Optimización de consultas a Bases de Datos relacionales

Enrique Ramas Ferrández

Trabajo de fin de grado en Matemáticas
Universidad de Zaragoza

Director del trabajo: Jorge Lloret Gazo
28 de junio de 2017

Prólogo

Este trabajo es una implementación a los conocimientos aprendidos en la asignatura Bases de Datos de cuarto curso. Con los conocimientos y habilidades aprendidas durante el Grado en Matemáticas, elaboro este documento con objeto de mejorar mis conocimientos de Informática.

La finalidad del trabajo es ser capaces de entender cómo funciona la optimización de consultas en el álgebra relacional. Para ello, comenzaremos el trabajo con una breve introducción del modelo relacional de Bases de Datos, así como sus operaciones dentro del álgebra relacional. Para finalizar, mostraremos el funcionamiento de la optimización de consultas presentando distintas formas de escribir una consulta, explicaremos el procedimiento a seguir para conseguir una consulta óptima, y definiremos una serie de funciones de coste que medirán el coste de la consulta.

Para entender mejor el procedimiento, acompañaremos el documento con un ejemplo completo de optimización. Desde que un usuario escribe una consulta, hasta que el gestor de Bases de Datos traduce dicha consulta y la optimiza.

Resumen

The purpose of this work is to understand and to explain the concepts on which the problem of the relational query optimization process is solved. To this end, first the relational algebra and its operators are presented. Next query optimization technics are presented and illustrated by means of an example.

0.1. THE RELATIONAL MODEL AND ITS OPERATIONS WITHIN THE RELATIONAL ALGEBRA

The relational model, born in 1970, is used by world-renowned database managers such as Oracle. This model represents the database as a collection of value tables, where each row in the table represents a collection of related values that can be both ordered and unordered.

A table on a database designed according to the relational model has the following form:

Table Name ⇩ Empleado

Columns

Rows

Nombre	Apellido1	Apellido2	Dni	FechaNac	Direccion	Sexo	Sueldo	Dno
Jose	Perez	Garcia	11122333	21/04/1975	Urb.Aurora,1	H	30000	3
Alberto	Campos	Sastre	33344555	08/12/1955	Avda. Rios,9	H	40000	5
Alicia	Jimenez	Martin	22233444	30/01/1980	Bretones,8	M	25000	5
Juana	Sainz	Oreja	98765432	20/06/1941	Cerquillas,67	M	43000	4
Fernando	Ojeda	Ordoñez	66688444	15/09/1962	Portillo,s/n	H	38000	5
Aurora	Oliva	Ferrer	44455666	07/07/1968	NULL	M	25000	2
Luis	Pajares	Rodriguez	55566777	NULL	Caseta,s/n	H	25000	1
Eduardo	Ochoa	Gonzalez	66677888	17/09/1975	Goya,9	H	55000	3

Where the rows are called tuples, the columns attributes, and the tables relations. From now on, we will use the terminology that appears in the picture.

In order to be able to manipulate a relational database, a series of relational operations are used that can be, unary, if they act on a table, or binary if they do on more than one table. The most used unary relational operations are SELECT and PROYECT, while the most recurrent binary relational operations are CARTESIAN PRODUCT and JOIN. Let's see what these operations are about:

- **SELECT:** It is used to select a subset of the rows of a table satisfying a selection condition that has previously been defined. It has the following form:

$$\sigma_{condition}(\text{TableName})$$

- **PROYECT:** In the same way that SELECT selects certain rows from one table and discards others, the PROYECT operation selects columns and discards others. It has the following form:

$$\pi_{columns}(\text{TableName})$$

- **CARTESIAN PRODUCT:** It is used to combine each row of a table with each of the rows of another table. It is a union of tables not very effective because it relates rows which do not have nothing in common. It has the following form:

$$(TableName1) \times (TableName2)$$

- **JOIN:** It is used to combine tuples from two tables into one. Unlike CARTESIAN PRODUCT, this operation allows combining rows between two tables in relation to each other by a condition that acts in the same way as the selection condition. It has the following form:

$$(NameTable1) \bowtie_{condition} (NameTable2)$$

Let's see an example that combines these two unary operations with the most effective binary operation, JOIN:

Recover the name and address of all employees working in the 'Investigacion' department.

$$\pi_{Nombre,Apellido1,Direccion}(\sigma_{NombreDpto='Investigacion'} (DEPARTAMENTO \bowtie_{NumeroDpto=Dno} (EMPLEADO)))$$

0.2. Query Optimization

In this chapter we will explain the optimization of queries, which is the process of choosing the most appropriate strategy for the execution of a query.

In the optimization process, the user writes a query in SQL language, then the database manager first transforms it into the language of relational algebra and, secondly, into a query tree. On this query tree, the database manager applies an algorithm composed by a set of heuristic rules. The algorithm transforms the initial tree into a tree whose execution cost is optimal.

0.2.1. SQL SELECT-FROM-WHERE Structure

A basic SQL query has the following form:

SELECT $< ColumnList >$
FROM $< TableList >$
WHERE $< condition >$

0.2.2. Structure in query trees

To optimize an SQL query, we will represent its expression in relational algebra as a query tree. A query tree is a tree data structure where the tables are the tree leaf nodes and the operations of the relational algebra are internal nodes.

Let's see how a query is represented first by a SQL query, second by a relational query and finally by a query tree with an example:

Write the query to retrieve the first last name of employees born after 1957 who work on a project called Aquarius.

SQL query:

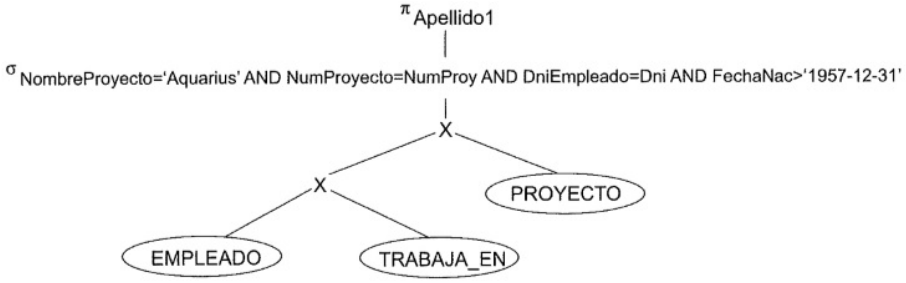
SELECT *Apellido1*

FROM *EMPLEADO, TRABAJA_EN, PROYECTO*
WHERE *FechaNac>'1957-12-31' AND NombreProyecto='Aquarius' AND DniEmpleado=Dni AND NumProyecto=NumProy*

Relational query:

$$\pi_{Apellido1}$$
$$(\sigma_{NombreProyecto='Aquarius' \text{ AND } NumProyecto=NumProy \text{ AND } DniEmpleado=Dni \text{ AND } FechaNac>'1957-12-31'}$$
$$(EMPLEADO \times TRABAJA\_EN \times PROYECTO))$$

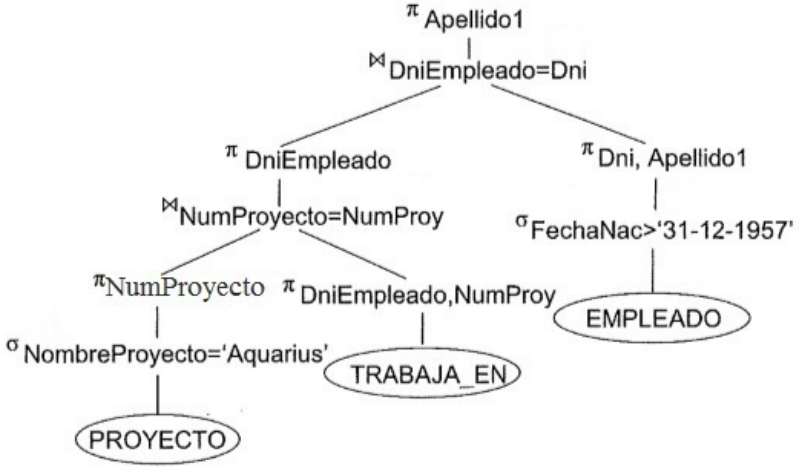
Query tree:



0.2.3. Algorithm of heuristic transformation of query trees

The heuristic rules are applied to query trees and aim at reducing the execution cost of the trees. This leads to by moving the SELECT and PROJECT operations down the tree as far away as possible. In addition, more restrictive SELECT and JOIN operations should be executed before other similar operations.

Using these rules we transform the tree of the previous example into this other equivalent tree:



We calculate the execution cost of each tree and verify that the final tree is more efficient than the initial one. For the first tree the database manager needs $2,4 \times 10^{18}$ entries to disk block, whereas for the second tree it needs $1,582 \times 10^{9}$ entries to disk block. Therefore, the second tree is more efficient than the first one.

Índice general

Prólogo III

Resumen V
0.1. THE RELATIONAL MODEL AND ITS OPERATIONS WITHIN THE RELATIONAL ALGEBRA . . . V
0.2. Query Optimization . . . VI
0.2.1. SQL SELECT-FROM-WHERE Structure . . . VI
0.2.2. Structure in query trees . . . VI
0.2.3. Algorithm of heuristic transformation of query trees . . . VII

1. Introducción al Modelo Relacional 1
1.1. Presentación del Modelo Relacional . . . 1
1.2. Estructura del Modelo Relacional . . . 3
1.3. Restricciones del Modelo Relacional . . . 3

2. Álgebra Relacional 5
2.1. Operaciones relacionales unarias . . . 6
2.2. Operaciones relacionales binarias . . . 8
2.3. Operaciones de álgebra relacional de la teoría de conjuntos . . . 9

3. Optimización de consultas 11
3.1. Estructura SELECT-FROM-WHERE de SQL . . . 11
3.2. Estructura heurística para la representación de consultas . . . 12
3.3. Optimización heurística de los árboles de consultas . . . 13
3.3.1. Reglas generales de transformación para las operaciones del álgebra relacional 13
3.3.2. Pasos del algoritmo de optimización heurística . . . 15
3.4. Índices y Funciones de Coste . . . 15
3.4.1. Índices . . . 15
3.4.2. Funciones de Coste . . . 16
3.5. Ejemplo completo de optimización heurística . . . 18
3.5.1. Árbol inicial de consultas . . . 19
3.5.2. Primer paso de optimización . . . 20
3.5.3. Segundo paso de optimización . . . 21
3.5.4. Tercer paso de optimización . . . 22
3.5.5. Último paso de optimización . . . 24

4. Apéndice 27

Bibliografía 33

Capítulo 1

Introducción al Modelo Relacional

El modelo relacional data de 1970 cuando Ted Codd presentó el artículo titulado *"A Relational Model for Large Shared Data Banks"*. Gracias a su simplicidad y a su fundamento matemático, el artículo permitió que se iniciaran un gran número de investigaciones para la optimización y comercialización del modelo. El modelo utiliza el concepto de *relación* como estructura básica y su base teórica es la teoría de conjuntos y la lógica de predicados de primer orden.

