Álgebra relacional y optimización heurística de consultas a base de datos

Jorge Lacasa Fonseca

Trabajo de fin de grado en Matemáticas
Universidad de Zaragoza

Director del trabajo: Jorge Lloret Gazo
3 de diciembre de 2020

Prólogo

Las bases de datos y los sistemas de bases de datos son un componente esencial de la vida cotidiana en la sociedad moderna. Actualmente, todos nosotros nos encontramos todos los días con actividades que implican una interacción directa o indirecta con una base datos, como por ejemplo al realizar una operación bancaria, realizar una reserva en un hotel, retirar un libro de una bilioteca, comprar un producto online, etc.

Hoy en día, la tecnología de los medios de comunicación nos permite almacenar diversos tipos de datos, como imágenes digitales, clips de audio o mapas meteorológicos, aunque esto no siempre ha sido así. En los últimos años el ámbito de las bases de datos ha sufrido una evolución importante que nos ha permitido pasar de guardar datos numéricos o textuales a todas las opciones que manejamos en la actualidad.

En las primeras bases de datos se utilizaba un modelo jerárquico, que pronto daría lugar a un modelo en red, que sería el modelo referente hasta principios de la década de los 70, plasmado en el modelo Codasyl.

El modelo relacional fue presentado por primera vez en 1970 por Edgar Frank "Ted" Codd, de IBM Research, y enseguida llamó la atención de todos los expertos debido a su simplicidad y fundamentación matemática.

Basándose en lo propuesto por Codd, los laboratorios de IBM desarrollaron un lenguaje de dominio específico para administrar y recuperar información de sistemas de gestión de bases de datos relacionales. Este lenguaje llamado SEQUEL fue desarrollado durante los siguientes años hasta dar lugar al actual SQL, uno de los lenguajes referentes para manipular bases de datos.

En 1978, y ante la negativa de IBM para invertir en el prototipo de Codd SystemR, el que pretendía ser la primera base de datos relacional, Larry Ellison se basa en el modelo relacional y en el lenguaje SQL para adelantar a IBM y desarrollar ORACLE.

A principios de los 80, varías compañías deciden crear sus versiones de bases de datos relacionales, dando lugar a la creación de importantes productos como Informix o Sybase. En la actualidad, muchas empresas dedicadas a este ámbito tienen sus productos basados en DBMS relacionales, como Access de Microsoft, Oracle y Rdb de Oracle, o DB2 de IBM.

Ante el crecimiento exponencial que tuvieron las bases de datos en los años posteriores y las nuevas posibilidades de representar consultas a bases de datos relacionales surgió la necesidad de reducir costes y optimizar dichas consultas. En este trabajo vamos a introducir los conceptos necesarios para conocer el modelo relacional y vamos a profundizar en la optimización de las consultas a bases de datos.

Abstract

In this document we are going to explain the relational model of databases and the operations of relational algebra that will allow us to manage the data entered in a relational database and obtain the results of queries made in a high-level language, such as SQL. Finally, we will see the implementation of these operations and their optimization to reduce the query's costs as much as possible.

In Chapter 1, we define the basic concepts that will allow us to understand the relational model. These are as follows:

- Domain D: Set of indivisible values that refer to all attributes.
- Attribute A: Name of the role played by a specific domain D in our relationship scheme.
- Relationship $r(R)$: Set of n-tuples, where each tuple t_i is an ordered list of n values.
- Relational scheme $R(A1,\ldots,An)$: Element that describes a relation formed by the name of the relation R and a list of attributes.
- Relational database schema S: Set of every relationship schemas contained in the relational database and the integrity constraints that all relationships must comply with.
- State of relational database DB: Set of all relations in the relational database that meet the integrity constraints.

In Chapter 2, we delve into relational algebra operations that allow us to manipulate the database. We can divide operations into two types: specific database operations and set operations.

These are the three specific database operations:

- SELECT $\sigma_{selection\ condition}(R)$: Select a subset of tuples from a relation that satisfy the selection conditions.
- PROJECT $\pi_{<listadeatributos>}(R)$: Select certain attributes of a relationship and discard the others.
- JOIN $R \bowtie_{condition} S$: Combine related tuples from two different relationships into one to get results from it.

The operations used in relational algebra are the following ones:

- CARTESIAN PRODUCT $R \times S$: Combines each tuple of a relation with all the tuples of another relation, as long as they are a compatible union.
- UNION $R \cup S$: Includes in the result every tuples that are in the relation R, in S or in both.
- INTERSECTION $R \cap S$: Includes in the result the tuples that are contained in the relations R and S at the same time.
- MINUS $R \setminus S$: Includes in the results the tuples that are in contained R but not in S.

In this chapter we also analyze the properties that the different operations could have, such as commutativity and associativity.

At the beginning of Chapter 3, we explain what is the process that a query is subjected to since it has been written in a high-level language, through the analysis and validation by the RDBMS, the optimization and the execution of that query, until the required result is obtained. In this section we also explain the proper way to translate a query in SQL language to relational algebra and its prior division into query blocks to facilitate the implementation task.

Next, we define the indexing structures, which allow us to access the data in a faster and more efficient way. After thar, we also define the algorithms that let us to order and organize the blocks of queries prior to the implementation of relational algebra operations.

In the next section, we study the different implementation methods for the execution of each operation that we have already seen, accompanied by specific algorithms that we use in the practice to execute the queries.

To finish the chapter, we see the definition of heuristic, which is the set of methods and techniques used in order to find a solution to a problem in those cases where it is not easy to find an optimal solution, and we apply it to the query optimization to reduce execution costs. To better understand this optimization, we see how to represent a query as a query tree and we study heuristic transformation rules to apply them to relational algebra operations. This transform our initial tree into a tree whose execution is more efficient.

Finally, in chapter 4 we explain two examples of database queries and how they are optimized until we obtain the appropriate query tree to implement it and thus obtain the required result with the least possible cost.

Índice general

Capítulo 1

El modelo relacional

1.1. Conceptos básicos

El modelo relacional es un modelo matemático de bases de datos basado en la teoría de conjuntos y en la lógica de predicado de primer orden. Este modelo se fundamenta en la representación de datos como un conjunto de relaciones, cuyas filas representa un hecho que, por lo general, corresponde con una relación o entidad.

A continuación vamos a definir con más detalle los elementos que aparecen en un modelo relacional.

- **Dominio** D:

  Conjunto de valores atómicos (indivisibles en lo que al modelo se refiere), que hacen referencia a todos los atributos.
  La forma más usual de especificar un dominio D es indicar el nombre, que ayuda a la interpretación de sus valores, el tipo de datos, un formato para los datos y una información adicional, si se requiere.
  Por ejemplo, el dominio NumerosTelefonosEspaña es el conjunto de 9 dígitos que componen los números de teléfono de España. Otro ejemplo es el dominio EstacionesAño, que es el conjunto de las siguientes cadenas de caracteres "Primavera", "Verano", "Otoño", "Invierno"

- **Atributo** A_i:

  El nombre del papel que juega un dominio D concreto en nuestro esquema de relación.
  Diremos que D es dominio del atributo A_i, y escribiremos $Dom(A_i)$.
  Llamaremos **grado** de una relación al número de atributos n de la misma.
  Diremos que un atributo es **clave** si tiene un valor único para cada tupla.

- **Relación** $r(R)$:

  Es un conjunto de n-tuplas $r(R) = [t_1, ..., t_m]$, donde cada tupla t_i es una lista ordenada de n valores $< v_1, ..., v_n >$, con v_i un elemento de $dom(A_i)$, o en su defecto, un valor de tipo **NULL**, es decir, que no se conoce o no existe.
  A niveles prácticos, una tupla es interpretada como una instancia de un esquema de relación.

- **Esquema de relación** $R(A_1, ..., A_n)$:

  Es el elemento que describe una relación. Está formado por el nombre de la relación R y una lista de atributos $A_1, ..., A_n$.

Estas definiciones se aplican a relaciones individuales y a sus atributos, pero la realidad es que una base de datos relacional contiene varios esquemas de relación, cuyas relaciones pueden estar relacionadas entre ellas.
A continuación, vamos a dar dos definiciones importantes:

- **Esquema de base de datos relacional** S:

  Es el conjunto de todos los esquemas de relación que contiene la base de datos relacional, $S = [R_1, ..., R_n]$ y de las restricciones de integridad RI que deben cumplir todas las relaciones.

- **Estado de base de datos relacional** DB:

  Es un conjunto de relaciones $DB = [r_1, ..., r_n]$, donde cada r_i es una relación del esquema de relación R_i y satisface las restricciones de integridad RI.

1.2. Ejemplo de esquema de base de datos relacional

En esta sección, una vez explicados todos los elementos que aparecen en el modelo relacional, vamos a presentar un esquema de base de datos relacional, denominado EMPRESA, formado por varios esquemas de relación junto con sus relaciones asociadas. Utilizaremos este ejemplo a lo largo del trabajo.

Nombre	Apell1	Apell2	Dni	FechaNac	Direccion	Sexo	Sueldo	SuperDni	Dno
José	Pérez	Pérez	123456789	01-09-1965	Eloy I, 98	H	30000	333445555	5
Alberto	Campos	Sastre	333445555	08-12-1956	Avda. Ríos, 9	H	40000	888665555	5
Alicia	Jiménez	Celaya	999887777	12-05-1968	Gran Vía, 38	M	25000	987654321	4
Juana	Sainz	Oreja	987654321	20-06-1971	Cerquillas, 67	M	43000	888665555	4
Fernando	Ojeda	Ordóñez	666884444	15-09-1962	Portillo, s/n	H	38000	333445555	5
Aurora	Oliva	Avezuela	453453453	31-07-1972	Antón, 6	M	25000	333445555	5
Luis	Pajares	Morera	987987987	29-03-1969	Enebros, 90	H	25000	987654321	4
Eduardo	Ochoa	Paredes	888665555	10-11-1987	Las Peñas, 1	H	55000	NULL	1

Figura 1.1: EMPLEADO

NombreDpto	NumeroDpto	DniDirector	FechaIngresoDirector
Investigación	5	333445555	22-05-1988
Administración	4	987654321	01-01-1995
Sede Central	1	8886655555	19-06-1981

Figura 1.2: DEPARTAMENTO

NombreProyecto	NumProyecto	UbicacionProyecto	NumDptoProyecto
Producto X	1	Valencia	5
Producto Y	2	Sevilla	5
Producto Z	3	Madrid	5
Computación	10	Gijón	4
Reorganización	20	Madrid	1
Comunicaciones	30	Gijón	4

