Consultas de selección

Las consultas de selección nos permiten seleccionar información de una o más tablas. Además posibilitan el uso de filtros, ordenamientos, transformaciones y resúmenes para presentar los datos seleccionados.

En una consulta de selección, un conjunto de datos de una base que está almacenada en un medio físico (disco rígido, disco de estado sólido, etc.) son transferidos a la memoria RAM mediante ciertos criterios, y en forma de arrays, listas, diccionarios, DataTables, DataSets, DataReaders, etc. con el fin de ser consumidos, por ejemplo, en una aplicación.

Es importante aclarar que, aunque se transforme uno o más datos para dar cierto formato a la información de salida, una consulta de selección nunca modifica datos, los cuales permanecerán inalterables luego de la ejecución de la misma. Lo mismo aplica para la estructura de tablas.

Selección de una, varias o todas las columnas de una tabla

La sintaxis básica para realizar una consulta de selección es mediante la palabra reservada **SELECT**. Luego, se especifica la o las columnas cuyo valor necesitamos recuperar, a continuación la palabra **FROM** para indicar de qué tabla o tablas extraemos los datos, y finalmente el nombre de la tabla:

```
SELECT * FROM [Tabla];

SELECT [Columna1], [Columna2] FROM [Tabla];
```

Veamos algunos ejemplos:

```
USE Libreria;
SELECT * FROM Libros;
```

En esta consulta ponemos en uso la base de datos Libreria y luego obtenemos todas las columnas de todos los registros existentes en la tabla Libros. El signo **asterisco (*)** se utiliza para indicar que queremos obtener todas las columnas de la tabla.

```
USE Libreria;
SELECT Titulo FROM Libros;
```

En este caso obtenemos solamente los datos de la columna Titulo de todos los registros de Libros.

Alias

Consultaremos algunas columnas de los registros de la tabla Libros:

```
USE Libreria;
SELECT IDLibro, Titulo, AñoPublicacion, Paginas FROM Libros;
```

	IDLibro	Titulo	AñoPublicacion	Paginas
1	1	Holly	2023	432
2	2	El visitante	2018	592
3	3	Fin de guardia	2016	448
4	4	Quien pierde paga	2015	434
5	5	Mr Mercedes	2014	437

Aquí obtenemos, para todos los registros de Libros, los datos de las columnas IDLibro, Titulo, AñoPublicacion y Paginas. Pero… ¿qué sucede si necesitamos que algunas columnas se muestren con otro nombre? Utilizamos la palabra **AS** para establecer un alias, y a continuación el nombre deseado entre comillas simples:

```
USE Libreria;
SELECT IDLibro AS 'ID de libro', Titulo, AñoPublicacion AS 'Año de
publicacion', Paginas AS 'Cantidad de paginas' FROM Libros;
```

	ID de libro	Titulo	Año de publicacion	Cantidad de paginas
1	1	Holly	2023	432
2	2	El visitante	2018	592
3	3	Fin de guardia	2016	448
4	4	Quien pierde paga	2015	434
5	5	Mr Mercedes	2014	437

De esta manera estamos modificando el nombre que se mostrará para las columnas IDLibro, AñoPublicacion y Paginas. Como mencionamos anteriormente, el uso de alias no modifica el nombre original de la columna.

Los nombres de tablas también pueden recibir alias, por ejemplo:

```
USE Libreria;
SELECT L.* FROM Libros AS L;
```

Obtenemos todas las columnas de todos los registros de la tabla Libros que, en esta consulta, está renombrada como L. Esta forma de abreviar los nombres de las tablas nos será mucho más útil más adelante, cuando veamos consultas complejas que abarquen varias tablas.

Concatenación de cadenas de texto - Nulos

Supongamos que necesitamos obtener un listado de los apellidos y nombres de autores de libros con el siguiente formato:

```
Verne, Julio
```

Para obtener dicho formato tendremos que hacer una *concatenación* de los campos que contienen el texto a seleccionar. La sintaxis será la siguiente:

```
SELECT Apellidos + ', ' + Nombres AS 'Apellidos y Nombres' FROM Autores;
```

	Apellidos y Nombres
1	Murakami, Haruki
2	*NULL*
3	King, Stephen
4	Crichton, Michael
5	Kawamura, Genki

Es probable que en algunos casos no queramos tener un NULL en nuestro listado. Por ejemplo, en casos de autores cuyo apellidos y/o nombres sean desconocidos, que la consulta devuelva una cadena vacía o bien la leyenda 'Apellidos desconocidos'. Para ello utilizamos **ISNULL**:

```
SELECT ISNULL(Apellidos, '') + ' ' + ISNULL(Nombres, '')
AS 'Apellidos y Nombres' FROM Autores;

SELECT ISNULL(Apellidos, '[Apellidos desconocidos]') + ' '
+ ISNULL(Nombres, '[Nombres desconocidos]') AS 'Apellidos y Nombres'
FROM Autores;
```

La sintaxis de ISNULL es muy simple: el primer parámetro representa la columna a devolver y el segundo el texto de reemplazo si la misma no contiene dato (NULL):

```
SELECT ISNULL([Columna], 'texto_de_reemplazo')
```

A continuación observemos cuáles fueron los resultados:

	Apellidos y Nombres
1	Murakami Haruki
2	
3	King Stephen
4	Crichton Michael
5	Kawamura Genki

	Apellidos y Nombres
1	Murakami Haruki
2	[Apellidos desconocidos] [Nombres desconocidos]
3	King Stephen
4	Crichton Michael
5	Kawamura Genki

También contamos con una utilidad que funciona de manera similar a ISNULL pero, a diferencia de ella, admite más parámetros: **COALESCE**.

Por ejemplo, en nuestro listado necesitamos saber los apellidos de los autores. Si dichos datos no se encuentran, entonces sus nombres. Pero si tampoco se conocen estos valores, es el seudónimo lo que, al menos, queremos devolver. Para ello escribiremos lo siguiente:

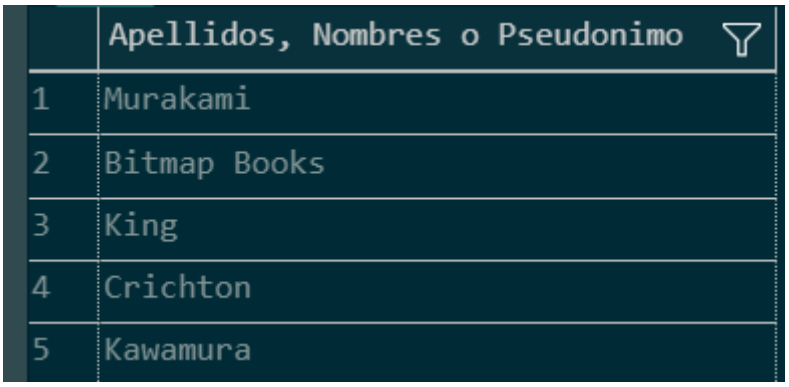
```
SELECT COALESCE(Apellidos, Nombres, Pseudonimo)
AS 'Apellidos, Nombres o Pseudonimo' FROM Autores;
```

En la imagen de página 3 vimos que, en el segundo registro de Autores, las columnas Apellidos y Nombres no tenían valor, con lo cual obteníamos NULL. Ahora intentamos recuperar el valor de las columnas según el orden de aparición dentro de COALESCE:

primero los apellidos, si se desconocen los nombres, y si éstos también son desconocidos, el seudónimo. En otras palabras, esta función evalúa los argumentos en orden, y devuelve el primero que no contiene NULL:

```
SELECT COALESCE([Columna1], [Columna2], … , [ColumnaN])
```

Obtenemos el siguiente resultado:



	Apellidos, Nombres o Pseudonimo
1	Murakami
2	Bitmap Books
3	King
4	Crichton
5	Kawamura

En la imagen observamos que el segundo registro, que no posee valores en los campos Apellidos y Nombres, ahora muestra el valor de la columna Pseudonimo.

