Un método heurístico para optimizar la respuesta a consultas relacionales.

Pablo Aranda Luna

Trabajo de fin de grado de Matemáticas
Universidad de Zaragoza
Junio 2024

Director del trabajo
Jorge Lloret Gazo

Abstract

The relational model in the field of databases has had a significant impact since its initial implementations in database management programs. It allows organizing information into structures called relations. This model facilitates data manipulation and querying through a standard language known as SQL (Structured Query Language).

In the context of relational databases, query optimization is an essential aspect because it enables the machine to provide quick and efficient responses to the user, which becomes increasingly important when dealing with large databases.

On the following pages, techniques based on cost and heuristic methods used in the optimization of relational queries are described. It demonstrates the use of relational algebra in manipulating the information stored in a relational database, and how query trees play an important role in this process.

Through the implementation and comparison of different optimization algorithms, including query transformation and index usage, significant reductions in query response times can be achieved.

Índice general

Capítulo 1

El modelo relacional

El concepto del **modelo relacional** para la creación de bases de datos, fue acuñado por primera vez por el científico informático inglés **Edgar Frank Codd**, en su conocido artículo *A relational model of data for large shared data banks* [1]. En dicho artículo Codd, con la intención de mejorar y optimizar el funcionamiento de grandes bases de datos, introdujo un nuevo enfoque para la estructuración de su información, la **base de datos relacional**.

En su artículo, Codd presenta su modelo relacional como un conjunto de relaciones, donde cada una se define como un conjunto de registros de la base de datos. Esta definición permite aplicar a las relaciones operaciones básicas de conjuntos. Por ejemplo, permite permutar, proyectar y combinar elementos de una relación.

Su objetivo consistía en desarrollar un modelo en el cual fuera únicamente necesario describir la estructura natural de los datos, sin requerir de un sistema adicional para su representación. Los modelos ya existentes eran vulnerables a cambios en la organización e información, de igual modo presentaban una fuerte dependencia del orden e indexación de los datos. La perspectiva relacional permitía sentar un cimiento sólido para gestionar la redundancia y consistencia, a diferencia de los modelos de red generales que tenían grandes limitaciones para abordar estos asuntos al almacenar la información.

Las ideas de Codd atrajeron rápidamente la atención de gran parte del sector de la industria informática. En los años posteriores a la publicación del artículo, IBM, empresa en la cual trabajaba Codd, desarrolló un sistema de gestión de bases de datos conocido como System R. Este sistema introdujo un lenguaje de consulta para implementar el modelo propuesto, el cual fue llamado SEQUEL (Structured English Query Language) y que, en siguientes versiones, se convirtió en lo que hoy conocemos como **SQL** (Structured Query Language).

Por otro lado, Larry Ellison, Ed Oates y Bob Miner, cofundadores de Software Development Laboratories, motivados por el modelo de Codd desarrollaron un sistema de gestión de bases de datos conocido como Oracle Database 2.0. Dicha empresa es la hoy conocida como **Oracle Corporation**. Su producto tuvo un gran éxito, y gracias a él SQL alcanzó gran reconocimiento en el tratamiento de las bases de datos. Desde entonces han sido desarrollados gran cantidad de sistemas basados en tal modelo, algunos incluso de software libre. En la actualidad SQL se ha convertido en el lenguaje estándar en los sistemas de gestión de bases de datos que basan su estructura en el modelo relacional.

El modelo relacional, para la creación y estructuración de bases de datos, se basa en el concepto de **relación matemática**. Se fundamenta en la **teoría de conjuntos** y en la **lógica de predicados**. En este primer capítulo trataremos de explicar en qué consiste el modelo relacional, para ello definiremos sus conceptos básicos y veremos cómo se relacionan los diferentes elementos de dicho modelo.

En el modelo relacional, una base de datos se representa como una colección de relaciones. Cada una de estas relaciones es representada con una tabla, donde cada fila está formada por un conjunto de datos, con una determinada relación entre sí, conformando un registro en la base de datos, es decir, cada fila representa un objeto real. Los nombres de las columnas nos ayudan a interpretar el tipo de información que contiene cada fila.

Cada fila se denomina **tupla** y cada columna se llama **atributo**. Finalmente a la tabla en su conjunto es lo que conocemos como **relación**.

1.1. Estructura

En esta primera sección se describe la estructura de una base de datos relacional, así como de los diferentes elementos que la componen.

Definición. Cada una de las propiedades por las que un objeto, o entidad, es caracterizado en una base de datos relacional se conoce como **atributo**.

Cuando una relación es gráficamente representada podemos identificar los atributos como las diferentes columnas que la componen.

Definición. Denominamos como **dominio** al conjunto de los posibles valores que un atributo puede tomar, lo denotamos D.

Además, cada elemento del dominio es indivisible. Un dominio está dado por un nombre, un tipo de valor (número, cadena, etc), y por un formato. Para definir un dominio, por ejemplo el dominio de las matrículas de coche, podemos hacerlo de las siguientes maneras:

- Definición lógica: Conjunto de todas las matrículas válidas de tres letras y cuatro números.
- Definición formal: Conjunto de todas las cadenas de la forma $LLLNNNN$, siendo L una letra y N un número natural de 0 a 9.

1.1.1. Esquemas de relación

Para representar una relación y el conjunto de sus atributos usamos un **esquema de relación**. Se denota $R(A_1,...,A_n)$, donde R es el nombre del esquema de relación y $A_1,...,A_n$ una lista de atributos, además el número n de atributos de la relación se conoce como grado de la relación. Cada atributo A_i tiene un dominio en la relación, denotado como $dom(A_i)$.
Un **esquema de base de datos relacional** S es un conjunto de esquemas de relación $S = \{R_1,...,R_m\}$ junto con un conjunto de restricciones de integridad (IC).

1.2. Valores de una tupla

Los valores que una tupla puede almacenar están delimitados por el dominio de sus atributos, como ya hemos visto. Por otro lado, cada valor es **indivisible**, es decir, no se puede separar en componentes. Es por ello que, en el modelo relacional, debemos representar en diferentes relaciones los atributos multievaluados; de la misma forma, un atributo compuesto es necesario definirlo como un conjunto de atributos más básicos. En algunos casos, en los que no conocemos un valor o no existe ese atributo para un registro, se toma como nulo ese atributo (null en inglés).

1.2.1. Relación

Definición. Una **relación** r de un esquema de relación $R(A_1,...,A_n)$, es un conjunto de n-tuplas $r = \{t_1,...,t_m\}$, donde cada n-tupla t_j es una lista ordenada de n valores, es decir, $t_j =< v_1,...,v_n >$ y en donde cada v_i es un elemento de $dom(A_i)$ o es un valor nulo.

En otras palabras, llamamos relación al conjunto de tuplas de un esquema de relación. Cada tupla está formada por un dato para cada atributo. En algunos casos se denomina también como *estado de relación*.

Basándonos en la ya mencionada teoría de conjuntos, podemos formalizar dichos conceptos. Por ejemplo, sean $dom(A_1),...,dom(A_n)$ los respectivos dominios de los atributos $A_1,...,A_n$ de un esquema de relación R. Si hacemos el siguiente producto cartesiano:

$$dom(A_1) \times dom(A_2) \times ... \times dom(A_n)$$

Un estado de relación $r(R)$ es un subconjunto de dicho producto, es decir:

$$r(R) \subseteq dom(A_1) \times dom(A_2) \times ... \times dom(A_n)$$

El producto cartesiano recorre todas las posibles combinaciones de todos los datos de cada uno de los dominios, en este caso de todas las posibles tuplas en la relación. Además, la cardinalidad de este producto aporta el número total de tuplas posibles en un estado $r(R)$.

1.2.2. Orden en una relación

Cuando se trata de los elementos de un conjunto no hay ningún orden entre ellos, por lo tanto en una relación **no hay un orden específico** entre los diferentes registros o tuplas.

Al representar una relación en forma de tabla es cuando se necesita un orden; se suele ordenar en función de alguno de los atributos, y aunque pueda ser representada por tablas ordenadas de diferente manera la relación es la misma, sin importar el atributo que tomemos como referencia en la ordenación.

De igual manera, si hablamos de manera abstracta, no es importante el orden en el que aparecen los diferentes atributos dentro de una tupla, mientras que cada atributo mantenga sus determinados valores. Aunque, como ya hemos visto, un esquema de relación está formado por un conjunto de n-tuplas ordenadas, luego el orden en este caso es importante porque mantenerlo nos permite identificar a qué atributo corresponde cada valor de cada n-tupla.

1.2.3. Interpretación de una relación

Podemos interpretar una relación como una simple afirmación que representa un hecho. Por ejemplo, en la relación:

ESTUDIANTE (Nombre, DNI, Telef_casa, Direccion, Telef_movil, Edad, Nota_media)

Interpretamos cada registro de la relación como un hecho. Si la primera tupla fuera:

(Arturo, 73456625N, 976012545, Calle Pedro Cerbuna, 656241569, 22, 8)

Estaría así afirmando que existe un alumno llamado Arturo, tiene 22 años y tiene una nota media de 8. En este caso la relación $ESTUDIANTE$ está representando hechos de una entidad, un alumno, pero puede ocurrir que represente hechos sobre una relación entre dos entidades. Por ejemplo, en la relación

ESPECIALIZACION(Estudiante_DNI, Codigo_departamento)

Cada tupla de esta relación está correlacionando cada alumno con el departamento de su especialización.

Otra interpretación de un esquema de relación es como si fuera un enunciado. Las tuplas que pertenecen a la relación son aquellas cuyos valores satisfacen dicho enunciado.

1.3. Base de datos relacional

Definición. Una **base de datos relacional** *DB* de *S* es un conjunto de estados de relación $DB = \{r_1, ..., r_m\}$ donde cada r_i satisface las restricciones de integridad.

Dicho de otro modo, una base de datos relacional es un conjunto de relaciones, una por cada esquema de relación que exista en el esquema de base de datos. Si un estado cumple todas las restricciones se denomina estado válido, en su defecto, se denomina no válido. En algunas ocasiones se denomina como *estado de la base de datos*.

