Pasos del procesamiento de una consulta

Análisis léxico, sintáctico y validación

Análisis léxico

Identificar los componentes (léxicos) en el texto de la consulta (SQL)

El concepto de **léxico** encierra varios significados, todos ligados al mundo de **lingüística**. Léxico es el **vocabulario** de un idioma o de una **región**, el **diccionario** de una **lengua** o el **caudal de modismos y voces de un autor**.

Análisis sintáctico

Revisar la sintaxis de la consulta (corrección gramatical)

Validación semántica

Verificar la validez de los nombres de las tablas, vistas, columnas, etc. y si tienen sentido

Traducción de la consulta a una representación interna que la máquina manipula mejor,
eliminando peculiaridades del lenguaje de alto nivel empleado (SQL),
el formalismo base de la representación interna debe ser rico, para representar toda consulta posible neutral, sin predisponer a ciertas opciones de optimización

La mejor elección: Álgebra Relacional

En un **sistema relacional**

Consultas expresadas con SQL: QUÉ datos y no CÓMO recuperarlos

El **SGBD selecciona la mejor estrategia de ejecución.**

La Optimización "automática" tiene mayor **control sobre** el **rendimiento** del sistema

Procesamiento de Consultas

Actividades involucradas en la recuperación de datos de la BD

Optimización de Consultas

Elección de una estrategia de ejecución eficaz para procesar cada consulta sobre la base de datos.

La **optimización** debe lograr un tiempo de ejecución de consultas aceptable

Una expresión relacional del álgebra relacional permite su optimización antes de la ejecución

Objetivos del procesamiento de consultas

Transformar una consulta SQL en una estrategia de ejecución eficaz, expresada en un lenguaje de bajo nivel y ejecutar dicha estrategia para recuperar los datos requeridos

Existen muchas transformaciones equivalentes para una misma consulta

Objetivo de la optimización de consultas

Elegir la estrategia de ejecución que minimiza el uso de los recursos.

No se garantiza que la estrategia elegida por el SGBD sea la óptima, pero es razonablemente eficiente.

Optimización

El Optimizador de Consultas suele combinar varias técnicas

Las técnicas principales:

- **Optimización heurística**
  - **Ordenar las operaciones de la consulta para incrementar la eficiencia de su ejecución**
- **Estimación de costos**
  - **Estimar sistemáticamente el costo de cada estrategia de ejecución y elegir el plan (estrategia) con el menor costo estimado**

Optimización automática

El usuario **no** se **preocupa** de **cómo** formular la consulta que ejecutara el motor la base de datos.

El Módulo **Optimizador** dispone de **información estadística** en el diccionario de datos del SGBD

esta información le da mayor precisión al estimar la eficiencia de cada posible estrategia y así tiene mayor probabilidad de elegir la más eficiente

Si **cambian las estadísticas** (tras reorganización física del esquema de la BD)

hay que Re-optimizaar (elegir **otra** estrategia)

En un SGBD Relacional: El Optimizador re-procesa la consulta original

El Optimizador es un programa que **considera más estrategias que el programador**

El Optimizador es el **compendio de aptitudes y servicios** de los mejores programadores

Optimización Heurística

Aplicación de reglas de transformación heurísticas para modificar la representación interna de una consulta (Álgebra Relacional o Árbol de consulta) a fin de mejorar su rendimiento.

Varias expresiones del Álgebra Relacional pueden corresponder a la misma consulta.

Lenguajes de consulta, como SQL permiten expresar una misma consulta de muchas formas diferentes, pero el rendimiento no debe depender de cómo sea expresada la consulta.

Optimización Heurística

El Analizador Sintáctico genera un árbol de consulta inicial sin optimización ⇨ ejecución ineficiente.

El Optimizador de Consultas transforma el árbol de consulta inicial en árbol de consulta final equivalente y eficiente.

Aplicación de reglas de transformación guiadas por reglas heurísticas.

Conversión de la consulta en su forma canónica equivalente

Optimización Heurística

Obtenida la forma canónica de la consulta, el Optimizador decide cómo evaluarla.

Estimación sistemática de costos:
Estimación y comparación de los costos de ejecutar una consulta con diferentes estrategias, y elegir la estrategia con menor costo estimado.

El punto de partida es considerar la consulta como una serie de operaciones elementales interdependientes.

Operaciones del Álgebra Relacional:

JOIN, PROYECCIÓN, SELECCIÓN, UNION, INTERSECCIÓN ...

Optimización Heurística

El Optimizador tiene un conjunto de técnicas para realizar cada operación

Por ejemplo: Hay técnicas para implementar la operación de σ

- **Búsqueda Lineal**
- **Búsqueda Binaria**
- **Empleo de Índice Primario o Clave de Dispersión**
- **Empleo de Índice de Agrupamiento**
- **Empleo de Índice Secundario**

Información estadística

Para cada **tabla**

- **Cardinalidad** (nº de filas),
- **Factor de bloques** (nº de filas que caben en un bloque),
- Nº de **bloques ocupados**,
- **Método de acceso primario y otras** estructuras de acceso (hash,índices,etc.),
- **Columnas indexadas**, de **dispersión**, de **ordenamiento**.

Para cada **columna**

- Nº de **valores distintos** almacenados,
- Valores **máximo y mínimo**.

