Universidad Nacional del Litoral

Facultad de Ingeniería y Ciencias Hídricas

Departamento de Informática

Bases de Datos

SQL: Guía de Trabajo Nro. 1

Consultas básicas

Msc. Lic. Hugo Minni
2020

1. La sentencia SELECT
Lista de salida del SELECT
La cláusula FROM

La forma más simple de sentencia `SELECT` sólo especifica las columnas a recuperar y la tabla donde se encuentran las mismas:

```
SELECT nomColumna1, nomColumna2
   FROM tabla
```

A la lista de columnas le llamaremos **lista de salida** del *`SELECT`*.

Ordenamiento: La cláusula `ORDER BY`

Recordemos que la cláusula `ORDER BY` nos permite especificar el orden en el que aparecerán las filas recuperadas por una sentencia `SELECT`, y -de estar definida- debe ser la última cláusula de la sentencia `SELECT`.

El orden por omisión es el ascendente. Podemos anexar el modificador `DESC` para especificar un orden descendente.

1. Obtenga el código de título, título, tipo y precio incrementado en un 8% de todas las publicaciones, ordenadas por tipo y título.

Guía para resolver la consulta

Para resolver una consulta es imprescindible estar familiarizados son el modelo físico de la base de datos con la que estamos trabajando (en este caso **Pubs**).
Tenemos que conocer a grandes rasgos qué información posee cada entidad (tabla) y cuales son las relaciones entre las mismas.

En este caso nos piden información sobre las publicaciones, así que sabemos que la misma se encontrará en la tabla *titles*.

A veces con tener el modelo físico no nos alcanza. En la práctica, nos es muy útil echar un vistazo a los datos. Esto nos da una idea más acabada de cómo recuperar lo que nos solicitan.

En nuestro caso, vamos a disparar una consulta simple sobre *titles*:

```
SELECT * FROM titles
```

Results | Messages

	title_id	title	type	pub_id	price	advance	royalty	ytd_sales	notes
1	BU1032	The Busy Executive's Database Guide	business	1389	20,00	5000,00	10	4095	An overview of available datab
2	BU1111	Cooking with Computers: Surreptitious Balance Sheets	business	1389	11,95	5000,00	10	3876	Helpful hints on how to use yo
3	BU2075	You Can Combat Computer Stress!	business	0736	2,99	10125,00	24	18722	The latest medical and psycho
4	BU7832	Straight Talk About Computers	business	1389	19,99	5000,00	10	4095	Annotated analysis of what co
5	MC2222	Silicon Valley Gastronomic Treats	mod_cook	0877	19,99	0,00	12	2032	Favorite recipes for quick, eas

Nos solicitan código de título (*title_id*), título (*title*), tipo (*type*) de **todas** las publicaciones.
El hecho de que nos solicitan **todas** las publicaciones implica que no tendremos cláusula `WHERE` en la sentecnia `SELECT`.

El ejercicio nos pide además que el precio (columna *price*) se muestre incrementado en un 8%.
Este cálculo se puede hacer directamente en la lista de salida del `SELECT`, ya que allí podemos ubicar expresiones que involucren valores constantes y una o varias columnas.

El precio incrementado en un 8% sería entonces:

```
price * 1.08
```

...y esto puede ir directamente en la lista de salida del `SELECT`.

Por último, nos solicitan las publicaciones ordenadas por tipo y título. Esto significa que ordenamos las publicaciones por tipo y, si se presenta el caso de que dos tuplas o filas posean el mismo tipo, el motor de base de datos ordena este subconjunto por título.

Observe que la cuarta columna del conjunto resultado no posee encabezado. Este encabezado se puede definir proporcionando un alias para esa columna.

Alias de columna

Podemos agregar un alias a una columna calculada, como en este caso. Sin embargo, un alias se puede agregar a cualquier columna cuyo nombre deseemos abreviar o hacer más significativo.

Especificamos un alias directamente después del nombre de la columna. Recordemos que en el caso de que el alias incluya espacios, debemos encerrarlo entre comillas:

```
SELECT au_lname 'Apellido del autor', city ciudad
   FROM authors
```

Especificamos el alias usando la cláusula `AS`:

```
SELECT au_lname AS Apellido, city ciudad
      FROM authors
```

Si el alias de columna posee más de una palabra, además debemos encerrar el nombre entre comillas **dobles**:

```
SELECT au_lname AS "Apellido del autor", city ciudad
      FROM authors)
```

2. Reescriba la consulta del ejercicio 1 pero proporcionando el alias *precio actualizado* para la columna calculada.

Guardar nuestro código SQL en Scripts

Si guardamos nuestro código SQL en un archivo (Script SQL) podremos reproducir en cualquier momento algo que hicimos hace tiempo.

