



Bases de datos

Consultas Simples y Múltiples

Consultas Simples

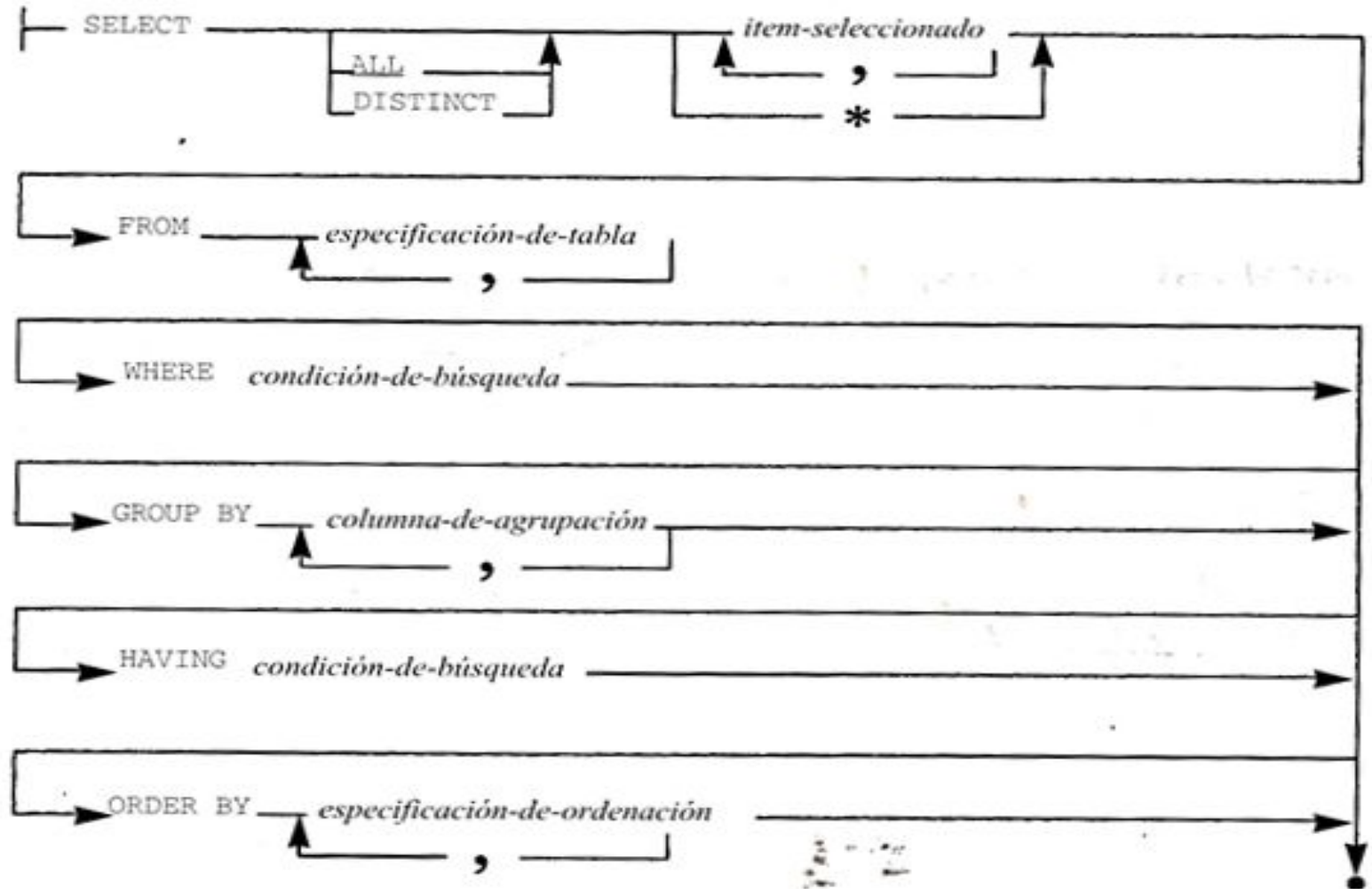
En muchos sentidos, las consultas son la base del lenguaje SQL. La sentencia **SELECT**, que se utiliza para expresar consultas, es la más potente y compleja de las sentencias SQL. A pesar de las muchas opciones permitidas por la sentencia **SELECT**, es posible comenzar de forma sencilla y luego pasar a elaborar consultas más complejas.

Sentencia SELECT

La sentencia **SELECT** recupera datos de una base de datos y los devuelve en forma de resultados de la consulta.

En la siguiente imagen puedes observar el formato completo de la sentencia SELECT, que consta de 6 cláusulas: **SELECT, FROM, WHERE, GROUP BY, HAVING Y ORDER BY.**

Sentencia SELECT





Resultados de Consultas

El resultado de una consulta es siempre una tabla de datos, semejante a las tablas de la base de datos.

Las consultas más sencillas solicitan columnas de datos de una única tabla en la base de datos.

