

## Fastbook 07

# Tratamiento de Datos (Excel y SQL)

PostgreSQL: recorremos el camino



## 07. PostgreSQL: recorremos el camino

En este fastbook seguiremos explorando el lenguaje DML, concretamente exploraremos la operación JOIN y todas sus distintas posibilidades. Esta es una de las herramientas más potentes y utilizadas que tendremos a nuestra disposición a la hora de **realizar consultas**.

Por último, trabajaremos con los operadores de conjunto UNION, INTERSECT y EXCEPT.

*Autores: Carlos Manchón y Breogán Cid*

[Trabajar con varias tablas](#)

[Sentencia JOIN](#)

[Tipos de JOIN](#)

[Manejo sobre conjuntos: UNION](#)

[Manejo sobre conjuntos: INTERSECT](#)

[Manejo sobre conjuntos: EXCEPT](#)

[Ejercicios propuestos y sus soluciones](#)

[Conclusiones](#)

# Trabajar con varias tablas

X Edix Educación

---

Si recuerdas la estructura de la sentencia SELECT, en el apartado FROM se podían especificar varias tablas:

- 1 `SELECT <columna1> [,<columna2>, ..., <columna N>]`
- 2 `FROM <tabla1>[,<tabla2>], ..., [<tabla N>]`
- 3 `[WHERE <condicion >]`
- 4 `[GROUP BY <columna1>, <columna2>, ..., <columnaN>]`
- 5 `[HAVING <condicion>]`
- 6 `[ORDER BY <columna1>, <columna2>, ..., <columnaN>]`
- 7 `[LIMIT N]`

Imaginemos las tablas 1 y 2 con sus respectivas columnas (COL A – COL F):

TABLA 1	COL A	COL B	COL C
FILA 1-1	AAA	1	100
FILA 1-2	BBB	2	120
FILA 1-3	CCC	3	150

TABLA 2	COL D	COL E	COL F
FILA 2-1	EEE	2	10
FILA 2-2	FFF	3	14
FILA 2-3	GGG	4	18

Si realizamos el producto cartesiano o CROSS JOIN:

```
SELECT *
FROM tabla1,tabla2
```

La tabla resultante será la siguiente:

RESULTADO	COL A	COL B	COL C	COL D	COL E	COL F
FILA R-1	AAA	1	100	EEE	2	10
FILA R-2	AAA	1	100	FFF	3	14
FILA R-3	AAA	1	100	GGG	4	18
FILA R-4	BBB	2	120	EEE	2	10
FILA R-5	BBB	2	120	FFF	3	14
FILA R-6	BBB	2	120	GGG	4	18
FILA R-7	CCC	3	150	EEE	2	10
FILA R-8	CCC	3	150	FFF	3	14
FILA R-9	CCC	3	150	GGG	4	18

Imaginemos que solo tiene sentido realizar esta unión de tablas cuando la columna B es igual a la columna E, es decir:

```
SELECT *
FROM tabla1, tabla2
WHERE b = e
```

RESULTADO	COL A	COL B	COL C	COL D	COL E	COL F
FILA R-1	BBB	2	120	EEE	2	10
FILA R-2	CCC	3	150	FFF	3	14

Si bien esta operación que hemos realizado es sintácticamente correcta, existe una sentencia equivalente y recomendada cuando se quieran realizar **reuniones entre tablas**, detallando las condiciones de esa *reunión*. Se trata de la sentencia JOIN.

# Sentencia JOIN

**X** Edix Educación

---

La sentencia JOIN encajaría dentro de una consulta SELECT de la manera siguiente:

SELECT <columna1> [<columna2>,...,<columnaN>]

FROM <tabla1>

JOIN <tabla2>

ON <condicion>

[WHERE <condicion>]

[GROUP BY <columna1>,<columna2>,...,<columnaN>]

[HAVING <condicion>]

[ORDER BY <columna1>,<columna2>,...,<columnaN>]

[LIMIT N]

Por lo que la query equivalente y recomendada es la siguiente:

```
SELECT *
FROM tabla1
JOIN tabla2 ON tabla1.b = tabla2.e
```

Como decíamos, las dos queries son equivalentes si el optimizador que se ejecuta en segundo plano hace su trabajo. No obstante, se recomienda esta última forma: la **sentencia WHERE** debería utilizarse solamente para filtrar **información**, en caso de querer establecer condiciones de unión entre tablas usaremos la **sentencia JOIN**.

# Tipos de JOIN

 Edix Educación

---

Existen distintos **tipos de JOIN** según como queramos realizar la unión entre las tablas. Vamos a entender rápidamente las distintas opciones con un ejemplo.

Imaginemos que realizamos una **encuesta en una empresa**, recopilamos la información de estatura de sus trabajadores y lo almacenamos en una tabla de nuestra base de datos.

Pasado un tiempo, nos damos cuenta de que hubiera sido útil recoger también la información de peso, por lo que **volvemos a realizar la encuesta**, recopilar la información y almacenarla en una nueva tabla.

Llegado a este punto surge la necesidad de **unificar las dos tablas en una única**, para que, a futuro, en el caso de actualizarla con nuevas encuestas, la información se encuentre centralizada y el proceso sea más fácil de realizar.

