Tema 2. Modelo relacional

- Creado por Edward (Ted) Codd a principios de los 70.
- Modelo de datos orientado a registro:
  - Estructuras.
  - Restricciones.
  - Operaciones. Lenguajes.
- Usa una colección de tablas para representar tanto los datos como sus relaciones.
- Sólida base teórica.
- Implementado en muchos SGBD (relacionales).

2.1. Estructuras de datos

Modelo relacional

- El concepto principal es la tabla o relación
  - Cada tabla o relación es un conjunto de tuplas donde cada una de ellas corresponde a una fila de la tabla
  - Cada tupla corresponde a la descripción de una entidad particular o a la descripción de una relación particular entre varias entidades particulares.

Conceptos

- Relación
  - Las entidades particulares se representan como tuplas o filas de la tabla.

- Atributo
  - Se representan como las columnas de la tabla.
  - Los valores de los atributos de las tuplas deben ser atómicos.
  - Tienen asociado un dominio (tipo de datos).

Terminología del modelo relacional

- Esquema de una tabla o relación
  - viene dado por el nombre de la tabla y una lista de atributos.
    - Alumnos (DNI, Apellidos, Nombre, teléfono, acceso)

- El orden de los atributos en la lista no importa.
- Lo fijamos porque nos viene bien para representarlo como tabla, pero cualquier permutación es válida.
- Instancia de una tabla => Conjuntos de entidades particulares.
  - Cada entidad particular se representa como una tupla.
  - Cada componente de la tupla corresponde con el valor del atributo correspondiente, según el orden enunciado en el esquema de la tabla.

Ejemplo: Diseño de una base de datos para la secretaría de una facultad

- Se desea gestionar (almacenar, consultar, actualizar, …) la información correspondiente a la secretaría de una facultad.
- Ésta es la información de la que partimos:
  1. Por cada alumno se requiere la información: DNI, Apellidos y nombre, domicilio, teléfono y acceso (que indica el tipo de acceso a la universidad). También se precisa conocer en cada momento las asignaturas en las que el alumno está matriculado, así como la nota en cada asignatura. Un alumno sólo puede matricularse en una asignatura una vez, y debe matricularse al menos en una.
  2. Por cada asignatura se requiere: código, título y núm de créditos. Puede haber varias asignaturas con el mismo número de créditos, pero todas tienen distinto código y distinto título.

Ejemplo: Diseño de una base de datos para la secretaría de una facultad

3. Cada asignatura puede estar impartida por uno o más profesores. Del profesor se deben conocer los mismo datos que en el caso de los alumnos, salvo el de acceso: DNI, Apellidos y nombre, domicilio y teléfono. El número máximo de asignaturas que puede impartir un profesor es 6, aunque puede que no imparta ninguna.
4. Algunos profesores tienen un supervisor (sólo uno), que es otro profesor.
5. Dados un profesor concreto y una asignatura de las que imparte, se debe conocer el aula en la que el profesor da esa asignatura (es siempre la misma). El aula se identifica mediante el nombre de edificio y el número de aula. Se supone que dentro del mismo edificio cada aula tiene un número diferente.

Ejemplo secretaría

- Ejemplo: Instancia de la tabla Alumnos:
  - { (01234567Z, Vázquez , Manuel, 9112345678, normal), ...}

DNI	Apellido	Nombre	teléfono	acceso
01234567Z	Vázquez	Manuel	9112345678	normal
…				

Terminología del modelo relacional

- Un relación no puede contener tuplas repetidas
  - Existe un conjunto de atributos que determina unívocamente a cada tupla
  - Cada tupla está unívocamente definida por sus valores

2.2. Restricciones

Unicidad de entidades

- Las entidades deben poder distinguirse unas de otras a través de los valores de un subconjunto de sus atributos.
- Estos conjuntos se denominan claves.
- Interesa a nivel práctico encontrar un conjunto de atributos lo más pequeño posible que nos permita distinguir unas entidades de otras.

Superclaves

- Dado un tipo de entidades E en una BD, se llama superclave a cualquier conjunto de atributos que permita distinguir a todas las entidades de cualquier instancia válida de E en la BD.

- En general los atributos de la superclave pertenecen a un tipo de entidad E,
  - Pero a veces un atributo de la superclave puede venir de otro tipo de entidad p.e. F …lo veremos

Superclaves

- En el caso de alumnos, el conjunto {teléfono} NO es una superclave, porque puede haber varias personas con el mismo número de teléfono (ej. 2 hermanos).
- Tampoco podemos tomar como superclave ApellidosyNombre porque puede repetirse.
- Una posible superclave es {DNI}.
- El caso de profesores es igual al anterior.
- Para asignaturas tenemos en principio 2 superclaves {título} y {código}.
- Para aulas la única superclave es {Edificio, número}.

Superclaves

- Propiedad:
  - Si S es una superclave y S $\subseteq$ S', entonces S' es también superclave

- Ej.: En el caso de asignaturas tenemos en realidad 6 superclaves:
  - {título},
  - {código},
  - {título, núm.creditos},
  - {código, núm.créditos},
  - {título, código},
  - {título, código, núm.creditos}.

Claves candidata y primaria

- Clave candidata.
  - Se llama clave candidata de un tipo de entidad a una superclave que no contiene ningún subconjunto que también sea superclave. (Conjunto mínimo de atributos que forma una superclave).

- Clave primaria.
  - Se llama clave primaria a la clave candidata finalmente seleccionada por el diseñador para distinguir entre las entidades de cada instancia.