Figura 1.3: PROYECTO

Nombre	Apell1	Apell2	Dni	FechaNac	Direccion	Sexo	Sueldo	SuperDni	Dno
Alberto	Fonseca	García	789456123	06-12-1948	Río Duero, 29	H	25000	333445555	1
Pilar	Vera	Luna	321654987	31-06-1952	Pablo Gargallo, 56	M	28000	333445555	5
Miguel	Menéndez	Moreno	741852963	10-03-1949	Padre Consolación, 9	H	20000	987654321	4
Joaquín	Duarte	Barrenas	369258147	01-01-1937	Paseo las Damas, 25	H	52000	NULL	1

Figura 1.4: EMPLEADO_JUBILADO

DniEmpleado	NumProy	Horas
123456789	1	32,5
123456789	2	7,5
666884444	3	40,0
453453453	1	20,0
333445555	2	10,0
333445555	3	10,0
333445555	10	10,0
333445555	20	10,0
999887777	30	30,0
987987987	10	35,0
987987987	30	5,0
987654321	30	20,0
888665555	20	NULL

Figura 1.5: TRABAJA_EN

En este esquema de base de datos relacional podemos observar cinco esquemas de relación: Empleado, Departamento, Proyecto, Empleado_jubilado y Trabaja_en.
El esquema de relación Empleado contiene una relación formada por ocho instancias (tuplas) que representan a ocho empleados cuyos atributos son: Nombre, Apell1, Apell2, Dni, FechaNac, Direccion, Sexo, Sueldo, SuperDni y Dno.
Cada uno de los atributos tiene un dominio. Por ejemplo, el atributo Sueldo es el conjunto de los números reales positivos.
El esquema de relación Departamento tiene una relación formada por tres instancias (tuplas) que representan tres departamentos cuyos atributos son: NombreDpto, NumeroDpto, DniDirector y FechaIngresoDirector.
Proyecto tiene una relación formada por seis instancias que representan seis proyectos diferentes que lleva a cabo la empresa cuyos atributos son: NombreProyecto, NumProyecto, UbicacionProyecto y NumDptoProyecto.
Empleado_jubilado tiene una relación formada por cuatro instancias que representan cuatro empleados de la empresa que ya se han jubilado cuyos atributos son los mismos que los del esquema de relación Empleado.
Por último, Trabaja_en contiene dieciséis tuplas que representa cada una de ellas el estado formado por el Dni de un empleado que se encuentra trabajando en un proyecto concreto. Los atributos de este esquema de relación son: DniEmpleado, NumProy y Horas.

Capítulo 2

El álgebra relacional y los cálculos relacionales

En el capítulo anterior, vimos los elementos necesarios para definir una estructura de datos. Pero eso no es todo, para elaborar un modelo de datos, debemos incluir un conjunto de operaciones para manipular la base de datos, las cuales permiten al usuario especificar las consultas de recuperación de datos. Este conjunto de operaciones para el modelo relacional recibe el nombre de álgebra relacional.

Se dice que una secuencia de operaciones de álgebra relacional conforma una expresión de álgebra relacional, cuyo resultado será también una nueva relación que representa el resultado de una consulta a la base de datos.

Las operaciones del álgebra relacional pueden dividirse en dos grupos. El primero incluye el conjunto de operaciones de la teoría matemática de conjuntos, que es aplicable gracias a que cada relación está definida como un conjunto de tuplas en el modelo relacional. Algunas de estas operaciones son las siguientes:

UNION, INTERSECTION, MINUS, CARTESIAN PRODUCT...

El otro grupo está formado por las operaciones desarrolladas específicamete para las bases de datos, y son:
SELECT, PROJECT y JOIN.
A continuación vamos a tratar con mayor profundidad algunas de ellas.

2.1. Operación SELECT

Esta operación se emplea para seleccionar un subconjunto de tuplas de una relación que satisfacen una serie de condiciones de selección. Puede visualizarse también como un filtro que mantiene las tuplas que cumplen dichas condiciones y descarta las demás.

En general, SELECT se denota como:

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

Donde σ representa el operador SELECT, la condición es una expresión de tipo booleana y R es el esquema de relación de cuya relación se seleccionan las tuplas.

La condición de selección se aplica independientemente a cada tupla. Esto se realiza sustituyendo cada ocurrencia del atributo A_i en la condición de selección por su valor en la tupla $t[A_i]$. Cuando se ejecuta una consulta, la tupla t se selecciona si la condición de evalúa como verdadero.

Los atributos del resultado de este operador son los mismos que los atributos del esquema de relación R.

La expresión booleana de la condición de selección viene dada por la siguiente estructura:

$< atributo >< operador\ de\ comparación >< constante >$

Donde $< atributo >$ es el nombre de un atributo de R, $< operador\ de\ comparación >$ es uno de los operadores definidos en el conjunto $\{=, <, \leq, >, \geq, \neq\}$ y $< constante >$ es un valor con el cual queremos comparar el valor del atributo seleccionado en cada una de las tuplas.

A continuación, vamos a mostrar un par de ejemplos del operador SELECT, basándonos en la tabla Empleado de la figura 1:

Ejemplo 1

Consulta textual: Selecciona los empleados cuyo sexo es Mujer.
Consulta en álgebra relacional:
$\sigma_{Sexo='M'}(EMPLEADO)$
Este operador selecciona aquellas tuplas de la relación Empleado cuyo atributo Sexo sea igual a 'M':

Nombre	Apell1	Apell2	Dni	FechaNac	Dirección	Sexo	Sueldo	SuperDni	Dno
Alicia	Jiménez	Celaya	999887777	12-05-1968	Gran Vía, 38	M	25000	987654321	4
Juana	Sainz	Oreja	987654321	20-06-1941	Cerquillas, 67	M	43000	888665555	4
Aurora	Oliva	Avezuela	453453453	31-07-1972	Antón, 6	M	25000	333445555	5

Figura 2.1: Resultado de SELECT 1

Ejemplo 2

Consulta textual: Selecciona los empleados cuyo sueldo es distinto de 25000.
Consulta en álgebra relacional:
$\sigma_{Sueldo\neq 25000}(EMPLEADO)$
Este operador nos mostrará aquellas tuplas en las que el atributo Sueldo sea distinto de 25000:

Nombre	Apell1	Apell2	Dni	FechaNac	Dirección	Sexo	Sueldo	SuperDni	Dno
José	Pérez	Pérez	123456789	01-09-1965	Eloy I, 98	H	30000	333445555	5
Alberto	Campos	Sastre	333445555	08-12-1955	Avda. Ríos, 9	H	40000	888665555	5
Juana	Sainz	Oreja	987654321	20-06-1941	Cerquillas, 67	M	43000	888665555	4
Fernando	Ojeda	Ordóñez	666884444	15-09-1962	Portillo, s/n	H	38000	333445555	5
Eduardo	Ochoa	Paredes	888665555	10-11-1937	Las Peñas, 1	H	55000	NULL	1

Figura 2.2: Resultado de SELECT 2

A partir de la condición de seleción que hemos presentado, podemos crear condiciones de selección complejas, que son unión con AND, OR y NOT de condiciones de selección.

- $(condición1\textbf{AND}condición2)$:
  Es VERDADERO si las dos condiciones lo son.
  Es FALSO en caso contrario.

- $(condición1\textbf{OR}condición2)$:
  Es VERDADERO si cualquiera de las dos condiciones lo es.
  Falso en caso contrario.

- $(\textbf{NOT}condición)$:
  Es VERDADERO si la condición es FALSO.
  Es FALSO en otro caso.

Obsérvese que la operación SELECT es conmutativa, y además, podemos combinar una *cascada* de operaciones SELECT en una sola:

Propiedad 1

Sea R un esquema de relación. Se cumple lo siguiente:

$\sigma_{<cond1>}(\sigma_{<cond2>}(...(\sigma_{<condN>}(R))...)) =$

$= \sigma_{<cond1>\textbf{AND}<cond2>\textbf{AND}...\textbf{AND}<condN>}(R)$

Ejemplo 3

Consulta textual: Selecciona los empleados cuyo sexo sea Mujer, cuyo sueldo sea mayor o igual que 30000 y cuyo número de departamento sea distinto de 4.

Consulta en álgebra relacional:

$\sigma_{(Sexo='M')AND(Sueldo\geq 30000)AND(Dno\neq 4)}(EMPLEADO)$

Dicha consulta no nos devolverá ninguna tupla.

Veámoslo paso por paso:

En primer lugar, por la Propiedad 1, la consulta es equivalente a

$\sigma_{(Sexo='M')}(\sigma_{Sueldo\geq 30000}(\sigma_{Dno\neq 4}(EMPLEADO)))$

Al realizar la consulta $\sigma_{(Dno\neq 4)}$ seleccionaremos todas las tuplas cuyo Dno sea distinto de 4:

Nombre	Apell1	Apell2	Dni	FechaNac	Dirección	Sexo	Sueldo	SuperDni	Dno
José	Pérez	Pérez	123456789	01-09-1965	Eloy I, 98	H	30000	333445555	5
Alberto	Campos	Sastre	333445555	08-12-1955	Avda. Ríos, 9	H	40000	888665555	5
Fernando	Ojeda	Ordóñez	666884444	15-09-1962	Portillo, s/n	H	38000	333445555	5
Aurora	Oliva	Avezuela	453453453	31-07-1972	Antón, 6	M	25000	333445555	5
Eduardo	Ochoa	Paredes	888665555	10-11-1937	Las Peñas, 1	H	55000	NULL	1

Figura 2.3: Primer filtro del ejemplo 3

A continuación, de las tuplas devueltas, seleccionamos aquellas cuyo sueldo sea mayor o igual que 30000:

Nombre	Apell1	Apell2	Dni	FechaNac	Dirección	Sexo	Sueldo	SuperDni	Dno
José	Pérez	Pérez	123456789	01-09-1965	Eloy I, 98	H	30000	333445555	5
Alberto	Campos	Sastre	333445555	08-12-1955	Avda. Ríos, 9	H	40000	888665555	5
Fernando	Ojeda	Ordóñez	666884444	15-09-1962	Portillo, s/n	H	38000	333445555	5
Eduardo	Ochoa	Paredes	888665555	10-11-1937	Las Peñas, 1	H	55000	NULL	1

Figura 2.4: Segundo filtro del ejemplo 3

Y por último, de entre las tuplas seleccionadas, mostramos aquellas cuyo Sexo sea 'M':

Nombre	Apell1	Apell2	Dni	FechaNac	Dirección	Sexo	Sueldo	SuperDni	Dno

Figura 2.5: Tercer filtro del ejemplo 3

Notar que independientemente del orden en el que realicemos los filtros, el resultado será siempre el mismo.