El tipo de unión dependerá de **cómo queramos unir la información**. Expondremos todas las opciones posibles junto a su código. Las tablas recolectadas son las siguientes:

TABLA_ESTATURA	COD_EMPLEADO	ESTATURA
	100	176
	101	183
	102	154
	103	191
	104	173
	105	182
	106	156
	107	159
	108	162
	109	165
	110	192
	111	188
	112	175
	113	177
	114	183
	115	154
	116	160
	117	170
	118	193
	119	169

TABLA_PESO	COD_EMPLEADO	PESO
	100	80
	101	65
	102	90
	104	66
	105	76
	106	74
	107	86
	108	83
	110	64
	111	75
	112	77
	114	92
	115	90
	116	85
	117	56
	119	65
	120	63
	121	93
	122	77
	123	76

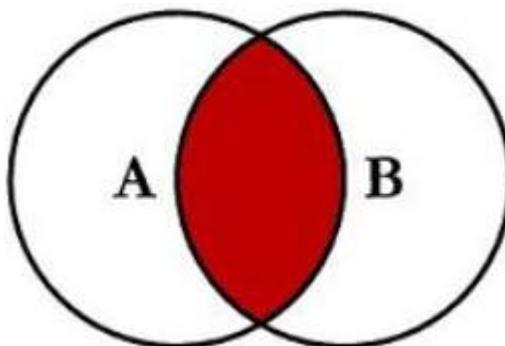
Como podemos ver, el listado de usuarios registrados en ambas tablas no es el mismo, esto obedece a que en el intervalo de tiempo entre las dos encuestas hubo movimientos de trabajadores.

Comencemos a enumerar los distintos tipos.

## **INNER JOIN o JOIN**

Usaremos INNER JOIN / JOIN si queremos obtener a los empleados para los que existe **información de estatura y peso**.

Es decir, si representamos ambas tablas como un conjunto de empleados respectivamente (conjunto A y conjunto B), la sentencia INNER JOIN/JOIN **representará a los empleados que son intersección de ambos conjuntos** (están presentes en ambos conjuntos).



TABLA_JOIN	COD_EMPLEADO	ESTATURA	COD_EMPLEADO	PESO
	<b>100</b>	<b>176</b>	<b>100</b>	<b>80</b>
	<b>101</b>	<b>183</b>	<b>101</b>	<b>65</b>
	<b>102</b>	<b>154</b>	<b>102</b>	<b>90</b>
	<b>104</b>	<b>173</b>	<b>104</b>	<b>66</b>
	<b>105</b>	<b>182</b>	<b>105</b>	<b>76</b>
	<b>106</b>	<b>156</b>	<b>106</b>	<b>74</b>
	<b>107</b>	<b>159</b>	<b>107</b>	<b>86</b>
	<b>108</b>	<b>162</b>	<b>108</b>	<b>83</b>
	<b>110</b>	<b>192</b>	<b>110</b>	<b>64</b>
	<b>111</b>	<b>188</b>	<b>111</b>	<b>75</b>
	<b>112</b>	<b>175</b>	<b>112</b>	<b>77</b>
	<b>114</b>	<b>183</b>	<b>114</b>	<b>92</b>
	<b>115</b>	<b>154</b>	<b>115</b>	<b>90</b>
	<b>116</b>	<b>160</b>	<b>116</b>	<b>85</b>
	<b>117</b>	<b>170</b>	<b>117</b>	<b>56</b>
	<b>119</b>	<b>169</b>	<b>119</b>	<b>65</b>

```
SELECT *
FROM tabla_estatura
JOIN tabla_peso
ON tabla_estatura.cod_empleado = tabla_peso.cod_empleado
```

### Nota

Aunque en la tabla estemos utilizando nombres de columnas en mayúsculas, después en la query estamos utilizando **nombres en minúsculas**. Es importante parar un momento y hacer un pequeño comentario acerca de la creación de tablas y el nombre de las tablas utilizadas.

Si hemos creado la tabla desde pgAdmin y hemos especificado el nombre de la tabla o algunas de las columnas con al menos una mayúscula, cuando hagamos referencia a la tabla o a la columna que contenga la letra en mayúscula, tendremos que **ponerlo con comillas dobles** (").

En caso contrario, la query que ejecutemos dará **siempre error de sintaxis**, pues el planificador no conocerá la tabla o columna a la que haces referencia.

En el caso de crear la tabla a través de código DDL (CREATE TABLE AS...), salvo que no pongamos entre comillas el nombre de la tabla y de las columnas, por defecto se convertirán todos los nombres a minúsculas.

Por mi parte, te recomendaría siempre utilizar nombres de tablas y columnas en minúsculas.

Observa cómo al hacer un SELECT \* devolvemos todas las columnas disponibles en las dos tablas unidas, generando información duplicada (columna COD\_EMPLEADO). Si quisieramos obtener solo las columnas no duplicadas podríamos hacerlo de dos maneras:

1

Seleccionando todas las columnas de la tabla que está a la izquierda del JOIN y añadiendo las columnas que queramos de la tabla que está a la derecha del JOIN.

```
SELECT tabla_estatura.*, tabla_peso.peso  
FROM tabla_estatura  
JOIN tabla_peso  
ON tabla_estatura.cod_empleado = tabla_peso.cod_empleado
```

Observa cómo hemos **seleccionado todas las columnas de la tabla 'tabla.estatura'**.

