

## Funciones de resumen

Las funciones de resumen permiten realizar cálculos sobre un conjunto de datos, devolviendo un único valor que los resume. Son similares a las funciones de las que disponemos en un software de planilla de cálculo: con ellas podemos contar registros, obtener máximos y mínimos, o bien sumar, promediar, etc. a los valores dentro de un conjunto de datos. Además, similar a la funcionalidad de las planillas de cálculo, también podemos agrupar registros bajo un determinado criterio, y una vez agrupados, aplicar filtros.

Las funciones que figuran en la tabla que se encuentra a continuación son las que utilizaremos en este apunte y en el curso, ya que en general son las más utilizadas:

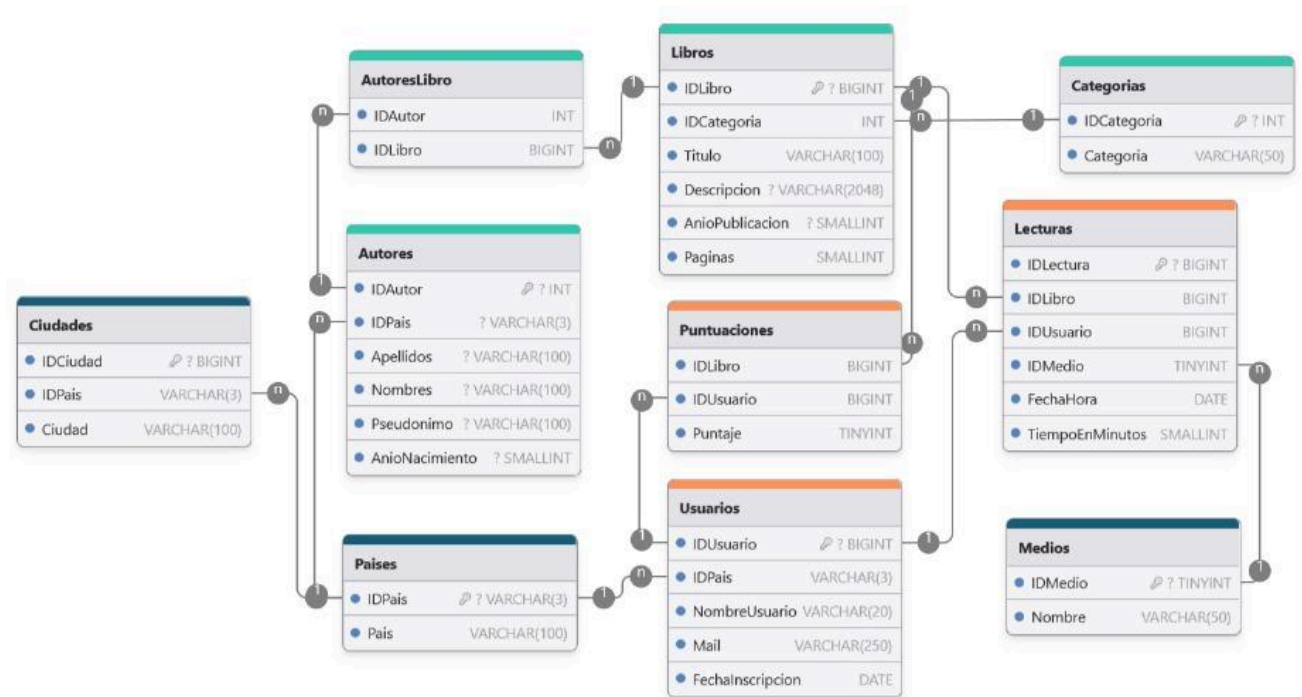
Función	Descripción	Ejemplo
COUNT	Cuenta el número de elementos no nulos de una columna (cuando su parámetro es una columna) o el número de filas cuando su parámetro es (*)	<b>SELECT COUNT (Descripcion) FROM Tabla</b> (cuenta las filas de <i>Tabla</i> donde la columna <i>Descripcion</i> no contiene NULL)  <b>SELECT COUNT (*) FROM Tabla</b> (cuenta todas las filas de <i>Tabla</i> )
SUM	Suma los valores de una columna numérica	<b>SELECT SUM(Importe) FROM Tabla</b> (suma todos los valores no nulos de la columna <i>Importe</i> de <i>Tabla</i> )
AVG	Promedia los valores de una columna numérica	<b>SELECT AVG(Edad) FROM Tabla</b> (promedia todos los valores no nulos de la columna <i>Edad</i> de <i>Tabla</i> )
MIN	Obtiene el valor mínimo de una columna	<b>SELECT MIN(Palabra) FROM Tabla</b> (obtiene el mínimo del campo <i>Palabra</i> de <i>Tabla</i> . Al ser una columna de tipo texto, el criterio para hallar el valor mínimo será el orden alfabético)
MAX	Obtiene el valor máximo de una columna	<b>SELECT MAX(FechaNacimiento) FROM Tabla</b> (obtiene el máximo del campo <i>FechaNacimiento</i> de <i>Tabla</i> . Al ser una columna de tipo fecha, para hallar el valor máximo se considerará un ordenamiento del valor más antiguo al más reciente)

El uso de funciones de resumen puede ser identificado en enunciados tales como:

"Obtener el **promedio** de la **cantidad de páginas** de los **libros** cuya **categoría es 'Novela'**"

Donde **promedio** es la **función de resumen**, la **cantidad de páginas** son los **datos a resumir**, y **libros cuya categoría es "Novela"** es el **conjunto de registros** sobre el que se **realizarán los cálculos** (es decir, la condición sobre los registros que se resumirán).

Veamos con ejemplos las particularidades de cada función. Como hicimos anteriormente, trabajaremos con la base de datos **Libreria**. Para una mejor comprensión de los enunciados, adjuntamos el DER:



## COUNT - Contar registros

### Contar todos los registros

Escribiremos una consulta para obtener la cantidad de libros que hay en total, pero antes realizaremos una consulta básica para ver "gráficamente" cuántos registros contiene la tabla Libros (NOTA: se muestra un conjunto acotado de resultados):

```
SELECT * FROM Libros;
```

	IDLibro ↕	IDCategoria ↕	Titulo ↕	Descripcion ↕	AñoPublicacion ↕	Paginas
76	76	2	La expedición	Novela de terror y ciencia ficción.	2018	320
77	77	1	Corazones en la Atlántida	Colección de relatos interconectados de King.	1999	672
78	78	3	Los ojos del dragón	Novela de fantasía ambientada en un reino medieval.	1987	376
79	79	1	El Lazarillo de Tormes	Relata las desventuras que un joven de origen humi...	1554	336
80	80	1	Cantar de Mio Cid	El Cantar de mio Cid es un poema épico anónimo del...	1195	74

(80 filas afectadas)

Con esta observación sabemos que la función COUNT aplicada a la tabla Libros debe devolver el valor 80. Vamos a comprobarlo:

```
SELECT COUNT(*) AS 'Cantidad de libros' FROM Libros;
```

	Cantidad de libros ↕
1	80

## Contar todos los registros que cumplan una condición determinada

En este ejemplo, buscamos la cantidad de autores de nacionalidad estadounidense (es decir, Pais = "Estados Unidos") que hayan nacido después del año 1950.

Aquí, el COUNT se aplica sobre los resultados de la unión de dos tablas (Autores y Países) y se limita a contar la cantidad de registros que cumplen las dos condiciones que incorporamos:

```
SELECT COUNT(A.IDAutor) AS 'Cantidad de autores'
FROM Autores AS A
INNER JOIN Países AS P ON A.IDPaís = P.IDPaís
WHERE P.País = 'Estados Unidos'
AND A.AñoNacimiento > 1950;
```

	Cantidad de autores
1	3

Tal como hicimos antes, realicemos una consulta sin el COUNT para corroborar los resultados:

```
SELECT A.*
FROM Autores AS A
INNER JOIN Países AS P ON A.IDPaís = P.IDPaís
WHERE P.País = 'Estados Unidos'
AND A.AñoNacimiento > 1950;
```

	IDAutor	IDPaís	Apellidos	Nombres	Pseudonimo	AñoNacimiento
1	9	USA	Cline	Ernest	NULL	1972
2	13	USA	Mechner	Jordan	NULL	1964
3	20	USA	Bourdain	Anthony	NULL	1956

Gracias a que son solamente tres registros, podemos corroborar simple y rápidamente que los tres autores informados son de nacionalidad estadounidense y sus años de nacimiento son posteriores a 1950.

