Capítulo 6

Consultas básicas con lenguaje SQL

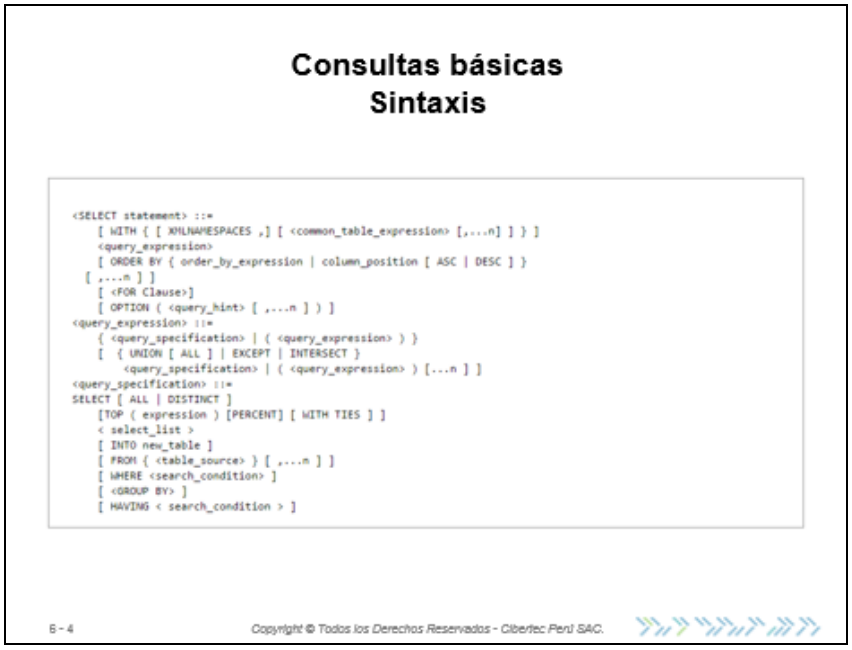
Al finalizar el capítulo, el alumno podrá:

- Elaborar consultas en la base de datos, mediante el correcto empleo de la sentencia SELECT.

Temas:

1. Consultas básicas
2. Consultas condicionales
3. Funciones de librería

1. Consultas básicas



1.1 ¿Objetivo de las consultas básicas o de selección?

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos; esta información es devuelta en forma de conjunto de registros que se pueden almacenar, por ejemplo, en un objeto **datatable** o **datareader**.

1.2 Parámetros básicos del comando SELECT

La cláusula SELECT lista los datos a recuperar por la sentencia SELECT. Los elementos o datos por seleccionar pueden ser columnas de la tabla o columnas a calcular por SQL, cuando se efectúa la consulta o también el asterisco (*****) para recuperar todos los campos de una tabla.

La sintaxis básica del comando se muestra a continuación.

```
SELECT [ALL|DISTINCT]
          {* | expr_1 [AS c_alias_1] ……[, expr_k [AS c_alias_k]]]}
   FROM table_name_1 [t_alias_1]
        [, ... [, table_name_n [t_alias_n]]]
   [WHERE condition]
```

Donde:

- **expr_1** puede ser un simple nombre de campo (por ejemplo: LastName). Expresiones más complejas pueden incluir operaciones matemáticas o de manipulación de caracteres (por ejemplo: FirstName + ‘, ‘+ LastName).

- Las expresiones de columnas deben ir separadas por comas, si existen más de una (por ejemplo: FirstName, LastName, TitleOfCourtesy).

- Los nombres de campos pueden ir precedidos por el nombre de la tabla o su alias. Por ejemplo: Employees.FirstName o E.FirstName donde E es el alias para la tabla o fichero Employees. Por ejemplo:

```
SELECT Employees.FirstName FROM Employees

SELECT E.FirstName FROM Employees E
```

1.3 Cláusulas de importancia del comando SELECT

- **Cláusula FROM**

  Lista las tablas que contienen los datos a recuperar por la consulta. El formato de esta cláusula es:

```
SELECT * FROM NombreTabla [Alias_Tabla]......
```

  NombreTabla puede ser uno o más nombres de tabla en el directorio de trabajo si se omite este o en un directorio distinto si se especifica.

  Alias_Tabla es un nombre que se usa para referirse a la tabla en el resto de la sentencia SELECT para abreviar el nombre original y hacerlo más manejable, en el caso de existir más de una tabla en la consulta y también para realizar consultas uniendo varias veces la misma tabla.

  Por ejemplo:

```
SELECT  C.CategoryName, P.ProductName
  FROM  Products P,
        Categories C
 WHERE  P.CategoryID = C.CategoryID
   AND  C.CategoryID = 1
```

  Es mucho más práctico y sencillo.

```
SELECT  Categories.CategoryName,
        Products.ProductName
  FROM  Products,
        Categories
 WHERE  Products.CategoryID = Categories.CategoryID
   AND  Categories.CategoryID = 1
```

  Las dos sentencias son idénticas y devolverían el nombre de la categoría y el producto en donde el número de ID de la categoría sea igual a 1.

