Diseño de base de datos

Resumen teorías

Clase 1

Base de datos

Definición

Es una colección de datos relacionados

Colección de archivos diseñados para servir a múltiples aplicaciones

Un dato representa hechos conocidos que pueden registrarse y que tienen un resultado implícito.

Propiedades implícitas 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 se diseña, construye y completa de datos para un propósito específico. Está destinada a un grupo de usuarios concretos y tiene algunas aplicaciones preconcebidas en las cuales están interesados los usuarios
- Una BD está sustentada físicamente en archivos, los cuales están en dispositivos de almacenamiento persistente de datos

Conceptos básicos

La definición de una BD consiste en especificar los tipos de datos, las estructuras y restricciones de los mismos.

La construcción de la BD es el proceso de almacenar datos concretos en algún dispositivo de almacenamiento bajo la gestión del DBMS.

La manipulación de BD incluye funciones tales como consultar la BD para recuperar datos específicos, actualizar los datos existentes, reflejar cambios producidos, etc

Actores

- 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 de seguridad o respuesta lenta del sistema
- Diseñador de BD: Definen la estructura de la BD de acuerdo al problema del mundo real que esté representando
- 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)

DBMS

Definición

Las siglas 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

- Evitar redundancia e inconsistencia de datos
- Permitir acceso a los datos en todo momento
- Evitar anomalías en el acceso concurrente
- Restricción a accesos no autorizados seguridad
- Suministro de almacenamiento persistente de datos (aún ante fallos)
- Integridad en los datos Backups

Componentes

DDL (data definition language)

Especifica el esquema de BD. Resultado: Diccionario de datos

DML (data manipulation language):

- Recuperación de información
- Agregar información
- Quitar información
- Modificar información

Características:

- Procedimentales (SQL) requieren que el usuario especifique qué datos se muestran y cómo obtener esos datos
- No Procedimentales (QBE) requieren que el usuario especifique qué datos se muestran y sin especificar cómo obtener esos datos

Clase 2

Modelado

Abstracciones

Visión

Ve solo los datos de interés (muchas vistas para la misma BD) a través de programas de aplicación

Ej.: en el siu:

- Alumno: ver materias cursadas, mesas de finales
- Profesor: ver quien aprobó…
- Administradores: manejar inscripciones
- Directivos: ver estadísticas

Conceptual

Qué datos se almacenan en la BD y qué relaciones existen entre ellos

La DB es una sola, pero las visiones múltiples (al haber necesidades múltiples), se debe generar un único modelo de datos con toda la información

Físico

Describe cómo se almacenan realmente los datos (archivos y hardware)

Modelos

Basado en objetos (visión, conceptual)

Estructura flexible, especifican restricciones explícitamente

- Modelo de Entidad-Relación
- Modelo Orientado a Objetos

Basado en registros (conceptual, físico)

La BD se estructura en registros de formato fijo
Se dispone de lenguaje asociado para expresar consultas

- OO
- Relacional
- Jerárquico
- De red

Físico de datos

Diseño de datos

- Nivel conceptual (representación abstracta)
  - Integración de vistas
  - Genérico
  - Alejado del tipo de DBMS
  - Alejado del producto particular
  - Interactivo con el cliente

- Nivel lógico (representación en una computadora)
  - Más específico
  - Orientado a un tipo de DBMS
  - Alejado del producto particular

- Nivel físico (determinar estructuras de almacenamiento físico)
  - Específico
  - Orientado a un producto

Utilidad

Un modelo de datos sirve para hacer más fácil la comprensión de los datos de una organización

Se modela para:

- Obtener la perspectiva de cada actor asociado al problema
- Obtener la naturaleza y necesidad de cada dato
- Observar cómo cada actor utiliza cada dato

Modelo Entidad Relación

Se basa en la concepción del mundo real como un conjunto de objetos llamadas entidades y las relaciones que existen entre ellas

Permite modelar el nivel conceptual y lógico de una BD

Conceptual

Objetivos

Representar la información de un problema en un 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

Características

- 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 por el desarrollador

Clase 3

Minimalidad

Cada aspecto aparece una sola vez en el esquema:

Atentan contra ella

Ciclo de relaciones
Atributos derivados

Expresividad

Representa los requerimientos de manera natural y se puede entender con facilidad.

Autoexplicación

El esquema se explica a sí mismos cuando puede representarse un gran número de propiedades usando el modelo conceptual, sin otros formalismos.

