Procesamiento de consultas

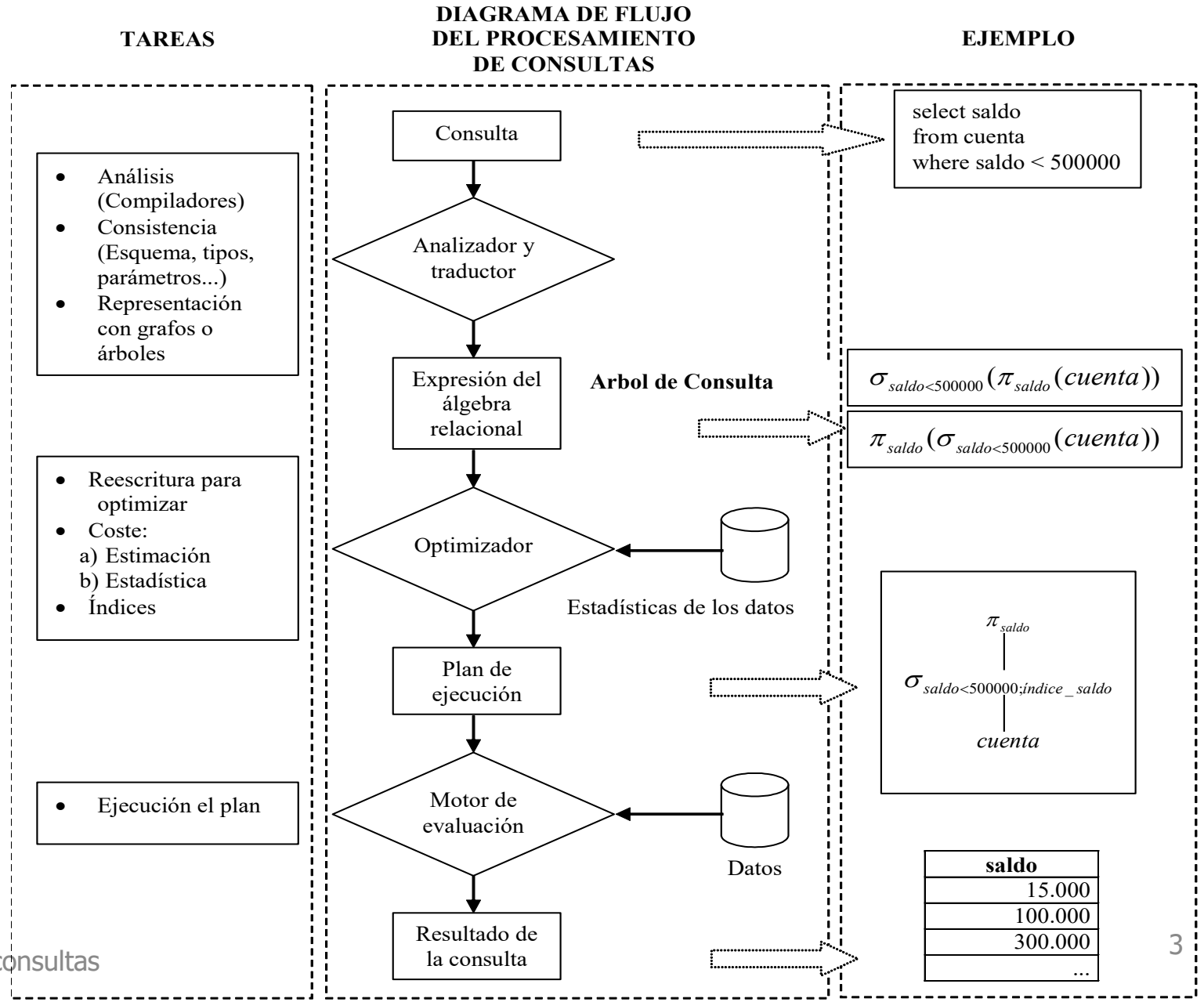
Índice

Introducción

Pasos del ANALISIS
1.-Análisis lexicográfico
(palabras SQL correctas)
2.- Análisis sintáctico
(cumple reglas gramática SQL)
3.- Validación semántica

Pasos del OPTIMIZADOR
1.-Genera planes:
- Qué expresión de A.R. en el Arbol de Consulta
- Qué algoritmo de acceso para cada operación de A.R.
- En qué orden ejecutar las ops

2.- Escoge el mejor plan



Introducción

- **Objetivo:** Mejorar los tiempos de respuesta en un SGBDR.
- **Factores a considerar**:
  - Evaluación de qué consulta es algebraicamente más adecuada
    - Para manejar el mínimo número de datos: filas y atributos
  - Evaluación de la carga sobre los recursos del sistema

Introducción

- Ejemplo:
  - Tablas de “proveedores” PROV(nombre,IDprov) con 100 proveedores y “productos” PROD(nombreProd,IDprov) con 10.000 productos.
  - Consulta: “Obtener los nombres de los proveedores que venden el producto P2”.
  - Estimaremos que sólo 50 tuplas de PROV corresponden al producto P2

```
SELECT DISTINCT PROV.NOMBRE
FROM PROV, PROD
WHERE PROV.IDprov=PROD.IDprov
AND PROD.nombreProd=”P2”;
```

- Veamos dos ejecuciones equivalentes y el número de filas leídas y escritas

Introducción

- Ejemplo (sigue): Ejecución **(a)**, ORDEN de los PASOS
  I. (**X** = producto cartesiano) PROV x PROD ⇒100 x10.000 = 1.000.000 de tuplas leídas (1). Probablemente, 1.000.000 tuplas escritas (2) en memoria virtual (en disco).
  II. (σ) Selección según la cláusula WHERE, 1.000.000 tuplas leídas (3), resultado reducido a 50 tuplas según la condición "P2" (4).
  III. (π) Realizar la proyección (5) sobre PROV.NOMBRE, dando como resultado un máximo de 50 tuplas.

```
SELECT DISTINCT PROV.NOMBRE
FROM PROV, PROD
WHERE PROV.IDprov=PROD.IDprov
AND PROD.nombreProd="P2";
```

Operaciones y Filas en (a)
(1) Lee 1.000.000
(2) Escribe 1.000.000
(3) Lee 1.000.000
(4) Selecciona (no accede)
(5) Proyección (no accede)
TOTAL 3.000.000

Contamos solo las Operaciones costosas: accesos a disco

Introducción

- Ejemplo (sigue): Ejecución **(b)** ORDEN de los PASOS
  - Equivalencia algebraica:
    i. Seleccionar en PROD las tuplas de la pieza P2. Lectura (1) de 10.000 tuplas, y selección (2) con resultado: 1 tupla.
    ii. JOIN de la tabla anterior con la tabla PROV. Lectura de 100 tuplas (3). Resultado join: 50 tuplas.
    iii. Proyección (4) sobre PROV.NOMBRE. Resultado máximo de 50 tuplas.

Operaciones y Filas en (a)
(1) Lee 1.000.000
(2) Escribe 1.000.000
(3) Lee 1.000.000
(4) Selecciona (no accede)
(5) Proyección (no accede)
TOTAL **3.000.000**

accesos a disco

Operaciones y Filas en (b)
(1) Lee 10.000
(2) Selección P2 (no accede)
(3) Lee 100
(4) Proyección (no accede)
TOTAL **10.100**

accesos a disco

Introducción

- Ejemplo (sigue): optimizar Rendimiento
  - El “rendimiento ” es el número de operaciones E/S de tuplas:
    - El procedimiento (b) es unas 300 veces mejor,
    - el primero realiza 3.000.000 operaciones de E/S
    - frente a 10.100 del segundo

- La optimización incluye manejar la **mínima** cantidad de datos.

Introducción

- **¿Procedimiento de optimización?:**
  - Consta de los siguientes pasos:
    - Representación interna de consultas (Análisis del SQL)
    - Conversión a forma canónica (el árbol)
    - Elección de procedimientos (operaciones de bajo nivel del Optimizador)
    - Generación y elección de planes de consulta (solo **un** plan de ejecución)

- **¿Donde aplica la optimización?:**
  - Factores a tener en cuenta:
    (todos relacionados con el volumen de datos)
    - El coste de acceso al almacenamiento secundario (disco).
    - El coste de almacenamiento.
    - El coste de computación.
    - El optimizador interviene también en las actualizaciones y borrados.

Introducción

- **¿En qué se basa la optimización?**
  - Información estadística (cardinalidades de tablas, dominios, etc)
  - Independencia de las estrategias de acceso con respecto de la organización física de la base de datos.
  - Potencia de cálculo en la toma de decisiones frente a la optimización manual.
  - Disponibilidad del optimizador para todos los usuarios del sistema.

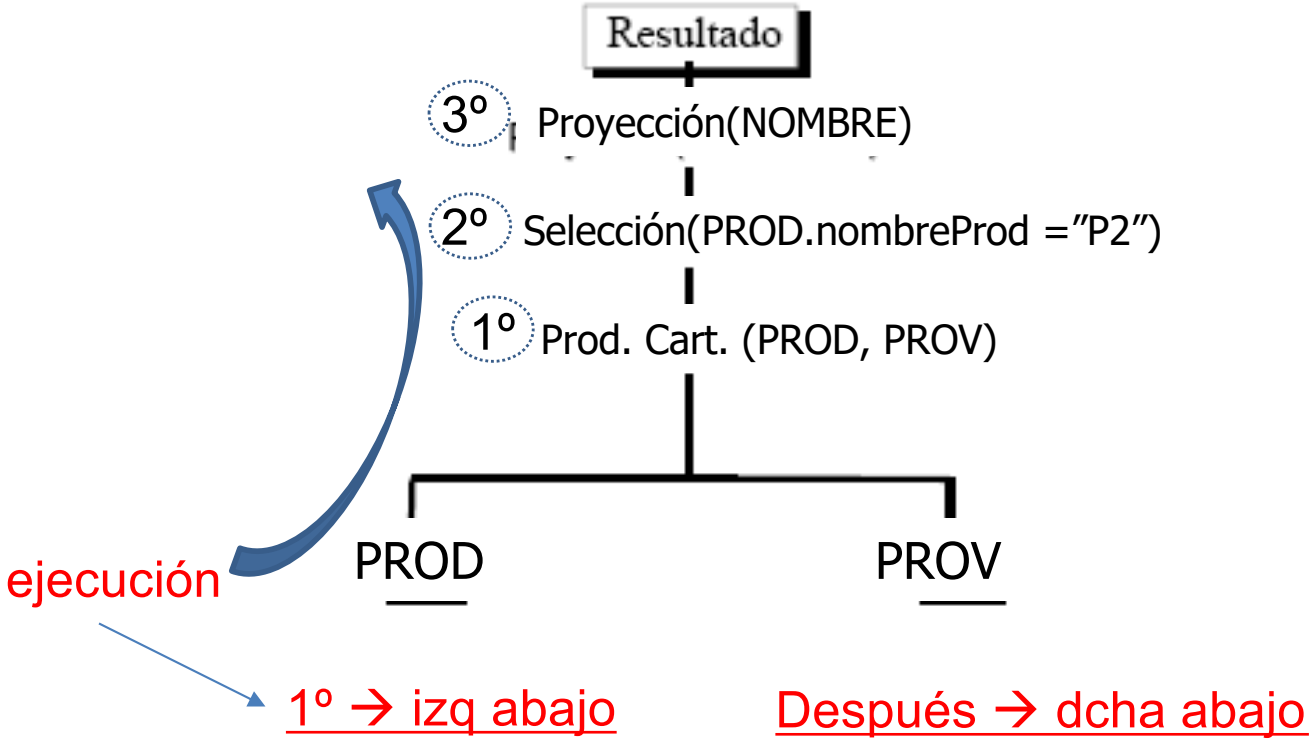
Reescritura de consultas

- **Representación de consultas**
  - Sistemas de representación:
    - álgebra relacional
    - cálculo relacional
    - árbol sintáctico abstracto o *Árbol de Consulta*
  - Características:
    - Ser “relacionalmente” completo: todas la operaciones
    - Permite realizar optimizaciones Reescribiendo la consulta:
      - Una consulta tiene varias formas equivalentes de ser ejecutada
      - Siguiendo unas Reglas de Equivalencia de Operaciones

Reescritura de consultas

Árbol de Consulta: → **Consulta sin optimizar**

$$\pi_{NOMBRE}(\sigma_{nombreProd=\text{"P2"}}(PROV \times PROD))$$



Conversión a forma canónica

- OBJETIVO:
  - Manejar el menor número de Datos: filas y atributos
- Aplicar optimizaciones que tienen un resultado mejorado y seguro.
- Buscar una expresión equivalente de una consulta (*FORMA CANONICA)*
  - para que se mejore de alguna manera el rendimiento

- La reescritura se basa en la equivalencia semántica:
  - igual esquema,
  - Obtener igual número de tuplas (quizás en diferente orden).

