

Bases de Datos

Tema 1: Introducción

1

*Ignacio Olmeda Martos
Antonio Moratilla Ocaña
Dept. Ciencias de la Computación
Universidad de Alcalá*

Indice

- 1.1. Definición de un SGBD
- 1.2. Propósito de los SGBD
- 1.3. Visión de los datos
- 1.4. Modelos de datos
- 1.5. Lenguajes de Bases de Datos
- 1.6. Gestión de Transacciones y Almacenamiento
- 1.7. Usuarios de Bases de Datos
- 1.8. Estructura general de un sistema

2

*Tema 3: Modelo relacional
Bases de Datos.
Ingeniería Técnica en Informática.*

*Antonio Moratilla Ocaña
Ignacio Olmeda Martos
Dept. Ciencias de la Computación
Universidad de Alcalá*

Referencias

- Silberschatz et al., 1998, pp. 1-14

1.1 Definición de un SGBD

- Un sistema de gestión de Bases de Datos (SGBD) consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a dichos datos.
- Los objetivos principales de un SGBD consisten en proporcionar un entorno eficaz y eficiente que permita el almacenamiento y la recuperación de información en una base de datos.

Bases de Datos

Tema 2: Modelo Entidad-Relación

1

*Ignacio Olmeda Martos
Antonio Moratilla Ocaña
Dept. Ciencias de la Computación
Universidad de Alcalá*

Indice

- 2.1. Conceptos básicos
- 2.2. Cuestiones de diseño
- 2.3. Ligaduras de correspondencia
- 2.4. Claves
- 2.5. Diagrama entidad-relación
- 2.6. Conjuntos de entidades débiles
- 2.7. El modelo E-R extendido
- 2.8. Diseño de un esquema de base de datos E-R
- 2.9. Reducción del esquema E-R a tablas

2

*Tema 3: Modelo relacional
Bases de Datos.
Ingeniería Técnica en Informática.*

*Antonio Moratilla Ocaña
Ignacio Olmeda Martos
Dept. Ciencias de la Computación
Universidad de Alcalá*

Referencias

- Silberschatz et al., 1998, pp. 15-44

2.1 Conceptos básicos

2.1.1 Conjuntos de entidades

- Una *entidad* es un objeto del mundo real que es distinguible de todos los demás objetos (p.e. una persona).
- Cada entidad tiene un conjunto de *propiedades* y valores para algún conjunto de propiedades que permiten identificarla de manera unívoca.
- Una entidad puede ser concreta (un libro) o abstracta (unas vacaciones).

2.1 Conceptos básicos

2.1.1 Conjuntos de entidades

- Un *conjunto de entidades* es la totalidad de las entidades del mismo tipo que comparten las mismas propiedades (p.e. Conjunto de entidades “cliente”).
- Los conjuntos no son necesariamente disjuntos (p.e. “empleado” e “impositor”).
- Una entidad se representa mediante un conjunto de atributos (p.e. “dni”, “ciudad”).

5

2.1 Conceptos básicos

- Para cada atributo hay un conjunto de valores permitidos llamado *dominio* del atributo (p.e. El dominio de “nombre cliente” es el conjunto de todas las cadenas de texto de una longitud).
- Cada entidad se puede describir como un conjunto de pares (atributo,valor), p.e. “cliente” se puede describir como {(nombre, López)}.
- Formalmente, un atributo es una función que asigna al conjunto de entidades un dominio.

6

2.1 Conceptos básicos

- Una BD incluye una colección de entidades cada una de las cuales incluye un número de entidades del mismo tipo.
- Podemos distinguir varios tipos de atributos:
 - simples y compuestos
 - univalorados y multivalorados
 - nulos
 - derivados

7

2.1 Conceptos básicos

- *Atributos simples y compuestos*: los atributos simples no pueden ser divididos en subpartes, mientras que los compuestos sí, p.e. “nombre_cliente” puede subdividirse en “nombre”, “primer_apellido” y “segundo_apellido”).
- Usar atributos compuestos permite poder referirse a a un atributo entero o a partes del atributo, lo que incrementa la flexibilidad.
- Usar atributos compuestos permite agrupar los atributos, haciendo modelos más claros.

8

Tema 3: Modelo Relacional

Bases de Datos

Indice

- 3.1 Estructura de las BD relacionales
- 3.2 Algebra relacional
- 3.3 Cálculo relacional de tuplas
- 3.4 Cálculo relacional de dominios
- 3.5 Operaciones del álgebra relacional extendida
- 3.6 Modificación de la base de datos
- 3.7 Vistas

Estructura de las BD relacionales

- Una *BD relacional* consiste en un conjunto de tablas, cada una de ellas con un nombre exclusivo.
- Cada fila de la tabla representa una *relación* entre un conjunto de valores.
- Puesto que cada tabla es un conjunto de relaciones podemos pensar que es una *relación matemática*.

Estructura básica

- Cada tabla consta de un conjunto de columnas que se denominan *atributos*.
- Para cada atributo hay un conjunto de valores permitido que se denomina *dominio* de ese atributo, por ejemplo

Relación (tabla) “cuenta”

nombre_sucursal	numero_cuenta	Saldo	
centro	c-101	100,000	
navacerrada	c-215	140,000	
galapagar	c-217	150,000	

Estructura básica

- La tabla anterior tiene tres atributos, el dominio de los dos primeros puede ser un conjunto alfanumérico, mientras que el del tercero pueden ser los números enteros.
- Si D_1, D_2, \dots, D_n son los dominios de una tabla con n atributos, entonces dicha tabla puede ser vista como un subconjunto de $D_1 \times D_2 \times \dots \times D_n$.
- Puesto que una tabla puede ser vista como una relación matemática se suele hablar de *relación* y *tupla* en lugar de *tabla* y *fila*.

Estructura básica

- La anterior tabla tenía tres tuplas.
- Supongamos que nos referimos a variable tupla t para hacer referencia a la primer tupla de la relación, utilizaremos la notación $t[\text{atributo}]$ para denotar el valor de t de un determinado atributo, por ejemplo $t[\text{nombre_sucursal}] = \text{“centro”}$, también se suele escribir $t[1]$ para el primer atributo, $t[2]$ para el segundo, etc.
- Se suele emplear la notación matemática $t \in r$ para denotar que la tupla t esta en la relación r .

Estructura básica

- Exigiremos que para todas las relaciones r los dominios sean *atómicos*, es decir, si sus dominios se consideran unidades indivisibles, por ejemplo Z (enteros) es atómico pero $\wp(Z)$ (partes de Z) no lo es.
- Existen modelos de BD que permiten trabajar con dominios no atómicos (el *modelo relacional anidado*), que no veremos.

Estructura básica

- Es posible que varios atributos tengan el mismo dominio (por ejemplo los atributos 1 y 2 de la anterior tabla).
- Un valor de dominio que, por convención, pertenece a cualquier dominio es el valor nulo que indica que un valor es desconocido o no existe.

Esquema de la BD

- Al igual que antes, hay que distinguir entre *esquema* de la BD y *ejemplar* de la BD.
- Se emplean nombres en minúsculas para las relaciones y nombres que comiencen por una letra mayúscula para los esquemas de las relaciones, por ejemplo

Esquema_cuenta=(nombre_sucursal, número_cuenta, saldo)

Esquema de la BD

- Para denotar que cuenta es una relación de Esquema_cuenta pondremos:

cuenta(Esquema_cuenta)

- Un *ejemplar de la relación* puede cambiar con el tiempo, generalmente se habla de una relación cuando en realidad se está hablando de un ejemplar de la relación.