Consultas que no necesitan especificar tablas

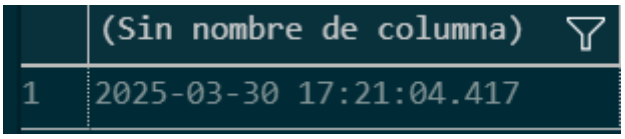
Con SELECT no solamente podemos obtener valores de columnas, sino también el resultado de operaciones o funciones.

Podemos notar, en este tipo de consultas, la ausencia de la cláusula FROM, ya que no necesitamos una tabla de la base de datos para obtener los resultados. Veamos un par de ejemplos:

Fecha actual

Mediante la siguiente consulta obtenemos la fecha y hora actuales del servidor:

```
SELECT GETDATE();
```



	(Sin nombre de columna)
1	2025-03-30 17:21:04.417

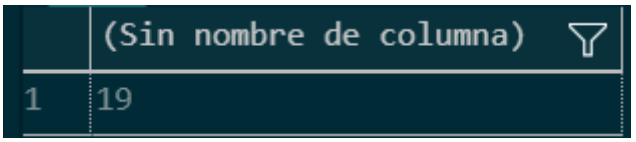
Operaciones aritméticas

Supongamos que queremos obtener el siglo en el que nacimos. Sabemos que el año 1900 corresponde al siglo XIX y que, entre el 1º de enero de 1901 y el 31 de diciembre de 2000 inclusive, transcurrió el siglo XX. Si, por ejemplo, mi año de nacimiento es 1984, mediante el siguiente cálculo y un redondeo obtendré el siglo en el que nací:

```
1984 / 100
```

El resultado, considerando la parte decimal, da 19.84. Pero como los siglos se representan mediante números enteros, hacemos un redondeo para quedarnos con la parte entera de ese valor: como 19.84 está más cerca de 20.00 que de 19.00, tendremos finalmente que mi siglo de nacimiento es el 20, lo cual es correcto. Veamos cómo hacer este cálculo en SQL:

```
SELECT 1984/100;
```

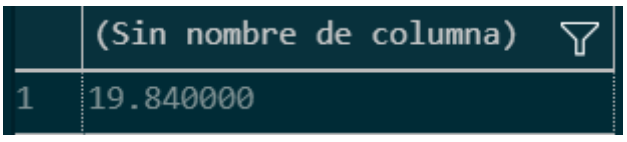


	(Sin nombre de columna)
1	19

Sin embargo, no obtuvimos el resultado esperado. ¿Por qué?

Tal como sucedía con estos cálculos en C++ (materias anteriores), el operador de división "**/**" efectúa la **división entera** entre dos números. Entonces necesitaremos realizar un *casteo* de alguno de los dos números para no perder la parte decimal:

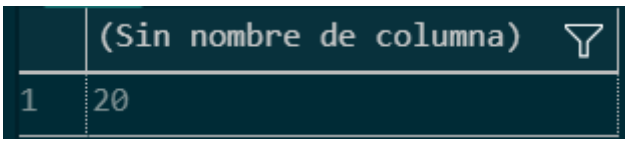
```
SELECT 1984/100.0;
```



	(Sin nombre de columna)
1	19.840000

Vemos cómo, convirtiendo el número 100 de entero a real, estamos más cerca de llegar al resultado final. Sólo nos falta una función que realice el redondeo hacia arriba, por lo que utilizamos **CEILING**:

```
SELECT CEILING(1984/100.0);
```

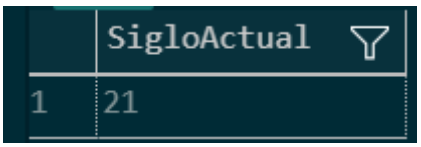


	(Sin nombre de columna)
1	20

NOTA: si el resultado se consumirá desde una aplicación, es importante establecer un alias.

Finalmente veamos cómo obtener el siglo en el que estamos actualmente:

```
SELECT CEILING(YEAR(GETDATE()) / 100.0) AS SigloActual;
```



	SigloActual
1	21

Lo que hicimos fue obtener la fecha actual (del servidor) mediante la función GETDATE(), luego extrajimos el año de dicha fecha a través de **YEAR** (que recibe una fecha como parámetro y devuelve un número entero que representa el año de esa fecha), y finalmente efectuamos el cálculo visto anteriormente para dar con el número de siglo. Notar que el alias *SigloActual* no se escribió dentro de comillas simples porque no contiene espacios. Además, las operaciones fueron realizadas entre constantes, con lo cual no necesitamos la cláusula FROM (no estamos consultando datos de tablas).

Seleccionando columnas con una decisión

Utilizaremos los últimos ejemplos para realizar una consulta un poco más compleja. Ahora listaremos: apellidos y nombres de los autores (con el formato Apellidos + “, “ + Nombres; en caso de ser ambos nulos mostraremos el seudónimo), su año de nacimiento, su siglo de nacimiento en numeración decimal (con el alias ‘Siglo’) y una columna llamada ‘Siglo en numeros romanos’ que será “calculada” en base al valor del campo anterior:

```
SELECT
ISNULL(Apellidos + ', ' + Nombres, Pseudonimo) AS 'Apellidos y Nombres o
Pseudonimo',
AñoNacimiento,
CEILING(AñoNacimiento/100.0) AS Siglo,
CASE WHEN AñoNacimiento IS NULL THEN 'Desconocido'
     WHEN AñoNacimiento < = 1800 THEN 'Muy antiguo'
     WHEN AñoNacimiento < = 1900 THEN 'XIX'
     WHEN AñoNacimiento < = 2000 THEN 'XX'
     ELSE 'XXI' END AS 'Siglo en numeros romanos'
FROM Autores;
```

Analicemos en detalle la forma en que se obtienen algunas de las columnas:

```
ISNULL(Apellidos + ', ' + Nombres, Pseudonimo) AS 'Apellidos y Nombres o
Pseudonimo',
```

Aquí obtenemos apellidos y nombres de los autores separados con una coma, y en caso de ser ambos nulos (lo cual retornará NULL), mostramos el pseudónimo.

El cálculo de siglo en numeración decimal no lo explicaremos aquí ya que lo vimos antes.