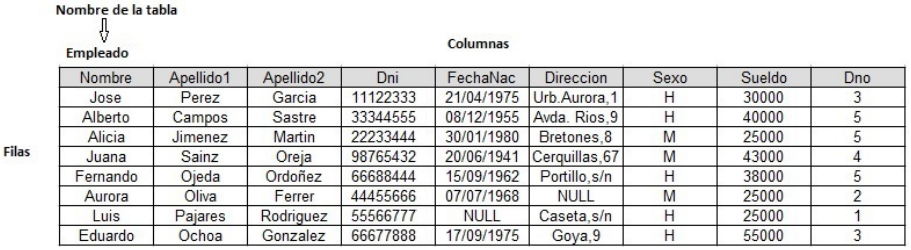
Las primeras implementaciones comerciales del modelo relacional, como SQL/DS y Oracle DBMS (DataBase Management System) estuvieron disponibles a principios de los años 80. En la actualidad, los gestores más populares son Oracle y SQL Server (de Microsoft), siendo Oracle el RDBMS (Relational DataBase Management System) más usado en la actualidad. Oracle está considerado como uno de los sistemas de bases de datos más completo. Entre sus características destacan: soporte de transacciones, estabilidad, escalabilidad, soporte multiplataforma y optimización de consultas.

Los DBMS deben desarrollar una estrategia de ejecución para obtener el resultado de la consulta a partir de los ficheros de la base de datos. Lo habitual es que en una consulta se disponga de muchas estrategias distintas de ejecución. El proceso de elección de la estrategia más adecuada se conoce como **optimización de consultas**.

En este trabajo, nos vamos a dedicar a estudiar la optimización de consultas a bases de datos relacionales.

1.1. Presentación del Modelo Relacional

Informalmente, el modelo relacional representa la base de datos como una colección de tablas de valores, donde cada fila de la tabla representa una colección de valores relacionados. El orden de las filas no es relevante, a diferencia de otros modelos como el jerárquico y el de red.
El hecho de que las filas no estén ordenadas tiene la ventaja de que una base datos es más fácil de entender y de manipular por un usuario no experto. Los valores de una tabla pueden ser recuperados o almacenados por medio de consultas que ofrecen una amplia flexibilidad para administrar dichos valores basándose exclusivamente en la información que tiene cada fila y no en su posición dentro de la tabla.
Un **ejemplo** de base de datos diseñada de acuerdo con el modelo relacional puede verse en la Figura 1:



Nombre	Apellido1	Apellido2	Dni	FechaNac	Direccion	Sexo	Sueldo	Dno
Jose	Perez	Garcia	11122333	21/04/1975	Urb.Aurora,1	H	30000	3
Alberto	Campos	Sastre	33344555	08/12/1955	Avda. Rios,9	H	40000	5
Alicia	Jimenez	Martin	22233444	30/01/1980	Bretones,8	M	25000	5
Juana	Sainz	Oreja	98765432	20/06/1941	Cerquillas,67	M	43000	4
Fernando	Ojeda	Ordoñez	66688444	15/09/1962	Portillo,s/n	H	38000	5
Aurora	Oliva	Ferrer	44455666	07/07/1968	NULL	M	25000	2
Luis	Pajares	Rodriguez	55566777	NULL	Caseta,s/n	H	25000	1
Eduardo	Ochoa	Gonzalez	66677888	17/09/1975	Goya,9	H	55000	3

Como podemos apreciar en la figura, tenemos una tabla llamada **Empleado** con **9 columnas** y **8 filas**. En las tablas de valores diferenciamos dos partes: La estructura y los datos.

La **estructura** engloba el nombre de la tabla y los nombres de las columnas que nos indican la información que va a albergar dicha columna en la tabla de valores. En este caso, el nombre de la tabla es Empleado y algunas de las columnas que lo componen son:

Nombre es el nombre del empleado.
Dni es el Dni del empleado.
Direccion que nos dará información sobre donde vive el empleado.
Sueldo que nos dará información acerca de los salarios recibidos por los empleados.
Dno que indica el número de departamento al que pertenece cada empleado.

La segunda parte son los **datos**. Dichos datos se organizan en filas y tienen una relación entre ellos. En nuestro ejemplo para una fila cualquiera, todos los datos son referentes a un único empleado. Por su parte, los datos que aparecen en las columnas también tendrán que estar relacionados, en este caso en la columna *Sexo* deberán aparecer datos relacionados exclusivamente con el sexo de la persona. De esta relación que poseen las columnas con los datos que albergan obtenemos el concepto de **dominio** de una columna, definido en el punto 1.2 del documento. Volviendo al ejemplo de la columna *Sexo*, tenemos que su dominio podría ser 1 carácter, el cual nos indicará el sexo de la persona, hombre (H) o mujer (M). Vamos a analizar dos filas de esta tabla de valores:

1.- Un empleado se llama Alberto, su primer apellido Campos y su segundo Sastre, su dni es 33344555, su fecha de nacimiento es 08/12/1955, su dirección es Avda. Rios, 9, su sexo es hombre, su sueldo es de 40.000 euros y su número de departamento es el 5. Como se ha mencionado en el párrafo anterior, todos los datos de esta fila están relacionados ya que nos dan información de la misma persona, Alberto Campos Sastre.

2.- Por otro lado, una empleada se llama Aurora, su primer apellido es Oliva y su segundo Ferrer, su dni es 44455666, su fecha de nacimiento es 07/07/1968, su dirección es **NULL**, su sexo es mujer, su sueldo es de 25.000 euros y su número de departamento es el 2. Como vemos, nos ha aparecido un dato de la tabla con información NULL. Esto puede deberse a dos situaciones, o que desconocemos dicha información, o que no hay información posible para dicha celda. En este caso, este NULL puede deberse a que se desconoce la dirección particular de Aurora Oliva Ferrer o que dicha persona no posee un domicilio fijo debido a un trabajo con continuos viajes y por tanto no dispone de número de direccion particular.

1.2. Estructura del Modelo Relacional

En este apartado vamos a formalizar los conceptos que informalmente se han presentado en el apartado anterior.

En el modelo relacional, las columnas corresponden a *atributos*, las filas corresponden a *tuplas*, el conjunto de filas que componen una tabla de valores es una *relación* y el conjunto de atributos de una tabla junto con el nombre de la tabla recibe el nombre de *esquema de relación*. El tipo de dato que describe los valores que pueden aparecer en cada columna está representada por un *dominio* de posibles valores. A continuación, describiremos con más detalle todos estos términos.

Dominio: Un dominio D es un conjunto de valores atómicos, es decir, cada valor de un dominio es indivisible en lo que al modelo relacional se refiere.
Por ejemplo: El dominio de la columna Dni de la tabla EMPLEADO es una cadena de 8 caracteres.

Atributo: Un atributo A es el nombre de un papel jugado por algún dominio D en el esquema de relación R.

Esquema de relación: Un esquema de relación es un conjunto de atributos más un nombre.

$$R = A_1, A_2, ..., A_n$$

Relación o Estado de relación: Una Relación r de un esquema de relación R con atributos A_1, A_2, ..., A_n; denotado por r(R) es un conjunto finito de asignaciones.

$$r = t_1, t_2, ..., t_m$$

en el que cada tupla t_i tiene la forma $(a_1, ..., a_n)$, donde $a_i \in dom(A_i)$

Tupla: Una tupla es un conjunto de parejas $(< atributo >, < valor >)$, donde cada una de ellas proporciona el valor de la asignación de un atributo A_i a un valor v_i de $dom(A_i)$ para i=1,...,n.

Finalmente, veamos la definición de esquema de bases de datos y de bases de datos.

Esquema de bases de datos: Un esquema de bases de datos es un conjunto de esquemas de relación y un conjunto de restricciones de integridad.

Base de datos: Una base de datos es un conjunto de relaciones, una por cada esquema de relación y que cumple las restricciones de integridad.

1.3. Restricciones del Modelo Relacional

En este apartado vamos a ver las restricciones sobre un estado de base de datos, que derivan de las reglas del *minimundo* que dicha base de datos representa. Las restricciones que podemos encontrar son las siguientes:

De Dominio: Especifican que dentro de cada tupla, el valor de un atributo A debe ser un valor atómico del dominio $dom(A)$.
Por ejemplo: En el caso del ejemplo de la Figura 1, en el atributo Dirección habrá una restricción de

dominio que indique que en esa columna solo podrá aparecer una cadena de 30 caracteres por ejemplo.

De Clave: Por definición, todos los elementos de un conjunto son distintos; por tanto, todas las tuplas en una relación también deben serlo. El diseñador de la tabla de valores definirá uno o varios atributos que actuarán como **claves**. Estas claves son información exclusiva de cada tupla, por lo que no se repetirá la misma información en dos tuplas distintas de la tabla. Una de estas claves será definida como **clave principal** que será la encargada de identificar una tupla entre todas las demás.
Por ejemplo: En el ejemplo de la Figura 1, el diseñador puede elegir el atributo Dni como clave principal de la relación, por tanto no pueden aparecer dos tuplas con la misma información en la columna Dni.

De Valores NULL o De Integridad de Entidad: Primero, definimos el valor **NULL** que será un valor asignado a un atributo dentro de una tupla y que indicará que o no tenemos información acerca del valor dado, o que no es aplicable a la tupla. Esta restricción especifica si se permite o no el NULL.
Por ejemplo: Si cada tupla de la tabla Empleado debe contar con un valor válido y no nulo para el atributo Nombre, entonces el Nombre del empleado debe declararse como NOT NULL.

De Integridad Referencial: Están especificadas entre dos relaciones y se utilizan para mantener la consistencia entre las tuplas de dos esquemas de relación. Veámoslo con un ejemplo:

En una relación llamada Empleado, en la cual se da información acerca de los empleados de una determinada empresa, el atributo Dno de Empleado nos informa sobre el número de departamento en el que trabaja cada empleado; por tanto, su valor en cada tupla Empleado debe coincidir con el valor NumeroDpto de alguna tupla de otra relación denominada Departamento.

Capítulo 2

Álgebra Relacional

Además de la estructura y de las restricciones vistas en el apartado anterior, un modelo de datos incluye un conjunto de operaciones para manipular la base de datos, que darán lugar a nuevas relaciones. En el modelo relacional, este conjunto básico de operaciones recibe el nombre de **álgebra relacional**.

El álgebra relacional proporciona un fundamento formal para las operaciones del modelo relacional y se utiliza como base para la implementación y optimización de consultas en los gestores de bases de datos.

En lo que sigue, trabajaremos sobre el siguiente ejemplo de bases de datos que llamamos base de datos Empresa:

EMPLEADO(Nombre, Apellido1, Apellido2, Dni, FechaNac, Dirección, Sexo, Sueldo, Dno)

Nombre	Apellido1	Apellido2	Dni	FechaNac	Dirección	Sexo	Sueldo	Dno
Alberto	Campos	Sastre	333445555	08/12/1955	Avda. Rios,9	H	40000	5
Juana	Sainz	Oreja	987654321	20/06/1941	Cerquillas,67	M	43000	4
Fernando	Ojeda	Ordoñez	666884444	15/09/1962	Portillo,s/n	H	38000	5
Jose	Perez	Garcia	111223333	21/04/1975	Urb.Aurora,1	H	30000	3
Alicia	Jimenez	Martin	222334444	30/01/1980	Bretones, 8	M	25000	5
Aurora	Oliva	Ferrer	444556666	07/09/1979	NULL	M	25000	2
Luis	Pajares	Rodriguez	555667777	NULL	Caseta,s/n	H	25000	1
Eduardo	Ochoa	Gonzalez	666778888	17/09/1965	Goya, 9	H	55000	3
Pedro	Sanz	González	888556666	19/10/1971	Coso, 19	H	58000	1
Carmen	Gutierrez	Sainz	223334444	01/07/1969	Coso, 35	M	60000	2

DEPARTAMENTO(NombreDpto, NúmeroDpto, DniDirector, FechIngDir)

NombreDpto	NúmeroDpto	DniDirector	FechIngDir
Investigación	5	33344555	07/09/1979
Administración	4	98765432	30/09/1995
Sede Central	1	88855666	18/09/1993
Análisis	3	66677888	14/12/2006
RRHH	2	22333444	05/06/2010

A continuación, vamos a describir algunas de las operaciones relacionales más importantes como son: la **SELECCIÓN**, la **PROYECCIÓN**, la **UNIÓN**, la **INTERSECCIÓN**, la **DIFERENCIA DE CONJUNTOS**, el **JOIN** y el **PRODUCTO CARTESIANO**.