  El nombre de tablas junto al nombre de campo es obligatorio cuando existen campos con nombre idéntico en las tablas que formen parte de la sentencia. En el ejemplo anterior CategoryName y ProductName no lo necesitarían, pero CategoryID sí, porque en las dos tablas existe un campo con ese nombre.

- **Cláusula WHERE**

  Esta cláusula dice a SQL, que incluya solo ciertas filas o registros de datos en los resultados de la consulta, es decir, que tienen que cumplir los registros que se desean ver. Contiene condiciones en la forma:

```
WHERE Expresión1 operador Expresion2
```

  Expresión1 y Expresión2 pueden ser nombres de campos, valores constantes o expresiones.

  Operador es un operador relacional que une dos expresiones.

  Por ejemplo, la siguiente sentencia muestra el número de empleados que viven en la ciudad de 'LONDON'.

```
SELECT  COUNT(*)
  FROM  Employees
 WHERE  City = 'London'
```

- **Cláusula ORDER BY**

  Ordena los resultados de la consulta en base a los datos de una o más columnas. Si se omite, los resultados saldrán ordenados por el primer campo que sea clave en el índice que se haya utilizado.

  Por tanto, indica cómo deben clasificarse los registros que se seleccionen. Tiene la forma:

```
SELECT  *
  FROM  [TABLA]
 ORDER  BY {Expresión_orden [DESC | ASC],... ]
```

  Expresión_orden puede ser el nombre de un campo, expresión o el número de posición que ocupa la expresión de columna en la cláusula SELECT. Por defecto, se ordenan ascendentemente (de menor a mayor). Sin embargo, si se desea de mayor a menor, se empleará DESC (Descendente).

  Por ejemplo, para mostrar los empleados ordenados por fecha de nacimiento de mayor edad a menor, se utilizaría:

```
SELECT  *
  FROM  Employees
 ORDER  BY BirthDate DESC
```

  Para obtener un listado de empleados ordenado por primer nombre y apellidos, se utilizaría:

```
SELECT  FirstName, LastName
  FROM  Employees
 ORDER  BY FirstName, LastName
```

O, lo mismo de otra forma:

```
SELECT  FirstName, LastName
  FROM  Employees
 ORDER  BY 1, 2
```

Donde los números son la posición actual de los campos mostrados en la cláusula SELECT.

- **Cláusula ALL**

  Si no se incluye ninguno de los predicados se asume ALL.

  El motor de base de datos selecciona todos los registros que cumplen las condiciones de la instrucción SQL, pero no es conveniente abusar de este predicado, ya que obliga al motor de la base de datos, a analizar la estructura de la tabla para averiguar los campos que contiene, es mucho más rápido indicar el listado de campos deseados.

```
SELECT ALL * FROM Products

/*Que sería lo mismo a:*/

SELECT * FROM Products
```

- **Cláusula OFFSET-FETCH**

  Devuelve una "ventana de registros" de los resultados. OFFSET especifica cuantos registros tiene que saltarse dentro del resultado y FETCH especifica cuantos registros desde el punto en adelante tiene que devolver.

  Por ejemplo, nos piden listar los 10 registros más caros exceptuando los 5 primeros.

```
SELECT  ProductName, UnitPrice
  FROM  Products
 ORDER  BY UnitPrice DESC
OFFSET  5 ROWS FETCH NEXT 10 ROWS ONLY
```

  Puede usar OFFSET sin FETCH, pero FETCH no se puede usar solo.

```
SELECT  ProductName, UnitPrice
  FROM  Products
 ORDER  BY UnitPrice DESC
OFFSET  5 ROWS
```

- **Cláusula TOP**

  Devuelve un cierto número de registros que entran al principio o al final de un rango especificado por una cláusula ORDER BY.

  Por ejemplo, suponiendo que se quiere recuperar los 25 primeros pedidos:

```
SELECT  TOP 25 OrderID, CustomerID, OrderDate, RequiredDate
  FROM  Orders
 ORDER  BY OrderDate DESC
```

Si no se incluye la cláusula ORDER BY, la consulta devolverá un conjunto arbitrario de 25 registros de la tabla **Orders**. El predicado TOP no elige entre valores iguales.

Se puede utilizar la palabra reservada PERCENT para devolver un cierto porcentaje de registros que caen al principio o al final de un rango especificado por la cláusula ORDER BY.

