FACULTAD DE CIENCIAS EXACTAS
UNIVERSIDAD NACIONAL DEL CENTRO DE LA PROVINCIA DE BUENOS AIRES

Tecnicatura Universitaria en Desarrollo de Aplicaciones Informáticas (TUDAI)

Base de Datos

Tema 8: Optimización de consultas

2025

Optimización

La tarea de optimización en bases de datos se puede atacar desde dos perspectivas:

- Analizando las consultas, vistas, triggers, funciones, etc.
-
- Mirar el procesamiento y optimización de consultas

PROCESAMIENTO Y OPTIMIZACIÓN DE CONSULTAS

Comprender las **tareas de procesamiento y optimización de** consultas realizadas por un SGBD

Conocer las **reglas heurísticas y de transformación** de expresiones del álgebra relacional y cómo aplicarlas para mejorar la eficiencia de una consulta

Conocer las diferentes **estrategias de implementación** de **operaciones relacionales**, en particular la de ensamble, y cómo evaluar el costo estimado de cada estrategia

Identificar la **información estadística** de la base utilizada para estimar el costo de ejecución de las operaciones del álgebra relacional

PROCESAMIENTO Y OPTIMIZACIÓN DE CONSULTAS

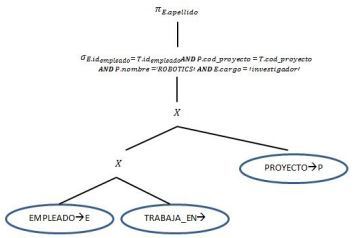
Consulta en un lenguaje de algo nivel → **SELECT…**

↓

Análisis léxico, análisis sintáctico y validación → **Compilador**

↓

Construcción inmediata de la consulta

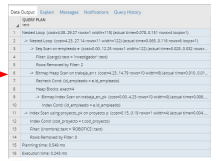


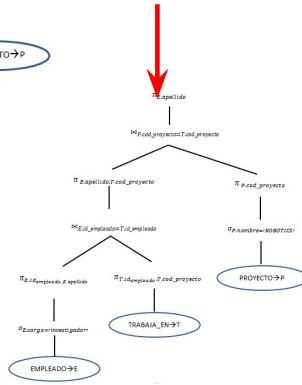
↓

Optimizador de la consulta

↓

Plan de ejecución





↓

Generador de código de la consulta

↓

Código para ejecutar la consulta → **Modo compilado o interpretado**

↓

Procesador de base de datos en tiempo de ejecución → **Tiempo de ejecución**

↓

Resultado de la consulta → **Si hay error de ejecución presenta el mensaje correspondiente**

Elmasri, R.; Navathe, S.B.: **Fundamentos de Sistemas de Bases de Datos**. 5ª Edición. Addison-Wesley. **(Cap. 15)**

PROCESAMIENTO Y OPTIMIZACIÓN DE CONSULTAS

1- Objetivos del procesamiento de consultas

- Transformar una consulta SQL en una **estrategia de ejecución eficaz**, expresada en un lenguaje de bajo nivel.
- Ejecutar dicha estrategia para recuperar los datos requeridos.

2- Objetivo de la optimización de consultas

- Elegir la **estrategia de ejecución** que **minimiza el uso de los recursos**

– Existen muchas transformaciones equivalentes para una misma consulta.

– En general, **no se garantiza que la estrategia elegida** por el SGBD **sea** la **óptima**, pero seguro que será una **estrategia razonablemente eficiente**

PROCESAMIENTO Y OPTIMIZACIÓN DE CONSULTAS

Ventajas de la **optimización automática**

El usuario no se preocupa de **cómo** formular la consulta. El Módulo **Optimizador trabaja** "**mejor**" (a veces!!!) que el DBA o programador, porque:

Dispone de **información estadística** en el diccionario de datos del SGBD. Entonces, puede estimar mejor la eficiencia de cada posible estrategia... y así (con mayor probabilidad) elegirá la más eficiente

Si **cambian las estadísticas** (por ej. si se reorganiza el esquema de BD). Entonces, **re-optimización** (quizá ahora convenga elegir **otra** estrategia)

Por lo general el Optimizador por ser un módulo del SGBD puede **considerar más estrategias** que un DBA manualmente

ANÁLISIS LÉXICO, SINTÁCTICO Y VALIDACIÓN

Análisis léxico: **Identificar** los **componentes** (léxicos) en el texto de la consulta (SQL)

Análisis sintáctico: **Revisar** la **sintaxis** de la consulta (corrección gramática)

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 eliminando peculiaridades del lenguaje de alto nivel (SQL), la mejor elección: Álgebra Relacional

TRADUCCIÓN

Nombres de los voluntarios que realizan la tarea cuyo nombre es PROMOCION

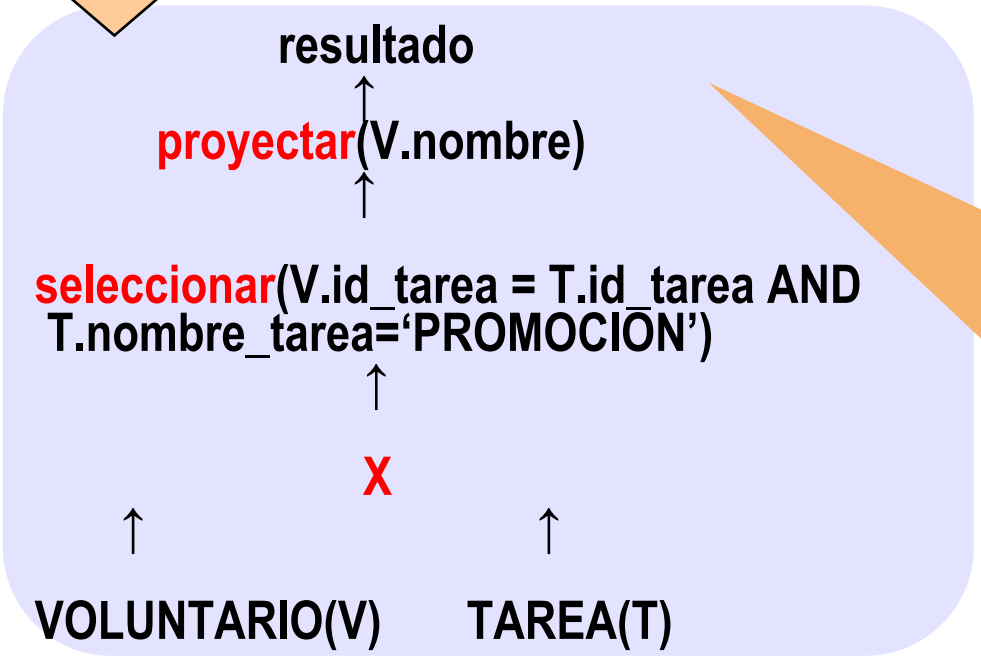
```
SELECT nombre
FROM voluntario V, tarea T
WHERE V.id_tarea = T.id_tarea
    AND T.nombre_tarea='PROMOCION';
```

$$\pi_{nombre}(\sigma_{V.id\_tarea = T.id\_tarea \text{ AND } nombre\_tarea='PROMOCION'}(VOLUNTARIO \times TAREA))$$

Traducción de la consulta SQL a álgebra relacional

TRADUCCIÓN (CONT.)

$$\pi_{\text{nombre}}(\sigma_{\text{V.id\_tarea = T.id\_tarea AND nombre\_tarea='PROMOCION'}}(\text{VOLUNTARIO(V)} \times \text{TAREA(T)}))$$



Árbol de Consulta

(o árbol sintáctico abstracto)

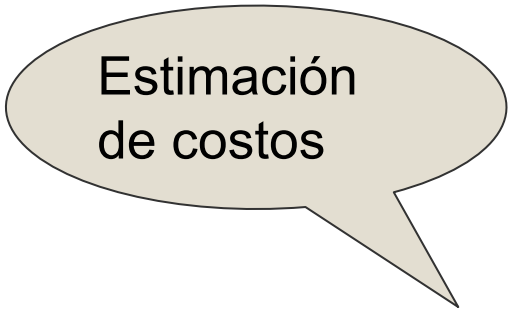
es la **representación de una expresión algebraica**

Cada nodo hoja es un **nodo de exploración de la tabla**

OPTIMIZACIÓN

El Optimizador de Consultas suele combinar varias técnicas de optimización

Ordenar los accesos a las tablas de la consulta para incrementar la eficiencia de su ejecución



Estimar el costo de cada estrategia de ejecución y se va a quedar con el plan (estrategia) que tenga el menor costo estimado

OPTIMIZACIÓN

El Analizador Sintáctico genera árbol de consulta inicial **sin optimización**

El Optimizador de Consultas transforma el árbol de consulta inicial en un árbol de consulta equivalente y eficiente, **aplicando las reglas heurística de transformación**

El Optimizador de Consultas convierte esa consulta en su forma canónica equivalente en lenguaje interno del SGBD

OPTIMIZACIÓN

- 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

OPTIMIZACIÓN

Método heurístico de optimización de consultas utiliza **reglas de transformación** para convertir una expresión de álgebra relacional en otra forma equivalente que sepa que es más eficiente

Cómo construye el árbol canónico? Los pasos son

1. Producto cartesiano (X) a relaciones del FROM
2. Condiciones de selección y ensamble de la cláusula WHERE
3. Proyección sobre los atributos de la cláusula SELECT.

