



decsai.ugr.es

Universidad de Granada

Fundamentos de Bases de Datos

Grado en Ingeniería Informática

Seminario 5: Cálculo relacional



**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 **mechanismo 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 $p(a_1, a_2, \dots, a_n)$ donde:
 - p es un símbolo de **predicado n-ario**
 - a_1, a_2, \dots, 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) $\wedge \neg$ conduce(propietario(GR-150-A), GR-150-A)
 - $\forall x$ coche(x) \rightarrow prefiere(propietario(x), marca(x), color(x))
 - $\exists y$ (persona(y) $\wedge \neg$ 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 $L=(S,W)$ 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 $L=(S,W)$ un lenguaje de CP; $C \subset S$ 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 S a objetos reales.
 - E se denomina “función extensión” y asocia a todo predicado n -ario $p \in S$ 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 con $K(c)$.



Interpretación de un lenguaje

Valor de verdad: toda interpretación $I=(D,K,E)$ de un lenguaje $L=(S,W)$ 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 $c_i \in S$ o $c_i = f(d_i)$ con $d_i \in S$ es cierta si $(c_1, c_2, \dots, c_n) \in E(P)$, en caso contrario es falsa.
 - Sean $f_1, f_2 \in W$, el valor de verdad de:
$$f_1 \vee f_2, f_1 \wedge f_2, \neg f_1, y f_1 \rightarrow f_2$$
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 $\forall c \in S$
 - $\exists x f_1(x)$ es cierta si $\exists c \in S$ 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 $L=(S,W)$ y $M \subset W$, I es un modelo de M si toda $f \in M$ es cierta con respecto a I .

Ejemplos:

- $\text{casados}(\text{Juan}, \text{Ana})$ será cierta si $(\text{Juan}, \text{Ana}) \in E(\text{casados})$
- $\text{prefiere}(\text{Ana}, \text{Honda}, \text{rojo})$ es cierta si $(\text{Ana}, \text{Honda}, \text{rojo}) \in E(\text{prefiere})$
- $\text{conduce}(\text{Juan}, \text{GR-150-A}) \wedge \neg \text{conduce}(\text{propietario}(\text{GR-150-A}), \text{GR-150-A})$ sera cierta si $\text{conduce}(\text{Juan}, \text{GR-150-A})$ es cierta y $\text{conduce}(\text{propietario}(\text{GR-150-A}), \text{GR-150-A})$ es falsa
- $\exists y (\text{persona}(y) \wedge \neg \text{casados}(\text{padre}(y), \text{madre}(y)))$ será cierta si podemos encontrar una constante c tal que
 - $(\text{persona}(c) \wedge \neg \text{casados}(\text{padre}(c), \text{madre}(c)))$ sea cierta
- $x \mid \text{casados}(\text{padre}(x), \text{madre}(x))$ define un conjunto de constantes