Nº de niveles de cada **índice de múltiples niveles**

El éxito de la estimación del tamaño y del coste de las operaciones incluidas en una consulta, depende de la cantidad y actualidad de la información estadística almacenada en el diccionario de datos del SGBD

Información estadística en Postgresql

```
CREATE TABLE prueba AS SELECT *, 'hola mundo cruel'::text AS name
        FROM generate_series(1, 1000000) AS id;

select * from prueba;

ALTER TABLE prueba ALTER COLUMN id SET STATISTICS 10;

ANALYZE;

explain  SELECT * FROM prueba WHERE id < 150000;

explain analyze SELECT * FROM prueba WHERE id < 150000;

select * from pg_stats where tablename = 'prueba‘

SELECT relpages, current_setting('seq_page_cost') AS seq_page_cost,
relpages * current_setting('seq_page_cost')::decimal AS page_cost,
reltuples, current_setting('cpu_tuple_cost') AS cpu_tuple_cost,
reltuples * current_setting('cpu_tuple_cost')::decimal AS tuple_cost
FROM pg_class WHERE relname=‘prueba';
```

El Optimizador genera **varios planes de ejecución**

Plan de Ejecución = combinación de **técnicas** candidatas.

una **técnica** por cada operación elemental de la consulta

Cada técnica tendrá asociada una **estimación del costo**

Costo(técnica$_i$) = accesos a bloque de disco necesarios

Medida en número de transferencias de bloques memoria a disco

La estimación precisa de costes es difícil, pues para estimar los accesos a bloque es necesario estimar el tamaño de las tablas (base o generadas como resultados intermedios), lo cual depende de los valores actuales de los datos

El Optimizador **elige el plan de ejecución más económico**

función de costo que se debe minimizar

Costo (plan$_p$) $\equiv \Sigma_i$ **costo (técnica$_{ip}$)**

Existen **muchos posibles planes de ejecución** para una consulta

La tarea de **obtener el plan más económico**, **tiene un costo**

Reglas generales de transformación

1. Una secuencia de selecciones sobre una tabla A puede transformarse en una sola selección

$$\sigma_{C1}\left(\sigma_{C2}(A)\right) \equiv \sigma_{C1\ AND\ C2}(A)$$

2. En una secuencia de proyecciones contra una tabla A pueden ignorarse todas, salvo la última (si cada columna mencionado en la última, también aparece en las demás)

$$\pi_{P2}\left(\pi_{P1}(A)\right) \equiv \pi_{P2}(A), \text{ si } P2 \subseteq P1$$

3. Una selección de una proyección puede transformarse en una proyección de una selección

$$\sigma_{C}\left(\pi_{P}(A)\right) \equiv \pi_{P}\left(\sigma_{C}(A)\right)$$

Hacer **selección antes que proyección**, pues la selección reduce el tamaño de la entrada para la proyección.

Distributividad

Sea f un operador unario y ☮ un operador binario,

f es distributivo respecto de ☮ si

F (A ☮ B) = f(A) ☮ f(B)

4. σ es distributivo respecto de la **UNIÓN**, **INTERSECCIÓN** y **DIFERENCIA**

$$\sigma_C(R \Theta S) \equiv (\sigma_C(R)) \Theta (\sigma_C(S))$$

donde $\Theta \in \{ \cup, \cap, - \}$

5. π es distributivo respecto de la **UNIÓN**

$$\pi_P(R \cup S) \equiv (\pi_P(R)) \cup (\pi_P(S))$$

6. σ es distributivo respecto de REUNIÓN (**JOIN**) si la condición de selección contiene columnas que sólo pertenecen a una tabla

$$\sigma_{nump=2}(\text{EMPLEADO } \mathbf{X}_{nss=nsse} \text{TRABAJA\_EN}) \equiv \text{EMPLEADO } \mathbf{X}_{nss=nsse} \left(\sigma_{nump=2}(\text{TRABAJA\_EN})\right)$$

O puede escribirse como (c1 AND c2), y en c1 sólo intervienen columnas de R1 y en c2 sólo hay columnas de R2

$$\sigma_c(R1 \ X_J R2) \equiv (\sigma_{c1}(R1)) \ X_J (\sigma_{c2}(R2))$$

7. π es distributivo respecto de **JOIN** si en la condición de reunión **J** sólo intervienen columnas incluidos en la lista de proyección **P**

$$\pi_P\left(R1 \ X_J R2\right) \equiv \left(\pi_{P\_R1}(R1)\right) \ X_J \left(\pi_{P\_R2}(R2)\right)$$

si **P = (P_R1** $\cup$ **P_R2)** y **P** incluye toda columna de reunión que aparece en **J**

8. En Álgebra Relacional, son conmutativas: **UNIÓN, INTERSECCIÓN** y **JOIN**
y no conmutativas: **DIFERENCIA** y **DIVISIÓN**

Conmutatividad Sea ☮ un operador binario, ☮ es conmutativo si A ☮ B = B ☮ A , ∀ A,B

9. En Álgebra Relacional, son asociativas: **UNIÓN, INTERSECCIÓN** y **JOIN**
y no asociativas: **DIFERENCIA** y **DIVISIÓN**

Asociatividad Sea ☮ un operador binario, ☮ es asociativo si A ☮ (B☮C)= (A☮B) ☮ C , ∀ A,B,C

10. En Álgebra relacional, son idempotentes : **UNIÓN, INTERSECCIÓN** y **JOIN**
y no idempotentes: **DIFERENCIA** y **DIVISIÓN**

Idempotencia Sea ☮ un operador binario, ☮ es idempotente si A ☮ A = A, ∀ A

Ejemplo si un elemento al multiplicarse por sí mismo sucesivas veces da él mismo, este elemento es **idempotente**.

