



**Argentina  
programa  
4.0**



Ministerio de Economía  
**Argentina**

Secretaría de  
Economía del Conocimiento

***primero  
la gente***

# Clase 23: Bases de datos Relacionales

## Tablas combinadas en consultas SQL

## Agenda de hoy

- A. La importancia de consultar datos combinando tablas
- B. Establecer relaciones entre tablas
  - a. INNER JOIN
  - b. LEFT JOIN
  - c. RIGHT JOIN
  - d. OUTER JOIN
  - e. UNION ALL
- C. Combinar JOIN y Subconsultas SQL



# La importancia de consultar datos combinando tablas

Continuamos con el proceso de aprendizaje iniciado la clase anterior. Luego de una teoría bastante intensa, tomaremos al lenguaje SQL para que nos ayude a representar gráficamente los aspectos más importantes entre tablas relacionales y cómo aprovechar datos esparcidos en dos o más tablas, para armar una consulta con la información que más importancia tenga.



# La importancia de consultar datos combinando tablas

Como hablamos al inicio de la cursada, mientras repasábamos los fundamentos de las bases de datos relacionales, es clave que tengamos estructurada la información en diferentes tablas.

A la larga, esta estructura será mucho más clara y limpia para nosotros, y gracias a SQL, podremos tomar aquello que necesitemos de cada tabla, de una forma prolija y clara.



# La importancia de consultar datos combinando tablas

En la estructura básica de creación de una base de datos relacional, las tablas y los índices primarios y foráneos son lo que mayormente utilizaremos cuando diseñamos una bb.dd de este tipo.

Las claves primarias son importantes para poder optimizar la búsqueda de registros resultantes de consultas donde aplicamos cláusulas WHERE simples o múltiples.



# La importancia de consultar datos combinando tablas

Mientras que, los índices o claves foráneas, son importantes para cuando tenemos que traer múltiples datos de diferentes tablas, y visualizar una estructura de registros uniforme y coherente.

Allí, las claves foráneas son el nexo específico, no solo para establecer relación entre datos, sino también para evitar que la información almacenada en tablas, quede huérfana o inconsistente.



# La importancia de consultar datos combinando tablas

Y, para comprender cómo podemos obtener múltiples datos de múltiples tablas, haremos un repaso sobre determinadas cláusulas importantes que incluye SQL.

Estas cláusulas, se apoyan en las bases de la teoría de conjuntos de las matemáticas, para poder comprender más fácilmente qué información obtendremos, cuando tomamos información de múltiples tablas de datos.

