

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

# **Bases de Datos**

*SQL: Guía de Trabajo Nro. 3*  
*Consultas avanzadas*  
*Parte 1: Joins*

# 1. Consultas que involucran más de una relación

## Expresiones JOIN

### 1.1. Producto cartesiano

El *Producto Cartesiano* de dos relaciones  $R$  y  $S$  es el resultado de “emparejar” 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

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

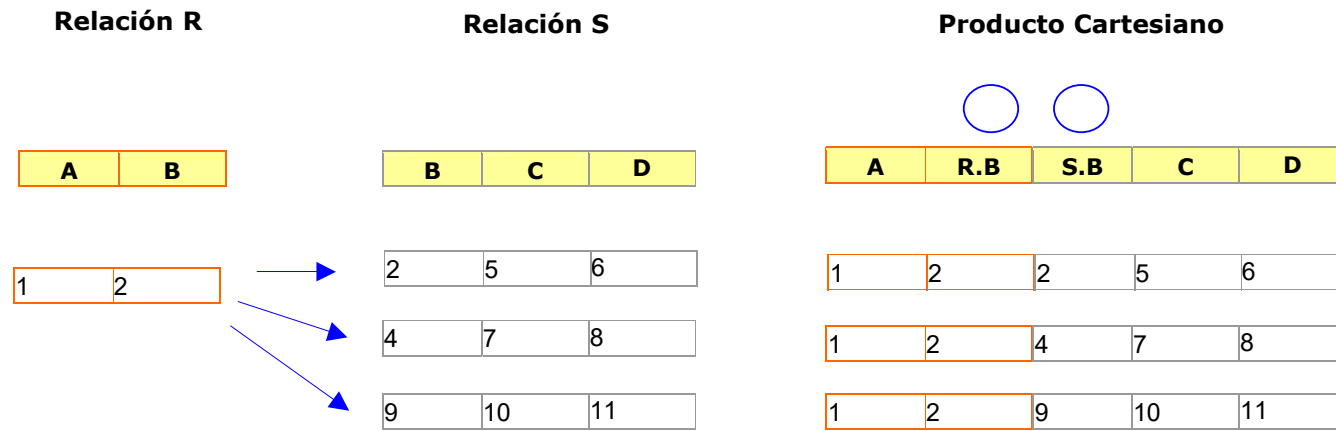
A	B
1	2
3	4

Relación  $R$

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

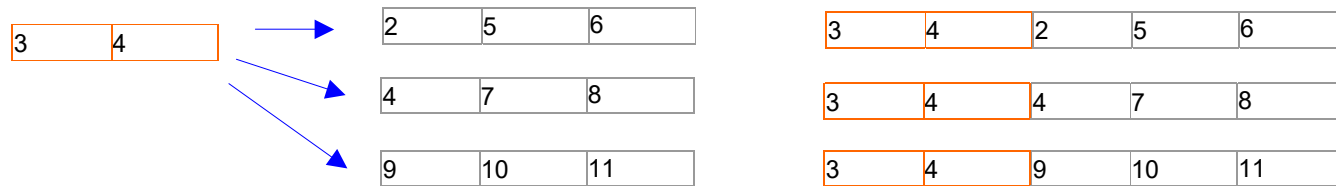
Relación  $S$

Lo que hacemos es combinar una tupla de R con cada una de las tuplas de S



Como vemos, cuando el nombre de un atributo se repite, lo prefijamos con el nombre de la relación (**A**) y (**B**).

De manera similar, para la segunda tupla de R:



Por lo tanto el producto cartesiano consiste de las siguientes seis tuplas:

<b>A</b>	<b>R.B</b>	<b>S.B</b>	<b>C</b>	<b>D</b>
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

Implementamos la solución en T-SQL de la siguiente manera:

```
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)

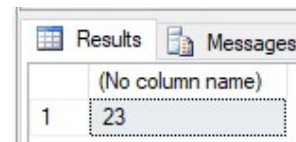
SELECT A, R1.B 'R1.B', S1.B 'S1.B', C, D
FROM R1 CROSS JOIN S1
```

