ALGEBRA y CÁLCULO RELACIONAL

Objetivos

- Aprender a construir consultas a Bases de Datos relacionales utilizando el Álgebra Relacional.

Introducción

- El Álgebra y el Cálculo Relacional constituyen lenguajes formales asociados con el Modelo Relacional.
- Informalmente, el Álgebra Relacional (AR) es un lenguaje de procedimientos (*procedural language*) de alto nivel, (se indica qué y cómo obtenerlo) mientras que el Cálculo Relacional (CR) no es un lenguaje orientado a procedimientos. (se indica qué pero no cómo obtenerlo)
- Formalmente son equivalentes el uno con el otro en poder expresivo.
- Un lenguaje que produce una relación que puede derivarse utilizando CR es completo relacionalmente.

Álgebra Relacional

- Es un lenguaje de consulta procedural.
- Consta de un conjunto de operaciones que toman como entrada una o dos relaciones y producen como resultado una nueva relación sin cambiar las relaciones originales, por lo tanto, es posible anidar y combinar operadores.
- Tanto las relaciones que actúan como operandos como la relación resultante a la salida pueden emplearse como entradas para otra operación.
- Permite, como la aritmética, que se aniden expresiones. Esta propiedad recibe el nombre de clausura.
- Lenguajes de Usuario: SQL (Structured Query Language), basado en AR.

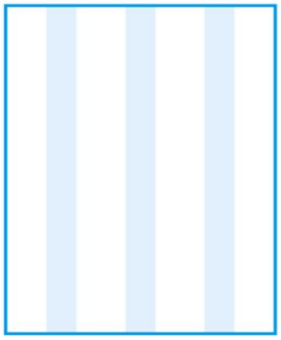
Álgebra Relacional

Hay varios operadores en el AR que construyen relaciones y manipulan datos:

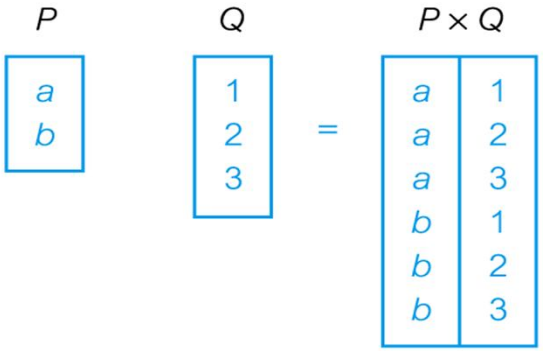
- Operaciones unarias:
  - Selección (ó Restricción)
  - Proyección
- Operaciones de conjuntos:
  - Unión
  - Diferencia
  - Intersección
  - Producto cartesiano
- Operaciones de Combinación
- Operación de División
- Operaciones de Agregación y Agrupamiento

