

Universidad Nacional del Litoral  
*Facultad de Ingeniería y Ciencias Hídricas*  
Departamento de Informática

# **Bases de Datos**

*SQL: Guía de Trabajo Nro. 3*  
*Consultas avanzadas*

# 1. Consultas que involucran más de una relación

Cuando necesitamos una consulta que retorne datos de más de una tabla, debemos enlazar las mismas a través de las columnas que las asocian en el modelo físico (*Primary Keys* y *Foreign Keys*). A esto le denominamos *unión equidistante* (`INNER JOIN`).

## Ejemplo 1

Supongamos que deseamos obtener el título de las publicaciones junto al nombre de la editorial que las publicó. Estos datos residen en dos tablas diferentes, por lo tanto es necesario establecer un `INNER JOIN`.

Las columnas en común que establecen la asociación se especifican a través de la cláusula `ON`:

```
SELECT title, pub_name
FROM titles INNER JOIN publishers
      ON titles.pub_id = publishers.pub_id
WHERE titles.type = 'business'
```

Diagrama de la consulta SQL:

- (A)** La cláusula `WHERE titles.type = 'business'` está marcada con un círculo rojo.
- (B)** La cláusula `ON titles.pub_id = publishers.pub_id` está marcada con un círculo rojo.

Este query considera todos los pares de tuplas, uno de `titles` y otro de `publishers`. La condición que tiene que cumplir este par está dada por:

**(A)** La cláusula `WHERE`:

El componente `type` de la tupla de `titles` debe tener el valor `'business'`.

**(B)** La cláusula `INNER JOIN`:

El atributo `pub_id` de la tupla `titles` debe tener el mismo código de editorial que el atributo `pub_id` en la tupla `publishers`. Esto es, las dos tuplas deben hacer referencia a la misma editorial.

Si se encuentra un par de tuplas que satisfacen ambas condiciones, se producen los atributos `title` de la tupla de `titles` y `pub_name` de la tupla de `publishers` como parte de la respuesta.

La consulta del Ejemplo 1 también se puede escribir como:

```
SELECT title, pub_name
FROM titles, publishers
WHERE titles.pub_id = publishers.pub_id AND
      titles.type = 'business'
```

Esta es la sintaxis SQL previa al estándar SQL 99.

## 1.1. Eliminar la ambigüedad entre atributos

Si un query involucra varias relaciones, y entre estas relaciones hay dos o más atributos con el mismo nombre, necesitamos una manera de indicar cual de estos atributos que comparte su nombre es el que deseamos usar. SQL resuelve este problema permitiéndonos indicar un nombre de relación y un punto delante de un atributo.

### Ejemplo 2

Supongamos que queremos encontrar pares consistentes en las ciudades de un autor y una editorial ubicados en el mismo estado (columna `state`). La siguiente consulta lleva a cabo esto:

(A)                      (B)

```
SELECT authors.city, publishers.city

FROM authors INNER JOIN titleauthor
      ON authors.au_id = titleauthor.au_id
      INNER JOIN titles
      ON titleauthor.title_id = titles.title_id
      INNER JOIN publishers
      ON titles.pub_id = publishers.pub_id
WHERE authors.state = publishers.state
```

Eliminamos la ambigüedad agregando el nombre de la tabla seguida de un punto (A) y (B) en la lista de salida del `SELECT`.

Por supuesto podemos utilizar el nombre de la tabla o relación seguida por un punto aún en casos donde no existe ambigüedad. Al ejemplo 1 lo podríamos escribir como:

(A)                      (B)

```
SELECT titles.title, publishers.pub_name
FROM titles INNER JOIN publishers
      ON titles.pub_id = publishers.pub_id
WHERE titles.type = 'business'
```

## 1.2. Alias de tabla (tuple variables)

Supongamos que tenemos un query que involucre dos o más tuplas de la **misma tabla**.

Podemos listar una relación o tabla en la cláusula `FROM` cuantas veces la necesitemos, pero necesitamos una forma de referirnos a **cada ocurrencia de ella**. SQL nos permite definir, para cada ocurrencia de una tabla en la cláusula `FROM`, un “alias de tabla”, también denominado *tuple variable*<sup>1</sup>.

En las cláusulas `SELECT` y `WHERE` podemos eliminar la ambigüedad entre atributos de la o las tablas precediendo los mismos por el alias apropiado y un punto.