Simplemente , colocando su nombre + '!' + '\*' y cómo hemos hecho referencia a la columna peso de la tabla 'tabla\_peso' .

Realmente, podemos omitir en el caso de la columna 'peso' la coletilla 'tabla\_peso'. Al no existir otra columna 'peso' , **no existirá confusión a la hora de hacer referencia a ella**.

Igualmente, para **reducir la cantidad de texto a escribir**, podemos llamar con un alias a las tablas que entran en acción dentro del JOIN. A partir de ese momento, podemos hacer referencia a las columnas con 'alias.nombre\_columna'.

Por último, piensa lo útil que es poder usar `'.*'` para hacer **referencia a todas las columnas de nuestra tabla**. En nuestro caso, el número de columnas es bastante reducido, pero imagina situaciones en la que tus tablas cuenten con decenas de columnas.

Por lo que, teniendo en cuenta toda esta información, la consulta finalmente podría quedar así:

```
SELECT te.* , peso  
FROM tabla_estatura te  
JOIN tabla_peso tp  
ON te.cod_empleado = tp.cod_empleado
```

2

**Seleccionando una a una las columnas** que queramos obtener en la consulta (tendremos en cuenta todas las recomendaciones vistas en el punto anterior):

```
SELECT te.cod_empleado, estatura, peso  
FROM tabla_estatura te  
JOIN tabla_peso tp  
ON te.cod_empleado = tp.cod_empleado
```

## LEFT OUTER JOIN o LEFT JOIN

Usaremos LEFT OUTER JOIN / LEFT JOIN cuando queramos obtener la información de los empleados que se encuentre a la izquierda del LEFT JOIN y pegarle toda la información disponible de la tabla que se encuentra a la derecha del LEFT JOIN. Si no existe información en la parte derecha del empleado, las columnas seleccionadas de la parte derecha valdrán NULL.

Es decir, si representamos ambas tablas como un conjunto de empleados respectivamente (conjunto A y conjunto B), la sentencia LEFT OUTER JOIN/LEFT JOIN representará a los empleados que pertenecen al conjunto A y les añadirá la información de B donde sea que exista:

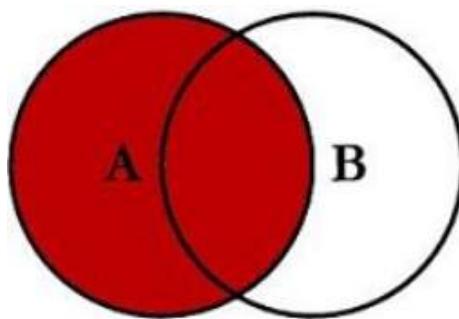


TABLA LEFT JOIN	COD_EMPLEADO	ESTATURA	PESO
	100	176	80
	101	183	65
	102	154	90
	103	191	NULL
	104	173	66
	105	182	76
	106	156	74
	107	159	86
	108	162	83
	109	165	NULL
	110	192	64
	111	188	75
	112	175	77
	113	177	NULL
	114	183	92
	115	154	90
	116	160	85
	117	170	56
	118	193	NULL
	119	169	65

```
SELECT te.cod_empleado,estatura,peso  
FROM tabla_estatura te  
LEFT JOIN tabla_peso tp  
ON te.cod_empleado = tp.cod_empleado
```

## RIGHT OUTER JOIN o RIGHT JOIN

Usaremos RIGHT OUTER JOIN / RIGHT JOIN cuando queramos obtener la **información de los empleados que se encuentre a la derecha** del RIGHT JOIN y *pegarle* toda la información disponible de la tabla que se encuentra a la izquierda del RIGHT JOIN. Si no existe información en la parte izquierda del empleado, las columnas seleccionadas de la parte izquierda valdrán NULL.

Es decir, si representamos ambas tablas como un conjunto de empleados respectivamente (conjunto A y conjunto B), la sentencia RIGHT OUTER JOIN/RIGHT JOIN representará a los **empleados que pertenecen a B** y les añadirá la información de A donde sea que exista:

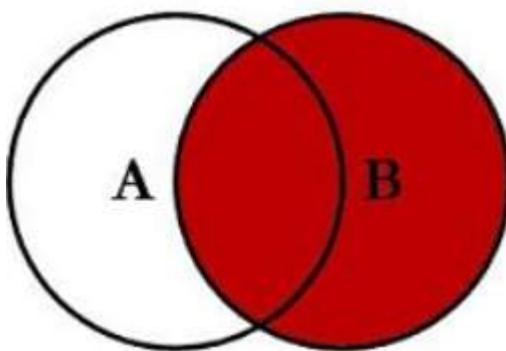


TABLA RIGHT JOIN	COD_EMPLEADO	PESO	ESTATURA
	100	80	176
	101	65	183
	102	90	154
	104	66	173
	105	76	182
	106	74	156
	107	86	159
	108	83	162
	110	64	192
	111	75	188
	112	77	175
	114	92	183
	115	90	154
	116	85	160
	117	56	170
	119	65	169
	120	63	NULL
	121	93	NULL
	122	77	NULL
	123	76	NULL

```
SELECT tp.cod_empleado, peso, estatura
FROM tabla_estatura te
RIGHT JOIN tabla_peso tp
ON te.cod_empleado = tp.cod_empleado
```

Observa cómo en esta última query hemos seleccionado cod\_empleado de la tabla 'tabla\_peso', ¿qué hubiera pasado si hubiésemos seleccionado el cod\_empleado de la tabla 'tabla\_estatura'?