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

Veamos la tabla authors:

```
Select COUNT(*) FROM authors
```

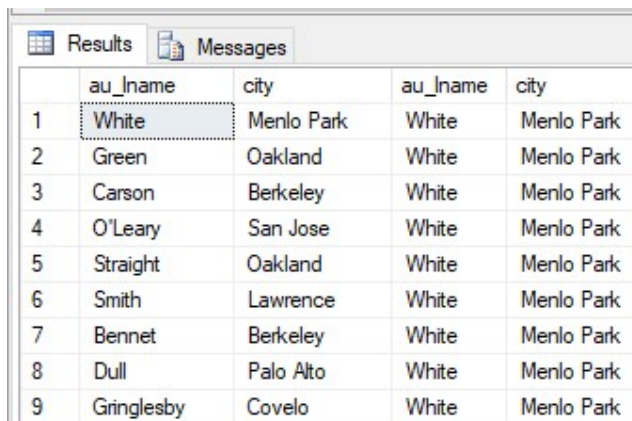


(No column name)	
1	23

Tenemos 23 autores

Podemos combinar cada autor con todos los demás usando la siguiente sentencia:

```
SELECT a1.au_lname, a1.city, a2.au_lname, a2.city  
FROM authors a1 CROSS JOIN authors a2
```



	au_lname	city	au_lname	city
1	White	Menlo Park	White	Menlo Park
2	Green	Oakland	White	Menlo Park
3	Carson	Berkeley	White	Menlo Park
4	O'Leary	San Jose	White	Menlo Park
5	Straight	Oakland	White	Menlo Park
6	Smith	Lawrence	White	Menlo Park
7	Bennet	Berkeley	White	Menlo Park
8	Dull	Palo Alto	White	Menlo Park
9	Gringlesby	Covelo	White	Menlo Park

..obtenemos 529 filas. (cada uno de los 23 autores combinados con 23 autores)

La sintaxis de `CROSS JOIN` tiene exactamente el mismo efecto que especificar las tablas en la cláusula `FROM` sin especificar ninguna cláusula de `JOIN` entre ellas:

```
SELECT a1.au_lname, a1.city, a2.au_lname, a2.city
FROM authors a1, authors a2
```

Podemos mejorar la consulta a fin de evitar que un autor se “empareje” con si mismo. Por ejemplo:

```
SELECT a1.au_lname, a1.city, a2.au_lname, a2.city
FROM authors a1 CROSS JOIN authors a2
WHERE a1.au_lname <> a2.au_lname
```

En el siguiente resultado parcial de este query vemos que el autor White se “emparejó” con todos los demás, pero no se emparejó con si mismo:

	au_lname	city	au_lname	city
1	Green	Oakland	White	Menlo Park
2	Carson	Berkeley	White	Menlo Park
3	O'Leary	San Jose	White	Menlo Park
4	Straight	Oakland	White	Menlo Park
5	Smith	Lawrence	White	Menlo Park
6	Bennet	Berkeley	White	Menlo Park
7	Dull	Palo Alto	White	Menlo Park
8	Gringlesby	Covelo	White	Menlo Park
9	Locksley	San Francisco	White	Menlo Park
10	Greene	Nashville	White	Menlo Park
11	Blotch-Halls	Corvallis	White	Menlo Park
12	Yokomoto	Walnut Creek	White	Menlo Park
13	del Castillo	Ann Arbor	White	Menlo Park
14	DeFrance	Gary	White	Menlo Park
15	Stringer	Oakland	White	Menlo Park
16	MacFeather	Oakland	White	Menlo Park
17	Karsen	Oakland	White	Menlo Park
18	Panteley	Rockville	White	Menlo Park
19	Hunter	Palo Alto	White	Menlo Park
20	McBadden	Vacaville	White	Menlo Park
21	Ringer	Salt Lake City	White	Menlo Park
22	Ringer	Salt Lake City	White	Menlo Park
23	White	Menlo Park	Green	Oakland
24	Carson	Berkeley	Green	Oakland