En una base de datos relacional las diferentes relaciones interactúan, se interconectan, comparten información unas con otras. Para comprender cómo se relacionan hemos definido los conceptos de esquema y estado de base de datos.

En algún caso es posible encontrar que dos conceptos reales distintos hayan sido llamados de igual manera en relaciones diferentes. En el ejemplo anterior *DNI* y *Estudiante_DNI* representan lo mismo, el número del DNI de cada alumno. Por otro lado, también podría ser posible que en diferentes relaciones el mismo concepto real es llamado de diferente manera. Esto último es importante, ya que el mismo concepto real puede ejercer un rol distinto dentro de una misma relación.

Ejemplo 1.1. En la siguiente relación de los trabajadores de una empresa:

EMPLEADOS(Nombre, DNI, Fecha_nac, Direccion, Sexo, Salario, Supervisor_DNI, Num_Dpto)

A cada empleado se le asigna a un superior, con lo cual, para distinguir el *DNI* de cada trabajador con el *Supervisor_DNI* de su superior asignado, se han creado dos atributos diferentes para representar el mismo concepto real, el número de identificación.

Las relaciones descritas a continuación, junto con la relación presentada *EMPLEADOS*, conforman la base de datos de una empresa, sobre la cual trabajaremos en los siguientes capítulos.

DEPARTAMENTOS (Dnombre, Num_Dpto, Jefe_DNI, Jefe_fecha_comienzo)

DEPTO_LOCALIZACION (Dnum, Dlocalizacion)

Donde en la relación *DEPARTAMENTOS* definimos el nombre y número de identificación del departamento, y la fecha de nacimiento y el número de identificaión del jefe de dicho departamento.
En *DEPTO_LOCALIZACION* el número de identificación y la localización de cada departamento.

PROYECTOS (Pnombre, Pnum, Plocalizacion, Dnum)

TRABAJA_EN (Emp_DNI, Pnum, Horas)

DEPENDIENTES (Emp_DNI, Nombre_dependiente, Sexo, Fecha_nac)

La relación *PROYECTOS* determina el número y nombre de los proyectos, además de su localización y el número de departamento que lo controla. En *TRABAJA_EN* el número de indentificación del empleado, el número del proyecto y el número de horas que invierte en dicho proyecto. Finalmente, la relación *DEPENDIENTES* contiene información sobre las personas dependientes de cada empleado.

Este ejemplo, así como otros empleados a lo largo de este trabajo, ha sido extraído del libro de Elmasri y Navathe *Fundamentals of database systems* [6].

Nombre	DNI	Fecha_nac	Direccion	Sexo	Salario	Supervisor_DNI	Num_Dpto
Juan	123456789	09-01-1965	731 Fondren, Houston	M	30.000	333445555	5
Fran	333445555	08-12-1955	638 Voss, Houston	M	40.000	888665555	5
Alicia	999887777	19-01-1968	3321 Castle, Spring	F	25.000	987654321	4
Jennifer	987654321	20-06-1941	291 Berry, Bellaire	F	43.000	888665555	4
Ramón	666884444	15-09-1962	975 Fire Oak, Humble	M	38.000	333445555	5
Julia	453453453	31-07-1972	5631 Rice, Houston	F	25.000	333445555	5
Ahmad	987987987	29-03-1969	980 Dallas, Houston	M	25.000	987654321	4
Jaime	888665555	10-11-1937	450 Stone, Houston	M	55.000	NULL	1

EMPLEADOS

Dnum	Dlocalizacion
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

DEPTO_LOCALIZACION

Dnombre	Num_Dpto	Jefe_DNI	Jefe_fecha_comienzo
Investigación	5	333445555	22-05-1988
Administración	4	987654321	01-01-1995
Oficinas Centrales	1	888665555	19-06-1981

DEPARTAMENTOS

Emp_DNI	Pnum	Horas
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

TRABAJA_EN

Pnombre	Pnum	Plocalizacion	Dnum
ProductoX	1	Bellaire	5
ProductoY	2	Sugarland	5
ProductoZ	3	Houston	5
Digitalizacion	10	Stafford	4
Reorganización	20	Houston	1
Nuevos beneficios	30	Stafford	4

PROYECTOS

Emp_DNI	Nombre_Dependiente	Sexo	Fecha_nac
333445555	Alicia	F	05-04-1968
333445555	Teodoro	M	25-10-1983
333445555	Teresa	F	03-05-1958
987654321	Aner	M	28-02-1942
123456789	Miguel	M	04-01-1988
123456789	Alicia	F	30-12-1988
123456789	Isabel	F	05-05-1967

DEPENDEINTES

Cuadro 1.1

Este es un estado del esquema de base de datos relacional de la compañía, además es el ejemplo que vamos a emplear en futuros capítulos y sobre el cual realizaremos diferentes consultas.

Capítulo 2

El álgebra relacional

En una base de datos es importante conocer su estructura, cómo se organizan y almacenan los datos, pero además es necesario conocer cómo interactuar con dicha información.
En este capítulo nos adentraremos en el **álgebra relacional**, lenguaje formal del modelo relacional, el cual permite la manipulación de bases de datos, y es empleado en la elaboración y optimización de las respuestas de las consultas de los usuarios. Vamos a explicar cómo funcionan sus operaciones básicas.

El álgebra relacional está constituido principalmente por un conjunto de operaciones que actúan sobre los esquemas de relación de la base de datos.
Podemos dividir estas operaciones en dos grupos:

- Las operaciones específicas de bases de datos: **SELECT**, **PROJECT** y **JOIN**.
- Las operaciones definidas en la teoría matemática de conjuntos: **UNIÓN**, **INTERSECCIÓN**, **DIFERENCIA**, y **PRODUCTO CARTESIANO**. Estas operaciones son onsecuencia de considerar una relación como un conjunto de tuplas.

Cabe destacar que los resultados de las operaciones del álgebra relacional son nuevas expresiones de álgebra relacional, las cuales constituyen una nueva relación. Esto permite una gran variedad de posibilidades, ya que habilita la anidación y combinación de las diferentes operaciones.

2.1. Operaciones unarias

Las operaciones unarias son aquellas que actúan únicamente en una de las relaciones de la base de datos. En este grupo vamos a explicar las operaciones SELECT y PROJECT.

2.1.1. El operador SELECT

La operación SELECT escoge un subconjunto de tuplas de una relación que satisface las condiciones de selección. La labor se lleva a cabo comprobando cada tupla de manera individual si cumple o no dicha condición. La acción de este operador puede verse como una partición del conjunto de tuplas en dos subconjuntos: el conjunto de tuplas que satisfacen la condición y aquel que engloba a las tuplas que no satisfacen la condición. De manera formal la operación SELECT se define:

$$\sigma_{\text{<Condición de selección>}}(R)$$

Donde σ denota la operación, R es el esquema de relación en el que se aplica la acción, la cual es una expresión de álgebra relacional y cuyo resultado es una relación con los mismos atributos que R. La condición de selección es una expresión booleana formada por una comparación entre dos atributos de la relación o entre un atributo y una constante, los comparadores empleados son: $[=, <, \leq, >, \geq, \neq]$.

También es posible conectar más de una condición usando los operadores lógicos: $[\wedge, \vee, \neg]$

El grado de la relación resultante de la acción de SELECT es el mismo que el de la relación original R, además su número de tuplas será siempre menor o igual al número de R, es decir:

$$|\sigma c(R)| \leq |R|$$

Ejemplo 2.1. Para seleccionar los empleados que se encuentran en el departamento 4 podríamos hacerlo usando SELECT de la siguiente manera:

$$\sigma_{\text{Num\_Dpto = 4}}(\text{EMPLEADOS})$$

Ejemplo 2.2. Aquellos trabajadores con un salario mayor a 30.000€:

$$\sigma_{\text{Salario >30.000}}(\text{EMPLEADOS})$$

Ejemplo 2.3. Teniendo en cuenta que, al ejecutar la operación sobre una relación da como resultado también una relación, permite anidar varios SELECT, en este caso podemos unir ambas condiciones en una sola.

$$\sigma_{\text{Num\_Dpto = 4}}(\sigma_{\text{Salario >30.000}}(\text{EMPLEADOS})) = \sigma_{\text{(Num\_Dpto = 4) } \wedge \text{ (Salario >30.000)}}(\text{EMPLEADOS})$$

Notar también que SELECT es una operación conmutativa.

$$\sigma_{\text{Num\_Dpto = 4}}(\sigma_{\text{Salario >30.000}}(\text{EMPLEADOS})) = \sigma_{\text{Salario >30.000}}(\sigma_{\text{Num\_Dpto = 4}}(\text{EMPLEADOS}))$$

2.1.2. El operador PROJECT

La operación PROJECT actúa sobre una única relación escogiendo un grupo de atributos y eliminando el resto, manteniendo el número de tuplas. Es decir, este operador proyecta la relación sobre un grupo de atributos. Si se piensa en una relación como una tabla, la operación SELECT realiza una partición horizontal, mientras que la operación PROJECT ejecuta una partición vertical. La definición formal del operador es:

$$\pi_{\text{<Lista de atributos>}}(R)$$

Donde π representa el operador y la lista de atributos es el grupo sobre el cual queremos proyectar la relación. Del mismo modo que en el caso de la operación SELECT, R es en general una expresión de álgebra relacional, el caso más básico es simplemente un esquema de relación.
La relación resultado de la operación posee el mismo grado que número de atributos haya en la lista, ordenados del mismo modo que aparecen en ella, y su número de tuplas es menor o igual al número de tuplas en R.