Sean **a y b columnas de dos relaciones distintas**, R(…a…) y S(…b…)

$$R \bowtie_{(a>b \text{ AND } b>3)} S$$

la condición **(a>b AND b>3)** equivale a **(a>b) AND (b>3) AND (a>3)**

por ser > un operador transitivo la nueva condición **(a>3)**, permite realizar una **restricción** (sobre **R**) **antes del JOIN** entre R y S (join necesario para evaluar (a>b))

$$(\sigma_{a>3}(R)) \bowtie_{a>b} (\sigma_{b>3}(S))$$

Sean **las columnas a, b, c, d, e, f**

La condición (**a>b OR (c=d AND e<f)**) equivale a (**a>b OR c=d**) **AND** (**a>b OR e<f**)

puesto que OR es distributivo respecto de AND

Forma normal conjuntiva

Una expresión en FNC tiene la forma C_1 **AND** C_2 **AND ...** C_n

donde cada C_i **no** incluye ningún AND y todo es TRUE si todo C_i es TRUE**,**

y **es FALSE si algún** C_i **es FALSE**

Ventajas de la FNC

Ya que AND es conmutativo, el Optimizador puede **evaluar cada** C_i **en cualquier orden** (por ejemplo, en orden creciente en dificultad)**. En cuanto un** C_i **de FALSE, el proceso puede acabar**

En un **entorno de procesamiento paralelo**, sería posible **evaluar todos los** C_i **a la vez**. En cuanto uno diera FALSE, el proceso acabaría

REGLAS HEURÍSTICAS

Algunas **reglas heurísticas** que pueden ser aplicadas durante el procesamiento de consultas

1. Ejecutar **operaciones de restricción σ tan pronto como sea posible**

2. Ejecutar **primero las restricciones σ más restrictivas**
(las que producen menor nº de filas)

3. Combinar un **producto cartesiano** × con una **restricción** σ subsiguiente cuya condición represente una condición de reunión, **convirtiéndolas en un join**

4. Ejecutar las operaciones de **proyección** π **tan pronto como sea posible**

Traducción de consultas al álgebra relacional

Cada consulta SQL se divide en **bloques de consulta**:

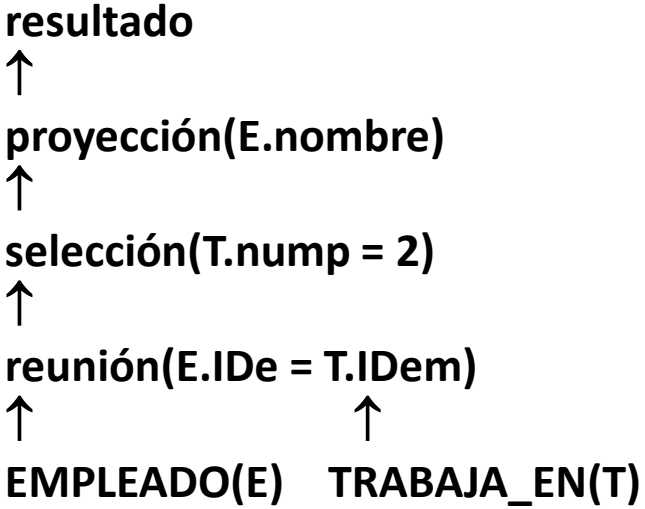
Unidades de traducción al álgebra relacional

Nombres de los empleados que trabajan en el proyecto 2
EMPLEADO(IDe,NOMBRE) y TRABAJA_EN(IDem,NUMP)

SELECT nombre FROM Empleado E, Trabaja_en T WHERE E.IDe = T. IDemAND T.nump=2 ;

$$\pi_{nombre}(\sigma_{nump=2}(EMPLEADO \ |X|_{IDe=IDem} TRABAJA\_EN))$$

Árbol de Consulta (o árbol sintáctico abstracto) es la representación de una expresión del algebra relacional



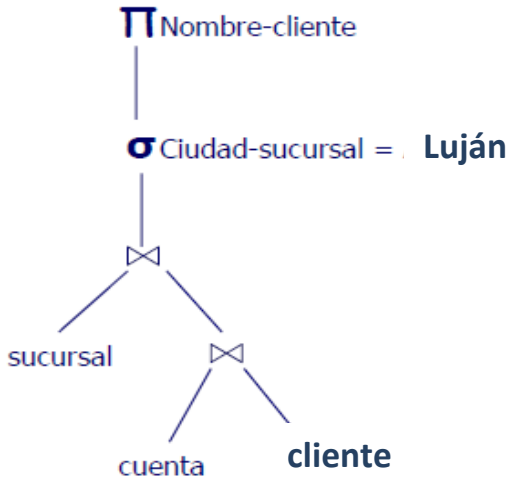
Considérese la siguiente expresión de álgebra relacional: Hallar los nombres de todos los clientes que tengan una cuenta ubicada en cualquier sucursal en Luján.

