Universidad de Granada

decsai.ugr.es

Fundamentos de Bases de Datos

Grado en Ingeniería Informática

Seminario: Cálculo relacional

DECSAI

Departamento de Ciencias de la Computación e Inteligencia Artificial

1. Introducción
2. El cálculo de predicados como lenguaje de representación de información
3. Cálculo relacional orientado a tuplas
4. Correspondencia entre operadores

1. Introducción
2. El cálculo de predicados como lenguaje de representación de información
3. Cálculo relacional orientado a tuplas
4. Correspondencia entre operadores

Inicialmente:

- Elementos lógicos en el modelo relacional (Codd 1971).
  - Cálculo relacional orientado a tuplas. Lenguaje Alpha.
  - Equivalencia entre enfoques de consulta.
- Extensiones e implementaciones iniciales:
  - Lacroix y Pirrotte (1977). Primera versión del Cálculo Relacional Orientado a Dominios.
  - Implementaciones: QUEL(1980), QUERY_BY_EXAMPLE (1985).

1. Introducción
2. El cálculo de predicados como lenguaje de representación de información
3. Cálculo relacional orientado a tuplas
4. Correspondencia entre operadores

Idea básicas:

- El cálculo de predicados surge como sistema de representación del conocimiento en I.A.

- Elementos de un sistema de representación del conocimiento:
  - Una Base de Conocimiento donde se almacena conocimiento a distintos niveles.
  - Un mecanismo de inferencia que permite derivar un conocimiento de otro.

- Para representar la información en la base de conocimiento los formalismos de la Lógica son muy adecuados ya que incluyen:
  - Mecanismos de representación.
  - Mecanismos de derivación de conocimiento.

- Entre los formalismos basados en la Lógica:
  - Cálculo de Proposiciones.
  - Cálculo de Predicados.
  - Formalismos Lógicos más avanzados:
    - Redes semánticas.
    - Lógica multivaluada.
    - Razonamiento por defecto, basado en casos etc..

Definición formal: ideas básicas

- Un lenguaje de Cálculo de Predicados se define para describir un "mundo". Debe tener símbolos y frases.
- Este lenguaje debe incluir un alfabeto donde haya:
  - Símbolos para describir los objetos del mundo (constantes)
    - Juan, José, ..., Seat,..., Rojo,..., GR-150-A...
  - Símbolos para describir funciones que nos dan unos objetos en función de otros:
    - Color, Padre, Madre, Propietario etc
  - Símbolos para describir variables: x, y, z,
  - Símbolos de predicados que describan relaciones entre objetos:
    - Casados, Conduce, Prefiere.
  - Los predicados describen relaciones binarias, ternarias etc...

- El lenguaje además de símbolos tendrá que generar "frases" (fórmulas, expresiones…) por ello debe tener
  - Símbolos de puntuación: ( ) , . ; [ ]
  - Conectores: $\wedge$, $\vee$, $\neg$, $\rightarrow$
  - Cuantificadores: $\forall$, $\exists$

Definición formal: Un lenguaje de Cálculo de Predicados se define como L=(S, W) donde

- S es un conjunto de símbolos incluyendo:
  - Constantes
  - Funciones
  - Variables
  - Predicados
  - Símbolos adicionales
- W un conjunto de frases "que están bien escritas" o fórmulas bien formadas (Well formed formulae) o "**wff**"
- W se define de forma recursiva:
  - Un átomo a se define como:
    - Un símbolo de constante (José) ó
    - Un símbolo de variable (x) ó
    - *f(a)* donde *f* es un símbolo de función y *a* un átomo: Padre(José), Color(x)

- Una formula atómica se define como $p(a_1,a_2,..a_n)$ donde:
  - p es un símbolo de predicado n-ario
  - $a_1,a_2,....a_n$ son átomos
- Toda formula atómica es **wff** ($\in W$)
- Si $f_1,f_2 \in W$ entonces:
  - $f_1 \vee f_2 \in W$; $f_1 \wedge f_2 \in W$; $\neg f_1 \in W$; $f_1 \rightarrow f_2 \in W$
- Si $f_1(x) \in W$ entonces:
  - $\forall x\ f_1(x) \in W$; $\exists x\ f_1(x) \in W$

– Algunos ejemplos de **wffs**:
  - casados(Juán, Ana), casados(padre(x),madre(x)),prefiere(Ana,Honda,rojo)
  - conduce(Juan,GR-150-A)^¬conduce(propietario(GR-150-A),GR-150-A)
  - ∀x coche(x)→prefiere(propietario(x),marca(x),color(x))
  - ∃y (persona(y) ^¬casados(padre(y),madre(y)))

– Toda variable en una **wff** que no esté cuantificada se denomina variable libre

– Toda variable en una **wff** que esté cuantificada se denomina variable ligada

Interpretación de un lenguaje

Idea básica: un lenguaje L=(A,W) es una abstracción formal que puede describir muchas realidades. Para describir una concreta es necesario asociar

- Símbolos de constantes con objetos del mundo
- Predicados con relaciones concretas entre objetos

Formalmente:

- Sea L=(A,W) un lenguaje de CP ;$C\subset A$ es el conjunto de constantes
- Llamaremos interpretación I de L al triple I=(D,K,E), donde
    - *D* es un "universo de discurso": conjunto de objetos asociados a una realidad
    - $K:C\rightarrow D$ y permite asociar las constantes de *A* a objetos reales.
    - *E* se denomina "función extensión" y asocia a todo predicado n-ario $p\in A$ un conjunto $E(p)\subseteq D^n$. *E*(*p*) se denomina extensión de *p* en *I*.
    - A partir de ahora cuando hablemos de una interpretación identificaremos cada objeto con su nombre es decir, $\forall c\in C$ identificaremos *c* y *K*(*c*)

Interpretación de un lenguaje

Valor de verdad: toda interpretación I=(D,K,E) de un lenguaje L=(A,W) permite asociar valores de verdad a ciertas wffs de W.