Ejemplo 2.4. En nuestra relación ejemplo de empleados, listamos únicamente los atributos de nombre y salario de la relación.

$$\pi_{\text{Nombre, Salario}}(\text{EMPLEADOS})$$

Ejemplo 2.5. Se pueden combinar las operaciones de SELECT y PROJECT. Si queremos obtener el nombre y el salario de los empleados que trabajan en el departamento 4, podemos anidar SELECT y PROJECT de la siguiente manera:

$$\pi_{\text{Nombre, Salario}}(\sigma_{\text{Num\_Dpto = 4}}(\text{EMPLEADOS}))$$

Por otro lado, podemos mostrar explícitamente el orden de operaciones mediante relaciones intermedias y usando la **operación de asignación** $(\leftarrow)$.

$$\text{DEP4\_EMP} \leftarrow \sigma_{\text{Num\_Dpto = 4}}(\text{EMPLEADOS})$$
$$\text{RESULTADO} \leftarrow \pi_{\text{Nombre, Salario}}(\text{DEP4\_EMP})$$

En algunos casos podemos necesitar cambiar el nomre de algún atributo o incluso de una relación. En estas situaciones usamos la operación unaria **RENOMBRAR**, la cual se define formalmente como:

$$\rho_{S(B_1,B_2,...,B_n)}(R)$$

Donde ρ denota al operador, S es el nuevo nombre del esquema del relación, $B_1, B_2, ..., B_n$ la lista de los nuevos nombres de los atributos, y R el esquema de relación que queremos renombrar. Si los atributos de R son $A_1, ..., A_n$, el nombre del atributo A_i es sustituido por B_i con $i \in [1, n]$.

Si el nombre del esquema de relación no se quiere modificar, o ningún atributo cambia su nombre, se pueden omitir aquellos elementos que no vayan a ser alterados por la operación.

2.2. Operaciones binarias

Las operaciones binarias son aquellas que actúan sobre dos relaciones de la base de datos. En este grupo encontramos operaciones de la teoría de conjuntos: unión, intersección, diferencia de conjuntos y producto cartesiano. En el caso de las tres primeras es necesario que las dos relacionas sobre las que actúan sean **unión compatibles**.

Definición. Dos relaciones $R(A_1, ..., A_n)$ y $S(B_1, ..., B_n)$ son unión compatibles si tienen el mismo grado y $dom(A_i) = dom(B_i)$ con $i \in [1, n]$.

Es decir, cada relación tienen el mismo número de atributos y cada pareja de atributos tiene el mismo dominio.

2.2.1. El producto cartesiano

El producto cartesiano opera sobre dos relaciones combinando cada tupla de una relación con cada una de las tuplas de la otra relación, y además no es necesario que ambas relaciones sean unión compatibles. La operación se denota $\times$.

En general, definimos el producto cartesiano como una operación sobre dos relaciones $R(A_1, ..., A_n)$ y $S(B_1, ..., B_m)$, y cuyo resultado es una relación $Q(A_1, ..., A_n, B_1, ..., B_m)$ de grado $n + m$ donde cada una de las tuplas son una combinación de una tupla de R y otra tupla de S.
Sea n_r el número de tuplas de R, y análogamente n_s el número de tuplas de S, entonces el número de tuplas de Q será el producto de ambos:

$$|Q| = |R| \cdot |S| = n_r \cdot n_s$$

El producto cartesiano como operación no tiene un gran interés, es mucho más útil cuando va seguido de la operación SELECT.

Ejemplo 2.6. Si queremos obtener como respuesta una lista de los dependientes de cada empleada:

$$\begin{aligned}
\text{MUJERES_EMP} &\leftarrow \sigma_{\text{Sexo = 'F'}}(\text{EMPLEADOS}) \\
\text{NOMBRES_EMP} &\leftarrow \pi_{\text{Nombre, DNI}}(\text{MUJERES_EMP}) \\
\text{DEPENDIENTES_EMP} &\leftarrow \text{DEPENDIENTES} \times \text{NOMBRES_EMP} \\
\text{DEPENDIENTES_REALES} &\leftarrow \sigma_{\text{DNI=Emp_DNI}}(\text{DEPENDIENTES_EMP}) \\
\text{RESULTADO} &\leftarrow \pi_{\text{Nombre, Nombre_dependiente}}(\text{DEPENDIENTES_REALES})
\end{aligned}$$

En primer lugar se seleccionan las tuplas de las empleadas, y a continuación se proyectan únicamente sus nombres y DNI. Seguidamente, la relación *NOMBRES_EMP* que hemos obtenido es cruzada con la de *DEPENDIENTES*, pero la relación resultante no tiene mucho interés, por tanto después realizamos una selección para elegir únicamente aquellas tuplas en las que el número de DNI coincida en ambos casos. Finalmente se proyectan los atributos que queremos obtener como respuesta, que son el nombre de la empleada con el nombre de su dependiente.

2.2.2. La operación JOIN

La operación JOIN es una de las más importantes en el procesamiento de bases de datos relacionales, ya que permite manipular y trabajar con las interrelaciones que existen entre las diferentes relaciones. Se puede describir como un producto cartesiano entre dos relaciones seguido del operador SELECT. De manera formal el operador JOIN sobre dos relaciones $R(A_1,...,A_n)$ y $S(B_1,...,B_m)$ se define como:

$$R \bowtie_{\text{<Condición de unión>}} S$$

Donde $\bowtie$ representa el operador. El resultado es una relación Q con $n+m$ atributos, igual que si fuera el producto cartesiano, con la excepción de que las únicas tuplas que contiene son aquellas que satisfacen la condición de unión. Esta condición, en general, es de la forma:

$$< \text{Condición}_1 > \wedge < \text{Condición}_2 > \wedge ... \wedge < \text{Condición}_m >$$

Cada una de estas condiciones se expresa como: $A_i\ \theta\ B_j$ con A_i un atributo de R, B_j un atributo de S, $dom(A_i) = dom(B_j)$, y θ uno de los operadores $[=, <, \leq, >, \geq, \neq]$.
La operación JOIN deriva en una relación con un número de tuplas entre cero y $n_r \cdot n_s$ dependiendo del número de tuplas que satisfagan la condición de unión.En la práctica el uso más habitual del operador es con condiciones de unión de igualdad, en estos casos donde el único operador de comparación es $=$ la operación se denomina **EQUIJOIN**.

Ejemplo 2.7. Anteriormente se ha empleado un producto cartesiano entre las relaciones *DEPENDIENTES* y *NOMBRES_EMP*, y a continuación un SELECT para tomar únicamente las tuplas en las cuales *DNI* y *Emp_DNI* coincidiesen:

$$\text{DEPENDIENTES\_EMP} \leftarrow \text{DEPENDIENTES} \times \text{NOMBRES\_EMP}$$
$$\text{DEPENDIENTES\_REALES} \leftarrow \sigma_{\text{DNI=Emp\_DNI}}(\text{DEPENDIENTES\_EMP})$$

Podemos reemplazar estas dos operaciones con un solo JOIN, del siguiente modo:

$$\text{DEPENDIENTES\_REALES} \leftarrow \text{DEPENDIENTES} \bowtie_{\text{DNI = Emp\_DNI}} \text{NOMBRES\_EMP}$$

2.3. Ejemplos

Una vez presentados los operadores del álgebra relacional, para facilitar la comprensión de su funcionamiento y cómo interactúan entre sí, veamos algunos ejemplos:

Consulta 1. Obtener el nombre y la dirección de todos los empleados que trabajan en el departamento "Investigación".

$$\text{INVESTIGACION\_DEPTO} \leftarrow \sigma_{\text{Dnombre='Investigación'}}(\text{DEPARTAMENTOS})$$
$$\text{INVESTIGACION\_EMP} \leftarrow \text{INVESTIGACION\_DEPTO} \bowtie_{\text{Num\_Dpto=Num\_Dpto}} \text{EMPLEADOS}$$
$$\text{RESULTADO} \leftarrow \pi_{\text{Nombre, Direccion}}(\text{INVESTIGACION\_EMP})$$

Podriamos escribir la consulta en una sola línea:

$$\pi_{\text{Nombre, Direccion}}(\sigma_{\text{Dnombre='Investigación'}}(\text{DEPARTAMENTOS} \bowtie_{\text{Num\_Dpto=Num\_Dpto}} \text{EMPLEADOS})$$

Localizamos, en primer lugar, el departamento de investigación, después realizamos un join con la relación de los empleados para conocer cuáles de ellos trabajan en dicho departamento, terminamos listando los atributos pedidos.

Hay algunas ocasiones en las cuales los atributos que han de ser iguales en la condición de unión de un JOIN tienen el mismo nombre, en esos casos tenemos lo que se conoce como **NATURAL JOIN**, se denota por $(*)$, aunque también se puede representar con el operador JOIN sin una condición de unión.

$$\text{INVESTIGACION\_EMP} \leftarrow \text{INVESTIGACION\_DEPTO} * \text{EMPLEADOS}$$

Al efectuar esta consulta en el estado del esquema de base de datos del Cuadro 1.1 el resultado es:

Nombre	Direccion
Juan	731 Fondren, Houston
Fran	638 Voss, Houston
Ramón	975 Fire Oak, Humble
Julia	5631 Rice, Houston

Resultado Consulta 1.

Consulta 2. Para cada proyecto localizado en "Stafford", obtener el número del proyecto, su departamento correspondiente, y el nombre, direccion y fecha de nacimiento del jefe de departamento.

$$\begin{aligned}
\text{PROY_STAFFORD} &\leftarrow \sigma_{\text{Plocalizacion='Stafford'}}(\text{PROYECTOS}) \\
\text{DEPTO_CORRESP} &\leftarrow \text{PROY_STAFFORD} \bowtie_{\text{Dnum=Num_Dpto}} \text{DEPARTAMENTOS} \\
\text{PROY_DEPTO_JEFE} &\leftarrow \text{DEPTO_CORRESP} \bowtie_{\text{Jefe_DNI=Emp_DNI}} \text{EMPLEADOS} \\
\text{RESULTADO} &\leftarrow \pi_{\text{Pnum, Dnum, Nombre, Direccion, Fecha_nac}}(\text{PROY_DEPTO_JEFE}) \\
\text{RESULTADO_RENOM} &\leftarrow \rho_{\text{(Num_proyecto, Num_depto, Nombre_jefe, Direccion_jefe, Fecha_nac_jefe)}}(\text{RESULTADO})
\end{aligned}$$

Seleccionamos todos los proyectos de "Stafford", después los unimos con sus departamentos, luego con los jefes de departamento. Y finalmente listamos los atributos requeridos, renombrando en el resultado cada uno de ellos para una mejor comprensión. El resultado de esta consulta aplicado a la base de datos del Cuadro 1.1 es:

Num_proyecto	Num_depto	Nombre_jefe	Direccion_jefe	Fecha_nac_jefe
Digitalización	4	Fran	638 Voss, Houston	8-12-1955
Nuevos beneficios	4	Fran	638 Voss, Houston	8-12-1955

Resultado Consulta 2.