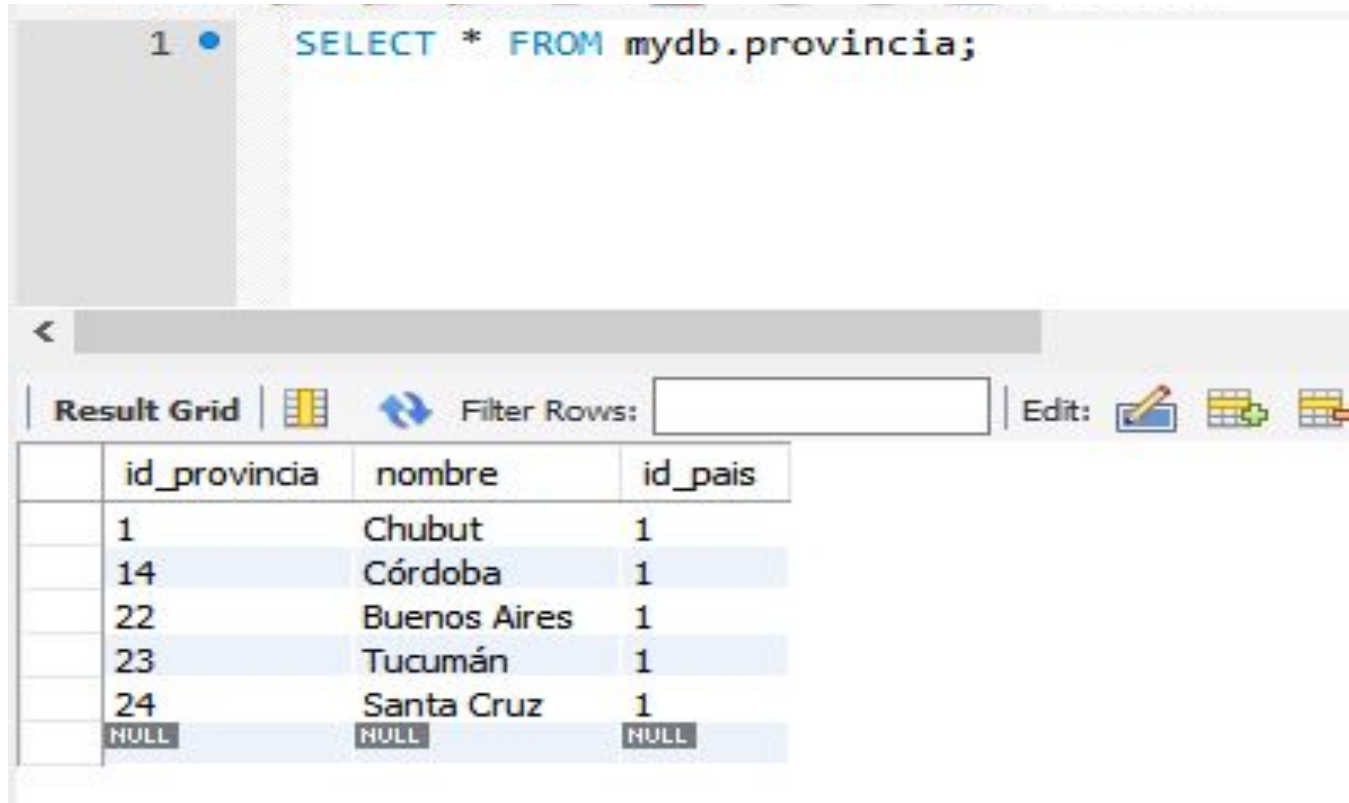
```
1 SELECT id_localidad, nombre, codigo_postal FROM mydb.localidad
2 where codigo_postal=5000
3 order by 1;
```

< | Result Grid | Filter Rows: | Edit:    | Export/Import:  

	id_localidad	nombre	codigo_postal
	16	Córdoba	5000
	18	Santa María	5000
	NULL	NULL	NULL

Selección de TODAS las columnas

Si deseamos visualizar el contenido de todas las columnas de una tabla podemos hacer uso del carácter * como sigue:



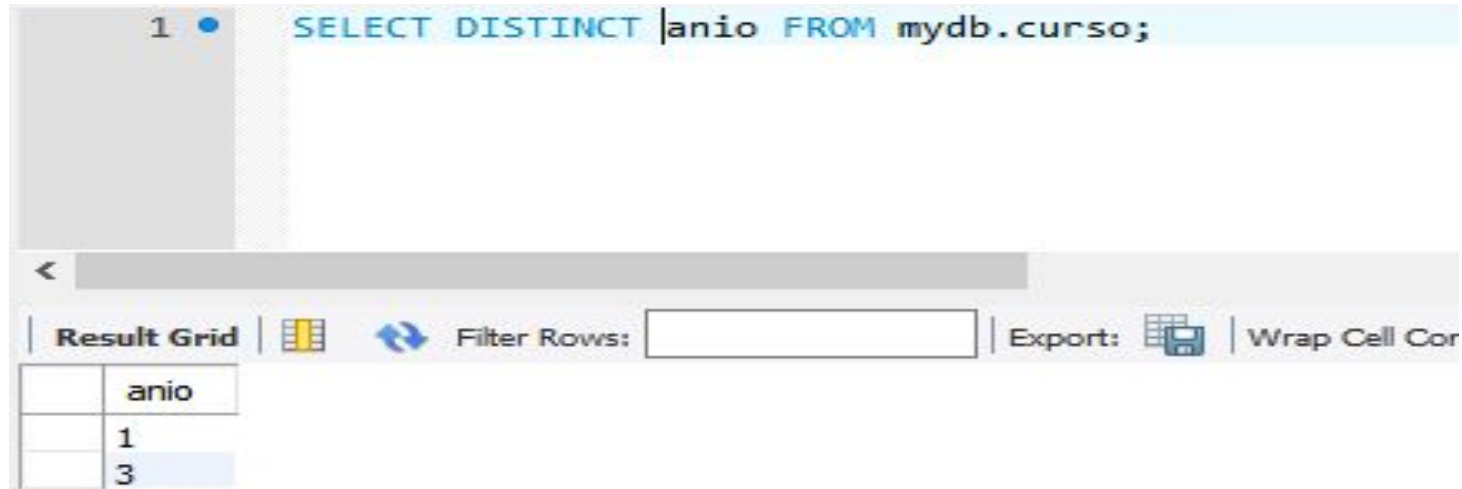
The screenshot shows a database query interface. At the top, a query editor displays the SQL statement: `SELECT * FROM mydb.provincia;`. Below the editor, a toolbar includes a 'Result Grid' button, a 'Filter Rows' input field, and an 'Edit' button. The 'Result Grid' is active, displaying a table with the following data:

	id_provincia	nombre	id_pais
1	1	Chubut	1
14	14	Córdoba	1
22	22	Buenos Aires	1
23	23	Tucumán	1
24	24	Santa Cruz	1
	NULL	NULL	NULL




Filas Duplicadas

Si una consulta incluye la clave primaria de una tabla en su lista de selección, entonces cada fila de resultados será única. Sin embargo, si no se incluye la clave primaria en los resultados, pueden producirse filas duplicadas. Para evitarlo, utilizaremos la palabra clave **DISTINCT**.