Operaciones

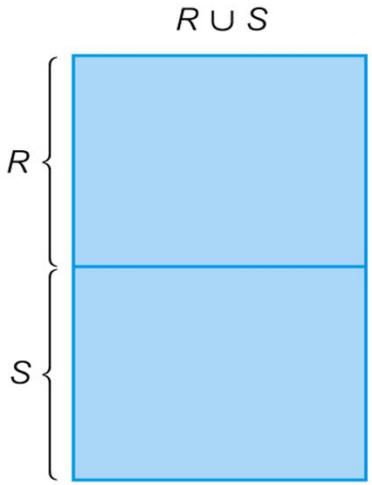
(a) Selection



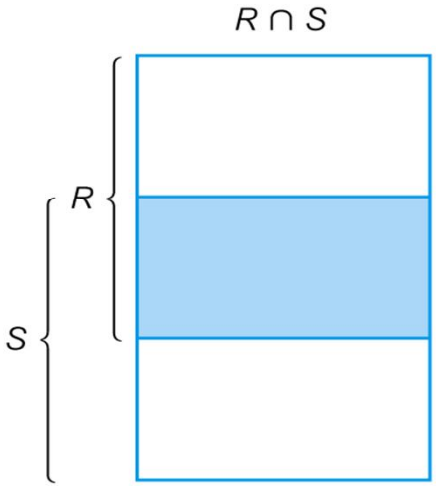
(b) Projection



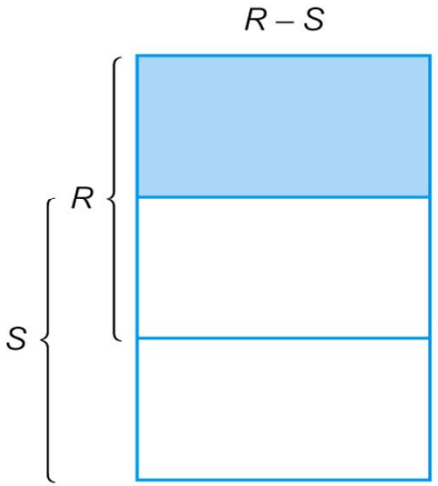
(c) Cartesian product



(d) Union

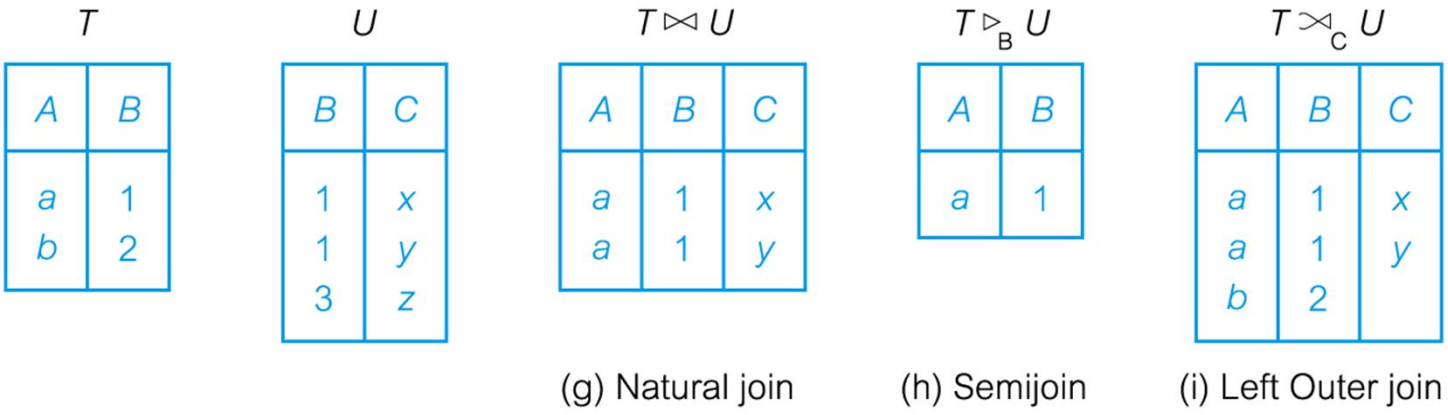


(e) Intersection



(f) Set difference

Operaciones



T

A	*B*
a	1
b	2

U

B	*C*
1	*x*
1	*y*
3	*z*

$T \bowtie U$

A	*B*	*C*
a	1	*x*
a	1	*y*

(g) Natural join

$T \triangleright_B U$

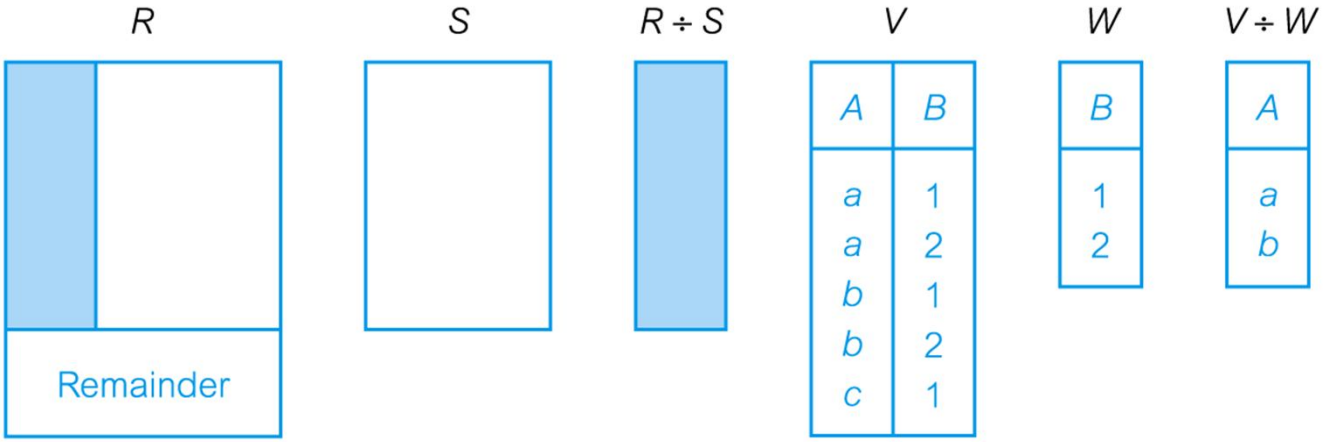
A	*B*
a	1

(h) Semijoin

$T \rtimes_C U$

A	*B*	*C*
a	1	*x*
a	1	*y*
b	2	

(i) Left Outer join



(j) Divis on (shaded area)