Extensibilidad

Un esquema se adapta fácilmente a requerimientos cambiantes cuando puede descomponerse en partes, donde se hacen los cambios

Legibilidad

- Utilizar herramientas automatizadas
- Estructuras simétricas
- Se minimiza el número de cruces
- Generalización sobre los hijos

Modelo lógico

Para llevar a cabo el diseño lógico de alto nivel usando ER, debo tomar en cuenta 4 aspectos:

- Esquema conceptual
- Criterios de rendimiento (Qué preferimos: eficiencia de espacio, eficiencia de tiempo, legibilidad, etc.)
- Descripción del modelo lógico objetivo (ER, redes, jerarquía, objetos)
- Información de carga

Atributos derivados

- Ventaja de dejar:
    - Rápido para encontrarlo
    - Útil si lo necesito seguido (no lo tengo que calcular cada vez)
    - Útil si no cambia mucho (una vez que lo calculo se mantiene)
- Ventaja de sacar:
    - Lento para encontrarlo
    - Malo si lo necesito seguido (lo tengo que calcular cada vez)
    - Útil si cambia mucho (cada vez que lo calcule tiende a ser diferente)

Ciclo de relaciones

Si todas las relaciones son muchos a muchos no hay redundancia, no se atenta contra la minimalidad

Si no, se puede sacar una de las relaciones (en caso de que se puedan sacar varias se elige una dependiendo de su uso)

Redundancia vs uso

Atributos compuestos

1. Generar un único atributo que se convierta en la concatenación de todos los atributos simples que contiene el compuesto
2. Definir los atributos simples sin un atributo compuesto que resuma. La cantidad de atributos aumenta pero esta solución permite definir cada uno de los datos en forma independiente
3. 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 a la cual pertenecía el atributo compuesto

Atributos polivalentes

Un atributo en el modelo relacional debe ser simple

Se genera una nueva entidad con ese atributo

Se genera una relación entre la nueva entidad y la entidad que contenía el atributo polivalente

Teléfono

1. Concatenar varios teléfonos en un string
2. Poner varios atributos: teléfono personal, teléfono laboral, teléfono casa, etc
3. Crear una relación Tiene_telefonos con un atributo número (código de área, código de país, etc.), y que relacione Personas con Tipos_telefono, que tiene como identificador la descripción (laboral, casa, celular, etc.)

Jerarquías

Elminar entidades hijas

Se pasan los atributos y relaciones de los hijos al padre, cambia la cardinalidad a opcional

Se puede añadir un atributo categoría o tipo en el padre que indica cuál de los tipos los hijos es un padre

Eliminar entidad padre

Todos los atributos y relaciones del padre se colocan en cada uno de los hijos

No es posible si la cobertura es parcial

Conservar todo

Se crea un relación Es_un para cada hijo, que lo relaciona con el padre
La cardinalidad es (0, 1) desde el padre y (1, 1) desde los hijos

Clase 4

Modelo físico

Eliminación de identificadores externos

???

Selección de claves

Uno de los identificadores del modelo lógico pasa a ser clave primaria, el resto serán claves candidatas (todos son claves unívocas)

También se puede elegir una clave primaria autoincremental, dada por el DBMS y manejada por él. Cada nueva tupla tendrá el valor siguiente, en dicha clave, a la previa

Conversión de entidades

Cada entidad del modelo ER es una tabla en el modelo relacional

Relaciones

Clave foránea

Atributo/s de una tabla que en otra tabla es/son CP y que sirven para establecer un nexo entre ambas estructuras

Establecen integridad referencial entre tablas

Cardinalidad Muchos a Muchos

Se convierten en una tabla con dos claves foráneas, una por cada entidad relacionada

Cardinalidad Uno a Muchos

- Si el lado de cardinalidad 1 es (1, 1), se coloca una clave foránea de ese lado
- Si el lado de cardinalidad 1 es (0, 1), depende de si elegimos que las claves primarias puedan ser null o no:
    - Si pueden serlo: entonces se coloca una clave foránea de ese lado
    - Si no pueden serlo: se crea una tabla con dos claves foráneas, una por cada entidad

Cardinalidad Uno a Uno

(1, 1) a (1, 1)

Las dos entidades se ponen en una sola tabla

Único caso donde no se genera una tabla a partir de una entidad

Jerarquías

Se mantiene una tabla por hijo y una para el padre, y dentro de las tablas correspondientes a los hijos se coloca una clave foránea que hace referencia al padre. Lo ideal es que esa clave foránea del padre, también sea clave primaria en los hijos (con Personas, Alumnos y Profesores, lo ideal sería que id_persona sea clave primaria de las tres tablas (en Alumnos y Profesores también sería clave foránea))

Relaciones ternarias

En general relaciones muchos a muchos Cada entidad general tabla , idem la relación, que terminará con tres claves foráneas

Relaciones recursivas

Se tratan igual que la binarias

Restricciones

De dominio

Especifican que el valor de cada atributo A debe ser un valor atómico del dominio de A.