### Ejemplo 3

Supongamos que queremos obtener podemos querer saber acerca de pares de autores que viven en la misma ciudad:

```
SELECT Autor1.au_lname, Autor2.au_lname
FROM authors Autor1, authors Autor2
WHERE Autor1.city = Autor2.city AND
      Autor1.au_lname < Autor2.au_lname
```

En la cláusula `FROM` vemos la declaración de dos alias, `Autor1` y `Autor2` (**A**). Cada una es un alias para la tabla `authors`. En la cláusula `SELECT` las usamos para hacer referencia a los componentes `name` de las dos tuplas (**B**).

Estos alias también son utilizados en la cláusula `WHERE` (**C**).

La segunda condición en la cláusula `WHERE` (**D**) se incluye a fin de evitar que las tuple variables `Autor1` y `Autor2` hagan referencia a la misma tupla. (obtendríamos un autor que encontró una coincidencia consigo mismo).

---

<sup>1</sup> A lo largo de las Guías de Trabajo utilizaremos la denominación “alias de tabla” o “tuple variable” de manera intercambiable.

### Tuple variables y nombres de tablas

En realidad, las referencias a atributos en las cláusulas `SELECT` y `WHERE` **son siempre referencias a una tuple variable**.

Lo que sucede es que, si una relación aparece una única vez en la cláusula `FROM`, podemos usar el nombre de la tabla como su propia **tuple variable**.

Por ejemplo, veamos las siguientes consultas:

```
SELECT title_id
FROM titles
  (A)
```

```
SELECT title_id
FROM titles AS titles
  (B)
```

(A) puede verse como un "atajo" de (B).

Cuando un atributo pertenece sin ambigüedades a una tabla, el nombre de la tabla (tuple variable) puede ser omitido.

## 2. Subconsultas (subqueries)

### Subquery - Subconsulta

Recordemos que una subconsulta es una sentencia **SELECT** anidada que es necesaria para asistir a la evaluación de una consulta principal. Un query o consulta que es parte de otro se denomina **subquery o subconsulta**. En el procesamiento de la sentencia completa, se evalúa primero la subconsulta y luego se aplican los resultados a la consulta principal.

Los subqueries pueden tener a su vez subqueries, y así sucesivamente tantos niveles de anidamiento como necesitemos.

### Outer query o Query "principal"

Llamamos **outer query** a un query "principal", que hace uso de subqueries.

### Inner query

Llamamos **inner query** a un subquery de un query "principal".

## 2.1. Subqueries que producen valores escalares

Una expresión *select-from-where* puede producir una relación con **n** atributos y **n** tuplas.

A veces nos interesan los **valores** de un único atributo y otras veces podemos deducir, a partir de las claves (primary o unique) que obtendremos un **único valor para un atributo** (valor escalar).

### Relación unaria

Llamamos relación unaria a una relación que posee un único atributo o columna.

### Valor escalar

Llamamos valor escalar a un valor atómico correspondiente a un componente de una tupla.

La forma más sencilla de subconsulta consiste en una sentencia SQL que retorna un *único valor escalar* que es necesario para evaluar una condición en el *outer query*.

Si sabemos a priori que obtendremos un valor escalar podemos usar la expresión *select-from-where* encerrada entre paréntesis, como si se tratara de una constante. Esta expresión puede aparecer en una cláusula **WHERE** en cualquier lugar donde esperaríamos encontrar una constante o un atributo representando un componente de una tupla. Por ejemplo, podemos comparar el resultado de un subquery como éstos con una constante o un atributo:

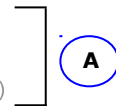
```
SELECT col1, col2
FROM tabla
WHERE col1 [=, <>, <, <=, >, >=] (inner query de valor único)
```

Lo que tenemos en la cláusula **WHERE** del *outer query* es un **predicado de comparación** entre una *columna o expresión de valor* y una *subconsulta de valor único*. Recordemos que los *tipos de datos* de la expresión de valor y la subconsulta de valor único deben ser comparables.

#### Ejemplo 4

Supongamos que deseamos obtener el nombre de la editorial que editó la publicación con código 'PC8888':

```
SELECT pub_name
FROM publishers
WHERE pub_id = (SELECT pub_id
                 FROM titles
                 WHERE title_id = 'PC8888')
```



**(A)** es el subquery. Centrándonos en este query por si mismo, vemos que el resultado será una relación unaria con el atributo `pub_id`, y que además esperamos encontrar una única tupla en la relación.

La tupla en nuestro caso es ('1389'). Esto es, un **único componente** `CHAR(4)` en este caso. Si se diera el caso de que **(A)** produjera más de una tupla o ninguna tupla, obtendríamos un error de ejecución.