En cuanto a la columna '*Siglo en numeros romanos*', su valor se obtiene a través de una **estructura de decisión,** que en este lenguaje posee algunas características: se inicia con la palabra **CASE**, y para delimitar cada caso a evaluar utilizamos **WHEN**. Cada uno de los casos está conformado por una **proposición lógica** (o un conjunto de ellas), y en cada una generalmente intervienen una o más columnas. La palabra **THEN** se utiliza para especificar el valor a mostrar en caso de que la condición se evalúe como **true** (o verdadera). Esta estructura funciona de manera secuencial, como un *switch*: se comienza por el caso que aparece en primer lugar (arriba de todo), y en caso de evaluarse como **false**, se continuará con el que le sigue en orden, y así sucesivamente. Finalmente tenemos un caso por defecto, indicado con **ELSE**: este caso es opcional y se evaluará si todos los anteriores retornaron **false.** Cerramos la estructura con **END** y utilizamos un alias para dar nombre a la nueva columna que se generó en esta decisión.

```
SELECT
CASE
WHEN [Condicion1] THEN '[Valor1]'
WHEN [Condicion2] THEN '[Valor2]'
WHEN [Condicion3] THEN '[Valor3]'
. . .
WHEN [CondicionN] THEN '[ValorN]'
ELSE [Valor_por_defecto]
END AS [Alias_Nombre_Columna]
```

En este ejemplo, el valor a evaluar será el de la columna AñoNacimiento. Aquí vemos que entran en juego los **operadores relacionales** para determinar si el valor es nulo o bien está dentro de cierto rango. Por medio de ellos evaluamos cuatro casos (null, <=1800, <=1900, <=2000) y por último utilizamos el valor por defecto para los años mayores a 2000.

En la siguiente sección veremos cómo trabajar con rangos y evaluación de nulos.

Cláusula WHERE

La posibilidad de filtrar los datos y obtener solamente los que necesitamos, otorga mucho poder y funcionalidad a las consultas de selección. Y, como vimos anteriormente, es fundamental en la mayoría de las consultas de acción, ya que en general sólo queremos aplicar las mismas a un pequeño conjunto de datos. Por todo ello, la cláusula **WHERE** tiene suma relevancia y además es ampliamente utilizada.

Tal como sucedía en las consultas de acción, la palabra reservada WHERE se utiliza para excluir filas (o registros) del resultado de la selección, a partir de una o más condiciones que deben cumplirse. Su estructura básica es la siguiente:

```
SELECT Columna1, Columna2 FROM Tabla WHERE Columna1 = [valor];
```

También se la podría definir de esta manera:

```
SELECT Columna1, Columna2 FROM Tabla WHERE [Condicion_ES_VERDADERA];
```

Ejemplo 1: seleccionamos todos los datos (todas las columnas) de autores argentinos (es decir, cuyo IDPais sea igual a 'ARG'):

```
SELECT * FROM Autores WHERE IDPais = 'ARG';
```

Ejemplo 2: seleccionamos el título de aquellos libros que hayan sido publicados en el siglo XX o anterior:

```
SELECT Titulo FROM Libros WHERE AñoPublicacion < = 2000;
```

Aquí vemos que no es necesario incluir en el SELECT la columna por la cual se filtra: sólo necesitamos conocer los títulos de los libros cuya selección se restringe a su año de publicación. Esto no afecta en absoluto al resultado, es decir, tanto la selección de datos como el filtro se realizan correctamente.

Operadores lógicos y relacionales

Como habrán observado en los ejemplos, el funcionamiento de WHERE se sustenta en el uso de operadores lógicos y relacionales. Dijimos anteriormente que WHERE puede tener una o más condiciones, por lo cual es razonable pensar que estos operadores pueden actuar por separado o combinarse entre sí, según la necesidad. En efecto, WHERE hace uso de operadores relacionales para establecer proposiciones lógicas, y de operadores lógicos para conectar o combinar dichas proposiciones.

Entre los operadores relacionales o de comparación podemos encontrar: igual (**=**), distinto (**<>** o **!=**), mayor (**>**), mayor o igual (**>=**), menor (**<**), menor o igual (**<=**). Mientras que los operadores lógicos están representados por Y (**AND**), O (**OR**) y NO (negación, **NOT**).

Veámoslo con ejemplos:

Un listado con ID de país, apellidos y año de nacimiento de los autores cuyo ID de país no sea 'JPN' y que hayan nacido en los años 1932, 1942, 1949 o 1979:

```
SELECT IDPais, Apellidos, AñoNacimiento
FROM Autores
WHERE IDPais < > 'JPN' AND AñoNacimiento = 1932 OR AñoNacimiento = 1942
OR AñoNacimiento = 1949 OR AñoNacimiento = 1979;
```

Lo que escribimos aparenta ser correcto para obtener los resultados deseados, sin embargo al ejecutar esta consulta vemos que:

	IDPais	Apellidos	AñoNacimiento
1	JPN	Murakami	1949
2	USA	Crichton	1942
3	JPN	Kawamura	1979
4	ITA	Eco	1932

Dos de los registros no forman parte del conjunto de resultados que buscábamos: si bien cumplen la condición de haber nacido en los años 1949 y 1979, tienen IDPais igual a 'JPN'. Recordemos que lo que queríamos obtener eran los autores que no tengan IDPais igual a 'JPN' y, dentro de este conjunto, los que nacieron en los años 1932, 1942, 1949 o 1979.

Lo que sucedió es que no establecimos ningún nivel de precedencia entre las condiciones, entonces el gestor de base de datos interpretó lo siguiente: los autores que no hayan nacido en un país con ID = 'JPN' y que su año de nacimiento sea 1932, o bien que hayan nacido en 1942, o bien que hayan nacido en 1949, o bien que hayan nacido en 1979. Es decir, algo como esto:

```
SELECT IDPais, Apellidos, AñoNacimiento
FROM Autores
WHERE (IDPais < > 'JPN' AND AñoNacimiento = 1932)
OR AñoNacimiento = 1942
OR AñoNacimiento = 1949
OR AñoNacimiento = 1979;
```

Entonces, para establecer las condiciones tal como las necesitamos, debemos incluir los paréntesis para separar las proposiciones lógicas según la manera en que queremos que sean evaluadas. Lo correcto es escribir la consulta de esta forma:

```
SELECT IDPais, Apellidos, AñoNacimiento
FROM Autores
WHERE (IDPais < > 'JPN') AND (AñoNacimiento = 1932 OR AñoNacimiento = 1942
OR AñoNacimiento = 1949 OR AñoNacimiento = 1979);
```

Así quedan delimitadas las condiciones de ID de país por un lado y de año de nacimiento por otro, que es lo que necesitamos.