Ej. hallar los años que deben cursar los estudiantes en la escuela.



```
1 • SELECT DISTINCT |anio FROM mydb.curso;
```

< | Result Grid |   Filter Rows: | Export:  | Wrap Cell Cor

	anio
	1
	3

Cláusula WHERE

Las consultas SQL que recuperan todas las filas de una tabla son útiles para inspeccionar y elaborar informes sobre la base de datos, pero para poco más. Generalmente se deseará seleccionar solamente parte de las filas de una tabla, y sólo se incluirán esas filas en los resultados.

Cláusula WHERE

La cláusula **WHERE** consta de la palabra clave **WHERE** seguida de una condición de búsqueda que especifica las filas a recuperar. Conceptualmente, SLQ recorre cada fila de la tabla una a una y aplica la condición de búsqueda a la fila.

Por cada fila, la condición de búsqueda puede producir uno de los tres resultados:

- Si la condición es **TRUE** (cierta), la fila se incluye en los resultados de la columna.
- Si la condición es **FALSE** (falsa), la fila se excluye de los resultados de la consulta.
- Si la condición es **NULL** (desconocido), la fila también se excluye de los resultados de la consulta.

Cláusula WHERE

Condiciones de Búsqueda




- **Test de comparación.** Compara el valor de una expresión con el valor de otra.
- **Test de rango.** Examina si el valor de una expresión cae dentro de un rango especificado.
- **Test de pertenencia.** Comprueba si el valor de una expresión se corresponde con uno de un conjunto de valores.
- **Test de correspondencia de patrón.** Comprueba si el valor de una columna que contiene datos de cadenas de caracteres se corresponde a un patrón especificado.
- **Test de valor nulo.** Comprueba si una columna tiene un valor NULL.

Test de comparación

Es la condición de búsqueda más utilizada. Ofrece 6 modos diferentes de comparar (utilizando los operadores de comparación: =, <>, >, >=, < y <=).

Ej. Hallar los nombres, apellidos y fecha de nacimiento de todas las mujeres registradas en la base de datos.

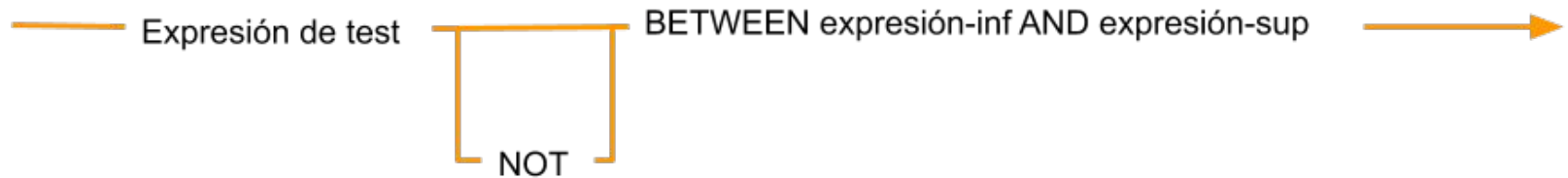
```
1 SELECT nombre, apellido, fecha_nacimiento
2 FROM mydb.persona
3 WHERE sexo="F";
```

< Result Grid   Filter Rows: | Export:  | Wrap

	nombre	apellido	fecha_nacimiento
	Estefanía	López	1978-01-01
	Rosario	Fuentes	0000-00-00

Test de rango

Comprueba si un valor se encuentra entre dos valores especificados. Para ello, deberemos utilizar la palabra clave **BETWEEN**.



Ej. Hallar las personas que nacieron entre 10/12/2000 y el 10/12/2009

```
1 SELECT nombre, apellido, fecha_nacimiento FROM mydb.persona
2 WHERE fecha_nacimiento BETWEEN '2000-12-10' AND '2009-12-10';
```

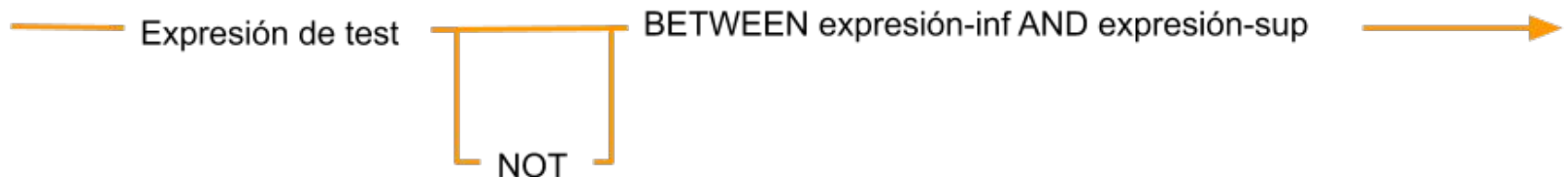
Result Grid

	nombre	apellido	fecha_nacimiento
	Rosario	Fuentes	2005-05-28

Test de rango

¿Incluye los puntos *extremos* del rango?

¿Qué sucede si hay valores *NULL*?

