# 1. Módulo Base de Datos

El módulo Base de Datos permite crear una base de datos, agregarle información y realizar distintas consultas. Una base de datos consiste en un conjunto de tablas (cada una de las cuales representa un concepto), identificadas con un nombre único (equivalente a un TAD DICCIONARIO(NOMBRE_TABLA, TABLA). Una vez creada una base de datos, es posible agregarle tablas, indicando los nombres de sus campos y cuál es el campo clave. A cada tabla, a su vez, es posible agregarle registros, indicando el nombre de la tabla y el valor asociado a los campos de la tabla a la que se le está agregando (es decir, un conjunto de asociaciones campo-valor; ver TAD REGISTRO). También se pueden eliminar registros de una tabla, identificándolos por el valor del campo clave, y eliminar tablas completas. Por último, la base de datos dispone de ocho tipos de consultas, que dan como resultado un conjunto de registros.
En ningún caso se imponen restricciones a las entradas de las consultas, sino que se devuelve un conjunto de registro al que no se le aplicó la consulta (por ejemplo, si se quiere usar RENAME para renombrar un campo que no existe, la consulta no tendrá efecto) o un conjunto de registros vacío (por ejemplo, si se hace la consulta FROM con el nombre de una tabla que no existe en la base).

# Interfaz

**se explica con**: TAD BASE DE DATOS

**géneros**: `base_de_datos`.

## Operaciones básicas de Base de Datos

CREAR() $\rightarrow$ *res* : `base_de_datos`
**Pre** $\equiv$ {true}
**Post** $\equiv$ {*res* $=_{\text{obs}}$ crear()}
**Complejidad:** $\Theta(1)$
**Descripción:** crea una base de datos vacía.

AGREGARTABLA(**in** $n$ : `nombre_tabla`, **in** $c$ : `nombre_campo`, **in** $cs$ : `conj(nombre_campo)`, **in/out** $b$ : `base_de_datos`)
**Pre** $\equiv$ {$b = b_0 \wedge \neg$ def?($n$, tablas($b$)) $\wedge$ $c \in cs$}
**Post** $\equiv$ {$b =_{\text{obs}}$ agregarTabla($n$, $c$, $cs$, $b_0$)}
**Complejidad:** $\Theta(|t| + |c|)$
**Descripción:** crea una tabla (vacía) con el nombre $n$ y los campos incluidos en $cs$ en la base de datos $b$ e indica cuál es el campo clave de la tabla ($c$).

AGREGARREGISTRO(**in** $r$ : `registro`, **in** $n$ : `nombre_tabla`, **in/out** $b$ : `base_de_datos`)
**Pre** $\equiv$ {b = $b_0 \wedge$ def?($n$,tablas($b$)) $\wedge_{\text{L}}$ campos(obtener($n$, $b$)) $=_{\text{obs}}$ campos($r$) $\wedge$ ($\forall$ *reg*: registro)
(*reg* $\in$ registros(obtener($n$, $b$)) $\Rightarrow_{\text{L}}$ *reg*[clave(obtener($n$, $b$))] $\neq$ $r$[clave(obtener($n$, $b$))])}
**Post** $\equiv$ {*res* $=_{\text{obs}}$ agregarRegistro($r$, $n$, $b$)}
**Complejidad:** $\Theta(|t| + |v| + |c| + n + size(r))$
**Descripción:** agrega un registro $r$ a la tabla de nombre $n$ definida en la base de datos $b$.

TABLAS(**in** $b$: `base_de_datos`) $\rightarrow$ *res* : `dicc`$\langle$`nombre_tabla, tabla`$\rangle$
**Pre** $\equiv$ {true}
**Post** $\equiv$ {*res* $=_{\text{obs}}$ tablas($b$)}
**Complejidad:** $\Theta(1)$
**Descripción:** a partir de una base de datos $b$, devuelve un diccionario cuyas claves son los nombres de las tablas que la conforman y sus significados, las instancias tipo tabla en la base de datos que corresponden a ese nombre.
**Aliasing:** *res* se devuelve por referencia y no es modificable.

ELIMINARTABLA(**in** $n$ : `nombre_tabla`, **in/out** $b$ : `base_de_datos`)
**Pre** $\equiv$ {$b = b_0 \wedge$ def?($n$, tablas($b$))}
**Post** $\equiv$ {$b =_{\text{obs}}$ eliminarTabla($n$, $b_0$)}
**Complejidad:** $\Theta(|t| + size(t))$, donde $t$ es la tabla a eliminar.
**Descripción:** elimina la tabla de nombre $n$ de la base de datos $b$.
**Aliasing:** se libera en memoria el espacio ocupado por $t$.

