| Título | |
|---|---|
| **ANÁLISIS Y TRADUCCIÓN DE UNA CONSULTA SQL** | |
| **Finalidad** | **Autor** |
| Explicar el proceso que realiza un SGBD al analizar una consulta SQL y transformarla a algebra relacional. Conocido como "Fase de análisis y traducción". | Eduardo Encalada<br>aeencalada@utpl.edu.ec<br>**Revisión**<br>Oct-2017 |

# Introducción

La primera fase del procesamiento de una consulta es el "análisis y traducción", cuyo objetivo es validar que la consulta SQL esté bien estructurada, simplificarla si es posible, y traducirla a algebra relacional.

El análisis y traducción de la consulta implica 4 pasos:

1. Análisis léxico y sintáctico de la consulta
2. Análisis semántico de la consulta
3. Simplificación de la consulta
4. Traducción de la consulta a algebra relacional

# Caso de estudio

Considere las siguientes tablas EMPLE (de empleados) y DEPART (de departamentos), usadas comúnmente para ejercitar la práctica de SQL:

```
desc EMPLE

Name        Null     Type
----------- -------- ------------
EMP_NO      NOT NULL NUMBER(8)     PK
APELLIDO    NOT NULL VARCHAR2(50)
OFICIO               VARCHAR2(30)
DIR                  NUMBER(8)
FECHA_ALT            DATE
SALARIO              NUMBER(9,2)
COMISION             NUMBER(9,2)
DEPT_NO              NUMBER(5)     FK
```

```
desc DEPART

Name     Null     Type
-------- -------- ------------
DEPT_NO  NOT NULL NUMBER(8)     PK
DNOMBRE  NOT NULL VARCHAR2(30)
LOC               VARCHAR2(30)
TIPO     NOT NULL CHAR(3)
```

**Tabla EMPLE**

| EMP_NO | APELLIDO | OFICIO | DIR | FECHA_ALT | SALARIO | COMISION | DEPT_NO |
|---|---|---|---|---|---|---|---|
| 7369 | SÁNCHEZ | EMPLEADO | 7902 | 17-DIC-10 | 1040.00 | (null) | 20 |
| 7499 | ARROYO | VENDEDOR | 7698 | 20-FEB-10 | 1500.00 | 390 | 30 |
| 7521 | SALA | VENDEDOR | 7698 | 22-FEB-11 | 1625.50 | 650 | 30 |
| 7566 | JIMÉNEZ | DIRECTOR | 7839 | 02-ABR-11 | 2900.00 | (null) | 20 |
| 7654 | MARTÍN | VENDEDOR | 7698 | 29-SEP-11 | 1600.00 | 1020 | 30 |
| 7698 | NEGRO | DIRECTOR | 7839 | 01-MAY-11 | 3005.00 | (null) | 30 |
| 7782 | CEREZO | DIRECTOR | 7839 | 09-JUN-11 | 2885.00 | (null) | 10 |
| 7788 | GIL | ANALISTA | 7566 | 09-NOV-11 | 3000.00 | (null) | 20 |
| 7839 | REY | PRESIDENTE | (null) | 17-NOV-11 | 4100.00 | (null) | 10 |
| 7844 | TOVAR | VENDEDOR | 7698 | 08-SEP-11 | 1350.00 | (null) | 30 |
| 7876 | ALONSO | EMPLEADO | 7788 | 23-SEP-11 | 1430.50 | (null) | 20 |
| 7900 | JIMENO | EMPLEADO | 7698 | 03-DIC-11 | 1335.00 | (null) | 30 |
| 7902 | FERNÁNDEZ | ANALISTA | 7566 | 03-DIC-11 | 3000.00 | (null) | 20 |
| 7934 | MUÑOZ | EMPLEADO | 7782 | 23-ENE-12 | 1690.00 | (null) | 10 |
| 8005 | GONZÁLEZ | VENDEDOR | 7698 | 17-MAY-13 | 1550.00 | 890 | (null) |
| 8010 | PÉREZ | (null) | 7782 | 06-OCT-12 | 1035.00 | (null) | (null) |