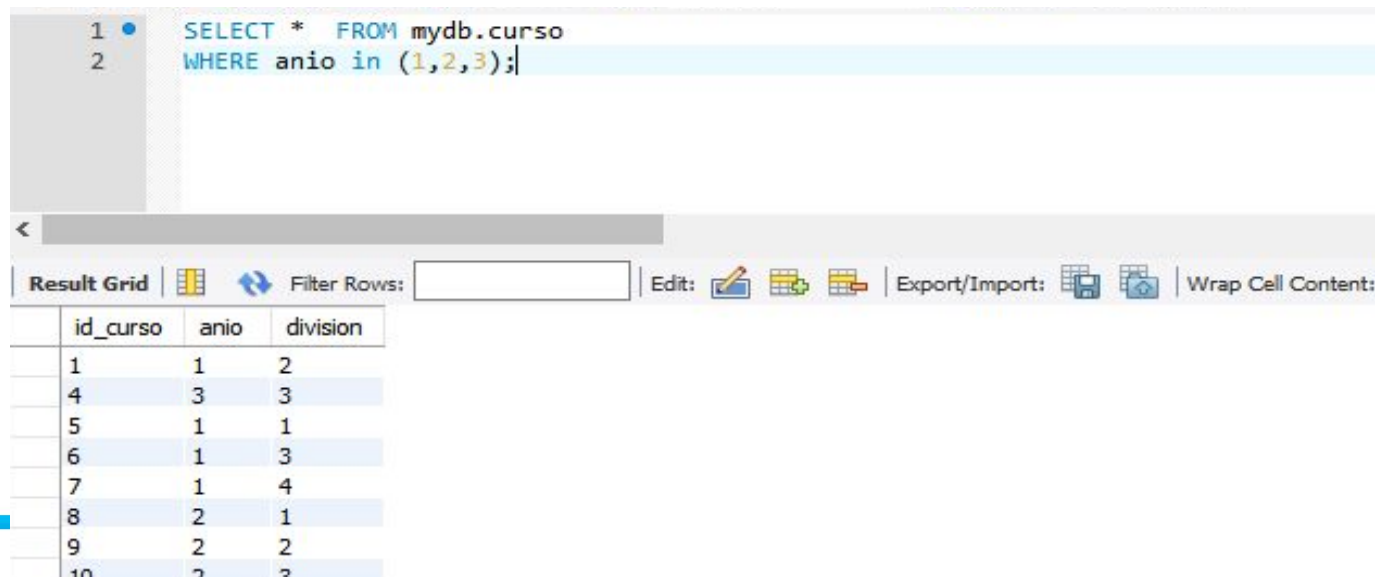


Test de pertenencia a conjunto

Examina si un valor de dato coincide con uno de la lista de valores objetivos. Para ello, debemos utilizar la palabra clave **IN**.



Ej. Hallar los cursos correspondientes al ciclo básico.



```
1 SELECT * FROM mydb.curso
2 WHERE anio in (1,2,3);
```

Result Grid

	id_curso	anio	division
1	1	1	2
4	3	3	3
5	1	1	1
6	1	3	3
7	1	4	4
8	2	1	1
9	2	2	2
10	2	3	3

Test de correspondencia





Se utiliza para recuperar las filas en donde el contenido de una consulta de texto se corresponde con un cierto texto particular. Para ello, debemos utilizar la palabra clave **LIKE**.



Ej. Hallar las personas cuyo primer apellido sea “López”

```
1 • SELECT nombre, apellido, fecha_nacimiento FROM mydb.persona
2 WHERE apellido LIKE 'López%';
```

<

Result Grid   Filter Rows: | Export:  | Wrap Cell Content: 

	nombre	apellido	fecha_nacimiento
	Estefanía	López Suarez	1978-01-01

Test de valor nulo

Comprueba explícitamente los valores NULL en una condición de búsqueda.



Ej. Hallar las localidades que no posean código postal.

```
1 • SELECT * FROM mydb.localidad
2 WHERE codigo_postal IS NULL;
```

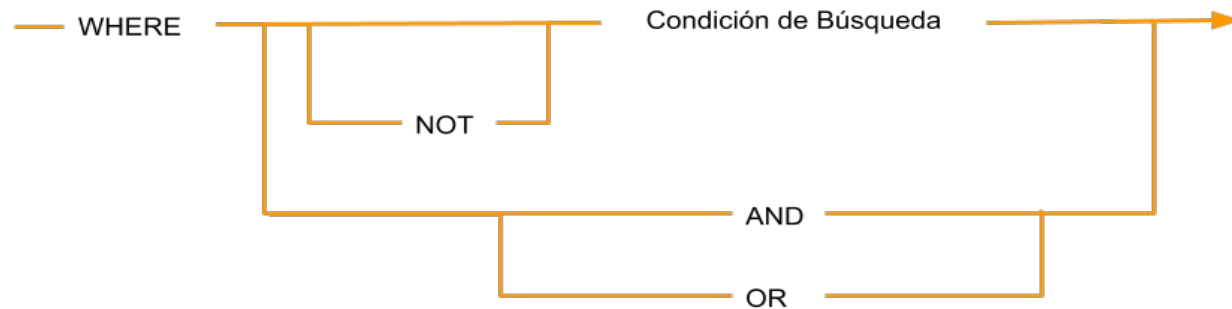


Result Grid

id_localidad	nombre	codigo_postal	id_provincia
19	Rosario	NULL	0
NULL	NULL	NULL	NULL

Condiciones de búsquedas compuestas (AND, OR y NOT)

Se pueden combinar condiciones de búsqueda simples para formar otras más complejas.



- La palabra clave **OR** se utiliza para combinar dos condiciones de búsqueda cuando una o la otra (o ambas) deban ser ciertas.
- La palabra clave **AND** para combinar dos condiciones de búsqueda que deban ser ciertas simultáneamente.
- La palabra clave **NOT** para seleccionar filas donde la condición de búsqueda es falsa.

Consultas múltiples

Muchas consultas útiles solicitan datos procedentes de dos o más tablas de la base de datos.

SQL permite recuperar datos que responden a estas peticiones mediante consultas multitabla que componen (join) datos procedentes de dos o más tablas.

El proceso de formar parejas de filas haciendo coincidir los contenidos de las columnas relacionadas se denomina **componer (joining)** las tablas. La tabla resultante (que tiene datos de las dos tablas originales) se denomina una **composición de dos columnas entre dos tablas**.