ELIMINARREGISTRO(**in** $v$ : `valor`, **in** $n$ : `nombre_tabla`, **in/out** $b$ : `base_de_datos`)
**Pre** $\equiv$ {$b = b_0 \wedge$ def?($n$, tablas($b$)) $\wedge_{\text{L}}$ ($\exists$ $r$: registro)($r \in$ registros(obtener($n$, $b$)) $\wedge_{\text{L}}$ $r$[clave(obtener($n$, $b$)]= $v$)}
**Post** $\equiv$ {$b =_{\text{obs}}$ EliminarRegistro($v$, $n$, $b_0$)}

**Complejidad:** $\Theta(|t| + |v| + size(r))$, donde $r$ es el registro a eliminar.
**Descripción:** elimina el registro de la tabla de nombre $n$ que tenga el valor $v$ en el campo clave.
**Aliasing:** se libera en memoria el espacio ocupado por $r$.

REALIZARCONSULTA(**in** $q$ : `consulta`, **in** $b$ : `base_de_datos`) $\rightarrow$ $res$ : `conj(registros)`
**Pre** $\equiv$ {true}
**Post** $\equiv$ {$res =_{\text{obs}}$ realizarConsulta($t$, $b$)}
**Complejidad:** $\Theta(\sum_{sq \in q} \Theta(sq))$, donde $sq$ son las consultas anidadas que contiene $q$. $\Theta(sq)$ depende del tipo de consulta y de la cantidad de registros resultante de sus subconsultas.
**Descripción:** realiza una consulta $q$ sobre la base de datos $b$ y devuelve un conjunto de registros que contiene los resultados de la misma. A continuación se detallan los tipos de consultas disponibles y las operaciones que realiza cada uno:

- `FROM`: a partir del nombre $n$ de una tabla, devuelve el conjunto de registros de dicha tabla. Si la tabla no existe en la base de datos, devuelve un conjunto vacío.
- `SELECT`: a partir de los resultados de una consulta previa $q$, el nombre de un campo $c$ y un valor $v$, devuelve el conjunto de registros que tienen ese valor asociado al campo elegido. Si hay registros que no tienen este campo, no se incluyen en los resultados.
- `MATCH`: a partir de los resultados de una consulta previa $q$ y dos nombres de campo $c_1$ y $c_2$, devuelve un conjunto que contiene solo los registros que tienen el mismo valor $v$ en ambos campos. Si hay registros que no tienen alguno de esos dos, no se incluyen en el conjunto final.
- `PROJ`: a partir de los resultados de una consulta previa $q$ y un conjunto de nombres de campo $cs$, devuelve un conjunto con los mismos registros que $q$ pero dejando solo las columnas correspondientes a los campos de nombre perteneciente al conjunto $cs$.
- `RENAME`: toma los resultados de una consulta previa y cambia el nombre de uno de los campos por otro pasado como parámetro. Si hay registros que no tienen definido el campo c1, no se efectúa ningún renombre y se devuelve el registro tal como está. Si ya tienen definido un campo con el nombre c2, la operación de renombre queda sin efecto para ese registro.
- `INTER`: toma los resultados de dos consultas previas $q_1$, $q_2$ y devuelve el conjunto de registros que se encuentran tanto en el conjunto resultante de la primera consulta como en el de la segunda.
- `UNION`: toma los resultados de dos consultas previas $q_1$, $q_2$ y devuelve el conjunto de registros que se encuentran en el conjunto resultante de la primera consulta o en el de la segunda.
- `PRODUCT`: toma los resultados de dos consultas previas $q_1$, $q_2$ y devuelve un conjunto de registros que es el producto cartesiano de esos dos conjuntos, es decir, combina cada uno de los registros del primer conjunto con cada uno de los del segundo (formando nuevos registros con ambos campos). Si hay registros provenientes de la primera consulta $q_1$ que tienen campos en común con registros provenientes de la segunda, se conserva el valor del campo correspondiente a la segunda consulta.