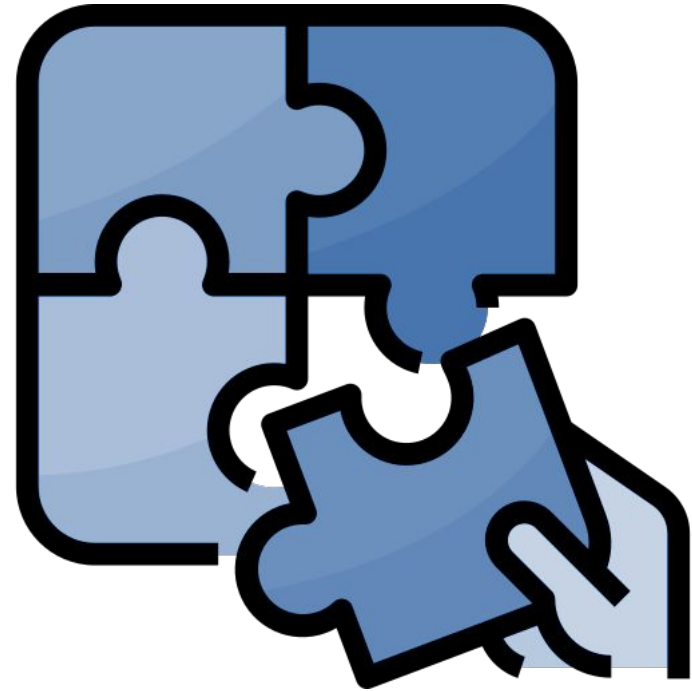




**JOIN**

## Establecer relaciones entre tablas

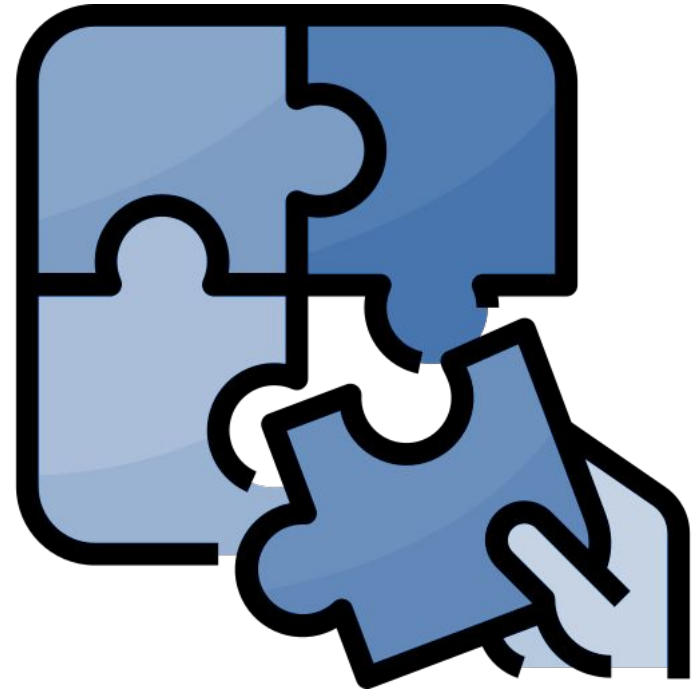
Como mencionamos recientemente, en el momento de trabajar bajo un modelo relacional es muy probable que, al ejecutar consultas de selección, debamos obtener datos que se complementan entre sí, desde dos o más tablas diferentes.



# Establecer relaciones entre tablas

Para poder solucionar esto, contamos con la cláusula JOIN y sus diferentes variantes.

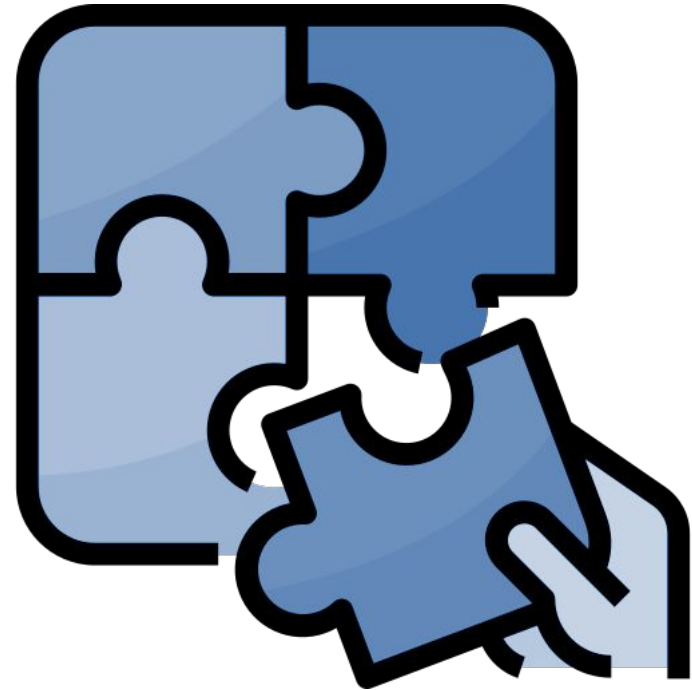
La misma forma parte del lenguaje SQL, y nos permite establecer el mecanismo de conexión entre dos o más tablas para ejecutar, en una consulta resultante, la conjunción de datos que éstas comparten entre sí.