Esquema de la BD

- Generalmente se suelen emplear atributos comunes en diferentes esquemas de relaciones, por ejemplo, si definimos la relación “sucursal”

nombre_sucursal	ciudad	activos	
centro	arganzuela	1,800,000,000	
navacerrada	aluche	340,000,000	
cuenca	villaverde	15,000,000	

- el atributo nombre de la sucursal aparece en Esquema-Sucursal y en Esquema-cuenta, esto permitirá relacionar las tuplas de relaciones diferentes

Esquema de la BD

- A veces parece que tener varios esquemas de relaciones es una redundancia pero esto permite aumentar la flexibilidad de la BD, por ejemplo dados los esquemas de relaciones:

Esquema-cliente=(nombre-cliente, calle-cliente, ciudad-cliente)

Esquema-impositor=(nombre-cliente, número-cuenta)

- podría pensarse que es más útil tener un solo esquema como por ejemplo

(nombre-sucursal, ciudad-sucursal, activos, nombre-cliente, calle-cliente, ciudad-cliente, número-cuenta, saldo)

Esquema de la BD

- Sin embargo esto no es así:
 - si un cliente tiene varias cuentas hay que repetir su dirección una vez por cuenta
 - si una sucursal no tiene ninguna cuenta no se pueden completar algunas tuplas (no hay datos de clientes ni cuentas) y habría que utilizar valores nulos
- Finalmente, al igual que antes es posible representar todas las relaciones mediante un diagrama E-R (ver ejemplo Silberschatz pág. 49).

Claves

- Los conceptos relacionados con claves son también aplicables en el modelo relacional, por ejemplo En el Esquema-sucursal $\{\text{nombre-sucursal}\}$ es una *superclave*.
- Sea R el esquema de una relación, si K es una superclave de R entonces no existen tuplas t_1 y t_2 de R que tengan iguales valores en todos los atributos de K , es decir $t_1[k] \neq t_2[k]$

Estructura de las BD relacionales

- Si el esquema de una BD relacional se basa en un esquema E-R es posible determinar la clave primaria a partir de la clave primaria de dicho esquema.
 - Para las entidades fuertes la clave primaria de las entidades es la clave primaria de la relación
 - Para las entidades débiles la clave primaria es la unión de la clave primaria de las entidades fuertes y el discriminante del conjunto de entidades débiles
 - para un conjunto de relaciones la unión de las claves primarias de las relaciones es una superclave de la relación, si la relación es varios a varios esta superclave es también clave primaria,

Estructura de las BD relacionales

- Para las tablas combinadas, en el esquema E-R un conjunto de relaciones varios a uno entre A y B se podía representar mediante una tabla combinada con los atributos de A y los de la relación (si existen), en este caso la clave primaria de la relación es la clave primaria de A, si la relación es uno a uno la clave primaria de la relación puede ser la clave primaria de A o de B indistintamente.
- Para los atributos multivalorados M, en el esquema E-R se representaban como una tabla con la clave primaria del conjunto De entidades o relaciones de las que M es atributo y una columna C con el valor concreto de M, en este caso la clave primaria es la clave primaria del conjunto de entidades o relaciones junto con el atributo M.

Lenguajes de consulta

- Un *lenguaje de consulta* es aquél en el que un usuario solicita información de la base de datos.
- Hay dos tipos:
 - *procedimentales*: el usuario indica al sistema cuales son las operaciones para calcular el resultado deseado (por ejemplo el álgebra relacional)
 - *no procedimentales*: el usuario sólo describe la información deseada, sin indicar el procedimiento (por ejemplo (el cálculo relacional de tuplas y el cálculo relacional de dominios)

Álgebra Relacional

- Es un lenguaje de consulta *procedimental*.
- Consta de un conjunto de operaciones sobre una o varias relaciones y que producen nuevas relaciones.
- Las operaciones fundamentales son: selección, proyección, unión, diferencia de conjuntos, producto cartesiano y renombramiento
- Hay además otras operaciones: intersección de conjuntos, reunión natural, división y asignación

Operaciones fundamentales

- Son de dos tipos.
 - *Unarias*: operan sobre una sola relación (selección, proyección y renombramiento)
 - *Binarias*: operan sobre pares de relaciones (unión, diferencia de conjuntos y producto cartesiano)

Operación selección

- Selecciona tuplas que satisfacen un predicado dado
- Se nota mediante la letra sigma minúscula (σ)

Álgebra Relacional

- Ejemplo: dada la relación préstamo

<i>nombre-sucursal</i>	<i>número-préstamo</i>	<i>importe</i>
centro	p-17	200000
moralzarzal	p-23	400000
navacerrada	p-15	300000
centro	p-14	300000
becerril	p-93	100000
collado mediano	p-11	180000
navacerrada	p-16	260000

$\sigma_{\text{nombre-sucursal}=\langle \text{Navacerrada} \rangle}(\text{préstamo})$

produce la relación:

<i>nombre-sucursal</i>	<i>número-préstamo</i>	<i>importe</i>
moralzarzal	p-23	400000
navacerrada	p-15	300000

Álgebra Relacional

- También se pueden emplear comparaciones que utilicen =, ≠, <, >, ≥, y se pueden combinar predicados utilizando las conectivas y (∧) y o (∨).
- por ejemplo para encontrar las tuplas correspondientes a préstamos de más de 240.00 pts concedidos por la sucursal “Navacerrada” pondríamos:

$$\sigma_{\text{nombre-sucursal} = \text{Navacerrada} \wedge \text{importe} > 24000}(\text{préstamo})$$

Álgebra Relacional

- También podemos incluir comparaciones entre atributos, por ejemplo En la relación responsable-préstamo podríamos querer encontrar si un empleado se llama igual que un cliente:

$$\sigma_{\text{nombre-cliente} = \text{nombre-empleado}}(\text{responsable-préstamo})$$

- Dado que el valor nulo denota desconocido o inexistente, cualquier comparación que conduzca a él se denota como falsa.

Álgebra Relacional

- Permite extraer una relación que contenga sólo los atributos incluidos en el subíndice
- Puesto que el resultado (una relación) es un conjunto, se eliminan las filas repetidas si las hubiera
- Se nota mediante la letra minúscula pi (π).

Álgebra Relacional

Ejemplo, en la relación préstamo,

$\pi_{\text{número-préstamo, importe}}(\text{préstamo})$

produce:

<i>número-préstamo</i>	<i>importe</i>
p-17	200000
p-23	400000
p-15	300000
p-14	300000
p-93	100000
p-11	180000
p-16	260000

Operación Unión

- Permite encontrar la relación compuesta por tuplas que que aparecen en un par de relaciones (en una, en la otra o en ambas)
- Se nota mediante el símbolo de unión conjuntista (\cup).
- Se debe asegurar que la unión se realice entre relaciones compatibles, para que $r \cup s$ sea válida deben cumplirse:
 - r y s deben ser de la misma *aridad* (igual número de atributos)
 - los dominios de sus atributos deben ser iguales

Álgebra Relacional

- por ejemplo podríamos estar interesados en los clientes que tienen una cuenta, un préstamo o ambas cosas, $\pi_{\text{nombre-cliente}}(\text{prestatarario})$ devuelve los nombre de clientes con préstamos, $\pi_{\text{nombre-cliente}}(\text{impositor})$ devuelve los nombres de clientes con cuenta, así pues nuestra consulta podría expresarse:

$$\pi_{\text{nombre-cliente}}(\text{prestatarario}) \cup \pi_{\text{nombre-cliente}}(\text{impositor})$$

Operación diferencia de conjuntos

- Permite buscar las tuplas que están en una relación pero no en otra
- Se nota con el signo menos (-). La operación $r-s$ produce una relación constituida por las tuplas que están en r pero no en s
- Al igual que en la unión, las relaciones deben ser compatibles

Operación diferencia de conjuntos

- por ejemplo, si queremos buscar los nombres de clientes con cuenta pero que no tienen concedido ningún préstamo podríamos:

$$\pi_{\text{nombre-cliente}}(\text{impositor}) - \pi_{\text{nombre-cliente}}(\text{prestatarario})$$

Operación producto cartesiano

- Permite combinar información de cualesquiera dos relaciones
- Se nota por un aspa (x)
- Dado que el nombre de un atributo puede aparecer en ambas relaciones hay que crear un esquema de denominaciones para distinguir entre ambos atributos