Por otro lado, luego de un tiempo de trabajar, el código SQL puede insumir muchas líneas en el editor, y será muy útil que esté debidamente documentado.

Por ejemplo:

Podemos escribir código "documentando" nuestro script usando un doble guión:

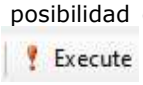
```
-------------------------------------------------------------------------
---------------------- E j e r c i c i o   1 ----------------------------
-------------------------------------------------------------------------.

SELECT * FROM titles
   WHERE type = 'business'
```

O podemos "documentar" varias líneas usando /* y */:

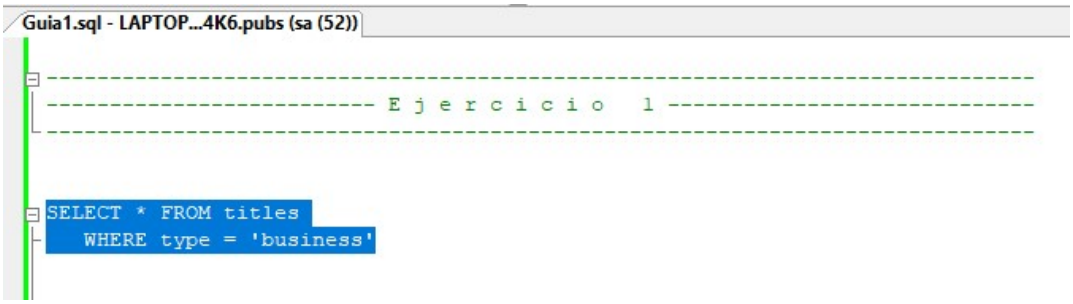
```
/*
----------------------- E j e r c i c i o   1 -------------------------
Buscar publicaciones de tipo business
*/

SELECT * FROM titles
   WHERE type = 'business'
```

Comentar código ya ejecutado

Por supuesto el uso primordial tanto del doble guión como de `/*` y `*/` es **encerrar código que no queremos ejecutar**, tal vez porque ya lo hemos ejecutado pero a la vez deseamos conservarlo en el script.

En el editor de SQL Server, por ejemplo, siempre tendremos la posibilidad de pintar con el mouse solamente lo que queremos ejecutar y luego hacer click en 

Por ejemplo:



Sin embargo, otras veces nos resultará más cómodo simplemente hacer click en 

.

Para eso debemos **comentar todo lo que no se deba ejecutar**, caso contrario **se ejecutará nuevamente**. En el siguiente ejemplo podemos ejecutar tranquilos todo lo que tenemos en el editor ya que todo el código previo ha sido comentado:

```
-------------------------------------------------------------------------
-------------------- E j e r c i c i o  1 ---------------------------
-------------------------------------------------------------------------

/*
SELECT * FROM titles
   WHERE type = 'business'
*/


/*
---------------------- E j e r c i c i o  2 ---------------------------
Buscar publicaciones de tipo business
*/


SELECT * FROM titles
   WHERE type = 'business'
```

El modificador `DESC` afecta el orden de las columnas especificadas en una cláusula `ORDER BY`.

3. Modifique la consulta del ejercicio 2 a fin de obtener los datos por orden descendente de precio actualizado.

Guía para resolver la consulta

Aquí tenemos el problema de que tenemos que ordenar la salida SQL por una "columna" que no es en realidad una columna sino una expresión.
Tenemos:

```
SELECT title_id, title, type, price * $1.08 'precio actualizado'
```

Podemos directamente expresar el alias en la cláusula `ORDER BY`:

```
ORDER BY 'precio actualizado' DESC
```

4. Es posible expresar el número de orden en la lista de salida del `SELECT` para identificar la columna sobre la cual se desea ordenar. Por ejemplo: `ORDER BY 5`. Reescriba la consulta de esta forma.

Guía para resolver la consulta

Es la segunda alternativa para hacer lo mismo:

```
ORDER BY 4 DESC
```

Constantes en la lista de salida del `SELECT`

Podemos especificar valores literales (fijos) como parte de la lista de salida del `SELECT`.