OPTIMIZACIÓN

Nombres de los voluntarios que mayores de 18 años y que realizan la tarea de PROMOCION

```
SELECT nombre
FROM voluntario V, tarea T
WHERE V.id_tarea = T.id_tarea                                  C1
AND T.nombre_tarea='PROMOCION'                                 C2
AND date_part('year',age(V.fecha_nacimiento)) > 18;            C3
```

Árbol canónico desarrollado en clase

$$\pi_{nombre}(\sigma_{C1\ AND\ C2\ AND\ C3}(V \times T))$$

$$\pi_{nombre}(\sigma_{C2\ AND\ C3}(V \bowtie T))$$

$$\pi_{nombre}(\ ((\sigma_{C3}(V)) \bowtie (\sigma_{C2}(T))\ ).....\ \text{Otros!!}$$

Reglas de Equivalencia

1. **Selección Conjuntiva:** una secuencia de selecciones sobre una relación A puede transformarse en una sola selección

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

2. **Selección conmutativa:**

$$\sigma_{C1}(\sigma_{C2}(A)) \equiv \sigma_{C2}(\sigma_{C1}(A))$$

3. En una **secuencia de proyecciones** sobre una relación A pueden ignorarse todas, salvo la última (si cada columna mencionado en la última, también aparece en las demás)

$$\Pi_{L2}(\Pi_{L1}(A)) \equiv \Pi_{L2}(A), \text{ sii } L2 \subseteq L1$$

4. La **selección** puede combinarse con productos cartesianos y ensambles generales

$$\sigma_{C1}(A \times B) \equiv (A \bowtie_{C1} B) \; ; \; \sigma_{C1}(A \bowtie_{C2} B) \equiv A \bowtie_{C1 \text{ AND } C2} B$$

5. Los ensambles naturales y generales son conmutativos

$$A \bowtie_{C} B \equiv B \bowtie_{C} A$$

6. a) Los ensambles naturales y b) generales son asociativos

a) $(A \bowtie B) \bowtie C \equiv A \bowtie (B \bowtie C)$

b) $(A \bowtie_{C1} B) \bowtie_{C2 \text{ AND } C3} C \equiv A \bowtie_{C1 \text{ AND } C3} (B \bowtie_{C2} C)$

c2 involucra atributos sólo de B y C

Reglas de Equivalencia

7. La selección es distributiva respecto de ensambles si la condición del σ

- C1 involucra sólo atributos de una de las expresiones (A)

$$\sigma_{C1}(A \bowtie_C B) \equiv (\sigma_{C1}(A)) \bowtie_C B$$

- C1 involucra sólo atributos de A y C2 sólo de B

$$\sigma_{C1 \text{ AND } C2}(A \bowtie_C B) \equiv (\sigma_{C1}(A)) \bowtie_C (\sigma_{C2}(B))$$

se **reduce el número de filas examinadas en la siguiente operación** en secuencia: join (así, esa operación será más rápida y también producirá menos filas)

8. La proyección(π) es distributiva para el ensamble general si

- si involucra solo atributos de L1 ∪ L2 (L1 y L2 lista de atributos de A y B, respect., con L1 ∩ L2 ≠ ∅

$$\pi_{L1 \cup L2}(A \bowtie_C B) \equiv (\pi_{L1}(A)) \bowtie_C (\pi_{L2}(B))$$

- L1 y L3 atributos de A, L2 y L4 atributos de B; L3 y L4 involucrados en C, pero no en L1 ∪ L2

$$\pi_{L1 \cup L2}(A \bowtie_C B) \equiv$$

$$(\pi_{L1 \cup L2}((\pi_{L1 \cup L3}(A)) \bowtie_C (\pi_{L2 \cup L4}B))$$

se **reduce el nro. de columnas que se involucran en la siguiente operación** en secuencia: join (por tanto, esa operación necesitará menos tiempo para su ejecución y producirá menos columnas)

Reglas de Equivalencia

9. **Unión e Intersección son conmutativas**
(la Diferencia y División no)

$$A \cup B \equiv B \cup A$$
$$A \cap B \equiv B \cap A$$

10. **Unión e Intersección son asociativas**
(la Diferencia y División no)

$$(A \cup B) \cup C \equiv A \cup (B \cup C)$$
$$(A \cap B) \cap C \equiv A \cap (B \cap C)$$

11. La **Proyección** es distributiva respecto de la **Unión**

$$\pi_p(A \cup B) \equiv (\pi_p(A)) \cup (\pi_p(B))$$

12. La **Selección** es distributiva respecto de la **Unión, Intersección y Diferencia**

$$\sigma_c(A \Theta B) \equiv (\sigma_c(A)) \Theta (\sigma_c(B))$$
$$\Theta \in \{ \cup, \cap, - \}$$

se **reduce el número de filas examinadas en la siguiente operación** en secuencia: Θ (así, esa operación será más rápida y también producirá menos filas)

Reglas de Equivalencia

Algunas **buenas 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 nro. de filas)
3. **Combinar** un **producto cartesiano** × con una σ subsiguiente cuya condición represente una condición de reunión, **convirtiéndolas en un ensamble** ⋈
4. **Ejecutar** las operaciones de π **tan pronto como sea posible**

Implementación de op. relacionales

Sin índices: Los algoritmos que 'barren' tablas (archivos) recuperando registros que cumplen con la condición de la selección

Algoritmos básicos:

- **Búsqueda lineal:** Se examinan todos los bloques y se toman los que cumplan la condición
- **Búsqueda binaria:** Si el archivo está ordenado en función del atributo del que se está realizando la selección

Con índices:

- **Igualdad basada en clave:** Se usará el índice primario para recuperar el único registro
- **Igualdad basada en atributo no clave:** Se usará el índice de agrupamiento para recuperar varios registros
- **Otros casos de igualdad**: Utilizar índices secundarios para recuperar un registro si es único o varios si no lo es

Plan de ejecución de la consulta

Algunos SGBD permiten a los usuarios inspeccionar la estrategia del optimizador para ejecutar determinada consulta

Algunos SGBD permiten "fijar" el plan de consulta. Postgresql NO

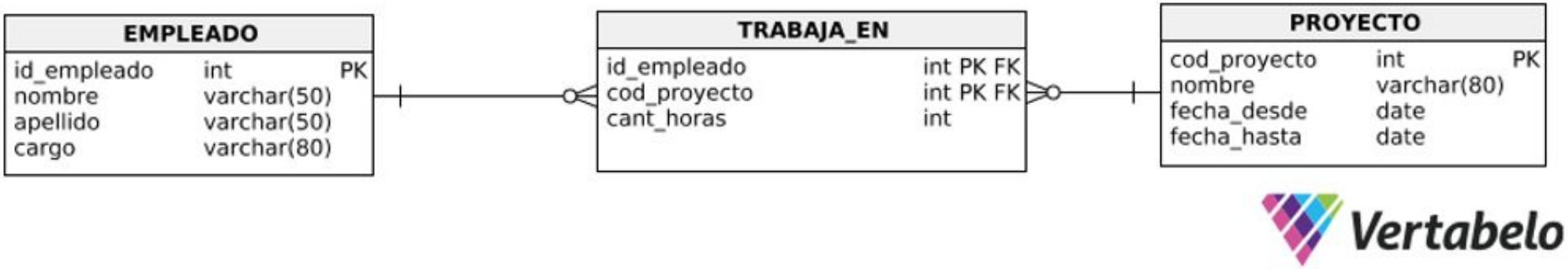
Query execution plan:

En Postgresql sentencia **EXPLAIN**

EXPLAIN SELECT ….

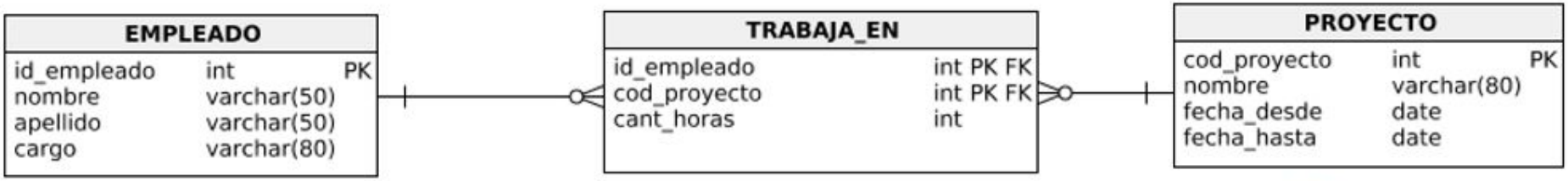
Ejercicio

- ✓ Considere el siguiente esquema de base de datos de empleados de un grupo de investigación en particular:



Ejercicio

- ✓ A su vez, se tienen la siguiente consulta que responde a **seleccionar los apellidos de los empleados que trabajan en el proyecto ROBOTICS y que tienen un cargo de investigador:**

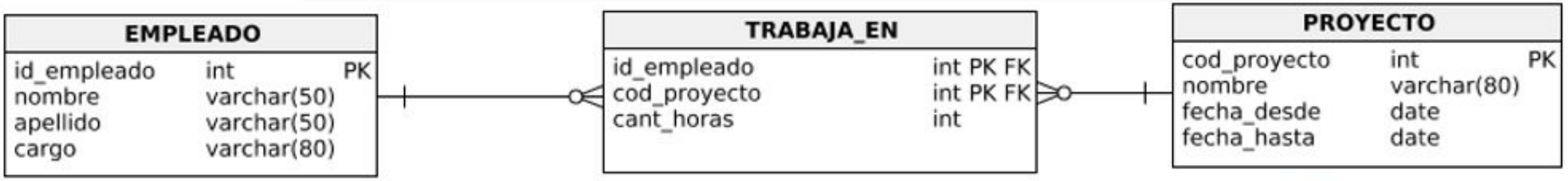


Vertabelo

Ejercicio

- ✓ A su vez, se tienen la siguiente consulta que responde a **seleccionar los apellidos de los empleados que trabajan en el proyecto ROBOTICS y que tienen un cargo de investigador:**

```
SELECT E.apellido
FROM    EMPLEADO E      ,
        TRABAJA_EN T    ,
        PROYECTO P
WHERE E.id_empleado = T.id_empleado
AND P.cod_proyecto = T.cod_proyecto
AND P.nombre = 'ROBOTICS'
AND E.cargo = 'investigador';
```



Vertabelo

Ejercicio

Se solicita:

1. Escriba la consulta en álgebra relacional.

1. Genere de árbol de consulta inicial o canónico.

1. Utilizando las reglas heurísticas y de equivalencia, convertir el árbol anterior en otro equivalente que sepa que es más eficiente. **Nota: Aplique una regla de equivalencia por vez, indique qué regla aplicó en cada caso.**

1. Con el comando EXPLAIN ANALYSE explique la heurística que utiliza para el query plan el optimizador de PostgreSQL en cada una de las sentencias anteriores.

Ejercicio

Se solicita:

1. Escriba la consulta en álgebra relacional.

1. Genere de árbol de consulta inicial o canónico.

1. Utilizando las reglas heurísticas y de equivalencia, convertir el árbol anterior en otro equivalente que sepa que es más eficiente. **Nota: Aplique una regla de equivalencia por vez, indique qué regla aplicó en cada caso.**

1. Con el comando EXPLAIN ANALYSE explique la heurística que utiliza para el query plan el optimizador de PostgreSQL en cada una de las sentencias anteriores.

