicos, la operación de modificación necesita representar el cambio con un operador, δ. El formato genérico de la operación es:

δ atributo = valor_nuevo (tabla)

donde δ es el operador que indica modificación, tabla es la tabla que se modifica, atributo es el atributo de la tabla a modificar y valor_nuevo se asigna como nuevo contenido del atributo.

Ejemplo 15: Se debe modificar el número de DNI de Pettorutti por 34567231.

δdni = "34567231" (σnombre = 'Pettorutti' (alumnos))

Nótese que, previo a realizar la modificación, se debe seleccionar la tupla sobre la que se desea hacer el cambio. Si no se hiciera de esa forma, todos los DNI de todos los alumnos de la tabla se modificarían

Definición formal

El AR puede ser definido formalmente de la siguiente manera. Una expresión básica en AR consta de:

- Una relación o tabla de una BD
- Una relación o tabla constante (ejemplo de la inserción, un conjunto de tuplas expresada literalmente cada una de ellas)

Una expresión (consulta) general *E* en AR se construye a partir de subexpresiones (subconsultas) E1, E2, ..., En

Las expresiones posibles son:

- σ p (E1) P Predicado con atributos en E1
- π s (E1) S Lista de atributos de E1.
- ρ z (E1) Z Nuevo nombre de E1.
- E1 × E2.

- E1 U E2.
- E1 – E2.

SQL

El origen de SQL está ligado a las bases de datos relacionales. Cómo se indico anteriormente, en 1970 E.F. Codd propone el modelo de datos relacional y asociado a éste, lenguaje de consulta procedural (algebra relacional) y no procedural (cálculos). Estos lenguajes propuestos por Codd fueron la base para la creación de SQL (Structured Query Languaje, Lenguaje de Consultas Estructurado)

SQL termina siendo una de las principales razones del éxito de las BD relacionales. Esto se debió a que la mayoría de los fabricantes de DBMS optaron por SQL como el lenguaje de trabajo, generando de esta forma un estándar respetado en general, por la mayor parte de los productos de mercado. Así, una consulta SQL puede migrar de producto sin necesitar mayores cambios para ser ejecutada.

Un poco de historia

En 1986 SQL es estandarizado por el ANSI (American National Standart Institute, la organización de estándares de Estados Unidos), dando lugar a la primera versión estándar de este lenguaje, el “SQL-86” o “SQL1”. Al año siguiente este estándar fue adopto por la ISO (International Standart Organization).

En 1992 se amplía la definición del estándar para cubrir mayores necesidades de los desarrolladores. Surge de esta forma el SQL-92 o el SQL2. Esta versión será la base de los temas tratados. Si bien hay versiones posteriores, para la finalidad presente material los parámetros definidos en ese estándar resultan suficientes.

El estándar tuvo varias modificaciones posteriores: en 1999 se creó el SQL2000, al que se incorporaron expresiones regulares, consultas

recursivas y características orientadas a objetos. En 2003, surge SQL3 que agrega características de XML y, posteriormente en 2006 ISO define ISO/IEC 9075-14:2006 con características que acercan a SQL al mundo W3C.

En las dos secciones subsiguientes se presentan el DDL (lenguaje de definición de datos) y el DML (lenguaje de manipulación de datos).

Lenguaje de definición de datos (DDL)

El modelo de datos relacional utiliza el concepto de relación, tupla y atributo, en tanto en SQL los términos corresponden a tabla, fila y columna. En esta materia tabla-relación, tupla-fila y atributo-columna, se utiliza indistintamente.

Para administrar un modelo físico de datos, SQL presenta tres cláusulas básicas: CREATE, DROP y ALTER. Los mismos se corresponden con crear, borrar o modificar el esquema existente. No es objetivo profundizar detalladamente en la definición del DDL de SQL2. En general, los productos de mercado ofrecen una interfaz amigable que permite generar BD, sus tablas, índices, etc. Además, existen herramientas que asisten al modelado de BD que, una vez terminado dicho modelo, permiten generar de manera automática el código SQL que genera la BD sobre el DBMS.

Crear o borrar una BD

Para generar una BD la sentencia SQL es:

```
CREATE DATABASE nombre_BD;
```

Esta sentencia genera una BD cuyo nombre se indica

Para eliminar una BD completa la sentencia es:

```
DROP DATABASE nombre_BD;
```

La cual borra la BD: esto incluye las tablas y los contenidos en ellas

Operar contra Tablas de BD

Para generar una tabla en una BD, la instrucción SQL es CREATE TABLE. El siguiente ejemplo presenta la creación de una tabla denominada empresas:

```
CREATE TABLE empresas (
idempresa INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
empresa VARCHAR(100) NOT NULL,
abreviatura VARCHAR(10) NULL,
cuit VARCHAR(13) NULL,
direccion VARCHAR(100) NULL,
observaciones TEXT NULL,
PRIMARY KEY(idempresa),
UNIQUE INDEX empresas_index19108(empresa));
```

Para definir una tabla es necesario indicar cada uno de los atributos que la componen. En este caso algunos de los atributos son:

- Idempresa: con dominio Integer sin signo, el atributo no puede ser nulo y el dato es autoincremental (tal como se definiera oportunamente en el capítulo de modelado físico).
- Empresa: con dominio String y con longitud máxima 100, no puede ser nulo.
- Obsevaciones: dominio texto (a diferencia del String que tiene longitud máxima, el tipo de dato texto no está limitado)

La sintaxis utilizada, si bien es ANSI, tiene similitud con la utilizada por el DBMS MySQL. Los valores posibles para los dominios de los atributos están ligados directamente con los productos comerciales. Asimismo ocurre con la definición de los atributos autoincrementales.
Por último, en el ejemplo anterior, se encuentran definidos dos índices para la tabla Empresas. Estos índices se definen asociados a la clave primaria y clave candidata respectivamente.

Eliminar una tabla del modelo

```
DROP TABLE nombre_tabla;
```

Modificar una tabla del modelo

```
ALTER TABLE nombre_tabla (...);
```

Esta sentencia debe indicar, además, que tipo de modificación se desea realizar sobre la tabla. Así entre paréntesis se pueden agregar, modificar o borrar atributos, índices o restricciones de integridad.

A modo de ejemplo, la siguiente sentencia SQL agrega el atributo razón social a la tabla empresa generada anteriormente, se quita el atributo cuit y se modifica el dominio del atributo dirección por un string de longitud 50.

```
ALTER TABLE empresas (
Add column razon_social VARCHAR(100) NOT NULL,
Drop column cuit,
Alter column direccion VARCHAR(50) NULL);
```

Lenguaje de Manipulación de datos (DML)

Para los ejemplos en este capítulo se utilizaran las siguientes tablas:

- Asociados=(idsocio, nombre, dirección, teléfono, sexo, estadocivil, fechanacimiento, idlocalidad)
- Deportes=(iddeporte, nombre, monto_cuota, idsede)
- Practica= (idsocio, iddeporte)
- Localidad= (idlocalidad, nombre)
- Sedes= (idsede, nombre, dirección, idlocalidad)

Estructura básica

La estructura básica de una consulta SQL tiene el siguiente formato

SELECT lista_de_atributos

FROM lista_de_tablas

WHERE predicado

Donde lista_de_atributos indica los nombres de los atributos que serán presentados en el resultado. Estos atributos deben estar contenidos en las tablas indicadas en la consulta; sobre las tablas indicadas en la lista se realiza un producto cartesiano. Por último, predicado indica que condición deben cumplir las tuplas de las tablas para estar en el resultado final de la consulta.