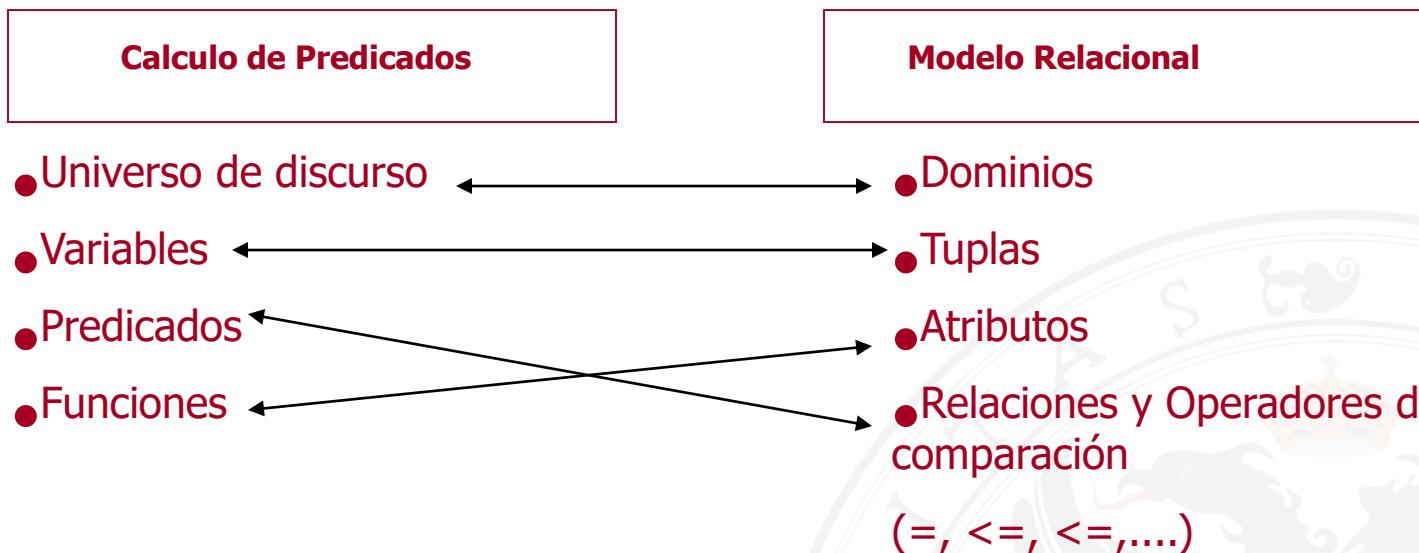


Analogías intuitivas entre el Cálculo de Predicados y el Modelo Relacional:

- 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



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 $x.f$

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





Definición de una consulta:

- Consideremos una base de datos con relaciones $R(A_1, \dots, A_n)$, $S(B_1, \dots, B_m)$, etc., y le **asociamos un lenguaje de Cálculo de Predicados**.
- Supongamos que $R_x, R_y \dots S_x, S_y \dots$, 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:

Select $R_x.A_i, R_x.A_j \dots, R_y.A_h, S_z.B_1, \dots$
Where $wff(R_x, R_y, S_z \dots)$

- $R_x.A_i, R_x.A_j \dots, R_y.A_h, S_z.B_1, \dots$ se denomina “**lista objetivo**”.
- $wff(R_x, R_y, S_z \dots)$ 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:

$$\{ \mathbf{X}.\mathbf{A}_i, \mathbf{X}.\mathbf{A}_j \dots, \mathbf{Y}.\mathbf{A}_h, \mathbf{Y}.\mathbf{B}_1, \dots \\ | \text{ wff } (\mathbf{R}(\mathbf{X}), \mathbf{R}(\mathbf{Y}), \mathbf{S}(\mathbf{Z}), \mathbf{X}, \mathbf{Y}, \mathbf{Z}, \dots) \}$$

- $\mathbf{X}.\mathbf{A}_i, \mathbf{X}.\mathbf{A}_j \dots, \mathbf{Y}.\mathbf{A}_h, \mathbf{Y}.\mathbf{B}_1, \dots$ se denomina “lista objetivo” .
- $\text{wff } (\mathbf{R}(\mathbf{X}), \mathbf{R}(\mathbf{Y}), \mathbf{S}(\mathbf{Z}), \mathbf{X}, \mathbf{Y}, \mathbf{Z}, \dots)$ es una fórmula en la que $\mathbf{R}(\mathbf{X}), \mathbf{R}(\mathbf{Y}), \mathbf{S}(\mathbf{Z})$ declara las variables libres \mathbf{X}, \mathbf{Y} para la relación \mathbf{R} y la variable libre \mathbf{Z} para la relación \mathbf{S} .
- La particularización de la lista objetivo para las **tuplas** que hacen cierta esta fórmula nos da la solución a la consulta.

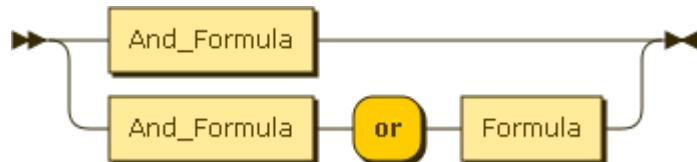


Sintaxis para WinRDBI

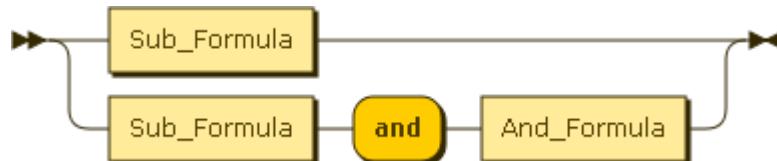
Query:



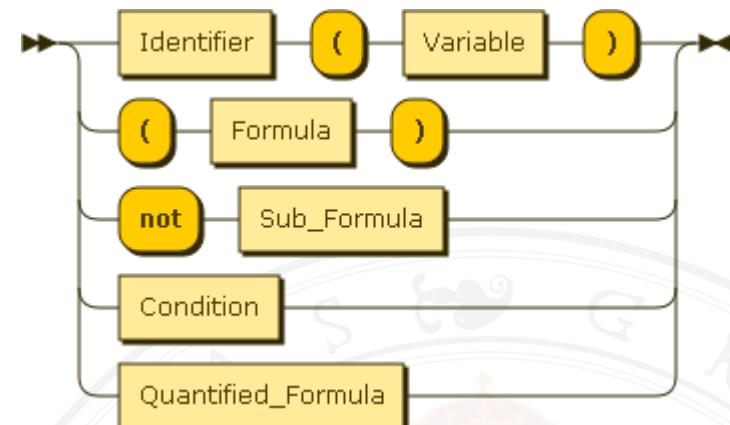
Formula:



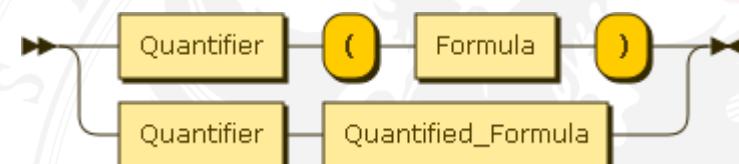
And_Formula:



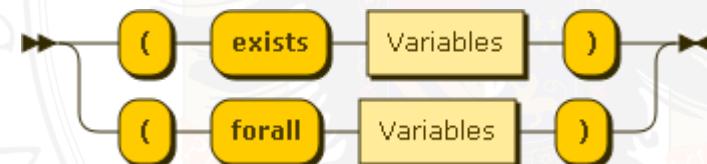
Sub_Formula:



Quantified_Formula:



Quantifier:



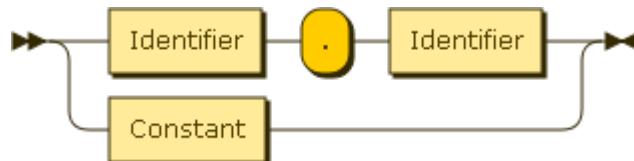


Sintaxis para WinRDBI

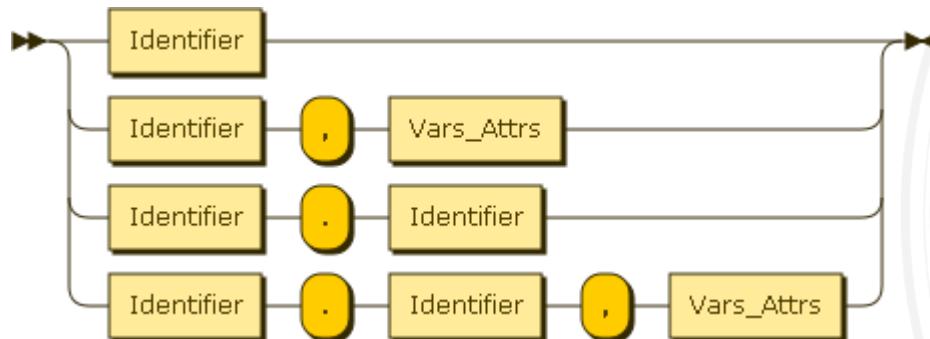
Condition:



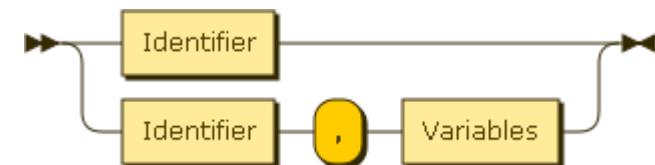
Operand:



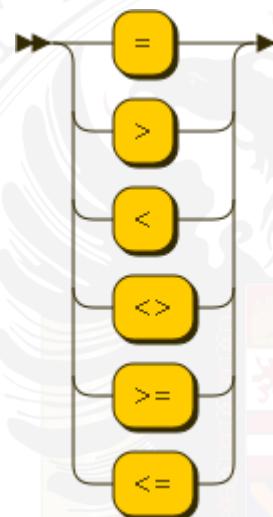
Vars_Attrs:



Variables:



Relational_Op:



Seminario 5: Cálculo relacional

El Cálculo Relacional orientado a Tuplas



ALUMNOS

<u>DNI</u>	NOMB_ALUM	FECHA_NAC	PROVINCIA	BECA
------------	-----------	-----------	-----------	------

ASIGNATURAS

<u>COD_ASIG</u>	NOMB_ASIG	CREDITOS	CARACTER	CURSO
-----------------	-----------	----------	----------	-------

PROFESORES

<u>NRP</u>	NOM_PROF	CATEGORIA	AREA	COD_DEP
------------	----------	-----------	------	---------

DEPARTAMENTOS

<u>COD_DEP</u>	NOM_DEP	DIRECTOR
----------------	---------	----------

AULAS

<u>COD-AULA</u>	CAPACIDAD
-----------------	-----------

GRUPOS

<u>COD_GRUP</u>	COD_ASIG	TIPO	NRP	MAX_AL
-----------------	----------	------	-----	--------

CLASE

<u>COD_AULA</u>	DIA	HORA	COD_GRUP	COD_ASIG	TIPO
-----------------	-----	------	----------	----------	------

MATRICULAS

<u>COD_GRUP</u>	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 de tupla:
 - 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 $\exists G_y (G_y.cod_asig=A_x.cod_asig \wedge G_y.tipo='Teoria' \wedge G_y.max_al>=60)$
 - 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.categoría='AS'
```

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

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

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

QUEL: RANGE Px IN profesores