Habiendo ejecutado este subquery, podemos ejecutar el query "principal" como si el valor '1389' reemplazara al subquery completo.



## 2.2. Condiciones que involucran relaciones

SQL posee predicados que se pueden aplicar a una relación  $R$  y producir un valor booleano. Esa relación  $R$  debe ser expresada como un subquery.

### 2.2.1 Condiciones que involucran relaciones usando el predicado IN

Vimos el predicado `IN` en la *Guía de Estudio Nro. 1*. en su forma más simple, evaluando si un valor escalar se encontraba en un conjunto de valores comparables, que expresábamos en una lista de valores encerrados entre paréntesis:

```
SELECT address
FROM authors
WHERE au_id IN ('172-32-1176', '238-95-7766')
```

Lo extenderemos ahora incluyendo subconsultas.

Consideremos un valor escalar  $s$ .

$s \text{ IN } R$  es true si y solo si  $s$  es igual a uno de los valores en  $R$ . De manera análoga,  $s \text{ NOT IN } R$  es true si y solo si  $s$  **no es igual** a ninguno de los valores en  $R$ .

Aquí estamos asumiendo que  $R$  es una **relación unaria**.

#### Ejemplo 5

```
SELECT address
FROM authors
WHERE au_id IN (SELECT au_id
                 FROM authors
                 WHERE au_id = '172-32-1176' OR
                    au_id = '238-95-7766')
```

El predicado `IN` puede ser negado con `NOT IN`.

### 2.2.2. Condiciones que involucran relaciones usando el cuantificador ALL

Consideremos un valor escalar  $s$  y una relación  $R$

$s > \text{ALL } R$  es true si y solo si  $s$  es mayor a todos los valores en la **relación unaria**  $R$ . El operador  $>$  puede ser reemplazado por cualquiera de los cinco operadores de comparación con significado análogo:  $s$  debe cumplir con la condición para toda tupla en  $R$ .  
 $s < > \text{ALL } R$  es lo mismo que  $s \text{ NOT IN } R$ .

#### Ejemplo 6

```
SELECT title, price
FROM titles
WHERE price >= ALL (SELECT price
                    FROM titles Titles2
                    WHERE price IS not NULL)
```

El cuantificador **ALL** puede ser negado con **NOT ALL**.

### 2.2.3. Condiciones que involucran relaciones usando el cuantificador ANY

Consideremos un valor escalar  $s$  y una relación  $R$

$s > \text{ANY } R$  es true si y solo si  $s$  es mayor que al menos un valor en la **relación unaria**  $R$ . El operador  $>$  puede ser reemplazado por cualquiera de los cinco operadores de comparación con significado análogo:  $s$  debe cumplir con la condición para al menos una tupla en  $R$ .  
 $s = \text{ANY } R$  es lo mismo que  $s \text{ IN } R$ .

#### Ejemplo 7

```
SELECT title, price
FROM titles
WHERE price > ANY (SELECT price
                  FROM titles Titles2
                  WHERE price IS NOT NULL AND
                    price > 20)
```

El inner query retorna los valores 22.95, 21,59 y 20.95.

El outer query retorna true para las tuplas que poseen un precio mayor que cualquiera de éstos.

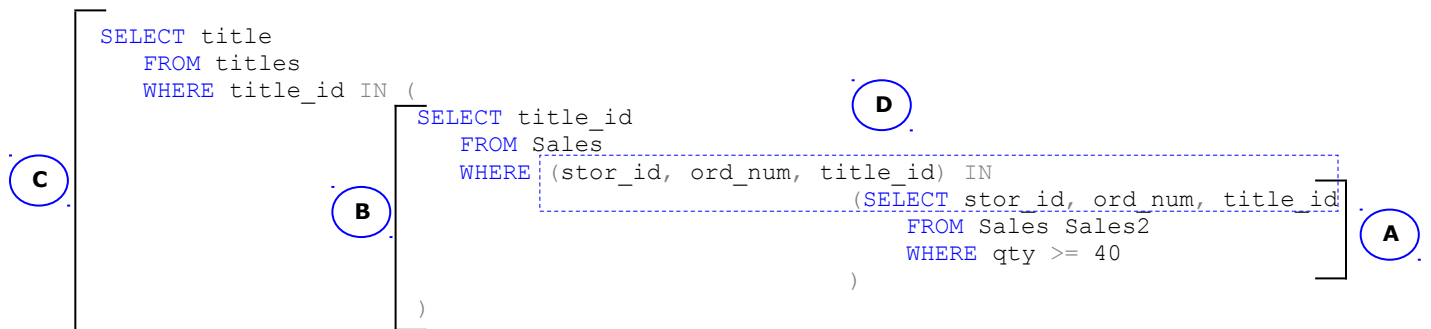
El cuantificador **ANY** puede ser negado con **NOT ANY**.