---

**Pruébalo tú mismo creando estas tablas en tu base de datos de prueba (después puedes borrarlas).**

## FULL OUTER JOIN o FULL JOIN

Usaremos FULL OUTER JOIN / FULL JOIN cuando queramos **obtener la información de todos los empleados**. Independientemente de que se encuentren a la derecha del FULL JOIN o a la izquierda. Para las distintas columnas se intentará llenar la información, en el caso de que no exista, tomarán **valor NULL**.

Es decir, si representamos ambas tablas como un conjunto de empleados respectivamente (conjunto A y conjunto B), la sentencia FULL OUTER JOIN/FULL JOIN **representará a los empleados que pertenezcan a A o B** y les añadirá la información de A en B donde exista y la de B en A donde exista:

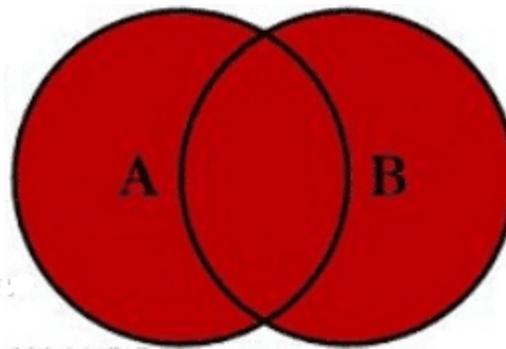


TABLA FULL JOIN	COD_EMPLEADO	ESTATURA	COD_EMPLEADO	PESO
	100	176	100	80
	101	183	101	65
	102	154	102	90
	103	191	NULL	NULL
	104	173	104	66
	105	182	105	76
	106	156	106	74
	107	159	107	86
	108	162	108	83
	109	165	NULL	NULL
	110	192	110	64
	111	188	111	75
	112	175	112	77
	113	177	NULL	NULL
	114	183	114	92
	115	154	115	90
	116	160	116	85
	117	170	117	56
	118	193	NULL	NULL
	119	169	119	65
	NULL	NULL	120	63
	NULL	NULL	121	93
	NULL	NULL	122	77
	NULL	NULL	123	76

```
SELECT te.cod_empleado, estatura, tp.cod_empleado, peso
FROM tabla_estatura te
FULL JOIN tabla_peso tp
ON te.cod_empleado = tp.cod_empleado
```

Para que la tabla quede de una manera más legible, tendremos que dar un paso más y **fusionar las dos columnas** de cod\_empleado en una sola. Pero de momento, para el caso que nos atañe, no es importante realizar ese paso.

Nos quedan tres tipos más de JOIN por conocer, pero no te preocupes, son casos particulares de los que ya hemos visto.

## LEFT OUTER JOIN / LEFT JOIN WHERE B IS NULL

Se trata como comentaba de un caso particular del LEFT JOIN. Combinaremos la **sentencia JOIN junto a una condición en el WHERE**. En nuestro caso, lo usaremos cuando queramos conocer los empleados para los que tenemos datos de estatura, pero no datos de peso.

Es decir, si representamos ambas tablas como un conjunto de empleados respectivamente (conjunto A y conjunto B), la sentencia LEFT OUTER JOIN/LEFT JOIN WHERE B IS NULL representará a los empleados que pertenecen a A pero que no pertenecen a B:

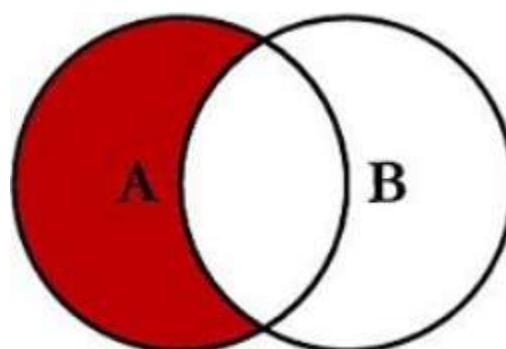


TABLA LEFT JOIN WITH B IS NULL	COD_EMPLEADO	ESTATURA	PESO
	<b>103</b>	<b>191</b>	<b>NULL</b>
	<b>109</b>	<b>165</b>	<b>NULL</b>
	<b>113</b>	<b>177</b>	<b>NULL</b>
	<b>118</b>	<b>193</b>	<b>NULL</b>

```
SELECT te.cod_empleado,estatura,peso
FROM tabla_estatura te
LEFT JOIN tabla_peso tp
ON te.cod_empleado = tp.cod_empleado
WHERE tp.cod_empleado is null
```

Como supondrás, la información de peso **no tiene sentido obtenerla en esta consulta**, puesto que ya estamos imponiendo en el WHERE una condición que precisamente seleccione solo los casos en los que la información de peso no está disponible.

## RIGHT OUTER JOIN / RIGHT JOIN WHERE A IS NULL

De manera similar al caso anterior, podemos realizar la misma operación para RIGHT JOINS: combinaremos la **sentencia JOIN junto a una condición en el WHERE**. En nuestro caso, lo usaremos cuando queramos conocer los empleados para los que tenemos datos de peso, pero no datos de estatura.