Suponiendo que en lugar de los 25 primeros pedidos se desea el 10 por ciento de los pedidos:

```
SELECT  TOP 10 PERCENT OrderID, CustomerID, OrderDate, RequiredDate
  FROM  Orders
 ORDER  BY OrderDate DESC
```

Por otro lado, con la cláusula WITH TIES permite mostrar los registros que hayan sido limitados usando la opción TOP pero tienen un valor igual al último registro que aparece.

Por ejemplo, nos piden listar los 11 productos ordenamos en forma descendente e incluir los que tengan el mismo precio que el onceavo registro.

```
SELECT  TOP 11 WITH TIES ProductID, ProductName, UnitPrice
  FROM  Products
   ORDER  BY UnitPrice DESC
```

- **Cláusula DISTINCT**

  Omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción SELECT se incluyan en la consulta, deben ser únicos.

  Por ejemplo, varios empleados listados en la tabla **Employees** pueden tener el mismo título (Title). Si dos o más registros contienen “Sales Representative” en el campo Title, la siguiente instrucción SQL devuelve un único registro:

```
SELECT  DISTINCT Title
  FROM  Employees
```

  Con otras palabras, el predicado DISTINCT devuelve aquellos registros cuyos campos indicados en la cláusula SELECT posean un contenido diferente. El resultado de una consulta que utiliza DISTINCT no es actualizable y no refleja los cambios subsiguientes realizados por otros usuarios.

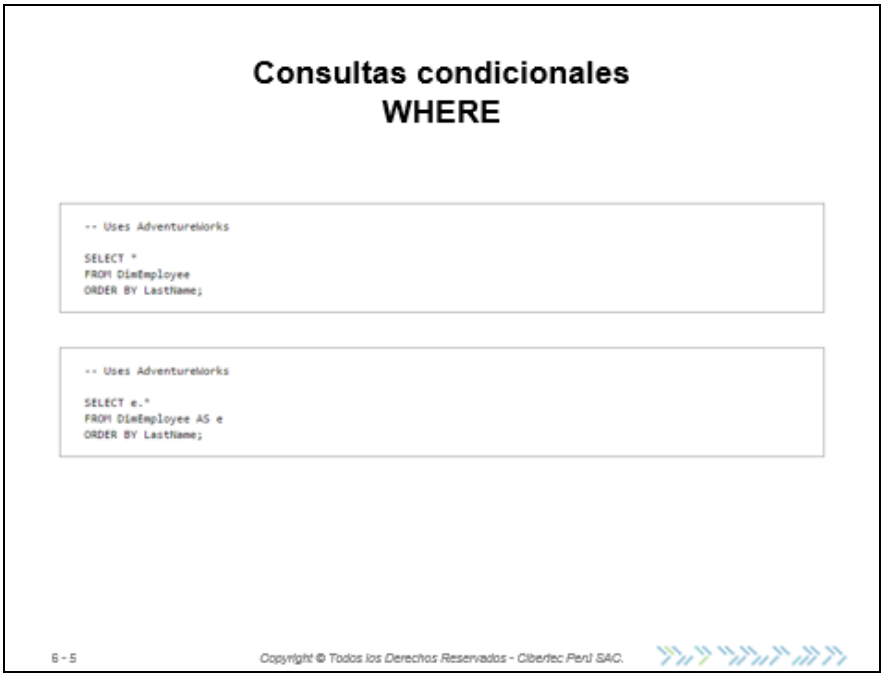
1.4 Recuperando información de otra base de datos

En ocasiones es necesario recuperar información que no se encuentra contenida en la base de datos que actualmente está en uso. Esta situación se puede salvar de la siguiente forma:

```
SELECT  CategoryID, CategoryName
  FROM  TSQL2016.dbo.Categories
```

Donde **TSQL2016** es la base de datos que contiene la tabla **Categories**.

2. Consultas condicionales



2.1 El parámetro WHERE

Cada vez que desee establecer una condición referida a un campo de texto, se debe tomar en cuenta 3 condiciones:

- La búsqueda debe ir encerrada entre comillas simples.
- No es posible establecer condiciones de búsqueda en los campos memo.
- Las fechas se deben escribir siempre, según el formato de configuración del sistema.

Para establecer condiciones en el comando SELECT se emplea el parámetro WHERE, el cual, en combinación con operando y operadores de diferente tipo, permiten filtrar los resultados de las consultas.

2.2 Operadores y cláusulas

2.2.1 Operadores lógicos

Operador	Uso
AND	Es el "**y**" lógico. Evalúa dos condiciones y devuelve un valor de verdad, solo si ambas son ciertas.
OR	Es el "**o**" lógico. Evalúa dos condiciones y devuelve un valor de verdadero, si alguna de las dos es cierta.
NOT	Negación lógica. Devuelve el valor contrario de la expresión.