## 2.3. Condiciones que involucran tuplas

### Tuplas SQL

En SQL una tupla se representa por una lista de valores escalares encerrada entre paréntesis. Por ejemplo: ('PC8888', 'popular\_comp') o (title\_id, type). La primera de estas dos tuplas posee constantes como componentes.

### Ejemplo 8



El inner query (**A**) retorna las tuplas ('7066', 'A2976', 'PC888') y ('7067', 'P2121', 'TC3218')

El query intermedio (**B**) recupera el `title_id` de las tuplas que se encuentren en ese conjunto. Termina recuperando los valores 'PC888' y 'TC3218'. El query principal (**C**) retorna el título de estas publicaciones.

En este caso de uso del predicado `IN` (**D**) sucede entonces que `s ((stor_id, ord_num, title_id))` es una tupla y la relación  $R$  (la salida de la subconsulta (**A**)) posee más de un atributo en su schema.

Dijimos que la tupla puede contener constantes como componentes. Una consulta como la siguiente es totalmente válida:

```
SELECT title_id
FROM Sales
WHERE ('6380', '6871', 'BU1032') IN (
    SELECT stor_id, ord_num, title_id
    FROM Sales Sales2
    WHERE date_part('year', ord_date) = 1994
)
```



SQL Server no soporta consultas SQL que involucren tuplas.

## 2.4. Subqueries correlacionados

### Subqueries correlacionados

Los subqueries más simples pueden ser evaluados una única vez, y el resultado es usado en un query de más alto nivel. Un uso más complicado de subqueries anidados requiere que el subquery sea evaluado varias veces, una por cada "asignación" en el subquery de un valor que proviene de una tuple variable externa al mismo. Un subquery de este tipo se denomina **subquery correlacionado**.

### Ejemplo 9

En el siguiente ejemplo buscamos el empleado más antiguo de cada editorial:

```
SELECT pub_id, fname, lname, hire_date
FROM employee e
WHERE e.hire_date = (SELECT MIN(hire_date)
                     FROM employee e2
                     WHERE e2.pub_id = e.pub_id)
```

A

B

Por cada tupla, preguntamos en un subquery (**A**) cuál es la fecha más antigua de contratación para la editorial que estamos procesando en el outer query.

A medida que se "recorren" las tuplas de `employee` del outer query, cada tupla proporciona un valor para `e.pub_id` (**B**).

## 2.5. El predicado EXISTS

El predicado `EXISTS` siempre precede a un subquery, y este subquery normalmente es un *subquery correlacionado*.

Dada una relación  $R$ , `EXISTS R` es una condición que es true si y solo si  $R$  no es vacío.

### Ejemplo 10

En la siguiente consulta obtenemos las publicaciones que se vendieron en años diferentes a 1993 y 1994.

```
SELECT title, titles.title_id
FROM titles
WHERE EXISTS (SELECT *
               FROM sales
               WHERE sales.title_id = titles.Title_id AND
                     YEAR(ord_date) NOT IN (1993, 1994)
             )
```

## 2.6. Subqueries en cláusulas FROM<sup>2</sup>

Otro uso de los subqueries es como relaciones en una cláusula `FROM`.

En una lista `FROM`, podemos tener, en vez de una relación “almacenada”, un subquery encerrado entre paréntesis<sup>3</sup>. Como no tenemos un nombre para el resultado de este subquery, debemos darle un alias de *tuple variable*.

Luego podemos hacer referencia a las tuplas en el resultado del subquery como si se tratara de una tabla ordinaria en la lista de la cláusula `FROM`.

### Ejemplo 11

```
SELECT au_lname
FROM authors INNER JOIN titleauthor
ON authors.au_id = titleauthor.au_id
INNER JOIN (SELECT titles.*
            FROM titles
            WHERE pub_id = '0736'
          ) TitlesAlgodata
ON titleauthor.title_id = TitlesAlgodata.title_id
```

---

<sup>2</sup> Algunos DBMSs los denominan **tablas derivadas**.