Operación producto cartesiano

- Por ejemplo: $r = \text{prestatario} \times \text{préstamo}$
(prestatario.nombre-cliente, prestatario.num-préstamo, préstamo.num-préstamo)
- Esta operación exige que las relaciones tengan nombres diferentes (no se permite por ejemplo préstamo \times préstamo), esto se puede evitar con la operación renombramiento
- Si hay n_1 tuplas en r_1 y n_2 tuplas en r_2 hay $n_1 \times n_2$ tuplas en $r = r_1 * r_2$

Operación producto cartesiano

Ejemplo

- Se quieren averiguar los nombres de todos los clientes con un préstamo en Navacerrada

1- Necesitamos la información de préstamo y prestatario, así que hacemos:

$\sigma_{\text{nombre- sucursal} = \text{“Navacerrada”}}(\text{prestatario} \times \text{préstamo})$

Operación producto cartesiano

- 2- Sin embargo habrá tuplas en esa relación para las que el cliente no tenga concedido ningún préstamo en Navacerrada (el producto nos da todas las posibles tuplas)
- 3- Para seleccionar sólo a éstos hacemos:

$\sigma_{\text{prestatario-n}^\circ\text{préstamo}=\text{préstamo.num-préstamo}}$

$(\sigma_{\text{nombre-sucursal}=\text{"Navacerrada"}} (\text{prestatario} \times \text{préstamo}))$

Operación producto cartesiano

4- Como solo necesitamos los nombres hacemos una proyección

$$\pi(\text{nombre-cliente}(\sigma_{\text{prestatario-n}^\circ\text{préstamo}=\text{préstamo-n}^\circ\text{préstamo}}(\sigma_{\text{nombre-sucursal}=\text{"Navacerrada"}}(\text{prestatario} \times \text{préstamo}))))$$

Operación renombramiento

- Los resultados del Álgebra relacional son relaciones, sin embargo no tienen un nombre para referirnos a ellas
- El operador renombramiento, notado por la letra *rho* minúscula, ρ , permite realizar esta tarea
- $\rho_x(E)$, devuelve el resultado de la expresión E con el nombre x

Operación renombramiento

- Las propias relaciones r se consideran expresiones triviales del álgebra relacional por lo que se puede aplicar el renombramiento a r para obtener la misma relación con un nombre nuevo:

$$\rho_x(r) = x$$

Operación renombramiento

Ejemplo

“Buscar el máximo saldo de cuenta del banco”

1- Seguiremos una estrategia consistente en calcular primero una relación intermedia con saldos que no son el máximo y luego realizar la diferencia entre

π_{saldo} (cuenta) y dicha relación obtenida

Operación renombramiento

- 2- Para calcular la relación intermedia hay que comparar los saldos de todos los clientes
- i) en primer lugar renombramos $\rho_d(\text{cuenta})$
 - ii) seguidamente construimos la relación de saldos que no son el máximo

$$\pi_{\text{cuenta-saldo}} (\sigma_{\text{cuenta.saldo} < d.\text{saldo}} (\text{cuenta} \times \rho_d(\text{cuenta})))$$

Esta última relación contiene todos los saldos salvo el máximo

Operación renombramiento

iii) por último calculamos el máximo:

$$\pi_{\text{saldo}}(\text{cuenta}) - \pi_{\text{cuenta.saldo}}(\sigma_{\text{cuenta.saldo} < \text{d.saldo}}(\text{cuenta} \times \rho_d(\text{cuenta}))).$$

Otras operaciones

Operación intersección de conjuntos

- Permite encontrar tuplas presentes simultáneamente en dos relaciones
- no añade mayor potencia computacional al álgebra relacional pero simplifica la notación
- se nota mediante el símbolo matemático de intersección (\cap)

Operación intersección

- Es equivalente a:

$$r \cap s = r - (r - s)$$

- Ejemplo: encontrar todos los clientes con un préstamo concedido y una cuenta abierta:

$$\pi_{\text{nombre-cliente}}(\text{prestatarario}) \cap \pi_{\text{nombre cliente}}(\text{impositor})$$

Operación reunión natural

- Sirve para simplificar ciertas consultas que involucran un producto cartesiano
- Sirve para combinar ciertas selecciones y un producto cartesiano en una sola operación
- Forma un producto cartesiano de sus dos argumentos, realiza una selección forzando la igualdad de los atributos que aparecen en ambos esquemas y finalmente elimina los atributos duplicados

Operación reunión natural

- Se nota mediante el símbolo $\triangleright \triangleleft$
- Ejemplo: averiguar los nombres de todos los clientes que tienen concedido un préstamo y averiguar su importe:

$\pi_{\text{nombre-cliente, préstamo.número préstamo, importe}} (\text{prestatario} \triangleright \triangleleft \text{préstamo})$

Operación reunión natural

- Puesto que los esquemas de prestatario y de préstamo tienen en común el atributo número-préstamo, la operación reunión natural solo considera los pares de tuplas que tienen el mismo valor de número-préstamo
- Formalmente, considerando los esquemas R y S como conjuntos tenemos:

$$r \bowtie s = \pi_{R \cup S}(\sigma_{r.A_1=s.A_1 \wedge r.A_2=s.A_2 \wedge \dots \wedge r.A_n=s.A_n}(r \times s)), \text{ donde } R \cap S = \{A_1, A_2, \dots, A_n\}$$

Operación división

- Es adecuada para las consultas que incluyen la expresión “para todo”
- Se nota mediante el símbolo matemático de división “ \div ”
- Formalmente dadas r y s y $S \subseteq R$ (es decir, todos los atributos del esquema S están también en el esquema R) la relación $r \div s$ es una relación del esquema $R-S$

Operación reunión natural

- Una tupla está en $r \div s$ si se cumplen:
 - i) t está en $\pi_{R-S}(r)$
 - ii) $\forall ts \in S \exists tr \in R$ que cumple las dos condiciones:
 - a) $tr[S] = ts[S]$
 - b) $tr[R-S] = t$

Operación reunión natural

- Ejemplo: halla a todos los clientes que tengan abierta una cuenta en todas las sucursales de “Arganzuela”

$$r_1 = \pi_{\text{nombre-sucursal}} (\sigma_{\text{ciudad-sucursal} = \text{“Arganzuela”}} (\text{sucursal}))$$

da todas las sucursales de Arganzuela

Operación reunión natural

$r_2 = \pi_{\text{nombre-cliente, nombre-sucursal}} (impositor \triangleright \triangleleft cuenta)$
da los pares nombre-cliente, nombre-sucursal para los que el cliente tiene una cuenta en una sucursal
 $r_2 \div r_1$ da los nombres de clientes que aparecen en r_2 con todas las sucursales de r_1

- Formalmente:

$$r \div S = \pi_{R-S}(r) - \pi_{R-S}((\pi_{R-S}(r) \times S) - \pi_{R-S}, S(r))$$

Operación asignación

- Sirve para asignar temporalmente una relación
- Se nota mediante una flecha dirigida a la izquierda “←”, la asignación va de derecha a izquierda
- La evaluación de la asignación no hace que se muestre ninguna relación al usuario
- Ejemplo, podemos escribir:

$$\begin{aligned}temp\ 1 &\leftarrow \pi_{R-S}(r) \\temp\ 2 &\leftarrow \pi_{R-S}((temp1 \times S) - r) \\resultado &= temp1 - temp2\end{aligned}$$

Cálculo relacional de tuplas

- Es un lenguaje de consulta no procedimental
- Las consultas se expresan como $\{t \mid P(t)\}$
es decir, son el conjunto de todas las tuplas tales que el predicado P es cierto para t
- Según la notación anterior también emplearemos $t[A]$ para denotar el valor de la tupla t en el atributo A y $t \in r$ para denotar que la tupla t está en la relación r

Consultas de ejemplo

- Se desea averiguar nombre-sucursal, n°préstamo e importe de los préstamos superiores a 240.000 pts escribiremos:

$\{t \mid t \in \text{préstamo} \wedge t[\text{importe}] > 240.000\}$

- Ejemplo: se desea averiguar el n° de préstamo de todos los préstamos por importe superior a 240.000 pts.