```
SELECT Px.nom_prof WHERE Px.categoría='AS'
```

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

Álgebra: $\Pi_{\text{nom_prof}}(\sigma_{categoría='AS'}(\text{profesores}))$



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

"Mostrar NRP s de profesor y nombres de departamento que cumplen que coinciden en el código de 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: $\Pi_{NRP, nom_dep}(\text{profesores} \bowtie \text{departamentos})$



"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 (\exists Ax,Ay ((Ax.cod_asig='FBD') \wedge (Ay.cod_asig='TA') \wedge
(Ax.NRP=Px.NRP) \wedge (Ay.NRP=Px.NRP)))

Álgebra: $\Pi_{\text{nom_prof}} (\text{profesores} \bowtie ((\Pi_{\text{NRP}} (\sigma_{\text{cod_asig}=\text{'FBD'}} (\text{grupos}))) \cap (\Pi_{\text{NRP}} (\sigma_{\text{cod_asig}=\text{'TA'}} (\text{grupos}))))))$



"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 (\exists Ax ((Ax.cod_asig='FBD') \vee (Ax.cod_asig='TA')) \wedge
(Ax.NRP=Px.NRP)))

Álgebra: $\Pi_{\text{nom_prof}} (\text{profesores} \bowtie (\sigma_{\text{cod_asig}=\text{'FBD'}} \vee \text{cod_asig}=\text{'TA'})(\text{grupos}))$



"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 $\neg(\exists Ax ((Ax.tipo='Practica') \wedge (Ax.NRP=Px.NRP)))$

Álgebra: $\Pi_{nom_prof} (\text{profesores} \bowtie (\Pi_{NRP}(\text{profesores}) - \Pi_{NRP}(\sigma_{\text{tipo}=practica}(\text{grupos}))))$



"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) \wedge (Px.NRP < Py.NRP))

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

$\Pi_{\text{profesores.nom_prof}, \text{profes.nom_prof}} (\sigma_{\text{profesores.cod_dep}=\text{profes.cod_dep} \wedge \text{profesores.NRP} < \text{profes.NRP}} (\text{profesores} \times \text{profes}))$



"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 $\forall P (P.area='COMPUT')$

$\rightarrow \exists G (G.cod_asig=A.cod_asig \wedge G.NRP=P.NRP)$

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 un Grupo de esa asignatura impartido por 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 $\neg \exists P (P.area='COMPUT' \wedge$
 $\neg \exists G (G.cod_asig=A.cod_asig \wedge G.NRP=P.NRP))$

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: $\sigma_{\text{tipo_de_oficio}=\text{'Electricista'}} \text{TRABAJADORES}$

"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: $\Pi_{\text{nombre}} (\sigma_{\text{tipo_de_oficio}=\text{'Electricista'}} \text{TRABAJADORES})$



"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: $\Pi_{\text{nombre}, \text{horas_por_sem}} (\text{TRABAJADORES} \bowtie \text{OFICIOS})$



"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 ($\exists Ax, Ay ((Ax.id_trabajador=Tx.id_trabajador) \wedge (Ay.id_trabajador=Tx.id_trabajador)) \wedge (Ax.id_edificio=312) \wedge (Ay.id_edificio=460)$)

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: $\Pi_{\text{nombre}} (\text{TRABAJADORES} \bowtie ((\Pi_{\text{id_trabajador}} \sigma_{\text{id_edificio}=312} \text{ASIGNACIONES}) \cap (\Pi_{\text{id_trabajador}} \sigma_{\text{id_edificio}=469} \text{ASIGNACIONES})))$



"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 $\exists Ax ((Ax.id_trabajador=Tx.id_trabajador) \wedge ((Ax.id_edificio=312) \vee (Ax.id_edificio=460)))$

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: $\Pi_{\text{nombre}} (\text{TRABAJADORES} \bowtie$

$\sigma_{\text{id_edificio}=312 \vee \text{id_edificio}=460} (\text{ASIGNACIONES})$)



"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 $\neg(\exists Ax ((Ax.id_trabajador=Tx.id_trabajador) \wedge (Ax.id_edificio=312)))$

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

Álgebra: $\Pi_{\text{nombre}} (\text{TRABAJADORES} \bowtie (\Pi_{\text{id_trabajador}} \text{TRABAJADORES} - \Pi_{\text{id_trabajador}} (\sigma_{\text{id_edificio}=312} (\text{ASIGNACIONES}))))$



"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: $\Pi_{Tx.\text{nombre}, Ty.\text{nombre}}$