<sup>3</sup> Este es un buen ejemplo de una relación (relation) que no está materializada ni como tabla ni como vista.

## 3. Grupos

Recordemos que la cláusula `GROUP BY` nos permite agrupar filas en base a los valores de una o más columnas. `GROUP BY` se ubica luego de la cláusula `WHERE`:

```
SELECT columnal
FROM tabla
GROUP BY columnal
```

### Atributos de grupo

Debemos tener en cuenta que, una vez definidos grupos en SQL, solo podemos especificar *atributos de grupo* en la *lista de salida* del `SELECT`.

Por ejemplo, si definimos grupos a partir de los tipos de publicación (columna `type`) en la tabla `titles`, *atributos de grupo* serán la cantidad de publicaciones de cada tipo, la suma de los precios de cada tipo de publicación, etc.

Estos atributos son totalmente incompatibles con los **atributos de tupla o de fila** que venimos viendo hasta ahora.

### 3.1. Condiciones de grupo

Recordemos que, de manera análoga a como `WHERE` nos permite especificar una condición que deben satisfacer las filas que formarán parte de la lista de salida, la cláusula `HAVING` nos permite especificar una condición que deben cumplir los grupos para formar parte de la lista de salida:

```
SELECT columnal
FROM tabla
GROUP BY columnal
HAVING condicion-de-grupo
```

Debemos tener en cuenta que, la condición de una cláusula `HAVING` sólo pueden incluir comparaciones contra *atributos de grupo*.

## 4. La expresión CASE

### 4.1. CASE simple

#### La expresión CASE simple

La expresión CASE simple compara un valor contra una lista de valores y retorna un resultado asociado al primer valor coincidente:

```
CASE expresion0
  WHEN expresion1 THEN ResultadoExpresion1
  [ [WHEN expresion2 THEN ResultadoExpresion2]
    [...]
  [ ELSE ResultadoExpresionN]
End
```

La expresion0 se compara contra expresion1, expresion2, etc. hasta que se encuentra una coincidencia o se encuentra el final de la lista de expresiones. Si CASE encuentra coincidencia, retorna el ResultadoExpresion correspondiente. Si no encuentra coincidencia alguna, retorna ResultadoExpresionN.

### 4.2. CASE searched

#### La expresión CASE searched

Esta forma de expresión CASE permite el análisis de varias condiciones lógicas y retorna un resultado asociado a la primera que evalúa a verdadero:

```
CASE
  WHEN condicion1 THEN expresion1
  [ [WHEN condicion2 THEN expresion2]
    [...]
  [ ELSE expresionN]
End
```

Si se encuentra una condición verdadera, el resultado es la expresión correspondiente. Si todas las condiciones son falsas, el resultado es expresionN.



## 5. Otras expresiones JOIN

Hasta ahora solo hemos visto la unión equidistante (`INNER JOIN`). Podemos también construir relaciones a través de algunas variaciones del operador `JOIN`.

### 5.1. Producto cartesiano “explícito”

Recordemos que el *Producto Cartesiano* de dos relaciones  $R$  y  $S$  es el resultado de “emparejar<sup>4</sup>” una tupla de  $R$  con una tupla de  $S$  es una tupla -más larga-, con un componente para cada una de los componentes de las tuplas que lo constituyen.

Por convención, los componentes de  $R$  (el operando de la izquierda) preceden a los componentes de  $S$  en el orden de atributos del resultado.

#### Ejemplo 12

Supongamos las relaciones  $R$  y  $S$  con los schemas y tuplas siguientes:

A	B
1	2
3	4

Relación  $R$

B	C	D
2	5	6
4	7	8
9	10	11

Relación  $S$

El producto cartesiano consiste de las siguientes seis tuplas:

A	R.B	S.B	C	D
1	2	2	5	6
1	2	4	7	8
1	2	9	10	11
3	4	2	5	6
3	4	4	7	8
3	4	9	10	11

---

<sup>4</sup> Como traducción de “to pair”.