- Toda wff que no incluya variables tiene un valor de verdad:
  - Toda fórmula atómica de la forma $P(c_1,..c_n)$ con $c_i \in C$ o $c_i=f(d_i)$ con $d_i \in C$ es cierta sii $(c_1,c_2...c_n) \in E(P)$, en caso contrario es falsa.
  - Sean $f_1,f_2 \in W$, el valor de verdad de:

    $$f_1 \vee f_2,\ f_1 \wedge f_2,\ \neg f_1,\ \text{y}\ f_1 \rightarrow f_2$$

  se calcula de acuerdo con las reglas del "or" "and" y "not", y not (f_1) or f_2 supuestos conocidos los valores de verdad de f_1 y f_2
- Toda wff que tenga todas sus variables ligadas tiene un valor de verdad de acuerdo con:
  - $\forall x\ f_1(x)$ es cierta si $f_1(c)$ es cierta $\forall c \in C$
  - $\exists x\ f_1(x)$ es cierta si $\exists c \in C$ para la que $f_1(c)$ es cierta
  - Equivalencia entre $\forall$ y $\exists$: $\forall x\ f_1(x) = \neg \exists x \neg f_1(x)$
- Una wff que tenga alguna variable libre genera un conjunto de constantes que son aquellas que hacen cierta la formula sustituyendo la variable por ellas.

Interpretación de un lenguaje

Modelo: dada una interpretación I=(D,K,E) de un lenguaje L=(A,W) y M⊂W, I es un modelo de M si toda f∈M es cierta con respecto a I.

Ejemplos:

- casados(Juan,Ana) será cierta si (Juan,Ana)∈E(casados)
- prefiere(Ana,Honda,rojo) es cierta si (Ana,Honda,rojo)∈E(prefiere)
- conduce(Juan,GR-150-A)^¬conduce(propietario(GR-150-A),GR-150-A) sera cierta si conduce(Juan,GR-150-A) es cierta y conduce(propietario(GR-150-A),GR-150-A) es falsa
- ∃y (persona(y) ^¬casados(padre(y),madre(y))) será cierta si podemos encontrar una constante c tal que
  - (persona(c) ^¬casados(padre(c),madre(c))) sea cierta
- x | casados(padre(x),madre(x)) define un conjunto de constantes

Ideas básicas:

- Un lenguaje de calculo de predicados y un modelo relacional son estructuras formales para describir la realidad: ambas pueden identificarse.
- Una instancia de una base de datos se identificaría entonces con una interpretación de su lenguaje asociado.
- Las reglas de integridad serían wffs y la interpretación debería ser un modelo para ellas.
- Las consultas se generarían mediante wffs con variables libres. Los conjuntos de constantes que las hacen ciertas serán la solución de la consulta.

Identificación en el caso del Cálculo Relacional Orientado a Tuplas

Calculo de Predicados	Modelo Relacional
• Universo de discurso	• Dominios
• Variables	• Tuplas
• Predicados	• Relaciones y Operadores de comparación
• Funciones	• Atributos

Cambio de notación:

- Operadores de comparación: notación de operador
  - =(a,b) se sustituye por a=b, etc... (=, <=, <=,....)
- Funciones: notación de atributo
  - f(x) se sustituye por x.f

Definición de una consulta:

- Consideremos una base de datos con relaciones $R(A_1,\ldots A_n)$, $S(B_1,\ldots B_m)$, etc, y le asociamos un lenguaje de Cálculo de Predicados.
- Supongamos que R_x, R_y … S_x, S_y…, son variables que toman valores en R, S etc.. denominadas variables tupla.
- Una consulta en C.R. Orientado a Tuplas (lenguaje QUEL) tiene la forma:

$$\mathbf{Select}\ R_x.A_i,\ R_x.A_j\ \ldots,\ R_y.A_h,\ S_z.B_l,\ldots\quad \mathbf{Where}\ wff(R_x,R_y,S_z\ldots)$$

  - $R_x.A_i,\ R_x.A_j\ \ldots,\ R_y.A_h, S_z.B_l,\ldots$ se denomina “lista objetivo”.
  - $wff(R_x,R_y,S_z\ldots)$ es una fórmula cuyas variables libres aparecen en la lista objetivo.
  - La particularización de la lista objetivo para las tuplas que hacen cierta esta fórmula nos da la solución a la consulta.

Definición de una consulta:

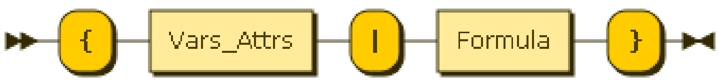
– Una consulta en C.R. Orientado a Tuplas (WinRDBI) tiene la forma:

$$\{X.A_i,\ X.A_j\ ...,\ Y.A_h,\ Y.B_l, ... \mid wff(R(X),R(Y),S(Z),X,Y,Z,\ldots)\}$$

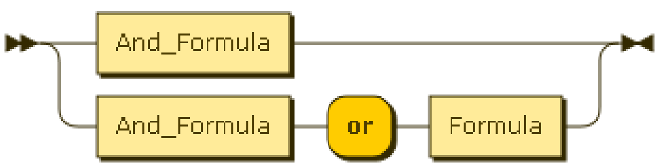
- $X.A_i,\ X.A_j\ ...,\ Y.A_h,\ Y.B_l, ...$ se denomina "lista objetivo" .
- $wff(R(X),R(Y),S(Z),X,Y,Z,\ldots)$ es una fórmula en la que **R(X),R(Y),S(Z)** declara las variables libres **X,Y** para la relación **R** y la variable libre **Z** para la relación **S** .
- La particularización de la lista objetivo para las tuplas que hacen cierta esta fórmula nos da la solución a la consulta.