**Tabla DEPART**

| DEPT_NO | DNOMBRE | LOC | TIPO |
|---|---|---|---|
| 10 | CONTABILIDAD | SEVILLA | ADM |
| 20 | INVESTIGACIÓN | MADRID | ADM |
| 30 | VENTAS | BARCELONA | OPE |
| 40 | PRODUCCIÓN | BILBAO | OPE |

Suponga la siguiente consulta SQL que tiene como objetivo *listar los funcionarios con su fecha de ingreso y el departamento donde trabajan, de aquellos funcionarios de Madrid, que no ingresaron en 2012, y cuyo oficio es EMPLEADO*.

```
SELECT e.apellido, e.fecha_alt, d.dnombre
FROM emple e, depart d
WHERE e.dept_no = d.dept_no
  AND e.oficio = 'EMPLEADO'
  AND e.fecha_alt < '01-ENE-12'
  AND e.fecha_alt > '31-DIC-12'
  AND d.loc = 'MADRID';
```

¿Cuál es el proceso que sigue el SGBD al analizar y traducir esta consulta?

Lo vemos ahora

## Análisis léxico y sintáctico de la consulta

Al igual que en todo lenguaje de programación, al procesar una consulta SQL lo primero que hará el compilador del SGBD es identificar los componentes o tokens del comando SQL (análisis léxico) y luego validará el lenguaje y la estructura del enunciado SQL (análisis sintáctico).

El análisis sintáctico implica validar principalmente:

- Uso correcto del lenguaje (palabras reservadas, orden de las cláusulas SQL, operadores, alias, puntuación, funciones, etc.)
- Nombres de tablas y columnas (consultando la metadata de la base de datos)
- Compatibilidad de tipos de datos (sobre todo en los criterios de selección de la cláusula WHERE)
- Que no exista ambigüedad en nombres de columnas
- Definición correcta de agrupamientos (GROUP BY y HAVING)
- Etc.

Si al analizar sintácticamente la consulta se detectase un error en la estructura del enunciado SQL, el SGBD abortaría la ejecución de la misma y devolvería un ERROR al usuario.

En el nuestro caso de estudio, como se puede ver, la consulta no presenta errores sintácticos: las cláusulas están correctas y en su orden, los nombres de tablas y columnas son correctos, se aplican bien los operadores lógicos y relacionales, no hay incompatibilidad en los tipos de datos, la cualificación de columnas es correcta. Por lo tanto, esta consulta no debe generar errores sintácticos.

Ejemplos de errores sintácticos para el caso de estudio planteado son los siguientes:

| Comando SQL | Motivo de error de sintaxis |
|---|---|
| `SELECT *`<br>`FROM empleados;` | La tabla "empleados" no existe |

| | |
|---|---|
| `SELECT *`<br>`FROM emple`<br>`WHERE dir = 'Loja';` | Incompatibilidad de tipos de datos. El campo DIR es numérico |
| `SELECT dept_no, e.apellido, d.dnombre`<br>`FROM emple e, depart d`<br>`WHERE e.dept_no = d.dept_no;` | Ambigüedad en el nombre de columna dept_no de la cláusula SELECT. El motor no sabría si mostrar dept_no de EMPLE o dept_no de DEPART |
| `SELECT oficio, count(*)`<br>`FROM emple;` | Se debe agrupar por la columna *oficio*. Falta la cláusula GROUP BY. |
| `SELECT oficio, count(*)`<br>`FROM emple`<br>`WHERE count(*) > 2`<br>`GROUP BY oficio;` | Las condiciones de filtrado de grupos en base a funciones de agrupamiento no se especifican en la cláusula WHERE. Se debe usar HAVING |
| `SELECT oficio, count(*)`<br>`FROM emple`<br>`ORDER BY oficio`<br>`GROUP BY oficio`<br>`HAVING count(*) > 2;` | La cláusula ORDER BY siempre va al final |

Si la validación sintáctica de la consulta tuvo éxito, entonces será procesada por el motor, sin retornar errores.

Dado que la consulta planteada en el caso de estudio, no tiene errores de sintaxis, el SGBD prosigue con el siguiente paso.