Algoritmo de optimización

- **(A) → Ejecutar selección primero** (sigue las Reglas en Apéndice)
  - *Paso 1:* Para selecciones con condiciones conjuntivas, tipo $\sigma_{c_1 \wedge c_2 \wedge \ldots c_n}(r)$
    - generar selecciones con condiciones simples ***ci*** en secuencia: $\sigma_{c_1}(\sigma_{c_2}(\sigma_{\ldots c_n}(r)))$
    - Si hay condiciones disyuntivas sobre tablas distintas:
      - Poner cada condición en un árbol distinto
  - *Paso 2:* Ejecutar primero las selecciones: Bajándolas en el árbol
    - Operaciones σ más restrictivas hacia los nodos hoja tanto como sea posible:
      - Según las tablas que participan en la selección σ
    - Asegurarse de que el orden de los nodos hoja no provocan operaciones "**X**"

Algoritmo de optimización

- **(B) → Ejecutar operaciones que reducen tamaño de resultado**
  - *Paso 3:* Poner juntas las selecciones y uniones (X prod. cart)
    - Las *más selectivas* (dan menos tuplas) ejecutarlas cuanto antes:
      - bajándolas y poniendo las relaciones en las hojas y a la izquierda
  - *Paso 4:* Cambiar (selección + prod. Cartesiano) por theta unión ⋈
  - *Paso 5:* Deshacer Proyecciones con lista atributos de varias relaciones
    - y obtener varias proyecciones de atributos de una sola relación
    - y bajarlas en el árbol hasta la relación a la que pertenece
    - y añadir π para evitar que suban atributos innecesarios
- **(C) → Escoger el mejor Algoritmo de acceso → lo hace el optimizador**
  - Define la operaciones o *Procedimientos de Bajo Nivel*
    - según costes y otros criterios
  - Genera los Planes de Consulta
  - Elige el mejor
  - Lo ejecuta

Ejemplo:

Dada esta Bases de Datos:

- Empleado: num. Seg. Social, fecha Nacim, Nombre

  E EMPLOYEE(SSN,BDATE,LNAME)

- Proyecto: Nombre Proyecto, Número Proyecto

  P PROJECT(PNAME,PNUMBER)

- Trabaja-en: num. Seg. Social, Número Proyecto

  W WORKS_ON (ESSN,PNO)

CONSULTA: Nombre de empleados que trabajan en el Proyecto Acuarius nacidos a partir de 1958

→ **Asumimos** que la condición de la fecha da más filas (es menos restrictiva) que la condición del nombre del proyecto (más restrictiva)

Ejemplo:

- SELECT LNAME

  FROM EMPLOYEE, WORKS_ON, PROJECT

  WHERE PNAME = 'Aquarius' AND PNUMBER = PNO AND ESSN = SSN

  AND BDATE > '1957-12-31'

→ Se empieza con su equivalente en algebra relacional **SIN OPTIMIZAR**:

$$\pi_{LNAME}(\sigma_{PNAME='Aquarius' \wedge PNUMBER=PNO \wedge ESSN=SSN \wedge BDATE>'1957-12-31'}(EMPLOYEE \times WORKS\_ON \times PROJECT)) =$$

$$\pi_{LNAME}(\sigma_{c_1 \wedge c_2 \wedge c_3 \wedge c_4}(E \times W \times P)),$$

$$c_1 = PNAME = 'Aquarius',$$

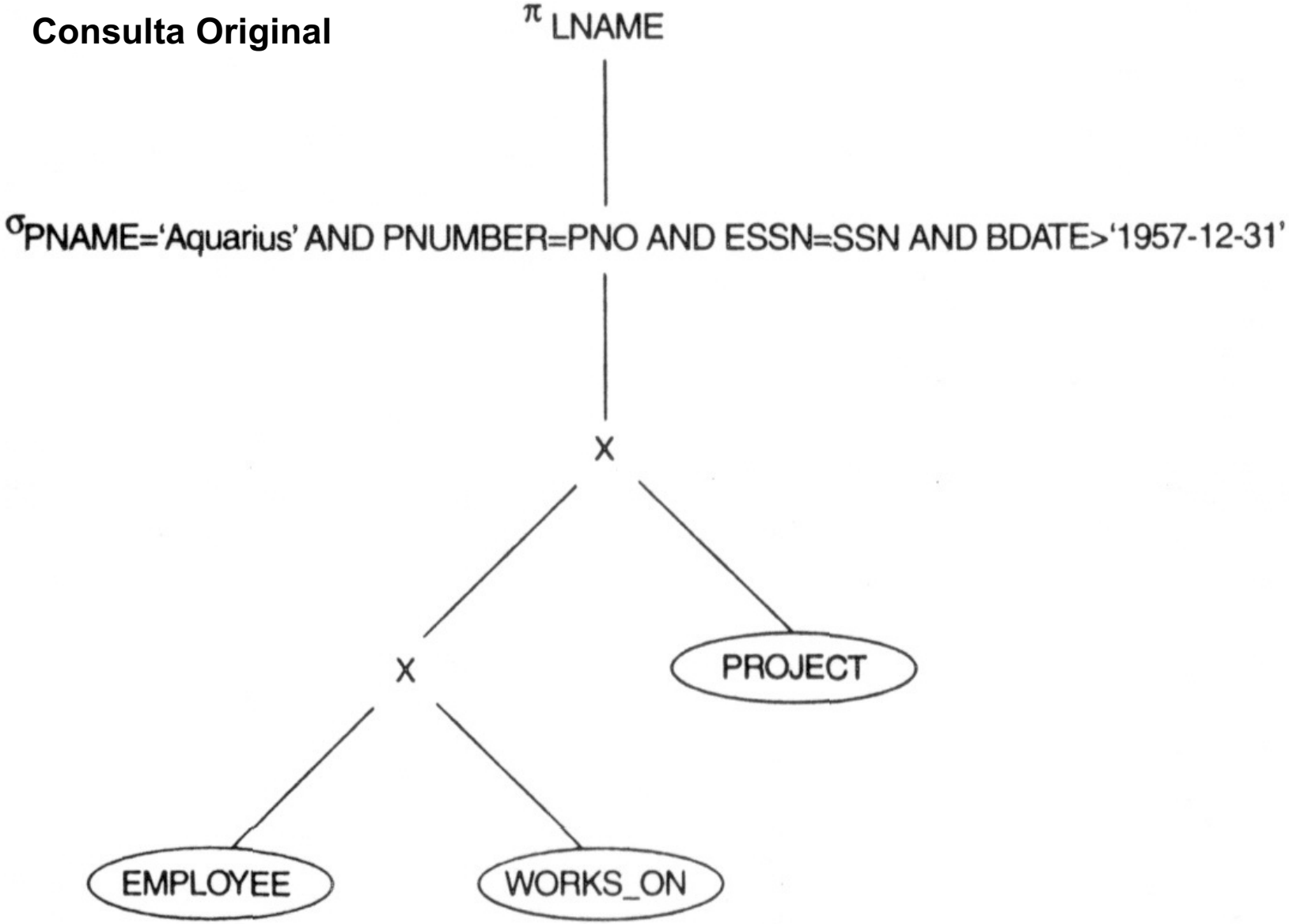
$$c_2 = PNUMBER = PNO,$$

$$c_3 = ESSN = SSN,$$

$$c_4 = BDATE > '1957-12-31',$$

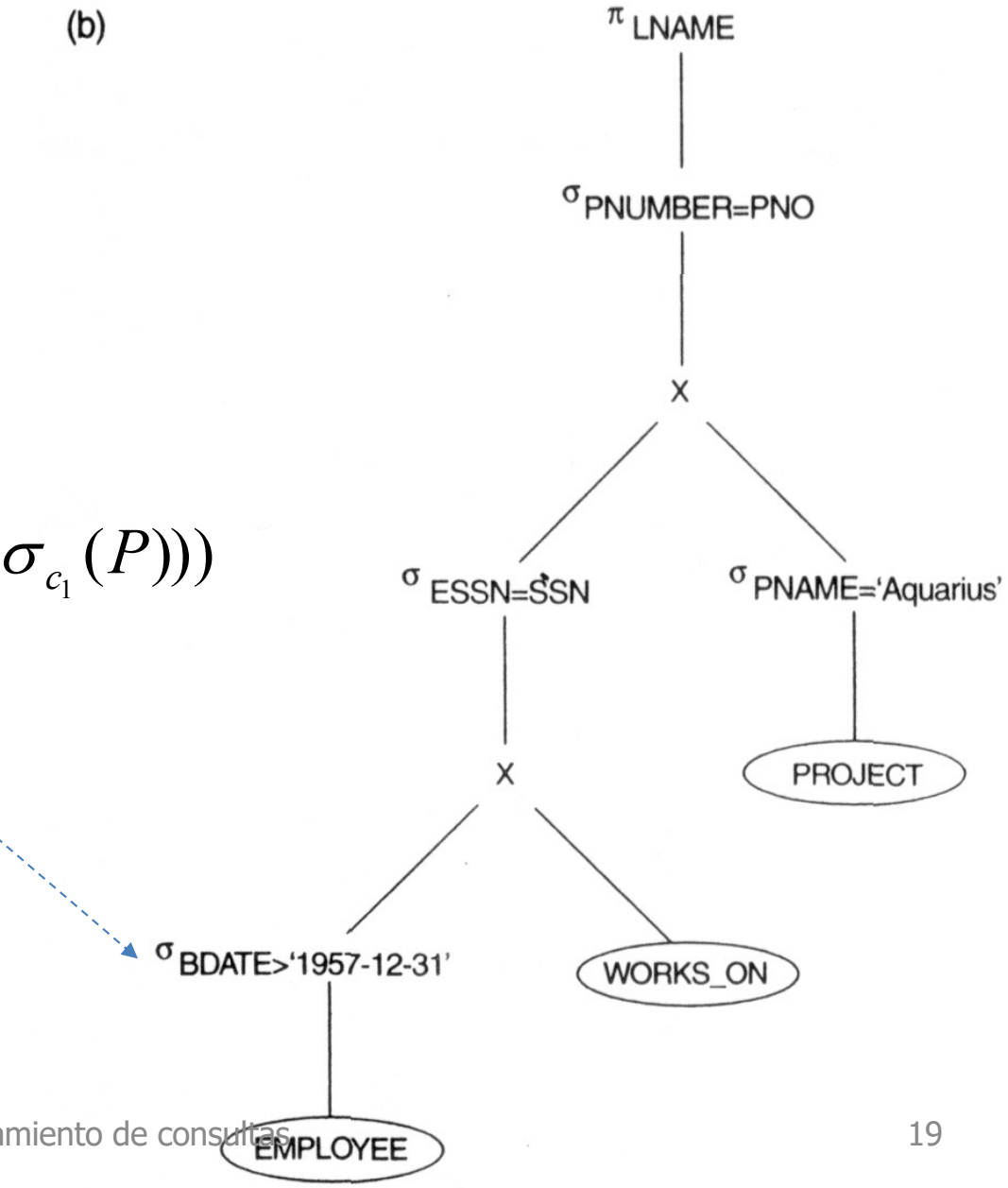
$$E = EMPLOYEE, W = WORKS\_ON, P = PROJECT$$

(a) **Consulta Original**



Con los pasos 1 y 2 del algoritmo se pasa a dibujo (b):