**Aliasing:** $res$ se devuelve por copia.

# Representación

## Representación de la base de datos

Las bases de datos se representan con un diccionario sobre Trie de tablas con sus respectivos nombres. Los nombres son las claves del diccionario y las tablas sus significados. La estructura diccionario cumple la restricción que debe tener el invariante de representación de una base de datos, es decir, que cada tablas se identifica con un nombre único.

`base_de_datos` **se representa con** `bdd`

donde `bdd` es `tupla`($tablas$: `diccTrie(nombre_tabla, tabla)`)

Rep : bdd $\longrightarrow$ bool
Rep($e$) $\equiv$ true $\Longleftrightarrow$ true

Abs : bdd $e$ $\longrightarrow$ base_de_datos {Rep($e$)}
Abs($e$) $\equiv$ b: base_de_datos | tablas(b) $=_{\text{obs}}$ e.tablas

# Algoritmos

---

**iCrear**() $\rightarrow$ *res* : bdd

1: *res* $\leftarrow$ ⟨diccTrie::Vacío(), diccTrie::Vacío()⟩

Complejidad: $\Theta(1)$

---

---

**iAgregarTabla**(**in/out** $b$: `bdd`, **in** $n$: `nombre_tabla`), **in** $c$: `nombre_campo`, **in** $cs$: `conj(nombre_campo)`)

1: Definir(*b.tablas*, *n*, Tabla::Nueva(*cs*, *k*))

Complejidad: $\Theta(|t| + |c|)$
Justificación: Crear una tabla nueva tiene un costo proporcional a la longitud del nombre de campo más largo.

---

---

**iAgregarRegistro**(**in** $r$: `registro`, **in** $n$: *nombre_tabla*, **in/out** $b$: `bdd`)

1: Insertar(Significado(*b.tablas*, *n*), *r*)

Complejidad: $\Theta(|t| + |v| + |c| + n + size(r))$
Justificación: $n$ es la cantidad de registros de la tabla y $size(r)$ es el costo de borrar un registro (esto ocurre solo si el valor clave del registro que se quiere agregar ya existe en la tabla).

---

---

**iTablas**(**in** $b$: `bdd`)$\rightarrow$ *res* : $diccTrie(nombre\_tabla, tabla)$

1: *res* $\leftarrow$ *b.tablas*

Complejidad: $\Theta(1)$
Justificación: *res* se devuelve por referencia y no es modificable.

---

---

**iEliminarTabla**(**in** $n$: `nombre_tabla`, **in/out** $b$: `bdd`)

1: Borrar(*b.tablas*, *n*)

Complejidad: $\Theta(|t| + size(tabla))$
Justificación: la operación tiene un costo adicional $size(tabla)$ porque se libera la memoria ocupada por la tabla eliminada.

---

---

**iEliminarRegistro**(**in** $v$: `valor`, **in** $n$: `nombre_tabla`, **in/out** $b$: `bdd`)

1: Borrar(Significado(*b.tablas*, *n*), *v*)

Complejidad: $\Theta(|t| + |v|)$
Justificación: v es el valor del campo clave de la tabla, por lo que no es necesario recorrer los valores.

---

**iRealizarConsulta**(**in** $q$ : `consulta`, **in** $b$ : `bdd`) $\rightarrow$ $res$ : conj(registro)

```
1: if Tipo_consulta(q) = 'FROM' then
2:     res ← iFROMAux(Nombre_tabla(q), b)
3: else
4:     if Tipo_consulta(q) = 'SELECT' then
5:         res ← iSELECTAux(Subconsulta1(q), Campo1(q), Valor(q), b)
6:     else
7:         if Tipo_consulta(q) = 'MATCH' then
8:             res ← iMATCHAux(Subconsulta1(q), Campo1(q), Campo2(q), b)
9:         else
10:            if Tipo_consulta(q) = 'PROJ' then
11:                res ← iPROJAux(Subconsulta1(q), Conj_campos(q), b)
12:            else
13:                if Tipo_consulta(q) = 'RENAME' then
14:                    res ← iRENAMEAux(Subconsulta1(q), Campo1(q), Campo2(q), b)
15:                else
16:                    if Tipo_consulta(q) = 'INTER' then
17:                        res ← iINTERAux(Subconsulta1(q), Subconsulta2(q), b)
18:                    else
19:                        if Tipo_consulta(q) = 'UNION' then
20:                            res ← iUNIONAux(Subconsulta1(q), Subconsulta2(q), b)
21:                        else
22:                            if Tipo_consulta(q) = 'PRODUCT' then
23:                                res ← iPRODUCTAux(Subconsulta1(q), Subconsulta2(q), b)
24:                            end if
25:                        end if
26:                    end if
27:                end if
28:            end if
29:        end if
30:    end if
31: end if
```

Complejidad: $\Theta(\sum_{sq \in q} \Theta(sq))$, donde $sq$ son las consultas anidadas que contiene $q$.
Justificación: Cada una de estas consultas tendrá una complejidad de peor caso correspondiente al tipo de consulta del que se trate (ver complejidad de funciones auxiliares para más detalles) y proporcional a la cantidad de registros resultante de cada una de ellas.