Sintaxis para WinRDBI

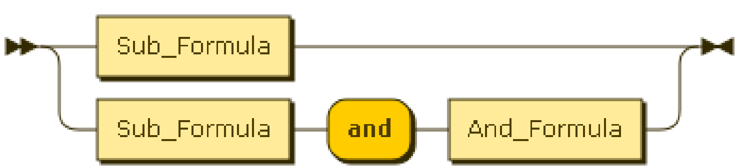
Query:



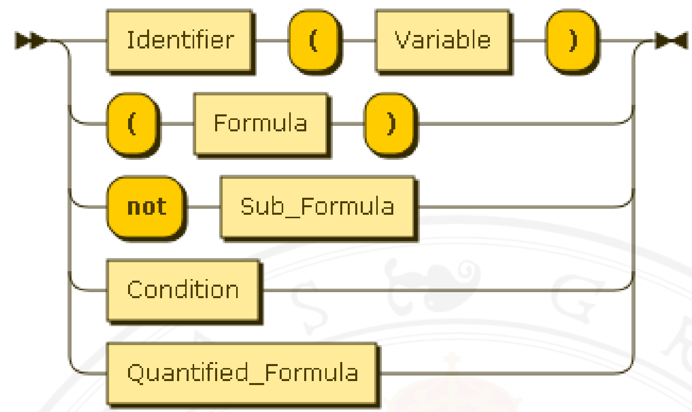
Formula:



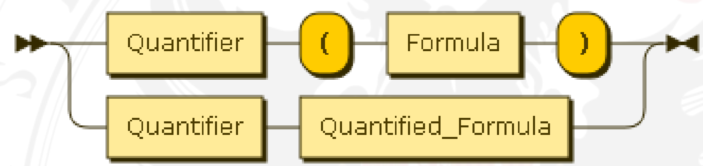
And_Formula:



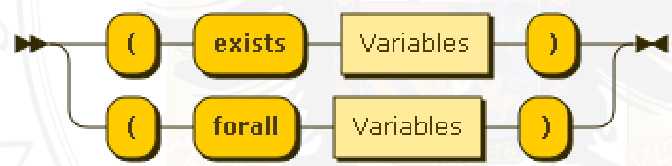
Sub_Formula:



Quantified_Formula:



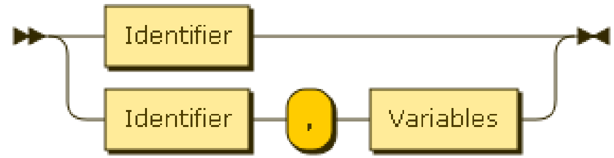
Quantifier:



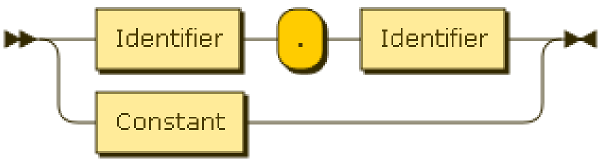
Sintaxis para WinRDBI

Condition:

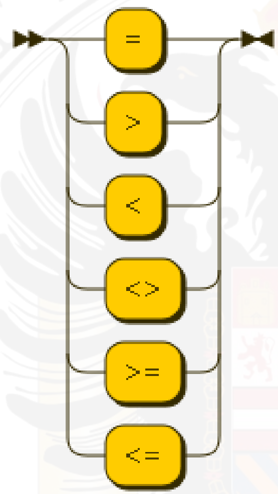
Variables:



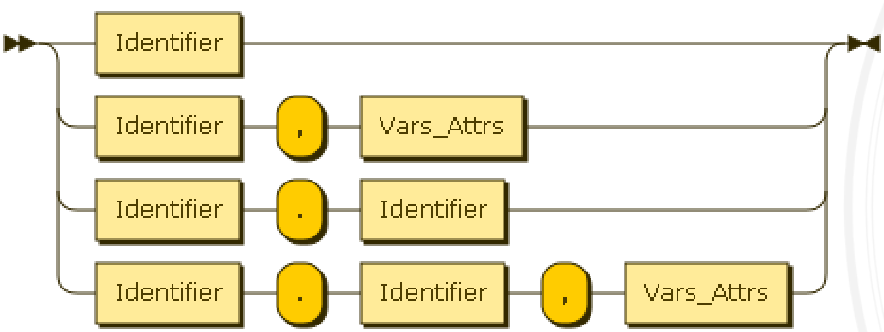
Operand:



Relational_Op:



Vars_Attrs:



ALUMNOS

DNI	NOMB_ALUM	FECHA_NAC	PROVINCIA	BECA

ASIGNATURAS

COD_ASIG	NOMB_ASIG	CREDITOS	CARACTER	CURSO

PROFESORES

NRP	NOM_PROF	CATEGORIA	AREA	COD_DEP

DEPARTAMENTOS

COD_DEP	NOM_DEP	DIRECTOR

AULAS

COD-AULA	CAPACIDAD

GRUPOS

COD_GRUP	COD_ASIG	TIPO	NRP	MAX_AL

CLASE

COD_AULA	DIA	HORA	COD_GRUP	COD_ASIG	TIPO

MATRICULAS

COD_GRUP	COD_ASIG	TIPO	DNI	CONVOCATORIA	CALIFICACION

Ejemplo:

- Modelo relacional
  - Asignaturas(Cod_Asig,Nom_asig, Creditos, Carácter, Curso)
  - Profesores(NRP, Nom_Prof, Categoria, Area, Cod_Dep)
  - Departamentos(Cod_Dep,Nom_Dep, Director)
  - Grupos(Cod_Asig, Cod_Grup, Tipo, NRP, Max_Al)