De clave

Evita que el valor del atributo clave genere valores repetidos

Sobre nulos

Evita que un atributo tome nulo en caso de no ingresarle valor

De integridad

Ningún valor de la clave primaria puede ser nulo

Restricción de integridad referencial

- Propiedad deseable de las BD
- Se especifica entre dos relaciones y sirve para mantener la consistencia entre tuplas de la dos relaciones
- Establece que una tupla en una relación que haga referencia a otra relación deberá referirse a una tupla existente en esa relación

- Clave foránea está representada por un atributo de una relación que en otra es clave primaria.
- Ej.: Si tengo que cada empleado trabaja solo en un departamento, voy a tener una clave foránea en la tabla Empleados, que hará referencia a la clave primaria de Departamentos (la cual sí o sí debe existir). De esta manera se establece integridad referencial entre ambas tablas. A la hora de querer realizar una baja de una tupla de Departamentos habrá que ver si se rompe la integridad referencial (si existen tuplas en Empleados que hagan referencia a la tupla que está siendo eliminada), en este caso habrá que decidir qué decisión tomar

Violaciones

Altas

Si se viola una regla, la operación se rechaza

Puede violar todas las operaciones (integridad referencial porque podría intentar darse de alta una tupla con una clave foránea que no exista en la otra tabla)

Bajas

Puede violar integridad referencial

Si una operación causa que se rompa la integridad referencial:

- Restrict (se restringe la operación)
- Cascade (se eliminan las tuplas con integridad referencial)
- Null (se pone en null la clave foránea)
- Nada

Modificaciones

Puede violar cualquiera de las operaciones

En el caso de la integridad referencial:

- Si no se modifica la clave primaria no hay problema (no se puede modificar si es autoincremental)
- Si se modifica la clave primaria se podrá:
    - Restrict (se restringe la operación)
    - Cascade (se modifica la clave foránea en las tuplas con integridad referencial)
    - Null (se pone en null la clave foránea)
    - Nada

Dependencias funcionales

Una DF es una restricción entre dos conjuntos de atributos de la BD

Formalmente: una DF X → Y (X entonces Y) entre dos conjuntos de atributos X e Y que son subconjuntos los atributos (R) de una relación (r), especifica una restricción sobre las posibles tuplas que podrían formar un estado de la relación r en R

En 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
- Y depende funcionalmente de X.

La restricción indica que si t1 y t2 son dos tuplas cualesquiera en r y que si t1[X] = t2[X] entonces debe ocurrir que t1[Y] = t2[Y]

Esto significa que los valores del componente Y de una tupla de r dependen de los valores del componente X

En general:

- Si una restricción en R dice que no puede haber más de una tupla con un valor X en r (convirtiendo a X en clave unívoca) entonces X → Y para cualquier Y de R
  Ej.: si conozco el id_alumno conozco su nombre, apellido, documento (en este caso Y → X vale), etc.
- Si X → Y en R, no se puede afirmar ni negar que Y → X (esto sucederá cuando Y sea clave unívoca)

Una clave unívoca siempre determina al resto de los atributos

Dependencia funcional completa

Si A y B son atributos de una relación r, B depende funcionalmente de manera completa de A, si B depende de A pero de ningún subconjunto de A. Es decir, no se puede sacar nada del antecedente para que se siga manteniendo la dependencia funcional

No generan problemas

Ej.: (nro_empl, nro_proy) → nombre_empleado
Nro_empl → nombre_empleado (completa)

Dependencia funcional parcial

A → B es una dependencia funcional parcial si existe algún atributo que puede eliminarse de A y la dependencia continúa verificándose. Es decir, se puede sacar algo del antecedente para construir una dependencia funcional completa

Parte_clave → no_clave

Generan problemas

Ej.: (nro_empl, nro_proy) → nombre_empleado (parcial)
Nro_empl → nombre_empleado

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

No_clave → no_clave

Repite información

Ej.: Nro_empleado → nombre, posición, nro_depto, nombre_depto
Nro_depto → nombre_depto

