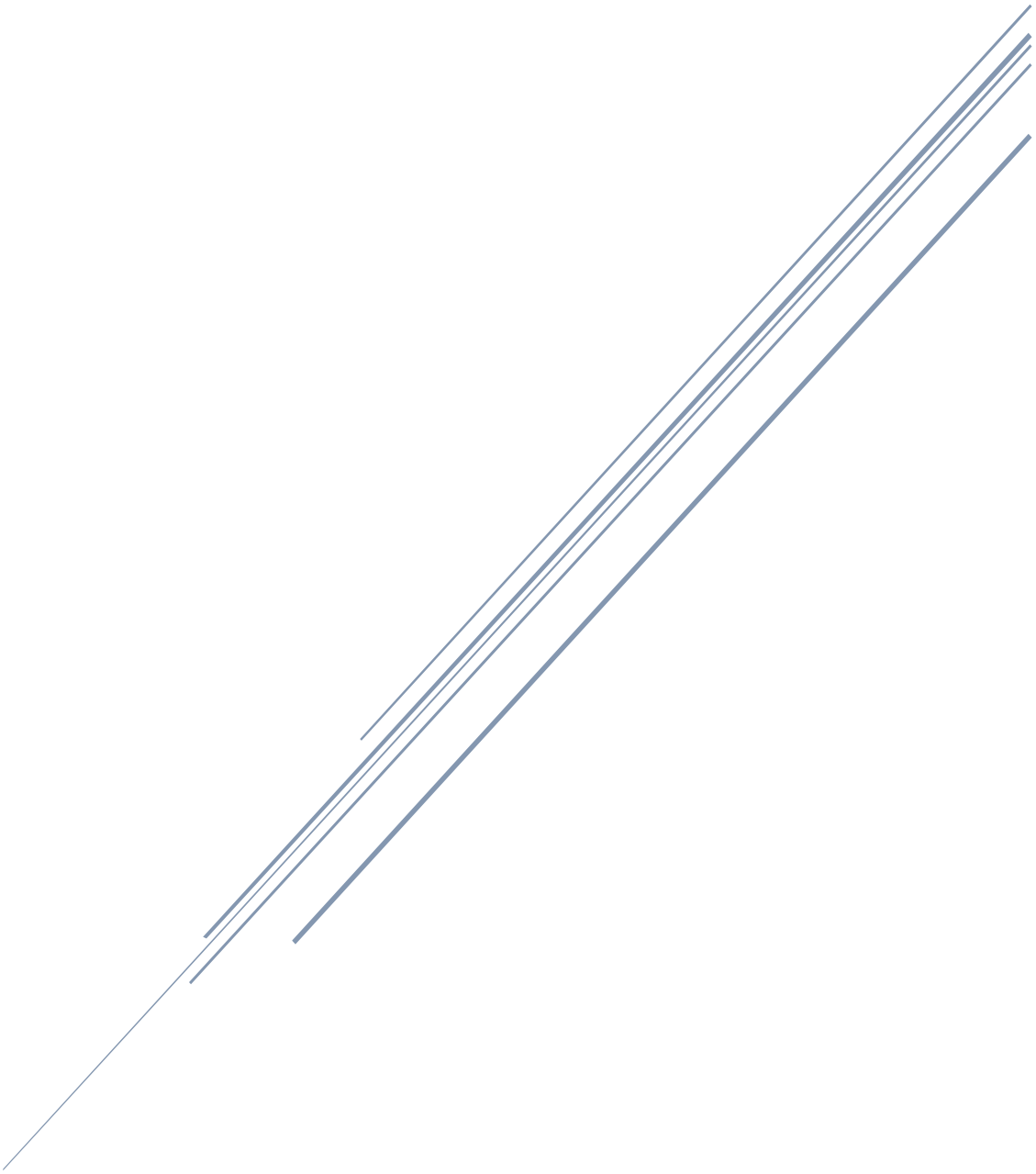
MANUAL BÁSICO SQL



Bases de Datos

Contenido

Qué es y para qué sirve el SQL

Las aplicaciones en red son cada día más numerosas y versátiles. En muchos casos, el esquema básico de operación es una serie de scripts que rigen el comportamiento de una base de datos.

Debido a la diversidad de lenguajes y de bases de datos existentes, la manera de comunicar entre unos y otras sería realmente complicada de no ser por la existencia de estándares que nos permiten realizar las operaciones básicas de una forma universal.

Es de eso de lo que trata el Structured Query Language que no es más que un lenguaje estándar de comunicación con bases de datos. Hablamos por tanto de un lenguaje normalizado que nos permite trabajar con cualquier tipo de lenguaje (ASP o PHP) en combinación con cualquier tipo de base de datos (MS Access, SQL Server, MySQL...).

El hecho de que sea estándar no quiere decir que sea idéntico para cada base de datos. En efecto, determinadas bases de datos implementan funciones específicas que no tienen necesariamente que funcionar en otras.

Aparte de esta universalidad, el SQL posee otras dos características muy apreciadas. Por una parte, presenta una potencia y versatilidad notables que contrasta, por otra, con su accesibilidad de aprendizaje.

Sentencia INSERT

Los registros pueden ser introducidos a partir de sentencias que emplean la instrucción Insert.