## Análisis semántico de la consulta

La semántica se refiere al significado de las cosas. En este caso implicaría validar si el enunciado de la consulta SQL se corresponde con lo que el usuario espera obtener de la consulta.

Pero el SGBD no tiene esa capacidad de interpretar lo que el usuario espera de la consulta, por lo que la validación semántica que realiza el motor es ciertamente limitada.

Durante el análisis semántico el SGBD valida principalmente los criterios de selección que se especifican en las cláusulas WHERE y HAVING. Verificando, por ejemplo:

- Que no existan condiciones contradictorias (2 o más condiciones que se contraponen y que no pueden ser ciertas a la vez)
- Que no existan condiciones contenidas en otras
- Que no existan condiciones contrapuestas a las restricciones de integridad

Para el caso de la consulta planteada como caso de estudio, si la analizamos detenidamente nos daremos cuenta que en efecto, presenta una incidencia de tipo semántico y es la siguiente:

> El enunciado del problema nos dice que deseamos obtener los empleados **que no ingresaron en el 2012**, para la cual en las condiciones se establece lo siguiente:
>
> `e.fecha_alt < '01-ENE-12' AND e.fecha_alt > '31-DIC-12'`
>
> El motor por supuesto no sabe lo que el usuario espera obtener realmente, pero si identifica que en esas dos condiciones existe una contradicción: es imposible que una misma fecha sea menor

a 01-ENE-12 y a la vez mayor a 31-DIC-12. Y al ser contradictorias el resultado de la conjunción de esas dos condiciones siempre será **FALSO**.

Descubierta la contradicción nos damos cuenta ahora que en lugar de AND se debió usar OR (`e.fecha_alt < '01-ENE-12' OR e.fecha_alt > '31-DIC-12'`), pero como ya se dijo, el motor no está en capacidad de hacer ese tipo de interpretaciones.

Otras incidencias semánticas que el SGBD si será capaz de detectar en nuestro caso de estudio son por ejemplo las siguientes:

| Comando SQL | Incidencia semántica |
|---|---|
| `SELECT *`<br>`FROM emple`<br>`WHERE salario BETWEEN 2000 and 2999`<br>`OR salario = 2500;` | La segunda condición está contenida en la primera, por lo tanto, eliminaría la segunda condición. |
| `SELECT oficio, count(*)`<br>`FROM emple`<br>`GROUP BY oficio`<br>`HAVING count(*) = 0;` | Cada oficio registrado en la tabla EMPLE lo está porque tiene al menos un empleado asociado. Por lo tanto, en este caso no es posible que exista algún oficio asociado a cero empleados. **count(*) = 0** siempre será FALSO. |
| `SELECT *`<br>`FROM emple`<br>`WHERE apellido is NULL;` | El campo **apellido** es obligatorio (restricción NOT NULL). Por lo tanto, no es posible que exista algún empleado sin apellido. **`apellido is NULL`** siempre devolverá FALSO. |
| `SELECT emp_no, count(*)`<br>`FROM emple`<br>`GROUP BY emp_no`<br>`HAVING count(*) = 1;` | El campo **emp_no** al ser llave primaria no acepta duplicados, por lo tanto, por cada valor de **emp_no** habrá solo un registro asociado. En este caso **`count(*) = 1`** siempre devolverá TRUE. |
| `SELECT *`<br>`FROM depart`<br>`WHERE tipo = 'ADMIN';` | La columna **depart.tipo** está definida con el tipo **CHAR(3)**. Significa que sus valores serán de máximo 3 caracteres. Por lo tanto, es imposible que exista algún valor de 5 caracteres. En este caso **`tipo = 'ADMIN'`** siempre será FALSE |

Una vez analizada semánticamente la consulta, el SGBD procede a normalizarla y simplificarla.

## Simplificación de la consulta

La simplificación de la consulta conlleva normalizar y optimizar su enunciado SQL de manera que implique el menor trabajo para el motor al ejecutarla. Y al igual que el análisis semántico, la simplificación ocurre sobre todo a nivel de las condiciones de selección.

Con base en el enunciado original de la consulta y en las incidencias semánticas que hayan sido detectadas en el paso anterior, el motor tratará de simplificar las condiciones de selección a su mínima expresión.