2.2. Operación PROJECT

Ésta operación se emplea para seleccionar ciertos atributos y descartar el resto de atributos de una relación. El resultado de la operación PROJECT puede verse como una partición de la relación co-

rrespondiente: Un conjunto que contiene a los atributos seleccionados y otro conjunto que contiene a aquellos atributos descartados.
La forma general de la operación PROJECT es:

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

Donde π representa el operador PROJECT, la lista de atributos es el conjunto de atributos que queremos obtener y R representa la relación de la que se extraen los atributos.

El resultado de la operación PROJECT contiene únicamente los atributos especificados en <lista de atributos>, por lo que su grado es igual al número de atributos incluidos en la lista.

Además, se cumple la característica de eliminación de duplicados. Es decir, el resultado no muestra aquellas tuplas que ya han aparecido con anterioridad.

Por último, veamos qué ocurre con la composición de la operación PROJECT:

$\pi_{<lista1>}(\pi_{<lista2>}(R)) = \pi_{<lista1>}(R)$

Por lo tanto, podemos afirmar que no se trata de una operación conmutativa.

Ejemplo 4

Consulta textual: Selecciona todos los nombres, DNI y sexos correspondientes a todos los empleados.

Consulta en álgebra relacional:

$\pi_{Nombre,Dni,Sexo}(EMPLEADO)$

Esta consulta selecciona todas las combinaciones de los atributos Nombre, DNI y Sexo que hay en el esquema de relación EMPLEADO:

Nombre	Dni	Sexo
José	123456789	H
Alberto	333445555	H
Alicia	999887777	M
Juana	987654321	M
Fernando	666884444	H
Aurora	453453453	M
Luis	987987987	H
Eduardo	888665555	H

Figura 2.6: Resultado de PROJECT 1

2.3. Operaciones UNION, INTERSECTION y MINUS

Este grupo de operaciones de álgebra relacional son las correspondientes a la operativa matemática sobre conjuntos. Estas operaciones se emplean para combinar los elementos de dos conjuntos. Todas ellas son binarias, es decir, se aplican a dos relaciones o conjuntos de tuplas, que deben tener el mismo tipo de tuplas. Esta condición se denomina compatibilidad de unión.

Definición

Dos relaciones $R(A_1,...,A_n)$ y $S(B_1,...,B_n)$ se dice que son de unión compatible si tienen el mismo grado n y si el $dom(A_i) = dom(B_i)$ para $1 \leq i \leq n$.

Definición

Podemos definir las tres operaciones UNION, INTERSECTION y MINUS en dos relaciones de unión compatible R y S del siguiente modo:

- **UNION**:
  El resultado de la operación UNION es una relación que incluye todas las tuplas que están en R, en S o en ambas. Las tuplas duplicadas se eliminan. Escribimos $R \cup S$.

- **INTERSECTION**:
  El resultado de la operación INTERSECTION es una relación que incluye a las tuplas que están en R y en S al mismo tiempo. Escribimos $R \cap S$.

- **MINUS**:
  El resultado de la operación MINUS es una relación que incluye las tuplas que están en R pero no en S. Escribimos $R \setminus S$.

Propiedades

- Las operaciones UNION e INTERSECTION son conmutativas y asociativas.
  $R \cup S = S \cup R$ y $R \cap S = S \cap R$.
  $(R \cup S) \cup T = R \cup (S \cup T)$ y $(R \cap S) \cap T = R \cap (S \cap T)$.

- La operación MINUS no es conmutativa en general.
  $R \setminus S \neq S \setminus R$.

- La operación INTERSECTION puede expresarse en términos de unión y diferencia de conjuntos:
  $R \cap S = R \cup S \setminus (R \setminus S) \setminus (S \setminus R)$.

2.4. Operación CARTESIAN PRODUCT

Esta operación se emplea para combinar cada tupla de una relación con todas las tuplas de otra relación. Notar que no es necesario que las relaciones en las que se aplica la operación CARTESIAN PRODUCT sean una unión compatible.

La forma general de la operación CARTESIAN PRODUCT entre dos relaciones $R(A_1, A_2, ..., A_n)$ y $S(B_1, B_2, ..., B_m)$ es:

$R(A_1, ..., A_n) \times S(B_1, ..., B_m)$

En general, el resultado de esta operación es una relación Q con un grado de $n + m$ atributos $Q(A_1, ..., A_n, B_1, ..., B_m)$ en este orden. La relación resultante Q tiene una tupla por cada combinación de una tupla de R con una tupla de S. Por lo tanto, si R tiene n tuplas y S tiene s tuplas, Q tendrá $n * s$ tuplas.

Ya que la operación CARTESIAN PRODUCT se suele emplear seguido de una operación SELECT con mucha frecuencia, veamos una operación que permite escribir estas dos operaciones como una operación única.

2.5. Operación JOIN

Esta operación se emplea para combinar tuplas relacionadas de dos relaciones diferentes en una sola y obtener resultados de la misma. Dicho de otra manera, nos permite crear relaciones entre relaciones.
La forma general de la operación JOIN aplicada a dos relaciones $R(A_1, A_2, ..., A_n)$ y $S(B_1, B_2, ..., B_m)$ es:

$R \bowtie_{<condición\ de\ concatenación>} S$

El resultado de la operación JOIN es otra relación Q de $n + m$ atributos $Q(A_1, ..., A_n, B_1, ..., B_m)$ en ese orden. En cuanto al número de tuplas, Q tiene una tupla por cada combinación de una tupla de R y otra de S siempre que dicha combinación satisfaga la condición de concatenación, que está especificada sobre los atributos de las dos relaciones iniciales y es evaluada para cada combinación de tuplas. Si dicha condición es evaluada a VERDADERO para una combinación de tuplas, se incluye en la relación Q una

única tupla combinada. Las tuplas cuyos atributos de conexión son NULL, o aquellas cuya condición de concatenación es evaluada a FALSO, no aparecen en el resultado.
La mayor diferencia de la operación JOIN y la operación CARTESIAN PRODUCT es que en la primera sólo aparecen en el resultado las combinaciones de tuplas que satisfacen la condición de concatenación, mientras que en la operación CARTESIAN PRODUCT se incluyen todas las combinaciones posibles.

Notar también que el resultado de una operación JOIN puede ser una relación vacía. En general, si la relación R tiene r tuplas y la relación S tiene s tuplas, el resultado de la operación JOIN entre dichas relaciones tendrá entre cero y $n * s$ tuplas.

Ejemplo 5

Consulta textual: Seleccionar todos los datos disponibles de los empleados y de los departamentos a los que pertenecen.
Consulta en álgebra relacional: $EMPLEADO \bowtie_{<Dno=NumeroDpto>} DEPARTAMENTO$

Nombre	Apell1	Apell2	Dni	FechaNac	Dirección	Sexo	Sueldo	SuperDni	Dno	NombreDpto	NumeroDpto	DniDirector	FechaIngresoDirector
José	Pérez	Pérez	123456789	01-09-1965	Eloy I, 98	H	30000	333445555	5	Investigación	5	333445555	22-05-1988
Alberto	Campos	Sastre	333445555	08-12-1955	Avda. Ríos, 9	H	40000	888665555	5	Investigación	5	333445555	22-05-1988
Alicia	Jiménez	Celaya	999887777	12-05-1968	Gran Vía, 38	M	25000	987654321	4	Administración	4	987654321	01-01-1995
Juana	Sainz	Oreja	987654321	20-06-1941	Cerquillas, 67	M	43000	888665555	4	Administración	4	987654321	01-01-1995
Fernando	Ojeda	Ordóñez	666884444	15-09-1962	Portillo, s/n	H	38000	333445555	5	Investigación	5	333445555	22-05-1988
Aurora	Oliva	Avezuela	453453453	31-07-1972	Antón, 6	M	25000	333445555	5	Investigación	5	333445555	22-05-1988
Luis	Pajares	Morera	987987987	29-03-1969	Enebros, 90	H	25000	987654321	4	Administración	4	987654321	01-01-1995
Eduardo	Ochoa	Paredes	888665555	10-11-1937	Las Peñas, 1	H	55000	NULL	1	Sede Central	1	8886655555	19-06-1981

Figura 2.7: Resultado JOIN 1

Capítulo 3

Procesamiento y optimización de consultas

En este capítulo vamos a conocer los pasos habituales que se realizan para procesar una consulta en un lenguaje de alto nivel, como por ejemplo SQL, que es el lenguaje de consultas que utiliza la mayoría de las RDBMS (Sistema de Gestión de Bases de Datos Relacionales) comerciales.
También vamos a profundizar en la traducción de consultas SQL a consultas de álgebra relacional, diferentes estrategias para su optimización y algoritmos para implementar las operaciones relacionales.

3.1. Estructura del procesamiento de una consulta de alto nivel

En el momento que se realiza una consulta en un lenguaje de alto nivel, el RDBMS somete la consulta a una serie de procesos hasta obtener el resultado requerido.
Estos procesos son los siguientes:

1. **Análisis léxico, análisis sintáctico y validación**:

   En este paso se analizan los elementos del lenguaje que estemos utilizando y sus palabras reservadas, así como los nombres de los atributos y de las relaciones con las que estamos trabajando.
   A continuación, se determina si la consulta ha sido formulada con arreglo a las reglas sintácticas del lenguaje.
   Por último se comprueba que los nombres de atributos y de las relaciones son válidos y tienen un sentido semántico.
   Con todo esto, se construye una representación interna de la consulta en forma de árbol o grafo de consultas.

2. **Optimización de la consulta**:

   Realiza una elección de una estrategia de ejecución adecuada a nuestra consulta, dando lugar a un plan de ejecución.
   Notar que elegimos una estrategia razonablemente eficiente para ejecutar la consulta, aunque en algunos casos puede no ser la óptima.

3. **Generación del código de la consulta**:

   Genera el código para ejecutar el plan de ejecución obtenido anteriormente.

4. **Procesador de base de datos en tiempo de ejecución**:

   Ejecuta el código dando lugar al resultado de la consulta.
   Si hubiera un error en tiempo de ejecución, genera un mensaje de error en lugar del resultado.

3.2. Traducción de consultas SQL al álgebra relacional

Por regla general, las consultas realizadas en un lenguaje SQL se descomponen en unidades básicas, llamados bloques de consulta, que se pueden traducir a los operadores algebraicos anteriormente mencionados.
Un bloque de consulta está formado por una expresión $SELECT - FROM - WHERE - GROUP\ BY - HAVING$, pudiendo ser omitidas las cláusulas $GROUP\ BY$ y $HAVING$ dependiendo de la consulta.
Estos bloques de consulta pueden ser traducidos de una manera sencilla al álgebra relacional.