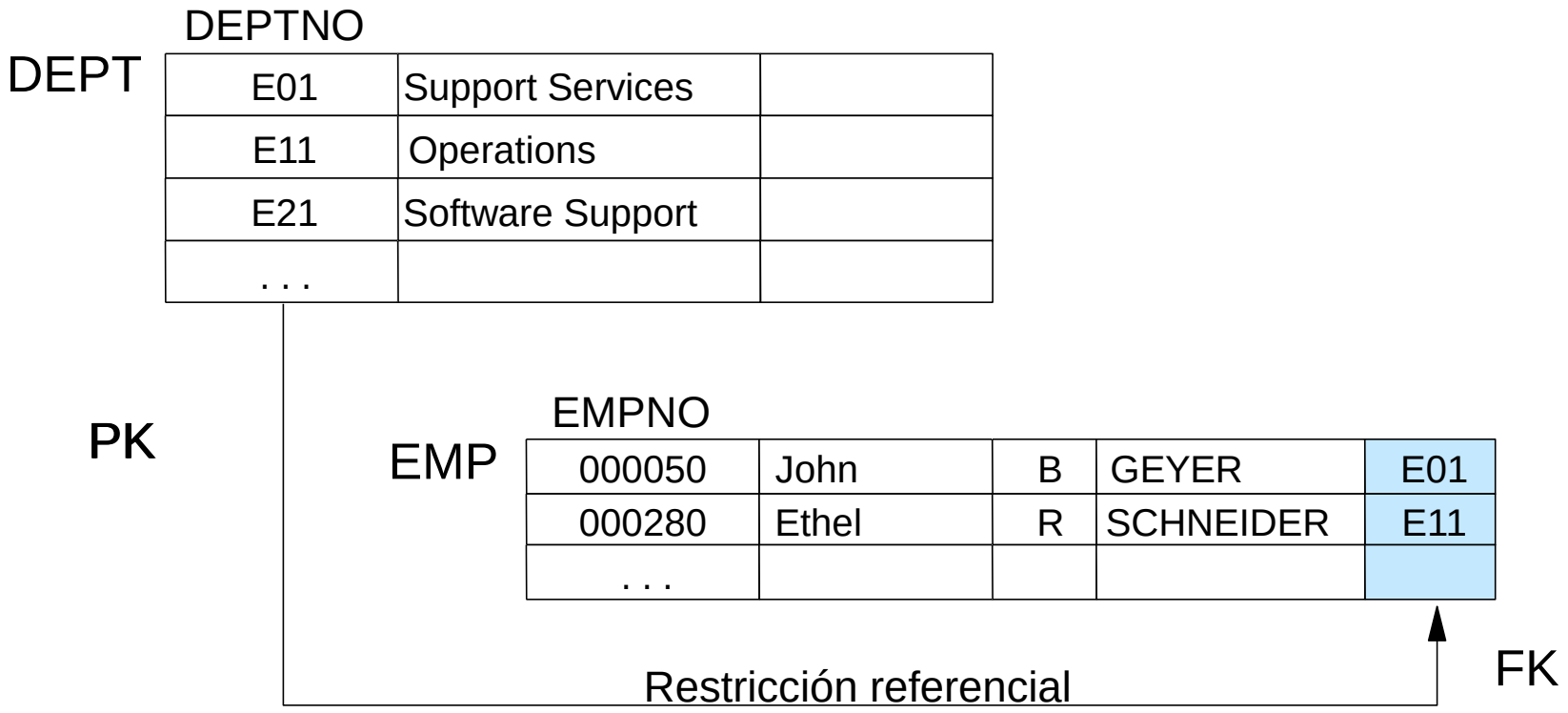
Claves candidata y primaria

- En el caso de alumnos tenemos una única clave candidata {dni} que será también la clave primaria.
- En el caso de profesores es idéntico: tenemos una única clave candidata {dni} que será también la clave primaria.
- En el caso de aulas, la única superclave {edificio, núm.aula} es la única clave candidata y la clave primaria.

Claves candidata y primaria

- En el caso de asignaturas tenemos dos claves candidatas {código} y {título}. Elegimos como clave primaria {código}, porque:
  - es posible que en el futuro haya dos asignaturas con el mismo título (ej. : cambio de planes de estudio) pero parece sensato obligar a que siempre tengan códigos distintos.
  - El código es más fácil de introducir (para buscar la información) (y los índices más rápidos, pero aún no hemos hablado de índices).

Clave ajena o externa



Una restricción referencial garantiza que todos los valores (no nulos) de una FK también son valores de la PK

2.3. Lenguajes

Lenguajes de consulta

- Un lenguaje de consulta es un lenguaje con el que los usuarios solicitan información de la BBDD.
  - Estos lenguajes suelen ser de nivel superior que el de los lenguajes de programación habituales.
- Clasificación
  - Procedimientales
    - El usuario indica al sistema que lleve a cabo una serie de operaciones en la BBDD para calcular el resultado.
  - No procedimientales
    - El usuario describe la información deseada sin dar un procedimiento concreto para obtener la información.

Lenguajes de consulta

- Lenguajes formales del modelo relacional:
  - Álgebra relacional
  - Cálculos
- Lenguajes prácticos de los SGBD:
  - QBE
  - SQL

Lenguajes formales del modelo relacional

- Se parte de los esquemas de relaciones y se define un lenguaje de manipulación de datos.
- Dentro de estos lenguajes podemos encontrar
  - Álgebra relacional (Procedimental)
  - Cálculo relacional de Tuplas (No procedimental)
  - Cálculo relacional de Dominios (No procedimental)
- Estos lenguajes son estrictos y formales y han servido como base para los lenguajes implementados en los SGBD comerciales que veremos posteriormente.

Álgebra relacional

- Álgebra relacional
  - Operadores básicos
  - Operadores adicionales
- Álgebra relacional extendida
  - Proyección generalizada
  - Agregados
  - Reunión externa (valores nulos)
- Manipulación de datos
  - Definición de relaciones
  - Inserción
  - Borrado
  - Actualización

Álgebra relacional

- Lenguaje de consulta **procedimental** basado en **álgebra de conjuntos**.
- Serie de operaciones que toman una o dos relaciones como entrada y generan una relación como salida, pero **siempre sin modificar los datos** de la base de datos (es un lenguaje de consulta).
- Operaciones básicas:
  - Selección
  - Proyección
  - Unión, intersección, diferencia y producto cartesiano.
- Conjunto mínimo completo (conjunto de operaciones que permiten realizar todas las operaciones posibles):
  - Proyección, selección, unión, diferencia, producto cartesiano
- Operaciones adicionales (no amplían la expresividad del conjunto mínimo):
  - Reunión (JOIN o producto cartesiano con condición)
  - División

Implementaciones del álgebra relacional

- Rel.Voorhis. University of Derby (UK)
  - Álgebra relacional pura
  - Lenguaje D: Date y Darwen
- LEAP. Leyton. Oxford Brookes University (UK)
  - Álgebra relacional pura
- Relational Algebra System for Oracle and Microsoft SQL Server. Solomon. Florida Atlantic University (USA)
  - Álgebra relacional con división
  - Formulación AR no muy natural
- WinRDBI. S.W. Dietrich. Arizona State University (USA)
  - Álgebra relacional
  - Cálculo relacional de dominios
  - Cálculo relacional de tuplas
  - SQL
- DES. F. Sáenz Pérez. Universidad Complutense de Madrid (EU)
  - Álgebra relacional extendida (nulos, recursión, correlaciones, duplicados)
  - SQL (con extensiones: recursivo, hipotético, …)
  - Datalog

Selección σ

- La operación de selección **da como resultado un conjunto de tuplas que cumplen la condición de selección** aplicada sobre la entrada.
- $\sigma_c(R)$= conjunto de tuplas de la relación R que cumplen la condición *c*.
  - *R* será un esquema de relación o una expresión que lo represente.
  - *c* será un conjunto de cláusulas conectadas entre sí mediante operadores booleanos ($\wedge$, $\vee$, $\neg$).
  - Cada **cláusula** será de la forma
    - <nombre atributo> OPCOMP <nombre atributo>
    - donde OPCOMP será un operador de comparación (=, ≠, <, ≤, >, ≥)
- El resultado de la consulta es una selección con los mismos atributos que la original.