También vale aclarar que, en este caso, es lo mismo contar los registros de la columna **IDAutor** de **Autores** (**COUNT A.IDAutor**) que contar la cantidad de filas de Autores (**COUNT \***), ya que no podría existir un registro en Autores donde IDAutor sea NULL (por ser su clave primaria).

## Contar sólo los registros con valores no nulos

Vamos con otro listado. Queremos saber la cantidad de autores que tienen registrado su apellido en la tabla Autores.

Existen dos maneras de resolver. La primera es con una condición (WHERE): contamos la cantidad de registros de Autores donde la columna Apellidos no tiene el valor NULL. Para diferenciar las consultas colocaremos un alias a la columna obtenida:

```
SELECT COUNT(*) AS 'Cantidad de autores_V1' FROM Autores
WHERE Apellidos IS NOT NULL;
```

	Cantidad de autores_V1
1	25

La otra manera de averiguarlo es, como vimos antes, colocando como parámetro de COUNT el nombre de la columna cuyos valores no nulos queremos contar:

```
SELECT COUNT(Apellidos) AS 'Cantidad de autores_V2' FROM Autores;
```

	Cantidad de autores_V2	↕	🔍
1	25		

Si bien la consulta no es compleja, podemos notar que esta última manera de escribirla resulta más breve y concisa.

## Contar los registros distintos

En este caso queremos obtener la cantidad de usuarios distintos que leyeron el libro con título "It". En primer lugar podemos hacer una consulta básica para llegar al resultado simplemente observando las filas (recomendamos tener a mano el DER):

```
SELECT DISTINCT (NombreUsuario) FROM Usuarios U
INNER JOIN Lecturas LE ON U.IDUsuario = LE.IDUsuario
INNER JOIN Libros LI ON LE.IDLibro = LI.IDLibro
WHERE LI.Titulo = 'It';
```

	NombreUsuario	↕	🔍
1	AuroraRomano		
2	EugeniaG		
3	SatoshiNakamura		

Lo que hicimos en esta consulta, que arrojó un resultado de 3 usuarios, fue vincular a los usuarios con el libro de título "It". Como habíamos visto anteriormente, al tener una relación de muchos a muchos (usuarios y libros) no tenemos posibilidad de una unión directa, sino que debemos pasar a través de la tabla Lecturas.

Si modificamos ligeramente la consulta anterior para resolver lo solicitado, tendríamos que escribir lo siguiente:

```
SELECT COUNT(DISTINCT U.NombreUsuario) AS CantidadDeUsuarios
FROM Usuarios U
INNER JOIN Lecturas LE ON U.IDUsuario = LE.IDUsuario
INNER JOIN Libros LI ON LE.IDLibro = LI.IDLibro
WHERE LI.Titulo = 'It';
```

	CantidadDeUsuarios
1	3

## Resumen de COUNT

- Cuando utilizamos **COUNT (\*)**, obtenemos el **número de filas** de una selección.
- Cuando utilizamos **COUNT (Columna)**, obtenemos el **número de elementos no nulos** de *Columna*.
- Cuando utilizamos **COUNT (DISTINCT Columna)**, obtenemos el **número de elementos no nulos y sin repeticiones** de *Columna*.
- La función **COUNT** siempre devuelve un número entero positivo. Si no hay registros para contar, devuelve 0 (cero).

## SUM - Sumar valores

Haremos una consulta donde obtendremos la sumatoria del tiempo de lectura en minutos en el año 2010, expresado en horas y minutos.

Para empezar, sabemos que el tiempo de lectura en minutos lo obtenemos de la tabla Lecturas (columna TiempoEnMinutos). En cuanto al año, surge de la misma tabla, de la columna FechaHora. Veamos primero cómo obtener ambos valores:

```
SELECT SUM(LE.TiempoEnMinutos)
FROM Lecturas LE
WHERE YEAR(LE.FechaHora) = 2010;
```

	(Sin nombre de columna)
1	3354

La función **YEAR**, como su nombre lo indica, extrae el año de un parámetro de tipo DATE o DATETIME (entre otros tipos de dato de fecha y hora), y lo devuelve como un número entero. Partiendo de este resultado, ya obtuvimos la cantidad total de minutos del año 2010, que es 3354. Primero realicemos los cálculos “a mano” para saber de antemano a cuántas horas y minutos equivale:

- Para obtener las horas, ya que 1 hora tiene 60 minutos, hacemos la división entera entre 3354 y 60: da 55.

- Multiplicando 55 horas por 60 minutos, obtendremos la cantidad de minutos que corresponden a una cantidad entera de horas: 3300 minutos.
- Finalmente, restando esta cantidad de minutos al total, nos da como resultado un “sobrante” de 54 minutos. De esa manera llegamos a la conclusión de que el resultado que debemos mostrar es: **55 horas 54 minutos**.

Ahora sí, veamos cómo resolverlo con T-SQL. Vamos a realizar unas operaciones muy similares a las que hicimos antes: para dividir utilizaremos el mismo operador que ya conocemos de materias anteriores, que es el de la **división entera (/)**:

```
SELECT SUM(LE.TiempoEnMinutos) / 60 AS Horas
FROM Lecturas LE
WHERE YEAR(LE.FechaHora) = 2010;
```

	Horas
1	55

Debemos tener cuidado con la ubicación de los paréntesis ya que la división por 60 se debe hacer sobre la sumatoria del campo TiempoEnMinutos. Si lo escribiéramos así:

```
SELECT SUM(LE.TiempoEnMinutos / 60) AS Horas
```

Tendríamos un resultado incorrecto.

Ahora nos queda calcular el resto de la división entera entre 3354 y 60, y para ello también haremos uso de un operador ya conocido: el operador de **resto (%)**:

```
SELECT SUM(LE.TiempoEnMinutos) / 60 AS Horas,
SELECT SUM(LE.TiempoEnMinutos) % 60 AS Minutos
FROM Lecturas LE
WHERE YEAR(LE.FechaHora) = 2010;
```

	Horas	Minutos
1	55	54

Y, por fin, dimos con el resultado esperado.

## Sumatoria de valores nulos

Dejaremos por un momento la base de datos Libreria y trabajaremos con una base de datos que contiene los registros de temperaturas máximas y mínimas en diferentes puntos de Argentina, dentro del período de los últimos 365 días. En particular trabajaremos con la tabla **Temperaturas**; veamos el DER:

Temperaturas	
IDMedicion	BIGINT
fecha	DATE
temperatura_maxima	? DECIMAL(5, 2)
temperatura_minima	? DECIMAL(5, 2)
observatorio	VARCHAR(150)

Brevemente, podemos comentar que la tabla posee un campo **IDMedicion** que es su clave primaria, de tipo BIGINT y autoincremental (comienza en 1 e incrementa de a 1). Luego tenemos la **fecha** en formato DATE, las **temperaturas máximas y mínimas** representadas por sus campos homónimos (en formato DECIMAL con 5 dígitos en total y 2 dígitos para la parte decimal), y finalmente el nombre del **observatorio** donde se realizó la medición, formado por un VARCHAR de 150 caracteres máximo. A excepción de las temperaturas, todas las columnas deben contener un valor no nulo. Un valor NULL en la columna *temperatura\_maxima* o en la columna *temperatura\_minima*, por supuesto que no significa que en dicha medición la ciudad no “tenía temperatura”, sino que el observatorio no ha podido registrarla por algún motivo (un error en los sensores, por ejemplo).