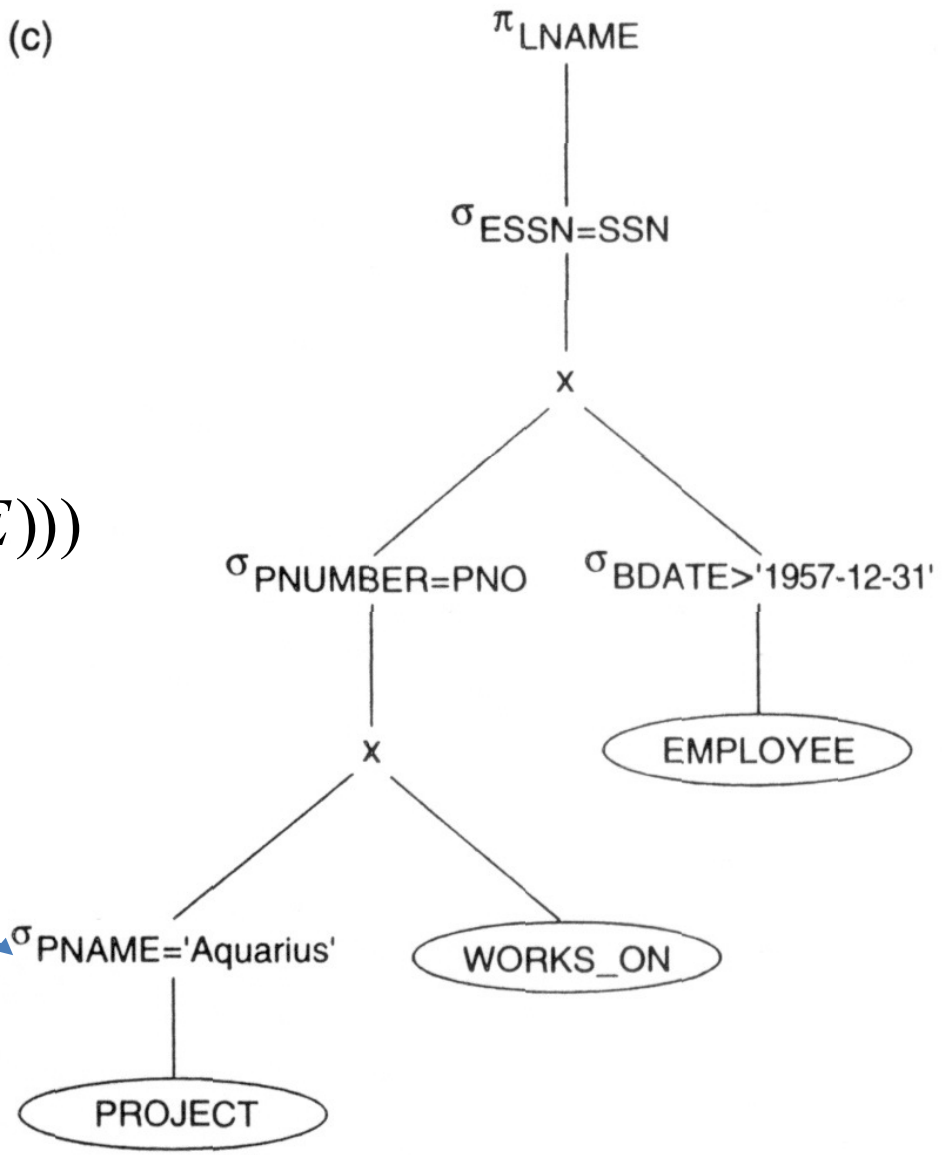
$$\pi_{LNAME}(\sigma_{c_2}(\sigma_{c_3}(\sigma_{c_4}(E)\times W)\times\sigma_{c_1}(P)))$$



Con el paso 3

se pasa al dibujo (c):

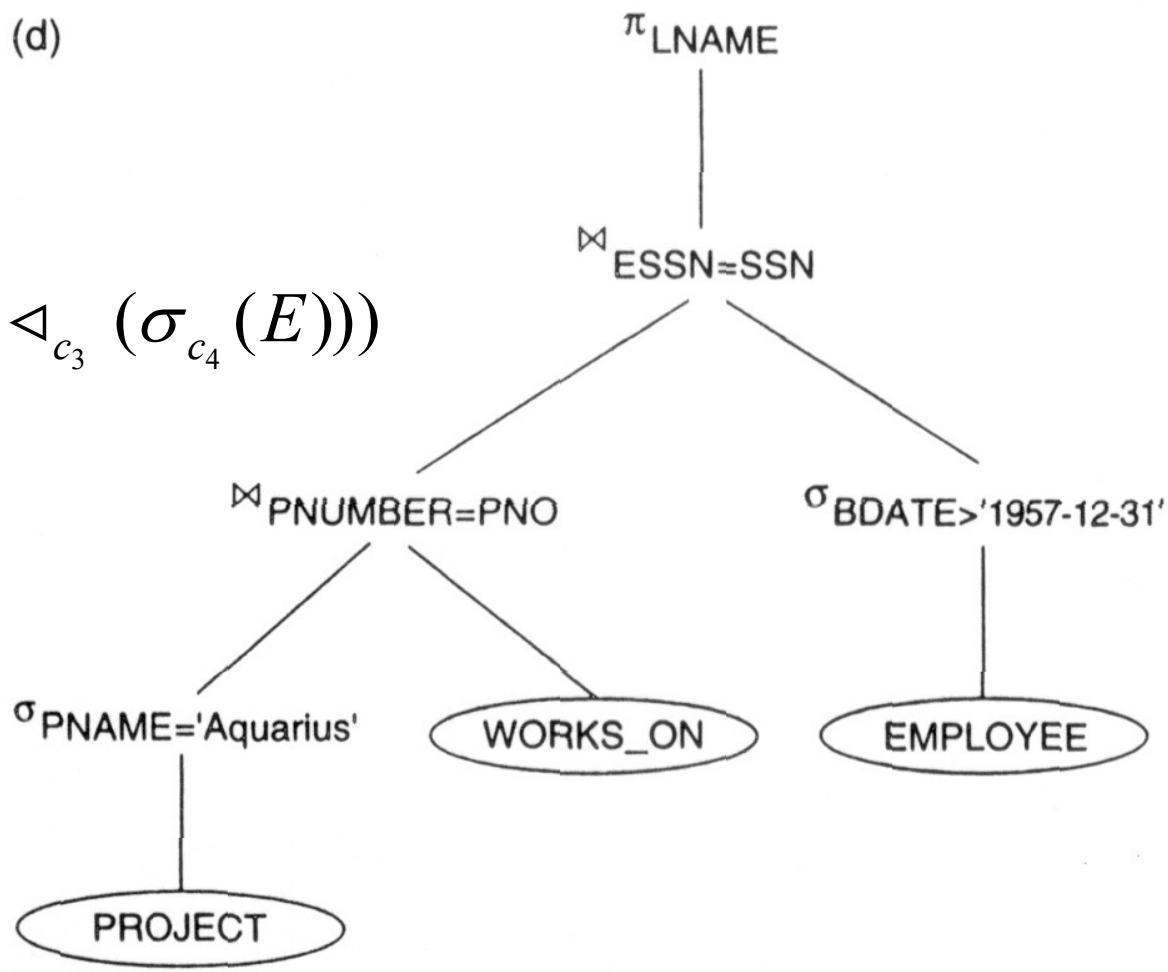
$$\pi_{LNAME}(\sigma_{c_3}(\sigma_{c_2}(\sigma_{c_1}(P)\times W)\times\sigma_{c_4}(E)))$$



Con el paso 4

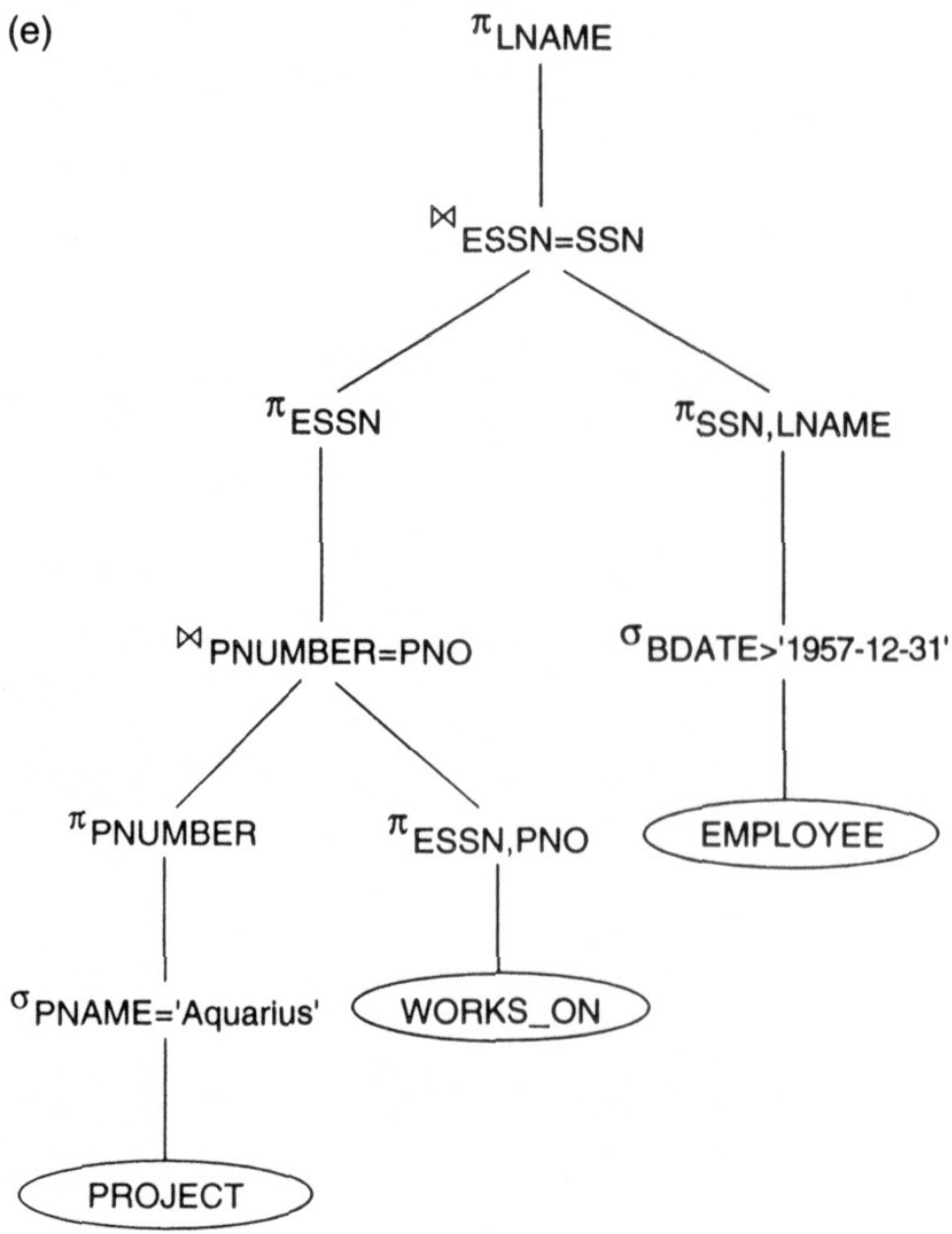
se pasa al dibujo (d)

$$\pi_{LNAME}((\sigma_{c_1}(P) \rhd\lhd_{c_2} W) \rhd\lhd_{c_3} (\sigma_{c_4}(E)))$$



Aplicando el paso 5

se pasa al dibujo (e):



Otro Ejemplo

- Dadas las Relaciones:

  P (P#,DESC,FAB)P: Pieza

  C (C#,NOM,DIR,CIUDAD)C: Cliente

  PE (P#,C#,CANT)PE: Pedido

```
SELECT P.DESC
FROM P,C,PE
WHERE PE.CANT > 100
   and C.CIUDAD = 'MADRID'
   and P.P# = PE.P#
   and C.C# = PE.C# ;
```

- "Obtener la descripción de las piezas servidas en una cantidad superior a 100 unidades a los clientes de Madrid"
- La condición de las unidades es la que da menos filas