Selección σ (cont.)

- ***Ejemplo***: base de datos de empleados.

  emp (dni, nom, ap1, ap2, sueldo)

```
DES> :-
   type(emp(dni:string,nom,string,ap1:string,ap2:
   string,sueldo:int))
```

- Empleados con sueldo superior a 1.200 € :

  $\sigma_{sueldo > 1200}$ (emp)

```
DES> select sueldo>1200 (emp)
```

- Empleados cuyos apellidos sean iguales:

  $\sigma_{ap1 = ap2}$ (emp)

```
DES> select ap1=ap2 (emp)
```

- Empleados con sueldo superior a 1.200 € cuyos apellidos sean iguales:

  $\sigma_{(sueldo > 1200) \wedge (ap1 = ap2)}$ (emp)

```
DES> select sueldo>1200 AND ap1=ap2 (emp)
```

Proyección π

- La operación de proyección devuelve las tuplas de la relación que se le pasa como argumento, omitiendo ciertas columnas.
  - Los atributos que queramos obtener se colocan como subíndice de la operación, separados por comas.
- $\pi_{LISTA_ATRIBUTOS}\ (R)$= conjunto de tuplas de R con los atributos de la lista.
  - *R* será un esquema de relación o una expresión que lo represente.
  - *LISTA_ATRIBUTOS* será una sucesión de atributos separados por comas.
- **Una proyección no repite tuplas** porque su resultado es una relación que a su vez es un conjunto de tuplas
  - Los conjuntos no repiten elementos
  - Dos tuplas son iguales si son iguales componente a componente

Proyección π (cont.)

- ***Ejemplo***: base de datos de empleados (ejemplo anterior).
- DNI y sueldo de todos los empleados:

  $\pi_{dni,\ sueldo}$ (emp)

  ```
  DES> project dni,sueldo (emp)
  ```

- DNI de los empleados con más de 1.200 € de sueldo:

  π_{dni} ($\sigma_{sueldo > 1200}$ (emp))

  ```
  DES> project dni (select sueldo>1200 (emp))
  ```

- DNI de los empleados con más de 1.200 € de sueldo, 'dos pasos':

  r1 ← $\sigma_{sueldo > 1200}$ (emp) /* Tuplas de emp con sueldo > 1200 */

  r2 ← π_{dni} (r1) /* DNI empleados con sueldo > 1200 */

  ```
  DES> r1 := select sueldo>1200 (emp)
  DES> r2 := project dni (r1)
  ```

Renombramiento

- La operación de renombramiento permite cambiar el nombre del esquema y de las columnas de otro esquema.
- $\rho_{E(LISTA_ATRIBUTOS)}(R)$ = La relación resultante tiene nombre E con nombre de atributos LISTA_ATRIBUTOS.
  - *R* será un esquema de relación o una expresión que lo represente.
  - *E* será el nombre del esquema resultantes.
  - *LISTA_ATRIBUTOS* será una sucesión de atributos separados por comas. Éstos serán los nuevos nombres para los atributos definidos en el esquema R.
- La relación resultante **contiene las mismas tuplas** que el esquema original.

Renombramiento

- ***Ejemplo:***
  - Los atributos de una relación se pueden renombrar para facilitar el manejo de atributos:

    /* Renombramiento de dni a d */

    $\rho_{emp\ (d,nom,ap1,ap2,sueldo)}$ (emp)

    /* Renombramiento de todos los atributos */

    $\rho_{emp\ (d,n,a1,a2,s)}$ (emp)
  - Es también posible realizar copias de relaciones renombrando o no sus atributos:

    /* Copia de emp a emp1sin renombrar atributos */

    $\rho_{emp1(dni,\ nom,\ ap1,\ ap2,\ sueldo)}$ (emp)

    ```
    DES> rename emp1(dni,nom,ap1,ap2,sueldo) (emp)
    ```
  - DES exige renombrar la relación. Motivo: recursión

Unión ∪, Intersección ∩ y Diferencia -

- Operaciones típicas de conjuntos. Las relaciones deben ser compatibles:
  - Mismo número de atributos y mismo dominio dos a dos:

$$R1\ (A_1, A_2, \ldots A_n) \qquad R2\ (B_1, B_2, \ldots B_n)$$

$$\text{Para todo } i,\ 1 \leq i \leq n,\ \text{Dom}\ (A_i) = \text{Dom}\ (B_i)$$

- El resultado de estas operaciones será otra relación (no repite tuplas).
- En la relación resultante, **los atributos serán los de la relación que pongamos como primer operando**.
- Lo habitual es realizar uniones, intersecciones y diferencias con esquemas exactamente iguales.

Unión, Intersección y Diferencia

- ***Ejemplo:***
  - est (dni_est, nomb_est)
  - prof (dni_prof, nomb_prof)
- Personas que son alumnos o son profesores

  r ← est ∪ prof

  DES> r := est union prof
  - Si hay algún alumno que es a la vez profesor y alumno sólo aparecerá una vez en la relación R.
- Profesores que son también alumnos (quizás de doctorado)

  r ← est ∩ prof

  DES> r := est intersect prof
- Profesores que no son alumnos

  r ← prof - est

  DES> r := prof difference est

Producto Cartesiano ×

- Esta es una operación binaria. No exige que las relaciones sean compatibles.

$$R1\ (A_1, ..., A_n) \times R2\ (B_1, ..., B_m) \rightarrow R\ (A_1, ..., A_n, B_1, ..., B_m)$$

- La relación resultante, R, contiene tuplas con los atributos de ambas relaciones. Pero esas tuplas **son el resultado de combinar cada una de las tuplas de la primera relación con todas las de la segunda**.