Ejercicio – Ítem 1

1. Escriba la consulta en álgebra relacional:

```
SELECT E.apellido
FROM    EMPLEADO E      ,
        TRABAJA_EN T    ,
        PROYECTO P
WHERE E.id_empleado = T.id_empleado
AND P.cod_proyecto = T.cod_proyecto
AND P.nombre = 'ROBOTICS'
AND E.cargo = 'investigador';
```

Ejercicio – Ítem 1

1. Escriba la consulta en álgebra relacional:

```
SELECT E.apellido
FROM    EMPLEADO E      ,
        TRABAJA_EN T    ,
        PROYECTO P
WHERE E.id_empleado = T.id_empleado
AND P.cod_proyecto = T.cod_proyecto
AND P.nombre = 'ROBOTICS'
AND E.cargo = 'investigador';
```

$$\pi_{E.apellido}(\sigma_{C_1 \wedge C_2 \wedge C_3 \wedge C_4}(EMPLEADO \rightarrow E \; \boldsymbol{X} \; TRABAJA\_EN \rightarrow T \; \boldsymbol{X} \; PROYECTO \rightarrow P))$$

$$\boldsymbol{C_1} \rightarrow E.id\_empleado = T.id\_empleado$$

$$\boldsymbol{C_2} \rightarrow P.cod\_proyecto = T.cod\_proyecto$$

$$\boldsymbol{C_3} \rightarrow P.nombre = 'ROBOTICS'$$

$$\boldsymbol{C_4} \rightarrow E.cargo = 'investigador'$$

Ejercicio

Se solicita:

1. Escriba la consulta en álgebra relacional.

1. Genere de árbol de consulta inicial o canónico.

1. Utilizando las reglas heurísticas y de equivalencia, convertir el árbol anterior en otro equivalente que sepa que es más eficiente. **Nota: Aplique una regla de equivalencia por vez, indique qué regla aplicó en cada caso.**

1. Con el comando EXPLAIN ANALYSE explique la heurística que utiliza para el query plan el optimizador de PostgreSQL en cada una de las sentencias anteriores.

Ejercicio

Pasos de construcción del árbol de consulta inicial o canónico

1. Se crea un nodo hoja para cada una de las tablas de la consulta.

1. Se crea un nodo no hoja para cada relación intermedia producida por una operación de álgebra relacional.

1. La raíz del árbol representa el resultado de la consulta.

1. La secuencia de operaciones se dirige desde las hojas a la raíz.

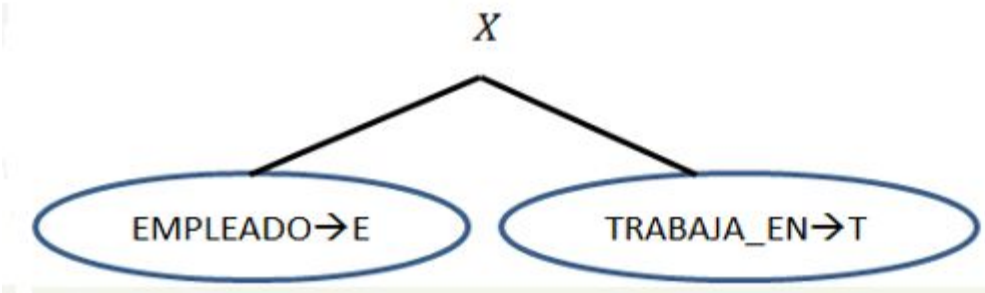
Connolly et al. **Database Systems: A Practical Approach to Design, Implementation and Management**. 2nd Ed. Addison-Wesley **(Cap. 18)**

Ejercicio – Ítem 2

```
SELECT E.apellido
FROM    EMPLEADO E      ,
        TRABAJA_EN T    ,
        PROYECTO P
WHERE E.id_empleado = T.id_empleado
AND P.cod_proyecto = T.cod_proyecto
AND P.nombre = 'ROBOTICS'
AND E.cargo = 'investigador';
```

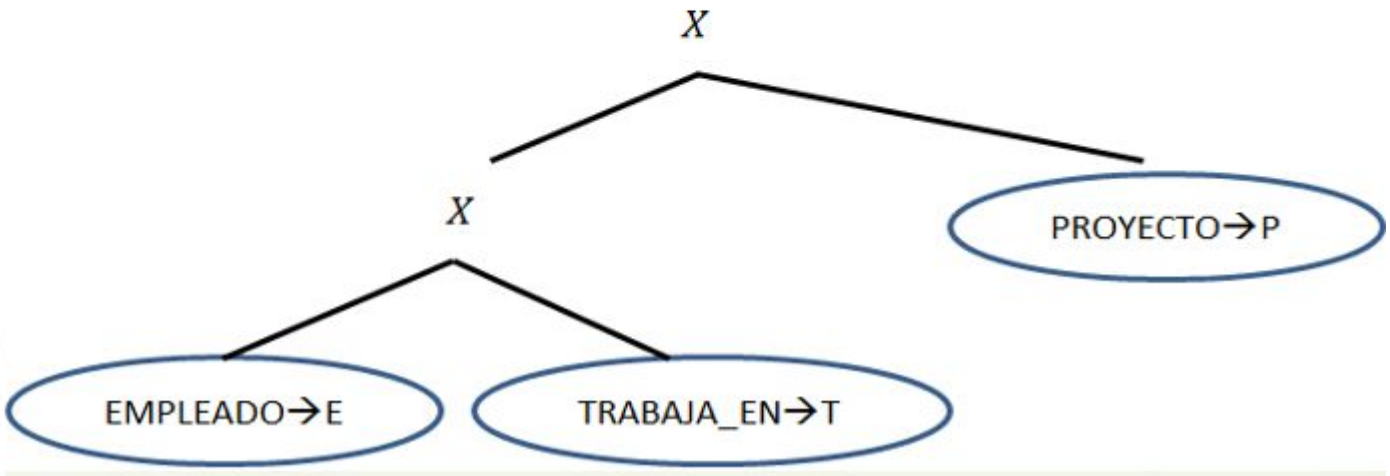
Ejercicio – Ítem 2

```
SELECT E.apellido
FROM    EMPLEADO E      ,
        TRABAJA_EN T    ,
        PROYECTO P
WHERE E.id_empleado = T.id_empleado
AND P.cod_proyecto = T.cod_proyecto
AND P.nombre = 'ROBOTICS'
AND E.cargo = 'investigador';
```



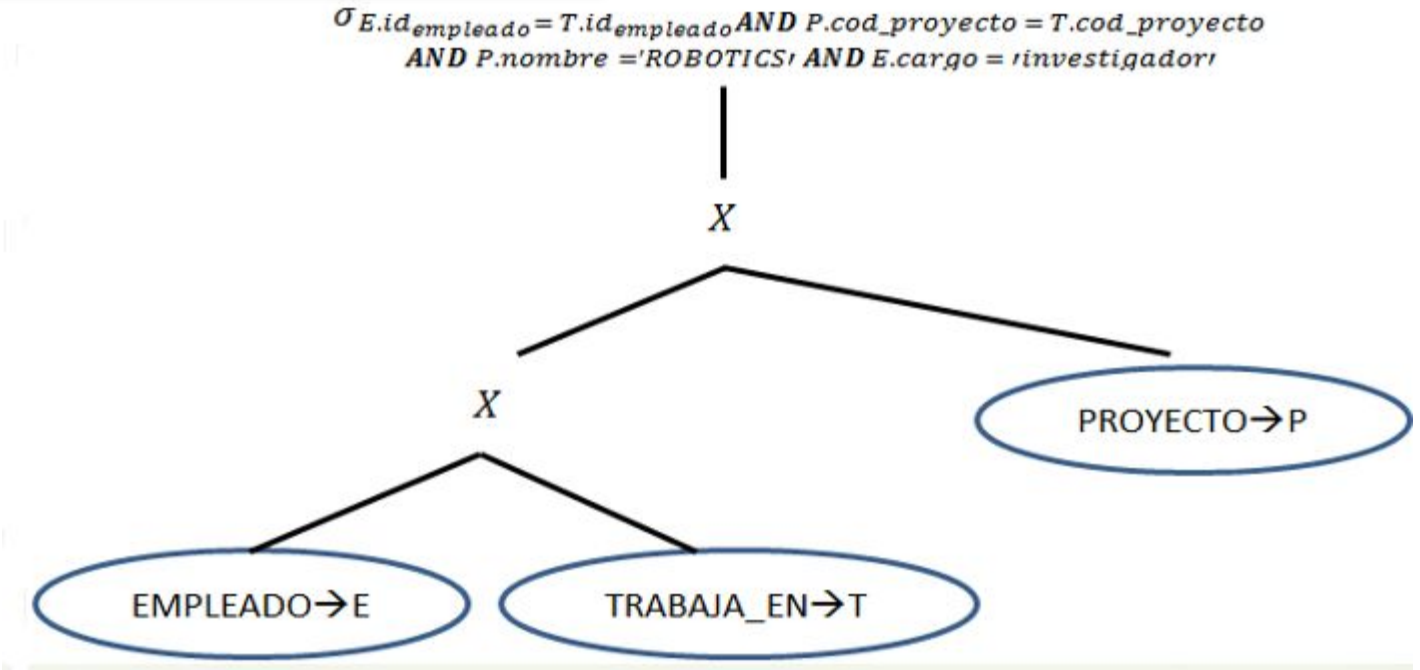
Ejercicio – Ítem 2

```
SELECT E.apellido
FROM    EMPLEADO E      ,
        TRABAJA_EN T    ,
        PROYECTO P
WHERE E.id_empleado = T.id_empleado
AND P.cod_proyecto = T.cod_proyecto
AND P.nombre = 'ROBOTICS'
AND E.cargo = 'investigador';
```