V

A	*B*
a	1
a	2
b	1
b	2
c	1

W

B
1
2

$V \div W$

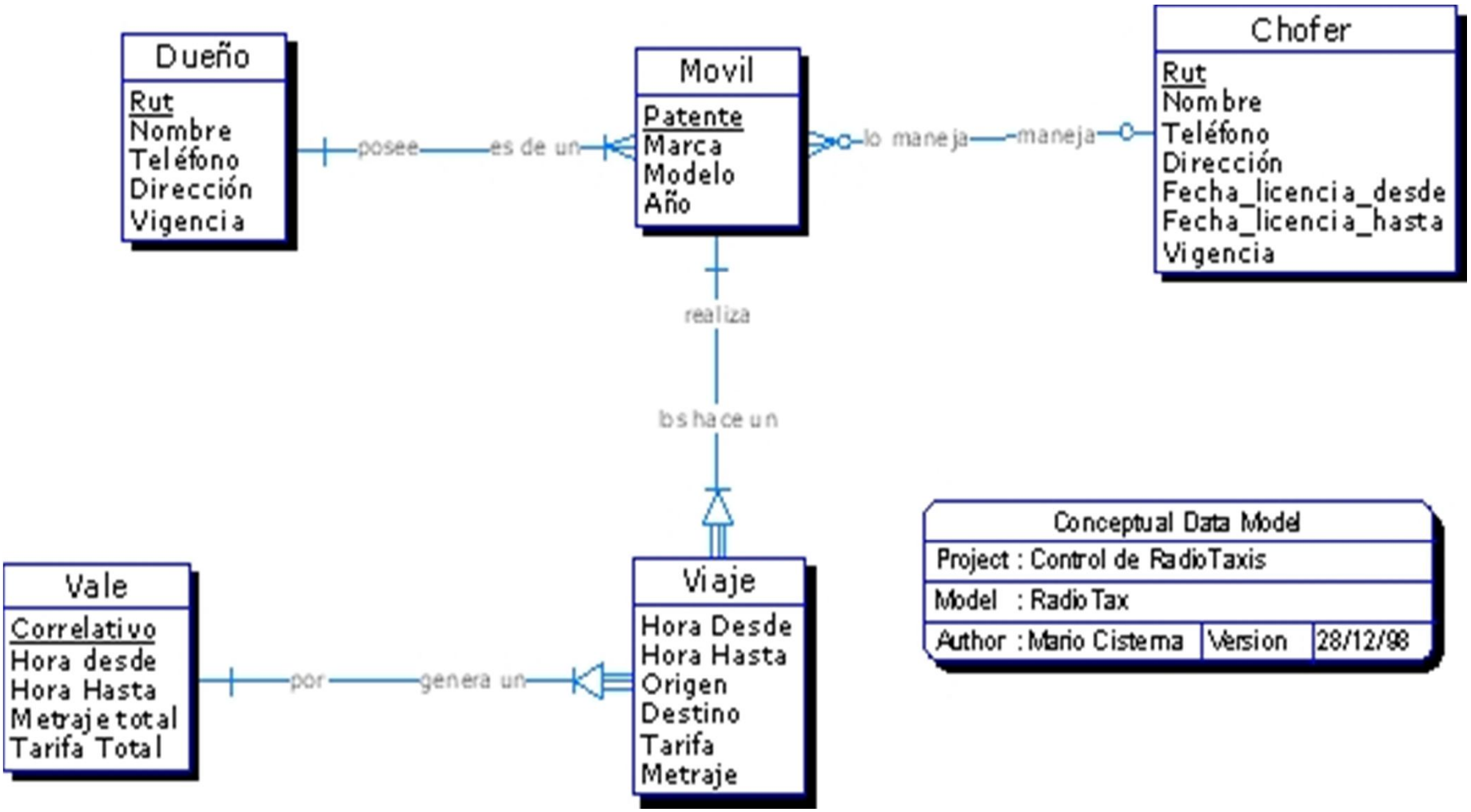
A
a
b

Example of division

Álgebra Relacional

- Las operaciones de proyección, producto, unión, diferencia, y selección son llamadas básicas ó primitivas, puesto que las otras operaciones pueden definirse en términos de éstas.
- Se hace necesario en este punto incluir un modelo de datos de ejemplo en el cual trabajar para generar ejemplos de comandos y operadores. Para este efecto se incluye un modelo básico de administración de Radio taxis.
- El Gráfico que se presenta a continuación representa el Modelo conceptual (Modelo Lógico) o Diagrama de Entidad-Relación:

Álgebra Relacional



Álgebra Relacional

Los Esquemas de relaciones que se pueden construir a partir de este modelo son los siguientes:

- ***Dueño = {rut, nombre, teléfono, dirección, vigencia}***
- ***Chofer = {rut, nombre, teléfono, dirección, fecha_licencia_desde, fecha_licencia_hasta, vigencia}***
- ***Vale = {correlativo, hora_desde, hora_hasta, metraje_total, tarifa_total}***
- ***Móvil = {patente, rut_dueño, rut_chofer, marca, modelo, año}***
- ***Viaje = {correlativo_vale, patente_movil, Hora_Desde, hora_hasta, origen, destino, tarifa, metraje}***

Selección ó Restricción (σ)

- El operador de selección opta por tuplas que satisfagan cierto predicado, se utiliza la letra griega sigma minúscula (**σ**) para señalar la selección.
- El predicado aparece como subíndice de **σ**.
- La Relación que constituye el argumento se da entre paréntesis después de la **σ**.

Ejemplos :

$$\sigma_{vigencia='S'}(Dueño)$$

$$\sigma_{patente='HL-8483}(Movil)$$

Proyección (Π)

- La operación de proyección permite quitar ciertos atributos de la relación.
- Esta operación es unaria, copiando su relación base dada como argumento y quitando ciertas columnas.
- La proyección se señala con la letra griega pi mayúscula (**Π**). Como subíndice de **Π** se coloca una lista de todos los atributos que se desea aparezcan en el resultado.
- La relación argumento se escribe después de **Π** entre paréntesis.
  Ejemplos :

$$\Pi_{rut,vigencia}(Chofer) \qquad \Pi_{nombre,direccion}(Dueño)$$

Unión (U)

- En álgebra relacional la unión de dos relaciones compatibles **A** y **B** es:

  A UNION B o A U B

  Produce el conjunto de todas las tuplas que pertenecen ya sea a **A** o a **B** o a Ambas.
- Al igual que en teoría de conjuntos el símbolo U representa aquí la unión de dos relaciones.

Ejemplo : $\sigma_{rut,vigencia}(Dueño) \cup \sigma_{rut,vigencia}(Chofer)$

Devuelve todos los Dueños y los Choferes.

Intersección (∩)

- En álgebra relacional la intersección de dos relaciones compatibles **A** y **B**
  A INTERSECCION B o A ∩ B
- Produce el conjunto de todas las tuplas pertenecientes a **A** y **B**. Al igual que en teoría de conjuntos el símbolo ∩ representa aquí la intersección entre dos relaciones.

Ejemplo: **Devuelve todos los dueños que también son choferes**

$$\sigma_{rut,vigencid}(Dueño) \cap \sigma_{rut,vigencid}(Chofer)$$

Diferencia (–)

- En álgebra relacional la diferencia entre dos relaciones compatibles **A** y **B**

  A MENOS B o A – B

  Produce el conjunto de todas las tuplas **t** que pertenecen a **A** y no pertenecen a **B**.

  Ejemplo: $\sigma_{rut,vigencid}(Dueño) - \sigma_{rut,vigencid}(Chofer)$

Devuelve todos los dueños que NO son choferes

Producto cartesiano (X)

- En álgebra relacional el producto de dos relaciones **A** y **B** es:

  A Veces B o A **X** B

  Produce el conjunto de todas las tuplas **t** tales que **t** es el encadenamiento de una tupla **a** perteneciente a **A** y de una **b** que pertenece a **B**. se utiliza el símbolo **X** para representar el producto.

  $$Dueño \times Movil$$

  Ejemplos:

  $$Movil \times Chofer$$

Join o Reunión

- Esta operación deriva del Producto Cartesiano
- Es equivalente a realizar una selección empleando un predicado de reunión como fórmula para elección dentro del producto cartesiano de las dos relaciones operando.
- Es difícil de implementar eficientemente en un Sistema de Manejo de Bases de Datos Relacionales (RDBMS) y es causa de problemas intrínsecos de *performance* en ellos.