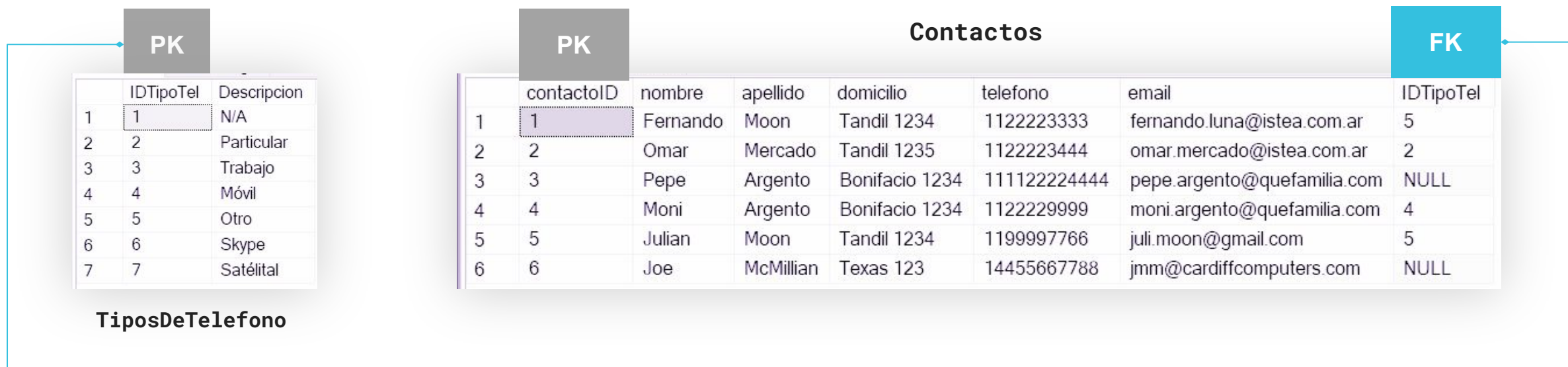
## Establecer relaciones entre tablas

JOIN se ocupa de definir cuál es el medio de relación entre estas tablas, basándose en campos en común que estas tengan:

- claves primarias
- claves foráneas



## Establecer relaciones entre tablas



Retomamos el ejemplo de la tabla **Contactos** y **TiposDeTelefono** que utilizamos en nuestro encuentro anterior. Tomaremos registros específicos de cada una de ellas, utilizando JOIN.

## Establecer relaciones entre tablas

Sobre el set de datos del ejemplo anterior, construimos la siguiente consulta de Selección, combinando ambas tablas. Aquí vemos en acción la cláusula **JOIN** junto con la palabra reservada **ON**.

Analicemos a continuación esta estructura de consulta SQL.

Subconsultas SQL

```
SELECT nombre, apellido, domicilio, email, teléfono, descripcion
FROM Contactos C
JOIN TiposDeTelefono T
ON C.idTipoTel = T.idTipoTel;
```

# Establecer relaciones entre tablas

```
Subconsultas SQL

SELECT nombre, apellido, domicilio, email, teléfono, descripcion
FROM Contactos C
JOIN TiposDeTelefono T
ON C.idTipoTel = T.idTipoTel;
```

**JOIN** se ocupa de unir la tabla **Contactos**, mencionada en la línea de arriba, con la tabla **TiposDeTelefono**.

## Establecer relaciones entre tablas

```
Subconsultas SQL

SELECT nombre, apellido, domicilio, email, teléfono, descripcion
FROM Contactos C
JOIN TiposDeTelefono T
ON C.idTipoTel = T.idTipoTel;
```

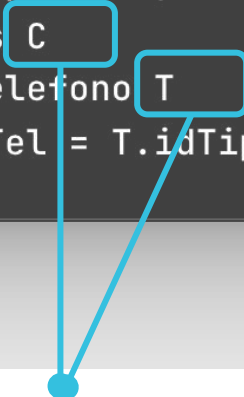
La cláusula **ON** se utiliza para definir el punto en común entre estas dos tablas. En este caso, el campo **idTipoTel** es el nexa, ya que en la tabla **TiposDeTelefono** es la clave primaria, y en la tabla **Contactos** es la clave foránea.



## Establecer relaciones entre tablas

```
Subconsultas SQL

SELECT nombre, apellido, domicilio, email, teléfono, descripcion
FROM Contactos C
JOIN TiposDeTelefono T
ON C.idTipoTel = T.idTipoTel;
```



Las letras que vemos por allí sueltas, son representaciones, o Alias, para cada una de las tablas. Cuando comenzamos a traer columnas de diferentes tablas, siempre conviene aplicar el uso de alias en tablas, para que a simple vista veamos de dónde proviene cada columna o campo.