Consultas de ejemplo

- En este caso solo queremos obtener el atributo n° préstamo, para hacerlo utilizamos la expresión

$$\exists t \in r (Q (r))$$

que significa que existe una tupla t en la relación r tal que el predicado Q es verdadero

Con esta notación tenemos:

$$\{t \mid \exists s \in \text{préstamo} (t[\text{n}^\circ\text{-préstamo}] = s[\text{n}^\circ\text{-préstamo}] \wedge s[\text{importe}] > 240.000)\}$$

Consultas de ejemplo

- Que se lee:

“el conjunto de todas las tuplas t tales que existe una tupla s en la relación préstamo para la que los valores de t y s para el atributo n° préstamo son iguales y el valor de s para el atributo “importe” es mayor que 240.000 pts”

Consultas de ejemplo

- Ejemplo: para averiguar todos los clientes del banco que tienen concedido un préstamo, una cuenta abierta o ambas cosas empleamos el o (\vee):

$$\{t \mid \exists s \in \text{prestatario} (t [\text{nombre-cliente}] = s [\text{nombre-cliente}]) \vee \exists u \in \text{impositor} (t [\text{nombre-cliente}] = u [\text{nombre-cliente}])\}$$

Consultas de ejemplo

- Ejemplo: averiguar todos los clientes que tienen una cuenta abierta pero no un préstamo, para ello empleamos la negación “ \neg ”

$\{t \mid \exists u \in \text{impositor}(t[\text{nombre-cliente}] = u[\text{nombre-cliente}]) \wedge \neg \exists s \text{ prestatario}(t[\text{nombre-cliente}] = s[\text{nombre-cliente}])\}$

Consultas de ejemplo

- Existe otro operador , la implicación $P \Rightarrow Q$
- Un ejemplo de su uso: averiguar todos los clientes que tienen una cuenta en todas las sucursales de Arganzuela

$$\{ t \mid \forall u \in \text{sucursal} (u [\text{ciudad-sucursal}] = \text{Arganzuela} \Rightarrow \exists s \in \text{impositor} (t [\text{nombre-cliente}] = s [\text{nombre-cliente}] \wedge \exists w \in \text{cuenta} (w [n^\circ \text{cuenta}] = s [n^\circ \text{cuenta}] \wedge w [\text{nombre-sucursal}] = u [\text{nombre-sucursal}])))) \}$$

Definición formal

- Se dice que una variable tupla es una variable libre si no está cuantificada por \exists o \forall
- por ejemplo en: $t \in \text{préstamo} \wedge \exists s \in \text{cliente} (t[\text{nombre-sucursal}] = s[\text{nombre-sucursal}])$

t es libre y s es ligada

Cálculo relacional de tuplas

- Las fórmulas de cálculo relacional se construyen con átomos que tienen una de las formas siguientes:
- $s \in r$ donde s es una variable tupla y r es una relación
- $s[x] \theta u[y]$ donde s y u son variables tuplas, x es un atributo en el que está definido s y y es un atributo en el que está definido u y θ es un operador comparación (e.g. \geq)

Cálculo relacional de tuplas

- También es necesario que los atributos x e y tengan dominios cuyos miembros puedan compararse mediante θ
- $s [x] \theta c$ donde s es una variable tupla, x es un atributo en el que está definida s , θ es un operador comparación y c es una constante, en el dominio del atributo x

Cálculo relacional de tuplas

- Las fórmulas se construyen a partir de átomos, empleando las reglas:
 - un átomo es una fórmula
 - si P_1 es una fórmula, también lo son $\neg P_1$ y (P_1)
 - si P_1 y P_2 son fórmulas, también lo son $P_1 \cup P_2$, $P_1 \wedge P_2$, $P_1 \Rightarrow P_2$
 - si $P_1(s)$ es una fórmula que contiene la variable tupla libre s y r es una relación $\exists s \in r(P_1(s))$ y $\forall s \in r(P_1(s))$ también lo son

Cálculo relacional de tuplas

Seguridad de las expresiones

- Las expresiones del cálculo relacional de tuplas pueden generar relaciones infinitas por ejemplo

$$\{ t \mid \neg(t \in \text{préstamo}) \} \quad (1)$$

genera infinitas tuplas que no están en préstamo

- para que esto no ocurra se introduce el concepto de dominio, $\text{dom}(P)$, que incluye el conjunto de todos los valores a que P hace referencia

Cálculo relacional de tuplas

- Por ejemplo $\text{dom}(t \in \text{préstamo} \wedge t[\text{importe}] > 240.000)$ es el conjunto que contiene 240.000 y todos los valores que aparecen en préstamo
- Se dice que la expresión $\{t \mid p(t)\}$ es segura si todos los valores que aparecen en el resultado son valores de $\text{dom}(p)$

Cálculo relacional de tuplas

- por ejemplo la anterior expresión (1) no es segura pues $\text{dom}(\neg(t \in \text{préstamo}))$ es el conjunto de todos los valores de préstamo, sin embargo es posible tener una tupla t que no esté en préstamo que contenga valores que no aparecen en préstamo

Cálculo relacional de dominios

- Es una segunda versión del cálculo relacional
- Utiliza variables de dominio que toman sus valores en el dominio de un atributo en vez de tomarlas de una tupla completa

Definición formal

- Las expresiones son de la forma:

$$\{ \langle x_1, x_2, \dots, x_n \rangle \mid P(x_1, x_2, \dots, x_n) \}$$

donde x_1, x_2, \dots, x_n representan las variables de dominio y P representa una fórmula compuesta de átomos

Cálculo relacional de dominios

- Los átomos son de la forma:
 - $\langle x_1, x_2, \dots, x_n \rangle \in r$ donde r es una relación con n atributos y x_1, \dots, x_n son variables de dominio
 - $x \theta y$, x, y variables de dominio y θ un operador de comparación (con las exigencias usuales)
 - $x \theta c$ donde c es una constante

Cálculo relacional de dominios

- Las fórmulas se construyen a partir de los átomos con las reglas:
 - un átomo es una fórmula
 - si P_1 es una fórmula también lo es $\neg P_1$
 - si P_1 y P_2 son fórmulas, también lo son $P_1 \wedge P_2$, $P_1 \vee P_2$, $P_1 \Rightarrow P_2$
 - si $P_1(x)$ es una fórmula en x $\exists x P(x)$ y $\forall x P(x)$ también lo son

Cálculo relacional de dominios

Ejemplos

- Averiguar el nombre de la sucursal, el n° del préstamo y el importe de los préstamos mayores que 240.000 pts.

$\{ \langle b, l, a \rangle \mid \langle b, l, a \rangle \in \text{préstamo} \wedge a > 240.000 \}$

- Averiguar todos los n° de préstamo de los préstamos con importe superior a 240.000

$\{ \langle l \rangle \mid \exists b, a (\langle b, l, a \rangle \in \text{préstamo} \wedge a > 240.000) \}$

Cálculo relacional de dominios

- Averiguar el nombre de todos los clientes que tienen concedido un préstamo en la sucursal Navacerrada y averiguar el importe del préstamo

$$\{\langle c,a \rangle \mid \exists l (\langle c,l \rangle \in \text{prestatario} \wedge \exists b (\langle b,l,a \rangle \in \text{préstamo} \wedge b = \text{Navacerrada}))\}$$

Cálculo relacional de dominios

Seguridad de las expresiones

- Al igual que antes existen expresiones que no son seguras pues permiten valores del resultado que no están en el dominio de la expresión
- por ejemplo
 $\{ \langle b, l, a \rangle \mid \neg (\langle b, l, a \rangle \in \text{préstamo}) \}$ no es segura
- Hay que añadir reglas adicionales para que las expresiones sean seguras en el cálculo relacional de dominios (no lo veremos)

Cálculo relacional de dominios

Potencia expresiva de los lenguajes.