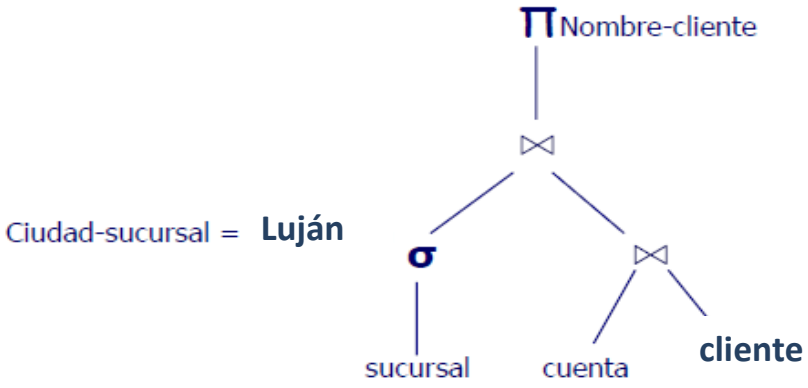
SUCURSAL(ID-SUCURSAL, CIUDAD-SUCURSAL)
CUENTA(DNI-CLIENTE,ID-SUCURSAL,SALDO)
CLIENTE(DNI-CLIENTE,NOMBRE-CLIENTE)

Π nombre-cliente (**σ** ciudad-sucursal= 'Luján' (sucursal |X| (cuenta |X| cliente)))

La consulta optimizada queda ahora de la siguiente manera:

Π nombre-cliente ((**σ** ciudad-sucursal = 'Luján' (sucursal)) |X| (cuenta |X| cliente))





Mostrar los empleados cuyo salario sea mayor al salario máximo del departamento 5.

EMPLEADO(ID,APELLIDO,NOMBRE,SALARIO,NUMD)

SELECT APELLIDO, NOMBRE **FROM** EMPLEADO
WHERE SALARIO > (**SELECT** MAX(SALARIO) **FROM** EMPLEADO WHERE NUMD = 5);

C = (π MAX (SALARIO)((σ NUMD=5(EMPLEADO))))

Π APELLIDO, NOMBRE(σ SALARIO **>** **C**)

Se evalúa sólo una vez, el resultado se trata como una constante **(>C)**

Optimización sintáctica

Objetivo
Reducir el tamaño de las tablas intermedias

Regla
Modifican la ***representación interna*** de la consulta

Regla principal:
Primero ejecutar seleccionar (σ**)** y proyectar (π**)**
Al final ejecutar reunir (|X|, X) y otras operaciones binarias

Árbol de consulta inicial o canónico

Lo genera el analizador sintáctico de manera estándar sin optimizar.
Primero los X , luego condiciones de σ y |X|. Por último π.
Muy ineficiente debido a los X

PROYECTO(NUMEROP, NUMD, LOCALIZACION)
EMPLEADO(NSS, APELLIDO, DIRECCIÓN, FECHA_NCTO)
DEPARTAMENTO(NUMEROD , NSS_JEFE)

SELECT P.NUMEROP, P.NUMD, E.APELLIDO, E.DIRECCION, E.FECHA_NCTO
FROM PROYECTO P, DEPARTAMENTO D, EMPLEADO E
WHERE P.NUMD=D.NUMEROD **AND** D.NSS_JEFE=E.NSS
AND P.LOCALIZACION='Luján'

(π P.NUMEROP, P.NUMD, E.APELLIDO, E.DIRECCION, E.FECHA_NCTO

(σ D.NSS_JEFE=E.NSS Y P.NUMD=D.NUMEROD Y P.LOCALIZACIONP = 'Luján'

(EMPLEADO X (PROYECTO X DEPARTAMENTO)))

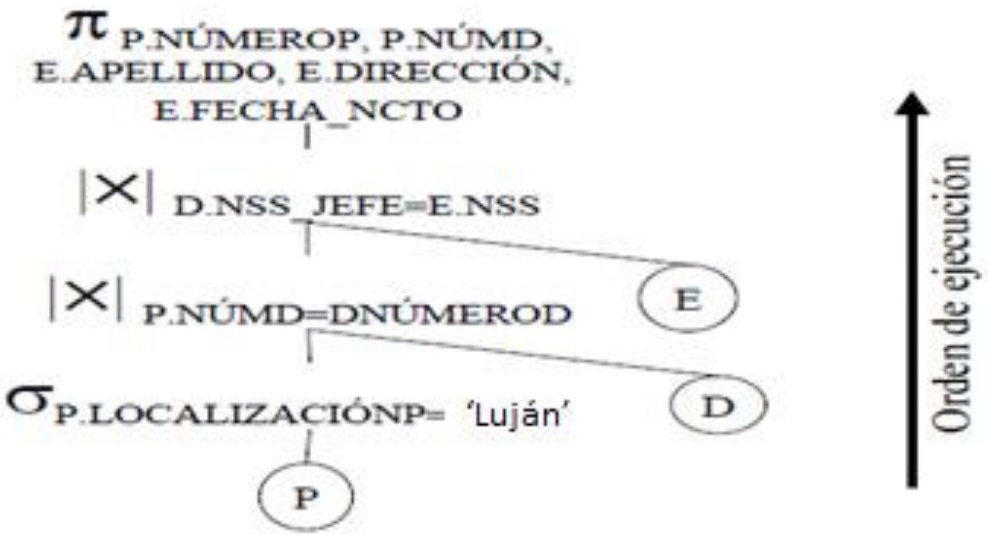
Árbol de consulta

Representa una expresión del álgebra relacional

Una consulta se puede representar por varios árboles distintos

PROYECTO(NUMEROP, NUMD, LOCALIZACION)
EMPLEADO(NSS, APELLIDO, DIRECCIÓN, FECHA_NCTO)
DEPARTAMENTO(NUMEROD , NSS_JEFE)