## Establecer relaciones entre tablas

```
Subconsultas SQL

SELECT C.nombre, C.apellido, C.domicilio, C.email, C.teléfono, T.descripcion
FROM Contactos C
JOIN TiposDeTelefono T
ON C.idTipoTel = T.idTipoTel;
```

Y cuando definimos alias en las tablas, es necesario aplicarlo también en la definición de las columnas que utilizaremos. En algunos motores SQL se resuelve automáticamente, pero en otros puede generar error, sobre todo cuando hay ambigüedad en el nombre de las columnas.

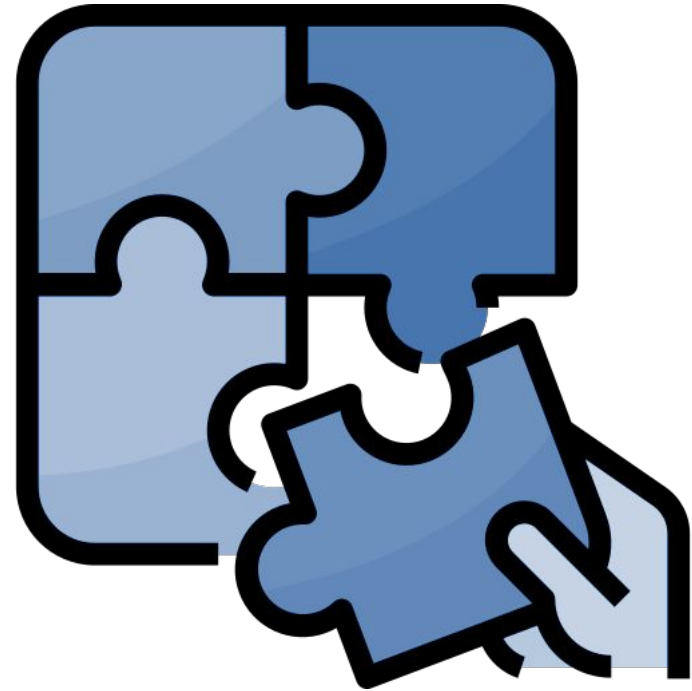
**Esta sería la estructura correcta para la consulta de selección que utiliza JOIN.**

# Tipos de JOIN

## Tipos de JOIN

JOIN se puede utilizar de manera simple, tal como vimos en el ejemplo de consulta anterior, aunque en realidad dispone de varios tipos de JOIN.

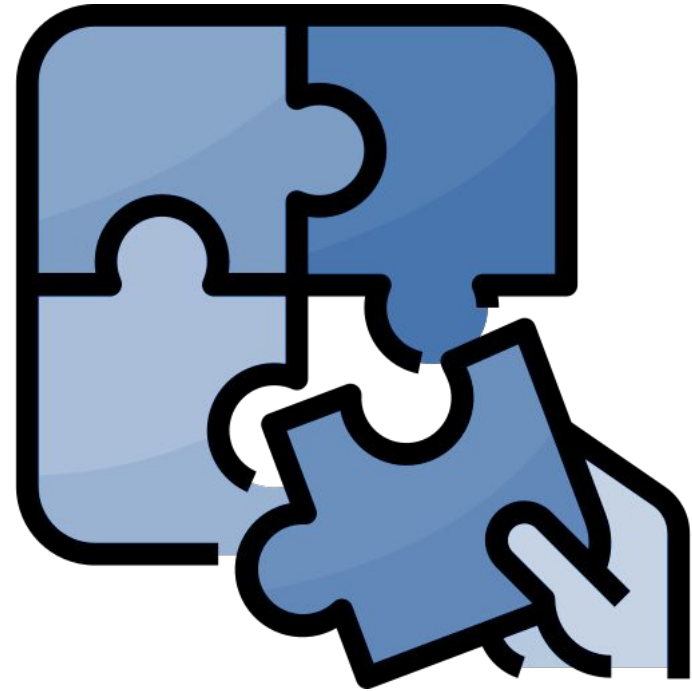
En la aplicación de sus diferentes tipos, entra en juego lo que se denomina **tabla de mayor peso** y **tabla de menor peso**. Esto es el condicional que nos permitirá ubicar a las tablas en la posición que les corresponde.



## Tipos de JOIN

Y, para una mejor representación de JOIN, también se acopla el modelo de lo que alguna vez vimos en la escuela primaria/secundaria, relacionada a la **Teoría de Conjuntos**.