Una composición basada en una coincidencia exacta entre dos columnas se denomina **equicomposición**.

idfab	idproducto	fab	producto
bic	41672	aci	41004
imm	779c	aci	41002

COMPOSICION

idfab	idproducto	fab	producto
bic	41672	aci	41002
bic	41672	aci	41004
imm	779c	aci	41002
imm	779c	aci	41004

Composiciones

Las composiciones son el **fundamento** del procesamiento de consultas multitaslas.

Todos los datos de una base de datos relacional, están almacenados en sus columnas con valores explícitos, y todas las relaciones están de la misma manera.

Esto permite que todas las relaciones posibles entre tablas pueden formarse comparando los contenidos de las columnas relacionadas.

Las cláusula where nos permite realizar composiciones entre dos o más tablas basándose en las relaciones de columna (columnas de emparejamiento) de las tablas.

Ejemplo:

```
select l.nombre, l.codigo_postal  
from localidad l, provincia p  
where l.id_provincia=p.id_provincia
```

Consultas Padre/Hijo

Las consultas multitaslas más comunes implican a dos tablas que tienen una relación natural padre/hijo.

La tabla (localidad) que contiene la clave ajena es el hijo en la relación; la tabla con la clave primaria es el padre(provincia) :

Ejemplo:

```
select l.nombre, l.codigo_postal  
from localidad l, provincia p  
where l.id_provincia=p.id_provincia|
```

Nota: Observa que en este ejemplo utilizamos alias de tabla. Éstas permiten simplificar los nombres de columna.

Consultas de tres o más tablas

SQL permite combinar datos de tres o más tablas utilizando las mismas técnicas básicas utilizadas para las consultas de dos tablas.

Ejemplo:

```
select l.nombre, l.codigo_postal  
from localidad l, provincia p, pais pa  
where l.id_provincia=p.id_provincia  
and p.id_pais=pa.id_pais
```

Observa que en el ejemplo estamos listando todas las localidades de Argentina. Para ello, necesitamos combinar 3 tablas (localidad, provincia y país)

La inmensa mayoría de las consultas multitabla se basan en relaciones padre/hijo.

Pero SQL no exige que las columnas de emparejamiento estén relacionadas como clave primaria y clave ajena. Cualquier par de columnas de dos tablas pueden servir como columnas de emparejamiento.



Composiciones basadas en desigualdad

El término composición (join) se aplica a cualquier consulta que combina datos de dos tablas mediante comparación de los valores en una pareja de columnas de tablas.

Aunque las composiciones basadas en la desigualdad entre columnas correspondientes a equicomposiciones son más habituales, SQL permite componer basándose en otros operadores de comparación.

Ejemplo: "<,>,<>,<=,>="

```
SELECT nombre  
FROM pais  
WHERE cant_habitantes>3000000
```

- Los **nombres de columna cuantificados** son necesarios a veces en consultas multitasbla para eliminar referencias de columnas ambiguas.
- Las **selecciones de todas las columnas (Select *)** tienen un significado especial para las consultas multitasbla.
- Las **autocomposiciones** pueden ser utilizadas para crear una consulta multitasbla que relaciona una tabla consigo misma.
- Los **alias de tablas** pueden simplificar los nombres de columnas cualificados y permitir referencias de columnas no ambiguas en autocomposiciones.

Consideraciones: Nombres de columna cuantificados

Por lo general, en una base de datos se incluyen varias instancias en donde dos tablas contienen columnas con el mismo nombre. Estos nombres duplicados de columna provocan un problema. Para resolverlo, utilizamos **nombres de columna cuantificados** como sigue:

```
1 • select localidad.nombre, provincia.nombre
2   from localidad, provincia
3  where localidad.id_provincia=provincia.id_provincia
4
```

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
nombre	nombre			
Esquel	Chubut			
Trevelin	Chubut			

Consideraciones: Selecciones de todas las columnas

Select * puede ser utilizado para seleccionar todas las columnas de la tabla designada en la cláusula from.

Sin embargo, en una columna multitabla, el asterisco selecciona todas las columnas de todas las tablas listadas en la cláusula from. Por ende esta resulta menos práctica cuando hay dos o más tablas en la cláusula from. Para resolverlo, se puede utilizar el * **cuantificado con el nombre de la tabla** como sigue:

```
1 • select localidad.*, provincia.nombre
2   from localidad, provincia
3  where localidad.id_provincia=provincia.id_provincia
4
```

< Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	id_localidad	nombre	codigo_postal	id_provincia	nombre
	1	Esquel	9200	1	Chubut
	5	Trevelin	0	1	Chubut

Consideraciones: Autocomposiciones

Algunas consultas multitabla afectan a una relación que una tabla tiene consigo misma.

Para resolverlo, se utiliza una estrategia de “tabla imaginaria” para componer la tabla consigo misma. En lugar de duplicar el contenido de la tabla, simplemente nos referimos a ella mediante un nombre diferente llamado alias de tabla como sigue:

```
SELECT  emp.nombre, dirs.nombre  
from  ventas emp, ventas dirs  
where  emp.id=dirs.id
```

Consideraciones: Alias de tabla

Se puede utilizar un alias de tabla en cualquier consulta.