- Restringidos a las expresiones seguras son equivalentes:
 - el álgebra relacional
 - el cálculo relacional de tuplas
 - el cálculo relacional de dominios

Operaciones del álgebra relacional extendida

Proyección generalizada

- Es una proyección en la que se utilizan funciones aritméticas en la lista de proyección

$$\pi_{F_1, F_2, F_3, \dots, F_n}(E)$$

donde E es una expresión del álgebra relacional y F_1, F_2, \dots, F_n son expresiones aritméticas que incluyen constantes y atributos en el esquema de E

Operaciones del álgebra relacional extendida

- Ejemplo, relación información-préstamo

Nombre - cliente	límite	Saldo - préstamo
Santos	1.200.000	140.000
Gómez	400.000	80.000

Si queremos averiguar el importe disponible podemos hacer:

daría $\pi_{\text{nombre-cliente, límite-saldo préstamo}}(\text{información-préstamo})$

Santos	1.060.000
Gómez	320.000

Operaciones del álgebra relacional extendida

- Es una ampliación de la operación reunión y sirve para trabajar con información que falta
- Ejemplo: dadas las relaciones *empleado*

nombre-empleado

Segura

Domínguez

Gómez

Valdivieso

calle

Tebeo

Viaducto

Bailén

Fuencarral

ciudad

La Loma

Villaconejos

Alcorcón

Móstoles

Operaciones del álgebra relacional extendida

Trabajo a tiempo completo

Nombre empleado	nombre sucursal	sueldo
Segura	Majadahonda	230.000
Domínguez	Majadahonda	200.000
Barea	Fuenlabrada	800.000
Valdivieso	Fuenlabrada	230.000

Si hiciésemos empleado $\triangleright \triangleleft$ trabajo-a-tiempo-completo tendríamos:

Segura	Tebeo	La Loma	Majadahonda	230.000
Domínguez	Viaducto	Villaconejos	Majadahonda	260.000
Valdivieso	Fuencarral	Móstoles	Fuenlabrada	230.000

Es decir, se pierden tuplas

Operaciones del álgebra relacional extendida

- La reunión externa por la izquierda tiene todas las tuplas de la relación de la izquierda que no coinciden con ninguna tupla de la derecha y las rellena con valores nulos y los añade a la reunión natural
- Se nota por $\leftarrow \bowtie$

(nota para el alumno: Silberchatz emplea una notación diferente)

Operaciones del álgebra relacional extendida

- Ejemplo: empleado $\leftarrow \bowtie$ trabajo-a-tiempo-completo

n.empleado	calle	ciudad	n.sucursal	saldo
Segura	Tebeo	La Loma	Majadahonda	230.000
Domínguez	Viaducto	Villaconejos	"	260.000
Valdivieso	Fuencarral	Móstoles	Fuenlabrada	80.000
Gómez	Bailén	Alcorcón	Nulo	Nulo

- similar ocurre con reunión externa por la dcha ($\bowtie \rightarrow$) y reunión externa completa ($\leftarrow \bowtie \rightarrow$)

Operaciones del álgebra relacional extendida

Funciones de agregación

- Son funciones que toman una colección de valores y devuelven un único valor
- Por ejemplo la función *sum* devuelve la suma de los valores

$$\text{Sum} < 1, 1, 3, 4, 4, 11 > = 24$$

- *Avg* devuelve la media
- *Cont* devuelve el número de elementos

Operaciones del álgebra relacional extendida

- *Min* y *max* devuelven mínimo y máximo, respectivamente
- Ejemplo de uso: queremos conocer la suma de los sueldos de los empleados

Sum sueldo (trabajo-por-horas)

- Si queremos que no tenga en cuenta valores repetidos utilizaremos la función: *distinct*
- P.e si queremos saber el n° de sucursales hacemos:
count-distinct nombre-sucursal (trabajo-por-horas)

Operaciones del álgebra relacional extendida

- Si queremos que la agregación se haga no para todas las tuplas sino por grupos, utilizaremos el operador de agregación G
- Ejemplo: queremos la suma de los sueldos de los empleados por sucursales
- nombre-sucursal G (trabajo-por-horas) produce

Operaciones del álgebra relacional extendida

n.empleado	n.sucursal	sueldo
González	centro
Díaz	centro
Jiménez	centro
Catalán	Leganés
Fernández	Leganés

Operaciones del álgebra relacional extendida

- Si ahora queremos la suma de sueldos por sucursales:

nombre-sucursal \mathbf{G} sum sueldo (trabajo-por-horas)

- Si además quisiéramos el máximo

nombre-sucursal \mathbf{G} sueldo,max sueldo (trabajo-por-horas)

Modificación de la base de datos

Borrado

- Solo se pueden borrar tuplas enteras, no atributos
- Un borrado se expresa mediante
 $r \leftarrow r - E$ E es una con....
- Ejemplo: borrar todas las cuentas de Gómez
 $\text{cuenta} \leftarrow \text{cuenta} - \sigma_{\text{nombre-cliente} = \text{“Gómez”}}(\text{cuenta})$
- Ejemplo: borrar los préstamos con importes entre 0 y 8.000
 $\text{préstamo} \leftarrow \text{préstamo} - \sigma_{\text{importe} \geq 0 \text{ and } \text{importe} \leq 8.000}(\text{préstamo})$

Modificación de la base de datos

Inserción

- Para insertar datos hay que especificar la tupla que se va a insertar o escribir una consulta
- Las tuplas deben ser de la aridad correcta y con dominios compatibles
- Las inserciones se expresan

$$r \leftarrow r \cup E$$

Modificación de la base de datos

- Ejemplo: insertar una tupla con Gómez con la cuenta c-973 en Navacerrada y con saldo 240.000 pts.

$Cuenta \leftarrow cuenta \cup \{ \text{"Navacerrada", c-973, 240.000} \}$
 $impositor \leftarrow impositor \cup \{ (\text{"Gómez", c-973}) \}$

Actualización

- Para modificar el valor de una tupla sin modificar todos los valores de la tupla se puede emplear el operador de proyección generalizada

$$r \leftarrow \pi_{F_1, F_2, \dots, F_n} (r)$$

- Ejemplo: pagamos intereses y queremos aumentar todos los saldos un 5%

$$\begin{aligned} \text{cuenta} &\leftarrow \pi_{\text{nombre-sucursal}, n^\circ \text{ cuenta}} \\ \text{saldo} &\leftarrow \text{saldo} * 1.05 (\text{cuenta}) \end{aligned}$$

Actualización

- Si queremos seleccionar varias tuplas y solo actualizar esas

$$r \leftarrow \pi_{F_1, F_2, \dots, F_n} (\sigma_p(r)) \cup (r - \sigma_p(r))$$

- Ejemplo: a las cuentas con mas de 2 millones las pagamos el 6%, al resto el 5%

$$\begin{aligned} \text{cuenta} \leftarrow & \pi_{NS, NC, saldo \leftarrow saldo * 1.06} (\sigma_{saldo \geq 2.000.000}(\text{cuenta})) \cup \pi \\ & NS, NC, saldo \leftarrow saldo * 1.05 (\sigma_{saldo < 2.000.000}(\text{cuenta})) \end{aligned}$$

Vistas

- En los ejemplos anteriores hemos actuado a nivel de modelo lógico, sin embargo puede ser interesante no actuar a este nivel sino crear relaciones virtuales que se muestren al usuario, a estas relaciones se las denomina *vistas*.
- Las vistas se definen empleando la instrucción *create view*.