Este, es el mejor modelo representativo, para que apliquemos en el aprendizaje del uso de consultas SQL combinando dos o más tablas.



## INNER JOIN

La profe te compartirá un archivo **.SQL** para que ejecutes su script sobre alguna base de datos nueva, existente, o de pruebas que tengas en MySQL.

Se crearán dos tablas de contactos y tipos de teléfono para abordar las diferentes **cláusulas SQL + JOIN**.

*Tómate unos minutos para ejecutarlo y ver las tablas creadas. Recuerda agregar el comando **USE** con tu bb.dd. preferida.*

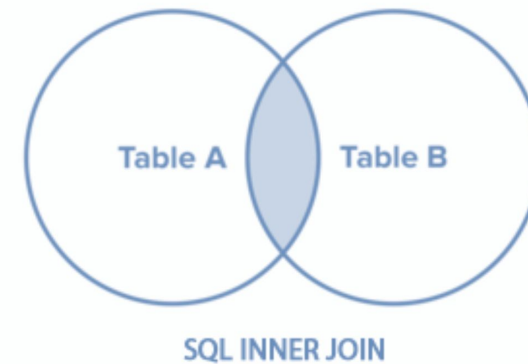


**JOIN || INNER JOIN**

# INNER JOIN

Representación de **INNER JOIN**:

Trae todos los registros tomando la relación de ambas tablas y filtrando por aquellos registros donde sí hay un “*match*”.





# INNER JOIN

El uso de JOIN o INNER JOIN es indistinto en SQL. Cualquiera de estas dos cláusulas nos devolverán el mismo resultado.

```
JOIN

SELECT nombreCompleto,
       Email,
       Telefono,
       Descripcion
FROM Contactos C
INNER JOIN TiposDeTelefono T
ON C.IdTipoTel = T.IdTipoTel;
```

## INNER JOIN

Si existen datos del tipo **null** en la tabla contactos, donde ésta enlaza con la tabla **TiposDeTelefono**, los mismos no serán visualizados en la consulta resultante.

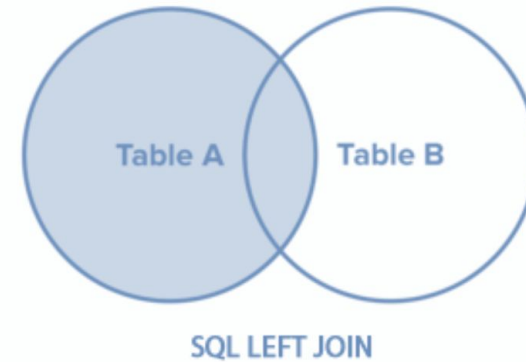
Result Grid				Filter Rows:	Search	Export:
	nombreCompleto	Email	Telefono	Descripcion		
▶	Cameron Howe	cameron@mutiny.com	11-4455-6699	Móvil		
	Gordon Clark	gordou@mutiny.com	11-4110-0909	Oficina		
	Joe McMillian	joe@mutiny.com	11-4844-1001	Satelital		
	Diane Gould	diane@mutiny.com	11-4844-1050	Skype		
	Malcolm Levitan	malc@mutiny.com	11-4110-5488	Oficina		
	Joanie Clark	joanie@mutiny.com	11-4810-1010	Personal		
	Haley Clark	hal@mutiny.com	11-4810-1010	Personal		

**LEFT JOIN**

# LEFT JOIN

Representación de **LEFT JOIN**:

Trae todos los registros de la tabla izquierda o de mayor peso, aplicando la relación de ambas tablas.



# LEFT JOIN

Representación de **LEFT JOIN**:

Trae todos los registros de la tabla izquierda o de mayor peso, aplicando la relación de ambas tablas.

```
JOIN

SELECT nombreCompleto,
        Email,
        Telefono,
        Descripcion
FROM Contactos C
LEFT JOIN TiposDeTelefono T
ON C.IdTipoTel = T.IdTipoTel;
```

## LEFT JOIN

Si existen datos del tipo **null** en la tabla contactos, donde ésta enlaza con la tabla **TiposDeTelefono**, estos serán incluídos igual en la consulta resultante, completando con **null** el valor del campo **descripcion**.