SELECT P.NUMEROP, P.NUMD ,E.APELLIDO, E.DIRECCION, E.FECHA_NCTO
FROM PROYECTO P, DEPARTAMENTO D, EMPLEADO E
WHERE P.NUMEROD = D.NUMEROD **AND** D.NSS_JEFE = E.NSS **AND**
P.LOCALIZACIÓN = 'Luján'



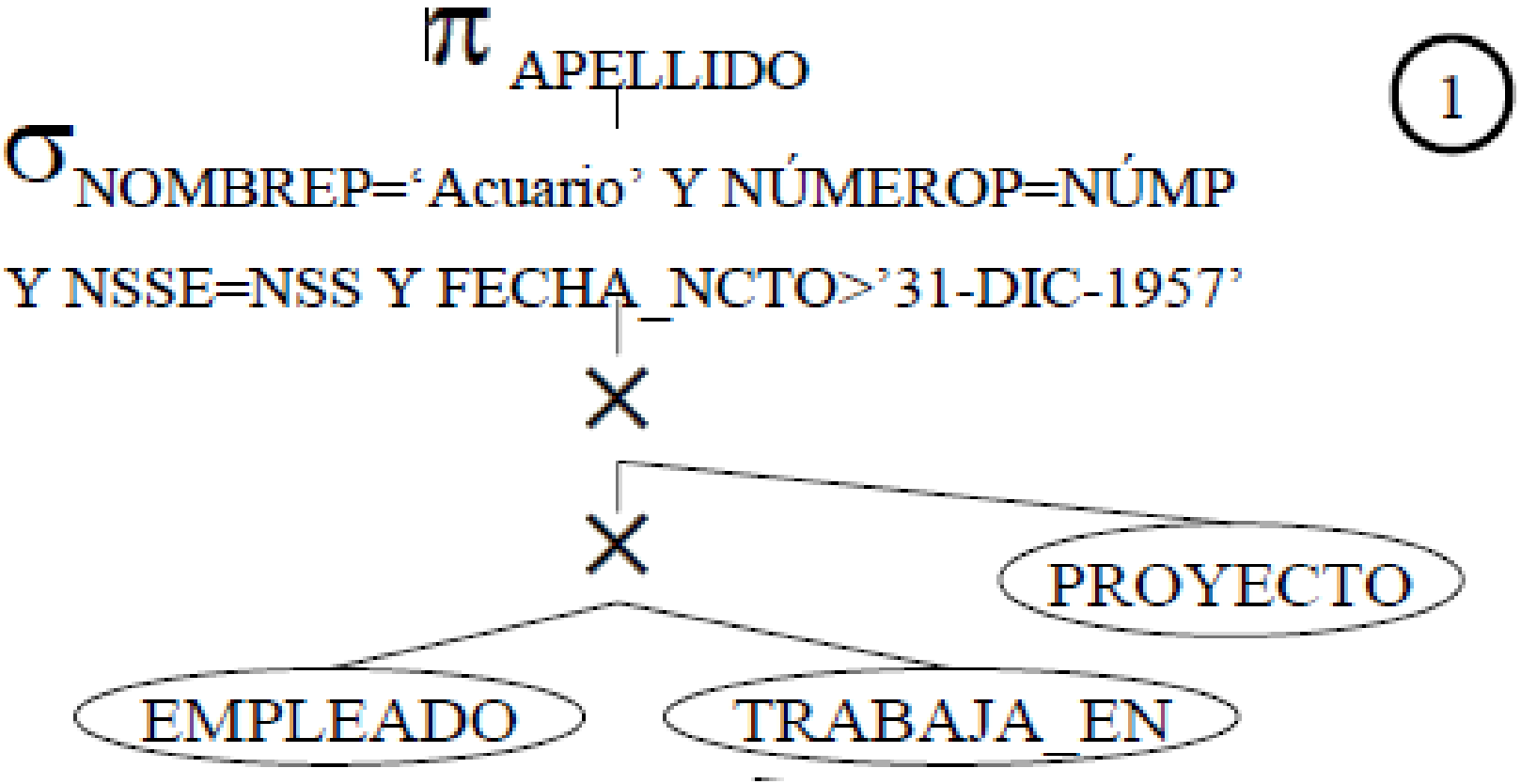
Optimización con árboles: Ejemplo

Transformar el árbol en uno de ejecución eficiente
Basada en **reglas de equivalencia** entre expresiones del álgebra relacional.

Ejemplo: Apellido de los empleados que trabajan en el proyecto Acuario nacidos después de 1957.