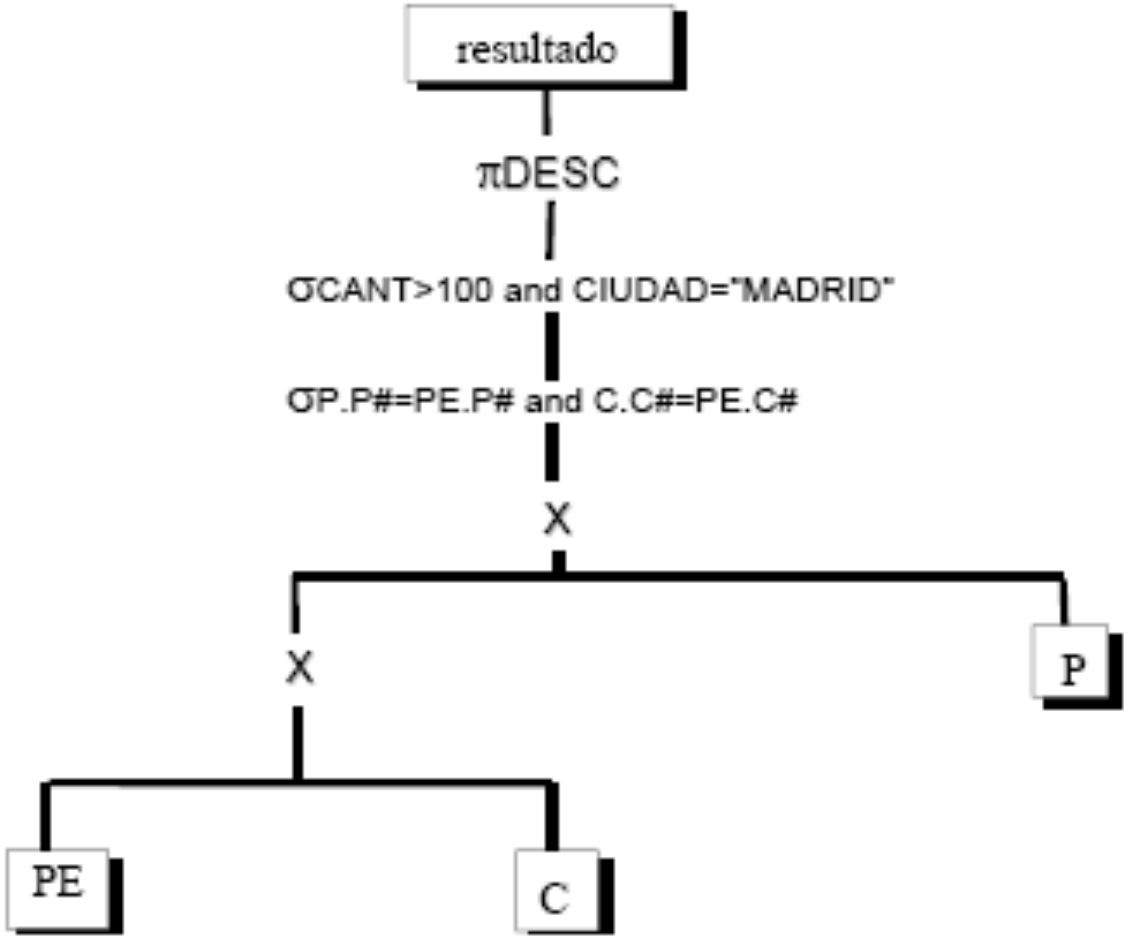
- Empezamos con la Representación en Algebra Relacional:

$$\pi_{DESC}\left(\sigma_{CANT>100 \text{ and } CIUDAD=\text{"MADRID"}}\left(\sigma_{P.P\#=PE.P\# \text{ and } C.C\#=PE.C\#}(P \times C \times PE)\right)\right)$$

Ejemplo

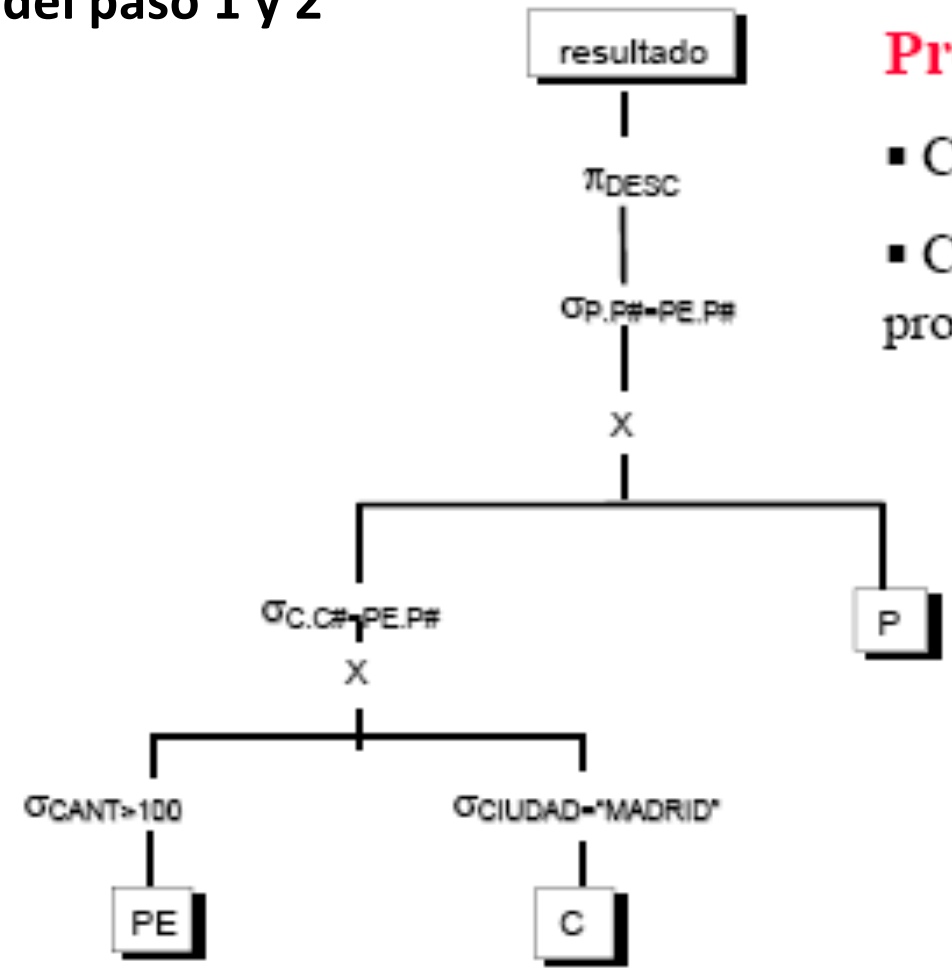
Arbol Canónico

para comenzar



Ejemplo

Después del paso 1 y 2

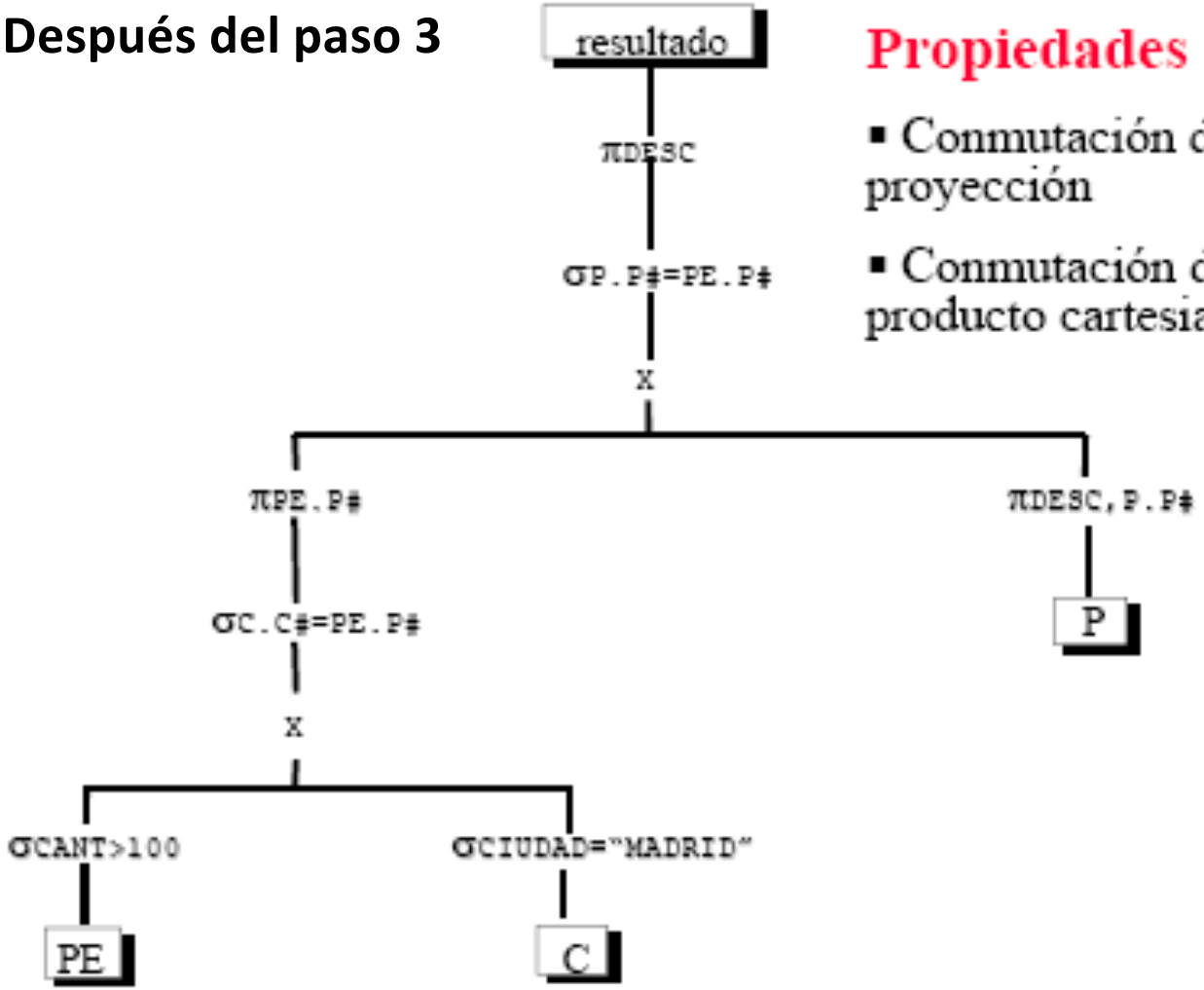


Propiedades aplicadas:

- Cascada de selecciones.
- Conmutación de selección y producto cartesiano.

Ejemplo

Después del paso 3



Propiedades aplicadas:

- Conmutación de selección y proyección
- Conmutación de proyección y producto cartesiano.

Falta aplicar el paso 4 y el 5 para terminar la optimización: hazlos como ejercicio

Elección de procedimientos de bajo nivel

- El Optimizador decide cómo evaluar el Árbol de Consulta obtenido
- Se basa en elementos tales como:
  - Existencia de índices u otras rutas de acceso: tipos de acceso posibles
  - Distribución de los valores de los datos almacenados.
  - Agrupamiento físico de los registros (bloques).
- Considera la consulta como un conjunto de
  - operaciones de “bajo nivel” (join, selección, etc.)
  - con ciertas interdependencias entre ellas.
- Ejemplo:
  - Para la eliminación de valores repetidos, una proyección
  - requerirá que los valores estén en un cierto orden, cosa que ha de
  - cumplir el resultado de la consulta que genera su relación objetivo,
  - Para ello añade una operación “sort”

Elección de procedimientos de bajo nivel

- Tiene una serie de procedimientos optimizados
  - para realizar cada una de estas operaciones de "bajo nivel"
  - con una medida de coste (para escoger el Plan de Consulta)
- Ejemplo:
  - Para un **join**, tiene varios procedimientos disponibles según el caso:
    - Si la condición es a través de una clave candidata.
    - Si el atributo de condición (de unión) está indexado.
    - Si el atributo de selección no está indexado
      - pero sí están agrupadas físicamente las filas.