Rangos

Si queremos obtener un listado de los libros publicados en el siglo XX (sabiendo que sus años de publicación deben estar entre 1901 y 2000), podemos escribir la siguiente consulta:

```
SELECT Titulo, AñoPublicacion FROM Libros
WHERE AñoPublicacion > = 1901 AND AñoPublicacion < = 2000;
```

	Titulo	AñoPublicacion
1	Salem's Lot	1975
2	Apocalipsis	1978
3	Jurassic Park	1990
4	El lector	1995
5	Cementerio de animales	1983
6	El piloto del Danubio	1908
7	Las Nieves del Kilimanjaro y otros cuentos	1936
8	Needful things	1991
9	El umbral de la noche	1978
10	Las cuatro estaciones: Primavera y verano	1982
11	Las cuatro estaciones: Otoño e invierno	1982
12	History of food	1987
13	De cómo un rey perdió Francia	1955
14	La loba de Francia	1959
15	La ley de los varones	1957
16	Los venenos de la corona	1956
17	El nombre de la rosa	1980
18	Rabia	1977
19	Confesiones de un chef	2000
20	Mientras escribo	2000

Operador BETWEEN

La consulta anterior es correcta tal como está formulada. Sin embargo, si deséaramos abreviarla, podemos reescribirla utilizando la cláusula **BETWEEN**:

```
SELECT Titulo, AñoPublicacion FROM Libros
WHERE AñoPublicacion BETWEEN 1901 AND 2000;
```

	Titulo	AñoPublicacion
1	Salem's Lot	1975
2	Apocalipsis	1978
3	Jurassic Park	1990
4	El lector	1995
5	Cementerio de animales	1983
6	El piloto del Danubio	1908
7	Las Nieves del Kilimanjaro y otros cuentos	1936
8	Needful things	1991
9	El umbral de la noche	1978
10	Las cuatro estaciones: Primavera y verano	1982
11	Las cuatro estaciones: Otoño e invierno	1982
12	History of food	1987
13	De cómo un rey perdió Francia	1955
14	La loba de Francia	1959
15	La ley de los varones	1957
16	Los venenos de la corona	1956
17	El nombre de la rosa	1980
18	Rabia	1977
19	Confesiones de un chef	2000
20	Mientras escribo	2000

Como podemos corroborar, el conjunto de resultados obtenidos con BETWEEN es exactamente igual al anterior.

Utilizamos BETWEEN para definir un intervalo cerrado en nuestras consultas, es decir:

```
SELECT * FROM [Tabla] WHERE [Columna1] BETWEEN [Valor1] AND [Valor2];
```

es lo mismo que escribir:

```
SELECT * FROM [Tabla] WHERE [Columna1] > = [Valor1] AND
[Columna1] < = [Valor2];
```

Como cualquier otro operador relacional, BETWEEN admite la negación.

Supongamos que necesitamos título y cantidad de páginas de los libros que **no** tengan una cantidad de páginas entre 400 y 499:

```
SELECT Titulo, Paginas FROM Libros WHERE Paginas NOT BETWEEN 400 AND 499;
```

Con la consulta que escribimos obtenemos estos resultados:

	Titulo	Paginas
1	El visitante	592
2	El Instituto	561
3	Salem's Lot	653
4	Apocalipsis	1152
5	If Cats Disappeared from the World	208

Y, como vemos, ninguno de los libros listados tiene una cantidad de páginas mayor o igual a 400 y menor o igual a 499, que es lo que buscábamos.

Otros operadores

Además de los operadores lógicos y relacionales, el lenguaje SQL dispone de otros tipos de operadores para simplificar operaciones entre expresiones. Algunos con los que trabajaremos más a menudo son:

Operador	Descripción
BETWEEN	Permite comparar entre rangos cerrados [Desde; Hasta]
IN	Permite comparar con un conjunto de valores separados por coma
IS NULL	Permite determinar si un valor es nulo o no
LIKE	Permite la búsqueda con patrones en el valor de comparación

Operador IN

Necesitamos obtener el ID, el título y el año de publicación de aquellos libros que hayan sido publicados en los años 2000, 2010 o 2020.

Podemos escribir esta consulta:

```
SELECT IDLibro, Titulo, AñoPublicacion FROM Libros
WHERE AñoPublicacion = 2000 OR AñoPublicacion = 2010
OR AñoPublicacion = 2020
```

Funciona perfectamente. Pero, tal como vimos con BETWEEN, existe una forma de escribir menos obteniendo los mismos resultados y es con el operador **IN**:

```
SELECT IDLibro, Titulo, AñoPublicacion FROM Libros
WHERE AñoPublicacion IN (2000, 2010, 2020)
```

	IDLibro	Titulo	AñoPublicacion
1	6	La sangre manda	2020
2	30	Todo oscuro, sin estrellas	2010
3	40	La historia de Nintendo	2020
4	44	Confesiones de un chef	2000
5	46	Mientras escribo	2000
6	57	The Making of Prince of Persia	2020

Su sintaxis básica es:

```
SELECT [Columna] FROM [Tabla]
WHERE [Campo] IN / NOT IN (Valor1, Valor2, . . ., ValorN);
```

Ahora listaremos todos los datos de los autores que hayan nacido en los países con ID 'JPN' o 'ARG':

```
SELECT * FROM Autores WHERE IDPais IN ('JPN', 'ARG');
```

	IDAutor	IDPais	Apellidos	Nombres	Pseudonimo
1	1	JPN	Murakami	Haruki	*NULL*
2	5	JPN	Kawamura	Genki	*NULL*
3	21	ARG	Balmaceda	Daniel	*NULL*
4	23	ARG	Bulat	Tomás	*NULL*

Como podemos observar utilizamos IN para un conjunto de datos formado por números y otro formado por cadenas de texto, teniendo la misma efectividad.

Operador LIKE

El operador LIKE es utilizado para comparar cadenas mediante patrones, lo que lo convierte en una herramienta muy poderosa. En muchos casos también es útil para obtener los registros en cuya columna "filtro" aparezca determinada cadena de texto o carácter.

De manera similar a lo que ocurre en las expresiones regulares, LIKE utiliza *comodines* para conformar los patrones de búsqueda. Los comodines son caracteres especiales que "reemplazan" a su vez uno o más caracteres de los datos que queremos recuperar. A continuación echemos un vistazo a algunos de los comodines que podemos usar con LIKE:

Comodín	Descripción	Ejemplo	Resultados
%	Sustituye cero, uno o varios caracteres	WHERE Palabra LIKE 'A%'	'Abel' 'A' 'Ahora'
_	Sustituye un solo carácter	WHERE Palabra LIKE '_aya'	'Baya' 'Haya' ' aya'
[ ]	Coincide con cualquier carácter que se encuentre dentro de los corchetes (sólo SQL Server)	WHERE Palabra LIKE 'So[lns]'	'Sol' 'Son' 'Sos'
[^]	Excluye cualquier carácter que se encuentre dentro de los corchetes (sólo SQL Server)	WHERE Palabra LIKE 'Cua[^k]'	'Cual' 'Cuan' pero no 'Cuak'
[-]	Define un rango de caracteres dentro de los corchetes (sólo SQL Server)	WHERE Palabra LIKE 'B[f-o]t'	'Bit' 'Bht' 'Bot' pero no 'But'

La sintaxis de LIKE es:

```
SELECT * FROM [Tabla] WHERE [Campo_Texto] LIKE / NOT LIKE '[Patron]';
```

Ahora veamos algunos ejemplos.

Ejemplo 1: obtener todos los datos de los libros cuyo título sea 'It' (coincidencia exacta).

```
SELECT * FROM Libros WHERE Titulo LIKE 'It';
```

En este caso en particular, la consulta que escribimos es equivalente a la siguiente:

```
SELECT * FROM Libros WHERE Titulo = 'It';
```

	IDLibro	IDCategoria	Titulo	Descripcion
1	70	2	It	Novela de terror sobre

Ejemplo 2: obtener ID y título de los libros cuyo título contenga la palabra 'historia':

```
SELECT IDLibro, Titulo FROM Libros WHERE Titulo LIKE '%historia%';
```

	IDLibro	Titulo
1	40	La historia de Nintendo
2	45	Historia de las letras, palabras y frases
3	63	Historias inesperadas de la historia argentina
4	69	Historia de las palabras

Aquí notamos que, dado que el comodín **%** puede representar uno, varios o ningún carácter, hay tres resultados que comienzan con la palabra 'Historia', es decir, sin un carácter delante.