Join o Reunión

- En álgebra relacional el JOIN entre el atributo **X** de la relación **A** con el atributo **Y** de la relación **B** produce el conjunto de todas las tuplas **t** tal que **t** es el encadenamiento de una tupla **a** perteneciente a **A** y una tupla **b** perteneciente a **B** que cumplen con el predicado:

  "A.X ***comp*** B.Y es verdadero"

  siendo ***comp*** un operador relacional y los atributos A.X y B.Y pertenecientes al mismo dominio.

Formas de la operación *Join*

- Combinación Theta (**θ-join**)
- Equicombinación (*Equi-join*) (tipo particular de Theta)
- Combinación Natural
- Reunión externa (*Outer join*)
- Semicombinación (*Semi-join*)

Join o Reunión Natural

- Si el operador relacional “comp” es “=” entonces el conjunto resultante es un EQUI-JOIN.
- Si se quita uno de éstos (usando una proyección) entonces el resultado es un JOIN-NATURAL.

Ejemplo.- $\sigma_{Dueño.rut=Movil.rut\_dueño}(Dueño \times Movil)$

Operaciones que extienden el Álgebra Relacional

- Proyección Generalizada
- Combinación externa
- Funciones de Agrupación y Agregación

Proyección Generalizada

- Extiende la operación de proyección permitiendo el uso de funciones aritméticas en la lista proyectada.

$$\Pi_{F1, F2, \ldots, Fn}(E)$$

- *E* es cualquier expresión del álgebra relacional
- Cada F_1, F_2, ..., F_n representa expresiones aritméticas que pueden involucrar constantes ó atributos en el esquema de *E*.
- Dada la relación *credit-info(customer-name, limit, credit-balance),* encontrar cuanto puede gastar una persona:

$$\Pi_{customer\text{-}name,\ limit - credit\text{-}balance}\ (credit\text{-}info)$$

Funciones y Operaciones de Agregación

- **La función de Agregación** toma una colección de valores y devuelve un único valor como resultado:

  avg: valor promedio
  min: valor mínimo
  max: valor máximo
  sum: suma de valores
  count: cantidad de valores

- **Operación de Agregar** en álgebra relacional

$$_{G1, G2, \ldots, Gn}\mathcal{g}_{F1(A1), F2(A2), \ldots, Fn(An)}(E)$$

  - E es cualquier expresión en álgebra relacional
  - $G_1, G_2 \ldots, G_n$ es una lista de atributos en los cuales agrupar (puede estar vacía)
  - Cada F_i es una función de agregación
  - Cada A_i es un nombre de atributo

Ejemplo de la Operación Agregación

- Relación *account* agrupada por *branch-name*:

branch-name	*account-number*	*balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

$$branch\text{-}name\ \mathcal{G}\ sum(balance)\ (account)$$

branch-name	*balance*
Perryridge	1300
Brighton	1500
Redwood	700

Funciones de Agregación (Cont.)

- El resultado de la agregación no tiene un nombre.

$$branch\text{-}name\ \mathcal{G}\ \textbf{\textit{sum}}(balance)\ \textbf{\textit{as}}\ sum\text{-}balance\ (account)$$

  - Puede utilizarse la operación de renombrar (*rename*) ***as*** para denominarla.
  - Por conveniencia, se permite renombrar como parte de la operación de agregación.

Combinación externa

- Extensión de la operación **join** que evita pérdidas de información, ya que trabaja con la información que falta.
- Calcula la combinación e incluye también en la relación resultante las tuplas de R que no tengan valores correspondientes en los atributos comunes de S.
- Asigna valores nulos (*null)* a los valores no existentes en la segunda relación.
  - *null* significa que el valor es desconocido ó no existe
  - Todas las comparaciones que involucran *null* son **false** por definición.

Ejemplo de Combinación externa

- Relación *loan*

loan-number	*branch-name*	*amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

- Relación *borrower*

customer-name	*loan-number*
Jones	L-170
Smith	L-230
Hayes	L-155

Ejemplo de Combinación externa

- **Combinación interna**

loan ⋈ *Borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

- **Combinación externa izquierda**

loan ⟕ *Borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

Ejemplo de Combinación externa

- **Combinación externa derecha:**

loan ⟖ *borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

- **Combinación externa completa**

loan ⟗ *borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

Valores nulos

- Es posible en las tuples tomar un valor nulo, denotado por *null*, para algunos atributos
- *null* significa un valor desconocido ó que no existe.
- El resultado de cualquier expresión aritmética donde participa *null* es el propio valor *null.*
- Las funciones de Agregación simplemente ignoran valores nulos
  - Es una decisión arbitraria.
- Para eliminación de duplicados y agrupamiento, el valor nulo es tratado como cualquier otro valor
  - Alternativa: asumir que cada valor nulo es diferente de los otros

Valores nulos

- Las comparaciones con valores nulos devuelven el valor especial verdadero *unknown*
  - Si se empleó *false* en vez de *unknown*, entonces *not (A < 5)* pudiera no ser equivalente a *A >= 5*
- Se emplea lógica tres estadoscon el valor *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown)* = *unknown*
  - AND: *(true* **and** *unknown)* = *unknown,*
    (false **and** *unknown)* = *false,*
    (unknown **and** *unknown)* = *unknown*
  - NOT*: (***not** *unknown)* = *unknown*
  - En SQL "*P* **is unknown"** evalúa como cierto si el predicado *P* evalúa a *unknown*
- El resultado del predicado de selección se trata como *false* si evalúa como *unknown*

Modificación de la Base de Datos

- El contenido de la base de datos puede modificarse utilizando las siguientes operaciones:
  - Borrado
  - Inserción
  - Actualización
- Estas operaciones se expresan empleando el operador de asignación: ←

Borrado

- Una solicitud de borrado se expresa de manera similar a una consulta, pero en vez de mostrar dichas tuplas al usuario, provoca que las tuplas seleccionadas sean eliminadas de la base de datos.
- Pueden eliminarse solamente tuplas completas; no pueden borrarse valores específicos de atributos.
- En álgebra relacional se expresa como:

$$r \leftarrow r - E$$

donde r es una relación y E es una consulta en álgebra relacional.