Vistas

- Hay que dar un nombre a la vista (V), así que la expresión habitual es:

create view V as <expresión de consulta>

- Por ejemplo, podemos querer ver las sucursales y todos sus clientes y denominar a esta vista *todos-los-clientes*, entonces hacemos:

create view *todos los clientes* as

π nombre-sucursal, nombre-cliente (impositor ▷◁ cuenta) \cup

π nombre-sucursal, nombre-cliente (prestatario ▷◁ préstamo)

Vistas

- Una vez definida la vista se puede hacer referencia a la misma, por ejemplo, podemos querer ver los nombres de todos los clientes de la sucursal de Navacerrada:

π nombre-cliente (σ nombre-sucursal="Navacerrada" (*todos-los-clientes*))

Vistas

- Las vistas son distintas de la operación asignación vista anteriormente, por ejemplo, podíamos haber hecho

$r_1 \leftarrow \pi_{\text{nombre-sucursal, nombre-cliente}}$ (impositor cuenta)

$\cup \pi_{\text{nombre-sucursal, nombre-cliente}}$ (prestatarario prestamo)

- en este caso, la asignación se evalúa sólo una vez y no cambiará si cambian *impositor*, *cuenta*, etc., sin embargo la vista *todos-los-clientes* **sí** cambiará

Vistas

- Algunos sistemas permiten guardar las vistas, de manera que cuando se actualiza alguna de las relaciones también se actualiza la vista, estas se llaman *vistas materializadas*.
- Las vistas tienen algunos problemas si con ellas se emplean actualizaciones, inserciones o borrados. Por ejemplo, definamos la vista *préstamo-sucursal*:

create view *préstamo-sucursal* as π nombre-sucursal, número-préstamo (*préstamo*)

Vistas

- Sería posible hacer la inserción directa en préstamo-sucursal
 - préstamo-sucursal \leftarrow préstamo-sucursal \cup {"Navacerrada", P-37}
- Esta inserción se debe hacer en *préstamo*, que es la relación a partir de la cual se genera la vista. Sin embargo para insertarla hace falta un valor para *importe*. Caben dos posibilidades:
 - rechazar la inserción
 - insertar la tupla ("Navacerrada", P-37, nulo) en la relación *préstamo*
- Por ello, generalmente no se permiten modificaciones en las relaciones de vistas.

Vistas

- También se pueden emplear vistas en la expresión que define otra vista, por ejemplo:

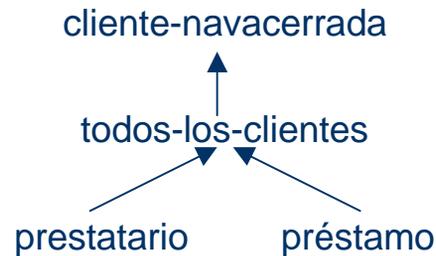
create view cliente-navacerrada **as**

π nombre-cliente (σ nombre-sucursal="Navacerrada" (*todos-los-clientes*))

- Se dice que una relación de vistas v_1 *depende directamente* de otra v_2 si v_2 se utiliza en la expresión que define v_1 .

Vistas

- Las dependencias entre vistas se presentan mediante *grafos de dependencia*, de la forma:



- Se dice que una relación de vistas v_1 *depende* de otra v_2 si hay un camino en el grafo de dependencia de v_2 a v_1 .

Vistas

- Se dice que una relación de vistas es *recursiva* si depende de sí misma, por ahora emplearemos sólo vistas que no lo sean.
- La *expansión de vistas* es una manera de ver vistas definidas en términos de otras vistas. Las relaciones de vistas sustituyen a las expresiones que definen las vistas y por tanto se pueden sustituir por las expresiones que las definen.

Vistas

- Si se modifica una expresión sustituyendo la relación de vistas por su definición, la nueva relación puede seguir conteniendo otras relaciones de vistas que pueden también ser expandidas.
- Por tanto, la expansión de vistas puede ser definida:
repeat
 buscar todas las relaciones de vistas v_i de e_i
 sustituir la relación de vistas v_i por la expresión que define v_i
until no queden mas vistas en e_i

Vistas

- Por ejemplo:

$\sigma_{\text{nombre-cliente}=\text{"Martín"}}(\text{cliente-navacerrada})$

- Puede expandirse:

$\sigma_{\text{nombre-cliente}=\text{"Martín"}}(\pi_{\text{nombre-cliente}}(\sigma_{\text{nombre-sucursal}=\text{"Navacerrada"}}(\text{todos-los-clientes})))$

- O expandirse más todavía:

$\sigma_{\text{nombre-cliente}=\text{"Martín"}}(\pi_{\text{nombre-cliente}}(\sigma_{\text{nombre-sucursal}=\text{"Navacerrada"}}(\pi_{\text{nombre-sucursal, nombre-cliente}}(\text{impositor cuenta}))))$
 $\cup \pi_{\text{nombre-sucursal, nombre-cliente}}(\text{prestatarario prestamo}))$

Bases de Datos

Tema 4: SQL

Indice

- 4.2 Operaciones sobre conjuntos
- 4.3 Funciones de agregación
- 4.4 Valores nulos
- 4.5 Subconsultas anidadas
- 4.6 Modificación de la Base de Datos

Referencias

- Silberschatz et al., 1998, pp. 81-110

Estructura básica

- SQL proviene de Structured Query Language, data de principios de los 70
- El lenguaje consta básicamente de expresiones de álgebra relacional
- La estructura básica de una expresión SQL consta de tres cláusulas: *select*, *from* y *where*.

Estructura básica

- La cláusula *select* equivale a la operación proyección del álgebra relacional
- La cláusula *from* equivale a la operación producto cartesiano del álgebra relacional
- La cláusula *where* equivale a un predicado selección del álgebra relacional

Estructura básica

- Una consulta típica en SQL tiene la estructura:

select A_1, A_2, \dots, A_n (A_i atributos)
from r_1, r_2, \dots, r_m (r_i relaciones)
where P (P predicado)

- es equivalente a

$\Pi_{A_1, A_2, \dots, A_n}(\sigma_P(r_1 \times r_2 \times \dots \times r_m))$

Estructura básica

- A diferencia del álgebra relacional, el resultado de una consulta en SQL puede contener varias tuplas iguales

Cláusula select

- Es distinta de la operación de selección del álgebra relacional en el sentido de que puede producir tuplas repetidas, esto es debido a que eliminarlas es computacionalmente muy costoso

Estructura básica

- Si se quiere forzar la eliminación (que actúa por defecto) se pone **distinct**:

```
select distinct A1,A2,...,An  
from r1,r2,...,rm  
where P
```

- Si se quiere especificar que no se eliminen se pone **all**:

```
select all A1,A2,...,An  
from r1,r2,...,rm  
where P
```

Estructura básica

- Se puede emplear "*" para denotar todos los atributos, por ejemplo **select ***
- No se puede emplear **distinct** con **count(*)**
- También se pueden incluir operaciones elementales, por ejemplo **select A₁*1000**

Estructura básica

Cláusula where

- Se pueden emplear, desigualdades y conectivas lógicas **and**, **or** y **not**, por ejemplo

```
select número-préstamo  
from préstamo  
where importe >=600000
```

- Existe un operador comparación **between** que simplifica las consultas, por ejemplo

```
select número-préstamo  
from préstamo  
where importe between 600000 and 300000
```

Estructura básica

Cláusula from

- Es muy fácil expresar una reunión natural en SQL:

Π nombre-cliente, número-préstamo (prestatario \triangleright \triangleleft préstamo)

select distinct nombre-cliente, prestatario.número-préstamo

from prestatario, préstamo

where prestatario.número-préstamo= préstamo.número-préstamo

Estructura básica

Operación renombramiento

- Se emplea la cláusula **as** que se puede poner tanto en **select** como en **from**:

select distinct nombre-cliente, prestatario.número-préstamo **as** id-préstamo

from prestatario, préstamo

where prestatario.número-préstamo=
préstamo.número-préstamo **and** nombre-
sucursal="Navacerrada"

Estructura básica

Variables tupla

- Son muy útiles para comparar tuplas de la misma relación.
- Las variables tupla se definen en la cláusula **from** mediante el uso de la cláusula **as**, por ejemplo: “obtener los nombres de todas las sucursales que poseen un activo mayor que al menos una sucursal situada en Barcelona”

```
select distinct T. Nombre-sucursal  
from sucursal as T, sucursal as S  
where T.activo>S.activo and S.ciudad-  
sucursal=“Barcelona”
```