Ejemplo 3: obtener ID, título y descripción de los libros cuya descripción contenga la palabra 'historia' pero no su título:

```
SELECT IDLibro, Titulo, Descripcion FROM Libros WHERE Descripcion LIKE '%historia%' AND Titulo NOT LIKE '%historia%';
```

	IDLibro	Titulo	Descripcion
1	10	Apocalipsis	Épica historia postapocalíptica de St
2	13	El lector	Historia de amor y culpa en la Alemar
3	15	Cementerio de animales	Una historia escalofriante sobre la m
4	33	History of food	Historia de la alimentación a lo larg
5	35	La loba de Francia	Historia del reinado de Isabel de Fra
6	37	Los venenos de la corona	Historia de conspiraciones y traicior
7	49	La reina estrangulada	Historia medieval sobre la familia re
8	62	El viejo y el mar	Historia de un viejo pescador y su lu
9	64	Buscando a Papá Noel	Historia navideña con un mensaje conm
10	73	A brief history of the Internet	Historia del desarrollo de Internet.

Podemos observar que se antepone el operador lógico **NOT** a la cláusula LIKE para negar lo que está a continuación.

Ejemplo 4: obtener todos los datos de los libros cuyo título comienza con 'The' y continúa con cualquier carácter o texto:

```
SELECT * FROM Libros WHERE Titulo LIKE 'The%'
```

	IDLibro	IDCategoria	Titulo	Descripcion
1	23	8	The Making of Karateka: Journals 1982-1985	Diario del creador del juego Karat
2	57	8	The Making of Prince of Persia	Diario de desarrollo del clásico

Notemos que luego de la palabra 'The', el título puede continuar con cualquier carácter y/o conjunto de ellos, sin importar su extensión.

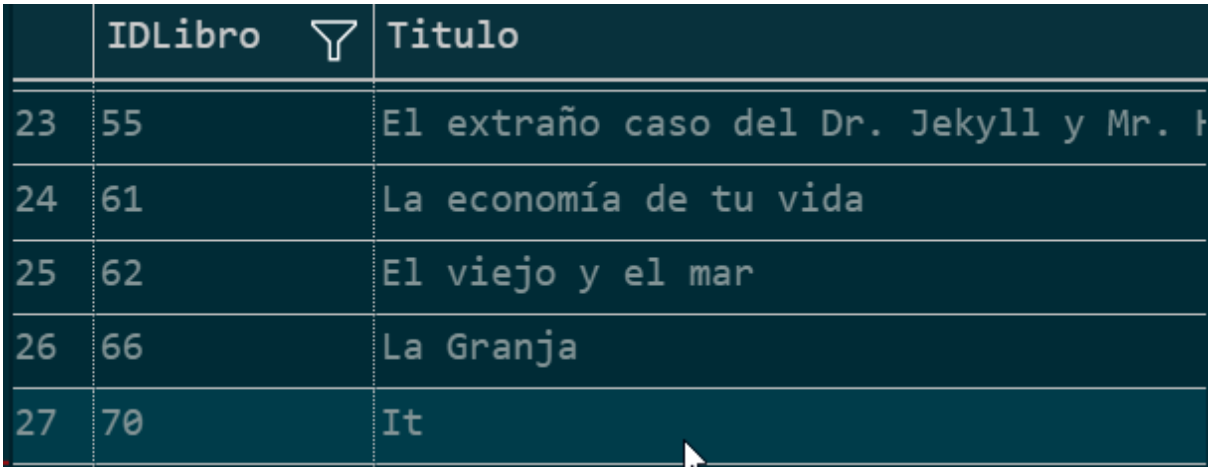
Ejemplo 5: obtener ID y título de los libros cuyo título comienza con una palabra de dos caracteres, luego continúa con un espacio y finaliza con cualquier combinación de letras y números (o ninguno):

```
SELECT IDLibro, Titulo FROM Libros WHERE Titulo LIKE '__ %';
```

	IDLibro	Titulo
1	2	El visitante
2	5	Mr Mercedes
3	6	La sangre manda
4	8	El Instituto
5	12	If Cats Disappeared from the World

Ejemplo 6: igual al ejemplo anterior, pero incluyendo los títulos que tengan exactamente dos caracteres:

```
SELECT IDLibro, Titulo FROM Libros WHERE Titulo LIKE '__ %'
OR Titulo LIKE '__';
```



	IDLibro	Titulo
23	55	El extraño caso del Dr. Jekyll y Mr. H
24	61	La economía de tu vida
25	62	El viejo y el mar
26	66	La Granja
27	70	It

Obtuvimos los mismos resultados que antes, pero además el registro correspondiente al libro con título 'It' que tiene una longitud exacta de dos caracteres.

Operador IS NULL

Tal como lo habíamos anticipado, este operador se utiliza para saber si un dato es desconocido o no, es decir, si en determinada fila y columna que seleccionamos hay ausencia de valor o, por el contrario, sí tienen uno.

Hasta ahora vimos que todo valor conocido, es decir, que se puede representar con un tipo de dato, se puede comparar con otro valor de igual tipo a través de los operadores relacionales. Si les consultara cuál les parece que es el resultado correcto de la siguiente comparación:

```
0 < 100
```

Ninguno de ustedes, ni el gestor de base de datos, tendrían inconveniente en responder que el resultado es **true** (verdadero), ya que 0 es un número menor a 100. Pero, si en lugar de lo anterior les consulto cuál es el resultado de esta expresión:

```
0 < NULL
```

Es decir, preguntamos si 0, que es un valor que conocemos, numérico, es menor a NULL, no será posible determinarlo, ya que NULL representa lo desconocido, la ausencia de valor. Ni siquiera estamos en condiciones de afirmar que en el lugar de NULL hay un número entero, por ejemplo.

La respuesta que el gestor de base de datos daría a esta última consulta sería **NULL**, que es la forma de representar lo que no se conoce. Lo mismo sucedería con las siguientes comparaciones:

Valor 1	Operador	Valor 2	Resultado
50	>	NULL	NULL
50	<>	NULL	NULL
50	=	NULL	NULL
‘Hola mundo’	LIKE	NULL	NULL
‘Hola mundo’	<>	NULL	NULL
GETDATE()	>	NULL	NULL
NULL	=	NULL	NULL
NULL	<>	NULL	NULL

Sin embargo, el lenguaje T-SQL nos provee una manera de saber si en determinada fila y columna (o conjunto de ellas) hay un dato, o bien si el dato está ausente (NULL): mediante el operador **IS NULL**.