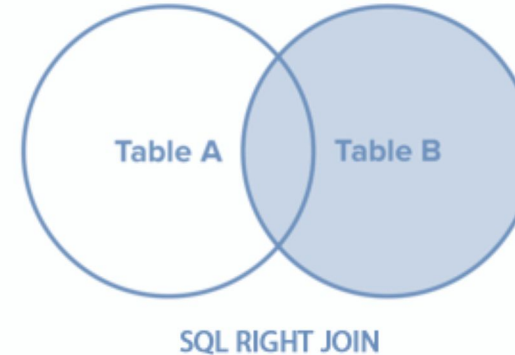
nombreCompleto	Email	Telefono	Descripcion
▶ Cameron Howe	cameron@mutiny.com	11-4455-6699	Móvil
Donna Clark	donna@mutiny.com	11-4455-0000	NULL
Gordon Clark	gordou@mutiny.com	11-4110-0909	Oficina
Joe McMillian	joe@mutiny.com	11-4844-1001	Satelital
John Bosworth	bos@mutiny.com	11-4844-1000	NULL
Yo-Yo Engberk	yoyo@mutiny.com	11-4844-1091	NULL
Diane Gould	diane@mutiny.com	11-4844-1050	Skype
Malcolm Levitan	malc@mutiny.com	11-4110-5488	Oficina
Joanie Clark	joanie@mutiny.com	11-4810-1010	Personal
Haley Clark	hal@mutiny.com	11-4810-1010	Personal

**RIGHT JOIN**

# RIGHT JOIN

Representación de **RIGHT JOIN**:

Trae todos los registros de la tabla derecha, aplicando la relación de ambas tablas, y haciendo el match con aquellos registros relacionados desde la tabla izquierda.





# RIGHT JOIN

Representación de **RIGHT JOIN**:

En este caso, se invierte la lógica respecto al uso de LEFT JOIN. Los registros que tendrán prioridad en la consulta resultante, serán los de la tabla **TiposDeTelefono**.

```
JOIN

SELECT nombreCompleto,
       Email,
       Telefono,
       Descripcion
FROM Contactos C
RIGHT JOIN TiposDeTelefono T
ON C.IdTipoTel = T.IdTipoTel;
```

## RIGHT JOIN

Todos aquellos datos provenientes de la tabla **Contactos** que no posean relación con los datos provenientes de la tabla **TiposDeTelefono** se completarán con valores **null**.

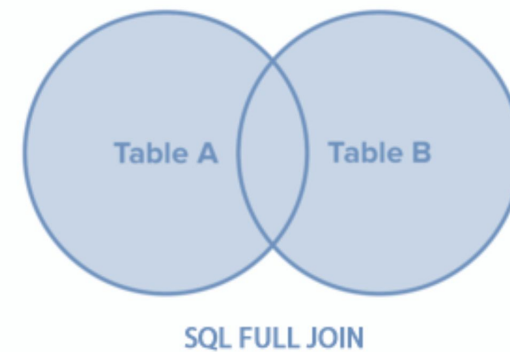
Result Grid				
Filter Rows:		Search	Export:	
nombreCompleto	Email	Telefono	Descripcion	
▶ Haley Clark	hal@mutiny.com	11-4810-1010	Personal	
Joanie Clark	joanie@mutiny.com	11-4810-1010	Personal	
Malcolm Levitan	malc@mutiny.com	11-4110-5488	Oficina	
Gordon Clark	gordou@mutiny.com	11-4110-0909	Oficina	
Cameron Howe	cameron@mutiny.com	11-4455-6699	Móvil	
Diane Gould	diane@mutiny.com	11-4844-1050	Skype	
NULL	NULL	NULL	WhatsApp	
Joe McMillian	joe@mutiny.com	11-4844-1001	Satelital	
NULL	NULL	NULL	Otro	
NULL	NULL	NULL	IP	
NULL	NULL	NULL	Semipúblico	
NULL	NULL	NULL	Recepción	

**OUTER JOIN || FULL OUTER JOIN**

# OUTER JOIN || FULL OUTER JOIN

Representación de **OUTER JOIN**:

Trae todos los registros de ambas tablas,  
coincidiendo aquellos donde hay relación, y  
completando con null aquellos donde no la haya.