Por ejemplo, en nuestro caso de estudio, recordemos el planteamiento original de la consulta:

```
SELECT e.apellido, e.fecha_alt, d.dnombre
FROM emple e, depart d
WHERE e.dept_no = d.dept_no
  AND e.oficio = 'EMPLEADO'
  AND e.fecha_alt < '01-ENE-12'
  AND e.fecha_alt > '31-DIC-12'
  AND d.loc = 'MADRID';
```

Sintácticamente sin errores, pero si con una incidencia semántica: **`e.fecha_alt < '01-ENE-12' AND e.fecha_alt > '31-DIC-12'`** son condiciones contradictorias, por lo tanto la conjunción de ambas se evaluará siempre como FALSE. Con lo cual la condición de selección se traduce a:

```
e.dept_no = d.dept_no AND e.oficio = 'EMPLEADO' AND FALSE AND d.loc = 'MADRID'
```

Y a partir de aquí el motor intentará reducir la expresión condicional al mínimo posible, **aplicando las leyes de la lógica proposicional** (ANEXO 1).

En nuestro caso aplicando asociatividad tendríamos:

```
e.dept_no = d.dept_no AND (e.oficio = 'EMPLEADO' AND (FALSE AND d.loc = 'MADRID'))
```

Y sabemos que en una conjunción (AND) si uno de los operandos es FALSO el resultado es falso, por lo tanto, la expresión se reduce a:

```
e.dept_no = d.dept_no AND (e.oficio = 'EMPLEADO' AND FALSE)
```

Y por la misma regla tendríamos

```
e.dept_no = d.dept_no AND FALSE
```

Lo que finalmente se reduce a

```
FALSE
```

Por lo tanto, la condición de selección de nuestra consulta será siempre FALSA, lo que significa que no devolverá ningún resultado (resultado vacío).

A la consulta simplificada se la podría ilustrar así:

```
SELECT e.apellido, e.fecha_alt, d.dnombre
FROM emple e, depart d
WHERE 1 = 0;
```

** En este caso usamos 1=0 para denotar una condición FALSA, ya que el lenguaje SQL de Oracle no incluye una constante FALSE.*

En este caso el SGBD no necesitará cargar y procesar los registros de las tablas, pues de antemano se sabe que ningún registro cumplirá la condición de selección (cero filas resultantes).

Veamos ahora como quedaría la simplificación de los ejemplos adicionales planteados en el paso anterior (análisis semántico), considerando las incidencias semánticas detectadas

| Consulta SQL | Consulta Simplificada | Explicación |
|---|---|---|
| `SELECT *`<br>`FROM emple`<br>`WHERE salario BETWEEN 2000 and 2999`<br>`OR salario = 2500;` | `SELECT *`<br>`FROM emple`<br>`WHERE salario BETWEEN 2000 and 2999;` | Al estar incluida la 2da condición en la primera, se la elimina y se continua con la ejecución de la consulta. |
| `SELECT oficio, count(*)`<br>`FROM emple`<br>`GROUP BY oficio`<br>`HAVING count(*) = 0;` | `SELECT oficio, count(*)`<br>`FROM emple`<br>`GROUP BY oficio`<br>`HAVING 1 = 0;` | Al ser siempre FALSE la condición de HAVING, no hará falta continuar ejecutando la consulta. Retorna resultado vacío. |
| `SELECT *`<br>`FROM emple`<br>`WHERE apellido is NULL;` | `SELECT *`<br>`FROM emple`<br>`WHERE 1 = 0;` | Al ser siempre FALSE la condición de WHERE, no hará falta continuar ejecutando la consulta. Retorna resultado vacío. |
| `SELECT emp_no, count(*)`<br>`FROM emple`<br>`GROUP BY emp_no`<br>`HAVING count(*) = 1;` | `SELECT emp_no, 1`<br>`FROM emple;` | Al ser siempre TRUE la condición del HAVING, equivale a eliminar esa cláusula, y al ser **emp_no** llave primaria, el conteo total (*) de filas asociadas a cada número de empleado siempre será 1, por tanto, no hará falta agrupar y contar. Continua la ejecución de la consulta |
| `SELECT *`<br>`FROM depart`<br>`WHERE tipo = 'ADMIN';` | `SELECT *`<br>`FROM depart`<br>`WHERE 1 = 0;` | Al ser siempre FALSE la condición de WHERE, no hará falta continuar ejecutando la consulta. Retorna resultado vacío. |