Reescribamos algunas de las anteriores comparaciones con el nuevo operador y observemos los resultados:

Valor 1	Operador	Valor 2	Resultado
50	IS	NULL	FALSE
50	IS NOT	NULL	TRUE
‘Hola mundo’	IS	NULL	FALSE
‘Hola mundo’	IS NOT	NULL	TRUE
GETDATE()	IS	NULL	FALSE
GETDATE()	IS NOT	NULL	TRUE
NULL	IS	NULL	TRUE
NULL	IS NOT	NULL	FALSE

Suena un tanto extraño pero tiene sentido: 50 no es NULL, por lo tanto el resultado de la comparación es **verdadero**. Si preguntamos si ‘Hola mundo’ es NULL, la respuesta es **falso**, ya que sí posee un dato, un valor. El resultado que arroja la función GETDATE() no es NULL

(es un valor de tipo DATETIME), con lo cual la comparación entre ambos, en caso de formularla como “es” (GETDATE() IS NULL) devuelve falso, pero si afirmamos “no es” (GETDATE() IS NOT NULL) devuelve verdadero. Y así sucesivamente. También podemos preguntarnos si NULL es NULL (¿*desconocido* es *desconocido*?) lo cual es verdadero. Sería similar a la comparación “50 = 50”, por ejemplo.

Veamos el operador en acción con algunos ejemplos:

Ejemplo 1: todos los datos de los autores que no tengan registrado su apellido (“traducido” a lenguaje SQL, queremos obtener los datos de los autores que en la columna Apellidos tengan NULL):

```
SELECT * FROM Autores WHERE Apellidos IS NULL;
```

	IDAutor	IDPais	Apellidos	Nombres	Pseudonimo	AñoNacimie
1	2	GBR	*NULL*	*NULL*	Bitmap Books	*NULL*

Ejemplo 2: todos los datos de los autores que tengan registrado un seudónimo (es decir, que en la columna Pseudonimo no tengan NULL):

```
SELECT * FROM Autores WHERE Pseudonimo IS NOT NULL;
```

	IDAutor	IDPais	Apellidos	Nombres	Pseudonimo	AñoNacimient
1	2	GBR	*NULL*	*NULL*	Bitmap Books	*NULL*
2	3	USA	King	Stephen	Richard Bachman	1947

Ejemplo 3: todos los datos de los autores que tengan registrado un seudónimo y también su apellido:

```
SELECT * FROM Autores WHERE Pseudonimo IS NOT NULL AND Apellidos IS NOT
NULL;
```

	IDAutor	IDPais	Apellidos	Nombres	Pseudonimo	AñoNacimiento
1	3	USA	King	Stephen	Richard Bachman	1947

En este último ejemplo vimos que, como cualquier otro operador, IS NULL puede formar parte de filtros más complejos, es decir, en la cláusula WHERE podríamos tener varias proposiciones lógicas formadas con IS NULL y combinarlas entre sí con operadores lógicos.

Ordenamiento - Cláusula ORDER BY

Cuando obtenemos más de una fila como resultado de una consulta, y especialmente si obtenemos muchas filas, suele ser importante recibir la información de forma ordenada a fines de hacer una lectura óptima y ágil de la misma.

Para ello contamos con la cláusula **ORDER BY**, que ordena los resultados de una consulta a partir de uno o más criterios que le indiquemos. La estructura de esta expresión es la siguiente:

```
SELECT [Columna1], [Columna2], [Columna3] FROM [Tabla]
ORDER BY [Columna1] [Criterio];
```

Donde el criterio de ordenamiento puede ser:

ASC, para indicar que será ascendente (ej. textos ordenados alfabéticamente, de la A a la Z).

DESC, para indicar que será descendente (ej. fechas ordenadas desde la más reciente a la más antigua).

A continuación veremos el funcionamiento de ORDER BY en algunos ejemplos:

Ejemplo 1: obtener un listado de libros (todos los datos) ordenados alfabéticamente por título:

```
SELECT * FROM Libros ORDER BY Titulo ASC;
```

	IDLibro	IDCategoria	Titulo
1	65	5	1984
2	56	5	20.000 Leguas de viaje submarino
3	73	4	A brief history of the Internet
4	58	11	A Cook's Tour
5	10	2	Apocalipsis

En esta consulta obtuvimos un listado con todos los datos de los libros ordenados alfabéticamente por título, es decir, los que se muestran en primer lugar son aquellos que cuya inicial está primera en orden alfabético (números, luego letras de la A a la Z).

NOTA: en el caso del criterio ascendente, si omitimos la expresión ASC, el gestor de base de datos interpretará que queremos hacer un ordenamiento ascendente. Sin embargo, esta práctica no es recomendable.

Realizaremos la misma consulta del Ejemplo 1 pero mencionando la columna a ordenar de otra forma:

```
SELECT * FROM Libros ORDER BY 3 ASC;
```

	IDLibro	IDCategoria	Titulo
1	65	5	1984
2	56	5	20.000 Leguas de viaje submarino
3	73	4	A brief history of the Internet
4	58	11	A Cook's Tour
5	10	2	Apocalipsis

Ejecutando la consulta observamos que tenemos exactamente los mismos resultados que antes. ¿Cómo ocurrió esto? Ante todo, sabemos que estas dos consultas son equivalentes:

```
SELECT * FROM Libros;
```

```
SELECT IDLibro, IDCategoria, Titulo, Descripcion, AñoPublicacion, Paginas
FROM Libros;
```

Es decir, el símbolo asterisco (*) es lo mismo que escribir una a una todas las columnas de la tabla. Luego, si establecemos ordenamiento a partir de la columna 3, lo que queremos decir es que se ordenen los resultados por la tercera columna que ***seleccionamos*** de la tabla de Libros (CUIDADO: no es lo mismo que referirnos a la tercera columna de la tabla de Libros). Con lo cual, este número representa la posición que tiene la columna dentro del conjunto de columnas seleccionadas.

Ejemplo 2: volvamos al ejemplo de página 8, donde establecemos el valor de las columnas a seleccionar según una condición:

```
SELECT
ISNULL(Apellidos + ', ' + Nombres, Pseudonimo) AS 'Apellidos y Nombres o
Pseudonimo',
AñoNacimiento,
CEILING(AñoNacimiento/100.0) AS Siglo,
CASE WHEN AñoNacimiento IS NULL THEN 'Desconocido'
     WHEN AñoNacimiento < = 1800 THEN 'Muy antiguo'
     WHEN AñoNacimiento < = 1900 THEN 'XIX'
     WHEN AñoNacimiento < = 2000 THEN 'XX'
     ELSE 'XXI' END AS 'Siglo en numeros romanos'
FROM Autores;
```

	Apellidos y Nombres o Pseudonimo	AñoNacimiento	Siglo	Siglo en numeros romanos
1	Murakami, Haruki	1949	20	XX
2	Bitmap Books	*NULL*	*NULL*	Desconocido
3	King, Stephen	1947	20	XX
4	Crichton, Michael	1942	20	XX
5	Kawamura, Genki	1979	20	XX

¿Qué sucedería si necesito ordenar por la columna 'Siglo en numeros romanos', que se calculó a partir de una decisión? ¿Tendré que escribir nuevamente todo el bloque CASE para hacer referencia a esa columna? Bueno, la respuesta es sí y no, porque ORDER BY reconoce los alias que se definieron para las columnas, ya sean calculadas o no:

```
SELECT
ISNULL(Apellidos + ', ' + Nombres, Pseudonimo) AS 'Apellidos y Nombres o
Pseudonimo',
AñoNacimiento,
CEILING(AñoNacimiento/100.0) AS Siglo,
CASE WHEN AñoNacimiento IS NULL THEN 'Desconocido'
     WHEN AñoNacimiento < = 1800 THEN 'Muy antiguo'
     WHEN AñoNacimiento < = 1900 THEN 'XIX'
     WHEN AñoNacimiento < = 2000 THEN 'XX'
     ELSE 'XXI' END AS 'Siglo en numeros romanos'
FROM Autores
ORDER BY [Siglo en numeros romanos] ASC;
```