Esta consulta no parece ser muy útil, pero podemos adecuarla para -por ejemplo- listar todos los autores que viven en una misma ciudad:

```
SELECT a1.au_lname, a2.au_lname, a1.city, a2.city
FROM authors a1 CROSS JOIN authors a2
WHERE a1.au_lname <> a2.au_lname AND
a1.city = a2.city
```

Obtenemos:

	au_lname	au_lname	city	city	
1	Straight	Green	Oakland	Oakland	<b>A</b>
2	Stringer	Green	Oakland	Oakland	
3	MacFeather	Green	Oakland	Oakland	
4	Karsen	Green	Oakland	Oakland	
5	Bennet	Carson	Berkeley	Berkeley	
6	Green	Straight	Oakland	Oakland	<b>B</b>
7	Stringer	Straight	Oakland	Oakland	
8	MacFeather	Straight	Oakland	Oakland	
9	Karsen	Straight	Oakland	Oakland	
10	Carson	Bennet	Berkeley	Berkeley	
11	Hunter	Dull	Palo Alto	Palo Alto	
12	Green	Stringer	Oakland	Oakland	
13	Straight	Stringer	Oakland	Oakland	
14	MacFeather	Stringer	Oakland	Oakland	
15	Karsen	Stringer	Oakland	Oakland	
16	Green	MacFe...	Oakland	Oakland	
17	Straight	MacFe...	Oakland	Oakland	
18	Stringer	MacFe...	Oakland	Oakland	
19	Karsen	MacFe...	Oakland	Oakland	
20	Green	Karsen	Oakland	Oakland	
21	Straight	Karsen	Oakland	Oakland	
22	Stringer	Karsen	Oakland	Oakland	
23	MacFeather	Karsen	Oakland	Oakland	
24	Dull	Hunter	Palo Alto	Palo Alto	

El query funciona bien. El autor Straight vive en Oakland y el autor Green también (**A**)

Sin embargo, todavía tenemos el problema de que cada tupla aparece dos veces... Straight se "empareja" con Green (**A**), y está bien, pero también Green se "empareja" con Straight (**B**).



Podemos resolver esto modificando la consulta una vez más:

```
SELECT a1.au_lname, a2.au_lname, a1.city, a2.city
FROM authors a1 CROSS JOIN authors a2
WHERE a1.au_lname < a2.au_lname AND
      a1.city = a2.city
```

Results		Messages		
	au_lname	au_lname	city	city
1	Bennet	Carson	Berkeley	Berkeley
2	Green	Straight	Oakland	Oakland
3	MacFeather	Straight	Oakland	Oakland
4	Karsen	Straight	Oakland	Oakland
5	Green	Stringer	Oakland	Oakland
6	Straight	Stringer	Oakland	Oakland
7	MacFeather	Stringer	Oakland	Oakland
8	Karsen	Stringer	Oakland	Oakland
9	Green	MacFeather	Oakland	Oakland
10	Karsen	MacFeather	Oakland	Oakland
11	Green	Karsen	Oakland	Oakland
12	Dull	Hunter	Palo Alto	Palo Alto

Ahora tenemos que Green “se empareja” con Straight (**A**), a la vez que evitamos que Straight se empareje con Green.

## 1.2. Natural Join

El *Natural Join* de dos relaciones R y S es el resultado de “emparejar” una tupla de R con una tupla de S es una tupla más larga solo cuando ambas tuplas coincidan en los valores de sus atributos comunes.

### Ejemplo

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

Lo que hacemos es combinar una tupla de R con alguna de tupla de S que posea el mismo valor para el atributo B

**Relación R**

A	B
---	---

1	2
---	---

**Relación S**

B	C	D
---	---	---

2	5	6
---	---	---

4	7	8
---	---	---

9	10	11
---	----	----

**Natural Join**

A	B	C	D
---	---	---	---

1	2	5	6
---	---	---	---

*El atributo común no posee el mismo valor*

*El atributo común no posee el mismo valor*

De manera similar, para la segunda tupla de R:

3	4
---	---

2	5	6
---	---	---

*El atributo común no posee el mismo valor*

4	7	8
---	---	---

3	4	7	8
---	---	---	---

*El atributo común no posee el mismo valor*

9	10	11
---	----	----

El natural join consiste entonces de las siguientes dos tuplas:

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

PostgreSQL



Implementamos la solución en PostgreSQL de la siguiente manera:

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

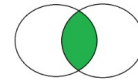
Obtenemos:

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



SQL Server no soporta Natural Joins.