La sintaxis utilizada es la siguiente:

```
Insert Into nombre_tabla (nombre_campo1, nombre_campo2,...) Values
(valor_campo1, valor_campo2...)
```

Un ejemplo sencillo a partir de nuestra tabla modelo es la introducción de un nuevo cliente lo cual se haría con una instrucción de este tipo:

```
Insert Into clientes (nombre, apellidos, direccion, poblacion,
codigopostal, email, pedidos) Values ('Ovidio', 'Peláez', 'Percebe
n°13', 'Alava', '01004', 'ovidio@colegio.com', 33)
```

Como puede verse, los campos no numéricos o booleanos van delimitados por apostrofes: '. También resulta interesante ver que el código postal lo hemos guardado como un campo no numérico. Esto es debido a que los códigos

postales contienen ceros a la izquierda que se perderían si los definiésemos como números.

Enumerar los campos de la tabla en que vamos a insertar, solo es imprescindible en el caso de que vayamos a rellenar solo algunos de ellos. Si los datos que vamos a insertar cubren todos los campos, bastaría con hacer lo siguiente.

```
Insert Into clientes Values ('Ovidio', 'Estrellas', 'Percebe n°13',
'Alava', '01004', 'ovidio@colegio.com', 33)
```

Sentencia DELETE

Para borrar un registro nos servimos de la instrucción Delete. En este caso debemos especificar cual o cuales son los registros que queremos borrar. Es por ello necesario establecer una selección que se llevara a cabo mediante la cláusula Where.

```
Delete From nombre_tabla Where condiciones_de_selección
```

Si queremos por ejemplo, borrar todos los registros de los clientes que se llamen Perico lo haríamos del siguiente modo:

```
Delete From clientes Where nombre='Pepito'
```

Sentencia UPDATE

Update es la instrucción del lenguaje SQL que nos sirve para modificar los registros de una tabla. Como para el caso de Delete, necesitamos especificar por medio de Where cuáles son los registros en los que queremos hacer efectivas nuestras modificaciones. Además, obviamente, tendremos que especificar cuáles son los nuevos valores de los campos que deseamos actualizar.

La sintaxis es de este tipo:

```
Update nombre_tabla Set nombre_campo1 = valor_campo1, nombre_campo2 =
valor_campo2,... Where condiciones_de_selección
```

Un ejemplo aplicado:

```
Update clientes Set nombre='Ana' Where nombre='Marta'
```

Mediante esta sentencia cambiamos el nombre Marta por el de Ana en todos los registros cuyo nombre sea Marta.

Aquí también hay que ser cuidadoso de no olvidarse de usar Where, de lo contrario, modificaríamos todos los registros de nuestra tabla.

```
Update producto Set precio=990, descuento=25
```

Esa sentencia modificaría el campo precio y el campo descuento en todos los productos de la tabla producto. Si tenemos una tabla con miles de productos con esa sentencia se actualizarían todos, de modo que la totalidad de los registros tendrían el mismo precio y el mismo descuento.

Consultas básicas de selección

La selección total o parcial de una tabla se lleva a cabo mediante la instrucción Select. En dicha selección hay que especificar:

-Los campos que queremos seleccionar

-La tabla en la que hacemos la selección

En nuestra tabla modelo de clientes podríamos hacer por ejemplo una selección del nombre y dirección de los clientes con una instrucción de este tipo:

```
Select nombre, dirección From clientes
```

Si quisiésemos seleccionar todos los campos, es decir, **toda la tabla**, podríamos utilizar el comodín * del siguiente modo:

```
Select * From clientes
```

Resulta también muy útil el filtrar los registros mediante condiciones que vienen expresadas después de la **cláusula Where**. Si quisiésemos mostrar los clientes de una determinada ciudad usaríamos una expresión como esta:

```
Select * From clientes Where poblacion Like 'Madrid'
```

Además, podríamos **ordenar los resultados** en función de uno o varios de sus campos. Para este ultimo ejemplo los podríamos ordenar por nombre así:

```
Select * From clientes Where poblacion Like 'Madrid' Order By nombre
```

Teniendo en cuenta que puede haber más de un cliente con el mismo nombre, podríamos dar un segundo criterio que podría ser el apellido:

```
Select * From clientes Where poblacion Like 'Madrid' Order By nombre,
apellido
```

Si invirtiésemos el orden « nombre,apellido » por « apellido, nombre », el resultado sería distinto. Tendríamos los clientes ordenados por apellido y aquellos que tuviesen apellidos idénticos se subclasificarían por el nombre.

La cláusula WHERE

La cláusula WHERE puede usarse para determinar qué registros de las tablas enumeradas en la cláusula FROM aparecerán en los resultados de la instrucción SELECT. Después de escribir esta cláusula se deben especificar las condiciones

expuestas en los apartados anteriores. Si no se emplea esta cláusula, la consulta devolverá todas las filas de la tabla. WHERE es opcional, pero cuando aparece debe ir a continuación de FROM.

```
SELECT
    Apellidos, Salario
FROM
    Empleados
WHERE
    Salario = 21000
SELECT
   IdProducto, Existencias
FROM
   Productos
WHERE
    Existencias <= NuevoPedido
```

Lista de operadores

Estos operadores serán utilizados después de la cláusula Where y pueden ser **combinados mediante paréntesis** para optimizar nuestra selección.

Operadores matemáticos:	
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
<>	Distinto
=	Igual

Operadores lógicos
And
Or
Not

Otros operadores

Like	Selecciona los registros cuyo valor de campo se asemeje, no teniendo en cuenta mayúsculas y minúsculas.
In y Not In	Da un conjunto de valores para un campo para los cuales la condición de selección es (o no) valida
Is Null y Is Not Null	Selecciona aquellos registros donde el campo especificado esta (o no) vacío.
Between...And	Selecciona los registros comprendidos en un intervalo
Distinct	Selecciona los registros no coincidentes
Desc	Clasifica los registros por orden inverso

Comodines	
*	Sustituye a todos los campos
%	Sustituye a cualquier cosa o nada dentro de una cadena
_	Sustituye un solo carácter dentro de una cadena

Veamos a continuación aplicaciones prácticas de estos operadores.