Ejemplo 1

Consulta textual: Selecciona los nombres y apellidos de los empleados que trabajen en el departamento cuyo director tenga Dni 888665555.

Consulta en SQL:

SELECT Nombre, Apell1, Apell2
FROM EMPLEADO
WHERE Dno = (**SELECT** NumeroDpto
FROM DEPARTAMENTO
WHERE DniDirector = 888665555)

Descomposición en bloques:

- Bloque 1:

  SELECT NumeroDpto
  FROM DEPARTAMENTO
  WHERE DniDirector = 888665555

- Bloque 2:

  SELECT Nombre, Apell1, Apell2
  FROM EMPLEADO
  WHERE Dno = c
  Donde c es el valor obtenido como resultado del bloque 1.

Traducción al álgebra relacional:

- Bloque 1:

  $\pi_{NumeroDpto}(\sigma_{DniDirector=888665555}(DEPARTAMENTO))$

- Bloque 2:

  $\pi_{Nombre,Apell1,Apell2}(\sigma_{Dno=c}(EMPLEADO))$

3.3. Estructuras de indexación

Para poder continuar, vamos a necesitar una serie de conceptos relacionados con las estructuras de indexación.
Para acelerar la búsqueda de tuplas dentro de una relación, vamos a utilizar lo que llamamos índices. Un índice es una estructura de acceso auxiliar que nos proporciona diferentes rutas de acceso para localizar una o varias tuplas que cumplen ciertos criterios de selección de uno o varios atributos, llamados atributos indexados. El índice almacena todos los valores del atributo indexado junto con una lista de punteros a las diferentes tuplas de nuestro esquema de relación.
Hay diferentes tipos de índices:

- **Índice principal o primario:** El atributo indexado es un atributo clave, por lo que cada valor de dicho atributo en las tuplas es único. En la estructura se almacena cada valor del atributo junto con un puntero a la tupla que tiene dicho valor en el esquema relacional.
- **Índice agrupado:** El atributo indexado es un atributo no clave, es decir, puede haber tuplas que tengan el mismo valor en dicho atributo. En la estructura se almacena cada valor diferente del atributo agrupado junto con un puntero al primer bloque que contiene una tupla con dicho valor del atributo en el esquema relacional.
- **Índice secundario:** Este tipo de índice proporciona un método de acceso secundario a un esquema de relación para el que ya existe un acceso principal. El atributo indexado puede ser clave o puede no serlo. Dependiendo de esta situación, el puntero que acompaña a los valores del atributo indexado será de bloque o de tupla.

3.4. Algoritmos de ordenación externa

La ordenación es uno de los principales algoritmos utilizados en el procesamiento de consultas que requieren ordenar las tuplas de una relación, en especial, aquellos que incluyen operaciones JOIN y en la eliminación de duplicados que se debe realizar en algunos casos con la operación PROJECT.
Diremos que es un algoritmo de ordenación externa si es adecuado para esquemas de relación de tamaño grande, cuyas tuplas no caben en su totalidad en la memoria principal del ordenador (donde se almacenan temporalmente los datos).
Uno de los algoritmos de ordenación externa más utilizados es la estrategia de ordenación-mezcla, que comienza con la división de la relación en pequeños bloques formados por un número determinado de tuplas. A continuación ordena esas tuplas según alguno de sus atributos y, por último, se mezclan estos bloques para obtener bloques ordenados de mayor tamaño. Estos pasos se repiten hasta obtener un sólo bloque ordenado compuesto por todas las tuplas de nuestra relación.

3.4.1. Algoritmo ordenación-mezcla

Descripción del algoritmo de ordenación-mezcla para la ordenación externa:

set $i \leftarrow 1$;
    $j \leftarrow b$; {siendo b el número de tuplas}
    $k \leftarrow n_B$; {siendo n_B el espacio temporal disponible medido en tuplas}
    $m \leftarrow \lceil (j/k) \rceil$; {que representa el número de bloques inicial}
{**Fase de ordenación**}
while $(j \leq m)$
do {
    leer los siguientes k bloques o, si quedan menos de k bloques, leer los restantes;
    ordenar las tuplas y reescribirlas;
    $i \leftarrow i + 1$;
}
{**Fase de mezcla**}
set $i \leftarrow 1$;
    $p \leftarrow \lceil log_{k-1m} \rceil$; {número de pasadas a realizar}
    $j \leftarrow m$;
while $(i \leq p)$
do {
    $n \leftarrow 1$;
    $q \leftarrow \lceil j/(k-1) \rceil$ {nuevo número de bloques a escribir esta pasada}
    while $(n \leq q)$

```
    do {
        leer los siguientes k-1 bloques, o los restantes si quedan menos;
        mezclar los bloques y escribirlos como nuevos bloques;
        n ← n + 1;
    }
    j ← q;      i ← i + 1;
}
```

3.5. Implementación de las operaciones relacionales

3.5.1. Implementación de la operación SELECT

Existen diferentes opciones a la hora de ejecutar una operación SELECT, las cuales vamos a ver a continuación. Utilizaremos los siguientes ejemplos de consultas para ilustrar los métodos de implementación:
OP1: $\sigma_{Dni='123456789'}(EMPLEADO)$
OP2: $\sigma_{NumeroDpto>5}(DEPARTAMENTO)$
OP3: $\sigma_{Dno=5}(EMPLEADO)$
OP4: $\sigma_{(Dno=5)AND(Sueldo>30000)AND(Sexo='F')}(EMPLEADO)$
OP5: $\sigma_{30000\leq Sueldo\leq 35000}(EMPLEADO)$

Métodos de búsqueda en una selección simple

Son algoritmos de búsqueda que exploran las tuplas de la relación sobre la que se ejecuta la consulta y encuentran aquellas tuplas que satisfacen una condición simple de selección.

- **S1-Búsqueda lineal:** Extrae todas las tuplas de la relación y comprueba si sus valores de los atributos satisfacen la condición de selección, obteniendo solamente aquellas tuplas que sí la cumplen.
  Este método puede utilizarse para implementar cualquier consulta SELECT.

- **S2-Búsqueda binaria:** Este método se emplea solamente si la condición de selección es una comparación de igualdad sobre un atributo clave. Extrae las tuplas de la relación cuyo valor del atributo clave coincide con el valor de la condición de selección.
  Un ejemplo de dónde utilizar este método es en **OP1** con el atributo clave Dni = '123456789'.

- **S3-Utilización de un índice primario:** Este método se emplea solamente si la condición de selección es una comparación de igualdad sobre un atributo clave con un índice primario. Extrae la única tupla con el índice primario que cumple la condición de selección.
  Un ejemplo de dónde utilizar este método es **OP1** con Dni = '123456789'.

- **S4-Utilización de un índice primario para encontrar varias tuplas:** Este método se emplea solamente si la condición de comparación es $>, \geq, <$ o $\leq$ sobre un atributo clave con índice primario. En primer lugar, encuentra la tupla con el índice primario que satisface la condición de igualdad. A continuación, extrae las tuplas anteriores o posteriores (dependiendo del tipo de comparación) en la relación ordenada.
  Un ejemplo de dónde utilizar este método es **OP2** con NumeroDpto > 5.

- **S5-Utilización de un índice agrupado para encontrar varias tuplas:** Este método se emplea solamente si la condición de selección es una comparación de igualdad sobre un atributo que no es clave mediante un índice agrupado. Utiliza el índice para extraer todas las tuplas que cumplen la condición.
  Un ejemplo de dónde utilizar este método es **OP3** con Dno = 5.

- **S6-Utilización de un índice secundario sobre una comparación de igualdad:** Este método extrae una única tupla si el atributo con el índice secundario es clave, o extrae varias tuplas si el atributo con el índice secundario no es clave.
  También puede emplearse este método para comparaciones del tipo $>, \geq, <$ o $\leq$. Extrae las tuplas anteriores o posteriores a las seleccionadas en el paso anterior.

Notar que los métodos **S4** y **S6** pueden utilizarse para extraer tuplas a través de consultas de rango. Por ejemplo, en la consulta **OP5**.

Métodos de búsqueda en selecciones complejas

Son algoritmos de búsqueda que se utilizan cuando la condición de selección de una operación SELECT es conjuntiva (está formada por varias condiciones simples conectadas mediante una conjunción lógica).

- **S7-Selección conjuntiva utilizando un índice individual:** Este método se emplea si una condición simple de la condición conjuntiva permite el uso de los métodos de búsqueda en una selección simple. Extrae las tuplas que cumplen dicha condición y comprueba si cada registro satisface el resto de condiciones.
  Un ejemplo de dónde utilizar este método es **OP4**.
- **S8-Selección conjuntiva utilizando un índice compuesto:** Este método se emplea si dos o más atributos forman parte de las condiciones de igualdad de la condición conjuntiva y existe un índice compuesto. Extrae las tuplas cuyos índices cumplen la condición conjuntiva.

En estos métodos, cuando más de uno de los atributos implicados en la condición es clave con índice simple o compuesto, es necesario elegir el índice que extraiga el menor número de tuplas del modo más eficaz para optimizar la consulta.

Definición

Definimos selectividad de una condición de selección como la relación entre el número de tuplas que satisfacen dicha condición y el número total de tuplas de la relación.
De esta manera, cuanto menor sea la selectividad, más recomendable es tener en cuenta esa condición en primer lugar para extraer las tuplas requeridas en la consulta.

3.5.2. Implementación de la operación JOIN

En este apartado veremos diferentes opciones para ejecutar una operación de concatenación entre dos esquemas de relación de la forma $R \bowtie_{A=B} S$, donde R y S son esquemas de relación y A y B son atributos compatibles en sus dominios. Utilizaremos los siguientes ejemplos de consultas para ilustrar los métodos de implementación:
OP6: $EMPLEADO \bowtie_{Dno=NumeroDpto} DEPARTAMENTO$
OP7: $DEPARTAMENTO \bowtie_{DniDirector=Dni} EMPLEADO$

Métodos de concatenación

Estos son los distintos métodos utilizados para la concatenación de dos relaciones:

- **J1-Concatenación de bucle anidado:** Para cada tupla t de R obtiene todas las tuplas s_i de S y, posteriormente, comprueba si las dos tuplas satisfacen la condición de concatenación $t[A] = s_i[B]$. Este método puede utilizarse para implementar cualquier consulta JOIN.