- Lenguaje
  - Constantes: CCIA, LSI, TU, CU, etc.
  - Variables:
    - **QUEL: Range A_x, A_y ... in Asignaturas, Range P_x, P_y, ...in Profesores,…**
    - **WinRDBI: Profesores(Px), Profesores(Py), Asignaturas(Ax)**
  - Funciones: Px.NRP, Ax.cod_asig,...
- Consultas
  - **QUEL: SELECT A_x.cod_asig, A_x.nom_asig, A_x.creditos WHERE A_x.curso=2**
  - **WinRDBI: {A.cod_asig, A.nom_asig, A.creditos | Asignaturas(A) and A.curso=2};**
  - **QUEL: SELECT A_x.cod_asig,A_x.nom_asig WHERE ∃G_y(G_y.cod_asig=A_x.cod_asig ^ G_y.tipo='Teoria' ^ G_y.max_al>=60)**
  - **WinRDBI: {A.cod_asig,A.nom_asig | Asignaturas(A) and (exists G)(Grupos(G) and G.cod_asig=A.cod_asig and G.tipo='Teoria' and G.max_al>=60)};**

"Encontrar los datos de aquellos profesores que son asociados":

QUEL:
```
RANGE Px IN profesores
    SELECT Px.* WHERE Px.categoria='AS'
```

WinRDBI:
```
{ P| profesores(P) and P.categoria='AS'};
```

Álgebra: $\sigma_{categoria='AS'}$ (profesores)

"Encontrar el nombre de aquellos profesores que son asociados":

QUEL:
```
RANGE Px IN profesores
SELECT Px.nom_prof WHERE Px.categoria='AS'
```

WinRDBI:
```
{ P.nom_prof| profesores(P) and P.categoria='AS'};
```

Álgebra: $\Pi_{nom\_prof}(\sigma_{categoria='AS'}$ (profesores))

"Para cada profesor encontrar el nombre del departamento en el que trabaja":

"Para cada nombre de departamento
encontrar el código del profesor que trabaja en ese departamento"

WinRDBI: `{ P.NRP, D.nom_dep | profesores(P) and departamentos(D) and P.cod_dep=D.cod_dep};`

QUEL:
```
RANGE Px IN profesores
RANGE Dx IN departamentos
SELECT Px.NRP, Dx.nom_dep
WHERE Px.cod_dep=Dx.cod_dep
```

Álgebra: $\Pi_{\text{NRP, mom\_dep}}(\text{profesores} \bowtie \text{departamentos})$

"Encontrar los nombres de los profesores que imparten, tanto la asignatura 'FBD', como la asignatura 'TA' " :

"Encontrar los nombres de los profesores para los que existe un grupo de 'FBD ' y un grupo de 'TA' impartido por ellos"

WinRDBI:
```
{P.nom_prof | profesores(P)
and (exists Ax,Ay) (grupos(Ax) and grupos(Ay) and Ax.cod_asig='FBD'
   and Ay.cod_asig='TA'
and Ax.NRP=P.NRP and Ay.NRP=P.NRP)};
```

QUEL:
```
RANGE Px IN profesores
RANGE Ax, Ay IN grupos
SELECT Px.nom_prof
WHERE (∃Ax,Ay ((Ax.cod_asig='FBD') ˄ (Ay.cod_asig='TA') ˄
(Ax.NRP=Px.NRP) ˄ (Ay.NRP=Px.NRP)))
```

Álgebra: Π_{nom_prof} (profesores $\bowtie$ ((Π_{NRP} ($\sigma_{cod\_asig='FBD'}$ (grupos))) $\cap$ (Π_{NRP} ($\sigma_{cod\_asig='TA'}$ (grupos)))))

"Encontrar los nombres de los profesores que imparten la asignatura 'FBD' o la asignatura 'TA' " :

"Encontrar los nombres de los profesores para los que existe un grupo de 'FBD ' o un grupo de 'TA' impartido por ellos"

WinRDBI: {P.nom_prof | profesores(P) and (exists A) (grupos(A) and A.cod_asig='FBD' or A.cod_asig='TA' and A.NRP=P.NRP)};

QUEL: RANGE Px IN profesores
RANGE Ax IN grupos
SELECT Px.nom_prof
WHERE (∃Ax ((Ax.cod_asig='FBD') ˅ (Ax.cod_asig='TA') ˄ (Ax.NRP=Px.NRP)))

Álgebra: Π_{nom_prof} (profesores ⋈ ($\sigma_{cod\_asig='FBD' \vee cod\_asig='TA'}$ (grupos)))

"Encontrar el nombre de aquellos profesores que no imparten asignaturas prácticas" :

"Encontrar los nombres de los profesores para los que no existe un grupo de tipo 'Practica' impartido por ellos"

WinRDBI:
```
{P.nom_prof | profesores(P)
and not (exists A) (grupos(A) and A.tipo='Practica'
and A.NRP=P.NRP)};
```

QUEL:
```
RANGE Px IN profesores
RANGE Ax IN grupos
SELECT Px.nom_prof
WHERE ¬(∃Ax ((Ax.tipo='Practica') ˅
(Ax.NRP=Px.NRP)))
```

Álgebra: Π_{nom_prof} (profesores $\bowtie$ (Π_{NRP}(profesores) – Π_{NRP}($\sigma_{tipo=practica}$ (grupos))))

“Encontrar parejas de profesores que sean del mismo departamento”:

“Encontrar los nombres de las parejas de profesores para los que el departamento sea el mismo”

WinRDBI: `{Px.nom_prof, Py.nom_prof | profesores(Px) and profesores (Py) and`
`Px.cod_dep = Py.cod_dep and Px.NRP<Py.NRP};`

QUEL: `RANGE Px,Py IN profesores`
`SELECT Px.nom_prof, Py.nom_prof`
`WHERE ((Px.cod_dep = Py.cod_dep) ˄ (Px.NRP<Py.NRP))`

Álgebra: $\rho(\texttt{profesores}) = \texttt{profes}$

$\Pi_{\texttt{profesores.nom\_prof, profes.nom\_prof}} (\sigma_{\texttt{profesores.cod\_dep=profes.cod\_dep} \wedge \texttt{profesores.NRP<profes.NRP}} (\texttt{profesores} \times \texttt{profes}))$

- ***"Encontrar las asignaturas en las que dan clase todos los profesores del área 'COMPUT' " :***
- $(\forall x\, f_1(x) = \neg\, \exists x\, \neg\, f_1(x)$; y $f_1 \rightarrow f_2 = \neg f_1 \vee f_2)$

  "Encontrar los códigos de asignaturas tal que
  para todo profesor que sea del área de 'COMPUT',
  implica que existe una impartición de esa asignatura para ese prof."