La analogía entre SQL y AR es importante. AR representa la base teórica de SQL, por lo tanto las consultas expresadas en ambos lenguajes es similar en aspectos semánticos.

Los atributos utilizados en el SELECT de una consulta SQL pueden tener asociados operaciones válidas para sus dominios.

Sobre los atributos definidos en un SELECT se pueden realizar cualquiera de las operaciones básicas definidas para su domino. Cada DBMS en particular define su mapa de operaciones.

Hasta el momento, se presentaron consultas que solamente involucraron una tabla del modelo de datos.

Cláusula WHERE

Se puede notar que la sentencia WHERE no es obligatoria.

En determinadas circunstancias puede no ser necesario que el mismo resultado aparezca varias veces. Para eliminar tuplas repetidas se debe utilizar la cláusula DISTINCT.

SQL define, además, el operador ALL. Este operador muestra todas las tuplas, aún las repetidas. En caso de no poner explícitamente el operador DISTINCT, SQL asume el operador ALL, por este motivo en general nunca se explicita.

Ejemplo 7: mostrar los asociados varones casados.

```
SELECT nombre
```

```
FROM asociados
WHERE sexo = “masculino” AND estadocivil = “casado”
```

Operador BETWEEN: mostrar los deportes cuya cuota esté entre 400 y 600 pesos mensuales

```
SELECT  nombre
FROM   deportes
WHERE montocuota BETWEEN 400 and 600
```

Cláusula FROM

Para realizar un producto cartesiano, basta con poner en la cláusula FROM dos o más tablas.

Ejemplo 7: **Producto Cartesiano**: mostrar para cada asociado su nombre y la localidad de residencia

```
SELECT  asociado.nombre, localidad.nombre
FROM   asociado, localidad
WHERE asociado.idlocalidad = localidad.idlocalidad
```

Ejemplo 8: **Producto Natural** mostrar para cada asociado su nombre y la localidad de residencia

```
SELECT  asociado.nombre, localidad.nombre
FROM   asociado  INNER JOIN  localidad
ON (asociado.idlocalidad = localidad.idlocalidad)
```

Operación de Renombre

Si bien no es necesario renombrar las tablas, es frecuente cambiar el nombre de la relación por otro más corto, con el propósito de escribir más rápidamente la consulta. De la misma forma que se puede renombrar una tabla en la cláusula FROM, es posible renombrar un atributo en al cláusula SELECT.

Ejemplo 9: mostrar todos los deportes salvo el mas costoso.

```
SELECT  DISTINCT deporte.nombre
FROM deportes, deporte dep
WHERE dep. Montocuota > deporte.montocuota
```

Ejemplo 10: presentar todos los asociados con nombre, dirección y idlocalidad. El listado debe figurar con la leyenda DIRECCIÓN LEGAL.

```
SELECT   nombre, direccion AS DIRECCION LEGAL, idlocalid
FROM asociados
```

Operaciones sobre cadenas

Una de las características interesantes y que dotan a SQL de gran potencia en la generación de consultas que relacionan Strings, resulta del uso del operador de comparación LIKE. Cuando una cadena se compara por igualdad (=) el resultado será verdadero si ambas cadenas son iguales, falso en caso contrario.

El operador LIKE se conjuga con el carácter reservado %. Se compara el atributo nombre de la tabla alumnos contra el string que empieza con P y continúa con cualquier substring (ese es el comportamiento del carácter %, el cual considera válida a partir de la posición donde aparece, cualquier cadena de caracteres, inclusive la cadena vacía). Entonces, cualquier tupla cuyo atributo nombre comience con P será presentado en el resultado final. Existe además otro carácter reservado, '_', el guion bajo sustituye solo el carácter del lugar donde aparece.

Ejemplos:

- ´"Alfa%": cualquier cadena que empiece con Alfa

- “%Alfa”: cualquier cadena que termine con Alfa
- ´“%casa%”: cualquier cadena que tenga casa en su interior
- ´“_ _ _”: cualquier cadena con tres caracteres
- ´“_ _ _%”: cualquier cadena con al menos tres caracteres.

Operadores que permiten ORDENAR las tuplas

Order by atributo: Especifica cuales tuplas serán ordenadas

Ejemplo 13: presentar todos los asociados ordenados por nombre.

```
SELECT  nombre
FROM   asociadosORDER BY nombre
```

Desc, asc: por defecto ascendente, se puede especificar descendente.

Operadores de Conjuntos

En SQL92 la UNION se resuelve de manera tal que las tuplas duplicadas en ambas subconsultas solo quedan una vez en el resultado final. Para obtener las tuplas duplicadas se puede utilizar la cláusula UNION ALL. La cláusula definida por SQL92 para la diferencia de conjuntos es EXCEPT, en tanto que para la operación de intersección se utiliza la cláusula INTERSECT.

Unión: agrupa las tuplas resultantes de dos subconsultas.

Union all:conserva duplicados

Ejemplo 15: asociados que practican futbol o voley

Intersección:

Ejemplo 16: asociados que practican futbol y voley

Diferencia: (except)

Ejemplo 17: asociados que practican futbol y no voley

```
(SELECT  a.nombre
FROM   asociados a INNER JOIN practica p
```

```
ON a.idsocio = p.idsocio
INNER JOIN deportes d ON p.iddeporte = d.iddeporte
WHERE d.nombre = “FUTBOL” )
UNION  /  UNION ALL  /   INTERSECT  / EXCEPT
(SELECT  a.nombre
FROM   asociados a INNER JOIN practica p ON a.idsocio =
INNER JOIN deportes d ON p.iddeporte = d.iddeporte
WHERE d.nombre = “VOLEY” )
```

Consultas con funciones de agregación

Las funciones de agregación son funciones que operan sobre un conjunto de tuplas de entrada y producen un único valor de salida. SQL define cinco funciones de agregación:

- AVG: retorna como resultado el promedio del atributo indicado para todas las tuplas del conjunto
- COUNT: retorna como resultado la cantidad de tuplas involucradas en el conjunto de entrada
- MAX: retorna como resultado el valor mas grande dentro del conjunto de tuplas para el atributo indicado
- MIN: retorna como resultado el valor más pequeño dentro del conjunto de tuplas para el atributo indicado
- SUM: retorna como resultado la suma del valor del atributo indicado para todas las tuplas del conjunto.

Limitaciones:

- NO puede aparecen en el where

Funciones de Agrupación

En muchos casos resulta necesario agrupar las tuplas de una consulta por algún criterio, aplicando una función de agregación sobre cada grupo. Para estas situaciones SQL prevé la cláusula **GROUP BY.**

Ejemplo 23: obtener la cantidad de asociados que practica cada deporte

```
SELECT COUNT ( * )
FROM  asociados a INNER JOIN practica p ON ( a.idsocio
INNER JOIN deportes d ON (d.iddeporte = p.iddeporte )
GROUP BY d.nombre
```

Cuando se genera un agrupamiento por un criterio (atributo) es posible proyectar en el SELECT el atributo por el cual se genera el grupo. Hay consultas que requieren obtener resultados para grupos que satisfagan una determinada condición. SQL prevé para esos casos la cláusula HAVING, la cual actúa como filtro de grupos. Dentro de la cláusula HAVING es posible indicar una condición que debe cumplir el grupo para ser tenido en cuenta.