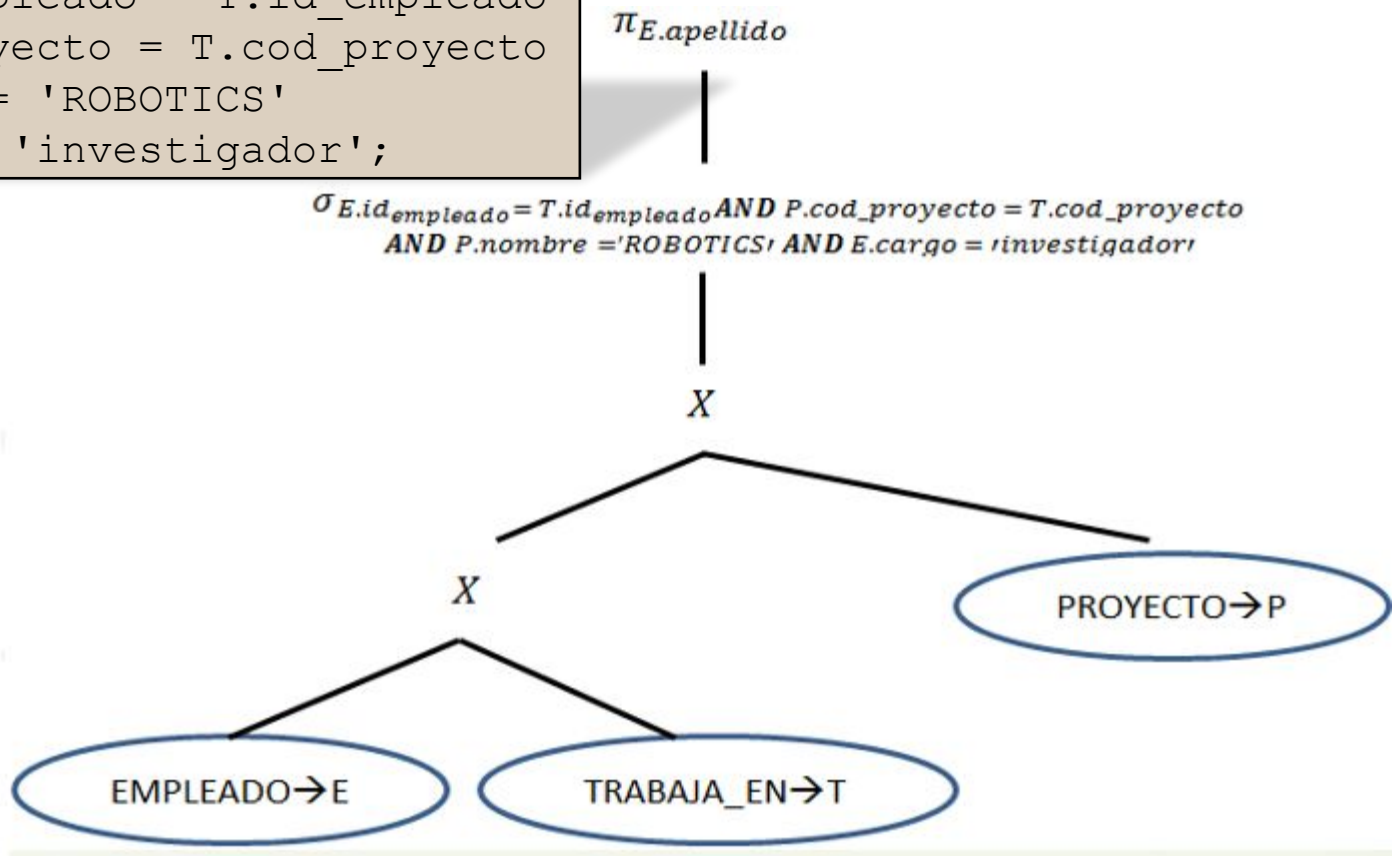
Ejercicio – Ítem 2

```
SELECT E.apellido
FROM    EMPLEADO E      ,
        TRABAJA_EN T    ,
        PROYECTO P
WHERE E.id_empleado = T.id_empleado
AND P.cod_proyecto = T.cod_proyecto
AND P.nombre = 'ROBOTICS'
AND E.cargo = 'investigador';
```



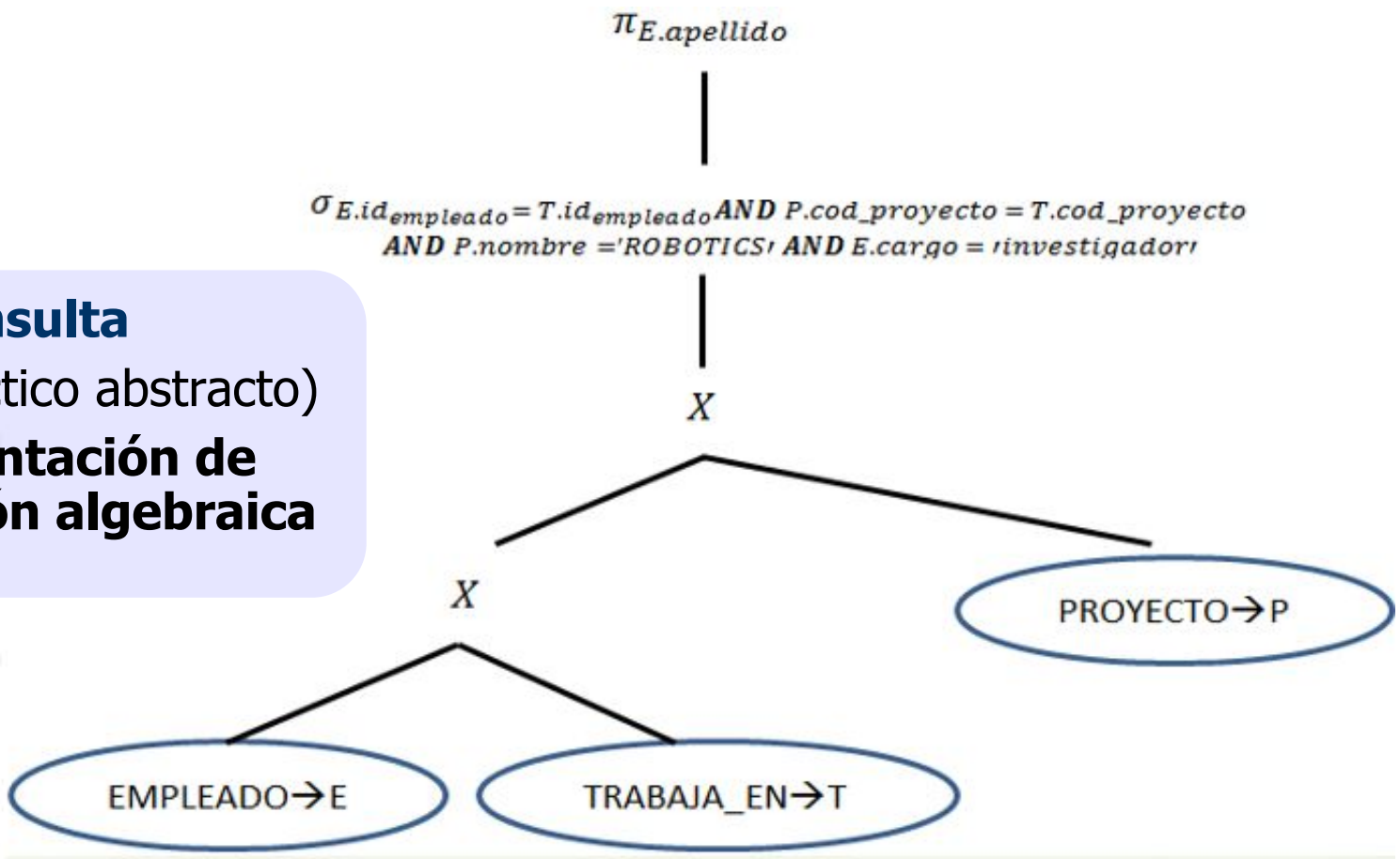
Ejercicio – Ítem 2

```
SELECT E.apellido
FROM    EMPLEADO E      ,
        TRABAJA_EN T    ,
        PROYECTO P
WHERE E.id_empleado = T.id_empleado
AND P.cod_proyecto = T.cod_proyecto
AND P.nombre = 'ROBOTICS'
AND E.cargo = 'investigador';
```



Ejercicio – Ítem 2

Árbol de Consulta
(o árbol sintáctico abstracto)
es la **representación de una expresión algebraica**



Ejercicio

Se solicita:

1. Escriba la consulta en álgebra relacional.

1. Genere de árbol de consulta inicial o canónico.

1. Utilizando las reglas heurísticas y de equivalencia, convertir el árbol anterior en otro equivalente que sepa que es más eficiente. **Nota: Aplique una regla de equivalencia por vez, indique qué regla aplicó en cada caso.**

1. Con el comando EXPLAIN ANALYSE explique la heurística que utiliza para el query plan el optimizador de PostgreSQL en cada una de las sentencias anteriores.

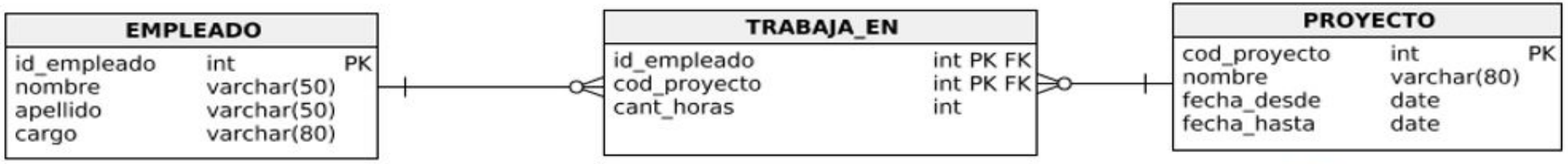
Ejercicio – Ítem 3

- ✓ Solo se necesitan las tuplas del proyecto ROBOTICS
- ✓ Solo se necesitan las tuplas de los empleados con cargo investigador
- ✓ Regla heurística → primero realizar la selección **σ** antes que el producto cartesiano **X**

```
SELECT E.apellido
FROM    EMPLEADO E      ,
        TRABAJA_EN T    ,
        PROYECTO P
WHERE E.id_empleado = T.id_empleado
AND P.cod_proyecto = T.cod_proyecto
AND P.nombre = 'ROBOTICS'
AND E.cargo = 'investigador';
```