WinRDBI:
```
{A.cod_asig | asignaturas(A) and
(forall P) (profesores(P) and (not (P.area='COMPUT') or
(exists G)(grupos(G) and G.cod_asig=A.cod_asig and G.NRP=P.NRP)))};
```

QUEL:
```
RANGE P IN profesores; RANGE A IN asignaturas
RANGE G IN grupos
SELECT A.cod_asig
WHERE ∀ P (P.area='COMPUT'
→ ∃ G (G.cod_asig=A.cod_asig ∧ G.NRP=P.NRP))
```

Álgebra: $(\Pi_{cod\_asig,\ NRP}\ (grupos) \div (\Pi_{NRP}\ (\sigma_{area='COMPUT'}\ (profesores)))$

- ***"Encontrar las asignaturas en las que dan clase todos los profesores del área 'COMPUT' " :***
- ($\forall x\ f_1(x) = \neg\ \exists x\ \neg\ f_1(x)$; y $f_1 \rightarrow f_2 = \neg f_1 \vee f_2$)

"Encontrar los códigos de asignaturas

para las que no existe un profesor que sea del área de 'COMPUT',

para el que no exista una impartición de esa asignatura para ese prof."

WinRDBI:
```
{A.cod_asig | asignaturas(A) and
not (exists P) (profesores(P) and P.area='COMPUT' and
not (exists G)(grupos(G) and G.cod_asig=A.cod_asig and
   G.NRP=P.NRP))};
```

QUEL:
```
RANGE P IN profesores; RANGE A IN asignaturas
RANGE G IN grupos
SELECT A.cod_asig
WHERE ¬∃ P (¬∃G (P.area='COMPUT' ˄
(G.cod_asig=A.cod_asig ˄ G.NRP=P.NRP)))
```

Álgebra: $(\Pi_{cod\_asig,\ NRP}\ (grupos) \div (\Pi_{NRP}\ (\sigma_{area='COMPUT'}\ (profesores)))$

Ejemplos

Trabajadores (id_trabajador, nombre, trf_hr, tipo_de_oficio, id_supv)

Edificios (id_edificio, dir_edificio, tipo, nivel_calidad, categoria)

Asignaciones (id_trabajador, id_edificio, fecha_inicio, num_dias)

Oficios (tipo_de_oficio, prima, horas_por_sem)

"Encontrar los datos de aquellos trabajadores que son electricistas":

QUEL:
```
RANGE Tx IN Trabajadores
SELECT Tx.* WHERE Tx.tipo_de_oficio='Electricista'
```

WinRDBI:
```
{ T | Trabajadores(T) and
  T.tipo_de_oficio='Electricista'};
```

Álgebra: $\sigma_{\text{tipo\_de\_oficio='Electricista'}}$ TRABAJADORES

"Encontrar el nombre de aquellos trabajadores que son electricistas":

QUEL:
```
RANGE Tx IN Trabajadores
SELECT Tx.nombre
WHERE Tx.tipo_de_oficio='Electricista'
```

WinRDBI:
```
{ T.nombre | Trabajadores(T) and
  T.tipo_de_oficio='Electricista'};
```

Álgebra: $\Pi_{\text{nombre}}(\sigma_{\text{tipo\_de\_oficio='Electricista'}}$ TRABAJADORES$)$

"Encontrar el número de horas semanales que trabaja cada trabajador":

QUEL:
```
RANGE Tx IN Trabajadores
RANGE Ox IN Oficios
SELECT Tx.nombre, Ox.horas_por_sem
WHERE Tx.tipo_de_oficio=Ox.tipo_de_oficio
```

WinRDBI:
```
{T.nombre, O.horas_por_sem |
Trabajadores(T) and Oficios(O) and
T.tipo_de_oficio=O.tipo_de_oficio};
```

Álgebra: $\Pi_{nombre,horas\_por\_sem}$ (TRABAJADORES $\bowtie$ OFICIOS)

“Encontrar los nombres de trabajadores que han trabajado tanto en el edificio 312 como en el edificio 460”:

QUEL: RANGE Tx IN Trabajadores
RANGE Ax, Ay IN Asignaciones
SELECT Tx.nombre
WHERE (∃Ax,Ay ((Ax.id_trabajador=Tx.id_trabajador) ˄
(Ay.id_trabajador=Tx.id_trabajador) ˄
(Ax.id_edificio=312) ˄ (Ay.id_edificio=460))

WinRDBI: {T.nombre | trabajadores(T) and (exists Ax,Ay)(
Asignaciones(Ax) and Asignaciones(Ay) and
Ax.id_trabajador=T.id_trabajador and
Ay.id_trabajador=T.id_trabajador and
Ax.id_edificio=312 and Ay.id_edificio=460)};

Álgebra: Π_{nombre} (TRABAJADORES ⋈
(($\Pi_{id\_trabajador}$ $\sigma_{id\_edificio=312}$ ASIGNACIONES) ∩
($\Pi_{id\_trabajador}$ $\sigma_{id\_edificio=469}$ ASIGNACIONES)))