Queda implícito que: Nro_empleado → nombre_depto

Dependencia Boyce Codd

A → B es una dependencia funcional Boyce Codd si el determinante no es una parte candidata (o primaria)

No_clave → parte_clave

Repite información

Dependencia multivaluada

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 en 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 entre sí

Se dice que A →→ B y que A →→ C
Se lee A multidetermina B y A multidetermina C
Cuando dado un valor para el atributo a puedo obtener múltiples valores para el atributo B (o C en A →→ C)
No es un problema el hecho que un atributo esté multideterminado

Las dependencias multivaluadas solo se pueden dar en relaciones de 3 o más entidades (no deberían darse por la forma de hacer el modelo conceptual)

Ej.: la fecha de nacimiento multidetermina al nombre, puesto que para la fecha de nacimiento puedo obtener múltiples nombres

Dudoso
A →→ B es una dependencia multivaluada trivial
(A, B) →→ C es una dependencia multideterminada puede no ser trivial
Será trivial si puedo sacar A o B, es decir si A →→ C o B →→ C
Una dependencia multivaluada es trivial si puedo sacar algo del antecedente y que siga siendo una dependencia multivaluada (ahora trivial)

Ej.: (sucursal, empleado) →→ propietario
(sucursal, propietario) →→ empleado

Sucursal →→ propietario
Sucursal →→ empleado

Solución
t1= (sucursal, empleado)
t2= (sucursal, propietario)

Normalización

La normalización es una técnica formal que puede utilizarse en cualquier etapa del diseño de BD
La redundancia de datos en un modelo es la causa primaria de posibles inconsistencias

Definición

Técnica de diseño de BD que comienza examinando los nexos que existen entre los atributos (dependencias funcionales). La normalización identifica el agrupamiento óptimo de estos atributos, con el fin de identificar un conjunto de relaciones que soporten adecuadamente los requisitos de datos de la organización (lograr que las tablas tengan exclusivamente dependencias funcionales completas)

Propósito

Producir un conjunto de relaciones (tablas) con una serie de propiedades deseables partiendo de los requisitos de datos de una organización

Proceso

- Incremental
- Cada vez más restrictivo
- Comienza con BD en forma NO normal
- A medida que se avanza las relaciones (tablas) tienen un formato cada vez más restringido y son menos vulnerables a anomalías de actualización
- En general, 1NF es muy restrictiva (se aplica siempre)
- El resto puede ser opcional, de hecho 2NF y 3NF normalmente se aplican siempre

Primera forma normal (1NF)

Una tabla que contiene uno o más grupos repetitivos no está en 1FN, o sea una tabla que tenga atributos polivalentes

Un modelo estará en 1FN si para toda relación r del modelo (tabla) cada uno de los atributos que la forman son si y solo sí monovalente

No puede haber ni atributos polivalentes

Segunda forma normal (2NF)

Una tabla que tenga atributos que dependen parcialmente de otro no está en 2NF

Un modelo está en 2NF si y solo sí está en 1NF y para toda relación r del mismo (tabla) no existen dependencia parciales

Tercera forma normal (3NF)

Una tabla que tenga atributos que dependen transitivamente de otro no está en 3NF

Un modelo está en 3NF si y solo sí está en 2NF y para toda relación r del mismo (tabla) no existen dependencia transitivas

Forma normal Boyce Codd (BCNF)

Una tabla que tenga atributos que dependan de acuerdo a la definición de Boyce Codd de otro no está en BCNF

Un modelo está en BCNF si y solo sí está en 3NF y para toda relación r del mismo (tabla) no existen dependencia de Boyce Codd

Fue propuesta como una “suavización” de 3NF pero resultó ser más restrictiva

Una relación (tabla) está en BCNF si y solo sí todo determinante (X(determinante) → Y) es una clave candidata (o primaria)

La decisión de si es mejor detener el proceso en 3NF o llegar a BCNF depende de:

- La cantidad de redundancia que resulte de la presencia de una DF de Boyce Codd
- La posibilidad de perder una CC con la cual se podrían realizar muchos más controles sobre los datos.