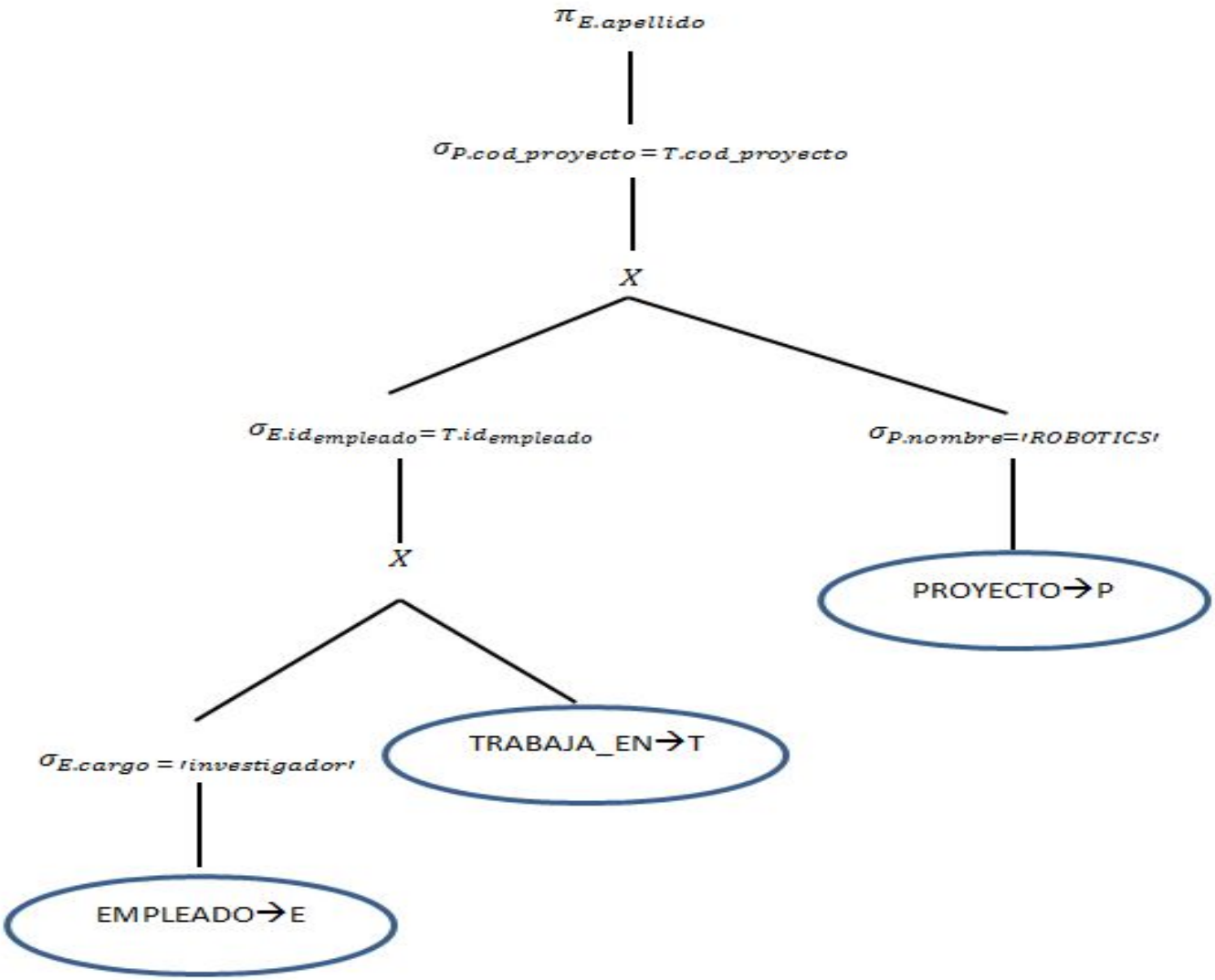
$$\sigma_{c1}(A\ X\ B) \equiv \left(\sigma_{c1}(A)\right) XB$$

C1 involucra sólo atributos de la relación A



Vertabelo

Ejercicio – Ítem 3



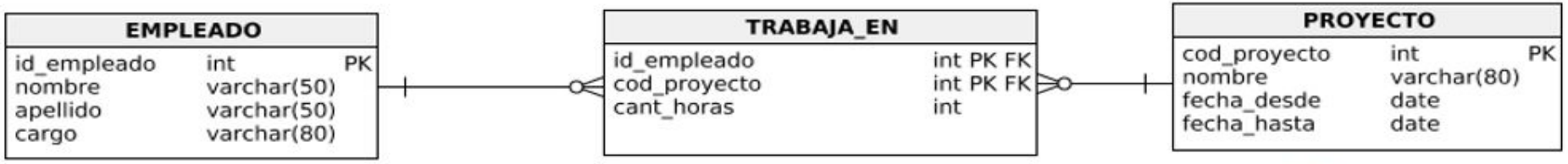
Ejercicio – Ítem 3

```
SELECT E.apellido
FROM    (
          SELECT *
          FROM EMPLEADO
          WHERE cargo = 'investigador'
         )  E      ,
        TRABAJA_EN T    ,
        (
          SELECT *
          FROM PROYECTO
          WHERE nombre = 'ROBOTICS'
         ) P
WHERE E.id_empleado = T.id_empleado
AND P.cod_proyecto = T.cod_proyecto;
```

Ejercicio – Ítem 3

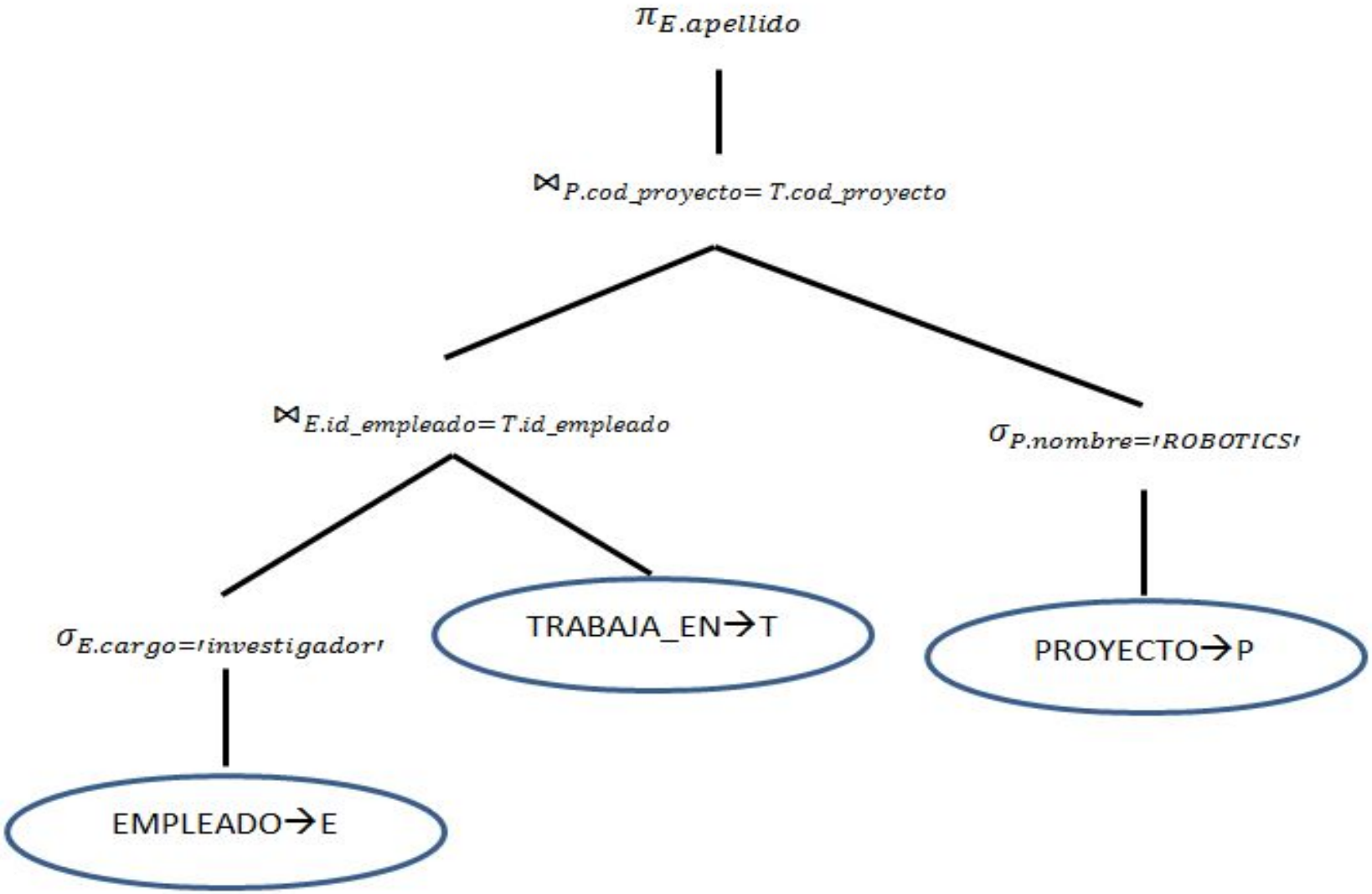
- ✓ Es conveniente utilizar el Equijoin $\bowtie_{Condición}$ que el producto cartesiano X

$$R_1 \bowtie_{R1.A=R2.B} R_2 \equiv \sigma_{R1.A=R2.B}(R_1 \times R_2)$$



Vertabelo

Ejercicio – Ítem 3



Ejercicio – Ítem 3

```
SELECT E.apellido
FROM  (
      SELECT *
      FROM EMPLEADO
      WHERE cargo = 'investigador'
      )  E
      INNER JOIN
      TRABAJA_EN T  ON (E.id_empleado = T.id_empleado)
      INNER JOIN
      (
      SELECT *
      FROM PROYECTO
      WHERE nombre = 'ROBOTICS'
      ) P ON (P.cod_proyecto = T.cod_proyecto);
```