**Funciones auxiliares de RealizarConsulta**

**iFROMAux**(**in** $n$ : `nombre_tabla`, **in** $b$ : `base_de_datos`) $\rightarrow$ $res$ : conj(registro)

```
1: if Definido?(b.Tablas, n) then
2:     res ← Registros(Significado(b.tablas, n))
3: else
4:     res ← Vacío()
5: end if
```

Complejidad: $\Theta(|t| + k \cdot (|c| + |v|))$
Justificación: $k$ es la cantidad de registros de la tabla y $|c| + |v|$ corresponde al costo de copiar cada registro.

**iSELECTAux**(**in** $q$ : `consulta`, **in** $c$ : `nombre_campo`, **in** $v$ : `valor`, **in** $b$ : `base_de_datos`) $\rightarrow$ $res$ : conj(registro)

```
1: res ← Vacío()
2: if Tipo_consulta(q) = 'FROM' ∧ Definido?(b.Tablas, Nombre_tabla(q)) then
3:     t ← Significado(b.tablas, Nombre_tabla(q))                    //Optimización 1: Select con clave
4:     if c = Clave(t) then
5:         AgregarRápido(res, RegPorClave(v, t))
6:     else                                                          //Optimización 2: Select sin clave
7:         if Pertenece?(Campos(t), c) then
8:             col ← ObtenerColumna(c, t)
9:             itCol ← CrearIt(col)
10:            while HaySiguiente(itCol) do
11:                if SiguienteSignificado(itCol) = v then
12:                    itR ← SiguienteClave(it)
13:                    AgregarRápido(res, *itR)
14:                end if
15:                Avanzar(itCol)
16:            end while
17:        end if
18:    end if
19: else
20:    if Tipo_consulta(q) = 'SELECT' ∧ Tipo_consulta(Subconsulta1(q)) = 'FROM' ∧ Definido?(b.Tablas,
   Nombre_tabla(Subconsulta1(q))) then
21:        t ← Significado(b.tablas, Nombre_tabla(Subconsulta1(q)))
22:        if c = Clave(t) then                                      //Optimización 4: Select con clave de select sin clave
23:            r ← RegPorClave(t, v)
24:            if Pertenece?(Campos(t), Campo(q)) ∧ r[Campo(q)] = Valor(q) then
25:                AgregarRápido(res, r)
26:            end if
27:        end if
28:    else
29:        if Tipo_consulta(q) = 'PRODUCT' ∧ Tipo_consulta(Subconsulta1(q)) = 'FROM' ∧ Tipo_consulta
   (Subconsulta2(q)) = 'FROM' then
30:            nt1 ← Nombre_tabla(Subconsulta1(q))                   //Optimización 5: Select de clave de un producto
31:            nt2 ← Nombre_tabla(Subconsulta2(q))
32:            if nt1 ≠ nt2 ∧ c = Clave(Significado(b.tablas, nt1)) then
33:                res ← SELECTPRODAux(nt1, nt2, c, v, Subconsulta1(Subconsulta1(q)), b)
34:            end if
35:        else                                                      //Caso general
36:            rs ← RealizarConsulta(q, b)
37:            it ← crearIt(rs)
38:            while HaySiguiente(it) do
39:                r ← Siguiente(it)
40:                if Pertenece?(Campos(r), c) ∧ r[c] = v then
41:                    AgregarRápido(res, r)
42:                end if
43:                Avanzar(it)
44:            end while
45:        end if
46:    end if
47: end if
```

Complejidad: **Select con clave**: $\Theta(|t| + |c| + |v|)$. **Select sin clave**: $\Theta(|t| + |c| + n \cdot |v| + k \cdot (|c| + |v|))$, donde $n$ es la cantidad de registros de la tabla que toma la subconsulta FROM y $k$ es la cantidad de registros. **Select con clave de select sin clave**: $\Theta(|t| + |c| + |v|)$. **Caso general**: $\Theta(\#rs \cdot (|c| + |v|) + k \cdot (|c| + |v|))$, donde $\#rs$ es la cantidad de registros resultantes de la subconsulta del SELECT.