En BCNF evito redundancia pero pierdo algo de control
Si dejo en 3NF tengo redundancia pero más control

Cuarta forma normal (4NF)

Un modelo está en 4FN si y solo sí está en BCNF (o si se optó por no ir por BCNF) y para toda relación r del mismo (tabla) sólo existen dependencia multivaluadas triviales

Quinta forma normal (5NF)

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

Clase 5

Lenguajes de consulta

Procedurales: (instrucciones para realizar secuencia de operaciones) (qué y cómo)
No procedurales: (solicita directamente la información deseada) (qué) (no los vemos)

Álgebra relacional

Lenguaje de consultas procedimental
Operaciones de uno o dos relaciones (tablas) de entrada que generan una nueva relación (tabla) como resultado

Operaciones fundamentales

- Unitarias (selección, proyección, renombre)
- Binarias (producto cartesiano, unión, diferencia)

Selección

Selecciona tuplas que satisfacen un predicado dado
Operador: σ

Ejemplo: mostrar todos los asociados casados
σ estado_civil = “casado” (Asociados)

Proyección

Devuelve la relación argumento con columnas omitidas
Operador: π

Ejemplo: nombres de los asociados
π nombre (Asociados)

Producto cartesiano

Conecta dos entidades de acuerdo a la definición matemática de la operación.
Operador: ✖

Ejemplo: mostrar cada asociado y la localidad donde vive
π asociado.nombre, localidad.nombre (σ asociado.idlocalidad = localidad.idlocalidad(Asociado ✖ Localidad)

Renombrar

Permite utilizar la misma tabla dos veces en, por ej., producto cartesiano
Operador: ρ

Ejemplo: mostrar todos los asociados que viven en la misma dirección que el socio con id 75

π aso.nombre (σ asociados.iddireccion = aso.iddireccion ((σ asociado.idsocio = 75 (Asociados)) ✖ ρ aso(Asociados))

Unión

Tuplas comunes a dos relaciones, equivalente a la unión matemática
Debe efectuarse entre relaciones con sentido (esquema compatibles)
Operador: ∪

Ejemplo: asociados que practiquen vóley o fútbol

π asociado.nombre (σ asociado.idsocio = practica.idsocio and deportes.iddeporte = practica.ideporte(σ deporte.nombre = “futbol” (Deportes) ✖ Practica ✖ Asociados))

∪

π asociado.nombre (σ asociado.idsocio = practica.idsocio and deportes.iddeporte=practica.ideporte(σ deporte.nombre = “voley” (Deportes) ✖ Practica ✖ Asociados))

Diferencia

Diferencia de Conjuntos
Debe efectuarse entre relaciones con sentido (esquema compatibles)
Operador: ─

Ejemplo: mostrar asociados que practiquen futbol y no voley

π asociado.nombre (σ asociado.idsocio = practica.idsocio and deportes.iddeporte = practica.ideporte(σ deporte.nombre = “futbol” (Deportes) ✖ Practica ✖ Asociados))

─

π asociado.nombre (σ asociado.idsocio = practica.idsocio and deportes.iddeporte=practica.ideporte(σ deporte.nombre = “voley” (Deportes) ✖ Practica ✖ Asociados))

Definición formal de Álgebra Relacional

Una expresión básica en AR consta de:

- Una relación de una Base de Datos
- Relación constante
- Una expresión general se construye a partir de sub-expresiones (E1,E2,...En)
- Expresiones:

- E1 ∪ E2
- E1 ─ E2
- E1 ✖ E2
- σ p (E1) p: predicado con atributos en E1
- π s (E1) s: lista de atributos de E1
- ρ x (E1) x: nuevo nombre de E1

Operaciones adicionales

Producto natural

Hace el producto cartesiano con una selección de tuplas “con sentido” eliminando las columnas (atributos) repetidas.

Si R y S dos relaciones no tienen atributos en común es igual al producto cartesiano

El atributo en común puede ser clave foránea en las dos tablas

Si hay más de un atributo en común NO se puede aplicar

Debe haber UN y SOLO UN atributo en común (con el mismo nombre y mismo dominio)

Operador: |✖|

•Ejemplo: asociados que practican fútbol

π asociados.nombre(Asociados |✖| Practica |✖| (σ nombre = “futbol” (Deportes)))

Intersección

Equivalente a la intersección matemática

Operador: ∩

A ∩ B = A ─ (A ─ B)

Asignación

Expresión que asigna a una variable temporal el resultado de una operación

Operador: ⇐

Temp ⇐ Operación del álgebra

Producto theta

Aplica una selección directamente al producto natural

Operador: θ

r |✖|θ s = σ θ (r |✖| s)

ABM

Altas

r ⇐ r ∪ E (r relación y E nueva tupla)
Ej.: Deportes ⇐ deportes ∪ (“golf”, 5000, 21)