Implementamos la solución en T-SQL de la siguiente manera, usando la sintaxis `CROSS JOIN` **(A)**:

```
CREATE TABLE R1
(
    A Integer,
    B Integer
)
```

```
CREATE TABLE S1
(
    B Integer,
    C Integer,
    D Integer
)
```

```
INSERT R1 VALUES (1, 2)
INSERT R1 VALUES (3, 4)
```

```
INSERT S1 VALUES (2, 5, 6)
INSERT S1 VALUES (4, 7, 8)
INSERT S1 VALUES (9, 10, 11)
```

```
SELECT A, R1.B 'R1.B', S1.B 'S1.B', C, D
FROM R1 CROSS JOIN S1
```

**A**

Obtenemos:

	A	R1.B	S1.B	C	D
1	1	2	2	5	6
2	1	2	4	7	8
3	1	2	9	10	11
4	3	4	2	5	6
5	3	4	4	7	8
6	3	4	9	10	11

## 5.2. Unión Natural (Natural join) “explícita”

Recordemos que el *Natural Join* de dos relaciones R y S era el resultado de “emparejar” una tupla de R con una tupla de S sólo cuando ambas tuplas coincidan en los valores de sus atributos comunes.

### Ejemplo 13

Supongamos las relaciones R y S con los schemas y tuplas siguientes:

A	B
1	2
3	4

Relación R

B	C	D
2	5	6
4	7	8
9	10	11

Relación S

El natural join consiste de las siguientes dos tuplas:

A	B	C	D
1	2	5	6
3	4	7	8



Implementamos un natural join “explícito” en PostgreSQL de la siguiente manera:

```
SELECT *  
FROM R NATURAL JOIN S
```

y obtenemos:

b	a	c	d
integer	integer	integer	integer
2	1	5	6
4	3	7	8



SQL Server no soporta Natural Joins “explícitos”.

## 5.3. Uniones externas (Outer Joins)

### Dangling tuple

Una tupla que en un join no encuentra coincidencia con ninguna tupla de la otra relación se dice que es una *dangling tuple*.

### Ejemplo 14

Supongamos que hacemos un JOIN entre las tablas `Publishers` y `Titles`:

```
SELECT publishers.pub_id, pub_name, titles.title_id
FROM publishers INNER JOIN titles
ON publishers.pub_id = titles.pub_id
```

Obtenemos diecinueve filas en el resultado. Las filas corresponden a las editoriales **que poseen títulos publicados**.

Sin embargo, puede darse el caso de tengamos editoriales que por alguna razón están cargadas en nuestra base de datos pero no poseen títulos cargados en la tabla `titles`. A estas tuplas de `publishers` las estamos perdiendo. Son **dangling tuples**.

Podemos recuperarlas escribiendo una **unión externa**, que materializamos escribiendo, en lugar de un `INNER JOIN`, un `LEFT JOIN`:

```
SELECT publishers.pub_id, pub_name, titles.title_id
FROM publishers LEFT JOIN titles
ON publishers.pub_id = titles.pub_id
```

Ahora obtenemos tuplas adicionales, correspondientes a las editoriales que no poseen publicaciones:

pub_id	pub_name	title_id
0877	Binnet & Hardley	TC3218
0877	Binnet & Hardley	TC4203
0877	Binnet & Hardley	TC7777
1389	Algodata Infosystems	BU1032
1389	Algodata Infosystems	BU1111
1389	Algodata Infosystems	BU7832
1389	Algodata Infosystems	MK1111
1389	Algodata Infosystems	PC1035
1389	Algodata Infosystems	PC8888
1389	Algodata Infosystems	PC9999
1622	Five Lakes Publishing	NULL
1756	Ramona Publishers	NULL
9901	GGG&G	NULL
9952	Scootney Books	NULL
9999	Luceme Publishing	NULL

El `LEFT JOIN` también puede escribirse como `LEFT OUTER JOIN`

El `RIGHT JOIN` sigue exactamente la misma idea. La sentencia podría escribirse también de esta forma, con idéntico resultado:

```
SELECT publishers.pub_id, pub_name, titles.title_id
FROM titles RIGHT JOIN publishers
ON titles.pub_id = publishers.pub_id
```

El `RIGHT JOIN` también puede escribirse como `RIGHT OUTER JOIN`

Finalmente podemos usar la sintaxis `FULL OUTER JOIN`, que considera las dangling tuplas de ambos "lados". En nuestro ejemplo obtenemos el mismo resultado:

```
SELECT publishers.pub_id, pub_name, titles.title_id
FROM publishers FULL OUTER JOIN titles
ON publishers.pub_id = titles.pub_id
```

## 6. El operador UNION

El operador `UNION` une los resultados (tuplas) de dos o más consultas en un único conjunto.

Dos reglas básicas para combinar los conjuntos de resultados de dos consultas con `UNION` son:

- a) El número y el orden de las columnas deben ser idénticos en todas las consultas.
- b) Los tipos de datos deben ser compatibles.