Ejercicio – Ítem 3

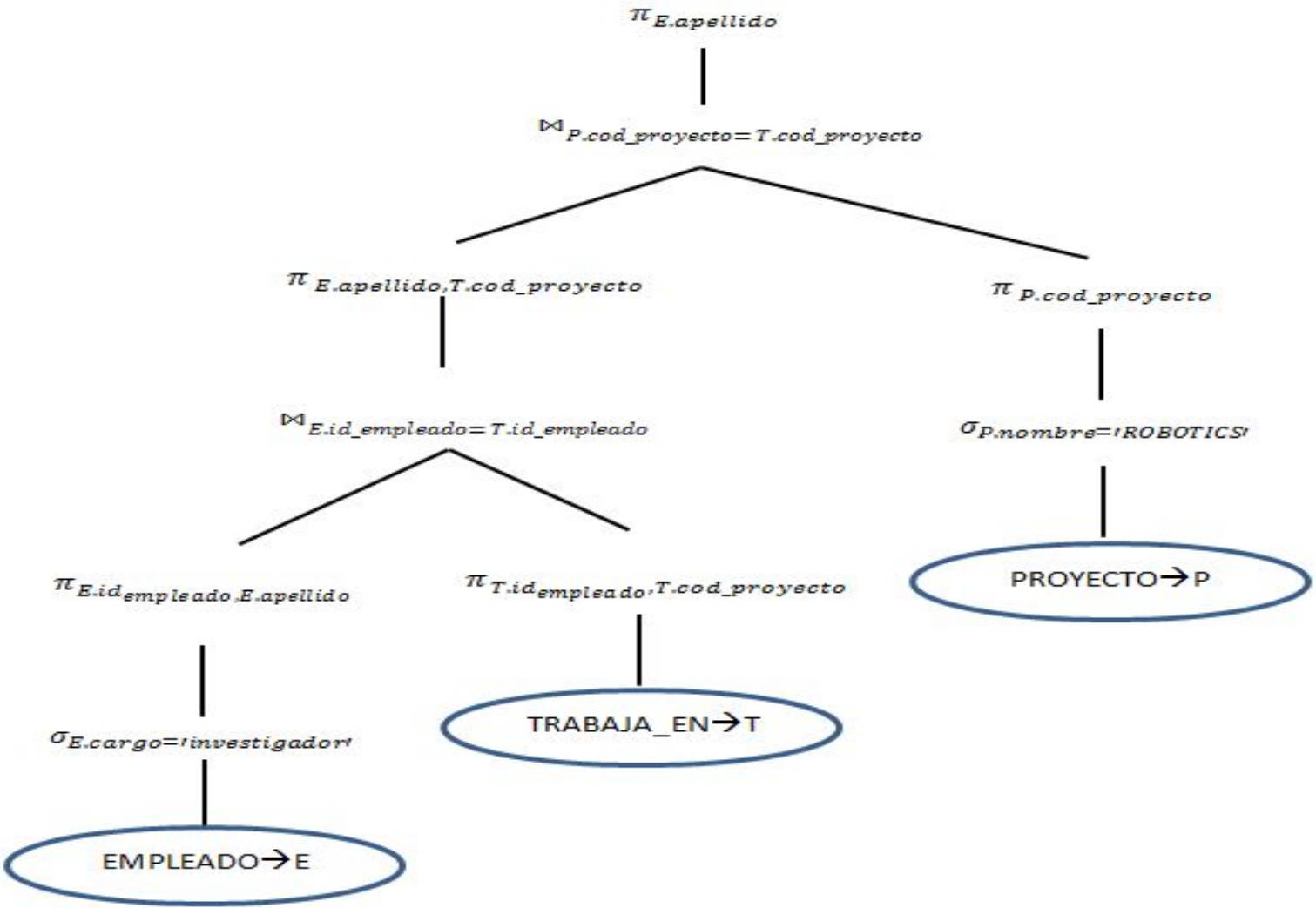
- ✓ Es conveniente proyectar solo los atributos necesarios para las operaciones que siguen.

π es distributiva para el ensamble si:

Si π involucra solo atributos de $L1 \cup L2$ ($L1$ y $L2$ conjunto de atributos de A y B respectivamente, $L1 \cap L2 \neq \emptyset$)

$$\pi_{L1 \cup L2}(A \bowtie_c B) \equiv \pi_{L1}(A) \bowtie_c \pi_{L2}(B)$$

Ejercicio – Ítem 3



Ejercicio – Ítem 3

```
SELECT E.apellido
FROM  ( SELECT E.apellido, T.cod_proyecto
        FROM (
              SELECT id_empleado, apellido
              FROM EMPLEADO
              WHERE cargo = 'investigador'
              )  E
              INNER JOIN
              (
              SELECT id_empleado, cod_proyecto
              FROM TRABAJA_EN
              ) T ON (E.id_empleado = T.id_empleado)
        ) E
        INNER JOIN
       ( SELECT cod_proyecto
         FROM PROYECTO
         WHERE nombre = 'ROBOTICS'
     ) ) P ON (P.cod_proyecto = E. cod_proyecto);
```

Ejercicio

Se solicita:

1. Escriba la consulta en álgebra relacional.

1. Genere de árbol de consulta inicial o canónico.

1. Utilizando las reglas heurísticas y de equivalencia, convertir el árbol anterior en otro equivalente que sepa que es más eficiente. **Nota: Aplique una regla de equivalencia por vez, indique qué regla aplicó en cada caso.**

1. Con el comando EXPLAIN ANALYSE explique la heurística que utiliza para el query plan el optimizador de PostgreSQL en cada una de las sentencias anteriores.

Ejercicio

Se solicita:

1. Escriba la consulta en álgebra relacional.

1. Genere de árbol de consulta inicial o canónico.

1. Utilizando las reglas heurísticas y de equivalencia, convertir el árbol anterior en otro equivalente que sepa que es más eficiente. **Nota: Aplique una regla de equivalencia por vez, indique qué regla aplicó en cada caso.**

1. Con el comando EXPLAIN ANALYSE explique la heurística que utiliza para el query plan el optimizador de PostgreSQL en cada una de las sentencias anteriores.

Ejercicio – Ítem 4

4. Con el comando EXPLAIN ANALYSE explique la heurística que utiliza para el query plan el optimizador de PostgreSQL en cada una de las sentencias anteriores.

```
EXPLAIN ANALYZE SELECT E.apellido
FROM    EMPLEADO E      ,
        TRABAJA_EN T    ,
        PROYECTO P
WHERE E.id_empleado = T.id_empleado
AND P.cod_proyecto = T.cod_proyecto
AND P.nombre = 'ROBOTICS'
AND E.cargo = 'investigador';
```

```
EXPLAIN ANALYZE SELECT E.apellido
FROM  ( SELECT E.apellido, T.cod_proyecto
        FROM (
          SELECT id_empleado, apellido
          FROM EMPLEADO
          WHERE cargo = 'investigador'
          )  E
          INNER JOIN
         (
           SELECT id_empleado, cod_proyecto
           FROM TRABAJA_EN
         ) T ON (E.id_empleado =
T.id_empleado)
        ) E
        INNER JOIN
       ( SELECT cod_proyecto
         FROM PROYECTO
         WHERE nombre = 'ROBOTICS'
    ) P ON (P.cod_proyecto = E.
cod_proyecto);
```

Ejercicio – Ítem 4

	QUERY PLAN text
1	Nested Loop (cost=1.05..3.33 rows=1 width=118) (actual time=0.051..0.088 rows=4 loops=1)
2	Join Filter: (t.id_empleado = e.id_empleado)
3	Rows Removed by Join Filter: 20
4	-> Seq Scan on empleado e (cost=0.00..1.07 rows=1 width=122) (actual time=0.022..0.023 rows=4 loops=1)
5	Filter: ((cargo)::text = 'investigador'::text)
6	Rows Removed by Filter: 2
7	-> Hash Join (cost=1.05..2.20 rows=4 width=4) (actual time=0.008..0.011 rows=6 loops=4)
8	Hash Cond: (t.cod_proyecto = p.cod_proyecto)
9	-> Seq Scan on trabaja_en t (cost=0.00..1.11 rows=11 width=8) (actual time=0.003..0.004 rows=11 loops=4)
10	-> Hash (cost=1.04..1.04 rows=1 width=4) (actual time=0.013..0.013 rows=1 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 9kB
12	-> Seq Scan on proyecto p (cost=0.00..1.04 rows=1 width=4) (actual time=0.008..0.009 rows=1 loops=1)
13	Filter: ((nombre)::text = 'ROBOTICS'::text)
14	Rows Removed by Filter: 2
15	Planning Time: 0.295 ms
16	Execution Time: 0.118 ms

Ejercicio – Ítem 4

EMPLEADO

	4 byte	118 byte	118 byte	178 byte
	id_empleado [PK] integer	nombre character varying (50)	apellido character varying (50)	cargo character varying (80)
1	1	Juan	Perez	investigador
2	2	Rosa	Gomez	investigador
3	3	Bruno	Fernandez	becario
4	4	Ingnacio	Rodriguez	becario
5	5	Alejandro	Perez	investigador
6	6	Sebastin	Solano	investigador

PROYECTO

	4 byte	178 byte	4 byte	4 byte
	cod_proyecto [PK] integer	nombre character varying (80)	fecha_desde date	fecha_hasta date
1	1	ROBOTICS	2018-01-01	2020-01-01
2	2	IA	2017-01-01	2020-01-01
3	3	IMAGE	2015-01-01	2017-01-01

TRABAJA_EN

	4 byte	4 byte	4 byte
	id_empleado [PK] integer	cod_proyecto [PK] integer	cant_horas integer
1	1	1	40
2	2	1	40
3	3	1	20
4	4	1	20
5	5	1	40
6	6	1	40
7	1	2	40
8	3	2	20
9	6	2	20
10	3	3	10
11	6	3	10

Ejercicio – Ítem 4

	QUERY PLAN text
1	Nested Loop (cost=1.05..3.33 rows=1 width=118) (actual time=0.051..0.088 rows=4 loops=1)
2	Join Filter: (t.id_empleado = e.id_empleado)
3	Rows Removed by Join Filter: 20
4	-> Seq Scan on empleado e (cost=0.00..1.07 rows=1 width=122) (actual time=0.022..0.023 rows=4 loops=1)
5	Filter: ((cargo)::text = 'investigador'::text)
6	Rows Removed by Filter: 2
7	-> Hash Join (cost=1.05..2.20 rows=4 width=4) (actual time=0.008..0.011 rows=6 loops=4)
8	Hash Cond: (t.cod_proyecto = p.cod_proyecto)
9	-> Seq Scan on trabaja_en t (cost=0.00..1.11 rows=11 width=8) (actual time=0.003..0.004 rows=11 loops=4)
10	-> Hash (cost=1.04..1.04 rows=1 width=4) (actual time=0.013..0.013 rows=1 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 9kB
12	-> Seq Scan on proyecto p (cost=0.00..1.04 rows=1 width=4) (actual time=0.008..0.009 rows=1 loops=1)
13	Filter: ((nombre)::text = 'ROBOTICS'::text)
14	Rows Removed by Filter: 2
15	Planning Time: 0.295 ms
16	Execution Time: 0.118 ms

Ejercicio – Ítem 4

```
10    -> Hash  (cost=1.04..1.04 rows=1 width=4) (actual time=0.013..0.013 rows=1 loops=1)
11          Buckets: 1024  Batches: 1  Memory Usage: 9kB
12          -> Seq Scan on proyecto p  (cost=0.00..1.04 rows=1 width=4) (actual time=0.008..0.009 rows=1 loops=1)
13                Filter: ((nombre)::text = 'ROBOTICS'::text)
14                Rows Removed by Filter: 2
```

PROYECTO

	4 byte	178 byte	4 byte	4 byte
	cod_proyecto [PK] integer	nombre character varying (80)	fecha_desde date	fecha_hasta date
1	1	ROBOTICS	2018-01-01	2020-01-01
2	2	IA	2017-01-01	2020-01-01
3	3	IMAGE	2015-01-01	2017-01-01

	cod_proyecto [PK] integer
1	1

Seq Scan: Escaneo secuencial de toda la tabla PROYECTO.