Ejemplo 27: mostras todos los deportes y la cantidad de asociados que lo practican siempre y cuando para el deporte se abonen en total mas de 100000 pesos

```
SELECT d.nombre, COUNT ( * )
FROM  asociados a INNER JOIN practica p ON ( a.idsocio =
INNER JOIN deportes d ON (d.iddeporte = p.iddeporte )
GROUP BY d.nombre
HAVING SUM( d.montocuota ) > 100000
```

Se puede observar que dentro de la cláusula HAVING es posible utilizar, nuevamente, una función de agregación. En el HAVING se controla la validez de un grupo de tuplas, por lo tanto es posible utilizar una función de agregación.

Consultas con subconsultas

Una consulta con subconsultas, también conocidas como subconsultas anidadas, consiste en ubicar una consulta SQL dentro de otra. Las subconsultas se pueden utilizar en varias situaciones: 1) comprobar la pertenencia o no a un conjunto, 2) analizar la cardinalidad de un conjunto, 3) analizar la existencia o no de elementos en un conjunto.

Operaciones de pertenencia a conjuntos

SQL define el operador IN para comprobar si un elemento es parte o no de un conjunto.

Ejemplo 29: mostrar aquellos asociados que practiquen Basquet.

```
SELECT nombre
FROM asociados
WHERE idsocio IN ( SELECT idsocio
from practica p INNER JOIN deportes d
ON (d.iddeporte = p.idpractica )
WHERE d.nombre = "basquet" )
```

Ejemplo 30: mostrar aquellos asociados que practiquen no Basquet.

```
SELECT nombre
FROM asociados
WHERE idsocio NOT IN ( SELECT idsocio
from practica p INNER JOIN deportes d
ON (d.iddeporte = p.idpractica )
WHERE d.nombre = "basquet" )
```

Operaciones de comparación contra conjuntos

Además del operador IN, se pueden utilizar operadores de comparación de elementos simples sobre conjuntos. Estos operadores, de funcionamiento similar a los operadores relacionales (=, <, >, >=, <= o <>), están combinados con palabras reservadas.

- =SOME, significa igual a alguno. El resultado será verdadero si el elemento simple resulta igual a alguno de los elementos contenidos en el conjunto, falso en caso contrario
- >ALL, significa mayor que todos. El resultado será verdadero si el elemento simple resulta mayor que todos los elementos del conjunto
- <= SOME, significa menor o igual que alguno. El resultado será verdadero si el elemento simple resulta menor o igual que alguno de los elementos del conjunto

Cada uno de ellos se puede combinar con los diferentes operadores relacionales, dependiendo el resultado deseado.

Cláusula EXISTS

La cláusula EXISTS se utiliza para comprobar si una subconsulta generó o no alguna tupla como respuesta. El resultado de la cláusula EXIST es verdadero si la subconsulta tiene al menos una tupla, y falso en caso contrario. No es importante para la subconsulta el o los atributos proyectados, debido a que solamente se chequea la existencia o no de tuplas.

Ejemplo 33: mostrar los asociados que practican futbol y voley (una tercera variante)

```
SELECT a.nombre
FROM  asociados a
WHERE EXISTS ( SELECT *
FROM practica p INNER JOIN deportes d ON (d.iddeporte
WHERE d.nombre = “Futbol” and p.idsocio = a.idsocio)
AND
EXISTS ( SELECT *
practica p INNER JOIN deportes d ON (d.iddeporte = p.i
WHERE d.nombre = “voley” and p.idsocio = a.idsocio)
```

Ejemplo 34: obtener los asociados que practiquen todos los deportes.

```
SELECT a.nombre
FROM asociados a
WHERE NOT EXISTS ( SELECT *
            FROM deportes d
            WHERE NOT EXISTS ( SELECT *
                                FROM practica p
                        WHERE  p.idsocio = a.idsocio A
                           p.iddeporte = d.iddeporte)
```

Es complejo resolver una consulta que involucre “.. todos los deportes..:”. Resulta más sencillo resolver esta consulta utilizando el razonamiento de la contra-recíproca. De este modo, se presentarán los asociados tal que para ellos no exista ningún deporte que no practiquen. Se puede notar que esta expresión es equivalente a “que practiquen todos”.

Operaciones con Valores Nulos

En SQL se debe tener especial cuidado cuando un atributo puede contener valores nulos. En general, cuando se resuelve un algoritmo y el programador debe preguntar si una variable tiene valor nulo genera una expresión del tipo:

Variable = ‘’

Al utilizar esta expresión, comillas sin ningún carácter en su interior se asume valor nulo. Sin embargo este tipo de expresiones no tienen un comportamiento similar cuando se trata de consultas SQL. Cuando un dominio de un atributo puede contener valores nulos no puede tratarse de esta forma. Los dominios con posibles valores nulos incorporan al conjunto de valores posibles el valor NULL. Este valor se almacena por defecto si el usuario no define otro. Entonces, una consulta que pregunta por un string nulo (‘’) no una respuesta satisfactoria. Para estas situaciones se define un nuevo operador IS NULL o su negación IS NOT NULL.

Ejemplo 28: Mostrar aquellos deportes que no tengan valor ingresado en su cuota

```
SELECT nombre
FROM  deportes
WHERE montocuota IS NOT NULL
```

Productos Naturales

Hasta el momento, en SQL se presentó al producto cartesiano como única opción para ligar tablas.
Por las mismas cuestiones de performance planteadas es necesario que SQL presente una alternativa al producto cartesiano, el producto

natural. Las sentencias disponibles al efecto son: INNER JOIN, LEFT JOIN, RIGTH JOIN, FULL JOIN. A continuación se detallan y muestran ejemplos de cada una de ellas.

suponer:
Tabla1 LEFT JOIN Tabla2
En primera instancia se reúne cada tupla de la tabla 1 con su correspondiente tupla en la tabla2, similar al INNER JOIN. Luego, si alguna tupla de la tabla 1 no se reúne con una tupla de la tabla 2, igualmente aparece en el resultado, con los atributos de la tabla 2 con valor nulo.

RIGTH JOIN genera, primero, un INNER JOIN para luego completar con las tuplas de la tabla 2 que no tienen correspondencia en la tabla

1. Se puede notar, entonces, que las siguientes expresiones resultan equivalentes:
   Tabla1 LEFT JOIN tabla2
   Tabla2 RIGHT JOIN tabla1

Por ultimo, FULL JOIN, genera el equivalente a un INNER JOIN, LEFT JOIN y RIGTH JOIN en conjunto. Es decir, reúne las tuplas con sentido, luego agrega las tuplas de las tabla 1 sin correspondencia en la tabla 2 y, por último, reúne las tuplas de tabla2 sin correspondencia en tabla1.

Otras operaciones: Insertar, Borrar y Modificar

Insertar tuplas a la BD

La cláusula utilizada para agregar tuplas a una tabla es la cláusula INSERT INTO.

Ejemplo 38: agregar un nuevo asociado a la tabla

```
INSERT INTO asociados
( "Juan Perez", "Balbin 143",
"221-5133747","masculino", "casado", 22-10-1994, 4)
```

El atributo idsocio está definido como autoincremental. Por este motivo, será el DBMS el encargado de asignarle el valor correspondiente. Si el usuario intentara asignar un valor a un

atributo definido como autoincremental, la operación de inserción falla.

