

## Consultas básicas con condiciones

Ya conocimos la sentencia que nos permite realizar consultas a nuestras tablas. En esta lección vamos a incorporar las condiciones para hacer nuestras consultas más completas.

### Condiciones en el select:

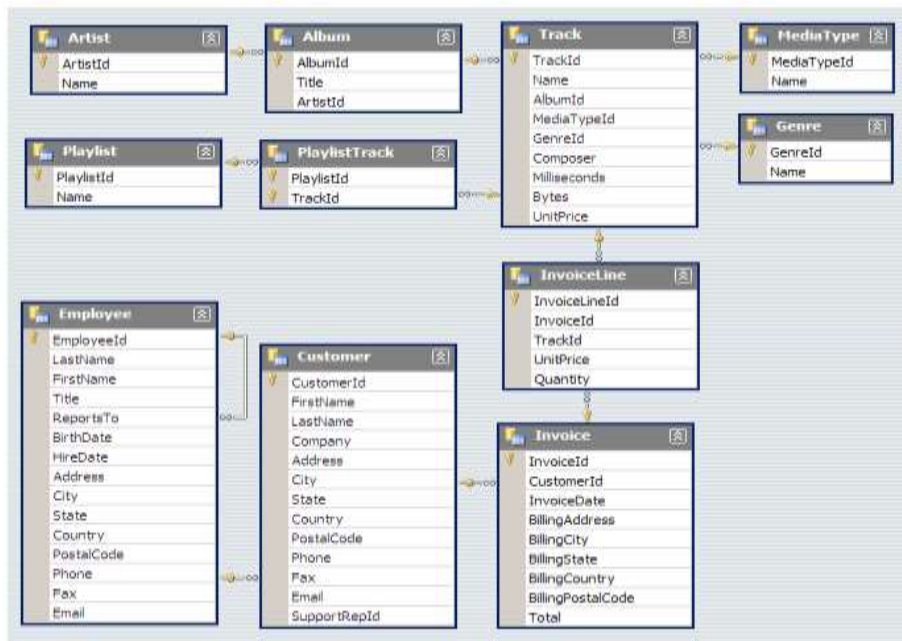
La sentencia SELECT nos permite mostrar los registros de una tabla que cumplan con determinadas condiciones o restricciones. Para poder agregar estas condiciones, usaremos el WHERE (condiciones a nivel de registro).

Si quisiéramos conectar más de una condición, emplearemos los conectores lógicos AND y OR. Y para los operadores de comparación:

=	Igual
<	Menor que
> =	Menor o igual que
< =	Mayor o igual que
>	Mayor que
< >	Distinto que

### Pasemos a un ejemplo:

Imaginemos que tengo el siguiente DER de nuestra base (vamos a trabajar mucho con este diagrama para los ejemplos de consultas), el cual representa un modelo de negocio de venta de música online:



Ahora supongamos que deseamos obtener un listado de las canciones que cuestan entre 0,10 y 1,50.

Revisando nuestro DER, detectamos que solo vamos a necesitar para nuestra consulta la tabla track, y nuestro FROM quedará de la siguiente forma

*FROM track t*

Ahora trabajemos con el WHERE. Sabemos que debemos condicionar los registros de la tabla en cuestión pidiendo que un campo cumpla con la condición del precio entre 0,10 y 1,50. El campo en cuestión es UnitPrice. Para condicionar que esté dentro de estos valores haremos uso de los conectores lógicos y algunos de comparación. Quedaría de la siguiente forma:

*WHERE t.unitprice >= '0.10'* de esta forma le pedimos al campo que sea mayor a 0,10

*AND t.unitprice <= '1,50'* y de esta otra que sea además menor a 1,50

Los datos numéricos siempre los vamos a encerrar entre comillas simples.