$(\sigma_{Tx.\text{tipo_de_oficio}=Ty.\text{tipo_de_oficio} \wedge Tx.\text{nombre}<Ty.\text{nombre}}(\rho_{Tx}(\text{Trabajadores}) \times \rho_{Ty}(\text{Trabajadores})))$



"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 $\neg \exists T_x (\neg \exists A_x ((A_x.id_trabajador=T_x.id_trabajador) \wedge (A_x.id_edificio=E_x.id_edificio)))$

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: $(\Pi_{id_edificio, id_trabajador} ASIGNACIONES) \div (\Pi_{id_trabajador} TRABAJADORES)$

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



Seminario 5: Cálculo relacional

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"			
$\Pi_{codpro,nompro}(\text{Proveedor})$	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 A\theta K;	{R r(R) and R.A\theta K };	SELECT Rx.* WHERE Rx.A\theta K
Ej.: "Mostrar los proveedores de Paris"			
$\sigma_{ciudad='Paris'}(\text{Proveedor})$	SELECT * FROM Proveedor WHERE ciudad='Paris';	{S proveedor(S) and S.ciudad='Paris' };	SELECT Sx.* WHERE Sx.ciudad='Paris'
Producto Cart.: $R \times S$	SELECT * FROM R,S;	{ R,S r(R) and r(S)};	SELECT Rx.*, Sx.*
Ej.: "Mostrar todas las combinaciones proveedor-pieza"			
Proveedor \times Pieza	SELECT * FROM Proveedor,Pieza;	{ S,P Proveedor(S) and Pieza(P)};	RANGE Sx IN Proveedor RANGE Px IN Pieza SELECT Sx.*,Px.*



Álgebra	SQL	Cálc. WinRDBI	Cálc. QUEL
Alias Def.: $p(R)=Rx$	SELECT * FROM R Rx,R Ry	$\{Rx, Ry \mid r(Rx) \text{ and } r(Ry)\}$	RANGE Rx,Ry IN R
Ej.: "Mostrar todas las combinaciones de cada pieza con cada pieza"			
$p(\text{Pieza})=P$ Pieza \times P	SELECT * FROM Pieza P1, Pieza P2	$\{P1,P2 \mid \text{pieza}(P1) \text{ and } \text{pieza}(P2)\}$	RANGE P1,P2 IN Pieza SELECT P1.* ,P2.*
Unión, Intersección y Diferencia: R y S compatibles respecto a la Unión. $R(A,B);S(A,B)$ $R \cup S, R \cap S, R - S$	SELECT * FROM R UNION INTERSECT MINUS SELECT * FROM S	<ul style="list-style-type: none"> ▪ $\{T \mid r(T) \text{ or } s(T)\}$ ▪ $\{T \mid r(T) \text{ and } s(T)\}$ ▪ $\{T \mid r(T) \text{ and not } s(T)\}$ 	RANGE TX IN (Rx,Sx) <ul style="list-style-type: none"> ▪ SELECT Tx ▪ SELECT Rx WHERE $\exists Sx (Sx.A=Rx.A)$ ▪ SELECT Rx WHERE $\neg \exists Sx (Sx.A=Rx.A)$
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"			
$\Pi_{\text{ciudad}}(\text{Pieza})$ $\cup \mid \cap \mid -$ $\Pi_{\text{ciudad}}(\text{Proyecto})$	SELECT ciudad FROM Pieza UNION INTERSECT MINUS SELECT ciudad FROM Proyecto	pCiudad := {P.ciudad pieza(P)} jCiudad := {J.ciudad proyecto(J)} 1) $\{C \mid pCiudad(C) \text{ or } jCiudad(C)\}$ 2) $\{C \mid pCiudad(C) \text{ and } jCiudad(C)\}$ ó $\{P.ciudad \mid \text{pieza}(P) \text{ and } (\exists J) (\text{proyecto}(J) \text{ and } P.ciudad=J.ciudad)\}$ 3) $\{C \mid pCiudad(C) \text{ and not } jCiudad(C)\}$ ó $\{P.ciudad \mid \text{pieza}(P) \text{ and not } (\exists J) (\text{proyecto}(J) \text{ and } P.ciudad=J.ciudad)\}$	RANGE P IN Pieza RANGE J IN Proyecto RANGE C IN (SELECT P.ciudad, SELECT J.ciudad) <ul style="list-style-type: none"> ▪ SELECT C.ciudad ▪ SELECT P.ciudad WHERE $\exists J (P.ciudad=J.ciudad)$ ▪ SELECT P.ciudad WHERE $\neg \exists J (P.ciudad=J.ciudad)$

Seminario 5: Cálculo relacional

Correspondencia entre operadores



Álgebra	SQL	Cálc. WinRDBI	Cálc. QUEL
$R(A,B); S(B,C)$ Reunión Natural: $R \bowtie S$	SELECT * FROM R NATURAL JOIN S;	$\{Rx,Sx.C \mid r(Rx) \text{ and } s(Sx) \text{ and } Rx.B=Sx.B\}$	SELECT Rx.* , Sx.C WHERE Rx.B=Sx.B
<i>Ej.: "Mostrar las ventas con toda la información de los proyectos a los que suministran"</i>			
Ventas \bowtie Proyecto	SELECT * FROM ventas NATURAL JOIN proyecto;	$\{V,J.nompj,J.ciudad \mid \text{ventas}(V) \text{ and } \text{proyecto}(J) \text{ and } V.codpj=J.codpj\}$	RANGE V IN Ventas RANGE J IN Proyecto SELECT V.* , J.nompj, J.ciudad WHERE V.codpj=J.codpj