Por omisión, `UNION` elimina las filas duplicadas (obtenemos un conjunto).  
Si especificamos la cláusula `ALL`, se incluyen en la salida las filas duplicadas.

### Ejemplo 15

```
SELECT type, title
  FROM titles
 WHERE type = 'business'

UNION ALL

SELECT type, title
  FROM titles
 WHERE type = 'popular_comp'
```

## 7. SELECTs en la lista de salida de un SELECT

Es posible ubicar una sentencia `SELECT` secundaria como parte de la lista de salida de una sentencia `SELECT` principal.

Generalmente esta sentencia `SELECT` secundaria está correlacionada a la principal:

### Ejemplo 16

```
SELECT title, (Select SUM(qty)
               from sales
               where sales.title_id = titles.title_id) AS 'Cantidad vendida'
from titles
```



**Ejercicio 1.** Obtenga un listado de apellidos y nombres de autores junto a los títulos de las publicaciones de su autoría. Ordene el listado por apellido del autor.

au_lname	au_fname	Titulo
Bennet	Abraham	The Busy Executive's Database
Blotchet-Halls	Reginald	Fifty Years in Buckingham Pala
Carson	Cheryl	But Is It User Friendly?
DeFrance	Michel	The Gourmet Microwave Second E
del Castillo	Innes	Silicon Valley Gastronomic Tre
Dull	Ann	Secrets of Silicon Valley
Green	Marjorie	The Busy Executive's Database
Green	Marjorie	You Can Combat Computer Stress

### Orientación para la resolución

1. Consultas que involucran más de una relación

**Ejercicio 2.** Obtenga un listado que incluya los nombres de las editoriales (tabla `publishers`) y los nombres y apellidos de sus empleados (tabla `employee`) pero sólo para los empleados con `job level` de 200 o más.

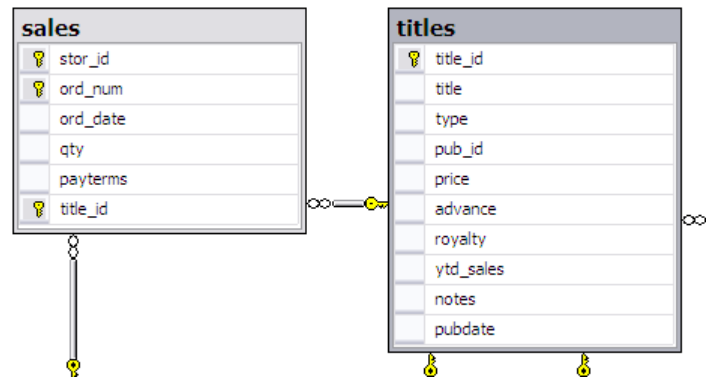
pub_name	Empleado	job_lvl
Scootney Books	Francisco Chang	227
Scootney Books	Philip Cramer	215
Luceme Publishing	Carlos Hemadez	211
New Moon Books	Matti Karttunen	220
Ramona Publishers	Maria Pontes	246

### Orientación para la resolución

1. Consultas que involucran más de una relación

**Ejercicio 3.** La tabla `sales` contiene información de ventas de publicaciones. Cada venta es identificada unívocamente por un número de orden (`ord_num`), el almacén donde se produjo (`stor_id`) y la publicación vendida (`title_id`).

La venta posee también la fecha de venta (`ord_date`) y la cantidad vendida de la publicación (`qty`).

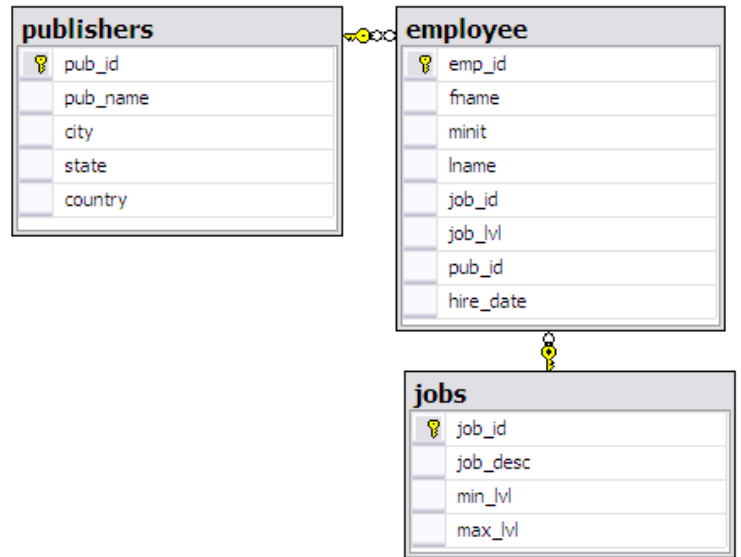


Se desea obtener un listado como el siguiente, que muestre los ingresos (precio de publicación \* cantidad vendida) que ha proporcionado cada autor a partir de las ventas de sus publicaciones. Ordene el listado por orden descendente de ingresos.