Es decir, si representamos ambas tablas como un conjunto de empleados respectivamente (conjunto A y conjunto B), la sentencia RIGHT OUTER JOIN/RIGHT JOIN WHERE A IS NULL representará a los empleados que pertenecen a B pero que no pertenecen a A:

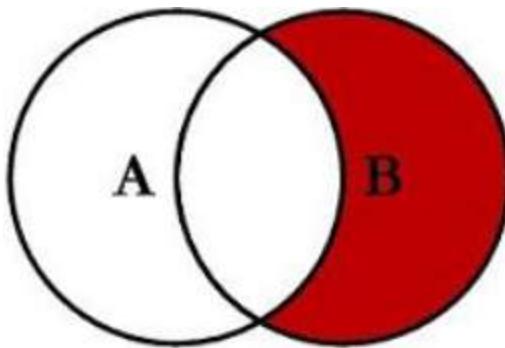


TABLA RIGHT JOIN WITH A IS NULL	COD_EMPLEADO	PESO	ESTATURA
	120	63	NULL
	121	93	NULL
	122	77	NULL
	123	76	NULL

```
SELECT tp.cod_empleado,peso,estatura
FROM tabla_estatura te
RIGHT JOIN tabla_peso tp
ON te.cod_empleado = tp.cod_empleado
WHERE te.cod_empleado is null
```

De manera similar al caso anterior, la información de estatura no tiene sentido obtenerla en esta consulta.

### FULL OUTER JOIN / FULL JOIN WHERE A IS NULL OR B IS NULL

De manera similar a los casos anteriores, podemos realizar la misma operación para FULL JOINS: combinaremos la sentencia JOIN junto a dos condiciones en el WHERE. En nuestro caso, lo usaremos cuando queramos conocer los empleados para los que no tenemos datos de peso o datos de estatura.

Es decir, si representamos ambas tablas como un conjunto de empleados respectivamente (conjunto A y conjunto B), la sentencia FULL OUTER JOIN/RIGHT JOIN WHERE A IS NULL OR B IS NULL representará a los empleados que pertenecen a A, pero que no pertenecen a B, junto a los empleados que pertenecen a B, pero no pertenecen a A:

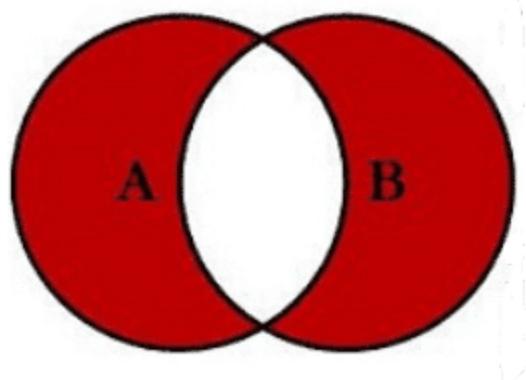


TABLA FULL JOIN WITH A IS NULL OR B IS NULL	COD_EMPLEADO	ESTATURA	COD_EMPLEADO	PESO
	103	191	NULL	NULL
	109	165	NULL	NULL
	113	177	NULL	NULL
	118	193	NULL	NULL
	NULL	NULL	120	63
	NULL	NULL	121	93
	NULL	NULL	122	77
	NULL	NULL	123	76

```
SELECT te.cod_empleado, estatura, tp.cod_empleado, peso
FROM tabla_estatura te
FULL JOIN tabla_peso tp
ON te.cod_empleado = tp.cod_empleado
WHERE te.cod_empleado is null or tp.cod_empleado is null
```

**Nota.** De momento solo hemos realizado JOINS entre dos tablas, pero podemos unir tantas tablas como queramos a la vez. Lo vamos a poner en práctica en los siguientes ejemplos, ¡no te preocupes!

## Ejemplos comando JOIN

Imaginemos las siguientes tablas y su estructura: alumnos, asignaturas y examen.

alumno		examen		asignatura	
<b>id_alumno</b>	integer			<b>id_asignatura</b>	integer
nombre	varchar			nombre	varchar
apellido1	varchar	<b>id_alumno</b>	integer	curso	integer
apellido2	varchar	fecha_examen	date		
curso	integer	nota	double		

- La tabla ‘alumno’ recoge al conjunto de alumnos de un colegio. Un alumno consta de un nombre y apellidos, además del curso (1,2,3,...,n) en el que estudie.
- La tabla ‘asignatura’ recoge todas las asignaturas que son impartidas en el colegio. Una asignatura consta de un nombre, además del curso en el que es impartida. Así podríamos tener la asignatura de matemáticas que se imparte en el curso 1, 2, 3, etc.

- Por último, la tabla ‘examen’ recoge todos los exámenes que se han realizado desde el comienzo del curso escolar. Un examen se conformará a partir del alumno que lo ha realizado, además de la asignatura donde ha sido realizado. Asociado a estas dos entidades tendrá una fecha de realización, además de una nota asociada.

**Nota.** Te recomiendo crear las tablas anteriores, además de generar datos para ellas. Así podrás poner en práctica los conocimientos adquiridos para la parte de generación (lenguaje DDL) y hacer uso de ellos realizando las consultas que verás a continuación.