Ejemplos de borrado

- Borrar en la tabla account todas las tuplas cuyo nombre de sucursal sea Perryridge.

$$account \leftarrow account - \sigma_{branch\text{-}name = \text{“Perryridge”}} (account)$$

- Borrar todos los registros de la tabla loan con cantidad en el rango de 0 a 50

$$loan \leftarrow loan - \sigma_{amount \geq 0 \text{ and } amount \leq 50} (loan)$$

- Borrar todas las cuentas para las sucursales ubicadas en Needham.

$$r_1 \leftarrow \sigma_{branch\text{-}city = \text{“Needham”}} (account \bowtie branch)$$

$$r_2 \leftarrow \Pi_{branch\text{-}name,\ account\text{-}number,\ balance} (r_1)$$

$$r_3 \leftarrow \Pi_{customer\text{-}name,\ account\text{-}number} (r_2 \bowtie depositor)$$

$$account \leftarrow account - r_2$$

$$depositor \leftarrow depositor - r_3$$

Inserción

- Para insertar datos en una relación, puede:
  - especificarse una tupla para ser insertada
  - escribir una consulta cuyo resultado sea un conjunto de tuplas a insertar
- En álgebra relacional, una inserción se expresa como:

$$r \leftarrow r \cup E$$

  donde *r* es una relación y *E* es una expresión del álgebra relacional.
- La inserción de una tupla simple se expresa haciendo que *E* sea una relación constante que contiene una tupla.

Ejemplos de Inserción

- Insertar información en la base de datos especificando que Smith tiene $1200 en la cuenta A-973 en la sucursal de Perryridge.

$$account \leftarrow account \cup \{(\text{“Perryridge”}, A\text{-}973, 1200)\}$$

$$depositor \leftarrow depositor \cup \{(\text{“Smith”}, A\text{-}973)\}$$

- Ofrecer una nueva cuenta de ahorro con $200 como regalo para todos los clientes con préstamos concedidos en la sucursal de Perryridge. Hacer que el número del préstamo sea el que se utilice como número de cuenta para cada nueva cuenta de ahorro.

$$r_1 \leftarrow (\sigma_{branch\text{-}name = \text{“Perryridge”}} (borrower \bowtie loan))$$

$$account \leftarrow account \cup \Pi_{branch\text{-}name,\ account\text{-}number,200} (r_1)$$

$$depositor \leftarrow depositor \cup \Pi_{customer\text{-}name,\ loan\text{-}number} (r_1)$$

Actualización

- Mecanismo para cambiar algunos valores en una tupla sin que cambien todos los valores de la misma.
- Se utiliza el operador de proyección generalizada

$$r \leftarrow \prod_{F1, F2, \ldots, Fl,} (r)$$

- Cada F_i es:
  - el *i*ésimo atributo de *r*, si dicho *i*ésimo atributo no se actualiza; o,
  - Si corresponde actualizar al atributo, entonces F_i es una expresión formada por constantes y por los atributos de *r*, que proporciona el valor del nuevo atributo

Ejemplos de actualización

- Pagar intereses incrementando todos los balances en un 5%.

$$account \leftarrow \Pi_{AN,\ BN,\ BAL\ *\ 1.05}(account)$$

donde *AN*, *BN* y *BAL* significan *account-number*, *branch-name* y *balance*, respectivamente.

- Las cuentas con balances sobre $10,000 perciben un 6% de interés. A las restantes, pagar el 5 %

$$account \leftarrow \Pi_{AN,\ BN,\ BAL\ *\ 1.06}(\sigma_{BAL > 10000}(account)) \cup \Pi_{AN,\ BN,\ BAL\ *\ 1.05}(\sigma_{BAL \leq 10000}(account))$$

Vistas

- En muchos casos no se desea que todos los usuarios vean el modelo lógico completo; por ejemplo: todas las relaciones almacenadas actualmente en la base de datos.
- Considere una persona que necesita saber el número de cuenta de un cliente, pero no necesita ver el importe de préstamos. Esta persona debería ver una relación descrita, en términos del álgebra relacional, por:

$$\Pi_{customer\text{-}name,\ loan\text{-}number}\ (borrower \bowtie loan)$$

- Cualquier relación que no forma parte del modelo conceptual pero que se hace visible para algún usuario como "relación virtual" se denomina **vista (view)**.

Definición de Vista

- Las vistas se definen utilizando la instrucción **create view**. Para definirla, hay que indicar el nombre y la consulta que la calcula:

  create view *v* **as** <query expression>

  donde <query expression> es cualquier expresión de consulta que sea legal en álgebra relacional. El nombre de la vista es *v.*
- Una vez definida la vista, su nombre de vista puede utilizarse para referirse a la relación virtual que es generada por dicha vista.
- Una definición de vista no es lo mismo que crear una nueva relación evaluando la expresión de consulta. En vez de eso, la definición de la vista genera que se guarde una expresión, la cual es sustituida en aquellas consultas que utilicen la vista.