Consulta 3. Encontramos el número de los proyectos en los que trabajan alguien que se llame "Juan", ya sea un empleado o el jefe del proyecto.

$$\begin{aligned}
\text{JUANES} &\leftarrow \sigma_{\text{Nombre='Juan'}}(\text{EMPLEADOS}) \\
\text{JUAN_TRABAJA_PROY} &\leftarrow \pi_{\text{Pnum}}(\text{TRABAJA_EN} \bowtie_{\text{Emp_DNI=DNI}} \text{JUANES}) \\
\text{JEFES} &\leftarrow \pi_{\text{Nombre, Num_Dpto}}(\text{DEPARTAMENTOS} \bowtie_{\text{Jefe_DNI=Emp_DNI}} \text{EMPLEADOS}) \\
\text{JUAN_JEFE_DEPTO} &\leftarrow \sigma_{\text{Nombre='Juan'}}(\text{JEFES}) \\
\text{JUAN_JEFE_PROY} &\leftarrow \pi_{\text{Pnum}}(\text{JUAN_JEFE_DEPTO} \bowtie_{\text{Num_Dpto=Dnum}} \text{PROYECTOS}) \\
\text{RESULTADO} &\leftarrow \text{JUAN_TRABAJA_PROY} \cup \text{JUAN_JEFE_PROY}
\end{aligned}$$

En primer lugar obtenemos los empleados que se llaman Juan y seguidamente listamos los números de los proyectos en los que trabajan. Por otro lado obtenemos los jefes y sus números de departamentos, después nos quedamos con aquellos que se llaman Juan, y finalmente obtenemos los números de los proyectos que dirigen. Una vez que tenemos los números de los proyectos en los que hay un Juan trabajando y los que están dirigidos por alguien llamado Juan, concluimos uniendo ambos conjuntos de soluciones. Al aplicar esta consulta, nuevamente, al estado del esquema de base de datos del Cuadro 1.1 nos devuelve el siguiente resultado:

Pnum
1
2

Resultado Consulta 3.

Capítulo 3

Un método heurístico

Una vez presentados el modelo y el álgebra relacional disponemos de las herramientas necesarias para describir métodos que se pueden seguir para conseguir optimizar las respuestas a las consultas del lenguaje SQL. El término **optimización**, en este contexto, no significa encontrar la ejecución óptima de una consulta, sino que se trata de alcanzar, con la información disponible, una estrategia razonablemente eficiente en un tiempo razonable. Para calcular la mejor opción, en muchas ocasiones, se pierde gran cantidad de tiempo y esfuerzo, lo que la convierte en algo ineficiente.

Cuando un sistema de gestión de bases de datos recibe una consulta externa, sobre la información que posee, lleva a cabo los siguientes pasos para tratar de dar una respuesta de manera eficiente:

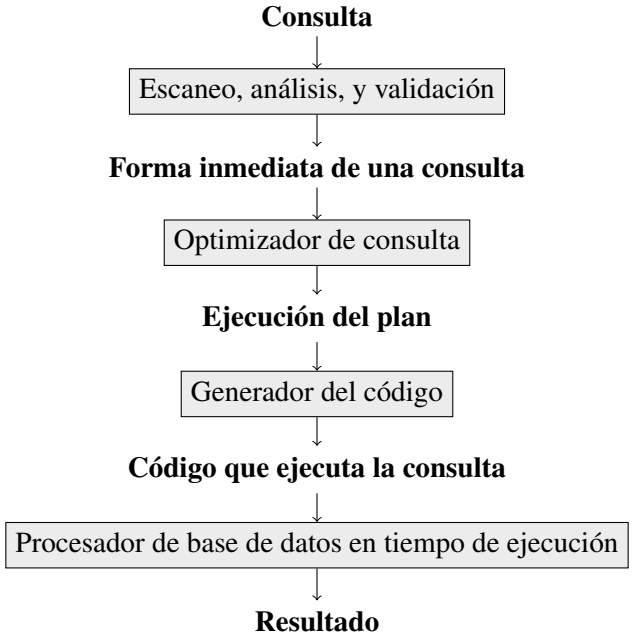

Figura 3.1: Pasos para procesar una consulta.

Cuando el sistema recibe una consulta SQL la escanea, analiza y valida que los nombres de los atributos y las relaciones son válidos en el esquema de la base de datos, después el optimizador de consulta elige la mejor estrategia de ejecución para la consulta. Seguidamente el generador de código del sistema procesa la estrategia y produce el código para ejecutarla. Y finalmente, el procesador de base de datos en tiempo de ejecución es el encargado de efectuar dicho código y obtener el resultado.

Este trabajo se enfoca en explicar qué ocurre en la optimización del plan de ejecución.

En primer lugar, una consulta SQL es traducida en una expresión de álgebra relacional, seguidamente

se representa como un **árbol de consulta** y finalmente se optimiza. Existen varias maneras de llevar a cabo tal proceso, una de las técnicas más importantes es tratar de estimar el coste de diferentes estrategias de ejecución de una consulta y elegir aquella que lo minimice.

En nuestro caso nos centraremos en desarrollar un **método heurístico**, el cual se basa en diferentes reglas que ordenan las operaciones dentro de la estrategia de ejecución de las consultas, con el fin de minimizar el tiempo de respuesta.

3.1. Consultas SQL a álgebra relacional

El lenguaje SQL se traduce en expresiones de álgebra relacional de la siguiente manera:

- En la cláusula *WHERE* de una consulta SQL se especifica la operación SELECT, por ejemplo la consulta:

  SELECT $*$
  FROM EMPLEADOS
  WHERE Num_Dpto = 4 **AND** Salario $>$ 30.000 ;

  Se traduce en la expresión: $\sigma_{(\text{Num\_Dpto} = 4) \wedge (\text{Salario} > 30.000)}(\text{EMPLEADOS})$.

- En la cláusula *SELECT* se especifica la operación PROJECT, por ejemplo, si queremos como respuesta de la anterior consulta únicamente los nombres de los empleados la consulta SQL es:

  SELECT Nombre
  FROM EMPLEADOS
  WHERE Num_Dpto = 4 **AND** Salario $>$ 30.000 ;

  Esta consulta se traduce en álgebra relacional de la siguiente manera:

  $$\pi_{\text{Nombre}}(\sigma_{(\text{Num\_Dpto} = 4) \wedge (\text{Salario} > 30.000)}(\text{EMPLEADOS}))$$

- En SQL se emplea la cláusula *JOIN* dentro de la cláusula *FROM* para cruzar la información de dos relaciones, por ejemplo:

  SELECT $*$
  FROM (DEPENDIENTES **JOIN** NOMBRES_EMP **ON** DNI = Emp_DNI);

  Otra manera es especificar la condición de unión en la cláusula *WHERE*:

  SELECT $*$
  FROM DEPENDIENTES, NOMBRES_EMP
  WHERE DNI = Emp_DNI;

  Si en la cláusula *WHERE* no se especifica ninguna condición el resultado que obtenemos es el producto cartesiano de ambas relaciones.

  Estas dos consultas de SQL se traducen en la misma expresión de álgebra relacional, la cual es:

  $$\text{DEPENDIENTES} \bowtie_{\text{DNI} = \text{Emp\_DNI}} \text{NOMBRES\_EMP}$$

- Las cláusulas *UNION, INTERSECT* y *EXCEPT* corresponden a las operaciones de unión, intersección y diferencia de conjuntos, las cuales están definidas en SQL tal y como las conocemos.

Normalmente las consultas SQL están formadas por varios bloques, son consultas anidadas, por tanto se debe traducir cada bloque por separado y optimizar cada uno de ellos.

Ejemplo 3.1. Consideremos la siguiente consulta sobre la relación *EMPLEADOS*, la cual obtiene el nombre y DNI de los empleados que cobren más que Juan:

```
SELECT Nombre, DNI
FROM EMPLEADOS
WHERE Salario > (SELECT Salario
                 FROM EMPLEADOS
                 WHERE Nombre = 'Juan');
```

Luego, los dos bloques a optimizar son:

```
SELECT Salario
FROM EMPLEADOS
WHERE Nombre = 'Juan';
```

Este bloque devuelve el salario de Juan.

```
SELECT Nombre, DNI
FROM EMPLEADOS
WHERE Salario > c;
```

Donde c representa el resultado del primer bloque, el salario de Juan. Las expresiones de álgebra relacional correspondientes al primer bloque y al segundo, respectivamente son:

$$\pi_{\text{Salario}}(\sigma_{\text{Nombre='Juan'}}(\text{EMPLEADOS}))$$

$$\pi_{\text{Nombre, DNI}}(\sigma_{\text{Salario}>c}(\text{EMPLEADOS}))$$

A continuación, cada bloque debe ser evaluado y optimizado por separado. Notar que el primer bloque solo debe ser evaluado una vez, el cual será usado en la optimización del segundo bloque como una constante.

3.2. Árboles de consulta

Durante el proceso de optimización, una herramienta que permite representar, analizar y transformar las consultas SQL mediante la descomposición de las expresiones algebraicas en operaciones básicas son los **árboles de consulta**. En el procedimiento que se sigue para dar respuestas a una consulta relacional (Figura 3.1), los árboles de consulta son empleados por el optimizador de consulta en la elaboración del plan de ejecución.

Definición. Un árbol de consulta es una representación gráfica y estructurada de una expresión de álgebra relacional.

En un árbol de consulta las relaciones de entrada son representados como **nodos hoja**, y las diferentes operaciones de la expresión algebráica son denotadas como **nodos internos**. Las operaciones de los nodos se ejecutan cuando sus operandos están disponibles y una vez ejecutada la operación el nodo es sustituido por su resultado. La ejecución de los árboles de consulta comienza por los nodos hoja y termina cuando se ejecuta el **nodo raíz**, el cual representa la última operación de la consulta y su aplicación da el resultado final.