Generación y elección de Planes de Consulta

- La generación se realiza combinando
  - los procedimientos de bajo nivel candidatos, uno por cada operación
  - Puede dar una explosión combinatoria de Planes

- La eficiencia de un optimizador depende
  - de la eficacia de la técnica heurística (Reglas Heurísticas) empleada
  - para la generación de planes de consulta y escoger uno como **Plan de Ejecución**
- La estimación de costos de un plan dependerá de:
  - El nº de operaciones de E/S de disco requeridas.
  - Utilización de CPU.
- Problema: una consulta implica generación de resultados intermedios

- Esto hace que la estimación de las operaciones de E/S
  - dependerá del tamaño de las tablas con estos resultados,
  - lo cual depende mucho de los valores **reales** de los datos
    - Que solo se pueden saber mediante ESTADISTICAS
    - Vemos después las de Oracle

ORACLE: Arboles de Consulta, Plan Ejecución

→ **En cada usuario hay una Tabla PLAN_TABLE para almacenar el Plan Ejecución**

→ **ATRIBUTOS interesantes:**

das un ID a tú explicación

Operación Bajo nivel

Modo de Ejecutar operación

Tabla, índice,..

```
statement_id    varchar2(30),
timestamp       date,
remarks         varchar2(4000),
operation       varchar2(30),
options         varchar2(30),
object_node     varchar2(128),
object_owner    varchar2(30),
object_name     varchar2(30),
object_instance number(38),
```

(CONT.)

```
object_type     varchar2(30),
optimizer       varchar2(255),
search_columns  number,
id              numeric(38),
parent_id       number(38),
position        number(38),
cost            number(38),
cardinality     number(38),
bytes           number(38),
CPU_COST        number(38),
IO_COST         number(38),
.....
other           long);
```

Num. fila de operación

Num. Fila de ope. padre de la presente

Estimación de filas devueltas por esa op.

Coste en Unidades "Oracle", depende de:
- CPU_COST : proporcional al núm. ciclos máquina
- IO_COST: proporcional al núm. de bloques datos leídos

ORACLE: Genera el Plan Ejecución en PLAN_TABLE

- **Crear el Arbol optimizado de una Consulta en la Tabla PLAN_TABLE:**

(NOTA: "SQL>" indica que se ejecuta en el interprete)

```
SQL> delete plan_table;          -- borra las tuplas de la tabla de otros planes
SQL> EXPLAIN PLAN
        SET STATEMENT_ID = 'unionc'          -- etiqueta de las filas de esta consulta
        INTO plan_table
        FOR (select * from cliente) union (select * from moroso);     -- consulta SQL que se explica
SQL> Explicado
```

- **Este árbol es el Plan de Ejecución:**

OPERATION	OPTIONS	OBJECT_NAME	PARENT_ID	ID
SORT	UNIQUE		0	1
UNION-ALL			1	2
TABLE ACCESS	FULL	CLIENTE	2	3
TABLE ACCESS	FULL	MOROSO	2	4

- **Para ver este resultado se necesita una Consulta Jerárquica**

ORACLE: Consulta Jerárquica de Plan_Table

Pasos de la ejecución de la Consulta Jerárquica de Plan_Table:

1) Selecciona las filas raíz con una condición : **start with id = 1**
2) Selecciona hijos de cada raíz: **connect by** condición entre dos filas (raíz e hijos)
3) Bucle iterativo: Sucesivas generaciones de hijos:
   connect by algún "padre actual" **prior id** que era hijo del anterior **parent_id**
4) Ordenar las filas según su atributo identificador **order by** id

CONSULTA: (resultado en transparencia anterior)

```
SQL> select operation, options, object_name, parent_id, id
        from plan_table
        connect by prior id=parent_id     --and statement_id= 'actuaT'
        start with id = 1                 -- and statement_id= 'actuaT'
        order by id;
```

→ **Necesita una Máscara para formatear mejor la salida**

ORACLE: Máscara para Formatear Salida

```
SQL> @p:\Publico\Prac3\MASCPLAN-13.sql
-------------------------------------------------------------
-- MASCARA PARA CONSULTAR LA TABLA DEL PLAN --
ttitle ' INFORME DEL PLAN  '

col operation     heading 'OPERACION' format a12 word_wrapped
col options       heading 'OPCIONES' format a12 word_wrapped
col object_name   heading 'TABLA'    format a12 word_wrapped
col cost          heading 'Coste'.   format a5
col cardinality   heading 'Filas'.   format a5
col parent_id     heading 'PADRE'.   format a5
col id            heading 'ID'       format a5

 select operation,options,object_name,cost,cardinality,parent_id,id
 from plan_table
 connect by prior id=parent_id    -- and statement_id= 'actuaT'
 start with id = 1                -- and statement_id= 'actuaT'
 order by id
 /
```

NOTA: "cost" y "cardinality" se ven solo en los ultimos ejemplo

ORACLE: Plan_Table, otros ejemplos

1º)
```
SQL> EXPLAIN PLAN
     SET STATEMENT_ID = 'unioncompl'
     INTO plan_table
     FOR (select * from cliente where DNI = '00000005') union
         (select * from moroso where NombreC = 'Client E');
```

2º)
```
---------- ahora consulta jerárquica con connect -----------------
SQL> select operation,options,object_name,parent_id,id
  2  from plan_table
  3  connect by prior id=parent_id --and statement_id= 'actuaT'
  4  start with id = 1            -- and statement_id= 'actuaT'
  5  order by id;
---------- INFORME DEL PLAN ----------------------------
OPERACION    OPCIONES     OBJECT_NAME  PARENT_ID         ID
-----------  -----------  -----------  ---------  ---------
SORT         UNIQUE                            0          1
UNION-ALL                                      1          2
TABLE ACCESS BY INDEX     CLIENTE              2          3
             ROWID
INDEX        RANGE SCAN   SYS_C0028915         3          4
TABLE ACCESS FULL         MOROSO               2          5
```

ORACLE: Plan_Table, otros ejemplos

3º) `SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());`

- Muestra la información de PLAN_TABLE de otro modo
- Da la operación que genera cada condición / predicado
- No está la jerarquía de operaciones, pero puede dar más detalles

```
-------------------------------------------------------------------------------------------
| Id  | Operation                      | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |               |     2 |   138 |     6  (84)| 00:00:01 |
|   1 |  SORT UNIQUE                   |               |     2 |   138 |     6  (84)| 00:00:01 |
|   2 |   UNION-ALL                    |               |       |       |            |          |
|   3 |    TABLE ACCESS BY INDEX ROWID| CLIENTE       |     1 |    69 |     1   (0)| 00:00:01 |
|*  4 |     INDEX UNIQUE SCAN          | SYS_C0028915 |     1 |       |     1   (0)| 00:00:01 |
|*  5 |    TABLE ACCESS FULL           | MOROSO        |     1 |    69 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   4 - access("DNI"='00000005')
   5 - filter("NOMBREC"='Client E')
```

4º) En SQLdeveloper, F10 sobre la consulta: da el árbol gráficamente.

ORACLE: Plan_Table, otros ejemplos

```
SQL> EXPLAIN PLAN                                  → Produce bucles anidados
     SET STATEMENT_ID = 'unioncompl'
     INTO plan_table
     FOR select * from cliente where DNI in
           (select DNI from moroso where NombreC = 'Client E');

-------  consulta a plan_table ----------- (falta la máscara)
SQL> select operation,options,object_name,parent_id,id
     from plan_table
     connect by prior id=parent_id --and statement_id= 'actuaT'
     start with id = 1              -- and statement_id= 'actuaT'
     order by id;
-------------- INFORME DEL PLAN ----------------------
OPERACION     OPCIONES      TABLA         PADRE         ID
------------ ------------ ------------ ----- ---------
NESTED LOOPS                                0          1
TABLE ACCESS FULL          CLIENTE          1          2
TABLE ACCESS BY INDEX      MOROSO           1          3
             ROWID
INDEX        UNIQUE SCAN   SYS_C0028917     3          4
```

ORACLE: Plan_Table, otros ejemplos

```
SQL> EXPLAIN PLAN
        INTO plan_table
          FOR (select * from cliente where DNI < '00000005');
```

→ Da información de Coste (en unidades de Oracle) y Filas

```
OPERACION     OPCIONES      TABLA          Coste Filas PADRE          ID
------------ ------------ ------------ ----- ----- ----- ----------
TABLE ACCESS BY INDEX      CLIENTE            2     5     0          1
             ROWID
INDEX        RANGE SCAN    SYS_C0035823       1     5     1          2
```

→ **Eficiencia:** coste y filas procesadas (espacio memoria)

```
EXPLAIN PLAN
INTO plan_table FOR
     (select DNI from moroso where NombreC = 'Client E');
```

```
OPERACION     OPCIONES      TABLA          Coste Filas PADRE          ID
------------ ------------ ------------ ----- ----- ----- ----------
TABLE ACCESS FULL          MOROSO             3     1     0          1
```

ORACLE Plan_Table: Operaciones y métodos acceso

- **Full table scan**: Lee todas las filas de la tabla y aplica el where después. Se aplica si:
  - se accede a porciones grandes de la tabla
  - Y no hay índice
- **Table access by ROWID**: lo usa para acceder a cada fila
  - ROWID: indica el DATAFILE, en qué bloque está, y posición dentro de él
- **Index unique scan**: accede a una sola fila por índice único

- **Index range scan**: accede a varias entradas de índice. Se aplica
  - Cuando hay condiciones de no igualdad o el índice no es único

- **Index skip scan** : salta el primer atributo de la clave si
  - ese atributo no se usa en la consulta
- **Full Index scan**: lee solo bloques hoja del índice pero no lee filas de la tabla.
  - Se aplica en consultas donde todos sus atributos están en el índice.

ORACLE Plan_Table: Operaciones y métodos acceso

- **Hash joins**: cuando se une una tabla pequeña a otra bastante más grande:
  - Usa la tabla menor para crear una tabla hash en memoria con la clave de join

- **Nested Loops joins** (puede que aparezcan dos seguidos)
  - son dos bucles anidados para tablas pequeñas si puede acceder por
    - Índice a la 2ª
  - para cada fila de la 1ª tabla accede por índice a las filas de la 2ª

- **Sort Merge joins** cuando la condición de unión no es igualdad: <, <=, >, or >=
  - Más eficiente que nested loops para tablas grandes
  - Hace dos operaciones: sort por clave de unión cada tabla y
  - un merge de las tablas ordenadas

- **Cartesian join** es producto Cartesiano
  - Si no hay atributos comunes en la condición de la unión
  - o si las tablas son pequeñas

ORACLE: Reglas de Optimización, Combina dos Tablas

Situaciones en consultas de *Combinación de dos tablas* con una columna común:

- Las columnas comunes no tienen (o no pueden usar) índices: muy costosa
- Una de las dos columnas tiene índice:
  - Recorre secuencialmente la tabla sin índice de la consulta
- Las dos tienen índices:
  - Recorre secuencialmente la primera tabla del 'from'
  - Se recomienda poner la más pequeña al final

Además:

- Si la consulta devuelve más del X % de filas(0,1-15): recorre secuencialmente
- Si tiene una condición,
  - hay una prioridad según el tipo de columna usada (índice, max,...)
  - Si el atributo está dentro de una función, no usa índice: definir otro tipo de índice

ORACLE: Elección de índices B+

- Un **índice** acelera consultas y relentiza actualizaciones: comprobar el mejor
- Es importante para el Diseñador decidir bien, *si crear o no índices*:
  - Para evitar recorridos secuenciales en las consultas
- **NO crear índices** en columnas:
  - Que se modifiquen frecuentemente
  - Ó con pocos valores diferentes (usar índice Bitmap)
  - Ó con consultas con funciones sobre el índice (usar índice de Función)
- **SÍ crear índices** cuando las columnas son:
  - Claves primarias (oracle por defecto las crea) :
    - si son muchos atributos: crear otro atributo único para la PK
  - Claves ajenas: aceleran las uniones
  - Que sirven frecuentemente para búsquedas
  - Que sirven de criterio de ordenación o agrupación.
  - Rango grande de valores o valores únicos o pocas filas con mismo valor

ORACLE: Tipos principales de Índices

TODOS los tipos: un índice pueden ser sobre varios atributos

A) Índices normales: forman árboles B+

- CREATE INDEX nombre_indice ON nombre_tabla (columnas);
- Nomenclatura para el *nombre_indice* : idx_tabla_columna