Ejemplos de vistas

- Considere la vista denominada *all-customer*, consistente de todas las sucursales y sus clientes.

create view *all-customer* **as**

$$\Pi_{\textit{branch-name, customer-name}} (\textit{depositor} \bowtie \textit{account})$$
$$\cup \Pi_{\textit{branch-name, customer-name}} (\textit{borrower} \bowtie \textit{loan})$$

- Pueden encontrarse todos los clientes de la sucursal Perryridge escribiendo:

$$\Pi_{\textit{customer-name}} (\sigma_{\textit{branch-name}} = \text{“Perryridge”} (\textit{all-customer}))$$

Actualizaciones mediante vistas

- Las modificaciones a la Base de Datos expresadas como vistas deben ser trasladadas a modificaciones de las relaciones actuales existentes en la Base de Datos.
- Considere la persona que necesita ver todos los datos de imposiciones en la relación *loan* excepto *amount*. La vista dada a dicha persona, *branch-loan,* se define como:

create view *branch-loan* **as**

$$\Pi_{\textit{branch-name, loan-number}}(\textit{loan})$$

- Dado que se admite que un nombre de vista aparezca dondequiera que se permite un nombre de relación, la persona pudiera escribir:

$$\textit{branch-loan} \leftarrow \textit{branch-loan} \cup \{(\text{“Perryridge”, L-37})\}$$

Actualizaciones mediante vistas (cont.)

- La inserción anterior puede representarse por una inserción en la relación actual *loan* a partir de la cual se construye la vista *branch-loan*.
- Una inserción en la relación *loan* requiere un valor para *amount*. Hay dos enfoques para trabajar esta inserción:
  1. Rechazarla y devolver un mensaje de error al usuario.
  2. Insertar la tupla ("L-37", "Perryridge", *null*) en la relación *loan.*
- Algunas actualizaciones a través de vistas no son posibles de convertir en actualizaciones de relaciones de la base de datos:
  - create view v as $\sigma_{branch\text{-}name = \text{"Perryridge"}}$ (*account*))
    v $\leftarrow$ v $\cup$ (L-99, Downtown, 23)
- Otras no pueden ser convertidas unívocamente:
  - *all-customer* $\leftarrow$ *all-customer* $\cup$ {("Perryridge", "John")}
    - ¡Hay que seleccionar loan o account, y crear un nuevo número loan/account!

Vistas Definidas Utilizando Otras Vistas

- Una vista puede ser utilizada en la definición de otra vista.
- Se dice que la vista v_1 *depende directamente* de la vista v_2 si v_2 se emplea en la expresión que define a v_1
- Se dice que la vista v_1 *depende* de la vista v_2 si v_1 depende directamente de v_2 o hay un camino de dependencias desde v_1 hasta v_2
- Una vista *v* se denomina *recursiva* si depende de si misma.

Expansión de vistas

- Es una manera de definir el significado de las vistas definidas en términos de otras vistas.
- Sea la vista v_1 definida por la expresión e_1 que puede contener a su vez otras vistas.
- La expansión de vista de una expresión repite el siguiente paso de reemplazo:

  repeat

  Buscar todas las vistas v_i de e_1

  Sustituir la vista v_i por la expresión que define v_i

  until no queden mas vistas en e_1

- Mientras las definiciones de vistas no sean recursivas, este bucle concluirá.

Cálculo relacional de tuplas

- Lenguaje de consultas no procedural, donde cada consulta está expresada en la forma

$$\{t \mid P(t)\}$$

- Es el conjunto de todas las tuplas *t* tales que el predicado *P* es verdadero para *t*
- *t* es una *variable de tupla*, *t*[*A*] denota el valor del atributo *A* para la tupla *t*
- $t \in r$ denota que la tupla *t* está en la relación *r*
- *P* es una *fórmula* similar a las del cálculo de predicados

Fórmulas del cálculo de predicados

1. Conjunto de atributos y constantes.
2. Operadores de comparación: $<, \leq, =, \neq, >, \geq$
3. Conectores: and ($\wedge$), or (v), not ($\neg$)
4. Implicación ($\Rightarrow$): x $\Rightarrow$ y, si x es verdadero, entonces y es verdadero

$$x \Rightarrow y \equiv \neg x \text{ v } y$$

5. Cuantificadores:
   - $\exists\ t \in r\ (Q(t)) \equiv$ " existe" una tupla *t* en relación *r* tal que el predicado *Q(t)* es true
   - $\forall t \in r\ (Q(t)) \equiv Q$ es true "para todas" las tuplas *t* en la relación *r*

Ejemplo del banco

- *branch (branch-name, branch-city, assets)*
- *customer (customer-name, customer-street, customer-city)*
- *account (account-number, branch-name, balance)*
- *loan (loan-number, branch-name, amount)*
- *depositor (customer-name, account-number)*
- *borrower (customer-name, loan-number)*

Ejemplo de consultas

- Buscar *loan-number*, *branch-name*, y *amount* para préstamos superiores a $1200

$$\{t \mid t \in loan \wedge t\,[amount] > 1200\}$$

- Buscar el *loan-number* para cada préstamo con una cantidad mayor que $1200

$$\{t \mid \exists\, s \in \text{loan}\ (t[loan\text{-}number] = s[loan\text{-}number] \wedge s\,[amount] > 1200)\}$$

Nótese que implícitamente queda definida la relación [*loan-number*] en el esquema producto de la consulta.

Ejemplo de consultas

- Buscar los nombres de todos los clientes con un préstamo, una cuenta, o ambas cosas:

$$\{t \mid \exists s \in borrower(\ t[customer\text{-}name] = s[customer\text{-}name])$$
$$\vee \exists u \in depositor(\ t[customer\text{-}name] = u[customer\text{-}name])$$

- Buscar los nombres de todos los clientes que tienen préstamo y cuenta en el banco:

$$\{t \mid \exists s \in borrower(\ t[customer\text{-}name] = s[customer\text{-}name])$$
$$\wedge \exists u \in depositor(\ t[customer\text{-}name] = u[customer\text{-}name])$$

Ejemplo de consultas

- Buscar los nombres de todos los clientes con préstamo en la sucursal Perryridge:

$$\{t \mid \exists s \in borrower(t[customer\text{-}name] = s[customer\text{-}name] \wedge \exists u \in loan(u[branch\text{-}name] = \text{“Perryridge”} \wedge u[loan\text{-}number] = s[loan\text{-}number]))\}$$