Estructura básica

Operaciones sobre cadenas

- Se pueden encajar patrones de una cadena mediante el operador **like**.
 - El (%) encaja con cualquier subcadena
 - El (_) encaja con cualquier carácter
- Por ejemplo, “Nava%” encaja con cualquier subcadena que empiece por “Nava”
- “%cer%” encaja con cualquier subcadena que incluya “cer”
- “___” encaja con cualquier cadena de tres caracteres
- “___%” encaja con cualquier cadena de al menos tres caracteres

Estructura básica

- Para que se puedan incluir en el encaje los caracteres especiales se emplea la palabra **escape**
 - **like** “ab\%cd%” **escape** “\”, encaja con todas las cadenas que empiezan con “ab%cd”
 - **like** “ab\\cd%” **escape** “\”, encaja con todas las cadenas que empiezan con “ab\cd”
- Se pueden buscar discordancias con **not like**

Estructura básica

Orden de presentación de las tuplas

- Para que presenten en cierto orden se emplea **order by**, ascendente por defecto), si queremos descendente se emplea **desc**
- Si hay varias tuplas con igual valor del atributo se pueden elegir distintos subórdenes, por ejemplo “listar por importe descendente y en caso de igualdad por número de préstamo”:

```
select *
```

```
from préstamo
```

```
order by importe desc, número préstamo asc
```

Operaciones sobre conjuntos

- ***Union***, ***intersect*** y ***except*** corresponden a las operaciones del álgebra relacional \cup , \cap y $-$
- Las relaciones involucradas han de ser compatibles.

Operación *union*

- por ejemplo Clientes con cuenta o préstamo o ambos:

```
(select nombre-cliente
from impositor)
union
(select nombre-cliente
from prestatario)
```

Operaciones sobre conjuntos

- A diferencia de **select**, **union** elimina duplicados, si se quieren conservar se emplea **union all**.

Operación intersect

- por ejemplo, “clientes con cuenta y préstamo a la vez”:

```
(select distinct nombre-cliente  
from impositor)
```

```
intersect
```

```
(select distinct nombre-cliente  
from prestatario)
```

Operaciones sobre conjuntos

- A diferencia de **select**, **intersect** elimina duplicados, si se quieren conservar se emplea **intersect all**.

```
(select nombre-cliente  
from impositor)  
intersect all  
(select nombre-cliente  
from prestatario)
```

Operaciones sobre conjuntos

Operación except

- por ejemplo, “clientes con cuenta pero no con préstamo”

```
(select nombre-cliente  
from impositor)  
except all  
(select nombre-cliente  
from prestatario)
```

Funciones de agregación

- Hay cinco funciones de agregación primitivas
 - **media**, avg (número)
 - **mínimo**, min
 - **máximo**, max
 - **total**, sum (número)
 - **cuenta**, count
- por ejemplo “obtener el saldo medio de las cuentas de Navacerrada”

```
select avg (saldo)  
from cuenta  
where nombre-sucursal=“Navacerrada”
```

Funciones de agregación

- Se puede agrupar por conjunto de tuplas según el atributo o atributos especificados en **group by**:

```
select nombre-sucursal avg (saldo)  
from cuenta  
group by nombre-sucursal
```

- A veces queremos eliminar los duplicados, por ejemplo “obtener el número de impositores de cada sucursal”

```
select nombre-sucursal, count (distinct nombre-  
cliente)  
from impositor, cuenta  
where impositor.número-cuenta=cliente.número-  
cuenta  
group by nombre-sucursal
```

Funciones de agregación

- A veces es útil emplear una condición sobre los grupos, no sobre las tuplas, para esto se emplea la cláusula **having**

```
select nombre-sucursal avg (saldo)  
from cuenta  
group by nombre-sucursal  
having avg(saldo)>240000
```

Funciones de agregación

- Ejemplo: “obtener el saldo medio de cada cliente que vive en Madrid y tiene como mínimo tres cuentas”

```
select impositor.nombre-cliente, avg(saldo)
from impositor, cuenta, cliente
where impositor.número-cuenta=cuenta.número cuenta and
impositor.nombre-cliente=cliente.nombre-cliente and ciudad-
cliente=“madrid”
group by impositor.nombre-cliente
having count(distinct impositor.número-cuenta)>=3
```

Valores nulos

- En un predicado se puede emplear **null** para comprobar si un valor es nulo.
- Por ejemplo, “encontrar los números de préstamo con importes nulos”

```
select número-préstamo  
from prestamo  
where importe is null
```

- El predicado **is not null** se emplea para preguntar por la ausencia de un valor nulo
- El resultado de cualquier comparación que involucre un valor nulo es falso.

Valores nulos

- El resultado de cualquier operación aritmética es nulo si cualquiera de los valores de entrada lo es.
- Para las operaciones de agregación, todas ellas excepto **count** ignoran los valores nulos
- El cálculo de **count** para una colección vacía es 0

Subconsultas anidadas

- SQL permite anidar consultas dentro de otras.
- Usualmente se emplean para efectuar comprobaciones sobre pertenencia a conjuntos, comparación de conjuntos y cardinalidad de conjuntos

Pertenencia a conjuntos

- Se puede emplear la conectiva **in** para comprobar la pertenencia a un conjunto.
- La no pertenencia a un conjunto se comprueba con **not in**.

Subconsultas anidadas

- Si queremos encontrar, de otra forma, los clientes de un banco que tienen un préstamo y una cuenta en el banco, podemos hacer

```
select distinct nombre-cliente  
from prestatario  
where nombre-cliente in (select nombre-cliente  
                           from impositor)
```

Subconsultas anidadas

- Si queremos encontrar, los clientes de un banco que tienen un préstamo pero no una cuenta en el banco, podemos hacer

```
select distinct nombre-cliente  
from prestatario  
where nombre-cliente not in (select nombre-  
cliente  
from impositor)
```

Subconsultas anidadas

Comparación de conjuntos

- Se puede emplear una forma alternativa a la anteriormente vista con la expresión **some**, antes teníamos:

```
select distinct T. Nombre-sucursal
from sucursal as T, sucursal as S
where T.activo>S.activo and S.ciudad-sucursal="Barcelona"
```

- Podemos hacer la misma consulta con

```
select nombre-sucursal
from sucursal
where activo> some (select activo
                    from sucursal
                    where ciudad-sucursal="Barcelona")
```

Subconsultas anidadas

- También podemos emplear $>$, $<$, $>=$, $<=$, $=$ y $<>$.
- Finalmente, también existe la expresión **all** que corresponde con “a todas” (por ejemplo $>$ **all**, superior a todas).

- por ejemplo

```
select nombre-sucursal
from sucursal
where activo  $>$  all (select activo
                    from sucursal
                    where ciudad-sucursal="Barcelona")
```

- Con **all** también podemos emplear $>$, $<$, $>=$, $<=$, $=$ y $<>$.

Subconsultas anidadas

Comprobación de relaciones vacías

- La constructora **exist** devuelve el valor cierto si la subconsulta argumento es no vacía.
- por ejemplo “obtener los clientes que tienen tanto una cuenta como un préstamo”

```
Select nombre-cliente
From préstamo
Where exist (select *
               from impositor
               where impositor.nombre-cliente=
                    prestatario.nombre-cliente)
```

Subconsultas anidadas

- Utilizando **not exist** se puede comprobar la no existencia de tuplas en el resultado de una consulta.
- También se puede expresar que la relación A contiene a la relación B como “**not exist (B except A**

Subconsultas anidadas

- Por ejemplo, “obtener todos los clientes que tienen una cuenta en todas las sucursales de Barcelona”

```
Select distinct S.nombre-cliente  
From impositor a s S  
Where not exist
```

```
((select nombre-sucursal  
From sucursal  
Where ciudad-sucursal="Barcelona")  
Except  
(select R. Nombre-sucursal  
From impositor as T, cuenta as R  
Where T.numero-cuenta = R.numero-cuenta and  
S.nombre-cliente = T.nombre-cliente))
```

Sucursales de Barcelona (B)

Sucursales en las que S.nombre-cliente tiene una cuenta (A)

Subconsultas anidadas

Comprobación de tuplas duplicadas

- SQL permite ver si una consulta ofrece duplas duplicadas mediante la constructora **unique (not unique)**.
- **Unique** devuelve el valor cierto si la subconsulta que se le pasa como argumento no produce tuplas duplicadas.