Ahora bien, realicemos un listado con la sumatoria de las temperaturas mínimas en los observatorios cuyo nombre comienza con “V”, el día 13/04/2025. Primero consultamos cuántas filas tiene la tabla para las mediciones de ese día:

```
SELECT COUNT(*) AS CantidadDeRegistros FROM Temperaturas
WHERE fecha = '2025-04-13';
```



	CantidadDeRegistros
1	116

La tabla tiene 116 registros para esa fecha. Ahora, cuáles de ellos corresponden a observatorios cuyo nombre comienza con la letra “V”:

```
SELECT * FROM Temperaturas
WHERE fecha = '2025-04-13' AND observatorio LIKE 'V%';
```

	IDMedicion	fecha	temperatura_maxima	temperatura_minima	observatorio
1	110	2025-04-13	23.50	7.00	VENADO TUERTO AERO
2	111	2025-04-13	NULL	NULL	VICTORICA
3	112	2025-04-13	18.50	9.40	VIEDMA AERO
4	113	2025-04-13	24.10	13.10	VILLA DE MARIA DEL RIO SECO
5	114	2025-04-13	24.50	9.80	VILLA DOLORES AERO
6	115	2025-04-13	21.40	7.40	VILLA GESELL AERO
7	116	2025-04-13	25.80	3.20	VILLA REYNOLDS AERO

Estos son los 7 registros que pertenecen a las mediciones de los observatorios que comienzan con “V” el día 13/04/2025. Como podemos notar, el observatorio “Victorica” no ha registrado temperatura máxima ni mínima para ese día.

Hagamos ahora la sumatoria de las temperaturas mínimas de ese conjunto de registros:

```
SELECT SUM(temperatura_minima) AS SumaTempMin FROM Temperaturas
WHERE fecha = '2025-04-13' AND observatorio LIKE 'V%';
```

	SumaTempMin
1	49.90

El resultado es fácil de corroborar ya que la suma no abarca muchas filas.

Veamos otro caso. Vamos a listar todas las mediciones del día 13/04/2025 ordenadas por la temperatura mínima de forma ascendente:

```
SELECT * FROM Temperaturas
WHERE fecha = '2025-04-13'
ORDER BY temperatura_minima ASC;
```

Observemos un fragmento de resultados con los 20 primeros:

	IDMedicion ↕	fecha ↕	temperatura_maxima ↕	temperatura_minima ↕	observatorio ↕
1	15	2025-04-13	22.00	NULL	CAMPO DE MAYO AERO
2	40	2025-04-13	NULL	NULL	JACHAL
3	42	2025-04-13	22.40	NULL	JUJUY U N
4	70	2025-04-13	NULL	NULL	PERITO MORENO AERO
5	111	2025-04-13	NULL	NULL	VICTORICA
6	5	2025-04-13	-19.00	-25.10	BASE BELGRANO II
7	8	2025-04-13	3.60	-5.00	BASE MARAMBIO
8	82	2025-04-13	10.60	-4.80	RIO GALLEGOS AERO
9	7	2025-04-13	7.10	-4.20	BASE ESPERANZA
10	94	2025-04-13	11.50	-3.80	SANTA CRUZ AERO
11	83	2025-04-13	11.50	-2.80	RIO GRANDE B.A.
12	10	2025-04-13	0.50	-2.10	BASE SAN MARTIN
13	19	2025-04-13	15.50	-1.90	CHAPELCO AERO
14	9	2025-04-13	0.60	-1.80	BASE ORCADAS
15	90	2025-04-13	11.50	-1.60	SAN JULIAN AERO
16	6	2025-04-13	2.50	-1.50	BASE CARLINI (EX JUBANY)
17	28	2025-04-13	15.40	1.40	EL BOLSON AERO
18	51	2025-04-13	16.00	1.40	MAQUINCHAO
19	50	2025-04-13	18.40	1.60	MALARGUE AERO
20	4	2025-04-13	16.50	1.60	BARIOCHE AERO

Como era de esperarse, hay registros que no tienen mediciones de temperatura, mientras que otros tienen un valor “negativo” (temperaturas bajo cero). Realicemos la suma de las temperaturas (máximas por un lado, mínimas por el otro) de aquellos registros que tienen ID de medición igual a 15, 40, 42, 70 o 111:

```
SELECT SUM(temperatura_minima) AS SumaTempMin,
SUM(temperatura_maxima) AS SumaTempMax
FROM Temperaturas
WHERE fecha = '2025-04-13' AND IDMedicion IN (15, 40, 42, 70, 111);
```

	SumaTempMin ↕	SumaTempMax ↕
1	NULL	44.40

El resultado de la suma de las temperaturas máximas de estos cinco registros, nuevamente, es fácil de chequear, pero con respecto al resultado de la sumatoria de las temperaturas mínimas, lo más lógico era esperar que sea igual a cero. ¿Es correcto que una suma dé como resultado NULL? Sí, absolutamente: cuando no hay elementos para procesar (es decir, la columna a sumar está completamente rellena con NULL) la función SUM devuelve NULL, y es un error suponer que dará cero como resultado, como pasaba con COUNT.

Aunque no tiene sentido en este contexto, ya que 0° es una temperatura válida y no podemos asegurar que la suma de valores que son todos desconocidos (NULL) sea igual a cero, por lo general no queremos que una aplicación que consuma estos datos obtenga como retorno un NULL. Entonces volvemos a un concepto que ya aprendimos antes, que es **ISNULL**. Le pediremos a esta consulta que, en caso de no tener registros para procesar, devuelva 0 como resultado de la sumatoria:

```
SELECT
ISNULL(SUM(temperatura_minima), 0) AS SumaTempMin,
ISNULL(SUM(temperatura_maxima), 0) AS SumaTempMax
FROM Temperaturas
WHERE fecha = '2025-04-13' AND IDMedicion IN (15, 40, 42, 70, 111);
```

	SumaTempMin	↑↓🔍	SumaTempMax	↑↓🔍
1	0.00		44.40	

## Resumen de SUM

- **SUM** se utiliza para sumar valores pertenecientes a una columna numérica (condición excluyente). Por ende, el resultado obtenido también será un número.
- Cuando escribimos **SUM(Columna)**, obtenemos la sumatoria de todos los valores no nulos de *Columna*.
- Existe un caso excepcional en el que **SUM** no devuelve un número sino NULL: cuando todos los valores a procesar son nulos (es decir, cuando no hay valores por sumar en la columna especificada).

## AVG - Promediar valores

La función **AVG** (de **average**), utilizada para calcular el promedio de un conjunto de valores numéricos, funciona de una manera muy similar a SUM.

Comencemos con un ejemplo de la base de datos Librería: queremos obtener el promedio de cantidad de páginas de los libros cuya categoría sea "Terror".

Revisando el DER observamos que la cantidad de páginas la obtenemos de la tabla **Libros** (campo **Paginas**) y el nombre de la categoría, de la tabla **Categorias** (campo **Categoria**). Estas dos tablas deben ser previamente relacionadas mediante el campo **IDCategoria** que es clave foránea en Libros.

```
SELECT
AVG(LI.Paginas) AS PromedioPaginas
FROM Libros AS LI
INNER JOIN Categorias AS C ON LI.IDCategoria = C.IDCategoria
WHERE Categoria = 'Terror';
```

	PromedioPaginas
1	473

Al ejecutar la consulta, obtenemos el promedio de páginas de los libros de la categoría “Terror”. Sin embargo, tratándose de un promedio, sería esperable que el resultado no sea un número entero. Entonces, si queremos darle mayor precisión, multiplicaremos el número que representa la cantidad de páginas por **1.0**. Esto no hará modificaciones en el resultado excepto por la cantidad de decimales que obtendremos, ya que “convertimos” a la cantidad de páginas, de entero a decimal:

```
SELECT
AVG(LI.Paginas * 1.0) AS PromedioPaginas,
FROM Libros AS LI
INNER JOIN Categorias AS C ON LI.IDCategoria = C.IDCategoria
WHERE Categoria = 'Terror';
```

	PromedioPaginas
1	473.827586

Con este detalle, podemos comprobar que en verdad el promedio de cantidad de páginas se aproxima más a 474 que a 473. De todos modos, para expresar un promedio, lo haremos sin redondear, a menos que se especifique lo contrario.

Una forma de corroborar el resultado sería sumar la cantidad de páginas de todos los libros de categoría “Terror” y luego dividir por la cantidad de registros:

```
SELECT SUM(LI.Paginas) AS SumaPag
FROM Libros AS LI
INNER JOIN Categorias AS C ON LI.IDCategoria = C.IDCategoria
WHERE Categoria = 'Terror';
```

	SumaPag
1	13741

```
SELECT COUNT(LI.Paginas) AS CuentaReg
FROM Libros AS LI
INNER JOIN Categorías AS C ON LI.IDCategoría = C.IDCategoría
WHERE Categoría = 'Terror';
```

	CuentaReg
1	29

```
SELECT (13741 / (29 * 1.0));
```

	Promedio
1	473.827586

Como se ve, esta forma de calcular el promedio es más extensa (y quizá engorrosa) que la anterior donde utilizamos la función AVG.