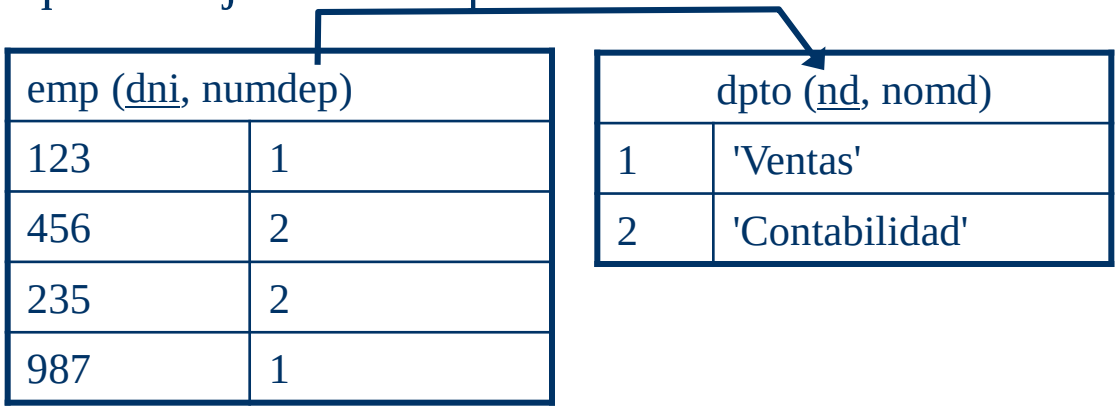
- **`DES> r1 product r2`**

- ***Ejemplo***:

$R1\ (A_1, ..., A_n)$	$R2\ (B_1, ..., B_m)$	$R1\ (A_1, ..., A_n) \times R2\ (B_1, ..., B_m)$
x1, ..., x1	y1, ..., y1	x1, ..., x1, y1, ..., y1
x2, ..., x2	y2, ..., y2	x1, ..., x1, y2, ..., y2
x3, ..., x3		x2, ..., x2, y1, ..., y1
		x2, ..., x2, y2, ..., y2
		x3, ..., x3, y1, ..., y1
		x3, ..., x3, y2, ..., y2

Ejemplos

- ***Ejemplo:*** para el siguiente esquema se pide obtener el DNI de los empleados que trabajan en el departamento 'Contabilidad'.



emp (dni, numdep)	
123	1
456	2
235	2
987	1

dpto (nd, nomd)	
1	'Ventas'
2	'Contabilidad'

- Pasos a seguir:
  - Combinar los datos de ambas tablas.
  - Quedarse con las filas donde cada empleado aparezca junto a su departamento y además el nombre del departamento sea 'Contabilidad'
  - Filtrar campos para obtener DNI.

Ejemplos

- Solución 1:

$res \leftarrow \pi_{dni} (\sigma_{numdep = nd \wedge nomd = 'contabilidad'} (emp \times dpto))$

- Solución 2:

$r1 \leftarrow \sigma_{nomd = 'contabilidad'} (dpto)$ /* Tuplas con NOMD = 'Contabilidad'*/

$r2 \leftarrow \pi_{nd} (r1)$

$r3 \leftarrow emp \times r2$ /* Prod. cartesiano como antes, pero sólo con ND */

$r4 \leftarrow \sigma_{numdep = nd} (r3)$ /* Tuplas de los empleados de 'Contabilidad'.*/

$res \leftarrow \pi_{dni} (r4)$ /* Escoge sólo el DNI. Resultado que queríamos*/

Producto Cartesiano ×

- En un producto cartesiano puede aparecer el mismo nombre de atributo en r1 y r2
  - Un convenio de denominación permite distinguir unos atributos de otros:
    - Se adjuntan el nombre de la relación de la que procede el argumento.
  - Si el producto cartesiano contiene dos veces la misma tabla se soluciona utilizando renombramiento

- Ejemplos:
  - emp(dni, num) dpto(num, nomd)

res1 ← π_{dni} ($\sigma_{emp.num = dpto.num \wedge nom = 'contabilidad'}$ (emp × dpto))

res2 ← $\rho_{emp1(dni, num,)}$ (emp) × emp

Ejemplos

- ***Ejemplo:*** para el siguiente esquema se pide obtener el DNI de los empleados que trabajan en el departamento 'Contabilidad'.

emp (<u>dni</u>, num) dpto (<u>num</u>, nomd)

- solución con renombramiento.

$r1 \leftarrow \sigma_{nomd = 'contabilidad'}(dpto)$ /* tuplas con nomd = 'contabilidad' */

$r2 \leftarrow \rho_{numdep}(\pi_{num}(r1))$ /* **renombrado** */

$r3 \leftarrow emp \times r2$ /* prod. cartesiano */

$r4 \leftarrow \sigma_{num = numdep}(r3)$ /* tuplas de los empleados de 'contabilidad'. */

$res \leftarrow \pi_{dni}(r4)$ /* Escoge sólo el DNI. Resultado que queríamos */

Ejemplos

- ***Ejemplo* (ex1.dl):** Para el siguiente esquema se pide obtener el nombre de los empleados que trabajan en proyectos dirigidos por 'López'.

- Observaciones:
  - El atributo que formará la relación resultado está en **emp**
  - El atributo sobre el que se aplica la condición está en **proy**
  - Se deben combinar los datos de las relaciones y luego filtrar para obtener aquellas filas donde los atributos de reunión coinciden, además de aplicar las condiciones correspondientes.

Ejemplos

- Solución:

$r1 \leftarrow \pi_{np} (\sigma_{director = 'lópez'} (proy))$ /* np de proyectos dirigidos por lópez */

$r2 \leftarrow trabaja \times r1$ /* combinación de proyectos con trabaja */

$r3 \leftarrow \sigma_{np = npro} (r2)$ /* selección de las filas que contienen datos relacionados */

$r4 \leftarrow \pi_{dnie} (r3)$ /* aquí solo tenemos dni, nos piden el nombre. */

$r5 \leftarrow r4 \times emp$ /* mismo proceso de combinación y selección */

$r6 \leftarrow \sigma_{dni = dnie} (r5)$

$res \leftarrow \pi_{nom} (r6)$ /* Resultado final*/

Ejemplos

- ***Ejemplo* (ex2.dl):** Para el siguiente esquema se pide obtener el DNI de los empleados cuyo supervisor gana más de 2.000 €.

- soluciones:

$emp1 \leftarrow \rho_{(dni1,\ s1,\ dni_s1)}\ (emp)$

$r1 \leftarrow emp \times emp1$

$r2 \leftarrow \sigma_{dnisupervisor = dni1 \wedge s1 > 2000}\ (r1)$

$res \leftarrow \pi_{dni}\ (r2)$

$r1 \leftarrow \sigma_{sueldo > 2000}\ (emp)$

$r2 \leftarrow \rho_{(dni_s)}\ (\pi_{dni}\ (r1))$

$r3 \leftarrow emp \times r2$

$r4 \leftarrow \sigma_{dnisupervisor = dni_s}\ (r3)$

$res \leftarrow \pi_{dni}\ (r4)$

Ejemplos

- ***Ejemplo* (ex3.dl):** Para el siguiente esquema se pide obtener el DNI de los empleados con al menos dos hijos.