### 1.3. Equi INNER Joins



Un equi JOIN retorna solo las tuplas que poseen valores iguales para las columnas especificadas.

**El operador de comparación siempre es la igualdad (=)**

En la práctica, este es el tipo de JOIN más usado, y vamos a estar enlazando las tablas generalmente a través de las columnas que las asocian en el modelo físico (*Primary Keys* y *Foreign Keys*).

#### Ejemplo

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	1

Relación S

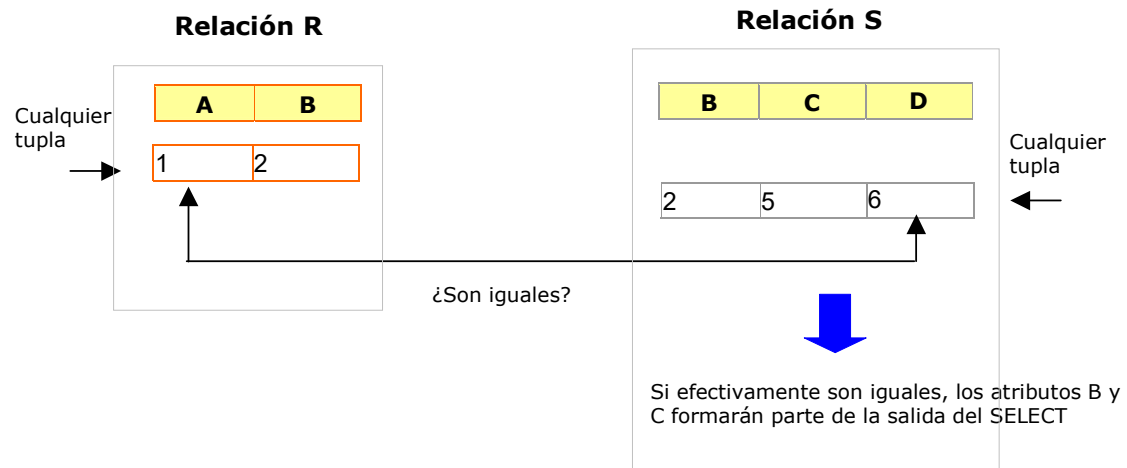
y queremos obtener todos los atributos de ambas relaciones para cuando se cumpla que el atributo A de la relación R “se empareje” con el atributo D de la relación S:

```
SELECT R.*, S.*
FROM R, S
WHERE R.A = S.D
```

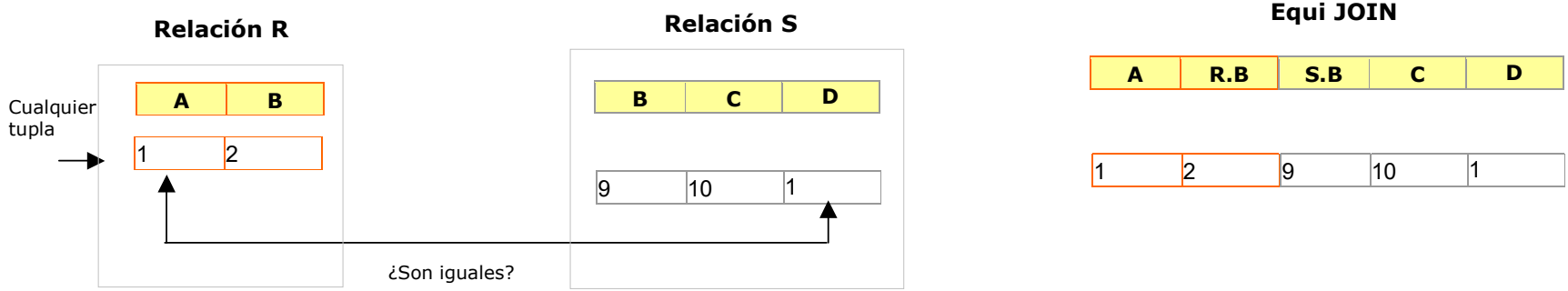
o, mejor aún, con la sintaxis de SQL-92:

```
SELECT R2.*, S2.*
FROM R2 INNER JOIN S2
ON R2.A = S2.D
```