La siguiente sentencia SQL utiliza el operador de concatenación + para generar un conjunto resultado con una única columna de salida:

```
SELECT 'El apellido del empleado es ' + lname  'Datos del empleado'
   FROM employee
```

El operador de concatenación en PostgreSQL es ||. Los literales en la lista de salida del SELECT deben encerrarse entre comillas simples:

```
SELECT 'El apellido del empleado es ' || lname
       AS "Datos del empleado"
   FROM employee
```

5. Obtenga en una única columna el apellido y nombres de los autores separados por coma con una cabecera de columna *Listado de Autores*. Ordene el conjunto resultado.

Guía para resolver la consulta

En este caso nos piden información sobre autores, así que sabemos que la misma se encontrará en la tabla *authors*.

Le echamos un vistazo a los datos:

```
SELECT * FROM authors
```

	au_id	au_lname	au_fname	phone	address	city	state	zip	contract
1	172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA	94025	1
2	213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA	94618	1
3	238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley	CA	94705	1
4	267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #14	San Jose	CA	95128	1
5	274-80-9391	Straight	Dean	415 834-2919	5420 College Av.	Oakland	CA	94609	1

Nos solicitan apellido y nombre de **todos** los autores.
Nuevamente, el hecho de que nos solicitan todos los autores implica que no tendremos cláusula `WHERE` en la sentencia `SELECT`.

Podríamos armar algo así:

```
SELECT au_lname, au_fname FROM authors
```

...pero nos solicitan una única columna. Así que tendríamos que concatenar la salida de ambas columnas, separadas por un caracter. El ejercicio pide que el separador sea una coma:

```
SELECT au_lname + ', ' + au_fname
```

Luego agregamos el alias solicitado.

Por último ordenamos la única columna como solicita el ejercicio.

Los motores de bases de datos muchas veces proporcionan conversión entre tipos de datos automática. Otras veces, tenemos que hacer nosotros mismos una conversión explícita entre tipos.

Siempre es más seguro que tengamos el control total sobre el código, y esto lo logramos haciendo la conversión explícita entre tipos de datos.

6. Obtenga un conjunto resultado para la tabla de publicaciones que proporcione, para cada fila, una salida como la siguiente.

¿Qué sucede?

	(Sin nombre de columna)
1	BU1032 posee un valor de $19.99
2	BU1111 posee un valor de $11.95
3	BU2075 posee un valor de $2.99
4	BU7832 posee un valor de $19.99
5	MC2222 posee un valor de $19.99
6	MC3021 posee un valor de $2.99

Guía para resolver la consulta

En este caso nos piden nuevamente información sobre publicaciones.

Por lo que vemos en la captura, nos solicitan que obtengamos el código de publicación concatenado con una cadena literal y concatenado con el precio de la publicación.

Acá necesitamos asegurarnos de qué tipo de dato es cada columna. Podemos volver al script de creación de *pubs* para averiguarlo. También podemos usar el Administrador Corporativo. Pero más rápido es usar un stored procedure del sistema que proporciona SQL Server y que brinda esta información: **sp_help**.

Por ejemplo, para *titles* ejecutamos:

```
sp_help titles
```

y obtenemos:

	Name	Owner	Type	Created_datetime
1	titles	dbo	user table	2019-02-11 02:03:43.050

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLenNullInSource	Co
1	title_id	tid	no	6			no	no	no	M
2	title	varchar	no	80			no	no	no	M
3	type	char	no	12			no	no	no	M
4	pub_id	char	no	4			yes	no	yes	M
5	price	money	no	8	19	4	yes	(n/a)	(n/a)	NU
6	advance	money	no	8	19	4	yes	(n/a)	(n/a)	NU

Aquí vemos que el tipo de dato de *title_id* es *tid* (un tipo definido por el usuario, que resulta siendo un varchar).
Vemos que el tipo de dato de *price* es money (que es un tipo de valor numérico).

Si probamos la consulta sin conversión explícita:

```
SELECT title_id + ' posee un valor de $' + price
```

...obtenemos:

Results Messages

```
Msg 235, Level 16, State 0, Line 1
Cannot convert a char value to money. The char value has incorrect syntax.
```

Conversión de datos numéricos a caracter

Podemos convertir datos numéricos a caracter de manera explícita usando la función `CAST`:

```
CAST (columna-a-convertir AS tipo-de-dato-destino)
```

Por ejemplo:

```
SELEct CAST(price AS varchar)
   from titles
```

T-SQL también proporciona la función T-SQL `convert()`. La sintaxis de uso es:

```
convert (tipo-de-dato-destino, columna-a-convertir)
```

Por ejemplo:

```
SELECT CONVERT(varchar, price)
   from titles
```

PostgreSQL también proporciona la siguiente sintaxis:

```
columna-a-convertir::tipo-de-dato-destino
```

Por ejemplo:

```
SELECT price::varchar(5)
   FROM titles
```

7. Reescriba el ejercicio 6 utilizando estas variantes explícitas de conversión de tipos.

Guía para resolver la consulta

En SQL Server la columna que provoca problemas es *price*, de tipo money, ya que *title_id* es de tipo varchar y `' posee un valor de $'` es una string literal.