2.1. Operaciones relacionales unarias

Las operaciones unarias operan en relaciones individuales y son la **SELECCIÓN** y la **PROYECCIÓN**.

1.- La operacion de SELECCIÓN: Esta operación se emplea para seleccionar un subconjunto de las tuplas de una relación que satisfacen una *condición de selección* que previamente se habrá definido. Se puede considerar esta operación como un *filtro* que mantiene sólo las tuplas que satisfacen dicha condición de selección.

En general se utiliza la siguiente notación para definir esta operación:

$$\sigma_{condicion}(\text{esquemaRelación})$$

donde el símbolo σ se utiliza para especificar el operador de SELECCIÓN, mientras que las condiciones del select son expresiones lógicas sobre los atributos del esquema de relación utilizadas para seleccionar las tuplas que buscamos.
Estas operaciones lógicas se dividen en dos grupos, **simples** y **complejas**.

Las **simples** son de la forma *a op b*, con a y b expresiones. En particular pueden ser atributos o constantes. Algunos ejemplos de expresiones lógicas son:

a	**op**	**b**
sueldo	$>$	40.000
$\frac{5*sueldo}{2}$	$>$	70.000

Las **complejas** son la unión de expresiones lógicas mediante **AND**, **OR** y **NOT**. Estas expresiones tienen el siguiente aspecto:

Condición1	**op**	**Condición2**
$sueldo > 30000$	AND	Dno=3
$sueldo > 40000$	OR	Dno=4
$- - - - - - -$	NOT	Dno=5

Por ejemplo: Para seleccionar todos los empleados que trabajan en el departamento 4 y ganan más de 25.000 euros al año, o los que trabajan en el departamento 5 y cobran más de 30.000 euros al año, podemos especificar la siguiente operacion de SELECCIÓN:

$$\sigma_{(Dno=4\ AND\ sueldo>25000)\ OR\ (Dno=5\ AND\ sueldo>30000)}(EMPLEADO)$$

En este caso, hay una condición compleja, que es un OR de otras dos condiciones complejas.. Una es *Dno=4 AND sueldo>25.000* y la otra es *Dno=5 AND sueldo>30.000.* A su vez, la primera condición compleja está formada por otras dos condiciones simples: Dno=4 es la primera y sueldo>25.000 es la segunda. Por otro lado la segunda condición compleja está formada por otras dos condiciones simples: Dno=5 es la primera y sueldo>30.000 es la segunda.
Además, como queremos que se cumpla una de las dos condiciones, usamos el operador complejo **OR**, dando lugar a la condición expuesta en el ejemplo.

El resultado de la consulta de SELECCIÓN del ejemplo sobre la tabla EMPLEADO es el siguiente:

Nombre	**Apellido1**	**Apellido2**	**Dni**	**FechaNac**	**Dirección**	**Sexo**	**Sueldo**	**Dno**
Alberto	Campos	Sastre	33344555	08/12/1955	Avda. Rios,9	H	40000	5
Juana	Sainz	Oreja	98765432	20/06/1941	Cerquillas,67	M	43000	4
Fernando	Ojeda	Ordoñez	66688444	15/09/1962	Portillo,s/n	H	38000	5

2.- La operación PROYECCIÓN: Si pensamos en un esquema de relación como una tabla, la operación de SELECCIÓN elige algunas de las filas de la tabla a la vez que descarta otras. Por otro lado la operación de PROYECCIÓN selecciona ciertas columnas de la tabla y descarta otras. Si sólo estamos interesados en algunos atributos de un esquema de relación, usamos la operacion de PROYECCIÓN para *planear* la relación sólo sobre estos atributos.

En general se utiliza la siguiente notación para definir esta operación:

$$\pi_{atributos}(\text{esquema de relacion})$$

Por ejemplo: Para listar el nombre, el primer apellido y el sueldo de cada empleado, podemos usar la operación de PROYECCIÓN de la siguiente forma:

$$\pi_{Nombre,Apellido1,Sueldo}(EMPLEADO)$$

donde el símbolo π es el símbolo usado para representar la operacion de PROYECCIÓN, mientras que Apellido1, Nombre y Sueldo son atributos del esquema de relación EMPLEADO.

El resultado de la consulta de PROYECCIÓN anterior para nuestro ejemplo de base de datos es el siguiente:

Nombre	**Apellido1**	**Sueldo**
Jose	Perez	30000
Alberto	Campos	40000
Alicia	Jimenez	25000
Juana	Sainz	43000
Fernando	Ojeda	38000
Aurora	Oliva	25000
Luis	Pajares	25000
Eduardo	Ochoa	55000
Pedro	Sanz	58000
Carmen	Gutierrez	60000

Podemos combinar en una misma consulta las dos operaciones anteriores para obtener un solo resultado.

Por ejemplo: Para listar el Dni de todos los empleados que trabajan en el departamento número 5, podemos usar las operaciones de SELECCIÓN y PROYECCIÓN de la siguiente forma:

$$\pi_{Dni}(\sigma_{Dno=5}(EMPLEADO))$$

El resultado de esta consulta sería el siguiente:

Dni
33344555
22233444
66688444

2.2. Operaciones relacionales binarias

Las operaciones relacionales binarias son aquellas que se especifican sobre dos esquemas relacionales. A continuación, vamos a describir la operación binaria **PRODUCTO CARTESIANO** y el **JOIN**.

La operación PRODUCTO CARTESIANO: El PRODUCTO CARTESIANO de dos esquemas de relación $R(A_1,...,A_n)$ y $S(B_1,...,B_n)$, denotado por $R(A_1,...,A_n) \times S(B_1,...,B_n)$ es otro esquema de relación $Q(A_1,...,A_n,B_1,...,B_n)$ cuyos atributos son $A_1,...,A_n,B_1,...,B_n$ y cuya relación son las tuplas formadas por la unión de las tuplas del esquema de relación R y las de S.

El problema de esta operación es que permite unir tuplas que no estén relacionadas entre sí, por eso debemos hacer una SELECCIÓN de tuplas de las dos relaciones especificando una condición de selección apropiada.
Existe una operación especial llamada **JOIN** que permite especificar esta secuencia de PRODUCTO CARTESIANO y de SELECCIÓN como una operación única. A continuación, vamos a estudiarla con más detalle.

La operación JOIN: Representada mediante el símbolo $\bowtie$, se emplea para combinar tuplas de dos relaciones en una sola. Esta operación es muy importante para cualquier base de datos relacional que cuente con más de una relación, ya que nos permite unir información que proviene de dos o más relaciones.

En general se utiliza la siguiente notación para definir esta operación:

$R \bowtie_{condicion} S$ con R, S dos esquemas de relación y condición tal y como hemos descrito en el apartado sobre la operacion de SELECCIÓN.

Por ejemplo: Consideremos la base de datos Empresa. Si queremos recuperar la información de cada director de departamento unida a su información de empleado, necesitamos combinar tuplas de departamento y empleado cuyos valores de DniDirector y Dni, respectivamente, sean iguales. Esto se consigue con un JOIN de la siguiente forma:

$$\text{DIRECTOR\_DPTO} \longleftarrow \text{DEPARTAMENTO} \bowtie_{DniDirector=Dni} EMPLEADO$$

En este ejemplo vemos como la operación JOIN forma una nueva relación a la que denominamos **DIRECTOR_DPTO**, por lo que podemos hablar de que obtenemos un esquema de relación que tiene como nombre DIRECTOR_DPTO y como atributos la unión de los atributos de los esquemas de relación DEPARTAMENTO y EMPLEADO. Podemos ver este nuevo esquema de relación a continuación:

DIRECTOR_DPTO

NombreDpto	NúmeroDpto	DniDirector	...	Nombre	Apellido1	Apellido2	Dni
Investigación	5	33344555	...	Alberto	Campos	Sastre	33344555
Administración	4	98765432	...	Juana	Sainz	Oreja	98765432
Análisis	3	66677888	...	Eduardo	Ochoa	Gonzalez	66677888
RRHH	2	22333444	...	Carmen	Gutierrez	Sainz	22333444
Sede Central	1	88855666	...	Eduardo	Ochoa	Paredes	88855666

En el caso de que no necesitemos toda la información de cada director tendríamos que añadir a nuestra consulta más operaciones. Vamos a ver algunos casos particulares respecto la consulta anterior:

1.- Mostrar únicamente el nombre de departamento y el Dni.

$$\text{DIR\_DPTO1} \longleftarrow \pi_{NombreDpto,Dni}(DIRECTOR\_DPTO)$$

2.- Mostrar todo de los que ganan más de 30.000 euros.

$$\text{DIR\_DPTO2} \longleftarrow \sigma_{sueldo>30000}(DIRECTOR\_DPTO)$$

3.- Mostrar únicamente el nombre de departamento y el Dni de los directores que ganan más de 30.000 euros.

$$\text{DIR\_DPTO3} \longleftarrow \pi_{NombreDpto,Dni}(\sigma_{sueldo>30000}(DIRECTOR\_DPTO))$$

2.3. Operaciones de álgebra relacional de la teoría de conjuntos

El siguiente grupo de operaciones de álgera relacional son las correspondientes a la operativa matemática sobre conjuntos. Para combinar los elementos de dos conjuntos se utilizan varias operaciones de la teoría de conjuntos como la **UNIÓN**, la **INTERSECCIÓN**, la **DIFERENCIA**. Todas ellas son operaciones **binarias**, es decir, se aplican a dos conjuntos de tuplas.

Podemos definir las tres operaciones en dos esquemas de relación R y S del siguiente modo:

· **UNIÓN:** El resultados de esta operación especificada como R$\bigcup$S, es una relación que incluye todas las tuplas que están en R o están en S.

· **INTERSECCIÓN:** El resultado de esta operación especificada como R$\bigcap$S es una relación que incluye todas las tuplas que están en R y en S.

· **DIFERENCIA DE CONJUNTO:** El resultado de esta operación especificada como R - S, es una relación que incluye todas las tuplas que están en R pero no en S.