Por ejemplo, si una consulta incluye nombres de tablas muy largos, se puede simplificar la tarea utilizando alias de tabla como sigue:

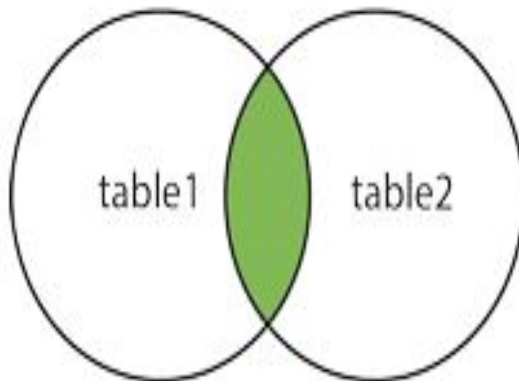
```
SELECT l.nombre, p.provincia  
from localidad l, provincia p  
where l.id_provincia=p.id_provincia
```

Composiciones

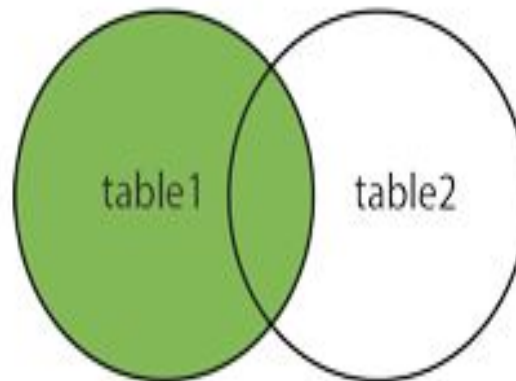
Otras formas de hacer composiciones:

- Inner Join:
- Left Join
- Right Join
- Full Outer Join

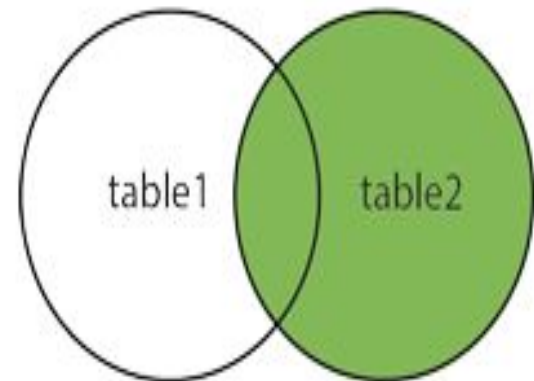
INNER JOIN



LEFT JOIN



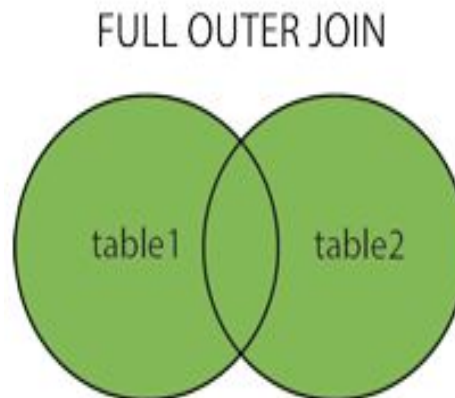
RIGHT JOIN



Composiciones externas

La composición (interna) estándar, que estuvimos viendo hasta ahora puede perder información si las tablas que se componen contienen filas sin emparejar.

Las composiciones externas son preservadoras de información. Para ello, se utiliza la cláusula FULL OUTER JOIN.

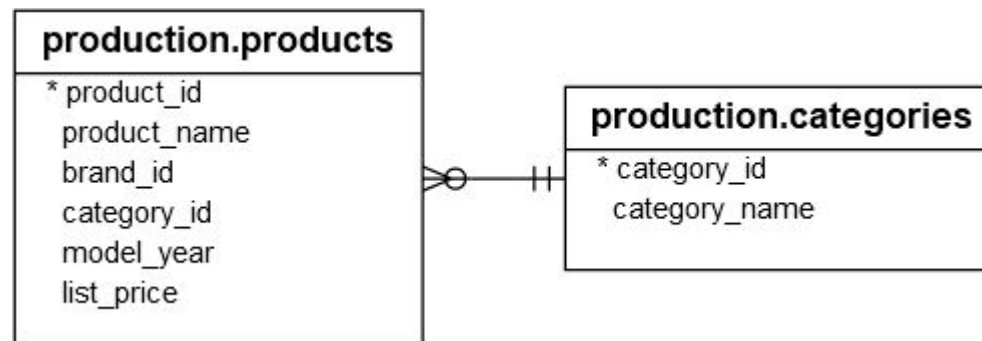


Combinaciones con INNER JOIN

La combinación interna es una de las combinaciones más utilizadas en SQL Server. La cláusula de combinación interna le permite consultar datos de dos o más tablas relacionadas.

Consulte las siguientes tablas de productos y categorías:

Ejercicio: Escribir una consulta que devuelva id, nombre y categoría de producto.



Las combinaciones internas se realizan mediante la instrucción **INNER JOIN**.

Devuelven únicamente aquellos registros/filas **que tienen valores idénticos en los dos campos** que se comparan para unir ambas tablas. Es decir aquellas que tienen elementos en las dos tablas, identificados éstos por el campo de relación.

Solución:

```
SELECT product_name, category_name, list_price  
FROM production.products p  
INNER JOIN production.categories c  
    ON c.category_id = p.category_id  
ORDER BY product_name DESC;
```

Los C y p son los alias de las tablas `production.categories` y `production.products`. Al hacer esto, cuando hace referencia a una columna en esta tabla, puede usar `alias.column_name` en lugar de usar `table_name.column_name`. Por ejemplo, la consulta usa `c.category_id` en lugar de `production.categories.category_id`. Por lo tanto, le ahorra algunas mecanografías.

Para cada fila de la tabla `production.products`, la cláusula de unión interna la empareja con cada fila de la tabla `product.categories` según los valores de la columna `category_id`:

- Si ambas filas tienen el mismo valor en la columna `category_id`, la combinación interna forma una nueva fila cuyas columnas son de las filas de las tablas `production.categories` y `production.products` de acuerdo con las columnas de la lista de selección e incluye esta nueva fila en la conjunto resultante.
- Si la fila de la tabla `production.products` no coincide con la fila de la tabla `production.categories`, la cláusula de unión interna simplemente ignora estas filas y no las incluye en el conjunto de resultados.

Cláusula GROUP BY

La cláusula GROUP BY le permite organizar las filas de una consulta en grupos. Los grupos están determinados por las columnas que especifique en la cláusula GROUP BY.