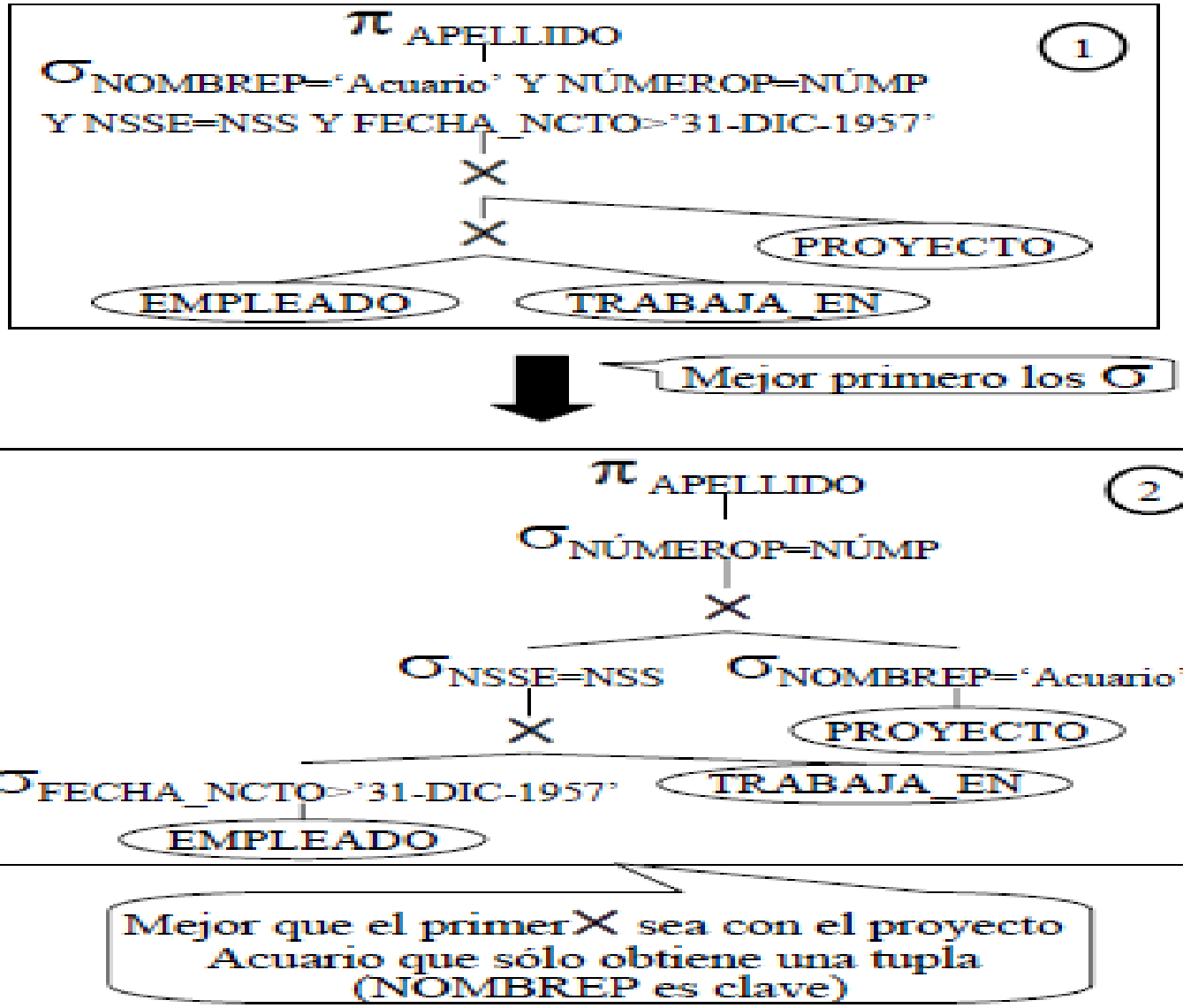
TRABAJA_EN (NSSE,NUMP)
PROYECTO(NUMEROP, NOMBREP,LOCALIZACION)
EMPLEADO(NSS,APELLIDO,DIRECCION,FECHA_NCTO)

SELECT APELLIDO **FROM** EMPLEADO , TRABAJA_EN , PROYECTO
WHERE NOMBREP = 'Acuario' **AND** NUMEROP = NUMP **AND** NSSE = NSS **AND** FECHA_NCTO > '31-DIC-1957'