División: $R(A,B) \div S(B)$	SELECT A FROM R WHERE NOT EXISTS ((SELECT S.B FROM S) MINUS (SELECT Rx.B FROM R Rx WHERE Rx.A=R.A));	$\{Rx.A \mid R(Rx) \text{ and } \text{not}(\exists Sx.Ry) (S(Sx) \text{ and } R(Ry) \text{ and } Sx.B=Ry.B \text{ and } Rx.A=Ry.A)\}$	SELECT Rx.A WHERE $\neg \exists Sx (\neg \exists Tx (Sx.B=Ry.B \wedge Rx.A=Ry.A))$
<i>Ej.: "Mostrar los proyectos que suministran todas las piezas"</i>			
$\Pi_{codpj,codpie}(\text{Ventas}) \div \Pi_{codpie}(\text{Pieza})$	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));	$\{J.codpj \mid \text{proyecto}(J) \text{ and } \text{not}(\exists P,V) (\text{pieza}(P) \text{ and } \text{ventas}(V) \text{ and } V.codpie=P.codpie \text{ and } V.codpj=J.codpj)\}$	RANGE P IN Pieza RANGE J IN Proyecto RANGE V IN Ventas SELECT J.codpj WHERE $\neg \exists P (\neg \exists V (V.codpie=P.codpie \wedge V.codpj=J.codpj))$



Álgebra

SQL

Cálc. WinRDBI

Cálc. QUEL

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"

1) Mediante un producto cartesiano de la relación ventas consigo mismo encontramos las ventas que no son las más recientes.

2) A todas las ventas les resto las que no son las más recientes.

$$\rho(\text{Ventas}) = V1, V2$$

Ventas –

$$\Pi_{V1.*} (\sigma_{V1.\text{fecha} < V2.\text{fecha}} (V1 \times V2))$$

Hay varias alternativas:

- (Basada en Álgebra)

$$(\text{SELECT } * \text{ FROM ventas}) \text{ MINUS}$$

$$(\text{SELECT } V1.* \text{ FROM ventas } V1, \text{ ventas } V2 \text{ WHERE } V1.\text{fecha} < V2.\text{fecha});$$
- $\text{SELECT } * \text{ FROM ventas } V1 \text{ WHERE } V1.\text{fecha} = (\text{SELECT } \text{max}(\text{fecha}) \text{ FROM Ventas});$
- (Basada en Cálculo)

$$\text{SELECT } * \text{ FROM ventas } V1 \text{ WHERE NOT EXISTS } (\text{SELECT } * \text{ FROM Ventas } V2 \text{ WHERE } V2.\text{fecha} > V1.\text{fecha});$$

- $\{V1 \mid \text{ventas}(V1) \text{ and } \text{not exists } V2 \text{ (ventas}(V2) \text{ and } V2.\text{fecha} > V1.\text{fecha})\};$
- $\{V1 \mid \text{ventas}(V1) \text{ and } (\text{forall } V2) (\text{not ventas}(V2) \text{ or } V2.\text{fecha} \leq V1.\text{fecha})\};$

RANGE **V1,V2** IN Ventas

- $\text{SELECT } V1.* \text{ WHERE }$
- $\neg \exists V2 (V2.\text{fecha} > V1.\text{fecha})$

- $\text{SELECT } V1.* \text{ WHERE }$
- $\forall V2 (V2.\text{fecha} \leq V1.\text{fecha})$

En Álgebra: Restamos a 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"

1) Encontramos los proyectos que tienen más de un proveedor: 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.

$$\rho(\text{Ventas}) = V1, V2$$

$$\Pi_{V1.\text{codpj}} \text{Ventas} - \Pi_{V1.\text{codpj}} (\sigma_{V1.\text{codpj} = V2.\text{codpj}} \wedge V1.\text{codpro} <> V2.\text{codpro}) (V1 \times V2))$$

Hay varias alternativas:

- (Basada en Álgebra)

$$(\text{SELECT } V3.\text{codpj} \text{ FROM ventas } V3) \text{ MINUS}$$

$$(\text{SELECT } V1.\text{codpj} \text{ FROM ventas } V1, \text{ ventas } V2 \text{ WHERE } V1.\text{codpj} = V2.\text{codpj} \text{ AND } V1.\text{codpro} <> V2.\text{codpro});$$
- (Basada en Cálculo)

$$\text{SELECT } V1.\text{codpj} \text{ FROM ventas } V1 \text{ WHERE NOT EXISTS } (\text{SELECT } * \text{ FROM Ventas } V2 \text{ WHERE } V1.\text{codpj} = V2.\text{codpj} \text{ AND } V1.\text{codpro} <> V2.\text{codpro});$$

- $\{V1.\text{codpj} \mid \text{ventas}(V1) \text{ and } \text{not exists } V2 \text{ (ventas}(V2) \text{ and } V1.\text{codpj} = V2.\text{codpj} \text{ and } V2.\text{codpro} <> V1.\text{codpro})\};$

- $\{V1.\text{codpj} \mid \text{ventas}(V1) \text{ and } (\text{forall } V2) (\text{not ventas}(V2) \text{ or } V1.\text{codpj} <> V2.\text{codpj} \text{ or } V2.\text{codpro} = V1.\text{codpro})\};$

RANGE **V1,V2** IN Ventas

- $\text{SELECT } V1.\text{codpj} \text{ WHERE }$
- $\neg \exists V2 (V1.\text{codpj} = V2.\text{codpj} \wedge V2.\text{codpro} <> V1.\text{codpro})$

- $\text{SELECT } V1.\text{codpj} \text{ WHERE }$