Realizaremos otro ejemplo donde volveremos a la base **TemperaturasArgentina**, puntualmente a la tabla **Temperaturas**, para obtener el promedio de temperaturas mínimas del día 13/04/2025:

```
SELECT AVG(temperatura_minima) AS PromedioTMin
FROM Temperaturas
WHERE fecha = '2025-04-13';
```

	PromedioTMin
1	8.424324

Ahora, utilicemos el método anterior para corroborar el resultado.

Sumamos las temperaturas mínimas del día 13/04/2025:

```
SELECT SUM(temperatura_minima) AS SumaTMin
FROM Temperaturas
WHERE fecha = '2025-04-13';
```

	SumaTMin
1	935.10

Contamos las temperaturas mínimas del día 13/04/2025:

```
SELECT COUNT(*) AS CuentaTMin
FROM Temperaturas
WHERE fecha = '2025-04-13';
```

	CuentaTMin
1	116

Y finalmente dividimos para obtener el promedio (no hace falta multiplicar por 1.0 ya que el dividendo es un número decimal, por lo que el resultado ya no podrá ser un número entero):

```
SELECT (935.10 / 116) AS Promedio;
```

	Promedio
1	8.061206

Esta vez no obtuvimos el mismo resultado. Con la primera forma (AVG) fue **8,424324** y con la segunda (más extensa), fue **8,061206**. En esta última forma de resolución hubo un error. ¿Puedes notar cuál es?

El error estuvo en considerar a **todos** los registros en la cuenta de temperaturas mínimas del 13/04/2025 ya que lo hicimos de esta manera:

```
SELECT COUNT(*)
```

Como el campo **temperatura\_minima** admite nulos (que naturalmente no deben ser tenidos en cuenta para el promedio), deberíamos haber excluido estos registros.

Veamos cuántos son los registros a considerar. Podemos obtener los nulos, y esa cantidad restarla a la cantidad total de registros del día 13/04/2025:

```
SELECT COUNT(*) AS CuentaTMinNull_1
FROM Temperaturas
WHERE fecha = '2025-04-13' AND temperatura_minima IS NULL;
```

	CuentaTMinNull_1
1	5

De un total de 116 registros quitamos 5; luego la cantidad de registros con valor distinto de NULL en temperatura mínima es de **111**.

O, lo que es lo mismo, averiguar la cantidad de registros donde la columna temperatura\_minima no tiene NULL:

```
SELECT COUNT(temperatura_minima) AS CuentaTMinNull_2
FROM Temperaturas
WHERE fecha = '2025-04-13';
```

	CuentaTMinNoNull
1	111

```
SELECT (935.10 / 111) AS Promedio;
```

	Promedio
1	8.424324

Y ahora sí llegamos al resultado correcto.

## Resumen de AVG

- **AVG** se utiliza para promediar valores pertenecientes a una columna numérica (condición excluyente). Por ende, el resultado obtenido también será un número. Se sugiere multiplicar por 1.0 para obtener la parte decimal, salvo indicación en contrario.
- Cuando escribimos **AVG (Columna)**, obtenemos el promedio de todos los valores no nulos de *Columna*.
- Existe un caso excepcional en el que **AVG** no devuelve un número sino NULL: cuando todos los valores a procesar son nulos (es decir, cuando no hay valores por promediar en la columna especificada).

## MIN y MAX - Obtener el valor mínimo o máximo

Las funciones **MIN** y **MAX** (para obtener el valor mínimo y máximo respectivamente en una tabla o conjunto de registros), a diferencia de SUM o AVG, pueden ser utilizadas con diversos tipos de datos, por ejemplo fechas, alfanuméricos, texto, números, etc.

Escribiremos una consulta en la que obtendremos la duración máxima de una sesión individual de lectura. Cada sesión individual de lectura está representada por la existencia de un registro en la tabla *Lecturas*, mientras que la duración está indicada por el campo *TiempoEnMinutos*.

Una de las formas de resolver, y que además nos sería útil para corroborar resultados, es realizar una consulta donde ordenamos todos los registros de *Lecturas* de mayor a menor (por el campo *TiempoEnMinutos*) y nos quedamos con el primero de ellos. Esto lo vimos antes:

```
SELECT TOP 1 TiempoEnMinutos FROM Lecturas  
ORDER BY TiempoEnMinutos DESC;
```

	TiempoEnMinutos
1	900

Esto es totalmente válido. Pero veamos cómo se puede simplificar aún más la consulta utilizando una función de resumen apropiada:

```
SELECT MAX(TiempoEnMinutos) AS TiempoEnMinutos_Max FROM Lecturas;
```

	TiempoEnMinutos_Max
1	900

Así de simple, llegamos al mismo resultado.

Ahora, queremos hallar la fecha en la que se registró la lectura más antigua en el medio de lectura "Braille EBook". Es decir, datos de la lectura cuya fecha es más atrás en el tiempo, más lejana de hoy, en ese formato de lectura.

Sabemos que los datos de lectura los encontramos en la tabla *Lecturas*, y que en *Medios* disponemos de IDs y nombres de los diferentes medios de lectura. En cuanto a la fecha, la



manera de obtener la más antigua sería preguntar por la fecha "más chica", por así decirlo. Entonces, debemos utilizar la función MIN:

```
SELECT MIN(LE.FechaHora) AS FechaMasAntigua
FROM Lecturas LE
INNER JOIN Medios M ON LE.IDMedio = M.IDMedio
WHERE Nombre = 'Braille Ebook';
```

	FechaMasAntigua
1	2024-05-25

Podemos notar que no es necesario que la columna por la cual filtramos (*Nombre* de la tabla *Medios*) se encuentre en el SELECT. Chequeamos el resultado ordenando de menor a mayor las fechas de lectura con tipo de medio "Braille Ebook" (al ser pocos resultados no es necesario esta vez utilizar un TOP):

```
SELECT LE.FechaHora
FROM Lecturas LE
INNER JOIN Medios M ON LE.IDMedio = M.IDMedio
WHERE M.Nombre = 'Braille Ebook'
ORDER BY FechaHora ASC;
```

	FechaHora
1	2024-05-25
2	2024-05-27
3	2024-05-28
4	2024-05-29
5	2024-05-30

Como podemos ver, la fecha más antigua de lectura con medio "Braille EBook" fue el 25/05/2024, tal como obtuvimos con la función de resumen.

Por último, buscaremos la fecha más reciente en la que se registraron lecturas en alguno de estos medios: Lector de pantalla o Libro de papel en Braille:

```
SELECT MAX(LE.FechaHora)
FROM Lecturas LE
INNER JOIN Medios M ON LE.IDMedio = M.IDMedio
WHERE M.Nombre IN ('Lector de pantalla', 'Libro de papel en Braille');
```

	FechaMasReciente
1	NULL

En este contexto, obtener NULL significa que los medios "Lector de pantalla" y "Libro de papel en Braille" no han sido utilizados en ninguna de las lecturas registradas. Hay muchas formas de verificarlo. Si, por ejemplo, conocemos los ID de los medios en cuestión:

	IDMedio	Nombre
1	2	Libro de papel en Braille
2	10	Lector de pantalla

Podemos contar la cantidad de registros de Lecturas donde *IDMedio* es igual a 2 o 10:

```
SELECT COUNT(*) AS CantidadDeLecturas
FROM Lecturas LE
WHERE LE.IDMedio IN (2, 10);
```

	CantidadDeLecturas
1	0

Y aquí, una cantidad de lecturas igual a cero significa que no hay registros que cumplan con los criterios especificados. Es decir, no hay lecturas registradas que hayan sido realizadas a través de "Lector de pantalla" o "Libro de papel en Braille". Por ende, al no haber

registros para resumir, no es posible evaluar cuál de todas las fechas es la más reciente, y la función MAX devuelve NULL.