Subconsultas anidadas

- Por ejemplo, “obtener todos los clientes que tienen sólo una cuenta en la sucursal de nombre Navacerrada”

```
select T.nombre-cliente
from impositor as T
where unique (select R.nombre-cliente
from cuenta, impositor as R
where T.nombre-cliente=R.nombre-cliente
and R.número cuenta=cuenta.número cuenta
and cuenta.nombre-sucursal=“Navacerrada”)
```

Modificación de la Base de Datos

Borrado

- Se expresa igual que una consulta
- Se pueden borrar sólo tuplas completas, no valores de atributos
- Se realiza con **delete**, por ejemplo
delete from r
where P
- **Delete** opera sobre una sola relación, si hay que borrar en varias hay que utilizar varias órdenes **delete**.

Modificación de la Base de Datos

- Por ejemplo, “borrar todas las cuentas de la sucursal Navacerrada”

Delete from cuenta

Where nombre-sucursal **in** (**select** nombre-sucursal

From sucursal

Where ciudad-sucursal=“Navacerrada”)

- Por ejemplo, “borrar todas las cuentas con saldos inferiores a la media del banco”

Delete from cuenta

Where saldo < (**select avg** (saldo) nombre-sucursal

From cuenta)

Modificación de la Base de Datos

Inserción

- Para insertar datos podemos especificar una tupla completa o bien una fórmula cuya consulta sea el cjto. de tuplas a insertar.
- Se hace con la instrucción **insert**, por ejemplo
Insert into cuenta
Values (“Navacerrada”, “C-9732”, 240000)
- Si no recordamos el orden de los atributos:
Insert into cuenta(nombre-sucursal,número-cuenta,saldo)
Values (“Navacerrada”, “C-9732”, 240000)

Modificación de la Base de Datos

- Antes de llevar a cabo ninguna inserción es importante que haya finalizado la evaluación de la orden **select**.

- por ejemplo

```
Insert into cuenta  
           select *  
           from cuenta
```

- Podría insertar un número infinito de tuplas
- Si desconocemos algún valor podemos hacer

```
Insert into cuenta  
           Values (null, "C-9732", 240000)
```

Modificación de la Base de Datos

Actualizaciones

- En ocasiones queremos cambiar algunos valores dentro de una tupla sin cambiar todos los valores de la misma.
- En este tipo de situaciones se emplea **update**
- por ejemplo Si queremos incrementar los saldos el 5%:

Update cuenta

Set saldo=saldo*1.05

Modificación de la Base de Datos

- Otro ejemplo: pagar mas a las cuentas superiores a 2M:

```
Update cuenta  
Set saldo=saldo*1.06  
Where saldo>2.000.000
```

```
Update cuenta  
Set saldo=saldo*1.05  
Where saldo<2.000.000
```

- El orden es muy importante: ejercicio, ¿cuál es el interés de una cuenta de 1.990.000 pts.? ¿y si se invierte el orden?)

Modificación de la Base de Datos

Actualización de vistas

- La anomalía vista para Algebra relacional también se produce en SQL.
- Supongamos al siguiente definición de vista:
create view préstamo-sucursal **as**
select nombre-sucursal, número-préstamo
from préstamo
- podríamos querer hacer
insert into préstamo-sucursal
values (“navacerrada”, “P-307”)

Modificación de la Base de Datos

- Sin embargo, esta inserción obligaría e introducir un importe nulo en la relación préstamo, lo cual produce problemas.
- Por tanto, en SQL sólo se permiten modificaciones de vistas si la vista se define en términos de una relación real de la BD, es decir, al nivel lógico.

Tema 5: Ligaduras de Integridad

Bases de Datos

Introducción

- En este tema se ataca el concepto de Ligadura de Integridad.
- Las Ligaduras de Integridad, en sus diferentes tipologías, son la base fundamental para un correcto diseño de una Base de Datos.
- Los distintos tipos de Ligaduras de Integridad cubren aspectos tanto dentro de una relación r dada, como entre un conjunto de varias relaciones R .

Ligadura de Integridad

- Definición:

Las ligaduras de integridad proporcionan un medio de **asegurar** que las modificaciones hechas a la base de datos por los usuarios **no provoquen una pérdida de la consistencia** de los datos.

- Las ligadura de integridad son predicados arbitrarios para las relaciones de la base de datos.

Tema 6: Diseño de Bases de Datos Relacionales (Parte 1)

Bases de Datos

Diapositiva resumen

- Introducción
- Problemas del Modelo E/R
- Ejemplo de diseño inadecuado
- Problemas del ejemplo
- Definiciones formales de la Normalización de Esquemas E/R
- Conservar la información
- Formas Normales

Diapositiva resumen (cont.)

- 1ª Forma Normal
- 2ª Forma Normal
- 3ª Forma Normal
- Forma Normal Boyce-Codd
- Resumen de las Formas Normales
- Descomposición en Proyecciones Independientes
- Proceso de Descomposición

Introducción

- En este tema se profundiza en el concepto de la Normalización de Bases de Datos.
- La Normalización de Bases de Datos es una formalización del diseño lógico de las bases de datos relacionales, lo que permite disponer de instrumentos algorítmicos de ayuda al diseño,
- La Normalización hace frente a las carencias de implementación del modelo E/R en el mundo informático.

Tema 6: Diseño de Bases de Datos Relacionales (Parte 2)

Bases de Datos

Diapositiva resumen

- Introducción
- Dependencias Multivaloradas
- 4^a Forma Normal
- Dependencias de Reunión
- 5^a Forma Normal
- Forma Normal de Clave de Dominios

Tema 7: Almacenamiento y Estructura de Archivos

Bases de Datos

Introducción

- En temas anteriores se ha estudiado cómo se organizan las bases de datos desde un punto de vista lógico, a través del modelo Entidad/Relación y a través de modelo Relacional.
- Es este tema se van a tratar aspecto sobre el almacenamiento de esas bases de datos y sus fundamentos, concentrando el esfuerzo en los distintos tipos de organización, tanto interna como externa, de los archivos utilizados.

Visión general de los medios de almacenamiento

- Distintos tipos de memoria:
 - Caché: La más rápida junto con los registros. Pequeño tamaño y control hardware.
 - Memoria principal: medio de almacenamiento donde se ejecutan los programas. No suele ser lo suficientemente grande para guardar toda una base de datos
 - Memoria flash: Memoria de sólo lectura, que si bien se puede escribir, es escritura es muy lenta.
 - Discos Magnéticos: Principal medio de almacenamiento. Hace las veces de disco swaping con la memoria principal.
 - Almacenamiento Óptico: CD-ROM normalmente, sirven como copia de respaldo, DVD (4-17Gbs)
 - Almacenamiento en cinta: Cintas de gran capacidad (80 Gb - 1Tb). Copias de respaldo.

Tema 8: Indexación y Asociación

Bases de Datos

Introducción

- En el tema anterior se estudió el almacenamiento de la información de una base de datos relacional en archivos, con especial interés en los distintos métodos de almacenamiento
- En este tema se van a tratar sobre la indexación y asociación de índices con los registros almacenados en la base de datos, para poder aumentar el rendimiento global en la ejecución de consultas sobre la misma, así como su rendimiento en inserciones, actualizaciones y borrados de datos.

Referencias

- Silberschatz 3^aEd.
pp 249-279.

Conceptos básicos

- Índices:
 - Índices Ordenados: Basados en la disposición ordenada de valores
 - Índices Asociativos (hash): Distribución uniforme de valores por funciones hash entre cajones (buckets)
- Consideraciones sobre los índices
 - Tipos de acceso: Búsqueda única, múltiple, inserción, borrado
 - Tiempo de acceso
 - Tiempo de inserción
 - Tiempo de borrado
 - Espacio adicional requerido por el índice