Bajas

r ⇐ r ─ E
Ej.: Deportes ⇐ deportes ─ (“bochas”, 500, 1)

Modificaciones

δa ⇐ E (r)
Ej.: δsaldo ⇐ saldo * 1.05 (depósito)

Clase 6

SQL

DDL

Data Definition Language

CREATE DATABASE

```
CREATE DATABASE organizacion
```

DROP DATABASE

```
DROP DATABASE organizacion
```

CREATE TABLE

```
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));
```

ALTER TABLE

```
ALTER TABLE empresas
(ADD COLUMN razon_social VARCHAR(100) NOT NULL,
DROP COLUMN cuit,
ALTER COLUMN direccion VARCHAR(50) NULL);
```

DROP TABLE

```
DROP TABLE empresas
```

DML

Data Manipulation Language

Estructura básica

```
SELECT lista_de_atributos
FROM lista_de_tablas
WHERE condición
```

Ej.:
```
SELECT atr1, atr2,atr3
    FROM tabla1, tabla2
    WHERE atr4="Valor"
```

Funciones de agregación

- Promedio (AVG): aplicable a atributos numéricos, retorna el promedio de la cuenta
- Mínimo (MIN): retorna el elemento más chico dentro de las tuplas para ese atributo
- Máximo (MAX): retorna el elemento más grande dentro de las tuplas para ese atributo
- Total (SUM): aplicable a atributos numéricos, realiza la suma matemática
- Cuenta (COUNT): cuenta las tuplas resultantes

No pueden aparecer en el WHERE (a menos que estén en una subconsulta)
Deben aparecer en el SELECT o en el HAVING
Devuelven un único valor, por ende, no puede haber otro atributo en el SELECT si hay una función de agregación (a menos que dicho atributo esté en el GROUP BY)

Variantes del producto natural

- Left outer Join
    - Primero se calcula el inner join (ídem anterior)
    - Luego cada tupla t perteneciente a la relación de la izquierda que no encontró un par igual en la tupla de la derecha, aparece igualmente en el resultado final con valores nulos en los atributos del correspondientes a la derecha
    - Right outer Join: ídem anterior pero aparecen las tuplas t de la relación de la derecha
    - Full outer join: aparecen las tuplas colgadas de ambos lados.

ABM

Altas

Ejemplo: agregar un nuevo asociado a la tabla

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

Bajas

Ejemplo: quitar de la tabla el deporte bochas

DELETE FROM deportes WHERE nombre = “Bochas”

Ejemplo: como haría en el caso anterior si bochas tuviera deportistas y estuviera definida integridad referencial

DELETE FROM practica WHERE iddeporte = (SELECT iddeporte FROM deportes WHERE nombre = “bochas”)

y luego lo del ejemplo anterior

Modificaciones

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

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

Ejemplo: incrementar la cuota de hockey en un 30%

UPDATE deportes SET montocuota = montocuota * 1,3
WHERE nombre = “hockey”

Clase 7

Optimización de consultas

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)

Selección

Si la selección se hace cuanto antes, como peor caso va a ser la misma cantidad de comparaciones que hacer las selecciones juntas

Ej.: Hombres solteros

1. σ Genero=‘M’ ^ ECivil=‘Soltero’ (Persona) → Se aplican 2 condiciones a 7 tuplas
2. σ Genero=‘M’(σ ECivil=‘Soltero’ (Persona)) → Se aplica 1 condición a 7 tuplas y 1 condición a 1 tupla
3. σ ECivil=‘Soltero’ (σ Genero=‘M’ (Persona)) → Se aplica 1 condición a 7 tuplas y 1 condición a 4 tuplas

En el caso 2 y 3, dependiendo del estado de la DB, van a ser más o menos eficientes, pero sí o sí son más o igual de eficientes que 1

Proyección

Ej.: DNI de las personas que vivan en la ciudad de Junín

π DNI (Persona |✖| (σ Nombre=‘Junín’ (Ciudad)))

π DNI ((π DNI, IdCiudad (Persona)) |✖| (π IdCiudad (σ Nombre=‘Junín’ (Ciudad)))

El caso 2 es mejor, por lo que conviene realizar la proyección para disminuir la cantidad de información que se almacena en buffers de memoria

Esto se debe a que cada tupla va a ocupar menos espacio a la hora de leerlas de disco