La cláusula INSERT INTO genera una sola tupla cada vez que es ejecutada.

Borrar tuplas de la BD

Para borrar una tupla o un conjunto de tuplas de una tabla se utiliza la cláusula DELETE FROM.

Ejemplo 39: quitar de la tabla el deporte bochas

```
DELETE FROM deportes
WHERE nombre = "Bochas"
```

Si la cláusula definida fuera:
DELETE FROM deportes
Se borran todas las tuplas definidas en deportes.

Por último, debe notarse que nuevamente la integridad referencial tendrá especial énfasis sobre la cláusula DELETE. Esto es, solamente pueden ser borradas de la tabla aquellas tuplas que cumplan las condiciones de integridad referencial definidas.

Modificar tuplas de la BD

Por último, la cláusula UPDATE ... SET se utiliza para modificar el contenido de uno o varios atributos de una tabla.

Ejemplo 41: incrementar la cuota de cada deporte en un 20%

```
UPDATE deportes
SET monto cuota = monto cuota * 1,2
```

Ejemplo 42: incrementar la cuota de hockey en un 30%

```
UPDATE deportes
SET montocuota = montocuota * 1,3
WHERE nombre = "hockey"
```

Se debe tener en cuenta que si varias tuplas cumplen la condición estipulada en el WHERE, todas ellas serán modificadas.

Optimización de Consultas

Análisis de procesamiento de consultas

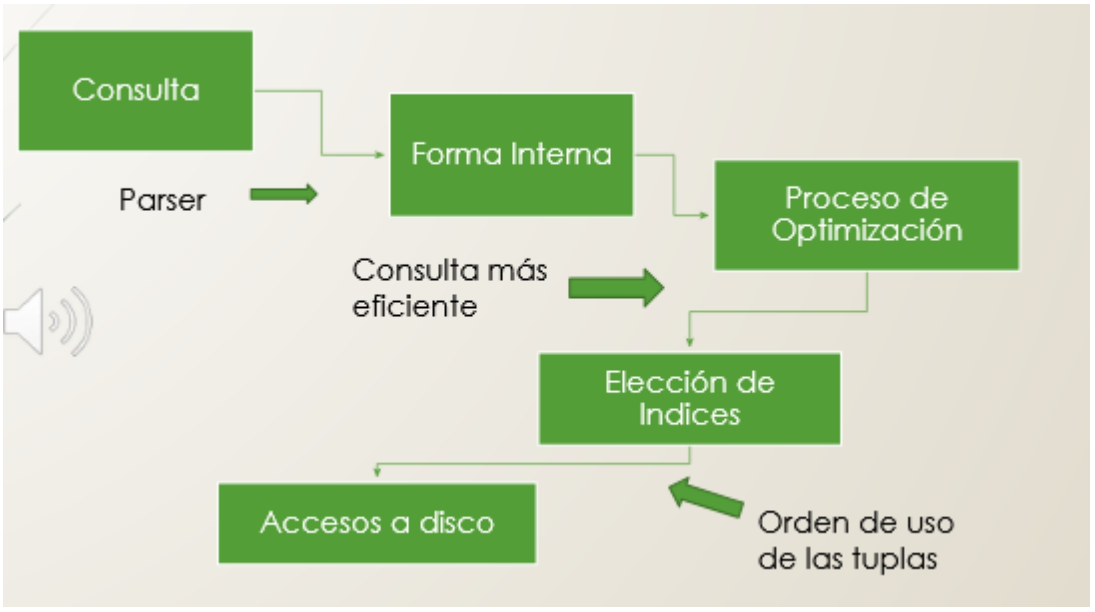
Los SGBD presentan en general un optimizador de consultas. Se denomina optimizador de consultas a un proceso del gestor de BD, encargado de encontrar una consulta equivalente a la generada por el usuario, que sea de óptima en términos de performance para obtener el resultado deseado.

Componentes del “costo” de ejecución de una consulta:

- Costo de acceso a almacenamiento secundario → acceder al bloque de datos que reside en disco.
- Costo de cómputo → Costo de realizar operaciones sobre memoria RAM
- Costo de comunicación → Costo de enviar la consulta y los resultados (si es un Sistema Distribuido)

El proceso de optimización, como se presenta en la Figura 16.1, comienza con la consulta generada por el usuario; el optimizador de consulta la analiza y la compara contra el estado actual de la BD, y genera una consulta equivalente en cuanto a la respuesta que se obtendrá de ella, pero más eficiente en términos de procesamiento.

Figura 16.1



Analizando con más detalle el comportamiento del optimizador de consulta (Figura 16.1), el proceso de optimización de consultas comienza con la consulta generada por el usuario, aplicando la siguiente secuencia de pasos:

1. Un analizador sintáctico (parser) genera una expresión manipulable por el optimizador de consultas. El análisis sintáctico convierte al texto de entrada –en este caso, la consulta del usuario– en una estructura tipo árbol, que resulta más útil para su análisis.
2. A partir de la expresión interna, el optimizador obtiene una consulta equivalente más eficiente. En el apartado sobre evaluación del costo de las expresiones, se presenta la comparación entre operaciones equivalentes.
3. Por último, el proceso de optimización considera el estado actual de la BD y los índices definidos, para resolver la consulta que se tiene hasta el momento, con el acceso a disco posible

Evaluación de operaciones

El proceso de optimización de consultas comienza analizando la consulta generada por el analizador sintáctico. Se pueden aplicar diferentes consideraciones, en función de las operaciones indicadas. Para analizar estas

operaciones se utilizará el AR, dado que este lenguaje resulta más claro para explicar el comportamiento.

Operación de selección

Dada esta tabla:

DNI	Nombre	Genero	ECivil
22456980	Josefina	F	Casado
32456789	Juan	M	Casado
24567876	María	F	Casado
21345654	Roberto	M	Soltero
20987654	Alfredo	M	Casado
20897656	Fernanda	F	Casado
21345678	Raul	M	Casado

Selección: Personas del género masculino que sean solteros

- σ Genero='M' ^ ECivil='Soltero' (Persona) →
  - Se aplican 2 condiciones a 7 tuplas
- σGenero='M'(σ ECivil='Soltero' (Persona)) →
  - Se aplica 1 condición a 7 tuplas y 1 condición a 1 tuple
- σECivil='Soltero' (σGenero='M' (Persona))
  - Se aplica 1 condicion a 7 tuplas y 1 condicion a 4 tuplas
- Conclusión: el caso 2 es mejor, por lo que conviene realizar la selección lo antes posible

De esta manera, resolviendo la selección lo antes posible, se filtran las tuplas innecesarias del problema. Se debe recordar que los productos cartesianos, y aun los naturales, generan mucha información. Cuanto más pequeño sea el conjunto de entrada, más rápidamente será resuelta la consulta.

Este tipo de expresiones puede generalizarse a operaciones que involucren unión y diferencia. De ser posible, no solo debe realizarse la

selección antes que el producto, sino que esto debe extenderse a las operaciones de unión y diferencia.

Operación de proyección

En líneas generales, el análisis de una proyección indica que esta operación también debería realizarse lo antes posible.