	Apellidos y Nombres o Pseudonimo	AñoNacimiento	Siglo	Siglo en numeros romanos
1	Bitmap Books	*NULL*	*NULL*	Desconocido
2	Greenberg, Paul	*NULL*	*NULL*	Desconocido
3	Bartlett, Robert	*NULL*	*NULL*	Desconocido
4	Pérez Rodríguez, Uxio	*NULL*	*NULL*	Desconocido
5	Alighieri, Dante	1265	13	Muy antiguo
6	Saint-Exupéry, Antoine de	1900	19	XIX
7	Carroll, Lewis	1832	19	XIX
8	Amicis, Edmundo de	1846	19	XIX
9	Verne, Julio	1828	19	XIX
10	Stoker, Bram	1847	19	XIX
11	Stevenson, Robert Louis	1850	19	XIX
12	Hemingway, Ernest	1899	19	XIX
13	Murakami, Haruki	1949	20	XX
14	Cline, Ernest	1972	20	XX
15	King, Stephen	1947	20	XX

Lo mismo podemos hacer para la columna ‘Siglo’, en forma descendente:

```
SELECT
ISNULL(Apellidos + ', ' + Nombres, Pseudonimo) AS 'Apellidos y Nombres o
Pseudonimo',
AñoNacimiento,
CEILING(AñoNacimiento/100.0) AS Siglo,
CASE WHEN AñoNacimiento IS NULL THEN 'Desconocido'
     WHEN AñoNacimiento < = 1800 THEN 'Muy antiguo'
     WHEN AñoNacimiento < = 1900 THEN 'XIX'
     WHEN AñoNacimiento < = 2000 THEN 'XX'
     ELSE 'XXI' END AS 'Siglo en numeros romanos'
FROM Autores
ORDER BY Siglo DESC;
```

	Apellidos y Nombres o Pseudonimo	AñoNacimiento	Siglo	Siglo en numeros romanos
1	King, Stephen	1947	20	XX
2	Crichton, Michael	1942	20	XX
3	Kawamura, Genki	1979	20	XX
4	Schlink, Bernhard	1944	20	XX
5	Cline, Ernest	1972	20	XX
6	Murakami, Haruki	1949	20	XX
7	Mechner, Jordan	1964	20	XX
8	Druon, Maurice	1918	20	XX
9	Eco, Umberto	1932	20	XX
10	Bourdain, Anthony	1956	20	XX
11	Balmaceda, Daniel	1962	20	XX
12	Bulat, Tomás	1964	20	XX
13	Yves, Cohat	1953	20	XX
14	Collet, Anne	1954	20	XX
15	Carroll, Lewis	1832	19	XIX
16	Saint-Exupéry, Antoine de	1900	19	XIX
17	Amicis, Edmundo de	1846	19	XIX
18	Verne, Julio	1828	19	XIX
19	Stoker, Bram	1847	19	XIX
20	Stevenson, Robert Louis	1850	19	XIX

Ejemplo 3: seleccionar ID de libro, ID de categoría, descripción de los libros, ordenados alfabéticamente por título.

```
SELECT IDLibro, IDCategoria, Descripcion FROM Libros ORDER BY Titulo ASC;
```

	IDLibro	IDCategoria	Descripcion
1	65	5	Distopía sobre un futuro totalitario.
2	56	5	Aventura de ciencia ficción con el Capitán Nemo.
3	73	4	Historia del desarrollo de Internet.
4	58	11	Relatos de viajes y gastronomía de Anthony Bourdai…
5	10	2	Épica historia postapocalíptica de Stephen King.

Aunque resulte un tanto engorroso, si comparamos el orden de estos resultados con el orden de los resultados del ejemplo 1, notaremos que es exactamente igual. Esto demuestra que no es necesario seleccionar la columna por la cual se realizará el ordenamiento (‘Titulo’ en este caso). En otras palabras, la columna ‘Titulo’ no está incluida en el SELECT y de todos modos logramos ordenar el listado por dicha columna.

En los siguientes dos ejemplos veremos que todas las columnas admiten ordenamiento, es decir, no sólo las de tipo texto y numéricas. También de fecha/hora, entre otras.

Ejemplo 4: obtener el ID, el título y la cantidad de páginas de los libros ordenados por longitud (del más extenso al más corto):

```
SELECT IDLibro, Titulo, Paginas FROM Libros ORDER BY Paginas DESC;
```

	IDLibro	Titulo	Paginas
1	10	Apocalipsis	1152
2	70	It	1138
3	38	La cúpula	1074
4	20	Needful things	736
5	33	History of food	728

Ejemplo 5: obtener todos los datos de las lecturas de libros, ordenados por fecha en forma descendente (de la lectura más reciente a la más antigua):

```
SELECT * FROM Lecturas ORDER BY FechaHora DESC;
```

	IDLectura	IDUsuario	IDLibro	IDMedio	FechaHora	TiempoEnMinutos
1	9060	60	15	4	2024-05-30	900
2	9059	60	15	4	2024-05-29	750
3	9058	60	15	4	2024-05-28	700
4	9057	60	15	4	2024-05-27	710
5	9056	60	15	4	2024-05-25	780

Ejemplo 5: obtener todos los datos de los libros, ordenados por año de publicación en forma descendente (del más reciente al más antiguo). En caso de haber más de un libro con el mismo año de publicación, ordenarlos por título alfabéticamente, es decir, de forma ascendente (A-Z).

```
SELECT * FROM Libros ORDER BY AñoPublicacion DESC, Titulo ASC;
```

	IDLibro	IDCategoria	Titulo	Descripcion	AñoPublicacion	Paginas
1	1	2	Holly	Thriller sobrenatural de Stephen King con un perso…	2023	432
2	23	8	The Making of Karateka: Journals 1982-1985	Diario del creador del juego Karateka.	2022	320
3	14	2	Después	Novela de terror sobrenatural de King.	2021	272
4	41	9	Visual Studio Code Succintly	Guía práctica sobre Visual Studio Code.	2021	128
5	40	8	La historia de Nintendo	Historia de la compañía Nintendo.	2020	400
6	6	2	La sangre manda	Colección de cuatro relatos de King con su toque c…	2020	448
7	57	8	The Making of Prince of Persia	Diario de desarrollo del clásico videojuego.	2020	336
8	8	2	El Instituto	Thriller de terror sobre niños con poderes psíquic…	2019	561
9	2	2	El visitante	Novela de terror y misterio sobre un crimen inexpl…	2018	592
10	45	6	Historia de las letras, palabras y frases	Estudio sobre el origen del lenguaje.	2018	390

En este ejemplo vimos que se puede incluir más de un criterio de ordenamiento, y que los mismos se escriben según su prioridad. Los libros con título 'Despues' y 'Visual Studio Code Succintly' fueron ambos publicados en el año 2021, por ello aparecen entre los primeros del listado. Pero, a la hora de decidir, entre ellos dos, cuál mostrar primero, se muestra 'Despues' antes que el otro registro porque lo precede en orden alfabético.