Valores

CT tabla = cantidad de tuplas en tabla

CB tabla = cantidad de bytes en tabla

CV (a, tabla) = cantidad de valores diferentes del atributo a en tabla

Costos

Selección

σ (at = “valor”) (Tabla)
(CT tabla / CV (at, tabla)) * CB tabla

Proyección

π at1, at2, .. atn (Tabla)
(CB at1 + CB at2 + .. + CB atn) * CT tabla

Producto cartesiano

T1 ✖ T2
(CT t1 * CT t2) * (CB t1 + CB t2)

Producto natural

T1 |✖| T2
Sin atributos en común → es lo mismo que T1 ✖ T2

Con atributo “a” en común, donde: a es PK en T1 y FK en T2
T1 |✖| T2 → una fila de T1 con muchas de T2 → Clave secundaria (árbol B+)
T2 |✖| T1 → una fila de T2 con una de T1 → Clave primaria (hashing)

Con atributo “a” en común, donde: a es FK en T1 y FK en T2
T1 |✖| T2
Cada tupla de T1 se junta con tuplas de T2 → CT (T2) / CV(a, T2)
En T1 hay CT(T1) Tuplas → CT(T1) * CT (T2) / CV(a, T2)

Cada tupla de T2 se junta con tuplas de T1 → CT (T1) / CV(a, T1)
En T2 hay CT(T2) Tuplas → T(T2) * CT (T1) / CV(a, T1)

Conviene hacer aquel que tenga un CV más grande, es decir, más valores diferentes para “a”

Clase 8

Transacciones

Definición

Colección de operaciones que forman una única unidad lógica de trabajo

Propiedades ACID

Una serie de propiedades que debe cumplir una transacción

A: Atomicidad

Todas las operaciones de la transacción se ejecutan o no lo hacen ninguna de ellas (EL ÚNICO INCONVENIENTE EN UN ENTORNO MONOUSUARIO)

C: Consistencia

La ejecución aislada de la transacción conserva la consistencia de la BD

I: Aislamiento (isolation)

Cada transacción ignora el resto de las transacciones que se ejecutan concurrentemente en el sistema, actúa cada una como única

D: Durabilidad

Una transacción terminada con éxito realiza cambios permanentes en la BD, incluso si hay fallos en el sistema

Estados

- 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 (luego de que se escribió en disco)

Registro histórico (bitácora)

Contenido

- <T Start>
- <T, E, Va, Vn>
    - Identificador de la transacción
    - Identificador del elemento de datos
    - Valor anterior
    - Valor nuevo
- <T Commit>
- <T Abort>

Las operaciones sobre la BD deben almacenarse luego de guardar en disco el contenido de la Bitácora

Orden de escrituras

Write buffer bitácora
Write buffer BD
Output bitácora
Output BD

Modificación diferida

Las operaciones write se aplazan hasta que la transacción esté parcialmente cometida, en ese momento se actualiza la bitácora y la BD

No se necesita valor viejo (<T, E, Vn>), se modifica la BD al final de la transacción o no se modifica

Ante un fallo, y luego de recuperarse:

- REDO (Ti), para todo Ti que tenga un Start y un Commit en la Bitácora. A veces es necesario (si el fallo sucedió entre medio de múltiples write o si sucedió inmediatamente luego del commit) y a veces no (si el fallo sucedió luego de todos los write), pero como no sabemos cuándo ocurrió el fallo, siempre se hace el REDO

- Si no tiene Commit entonces se ignora, dado que no llegó a hacer algo en la BD (queda con valores no actualizados pero consistentes)

Tiene una carga de trabajo más dispareja: todos los write se hacen al final de la transacción

Es más simple: solo hago REDO si hay un fallo y tiene start y commit

Modificación inmediata

La actualización de la BD se realiza mientras la transacción está activa y se va ejecutando

Se necesita el valor viejo (<T, E, Va, Vn>), pues los cambios se fueron efectuando y 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 (puede tener abort)

Tiene una carga de trabajo más pareja: los write se hacen a lo largo de la transacción

Es más compleja: puedo hacer REDO o UNDO

Checkpoint

Se colocan cuando se está seguro que todo está bien

Se agregan periódicamente indicando desde allí hacia atrás todo OK

Condición de idempotencia de una transacción

Puede ocurrir un fallo mientras hago un REDO o UNDO. Cuando se recupera, se vuelve a hacer lo correspondiente. La recuperación de un fallo se puede repetir las veces que sea