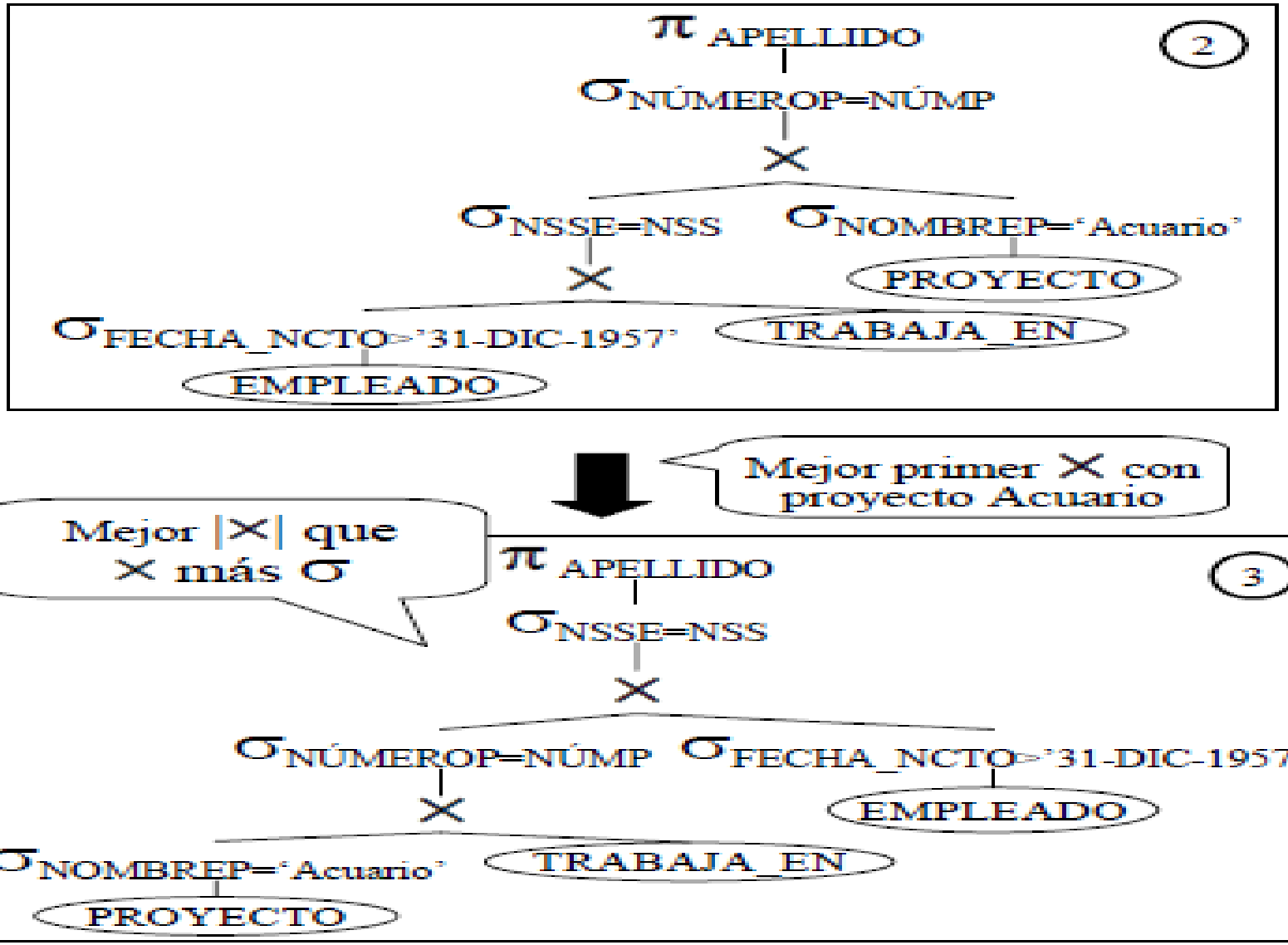


Los X crean un fichero muy grande sólo se necesita usar las tuplas del proyecto Acuario y reunir los empleados nacidos después de 1957
Es más óptimo primero hacer la selección σ

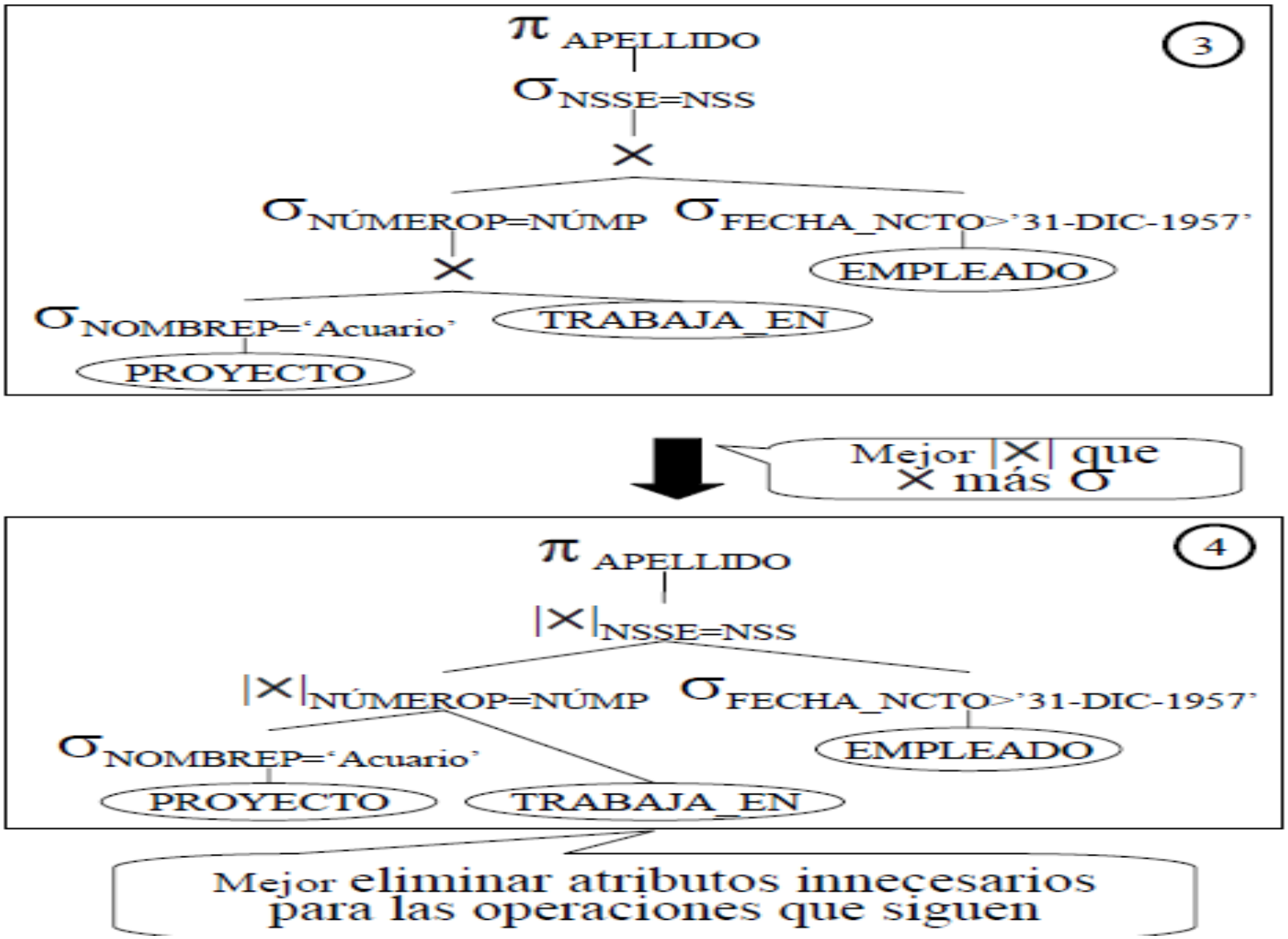
Optimización con árboles: ejemplo (2)