- Buscar los nombres de todos los clientes que tienen un préstamo otorgado por la sucursal Perryridge, pero que no tienen cuenta en ninguna sucursal del banco:

$$\{t \mid \exists s \in borrower(t[customer\text{-}name] = s[\text{customer-name}] \wedge \exists u \in loan(u[branch\text{-}name] = \text{“Perryridge”} \wedge u[loan\text{-}number] = s[\text{loan-}number])) \wedge \textbf{not } \exists v \in depositor (v[customer\text{-}name] = t[\text{customer-name}])\}$$

Ejemplo de consultas

- Buscar nombres de todos los clientes que han tomado préstamo en la sucursal Perryridge y las ciudades en que viven:

$$\{t \mid \exists s \in \mathit{loan}(s[\mathit{branch\text{-}name}] = \text{“Perryridge”} \wedge \exists u \in \mathit{borrower}\,(u[\mathit{loan\text{-}number}] = s[\mathit{loan\text{-}number}] \wedge t[\mathit{customer\text{-}name}] = u[\mathit{customer\text{-}name}]) \wedge \exists v \in \mathit{customer}\,(u[\mathit{customer\text{-}name}] = v[\mathit{customer\text{-}name}] \wedge t[\mathit{customer\text{-}city}] = v[\mathit{customer\text{-}city}]))\}$$

Ejemplo de consultas

- Buscar los nombres de todos los clientes que tienen una cuenta en todas las sucursales ubicadas en Brooklyn:

$$\{t \mid \exists\, c \in \text{customer}\ (t[\text{customer.name}] = c[\text{customer-name}]) \wedge$$
$$\forall\, s \in \textit{branch}(s[\textit{branch-city}] = \text{“Brooklyn”} \Rightarrow$$
$$\exists\, u \in \textit{account}\ (\ s[\textit{branch-name}] = u[\text{branch-name}]$$
$$\wedge \exists\, s \in \textit{depositor}\ (\ \ t[\textit{customer-name}] = s[\text{customer-name}]$$
$$\wedge\ \ s[\textit{account-number}] = u[\text{account-number}]\ ))\ )\}$$

Seguridad de Expresiones

- Es posible escribir expresiones de cálculo de tuplas que generan relaciones infinitas.
- Por ejemplo, $\{t \mid \neg\, t \in r\}$ resulta en una relación infinita si el dominio de uno cualquiera de los atributos de la relación *r* es infinito.
- Para evitar este problema, se restringe el conjunto de expresiones permitidas a expresiones seguras.
- Una expresión $\{t \mid P(t)\}$ en el cálculo relacional de tuplas es *segura* si cada componente de *t* aparece en una de las relaciones, tuplas, o constantes que aparecen en *P*
  - NOTA: Esta condición es mas que sólo una condición sintáctica.
    - Ejemplo: $\{ t \mid t[A]=5 \vee$ **true** $\}$ no es segura --- define un conjunto infinito con valores de atributos que no aparecen en ninguna relación o tuplas o constantes en *P*.

Cálculo relacional de dominios

- Lenguaje de consultas no procedural equivalente en potencialidad al cálculo relacional de tuplas.
- Cada consulta es una expresión de la forma:

$$\{ < x_1, x_2, \ldots, x_n > \mid P(x_1, x_2, \ldots, x_n)\}$$

  - $x_1, x_2, \ldots, x_n$ representan variables de dominio
  - P representa una fórmula similar a las del cálculo de predicados.

Ejemplo de consultas

- Buscar *loan-number*, *branch-name*, y *amount* para préstamos superiores a $1200:

$$\{< l, b, a > \mid < l, b, a > \in loan \wedge a > 1200\}$$

- Buscar nombres de clientes con préstamos superiores a $1200:

$$\{< c > \mid \exists\, l, b, a\, (< c, l > \in borrower \wedge < l, b, a > \in loan \wedge a > 1200)\}$$

- Buscar los nombres de todos los clientes que tienen un préstamo de la sucursal Perryridge y el monto de dicho préstamo:

$$\{< c, a > \mid \exists\, l\, (< c, l > \in borrower \wedge \exists b(< l, b, a > \in loan \wedge b = \text{“Perryridge”}))\}$$

or $\{< c, a > \mid \exists\, l\, (< c, l > \in borrower \wedge < l, \text{“Perryridge”}, a > \in loan)\}$

Ejemplo de consultas

- Buscar los nombres de todos los clientes que tienen un préstamo, una cuenta ó ambas cosas en la sucursal Perryridge:

$$\{< c > \mid \exists\, l\, (\{< c, l > \in \textit{borrower} \wedge \exists\, b,a(< l, b, a > \in \textit{loan} \wedge b = \text{“Perryridge”})) \vee \exists\, a(< c, a > \in \textit{depositor} \wedge \exists\, b,n(< a, b, n > \in \textit{account} \wedge b = \text{“Perryridge”}))\}$$

- Buscar los nombres de todos los clientes que tienen una cuenta en todas las sucursales ubicadas en Brooklyn:

$$\{< c > \mid \exists\, s, n\, (< c, s, n > \in \text{customer}) \wedge \forall\, x,y,z(< x, y, z > \in \textit{branch} \wedge y = \text{“Brooklyn”}) \Rightarrow \exists\, a,b(< x, y, z > \in \textit{account} \wedge < c,a > \in \textit{depositor})\}$$

Seguridad de las Expresiones

$$\{ < x_1, x_2, ..., x_n > \mid P(x_1, x_2, ..., x_n)\}$$

es segura si se cumplen todos los tres puntos siguientes:

1. Todos los valores que aparecen en tuplas de la expresión son valores de *dom*(*P*), que significa: los valores aparecen ó en *P* o en una tupla de una relación mencionada en *P*.
2. Para cada subfórmula "existe" de la forma $\exists\, x\, (P_1(x))$, dicha subfórmula es verdadera si y solamente si hay un valor de *x* en *dom*(P_1) tal que $P_1(x)$ sea verdadera.
3. Por cada subfórmula "para todo" de la forma $\forall_x\, (P_1\, (x))$, dicha subfórmula es verdadera si y solamente si $P_1(x)$ es verdadera para todos los valores *x* de *dom* (P_1).