En esta sentencia seleccionamos todos los clientes de Madrid cuyo nombre no es Pepe. Como puede verse, empleamos **Like** en lugar de **=** simplemente para evitar inconvenientes debido al empleo o no de mayúsculas.

```
Select * From clientes Where poblacion Like 'madrid' And nombre Not Like 'Pepe'
```

Si quisiéramos recoger en una selección a los clientes de nuestra tabla cuyo **apellido comienza por A y cuyo número de pedidos está comprendido entre 20 y 40**:

```
Select * From clientes Where apellidos like 'A%' And pedidos Between 20 And 40
```

El operador **In**, lo veremos más adelante, es muy práctico para consultas en varias tablas. Para casos en una sola tabla es empleado del siguiente modo:

```
Select * From clientes Where poblacion In ('Madrid','Barcelona','Valencia')
```

De esta forma **seleccionamos aquellos clientes que vivan en esas tres ciudades**.

Operador DISTINCT

Omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción SELECT se incluyan en la consulta deben ser únicos. Por ejemplo, varios empleados listados en la tabla Empleados pueden tener el mismo apellido. Si dos registros contienen López en el campo Apellido, la siguiente instrucción SQL devuelve un único registro:

```
SELECT DISTINCT
   Apellido
FROM
   Empleados
```

En otras palabras, el predicado DISTINCT devuelve aquellos registros cuyos campos indicados en la cláusula SELECT posean un contenido diferente. El resultado de una consulta que utiliza DISTINCT no es actualizable y no refleja los cambios subsiguientes realizados por otros usuarios.

Operador BETWEEN

Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo emplearemos el operador Between cuya sintaxis es:

```
   campo [Not] Between valor1 And valor2 (la condición Not es opcional)
```

En este caso la consulta devolvería los registros que contengan en "campo" un valor incluido en el intervalo valor1, valor2 (ambos inclusive). Si anteponemos la condición Not devolverá aquellos valores no incluidos en el intervalo.

```
SELECT *
FROM
    Pedidos
WHERE
    NumPedido Between 200 And 299
```

El Operador LIKE

Se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis es:

```
   expresión Like modelo
```

En donde expresión es una cadena modelo o campo contra el que se compara expresión. Se puede utilizar el operador Like para encontrar valores en los campos que coincidan con el modelo especificado. Por modelo puede especificar un valor completo (Ana María), o se puede utilizar una cadena de caracteres

comodín como los reconocidos por el sistema operativo para encontrar un rango de valores (Like ‘An%’).

El operador Like se puede utilizar en una expresión para comparar un valor de un campo con una expresión de cadena. Por ejemplo, si introduces Like ‘C%’ en una consulta SQL, la consulta devuelve todos los valores de campo que comiencen por la letra C. En una consulta con parámetros, puede hacer que el usuario escriba el modelo que se va a utilizar.

Consultas Multitabla

Una base de datos puede ser considerada como un conjunto de tablas. Estas tablas en muchos casos están relacionadas entre ellas y se complementan unas con otras.

Refiriéndonos a nuestro clásico ejemplo de una base de datos para una aplicación de e-comercio, la tabla clientes de la que hemos estado hablando puede estar perfectamente coordinada con una tabla donde almacenamos los pedidos realizados por cada cliente. Esta tabla de pedidos puede a su vez estar conectada con una tabla donde almacenamos los datos correspondientes a cada artículo del inventario.

De este modo podríamos fácilmente obtener informaciones contenidas en esas tres tablas como puede ser, la designación del artículo más popular en una determinada región. Donde la designación del artículo sería obtenida de la tabla de artículos, la popularidad (cantidad de veces que ese artículo ha sido vendido) vendría de la tabla de pedidos y la región estaría comprendida obviamente en la tabla clientes.

Este tipo de organización basada en múltiples tablas conectadas nos permite trabajar con tablas mucho más manejables a la vez que nos evita copiar el mismo campo en varios sitios ya que podemos acceder a él a partir de una simple llamada a la tabla que lo contiene.

Aquí veremos cómo, sirviéndonos de lo aprendido hasta ahora, podemos realizar fácilmente selecciones sobre varias tablas. Definamos antes de nada las diferentes tablas y campos que vamos a utilizar en nuestros ejemplos:

Tabla de clientes	
Nombre campo	**Tipo campo**

id_cliente	int
nombre	Varchar(30)
apellidos	Varchar(30)
direccion	Varchar(30)
poblacion	Varchar(30)
codigopostal	Varchar(30)
telefono	int
email	Varchar(30)

Tabla de pedidos	
Nombre campo	**Tipo campo**
id_pedido	int
id_cliente	int
id_articulo	int
fecha	Date
cantidad	int

Tabla de artículos	
Nombre campo	**Tipo campo**
id_articulo	int
titulo	Varchar(30)
autor	Varchar(30)
editorial	Varchar(30)
precio	double

Estas tablas pueden ser utilizadas simultáneamente para extraer informaciones de todo tipo. Supongamos que queremos enviar un mail a todos aquellos que hayan realizado un pedido ese mismo día. Podríamos escribir algo así:

```
Select clientes.apellidos, clientes.email From clientes,pedidos Where pedidos.fecha like '2023-12-18' And pedidos.id_cliente= clientes.id_cliente
```

Como puede verse esta vez, después de la cláusula From, introducimos el nombre de las dos tablas de donde sacamos las informaciones. Además, el nombre de cada campo va precedido de la tabla de origen, separados ambos por un punto. **En los campos que poseen un nombre que solo aparece en una de las tablas, no es necesario especificar su origen aunque a la hora de leer la sentencia puede resultar más claro el precisarlo**. En este caso el campo fecha podría haber sido designado como "fecha" en lugar de "pedidos.fecha".