Usamos la función de conversión `CONVERT`:

```
SELECT title_id + ' posee un valor de $' + CONVERT(varchar, price)
```

Otra alternativa es usar `CAST`:

```
SELECT title_id + ' posee un valor de $' + CAST(price AS varchar)
```

2. La cláusula WHERE

Hasta ahora no hemos especificado cláusulas `WHERE`. Las cláusulas `WHERE` nos permiten especificar una **condición** que las filas deben cumplir a fin de formar parte de la lista de salida del `SELECT`.

Operadores

Para especificar las **condiciones** de las cláusulas `WHERE` necesitamos de operadores relacionales y lógicos.

Los operadores relacionales son >, >=, <, <=, = y <>.

Los operadores lógicos son `AND`, `OR` y `NOT`.

8. Obtenga título y precio de todas las publicaciones que no valen más de `$13`. Pruebe definir la condición de `WHERE` con el operador `NOT`.

Guía para resolver la consulta

Nuevamente nos piden información sobre publicaciones.

El ejercicio solicita título y precio. Esto significa que tendré solo esas dos columnas en la lista de salida (*title* y *price*).

La primer cláusula `WHERE` que se nos ocurre es la que utiliza los operadores de comparación:

```
WHERE price <= 13
```

El ejercicio nos pide usar el operador `NOT`. Deberíamos expresar la afirmación opuesta y negarla:

```
WHERE NOT price > 13
```

Fechas en cláusulas `WHERE`

Las fechas se especifican entre comillas simples. El formato depende de la configuración del motor de base de datos. Un formato usual es `mm/dd/aaaa`.

El predicado `BETWEEN`

Recordemos que el predicado `BETWEEN` especifica la comparación dentro de un intervalo entre valores cuyo tipos de datos son comparables:

```
WHERE precio BETWEEN 5 AND 10
```

La cláusula `NOT` se puede utilizar con `BETWEEN` para indicar que la condición debe evaluar contra lo que existe fuera del intervalo:

```
WHERE precio NOT BETWEEN 5 AND 10
```

9. Obtenga los apellidos y fecha de contratación de todos los empleados que fueron contratados entre el `01/01/1991` y el `01/01/1992`. Use el predicado `BETWEEN` para elaborar la condición.

Guía para resolver la consulta

Nos solicitan información sobre empleados, así que vamos a la tabla *Employee*.

```
SELECT * FROM employee
```

Results | Messages

	emp_id	fname	minit	lname	job_id	job_lvl	pub_id	hire_date
1	PMA42628M	Paolo	M	Accorti	13	35	0877	1992-08-27 00:00:00.000
2	PSA89086M	Pedro	S	Afonso	14	89	1389	1990-12-24 00:00:00.000
3	VPA30890F	Victoria	P	Ashworth	6	140	0877	1990-09-13 00:00:00.000
4	H-B39728F	Helen		Bennett	12	35	0877	1989-09-21 00:00:00.000
5	L-B31947F	Lesley		Brown	7	120	0877	1991-02-13 00:00:00.000
6	F-C16315M	Francisco		Chang	4	227	9952	1990-11-03 00:00:00.000
7	PTC11962M	Philip	T	Cramer	2	215	9952	1989-11-11 00:00:00.000

Las columnas solicitadas son apellido (*lname*) y fecha de contratación (*hire_date*). Debemos expresar una consulta comparando fechas.

Una solución posible es:

```
WHERE hire_date > '01/01/1991' and
      hire_date < '01/01/1992'
```

Pero el ejercicio nos pide usar `BETWEEN`:
La solución sería entonces:

```
WHERE hire_date BETWEEN '01/01/1991' and '01/01/1992'
```

El operador `[NOT] IN`

Recordemos que el operador `IN` especifica la comparación cuantificada: hace una lista de un conjunto de valores y evalúa si un valor está en la lista. La lista debe expresarse entre paréntesis:

```
WHERE precio IN (25, 30)
```

10. Obtenga los códigos, domicilio y ciudad de los autores con código `172-32-1176` y `238-95-7766`. Utilice el operador `IN` para definir la condición de búsqueda.
Modifique la consulta para obtener todos los autores que no poseen esos códigos.