Lo que hacemos es combinar una tupla de R con alguna de tupla de S que posea el mismo valor para el atributo especificado



En este ejemplo solo se produce coincidencia en este caso:



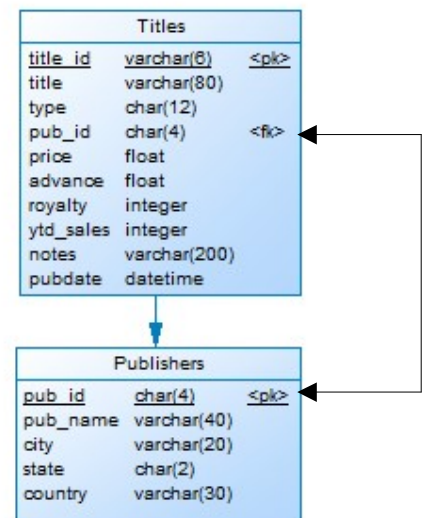
Supongamos que deseamos obtener el título de las publicaciones junto al nombre de la editorial que las publicó.

Como estos datos residen en dos tablas diferentes, es necesario establecer un join.



En este caso las tuplas a “emparejar” deben ser aquellas que hagan referencia a la misma editorial.

En otras palabras, cada tupla a comparar debe poseer el mismo valor para la columna pub\_id:





El siguiente es el Equi JOIN entre que lleva a cabo lo solicitado:

```
SELECT T.title, P.pub_name  
FROM Titles T INNER JOIN Publishers P  
ON T.pub_id = P.pub_id
```

**A**

Las columnas en común que establecen la asociación se especifican a través de la cláusula **ON (A)**.

El motor de base de datos considera todos los pares de tuplas, uno de `titles` y otro de `publishers`.

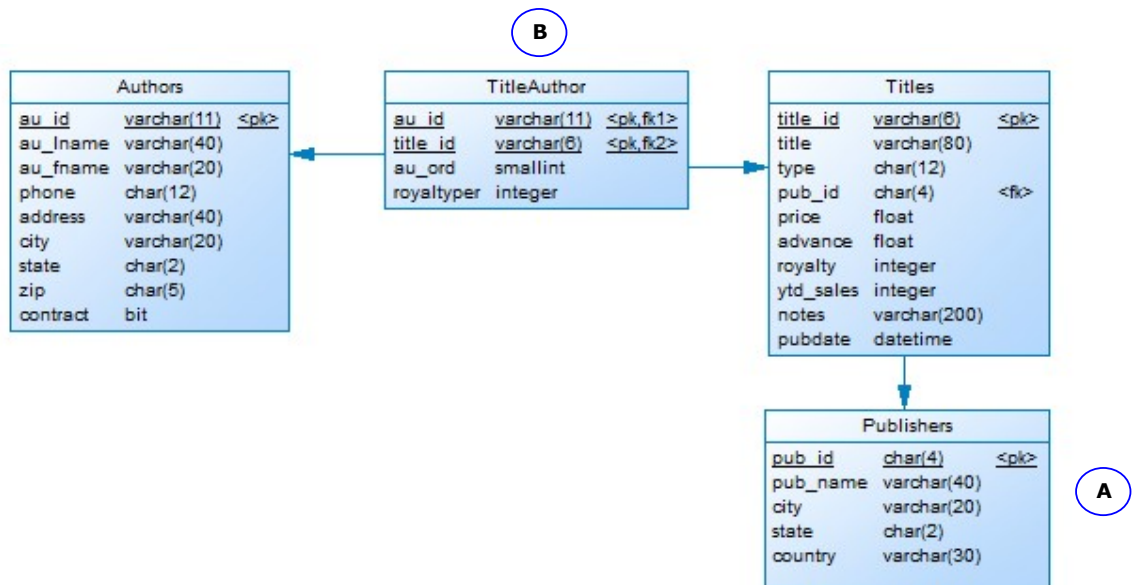
El atributo `pub_id` de la tupla `titles` debe tener el mismo código de editorial que el atributo `pub_id` en la tupla `publishers`. Esto es, las dos tuplas deben hacer referencia a la misma editorial.