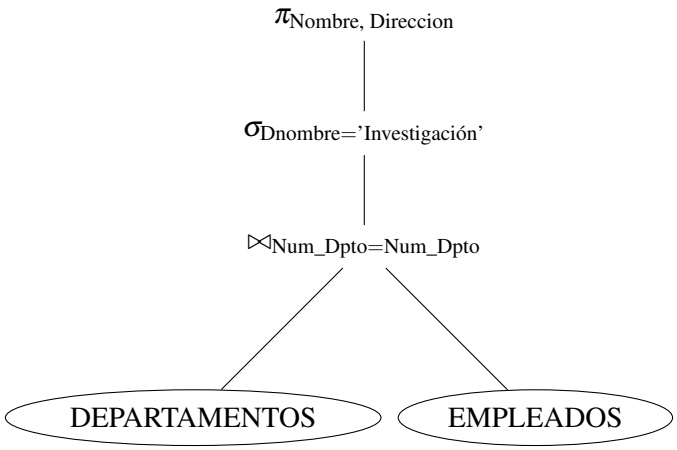

Figura 3.2: Árbol de Consulta 1.

Ejemplo 3.2. Veamos cómo es el árbol de consulta de una expresión de álgebra relacional. Por ejemplo, tomamos la expresión de la Consulta 1, con la que queremos obtener el nombre y dirección de todos los empleados que trabajan en el departamento "Investigación".

$$\pi_{\text{Nombre, Direccion}}(\sigma_{\text{Dnombre='Investigación'}}(\text{DEPARTAMENTOS} \bowtie_{\text{Num\_Dpto=Num\_Dpto}} \text{EMPLEADOS})$$

El árbol de consulta correspondiente es:

La primera operación en ser ejecutada en este árbol de consulta es la operación JOIN entre las relaciones *DEPARTMENTOS* y *EMPLEADOS*, después se aplica la operación SELECT y finalmente se ejecuta un PROJECT. De manera informal, las operaciones en un árbol de consulta se ejecutan de abajo hacia arriba.

3.3. Reglas y algoritmo de transformación

En la optimización de las consultas nuestro interés se centra en transformar las expresiones de álgebra relacional en otras equivalentes, cuyo resultado contenga la misma información que la expresión original, aunque pueda variar el orden de los atributos.

Presentamos en esta sección una serie de reglas para transformar estas expresiones de álgebra relacional en otras equivalentes.

1. **Secuencia de σ.**

$$\sigma_{c_1 \wedge c_2 \wedge \ldots \wedge c_n}(R) = \sigma_{c_1}(\sigma_{c_2}(\ldots(\sigma_{c_n}(R))\ldots))$$

Demostración. Lo probamos por doble contenido.

Sea $x \in \sigma_{c_1 \wedge c_2 \wedge \ldots \wedge c_n}(R)$, entonces el elemento x cumple todas las condiciones $c_1, \ldots, c_n$. Obviamente $x \in \sigma_{c_n}(R)$, pero x también cumple c_{n-1}, luego $x \in \sigma_{c_{n-1}}(\sigma_{c_n}(R))$, repitiendo este razonamiento llegamos a que $x \in \sigma_{c_1}(\sigma_{c_2}(\ldots(\sigma_{c_n}(R))\ldots))$

Ahora, sea $x \in \sigma_{c_1}(\sigma_{c_2}(\ldots(\sigma_{c_n}(R))\ldots))$, luego $x \in \sigma_{c_2}(\ldots(\sigma_{c_n}(R))\ldots)$ y cumple la condición c_1, por lo tanto $x \in \sigma_{c_3}(\ldots(\sigma_{c_n}(R))\ldots)$ y cumple las condiciones c_1, c_2. Reiterando dicha lógica llegamos a que $x \in R$ y cumple todas las condiciones $c_1, c_2, \ldots, c_n$, es decir, $x \in \sigma_{c_1 \wedge c_2 \wedge \ldots \wedge c_n}(R)$. □

2. **Conmutatividad de σ.**

$$\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$$

Demostración. Siendo que el operador lógico de conjunción es conmutativo, por la *Regla 1*:

$$\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_1 \wedge c_2}(R) \equiv \sigma_{c_2 \wedge c_1}(R) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$$

□

3. **Secuencia de π.**

   Sean $Lista_1 \subseteq Lista_2 \subseteq ... \subseteq Lista_n$. Entonces:

   $$\pi_{Lista_1}(\pi_{Lista_2}(...(\pi_{Lista_n}(R))...) \equiv \pi_{Lista_1}(R)$$

   Demostración. Es fácil ver que el resultado de la operación PROJECT proyectando los atributos $Lista_1$ será igual con independencia de los atributos que contenga el esquema de relación sobre el que actúa, siempre que éste contenga al menos los atributos que contiene la lista de dicha operación. □

4. **Conmutar σ con π.**
   Si la condición de unión c involucra únicamente los atributos $A_1, A_2, ..., A_n$, entonces ambas operaciones conmutan de la forma:

   $$\pi_{A_1,A_2,...,A_n}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1,A_2,...,A_n}(R))$$

   Demostración. Veamos en primer lugar que: $\pi_{A_1}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1}(R))$

   Las tuplas que contiene $\sigma_c(R)$ son todas aquellas contenidas en la relación R que cumplen la condición de selección c. Por lo tanto, el conjunto $\pi_{A_1}(\sigma_c(R))$ contiene únicamente el atributo A_1 de todas las tuplas de R que satisfacen c.

   Por otro lado, $\pi_{A_1}(R)$ proyecta solamente el atributo A_1 de R, si aplicamos la operación SELECT el resultado es $\sigma_c(\pi_{A_1}(R))$, un conjunto que contiene únicamente el atributo A_1 de todas las tuplas de R que satisfacen c.

   Ahora, haciendo uso de lo probado:

   $$\pi_{A_1,A_2,...,A_n}(\sigma_c(R)) \equiv \pi_{A_1,A_2,...,A_{n-1}}(\pi_{A_n}(\sigma_c(R))) \equiv \pi_{A_1,A_2,...,A_{n-1}}(\sigma_c(\pi_{A_n}(R)))$$

   Repetimos el proceso con el resto de atributos $A_1, ..., A_{n-1}$, y llegamos finalmente a:

   $$\pi_{A_1,A_2,...,A_n}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1,A_2,...,A_n}(R))$$

   □

5. **Convertir una secuencia de σ y $\times$ en un $\bowtie$.**

   $$\sigma_c(R \times S) \equiv R \bowtie_c S$$

   Demostración. Por la definición de la operación JOIN se da la igualdad. Ambos conjuntos contienen los elementos del producto cartesiano entre ambas relaciones que cumplen la condición de unión c. □

6. **Conmutatividad de $\bowtie$.**

   $$R \bowtie_c S \equiv S \bowtie_c R$$

   Notar que el orden de los atributos en la relación resultado podría no ser el mismo que en las relaciones originales. Análogo para el caso del producto cartesiano.

Demostración. Teniendo en cuenta la conmutatividad del producto cartesiano, salvo reordenación de los atributos:

$$R \bowtie_c S \equiv \sigma_c(R \times S) \equiv \sigma_c(S \times R) \equiv S \bowtie_c R$$

□

7. **Conmutación de σ con $\bowtie$.**

$$\sigma_c(R \bowtie S) \equiv (\sigma_c(R)) \bowtie S$$

Análogo para el caso del producto cartesiano.

Demostración. Si $x \in \sigma_c(R \bowtie S)$, entonces $x \in R \bowtie S$ y además cumple c. Ahora, si $x \in (\sigma_c(R))$ entonces $x \in R$ y cumple la condición c, luego si se ejecuta una operación JOIN entre este conjunto y la relación S tenemos que sus elementos pertenecerán a R y S, y además cumplirán la condición c. □

8. **Conmutar π con $\bowtie$.**
Sea L una lista de atributos $L = A_1, A_2, ..., A_n, B_1, B_2, ..., B_m$, donde $A_1, A_2, ..., A_n$ son atributos de una relación R y $B_1, B_2, ..., B_m$ atributos de una realación S, entonces si la condición de unión c solo involucra atributos de L, las operaciones conmutan de la siguiente manera:

$$\pi_L(R \bowtie_c S) \equiv (\pi_{A_1, A_2, ..., A_n}(R)) \bowtie_c (\pi_{B_1, B_2, ..., B_m}(S))$$

Demostración.

$$\pi_L(R \bowtie_c S) \equiv \pi_L(\sigma_c(R \times S)) \equiv \pi_L(\sigma_c(R) \times \sigma_c(S))$$

Teniendo en cuenta ahora que la operación PROJECT es distributiva respecto del producto cartesiano, ya que no modifica la estructura de la relación. Se tiene entonces:

$$\pi_L(\sigma_c(R) \times \sigma_c(S)) \equiv \pi_L(\sigma_c(R)) \times \pi_L(\sigma_c(S))$$

Sea $A = A_1, A_2, ..., A_n$ y $B = B_1, B_2, ..., B_m$, entonces $L = A \cup B$ la operación PROJECT sobre la relación R de los atributos de L será igual que si proyecta el conjunto A, ya que los atributos de B no pertenecen al esquema de relación de R, análogamente sobre la relación S únicamente se proyectan los atributos contenidos en la lista B.