Veamos otro ejemplo más para consolidar estos nuevos conceptos. Esta vez queremos ver el título del libro correspondiente a cada uno de los pedidos realizados:

```
Select pedidos.id_pedido, articulos.titulo From pedidos, articulos Where pedidos.id_articulo=articulos.id_articulo
```

En realidad la filosofía continua siendo la misma que para la consulta de una única tabla.

Funciones sumarias y agrupadas

Además de los criterios hasta ahora explicados para realizar las consultas en tablas, SQL permite también aplicar un conjunto de funciones predefinidas. Estas funciones, aunque básicas, pueden ayudarnos en algunos momentos a expresar nuestra selección de una manera más simple sin tener que recurrir a operaciones adicionales por parte del script que estemos ejecutando.

Algunas de estas funciones son representadas en la tabla siguiente :

Función	Descripción
Sum(campo)	Calcula la suma de los registros del campo especificado
Avg(Campo)	Calcula la media de los registros del campo especificado
Count(*)	Nos proporciona el valor del numero de registros que han sido seleccionados
Max(Campo)	Nos indica cual es el valor máximo del campo
Min(Campo)	Nos indica cual es el valor mínimo del campo

Dado que el campo de la función no existe en la base de datos, sino que lo estamos generando virtualmente, esto puede crear inconvenientes cuando estamos trabajando con nuestros scripts a la hora de tratar su valor y su nombre de campo. Es por ello que el valor de la **función ha de ser recuperada a partir de un alias** que nosotros especificaremos en la sentencia SQL a partir de la instrucción **AS**. La cosa podría quedar así:

```
Select Sum(total) As suma_pedidos From pedidos
```

A partir de esta sentencia calculamos la suma de los valores de todos los pedidos realizados y almacenamos ese valor en un campo virtual llamado suma_pedidos que podrá ser utilizado como cualquier otro campo por nuestras paginas dinámicas.

Por supuesto, todo lo visto hasta ahora puede ser aplicado en este tipo de funciones de modo que, por ejemplo, podemos establecer condiciones con la cláusula Where construyendo sentencias como esta:

```
Select Sum(cantidad) as suma_articulos From pedidos Where id_articulo=6
```

Esto nos proporcionaría la cantidad de **ejemplares de un determinado libro que han sido vendidos**.

Otra propiedad interesante de estas funciones es que **permiten realizar operaciones con varios campos dentro de un mismo paréntesis**:

```
Select Avg(total/cantidad) From pedidos
```

Esta sentencia da como resultado el **precio medio al que se están vendiendo los libros**. Este resultado no tiene por qué coincidir con el del **precio medio de los libros presentes en el inventario**, ya que, puede ser que la gente tenga tendencia a comprar los libros caros o los baratos:

```
Select Avg(precio) as precio_venta From articulos
```

Cláusula GROUP BY

Una cláusula interesante en el uso de funciones es Group By. Esta cláusula nos permite agrupar registros a los cuales vamos a aplicar una función. Podemos por ejemplo calcular el **dinero gastado por cada cliente**:

```
Select id_cliente, Sum(total) as suma_pedidos From pedidos Group By id_cliente
```

O saber el **numero de pedidos que han realizado**:

```
Select id_cliente, Count(*) as numero_pedidos From pedidos Group By id_cliente
```

Las posibilidades como vemos son numerosas y pueden resultar prácticas. Todo queda ahora a disposición de nuestras ocurrencias e imaginación.

Cláusula HAVING

Las funciones sumarias, resultan de gran utilidad en las consultas sql pero presentan una dificultad añadida que no es otra que, el hecho de que no se pueden utilizar para establecer una condición en la cláusula where.

Se hace por ello necesario emplear una nueva cláusula que, en muchas ocasiones, trabaja en conjunto con la cláusula group by. Veamos un ejemplo

```
Select count(ciudad), region
From oficinas
Group by region
Having count(ciudad)>2
```

La anterior consulta nos ofrecerá como resultado el número de ciudades y la región a la que pertenecen, para todas las regiones en que el número de ciudades sea mayor de 2.

Campos calculados

En determinadas ocasiones nos puede interesar incluir una columna con una fórmula, por ejemplo, supongamos que tenemos una tabla de empleados y deseamos recuperar las tarifas semanales de los electricistas (tarifas por hora por las 40 horas de la semana), podríamos realizar la siguiente consulta:

```
SELECT
   Empleados.Nombre, (Empleados.TarifaHora * 40) as tarifa_semanal
FROM
   Empleados
WHERE
   Empleados.Cargo = 'Electricista'
```

Ordenar los registros

Adicionalmente se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula ORDER BY Lista de Campos. En donde Lista de campos representa los campos a ordenar. Ejemplo:

```
SELECT
   CodigoPostal, Nombre, Telefono
FROM
```

```
   Clientes
ORDER BY
   Nombre
```

Esta consulta devuelve los campos CodigoPostal, Nombre, Telefono de la tabla Clientes ordenados por el campo Nombre.