Otros ejemplos de simplificación de expresiones condicionales

| Expresión | Expresión simplificada | Observación |
|---|---|---|
| NOT (salario <> 1800) AND oficio = 'VENDEDOR' | salario = 1800 AND oficio = 'VENDEDOR' | Aplicando doble negación |
| NOT (salario <> 800 OR oficio = 'DIRECTOR') | salario = 800 AND oficio <> 'DIRECTOR' | Aplicando leyes de morgan y doble negación |

El simplificar la condición de selección en la consulta es muy importante de cara al rendimiento. Por ejemplo, en los dos ejemplos previos, al simplificar se reduce de 4 a 3 las operaciones que deberá evaluar el motor. Y en los casos anteriores como se ha visto incluso puede implicar que no haya necesidad de continuar con la ejecución de la consulta.

**RECUERDE**: cuando al simplificar la consulta las condiciones de selección (WHERE y/o HAVING) se reducen a FALSE, entonces el SGBD no necesita proseguir con la ejecución de la consulta y retornará al usuario un resultado vacío ( `no rows selected` en Oracle ). Evitando con ello cargar bloques desde disco a memoria y procesar datos innecesariamente.

***Otro ejemplo para simplificación de consulta SQL se explica en el siguiente enlace: https://rebrand.ly/utplpff9d.***

## Traducción de la consulta a algebra relacional

Si luego simplificar la consulta SQL, ésta no prevé un resultado vacío (condición de selección siempre FALSE), entonces el proceso de ejecución continúa.

Antes de pasar la consulta a la **fase de optimización** donde se arma el plan de ejecución, se requiere traducir la consulta al lenguaje formal de **algebra relacional**.

En la consulta SQL de nuestro caso de estudio, como vimos tenía una contradicción semántica que conlleva un resultado vacío. Por lo que tal como estaba planteada no llegaría a este punto.

Entonces, aprovecharemos aquí para corregir el enunciado de la consulta.

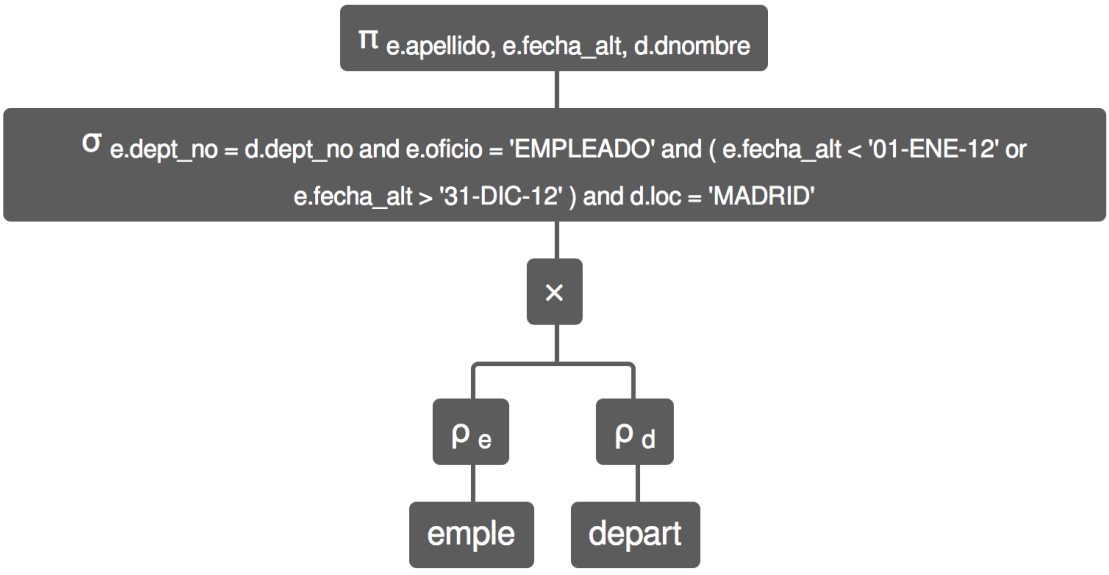
¿Cuál de debió ser el enunciado SQL para cumplir con el requerimiento de datos del problema planteado?, el siguiente:

```
SELECT e.apellido, e.fecha_alt, d.dnombre
FROM emple e, depart d
WHERE e.dept_no = d.dept_no
  AND e.oficio = 'EMPLEADO'
  AND (e.fecha_alt < '01-ENE-12'
   OR  e.fecha_alt > '31-DIC-12')
  AND d.loc = 'MADRID';
```