Además, aplicando la *Regla 4* que implica la conmutación de PROJECT y SELECT, se tiene:

$$\pi_L(\sigma_c(R)) \times \pi_L(\sigma_c(S)) \equiv \sigma_c(\pi_A(R)) \times \sigma_c(\pi_B(R)) \equiv \sigma_c(\pi_A(R) \times \pi_B(S))$$

Finalmente, aplicando la *Regla 5* tenemos la equivalencia buscada. □

9. **Asociatividad de $\bowtie$, $\times$, $\cup$, $\cap$.**
Sea θ una operación de las cuatro cualquiera, entonces:

$$(R \ \theta \ S) \ \theta \ T \equiv R \ \theta \ (S \ \theta \ T)$$

Demostración. Se sigue de la propia definición de las operaciones, teniendo en cuenta que la operación JOIN se fundamenta en el producto cartesiano, y ésta es una operación que cumple la propiedad asociativa. □

10. **Conmutar σ con las operaciones de conjuntos.**
La operación SELECT conmuta con la unión, intersección y la diferencia de conjuntos. Sea θ una operación cualquiera de las tres, entonces:

$$\sigma_c(R \ \theta \ S) \equiv (\sigma_c(R)) \ \theta \ (\sigma_c(S))$$

11. **Conmutación de π con $\cup$.**

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

Demostración. Como la operación PROJECT no repercute en la estructura de la relación no importa su lugar en el orden de ejecución de las operaciones. □

12. **Distributividad de σ con la diferencia de conjuntos.**

$$\sigma_c(R - S) = \sigma_c(R) - \sigma_c(S)$$

Pero esta regla podría verse aplicando únicamente la operación SELCT sobre el esquema de relación R:

$$\sigma_c(R - S) = \sigma_c(R) - S$$

13. **La operación σ sobre solo uno de los argumentos de la intersección.**
Si los atributos de la condición de selección c pertenecen al esquema de relación R, entonces:

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

Demostración. Como los atributos que involucra c sólo pertenecen al esquema de relación de R no tiene sentido la operación SELECT con la condición c sobre la relación S.

Si $x \in \sigma_c(R \cap S)$ entonces $x \in R \cap S$, y además cumple c. Por otro lado, si $x \in \sigma_c(R)$ se tiene que $x \in R$ y cumple c, y por lo tanto, sea $x \in \sigma_c(R) \cap S$ entonces además pertenece a S, luego tenemos $x \in R \cap S$ y cumple la condición de unión c. Dándose así la igualdad. □

14. **Otras transformaciones triviales.**
Si S no contiene ningún elemento, entonces: $R \cup S = R \cup \emptyset = R$.
Si la condición de selección c se cumple en todas las tuplas de la realción R, entonces: $\sigma_c(R) = R$.

3.3.1. Optimización heurística

Una vez presentado el álgebra relacional, los árboles de consulta y las reglas para la transformación equivalente de expresiones de álgebra relacional, estamos en disposición de describir un algoritmo que modifique un árbol de consulta inicial y lo convierta en uno más eficiente. Los pasos son los siguientes:

1. Convertir cada operación SELECT que posea una condición de selección conjuntiva en una secuencia anidada de operaciones SELECT, *Regla 1*.

2. Usando las transformaciones de las *Reglas 2, 4, 7* y las *Reglas 10, 12, 13* mover cada operación SELECT lo más abajo posible del árbol de consulta. Por ejemplo, si la condición de selección de SELECT posee atributos de una sola relación, dicha operación se situa inmediatamente después del nodo que representa la relación. Si, por el contrario, la condición involucra dos relaciones se situa justo después de la operación JOIN que involucre ambas.

3. Empleando las *Reglas 6, 10* reordenar los nodos hoja del árbol de consulta, con el objetivo de ejecutar en primer lugar las operaciones SELECT más restrictivas, es decir, aquellas cuyo resultado posea el menor número de tuplas.

   Al realizar este reajuste se debe asegurar que no cause un producto cartesiano entre las diferentes relaciones, es decir, si las dos relaciones que poseen una condición de selección más restrictiva no tienen una operación SELECT con una condición de unión entre ellas, lo mejor es cambiar el orden de los nodos hoja para evitar un producto cartesiano entre ellas.

4. Reemplazar cada producto cartesiano seguido de una operación SELECT por una operación JOIN, si la condición del SELECT representa una condición de unión, *Regla 5*.

5. Usando las *Reglas 3, 4, 8, 11* mover cada operación PROJECT los más abajo posible en el árbol de consulta, además solo se deben proyectar aquellos atributos que aparecen en el resultado de la consulta, y aquellos que se necesiten para ejecutar las operaciones intermedias.

6. Identificar posibles subárboles de consulta que puedan ser ejecutados por un único algoritmo.

En resumen, el objetivo del algoritmo es conseguir ejecutar en primer lugar aquellas operaciones que reduzcan el tamaño de las relaciones intermedias, reordenando las operaciones SELECT y PROJECT. Después se intenta evitar cualquier producto cartesiano reordenando los nodos hoja y ajustando el resto del árbol de consulta.

3.3.2. Optimización de árboles de consulta

Una consulta de una base de datos puede ser descrita de varias maneras como expresión de álgebra relacional, ya que ésta no es única. Cuando se analiza y transforma una consulta SQL, el árbol de consulta inicial que se obtiene no está optimizado. El método heurístico para optimizar una consulta transforma este árbol de consulta inicial en uno equivalente para que su ejecución sea más eficiente.

Esta optimización se lleva a cabo siguiendo los pasos del algoritmo que se acaba de presentar y empleando las reglas de equivalencia ya descritas. Para explicar mejor cómo se optimiza un árbol de consulta lo haremos con un ejemplo.

Ejemplo 3.3. Queremos conocer el nombre de los empleados nacidos después de 1957 y que trabajan en el proyecto “Digitalizacion”. El código SQL de esta consulta es:

```
SELECT E.Nombre
FROM EMPLEADOS E, TRABAJA_EN T, PROYECTOS P
WHERE P.Pnombre = 'Digitalizacion'  AND  P.Pnum = T.Pnum  AND
    E.DNI = T.Emp_DNI AND  E.Fecha_nac > '31-12-1957'
```

Tener en cuenta que en SQL, cuando se emplea más de una relación y debemos distinguir sus atributos, se denota cada relación por una inicial y se escribe delante de cada atributo la relación a la que pertenece seguida de un punto. El árbol de consulta inicial correspondiente al código de la consulta es:

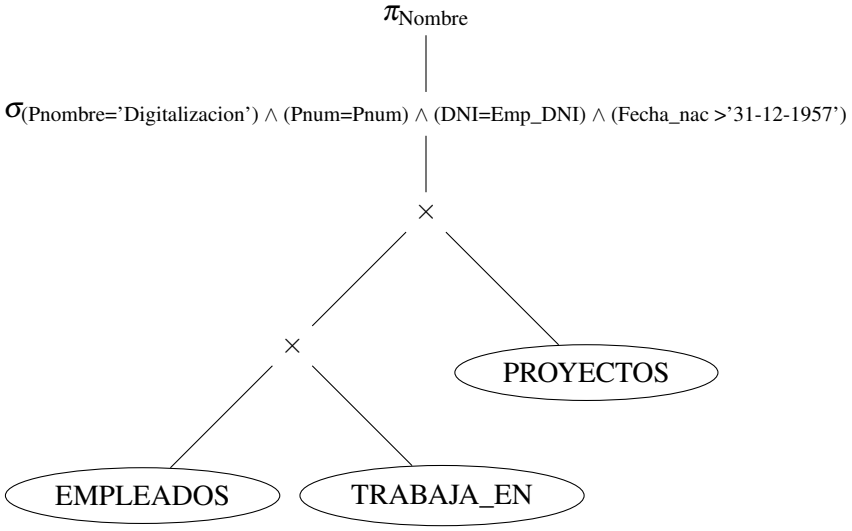

Figura 3.3

Si este primer árbol fuera ejecutado el resultado ocuparía mucho espacio al tratarse del producto cartesiano de las tres relaciones. Por tanto, este árbol debe ser transformado en otro equivalente.

Para reducir el tamaño de las relaciones, antes de ejecutar el producto cartesiano, se aplica la operación SELECT en cada relación, es decir desplazamos las operaciones SELECT lo más abajo posible en el árbol. En nuestro caso queremos que la operación SELECT seleccione la tupla de la relación *PROYECTOS* correspondiente al proyecto "Digitalizacion", y filtre únicamente los empleados nacidos después de 1957 de la relación *EMPLEADOS*.

Corresponde con el *Paso 1* del algoritmo, en el cual hemos empleado la *Regla 1* para separar un único SELECT con una condición de selección conjuntiva en una secuencia de operaciones. Además, en este mismo árbol de consulta se ha aplicado también el *Paso 2* usando la *Regla 7* para mover cada operación SELECT lo más abajo posible del árbol de consulta.

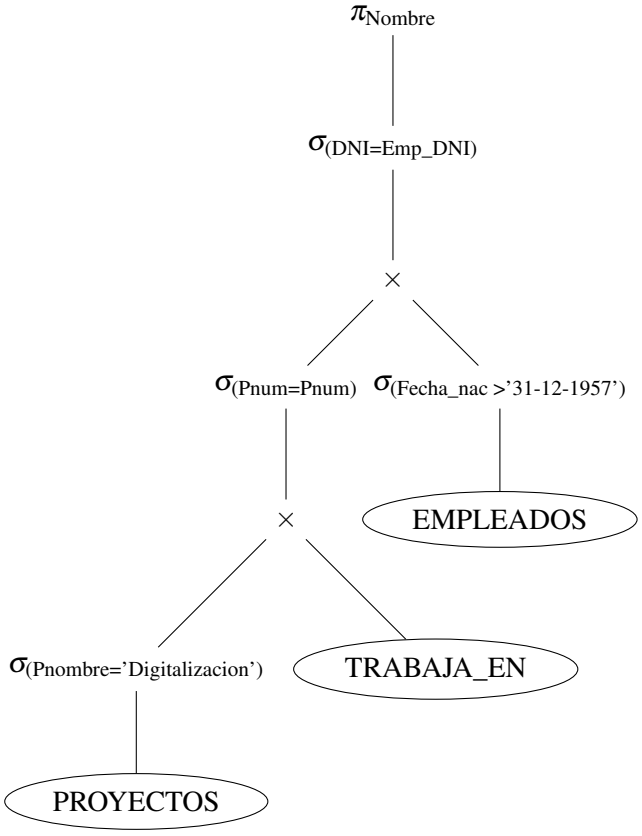

Figura 3.4

La siguiente mejora que se puede realizar es reemplazar el producto cartesiano y la condición de unión que le sigue en cada caso por la operación JOIN. Aplicamos el *Paso 4* del algoritmo en el cual la *Regla 5* permite transformar el producto cartesiano, entre la relación *TRABAJA_EN* y el resultado de seleccionar el proyecto "Digitalizacion" de la relación *PROYECTOS*, seguido de una operación SELECT, en la cual se seleccionan las tuplas en las que coinciden el número de proyecto. Se convierte en una operación JOIN entre ambas relaciones.