# OUTER JOIN || FULL OUTER JOIN

En el motor MySQL esta consulta no surte efecto alguno, porque **OUTER JOIN** no está disponible en el mismo.

Solo forma parte del motor de **Microsoft SQL SERVER**. En este último, trae todos los registros de ambas tablas combinando las coincidencias, y completando ambos extremos con valores **null** donde no haya coincidencias.



**UNION ALL**

# UNION ALL

La cláusula **UNION ALL** se utiliza para combinar los resultados de dos o más consultas en una sola tabla resultante, devolviendo todas las filas, incluso si hay duplicados.

```
JOIN

SELECT NombreCompleto, Telefono
FROM Contactos
UNION ALL
SELECT IdTipoTel, Descripcion
FROM TiposDeTelefono;
```

# UNION ALL

En este caso, para utilizar UNION ALL, debemos solicitar la misma cantidad de campos de una tabla y de la otra, más aún si no coinciden en datos. El resultado sobre nuestro modelo de datos, no es muy efectivo que digamos.

Result Grid		
Filter Rows: Search		
Export:		
NombreCompleto	Telefono	
▶ Cameron Howe	11-4455-6699	
Donna Clark	11-4455-0000	
Gordon Clark	11-4110-0909	
Joe McMillian	11-4844-1001	
John Bosworth	11-4844-1000	
Yo-Yo Engberk	11-4844-1091	
Diane Gould	11-4844-1050	
Malcolm Levitan	11-4110-5488	
Joanie Clark	11-4810-1010	
Haley Clark	11-4810-1010	
1	Personal	
2	Oficina	
3	Móvil	
4	Skype	
5	WhatsApp	
6	Satelital	
7	Otro	
8	IP	
9	Semipúblico	
10	Recepción	



# Combinar JOIN y consultas SQL

## Combinar JOIN y consultas SQL

Trabajaremos sobre la base de datos **Northwind**, ejecutando consultas combinadas.

Trabajaremos con las tablas **Orders**, **OrderDetails**, **Products**, **Customers**, **Employees**, combinando los diferentes campos de estas para obtener consultas específicas.






## Combinar JOIN y consultas SQL

La tabla **OrderDetails** y **Products** tienen en común el campo **ProductID**.

En nuestra tabla **OrderDetails** tenemos información muy resumida. Sobre los productos que se vendieron en dicha orden, sólo accedemos a ver su código. Veamos cómo realizar una consulta que nos muestre el código de producto, su nombre (*aquí la relación*), la cantidad vendida, y el precio unitario.

De paso, generamos un campo calculado llamado **Subtotal**, para ver cuál es la ganancia por cada producto vendido.

Result Grid   Filter Rows: <input type="text" value="Search"/> Export: 						
	OrderID	ProductID	ProductName	Quantity	UnitPrice	Subtotal
▶	10255	2	Chang	20	15.2	304
	10255	16	Pavlova	35	13.9	486.5
	10255	36	Inlagd Sill	25	15.2	380
	10255	59	Raclette Courdavault	30	44	1320

## Combinar JOIN y consultas SQL

Veamos ahora cómo combinar las tablas **Orders** y **Customers**. En esta oportunidad, visualizaremos los nombres de los clientes de tres órdenes de compra específicas.



## Combinar JOIN y consultas SQL

Aprovechando que la tabla Orders no almacena información del cliente que solicitó la misma, más que su código de cliente (**CustomerID**), ejecutamos una consulta de selección que muestre la razón social (**companyName**) de los clientes para tres órdenes de compra específicas.

De paso, aprovechamos y sumamos una función escalar para eliminar la hora del campo Fecha de la orden (**Date**).

Result Grid   Filter Rows: <input type="text" value="Search"/> Export: 					
	OrderID	CustomerID	CompanyName	ContactName	Date
▶	10249	TOMSP	Toms Spezialit?ten	Karin Josephs	1996-07-05
	10252	SUPRD	Supr?mes d?lices	Pascale Cartrain	1996-07-09
	10254	CHOPS	Chop-suey Chinese	Yang Wang	1996-07-11

## Combinar JOIN y consultas SQL

Y, en este último ejemplo, incrementamos la complejidad de una consulta a múltiples tablas, trayendo información de las tablas **Orders**, **Customers** y **Employees**. Para ello, utilizaremos el uso de doble JOIN sobre la información pertinente.