1

Construir una tabla que contenga la siguiente información: listado de exámenes en los que aparezca el nombre y apellidos del alumno, el nombre de la asignatura, la fecha de realización, solo para los exámenes cuya nota sea menor a un 5.0.

```
SELECT a.nombre,a.apellido1,a.apellido2,asig.nombre,e.fecha_examen
FROM examen e
JOIN alumno a ON a.id_alumno = e.id_alumno
JOIN asignatura asig ON asig.id_asignatura = e.id_asignatura
WHERE
e.nota < 5.0
```

2

¿Existe algún alumno que no haya realizado al menos un examen?

```
SELECT id_alumno
FROM alumno a
LEFT JOIN examen e ON a.id_alumno = e.id_alumno
WHERE e.id_alumno is null
```

3

¿Cuántos exámenes se han realizado para la asignatura de matemáticas de primer curso? (De momento no sabemos contar dentro de una query, construyamos la tabla necesaria para resolver la pregunta y contemos el número de filas que devuelve la query).

```
SELECT e.*  
FROM examen e  
JOIN asignatura a ON e.id_asignatura = a.id_asignatura  
WHERE a.nombre = 'matemáticas' and a.curso = 1
```

4

¿De qué asignatura y para qué curso es el primer examen que se realizó en el curso académico?

```
SELECT a.nombre,a.curso  
FROM examen e  
JOIN asignatura a ON a.id_asignatura = e.id_asignatura  
ORDER BY fecha_examen asc  
LIMIT 1
```

5

¿Cuál es el examen (asignatura y curso) más reciente que ha realizado el alumno Juan López Martínez?

```
SELECT a.nombre,a.curso  
FROM examen e  
JOIN alumno a ON e.id_alumno = a.id_alumno  
JOIN asignatura asig ON asig.id_asignatura = e.id_asignatura  
WHERE a.nombre = 'Juan' and a.apellido1 = 'López' and a.apellido2 = 'Martínez'  
ORDER BY fecha_examen desc  
LIMIT 1
```

# Manejo sobre conjuntos: UNION

 Edix Educación

---

Al igual que hemos unido tablas de manera horizontal (la tabla resultante crece en columnas), existe también la forma de unir tablas de manera vertical (la tabla resultante crece en número de filas).

El comando encargado de realizar esta característica es el comando UNION. Para poder utilizarlo es necesario respetar tres reglas:

- 1 Las dos tablas a unir deben tener el mismo número de columnas.
  - 2 Las dos tablas a unir deben tener el mismo orden de columnas.
  - 3 Las dos tablas a unir deben tener el mismo tipo entre las columnas semejantes.
- 

El no cumplimiento de alguna de las tres reglas **generará un error** al intentar ejecutar la consulta.

Es muy importante advertir que el comando UNION **descarta filas iguales**, es decir, filas que columna a columna presenten el mismo valor. Si no queremos que se realice esta operación, deberemos usar el comando **UNION ALL**, que trabaja igual que UNION, pero **no elimina filas repetidas**.

Posiblemente la mejor forma de entender todo esto sea con un ejemplo.

Imaginemos que queremos realizar una colecta y apuntamos los ingresos en una tabla de base de datos. Almacenaremos el nombre de la persona y también el importe aportado.

Nuestra tabla tiene el siguiente contenido:

TABLA APORTACIONES	NOMBRE	VALOR
	Luis	100
	Juan	50
	Pedro	60
	Lucía	120

Sin embargo, nos damos cuenta de que la recolección de información ha sido errónea y es posible que otras aportaciones hayan sido realizadas, pero no apuntadas en nuestra tabla. Por suerte, otra persona ha realizado la misma tarea que nosotros.

Su tabla tiene el siguiente contenido:

TABLA APORTACIONES	NOMBRE	VALOR
	Luis	100
	María	100
	Esteban	80
	Pedro	60
	Lucía	120

Efectivamente, su tabla tiene ingresos que la nuestra no, pero lamentablemente también tiene el inconveniente de que hay ingresos que esa persona tampoco ha registrado. La mejor forma de avanzar será unificar las tablas:

```
SELECT nombre,valor  
FROM aportaciones1  
UNION  
SELECT nombre,valor  
FROM aportaciones2
```

Que devuelve una tabla con este contenido:

TABLA APORTACIONES UNION	NOMBRE	VALOR
	Luis	100
	Juan	50
	Pedro	60
	Lucía	120
	María	100
	Esteban	80

Sin embargo, si hubiésemos utilizado el comando UNION ALL, el contenido de la tabla sería el siguiente:

```
SELECT nombre,valor
FROM aportaciones1
UNION ALL
SELECT nombre,valor
FROM aportaciones2
```

TABLA APORTACIONES UNION	NOMBRE	VALOR
	Luis	100
	Juan	50
	Pedro	60
	Lucía	120
	Luis	100
	María	100
	Esteban	80
	Pedro	60
	Lucía	120

Como verás, las **filas repetidas se añaden también**, en el caso del UNION se omiten.