- solución:

$r1 \leftarrow \rho_{(dni1,\ nom1)}\ (hijo)$ /* primero renombramos para no confundirnos. */

$r2 \leftarrow hijo \times r1$ /* r2 tiene tuplas así: dni, nom, dni1, nom1 */

$r3 \leftarrow \sigma_{dni1 = dni \wedge nom \neq nom1}\ (r2)$ /* interesan tuplas así: 1 a 1 b (igual dni del padre, y distinto nombre) */

$res \leftarrow \pi_{dni}\ (r3)$

Ejemplos

- ***Ejemplo* (ex2.dl):** para el siguiente esquema se pide obtener el DNI de los empleados supervisados por DNI = 123

emp (dni, sueldo, dnisup)

- DNI de los empleados supervisados a uno y dos niveles por DNI = 123

res $\leftarrow \pi_{dni} (\sigma_{dnisup = 123} (emp))$ /* dni de los supervisados directamente */

r1 $\leftarrow \rho_{(dni1)} (\pi_{dni} (\sigma_{dnisup = 123} (emp)))$

r2 $\leftarrow \sigma_{dnisup = dni1} (emp \times r1)$

r3 $\leftarrow \pi_{dni} (r2)$ /* en r3 están los dni de los superv. a 2 niveles por 123. */

res $\leftarrow r1 \cup r3$

- No se puede responder a 'empleados supervisados a cualquier nivel'.

Reunión/Join $\bowtie_c$

- Operación binaria que implementa el producto cartesiano con condición de selección.
- $R1 \bowtie_c R2$ = conjunto de tuplas resultantes del producto cartesiano $R1 \times R2$ que cumplen la condición *c*.
  - *c* es la **condición de reunión**, y será un conjunto de cláusulas conectadas entre sí mediante operadores booleanos ($\wedge$, $\vee$, $\neg$), cada una de la forma
    - <nombre atributo> OPCOMP <nombre atributo>
    - donde OPCOMP será un operador de comparación ($=$, $\neq$, $<$, $\leq$, $>$, $\geq$)
      En DES, que sigue la sintaxis SQL: (=, <>, =<,>,>=) y (and, or, not)
- Se cumple la siguiente correspondencia:

$$R1 \bowtie_c R2 \equiv \sigma_c (R1 \times R2)$$

- **`DES> r1 zjoin condición r2`**

Reunión/Join $\bowtie_c$ (cont.)

- La reunión entre dos relaciones puede dar como resultado el conjunto vacío si ninguna de las tuplas del producto cartesiano cumple la condición de reunión.
- Las tuplas cuyos atributos de reunión sean nulos no aparecen en el resultado.

- ***Ejemplo* (ex4.dl):** para el siguiente esquema se pide obtener el DNI de los empleados que trabajan en el departamento 'Contabilidad'.

- Solución:

$r1 \leftarrow \sigma_{nomd = \text{'Contabilidad'}}(DPTO)$ /* Tuplas con nomd = 'Contabilidad' */

$r2 \leftarrow emp \bowtie_{numdep = nd} r1$ /* Tuplas de los empleados de 'Contabilidad'. */

$res \leftarrow \pi_{dni}(r2)$

Reunión/Join Natural ⋈

- Reunión con **condición implícita**: se seleccionan aquellas tuplas cuyos valores coincidan en los atributos con igual nombre.
  - Se eliminan los atributos duplicados
  - Dados R1(R) y R2(S) donde a_i, $1<=i<=k$ son atributos comunes a R1 y R2

    $$R1 \bowtie R2 \equiv \Pi_{R \cup S}(\sigma_{R1.A1=R2.A1 \ldots R1.AK=R2.AK}(R1 \times R2))$$
  - Si no hay atributos comunes, es decir, $R \cap S = \varnothing$

    $$R1 \bowtie R2 \equiv R1 \times R2$$

  - **`DES> r1 njoin r2`**

Reunión/Join natural ⋈ (cont.)

- ***Ejemplos* (ex4.dl, ex5.dl):**

emp (<u>dni</u>, nd) dpto (<u>nd</u>, nomd)

  - emp ⋈ dpto devuelve una relación con los atributos (dni,nd,nomd)

- Hay que tener cuidado al usar la reunión natural porque puede haber otros atributos comunes en los que no pensemos:

emp (<u>dni</u>, **nom**, apes, **nd**) dpto (**<u>nd</u>**, **nom**)

Reunión/Join natural ⋈ (cont.)

- ***Ejemplo:*** dado el siguiente esquema, obtener el nombre de los propietarios de coches modelo 'Ferrari F50'.

- Solución:

r1 ← $\sigma_{\text{modelo = 'Ferrari F50'}}$ (coches)

  - /* Tuplas del modelo 'Ferrari F50' */

r2 ← propiedad ⋈ r1

  - /* DNI de propietarios junto a los datos de sus coches */

res ← π_{nom} (clientes ⋈ r2)

  - /* Nombre de los propietarios de 'Ferrari F50' */

División ÷

- Sean R1 y R2 dos esquemas de relación, con X e Y como conjuntos de atributos respectivamente, para realizar la división debe ocurrir que Y sea subconjunto propio de X

  R1 (X) ÷ R2 (Y)

  - el resultado es una relación r cuyos atributos son: X – Y.
- Una tupla *u* está en R1÷ R2 cuando **para todo** *v* de R2, la tupla que se obtiene al unir los valores de *u* y *v* está en R1:

$R1\ (A_1, A_2, \ldots, A_n) \div R2\ (A_{n-s}, \ldots, A_n) \rightarrow R\ (A_1, A_2, \ldots, A_{n-s-1})$

$<A_1, A_2, \ldots, A_{n-s-1}> \in R$ si $\forall <A_{n-s}, \ldots, A_n> \in R2 \Rightarrow <A_1, \ldots, A_{n-s-1}, A_{n-s}, \ldots, A_n> \in R1$

- Idea
  - Consultas en las que se busca que algún atributo de una relación tome (al menos) todos los valores de otro atributo en otra relación (cuantificación universal).
- **`DES> r1 division r2`**

División ÷ (cont.)