Un ejemplo de cada una de estas operaciones considerando la base de datos Empresa, serían los siguientes:

1.- Recupere los Dnis de todos los empleados del departamento 3 o del departamento 4:

$$\textbf{Resultado} \longleftarrow \pi_{Dni}(\sigma_{Dno=3}(EMPLEADO)) \cup \pi_{Dni}(\sigma_{Dno=4}(EMPLEADO))$$

Resultado

Dni
11122333
66677888
98765432

2.- Recupere el dni de los directores de departamento que tengan un sueldo superior a 50000 euros:

$$\textbf{Resultado} \longleftarrow \pi_{Dni}(\sigma_{Sueldo>50000}(EMPLEADO)) \cap \pi_{DniDirector}(DEPARTAMENTO)$$

Resultado

Dni
66677888
88855666
22333444

3.- Recupere los Dnis de todos los empleados que no sean directores de departamento:

$$\textbf{Resultado} \longleftarrow \pi_{Dni}(EMPLEADO) - \pi_{Dni}(DEPARTAMENTO)$$

Resultado

Dni
22233444
44455666
55566777
66677888
66688444

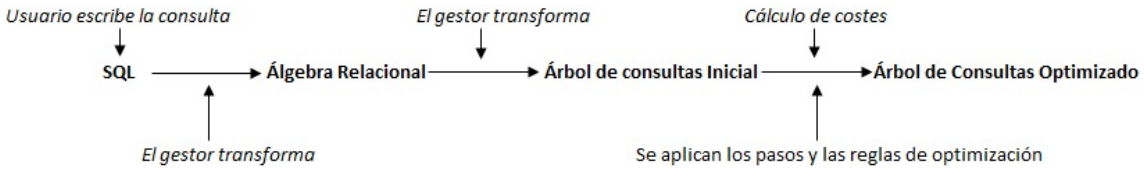
Capítulo 3

Optimización de consultas

En este capítulo vamos a explicar la optimización de consultas, que es el proceso de elección de la estrategia más adecuada para la ejecución de una consulta.

En el proceso de optimización, el usuario escribe una consulta en lenguaje SQL, después el gestor de la base de datos la transforma, en primer lugar, al lenguaje del álgebra relacional y, en segundo lugar, en un **árbol de consultas**. Sobre este árbol de consultas, al que llamaremos inicial, el gestor aplica un algoritmo compuesto por un conjunto de reglas heurísticas. El algoritmo transforma el árbol inicial en un árbol final cuyo coste de ejecución es óptimo..

Un esquema del proceso de optimización es el siguiente:



Para ilustrar la optimización de consultas, usaremos ejemplos sobre nuestra base de datos empresa, cuyas tablas recordamos a continuación:

EMPLEADO(Nombre, Apellido1, Apellido2, Dni, FechaNac, Dirección, Sexo, Sueldo, Dno**)**

PROYECTO(NombreProyecto, NumProyecto, UbicacionProyecto, NumDptoProyecto**)**

TRABAJA_EN(DniEmpleado, NumProy, Horas**)**

A continuación, estudiaremos cada una de las partes del proceso de optimización. En primer lugar, presentaremos el lenguaje SQL, a continuación los árboles de consultas. Posteriormente trataremos con las reglas de optimización y las funciones de coste.

3.1. Estructura SELECT-FROM-WHERE de SQL

Una consulta SQL básica tiene la siguiente forma:

SELECT $< listaColumnas >$

FROM $< listaTablas >$
WHERE $< condicion >$

donde:

− < **listaColumnas** > es una lista de columnas cuyos valores serán recuperados por la consulta.
− < **listaTablas** > es una lista de tablas donde está la información necesaria para procesar la consulta.
− < **condicion** > es una expresión condicional que identifica las tuplas que la consulta recuperará.

Ejemplo: Vamos a escribir la consulta para recuperar la fecha de nacimiento y la dirección del empleado cuyo nombre es José Pérez Pérez. Lo haremos en el lenguaje SQL y en lenguaje del álgebra relacional.

Forma SQL:

SELECT *FechaNac, Dirección*
FROM *EMPLEADO*
WHERE *Nombre='José' AND Apellido1='Pérez' AND Apellido2='Pérez'*

Forma relacional:

$$\pi_{FechaNac,Direccion}(\sigma_{Nombre='Jose' \text{ AND } Apellido1='Perez' \text{ AND } Apellido2='Perez'}(EMPLEADO))$$

3.2. Estructura heurística para la representación de consultas

Para optimizar una consulta SQL, representaremos su expresión en el álgebra relacional como un árbol de consultas. Un árbol de consultas es una estructura de datos en árbol donde las tablas son los nodos hoja del árbol y las operaciones del álgebra relacional son nodos internos.

Una ejecución del árbol de consultas consiste en ejecutar una operación de un nodo interno, para después reemplazar ese nodo interno por la tabla que resulta de la ejecución de la operación. La ejecución termina cuando se ejecuta el nodo raíz y se genera la tabla resultante de la consulta.

Ejemplo: Escribe la consulta para recuperar el primer apellido de los empleados nacidos después de 1957 que trabajan en un proyecto llamado 'Aquarius'.

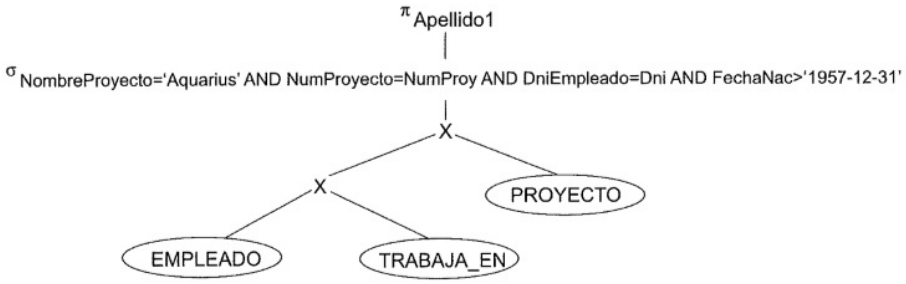
Su expresión en SQL es:

SELECT *Apellido1*
FROM *EMPLEADO, TRABAJA_EN, PROYECTO*
WHERE *FechaNac>'1957-12-31' AND NombreProyecto='Aquarius' AND DniEmpleado=Dni AND NumProyecto=NumProy*

Su expresión en el álgebra relacional es:

$$\pi_{Apellido1}$$
$$(\sigma_{NombreProyecto='Aquarius' \text{ AND } NumProyecto=NumProy \text{ AND } DniEmpleado=Dni \text{ AND } FechaNac>'1957-12-31'}$$
$$(EMPLEADO \times TRABAJA\_EN \times PROYECTO))$$

El árbol correspondiente a esta expresión de álgebra relacional es:



Una vez sabemos cómo se representan las consultas, en el siguiente apartado vamos a presentar las reglas de optimización heurística.

3.3. Optimización heurística de los árboles de consultas

En esta sección vamos a estudiar un conjunto de reglas que podemos utilizar para transformar un árbol de consultas en otro equivalente. Además, estudiaremos los pasos del algoritmo que, basándose en estas reglas, obtiene un árbol de consultas optimizado.

Para optimizar una consulta, el analizador de consultas genera un **árbol de consultas inicial** estándar sin ninguna optimización. Después de la generación del árbol de consultas inicial, el algoritmo transforma, en pasos sucesivos, este árbol en otro **árbol de consultas final** con un coste de ejecución menor.

Para ello, el algoritmo aplica reglas heurísticas que se describen a continuación. Si el coste de ejecución del árbol transformado es superior al coste de ejecución del árbol original, se descarta la transformación y se busca otra.

Intuitivamente, una de las principales reglas heurísticas es aplicar las operaciones de SELECCIÓN y PROYECCIÓN antes de aplicar operaciones binarias, ya que éstas producen un fichero de gran tamaño debido a su función multiplicativa de los ficheros de entrada.

Estas intuiciones, se plasman en las siguientes reglas de optimización.

3.3.1. Reglas generales de transformación para las operaciones del álgebra relacional

En este apartado nos centraremos en el significado de las operaciones y de las relaciones resultantes, es es por esto que si dos relaciones tienen el mismo conjunto de atributos en diferente orden pero representan la misma información, las consideraremos equivalentes.

Algunas de las reglas de transformación más útiles para la optimización de consultas son:

1- Cascada de σ: Una condición de SELECCIÓN conjuntiva puede ser dividida en una cascada de operaciones σ individuales:

$$\sigma_{c1\,AND\,c2\,AND\,...\,AND\,cn}(R) \equiv \sigma_{c1}(\sigma_{c2}(...(\sigma_{cn}(R))...))$$

2- Conmutatividad de σ: La operación SELECCIÓN es conmutativa:

$$\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$$

3- Cascada de π: En una cascada de operaciones de PROYECCIÓN, todas, excepto la última, pueden ser ignoradas:

$$\pi_{Lista1}(\pi_{Lista2}(...(\pi_{Listan}(R))...)) \equiv \pi_{Lista1}(R)$$

4- Conmutatividad de σ por π: Si la condición de SELECCIÓN c incluye sólo los atributos $A_1, ..., A_n$ en la lista de proyección, las dos operaciones pueden ser conmutadas:

$$\pi_{A1,A2,...,An}(\sigma_c(R)) \equiv \sigma_c(\pi_{A1,A2,...,An}(R))$$

5- Conmutatividad de $\bowtie$ (y $\times$): Las operaciones JOIN y PRODUCTO CARTESIANO son conmutativas:

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

6- Conmutación de σ con $\bowtie$ (o $\times$): Si todos los atributos con condición de SELECCIÓN c involucran sólo a los atributos de una de las relaciones que están siendo unidas, las dos operaciones pueden ser conmutadas entre sí:

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

7- Conmutación de π con $\bowtie$ (o $\times$): Supongamos que la lista de proyección es L= $\{A_1, ..., A_n, B_1, ..., B_m\}$, donde $A_1, ..., A_n$ son atributos de R y $B_1, ..., B_m$ son atributos de S. Si la condición JOIN c incluye sólo los atributos de L, las dos operaciones pueden ser conmutadas entre sí:

$$\pi_L(R \bowtie_c S) \equiv (\pi_{A1,...,An}(R)) \bowtie_c (\pi_{B1,...,Bm}(S))$$

8- Asociatividad de $\bowtie$, $\times$, $\cup$ y $\cap$: Estas cuatro operaciones son asociativas de forma individual, es decir, si θ representa a cualquiera de estas cuatro operaciones, tenemos:

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

9- Conmutación de σ con las operaciones de conjunto: La operación σ se puede conmutar con $\cup$, $\cap$ y -. Si θ representa cualquiera de estas tres operaciones, tenemos:

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

10- La operación π se puede conmutar con $\cup$:

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

11- Conversión de una secuencia (σ, $\times$) en $\bowtie$: Si la condición c de una operación σ que sigue a una operación $\times$ corresponde a una operación JOIN:

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

3.3.2. Pasos del algoritmo de optimización heurística

En estos momentos, podemos describir los pasos de un algoritmo que utiliza algunas de las reglas descritas anteriormente para transformar un árbol de consultas inicial en un árbol optimizado que se ejecute de forma más eficiente.

Los pasos del algoritmo son los que se muestran a continuación:

1- Utilizando la regla 1, descomponer cualquier operación de SELECCIÓN con condiciones conjuntivas en una cascada con operaciones de SELECCIÓN, lo que permite un mayor grado de libertad para mover las operaciones de SELECCIÓN hacia abajo por las distintas ramas del árbol.