B) Índices con Bitmap: para atributos con pocos valores diferentes

- Índices B+ sobre esos atributos son muy ineficientes, por eso existen otros. . .
- CREATE BITMAP INDEX nombre_index ON nombre_tabla (columnas);

ROWID	H	M
AAA1	1	0
AAA2	0	1
AAA3	1	0
AAA4	0	1
AAA5	0	1

ORACLE: Tipos principales de Índices

C) Indices basados en funciones:

- En una SELECT, no puede usar un índice si está dentro de una función

```
select direccion  from Empleado
 where upper(nombre) = 'JUAN'  or
 ROUND(suma_ventas) = 105 ;
```

– Ejemplos de tipos de funciones que se pueden poner:

```
ROUND(sale_amount)
(sale_amount * 0.2)
SUBSTR(first_name, 1, 10)
```

– CREATE INDEX idx_nombre_indice ON tabla(Funcion(atributo));

EJ: CREATE INDEX idx_empleado_suma ON Empleado(ROUND(suma_ventas));

Implementación de índices: Árboles equilibrados B+

- Evitar degradación del rendimiento cuando crece el fichero
- Arbol B+: es una estructura de datos, enlazada
  - con más de un puntero en cada uno de sus nodo
- Cada nodo contiene (max: n punteros y n-1 claves)
  - (Ci) Claves de registros (que estarán en algún bloque de datos)
  - (Pi) Punteros: direcciones físicas de la ubicación de:
    - nodos que contienen esas claves
    - Bloques de datos que contienen los registros con esas claves
- Los árboles equilibrados B+ mantienen la eficiencia
  - B-tree = balanced tree = árbol equilibrado = los caminos de la raíz a los nodos hoja son de la misma longitud siempre en todas las ramas
  - Cada nodo interno (no hoja) tiene entre $\lceil n/2 \rceil$ y n hijos, con n fijo para cada árbol particular.
  - El nodo raíz tiene entre 1 y n hijos.
  - Los nodos hoja contienen entre $\lceil (n-1)/2 \rceil$ y n-1 valores (todas las claves)

Índices de árboles equilibrados B+ : estructura

P_1	C_1	P_2	...	P_{n-1}	C_{n-1}	P_n

- La estructura de un nodo raíz, interno u hoja es: (ver ejemplo más adelante)

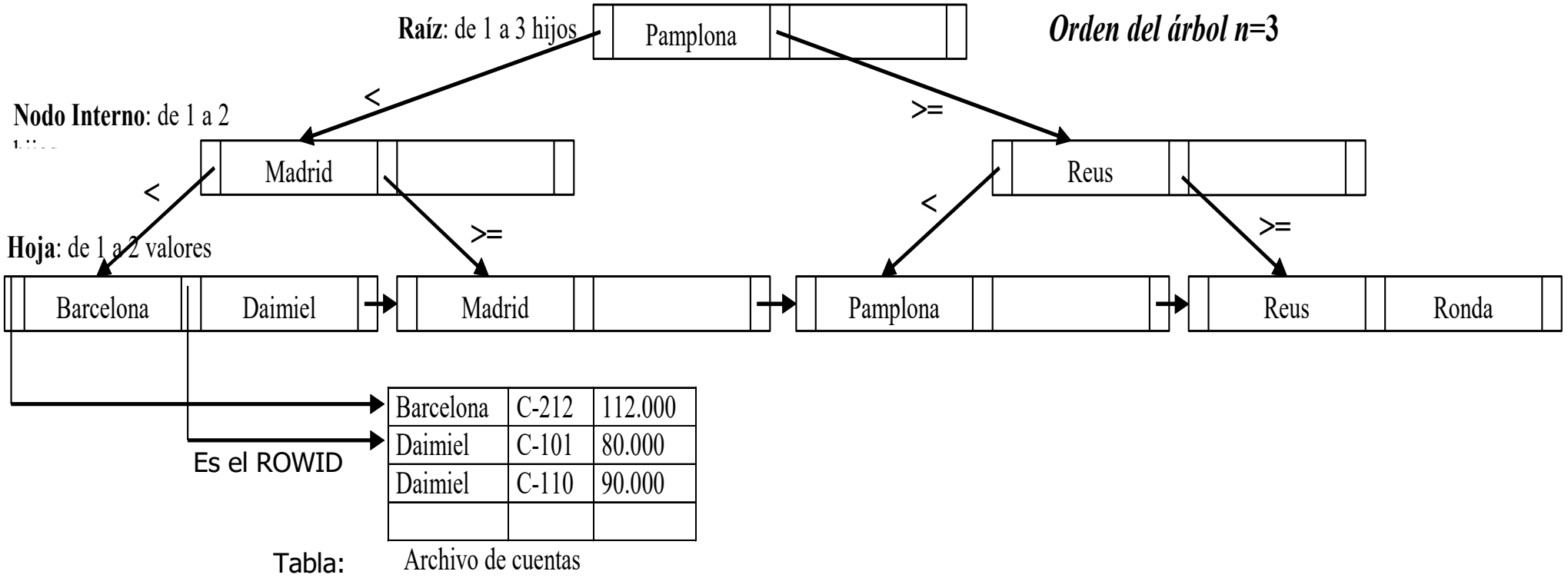
- **P_i** son punteros.
  - Si el nodo es interno o raíz, el puntero apunta a otro nodo de un nivel inferior.
  - Si es hoja, apunta a un registro o a un bloque de punteros a registros.
    - El bloque sólo se usa si la clave de búsqueda no forma una clave primaria y si el fichero no está ordenado según la clave de búsqueda.
- **C_i** son valores de la clave y están ordenados.
  - Los valores **no** se repiten dentro del mismo nivel.
  - Para un mismo nivel del árbol los valores de un nodo son todos anteriores (o posteriores, según el orden) al de otro nodo.

- El puntero P_i de un nodo raíz o interno apunta a un nodo con
  - valores de la clave estrictamente menores que los de C_i,
- el puntero P_{i+1} al nodo con valores mayores o iguales.
- Los nodos internos del árbol B+ forman un índice multinivel (disperso) sobre los nodos hoja.

Índices de árboles equilibrados B+ : ejemplo

Atributo con índice: **clave de búsqueda**
Ej.: NombreCiudad, valor : Pamplona

Punteros: apuntan al bloque indicado
(no al valor donde va la flecha)



Índices de árboles equilibrados B+: Algoritmo (Búsqueda) Consulta de un valor de clave "c"

1. Se examina el nodo raíz para buscar el menor valor de la clave búsqueda mayor que c.
   - Supongamos que este valor de la clave de búsqueda es c_i
   - Seguimos el puntero p_i hasta cierto nodo.
2. (un extremo del nodo) Si $c < c_1$, entonces seguimos p_1 hasta otro nodo.
3. (el otro extremo del nodo) Si se tienen m punteros en el nodo y $c >= c_{m-1}$ entonces se sigue p_m hasta otro nodo.
4. Una vez más, se busca el menor valor de la clave de búsqueda que es mayor que c para seguir el puntero correspondiente.
5. Finalmente se alcanza un nodo hoja, se busca la **c** dentro del nodo
6. Cuando encuentra la c_X igual a **c**:

   el puntero p_X apunta a la fila/registro deseado (o bloque si no es PK)

EJ.: a) Busca "Daimiel" b) Busca "Ronda"

Índices de árboles equilibrados B+
Consulta de un valor de clave "c"

- Para procesar una consulta se tiene que recorrer un camino en el árbol desde la raíz hasta algún nodo hoja.
  - Si hay C valores de la clave de búsqueda en el fichero, este camino no será más largo que $\lceil \log_{\lceil n/2 \rceil}(C) \rceil$.
  - Lo normal es que cada nodo tiene mismo tamaño que un bloque de disco
- Ej:
  - Con bloque 4Kb, clave 12 bytes, puntero 8 bytes, n es del orden de 200. Con clave de 32 bytes, n=100.
  - Con n = 100 y 1.000.000 de valores de la clave → $\lceil \log_{50}(1.000.000) \rceil$ = 4 accesos a bloques.
- Si el nodo raíz se almacena en memoria intermedia → 3 accesos a disco.

Índices de árboles equilibrados B+: Diferencias con los árboles binarios

- Los nodos de los B+ son grandes y con muchos hijos
  - A diferencia de los pequeños nodos de los binarios con sólo dos hijos.

- Los B+ son anchos y bajos
  - Los binarios altos y estrechos.

- C=1.000.000 y n=100 (aridad B+) $\lceil \log_{50}(1.000.000) \rceil = 4$ accesos a bloques.
  - En árboles binarios $\lceil \log_2 (1.000.000) \rceil = 20$ accesos a bloques

Índices de árboles equilibrados B+: Borrado e inserción

- Ambas operaciones son más difíciles de llevar a cabo para los nodos internos. Problemas que podemos encontrar
  - Borrado: Puede ser necesaria la recombinación de nodos si se viola $\lceil n/2 \rceil$.
  - Inserción: Puede ser necesaria la división de nodos si se viola n.
- Por tanto las operaciones posibles son:
  - Inserción
  - Inserción con división
  - Borrado
  - Borrado con Recombinación

- El borrado e inserción son complicados pero consumen poco tiempo:
  - Necesitan reorganizar nodos: punteros y claves

Índices de árboles equilibrados B+: Inserción

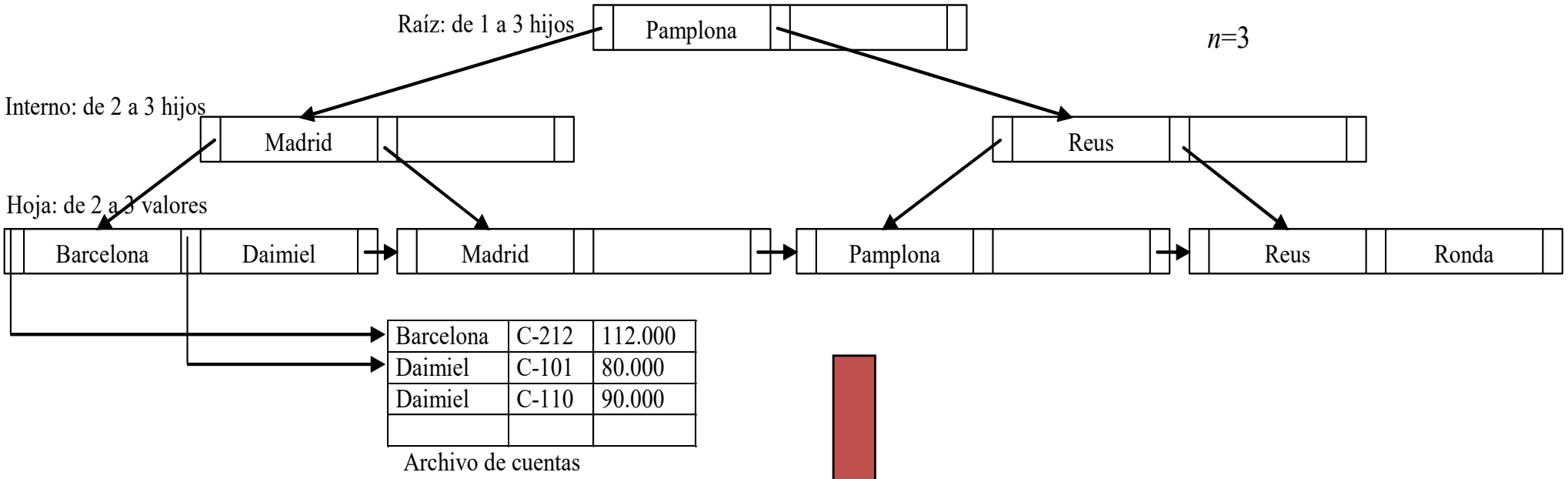
- Se busca un nodo hoja donde debería estar el valor de la clave “c” de búsqueda.
- Si “c” ya aparece en el nodo hoja:
  - se inserta el nuevo registro en el archivo de datos / tabla
- Si “c” no aparece en el nodo hoja
  - se inserta el valor “c” en el nodo hoja conservando el orden
  - Se inserta el nuevo registro en el archivo