Justificación: Como hay recursión mutua entre RealizarConsulta y las auxiliares correspondientes a cada tipo de consulta, a la complejidad del caso general hay que sumarle la de RealizarConsulta(q, b). En todos los casos, $(k \cdot (|c| + |v|)$ corresponde al costo de copiar los registros.

**iSELECTPRODAux**(**in** $t_1$ : `nombre_tabla`, **in** $t_2$ : `nombre_tabla`, **in** $c$ : `nombre_campo`, **in** $v$ : `valor`, **in** $q$ : `consulta`, **in** $b$ : `base_de_datos`) $\rightarrow$ $res$ : conj(registro)

```
1: r_1 ← SELECTAux(q, c, v, b)                    //Realiza un select con clave sobre la tabla 1
2: it ← CrearIt(Registros(Significado(b.tablas, t_2)))
3: while HaySiguiente(it) do
4:     r_2 ← Siguiente(it)
5:     itNuevo ← AgregarRápido(res, Vacío())
6:     rNuevo ← Siguiente(itNuevo)
7:     itCamp_1 ← CrearIt(Campos(r_1))
8:     while HaySiguiente(itCamp_1) do
9:         c ← Siguiente(itCamp_1)
10:        Definir(rNuevo, c, r_1[c])
11:        Avanzar(itCamp_1)
12:    end while
13:    itCamp_2 ← CrearIt(Campos(r_2))
14:    while HaySiguiente(itCamp_2) do
15:        c ← Siguiente(itCamp_2)
16:        Definir(rNuevo, c, r_2[c])
17:        Avanzar(itCamp_2)
18:    end while
19:    Avanzar(it)
20: end while
```

Complejidad: $\Theta(|t| + |c| + |v| + n_2 \cdot (|c| + |v|))$, donde $n_2$ es la cantidad de registros de la tabla 2.
Justificación: Como $c$ es la clave de la tabla 1, es posible realizar un select con clave que dé como resultado un solo registro. Luego, se puede realizar PRODUCT y combinando este único registro con los de la tabla 2. El costo de la copia está incluido en $n_2 \cdot (|c| + |v|)$ ya que la operación PRODUCT se realiza directamente sobre el registro nuevo.

---

**iMATCHAux**(**in** $q$: `consulta`, **in** $c_1$: `nombre_campo`, **in** $c_2$: `nombre_campo`, **in** $b$: `base_de_datos`) $\rightarrow$ $res$ : conj(registro)

```
1: if Tipo_consulta(q) = 'PRODUCT' ∧ Subconsulta_1(q) = 'FROM' ∧ Subconsulta_2(q) = 'FROM' then
2:     t_1 ← Nombre_tabla(Subconsulta_1(q))
3:     t_2 ← Nombre_tabla(Subconsulta_2(q))
4:     if t_1 ≠ t_2 ∧ Definido?(b.tablas, t_1) ∧ Definido?(b.tablas, t_2) ∧ c_1 = Clave(Significado(b.tablas, t_1)) ∧ c_2 =
   Clave(Significado(b.tablas, t_2)) then
5:         res ← iJOINAux(t_1, t_2, c_1, c_2, b)          //Optimización 3: Join con claves
6:     end if
7: else                                                   //Caso general
8:     res ← Vacío()
9:     rs ← RealizarConsulta(q, b)
10:    it ← crearIt(rs)
11:    while HaySiguiente(it) do
12:        r ← Siguiente(it)
13:        if Pertenece?(Campos(r), c_1) ∧ Pertenece?(Campos(r), c_2) ∧ r[c_1] = r[c_2] then
14:            AgregarRápido(res, r)
15:        end if
16:        Avanzar(it)
17:    end while
18: end if
```

Complejidad: **Join con claves:** $\Theta(|t| + |c| + \min(n_1, n_2) \cdot |v| + k \cdot (|c| + |v|))$, donde $n_1$ y $n_2$ son la cantidad de registros en las tablas 1 y 2, respectivamente. **Caso general:** $\Theta(\#rs \cdot (|c| + |v|) + k \cdot (|c| + |v|))$, donde $\#rs$ es la cantidad de registros resultantes de la subconsulta del MATCH.
Justificación: Como hay recursión mutua entre RealizarConsulta y las auxiliares, a la complejidad del caso general hay que sumarle la de RealizarConsulta(q, b). En ambos casos, $(k \cdot (|c| + |v|)$ corresponde al costo de copiar los registros.