Si bien los efectos de una selección son más importantes que los de una proyección debido a que la primera quita tuplas del resultado, la proyección reduce el tamaño de cada tupla. Esto significa que cada tupla ocupará menos espacio en el buffer en memoria y, por consiguiente, será posible almacenar más elementos.

La conclusión anterior respecto a la proyección se puede aplicar a otras operaciones binarias:

Unión

Intersección

Diferencia

Operación de producto natural

Una especial atención requiere el proceso de optimización del producto natural. Esto se debe a la frecuencia de aparición de este operador en las consultas sobre una BD. El producto natural es utilizado en muchas expresiones y genera como resultado un número importante de tuplas; por lo tanto, se debe tener en cuenta cómo se puede expresar una consulta. Expresiones del tipo
tabla1 |×| tabla2 |×| tabla3

deben intentar resolverse en pasos, es decir, resulta más eficiente resolver primero
tabla1 |×| tabla2
y con el resultado obtenido realizar el producto natural con la tabla restante.

Evaluación del costo de las expresiones

Los parámetros necesarios para evaluar el costo de las consultas ya fueron definidos. Se utilizarán CT tabla, CB tabla y CV (a, tabla) para evaluar el costo de las principales operaciones de AR. En cada caso, se analizarán la cantidad de tuplas resultantes y el tamaño en bytes de cada tupla.

Costo de la operación de selección

Sea la expresión

σat = valor (tabla)

El tamaño en bytes de cada tupla es CB tabla, debido a que no se genera proyección alguna.
Para analizar la cantidad de tuplas resultantes, se tendrán en cuenta la cantidad de valores diferentes para el atributo del predicado (CV(atributo, tabla)), y se supondrá que la distribución de valores es uniforme.
Suponga que una tabla tiene 1.000 tuplas y que para el atributo1 se tienen 20 valores diferentes. La distribución uniforme supone que es esperable tener 50 tuplas por cada valor diferente (1.000/20). Si bien esta consideración no tiene por qué ser real, es la suposición indispensable que se debe realizar para poder poner en práctica este análisis. Como conclusión, la cantidad de tuplas esperables de la consulta planteada será:

(CT tabla / CV (at, tabla)) * CB tabla

Nota: CT(Cantidad de tuplas), CB (Cantidad de bytes), CV(Cantidad de veces que aparece el atributo)

Costo de la proyección

Sea la expresión

π at1, at2, .. atn (Tabla)

(CB at1 + CB at2 + .. + CB atn) * CT tabla

Costo de la operación de producto cartesiano

Sea la expresión

T1 X T2

(CT t1 * CT t2) * (CB t1 + CB t2)

Costo de la operación de producto natural

Sin atributos en común → T1 |x| T2

CT (t1) * CT (t2)

T2 x T1

CT (t2) * CT (t1)

ES LO MISMO

Con atributo “a” en común, donde: a es PK en T1 y FK en T2.

T1 |X| T2 → un fila de T1 con muchas de T2.

Clave secundaria.

T2 |X| T1 → un fila de T2 con una de T1.

Clave primaria.

- P |x| C

c/tupla de P se junta con tuplas de C → CT (C) / CV(idlocalidad, C)

En P hay CT(P) Tuplas → **(CT(P) * CT (C))/ CV(idlocalidad, C)**

- C|x| P

c/tupla de C se junta con tuplas de P → CT (P) / CV(id,localidad P)

En C hay CT(C) Tuplas → **(CT(C) * CT (P))/ CV(idlocalidad, P)**

- (CT t1 * CT t2) / MAX(CV (a, t1) , CV (a, t2))

Seguridad e integridad de datos

Concepto de Transacción

Una transacción es una unidad lógica de trabajo. Para una BD, una transacción actúa como una única instrucción que tendrá éxito si se ejecuta completamente, o, en su defecto, nada de ella deberá reflejarse en la BD.

2da Definición:

Una transacción es un conjunto de instrucciones que actúa como unidad lógica de trabajo. Esto es, una transacción se completa cuando se ejecutaron todas las instrucciones que la componen. En caso de no poder ejecutar todas las instrucciones, ninguna de ellas debe llevarse a cabo.

Para garantizar que una transacción mantenga la consistencia de la BD, es necesario que cumpla con cuatro propiedades básicas ACID:

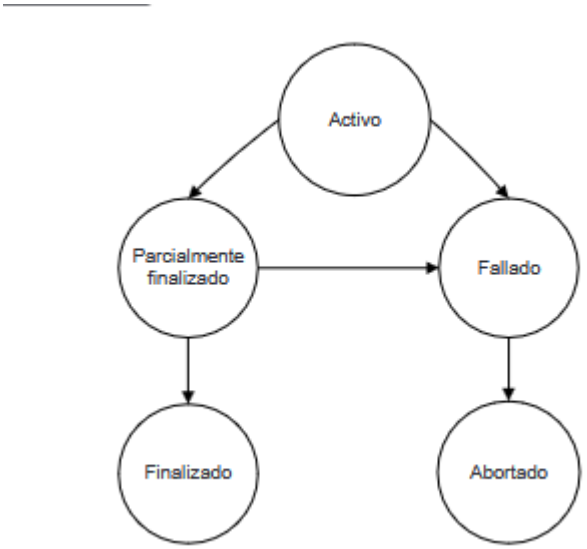
1. Atomicidad: garantiza que todas las instrucciones de una transacción se ejecutan sobre la BD, o ninguna de ellas se lleva a cabo.
2. Consistencia: La ejecución completa de una transacción lleva a la BD de un estado consistente a otro estado de consistencia. Para esto, la transacción debe estar escrita correctamente. En el ejemplo del pago de un servicio, si la transacción resta $100 de la cuenta del usuario y suma 120 en la cuenta del prestatario, no resulta en una transacción consistente.
3. Aislamiento (insolation en inglés): cada transacción actúa como única en el sistema. Esto significa que la ejecución de una transacción T1, no debe afectar la ejecución simultánea de otra transacción, T2.
4. Durabilidad: Una vez finalizada una transacción, los efectos producidos n la BD son permanentes.

Estados de una transacción

- Activa: estado inicial, estado normal durante la ejecución
- Parcialmente Cometida: después de ejecutarse la última instrucción
- Fallada: luego de descubrir que no puede seguir la ejecución normal

- Abortada: después de haber retrocedido la transacción y restablecido la BD al estado anterior al comienzo de la transacción
- Cometida: tras completarse con éxito

Una transacción siempre comienza en estado activo. De dicho estado puede migrar al estado de parcialmente finalizada, si ejecuta la última instrucción sin error, o al estado de fallada, en caso contrario. Una transacción fallada debe abortarse. Esto significa que todos los cambios que esta pudiera haber efectuado sobre la BD deben deshacerse, para garantizar la consistencia. Una vez deshechos los cambios, el estado de una transacción cambia a estado abortado.



Cualquier transacción que falla debe abortarse, para mantener la BD consistente. Sin embargo, suponga la siguiente situación: un usuario requiere pagar un servicio, genera el pago y este no puede llevarse a cabo debido a un fallo producido. Por ende, la transacción generada falla y aborta, impidiendo que el usuario realice el pago. La pregunta en este punto es: ¿Qué hacer con la transacción abortada? Las opciones son dos:

1. Reiniciar la transacción → No sirve
2. Dejar el estado de la BD como está → no sirve

Soluciones:

Registro histórico

Doble paginación

Entornos de transacciones

Las transacciones pueden ser analizadas desde tres perspectivas básicas.