## Resumen de MIN/MAX

- **MIN (Columna)** se utiliza para obtener el mínimo de los valores no nulos de *Columna*.
- **MAX (Columna)** se utiliza para obtener el máximo de los valores no nulos de *Columna*.
- En ambos casos, el valor mínimo o máximo puede ser de un tipo numérico, de texto, de fecha, etc.
- Existe un caso excepcional en el que **MIN** y **MAX** devuelven NULL: cuando todos los valores a evaluar son nulos (es decir, cuando no hay valores disponibles en la columna especificada).

## GROUP BY

Esta cláusula nos permite aplicar criterios de agrupamiento a los resultados obtenidos mediante las funciones de resumen. Por ejemplo:

*"Armar un listado en el que se informe, **por cada libro**, su título y la **cantidad total** de **minutos** que los **usuarios** dedicaron a su **lectura**. Considerar únicamente los **libros con más de 700 páginas**".*

En este enunciado, **por cada libro** representa el **criterio de agrupamiento**; la **cantidad total**, aunque no lo indique explícitamente, dentro de este contexto se entiende que equivale a hacer una suma, es decir, es la **función de resumen**. Los **minutos** que los **usuarios leyeron** son los **datos a resumir**, y finalmente tenemos una **condición** que debe cumplirse para obtener los registros a resumir: que la **cantidad de páginas sea mayor a 700**. Sobre este conjunto de registros se realizarán los cálculos.

Para resolver la situación planteada, podemos realizar un breve análisis: el resultado esperado es una tabla con dos columnas, una de ellas el título de cada libro (tabla *Libros*) y la otra, la sumatoria de tiempo en minutos de lectura (tabla *Lecturas*). Luego tenemos una condición vinculada con el campo *Paginas* de la tabla *Libros*, y finalmente, una cuestión no menor, es que los libros a considerar deben haber sido leídos alguna vez. Esto nos obliga a utilizar un determinado tipo de join. Realicemos un bosquejo de la consulta con lo que aprendimos hasta el momento:

```
SELECT LI.Titulo, SUM(LE.TiempoEnMinutos) AS TiempoDeLectura
FROM Libros LI
INNER JOIN Lecturas LE ON LI.IDLibro = LE.IDLibro
WHERE LI.Paginas > 700;
```

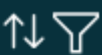
Si ejecutamos la consulta tal como está, obtenemos el siguiente error:

***"Column 'Libros.Titulo' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause"***

Que nos indica que la columna *LI.Titulo* no es válida en la lista de campos a seleccionar ya que no está dentro de una función de agregado (o resumen) o la cláusula GROUP BY. En otras palabras, lo que sucede es que, por un lado, le pedimos al gestor de base de datos que calcule una suma sobre un campo de la tabla *Lecturas* (es decir, le sugerimos que habrá un agrupamiento) y, por otro, que devuelva los valores sin resumir de otro campo de la tabla *Libros*. Por lo que, si no lo indicamos de forma explícita, el gestor no logrará deducir cuál es la columna por la que debe agrupar los registros, que en nuestro caso es *Titulo* de *Libros*.

Veamos entonces cómo queda finalmente la consulta con el agrupamiento:

```
SELECT LI.Titulo, SUM(LE.TiempoEnMinutos) AS TiempoDeLectura
FROM Libros LI
INNER JOIN Lecturas LE ON LI.IDLibro = LE.IDLibro
WHERE LI.Paginas > 700
GROUP BY LI.Titulo;
```

	Titulo 	TiempoDeLectura
1	Apocalipsis	8160
2	History of food	25332
3	It	13799
4	La cúpula	4666
5	Needful things	26939



Ahora sí, para cada uno de los cinco libros que tienen más de 700 páginas (y que fueron leídos), obtuvimos el título y la cantidad total de minutos que los usuarios dedicaron a leerlos.

## Ejemplos de GROUP BY

Pensemos una variante del problema anterior. Ahora queremos obtener la sumatoria de tiempo de lectura de todos los libros, hayan sido leídos o no, y sin el filtro de cantidad de páginas. Si un libro no fue leído, el valor a informar como tiempo de lectura deberá ser 0.



```
SELECT LI.Titulo, SUM(LE.TiempoEnMinutos) AS TiempoDeLectura
FROM Libros LI
LEFT JOIN Lecturas LE ON LI.IDLibro = LE.IDLibro
GROUP BY LI.Titulo;
```

En la consulta anterior, al usar un join de tipo INNER entre *Libros* y *Lecturas*, habíamos exigido que los libros listados hayan sido leídos. Pero, en esta última, reemplazando INNER por LEFT, los libros resultantes pueden sumar tiempo de lectura o no, dando resultados como por ejemplo:

	Titulo 	TiempoDeLectura 
46	Historias inesperadas de la historia argentina	29276
47	La sangre manda	19750
48	Cantar de Mio Cid	NULL
49	1984	29970
50	El extraño caso del Dr. Jekyll y Mr. Hyde	22620

Para lograr que el valor de *TiempoDeLectura* del libro *Cantar de Mio Cid* se informe como cero y no como NULL, recurrimos a un recurso ya conocido: la función ISNULL:

```
SELECT LI.Titulo, ISNULL(SUM(LE.TiempoEnMinutos), 0) AS TiempoDeLectura
FROM Libros LI
LEFT JOIN Lecturas LE ON LI.IDLibro = LE.IDLibro
GROUP BY LI.Titulo;
```

	Titulo 	TiempoDeLectura 
46	Historias inesperadas de la historia argentina	29276
47	La sangre manda	19750
48	Cantar de Mio Cid	0
49	1984	29970
50	El extraño caso del Dr. Jekyll y Mr. Hyde	22620

Ahora sí, tenemos el formato solicitado para informar la columna resultante *TiempoDeLectura*. Una cuestión a tener en cuenta, es que con esta modificación (reemplazar NULL por 0) no estamos afirmando que la suma de minutos de lectura del libro *Cantar de Mio Cid* es cero, ya que eso sería incorrecto (recordemos que NULL representa un valor desconocido, y no se puede sumar algo desconocido). Simplemente, a fines de presentar la información de manera conveniente, estamos indicando que, en este listado, los registros donde el tiempo de lectura sea 0, corresponden a los libros que no se leyeron.

Veamos otro caso. Necesitamos un listado en el que, por cada autor, se muestre sus nombres y apellidos, pseudónimo, y el promedio de las puntuaciones que recibieron por parte de los usuarios. Si existiesen autores que no hayan recibido un puntaje, deben ser incluidos en el listado, con un puntaje promedio de NULL.

Si miramos con atención el DER, veremos que los autores no reciben la puntuación en forma directa sino que el puntaje pertenece a un libro de su autoría. La tabla *Puntuaciones* contiene el puntaje asociado a un libro (clave foránea *IDLibro*) y el usuario que lo calificó (*IDUsuario*). Luego, hemos de relacionar *Libros* con *Autores* para obtener los datos necesarios. Como vimos anteriormente, *Libros* y *Autores* tampoco se vinculan directamente ya que su tipo de relación es "muchos a muchos". Entonces tenemos que pasar por la tabla intermedia *AutoresLibro*. Finalmente, nos piden incluir también a los autores que no tengan puntuación registrada. Esto podría suceder por dos razones: porque ninguno de los libros escritos por el autor recibió al menos una calificación, o bien porque el autor no escribió ni participó en ningún libro. En esta situación, el tipo de join que corresponde utilizar para las uniones es LEFT.

```

SELECT A.Nombres, A.Apellidos, A.Pseudonimo,
AVG(P.Puntaje * 1.0) AS PromedioPuntuacion
FROM Autores A
LEFT JOIN AutoresLibro AL ON A.IDAutor = AL.IDAutor
LEFT JOIN Libros L ON AL.IDLibro = L.IDLibro
LEFT JOIN Puntuaciones P ON L.IDLibro = P.IDLibro
GROUP BY A.Nombres, A.Apellidos, A.Pseudonimo;

```

En cuanto al criterio de agrupamiento, la frase "por cada autor" nos está diciendo que los registros deben agruparse por autor. Entonces, en GROUP BY, debemos enumerar las columnas de la tabla Autores que están en el SELECT.

Se pide expresamente que se informe tal cual el resultado del cálculo, en los casos de autores que no posean datos de calificaciones para calcular el promedio. Por eso, en este caso no debemos forzar un valor de salida para el promedio (como hicimos en el ejemplo anterior con ISNULL).