Que sería una consulta sin errores sintácticos ni semánticos.

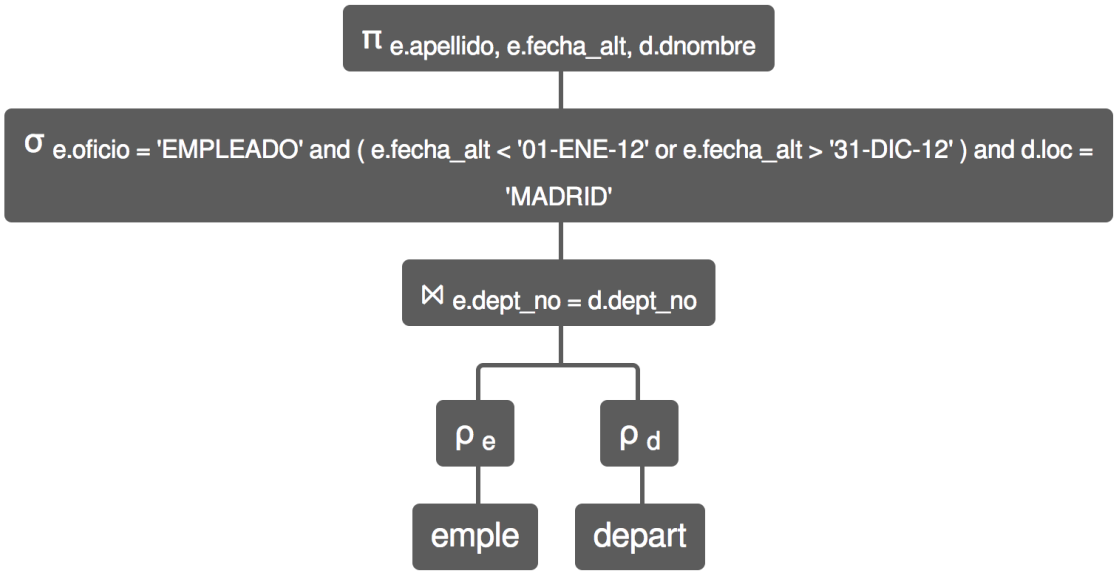
Dicha consulta en álgebra racional (vista como expresión y en forma de árbol) se expresaría así:

$$\pi_{\text{e.apellido, e.fecha\_alt, d.dnombre}} \sigma_{\text{e.dept\_no = d.dept\_no and e.oficio = 'EMPLEADO' and ( e.fecha\_alt < '01-ENE-12' or e.fecha\_alt > '31-DIC-12' ) and d.loc = 'MADRID'}} ((\rho_{e}\ \text{emple}) \times (\rho_{d}\ \text{depart}))$$