Ambientes monousuario: Estos entornos tienen por característica básica que admiten un solo usuario operando con la BD, y a este usuario solamente se le permite realizar una actividad por vez.

Sistemas concurrentes: Estos entornos admiten varios usuarios simultáneos, cada uno de los cuales puede generar múltiples transacciones a la vez. En ese caso, el énfasis quedará sobre la propiedad de aislamiento. El comportamiento de las otras tres propiedades –atomicidad, consistencia y durabilidad– es similar tanto para entornos monousuarios como concurrentes.

Sistemas distribuidos: Estos entornos son los más comunes en la práctica actual de BD, y el estudio de las transacciones se torna más complejo. Una transacción generada por un usuario puede requerir modificar una BD residente en varios servidores. Los tipos de fallos que se pueden producir son mayores. Asegurar la integridad de una BD distribuida físicamente en varias computadoras es mucho más complejo. No obstante, no es propósito de este material profundizar el tratamiento de las transacciones distribuidas.

Registro Histórico

El método de bitácora (log) o registro histórico plantea que todas las acciones llevadas a cabo sobre la BD deben quedar registradas en un archivo histórico de movimientos. Una bitácora es un repositorio donde se registran acciones.

El método de bitácora consiste en registrar, en un archivo auxiliar y externo a la BD, todos los movimientos producidos por las transacciones antes de producir los cambios sobre la BD misma. De esta forma, en caso de producirse un error, se tiene constancia de todos los movimientos efectuados. Con este registro de información, es posible garantizar que el estado de consistencia de la BD es alcanzable en todo momento, aun ante un fallo.

La estructura del archivo de bitácora es simple. Cada transacción debe indicar su comienzo, su finalización y cada una de las acciones llevadas a cabo sobre la BD que modifiquen su contenido. El gestor del SGBD es el responsable de controlar la ejecución de las transacciones, de administrar el archivo de bitácora y de utilizar los algoritmos de recuperación cuando el SGBD se recupera de una situación de fallo.

Para poder cumplir con su cometido, el gestor identifica cada transacción asignándole un número correlativo único a cada una de ellas. El formato de una transacción en bitácora es el siguiente:
<T0 comienza>
<T0, dato1, valor viejo, valor nuevo>
<T0, dato2, valor viejo, valor nuevo>
<T0, daton, valor viejo, valor nuevo>
.....

<T0 Commit>
<T0 finaliza>

Para todas las transacciones que alcanzan el estado de finalizada, el trabajo registrado en bitácora es innecesario. Solamente cuando se produce un error, se deben activar los algoritmos que subsanan dicho error, utilizando para ello el registro histórico.

Transacción:

- Condición de idempotencia: Esta condición asegura que aunque se reejecute n veces una transacción, se genera siempre el mismo resultado sobre la BD.

La condición de idempotencia es fundamental en el siguiente caso. Suponga que se genera un fallo y durante la recuperación se decide reejecutar con REHACER la transacción Ti. Mientras se está reejecutando Ti, se produce un nuevo fallo. Nuevamente, cuando se recupera del fallo se deberá ejecutar otro REHACER. Es fundamental, entonces, que ante cada REHACER, el resultado sobre la BD sea siempre el mismo.

Buffers de Bitácora

- Grabar en disco c/registro de bitácora insume gran costo de tiempo → se utilizan buffer, como proceder?

- Transacción está parcialmente cometida después de grabar en memoria no volátil el Commit en la Bitácora.
- Un Commit en la bitácora en memoria no volátil, implica que todos los registros anteriores de esa transacción ya están en memoria no volátil.
- **Siempre** graba primero la Bitácora y luego la BD.

Modificación diferida

El método de bitácora con modificación diferida de la BD demora todas las escrituras en disco de las actualizaciones de la BD, hasta que la transacción alcance el estado de finalizada en bitácora. Así, mientras la transacción está en estado activo, se demora cualquier escritura física en la BD.
Esta variante posee una ventaja interesante. Si una transacción activa falla, se puede asegurar que los cambios producidos por la transacción no tuvieron impacto sobre la BD. Por lo tanto, el fallo de la transacción es ignorado y la BD permanece íntegra.

Para la modificación diferida de la BD, no es necesario guardar en bitácora el valor viejo del dato. Esto se debe a que no es posible modificar la BD hasta que la transacción alcance el estado de finalizada.

Ante un fallo, y luego de recuperarse:

- REDO (Ti), para todo Ti que tenga un Start y un Commit en la Bitácora.
- Si no tiene Commit entonces se ignora, dado que no llegó a hacer algo en la BD.

Modificación inmediata

La modificación inmediata de la BD plantea actuar de esta forma. Los cambios en la BD se producen a medida que se producen en la transacción que se está ejecutando, siempre bajo la consigna que indica que un cambio se registra primero en bitácora para luego grabarse en la BD.

Se necesita el valor viejo, pues los cambios se fueron efectuando.

Ante un fallo, y luego de recuperarse:

- REDO(Ti), para todo Ti que tenga un Start y un Commit en la Bitácora.
- UNDO(Ti), para todo Ti que tenga un Start y no un Commit.

En resumen, si se produce un error, se debe REHACER toda transacción de la bitácora que haya finalizado, y se debe DESHACER toda transacción iniciada que no haya finalizado

Puntos de verificación

El método de bitácora requiere una última consideración. Suponga que durante un determinado tiempo no se han producido incidentes en la operación con la BD. Básicamente, las transacciones ejecutadas durante los últimos tres meses finalizaron sin problemas. Toda esta información quedó registrada en bitácora. Luego, la transacción Ti comienza su ejecución y falla.
Sin importar si el método de recuperación
utilizado es con modificación inmediata o diferida de la BD, deberán rehacerse todas las transacciones que tengan registrados <T comienza> y <T finaliza>.
Es altamente probable que un gran porcentaje de las transacciones hayan finalizado bien, dejando la BD consistente. Rehacer todo ese trabajo es muy lento e innecesario.
Con el fin de evitar la revisión de toda la bitácora ante un fallo, el gestor de BD agrega en forma periódica, al registro histórico, una entrada

<punto de verificación>

Checkpoints (monousario)

- Se agregan periódicamente indicando desde allí hacia atrás todo OK.
- Periodicidad?

Esta situación resulta un tanto más compleja para entornos concurrentes.

Por último, se debe analizar la periodicidad de los puntos de verificación. Colocarlos “muy cerca” significa tener que rehacer poco trabajo en el caso de producirse un fallo, pero agrega una sobrecarga a la escritura de la bitácora de la BD. Por el contrario, puntos de verificación “lejanos” entre sí significan menos sobrecarga de escrituras en bitácora, pero generan la necesidad de rehacer mayor cantidad de trabajo ante un fallo.

Doble paginación

Este método plantea dividir al BD en nodos virtuales (páginas) que contienen determinados datos.
Se generan dos tablas en disco y cada una de las tablas direcciona a los nodos (páginas) generados.