Se pueden ordenar los registros por más de un campo, como por ejemplo:

```
SELECT
   CodigoPostal, Nombre, Telefono
FROM
   Clientes
ORDER BY
   CodigoPostal, Nombre
```

Incluso se puede especificar el orden de los registros: ascendente mediante la cláusula (ASC - se toma este valor por defecto) ó descendente (DESC)

```
SELECT
   CodigoPostal, Nombre, Telefono
FROM
   Clientes
ORDER BY
   CodigoPostal DESC, Nombre ASC
```

Subconsultas

Una subconsulta es una consulta anidada dentro de otra consulta.

Debemos tener en cuenta que no existe una única solución para resolver una consulta en SQL. Aquí veremos cómo resolver con subconsultas, algunas consultas que, de otra manera, no sabíamos resolver hasta la fecha.

Tipos de subconsultas

El estándar SQL define tres tipos de subconsultas:

- **Subconsultas de fila**. Son aquellas que devuelven más de una columna pero una única fila.
- **Subconsultas de tabla**. Son aquellas que devuelve una o varias columnas y cero o varias filas.
- **Subconsultas escalares**. Son aquellas que devuelven una columna y una fila.

Subconsultas en la cláusula WHERE

Supongamos que, en una tabla de productos, queremos conocer el nombre del producto que tiene el mayor precio. En este caso podríamos realizar una primera consulta para buscar cuál es el valor del precio máximo y otra segunda consulta para buscar el nombre del producto cuyo precio coincide con el valor del precio máximo. Si unimos ambas cosas, la consulta obtenida sería la siguiente:

```
SELECT nombre
FROM producto
WHERE precio = (SELECT MAX(precio) FROM producto)
```

En este caso sólo hay un nivel de anidamiento entre consultas pero pueden existir varios niveles de anidamiento.

Subconsultas en la cláusula HAVING

Ejemplo: Devuelve un listado con todos los nombres de los fabricantes que tienen el mismo número de productos que el fabricante Xiaomi.

```
SELECT fabricante.nombre, COUNT(producto.codigo)
FROM fabricante INNER JOIN producto
ON fabricante.codigo = producto.codigo_fabricante
GROUP BY fabricante.codigo
HAVING COUNT(producto.codigo) >= (
    SELECT COUNT(producto.codigo)
    FROM fabricante INNER JOIN producto
    ON fabricante.codigo = producto.codigo_fabricante
    WHERE fabricante.nombre = 'Xiaomi');
```

Subconsultas en la cláusula FROM

Ejemplo: Devuelve una listado de todos los productos que tienen un precio mayor o igual al precio medio de todos los productos de su mismo fabricante.

```
SELECT *
FROM producto INNER JOIN (
    SELECT codigo_fabricante, AVG(precio) AS media
    FROM producto
    GROUP BY codigo_fabricante) AS t
ON producto.codigo_fabricante = t.codigo_fabricante
WHERE producto.precio >= t.media;
```

Operadores que podemos usar en las subconsultas

Los operadores que podemos usar en las subconsultas son los siguientes:

- Operadores básicos de comparación (>,>=, <, <=, !=, <>, =).
- Predicados ALL y ANY.
- Predicado IN y NOT IN.
- Predicado EXISTS y NOT EXISTS.

Operadores básicos de comparación en subconsultas

Los operadores básicos de comparación (>,>=, <, <=, !=, <>, =) se pueden usar cuando queremos comparar una expresión con el valor que devuelve una subconsulta.

Los operadores básicos de comparación los vamos a utilizar para realizar comparaciones con **subconsultas que devuelven un único valor, es decir, una columna y una fila**.

Ejemplo: Devuelve todos los productos de la base de datos que tienen un precio mayor o igual al producto más caro del fabricante Xiaomi.

```
SELECT *
FROM producto
WHERE precio >= (SELECT MAX(precio)
    FROM fabricante INNER JOIN producto
    ON fabricante.codigo = producto.codigo_fabricante
    WHERE fabricante.nombre = 'Xiaomi');
```

La consulta anterior también se puede escribir con subconsultas sin hacer uso de INNER JOIN.

```
SELECT *
FROM producto
WHERE precio >= (
  SELECT MAX(precio)
  FROM producto
  WHERE codigo_fabricante = (
    SELECT codigo
    FROM fabricante
    WHERE nombre = 'Xiaomi'));
```

Subconsultas con ALL y ANY

ALL y ANY se utilizan con los operadores de comparación (>,>=, <, <=, !=, <>, =) y nos permiten comparar una expresión con el conjunto de valores que devuelve una subconsulta.

ALL y ANY los vamos a utilizar para realizar comparaciones con **subconsultas que pueden devolver varios valores, es decir, una columna y varias filas**.

Ejemplo: Podemos escribir la consulta que devuelve todos los productos de la base de datos que tienen un precio mayor o igual al producto más caro del fabricante Xiaomi, haciendo uso de **ALL**. Por lo tanto, estas dos consultas darían el mismo resultado.

```
SELECT *
FROM fabricante INNER JOIN producto
ON fabricante.codigo = producto.codigo_fabricante
WHERE precio >= (SELECT MAX(precio)
    FROM fabricante INNER JOIN producto
    ON fabricante.codigo = producto.codigo_fabricante
    WHERE fabricante.nombre = 'Xiaomi');

SELECT *
FROM fabricante INNER JOIN producto
ON fabricante.codigo = producto.codigo_fabricante
WHERE precio >= ALL (SELECT precio
    FROM fabricante INNER JOIN producto
    ON fabricante.codigo = producto.codigo_fabricante
    WHERE fabricante.nombre = 'Xiaomi');
```