- $\forall V2 (V1.\text{codpj} = V2.\text{codpj} \rightarrow V2.\text{codpro} = V1.\text{codpro})$



Álgebra

SQL

Cálc. WinRDBI

Cálc. QUEL

Encontrar aquellas tuplas en una tabla que **no cumplen una determinada condición**.

En Álgebra: Identificamos las tuplas **que la cumplen y las restamos** a todas las tuplas.

Ej.: *"Encontrar los proyectos que no usan ninguna pieza color 'Blanco' suministrada por un proveedor de 'Madrid'"*

1) Encontramos e identificamos los proyectos que usan una pieza 'Blanca' enviada por un proveedor de 'Madrid':

$$\Pi_{V.codpj}(\sigma_{V.codpro=S.codpro \wedge V.codpie = P.codpie \wedge S.ciudad='Madrid' \wedge P.color='Blanco'}(S \times P \times V))$$

2) A todos los proyectos de ventas les resto lo anterior.

$$\Pi_{codpj}(\text{Proyecto}) -$$

$$\Pi_{V.codpj}(\sigma_{V.codpro=S.codpro \wedge V.codpie = P.codpie \wedge S.ciudad='Madrid' \wedge P.color='Blanco'}(S \times P \times V))$$

Hay varias alternativas:

- (Basada en Álgebra)

(SELECT codpj FROM proyecto)

MINUS

(SELECT V.codpj FROM proveedor S, pieza P, ventas V WHERE S.ciudad='Madrid' and P.color='Blanco' and V.codpro=S.codpro and V.codpie = P.codpie and V.codpj=J.codpj);

- (Basada en Cálculo)

SELECT J.codpj FROM proyecto WHERE NOT EXISTS (SELECT * FROM proveedor S, pieza P, ventas V WHERE S.ciudad='Madrid' and P.color='Blanco' and V.codpro=S.codpro and V.codpie = P.codpie and V.codpj=J.codpj);

- {J.codpj | proyecto(J) and **not (exists S,P,V) (proveedor(S) and pieza(P) and ventas(V) and S.ciudad='Madrid' and P.color='Blanco' and V.codpro=S.codpro and V.codpie = P.codpie and V.codpj=J.codpj)};**

- {J.codpj | proyecto(J) and **(forall S,P (not proveedor(S) or not pieza(P) or S.ciudad<>'Madrid' or P.color<>'Blanco' or not (exists V) (ventas(V) and V.codpro=S.codpro and V.codpie = P.codpie and V.codpj=J.codpj))};**

RANGE **V** IN Ventas; RANGE **S** IN Proveedor; RANGE **P** IN Pieza

- SELECT **J.codpj WHERE**
 $\neg \exists S,P,V (S.ciudad='Madrid' \wedge P.color='Blanco' \wedge V.codpro=S.codpro \wedge V.codpie = P.codpie \wedge V.codpj=J.codpj);$

- SELECT **J.codpj WHERE** $\forall S,P ((S.ciudad='Madrid' \wedge P.color='Blanco') \rightarrow \neg \exists V (V.codpro=S.codpro \wedge V.codpie = P.codpie \wedge V.codpj=J.codpj));$



Álgebra

SQL

Cálc. WinRDBI

Cálc. QUEL

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 (1) y sobre los campos del divisor.

Ej.: "Encontrar los proyectos suministrados por todos los proveedores de Paris"

$p(\text{proveedor})=S; p(\text{Ventas})=V$
 1) Encontramos e identificamos (clave primaria) los proveedores de Paris (divisor):
 $\Pi_{\text{codpro}}(\text{ciudad} = \text{'Paris'}) (S)$
 2) Buscamos (3), en este caso ventas; proyectamos sobre campos del divisor y sobre el campo a buscar (codpj), ya tenemos en dividendo:
 $\Pi_{\text{codpj}, \text{codpro}}(V)$
 3) $\Pi_{\text{codpj}, \text{codpro}}(V) \div$
 $\Pi_{\text{codpro}}(\text{ciudad} = \text{'Paris'}) (S)$

Hay varias alternativas:

- (Basada en not exists y diferencia)


```
SELECT J.codpj FROM proyecto J WHERE
NOT EXISTS (
    SELECT S.codpro FROM proveedor S WHERE
        S.ciudad='Paris') -- divisor
```

 MINUS

```
(SELECT V.codpro FROM ventas V
WHERE V.codpj=J.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));
```

- {**J.codpj** | **proyecto(J) and not (exists S) (proveedor(S) and S.ciudad='Paris' and not (exists V) (ventas(V) and V.codpro=S.codpro and V.codpj=J.codpj))});**

- {**J.codpj** | **proyecto(J) and (forall S) (not (proveedor(S) and S.ciudad='Paris') or (exists V) (ventas(V) and V.codpro=S.codpro and V.codpj=J.codpj))};**

RANGE J IN Proyecto
 RANGE S IN Proveedor
 RANGE V IN Ventas;

- **SELECT J.codpj WHERE**
 $\neg \exists S (\neg \exists V (S.ciudad='Paris' \wedge V.codpro=S.codpro \wedge V.codpj=J.codpj))$

- **SELECT V1.codpj WHERE**
 $\forall S (S.ciudad='Paris' \rightarrow \exists V (V.codpro=S.codpro \wedge V.codpj=J.codpj))$