“Encontrar los nombres de trabajadores que han trabajado o en el edificio 312 o en el edificio 460”:

QUEL:
```
RANGE Tx IN Trabajadores
RANGE Ax IN Asignaciones
SELECT Tx.nombre
WHERE ∃Ax ((Ax.id_trabajador=Tx.id_trabajador) ˄
  ((Ax.id_edificio=312) ˅ (Ax.id_edificio=460)))
```

WinRDBI:
```
{T.nombre | trabajadores(T) and (exists A)(
Asignaciones(A) and
A.id_trabajador=T.id_trabajador and
A.id_edificio=312 or A.id_edificio=460)};
```

Álgebra: $\Pi_{nombre}(\text{TRABAJADORES} \bowtie \sigma_{id\_edificio=312 \vee id\_edificio=460}\ \text{ASIGNACIONES})$

"Encontrar el nombre de aquellos trabajadores que no han trabajado en el edificio 312":

QUEL: RANGE Tx IN Trabajadores

RANGE Ax IN Asignaciones

SELECT Tx.nombre

WHERE ¬(∃Ax ((Ax.id_trabajador=Tx.id_trabajador) ˄ (Ax.id_edificio=312)))

WinRDBI:{ T.nombre | trabajadores(T) and not (exists A) (Asignaciones(A) and A.id_trabajador=T.id_trabajador and Ax.id_edificio=312)};

Álgebra: Π_{nombre} (TRABAJADORES $\bowtie$($\Pi_{id_trabajador}$TRABAJADORES - $\Pi_{id\_trabajador}$($\sigma_{id\_edificio=312}$ ASIGNACIONES)))

"Encontrar parejas de trabajadores que tengan el mismo oficio":

QUEL: RANGE Tx, Ty IN Trabajadores

SELECT Tx.nombre, Ty.nombre

WHERE (Tx.tipo_de_oficio=Ty.tipo_de_oficio) ∧ (Tx.nombre<Ty.nombre))

WinRDBI: {Tx.nombre, Ty.nombre | Trabajadores(Tx) and Trabajadores(Ty) and Tx.tipo_de_oficio=Ty.tipo_de_oficio and Tx.nombre<Ty.nombre};

Álgebra: $\Pi_{\texttt{Tx.nombre, Ty.nombre}}(\sigma_{\texttt{Tx.tipo\_de\_oficio=Ty.tipo\_de\_oficio} \wedge \texttt{Tx.nombre<Ty.nombre}}(\rho_{Tx}(\text{Trabajadores}) \times \rho_{Ty}(\text{Trabajadores})))$

"Encontrar aquellos edificios en los que han trabajado todos los trabajadores de la empresa":

QUEL: RANGE Tx IN Trabajadores
RANGE Ex IN Edificios
RANGE Ax IN Asignaciones
SELECT Ex.*
WHERE ¬∃Tx (¬∃Ax ((Ax.id_trabajador=Tx.id_trabajador) ˄ (Ax.id_edificio=Ex.id_edificio)))

WinRDBI: { E | Edificios(E) and not (exists T) (Trabajadores(T) and not (exists A) (Asignaciones(A) and A.id_trabajador=T.id_trabajador and A.id_edificio=E.id_edificio))};

Álgebra: $(\Pi_{\text{id\_edificio, id\_trabajador}}$ ASIGNACIONES$)$

$\div$

$(\Pi_{\text{id\_trabajador}}$ TRABAJADORES$)$

1. Introducción
2. El cálculo de predicados como lenguaje de representación de información
3. Cálculo relacional orientado a tuplas
4. Correspondencia entre operadores

Álgebra	SQL	Cálc. WinRDBI	Cálc. QUEL
		Variables de Tupla: r(R), r(S)	Variables de Tupla: RANGE Rx IN R RANGE Sx IN S
Proyección: $\Pi_{A,B}(R)$	SELECT **A,B** FROM R	{**R.A,R.B** | r(R)}	SELECT **Rx.A,Rx.B**
Ej.: *"Mostrar el código y el nombre de todos los proveedores"*			
$\Pi_{codpro,nompro}$(Proveedor)	SELECT **codpro,nompro** FROM Proveedor	{**S.codpro,S.nompro** | proveedor(S)}	RANGE Sx IN Proveedor SELECT **Sx.codpro,Sx.nompro**
Selección: $\sigma_{A\theta K}(R)$	SELECT * FROM R **WHERE AθK**	{R | r(R) and **R.AθK**}	SELECT Rx.* WHERE **Rx.AθK**
Ej.: *"Mostrar los proveedores de Paris"*			
$\sigma_{ciudad='Paris'}$(Proveedor)	SELECT * FROM Proveedor **WHERE ciudad='Paris'**	{S | proveedor(S) and **S.ciudad='Paris'**}	SELECT Sx.* WHERE **Sx.ciudad='Paris'**
Producto Cart.: R×S	SELECT * **FROM R,S**	{**R,S** | r(R) and r(S)}	SELECT **Rx.*,Sx.***
Ej.: *"Mostrar todas las combinaciones proveedor-pieza"*			
Proveedor × Pieza	SELECT * **FROM Proveedor,Pieza**	{**S,P** | Proveedor(S) and Pieza(P)}	RANGE Sx IN Proveedor RANGE Px IN Pieza SELECT **Sx.*,Px.***