Ejemplo 2: Podemos escribir la consulta que devuelve todos los productos de la base de datos que tienen un precio mayor que **alguno de los productos** del fabricante Xiaomi, haciendo uso de **ANY**.

```
SELECT *
FROM fabricante INNER JOIN producto
ON fabricante.codigo = producto.codigo_fabricante
WHERE precio > ANY (SELECT MAX(precio)
    FROM fabricante INNER JOIN producto
    ON fabricante.codigo = producto.codigo_fabricante
```

```
    WHERE fabricante.nombre = 'Xiaomi');
```

Subconsultas con IN y NOT IN

IN y NOT IN nos permiten comprobar si un valor está o no incluido en un conjunto de valores, que puede ser el conjunto de valores que devuelve una subconsulta. IN y NOT IN los vamos a utilizar para realizar comparaciones con **subconsultas que pueden devolver varios valores, es decir, una columna y varias filas**.

Ejemplo: Devuelve un listado de los clientes que no han realizado ningún pedido.

```
SELECT *
FROM cliente
WHERE id NOT IN (SELECT id_cliente FROM pedido);
```

Cuando estamos trabajando con subconsultas, IN y = ANY realizan la misma función. Por lo tanto, las siguientes consultas devolverían el mismo resultado:

```
SELECT *
FROM cliente
WHERE id = ANY (SELECT id_cliente FROM pedido);
```

```
SELECT *
FROM cliente
WHERE id IN (SELECT id_cliente FROM pedido);
```

Ocurre lo mismo con NOT IN (para pedir los valores que no estén en una enumeración dada) y <> ALL (igual que lo anterior, puesto que estamos pidiendo los valores que sean distintos de todos los listados a continuación). Por lo tanto, las siguientes consultas devolverían el mismo resultado:

```
SELECT *
FROM cliente
WHERE id <> ALL (SELECT id_cliente FROM pedido);
```

```
SELECT *
FROM cliente
WHERE id NOT IN (SELECT id_cliente FROM pedido);
```

Importante:

Ten en cuenta que cuando hay un valor NULL en el resultado de la consulta interna, la consulta externa no devuelve ningún valor.

Ejemplo: Devuelve un listado con el nombre de los departamentos que no tienen empleados asociados.

```
SELECT nombre
FROM departamento
WHERE codigo NOT IN (
  SELECT codigo_departamento
  FROM empleado);
```

Imaginemos que hay algunos empleados que no tienen código de departamento. En ese caso, la consulta interna SELECT codigo_departamento FROM empleado, devuelve algunas filas con valores NULL y por lo tanto la consulta externa no devuelve ningún valor.

La forma de solucionarlo sería quitando los valores NULL de la consulta interna:

```
SELECT nombre
FROM departamento
WHERE codigo NOT IN (
  SELECT codigo_departamento
  FROM empleado
  WHERE codigo_departamento IS NOT NULL);
```

Condiciones de JOIN

una instrucción de “SQL JOIN” en un comando que combina las columnas entre dos o más tablas en una base de datos relacional y retorna a un conjunto de datos.

Para ver los efectos de los distintos tipos de join, nos basaremos en las siguientes tablas:

Por un lado, vamos a tener una tabla Empleados (que almacenará una lista de empleados y el id del departamento al que pertenecen):

Nombre	DepartamentoId
Rafferty	31
Jones	33
Heisenberg	33

Nombre	DepartamentoId
Robinson	34
Smith	34
Williams	NULL

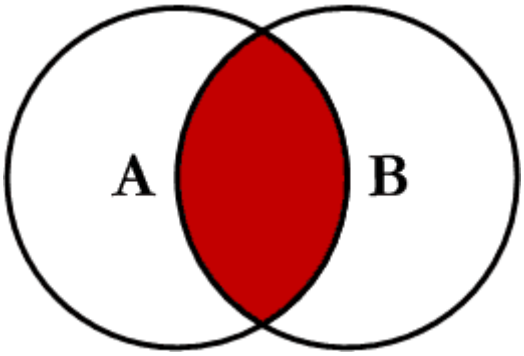
Y por otro lado, una tabla Departamentos (con la lista de departamentos que existen en la empresa).

Id	Nombre
31	Sales
33	Engineering
34	Clerical
35	Marketing

Cláusula INNER JOIN

Lo más usual, lo primero que se suele aprender, es el uso de INNER JOIN, o generalmente abreviado como JOIN.

Esta cláusula busca coincidencias entre 2 tablas, en función de una columna o campo que tienen en común. De tal modo que **sólo la intersección se mostrará en los resultados**.



Por ejemplo, si queremos listar a los empleados e indicar el nombre del departamento al que pertenecen, podemos hacer lo siguiente:

```
SELECT *
FROM Empleados
JOIN Departamentos
ON Empleados.DepartamentoId = Departamentos.Id
```

Con esto, nuestro resultado será una tabla que combine los campos de las dos mencionadas en la consulta anterior (Empleados y Departamentos):

Nombre	DepartmentoId	Id	Nombre
Rafferty	31	31	Sales
Jones	33	33	Engineering
Heisenberg	33	33	Engineering
Robinson	34	34	Clerical
Smith	34	34	Clerical

Y a partir de aquí podemos notar lo siguiente:

- El empleado "Williams" no aparece en los resultados, ya que no pertenece a ningún departamento existente.
- El departamento "Marketing" tampoco aparece, ya que ningún empleado pertenece a dicho departamento.