Observemos lo que sucede con los libros publicados en 2020, de no utilizar el segundo criterio:

```
SELECT * FROM Libros ORDER BY AñoPublicacion DESC;
```

	IDLibro	IDCategoria	Titulo	Descripcion	AñoPublicacion	Paginas
1	1	2	Holly	Thriller sobrenatural de Stephen King con un perso…	2023	432
2	23	8	The Making of Karateka: Journals 1982-1985	Diario del creador del juego Karateka.	2022	320
3	14	2	Después	Novela de terror sobrenatural de King.	2021	272
4	41	9	Visual Studio Code Succintly	Guía práctica sobre Visual Studio Code.	2021	128
5	40	8	La historia de Nintendo	Historia de la compañía Nintendo.	2020	400
6	57	8	The Making of Prince of Persia	Diario de desarrollo del clásico videojuego.	2020	336
7	6	2	La sangre manda	Colección de cuatro relatos de King con su toque c…	2020	448
8	8	2	El Instituto	Thriller de terror sobre niños con poderes psíquic…	2019	561
9	2	2	El visitante	Novela de terror y misterio sobre un crimen inexpl…	2018	592
10	45	6	Historia de las letras, palabras y frases	Estudio sobre el origen del lenguaje.	2018	390

TOP - Rankeo

La cláusula **TOP** permite limitar la cantidad de registros que devolverá una consulta. Es decir, quedarnos sólo con un determinado número de filas de todos los que podríamos obtener como resultado.

Por ejemplo:

```
SELECT TOP (1) * FROM Libros;
```

Esta consulta devolverá únicamente el primer registro de la tabla de Libros, con un ordenamiento arbitrario. Aquí, el gestor de base de datos, al no recibir una directiva de ordenamiento, "decidió" que dicho criterio sea el ID de libro:

	IDLibro	IDCategoria	Titulo	Descripcion	AñoPublicacion	Paginas
1	1	2	Holly	Thriller sobrenatural de Stephen King con un perso…	2023	432

Sin embargo, hemos incluido a TOP dentro de la sección de ordenamiento ya que realmente cobra sentido cuando se aplica a una consulta que está ordenada de alguna manera. Su sintaxis es la siguiente:

```
SELECT TOP ([Cantidad_de_filas]) [Columnas] FROM [Tabla] ORDER BY [Columna]
[CRITERIO];
```

Veamos como ejemplo un listado del libro más extenso, es decir, el que posee mayor cantidad de páginas:

```
SELECT TOP(1) * FROM Libros ORDER BY Paginas DESC;
```

	IDLibro	IDCategoria	Titulo	Descripcion	AñoPublicacion	Paginas
1	10	2	Apocalipsis	Épica historia postapocalíptica de Stephen King.	1978	1152

Si solamente necesitamos obtener el ID, el título y la cantidad de páginas de este libro más extenso:

```
SELECT TOP(1) IDLibro, Titulo, Paginas FROM Libros ORDER BY Paginas DESC;
```

	IDLibro	Titulo	Paginas
1	10	Apocalipsis	1152

Por las características de estos listados podemos decir que la combinación de TOP y ORDER BY nos permite realizar algo así como un **ranking.**

Analicemos ahora esta consulta, que nos devuelve los diez registros correspondientes a los libros más “nuevos” (o sea, con año de publicación más reciente):

```
SELECT TOP(10) IDLibro, Titulo, AñoPublicacion FROM Libros ORDER BY
AñoPublicacion DESC;
```

	IDLibro	Titulo	AñoPublicacion
1	1	Holly	2023
2	23	The Making of Karateka: Journals 1982-1985	2022
3	14	Después	2021
4	41	Visual Studio Code Succintly	2021
5	6	La sangre manda	2020
6	40	La historia de Nintendo	2020
7	57	The Making of Prince of Persia	2020
8	8	El Instituto	2019
9	2	El visitante	2018
10	45	Historia de las letras, palabras y frases	2018

Ahora, si en lugar de los primeros diez registros necesitamos los cinco primeros, vemos que en las posiciones 5, 6 y 7 están "empatados" tres libros con el año de publicación 2020. Entonces, ¿cómo *desempatamos*? Es decir, ¿cómo decidimos cuál de esos tres registros debería ser el quinto y último, descartando los dos restantes?

Uno de los criterios para resolver este tipo de situaciones es la cláusula WITH TIES, que incluye a los registros que cumplen un mismo criterio de ordenamiento pero son excluidos con el TOP "a secas":

```
SELECT TOP(5) WITH TIES IDLibro, Titulo, AñoPublicacion FROM Libros ORDER BY AñoPublicacion DESC;
```

	IDLibro	Titulo	AñoPublicacion
1	1	Holly	2023
2	23	The Making of Karateka: Journals 1982-1985	2022
3	14	Después	2021
4	41	Visual Studio Code Succintly	2021
5	40	La historia de Nintendo	2020
6	57	The Making of Prince of Persia	2020
7	6	La sangre manda	2020

En efecto, aquí podemos ver que la inclusión de WITH TIES hace que todos los libros con año de publicación 2020 formen parte de la selección. Sí, son dos registros más de los que quería obtener inicialmente, pero "comparten" el quinto lugar.

Eliminar repeticiones con DISTINCT

Suponiendo que deseamos obtener un listado con los años de publicación de los libros, ordenado del más reciente al más antiguo:

```
SELECT AñoPublicacion FROM Libros ORDER BY AñoPublicacion DESC;
```

	AñoPublicacion
1	2023
2	2022
3	2021
4	2021
5	2020
6	2020
7	2020
8	2019
9	2018
10	2018

Sin embargo, la utilidad de esta consulta es bastante cuestionable, ya que hay muchos resultados repetidos.

Mediante el uso de la cláusula **DISTINCT** podemos excluir de la consulta las filas repetidas:

```
SELECT DISTINCT AñoPublicacion FROM Libros;
```

	AñoPublicacion
1	1195
2	1320
3	1554
4	1864
5	1870
6	1886
7	1887
8	1891
9	1897
10	1908

Quizá no podremos apreciarlo en este conjunto de resultados, pero, además de quitar las filas repetidas, DISTINCT muestra los resultados de forma ordenada (ascendente).

La estructura de DISTINCT es la siguiente:

```
SELECT DISTINCT [Columna1] FROM [Tabla];
```

También podríamos necesitar un listado donde no se repita un mismo conjunto de columnas, por ejemplo, por cada categoría de libros, queremos obtener los años de publicación de los mismos, sin que se repita la categoría ni el año de publicación dentro de cada una de ellas:

```
SELECT DISTINCT IDCategoria, AñoPublicacion FROM Libros ORDER BY IDCategoria ASC, AñoPublicacion ASC;
```

	IDCategoria	AñoPublicacion
1	1	1195
2	1	1320
3	1	1554
4	1	1864
5	1	1886
6	1	1887
7	1	1908
8	1	1936
9	1	1941
10	1	1952
11	1	1955
12	1	1956
13	1	1957
14	1	1959
15	1	1980
16	1	1995
17	1	1996
18	1	1999
19	1	2012
20	2	1886

Nuevamente, aclaramos que, al tratarse de una consulta de selección, los datos originales de las tablas no serán modificados. Es decir, DISTINCT elimina las filas duplicadas de un conjunto de resultados, pero no de las tablas.