Seminario: Cálculo relacional

Correspondencia entre operadores

Álgebra	SQL	Cálc. WinRDBI	Cálc. QUEL
Alias Def.: ρ(R)=Rx	SELECT * **FROM R Rx,R Ry**	{Rx,Ry | **r(Rx) and r(Ry)**}	**RANGE Rx,Ry IN R**
	Ej.: *“Mostrar todas las combinaciones de cada pieza con cada pieza”*		
ρ(Pieza)=P Pieza×P	SELECT * **FROM Pieza P1, Pieza P2**	{**P1,P2** | pieza(P1) and pieza(P2)}	RANGE **P1,P2** IN **Pieza** SELECT P1.*,P2.*
Unión, Intersección y Diferencia: R y S compatibles respecto a la Unión. R(<u>A</u>,B);S(<u>A</u>,B) R ⋃ S, R ⋂ S, R – S	SELECT * FROM R **UNION** | **INTERSECT** | **MINUS** SELECT * FROM S	▪ {**T** | r(T) **or** s(T)} ▪ {**T** | r(T) **and** s(T)} ▪ {**T** | r(T) **and not** s(T)}	RANGE TX IN (Rx,Sx) ▪ SELECT **Tx** ▪ SELECT Rx WHERE **∃ Sx (Sx.A=Rx.A)** ▪ SELECT Rx WHERE **¬ ∃ Sx (Sx.A=Rx.A)**
	Ej1.: *“Mostrar las ciudades de las piezas y de los proyectos”*; Ej2.: *“Mostrar las ciudades donde hay piezas y proyectos”*; Ej3.: *“Mostrar las ciudades donde hay piezas y no hay proyectos”*		
Π_{ciudad}(Pieza) ⋃ | ⋂ | - Π_{ciudad}(Proyecto)	SELECT ciudad FROM Pieza **UNION** | **INTERSECT** | **MINUS** SELECT ciudad FROM Proyecto	pCiu:={P.ciudad|pieza(P)} jCiu:={J.ciudad|proyecto(J)} 1) {**C** | pCiu(C) **or** jCiu(C)} 2) {**C** | pCiu(C) **and** jCiu(C)} 3) {**C** | pCiu(C) **and not** jCiu(C)}	RANGE **P** IN Pieza RANGE **J** IN Proyecto RANGE **C** IN (SELECT P.ciudad, SELECT J.Ciudad) ▪ SELECT **C.ciudad** ▪ SELECT **P.ciudad** WHERE **∃ J (P.ciudad=J.ciudad)** ▪ SELECT **P.ciudad** WHERE **¬ ∃ J (P.ciudad=J.ciudad)**

Álgebra	SQL	Cálc. WinRDBI	Cálc. QUEL
R(A,B); S(B,C) **Reunión Natural**: R ⋈ S	SELECT * FROM R **NATURAL JOIN** S	{**Rx,Sx.C** | r(Rx) and s(Sx) and **Rx.B=Sx.B**}	SELECT **Rx.*,Sx.C** WHERE **Rx.B=Sx.B**
Ej.: *"Mostrar las ventas con toda la información de los proyectos a los que suministran"*			
Ventas ⋈ Proyecto	SELECT * FROM ventas **NATURAL JOIN** proyecto	{**V,J.nompj,J.ciudad** | ventas(V) and proyecto(J) and **V.codpj=J.codpj**}	RANGE **V** IN Ventas RANGE **J** IN Proyecto SELECT **V.*,J.nompj, J.ciudad** WHERE **V.codpj=J.codpj**
División: R(A,B) ÷ S(B)	SELECT A FROM R WHERE NOT EXISTS ((SELECT S.B FROM S) MINUS (SELECT Rx.B FROM R Rx WHERE Rx.A=R.A))	{**Rx.A** | R(Rx) and **NOT EXISTS (Sx,Ry)** (S(Sx) and R(Ry) and **Sx.B=Ry.B and Rx.A=Ry.A**)}	SELECT **Rx.A** WHERE **¬ ∃ Sx(¬ ∃ Tx(Sx.B=Ry.B ^ Rx.A=Ry.A))**
Ej.: *"Mostrar los proyectos que suministran todas las piezas"*			
$\Pi_{codpj,codpie}$(Ventas) ÷ Π_{codpie}(Pieza)	SELECT j.codpj FROM proyecto j WHERE NOT EXISTS ((SELECT p.codpie FROM pieza p) MINUS (SELECT v.codpie FROM ventas v WHERE v.codpj=j.codpj))	{**J.codpj** | proyecto(J) and **NOT EXISTS (P,V)** (pieza(P) and ventas(V) and **V.codpie=P.codpie and V.codpj=J.codpj**)}	RANGE P IN Pieza RANGE J IN Proyecto RANGE V IN Ventas SELECT **J.codpj** WHERE **¬ ∃ P(¬ ∃ Vx(V.codpie=P.codpie ^ V.codpj=J.codpj))**

<table>
<tr><th>Álgebra</th><th>SQL</th><th>Cálc. WinRDBI</th><th>Cálc. QUEL</th></tr>
<tr><td colspan="4">Encontrar el más pequeño/más grande, el menor/mayor, primero/último, etc. Consultas sobre atributos con dominio subyacente ordenado (“string”, numérico, fecha, tiempo, etc.). En Álgebra hay que encontrar primero los que no cumplen la condición y restarlo a todos, con lo que obtenemos los que la cumplen.
Ej.: “Mostrar la venta más reciente”</td></tr>
<tr><td>1) Mediante un producto cartesiano de la relación ventas consigo mismo encontramos las ventas que <u>no son las más recientes</u>.
2) A todas las ventas les resto las que no son las más recientes.
ρ(Ventas)=V1,V2
Ventas –
$\Pi_{V1.*}(\sigma_{V1.fecha < V2.fecha}(V1 \times V2))$</td><td>Hay varias alternativas:
▪ (Basada en Álgebra)
(SELECT * FROM ventas) MINUS
(SELECT V1.* FROM ventas V1, ventas V2 WHERE V1.fecha<V2.fecha)

▪ SELECT * FROM ventas V1 WHERE V1.fecha =(SELECT max(fecha) FROM Ventas)