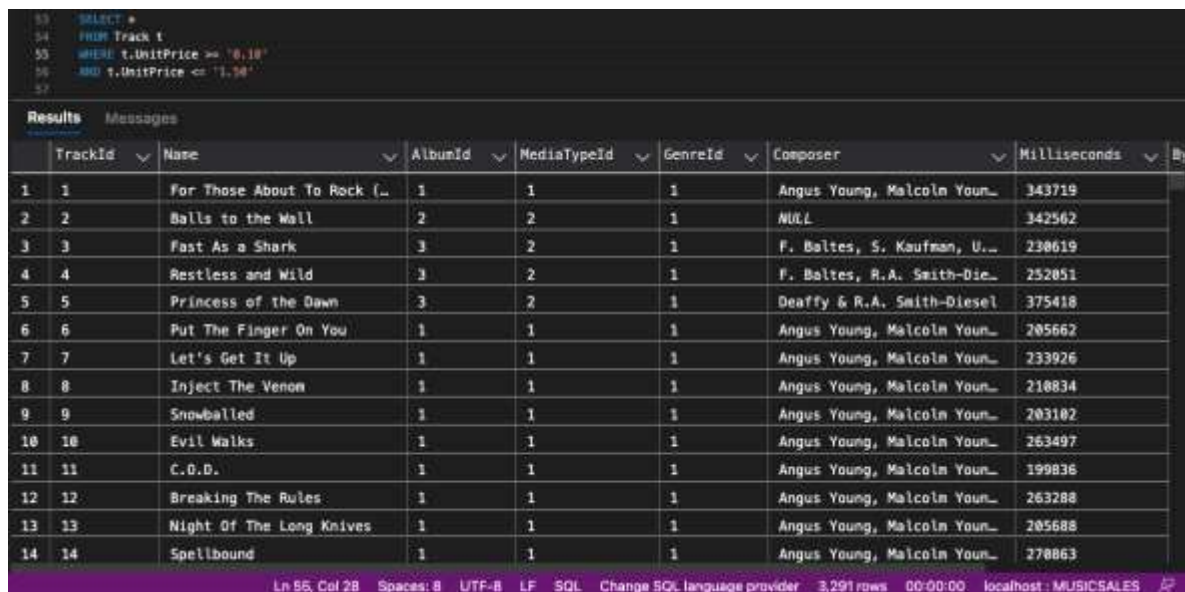
Solo nos resta seleccionar qué datos vamos a mostrar. Visto que el pedido no lo aclaraba utilizaremos \* para que nos traiga todos los datos de la tabla. La consulta completa queda de la siguiente forma:

```

SELECT *
FROM Track t
WHERE t.unitprice >= '0.10'
AND t.unitprice <= '1.50'

```

Y el resultado en el motor nos daría algo similar a esto:



The screenshot shows a SQL query executed in a database client. The query is:
 

```

SELECT *
FROM Track t
WHERE t.UnitPrice >= '0.10'
AND t.UnitPrice <= '1.50'

```

 The results are displayed in a table with 14 rows. The columns are: TrackId, Name, AlbumId, MediaTypeId, GenreId, Composer, and Milliseconds. The status bar at the bottom indicates 3,291 rows returned.

TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds
1	For Those About To Rock (...	1	1	1	Angus Young, Malcolm Youn...	343719
2	Balls to the Wall	2	2	1	NULL	342562
3	Fast As a Shark	3	2	1	F. Baltes, S. Kaufman, U...	230619
4	Restless and Wild	3	2	1	F. Baltes, R.A. Smith-Die...	252051
5	Princess of the Dawn	3	2	1	Deaffy & R.A. Smith-Diesel	375418
6	Put The Finger On You	1	1	1	Angus Young, Malcolm Youn...	205662
7	Let's Get It Up	1	1	1	Angus Young, Malcolm Youn...	233926
8	Inject The Venom	1	1	1	Angus Young, Malcolm Youn...	210834
9	Snowballed	1	1	1	Angus Young, Malcolm Youn...	203102
10	Evil Walks	1	1	1	Angus Young, Malcolm Youn...	263497
11	C.O.D.	1	1	1	Angus Young, Malcolm Youn...	199836
12	Breaking The Rules	1	1	1	Angus Young, Malcolm Youn...	263288
13	Night Of The Long Knives	1	1	1	Angus Young, Malcolm Youn...	205688
14	Spellbound	1	1	1	Angus Young, Malcolm Youn...	270863

Se habrán preguntado qué significa la “t” que está justo al lado del nombre de la tabla. A eso lo llamamos alias y nos permite, una vez que lo escribimos, hacer referencia a nuestra tabla con la palabra o letras que hayamos puesto al lado de la tabla en el FROM.

Nos será de mucha utilidad y en especial cuando debemos hacer uso de más de una tabla, para poder referenciar a campos de nuestras tablas sin problemas, incluso cuando tengan el mismo nombre. (Ej.: Tabla Personas columna nombre y tabla Empleados columna nombre).

Veamos cómo hacer uso del conector OR.

Vamos a plantear un ejemplo donde las condiciones sean dos y se deban cumplir una u otra, el motor va a validar que se cumpla alguna de las dos. En otras palabras, buscará que alguna sea positiva.