Guía para resolver la consulta

Nos solicitan información sobre autores, así que trabajamos sobre la tabla authors.

```
SELECT * FROM authors
```

Results | Messages

	au_id	au_lname	au_fname	phone	address	city
1	172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park
2	213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland
3	238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley
4	267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #14	San Jose

Nos solicitan código (*au_id*), domicilio (*address*) y ciudad (*city*) de los autores que se encuentran en el "conjunto" de autores que posee dos elementos: `172-32-1176` y `238-95-7766`.

La cláusula `WHERE` se expresaría así:

```
WHERE au_id IN ('172-32-1176', '238-95-7766')
```

Otra forma de obtener lo mismo:

```
WHERE au_id = '172-32-1176' OR
      au_id = '238-95-7766'
```

La negación (todos los autores que no son estos dos), sería:

```
WHERE au_id NOT IN ('172-32-1176', '238-95-7766')
```

El predicado `LIKE`. Caracteres comodín.

Recordemos que el predicado `LIKE` permite especificar una comparación de caracteres utilizando caracteres comodín (wild cards) para recuperar filas cuando solo se conoce un patrón del dato buscado. `LIKE` generalmente se utiliza sobre columnas de tipo caracter.

Los caracteres comodín generalmente soportados son:

`%`	0 a n caracteres (de cualquier caracter).
	Exactamente un caracter (de cualquier caracter).
`[]`	Exactamente un caracter del conjunto o rango especificado, por ej.: `[aAc]` o `[a-c]`

11. Obtenga código y título de todos los títulos que incluyen la palabra `Computer` en su título.

Guía para resolver la consulta

Nos solicitan información sobre publicaciones, así que trabajamos sobre la tabla *titles*.

El título de la publicación es la columna *title*. Debemos usar el predicado `LIKE` para recuperar las publicaciones que incluyen la palabra `computer`.
La solución sería:

```
WHERE title LIKE '%computer%'
```

El pattern `%computer%` significa:

Cualquier cantidad de caracteres de cualquier caracter,
a continuación la string computer,
a continuación cualquier cantidad de caracteres de cualquier caracter.

Valores `NULL`

Un valor `NULL` para una columna indica que el valor para esa columna es desconocido. No es lo mismo que blanco, cero o cualquier otro valor discreto. A veces puede significar que el valor para una columna particular no es significativo.

Los valores `NULL` son casos especiales y deben tratarse en forma especial cuando se realizan comparaciones.
A los propósitos del ordenamiento, los valores `NULL` son considerados los más bajos.

12. Obtenga el nombre, ciudad y estado de las editoriales cuyo estado (columna `state`) no está definido. Recordemos que para ello debemos utilizar la cláusula `IS` (o su negación `IS NOT`)
¿Qué sucede si explícitamente compara la columna contra el valor `NULL`?.

Guía para resolver la consulta

Nos solicitan información sobre editoriales, así que trabajamos sobre la tabla *publishers*.

Debemos listar el nombre (columna *pubname*), la ciudad (columna *city*) y el estado (columna *state*) de las editoriales que posean valor `NULL` en su columna *state*.

Si no supiésemos que el valor `NULL` se trata de una manera especial, intentaríamos los siguiente:

```
WHERE state = NULL
```

Esta cláusula no dispara ningún error, pero no devuelve lo que esperamos.

La forma correcta es la siguiente:

```
WHERE state IS NULL
```

3. Limitar la cantidad de tuplas

Obtenemos las *n* primeras tuplas de un query SQL utilizando el modificador `TOP`. Por ejemplo:

```
SELECT TOP 1 *
  FROM titles
```

En PostgreSQL obtenemos un comportamiento similar usando la cláusula `LIMIT`. Por ejemplo:

```
SELECT *
   FROM titles
   LIMIT 1;
```

4. Funciones

4.1. Fechas

Componentes de una fecha

`YEAR(columna)` retorna el año de la fecha como un entero de cuatro dígitos, `MONTH(columna)` proporciona el mes de la fecha como un entero de `1` a `12`. `DAY (columna)` retorna el día del mes de la fecha como un entero.

En PostgreSQL debemos usar la funcion `date_part`. Por ejemplo

`date_part('year', columna)` retorna el año de la fecha como un número de doble precisión.

`date_part('month', columna)` retorna el mes de la fecha como un número de doble precisión.