# Combinar JOIN y consultas SQL

La Tabla Orders almacena información del encabezado de una orden de compra, donde podemos visualizar tanto la información del cliente que la solicitó, y también del empleado o ejecutivo de ventas que la generó.

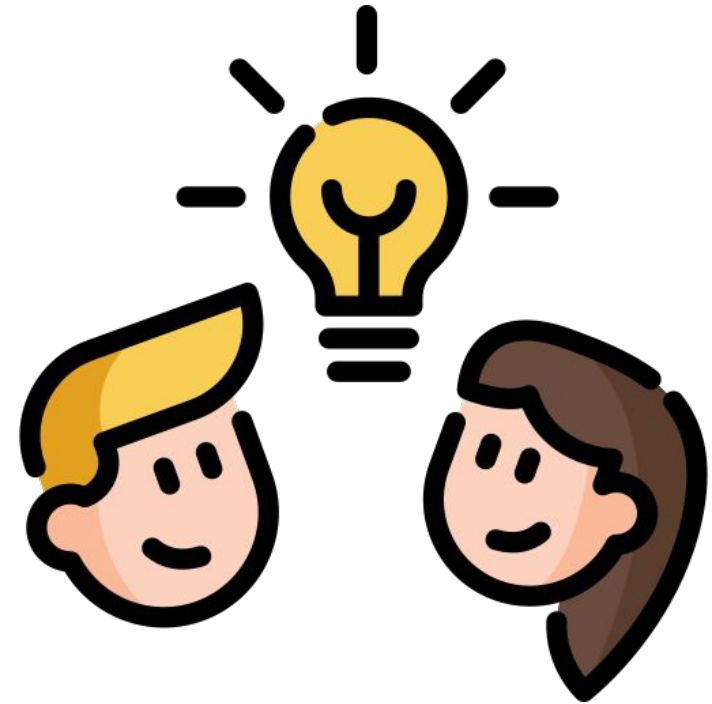
Aquí entra en juego el uso de **JOIN**, combinando las tablas **Employees** y **Customers**, con la tabla **Orders** en sí.

Result Grid   Filter Rows: <input type="text" value="Search"/> Export: 						
	OrderID	CustomerID	CompanyName	EmployeeID	LastName	FirstName
▶	10260	OTTIK	Ottilies K?seladen	4	Peacock	Margaret
	10267	FRANK	Frankenversand	4	Peacock	Margaret
	10269	WHITC	White Clover Markets	5	Buchanan	Steven

## Combinar JOIN y consultas SQL

Como podemos apreciar, el uso de **JOIN** nos facilita mucho el poder acceder a datos más concisos, algo que las tablas en general, de forma aislada no suelen tener mucho sentido de consistencia.

Además, podemos apreciar cómo sumar el uso de campos calculados, concatenados, e integrar también fácilmente cualquier función escalar útil para nuestros propósitos.





# Sección práctica

Es momento de poner en práctica el uso de tablas combinadas mediante JOIN.

Veamos a continuación tres ejercicios para resolver la visualización de datos provenientes de diferentes tablas.



# Prácticas

Necesitamos simplificar la visualización de datos de la tabla **Products**, **Customers**, **Categories** y **Employees**, a través de diferentes consultas de selección:

Realiza para ello, las siguientes consignas:

1. Ejecuta una consulta de selección para obtener los campos **ProductID**, **ProductName**, **Quantity** y **UnitPrice**, combinando la tabla **Products** y la tabla **OrderDetails**.
  - a. Deberás visualizar los datos de la orden número: 10255
2. Ejecuta una consulta de selección para visualizar el campo **CustomerName**, de la tabla **Customers**, y los campos **FirstName** y **LastName** de la tabla **Employees**.
  - a. Concatena **FirstName** y **LastName** como un único campo llamado **EjecutivoDeCuentas**
1. Ejecuta una consulta de selección para visualizar los datos **ProductID**, **ProductName** de la tabla **Products** y los campos **CompanyName** y **ContactName** de la tabla **Suppliers**.
  - b. Visualizar la información solo de los productos correspondientes a la categoría 7



# Muchas gracias.



Ministerio de Economía  
**Argentina**

Secretaría de  
Economía del Conocimiento

*primero  
la gente*