- Las variaciones surgen cuando “c” no cabe en el nodo

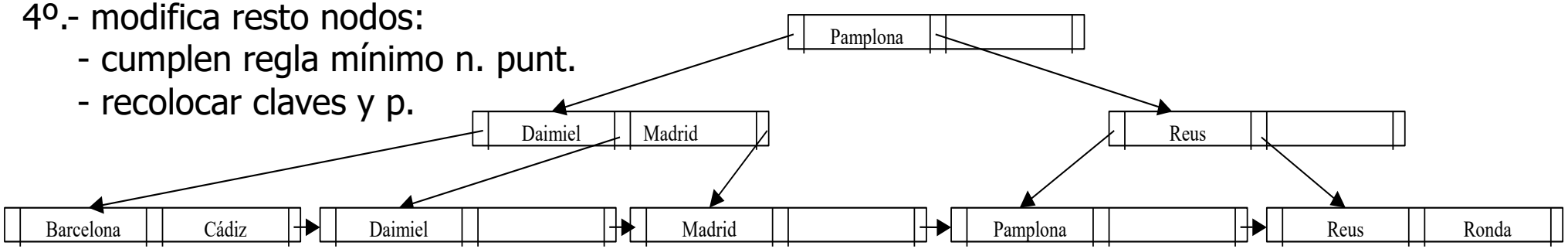
Índices de árboles equilibrados B+: Inserción con división

- Queremos insertar un registro cuyo valor de nombre-sucursal es “Cádiz” .
  - Usando el algoritmo de búsqueda, “Cádiz” debería estar en el nodo que incluye “Barcelona” y “Daimiel”.
    - No hay sitio para insertar el valor de la clave de búsqueda “Cádiz”.
  - Por tanto, se divide el nodo en otros dos nodos.
    - El primero con $\lceil n/2 \rceil$ valores y el segundo con el resto.
    - Se pueden provocar divisiones en los padres.
  - Necesitan reorganizar nodos: punteros y claves
  - Existe el algoritmo paso a paso, pero no lo veremos

Índices de árboles equilibrados B+ Inserción con división: "Cadiz"



1º.- Busca el lugar donde debe estar
2º.- ese nodo está lleno
3º.- modifica nodos hoja
4º.- modifica resto nodos:
- cumplen regla mínimo n. punt.
- recolocar claves y p.



Índices de árboles equilibrados B+ : Borrado

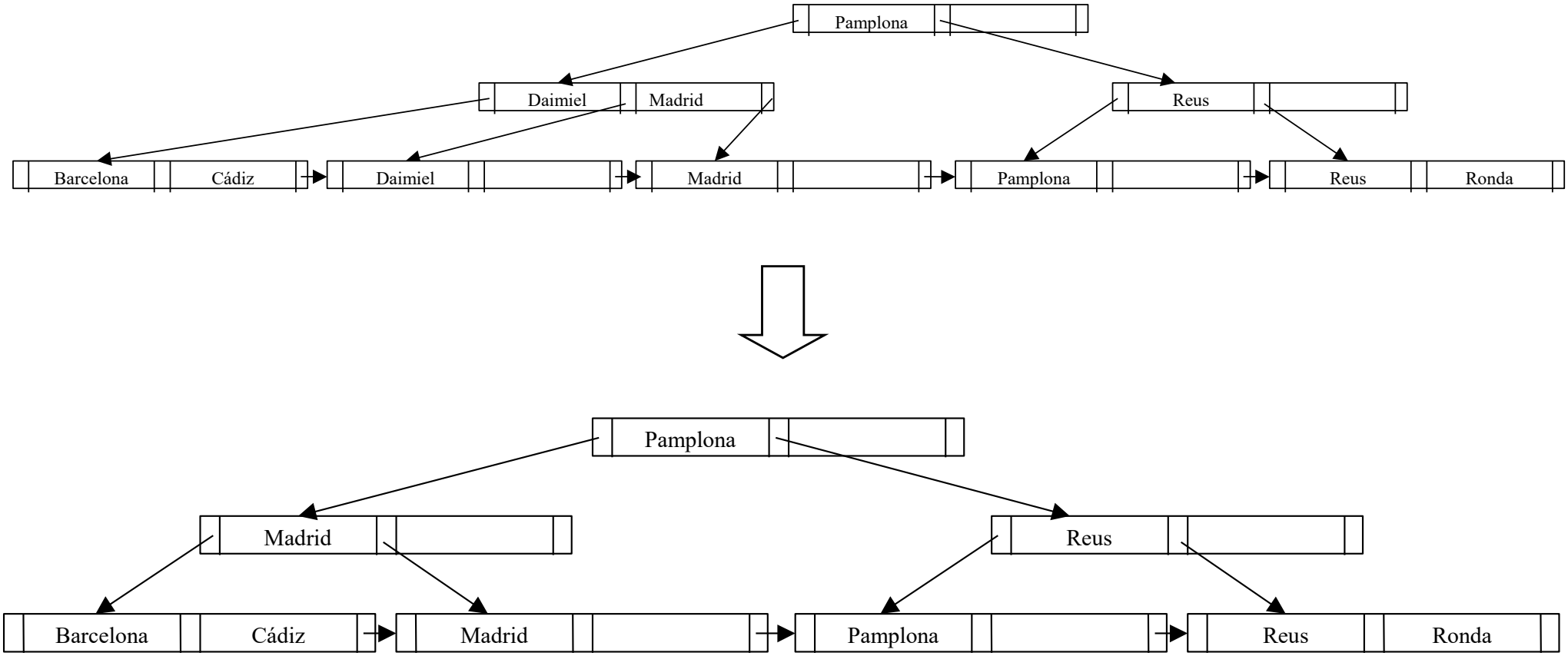
- Se busca el registro con clave “c” a borrar y se elimina del fichero
  - Técnica de buscar ya explicada antes
- Se borra el valor de la clave de búsqueda del nodo hoja
- si el bloque de la hoja se queda vacío: se borra

- Hay variaciones según los valores de claves que quedan:
  - Reorganizar punteros y nodos

Índices de árboles equilibrados B+ Borrado con recombinación

- Queremos borrar “Daimiel” del árbol B+.
- Para ello se localiza la entrada “Daimiel” usando el algoritmo de búsqueda.
- Cuando se borra la entrada “Daimiel” de su nodo hoja, la hoja se queda vacía.
  - Ya que en el ejemplo n = 3 y $0 < \lceil (n - 1)/2 \rceil$, este nodo se debe borrar del árbol B+. Para borrar un nodo hoja se tiene que borrar el puntero que le llega desde su padre.
- La operación de borrado deja al nodo padre, el cual contenía tres punteros, con sólo dos punteros.
  - Ya que $2 >= \lceil n/2 \rceil$, el nodo es todavía lo suficientemente grande y la operación de borrado se completa.

Índices de árboles equilibrados B+
Borrado “Daimiel” con recombinación :

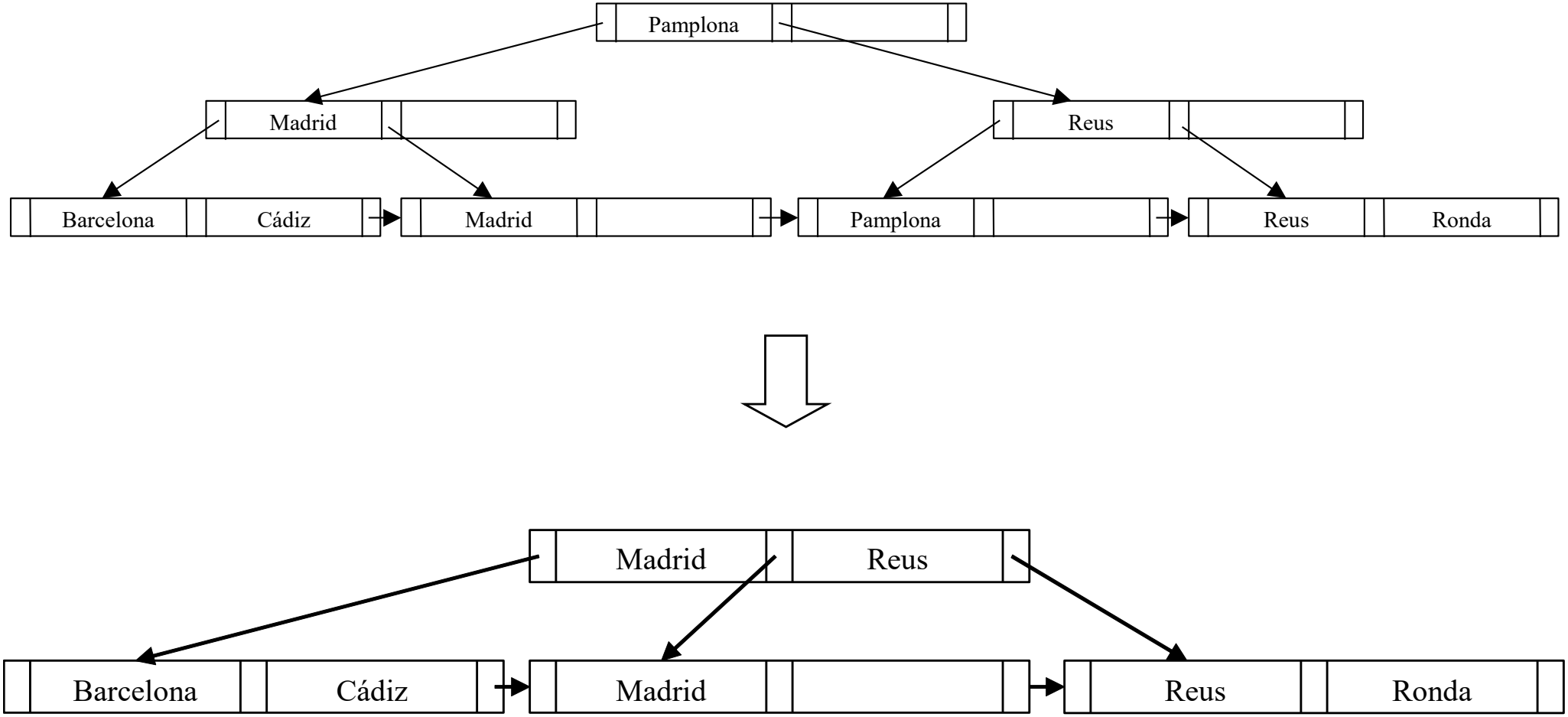


Índices de árboles equilibrados B+ Borrado con recombinación (borrar Pamplona)

- Cuando un borrado se hace sobre el padre de un nodo hoja,
  - el propio nodo padre podría quedar demasiado pequeño.
  - Si se borra "Pamplona" de **la figura obtenida**, un nodo hoja se quede vacío.
- Cuando se borra el puntero a este nodo en su padre,
  - el padre sólo se queda con un puntero. Pero está prohibido porque n = 3, $\lceil n/2 \rceil = 2$
  - Ya que el nodo padre contiene información útil, no podemos simplemente borrarlo.

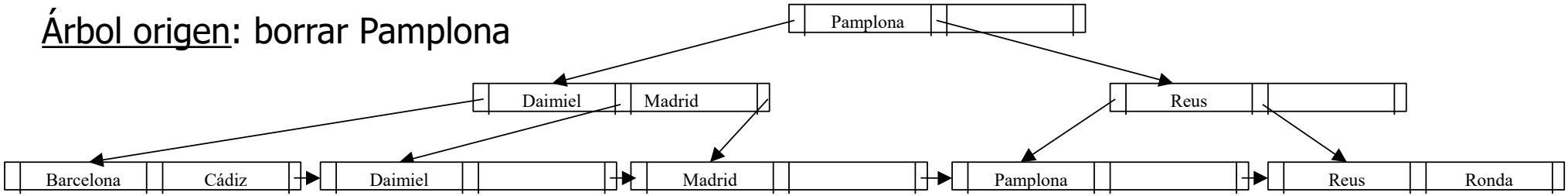
- Se busca el nodo hermano
  - Que es el nodo interno que contiene al menos una clave de búsqueda "Madrid"
  - Como tiene sitio se funden estos nodos y se pone la información del
    - nodo prohibido que quedó demasiado pequeño, así que…
  - El nodo hermano ahora contiene las claves "Madrid" y "Reus".
    - El otro nodo (el nodo que contenía solamente la clave de búsqueda "Reus"):
      - ahora tiene información redundante y se puede borrar desde su padre.
- La raíz tiene solamente un puntero hijo como resultado del borrado,
  - así que éste se borra y el hijo solitario se convierte en la nueva raíz.
  - De esta manera la profundidad del árbol ha disminuido en una unidad.