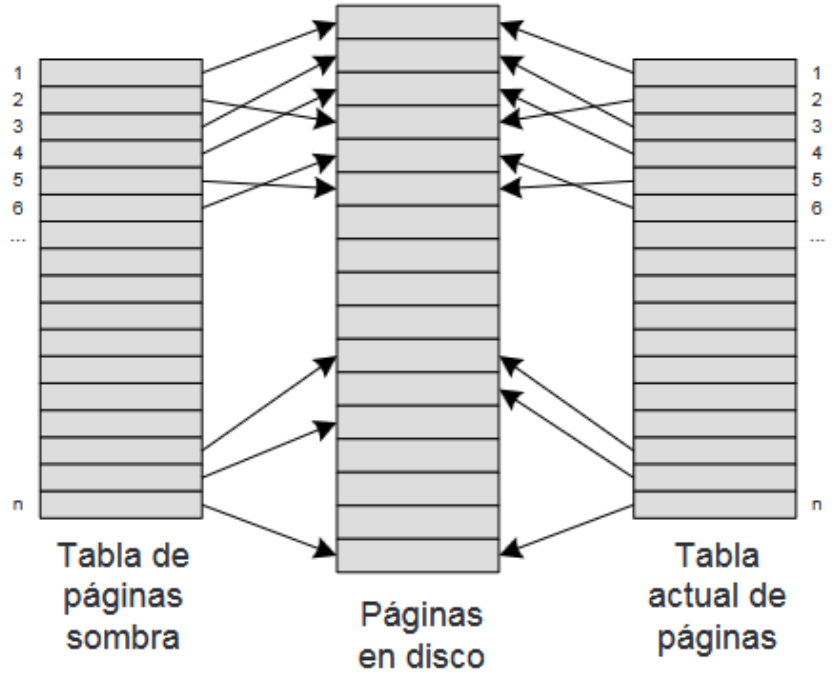
Ventaja: menos accesos a disco

Desventaja: complicada en un ambiente concurrente/distribuido.

N páginas equivalente a páginas del SO.

Tabla de páginas actual

Tabla de páginas sombra



El método trabaja de la siguiente forma. Ante una transacción que realiza una operación de escritura sobre la BD, la secuencia de pasos es la siguiente:

1. Se obtiene desde la BD el nodo (página) sobre el cual debe realizarse la escritura (si el nodo está en memoria principal, este paso se obvia).
2. Se modifica el dato en memoria principal y se graba en disco, en un nuevo nodo, la página actual que referencia al nuevo nodo.
3. Si la operación finaliza correctamente, se modifica la referencia de la página a la sombra para que apunte al nuevo nodo.
4. Se libera el nodo viejo.

En caso de producirse un fallo, luego de la recuperación, copia la tabla de páginas sombra en memoria principal. Abort automáticos, se tienen la dirección de la página anterior sin las modificaciones.

Ejecución de la operación *escribir*

- Ejecutar **entrada**(X) si página i-ésima no está todavía en memoria principal.
- Si es la primer escritura sobre la página i-ésima, modificar la tabla actual de páginas así:
    - Encontrar una página en el disco no utilizada
    - Indicar que a partir de ahora está ocupada
    - Modificar la tabla actual de página indicando que la i-ésima entrada ahora apunta a la nueva página

Este método presenta dos ventajas evidentes sobre el método de bitácora: se elimina la sobrecarga producida por las escrituras al registro histórico, y la recuperación ante un error es mucho más rápida (se recupera la página a la sombra como página actual, descartando la página actual insegura).

También presenta algunas desventajas:

- Sobrecarga en el compromiso: la técnica de paginación es por cada transacción.

- Fragmentación de datos: cambia la ubicación de los datos continuamente.
- Garbage Collector: ante un fallo queda una página que no es mas referenciada.

Transacciones en entornos concurrentes

Los SGBD, a través de los gestores de BD, permiten que varias transacciones operen simultáneamente sobre la BD, situación que puede generar problemas de consistencia.
Si bien dos transacciones, T1 y T2, son consistentes ejecutadas en entornos monousuarios, si la ejecución de estas se produce en simultáneo, puede alterarse la consistencia de la BD. Este problema se genera debido a que un entorno concurrente debe asegurar, además, la propiedad de aislamiento de una transacción. Esto significa que las dos transacciones T1 y T2 se interfieren en su ejecución.

Seriabilidad

Garantiza la consistencia de la BD.

Planificación: secuencia de ejecución de transacciones

Un conjunto de m transacciones generan m! planificaciones en serie.

La ejecución no necesita una planificación en serie

Cuando la ejecución de una transacción no afecta el desempeño de otra transacción sobre la BD, ambas pueden ejecutarse concurrentemente sin afectar la propiedad de aislamiento

Las conclusiones que se pueden obtener hasta el momento son las siguientes:

• Se debe mantener la integridad de la BD.
• La inconsistencia temporal puede ser causa de inconsistencia en planificaciones concurrentes. La inconsistencia temporal se presenta durante la ejecución de una transacción que modifica la BD, hasta que termine su ejecución (finalizando correctamente o abortando).
• Una planificación concurrente, para que sea válida, debe equivaler a una planificación en serie. Si se puede demostrar que la ejecución concurrente tiene el mismo resultado que una planificación en serie, entonces la planificación concurrente es válida.
• Solo las instrucciones READ y WRITE son importantes y deben considerarse para realizar cualquier tipo de análisis. Esto se debe a que el resto de las operaciones se pueden llevar a cabo sobre la computadora del usuario, en su memoria local, sin afectar el contenido de la BD.

Conflicto en planificaciones serializables:

- I1, I2 instrucciones de t1 y t2
  - Si operan sobre datos distintos no hay conflicto
  - Si operan sobre el mismo dato
    - i1 = READ(Q) = i2, no importa el orden de ejecución
    - I1 = READ(Q), I2 = WRITE(Q) depende del orden de ejecución (I1 leerá valores distintos)
    - I1 = WRITE(Q), I2 = READ(Q) depende del orden de ejecución (I2 leerá valores distintos)
    - I1 = WRITE(Q) = I2, depende el estado final de la BD
  - I1, I2 está en conflicto si actúan sobre el mismo dato y al menos una es un write. Ejemplos

Como conclusión, dos instrucciones son conflictivas si operan sobre el mismo dato, y al menos una de ellas es una operación de escritura.
Una Planificación S se transforma en una S´ mediante intercambios de instrucciones no conflictivas, entonces S y S´ son *equivalentes en cuanto a conflictos.*

Esto significa que si

- S′ es consistente, S también lo será
- S′ es inconsistente, S también será inconsistente
- **S´ es serializable en conflictos si existe S/ son equivalentes en cuanto a conflictos y S es una planificación serie.**

Métodos de control de concurrencia

Hasta el momento, se ha discutido cuándo una planificación concurrente es válida y cuándo no. El SGBD debe constar de algo más que de una definición, para poder ejecutar de manera concurrente y correcta una planificación que involucre múltiples transacciones simultáneas.
Como primera medida, es de notar que para que dos transacciones generen conflicto, estas deben operar sobre los mismos datos, aunque, generalmente, cuando se opera con una BD es poco probable que dos transacciones, al mismo tiempo, utilicen la misma información en forma completa.
Existen dos variantes para lograr implementar el aislamiento:

1. Protocolo de bloqueo.
2. Protocolo basado en hora de entrada.

Bloqueo

El método de bloqueo se basa en que una transacción debe solicitar el dato con el cual desea operar, y tenerlo en su poder hasta que no lo necesite más. El funcionamiento es similar al bloqueo que realiza el SO.