Las siguientes ejecuciones devolverían un error al ser lanzadas:

```
SELECT nombre,valor,otra_columna  
FROM aportaciones1  
UNION  
SELECT nombre,valor  
FROM aportaciones2
```

1

Regla número 1: las dos tablas deben tener el mismo número de columnas a la hora de hacer la selección.

```
SELECT valor,nombre  
FROM aportaciones1  
UNION  
SELECT nombre,valor  
FROM aportaciones2
```

2

Regla número 2: las dos tablas deben tener el mismo orden a la hora de seleccionar las columnas.

Imaginemos que tenemos una tabla 'aportaciones3' que es una copia de 'aportaciones2', la única diferencia es que la columna 'valor' en vez de ser una columna numérica está traducida a **formato STRING**.

3

La siguiente query devolvería un error al incumplir la regla número 3: las dos tablas a unir deben tener el mismo tipo entre las columnas semejantes.

```
SELECT nombre,valor  
FROM aportaciones1  
UNION  
SELECT nombre,valor  
FROM aportaciones3
```

# Manejo sobre conjuntos: INTERSECT

**X** Edix Educación

---

De una manera muy parecida al comando UNION funciona el comando INTERSECT. La salvedad es que, en vez de devolver una tabla que es el resultado de unificar las filas de una tabla junto a la otra, **devuelve las filas que aparecen en ambas tablas**.

Si volvemos a reutilizar las tablas que hemos construido antes (aportaciones1 y aportaciones2), el resultado de realizar la operación INTERSECT sería el siguiente:

```
SELECT nombre,valor  
FROM aportaciones1  
INTERSECT  
SELECT nombre,valor  
FROM aportaciones2
```

TABLA APORTACIONES INTERSECT	NOMBRE	VALOR
	Luis	100
	Pedro	60
	Lucía	120

---

Las reglas que comentamos para el comando UNION son las mismas que para el comando INTERSECT.

---

1

Las dos tablas a unir deben tener el mismo número de columnas.

2

Las dos tablas a unir deben tener el mismo orden de columnas.

3

Las dos tablas a unir deben tener el mismo tipo entre las columnas semejantes.

# Manejo sobre conjuntos: EXCEPT

X Edix Educación

---

El comando EXCEPT es muy parecido a los dos que ya hemos visto. En este caso, cuando realicemos la consulta sobre dos tablas el orden en el que las coloquemos será muy importante, pues **esta operación no es conmutativa**. Es decir, los resultados producidos al realizar 'tablaA except tablaB' son distintos a los producidos por la consulta 'tablaB except tablaA'.

---

La operación EXCEPT devuelve todas las filas de la tabla A, excepto las que están presentes en la tabla B.

Realicemos la consulta sobre nuestras tablas de aportaciones para entender bien el funcionamiento a partir de los resultados obtenidos:

```
SELECT nombre,valor  
FROM aportaciones1  
EXCEPT  
SELECT nombre,valor  
FROM aportaciones2
```

aportaciones1 EXCEPT aportaciones2

TABLA APORTACIONES EXCEPT	NOMBRE	VALOR
	Juan	50
INTERSECT 2		
	María	100
	Esteban	80

```
SELECT nombre,valor  
FROM aportaciones2  
EXCEPT  
SELECT nombre,valor  
FROM aportaciones1
```

aportaciones2 EXCEPT aportaciones1

TABLA APORTACIONES INTERSECT 2	NOMBRE	VALOR
	María	100
	Esteban	80

Las reglas que hemos venido comentando anteriormente también aplican para el comando EXCEPT.

1

Las dos tablas a unir deben tener el mismo número de columnas.

2

Las dos tablas a unir deben tener el mismo orden de columnas.

3

Las dos tablas a unir deben tener el mismo tipo entre las columnas semejantes.

# Ejercicios propuestos y sus soluciones

**X** Edix Educación

---

Te propongo los siguientes ejercicios resueltos (código + resultado) para que practiques.

Lista todas las tarjetas (toda la información de la tabla tarjeta) para las tarjetas de tipo 'CREDITO' que no hayan realizado ninguna compra. Ordénalo por id de tarjeta.

## Respuesta

---

```
SELECT tarjeta.*  
FROM tarjeta  
LEFT JOIN ticket ON ticket.id_tarjeta = tarjeta.id  
WHERE ticket.id_tarjeta is null  
AND tipo_tarjeta = 'CREDITO'  
ORDER BY id
```

## Resultado

	<b>id</b> integer	<b>id_cliente</b> integer	<b>fecha_creacion</b> date	<b>tipo_tarjeta</b> character varying (10)
1	100400503	10536	2019-11-10	CREDITO
2	100400925	10805	2020-10-27	CREDITO
3	100401039	10863	2019-10-03	CREDITO
4	100401186	10995	2019-10-10	CREDITO
5	100401241	10457	2019-11-26	CREDITO
6	100401304	10277	2019-10-01	CREDITO
7	100401454	10112	2020-06-06	CREDITO
8	100401456	10129	2019-02-19	CREDITO
9	100401459	10774	2020-10-02	CREDITO
10	100401460	10633	2020-12-14	CREDITO
11	100401464	10624	2019-12-13	CREDITO
12	100401467	10630	2020-06-02	CREDITO
13	100401471	10498	2020-08-04	CREDITO
14	100401472	10918	2020-11-19	CREDITO
15	100401476	10428	2020-01-22	CREDITO
16	100401478	10453	2020-01-08	CREDITO
17	100401483	10341	2019-01-16	CREDITO

¿Cuántos tickets se han realizado en la tienda con nombre 'Toledo III' para la caja 1? De momento, no sabemos contar directamente en SQL, por lo que haremos una captura de las filas que devuelve pgAdmin.