- ***Ejemplo* (ex6.dl):** dado el siguiente esquema de relación, obtener el DNI de los empleados que trabajan en todos los proyectos en los que trabaja el empleado con DNI = 123.

  emp (<u>dni</u>, nom) trab (<u>dni</u>, <u>npro</u>)

- solución:

  $temp \leftarrow \pi_{npro}(\sigma_{dni = 123}(trab))$

  - /* npro de todos los proyectos en los que trabaja dni=123 */

  $res \leftarrow trab \div temp$

  - /* dni tales que todas las concatenaciones de los dni con los npro de temp existe en traB. */

División ÷ (cont.)

TRAB (DNI, NPRO)	
123	1
123	2
456	1
456	2
456	3
987	1

÷

TEMP (NPRO)
1
2

→

RES (DNI)
123
456

División ÷ (cont.)

- ***Ejemplo 2* (ex7.dl):** obtener el DNI de los empleados que trabajan en todos los proyectos en los que trabaja el empleado con DNI = 123.

emp (dni, nom) trab (dni, npro, nhoras)

- Solución 1:
  - r1 ← $\pi_{npro}(\sigma_{dni = 123}(trab))$
    - /* npro de todos los proyectos en los que trabaja dni=123 */
  - res ← TRAB ÷ R1
    - /* **CONJUNTO VACÍO** (probablemente) */

- Solución 2:
  - r1 ← $\pi_{npro}(\sigma_{dni = 123}(trab))$
    - /* npro de todos los proyectos en los que trabaja dni=123 */
  - r2 ← $\pi_{dni, npro}(trab)$
    - /* todas la tuplas dni, npro */
  - res ← r2 ÷ r1
    - /* DNI de los empleados que trabajan todos los proyectos en que trabaja 123 */

Extended Relational Algebra (Álgebra relacional extendida)

- Generalized Projection
- Bags
- Aggregate Functions
- Outer Joins
- Further Operators
- Recursion

Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the projection list.

$$\Pi_{F_1, F_2, \ldots, F_n}(E)$$

- E is any relational-algebra expression
- Each of $F_1, F_2, \ldots, F_n$ are are arithmetic expressions involving constants and attributes in the schema of E.
- Given relation *credit_info(customer_name, limit, credit_balance)*, find how much more each person can spend:

  $\Pi_{customer_name,\ limit - credit_balance}\ (credit_info)$

```
DES> project customer_name,limit-credit_balance (credit_info)
```

Bags

- Multisets.
- Former operations running on bags:

  Given: r(a,b) = {(1,2), (1,2)}, s(a) = {(1)}

  - Cartesian product: r $\times$ s = {(1,2,1), (1,2,1)}

    ```
    DES> /duplicates on
    DES> r product s
    ```

  - Natural join: r $\bowtie$ s = {(1,2), (1,2)}

    ```
    DES> r njoin s
    ```

  - Union: r $\cup$ r = {(1,2)}

    ```
    DES> r union r
    ```

  - Union all: r $\cup_{all}$ r = {(1,2), (1,2), (1,2), (1,2)}

    ```
    DES> r union_all r  % Not yet supported
    ```

- Duplicate elimination operator: distinct (δ)

  - δ (r) = {(1,2)}

    ```
    DES> /ra distinct (r)  % name clash with Datalog distinct
    ```

Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- **Aggregate operation** in relational algebra

$$_{G_1,G_2,\ldots,G_n}\vartheta_{F_1(A_1),F_2(A_2,\ldots,F_n(A_n)}(E)$$

  E is any relational-algebra expression

  - $G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can be empty)
  - Each F_i is an aggregate function
  - Each A_i is an attribute name

Aggregate Operation – Example

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

- $\mathcal{g}_{\text{sum(c)}}(r)$

sum(*c*)
27

```
DES> group_by [] sum(c) true (r)
```

- Question: Which aggregate operations cannot be expressed using basic relational operations?

Aggregate Operation – Example

- Relation *account:*

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

- Grouped by *branch-name*:

$$branch_name\ \mathcal{G}\ branch_name, \mathbf{sum}(balance)\ (account)$$

branch_name	**sum**(*balance*)
Brighton	1500
Perryridge	1300
Redwood	700

```
DES> group_by branch_name branch_name,sum(balance) true (account)
```

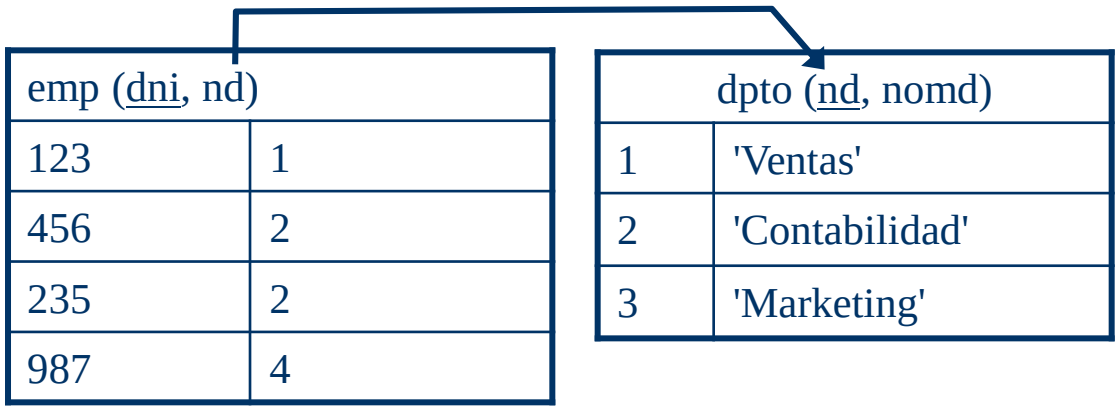
Reunión externa (Outer Join)

Existen tres variantes:

- Izquierda: *R1* ⟕ *R2*,
  - Añade todas las tuplas de la relación de la izquierda que no coinciden con ninguna tupla de la relación de la derecha
  - Estas tuplas aparecerán con los atributos correspondientes a la relación R2 con atributos comunes nulos.
  - **`DES> R1 ljoin cond R2`**
- Derecha: *R1* ⟖ *R2*
  - Análoga a la anterior, cambiando el papel de R1 por el de R2.
  - **`DES> R1 rjoin cond R2`**
- Completa: *R1* ⟗ *R2*
  - La unión de las dos anteriores.
  - **`DES> R1 fjoin cond R2`**

Reunión/Join externa (cont.)