- **J2-Concatenación de bucle simple:** Este método se emplea si existe un índice para uno de los atributos de la concatenación. Extrae todas las tuplas t de R individualmente y se utiliza el índice simple para obtener las tuplas s_i de S que cumplen $s_i[B] = t[A]$.
- **J3-Concatenación de ordenación mezcla:** En primer lugar, si las relaciones no se encuentran ordenadas, este método utiliza un algoritmo de ordenación externa para ordenarla según los atributos de concatenación. A continuación, las dos relaciones son exploradas concurrentemente siguiendo el orden y empareja las tuplas que tienen los mismos valores $t[A] = s[B]$.

3.5.3. Implementación de la operación PROJECT

Para ejecutar una operación PROJECT $\pi_{<lista\ de\ atributos>}(R)$ distinguiremos dos casos: si $< lista\ de\ atributos >$ incluye una clave de la relación R o si no incluye una clave de R.
En el primer caso, tendrá una fácil implementación, ya que el resultado será todas las tuplas de R pero únicamente con los valores de los atributos incluidos en la lista de la propia consulta.
En caso de no incluir una clave de R, se deben eliminar las tuplas duplicadas una vez descartados los atributos que no aparezcan en la lista de la operación. Para realizar este procedimiento existen dos métodos:

- Ordenando el resultado de la consulta y eliminando las tuplas duplicadas, que son fáciles de localizar ya que aparecen consecutivamente.
- Obteniendo una a una las tuplas y extrayendo únicamente aquellas que no son duplicadas de otras ya extraídas anteriormente.

3.5.4. Implementación de las operaciones de conjunto

En este apartado veremos los métodos de ejecución de las operaciones de conjunto UNION, INTERSECTION, MINUS y CARTESIAN PRODUCT.
En particular, la operación CARTESIAN PRODUCT incluye una tupla para cada combinación de tuplas de R y S, por lo que será muy costosa. Por lo tanto, conviene sustituir esta operación por otras equivalentes a ella durante la optimización.
Las otras tres operacones de conjunto se aplican exclusivamente a relaciones compatibles en unión, es decir, que tengan el mismo número de atributos y los mismos dominios de atributo. Para la implementación de estas operaciones se utilizan los siguientes métodos basados en variaciones del método ordenación-mezcla.

- **UNION** ($R \cup S$): Ordena las relaciones R y S, las mezcla de forma concurrente y extrae todas las tuplas exceptuando cuando hay una tupla que se encuentra duplicadas, en cuyo caso sólo extrae una de ellas.
- **INTERSECTION** ($R \cap S$): Ordena las relaciones R y S, las mezcla de forma concurrente y extrae aquellas tuplas que aparecen duplicadas (únicamente una de ellas).
- **MINUS** ($R \setminus S$): Selecciona una a una las tuplas de R y comprueba si están en S, extrayendo únicamente aquellas que no están en S.

3.6. Utilización de la heurística en la optimización de consultas

Para mejorar la ejecución de una consulta de alto nivel, el analizador genera en primer lugar una representación interna inicial, que posteriormente es optimizada para, finalmente, generar un plan de ejecución adecuado y cuyos costes sean mínimos. En esta sección vamos a explicar técnicas de optimización basadas en reglas heurísticas para modificar las representaciones internas de una consulta.

En primer lugar, definamos el concepto de heurística.

3.6.1. Definición de heurística

La heurística es el conjunto de métodos y técnicas que se emplean con el fin de encontrar una solución para un problema en aquellos casos que es difícil hallar una solución óptima o satisfactoria.

En particular, definimos reglas heurísticas como aquellos procedimientos que señalan los medios para resolver un problema.

3.6.2. Árboles de consultas

Normalmente, las representaciones internas de una consulta se encuentran en forma de un árbol de consultas.

Definimos árbol de consultas como una estructura de datos en forma de árbol que equivale a una expresión de álgebra relacional, donde se representan las relaciones de entrada como nodos hoja del árbol y las operaciones del álgebra relacional como nodos internos.

Para ejecutar un árbol de consultas, ejecutamos las operaciones de un nodo interno siempre que estén disponibles sus operandos y, posteriormente, reemplazamos ese nodo por el resultado de la ejecución de dicha operación. Este paso se repite con todos los nodos internos hasta ejecutar el nodo raíz, donde se genera la relación resultante de la consulta.

Ejemplo

Consulta textual: Selecciona para cada proyecto localizado en Gijón el número de proyecto, el número de departamento controlador, el primer apellido del responsable del departamento, su dirección y su fecha de nacimiento.

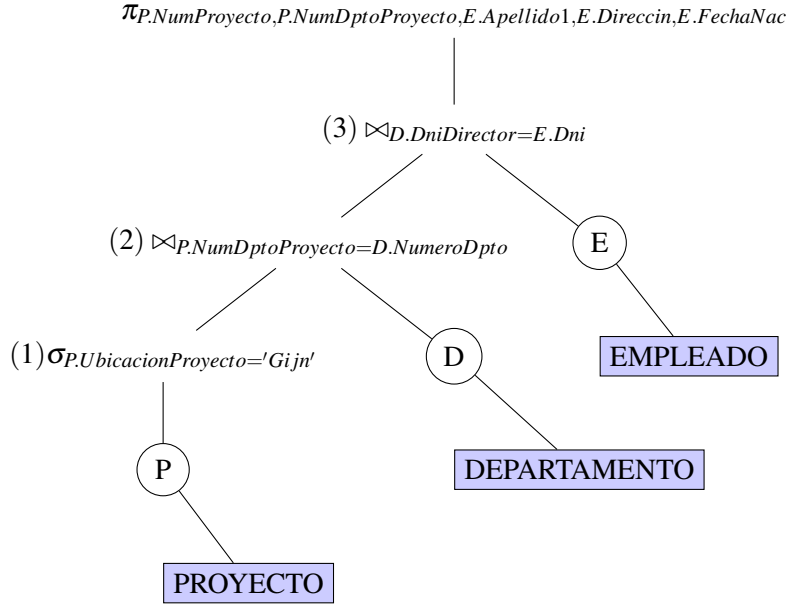
Consulta en SQL:

SELECT P.NumProyecto, P.NumDptoProyecto, E.Apellido1, E.Direccion, E.FechaNac
FROM PROYECTO **AS** P, DEPARTAMENTO **AS** D, EMPLEADO **AS** E
WHERE P.NumDptoProyecto=D.NumeroDpto **AND** D.DniDirector=E.Dni **AND** P.UbicacionProyecto='Gijón';

Consulta en álgebra relacional:

$\pi_{NumProyecto,NumDptoProyecto,Apellido1,Direccion,FechaNac}(((\sigma_{UbicacionProyecto='Gijn'}(PROYECTO)) \bowtie_{NumDptoProyecto=NumeroDpto} (DEPARTAMENTO)) \bowtie_{DniDirector=Dni} (EMPLEADO))$

Árbol de consultas:



Notar que al ejecutar esta consulta, el nodo interno (1) debe de ejecutarse antes que el (2), ya que se necesitan las tuplas del resultado de (1) para poder utilizarlas en la operación del nodo (2). De la misma manera, debemos ejecutar el nodo interno (2) antes que el (3). Por lo tanto, podemos afirmar que un árbol de consulta establece un orden concreto de ejecución de las operaciones.

3.6.3. Optimización heurística de los árboles de consultas

En general, podemos afirmar que existen diferentes expresiones en álgebra relacional que son equivalentes, es decir, que representan la misma consulta. Esto ya lo vimos en el caso concreto de la propiedad 1 del capítulo 2.
A la hora de ejecutar una consulta, en primer lugar, el analizador de consultas genera un árbol de consultas inicial que representa una expresión en álgebra relacional que resulta ser muy ineficaz si se ejecuta directamente por norma general. Gracias a la optimización heurística podemos transformar este árbol inicial en otro árbol de consultas equivalente cuya ejecución sea más eficiente y en la cuál se reducirán los costes.
Esto se puede realizar gracias a las llamadas reglas de optimización heurística de consultas que utilizarán expresiones de álgebra relacional que puedan ser aplicadas al árbol de consultas inicial para transformarlo en el árbol de consultas final ya optimizado.

Reglas de transformación para las operaciones del álgebra relacional

En esta sección estudiaremos las reglas de optimización heurísticas. Notar que aunque el orden de los atributos no sea el mismo en los resultados, diremos que las expresiones son equivalentes ya que la información obtenida es la misma.

1. **Cascada de σ:** Una condición de selección conjuntiva puede dividirse en una cascada de operaciones SELECT:
   $\sigma_{c1\ AND\ c2\ AND\ ...\ AND\ cn}(R) \equiv \sigma_{c1}(\sigma c2(...(\sigma_{cn}(R))...))$

2. **Conmutatividad de σ:** La operación SELECT es conmutativa:
   $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$

3. **Cascada de π:** Podemos sustituir una cascada de operaciones PROJECT por la última de las operaciones:
   $\pi_{Lista1}(\pi_{Lista2}(...(\pi_{ListaN}(R)))) \equiv \pi_{Lista1}(R)$

4. **Conmutación de σ por π:** Podemos conmutar una operación SELECT y una operación PROJECT siempre y cuando la condición de selección incluya únicamente los atributos en la lista de proyección:
   $\pi_{A1,\ A2,...,An}(\sigma_c(R)) \equiv \sigma_c(\pi_{A1,\ A2,...,An}(R))$

5. **Conmutatividad de $\bowtie$ (y de $\times$):** Las operaciones JOIN y CARTESIAN PRODUCT son conmutativas:
   $R \bowtie_c S \equiv S \bowtie_c R$
   $R \times S \equiv S \times R$

6. **Conmutación de σ con $\bowtie$ (o $\times$):** Podemos conmutar una operación SELECT y una operación JOIN (o CARTESIAN PRODUCT) siempre y cuando todos los atributos de la condición de selección involucran únicamente a los atributos de una de las relaciones a las que se aplica la concatenación:
   $\sigma_c(R \bowtie S) \equiv (\sigma_c(R)) \bowtie S$
   De la misma forma, podemos conmutar dichas operaciones siempre y cuando la condición de selección sea conjuntiva $(c1\ AND\ c2)$, donde $c1$ incluye sólo los atributos de R y $c2$ incluye sólo los atributos de S:
   $\sigma_c(R \bowtie S) \equiv (\sigma_{c1}(R)) \bowtie (\sigma_{c2}(S))$

7. **Conmutación de π con $\bowtie$ (o $\times$):** Sea $L = \{A1,...,An,\ B1,...,Bm\}$, donde $A1,...,An$ son atributos de R y $B1,...,Bn$ son atributos de S. Podemos conmutar una operación PROJECT con una operación JOIN siempre y cuando la condición de concatenación c incluya únicamente los atributos de L:

$\pi_L(R \bowtie_c S) \equiv (\pi_{A1,...,An}(R)) \bowtie_c (\pi_{B1,...,Bn}(S))$
En el caso de que c incluya atributos $A_{n+1},...,A_{n+k}$ de R y $B_{m+1},...,B_{m+p}$ de S que no estén en L, se necesita una operación PROJECT adicional para poder aplicar la conmutaividad:
$\pi_L(R \bowtie_c S) \equiv \pi_L((\pi_{A1,...,An,An+1,...,An+k}(R)) \bowtie_c (\pi_{B1,...,Bm,Bm+1,...,Bm+p}(S)))$
Además, siempre podemos conmutar una operación PROJECT con una operación CARTESIAN PRODUCT:
$\pi_L(R \times S) \equiv (\pi_{A1,...,An}(R)) \times (\pi_{B1,...,Bn}(S))$

8. **Conmutatividad de las operaciones de conjunto $\cup$ y $\cap$:** Las operaciones de conjunto UNION e INTERSECTION son conmutativas de manera individual, pero MINUS no lo es:
$R \cup S \equiv S \cup R$
$R \cap S \equiv S \cap R$
$R \setminus S \not\equiv S \setminus R$

9. **Asociatividad de $\bowtie$, $\times$, $\cup$ y $\cap$:** Las operaciones JOIN, CARTESIAN PRODUCT, UNION e INTERSECTION son asociativas. Sea θ cualquiera de estas cuatro operaciones:
$(R\theta S)\theta T \equiv R\theta(S\theta T)$

10. **Conmutatividad de σ con las operaciones de conjunto:** Podemos conmutar una operación SELECT con una operación UNION, INTERSECTION o MINUS. Sea θ cualquiera de las tres operaciones de conjunto:
$\sigma_c(R\theta S) \equiv (\sigma_c(R))\theta(\sigma_c(S))$

11. **Conmutatividad de π con $\cup$:** Podemos conmutar una operación PROJECT con una operación UNION:
$\pi_L(R \cup S) \equiv (\pi_L(R)) \cup (\pi_L(S))$

12. **Conversión de una secuencia $(\sigma, \times)$ en $\bowtie$:** Podemos convertir una secuencia $(\sigma, \times)$ en una operación JOIN siempre y cuando la condición de selección de la operación SELECT que sigue a una operación CARTESIAN PRODUCT corresponda a una operación de concatenación:
$(\sigma_c(R \times S)) \equiv (R \bowtie_c S)$

3.6.4. Algoritmo de optimización algebraica heurística

Una vez conocidas todas las reglas de optimización heurística, estamos en disposición de explicar un algoritmo que se base en ellas con el objetivo de transformar un árbol de consultas inicial en uno ya optimizado de forma que se ejecute de una manera más eficiente en la mayor parte de los casos. Los pasos son los siguientes:

1. **Utilizar la regla 1:** Dividir las operaciones SELECT conjuntivas en una cascada de operaciones SELECT.

2. **Utilizar las reglas 2,4,6 y 10:** Estas reglas, gracias a las propiedades conmutativas, nos permiten desplazar las operaciones SELECT hacia la parte inferior del árbol, tanto como lo permitan los atributos de la condición de selección, para que sean ejecutadas lo antes posible.

3. **Utilizar las reglas 5 y 9:** Por la conmutatividad y la asociatividad de las operaciones de concatenación, podemos reordenar las relaciones de los nodos hoja utilizando los siguientes criterios:
En primer lugar, recolocar las relaciones de nodos hoja con las operaciones SELECT más restrictivas de forma que sean ejecutadas lo antes posible.
Y por último, asegurarse de que la ordenación anterior de los nodos hoja no produce ninguna operación CARTESIAN PRODUCT, y si es necesario cambiar el orden de los nodos para evitarlo.

4. **Utilizar la regla 12:** Combinar operaciones CARTESIAN PRODUCT con la siguiente operación SELECT para obtener operaciones JOIN en el caso de que la condición de selección represente una condición de concatenación.

5. **Utilizar las reglas 3,4,7 y 11:** Dividir las operaciones PROJECT y desplazar las listas de atributos de proyección lo más abajo posible en el árbol.

6. **Agrupar en subárboles:** Identificar los grupos de operaciones que se puedan ejecutar mediante un sólo algoritmo.

Resumen

En resumen, aplicar las diferentes reglas heurísticas para optimizar una consulta consiste en lo siguiente: Aplicar en primer lugar las operaciones que reducen el tamaño de los resultados intermedios (SELECT, que reduce el número de tuplas, y PROJECT, que reduce el número de atributos). En segundo lugar, ejecutar antes aquellas operaciones SELECT y JOIN que sean más restrictivas, y evitar las operaciones CARTESIAN PRODUCT. Y por último, aplicar los algoritmos ya descritos para implementar los subárboles obtenidos para ejecutar la consulta de una forma eficiente.

Capítulo 4

Ejemplos prácticos de optimización de una consulta

En este último capítulo vamos a partir de una consulta textual, veremos cuál sería su equivalente en lenguaje SQL, para después traducirla al álgebra relacional, crear su árbol de consultas inicial que generaría el analizador y aplicaremos las diferentes reglas heurísticas para optimizar la consulta y obtener un árbol de consultas final cuya futura ejecución sea lo más eficaz posible en la mayoría de los casos.

Consulta textual: Selecciona los apellidos de los empleados nacidos después de 1957 que trabajan en un proyecto llamado 'Aquarius'.

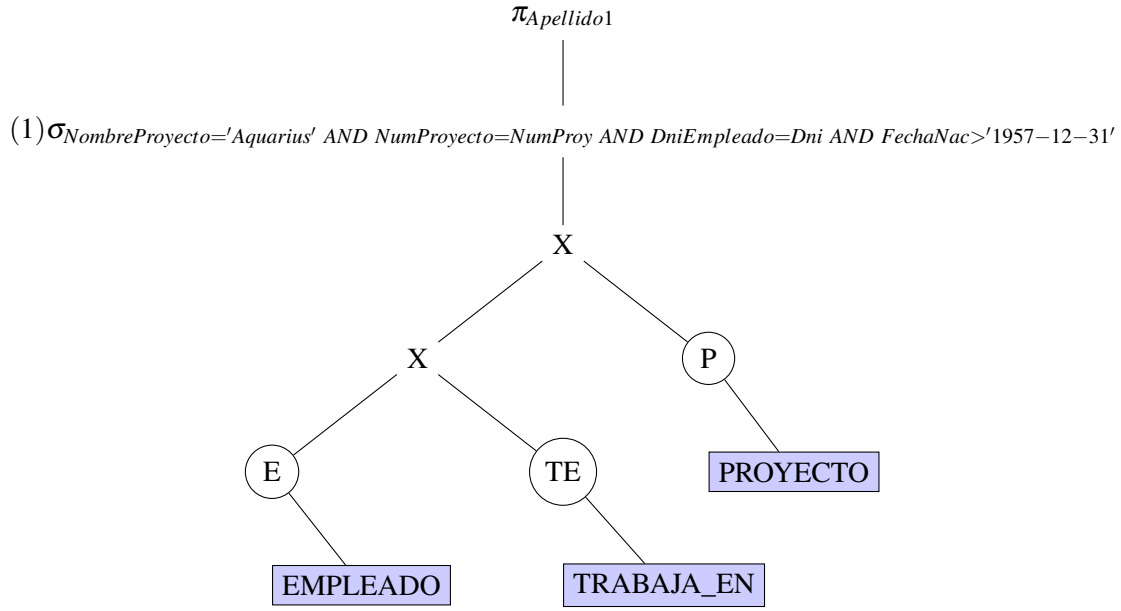
Consulta en SQL:

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

Consulta en álgebra relacional:

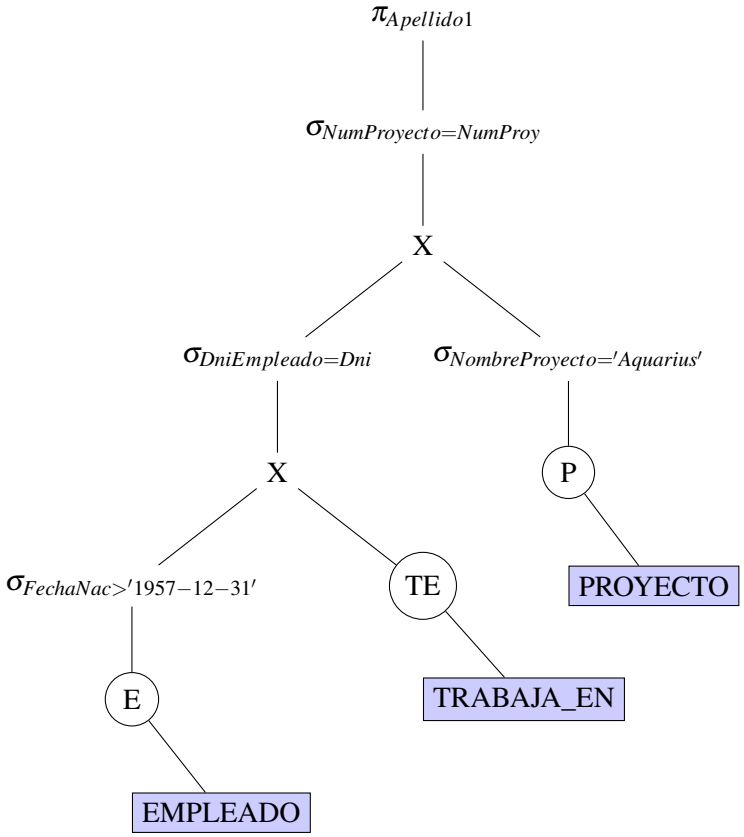
$\pi_{Apellido1}(((\sigma_{NombreProyecto='Aquarius'}(PROYECTO)) \bowtie_{NumProyecto=NumProy} (TRABAJA\_EN)) \bowtie_{DniEmpleado=Dni} (\sigma_{FechaNac>'1957-12-31'}(EMPLEADO)))$

Árbol de consultas inicial:



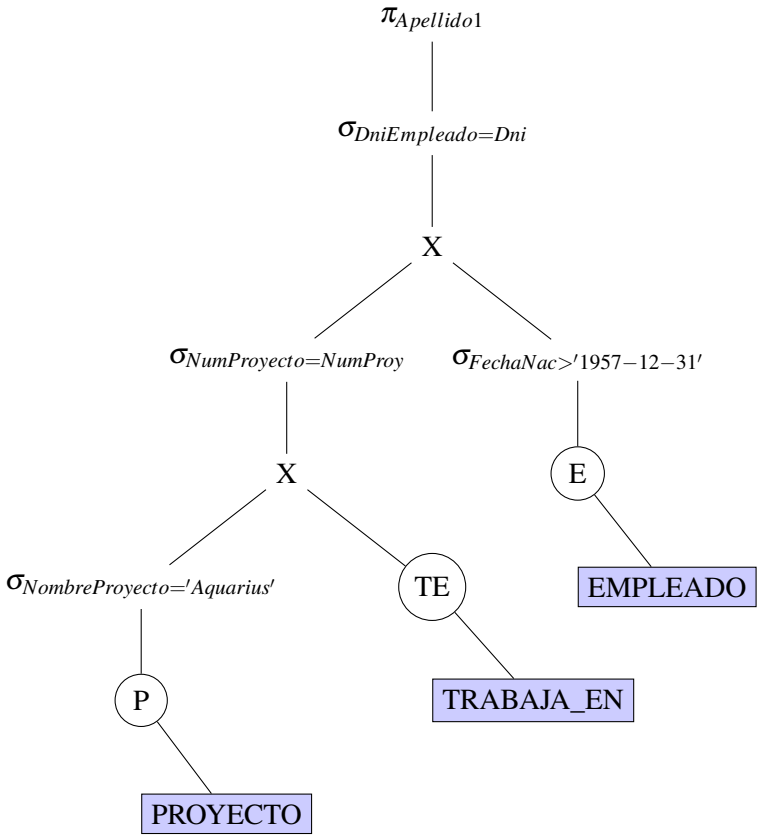
A continuación comenzamos a aplicar las reglas de optimización heurística. En primer lugar, utilizamos la regla 1 para descomponer la operación SELECT conjuntiva (1) en una cascada de operaciones SELECT y utilizamos las reglas 2 y 6 para desplazar las operaciones SELECT resultantes hacia la parte inferior del árbol de consultas.

Árbol de consultas 2



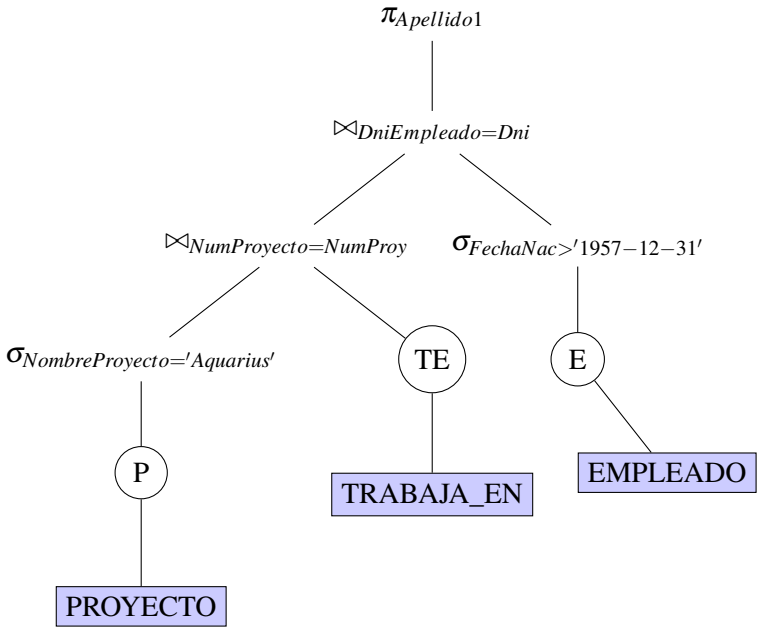
Ahora, aplicamos la regla 5 para asegurarnos de que se ejecuta antes las operaciones SELECT más restrictivas.

Árbol de consultas 3



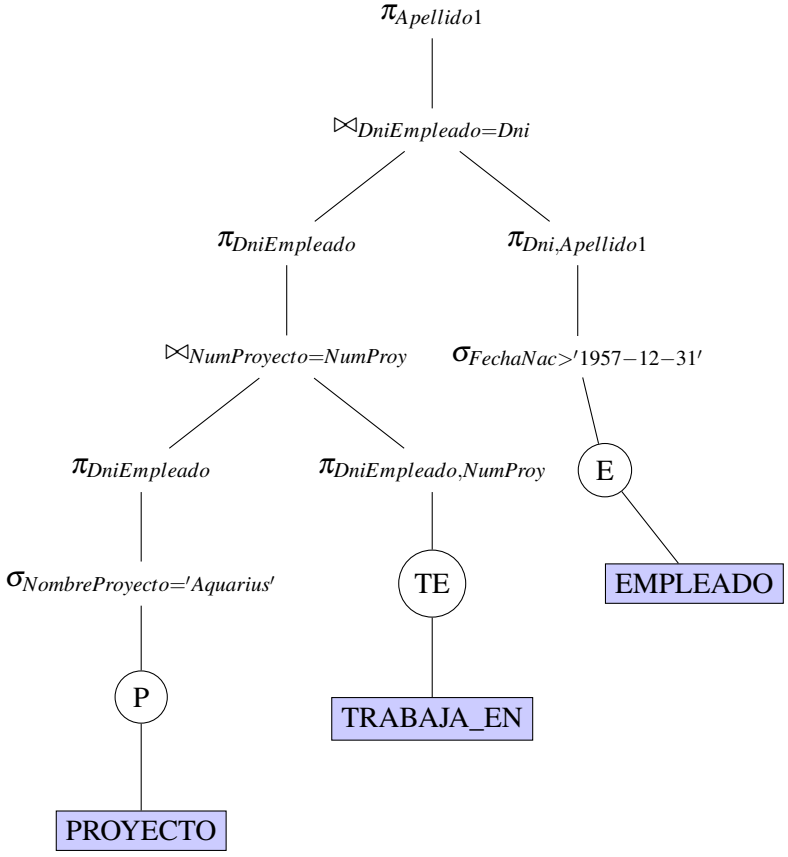
El siguiente paso consiste en aplicar la regla 12 para reemplazar las combinaciones de operaciones CARTESIAN PRODUCT y SELECT por operaciones JOIN.

Árbol de consultas 4



Por último, aplicaremos al árbol 4 las reglas de optimización 3, 4 y 7 para crear las operaciones PROJECT necesarias que nos faciliten la ejecución y desplazarlas hacia abajo en el árbol.

Árbol de consultas final



Éste sería el árbol de consultas optimizado para obtener la ejecución más eficaz para la consulta inicial.

Veamos un segundo ejemplo en el que utilizaremos el resto de reglas heurísticas estudiadas en el capítulo anterior para optimizar la siguiente consulta:

Consulta textual: Selecciona el Dni y la dirección de todos los empleados que han trabajado a lo largo de la historia de la empresa en el departamento de Administración.

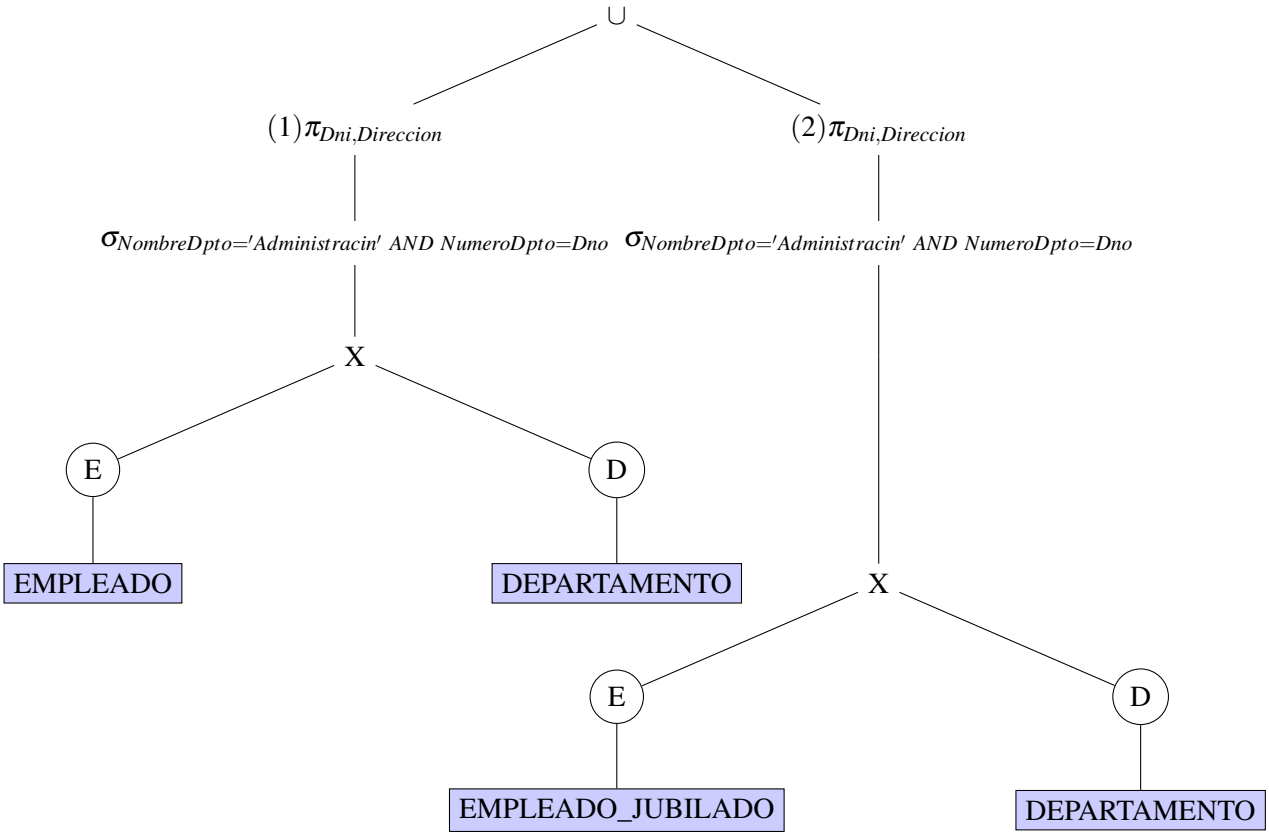
Consulta en SQL:

SELECT Dni, Direccion
FROM EMPLEADO **AS** E, DEPARTAMENTO **AS** D
WHERE NumeroDpto = Dno **AND** NombreDpto = Administración;
UNION
SELECT Dni, Direccion
FROM EMPLEADO_JUBILADO **AS** EJ, DEPARTAMENTO **AS** D
WHERE NumeroDpto = Dno **AND** NombreDpto = Administración;