Aplicando estas ideas al árbol de consulta de la Figura 3.4:

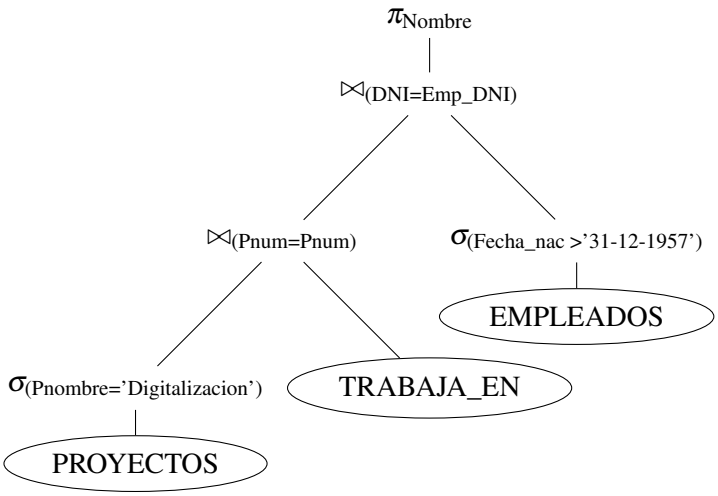

Figura 3.5

Un último avance en la eficiencia de la ejecución del árbol de consulta es aplicar lo antes posible la operación PROJECT, se consigue así que las relaciones intermedias posean el menor número de atributos posible.

En el ejemplo, antes de realizar ninguna operación JOIN que involucre más de una relación se proyectan únicamente los atributos, de cada relación que participa en la consulta, necesarios para realizar las siguientes operaciones. Es decir, de la relación *PROYECTOS* se necesita el atributo *Pnum* para la posterior operación JOIN, de la relación *TRABAJA_EN* se proyectan los atributos *Pnum* y *Emp_DNI* para la primera y la segunda operación JOIN respectivamente, y de la relación *EMPLEADOS* se proyecta el atributo *DNI* para poder realizar la operación JOIN y el atributo *Nombre* que se pide como resultado de la consulta. El árbol de consulta resultante de este paso es:

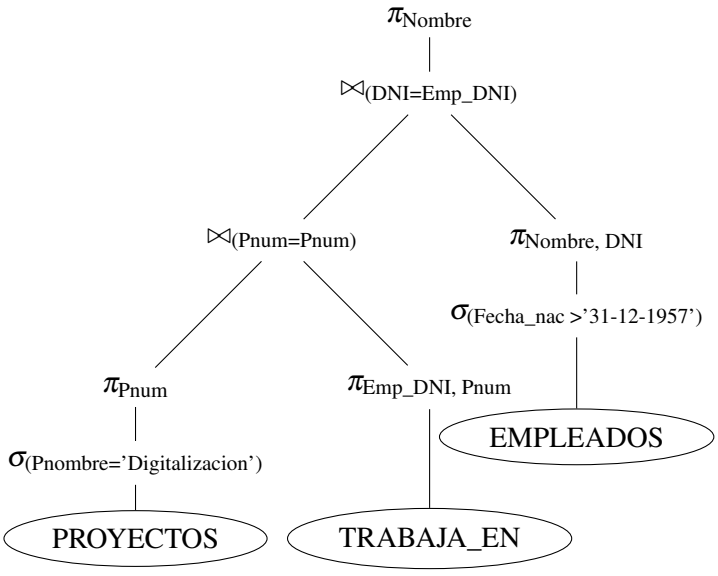

Figura 3.6

Se ha ejecutado el *Paso 5* del algoritmo heurístico, la *Regla 8* permite conmutar la operación PROJECT con la operación JOIN. Por otro lado, la *Regla 3* se emplea para proyectar únicamente el atributo necesario en el resultado. Tal y como acabamos de demostrar, un árbol de consulta puede ser optimizado paso a paso siendo transformado en un árbol de consulta equivalente. A pesar de todo, se debe asegurar que las transformaciones dan como resultado un árbol equivalente y, por lo tanto, el optimizador debe conocer las reglas que preserven dicha equivalencia.

Capítulo 4

Optimización basada en el coste

La optimización de consultas relacionales no sólo involucra las técnicas heurísticas, las cuales hemos visto que se basan en la aplicación de una serie de reglas que llevan a una mejora en la eficiencia de respuesta de las consultas al simplificar y transformar el árbol de consulta, sino que también realiza estimaciones sobre el coste de ejecución de diferentes maneras y lleva a cabo aquella con el menor coste estimado, conocida como **optimización basada en el coste**.

La optimización, usando técnicas tradicionales, estima una función objetivo del coste, tratando de minimizarlo y elegir la ejecución más rápida y eficiente. Normalmente esta optimización se lleva a cabo a partir de un árbol de consulta previamente optimizado heurísticamente.

Estas funciones de coste son aproximaciones, por lo que la estrategia de ejecución que resulta de la optimización no es la óptima del problema, ya que su búsqueda podría llevar mucho tiempo al tener que calcular el coste de ejecución de cada posible estrategia.

4.1. Determinantes del coste

Los diferentes componentes de los que depende el coste de ejecución de una consulta relacional son los siguientes:

- **Costo de acceso al almacenamiento secundario**. Se refiere al coste de leer y escribir bloques de datos entre el almacenamiento en disco secundario y la memoria principal. El coste de buscar registros en un archivo de disco depende del tipo de estructuras de acceso en ese archivo, como el ordenamiento e índices primarios o secundarios.

- **Coste de almacenamiento en disco**. Es el coste de almacenar en disco cualquier archivo intermedio que se necesite para la ejecución.

- **Coste de computación**. Se trata del coste de realizar operaciones en memoria RAM sobre los registros durante la ejecución de la consulta.

  Tales operaciones incluyen la búsqueda y ordenamiento de registros, la combinación de registros para una operación de unión o de ordenamiento, y la realización de cálculos sobre los valores de las columnas.

- **Coste de memoria**. Es el coste referido a la cantidad de memoria principal necesaria durante la ejecución de la consulta.

- **Coste de comunicación**. El coste de enviar la consulta y sus resultados desde el sitio de la base de datos al sitio o terminal donde se originó la consulta.

Principalmente, en grandes bases de datos, la optimización se enfoca en reducir el coste de acceso al almacenamiento secundario. Por otro lado, el optimizador de consulta necesita conocer algunos datos sobre los contenidos de las relaciones para poder realizar una estrategia adecuada:

- El número de tuplas en una relación, es decir, su cardinalidad: $|R|$.
- La longitud media de una tupla en la relación.
- El número de bloques que ocupa la relación en el disco, se denota b_r.
- El número de tuplas por bloque, conocido como factor de bloque (*bfr*).
- El número de valores distintos de un atributo A en la relación R, se denota (*NDV(A,R)*). También son importantes el valor máximo y el mínimo, *max(A,R)* y *min(A,R)*.
- La selectividad (*sl*), que es la fracción de tuplas que cumplen la condición de selección de un atributo.

  Y la cardinalidad de selección ($sc = sl * |R|$), que es el número medio de tuplas que satisfacen una condición de selección.
- La selectividad de una operación JOIN se refiere a la relación entre el número de tuplas que contiene el resultado de la operación y el producto cartesiano entre ambas relaciones, es decir:

$$js = |(R \bowtie_c S)|/|(R \times S)| = |(R \bowtie_c S)|/(|R| * |S|)$$

  La estimación del número de tuplas del resultado de una operación JOIN es la cardinalidad de la unión: $jc = js * |R| * |S|$.
- La existencia de **índices**. Un índice es una estructura de datos utilizada para mejorar la velocidad de recuperación de registros y que permitien un acceso rápido a las filas de una tabla basándose en los valores de una o más columnas. Hay varios tipos:
  - **Primario**: índice creado en la clave primaria[1] de la relación.
  - **Secundario**: índice creado en uno o más atributos que no son la clave primaria.
- Información sobre los índices. El número de niveles (x) de cada índice multinivel es necesario para las funciones de costo que estiman el número de accesos a bloques durante la ejecución de consultas.

4.2. Funciones de coste

Para llevar a cabo una operación SELECT o JOIN se pueden usar diferentes algoritmos para encontrar los registros de la base de datos que satisfacen la condición de selección, dependiendo de la información que dispongamos. Para llevar a cabo la optimización de la consulta se estima el coste de cada algoritmo para tratar de elegir la mejor solución. Presentamos algunos de estos algoritmos:

4.2.1. Operación SELECT

- **Búsqueda lineal**. Comprobar fila por fila cuales de ellas satisfacen la condición. La estimación del coste de ejecución es igual al número de bloques que ocupe la relación, es decir: $Coste = b_r$.

[1] Una clave primaria es un conjunto de uno o más atributos de una relación de la base de datos que identifica de manera única cada tupla.

4.2.2. Operación JOIN

- **Bucle anidado**. Es el algoritmo por defecto. Si la operación es entre dos relaciones, por ejemplo R y S, evalúa en un bucle externo cada registro t de R con cada registro s de S y comprueba cuando se satisface la condición de unión de la operación JOIN $t[A] = s[B]$, siendo A y B atributos de las relaciones R y S respectivamente.

  Supongamos que tomamos R en el bucle externo, entonces tenemos la siguiente función de coste que estima el número de accesos al bloque:

  $$Coste = b_R + (b_R * b_S) + ((js * |R| * |S|)/bfr_{RS}) = b_R + (b_R * b_S) + (jc/bfr_{RS})$$

  El primer término de la fórmula, b_r, indica el coste de leer los bloques de la relación R. El producto $b_r * b_s$ indica el coste de leer los bloques de la relación S, ya que en un bucle anidado evalúa cada fila de R en cada fila de S. Finalmente, el último término de la función representa el coste de escribir el resultado en la memoria, estima el número de bloques que ocupa.