**Respuesta**

```
SELECT *
FROM ticket t
JOIN tienda ti on t.id_tienda = ti.id
WHERE nombre = 'Toledo III' and caja = 1
```

**Resultado**

El resultado son 5 tickets.

	id integer	id_tarjeta integer	fecha date	hora time without time zone	id_tienda integer	caja integer	importe double precision	id integer	nombre character varying (60)	provincia character varying (60)	tipologia character
1	1000621	100400191	2020-02-09	20:56:09	30033	1	52.65	30033	Toledo III	Toledo	Superme
2	1001921	100401309	2020-04-06	13:42:02	30033	1	26.19	30033	Toledo III	Toledo	Superme
3	1002021	100400460	2020-08-20	20:30:33	30033	1	22.9	30033	Toledo III	Toledo	Superme
4	1005103	100400906	2020-04-26	18:31:25	30033	1	74.65	30033	Toledo III	Toledo	Superme
5	1005349	100400544	2020-02-05	20:57:29	30033	1	48.3	30033	Toledo III	Toledo	Superme

✓ Successfully run. Total query runtime: 212 msec. 5 rows affected.

Un poco más difícil que la anterior. ¿Cuántos tickets se han generado en la 'Zona Sur' y 'Área Centro' para los que el importe es mayor a 100€ y la caja donde se ha realizado la compra sea la 11? Devuelve solo los id de los tickets.

### Respuesta

```
SELECT t.id
FROM ticket t
JOIN tienda ti ON t.id_tienda = ti.id
JOIN estructura_comercial c ON c.id = ti.id_estructura_comercial
WHERE area = 'Área Centro'
AND zona = 'Zona Sur'
AND importe > 100
AND caja = 11
```

### Resultado

**Explicación.** La dificultad en esta query reside en que para llegar a poder relacionar la tabla 'ticket' con la tabla 'estructura comercial', tenemos que pasar previamente por la tabla 'tienda'. Es por ello que tenemos que **realizar dos JOINS**. En el primero, relacionamos información de tickets con tiendas. En la segunda, utilizamos la columna id\_estructura\_comercial que nos proporciona la tabla 'tienda' para unir por fin con la tabla 'estructura\_comercial'.

	<b>id</b> integer
1	1003426

Más difícil todavía. ¿Cuántos tickets han sido realizados por clientes cuya provincia sea la misma que la provincia a la que pertenezca la tienda donde se realizó la compra?

**Nota.** Como columnas de la sentencia SELECT pon solo esto: 'count(\*)', aunque nos estamos adelantando al siguiente fastbook, con eso podremos generar el número de filas que devuelve la sentencia SELECT.

**Planteamiento.** Es parecido al problema anterior, pero entran en acción más tablas. Piensa que tenemos que relacionar la provincia del cliente con la provincia de la tienda. Para ello, tendremos que unir la tabla de ticket con la tabla de tarjeta. Así sabremos el identificador de cliente.

Con el identificador de cliente relacionamos con la tabla de cliente. Es en ese momento cuando ya podemos utilizar la columna 'provincia' de la tabla 'cliente'. Para el caso de la provincia de la tienda, de forma similar, relacionamos la tabla ticket con la tabla tienda, a través del campo id\_tienda. A partir de ese momento, ya tenemos disponible la columna 'provincia' de la tabla 'tienda'.

### Respuesta

```
SELECT count(*)
FROM ticket t
JOIN tarjeta tar ON t.id_tarjeta = tar.id
JOIN cliente c ON tar.id_cliente = c.id
JOIN tienda ti ON t.id_tienda = ti.id
WHERE c.provincia = ti.provincia
```

### Resultado

	count	bigint
1	92	

# Conclusiones

X Edix Educación

---

En este fastbook hemos continuado explorando maneras de **realizar consultas sobre tablas de nuestra base de datos**. Hemos aprendido a utilizar varias tablas a la vez a través del comando JOIN y sus distintos tipos, además de los operadores de conjuntos UNION, EXCEPT e INTERSECT.

Entre lo más relevante cabe destacar:

- Existen diferentes maneras de unir las tablas con las que estamos trabajando y es importante tener claro el cometido de cada una.
- Cuando usemos un LEFT JOIN, obtendremos todos los registros de la primera tabla (la del FROM), agregando siempre que sea posible los valores obtenidos de la segunda tabla (la del JOIN)
- Cuando usemos un RIGTH JOIN, obtendremos todos los registros de la segunda tabla (la indicada en el JOIN), agregando siempre que sea posible los valores obtenidos de la primera tabla (la del FROM).
- Cuando usemos un INNER JOIN, obtendremos los registros que coincidan en las dos tablas.
- Si queremos agregar la información de las dos tablas, podemos usar los distintos operadores de conjuntos: UNION, EXCEPT e INTERSECT. Lo más importante es que recordemos que la estructura de las dos tablas tiene que ser idéntica.

¡Enhорabuena! Fastbook superado

edix

Creamos Digital Workers