A continuación se ilustra la sintaxis de la cláusula GROUP BY:

```
SELECT
    select_list
FROM
    table_name
GROUP BY
    column_name1,
    column_name2 ,...;
```

Cláusula GROUP BY:

En esta consulta, la cláusula GROUP BY produjo un grupo para cada combinación de los valores en las columnas enumeradas en la cláusula GROUP BY. Considere el siguiente ejemplo:

```
SELECT  customer_id, YEAR (order_date) order_year  
  
FROM    sales.orders  
  
WHERE   customer_id IN (1, 2)  
  
ORDER BY customer_id;
```

“No tiene la cláusula GROUP BY, agregar antes de ORDER BY”

GROUP BY cláusula y funciones agregadas

En la práctica, la cláusula GROUP BY se utiliza a menudo con funciones agregadas para generar informes resumidos.

Una función agregada realiza un cálculo en un grupo y devuelve un valor único por grupo. Por ejemplo, COUNT () devuelve el número de filas de cada grupo. Otras funciones agregadas comúnmente utilizadas son SUM (), AVG () (promedio), MIN () (mínimo), MAX () (máximo).

La cláusula GROUP BY organiza las filas en grupos y una función agregada devuelve el resumen (recuento, mínimo, máximo, promedio, suma, etc.) para cada grupo.

GROUP BY cláusula y funciones agregadas

Por ejemplo, la siguiente consulta devuelve el número de pedidos realizados por el cliente por año:

```
SELECT customer_id, YEAR (order_date) order_year, COUNT (order_id)  
order_placed  
FROM sales.orders  
WHERE customer_id IN (1, 2)  
GROUP BY customer_id, YEAR (order_date)  
ORDER BY customer_id;
```

Usando la cláusula GROUP BY con COUNT () ejemplo de función

La siguiente consulta devuelve el número de clientes en cada ciudad:

```
SELECT  city, COUNT (customer_id) customer_count  
FROM    sales.customers  
GROUP BY  city  
ORDER BY  city;
```


Sub Consultas

Permiten utilizar los resultados de una consulta como parte de otra.

Esta característica es menos conocida que las composiciones de SQL pero juegan un papel importante por 3 razones:

- Una sentencia SQL con una subconsulta es frecuentemente el modo más natural de expresar una consulta, ya que se asemeja más a la descripción de la consulta en lenguaje natural.
- Las subconsultas hacen más fácil la escritura de sentencias SELECT, ya que permiten “descomponer una consulta en partes” y luego “recomponer las partes”.
- Hay algunas consultas que no pueden ser expresadas sin utilizar subconsultas.

Subconsultas

La subconsulta está siempre encerrada entre paréntesis, pero por otra parte tiene el formato familiar a una sentencia SELECT, con una cláusula FROM y cláusulas opcionales WHERE, GROUP BY y HAVING.

El formato de estas cláusulas en una subconsulta es idéntico al que tienen en una sentencia SELECT, y efectúa sus funciones normales cuando se utilizan dentro de una subconsulta.

```
SELECT column_name [, column_name ]  
FROM table1 [, table2 ]  
WHERE column_name OPERADOR  
    (SELECT column_name  
     FROM table1 [, table2 ]  
     [WHERE])
```

Subconsultas: Utilización

Una subconsulta es una consulta que aparece dentro de la cláusula WHERE o HAVING de otra sentencia SQL.

Las subconsultas proporcionan un modo eficaz y natural de gestionar peticiones de consultas que se expresan en términos de los resultados de otras consultas.

```
SELECT  dep_no "N° Empleado", apellido, salario
FROM    empleados
WHERE   salario > (SELECT AVG(salario)
                  FROM empleados );
```

Subconsultas

Sin embargo hay algunas diferencias entre una subconsulta y una sentencia select:

- Una subconsulta debe producir una única columna de datos como resultado.
- La cláusula ORDER BY no puede ser especificada en la subconsulta.
- Una subconsulta no puede ser la UNION de varias sentencias SELECT diferentes.
- Los nombres de columna que aparecen en una subconsulta puede referirse a columnas de tablas de la columna principal.

Subconsultas: Subconsultas en la cláusula WHERE

Las subconsultas suelen ser utilizadas principalmente en la cláusula WHERE de una sentencia sql.

Cuando aparece una subconsulta en la cláusula WHERE, ésta funciona como parte del proceso de selección de filas.

Ejemplo:

```
1 • select p.nombre, p.apellido
2   from persona p
3  inner join alumno a on a.id_persona=p.id_persona
4  where a.n_libro_matriz = (select min(n_libro_matriz) from persona)
```

El ejemplo lista los nombres y apellidos de los estudiantes que tienen asignado el libro matriz más antiguo en la base de datos.

Subconsultas: Condiciones de búsqueda en subconsultas

Una subconsulta forma parte de una condición de búsqueda en la cláusula WHERE o HAVING.

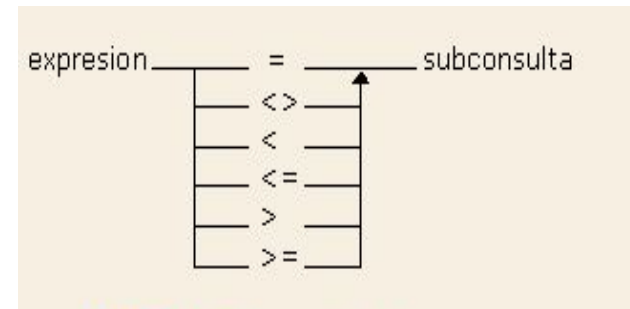
- **Test de comparación subconsulta.** Compara el valor de una expresión con un valor único producido por una subconsulta.
- **Test de pertenencia a conjunto subconsulta.** Comprueba si el valor de una expresión coincide con uno del conjunto de valores producidos por una subconsulta.
- **Test de existencia.** Examina si una subconsulta produce alguna fila de resultados.
- **Test de comparación cuantificada.** Compara el valor de una expresión con cada uno del conjunto de valores producido por una subconsulta.