Supongamos que deseamos el mismo listado de canciones, pero nos solicitan que se encuentren entre esos valores, 0,10 y 1,50 pero además aquellas que tenga una duración menor a 120000 milisegundos (sin importar el valor).

Nuestro *where* quedaría de la siguiente forma utilizando lo que ya teníamos previo.

*WHERE (t.UnitPrice >= '0.10'*

*AND t.UnitPrice <= '1.50')* hasta este paso estaba igual al anterior.

*OR t.Milliseconds <= 120000* y con el conector incorporamos la otra condición.

Fue necesario agregar entre paréntesis las primeras condiciones para que no se mezclarán con el OR.

El resultado sería el siguiente, mostrando solo las columnas Name, UnitPrice y Milliseconds:

```
60 -- Mostrar el nombre de los Canciones que cuestan
61 -- entre '0.10' y '1.50', o aquellas canciones que tienen
62 -- una duración menor a 120000 milisegundos
63 SELECT t.Name, t.UnitPrice, t.Milliseconds
64 FROM Track t
65 WHERE (t.UnitPrice >= '0.10'
66 AND t.UnitPrice <= '1.50')
67 OR t.Milliseconds <= 120000
68 ORDER by t.UnitPrice DESC
```

Results		Messages	
	Name	UnitPrice	Milliseconds
1	LOST Season 4 Trailer	2.19	112712
2	Money	1.20	147591
3	Long Tall Sally	1.20	106396
4	Bad Boy	1.20	116088
5	Twist And Shout	1.20	161123
6	Please Mr. Postman	1.20	137639
7	C'Mon Everybody	1.20	140199
8	Rock 'N' Roll Music	1.20	141923
9	Slow Down	1.20	163265

Como se puede ver, hay una canción que no cumple con la condición previa del OR referida al precio, pero sí cumple con la otra condición referida a la duración.

Aplicar las condiciones a otros tipos de datos:

Hasta ahora vimos ejemplos con números, pero en nuestras tablas vamos a tener diferentes tipos de datos.

Los operadores de comparación pueden ser usados con otros tipos de datos como fechas y *string*.

Veamos un ejemplo con fechas, donde se requiere las facturas realizadas en un período dado por ejemplo el año 2020.

La tabla de nuestro dominio será Invoice y el campo donde debemos aplicar las condiciones es InvoiceDate.

Cabe aclarar que para las fechas debemos revisar la configuración de nuestro motor para saber cómo las toma. En mi caso las tengo en el formato aaaa-mm-dd, 4 dígitos para el año, 2 para el mes y 2 para el día por lo que mi consulta quedaría de la siguiente forma:

```
SELECT *  
  
FROM Invoice i  
  
WHERE i.InvoiceDate >= '2020-01-01'.  
  
AND i.InvoiceDate < '2021-01-01'
```

Condiciono para que la columna InvoiceDate sea mayor e igual al 1 de enero del 2020 y menor (no inclusive) al 1 de enero del 2021, y de esta forma cubro solo las facturas emitidas en el año 2020. El resultado de la consulta queda algo así:

```

75 -- Listado de Facturas realizadas en el año 2020
76 SELECT *
77 FROM Invoice i
78 WHERE i.InvoiceDate >= '2020-01-01'
79 AND i.InvoiceDate < '2021-01-01'
80

```

	InvoiceId	CustomerId	InvoiceDate	BillingAddress	BillingCity	BillingState	BillingCountry
1	250	55	2020-01-01 00:00:00.000	421 Bourke Street	Sidney	NSW	Australia
2	251	10	2020-01-09 00:00:00.000	Rua Dr. Falcão Filho, 155	São Paulo	SP	Brazil
3	252	11	2020-01-22 00:00:00.000	Av. Paulista, 2022	São Paulo	SP	Brazil
4	253	13	2020-01-22 00:00:00.000	Qe 7 Bloco G	Brasília	DF	Brazil
5	254	15	2020-01-23 00:00:00.000	700 W Pender Street	Vancouver	BC	Canada
6	255	19	2020-01-24 00:00:00.000	1 Infinite Loop	Cupertino	CA	USA
7	256	25	2020-01-27 00:00:00.000	319 N. Frances Street	Madison	WI	USA
8	257	36	2020-02-01 00:00:00.000	Rua do Assunção, 53	Lisbon	NUL	Portugal

Ahora veamos qué pasa con los *strings*. En el próximo módulo haremos mayor hincapié en este tipo de consulta por ser, a mi entender, una de las más utilizadas en los sitios web.