`date_part('day', columna)` retorna el mes de la fecha como un número de doble precisión.

Podemos obtener el mismo resultado con la función `EXTRACT`. Por ejemplo:

`EXTRACT('year' FROM columna)` retorna el año de la fecha como un número de doble precisión.

`EXTRACT ('month' FROM columna)` retorna el mes de la fecha como un número de doble precisión.

`EXTRACT ('day' FROM columna)` retorna el mes de la fecha como un número de doble precisión.

13. La información de publicaciones vendidas reside en la tabla `Sales`. Liste las filas de ventas correspondientes al mes de Junio de cualquier año.

Guía para resolver la consulta

Nos solicitan información sobre ventas, así que trabajamos sobre la tabla *Sales*.

Observamos un poco los datos en *Sales*:

```
SELECT * FROM Sales
```

Results | Messages

	stor_id	ord_num	ord_date	qty	payterms	title_id
1	6380	6871	1994-09-14 00:00:00.000	5	Net 60	BU1032
2	6380	722a	1994-09-13 00:00:00.000	3	Net 60	PS2091
3	7066	A2976	1993-05-24 00:00:00.000	50	Net 30	PC8888
4	7066	QA7442.3	1994-09-13 00:00:00.000	75	ON invoice	PS2091
5	7067	D4482	1994-09-14 00:00:00.000	10	Net 60	PS2091
6	7067	P2121	1992-06-15 00:00:00.000	40	Net 30	TC3218
7	7067	P2121	1992-06-15 00:00:00.000	20	Net 30	TC4203

Debemos listar todas las tuplas que correspondan a ventas del mes de Junio.
La fecha de venta reside en la columna `ord_date`.
Así que tendremos que obtener el mes de esa fecha y compararlo contra Junio, que corresponde al número 6.

Así que tendríamos:

```
WHERE MONTH (ord_date) = 6
```

Fecha actual

`CURRENT_TIMESTAMP` retorna la fecha y hora actual como un valor `datetime`. Se invoca sin paréntesis.

PostgreSQL proporciona también la función `CURRENT_TIMESTAMP` que retorna la fecha y hora actuales.

La función `now()` retorna también el mismo resultado.

Fechas como texto

Ya vimos que podíamos convertir el dato de una columna a un tipo destino a través de la función `convert()`. `convert()` posee una versión extendida que permite convertir datos de columnas de tipo `datetime` a diferentes formatos de visualización de texto. La sintaxis es:

```
convert (varchar, columna-datetime, codigo-de-formato)
```

`codigo-de-formato` es un código que establece como se va a mostrar la fecha en formato varchar. Por ejemplo, el formato `3` muestra la fecha con formato `dd/mm/yyyy`:

```
SELECT CONVERT(varchar, hire_date, 3)
   FROM Employee
```

4.2. Strings

Subcadenas

La función `substring()` extrae una subcadena de una cadena principal. Su sintaxis es:

```
substring (columna-o-expresion, desde, cantidad)
```

`columna-o-expresion` es la cadena desde la cual se extraerán los caracteres. `desde` es la posición de inicio de la extracción. `substring()` retorna una cadena de tipo `varchar`. Por ejemplo:

```
SELECT substring(title,5,4)
    FROM titles
```

Conversión de números a strings

La función `str()` convierte una expresión numérica a una cadena. Su sintaxis es:

```
str (expresion-numerica, longitud-total, cantidad-decimales)
```

donde `longitud` es la longitud total incluyendo la coma y las cifras decimales. Por ejemplo:

```
SELECT STR(price, 5, 2)
   from titles
```

En PostgreSQL utilizamos la función `CAST` ya vista

Otras funciones de manejo de Strings

T-SQL soporta también las funciones manejo de texto `right()`, `upper()`, `lower()`, `rtrim()` y `ltrim()`.

PostgreSQL soporta `upper()` y `lower()`.

Proporciona la función `TRIM`, que permite eliminar los caracteres especificados del principio, final o principio y final de una cadena. Por ejemplo, para eliminar cualquier caracter `'E'` al principio de la cadena:

```
SELECT TRIM(LEADING 'E' FROM title)
   from titles
```

Para eliminar cualquier caracter `'s'` al final de la cadena:

```
SELECT TRIM(TRAILING 's' FROM title)
   from titles
```

y para eliminar cualquier caracter `'s'` al principio o final de la cadena:

```
SELECT TRIM(BOTH 's' FROM title)
   from titles
```

En cualquier caso, si se omite el caracter a eliminar, se asume espacio.