▪ (Basada en Cálculo) SELECT * FROM ventas V1 WHERE NOT EXISTS (SELECT * FROM Ventas V2 WHERE V2.fecha>V1.fecha)</td><td>▪ {V1 | ventas(V1) and NOT EXISTS (V2) (ventas(V2) and V2.fecha>V1.fecha}

▪ {V1 | ventas(V1) and FORALL(V2) (ventas(V2) and V2.fecha<=V1.fecha}</td><td>RANGE V1,V2 IN Ventas
▪ SELECT V1.* WHERE
¬ ∃ V2 (V2.fecha>V1.fecha)

▪ SELECT V1.* WHERE
∀ V2 (V2.fecha<=V1.fecha)</td></tr>
<tr><td colspan="4">Encontrar aquellas tuplas que tienen relación con una única tupla de otra tabla.
En Álgebra: Restamos a aquellas tuplas que tienen relación con elementos de esa otra tabla, las tuplas que tienen más de una relación con los elementos de esa otra tabla.
Ej.: “Encontrar los proyectos que tienen un único proveedor”</td></tr>
<tr><td>1) Encontramos los proyectos que <u>tienen más de un proveedor:</u> Mediante un producto cartesiano de la relación ventas consigo mismo igualo proyectos y selecciono los que tienen distinto proveedor.
2) A todos los proyectos de ventas les resto lo anterior.
ρ(Ventas)=V1,V2
$\Pi_{V1.codpj}Ventas - \Pi_{V1.codpj}(\sigma_{V1.codpj=V2.codpj \wedge V1.codpro <> V2.codpro}(V1 \times V2))$</td><td>Hay varias alternativas:
▪ (Basada en Álgebra)
(SELECT V3.codpj FROM ventas V3) MINUS
(SELECT V1.codpj FROM ventas V1, ventas V2 WHERE V1.codpj=V2.codpj AND V1.codpro<>V2.codpro)

▪ (Basada en Cálculo) SELECT V1.codpj FROM ventas V1 WHERE NOT EXISTS (SELECT * FROM Ventas V2 WHERE V1.codpj=V2.codpj AND V1.codpro<>V2.codpro)</td><td>▪ {V1.codpj | ventas(V1) and NOT EXISTS (V2) (ventas(V2) and V1.codpj=V2.codpj AND V2.codpro<>V1.codpro}

▪ {V1.codpj | ventas(V1) and FORALL(V2) (ventas(V2) and (V1.codpj<>V2.codpj OR V2.codpro=V1.codpro))}</td><td>RANGE V1,V2 IN Ventas
▪ SELECT V1.codpj WHERE
¬ ∃ V2 (V1.codpj=V2.codpj ˄ V2.codpro<>V1.codpro)

▪ SELECT V1.codpj WHERE
∀ V2 (V1.codpj=V2.codpj → V2.codpro=V1.codpro)</td></tr>
</table>

<table>
<tr><th>Álgebra</th><th>SQL</th><th>Cálc. WinRDBI</th><th>Cálc. QUEL</th></tr>
<tr><td colspan="4">Encontrar aquellos (1) campos que tienen relación con (2) todas las tuplas de otra relación/consulta.
En Álgebra (División): Encontramos e identificamos las tuplas de (2) (se trata del divisor); Buscamos la tabla (3) que relaciona esos campos (1) con los que identifican a (2) elaboramos el dividendo proyectando en la tabla (3) sobre los campos de (2) y sobre los campos del divisor.
Ej.: “Encontrar los proyectos suministrados por todos los proveedores de Paris”</td></tr>
<tr>
<td>ρ(proveedor)=S; ρ(Ventas)=V
1) Encontramos e identificamos (clave primaria) los proveedores de Paris (divisor):
$\Pi_{codpro}(_{ciudad='Paris'}(S))$
2) Buscamos (3), en este caso ventas; proyectamos sobre campos del divisor y sobre el campo a buscar (codpj), ya tenemos en dividendo:
$\Pi_{codpj,codpro}(V)$
3) $\Pi_{codpj,codpro}(V) \div$
$\Pi_{codpro}(_{ciudad='Paris'}(S))$</td>
<td>Hay varias alternativas:
▪ (Basada en not exists y diferencia)
SELECT J.codpj FROM proyecto WHERE
NOT EXISTS (
(SELECT V1.codpro FROM proveedor S
WHERE S.ciudad=‘Paris’) -- divisor
MINUS
(SELECT V.codpro FROM ventas V
WHERE V.codpj=S.codpj))

▪ (Basada en Cálculo)
SELECT J.codpj FROM proyecto J WHERE
NOT EXISTS (
SELECT * FROM proveedor S WHERE
S.ciudad=‘Paris’ AND
NOT EXISTS (SELECT * FROM ventas V WHERE
V.codpro=S.codpro AND V.codpj=J.codpj))</td>
<td>▪ {J.codpj | proyecto(J) and
NOT EXISTS (S,V) (proveedor(S) and
ventas(V) and S.ciudad=‘Paris’ AND
V.codpro=S.codpro AND
V.codpj=J.codpj)}

▪ {J.codpj | proyecto(J) and
FORALL(S) (proveedor(S) and
S.ciudad<>‘Paris’ OR EXISTS (V) (
V.codpro=S.codpro AND
V.codpj=J.codpj))}</td>
<td>RANGE J IN Proyecto
RANGE S IN Proveedor
RANGE V IN Ventas;
▪ SELECT J.codpj WHERE
¬ ∃ S (¬ ∃ V(S.ciudad=‘Paris’ ^
V.codpro=S.codpro ^
V.codpj=J.codpj))

▪ SELECT V1.codpj WHERE
∀ S (S.ciudad=‘Paris’ →
∃ V (V.codpro=S.codpro ^
V.codpj=J.codpj))</td>
</tr>
</table>