Subconsultas: Test de comparación subconsulta (=, <>, <,>, <=,>=).

Compara el valor de una expresión con el valor producido por una subconsulta y devuelve TRUE si la comparación es cierta.

Este test se utiliza para comparar un valor de la fila que está siendo examinado con un valor único producido por una subconsulta, como en el siguiente ejemplo:

El ejemplo lista los nombres y apellidos de los estudiantes que tienen asignado el libro matriz más antiguo en la base de datos.



```
1 • select p.nombre, p.apellido
2   from persona p
3  inner join alumno a on a.id_persona=p.id_persona
4  where a.n_libro_matriz = (select min(n_libro_matriz) from persona)
```

Subconsultas: Test de pertenencia a un conjunto (IN)

Compara un único valor de datos con una columna de valores producida por una subconsulta y devuelve el resultado TRUE si el valor coincide con uno de los valores de la columna.

Este test se utiliza cuando se necesita comparar un valor de la fila que está siendo examinada con un conjunto de valores producidos por una subconsulta como se muestra en el ejemplo:

```
1 • select p.nombre, p.apellido
2   from persona p
3  inner join alumno a on a.id_persona=p.id_persona
4  where a.id_alumno in (select DISTINCT id_alumno
5                        from trayectoria where notas < 6
6                        and fecha between '30-03-2019' and '30-12-2019')
```

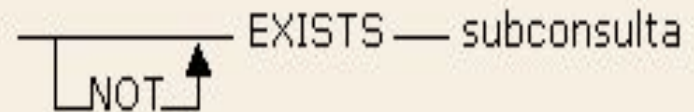
El ejemplo lista los nombres y apellidos de los estudiantes que tienen una nota menor a 6 en el año 2019.

Subconsultas: Test de existencias (EXIST)

El test de existencia (EXIST) comprueba si una subconsulta produce alguna fila de resultados.

No hay test de comparación que se asemeje a este test, solamente se utiliza con subconsultas.

Ejemplo:



```
1 • select nombre
2   from localidad
3  where EXISTS (select distinct * from persona
4                  where id_localidad=localidad.id_localidad)
```

El ejemplo lista los nombres de las localidades en donde residen las personas registradas en la base de datos.

Subconsultas: Test cuantificados (ANY y ALL)

La versión subconsulta del test IN comprueba si un valor de dato es igual a algún valor en la comuna de los resultados de la subconsulta.

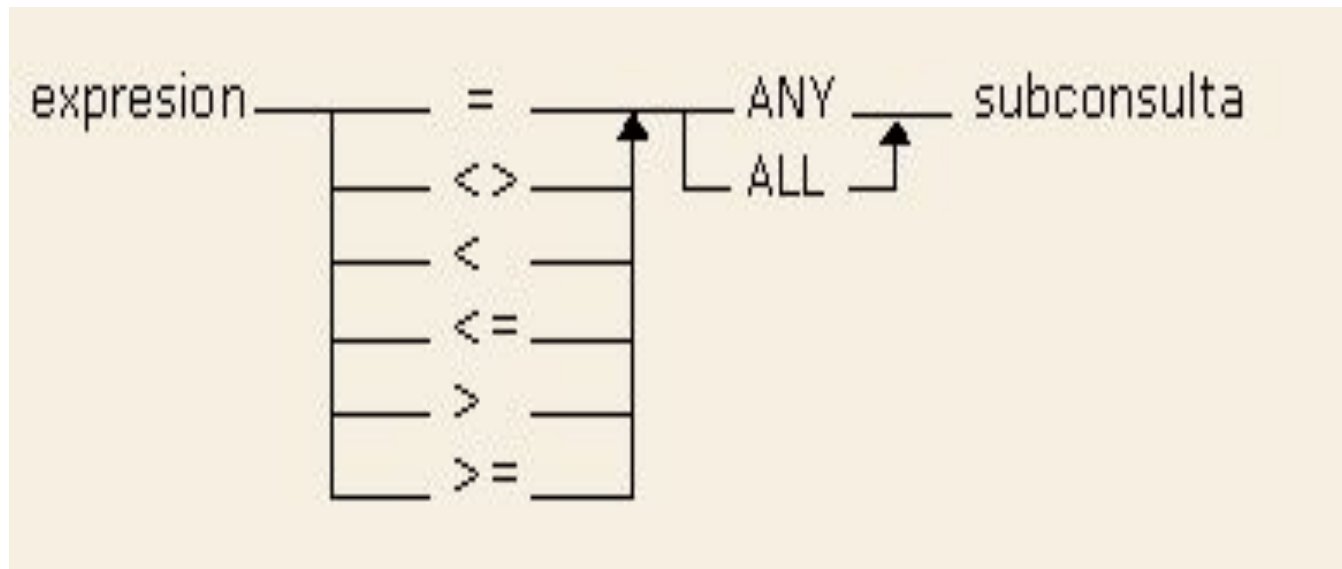
SQL proporciona dos test cuantificados que extienden esta noción a otros operadores de comparación, tales como mayor que ($>$) y menor que ($<$).

Ambos test comparan un valor de dato con la columna de valores producidos por una subconsulta.

Subconsultas: El test ANY.

Se utiliza conjuntamente con uno de los 6 operadores de comparación (=, <>, <, >, <=, >=) para comparar un único valor de test con una columna de valores producidos por una subconsulta.

Si alguna de las comparaciones individuales producen un resultado TRUE, el test ANY devuelve un resultado TRUE.



Subconsultas: El test ALL

Al igual que el test ANY, el test ALL se utiliza conjuntamente con uno de los 6 operadores de comparación para comparar un único valor de test con una columna de valores de datos producidos por una subconsulta.

Para efectuar el test, SQL utiliza el operador de comparación especificado para comparar el valor del test con TODOS y cada uno de los valores de la columna.

Si todas las comparaciones individuales producen un resultado TRUE, el test ALL devuelve TRUE.