Hash: Carga los registros candidatos dentro de una tabla hash.

```
SELECT cod_proyecto
FROM PROYECTO
WHERE nombre = 'ROBOTICS';
```

Ejercicio – Ítem 4

	QUERY PLAN text
1	Nested Loop (cost=1.05..3.33 rows=1 width=118) (actual time=0.051..0.088 rows=4 loops=1)
2	Join Filter: (t.id_empleado = e.id_empleado)
3	Rows Removed by Join Filter: 20
4	-> Seq Scan on empleado e (cost=0.00..1.07 rows=1 width=122) (actual time=0.022..0.023 rows=4 loops=1)
5	Filter: ((cargo)::text = 'investigador'::text)
6	Rows Removed by Filter: 2
7	-> Hash Join (cost=1.05..2.20 rows=4 width=4) (actual time=0.008..0.011 rows=6 loops=4)
8	Hash Cond: (t.cod_proyecto = p.cod_proyecto)
9	-> Seq Scan on trabaja_en t (cost=0.00..1.11 rows=11 width=8) (actual time=0.003..0.004 rows=11 loops=4)
10	-> Hash (cost=1.04..1.04 rows=1 width=4) (actual time=0.013..0.013 rows=1 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 9kB
12	-> Seq Scan on proyecto p (cost=0.00..1.04 rows=1 width=4) (actual time=0.008..0.009 rows=1 loops=1)
13	Filter: ((nombre)::text = 'ROBOTICS'::text)
14	Rows Removed by Filter: 2
15	Planning Time: 0.295 ms
16	Execution Time: 0.118 ms

Ejercicio – Ítem 4

```
9    -> Seq Scan on trabaja_en t (cost=0.00..1.11 rows=11 width=8) (actual time=0.003..0.004 rows=11 loops=4)
```

TRABAJA_EN

	id_empleado [PK] integer (4 byte)	cod_proyecto [PK] integer (4 byte)	cant_horas integer (4 byte)
1	1	1	40
2	2	1	40
3	3	1	20
4	4	1	20
5	5	1	40
6	6	1	40
7	1	2	40
8	3	2	20
9	6	2	20
10	3	3	10
11	6	3	10

	id_empleado [PK] integer	cod_proyecto [PK] integer
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1
6	6	1
7	1	2
8	3	2
9	6	2
10	3	3
11	6	3

Seq Scan: Escaneo secuencial de toda la tabla TRABAJA_EN.

```
SELECT id_empleado, cod_proyecto
FROM trabaja_en
```

Ejercicio – Ítem 4

	QUERY PLAN text
1	Nested Loop (cost=1.05..3.33 rows=1 width=118) (actual time=0.051..0.088 rows=4 loops=1)
2	Join Filter: (t.id_empleado = e.id_empleado)
3	Rows Removed by Join Filter: 20
4	-> Seq Scan on empleado e (cost=0.00..1.07 rows=1 width=122) (actual time=0.022..0.023 rows=4 loops=1)
5	Filter: ((cargo)::text = 'investigador'::text)
6	Rows Removed by Filter: 2
7	-> Hash Join (cost=1.05..2.20 rows=4 width=4) (actual time=0.008..0.011 rows=6 loops=4)
8	Hash Cond: (t.cod_proyecto = p.cod_proyecto)
9	-> Seq Scan on trabaja_en t (cost=0.00..1.11 rows=11 width=8) (actual time=0.003..0.004 rows=11 loops=4)
10	-> Hash (cost=1.04..1.04 rows=1 width=4) (actual time=0.013..0.013 rows=1 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 9kB
12	-> Seq Scan on proyecto p (cost=0.00..1.04 rows=1 width=4) (actual time=0.008..0.009 rows=1 loops=1)
13	Filter: ((nombre)::text = 'ROBOTICS'::text)
14	Rows Removed by Filter: 2
15	Planning Time: 0.295 ms
16	Execution Time: 0.118 ms

Ejercicio – Ítem 4

```
7   -> Hash Join  (cost=1.05..2.20 rows=4 width=4) (actual time=0.008..0.011 rows=6 loops=4)
8       Hash Cond: (t.cod_proyecto = p.cod_proyecto)
9       -> Seq Scan on trabaja_en t  (cost=0.00..1.11 rows=11 width=8) (actual time=0.003..0.004 rows=11 loops=4)
10      -> Hash  (cost=1.04..1.04 rows=1 width=4) (actual time=0.013..0.013 rows=1 loops=1)
11          Buckets: 1024  Batches: 1  Memory Usage: 9kB
12          -> Seq Scan on proyecto p  (cost=0.00..1.04 rows=1 width=4) (actual time=0.008..0.009 rows=1 loops=1)
13              Filter: ((nombre)::text = 'ROBOTICS'::text)
14              Rows Removed by Filter: 2
```

Hash Join: Realiza un ensamble hash cargando las tuplas de un lado del ensamble.

	id_empleado [PK] integer	cod_proyecto [PK] integer
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1
6	6	1
7	1	2
8	3	2
9	6	2
10	3	3
11	6	3

⋈

	cod_proyecto [PK] integer
1	1

→

	id_empleado integer
1	1
2	2
3	3
4	4
5	5
6	6

Ejercicio – Ítem 4

```
SELECT t.id_empleado
FROM ( SELECT id_empleado, cod_proyecto
       FROM trabaja_en) t
     JOIN
     ( SELECT cod_proyecto
       FROM proyecto
       WHERE nombre = 'ROBOTICS') p
     ON (t.cod_proyecto = p.cod_proyecto)
```

Ejercicio – Ítem 4

	QUERY PLAN text
1	Nested Loop (cost=1.05..3.33 rows=1 width=118) (actual time=0.051..0.088 rows=4 loops=1)
2	Join Filter: (t.id_empleado = e.id_empleado)
3	Rows Removed by Join Filter: 20
4	-> Seq Scan on empleado e (cost=0.00..1.07 rows=1 width=122) (actual time=0.022..0.023 rows=4 loops=1)
5	Filter: ((cargo)::text = 'investigador'::text)
6	Rows Removed by Filter: 2
7	-> Hash Join (cost=1.05..2.20 rows=4 width=4) (actual time=0.008..0.011 rows=6 loops=4)
8	Hash Cond: (t.cod_proyecto = p.cod_proyecto)
9	-> Seq Scan on trabaja_en t (cost=0.00..1.11 rows=11 width=8) (actual time=0.003..0.004 rows=11 loops=4)
10	-> Hash (cost=1.04..1.04 rows=1 width=4) (actual time=0.013..0.013 rows=1 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 9kB
12	-> Seq Scan on proyecto p (cost=0.00..1.04 rows=1 width=4) (actual time=0.008..0.009 rows=1 loops=1)
13	Filter: ((nombre)::text = 'ROBOTICS'::text)
14	Rows Removed by Filter: 2
15	Planning Time: 0.295 ms
16	Execution Time: 0.118 ms

Ejercicio – Ítem 4

```
4  -> Seq Scan on empleado e (cost=0.00..1.07 rows=1 width=122) (actual time=0.022..0.023 rows=4 loops=1)
5       Filter: ((cargo)::text = 'investigador'::text)
6       Rows Removed by Filter: 2
```

EMPLEADO

	4 byte	118 byte	118 byte	178 byte
	id_empleado [PK] integer	nombre character varying (50)	apellido character varying (50)	cargo character varying (80)
1	1	Juan	Perez	investigador
2	2	Rosa	Gomez	investigador
3	3	Bruno	Fernandez	becario
4	4	Ingnacio	Rodriguez	becario
5	5	Alejandro	Perez	investigador
6	6	Sebastin	Solano	investigador

	id_empleado [PK] integer	apellido character varying (50)
1	1	Perez
2	2	Gomez
3	5	Perez
4	6	Solano

Seq Scan: Escaneo secuencial de toda la tabla EMPLEADO.

```
SELECT id_empleado, apellido
FROM empleado
WHERE cargo = 'investigador'
```

Ejercicio – Ítem 4

	QUERY PLAN text
1	Nested Loop (cost=1.05..3.33 rows=1 width=118) (actual time=0.051..0.088 rows=4 loops=1)
2	Join Filter: (t.id_empleado = e.id_empleado)
3	Rows Removed by Join Filter: 20
4	-> Seq Scan on empleado e (cost=0.00..1.07 rows=1 width=122) (actual time=0.022..0.023 rows=4 loops=1)
5	Filter: ((cargo)::text = 'investigador'::text)
6	Rows Removed by Filter: 2
7	-> Hash Join (cost=1.05..2.20 rows=4 width=4) (actual time=0.008..0.011 rows=6 loops=4)
8	Hash Cond: (t.cod_proyecto = p.cod_proyecto)
9	-> Seq Scan on trabaja_en t (cost=0.00..1.11 rows=11 width=8) (actual time=0.003..0.004 rows=11 loops=4)
10	-> Hash (cost=1.04..1.04 rows=1 width=4) (actual time=0.013..0.013 rows=1 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 9kB
12	-> Seq Scan on proyecto p (cost=0.00..1.04 rows=1 width=4) (actual time=0.008..0.009 rows=1 loops=1)
13	Filter: ((nombre)::text = 'ROBOTICS'::text)
14	Rows Removed by Filter: 2
15	Planning Time: 0.295 ms
16	Execution Time: 0.118 ms

Ejercicio – Ítem 4

1	Nested Loop (cost=1.05..3.33 rows=1 width=118) (actual time=0.051..0.088 rows=4 loops=1)
2	Join Filter: (t.id_empleado = e.id_empleado)
3	Rows Removed by Join Filter: 20
4	-> Seq Scan on empleado e (cost=0.00..1.07 rows=1 width=122) (actual time=0.022..0.023 rows=4 loops=1)
5	Filter: ((cargo)::text = 'investigador'::text)
6	Rows Removed by Filter: 2
7	-> Hash Join (cost=1.05..2.20 rows=4 width=4) (actual time=0.008..0.011 rows=6 loops=4)
8	Hash Cond: (t.cod_proyecto = p.cod_proyecto)
9	-> Seq Scan on trabaja_en t (cost=0.00..1.11 rows=11 width=8) (actual time=0.003..0.004 rows=11 loops=4)
10	-> Hash (cost=1.04..1.04 rows=1 width=4) (actual time=0.013..0.013 rows=1 loops=1)
11	Buckets: 1024 Batches: 1 Memory Usage: 9kB
12	-> Seq Scan on proyecto p (cost=0.00..1.04 rows=1 width=4) (actual time=0.008..0.009 rows=1 loops=1)
13	Filter: ((nombre)::text = 'ROBOTICS'::text)
14	Rows Removed by Filter: 2

Nested Loop: Ensambla dos tablas ante una condición establecida.