2- Utilizando las reglas 2, 4, 6 y 10 relativas a la conmutatividad de la SELECCIÓN con otras operaciones, desplazar cada operación de SELECCIÓN hacia abajo por el árbol tan lejos como lo permitan los atributos incluidos en la condición de SELECCIÓN.

3- Utilizando las reglas 5 y 8 relativas a la conmutatividad y asociación de las operaciones binarias, reordenar las relaciones de los nodos hoja utilizando los siguientes criterios. En primer lugar, posicionar las relaciones de los nodos hoja con las operaciones de SELECCIÓN más restrictivas (que generan una relación con el menor número de tuplas o con el menor tamaño absoluto), de forma que sean ejecutadas en primer lugar en la representación del árbol de consultas. En segundo lugar, asegurarse de que la ordenación de los nodos hoja no produce ningún producto cartesiano.

4- Utilizando la regla 11, combinar una operación de producto cartesiano con la siguiente operación de SELECCIÓN del árbol para formar una operación JOIN, en el caso de que la condición represente una condición JOIN.

5- Utilizando las reglas 3, 4, 7 y 10 relativas a la secuencia de PROYECCIONES y a su conmutación con otras operaciones, descomponer y desplazar las listas de atributos de PROYECCIÓN hacia abajo por el árbol lo más lejos posible mediante la creación de nuevas operaciones de PROYECCIÓN según sea necesario.

6- Identificar los subárboles que representan grupos de operaciones que se puedan ejecutar mediante un único algoritmo.

Antes de ver un ejemplo completo del proceso de optimización de una consulta, presentamos varios conceptos previos.

3.4. Índices y Funciones de Coste

En esta sección, vamos a definir, en primer lugar, unas estructuras de acceso a tabla denominadas índices, útiles para acelerar la recuperación de registros en diferentes consultas de SELECCIÓN. En segundo lugar, definiremos una serie de funciones que nos proporcionarán una información aproximada acerca del coste de una consulta, denominadas funciones de coste.

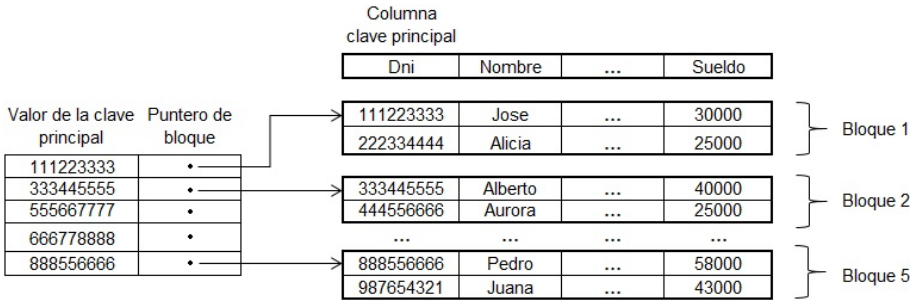
3.4.1. Índices

Actuando de forma similar a los índices de los libros comunes, los **índices** son estructuras de datos que se utilizan sobre columnas en las tablas que sirven para recuperar de una manera rápida las filas que

necesitamos en las distintas consultas realizadas sobre una base de datos.

Un índice es un fichero ordenado con dos columnas de longitud fija. La primera columna es del mismo tipo de datos que la columna clave de ordenación de la tabla sobre la que se define. La segunda columna es un puntero a un bloque del fichero de datos.

Ejemplo: Sobre nuestra base de datos Empresa, creamos sobre la tabla EMPLEADO un índice sobre el atributo DNI.



El índice actuará de la siguiente forma, el gestor recibe una consulta con una condición que actúa sobre la columna Dni, como la tabla tiene un índice sobre dicha columna, el gestor buscará en el fichero índice el bloque al que pertenece. ¿Cómo encuentra dicho bloque? Veámoslo con un ejemplo:

Recuperar la información del empleado que tiene DNI 222334444:

El gestor busca en el fichero índice el valor de ese DNI y determina que se encuentra en el bloque [11122333, 33344555). Entonces, el gestor busca sobre ese bloque el DNI 222334444 ahorrándose buscar en el resto de bloques de la tabla.

3.4.2. Funciones de Coste

El coste de una consulta proviene de varios factores como el tiempo que se utiliza por cada entrada a bloque, el tiempo utilizado para cada acceso a memoria o el tiempo de procesamiento de determinadas operaciones. En este trabajo, vamos a estudiar el coste medido únicamente como el número de entradas a bloques de datos necesarias para la ejecución de la consulta, es decir, el número de bloques que hay que leer para la ejecución de la misma. Consideramos únicamente ese factor porque es el que más tiempo consume en la respuesta a una consulta.

Las operaciones que aparecen en una consulta tienen distintos costes que dependen de varios factores. Por ejemplo, el uso de índices rebaja considerablemente los costes de una consulta. Cada consulta, como hemos visto, puede tener varias operaciones (varias selecciones, distintos JOIN) por lo que para cada consulta, el coste de la misma será la suma de los costes que se acumulen a lo largo de cada una de las operaciones de la consulta.

A continuación, vamos a definir algunos conceptos necesarios para el cálculo de costes.

nTuples(R): Número de tuplas de la tabla R.

bFactor: Número de tuplas que caben en cada bloque.

nBlocks(R): Número de bloques de la tabla R.

$$nBlocks(R) = \frac{nTuples(R)}{bFactor}$$

A continuación, estudiaremos los costes de las operaciones de SELECCIÓN y de JOIN, que son las operaciones que se usan con mayor frecuencia en las consultas.

Coste de la SELECCIÓN sobre la tabla R:

Operación	Ejemplo	Estrategias	Coste
Igualdad de clave principal	$\sigma_{NumProyecto='1000'}$	Búsqueda lineal sin índice en tabla desordenada	$nBlocks(R)/2$
Igualdad de clave principal	$\sigma_{Dni='222334444'}$	Búsqueda binaria sin índice en tabla ordenada por clave principal	$log_2(nBlocks(R))$
Igualdad de clave principal	$\sigma_{Dni='222334444'}$	Búsqueda con índice	$log_2(nBlocks(I))+1$
Igualdad de columna no clave	$\sigma_{NombreProyecto='Aquarius'}$	Búsqueda lineal sin índice en tabla desordenada	$nBlocks(R)$
Desigualdad de columna no clave	$\sigma_{FechaNac>'1957-12-31'}$	Búsqueda lineal sin índice en tabla desordenada	$nBlocks(R)$

En primer lugar, en el caso de una *igualdad sobre una columna clave principal de una tabla sin índice desordenada*, el gestor buscará un único registro que cumpla la condición de selección, por lo que, en media, tendrá que leer la mitad de la tabla para encontrar dicho registro. En consecuencia, el coste de leer la mitad de la tabla viene determinado por nBlocks(R)/2.

En segundo lugar, una *búsqueda binaria sin índice en tabla ordenada por clave principal* tiene un coste de $log_2(nBlocks(R))$. ¿De dónde viene este coste? La tabla, al estar ordenada, permite al gestor realizar una búsqueda en el registro que ocupe la posición central de toda la tabla. De este modo, el gestor verá si el registro buscado está en la mitad superior de la tabla o en el inferior, evitándose así la búsqueda en la mitad de los registros de la tabla. El gestor seguirá reduciendo la búsqueda a la mitad sucesivamente hasta encontrar el registro que cumpla la condición, por tanto el coste de este tipo de SELECCIÓN viene determinado por una función logarítmica de base 2 del número de bloques en los que se divide la tabla, de donde se obtiene el coste de $log_2(nBlocks(R))$.

En tercer lugar, siguiendo el mismo razonamiento que para la búsqueda binaria, el gestor buscará en el fichero índice ordenado el registro que cumpla la condición de SELECCIÓN, obteniendo un coste de $log_2(nBlocks(I))$, siendo I el fichero índice. A este coste habrá que sumarle el coste de recuperar el registro de la tabla indexada sobre la que trabajamos que será de 1 acceso a bloque puesto que solamente 1 registro cumplirá la condición de SELECCIÓN.

En último lugar, en el caso de una *igualdad o una desigualdad de columna no clave de una tabla sin índice desordenada*, el gestor tendrá que leer toda la tabla para comprobar qué filas cumplen la condición. Por tanto su coste vendrá determinado por el número de bloques de dicha tabla, nBlocks(R).

Coste de la operación JOIN:

Operación	Ejemplo	Estrategias	Coste
JOIN de clave principal	$\bowtie_{NumProyecto=NumProy}$	Combinación bucle anidado sin índice	$nBlocks(R)+$ $nBlocks(R)*nBlocks(S)$
JOIN de clave principal	$\bowtie_{Dni=DniEmpleado}$	Combinación bucle anidado con índice	$nBlocks(R)+nTuples(R)*$ $(log_2(nBlocks(I))+1)$

Sobre el ejemplo de optimización heurística que aparece en la siguiente sección de este documento, vamos a utilizar las 2 funciones de coste JOIN que hemos definido. En primer lugar, aparece el algoritmo más simple en el cálculo de costes JOIN, el cual tendrá un coste de $nBlocks(R)+nBlocks(R)*nBlocks(S)$. El gestor deberá leer cada bloque de la tabla R por lo que acumulará un coste de $nBlocks(R)$, además tendrá que leer cada bloque de la tabla S para cada bloque de R, acumulando un coste de $nBlocks(S)*nBlocks(R)$. Sumando los 2 costes que hemos acumulado, obtenemos un coste total de $nBlocks(R)+nBlocks(R)*nBlocks(S)$ tal y como queríamos probar. ¿Qué tabla tomamos como R y cuál como S? El segundo sumando es fijo, sin embargo el primer sumando depende de qué tabla se tome como la primera, por tanto tomaremos como tabla R la que necesite menos accesos a bloque.

En segundo lugar, si una de las 2 tablas del JOIN posee un índice primario sobre el atributo de combinación, utilizaremos la función de coste para *bucle anidado con índice*. El gestor leerá todos los bloques de la tabla no indexada acumulando un coste de $nBlocks(R)$ y, para cada tupla de R, utilizará el índice para extraer las filas correspondientes de la tabla indexada S. De este modo, para cada tupla de R realizará una *búsqueda con índice en tabla ordenada*, que como hemos visto en los costes de la SELECCIÓN tiene un coste de $(log_2(nBlocks(I))+1)$. Como esta búsqueda se realiza para cada tupla de la tabla R, se acumulará un coste de $nTuples(R)*(log_2(nBlocks(I))+1)$. Sumando los 2 costes que hemos acumulado, obtenemos un coste total de $nBlocks(R)+nTuples(R)*(log_2(nBlocks(I))+1)$.

A continuación, vamos a realizar el estudio de optimización y cálculo de costes a partir de un ejemplo completo.

3.5. Ejemplo completo de optimización heurística

Vamos a suponer la siguiente información sobre la base de datos Empresa.

Tabla EMPLEADO: 3.000.000 tuplas.
Tabla PROYECTO: 200.000 tuplas.
Tabla TRABAJA_EN: 4.000.000 tuplas.
Fichero Índice: 40.000 tuplas.
$-I_1$: Índice primario sobre la clave principal DNI en EMPLEADO.
-Longitud de campos: 128 bytes cada uno.
-Proyectos que sean de Aquarius: 50.000 tuplas.
-Empleados nacidos después de 1957: 2.500.000 tuplas.
-El tamaño de cada bloque es de 2.048 bytes.