Consulta en SQL:

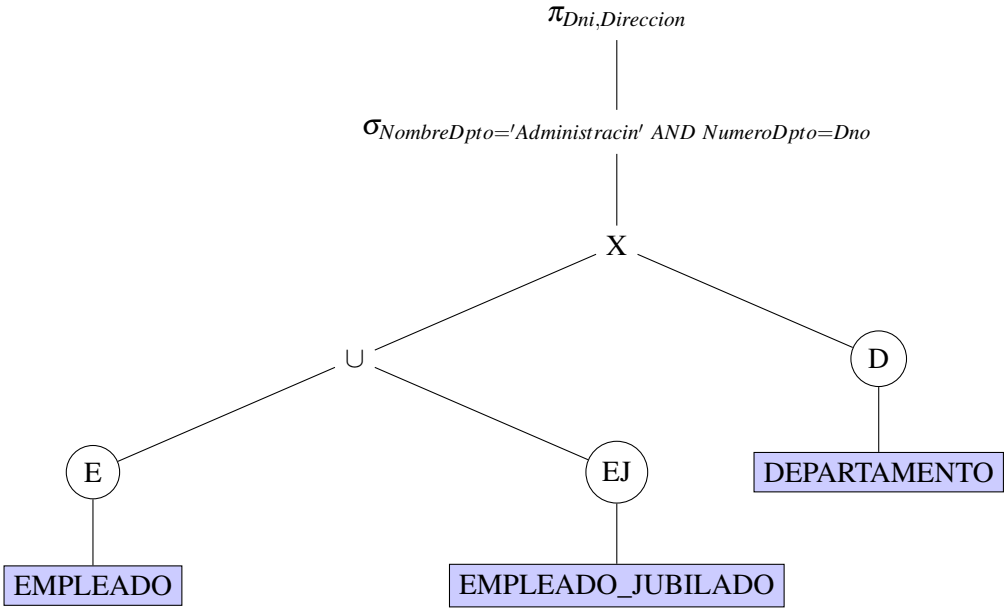
$$(\pi_{Dni,Direccion}((\sigma_{NombreDpto='Administracin'}(DEPARTAMENTO)) \bowtie_{NumeroDpto=Dno} (EMPLEADO)) \cup$$
$$(\pi_{Dni,Direccion}((\sigma_{NombreDpto='Administracin'}(DEPARTAMENTO)) \bowtie_{NumeroDpto=Dno} (EMPLEADO\_JUBILADO)))$$

Árbol de consultas inicial



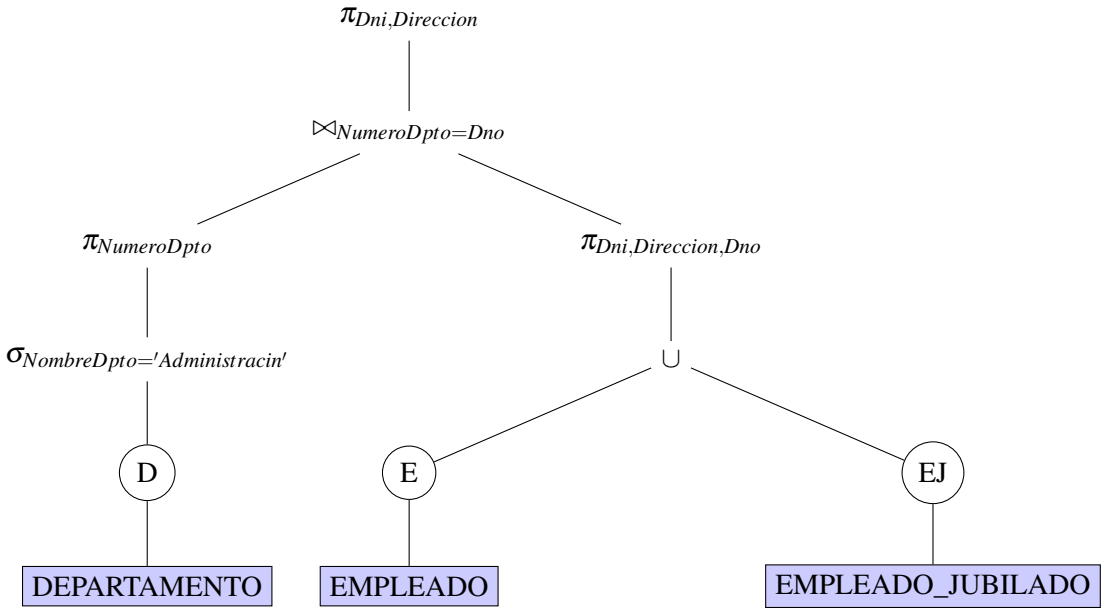
El primer paso para optimizar la consulta es utilizar las reglas 10 y 11 para trasladar la operación UNION a la parte baja del árbol.

Árbol de consultas 2



A continuación ya podemos aplicar los mismos pasos que en el ejemplo anterior para optimizar nuestra consulta, utilizando las reglas 1, 2, 3, 4, 5, 6, 7 y 12.

Árbol de consultas final



Y éste sería el árbol de consultas optimizado listo para ejecutar de la forma más eficaz posible.

Anexos

Anexo A

Algoritmos de implementación

A.1. Algoritmo de implementación para JOIN

Descripción del algoritmo para implementar una operación JOIN $T \leftarrow R \bowtie_{A=B} S$:

clasificación de las tuplas de R sobre el atributo A; (*suponiendo que R tiene n tuplas*)
clasificación de las tuplas de S sobre el atributo B; (*suponiendo que S tiene m tuplas*)
set $i \leftarrow 1, j \leftarrow 1$;
while (i $\leq$ n) and (j $\leq$ m)
do {
    if $R(i)[A] > S(j)[B]$
        then set $j \leftarrow j+1$;
    elseif $R(i)[A] < S(j)[B]$
        then set $i \leftarrow i+1$;
    else {
        generar la tupla combinada $< R(i), S(j) >$ en T;
        set $l \leftarrow j+1$;
        while (l $\leq$ m) and $(R(i)[A] = S(l)[B])$
        do {
            generar la tupla combinada$< R(i), S(l) >$ en T;
            set $l \leftarrow l+1$;
        }
        set $k \leftarrow i+1$;
        while (k $\leq$ n) and $(R(k)[A] = S(j)[B])$
        do {
            generar la tupla combinada$< R(k), S(j) >$ en T;
            set $k \leftarrow k+1$;
        }
        set $i \leftarrow k, j \leftarrow l$;
    }
}

A.2. Algoritmo de implementación para PROJECT

Descripción del algoritmo para implementar una operación PROJECT $T \leftarrow \pi_{<listadeatributos>}(R)$:
crear una tupla $t[< listadeatributos >]$ en T' para cada tupla t de R;
if $< listadeatributos >$ incluye una clave de R
then $T \leftarrow T'$;
else {
ordenar las tuplas de T';
set i $leftarrow 1, j \leftarrow 2$;
while $i \leq n$
do {
generar la tupla $T'[i]$ en T;
while $T'[i] = T'[j]$ and $j \leq n$ do $j \leftarrow j+1$; (*eliminación de duplicados*)
$i \leftarrow j, j \leftarrow i+1$;
}
}

A.3. Algoritmo de implementación para operaciones de conjunto

Operación UNION

Descripción del algoritmo para implementar una operación UNION $T \leftarrow R \cup S$:
ordenar las tuplas de R y S utilizando los mismos atributos de ordenación únicos;
set $i \leftarrow 1$, $j \leftarrow 1$;
while $(i \leq n)$ and $(j \leq m)$
do {
if $R(i) > S(j)$
then {
generar $S(j)$ en T;
set $j \leftarrow j+1$;
}
elseif $R(i) < S(j)$
then {
generar $R(i)$ en T;
set $i \leftarrow i+1$;
}
else set $j \leftarrow j+1$; (*$R(i) = S(j)$, por lo que nos saltmos una de las tuplas dupicadas*)
}
if $(i \leq n)$ then agregar tuplas $R(i)$ a $R(n)$ en T;
if $(j \leq m)$ then agregar tuplas $S(j)$ a $S(m)$ en T;

Operación INTERSECTION

Descripción del algoritmo para implementar una operación INTERSECTION $T \leftarrow R \cap S$:
ordenar las tuplas de R y S utilizando los mismos atributos de ordenación únicos;
set $i \leftarrow 1$, $j \leftarrow 1$;
while $(i \leq n)$ and $(j \leq m)$
do {
    if $R(i) > S(j)$
        then set $j \leftarrow j + 1$;
    elseif $R(i) < S(j)$
        then set $i \leftarrow i + 1$;
    else {
        generar $R(j)$ en T;
        set $i \leftarrow i + 1, j \leftarrow j + 1$;
    }
}

Operación MINUS

Descripción del algoritmo para implementar una operación MINUS $T \leftarrow R \setminus S$:
ordenar las tuplas de R y S utilizando los mismos atributos de ordenación únicos;
set $i \leftarrow 1$, $j \leftarrow 1$;
while $(i \leq n)$ and $(j \leq m)$
do {
    if $R(i) > S(j)$
        then set $j \leftarrow j + 1$;
    elseif $R(i) < S(j)$ {
        generar $R(i)$ en T;
        set $i \leftarrow i + 1$;
    }
    else set $i \leftarrow i + 1, j \leftarrow j + 1$;
}
if $(i \leq n)$ then agregar tuplas $R(i)$ a $R(n)$ en T;

Bibliografía

[1] RAMEZ ELMASRI Y SHAMKANT B. NAVATHE, *Fundamentals of database systems*, Pearson Education, Inc., 4.ª ed., 2004.

[2] LAKATOS, IMRE, *La metodología de los programas de investigación científica*, Alianza Editorial, Madrid, 1983.

[3] POLYA, GEORGE, *How to solve it?*, Expanded Princeton Science Library Edition, with a new foreword by John H. Conway, 2004.

[4] RIVERO CORNELIO, ENRIQUE, *Bases de datos relacionales: Fundamentos y diseño lógico*, Ediciones Icai,2005.

[5] RIVERO CORNELIO, ENRIQUE, *Bases de datos relacionales: Diseño físico*, Ediciones Icai,2004.

[6] CELMA GIMÉNEZ, MATILDE, *Bases de datos relacionales*, Pearson Education, Inc.,2003.

[7] KROENKE, DAVID M., *Procesamiento de bases de datos*, Pearson Education, Inc., 8.ª ed.,2003.

[8] PIÑEIRO GÓMEZ, JOSE M., *Bases de datos relacionales y modelado de datos*, Paraninfo, S.A. Editorial, 2013.