Índices de árboles equilibrados B+ Borrado con recombinación (borrar Pamplona)

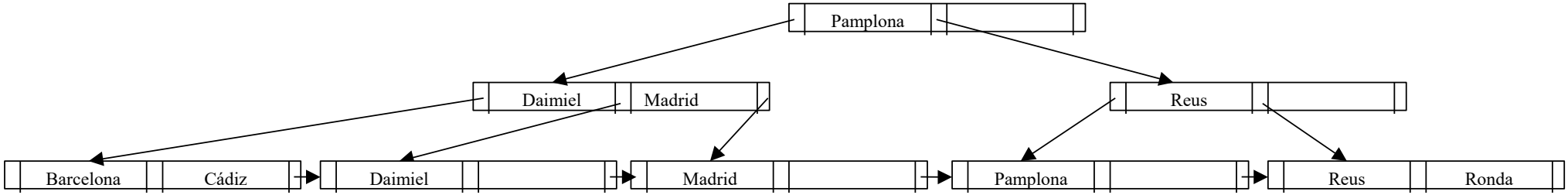


Índices de árboles equilibrados B+ Borrado con recombinación

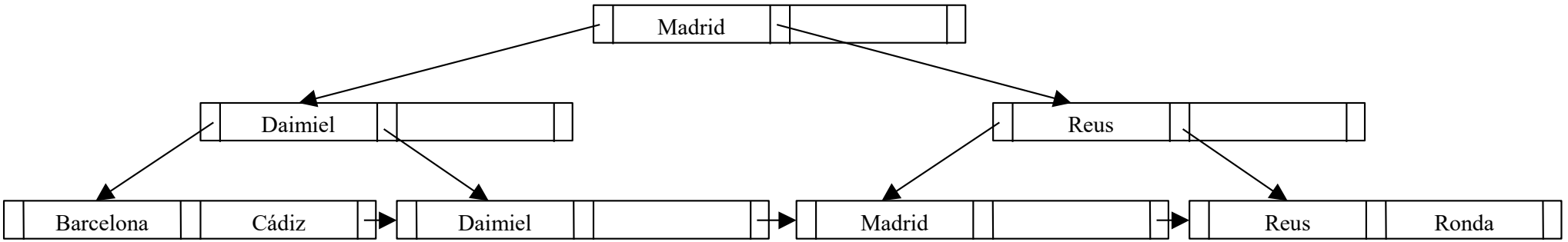
- No siempre es posible fundir nodos. Como ejemplo se borrará “Pamplona” del árbol B+ que tiene Daimiel y Cadiz:
  - El nodo hoja que contiene “Pamplona” se queda vacío.
  - El padre (Reus) del nodo hoja se queda con sólo un puntero.
  - De cualquier modo, en este ejemplo, el nodo hermano(Daimiel,Madrid) contiene ya el máximo número de punteros: tres. Así, no puede acomodar a un puntero adicional.
- La solución en este caso es *redistribuir* los punteros de tal manera que cada hermano tenga dos punteros.
  - La redistribución de los valores necesita de un cambio en el valor de la clave de búsqueda en el padre de los dos hermanos.



Índices de árboles equilibrados B+ Borrado con recombinación (borrado Pamplona)



1º.- modifica nodos hoja
2º.- modifica resto nodos:
- cumplen regla mínimo n. punt.
- recolocar claves y p.



ORACLE: Activar Estadísticas de tablas e índices

- Oracle puede usar estadísticas para escoger el plan de ejecución
- Ralentiza la ejecución del plan.

```
ANALYZE {TABLE nombreTabla
                        {{COMPUTE | ESTIMATE | DELETE} STATISTICS}
                        [SAMPLE { n percent | n rows}]
                        VALIDATE STRUCTURE [CASCADE] . . .
                }
                [ { FOR TABLE | FOR ALL [indexed] COLUMNS|
                    FOR COLUMNS nombreAtributo} ] ;
```

- Ejemplo 1:

```
analyze table cliente
   compute statistics for columns direccion;
```

- Ejemplo 2:

```
analyze table cliente
validate structure cascade;
```

(si *cascade* : también índices)

- Otro modo de obtener estadísticas:
  - **los Paquetes DBMS_XPLAN , DBMS-STATS** y **DBMS_UTILITY**
  - **El Optimizer usa DBMS_STATS para calcular costes**

ORACLE: (en B+) Resultado de Estadísticas con “Analize”

- Las estadísticas están en tablas del sistema
  - ALL_TABLES, USER_TABLES, DBA_TABLES
- Los conceptos más prácticos son: (tabla USER_TABLES)
  - **nr** número de filas, → NUM_ROWS
  - **br** número de bloques(asignados y ocupados), → BLOCKS , EMPTY_BLOCKS
  - tamaños (medias) de fila. → AVG_ROW_LEN
- Sobre índices: (tabla USER_INDEXES)
  - **AAi** número de niveles (altura del árbol B+) → BLEVEL
  - **bi** bloques hoja, → LEAF_BLOCKS
  - V(A,r) número de valores diferentes de la clave, → DISTINCT_KEYS
  - tamaños (medias)

Bibliografía

- [ACPT00] P. Atzeni, S. Ceri, S. Paraboschi y R. Torlone, “Database Systems. Concepts, Languages and Architectures”, McGraw-Hill, 2000.
- [Dat93] C.J. Date, "Introducción a los Sistemas de Bases de Datos", Addison-Wesley, Reading Massachusetts, 1993.
- [EN00] R. Elmasri y S.B. Navathe, "Fundamentals of Data Base Systems", Addison-Wesley, 2000. Apartados 8.1, 8.2, 8.3, 8.5
- [Pres98] Pressman, R.S., “Ingeniería del software. Un enfoque práctico”, McGraw-Hill, 1998. Apartado 8.4
- [SKS98] SILBERSCHATZ, A., KORTH, H.F., SUDARSHAN, S. "Fundamentos de Bases de Datos", 3ª edición, McGraw-Hill, 1998. Apartados 8.1, 8.2, 8.4
- [Ull98] J.D. Ullman, "Principles of Database and Knowledge Base Systems", Vol. I y II, Computer Science Press, 1998.
- [Wir94] N. WIRTH, “Algoritmos + estructuras de datos = programas”, 1994 (reimpresión). Apartado 8.4.1
- **John Kirkwood.** High Perfomance Relatinal Databases Design. Ed. Ellis Horwood, 1993.
- Diseño y Optimización de Bases de Datos: *Optimización de Consultas. http://www.oei.eui.upm.es/Asignaturas/BD/DYOBD/OPCONS.pdf*

Operaciones para la equivalencia de las consultas (solo como referencia, no las usamos)

- Sean:
  - θ condición simple (en el algoritmo era C_i)
  - L lista de atributos
  - R, S y T relaciones
  - Attrib(R) es el conjunto de atributos que aparecen en R

Apéndice: Operaciones

1) Cascada (secuencia) de σ :

$$\sigma_{\theta_1 \wedge \ldots \wedge \theta_n}(R) = \sigma_{\theta_1}(\cdots(\sigma_{\theta n}(R)\cdots)$$

2) Conmutatividad de σ :

$$\sigma_{\theta_1}(\sigma_{\theta 2}(R)) = \sigma_{\theta 2}(\sigma_{\theta 1}(R))$$

3) Cascada de π :

$$\pi_{L_1}(\cdots(\pi_{Ln}(R))\cdots) = \pi_{L_1 \cap L_2 \cap \cdots \cap L_n}(R)$$

donde $L_1 \subseteq L_2 \subseteq \cdots \subseteq L_n$

4) Conmutatividad de σ con π : $\pi_L(\sigma_\theta(R)) = \sigma_\theta(\pi_L(R))$

5) Conmutatividad de ×: $R \times S = S \times R$

6) Conmutatividad de ⋈ $R \rhd\lhd S = S \rhd\lhd R$

Operaciones

7) Distributividad de σ con $\bowtie$:

i) $\sigma_\theta(R \bowtie S) = \sigma_\theta(R) \bowtie S$, si $attrib(\theta) \subseteq attrib(R)$

, si

ii) $\sigma_{\theta_1 \wedge \theta_2}(R \bowtie S) = (\sigma_{\theta_1}(R)) \bowtie (\sigma_{\theta_2}(S))$,

si $attrib(\theta_1) \subseteq attrib(R) \wedge attrib(\theta_2) \subseteq attrib(S)$

8) Distributividad de σ con × :

i) $\sigma_\theta(R \times S) = \sigma_\theta(R) \times S$, si $attrib(\theta) \subseteq attrib(R)$

ii) $\sigma_{\theta_1 \wedge \theta_2}(R \times S) = (\sigma_{\theta_1}(R)) \times (\sigma_{\theta_2}(S))$

, si $attrib(\theta_1) \subseteq attrib(R) \wedge attrib(\theta_2) \subseteq attrib(S)$

Operaciones

9) Distributividad de π con $\times$: $\pi_L(R \times S) = (\pi_{L_1}(R)) \times (\pi_{L_2}(S))$

si $L_1 \subseteq attrib(R) \wedge L_2 \subseteq attrib(S)$

10) Distributividad de π con $\bowtie$:

i) $\pi_L(R \rhd\lhd_\theta S) = (\pi_{L_1}(R)) \rhd\lhd_\theta (\pi_{L_2}(S))$,

si $L_1 \subseteq attrib(R) \wedge L_2 \subseteq attrib(S)$

ii) $\pi_L(R \rhd\lhd_\theta S) = \pi_L((\pi_{L_1 \cup C_1}(R)) \rhd\lhd_\theta (\pi_{L_2 \cup C_2}(S)))$,

si $L_1 \subseteq attrib(R) \wedge L_2 \subseteq attrib(S) \wedge$

$C_1 = attrib(\theta) \cap attrib(R) \wedge C_2 = attrib(\theta) \cap attrib(S)$

Operaciones

11) Asociatividad de ×: $(R \times S) \times T = R \times (S \times T)$

12) Asociatividad de $\bowtie$: $(R \rhd\lhd S) \rhd\lhd T = R \rhd\lhd (S \rhd\lhd T)$

13) Asociatividad de $\rhd\lhd_\theta$:

$$(R \rhd\lhd_{\theta_{RS}} S) \rhd\lhd_{\theta_{ST} \wedge \theta_{RT}} T = R \rhd\lhd_{\theta_{RS} \wedge \theta_{RT}} (S \rhd\lhd_{\theta_{ST}} T)$$

si $attrib(\theta_{RS}) \subseteq attrib(R) \cup attrib(S) \wedge$

$attrib(\theta_{ST}) \subseteq attrib(S) \cup attrib(T) \wedge$

$attrib(\theta_{RT}) \subseteq attrib(R) \cup attrib(T)$

14) Equivalencia con σ, × y $\bowtie$:

i) $\sigma_\theta(R \times S) = R \rhd\lhd_\theta S$

ii) $\sigma_{\theta 1}(R \rhd\lhd_{\theta 2} S) = R \rhd\lhd_{\theta 1 \wedge \theta 2} S$

Operaciones con conjuntos

2.1) Conmutatividad de $\cup$ y $\cap$: $R \cup S = S \cup R$

$R \cap S = S \cap R$

2.2) Asociatividad de $\cup$ y $\cap$: $(R \cup S) \cup T = R \cup (S \cup T)$

$(R \cap S) \cap T = R \cap (S \cap T)$

2.3) Distributividad de σ con $\cup$: $\sigma_\theta(R \cup S) = (\sigma_\theta(R)) \cup (\sigma_\theta(S))$

2.4) Distributividad de σ con $\cap$ y -: $\sigma_\theta(R \Delta S) = (\sigma_\theta(R)) \Delta S$

2.5) Distributividad de π con $\cup$: $\pi_L(R \cup S) = (\pi_L(R)) \cup (\pi_L(S))$