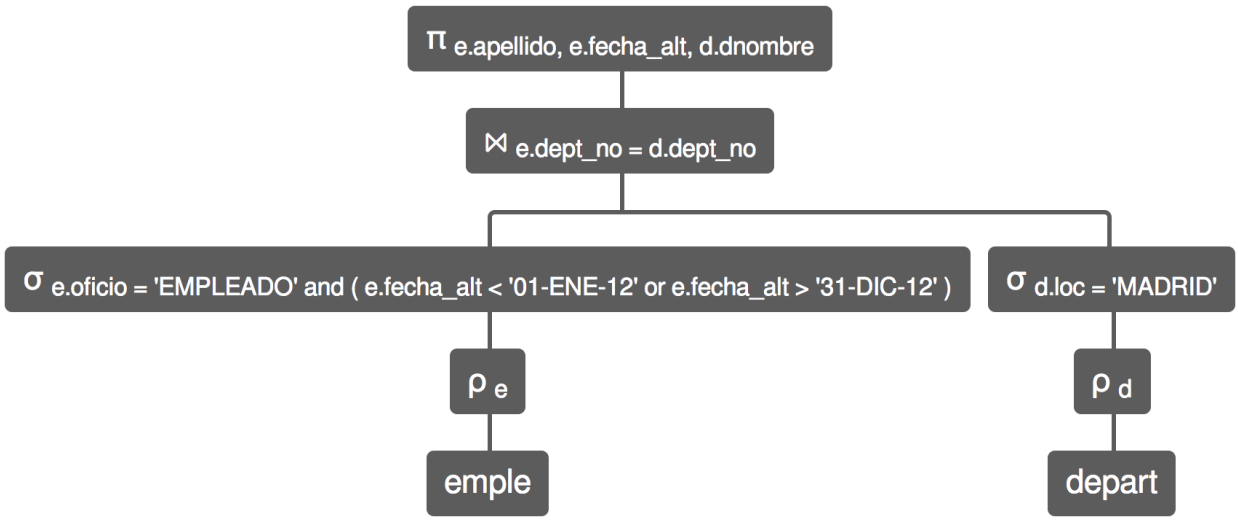
o así:

$\pi$ e.apellido, e.fecha_alt, d.dnombre $\sigma$ e.oficio = 'EMPLEADO' and ( e.fecha_alt < '01-ENE-12' or e.fecha_alt > '31-DIC-12' ) and d.loc = 'MADRID' (( $\rho$ e emple) $\bowtie$ e.dept_no = d.dept_no ( $\rho$ d depart))




o así:

$\pi$ e.apellido, e.fecha_alt, d.dnombre (( $\sigma$ e.oficio = 'EMPLEADO' and ( e.fecha_alt < '01-ENE-12' or e.fecha_alt > '31-DIC-12' ) ( $\rho$ e emple)) $\bowtie$ e.dept_no = d.dept_no ( $\sigma$ d.loc = 'MADRID' ( $\rho$ d depart)))




Pudiendo existir muchas más alternativas de representación de esa misma consulta. Será la **fase de optimización** donde se defina cuál de las posibles alternativas (planes de ejecución) es la más eficiente.

## ANEXO 1

# Lógica proposicional

**Operaciones**

- Negación ( NOT , ¬ , ~ )
- Conjunción ( AND , Y , ∧ )
- Disyunción ( OR , O , ∨ )

**Tablas de verdad**

**La conjunción**

| p | q | p ∧ q |
|---|---|---|
| V | V | V |
| V | F | F |
| F | V | F |
| F | F | F |

**La disyunción**

| p | q | p ∨ q |
|---|---|---|
| V | V | V |
| V | F | V |
| F | V | V |
| F | F | F |

**La negación**

| p | ~ p |
|---|---|
| V | F |
| F | V |

**Equivalencias**

| Leyes | Nombre |
|---|---|
| $\sim(\sim P) \equiv P$ | Doble negación |
| $P \vee Q \equiv Q \vee P$<br>$P \wedge Q \equiv Q \wedge P$ | Conmutativas |
| $(P \vee Q) \vee R \equiv P \vee (Q \vee R)$<br>$(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$ | Asociativas |
| $P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$<br>$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$ | Distributivas |
| $\sim(P \wedge Q) \equiv \sim P \vee \sim Q$<br>$\sim(P \vee Q) \equiv \sim P \wedge \sim Q$ | Leyes de De Morgan |

| Leyes | Nombre |
|---|---|
| $P \vee \sim P \equiv V$<br>$P \wedge \sim P \equiv F$ | Ley de medio excluido<br>Ley de contradicción |
| $P \vee F \equiv P$<br>$P \wedge V \equiv P$ | Leyes de identidad |
| $P \vee V \equiv V$<br>$P \wedge F \equiv F$ | Leyes de dominación |
| $P \vee P \equiv P$<br>$P \wedge P \equiv P$ | Leyes de idempotencia |