Para ilustrar la optimización heurística, usaremos el siguiente ejemplo:

Ejemplo: Recuperar los apellidos de los empleados nacidos después de 1957 que trabajan en un proyecto llamado 'Aquarius'.

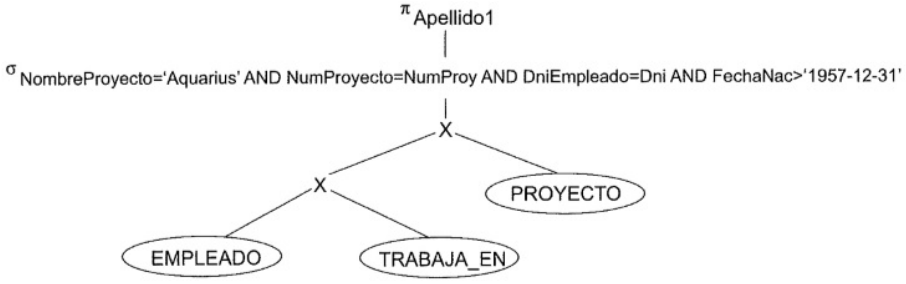
En formato SQL se escribe:

SELECT *Apellido1*
FROM *EMPLEADO, PROYECTO, TRABAJA_EN*
WHERE *NombreProyecto='Aquarius' AND NumProyecto=NumProy AND DniEmpleado=Dni AND FechaNac>'1957-12-31'*

El gestor de base de datos transformará esta consulta al álgebra relacional, con lo que obtendrá la siguiente consulta relacional:

$$\pi_{Apellido1}(\sigma_{NombreProyecto='Aquarius' \text{ AND } FechaNac>'1957-12-31'} (EMPLEADO \bowtie_{Dni=DniEmpleado} (PROYECTO \bowtie_{NumProyecto=NumProy} (TRABAJA\_EN))))$$

Posteriormente, el gestor de base de datos transforma esta consulta en el siguiente árbol de consultas, al que denominamos inicial:



A partir de este árbol, comienza el proceso de optimización y de cálculo de costes.

3.5.1. Árbol inicial de consultas

Dado el árbol de consultas inicial que acabamos de ver, calculamos su coste:

Coste:

El producto cartesiano de las 3 tablas genera una tabla de 16 atributos cuyo número de tuplas es:

$nTuples(EMPLEADO) \times nTuples(PROYECTO) \times nTuples(TRABAJA\_EN) = 2,4 \times 10^{18}$ tuplas.

El número de bloques ocupados por esas tuplas se calcula del siguiente modo.

Longitud de una tupla = 16×128=2048 (Número de atributos $\times$ longitud de cada campo)
bFactor = $2048/2048 = 1$
nTuples = $2,4 \times 10^{18}$
nBlocks = $2,4 \times 10^{18}/1 = 2,4 \times 10^{18}$

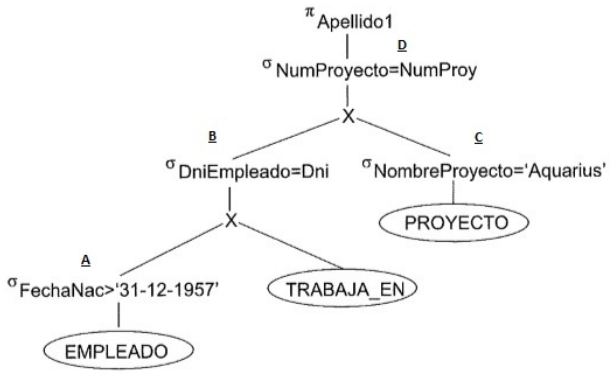
Finalmente, como se trata de una SELECCIÓN sin índices, se usa la búsqueda lineal. En la sección anterior hemos visto que el coste de esa búsqueda es:

$nBlocks = 2,4 \times 10^{18}$ entradas a bloque.

3.5.2. Primer paso de optimización

Ahora, comenzamos a optimizar el árbol utilizando las reglas heurísticas mencionadas en este capítulo. Para ello, se usa el paso 2 del algoritmo de optimización desplazando las operaciones de SELECCIÓN hacia abajo por el árbol de consultas tanto como nos lo permitan los atributos que aparecen en las condiciones de SELECCIÓN.

En este caso, valiéndonos de las reglas heurísticas 2, 4, 6 y 10 relativas a la conmutatividad de las operaciones de SELECCIÓN con el resto de operaciones, podemos desplazar las condiciones de SELECCIÓN FechaNac>'31-12-1957' y NombreProyecto='Aquarius' hasta abajo en el árbol de consultas para ejecutarlas en primer lugar y obtener, de este modo, una reducción del número de filas de las tablas EMPLEADO y PROYECTO considerable. Por último, utilizando las mismas reglas de optimización, desplazamos las condiciones de SELECCIÓN NumProyecto=NumProy y DniEmpleado=Dni hasta los productos cartesianos. De este modo, en los siguientes pasos de optimización usemos la regla 11 para combinar la operación de SELECCIÓN con el PRODUCTO CARTESIANO y formar un JOIN. Veamos como queda tras estos cambios el árbol de consultas:



Coste:

NODO A: Coste $1,7 \times 10^6$ entradas a bloque	
Número de bloques ocupados por las tuplas de la tabla EMPLEADO	Longitud de una tupla = 9×128=1152 bFactor = $2048/1152 = 1,78$ nTuples = 3×10^6 nBlocks = $3 \times 10^6/1,78 = 1,7 \times 10^6$
Coste de la búsqueda lineal en una selección sin índices	nBlocks=$1,7 \times 10^6$ entradas a bloque

Para el NODO B, el producto cartesiano de las 2 tablas anteriores genera una tabla de 12 atributos cuyo número de tuplas es:

$(nTuples(EMPLEADOSconFechaNac > \text{'}31-12-1957\text{'}) \times nTuples(TRABAJA\_EN))$ = 10^{13} tuplas.

NODO B: Coste $7,5 \times 10^{12}$ entradas a bloque	
Número de bloques ocupados por las tuplas de la tabla RESULTANTE	Longitud de una tupla = 12×128=1536 bFactor = $2048/1536 = 1,33$ nTuples = 10^{13} nBlocks = $10^{13}/1,33 = 7,5 \times 10^{12}$
Coste de la búsqueda lineal en una selección sin índices	nBlocks=$7,5 \times 10^{12}$ entradas a bloque

La relación de tuplas entre las tablas EMPLEADO y TRABAJA_EN es de 4 tuplas en TRABAJA_EN por cada 3 tuplas en EMPLEADO. Por tanto tras la operación de SELECCIÓN del NODO B se obtendrán $(nTuples(EMPLEADOSconFechaNac > \text{'}31-12-1957\text{'}) * 4/3$ = $2,5 \times 10^6$ * $4/3$ tuplas, esto es, $3,33 \times 10^6$ tuplas.

NODO C: Coste 5×10^4 entradas a bloque	
Número de bloques ocupados por las tuplas de la tabla PROYECTO	Longitud de una tupla = 4×128=512 bFactor = $2048/512 = 4$ nTuples = 2×10^5 nBlocks = $2 \times 10^5/4 = 5 \times 10^4$
Coste de la búsqueda lineal en una selección sin índices	nBlocks=5×10^4 entradas a bloque

Ahora, el producto cartesiano de las 2 tablas del nodo D genera una tabla de 16 atributos cuyo número de tuplas es:

$(nTuples(nodo\ B) \times nTuples(Proyecto.NombreProyecto = \text{'}Aquarius\text{'}))$ = $1,67 \times 10^{11}$ tuplas.

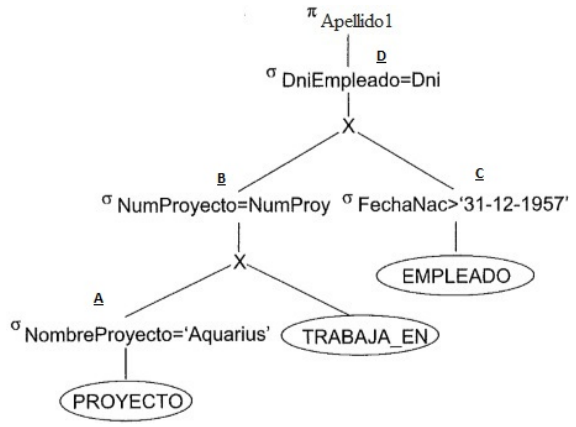
NODO D: Coste $1,67 \times 10^{11}$ entradas a bloque	
Número de bloques ocupados por las tuplas de la tabla RESULTANTE	Longitud de una tupla = 16×128=2048 bFactor = $2048/2048 = 1$ nTuples = $1,67 \times 10^{11}$ nBlocks = $1,67 \times 10^{11}/1 = 1,67 \times 10^{11}$
Coste de la búsqueda lineal en una selección sin índices	nBlocks=$1,67 \times 10^{11}$ entradas a bloque

Por tanto, el **coste total** del árbol será:

Coste(A) + **Coste(B)** + **Coste(C)** + **Coste(D)** $\approx 7,667 \times 10^{12}$ entradas a bloque.

3.5.3. Segundo paso de optimización

Desde el árbol inicial utilizando los pasos 2 y 3 del algoritmo de optimizacion, también podemos llegar a este árbol.



Coste:

NODO A: Coste 50.000 entradas a bloque	
Número de bloques ocupados por las tuplas de la tabla PROYECTO	Longitud de una tupla = $4 \times 128 = 512$ bFactor = $2048/512 = 4$ nTuples = 200.000 nBlocks = $200.000/4 = 50.000$
Coste de la búsqueda lineal en una selección sin índices	nBlocks=50.000 entradas a bloque

Para el NODO B obtenemos, tras el producto cartesiano de PROYECTO y TRABAJA_EN, una tabla de 7 atributos y 2×10^{11} tuplas.

NODO B: Coste $8,73 \times 10^{10}$ entradas a bloque	
Número de bloques ocupados por las tuplas de la tabla RESULTANTE	Longitud de una tupla = 7×128=896 bFactor = $2048/896 = 2,29$ nTuples = 2×10^{11} nBlocks = $2 \times 10^{11}/2,29 = 8,73 \times 10^{10}$
Coste de la búsqueda lineal en una selección sin índices	nBlocks=$8,73 \times 10^{10}$ entradas a bloque

La relación de tuplas entre las tablas PROYECTO y TRABAJA_EN es de 20 tuplas en TRABAJA_EN por cada tupla en PROYECTO. Por tanto tras la operación de SELECCIÓN del NODO B se obtendrán (nTuples(Proyectos Aquarius) * 20 = 50.000 * 20 tuplas, esto es, 10^6 tuplas.