**iJOINAux**(**in** $t_1$ : `nombre_tabla`, **in** $t_2$ : `nombre_tabla`, **in** $c_1$ : `nombre_campo`, **in** $c_2$ : `nombre_campo`, **in** $b$ : `base_de_datos`) $\rightarrow$ $res$ : conj(registro)

1: **if** Cardinal(Registros(Significado($b.tablas$, $t_1$)) $\leq$ Cardinal(Registros(Significado($b.tablas$, $t_2$)) **then**
2: $tabMen \leftarrow$ Significado($b.tablas$, $t_1$)
3: $tabMay \leftarrow$ Significado($b.tablas$, $t_2$)
4: **else**
5: $tabMen \leftarrow$ Significado($b.tablas$, $t_2$)
6: $tabMay \leftarrow$ Significado($b.tablas$, $t_1$)
7: **end if**
8: $it \leftarrow$ CrearIt(valoresClave($tabMen$))
9: **while** HaySiguiente($it$) **do**
10: $v \leftarrow$ Siguiente($it$)
11: **if** ExisteRegConClave($v$, $tabMay$) **then**
12: $itNuevo \leftarrow$ AgregarRápido($res$, Nuevo())
13: $rNuevo \leftarrow$ Siguiente($itNuevo$)
14: $r_1 \leftarrow$ RegPorClave($v$, $tabMen$)
15: $r_2 \leftarrow$ RegPorClave($v$, $tabMay$)
16: $itCamp1 \leftarrow$ CrearIt(Campos($r_1$))
17: **while** HaySiguiente($itCamp1$) **do**
18: $c \leftarrow$ Siguiente($itCamp1$)
19: Definir($rNuevo$, $c$, $r_1[c]$)
20: Avanzar($itCamp1$)
21: **end while**
22: $itCamp2 \leftarrow$ CrearIt(Campos($r_2$))
23: **while** HaySiguiente($itCamp2$) **do**
24: $c \leftarrow$ Siguiente($itCamp2$)
25: Definir($rNuevo$, $c$, $r_2[c]$)
26: Avanzar($itCamp2$)
27: **end while**
28: **end if**
29: Avanzar($it$)
30: **end while**

Complejidad: $\Theta(|t| + |c| + \text{mín}(n_1, n_2) \cdot |v| + k \cdot (|c| + |v|))$, donde $n_1$ y $n_2$ son la cantidad de registros en las tablas 1 y 2, respectivamente.

Justificación: Como los campos que toma MATCH son las claves de las tablas, solo será necesario hacer el producto cartesiano de los registros cuyas claves sean iguales. Si se chequea antes esta condición empezando por la tabla con menor cantidad de registros (iterando sobre el conjunto de sus valores clave), se logra la complejidad indicada arriba. $(k \cdot (|c| + |v|)$ corresponde al costo de copiar los registros.

**iPROJAux**(**in** $q$ : `consulta`, **in** $cs$ : `conj(nombre_campo`, **in** $b$ : `base_de_datos`) $\rightarrow$ $res$ : conj(registro)

```
1: res ← Vacío()
2: rs ← RealizarConsulta(q, b)
3: itReg ← CrearIt(rs)
4: while HaySiguiente(itReg) do
5:     r ← Siguiente(itReg)
6:     itCamp ← CrearIt(Campos(r))
7:     rNuevo ← Nuevo()
8:     while HaySiguiente(itCamp) do
9:         c ← Siguiente(itCamp)
10:        if Pertenece?(cs, c) then
11:            Definir(rNuevo, c, r[c])
12:        end if
13:        Avanzar(itCamp)
14:    end while
15:    if ¬EsVacío?(Campos(rNuevo)) then
16:        AgregarRápido(res, rNuevo)
17:    end if
18:    Avanzar(itReg)
19: end while
```

Complejidad: $\Theta(\#rs \cdot (|c| + |v|) + k \cdot (|c| + |v|))$, donde $\#rs$ es la cantidad de registros resultantes de la subconsulta del PROJ.
Justificación: Como hay recursión mutua entre RealizarConsulta y las auxiliares, a la complejidad del caso general hay que sumarle la de RealizarConsulta(q, b). $(k \cdot (|c| + |v|)$ corresponde al costo de copiar los registros.