Ejercicio – Ítem 4

	id_empleado [PK] integer	apellido character varying (50)
1	1	Perez
2	2	Gomez
3	5	Perez
4	6	Solano

X

	id_empleado integer
1	1
2	2
3	3
4	4
5	5
6	6

	apellido character varying (50)
1	Perez
2	Gomez
3	Perez
4	Solano

```
SELECT e.apellido
FROM(  SELECT id_empleado,apellido
       FROM empleado
       WHERE cargo = 'investigador') e,
    (  SELECT t.id_empleado
       FROM ( SELECT id_empleado, cod_proyecto
              FROM trabaja_en) t
       JOIN
            ( SELECT cod_proyecto
              FROM proyecto
              WHERE nombre = 'ROBOTICS') p
       ON (t.cod_proyecto = p.cod_proyecto))t
WHERE e.id_empleado = t.id_empleado;
```

ALGUNOS TIPS

ALGUNOS TIPS

La clave para escribir una buena consulta SQL es seguir el proceso paso a paso:

1. Hacer una lista de tablas de las que se deben recuperar datos
2. Pensar en cómo unir dichas tablas
3. Pensar en cómo obtener la mínima cantidad de registros que participan en la condición de unión

ALGUNOS TIPS

- No suponer que la consulta funciona bien con pocos datos en la tabla.
- Usar EXPLAIN para comprender lo que está sucediendo
- En general, EXISTS y JOIN dan buenos resultados
- PostgreSQL optimiza la cláusula IN en un subplan con hash en muchos casos.

ALGUNOS TIPS

- ✓ Consulta: **Listar el identificador y nombre de todos los distribuidores nacionales que no realizaron entregas (**Esquema peliculas**):**

```
SELECT d.id_distribuidor, d.nombre
FROM distribuidor d JOIN nacional n
     ON (d.id_distribuidor = n.id_distribuidor)
WHERE NOT EXISTS (
                  SELECT id_distribuidor
                  FROM entrega e
                  WHERE e.id_distribuidor = d.id_distribuidor
                  );
```

ALGUNOS TIPS

- ✓ Consulta Equivalente: **Listar el identificador y nombre de todos los distribuidores nacionales que no realizaron entregas (**Esquema peliculas**):**

```
SELECT d.id_distribuidor, d.nombre
FROM distribuidor d
WHERE d.tipo = 'N'
AND NOT EXISTS (
                SELECT id_distribuidor
                FROM unc_esq_peliculas.entrega e
                WHERE e.id_distribuidor = d.id_distribuidor
                );
```

ALGUNOS TIPS

```
QUERY PLAN
Nested Loop  (cost=182.75..215.78 rows=1 width=21) (actual time=1.411..1.599 rows=3 loops=1)
  ->  Hash Anti Join  (cost=182.47..215.47 rows=1 width=21) (actual time=1.308..1.580 rows=6 loops=1)
        Hash Cond: (d.id_distribuidor = e.id_distribuidor)
        ->  Seq Scan on distribuidor d  (cost=0.00..22.50 rows=1050 width=21) (actual time=0.005..0.073 rows=1050 loops=1)
        ->  Hash  (cost=106.10..106.10 rows=6110 width=5) (actual time=1.227..1.228 rows=5024 loops=1)
              Buckets: 8192  Batches: 1  Memory Usage: 246kB
              ->  Seq Scan on entrega e  (cost=0.00..106.10 rows=6110 width=5) (actual time=0.006..0.552 rows=5024 loops=1)
  ->  Index Only Scan using pk_nacional on nacional n  (cost=0.28..0.31 rows=1 width=5) (actual time=0.002..0.002 rows=0 loops=6)
        Index Cond: (id_distribuidor = d.id_distribuidor)
        Heap Fetches: 0
Planning Time: 0.326 ms
Execution Time: 1.625 ms
```

ALGUNOS TIPS

- ✓ Consulta: **Seleccionar el identificador y nombre de aquellos empleados donde la fecha de nacimiento supere el 1/1/1980 (**Esquema peliculas**):**

```
SELECT id_empleado, nombre
FROM empleado
WHERE fecha_nacimiento > to_date('1/1/1980', 'dd/MM/yyyy');
```

Otros Ejemplos

- ✓ **Sin indices:**

```
QUERY PLAN
1 Seq Scan on empleado  (cost=0.00..1061.22 rows=11695 width=17) (actual time=0.024..14.759 rows=11694 loops=1)
2   Filter: (fecha_nacimiento > to_date('1/1/1980'::text, 'dd/MM/yyyy'::text))
3   Rows Removed by Filter: 23387
4 Planning Time: 0.091 ms
5 Execution Time: 15.105 ms
```

ALGUNOS TIPS

✓ **Con un indice sobre el atributo fecha_nacimiento:**

```
CREATE INDEX I_Empleado_Fecha_Nacimiento ON
EMPLEADO(fecha_nacimiento DESC);
```

```
QUERY PLAN
1 Bitmap Heap Scan on empleado  (cost=222.93..933.35 rows=11695 width=17) (actual time=0.860..2.991 rows=11694 loops=1)
2   Recheck Cond: (fecha_nacimiento > to_date('1/1/1980'::text, 'dd/MM/yyyy'::text))
3   Heap Blocks: exact=535
4   ->  Bitmap Index Scan on i_empleado_fecha_nacimiento  (cost=0.00..220.00 rows=11695 width=0) (actual time=0.806..0.806 rows=11694 loops=1)
5         Index Cond: (fecha_nacimiento > to_date('1/1/1980'::text, 'dd/MM/yyyy'::text))
6 Planning Time: 0.272 ms
7 Execution Time: 3.299 ms
```

OPTIMIZACIÓN

Cualquiera SGBD relacional, requiere generar **el mejor plan posible** para la ejecución de una consulta con el **menor tiempo y recursos**.

Algunos SGBD permiten a los usuarios inspeccionar la estrategia del optimizador para ejecutar una determinada consulta.

En Postgresql sentecia **EXPLAIN (**Query execution plan), se utiliza **EXPLAIN SELECT** ….

EXPLAIN

Inspeccionemos que ocurre en estos ejemplos con el EXPLAIN

Las consultas de inclusión se puede escribir de cuatro maneras diferentes

Por ejemplo :

Listar el identificador de tarea y la cantidad de voluntarios que realizan tareas cuya cantidad de horas minima es mayor o igual a 9000

EXPLAIN

1. Usando la cláusula IN
1. Usando la cláusula ANY
1. Usando la cláusula EXISTS
1. Usando INNER JOIN

Nos dio el mismo plan entonces,

¿podemos concluir que podemos escribir la consulta ya que estamos cómodos y "la inteligencia de PostgreSQL" se encargará del resto?

QUERYs

```
SELECT COUNT(*), id_tarea
FROM voluntario
WHERE id_tarea IN (SELECT id_tarea
                   FROM tarea
                   WHERE min_horas > 9000)
GROUP BY id_tarea;
```

```
SELECT COUNT(*), id_tarea
FROM voluntario
WHERE id_tarea = ANY (
                   SELECT id_tarea
                   FROM tarea
                   WHERE min_horas > 9000)
GROUP BY id_tarea;
```

```
SELECT COUNT(*), id_tarea
FROM voluntario V
WHERE EXISTS (
          SELECT 1
           FROM tarea T
            WHERE V.id_tarea = T.id_tarea
                AND min_horas > 9000)
GROUP BY id_tarea;
```

```
SELECT COUNT(*), T.id_tarea
FROM voluntario V JOIN tarea T
              ON (V.id_tarea = T.id_tarea)
WHERE min_horas > 9000
GROUP BY T.id_tarea;
```

EXPLAIN

Inspeccionemos que ocurre con el EXPLAIN en consultas de exclusión

Por ejemplo :

Listar el identificador de tarea y la cantidad de voluntarios que la realizan excepto para las tareas con máxima cantidad de horas > a 2500 (max_horas > 2500)

EXPLAIN

1. Usando la cláusula NOT IN
1. Usando la cláusula ALL
1. Usando la cláusula NOT EXISTS
1. Usando LEFT JOIN y IS NULL

1 y 2 produce un plan de ejecución con subconsultas (SubPlan)

Mientras 3 y 4 produce el mismo plan de ejecución sin subplan con anti-join

QUERYs

```
SELECT COUNT(*), id_tarea
FROM voluntario
WHERE id_tarea NOT IN (
            SELECT id_tarea
            FROM tarea
            WHERE min_horas > 2500)
GROUP BY id_tarea;
```

```
SELECT COUNT(*), id_tarea
FROM voluntario
WHERE id_tarea <> ALL (
            SELECT id_tarea
            FROM tarea
            WHERE min_horas > 2500)
GROUP BY id_tarea;
```

```
SELECT COUNT(*), id_tarea
FROM voluntario V
WHERE NOT EXISTS (
        SELECT 1
         FROM tarea T
          WHERE V.id_tarea = T.id_tarea
              AND min_horas > 2500)
GROUP BY id_tarea;
```

```
SELECT COUNT(*), T.id_tarea
FROM voluntario V LEFT JOIN tarea T
          ON (V.id_tarea = T.id_tarea
          AND min_horas > 2500)
WHERE T.id_tarea IS NULL
GROUP BY T.id_tarea;
```

ALGUNOS TIPS

Todas las variantes de algoritmos de hash join implican **construir tablas hash** a partir de los registros de una o ambas tablas unidas, y posteriormente buscar para cada registro con el mismo código hash en la otra tabla

Las uniones de hash suelen ser más eficientes que las uniones de bucles anidados, excepto cuando el lado interno de la unión es muy pequeño

BIBLIOGRAFÍA

Elmasri, R.; Navathe, S.B.: **Fundamentos de Sistemas de Bases de Datos**. 3ª Edición. Addison-Wesley. (Cap. 18)

Elmasri, R.; Navathe, S.B.: **Sistemas de bases de datos. Conceptos fundamentales**. 2ª Ed. Addison-Wesley Iberoamericana. (Cap. 16)

Connolly et al.: **Database Systems: A Practical Approach to Design, Implementation and Management**. 2nd Ed. Addison-Wesley (Cap. 18)