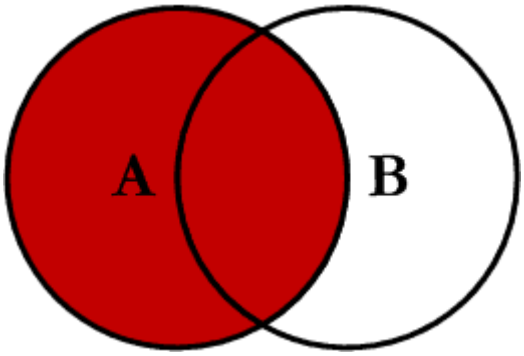
¿Por qué ocurre esto? Porque JOIN muestra como resultado la intersección de ambas tablas.

También hay que tener en cuenta que, en los resultados vemos 4 columnas. Las 2 primeras se corresponden con la tabla Empleados y las últimas con Departamentos.

Cláusula LEFT JOIN

A diferencia de un INNER JOIN, donde se busca una intersección respetada por ambas tablas, con LEFT JOIN damos prioridad a la tabla de la izquierda, y buscamos en la tabla derecha.

Si no existe ninguna coincidencia para alguna de las filas de la tabla de la izquierda, de igual forma **todos los resultados de la primera tabla se muestran**.



He aquí una consulta de ejemplo:

```
SELECT
  Empleados.Nombre as 'Empleado',
  Departamentos.Nombre as 'Departamento'
FROM Empleados
LEFT JOIN Departamentos
ON Empleados.DepartamentoId = Departamentos.Id
```

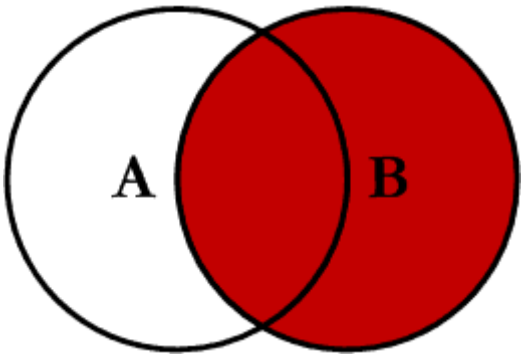
- La tabla Empleados es la primera tabla en aparecer en la consulta (en el FROM), por lo tanto ésta es la tabla LEFT (izquierda), y todas sus filas se mostrarán en los resultados.
- La tabla Departamentos es la tabla de la derecha (aparece luego del LEFT JOIN). Por lo tanto, si se encuentran coincidencias, se mostrarán los valores correspondientes, pero sino, aparecerá NULL en los resultados.

Empleado	Departamento
Rafferty	Sales

Empleado	Departamento
Jones	Engineering
Heisenberg	Engineering
Robinson	Clerical
Smith	Clerical
Williams	NULL

Cláusula RIGHT JOIN

En el caso de RIGHT JOIN la situación es muy similar, pero aquí se da prioridad a la tabla de la derecha.



De tal modo que si usamos la siguiente consulta, estaremos **mostrando todas las filas de la tabla de la derecha**:

```
SELECT
  Empleados.Nombre as 'Empleado',
  Departamentos.Nombre as 'Departamento'
FROM Empleados
RIGHT JOIN Departamentos
ON Empleados.DepartamentoId = Departamentos.Id
```

La tabla de la izquierda es Empleados, mientras que Departamentos es la tabla de la derecha.

La tabla asociada al FROM será siempre la tabla LEFT, y la tabla que viene después del JOIN será la tabla RIGHT.

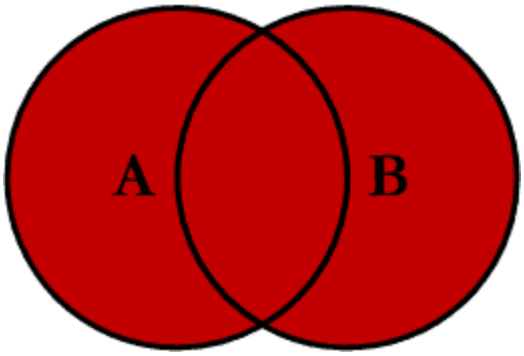
Entonces el resultado mostrará todos los departamentos al menos 1 vez.

Y si no hay ningún empleado trabajando en un departamento determinado, se mostrará NULL. Pero el departamento aparecerá de igual forma.

Empleado	Departamento
Rafferty	Sales
Jones	Engineering
Heisenberg	Engineering
Robinson	Clerical
Smith	Clerical
NULL	Marketing

Cláusula FULL JOIN

Mientras que LEFT JOIN muestra todas las filas de la tabla izquierda, y RIGHT JOIN muestra todas las correspondientes a la tabla derecha, FULL OUTER JOIN (o simplemente FULL JOIN) se encarga de mostrar todas las filas de ambas tablas, sin importar que no existan coincidencias (usará NULL como un valor por defecto para dichos casos).