- **Bucle anidado basado en índice**. Si existe un índice para uno de los dos atributos de la condición del JOIN, por ejemplo el atributo B de la relación S y cuyo nivel de índice es x_B, recupera con un bucle cada registro t de R y luego usa el índice para recuperar directamente todos los registros coincidentes s de S que satisfacen $s[B] = t[A]$.

  El coste depende del tipo del índice, para un índice primario:

  $$Coste = b_R + (|R| * (x_B + 1)) + ((js * |R| * |S|)/bfr_{RS})$$

  Para un índice secundario, donde s_B es la cardinalidad de selección del atributo B de la relación S:

  $$Coste = b_R + (|R| * (x_B + 1 + sc_B)) + ((js * |R| * |S|)/bfr_{RS})$$

4.3. Ordenación de JOIN

Cuando hay más de dos operaciones JOIN, el orden de ejecución es relevante para optimizar el coste. En una operación JOIN entre n relaciones, éstas pueden ser ordenadas de $n!$ formas distintas, pero no todas son evaluadas para después elegir la mejor de todas sino que, si por ejemplo tenemos cinco relaciones, se estima la mejor manera para las tres primeras y después se estima el mejor orden entre las dos relaciones restantes y el resultado de la primera estimación:

$$r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4 \bowtie r_5 = (r_1 \bowtie r_2 \bowtie r_3) \bowtie r_4 \bowtie r_5 = r_{Resultado} \bowtie r_4 \bowtie r_5$$

Dando un resultado de $3! + 3! = 12$ maneras evaluadas, y no de $5! = 120$ formas distintas.

Otro aspecto importante a tener en cuenta es el tamaño de las relaciones involucradas, si por ejemplo la operación JOIN involucra dos relaciones con 100 tuplas cada una y otra con 20, es preferible que la primera operación JOIN contenga a la relación con menos registros y la relación intermedia generada sea del menor tamaño posible.

4.4. Ejemplo de optimización basada en el coste

En esta sección se explica un ejemplo para mostrar de manera más clara cómo funciona esta optimización. Supongamos que se tiene la siguiente consulta SQL en la que se quiere obtener una serie de información sobre los proyectos localizados en Stafford.

```
SELECT Pnum, Num_Dpto, Nombre, Direccion, Fecha_nac
FROM PROYECTOS, DEPARTAMENTOS, EMPLEADOS
WHERE Dnum = Num_Dpto AND Jefe_DNI = DNI AND Plocalizacion = ' Stafford '
```

El esquema de base de datos sobre el que se realiza este ejemplo es el mismo empleado durante todo el trabajo (Cuadro 1.1), pero no así las tuplas de las diferentes relaciones mostradas en el primer capítulo, sólo se conoce que existen diez proyectos localizados en Stafford y la información sobre las tres relaciones involucradas mostrada en el cuadro siguiente:

Nombre Tabla	Nombre Columna	Distintos	Valor_Mín	Valor_Máx
PROYECTOS	Plocalizacion	200	1	200
PROYECTOS	Pnum	2000	1	2000
PROYECTOS	Dnum	50	1	50
DEPARTAMENTOS	Num_Dpto	50	1	50
DEPARTAMENTOS	Jefe_DNI	50	1	50
EMPLEADOS	DNI	10 000	1	10 000
EMPLEADOS	Num_Dpto	50	1	50
EMPLEADOS	Salario	500	1	500

(a) Información de columnas.

Nombre Indice	Unicidad	BNivel[a]	Claves distintas
PROY_PLOC	No único	1	200
EMP_DNI	Único	1	10 000
EMP_SAL	No único	1	500

(b) Información de índices.

[a]Número de niveles sin el nodo hoja.

Nombre Tabla	Número de Filas	Bloques	Factor de bloque
PROYECTOS	2000	100	20
DEPARTAMENTOS	50	5	10
EMPLEADOS	10 000	2000	5

(c) Información de relaciones.

Cuadro 4.1: Información estadística de las relaciones.

La primera optimización basada en el coste que se debe realizar es elegir el orden de la operación JOIN entre las tres relaciones. Existen 3! = 6 posibilidades distintas:

1. $PROYECTOS \bowtie DEPARTAMENTOS \bowtie EMPLEADOS$
2. $DEPARTAMENTOS \bowtie PROYECTOS \bowtie EMPLEADOS$
3. $DEPARTAMENTOS \bowtie EMPLEADOS \bowtie PROYECTOS$
4. $EMPLEADOS \bowtie DEPARTAMENTOS \bowtie PROYECTOS$
5. $EMPLEADOS \bowtie PROYECTOS \bowtie DEPARTAMENTOS$
6. $PROYECTOS \bowtie EMPLEADOS \bowtie DEPARTAMENTOS$

Las opciones 5 y 6 se pueden desechar, ya que no resulta eficiente realizar en primer lugar un JOIN entre las relaciones *EMPLEADOS* y *PROYECTOS*, porque el número de registros del resultado sería mucho más grande que las otras posibilidades por culpa de que de las tres son las dos relaciones con una mayor cardinalidad, número de filas.

Es interesante también que la relación *DEPARTAMENTOS* esté presente en la primera operación JOIN que se ejecuta debido a que por el Cuadro 4.1 sabemos que su cardinalidad es 50. Consideramos el coste de ejecutar la primera alternativa.

Se estima primero el coste de la operación **JOIN 1** $= PROYECTOS \bowtie DEPARTAMENTOS$.

En primer lugar se debe estimar el coste de acceso a la memoria que guarda la información contenida en las relaciones, y posteriormente se estima el coste de su ejecución. La relación *DEPARTAMENTOS* no tiene un índice, por lo que el único método es la búsqueda lineal que tiene un coste de 5 accesos a bloque. En el caso de la relación *PROYECTOS*, asumiendo que la operación SELECT ya ha sido ejecutada, se puede realizar una búsqueda lineal o usar el índice de la relación que en este caso es *PROY_PLOC*, es aquí dónde se debe comparar el coste de las dos alternativas.

- La búsqueda lineal se estima que tiene un coste de 100 accesos de bloque.
- Como se indica en el Cuadro 4.1, tiene un coste de acceso al índice de 2 (Bnivel más nivel nodo hoja) y además no es único, con lo cual el optimizador asume una distribución normal de los datos. La estimación del coste, con los datos del Cuadro 4.1, se calcula de la forma:

$$Coste = \text{sl} * \text{Núm\_Filas} = \frac{\text{Núm\_Filas}}{\text{Distintos}} = \frac{2000}{200} = 10$$

  Luego, el coste estimado de acceso con el índice es: 10 + 2 = 12.

Por lo tanto, concluímos que para acceder a la relación PROYECTOS se hace mediante el índice de la relación. Calculamos ahora el coste de ejecución de la operación JOIN siguiendo el algoritmo de bucle anidado, en el cual la relación del bucle exterior será aquella resultante de la operación SELECT en la relación *PROYECTOS* y en el bucle interior la relación *DEPARTAMENTOS*. Como la relación *PROYECTOS* tiene un factor de bloque de 20 y únicamente estamos tratando con 10 de sus registros la primera relación ocupa un solo bloque, y se necesita leer cada uno de los cinco bloques que ocupa la relación de los departamentos, así que el coste total de la operación será de seis bloques más lo que cueste escribir la nueva relación.

El atributo *Num_Dpto* de la relación *DEPARTAMENTOS* toma un único valor para cada tupla, por lo tanto cada *Dnum* de la relación resultado de SELECT coincidirá sólo con un registro de la relación *DEPARTAMENTOS*, consecuentemente la relación resultado tendrá diez registros. Ahora, si suponemos que la relación resultante tiene un factor de bloque de cinco filas por bloque nos da un total de dos bloques que se necesitan para almacenar dicha relación. Con lo cual, tenemos un coste estimado de la ejecución del JOIN 1 de: 5 + 12 + 6 + 2 = 25.

Estimamos el coste de la operación **JOIN 2** $= JOIN\ 1 \bowtie EMPLEADOS$.

Se puede ejecutar un JOIN de un solo bucle, ya que en este caso se puede usar el índice *EMP_DNI* para localizar los registros coincidentes entre la relación *EMPLEADOS* y el primer JOIN. Por lo tanto, el método de unión implicaría leer cada bloque del primer resulatdo y buscar cada uno de los cinco valores del atributo Jefe_DNI utilizando el índice *EMP_DNI*. Cada búsqueda de índice requeriría un acceso a la raíz, un acceso a la hoja y un acceso a un bloque de datos , en total, 3 niveles. Así, 10 búsquedas requieren 30 accesos a bloques. Sumando los dos accesos a bloques para acceder al resultado de la primera operación se obtiene un total de 32 accesos a bloques para este JOIN.

El coste final es la suma de ambas operaciones y es igual a: 25 + 32 = 57 accesos a bloque. El optimizador debe realizar este proceso aquí descrito para las demás alternativas y elegir así la forma más eficiente.

Bibliografía

[1] CODD, E. F. (1970). *A relational model of data for large shared data banks*, Communications of the ACM, 13(6), 377-387.

[2] CODD, E. F. (1990). *The relational model for database management: Version 2.* Addison-Wesley.

[3] DATE, C. J. (2012). *SQL and Relational Theory: How to Write Accurate SQL Code (2ª ed.).* O'Reilly Media.

[4] DATE, C. J. (2003). *An Introduction to Database Systems (8ª ed.).* Pearson.

[5] DATE, C. J. (2000). *The database relational model: A retrospective review analysis (1ª ed.).* Prentice Hall.

[6] ELMASRI, R. & NAVATHE, S. B. (2015). *Fundamentals of database systems (7ª ed.).* Pearson.

[7] SIMON, A. R. & MELTON, J. (2001). *SQL: 1999: Understanding Relational Language Components (1ª ed.).* Morgan Kaufmann