Doble paginación

Ventaja: menos accesos a disco y recuperación muy rápida

Desventaja: complicada, sobre todo en un ambiente concurrente/distribuido

N páginas equivalente a páginas del SO:

- Tabla de páginas actual
- Tabla de páginas sombra

Ejecución de la operación write

- Ejecutar input(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

- Si la operación de escritura termina bien, la última operación que debe hacer es modificar la página sombra para que la i-ésima entrada apunte a la nueva página
  En caso de fallo y luego de la recuperación:
  - Copia la tabla de páginas sombra en memoria principal (es la que seguro apunta a algo que está bien)
  - Abort automáticos, se tienen la dirección de la página anterior sin las modificaciones

Ventajas

Elimina la sobrecarga de escrituras del log
Recuperación más rápida (no existe el REDO o UNDO)

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 más referenciada

Entornos Concurrentes

Varias transacciones ejecutándose simultáneamente compartiendo recursos
Deben evitarse los mismos problemas de consistencia de datos
Transacciones correctas, en ambientes concurrente pueden llevar a fallos

La ejecución serie de las transacciones asegura integridad de los datos. La BD va a ser consistente

Planificación

Es una secuencia de ejecución de transacciones (T1, T0, T3, T2 es una planificación)
Involucra todas las instrucciones de las transacciones
Conservan el orden de ejecución de las mismas
Un conjunto de m transacciones generan m! planificaciones en serie
La ejecución concurrente no necesita una planificación en serie (por algo es concurrente)
Una planificación concurrente debe equivaler a una planificación en serie

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

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 tal que son equivalentes en cuanto a conflictos y S es una planificación serie

Métodos de control de concurrencia

Bloqueo

Compartido Lock_c(dato)(solo lectura)
Exclusivo Lock_e(dato) (lectura/escritura)
Las transacciones piden lo que necesitan
Los bloqueos pueden ser compatibles y existir simultáneamente (compartidos)

Deadlock

Situación en la que una transacción espera un recurso de otra y viceversa
Hay que elegir una de las transacciones y abortarla

Si lo datos se liberan pronto → se evitan posibles deadlock
Si los datos se mantienen bloqueados → se evita inconsistencia

Protocolo de dos fases

Requiere que las transacciones hagan bloqueos en dos fases:

- Crecimiento: se obtienen datos. Se piden bloqueos en orden: compartido, exclusivo
- Decrecimiento: se liberan los datos o se pasa de exclusivo a compartido

Garantiza seriabilidad en conflictos, pero no evita situaciones de deadlock

Protocolo basado en hora de entrada

El orden de ejecución se determina por adelantado, no depende de quien llega primero

Cada transacción recibe una HDE: Hora del servidor o un contador
Si HDE(Ti) < HDE(Tj), Ti es anterior

Cada dato registra

- 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

Ti Solicita READ(Q)

- HDE(Ti) < HW(Q): rechazo (solicita un dato que fue escrito por una transacción posterior)
- HDE(Ti) >= 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 suposo 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

Se pueden bloquear registros, áreas o toda la BD

Otras operaciones conflictivas

Delete

Esto es porque para borrar una tupla de una tabla, dicha tupla debe estar bloqueada, así nadie lo puede acceder mientras lo elimino

Insert

Nadie puede usar el dato mientras lo inserto, pero pueden hacerse inserciones en conjunto

Ej.: Puedo dar de alta en factura y debo dar de alta los renglones en detalle, pero no quiero que se use ese registro de factura si no termine de dar de alta los registros en detalle

Si antes de terminar de darse de alta los detalles hay un fallo, la factura estaría dada de alta, pero los detalles no, entonces al bloquearla no permito que alguien la pueda utilizar si eso pasa

Registro Histórico en entornos concurrentes

Existe un único buffer de datos compartidos y uno para la bitácora

Cada transacción tiene un área donde lleva sus datos

El retroceso de una transacción puede llevar al retroceso de otras transacciones: retroceso en cascada

Ej.: Puede ocurrir que falle Ti, y que Tj deba retrocederse, pero que Tj ya terminó (por la propiedad de durabilidad una transacción realiza cambios permanentes en la BD incluso si hay fallos)

Solución: los bloqueos exclusivos deben conservarse hasta que Ti termine

Cuando hay dos operaciones que interactúan sobre los mismos datos, deben ser ejecutadas en serie

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

Checkpoint

Colocarlo cuando ninguna transacción esté activa. Puede que no exista el momento

Checkpoint<L> L lista de transacciones activa al momento del checkpoint

Ante un fallo:

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