Para nuestro ejemplo, ocurre lo siguiente:

```
SELECT
  Empleados.Nombre as 'Empleado',
```

```
  Departamentos.Nombre as 'Departamento'
FROM Empleados
FULL JOIN Departamentos
ON Empleados.DepartamentoId = Departamentos.Id
```

Se muestra el empleado "Williams" a pesar de que no está asignado a ningún departamento, y se muestra el departamento de "Marketing" a pesar que aún nadie está trabajando allí:

Empleado	Departamento
Rafferty	Sales
Jones	Engineering
Heisenberg	Engineering
Robinson	Clerical
Smith	Clerical
Williams	NULL
NULL	Marketing

Consultas con UNION, INTERSECT y EXCEPT

Los operadores de conjunto UNION, INTERSECT y EXCEPT se utilizan para comparar y fusionar los resultados de dos expresiones de consulta diferentes. Por ejemplo, si deseas saber qué usuarios de un sitio web son compradores y vendedores, pero los nombres de usuario están almacenados en diferentes columnas o tablas, puede buscar con INTERSECT, la intersección de estos dos tipos de usuarios. Si deseas saber qué usuarios de un sitio web son compradores, pero no vendedores, puedes usar el operador EXCEPT para buscar la diferencia entre las dos listas de usuarios. Si deseas crear una lista de todos los usuarios, independientemente de la función, puedes usar el operador UNION.

Puedes crear consultas que contengan una cantidad ilimitada de expresiones de consulta y vincularlas con operadores UNION, INTERSECT y EXCEPT en cualquier combinación. Por ejemplo, la siguiente estructura de consulta es válida, suponiendo que las tablas Tabla1, Tabla2 y Tabla3 contienen conjuntos de columnas compatibles:

```
select * from tabla1
union
select * from tabla2
except
select * from tabla3
order by campo1;
```

UNION

Operación de conjunto que devuelve filas de dos expresiones de consulta, independientemente de si las filas provienen de una o ambas expresiones.

INTERSECT

Operación de conjunto que devuelve filas que provienen de dos expresiones de consulta. Las filas que no se devuelven en las dos expresiones se descartan.

EXCEPT

Operación de conjunto que devuelve filas que provienen de una de las dos expresiones de consulta. Para calificar para el resultado, las filas deben existir en la primera tabla de resultados, pero no en la segunda. MINUS y EXCEPT son sinónimos exactos.

UNION ALL

La palabra clave ALL conserva cualquier fila duplicada que UNION produce. El comportamiento predeterminado cuando no se usa la palabra clave ALL es descartar todos estos duplicados. No se admiten las expresiones INTERSECT ALL o EXCEPT ALL.

Orden de evaluación para los operadores de conjunto

Los operadores de conjunto UNION y EXCEPT se asocian por la izquierda. Si no se especifican paréntesis para establecer el orden de prioridad, los

operadores se evalúan de izquierda a derecha. Por ejemplo, en la siguiente consulta, UNION de Tabla1 y Tabla2 se evalúa primero, luego se realiza la operación EXCEPT en el resultado de UNION:

```
select * from tabla1
union
select * from tabla2
except
select * from tabla3
order by campo1;
```

El operador INTERSECT prevalece sobre los operadores UNION y EXCEPT cuando se utiliza una combinación de operadores en la misma consulta. Por ejemplo, la siguiente consulta evalúa la intersección de Tabla2 y Tabla3, y luego unirá el resultado con Tabla1:

```
select * from tabla1
union
select * from tabla2
intersect
select * from tabla3
order by campo1;
```

Al agregar paréntesis, puede aplicar un orden diferente de evaluación. En el siguiente caso, el resultado de la unión de Tabla1 y Tabla2 es objeto de la insersección con Tabla3, y la consulta probablemente produzca un resultado diferente.

```
(select * from tabla1
union
select * from tabla2)
intersect
(select * from tabla3)
order by campo1;
```

SENTENCIA SELECT….INTO

La función principal de la sentencia SELECT….INTO es Copia datos de una tabla existente en una nueva tabla. En muchos casos la finalidad es realizar una copia temporal de los datos que había en la tabla original, en previsión de que alguno de ellos pudiese resultar accidentalmente modificado o eliminado La sentencia, rea automáticamente la estructura de la nueva tabla, sin necesidad

de definirla antes. Pero cuidado, porque no copia restricciones como claves primarias, índices o claves foráneas**.**

Copiar tabla completa

La sintaxis, en caso de que queramos copiar la tabla al completo, es esta:

```
SELECT * INTO NuevaTabla FROM TablaOriginal;
```

Esto crea una nueva tabla llamada NuevaTabla y copia todos los datos de TablaOriginal en ella. Otro aspecto a tener en cuenta es que NuevaTabla no debe existir previamente en la base de datos. De lo contrario la operación no se realizará porque SELECT…INTO no permite copiar información en una tabla que ya exista.

Supongamos que tenemos una tabla llamada PACIENTES y queremos hacer una copia con el nombre PACIENTES_BACKUP.

```
SELECT * INTO PACIENTES_BACKUP FROM PACIENTES;
```

Esto creará una nueva tabla llamada PACIENTES_BACKUP con la misma estructura y datos de PACIENTES.

Copiar solo algunos campos y su contenido

Hay otra manera de emplear SELECT….INTO, y es para copiar solo algunas columnas.

Si solo queremos copiar algunas columnas, podemos especificarlas:

```
SELECT ID_PACIENTE, NOMBRE, APELLIDO
INTO PACIENTES_COPIA
FROM PACIENTES;
```

Esto creará PACIENTES_COPIA, pero solo con las columnas ID_PACIENTE, NOMBRE y APELLIDO.

Copiar solo algunos datos

Por último, podemos decidir copiar solo algunos datos de la tabla original. Para ello, estableceremos criterios en el where de nuestra consulta.

También podemos filtrar los datos que queremos copiar. Por ejemplo, copiar solo los pacientes con seguro médico:

```
SELECT * INTO PACIENTES_ASEGURADOS
FROM PACIENTES
WHERE SEGURO_MEDICO IS NOT NULL;
```

Esto creará PACIENTES_ASEGURADOS y solo incluirá a los pacientes que tienen seguro médico.