NODO C: Coste $1,7 \times 10^6$ entradas a bloque	
Número de bloques ocupados por las tuplas de la tabla EMPLEADO	Longitud de una tupla = 9×128=1152 bFactor = $2048/1152 = 1,78$ nTuples = 3×10^6 nBlocks = $3 \times 10^6/1,78 = 1,7 \times 10^6$
Coste de la búsqueda lineal en una selección sin índices	nBlocks=$1,7 \times 10^6$ entradas a bloque

Para el NODO D, tras el producto cartesiano, obtenemos una tabla de 16 atributos y $2,5 \times 10^{12}$ tuplas

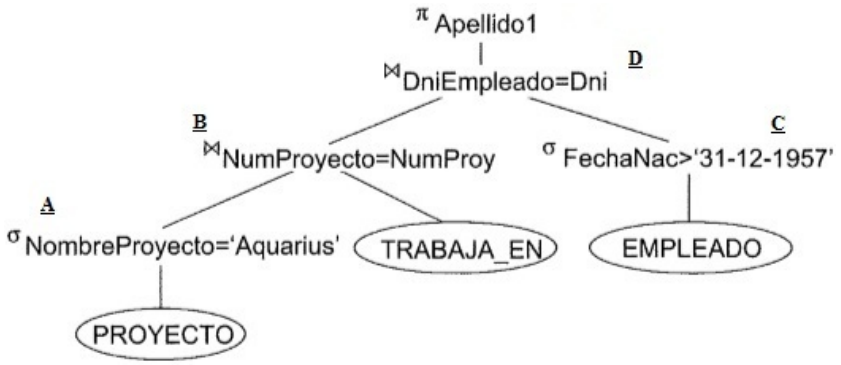
NODO D: Coste $2,5 \times 10^{12}$ entradas a bloque	
Número de bloques ocupados por las tuplas de la tabla RESULTANTE	Longitud de una tupla = 16×128=2048 bFactor = $2048/2048 = 1$ nTuples = $2,5 \times 10^{12}$ nBlocks = $12,5 \times 10^{12}/1 = 2,5 \times 10^{12}$
Coste de la búsqueda lineal en una selección sin índices	nBlocks=$2,5 \times 10^{12}$ entradas a bloque

Por tanto, el coste del árbol será:

Coste(A) + **Coste(B)** + **Coste(C)** + **Coste(D)** $\approx 2,5873 \times 10^{12}$ entradas a bloque.

3.5.4. Tercer paso de optimización

Ahora, suponiendo que tenemos un índice primario sobre la columna Dni en la tabla EMPLEADO y utilizando el paso 4 del algoritmo de optimización, reemplazamos el PRODUCTO CARTESIANO y la SELECCIÓN con operaciones JOIN.



Coste:

NODO A: Coste 50.000 entradas a bloque	
Número de bloques ocupados por las tuplas de la tabla PROYECTO	Longitud de una tupla = $4 \times 128 = 512$ bFactor = $2048/512 = 4$ nTuples = 200.000 nBlocks = $200.000/4 = 50.000$
Coste de la búsqueda lineal en una selección sin índices	nBlocks=50.000 entradas a bloque

NODO B: En este caso, tenemos un par de tablas no indexadas, la que viene del nodo A y la tabla TRABAJA_EN, por lo que utilizaremos la función de coste *combinación bucle anidado sin índice* para el JOIN:

Coste $nBlocks(R) + nBlocks(R) * nBlocks(S)$

Del nodo A hemos obtenido una tabla con las mismas columnas que la tabla PROYECTO pero reducida a 50.000 filas (proyectos que sean de Aquarius son 50.000 por definición). Hallamos el número de accesos a bloque necesarios para leer esta tabla:

Longitud de una tupla = $4 \times 128 = 512$

bFactor $= 2048/512 = 4$

nTuples = 50.000

nBlocks = $50.000/4 = 12.500$

Ahora, hallamos el número de accesos a bloque necesarios para leer la otra tabla del JOIN, la tabla TRABAJA_EN:

Longitud de una tupla = $3 \times 128 = 384$

bFactor $= 2048/384 = 5,33$

nTuples = 4×10^6

nBlocks = $4 \times 10^6/5,33 = 7,5 \times 10^5$

Como podemos observar, el número de bloques de la tabla resultante del nodo A es menor, por lo que utilizaremos esta tabla como la tabla R en la función de coste JOIN. Ahora, sustituimos en la formula del coste JOIN, obteniendo un coste para el JOIN del nodo B de:

$12.500 + 12.500 \times 7,5 \times 10^5 = 9,375 \times 10^9$ entradas a bloque.

Como la relación lineal entre las tablas TRABAJA_EN y PROYECTO es de $1 \longrightarrow 20$, podemos suponer que en el JOIN obtendremos alrededor de $5 \times 10^4 \times 20 = 10^6$ tuplas.

NODO C: Coste $1,7 \times 10^6$ entradas a bloque	
Número de bloques ocupados por las tuplas de la tabla EMPLEADO	Longitud de una tupla = 9×128=1152 bFactor = $2048/1152 = 1,78$ nTuples = 3×10^6 nBlocks = $3 \times 10^6/1,78 = 1,7 \times 10^6$
Coste de la búsqueda lineal en una selección sin índices	nBlocks=$1,7 \times 10^6$ entradas a bloque

NODO D: En este caso, disponemos de un JOIN que afecta a 2 tablas, la que proviene del JOIN del nodo B y la que proviene de la selección del nodo C. Esta última, posee un índice primario sobre el campo Dni de la tabla EMPLEADO, por tanto utilizaremos la estrategia *Combinación bucle anidado con índice* para hallar el coste del JOIN:

Coste $= nBlocks(R) + nTuples(R) \times (log_2(nBlocks(I)) + 1)$

De acuerdo con lo que hemos explicado en la estrategia *Combinación bucle anidado con índice*, vamos

a calcular los datos de la tabla no indexada, que es el resultado del nodo B.

Longitud de una tupla = 7×128=896
bFactor = $2048/896 = 2,29$
nTuples = 10^6
nBlocks = $10^6/2,29 = 4,367 \times 10^5$

Hallamos los datos referentes al fichero índice, definido sobre la columna Dni de la tabla EMPLEADO:

Longitud de una tupla = 2×128=256
bFactor = $2048/256 = 8$
nTuples = 40.000
nBlocks = $40.000/8 = 5.000$
$log_2(nBlocks(I)) = 15,61$

Sustituimos en la formula del coste JOIN, obteniendo un coste de:

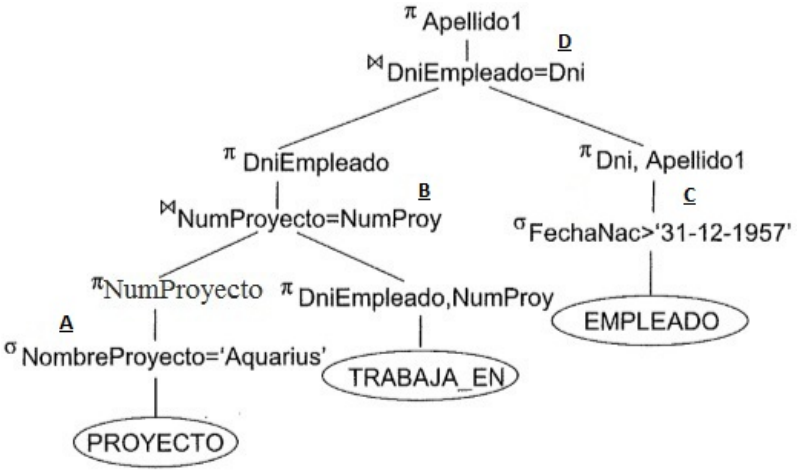
Coste $= nBlocks(R) + nTuples(R) \times (log_2(nBlocks(I)) + 1) = 4,367 \times 10^5 + 10^6 \times [15,61 + 1] = 1,705 \times 10^7$ entradas a bloque.

Por tanto, el coste total del árbol será:

Coste(A) + **Coste(B)** + **Coste(C)** + **Coste(D)** $\approx 9,395 \times 10^9$ entradas a bloque.

3.5.5. Último paso de optimización

Por último, aplicamos el paso 5 del algoritmo de optimización y desplazamos las operaciones PROJECT hacia abajo por el árbol de consultas:



Coste:

NODO A: Coste 50.000 entradas a bloque	
Número de bloques ocupados por las tuplas de la tabla PROYECTO	Longitud de una tupla = $4 \times 128 = 512$ bFactor = $2048/512 = 4$ nTuples = 200.000 nBlocks = $200.000/4 = 50.000$
Coste de la búsqueda lineal en una selección sin índices	nBlocks=50.000 entradas a bloque

NODO B: Del mismo modo que en el nodo B del árbol de consultas anterior, tenemos 2 tablas no indexadas, por lo que utilizaremos la función de coste de *combinación bucle anidado sin índice* para el JOIN:

Coste $nBlocks(R) + nBlocks(R) * nBlocks(S)$

Con un cálculo análogo al del nodo B del tercer paso de optimización, obtenemos que el número de entradas a bloque es de $1,563 \times 10^9$ y el número de tuplas resultante tras este JOIN es de 10^6.

NODO C: Coste $1,7 \times 10^6$ entradas a bloque	
Número de bloques ocupados por las tuplas de la tabla EMPLEADO	Longitud de una tupla = 9×128=1152 bFactor = $2048/1152 = 1,78$ nTuples = 3×10^6 nBlocks = $3 \times 10^6/1,78 = 1,7 \times 10^6$
Coste de la búsqueda lineal en una selección sin índices	nBlocks=$1,7 \times 10^6$ entradas a bloque

NODO D: En este caso, disponemos de un índice primario sobre el campo Dni de la tabla EMPLEADO, por tanto utilizaremos la función *Combinación bucle anidado con índice* para hallar el coste del JOIN:

Coste $= nBlocks(R) + nTuples(R) \times (log_2(nBlocks(I)) + 1)$

Con un cálculo análogo al del nodo D del tercer paso de optimización, obtenemos que el número de entradas a bloque es de $1,667 \times 10^7$.

Por tanto, el coste total del árbol será:

Coste(A) + **Coste(B)** + **Coste(C)** + **Coste(D)** $\approx 1,581 \times 10^9$ entradas a bloque.

En consecuencia, nuestro algoritmo determina que esta última estrategia es la mejor ya que tiene un menor coste de entradas a bloque (del orden de 10^9 frente a las primeras estrategias cuyo coste era del orden de 10^{18} y de 10^{12}). Para ejecutar la consulta SQL, se usaría el árbol de consultas de este último paso de optimización.

Capítulo 4

Apéndice

En este Apéndice vamos a implementar brevemente los conocimientos aprendidos sobre los operadores del Álbegra Relacional. En primer lugar, estudiaremos la *completitud de los operadores relacionales* y definiremos una serie de operaciones relacionales adicionales a las que ya hemos definido. Para finalizar, resolveremos una serie de ejercicios que mezclan operaciones relacionales.

Completitud de los operadores relacionales:

Las operaciones del álgebra relacional SELECCIÓN, PROYECCIÓN, UNIÓN, DIFERENCIA y PRODUCTO CARTESIANO conforman un conjunto **completo**, es decir, cualquiera de las operaciones originales podemos expresarlas como una secuencia de operaciones de este conjunto.
Dicho de otra forma, todas las consultas pueden reducirse a combinaciones de las operaciones mencionadas aunque podamos utilizar otras operaciones del algebra relacional que nos simplifiquen y/o faciliten la consulta.