---

**iINTERAux**(**in** $q_1$ : `consulta`, **in** $q_2$ : `consulta`, **in** $b$ : `base_de_datos`) $\rightarrow$ $res$ : conj(registro)

```
1: res ← Vacío()
2: rs1 ← RealizarConsulta(q1, b)
3: rs2 ← RealizarConsulta(q2, b)
4: it ← CrearIt(rs1)
5: while HaySiguiente(it) do
6:     r ← Siguiente(it)
7:     if Pertenece?(rs2, r) then
8:         Agregar(res, r)
9:     end if
10:    Avanzar(it)
11: end while
```

Complejidad: $\Theta(\#rs_1(|c| + |v|) \cdot \#rs_2(|c| + |v|))$, donde $\#rs_1$ es la cantidad de registros resultantes de la primera subconsulta de INTER y $\#rs_2$ es la cantidad de registros resultantes de la segunda.
Justificación: Como hay recursión mutua entre RealizarConsulta y las auxiliares, a la complejidad del caso general hay que sumarle la de RealizarConsulta($q_1$, b) y la de RealizarConsulta($q_2$, b).

**iUNIONAux**(**in** $q_1$ : `consulta`, **in** $q_2$ : `consulta`, **in** $b$ : `base_de_datos`) $\rightarrow$ $res$ : conj(registro)

```
1: res ← Vacío()
2: rs1 ← RealizarConsulta(q1, b)
3: rs2 ← RealizarConsulta(q2, b)
4: it1 ← CrearIt(rs1)
5: while HaySiguiente(it1) do
6:     AgregarRápido(res, Siguiente(it1))
7: end while
8: it2 ← CrearIt(rs2)
9: while HaySiguiente(it2) do
10:    AgregarRápido(res, Siguiente(it2))
11: end while
```

Complejidad: $\Theta(\#rs_1(|c| + |v|) + \#rs_2(|c| + |v|))$, donde $\#rs_1$ es la cantidad de registros resultantes de la primera subconsulta de UNION y $\#rs_2$ es la cantidad de registros resultantes de la segunda.
Justificación: Como hay recursión mutua entre RealizarConsulta y las auxiliares, a la complejidad del caso general hay que sumarle la de RealizarConsulta($q_1$, b) y la de RealizarConsulta($q_2$, b).

---

**iPRODUCTAux**(**in** $q_1$ : `consulta`, **in** $q_2$ : `consulta`, **in** $b$ : `base_de_datos`) $\rightarrow$ $res$ : conj(registro)

```
1: res ← Vacío()
2: rs1 ← RealizarConsulta(q1, b)
3: rs2 ← RealizarConsulta(q2, b)
4: it1 ← CrearIt(rs1)
5: while HaySiguiente(it1) do
6:     it2 ← CrearIt(rs2)
7:     while HaySiguiente(it2) do
8:         rNuevo ← Vacío()
9:         r1 ← Siguiente(it1)
10:        r2 ← Siguiente(it2)
11:        itCamp1 ← CrearIt(Campos(r1))
12:        while HaySiguiente(itCamp1) do
13:            c ← Siguiente(itCamp1)
14:            Definir(rNuevo, c, r1[c])
15:            Avanzar(itCamp1)
16:        end while
17:        itCamp2 ← CrearIt(Campos(r2))
18:        while HaySiguiente(itCamp2) do
19:            c ← Siguiente(itCamp2)
20:            Definir(rNuevo, c, r2[c])
21:            Avanzar(itCamp2)
22:        end while
23:        AgregarRápido(res, rNuevo)
24:        Avanzar(it2)
25:    end while
26: end while
```

Complejidad: $\Theta(\#rs_1(|c| + |v|) \cdot \#rs_2(|c| + |v|))$, donde $\#rs_1$ es la cantidad de registros resultantes de la primera subconsulta de PRODUCT y $\#rs_2$ es la cantidad de registros resultantes de la segunda.
Justificación: Como hay recursión mutua entre RealizarConsulta y las auxiliares, a la complejidad del caso general hay que sumarle la de RealizarConsulta($q_1$, b) y la de RealizarConsulta($q_2$, b).