- ***Ejemplo:***



emp (dni, nd)	
123	1
456	2
235	2
987	4

dpto (nd, nomd)	
1	'Ventas'
2	'Contabilidad'
3	'Marketing'

- La operación join natural da como resultado la instancia siguiente:

EMP ⋈ DPTO

```
DES> emp njoin dpto
```

RES (DNI, ND, NOMD)		
123	1	'Ventas'
456	2	'Contabilidad'
235	2	'Contabilidad'

Reunión/Join externa (cont.)

- emp ⟕ dpto

```
DES> emp nrjoin dpto
```

res (dni, nd, nomd)		
123	1	'Ventas'
456	2	'Contabilidad'
235	2	'Contabilidad'
-	3	'Marketing'

- emp ⟖ dpto

```
DES> emp nljoin dpto
```

res (dni, nd, nomd)		
123	1	'Ventas'
456	2	'Contabilidad'
235	2	'Contabilidad'
987	4	-

Observaciones sobre valores nulos

- Los valores nulos afectan a las diferentes operaciones del álgebra relacional.
- Cualquier comparación (=, ≠, <, ≤, >, ≥) que incluya un valor nulo, tiene como resultado un valor ***desconocido***.
- Los valores nulos también pueden aparecer en operaciones booleanas
  - NULO ∧ cierto = desconocido , NULO ∧ falso = falso,
    NULO ∧ NULO = desconocido
  - NULO ∨ cierto = cierto , NULO ∨ falso = desconocido,
    NULO ∨ NULO = desconocido
  - ¬ NULO = desconocido

Observaciones sobre valores nulos (cont.)

- Selección
  - Si la evaluación del predicado sobre una tupla devuelve cierto, la tupla se incluye en el resultado.
  - En otro caso (falso o desconocido), no se incluye.
- Proyección
  - La operación de proyección trata los nulos como cualquier otro valor a la hora de eliminar duplicados.
  - Si una parte de valores de las tuplas coinciden y las dos tienen valores nulos en el resto, se tratan como duplicados.
- Unión, intersección y diferencia
  - Estas operaciones tratan los valores nulos como las proyecciones.

Observaciones sobre valores nulos (cont.)

- Reunión
  - Se utiliza la equivalencia reunión = selección (producto cartesiano)
  - Por tanto dos tuplas con valor nulo en el atributo común no coinciden.
- Reunión externa
  - Funciona como la reunión para las tuplas que cumplan las condiciones de reunión.
  - Para el resto de tuplas, dependiendo del tipo de reunión externa, se incluirán el resto de tuplas rellenando con valores nulos.

Further Operators

- Sort: $\tau_L(R)$
  - Sorts the relation R following the sequence L.
- Example:
  - Sort the relation *account* first by *branch-name* and then by *balance*
  - $\tau_{\textit{branch-name, balance}}(\textit{account})$
  - **`DES> sort branch_name, balance (account)`**
- Ordering criteria:
  - **`DES> sort branch_name asc, balance desc (account)`**

Further Operators

- Top: $\varphi_N\,(R)$
  - Return the first N tuples of the relation R .

- Example:
  - Retrieve the first three tuples from employee:
  - $\varphi_3\,(employee)$
  - **`DES> top 3 (employee)`**

Recursión

- Definiciones recursivas para procedimientos, funciones, ..., ¿y para un lenguaje de consulta de bases de datos?
- El álgebra relacional como se definió originalmente no dio soporte a la recursión
- ... pero no parece muy difícil hacerse una idea de cómo expresarlo:
- De modo análogo a cómo se define una función recursiva para un lenguaje como Java, se puede definir una *relación recursiva:*
  - Una relación es *recursiva* si en su definición interviene el mismo nombre de relación.

Recursión

- Definición de los números naturales en Java:

```
Set<int> nat(int n){
      return union(n, nat(n+1)) ;
}
```

- Caso base: parámetro de llamada
- Caso recursivo: **`nat(n+1)`**
- Resultado (no observable) a la llamada **`nat(0)`**:

  `{0,1,2,...}`

Recursión

- Definición de los números naturales en álgebra relacional:
- Esquema para la relación nat: (n int)

$$\text{nat} \leftarrow \Pi_0(\text{dual}) \cup \Pi_{n+1}(\mathbf{nat})$$

- Caso base: $\Pi_0(\text{dual})$, es decir: {(0)}
- Caso recursivo: $\Pi_{n+1}(\mathbf{nat})$, es decir, dada una relación nat, el resultado de incremetar su atributo en una unidad para cada una de sus tuplas.
- Resultado (no observable) a la consulta $\Pi_n(\text{nat})$, o lo que es lo mismo, a $\sigma_{\text{true}}(\text{nat})$:
  {0,1,2,...}

Cómputo iterativo de la recursión

- Inicialmente nat está vacía y se aplica su definición:

  $\text{nat} \leftarrow \prod_0 (\text{dual}) \cup \prod_{n+1} (\textbf{nat})$

  ```
  DES> nat(n) := project 0 (dual) union project n+1 (nat);
  ```

- Primera iteración: nat = {(0)} ∪ {}
- Se aplica de nuevo su definición sobre el nuevo valor de nat:
- Segunda iteración:

  $\text{nat} = \{(0)\} \cup \prod_0 (\text{dual}) \cup \prod_{n+1} (\textbf{nat}) = \{(0)\} \cup \{(0),(1)\} = \{(0),(1)\}$

- Tercera iteración:

  $\text{nat} = \{(0),(1)\} \cup \prod_0 (\text{dual}) \cup \prod_{n+1} (\textbf{nat}) = \{(0),(1)\} \cup \{(0),(1),(2)\} = \{(0),(1),(2)\}$

- Cuarta iteración:

  $\text{nat} = \{(0),(1),(2)\} \cup \prod_0 (\text{dual}) \cup \prod_{n+1} (\textbf{nat}) = \{(0),(1),(2)\} \cup \{(0),(1),(2),(3)\} = \{(0),(1),(2),(3)\}$

- ....

Cómputo iterativo de la recursión

- ....
- ¿Cuándo terminamos? ¡Nunca!
- Resultado (no observable) a la consulta Π_n (nat), o lo que es lo mismo, a σ_{true} (nat) = {0,1,2,...}
- En DES se puede usar el metapredicado **`top`** (disponible en la mayoría de SGBD's en SQL como **`TOP`** o **`FETCH FIRST n ROWS`**):

```
DES> top 5 (nat);
answer(nat.n:number(integer)) ->
   {answer(0), answer(1), answer(2),
    answer(3), answer(4)}
```

- Ejercicio: Definir una relación que contenga los primeros 100 números naturales, controlando el problema de la parada con una condición.