Observando los primeros cinco resultados:

	Nombres ↑↓	Apellidos ↑↓	Pseudonimo ↑↓	PromedioPuntuacion ↑↓
1	NULL	NULL	Bitmap Books	5.153846
2	Anne	Collet	NULL	6.714285
3	Anthony	Bourdain	NULL	6.625000
4	Antoine de	Saint-Exupéry	NULL	NULL
5	Bernhard	Schlink	NULL	7.666666

Podemos ver que aparecen autores sin nombres, sin apellidos, sin seudónimo, y también sin puntaje, como en el caso de Antoine de Saint-Exupéry: no lo haremos esta vez, pero es sencillo comprobar que para dicho autor no existen registros en la tabla *AutoresLibro*.





Continuando con la base de datos *Libreria*, realicemos un listado que, por cada categoría, muestre su nombre y la cantidad de páginas del libro que posea la mayor cantidad de páginas dentro de la categoría.

```

SELECT C.Categoria, MAX(L.Paginas) AS MaximaCantPaginas
FROM Categorias C
INNER JOIN Libros L ON C.IDCategoria = L.IDCategoria
GROUP BY C.Categoria;

```

Similar al caso anterior, aquí se nos pide informar un valor para cada grupo. En este caso, los grupos son las diferentes categorías de libros, y el valor a informar corresponde a la cantidad de páginas del libro que posea mayor cantidad de páginas en su grupo. Estos son los resultados surgidos de la consulta que escribimos:

	Categoria  	MaximaCantPaginas  
1	Ciencia Ficción	502
2	Cuento	112
3	Culinaria	312
4	Economía	256
5	Ensayo	390
6	Fantasía	376
7	Historia	728
8	Novela	672
9	Salud	304
10	Terror	1152
11	Videojuegos	400

En el último ejemplo que veremos (por ahora) sobre este tema, armaremos un listado donde cada registro contendrá la información siguiente: nombre de usuario, el título del libro que leyó, la cantidad de páginas y el total acumulado de tiempo en minutos que destinó a su lectura.

Como siempre, analizamos los datos a obtener, las relaciones entre entidades, etc. Luego de mucho trabajar con la base de datos *Libreria* sabemos que la tabla que registra las lecturas realizadas por cada usuario es *Lecturas* (a la cual debemos acudir para informar el acumulado de tiempo en minutos de lectura). A esta tabla debemos relacionarla con *Usuarios* (mediante *IDUsuario*) para obtener el nombre del usuario asociado, y finalmente con *Libros* (a través de *IDLibro*) para saber cuál fue el título y la cantidad de páginas. Dado que no se indica el requerimiento de listar usuarios que no hayan registrado lecturas, el tipo







de join a utilizar será INNER. Como los registros de este listado no están sujetos a una condición, entonces tampoco será necesaria la cláusula WHERE para filtrar.

Para agrupar, se debe tener en cuenta en primer lugar a los usuarios, y luego a los libros que cada usuario leyó.

Agregaremos un ordenamiento, primero por nombre de usuario (en forma ascendente) y luego por título de libro, para facilitar la lectura.

Con todo esto analizado, la consulta resultante será:

```
SELECT U.NombreUsuario, LI.Titulo, LI.Paginas,
SUM(LE.TiempoEnMinutos) AS TotalAcumMinutos
FROM Usuarios AS U
INNER JOIN Lecturas AS LE ON U.IDUsuario = LE.IDUsuario
INNER JOIN Libros AS LI ON LE.IDLibro = LI.IDLibro
GROUP BY U.NombreUsuario, LI.Titulo, LI.Paginas
ORDER BY U.NombreUsuario ASC, LI.Titulo ASC;
```

	NombreUsuario 	Titulo 	Paginas 	TotalAcumMinutos 
1	Ana	20.000 Leguas de viaje submarino	416	5282
2	Ana	A brief history of the Internet	200	2230
3	Ana	La Granja	144	3137
4	Ana	La loba de Francia	368	2219
5	Ana	Micro	416	5817
6	Ana	Sombras y luces de la Edad Media	320	3313
7	Ana	Todo oscuro, sin estrellas	368	2253
8	AngeloSimone	Carrie	199	4878
9	AngeloSimone	Corazones en la Atlántida	672	5270
10	AngeloSimone	El umbral de la noche	512	3929
11	AngeloSimone	El viejo y el mar	127	2821
12	AngeloSimone	Historias inesperadas de la historia argentina	320	3916
13	AngeloSimone	Holly	432	2620
14	AngeloSimone	La loba de Francia	368	3622
15	AngeloSimone	Mientras escribo	384	2380
16	AngeloSimone	The Making of Karateka: Journals 1982-1985	320	3274
17	AuroraRomano	1984	328	6840
18	AuroraRomano	Drácula	418	1225
19	AuroraRomano	El Lazarillo de Tormes	336	1269
20	AuroraRomano	El lector	224	5211

Para cada usuario, aparece listado cada libro leído, con su cantidad de páginas y el tiempo en minutos que demoró en completarlo (en total, entre todas las sesiones de lectura).

Si a partir de estos resultados nos surge la inquietud de conocer el promedio de tiempo de lectura por página de cada libro, veamos qué sucede si agregamos lo siguiente:

```
SELECT U.NombreUsuario, LI.Titulo, LI.Paginas,
SUM(LE.TiempoEnMinutos) AS TotalAcumMinutos,
AVG(LE.TiempoEnMinutos * 1.0) AS PromedioMinutosPorPagina
FROM Usuarios AS U
INNER JOIN Lecturas AS LE ON U.IDUsuario = LE.IDUsuario
INNER JOIN Libros AS LI ON LE.IDLibro = LI.IDLibro
GROUP BY U.NombreUsuario, LI.Titulo, LI.Paginas
ORDER BY U.NombreUsuario ASC, LI.Titulo ASC;
```

	NombreUsuario	Titulo	Paginas	TotalAcumMinutos	PromedioMinutosPorPagina
1	Ana	20.000 Leguas de viaje submarino	416	5282	170.387096
2	Ana	A brief history of the Internet	200	2230	131.176470
3	Ana	La Granja	144	3137	184.529411
4	Ana	La loba de Francia	368	2219	158.500000
5	Ana	Micro	416	5817	153.078947

No hace falta mirar muchos registros para notar que el valor informado para el promedio es incorrecto. Basta con tomar el primer registro para verificar fácilmente que el tiempo promedio de lectura de 416 páginas (aproximadamente 170 minutos, siendo el total de lectura del libro de 5282 minutos), excede ampliamente a un valor razonable. ¿Cuál fue el error en el razonamiento del cálculo? Lo que sucedió es que basamos el cálculo del promedio sobre la columna *TiempoEnMinutos* (sin resumir), en lugar de hacerlo por el valor de la columna ya resumida (*TotalAcumMinutos*). En otras palabras, *LE.TiempoEnMinutos* no contiene el total de minutos de lectura por libro, sino por sesión de lectura.

Para solucionarlo, como no es posible usar el alias directamente, podemos hacer lo siguiente:

```
SELECT U.NombreUsuario, LI.Titulo, LI.Paginas,
SUM(LE.TiempoEnMinutos) AS TotalAcumMinutos,
SUM(LE.TiempoEnMinutos * 1.0) / (LI.Paginas) AS PromedioMinutosPorPagina
FROM Usuarios AS U
INNER JOIN Lecturas AS LE ON U.IDUsuario = LE.IDUsuario
INNER JOIN Libros AS LI ON LE.IDLibro = LI.IDLibro
GROUP BY U.NombreUsuario, LI.Titulo, LI.Paginas
ORDER BY U.NombreUsuario ASC, LI.Titulo ASC;
```

	NombreUsuario ↑↓▽	Titulo ↑↓▽	Paginas ↑↓▽	TotalAcumMinutos ↑↓▽	PromedioMinutosPorPagina ↑↓▽
1	Ana	20.000 Leguas de viaje submarino	416	5282	12.697115
2	Ana	A brief history of the Internet	200	2230	11.150000
3	Ana	La Granja	144	3137	21.784722
4	Ana	La loba de Francia	368	2219	6.029891
5	Ana	Micro	416	5817	13.983173

Ahora sí, obtuvimos el promedio de tiempo de lectura por página.