Si se encuentra un par de tuplas que satisface esta condición, los atributos `title` de la tupla de `titles` y `pub_name` de la tupla de `publishers` se producen como parte de la salida del **SELECT**.

## 1.4. Enlazar más de dos tablas en un Equi INNER JOIN

Muchas veces necesitaremos “navegar” a través de nuestro modelo físico para extraer información de tablas vinculadas aunque no necesariamente de manera directa.

Por ejemplo, recordemos un fragmento del modelo físico de Pubs:

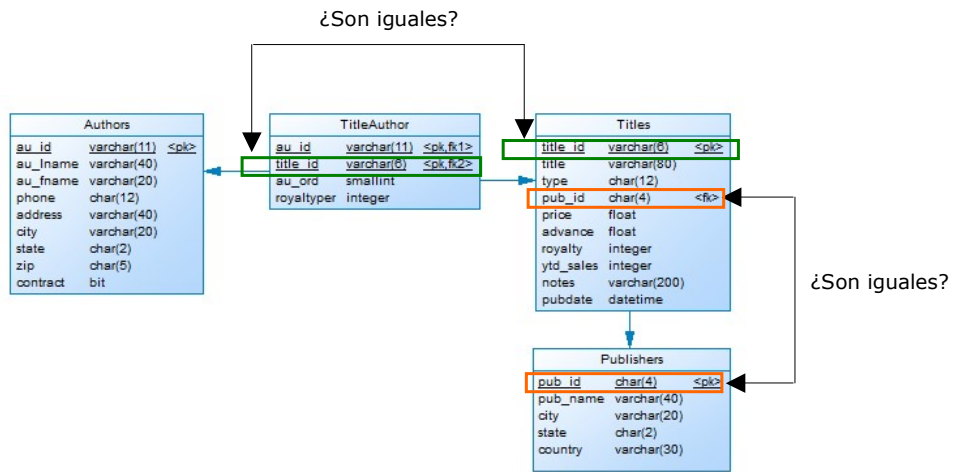


Supongamos que queremos obtener los nombres de la editoriales que han editado publicaciones del autor con código '998-72-3567'.

Los nombres de las editoriales residen en la tabla Publishers (**A**).

...pero los códigos de los autores recién los encontramos en la relación TitleAuthor (**B**)

Necesitaremos “navegar” a través de las columnas Primary Key y Foreign Key a fin de llegar a unir las tablas que resultan imprescindibles a fin de extraer la información que necesitamos.



Podríamos definir la siguiente consulta:

```

SELECT P.pub_name
FROM Publishers P INNER JOIN Titles T
ON P.pub_id = T.pub_id
INNER JOIN titleauthor TA
ON T.title_id = TA.title_id
WHERE TA.au_id = '998-72-3567'
  
```

Obtenemos:

Results		Messages	
	pub_name		
1	New Moon Books		
2	New Moon Books		

## 1.5. 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*.

### En nuestro ejemplo de JOIN

...suponíamos 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	1

Relación S

...y queríamos obtener:

```
SELECT R2.*, S2.*  
FROM R2 INNER JOIN S2  
ON R2.A = S2.D
```

Obteníamos coincidencia en:

1	2	9	10	1
---	---	---	----	---

Tenemos una tupla de R que nunca encontró coincidencia:

3	4
---	---

Esta tupla nunca será recuperada, ya que no encuentra coincidencia en la relación S. Lo mismo sucede por supuesto con dos tuplas de S.