Un ejemplo recursivo más elaborado

- Java (ejercicio: expresarlo de forma que devuelva un conjunto infinito de pares de un número y su factorial):

```
int factorial(int n){
    if(n==0) return 1;              // Caso base
    else return n*factorial(n-1);   // Caso recursivo
  }
```

- AR:

  factorial(n int, f int)

  factorial $\leftarrow \prod_{0,\,1}(\text{dual}) \cup \prod_{n+1,\,(n+1)*f}(\text{factorial})$

```
DES> factorial(n,f) := project 0,1 (dual) union
                       project n+1,(n+1)*f (factorial);
DES> top 5 (factorial);
```

Recursión: el cierre transitivo

- Dada la relación `tiene_jefe(emp, jefe)`, calcular su cierre transitivo (si `a` es jefe de `b` y `b` lo es de `c`, entonces se deduce que `a` también lo es de `c`, aplicado a todas las tupla tanto originales como a las deducidas):

  tiene_jefe(emp, jefe)

  jefe $\leftarrow \Pi_{emp,\ jefe}$ (tiene_jefe) $\cup$

  $\Pi_{e1,\ j2}(\rho_{j1(e1,j1)}$ (tiene_jefe) $\rtimes_{j1=e2}$ $\rho_{j2(e2,j2)}$ (jefe))

```
DES> :-type(tiene_jefe(emp:string,jefe:string)).
DES> jefe(emp,jefe) := project emp,jefe (tiene_jefe) union
      project e1,j2
      (rename j1(e1,j1) (tiene_jefe) zjoin j1=e2
       rename j2(e2,j2) (jefe));
```

- Ejercicio: Escribir el cómputo iterativo para calcular la relación **`jefe`** asumiendo la instancia **`tiene_jefe={('a','b'),('b','c'),('c','d')}`**

Recursión: linealidad

- Si la definición de una relación recursiva contiene solo una aparición del nombre de la relación en la parte derecha del operador de asignación, la recursión se denomina *lineal*.
- Si hay más de una aparición de este nombre, la recursión se denomina *no lineal*.
- Se puede reformular el ejemplo anterior para convertirlo en no lineal:

  tiene_jefe(emp, jefe)

  $$\text{jefe} \leftarrow \Pi_{\text{emp, jefe}} (\text{tiene_jefe}) \cup \Pi_{\text{e1, j2}}(\rho_{\text{j1(e1,j1)}} (\text{jefe}) \bowtie_{\text{j1=e2}} \rho_{\text{j2(e2,j2)}} (\text{jefe}))$$

```
DES> :-type(tiene_jefe(emp:string,jefe:string))
DES> jefe(emp,jefe) := project emp,jefe (tiene_jefe) union
      project e1,j2
      (rename j1(e1,j1) (jefe) zjoin j1=e2
       rename j2(e2,j2) (jefe));
```

- Los sistemas SQL actuales admiten a lo sumo recursión lineal (aunque DES admite la no lineal).

Recursión mutua

- No es posible expresar la recursión mutua en álgebra relacional, dado que no se puede hacer referencia a una relación que no se haya definido aún.
- Nos gustaría resolver cosas como:

$$\text{par}(x) \leftarrow \Pi_0(\text{dual}) \cup \Pi_{x+1}(\text{impar})$$

$$\text{impar}(x) \leftarrow \Pi_{x+1}(\text{par})$$

- Pero esto se puede resolver en DES con el siguiente "truco":

```
impar(x):= select true (impar);
par(x)  := project 0 (dual) union project x+1 (impar);
impar(x):= project x+1 (par);

top 5 (par);
top 5 (impar);
```

Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- All these operations are expressed using the assignment operator.
- RA in DES does not support DML queries, but both SQL and Datalog in DES do.

Deletion

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.
- Can delete only whole tuples; cannot delete values on only particular attributes
- A deletion is expressed in relational algebra by:

$$r \leftarrow r - E$$

  where r is a relation and E is a relational algebra query.
- Don't do this at DES!

```
DES> r := r difference e
```

  - Recursive definition for R!
  - DES does not enjoy DML sentences, instead use assertion **`/retract`** for single tuples and SQL **`DELETE FROM`**

Deletion Examples

- Delete all account records in the Perryridge branch.

$$account \leftarrow account - \sigma_{branch_name = 'Perryridge'}(account)$$

- Delete all loan records with amount in the range of 0 to 50

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

- Delete all accounts at branches located in Needham.

$$r_1 \leftarrow \sigma_{branch_city = 'Needham'}(account \bowtie branch)$$
$$r_2 \leftarrow \Pi_{account_number,\ branch_name,\ balance}(r_1)$$
$$r_3 \leftarrow \Pi_{customer_name,\ account_number}(r_2 \bowtie depositor)$$
$$account \leftarrow account - r_2$$
$$depositor \leftarrow depositor - r_3$$

Insertion

- To insert data into a relation, we either:
  - specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted
- in relational algebra, an insertion is expressed by:

$$r \leftarrow r \cup E$$

  where r is a relation and E is a relational algebra expression.
- The insertion of a single tuple is expressed by letting E be a constant relation containing one tuple.
- In DES use assertion **`/assert`** for single tuples and SQL **`INSERT INTO`**

Insertion Examples

- Insert information in the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.

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

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

- Provide as a gift for all loan customers in the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account.

  $r_1 \leftarrow (\sigma_{branch_name = 'Perryridge'} (borrower \bowtie loan))$

  $account \leftarrow account \cup \prod_{loan_number,\ branch_name,\ 200} (r_1)$

  $depositor \leftarrow depositor \cup \prod_{customer_name,\ loan_number} (r_1)$

Updating

- A mechanism to change a value in a tuple without charging *all* values in the tuple
- Use the generalized projection operator to do this task

$$r \leftarrow \prod_{F_1, F_2, \ldots, F_l}(r)$$

- Each F_i is either
  - the I^{th} attribute of r, if the I^{th} attribute is not updated, or,
  - if the attribute is to be updated F_i is an expression, involving only constants and the attributes of r, which gives the new value for the attribute
- In DES use assertion SQL **`UPDATE`**

Update Examples

- Make interest payments by increasing all balances by 5 percent.

$$account \leftarrow \Pi_{account_number,\ branch_name,\ balance * 1.05}(account)$$

- Pay all accounts with balances over $10,000 6 percent interest and pay all others 5 percent

$$account \leftarrow \Pi_{account_number,\ branch_name,\ balance * 1.06}(\sigma_{BAL > 10000}(account)) \cup \Pi_{account_number,\ branch_name,\ balance * 1.05}(\sigma_{BAL \leq 10000}(account))$$