Por ejemplo: La INTERSECCIÓN puede expresarse usando UNIÓN y DIFERENCIA del siguiente modo:

$$R \cap S \equiv (R \cup S) - ((R - S) \cup (S - R))$$

Operaciones relacionales adicionales:

Existen algunas consultas habituales a bases de datos que no pueden llevarse a cabo con las operaciones de álgebra relacional mencionadas en los apartados anteriores. Algunas de estas consultas, que mejoran la potencia del álgebra relacional original, son el **proyect generalizado**, las **funciones de agregación**, la **división**, la **join externo**, la **unión externa** y las de **cierre recursivo**.

A continuación vamos a definir las dos primeras, ya que la DIVISIÓN puede expresarse como combinación de las operaciones ya mencionadas en este documento y el JOIN EXTERNO y las operaciones de CIERRE RECURSIVO no aparecen con frecuencia en las consultas del álgebra relacional.

Proyect generalizado: Es una operación que amplía las posibilidades del proyect original permitiendo la inclusión de funciones de atributos en la lista del proyect.

En general se utiliza la siguiente notación para definir esta operación:

$\pi_{F_1,\ldots,F_n}(R)$ donde $F_1, \ldots, F_n$ son funciones que pueden involucrar constantes.

Por ejemplo: Considerando la base de datos Empresa, si queremos mostrar el Dni de los empleados así como el sueldo mensual, lo representamos de la siguiente forma:

$$\pi_{Dni,Sueldo/12}(EMPLEADO)$$

Funciones de agregación: SUMA, **MEDIA**, **MÁXIMO**, **MÍNIMO** o **CONTAR** son consultas que no pueden expresarse a través del álgebra relacional básico. Para especificar este tipo de peticiones definimos una operación **FUNCIÓN AGREGADA** usando el símbolo $\Im$.

En general se utiliza la siguiente notación para definir estas peticiones:

$$<atributos\ de\ agrupamiento>\Im_{<lista\ de\ funciones>}(R)$$

Por ejemplo: Para recuperar cada número de departamento, el número de empleados del mismo, y la media de sueldos, lo representamos de la siguiente forma:

$$_{Dno}\Im_{COUNT\ Dni,\ AVERAGE\ Sueldo}(EMPLEADO)$$

El resultado de esta operación es:

Dno	Count Dni	Media Sueldo
1	2	41.500
2	2	42.500
3	2	42.500
4	1	43.000
5	3	34.333

EJERCICIOS:

Para finalizar con las operaciones relacionales, vamos a ver un ejemplo que incluye las operaciones SELECCIÓN, PROYECCIÓN, JOIN y AGREGACIÓN, para después terminar con una serie de ejercicios:

Recupere nombre y número de departamento, así como el sueldo medio de los departamentos que tengan un sueldo medio superior a 35.000 euros.

$$\textbf{RESULT1} \longleftarrow {}_{Dno}\Im_{AVERAGE\ Sueldo}(EMPLEADO)$$
$$\textbf{RESULT} \longleftarrow \pi_{NombreDpto,\ NumeroDpto}(\sigma_{Sueldo>35000}\ (DEPARTAMENTO \bowtie_{NumeroDpto=Dno} RESULT1))$$

A continuación, se muestran una serie de ejercicios. Para poder resolverlos, ampliamos la base de datos EMPRESA con los siguientes esquemas de relación:

TRABAJA_EN

DniEmpleado	NumProy	Horas

Este esquema de relación consta de 3 atributos. **Dni** con información del dni del empleado, **NumProy** con información sobre el número de proyecto en el que trabaja el empleado, y **Horas**, que expresa el número de horas que invierte por semana un empleado en el proyecto dado.

PROYECTO

NombreProyecto	**NumProyecto**	**UbicacionProyecto**	**NumDptoProyecto**

Este esquema de relación consta de 4 atributos. **NombreProyecto**, **NumProyecto**, **UbicacionProyecto** y **NumDptoProyecto** con información acerca del nombre, número asignado, ciudad en la que se realiza y el número de departamento al que pertenece un proyecto respectivamente.

SUBORDINADO

Dni	**NombSubordinado**	**Sexo**	**FechaNac**	**Relacion**

Este esquema de relación consta de 5 atributos. **Dni** con información del Dni de un empleado que tiene subordinados, **NombSubordinado** con información del nombre de un empleado subordinado, **Sexo** con información del sexo del subordinado, **FechaNac** con información de la fecha de nacimiento del subordinado y **Relacion** que albergará valores numéricos que indican los niveles de diferencia que tienen el empleado jefe de su subordinado.

1. Recupere el nombre y la dirección de todos los empleados que trabajan en el departamento 'Investigacion'.

$$\textbf{RESULT1} \longleftarrow \sigma_{NombreDpto='Investigacion'}(DEPARTAMENTO)$$
$$\textbf{RESULT2} \longleftarrow RESULT1 \bowtie_{NumeroDpto=Dno} EMPLEADO$$
$$\textbf{RESULT} \longleftarrow \pi_{Nombre,Apellido1,Direccion}(RESULT2)$$

Como expresión única, esta consulta se convierte en:

$$\pi_{Nombre,Apellido1,Direccion}(\sigma_{NombreDpto='Investigacion'} (DEPARTAMENTO \bowtie_{NumeroDpto=Dno} (EMPLEADO)))$$

En los próximos ejercicios utilizaremos la primera opción para una comprensión más sencilla.

2. Recupere los nombres de todos los departamentos cuyos directores son de género masculino y que cobren un sueldo superior a 30000 euros.

$$\textbf{DIRDPTO1} \longleftarrow DEPARTAMENTO \bowtie_{DniDirector=Dni} EMPLEADO$$
$$\textbf{RESULTADO} \longleftarrow \pi_{NombreDpto}(\sigma_{Sexo='H' AND Sueldo>30000}(DIRDPTO1))$$

3. Recupere los nombres de todas las empleadas sean o no directoras de departamento.

$$\textbf{DIRECDPTO2} \longleftarrow DEPARTAMENTO \bowtie_{DniDirector=Dni} EMPLEADO$$
$$\textbf{RESULTADO1} \longleftarrow \pi_{Nombre}(\sigma_{Sexo='M'}(DIRECDPTO2))$$
$$\textbf{RESULTADO2} \longleftarrow \pi_{Nombre}(\sigma_{Sexo='M'}(EMPLEADO))$$
$$\textbf{RESULTADO} \longleftarrow RESULTADO1 \cup RESULTADO2$$

4. Recupere los nombres de todas las empleadas que no sean directoras de departamento.

$$\textbf{DIRECDPTO2} \longleftarrow DEPARTAMENTO \bowtie_{DniDirector=Dni} EMPLEADO$$
$$\textbf{RESULTADO1} \longleftarrow \pi_{Nombre}(\sigma_{Sexo='M'}(DIRECDPTO2))$$
$$\textbf{RESULTADO2} \longleftarrow \pi_{Nombre}(\sigma_{Sexo='M'}(EMPLEADO))$$
$$\textbf{RESULTADO} \longleftarrow RESULTADO2 - RESULTADO1$$

5. Recupere los nombres y la dirección de todos los directores que hayan conseguido su puesto en el siglo XXI y hayan nacido más tarde del año 1976.

$$\textbf{DIRECDPTO3} \longleftarrow DEPARTAMENTO \bowtie_{DniDirector=Dni} EMPLEADO$$
$$\textbf{RESULT1} \longleftarrow \pi_{Nombre,Direccion}(\sigma_{FechIngDir>01/01/2000}(DIRECDPTO3))$$
$$\textbf{RESULT2} \longleftarrow \pi_{Nombre,Direccion}(\sigma_{FechaNac>01/01/1977}(DIRECDPTO3))$$
$$\textbf{RESULT} \longleftarrow \text{RESULT1} \cap \text{RESULT2}$$

6. Recupere los nombres de todos los empleados que trabajan en algún proyecto y que no tengan subordinados.

$$\textbf{RESULT1} \longleftarrow EMPLEADO \bowtie_{Dni=DniEmpleado} TRABAJA\_EN$$
$$\textbf{RESULT2} \longleftarrow \pi_{Dni}(EMPLEADO) - \pi_{Dni}(SUBORDINADO)$$
$$\textbf{RESULT3} \longleftarrow EMPLEADO \bowtie_{Dni=Dni} RESULT2$$
$$\textbf{RESULT} \longleftarrow \pi_{Nombre,Apellido1}(RESULT3)$$

7. Recupere los nombres de todos los empleados que no trabajan en ningún proyecto.

$$\textbf{RESULT1} \longleftarrow \pi_{Dni}(EMPLEADO) - \pi_{DniEmpleado}(TRABAJA\_EN)$$
$$\textbf{RESULT} \longleftarrow \pi_{Nombre,Apellido1}(EMPLEADO \bowtie_{Dni=Dni} RESULT1)$$

8. Recupere el salario medio de los empleados según el sexo y después recupere sólo el de las empleadas.

$$\text{a} - \textbf{RESULT} \longleftarrow {}_{Sexo}\Im_{AVERAGE\ Sueldo}(EMPLEADO)$$
$$\text{b} - \textbf{RESULT1} \longleftarrow {}_{Sexo}\Im_{AVERAGE\ Sueldo}(EMPLEADO)$$
$$\textbf{RESULT} \longleftarrow \sigma_{Sexo='M'}(RESULT1)$$

9. Recupere los nombres de los empleados que tengan más de un subordinado.

$$\textbf{RESULT1} \longleftarrow {}_{Dni}\Im_{COUNT\ NombSubordinado}(SUBORDINADO)$$
$$\textbf{RESULT2} \longleftarrow EMPLEADO \bowtie_{Dni=Dni} RESULT1$$
$$\textbf{RESULT} \longleftarrow \pi_{Nombre,Apellido1}(\sigma_{ContarNombSubordinado>2}(RESULT2))$$

10. Por cada proyecto ubicado en 'Zaragoza', recupere su número, el número de departamento que lo gestiona y el nombre y la fecha de nacimiento del director.

$$\textbf{RESULT1} \longleftarrow \sigma_{UbicacionProyecto='Zaragoza'}(PROYECTO)$$
$$\textbf{RESULT2} \longleftarrow (RESULT1 \bowtie_{NumDptoProyecto=NumeroDpto} DEPARTAMENTO)$$
$$\textbf{RESULT3} \longleftarrow (RESULT2 \bowtie_{DniDirector=Dni} EMPLEADO)$$
$$\textbf{RESULT} \longleftarrow \pi_{NumProyecto,NumDptoProyecto,Nombre,Apellido1,FechaNac}(RESULT3)$$

11. Por cada proyecto, recupere el número de horas que se trabaja en él.

$$\textbf{RESULT1} \longleftarrow {}_{NumProy}\Im_{SUM\ Horas}(TRABAJA\_EN)$$
$$\textbf{RESULT2} \longleftarrow (PROYECTO \bowtie_{NumProyecto=NumProy} RESULT1)$$
$$\textbf{RESULT} \longleftarrow \pi_{NombreProyecto,Horas}(RESULT2)$$

Bibliografía

[1] CONNOLLY, T. M., BEGG, C. E. (2006), *Sistemas de Bases de Datos: Un Enfoque Práctico Para Diseño, Implementación y Gestión de Bases de Datos*, Pearson Education.

[2] ELMASRI, R. (2007), NAVATHE, S., *Fundamentos de bases de datos* (5 ed), Addison-Wesley, `http://www.librosite.net/elmasri`.