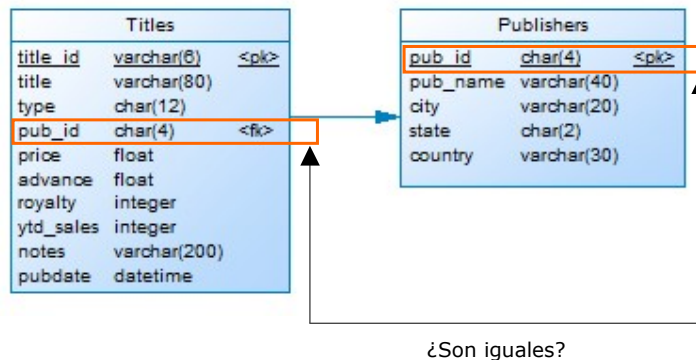
## Ejemplo

Supongamos que queremos obtener un listado de código de editorial, nombre de editorial junto a los códigos de publicaciones que han editado.

Tendríamos:

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

Estamos haciendo un JOIN entre las tablas Publishers y Titles:



Analicemos un poco los datos:

```
SELECT *
FROM publishers
```

	pub_id	pub_name	city	state	country
1	0736	New Moon Books	Boston	MA	USA
2	0877	Binnet & Hardley	Washington	DC	USA
3	1389	Algodata Infosystems	Berkeley	CA	USA
4	1622	Five Lakes Publishing	Chicago	IL	USA
5	1756	Ramona Publishers	Dallas	TX	USA
6	9901	GGG&G	Mnchen	NULL	Germany
7	9952	Scootney Books	New York	NY	USA
8	9988	Editora Ingenieria Web 2000	Texas	TA	USA
9	9999	Luceme Publishing	Paris	NULL	France

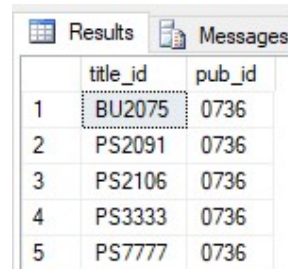
Tomemos, por ejemplo, la primer editorial listada, la editorial '0736'

Busquemos si hay publicaciones de ella en la tabla Titles:

```
SELECT title_id, pub_id
FROM titles
WHERE pub_id = '0736'
```

Obtenemos:

Hay publicaciones.

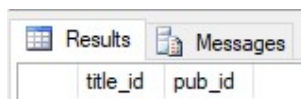


	title_id	pub_id
1	BU2075	0736
2	PS2091	0736
3	PS2106	0736
4	PS3333	0736
5	PS7777	0736

Tomemos, por ejemplo, la editorial '1622':

```
SELECT title_id, pub_id
FROM titles
WHERE pub_id = '1622'
```

Obtenemos:



title_id	pub_id
----------	--------

Esto significa que nunca recuperaremos a la editorial '1622' con el JOIN que escribimos. Simplemente porque no tiene publicaciones editadas y el JOIN no encontrará ninguna tupla en Titles para "emparejarse".

A estas tuplas de publishers las estamos perdiendo. Son **dangling tuples**.

Esto puede darse, por ejemplo, porque cargamos las Editoriales pero todavía no cargamos sus publicaciones.

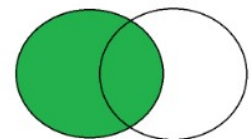
Básicamente, el query:

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

...retorna dieciocho filas en el resultado.  
Las filas corresponden a las editoriales **que poseen títulos publicados**:

Results		Messages	
	pub_id	pub_name	title_id
1	1389	Algodata Infosystems	BU1032
2	1389	Algodata Infosystems	BU1111
3	0736	New Moon Books	BU2075
4	1389	Algodata Infosystems	BU7832
5	0877	Binnet & Hardley	MC2222
6	0877	Binnet & Hardley	MC3021
7	0877	Binnet & Hardley	MC3026
8	1389	Algodata Infosystems	PC1035
9	1389	Algodata Infosystems	PC8888
10	1389	Algodata Infosystems	PC9999
11	0877	Binnet & Hardley	PS1372
12	0736	New Moon Books	PS2091
13	0736	New Moon Books	PS2106
14	0736	New Moon Books	PS3333
15	0736	New Moon Books	PS7777
16	0877	Binnet & Hardley	TC3218
17	0877	Binnet & Hardley	TC4203
18	0877	Binnet & Hardley	TC7777