## HAVING

La cláusula HAVING permite aplicar una o más condiciones al conjunto de resultados de las funciones de resumen, para poder filtrar los mismos.

Para comprender el concepto veremos un ejemplo como el de GROUP BY, con una pequeña variante:

*"Armar un listado en el que se informe, **por cada libro**, su título y la **cantidad total de minutos** que los **usuarios** dedicaron a su **lectura**. Considerar únicamente los **libros con más de 700 páginas** y que **acumulen más de 500 horas de lectura**".*

Podemos identificar la misma estructura de antes: **por cada libro** indica el agrupamiento, los **minutos de lectura** son los datos a resumir, la **cantidad total** indica (aunque no directamente) que hay una suma, es decir la función de resumen, y luego tenemos condiciones: los **libros con más de 700 páginas** y que **acumulen más de 200 horas de lectura**. Tomaremos tal cual está la consulta que escribimos antes, y nos detendremos únicamente en la nueva condición que se agregó. Hasta aquí, es la misma consulta:

```
SELECT LI.Titulo, SUM(LE.TiempoEnMinutos) AS TiempoDeLectura
FROM Libros LI
INNER JOIN Lecturas LE ON LI.IDLibro = LE.IDLibro
WHERE LI.Paginas > 700
GROUP BY LI.Titulo;
```

Ahora bien, para incorporar la condición de horas de lectura, tiene bastante sentido pensar que podemos resolverlo con una sentencia más dentro del WHERE: el acumulado de tiempo de lectura (en minutos) está dado por SUM(LE.TiempoEnMinutos), y luego, si una hora tiene 60 minutos, entonces 200 horas equivalen a 12000 minutos:

```
SELECT LI.Titulo, SUM(LE.TiempoEnMinutos) AS TiempoDeLectura
FROM Libros LI
INNER JOIN Lecturas LE ON LI.IDLibro = LE.IDLibro
WHERE (LI.Paginas > 700) AND (SUM (LE.TiempoEnMinutos) > 12000)
GROUP BY LI.Titulo;
```

Sin embargo, esta consulta no funciona y su sintaxis no es correcta; nos devuelve este mensaje de error:

***"An aggregate may not appear in the WHERE clause unless it is in a subquery contained in a HAVING clause or a select list, and the column being aggregated is an outer reference"***

De forma muy sintética, lo que este mensaje nos dice es que no podemos utilizar una función de agregado dentro del WHERE salvo que cumpla ciertas condiciones, pero, no viene al caso profundizar en esto. El motivo por el que no funciona lo anterior es porque las dos condiciones no aplican sobre el mismo conjunto de datos. Mientras que esta condición:

**LI.Paginas > 700**

Está dirigida al **conjunto de todos los libros**, y se ejecuta **antes** de realizar el agrupamiento, la otra condición:



**SUM (LE.TiempoEnMinutos) > 12000**

Se ejecuta **después** de agrupar los registros, ya que se aplica sobre el **conjunto de datos resumidos por libro**. Una diferencia que parece sutil pero no lo es.

Aquí es donde entra en juego la cláusula **HAVING**: se utiliza para filtrar los registros que son el resultado de haber aplicado una función de resumen, para determinar si deben formar parte del listado final (o no). La diferencia con WHERE es que esta última trabaja con los registros que se van a resumir.




Finalmente la consulta resulta así:

```
SELECT LI.Titulo, SUM(LE.TiempoEnMinutos) AS TiempoDeLectura
FROM Libros LI
INNER JOIN Lecturas LE ON LI.IDLibro = LE.IDLibro
WHERE LI.Paginas > 700
GROUP BY LI.Titulo
HAVING SUM (LE.TiempoEnMinutos) > 12000;
```

	Titulo 	TiempoDeLectura 
1	History of food	25332
2	It	13799
3	Needful things	26939

Para corroborar el resultado, podemos incorporar la columna *Paginas* de *Libros* (tanto en la selección como en el agrupamiento) para chequear que efectivamente los tres libros en el listado tienen más de 700 páginas:

```
SELECT LI.Titulo, LI.Paginas, SUM(LE.TiempoEnMinutos) AS TiempoDeLectura
FROM Libros LI
INNER JOIN Lecturas LE ON LI.IDLibro = LE.IDLibro
WHERE LI.Paginas > 700
GROUP BY LI.Titulo, LI.Paginas
HAVING SUM (LE.TiempoEnMinutos) > 12000;
```

	Titulo 	Paginas 	TiempoDeLectura 
1	History of food	728	25332
2	Needful things	736	26939
3	It	1138	13799

Y también podríamos quitar por un momento la condición del acumulado de 200 horas de lectura, y ordenar los resultados, para verlo más claro:

```
SELECT LI.Titulo, LI.Paginas, SUM(LE.TiempoEnMinutos) AS TiempoDeLectura
FROM Libros LI
INNER JOIN Lecturas LE ON LI.IDLibro = LE.IDLibro
GROUP BY LI.Titulo, LI.Paginas
ORDER BY TiempoDeLectura DESC;
```

	Titulo	↑↓	Paginas	↑↓	TiempoDeLectura	↑↓
20	La expedición		320		27024	
21	Needful things		736		26939	
22	De cómo un rey perdió Francia		352		26858	
23	La vida en la Edad Media		356		26592	
24	Viaje al centro de la Tierra		312		26468	
25	La historia de Nintendo		400		26151	
26	History of food		728		25332	
27	Insomnia		663		24937	
28	Buscando a Papá Noel		128		24858	
29	La reina estrangulada		384		24521	
30	El visitante		592		23369	

(se omitieron adrede algunos resultados).

## Más ejemplos con Having

Armemos un listado con los títulos de los libros que registren una puntuación promedio mayor a 7,5.

Observando el DER, sólo necesitamos dos tablas: *Libros* y *Puntuaciones*, que se relacionan con el campo *IDLibro*. Si bien no se pide informar la puntuación promedio, la mostraremos para ilustrar los resultados, lo mismo el ordenamiento. En principio tendríamos una consulta con esta estructura:

```
SELECT LI.Titulo, AVG(P.Puntaje * 1.0) AS PromedioPuntaje
FROM Libros LI
INNER JOIN Puntuaciones P ON LI.IDLibro = P.IDLibro
GROUP BY LI.Titulo
ORDER BY PromedioPuntaje DESC;
```

	Titulo 	PromedioPuntaje 
1	Confesiones de un chef	8.333333
2	El Lazarillo de Tormes	8.250000
3	Las cuatro estaciones: Primavera y verano	8.000000
4	Mr Mercedes	8.000000
5	Rabia	8.000000
6	La vida en la Edad Media	7.666666
7	El lector	7.666666
8	A brief history of the Internet	7.666666
9	Carrie	7.666666
10	The Making of Karateka: Journals 1982-1985	7.500000
11	It	7.333333
12	La ley de los varones	7.333333
13	La economía de tu vida	7.250000
14	Los venenos de la corona	7.125000
15	Historias inesperadas de la historia argentina	7.000000
16	Armada	7.000000
17	Salem's Lot	7.000000
18	Viaje al centro de la Tierra	6.833333
19	La sangre manda	6.833333
20	Mientras escribo	6.833333

Ahora sabemos que, cuando incorporemos la condición de promedio mayor a 7,5, sólo deberíamos visualizar los primeros nueve registros de la imagen anterior: veamos si así sucede.

```
SELECT LI.Titulo, AVG(P.Puntaje * 1.0) AS PromedioPuntaje
FROM Libros LI
INNER JOIN Puntuaciones P ON LI.IDLibro = P.IDLibro
GROUP BY LI.Titulo
HAVING AVG(P.Puntaje * 1.0) > 7.5
ORDER BY PromedioPuntaje DESC;
```

	Titulo	↑↓🔍	PromedioPuntaje	↑↓🔍
1	Confesiones de un chef		8.333333	
2	El Lazarillo de Tormes		8.250000	
3	Las cuatro estaciones: Primavera y verano		8.000000	
4	Mr Mercedes		8.000000	
5	Rabia		8.000000	
6	A brief history of the Internet		7.666666	
7	Carrie		7.666666	
8	El lector		7.666666	
9	La vida en la Edad Media		7.666666	

Como podemos ver, no solamente seleccionamos los 9 registros que queríamos, que es lo más importante, sino que "de yapa", al haber utilizado HAVING, los registros "empatados" en puntaje se muestran ordenados alfabéticamente.