Otros Lenguajes

- Los lenguajes orientados a Transformaciones son lenguajes no procedurales que emplean relaciones para transformar datos de entrada en las salidas requeridas. Por ejemplo: el Lenguaje estructurado de Consultas (*Structured Query Language* - SQL).
- Los lenguajes gráficos muestran a los usuarios diagramas de la estructura de la relación. El usuario llena un ejemplo de lo que desea y el sistema devuelve los datos solicitados en ese formato. Por ejemplo: Consultas por ejemplo (*Query by example* - QBE).

Otros Lenguajes

- Los lenguajes de cuarta generación (4GLs) pueden crear aplicaciones completamente personalizadas utilizando un conjunto limitado de comandos en un ambiente amistoso al usuario, comúnmente manejado por menús.
- Algunos sistemas aceptan una variante de *lenguaje natural*, denominada 5GL, aunque este desarrollo se encuentra aún en su etapa inicial.

Resumen (1)

- El Álgebra Relacional (AR) define un conjunto de operaciones sobre tablas que devuelven como resultado tablas.
- Estas operaciones pueden combinarse para obtener expresiones acordes con las consultas deseadas.
- Las operaciones del AR pueden dividirse en:
  - Básicas.
  - Adicionales, que pueden expresarse en términos de las operaciones básicas.
  - Extendidas, algunas de las cuales añaden mayor poder expresivo al AR

Resumen (2)

- Las Bases de Datos (BD) pueden modificarse con:
  - la inserción,
  - el borrado y
  - la actualización

  de tuplas.
- Se usó el AR con el operador de asignación para expresar estas modificaciones.

Resumen (3)

- Las vistas son relaciones virtuales definidas mediante expresiones de consulta. Constituyen mecanismos útiles para simplificar accesos a la BD. Pueden tener consecuencias desventajosas, por lo que los sistemas de BD restringen estrictamente las actualizaciones mediante ellas.
- Por razones de eficiencia del procesamiento de las consultas, una vista puede estar materializada: la consulta se evalúa y el resultado se almacena físicamente. Esto implica que si las relaciones correspondientes se actualizan, debe actualizarse también dicha vista materializada.

Resumen (4)

- El cálculo relacional de tuplas y el cálculo relacional de dominios son lenguajes no procedurales que poseen la potencia básica necesaria en un lenguaje de consultas relacional.
- El álgebra relacional básica es un lenguaje procedural que es equivalente en potencia con ambas formas del cálculo relacional cuando se restringen a las expresiones seguras.

Resumen (5)

- El Álgebra Relacional y los Cálculos Relacionales son lenguajes rígidos, formales, que no resultan adecuados para los usuarios ocasionales de los sistemas de Bases de Datos.
- Los sistemas comerciales emplean alternativas que favorecen al usuario final:
  - SQL: Basado en álgebra relacional.
  - QBE: Basado en cálculo relacional de dominios.

Bibliografía

- Silberschatz, Korth y Sudarshan “Fundamentos de Bases de Datos”
- Connolly y Begg “Sistemas de Bases de Datos”
- Date “Introducción a los Sistemas de Bases de Datos”
- Ullmann “Principios de los Sistemas de Bases de Datos”
- Cisterna “METODOS DE OPTIMIZACION DE CONSULTAS PARA EL LENGUAJE SQL” http://macine.epublish.cl/tesis/index-Contents.html (visitado 2006.05.11 11:53)

Ejercicios (1)

Las siguientes tablas forman parte de una BD:

Hotel (hotelNo, hotelNombre, ciudad)

Room (roomNo, hotelNo, tipo, precio)

Booking(hotelNo, guestNo, dateFrom, dateTo, roomNo)

Guest (guestNo, guestNombre, guestDireccion)

1. Describa las relaciones que se generan mediante las siguientes operaciones del álgebra relacional:
   a) $\Pi_{hotelNo}(\sigma_{price>50000}(Room))$
   b) $\sigma_{Hotel.hotelNo=Room.hotelNo}(Hotel \times Room)$
2. Proporcione las expresiones equivalentes en el cálculo relacional de tuplas y en el de dominios.

Ejercicios (2)

3. Escriba las expresiones tanto del álgebra relacional, como del cálculo relacional de tuplas y del de dominios para responder las siguientes consultas:
   a) Enumerar todos los hoteles
   b) Enumerar todas las habitaciones cuyo precio sea inferior a $20000
   c) Enumerar los nombres y ciudades de procedencia de todos los huéspedes en el hotel “Raddison”
   d) Enumerar los detalles guestNo, guestNombre y guestDireccion para todos los huéspedes en el hotel “Sheraton”

Ejercicios (3)

Dada la siguiente Base de Datos:

Empleado (nombre-empleado, calle, ciudad)

Trabaja (nombre-empleado, nombre-empresa, sueldo)

Empresa (nombre-empresa, ciudad)

Jefe (nombre-empleado, nombre-jefe)

1. Escriba una expresión del álgebra relacional que permita realizar las siguientes consultas:

Ejercicios (4)

a) Averiguar los nombres de todos los empleados que trabajan para la UDLA

b) Averiguar el nombre, calle y ciudad de residencia de todos los empleados que ganan mas de $2000000 mensuales

c) Averiguar el nombre de todos los empleados que viven en la misma ciudad donde radica la empresa para la que trabajan

d) Determinar la empresa con mayor número de empleados

e) Determinar la compañía que paga menos nómina (salario de todos los empleados en conjunto)

f) Dar a todos los jefes un aumento de salario del 10% a menos que el sueldo resultante sea mayor que $2000000. Para estos casos, aumentar el salario solamente en un 3%