au_lname	au_fname	Ingresos
Ringer	Albert	1357.6000
Ringer	Anne	1302.2000
Dull	Ann	1000.0000
Hunter	Sheryl	1000.0000
Panteley	Sylvia	838.0000
MacFeather	Stearns	730.5500

**Ejercicio 4.** Obtenga los tipos de publicaciones (columna `type`) cuya media de precio sea mayor a \$12.

**Ejercicio 5.** La tabla `employee` posee información de los empleados de cada editorial. Por cada empleado tenemos su identificación (`emp_id`), su nombre (`fname`) y apellido (`lname`). Cada empleado pertenece a una editorial (`pub_id`) y posee una fecha de contratación (`hire_date`). Las funciones de los empleados se describen en la tabla `jobs`. Cada empleado posee una función (`job_id`).



Obtenga el apellido y nombre del empleado contratado más recientemente.

**Ejercicio 6.** Obtenga un listado de nombres de editoriales que han editado publicaciones de tipo `business`.

#### Orientación para la resolución

Esta consulta se puede resolver aplicando:

1. Consultas que involucran más de una relación
  - 2.2. Condiciones que involucran relaciones (IN-ALL-ANY)
- Resuélvala de ambas maneras.

**Ejercicio 7.** Obtenga un listado de títulos de las publicaciones que no se vendieron ni en 1993 ni en 1994. (Columna `ord_date` en tabla `Sales`)

### Orientación para la resolución

La tendencia natural sería intentar resolver esta consulta a través de un `INNER JOIN` entre las tablas de títulos (para obtener el título de la publicación) y la de ventas (a fin de evaluar la condición de fecha de la venta). La consulta sería la siguiente:

```
SELECT titles.title_id, title
FROM titles INNER JOIN sales
ON titles.title_id = sales.title_id
WHERE YEAR(sales.ord_date) NOT IN (1993, 1994)
```

Sin embargo, esta solución no retorna los datos esperados, ya que obtenemos el listado de las publicaciones que tuvieron ventas en años diferentes a 1993 y 1994, pero no obtenemos las publicaciones que no tuvieron ventas en absoluto. Ante requerimientos de este tipo, la consulta se debe resolver a través de un predicado `IN`.

**Ejercicio 8.** Obtenga un listado como el siguiente con las publicaciones que poseen un precio menor que el promedio de precio de la editorial a la que pertenecen.

title	pub_name	price
You Can Combat Computer Stress!	New Moon Books	2.9900
Life Without Fear	New Moon Books	7.0000
Emotional Security: A New Algorithm	New Moon Books	7.9900
The Gourmet Microwave	Binnet & Hardley	2.9900
Fifty Years in Buckingham Palace Kitchen	Binnet & Hardley	11.9500
Sushi, Anyone?	Binnet & Hardley	14.9900
Cooking with Computers: Surreptitious Ba	Algodata Infosystems	11.9500

**Ejercicio 9.** La tabla `authors` posee una columna llamada `contract` con valores 0 ó 1 indicando si el autor posee o no contrato con la editora. Se desea obtener un listado como el siguiente para los autores de California (columna `state` con valor CA).

Nombre	Apellido	Posee contrato?
Johnson	White	Si
Marjorie	Green	Si
Cheryl	Carson	Si
Michael	O'Leary	Si
Dean	Straight	Si
Abraham	Bennet	Si

**Ejercicio 10.** La columna `job_lvl` indica el puntaje del empleado dentro de su área de especialización. Se desea obtener un reporte como el siguiente, ordenado por puntaje y apellido del empleado:

lname	Nivel
Ashworth	Puntaje entre 100 y 200
Brown	Puntaje entre 100 y 200
Domingues	Puntaje entre 100 y 200
...	
Chang	Puntaje mayor que 200
Cramer	Puntaje mayor que 200
Devon	Puntaje mayor que 200
...	
Schmitt	Puntaje menor que 100
Smith	Puntaje menor que 100
Tonini	Puntaje menor que 100