Veamos otro caso, en el que se nos pide listar los nombres de los medios de lectura que han sido utilizados más de 1000 veces.

De nuevo, acudimos al DER si es necesario, para ver que las tablas necesarias son solamente *Lecturas* y *Medios*, relacionadas a través de la columna *IDMedio*. En otras palabras, los "usos" de los medios están representados por las ocurrencias de *IDMedio* dentro de la tabla *Lecturas*. Luego, por cada medio, debemos contar los registros de *Lecturas* donde aparezca su ID. Como se nos pide listar los nombres de los medios, eso nos sugiere que el agrupamiento es por medio.

Hagamos primero un bosquejo de la consulta, es decir, sin el filtrado, y ordenando de forma descendente por cantidad de usos, para tener una idea más clara de lo que debemos obtener:

```
SELECT M.Nombre AS Medio, COUNT(*) AS CantUsosMedio
FROM Medios M
INNER JOIN Lecturas LE ON M.IDMedio = LE.IDMedio
GROUP BY M.Nombre
ORDER BY CantUsosMedio DESC;
```





	Medio 	CantUsosMedio 
1	Libro de papel	1930
2	Ebook	1862
3	Tablet	1494
4	Celular	1144
5	PC	1117
6	Tablet con E-Ink	781
7	Audiolibro	727
8	Braille Ebook	5

A excepción de los medios de lectura nunca utilizados, sabemos que "Tablet con E-Ink", "Audiolibro" y "Braille Ebook" no deberán aparecer en el listado final.

Ahora sí, incluyendo la condición:

```
SELECT M.Nombre AS Medio, COUNT(*) AS CantUsosMedio
FROM Medios M
INNER JOIN Lecturas LE ON M.IDMedio = LE.IDMedio
GROUP BY M.Nombre
HAVING COUNT(*) > 1000
ORDER BY CantUsosMedio DESC;
```

	Medio 	CantUsosMedio 
1	Libro de papel	1930
2	Ebook	1862
3	Tablet	1494
4	Celular	1144
5	PC	1117

Vemos solamente los cinco resultados esperados.

Notar que COUNT(\*) es suficiente sin más para contar las apariciones de los ID de medios en Lecturas, ya que al tener un join de tipo INNER, los registros donde no exista vinculación entre un medio de lectura determinado y una sesión de lectura, no serán tenidos en cuenta en el listado.

En este último ejemplo debemos preparar un listado que contenga apellidos, nombres y pseudónimos de aquellos autores que hayan sido leídos por más de 20 usuarios distintos.

No ocuparemos tiempo esta vez en describir las relaciones entre tablas y el "camino" a recorrer para llegar a los registros solicitados, ya que lo complejo de esta consulta radica en el agrupamiento y filtrado de los registros agrupados.

Escribimos, entonces, el esquema básico de esta consulta:

```
SELECT A.Apellidos, A.Nombres, A.Pseudonimo,  
COUNT(*) AS CantUsuarios  
FROM Autores A  
INNER JOIN AutoresLibro AL ON A.IDAutor = AL.IDAutor  
INNER JOIN Libros LI ON AL.IDLibro = LI.IDLibro  
INNER JOIN Lecturas LE ON LE.IDLibro = LI.IDLibro  
GROUP BY A.Apellidos, A.Nombres, A.Pseudonimo  
ORDER BY CantUsuarios DESC;
```

Teniendo en cuenta que nos solicitan datos de autores, es evidente que el agrupamiento es por autor. Si observamos un grupo de resultados:

	Apellidos ↕	Nombres ↕	Pseudonimo ↕	CantUsuarios ↕
3	Verne	Julio	NULL	625
4	Cline	Ernest	NULL	474
5	Bartlett	Robert	NULL	473
6	Crichton	Michael	NULL	434
7	Balmaceda	Daniel	NULL	381
8	NULL	NULL	Bitmap Books	311
9	Stevenson	Robert Louis	NULL	297
10	Alighieri	Dante	NULL	267
11	Carroll	Lewis	NULL	249
12	Mechner	Jordan	NULL	235
13	Kawamura	Genki	NULL	218
14	Eco	Umberto	NULL	188
15	Bulat	Tomás	NULL	156
16	Hemingway	Ernest	NULL	156
17	Collet	Anne	NULL	150
18	Greenberg	Paul	NULL	150
19	Bourdain	Anthony	NULL	115
20	Stoker	Bram	NULL	100
21	Pérez Rodríguez	Uxio	NULL	94
22	Amicis	Edmundo de	NULL	84

Parece correcto pero, sin embargo, hay algo extraño con la cantidad de usuarios informados: ¿realmente contamos con 625 usuarios distintos en nuestra base de datos?

Es fácil de verificar si hacemos una consulta como la siguiente a la tabla *Usuarios*:

```
SELECT COUNT(IDUsuario) AS CantidadUsuarios FROM Usuarios;
```

	CantidadUsuarios
1	60

Y comprobamos, con cierto desdén, que algo salió mal en nuestra consulta.

Pero la buena noticia es que es un desliz sencillo de solucionar. Lo que ocurre es que esta expresión:

**COUNT(\*) AS CantUsuarios**

Está contando la **cantidad de sesiones de lectura** donde participaron libros de cada autor. No es lo mismo que la **cantidad de usuarios distintos**, ya que un usuario puede haber leído un libro en más de una sesión, incluso en muchas. Y también puede haber leído más de un libro del autor.

Aquí viene al rescate la cláusula DISTINCT, para pedirle al gestor de base de datos que solamente cuente los usuarios distintos:

```
SELECT A.Apellidos, A.Nombres, A.Pseudonimo,
COUNT(DISTINCT LE.IDUsuario) AS CantUsuarios
FROM Autores A
INNER JOIN AutoresLibro AL ON A.IDAutor = AL.IDAutor
INNER JOIN Libros LI ON AL.IDLibro = LI.IDLibro
INNER JOIN Lecturas LE ON LE.IDLibro = LI.IDLibro
GROUP BY A.Apellidos, A.Nombres, A.Pseudonimo
ORDER BY CantUsuarios DESC;
```

	Apellidos ↕⌵	Nombres ↕⌵	Pseudonimo ↕⌵	CantUsuarios ↕⌵
1	King	Stephen	Richard Bachman	51
2	Druon	Maurice	NULL	29
3	Verne	Julio	NULL	27
4	Bartlett	Robert	NULL	21
5	Cline	Ernest	NULL	20
6	Crichton	Michael	NULL	18
7	Balmaceda	Daniel	NULL	16
8	Stevenson	Robert Louis	NULL	13
9	Alighieri	Dante	NULL	12
10	NULL	NULL	Bitmap Books	11

Ahora, el resultado tiene más sentido, ya que el autor que registró la mayor cantidad de usuarios que lo leyeron (Stephen King) tiene 51 usuarios, que es menor a 60 (la cantidad total de usuarios).

Lo único que nos queda por hacer es armar la condición para listar solamente los autores leídos por más de 20 usuarios distintos, para lo cual volvemos a utilizar la misma estrategia:

```
SELECT A.Apellidos, A.Nombres, A.Pseudonimo,  
COUNT(DISTINCT LE.IDUsuario) AS CantUsuarios  
FROM Autores A  
INNER JOIN AutoresLibro AL ON A.IDAutor = AL.IDAutor  
INNER JOIN Libros LI ON AL.IDLibro = LI.IDLibro  
INNER JOIN Lecturas LE ON LE.IDLibro = LI.IDLibro  
GROUP BY A.Apellidos, A.Nombres, A.Pseudonimo  
HAVING COUNT(DISTINCT LE.IDUsuario) > 20  
ORDER BY CantUsuarios DESC;
```

	Apellidos ↕	Nombres ↕	Pseudonimo ↕	CantUsuarios ↕
1	King	Stephen	Richard Bachman	51
2	Druon	Maurice	NULL	29
3	Verne	Julio	NULL	27
4	Bartlett	Robert	NULL	21

Y se listan exitosamente los cuatro autores cuyos libros han sido leídos por más de 20 usuarios distintos. Si desean pueden escribir una consulta donde figuren los nombres de los usuarios, para mayor detalle.