Básicamente, existen dos tipos de bloqueo a realizar sobre un elemento de datos:

1. Bloqueo compartido(lectura).
2. Bloqueo exclusivo (lectura/escritura).

- Las transacciones piden lo que necesitan.
- Los bloqueos pueden ser compatibles y existir simultáneamente (compartidos)

Un bloqueo compartido debe ser realizado por una transacción para leer un dato que no modificará. Mientras dure este bloqueo, se impide que

otra transacción pueda escribir el mismo dato. Un bloqueo compartido puede coexistir con otro bloqueo compartido sobre el mismo dato.

Un bloqueo exclusivo se genera cuando una transacción necesita escribir un dato. Dicho bloqueo garantiza que otra transacción no pueda utilizar ese dato (ya sea lectura o escritura de ese dato). Un bloqueo exclusivo es único. Una transacción que modifica un dato de la BD requiere que ninguna otra esté operando con ese dato en ese momento.

Cada transacción debe solicitar el tipo de bloqueo que necesite. Si obtiene el dato, debe utilizarlo y luego liberarlo, cuando la transacción finaliza. Si el dato está siendo usado en ese momento, la transacción tiene dos posibilidades: esperar por el dato, o fallar y abortar, para luego comenzar como una transacción diferente.

Se puede definir la condición de deadlock(abrazo mortal) como una situación en que dos procesos obtienen un elemento y ambos solicitan el elemento que tiene el otro. Como ninguno de los dos procesos puede obtener el segundo elemento solicitado, se produce deadlock, que impide continuar con la ejecución de los procesos.

Conclusiones:

Si los datos se liberan pronto → se evitan posibles deadlock.

Si los datos se mantienen bloqueados → se evita inconsistencia.

Protocolos de bloqueo

- Dos fases
  - Requiere que las transacciones hagan bloqueos en dos fases:
  - Crecimiento: se obtienen datos
  - Decrecimiento: se liberan los datos
  - Garantiza seriabilidad en conflictos, pero no evita situaciones de deadlock.
  - Como se consideran operaciones
    - Fase crecimiento: se piden bloqueos en orden: compartido, exclusivo

        - Fase decrecimiento: se liberan datos o se pasa de exclusivo a compartido.

Protocolo basado en hora de entrada

El protocolo basado en hora de entrada es una variante del protocolo de bloqueo, donde la ejecución exitosa de una transacción se establece de antemano, en el momento en que la transacción fue generada.

Cada transacción recibe una Hora de entrada (HDE) en el momento en que inicia su ejecución. La HDE es una marca única asignada a una transacción. Esta marca única puede ser el valor de un contador o la hora del reloj interno del servidor de la BD.

- El orden de ejecución se determina por adelantado, no depende de quien llega primero
- C/transacción recibe una HDE
    - Hora del servidor
    - Un contador
- Si HDE(Ti) < HDE(Tj), Ti es anterior
- C/Dato
    - Hora en que se ejecutó el último WRITE
    - Hora en que se ejecutó el último READ
    - Las operaciones READ y WRITE que pueden entrar en conflicto se ejecutan y eventualmente fallan por HDE.

Algoritmo de ejecución:

- **Ti Solicita READ(Q)**
    - HDE(Ti) < HW(Q): rechazo (solicita un dato que fue escrito por una transacción posterior)
    - HDE(Ti) ³ HW(Q): ejecuta y se establece HR(Q)=Max{HDE(Ti), HR(Ti)}
- **Ti solicita WRITE(Q)**

    - HDE(Ti) < HR(Q): rechazo (Q fue utilizado por otra transaccion anteriomente y suposu que no cambiaba)
    - HDE(Ti) < HW(Q): rechazo (se intenta escribir un valor viejo, obsoleto)
    - HDE(Ti) > [HW(Q) y HR(Q)]: ejecuta y HW(Q) se establece con HDE(Ti).
- **Si Ti falla, y se rechaza entonces puede recomenzar con una nueva hora de entrada.**

Granularidad

El concepto de granularidad en el acceso a los datos está vinculado con el tipo de bloqueo que se puede realizar sobre la BD.

La granularidad gruesa permite solamente bloquear toda la BD, y a medida que la granularidad disminuye, es posible bloquear a nivel tabla, registro o hasta campos que la componen.
En general, los SGBD permiten diversos niveles de bloqueo sobre los datos. El nivel mayor de bloqueo es sobre la BD completa. Esta acción está normalmente reservada para el diseñador de la BD, y se produce en actividades de mantenimiento. Durante este bloqueo, ningún usuario puede acceder a la BD.

Otras operaciones concurrentes

No solamente las operaciones de lectura y escritura deben garantizar el acceso aislado a la BD. Operaciones como insertar o borrar tuplas de una tabla también necesitan garantizar su operación en forma aislada.

- Delete(Q) requiere un uso completo del registro. no es posible que se intente leer o modificar la misma tupla.
- Insert(Q) el dato permanece bloqueado hasta la operación finalice.

Bitácora / Registro histórico

En el caso de entornos concurrentes, se agrega un nuevo tipo de fallo, una transacción que no puede continuar su ejecución por problemas de bloqueos o acceso a la BD. En ese caso, como en cualquier otra situación de fallo, la transacción debe abortar, dejando la BD en el estado de consistencia anterior al comienzo de su ejecución.

Consideraciones del protocolo basado en bitácora

- Existe un único buffer de datos compartidos y uno para la bitácora
- C/transacción tiene un área donde lleva sus datos
- El retroceso de una transacción puede llevar al retroceso de otras transacciones

Retroceso en cascada

- Falla una transacción → pueden llevar a abortar otras
- Puede llevar a deshacer gran cantidad de trabajo.

El retroceso de una transacción puede llevar a que otras transacciones también fallen. Suponga el siguiente ejemplo. La transacción T1 opera contra los datos 1, 2, 3 y 4; para su desarrollo utiliza varios ciclos de CPU. Obtiene, actualiza y libera el dato1. La transacción T2 obtiene el dato1, lo utiliza y finaliza su ejecución. Mientras tanto, la transacción T1 continuó su ejecución y produjo un fallo; la transacción T1, entonces, falla; por ende, aborta, retrocediendo todo lo actuado.

Durabilidad

- Puede ocurrir que falle Ti, y que Tj deba retrocederse, pero que Tj ya terminó. Como actuar?
- Protocolo de bloqueo de dos fases: los bloqueos exclusivos deben conservarse hasta que Ti termine.
- HDE, agrega un bit, para escribir el dato, además de lo analizado, revisar el bit si está en 0 proceder, si está en 1 la transacción anterior no termino, esperar....

Bitácoras con checkpoint

El único cambio que requiere el método de registro histórico con respecto a entornos monousuarios está vinculado con los puntos de verificación. En un entorno monousuario, un punto de verificación se agrega periódicamente luego de <Ti finaliza> y antes de <Tj comienza>.
El registro histórico de un esquema concurrente no puede garantizar la existencia de un momento temporal, donde ninguna transacción se encuentre activa. Por este motivo, la sentencia
<punto de verificación (L)>

agrega un parámetro L. Este parámetro contiene la lista de transacciones que, en el momento de colocar el punto de verificación, se encuentran activas.

- Ante un fallo
    - UNDO y REDO según el caso.
    - Debemos buscar antes del Checkpoint solo aquellas transacciones que estén en la lista.