Podemos recuperar esas editoriales “perdidas” 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:

Observamos que ahora sí, recuperamos a la editorial 1622 (**A**) aún cuando ésta no posea publicaciones editadas

Al valor title\_id se le ha asignado un NULL (**B**)

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
<b>A</b> 1622	Five Lakes Publishing	<b>B</b> NULL
1756	Ramona Publishers	NULL
9901	GGG&G	NULL
9952	Scotney Books	NULL
9999	Luceme Publishing	NULL

Las distinguimos porque poseen código de publicación con un valor NULL.

Volviendo al ejemplo de las relaciones R y S, tendríamos

```
SELECT R2.*, S2.*
FROM R2 LEFT JOIN S2
ON R2.A = S2.D
```

...y obtenemos:

	A	B	B	C	D
1	1	2	9	10	1
2	3	4	NULL	NULL	NULL

Como vemos, La tupla 

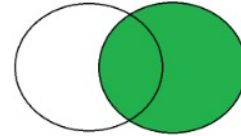
3	4
---	---

...ahora es recuperada.



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

El `RIGHT JOIN` sigue exactamente la misma idea.

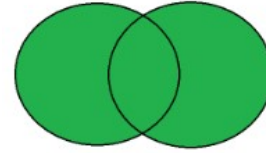


La sentencia podría escribirse también de esta forma, con idéntico resultado:

```
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`.



El `FULL OUTER JOIN` 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
```

Obtenemos:

	pub_id	pub_name	title_id
1	0736	New Moon Books	BU2075
2	0736	New Moon Books	PS2091
3	0736	New Moon Books	PS2106
4	0736	New Moon Books	PS3333
5	0736	New Moon Books	PS7777
6	0877	Binnet & Hardley	MC2222
7	0877	Binnet & Hardley	MC3021
8	0877	Binnet & Hardley	MC3026
9	0877	Binnet & Hardley	PS1372
10	0877	Binnet & Hardley	TC3218
11	0877	Binnet & Hardley	TC4203
12	0877	Binnet & Hardley	TC7777
13	1389	Algodata Infosy...	BU1032
14	1389	Algodata Infosy...	BU1111
15	1389	Algodata Infosy...	BU7832
16	1389	Algodata Infosy...	PC1035
17	1389	Algodata Infosy...	PC8888
18	1389	Algodata Infosy...	PC9999
19	1622	Five Lakes Publ...	NULL
20	1756	Ramona Publish...	NULL
21	9901	GGG&G	NULL
22	9952	Scootney Books	NULL
23	9999	Luceme Publish...	NULL

Aquí deberíamos obtener también publicaciones que hiciesen referencia a editoriales que no tuviesen ocurrencia en Publishers. Por supuesto, no obtenemos ninguna tupla de estas características

Es difícil que necesitemos recuperar dangling tuples de ambos lados.  
 Por ejemplo, podemos querer listar clientes aún cuando no tengan ocurrencias de facturas, pero difícilmente encontremos ocurrencias de facturas que no pertenezcan a algún cliente.

Volviendo al ejemplo de las relaciones R y S:

A	B
1	2
3	4

Relación R

B	C	D
2	5	6
4	7	8
9	10	1

Relación S

...tendríamos:

```
SELECT R2.*, S2.*
FROM R2 FULL OUTER JOIN S2
ON R2.A = S2.D
```

...y obtenemos:

	A	B	B	C	D
1	1	2	9	10	1
2	3	4	NULL	NULL	NULL
3	NULL	NULL	2	5	6
4	NULL	NULL	4	7	8

## 1.6. Tuple variables

A veces es necesario especificar 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" que denominamos *tuple variable*.

Luego, en las cláusulas `SELECT` y `WHERE` podemos eliminar la ambigüedad entre atributos de la o las tablas precediéndolos por la **tuple variable** apropiada y un punto.

### Ejemplo

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 *tuple variables*, `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).