Pocos operadores de comparación nos serán realmente útiles para los *string* o cadena de caracteres, incluso cuando los motores nos permitan usar todos no vamos a obtener el resultado esperado.

El que vamos a poder usar sin problema es el Igual, comparando una columna con una cadena de caracteres, por ejemplo, buscar el género Jazz. Nos quedaría de la siguiente forma:

```

SELECT *
FROM Genre g
WHERE g.Name = 'Jazz'

```

Esto tiene una contra importante, casi que debemos saber y/o adivinar como están guardados los datos en nuestras tablas o será imposible buscar algo con tanta precisión. En este caso, suponiendo que Jazz va a esta capitalizado cuando podría no estar así.

Acá es cuando vamos a hacer uso de las funciones del motor. La mayoría viene preconfigurado con un set de funciones que nos van a ayudar para realizar todo tipo de operaciones y para todo tipo de datos.

Para mejorar la anterior consulta vamos a hacer uso de UPPER y LOWER, puede variar el nombre dependiendo del motor (recurrir al manual del motor para investigar cómo se escriben).

Lo que hace UPPER es que dado un *string* o cadena de caracteres pasa todos sus caracteres a mayúsculas y LOWER hace lo contrario. Acá dejamos un ejemplo:

```
75 Select lower ('RUBEN');
76
```

Results		Messages
	(No column name) ▾	
1	ruben	

```
75 Select upper ('ruben');
76
```

Results		Messages
	(No column name) ▾	
1	RUBEN	

Y con cualquiera de ellas podemos mejorar cualquier consulta de caracteres olvidándonos del problema de mayúsculas y minúsculas.

*SELECT \**

*FROM Genre g*

*WHERE lower(g.Name) = 'jazz'*

Y lo que ocurrirá es que por cada registro el motor primero pasará a minúsculas el dato que se encuentra guardado para luego compararlo con la palabra que hayamos puesto, en este caso “jazz”.

Veamos otro ejemplo, referido a nombres de personas. Queremos hallar los datos del cliente con nombre Astrig y apellido Gruber.

En principio, nuestro FROM será la tabla Customer, y las condiciones las aplicamos sobre las columnas firstname y lastname. Aprovechamos para hacer uso de lower o upper, y nos quedaría de la siguiente forma:

*Select \**

*From Customer c*

*Where lower(c.FirstName) = 'astrid'*

*And lower(c.LastName) = 'gruber'*

Y este es el resultado:

```
212 Select *
213 From Customer c
214 Where lower(c.FirstName) = 'astrid'
215 And lower(c.LastName) = 'gruber'
```

Results

Messages

	CustomerId	FirstName	LastName	Company	Address	City	State	Country	PostalCode
1	7	Astrid	Gruber	NULL	Rotenturmstraße 4, 1010 I...	Vienne	NULL	Austria	1010

De igual forma se puede combinar estas funciones con operaciones aritméticas necesarias, por ejemplo, las facturas realizadas el día de ayer.

```
SELECT *
FROM Invoice i
WHERE i.InvoiceDate+2 >= GETDATE()
AND i.InvoiceDate+1 <= GETDATE()
```

En la consulta incorporo una función de tipo fecha que nos será muy útil. GETDATE (en SQL Server) nos retorna la fecha y hora exacta. Y realizo una operación aritmética con fecha al sumar 2 a una columna de tipo fecha, el motor automáticamente lo toma como sumar 2 días al campo.

En otras palabras, estamos condicionando a las facturas para que sean las emitidas hace 2 días, y además sean menores a hoy en un día.

Ejemplo:

Si GETDATE() fuera 20/09/2021, la consulta me retornaría solo las facturas con fecha 18 y 19 de septiembre.

Si quisiera comparar directamente fechas debería utilizar otra función que solo me permita obtener la fecha del dato y no la hora.



```
SELECT CONVERT(date, getdate());
```

La función Convert nos permite convertir diferentes tipos de datos a otros. En otros motores existen similares funciones.

Y gracias a esto mi consulta quedaría de la forma:

```
SELECT *
```

```
FROM Invoice i
```

```
WHERE CONVERT(date,i.InvoiceDate+1) = CONVERT(date,GETDATE());
```

Logrando obtener las facturas emitidas en el día de ayer.

Cada motor implementa diferentes funciones que nos van a permitir volver más poderosas nuestras consultas. Las vamos a encontrar para el manejo de cadena de caracteres, fechas, numéricas y muchas más.