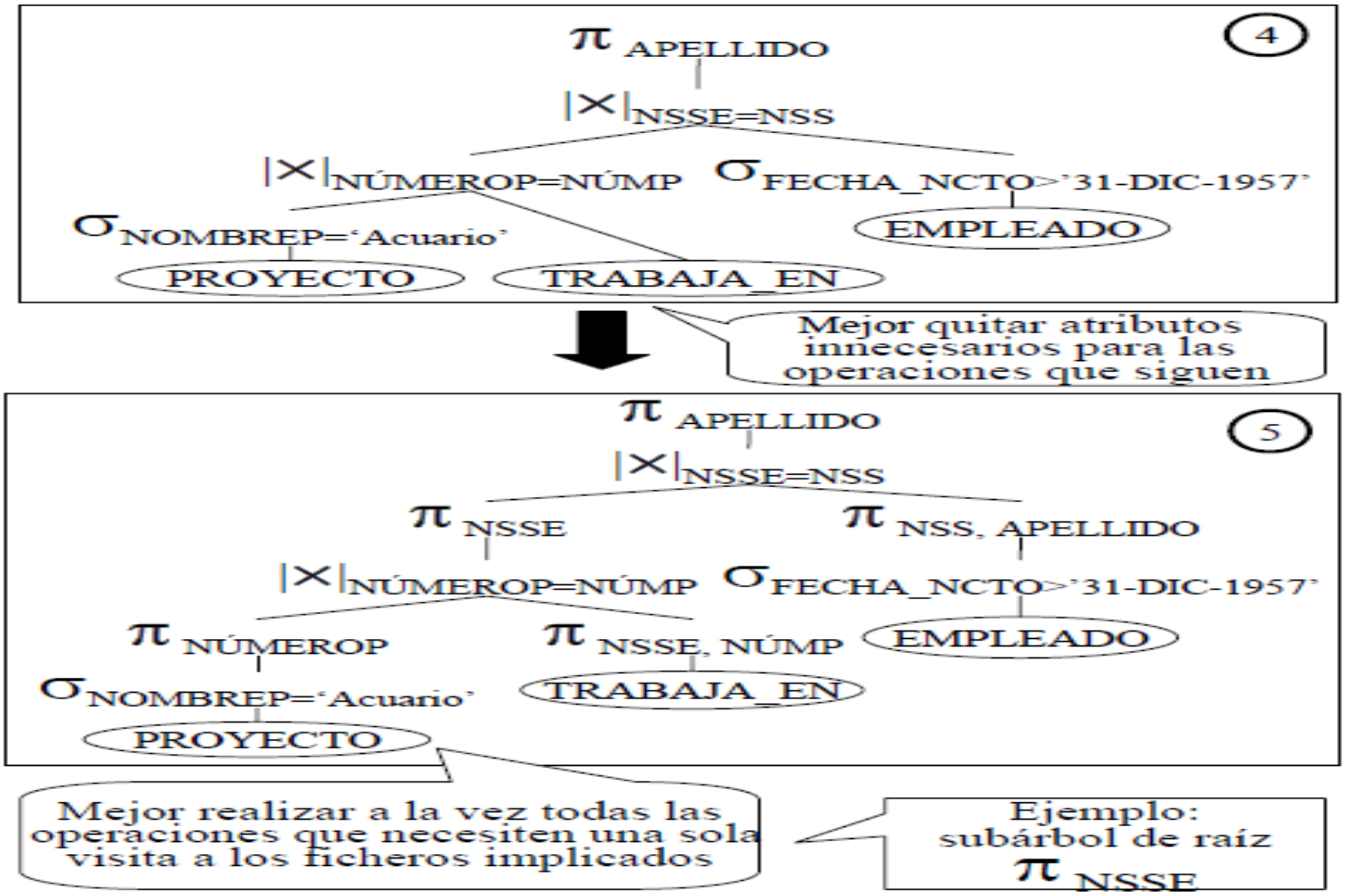
Optimización con árboles: ejemplo (3)



Optimización con árboles: ejemplo (4)



Optimización con árboles: ejemplo (5)



Ejercicio: optimización heurística

Escribir los árboles canónicos y optimizados correspondientes a las siguientes consultas:
Probar el costo usando explain analyze de postgresql

a)

```
SELECT NUMEROP, NUMD, APELLIDO, DIRECCION, FECHA_NCTO
    FROM PROYECTO, DEPARTAMENTO, EMPLEADO
        WHERE NSS_JEFE = NSS AND NÚMD = NÚMEROD AND LOCALIZACIONP = 'Luján'
```

b)

```
SELECT NOMBRED, APELLIDO, NOMBRE, NOMBREP
    FROM DEPARTAMENTO, EMPLEADO, TRABAJA_EN, PROYECTO
        WHERE NUMEROD = ND AND NSS = NSSE AND NP = NUMEROP
```

c)

```
SELECT NOMBRE, APELLIDO, 1.1 * SALARIO
        FROM EMPLEADO, TRABAJA_EN, PROYECTO
            WHERE NSS = NSSE AND NP = NÚMEROP AND NOMBREP = 'ProyectoX'
```