La sintaxis es la siguiente:

```
      <expresión1>    operador    <expresión2>
```

En donde **expresión1** y **expresión2** son las condiciones a evaluar, el resultado de la operación varía en función del operador lógico. La tabla adjunta muestra los diferentes posibles resultados:

<expresión1>	Operador	<expresión2>	Resultado
Verdad	AND	Falso	Falso
Verdad	AND	Verdad	Verdad
Falso	AND	Verdad	Falso
Falso	AND	Falso	Falso
Verdad	OR	Falso	Verdad
Verdad	OR	Verdad	Verdad
Falso	OR	Verdad	Verdad
Falso	OR	Falso	Falso

- Si a cualquiera de las anteriores condiciones se le antepone el operador **NOT**, el resultado de la operación será el resultado contrario.

- Dos o más condiciones pueden ser combinadas para formar expresiones más complejas con distintos criterios. Cuando existen dos o más condiciones deberán estar unidas por **AND** u **OR**.

Ejemplos de operadores lógicos:

```
/* Empleados que no sean de la Ciudad de London */
SELECT  *
  FROM  Employees
 WHERE  NOT City = 'London'
```

```
/* Productos con precio entre 20.00 y 40.00 o que las unidades en
   stock sea mayor igual que 100 */

SELECT  *
  FROM  Products
 WHERE  (UnitPrice >= 20.00 AND UnitPrice <= 40.00)
    OR  (UnitsInStock >= 100)
```

2.2.2 Operadores numéricos

Para realizar operaciones numéricas se pueden utilizar los operadores que se muestran en la tabla.

Operador	Significado
+	Suma
-	Resta
*****	Multiplicación
/	División
%	Devuelve el resto entero de una división

Ejemplos de operadores numéricos

```
/* Mostrar los datos de un producto, además del TAX y el precio final */

SELECT  ProductID, ProductName, UnitPrice,
        TAX = UnitPrice * 0.10,
        FinalPrice = (UnitPrice * 0.10) + UnitPrice
  FROM  Products
 WHERE  UnitPrice > 0.00
```

2.2.3 Operadores de caracteres

Las expresiones de caracteres pueden incluir los siguientes operadores.

Operador	Significado
+	Concatenación manteniendo espacios en blanco

Ejemplos de operadores de caracteres

```
/* Mostrar el FirstName y LastName de la persona en una sola columna */

SELECT  EmployeeID, Name = LastName + ' ' + FirstName
    FROMEmployees
```

2.2.4 Operadores de relación

Operador	Uso
<	Menor que
>	Mayor que
<>	Distinto de
<=	Menor o Igual que
>=	Mayor o Igual que
=	Igual que
BETWEEN	Utilizado para especificar un intervalo de valores
LIKE	Utilizado en la comparación de un modelo
In	Utilizado para especificar registros de una base de datos

Ejemplos de operadores de relación:

```
/* Mostrar todos los datos de los productos en donde las unidades en
   stock sea menor que 50 */

SELECT  *
  FROM  Products
 WHERE  UnitsInStock < 50
```

```
/* Mostrar todos los datos de los empleados donde la primera letra de su
   apellido sea desde la 'A' hasta la 'L' */

SELECT  *
  FROM  Employees
 WHERE  LEFT(FirstName, 1) < 'M'
```

```
/* Mostrar todos los datos de las personas donde la primera letra de su
   apellido sea diferente de 'Z' */

SELECT  *
  FROM  Employees
 WHERE  LEFT(FirstName, 1) <> 'Z'
```

2.2.5 Comparando fechas

```
/* Mostrar todos los datos de los empleados que la fecha de contratación
   se haya realizado en el año 1992 */

SELECT  *
  FROM  Employees
 WHERE  YEAR(HireDate) = 1992
```

```
/* Mostrar todos los datos de los empleados que la fecha de contratación
   se haya realizado antes de mayo del 1992 */

SELECT  *
  FROM  Employees
 WHERE  YEAR(HireDate) = 1992
   AND  MONTH(HireDate) < 5
```

```
/* Mostrar todos los datos de los empleados que la fecha de contratación
    sea antes del 1994-01-01 */

SELECT  *
  FROM  Employees
 WHERE  HireDate < '1994-01-01'
```

2.2.6 Cláusula BETWEEN

Para indicar que se desean recuperar los registros, según el intervalo de valores de un campo, se empleará el operador BETWEEN, cuya sintaxis es:

```
CAMPO [NOT] BETWEEN VALOR1 AND VALOR2
  (LA CONDICIÓN NOT ES OPCIONAL)
```

En este caso, la consulta devolvería los registros que contengan en **campo** un valor incluido en el intervalo **valor1**, **valor2** (ambos inclusive). Pero, si se antepone la condición **NOT**, esta devolverá aquellos valores no incluidos en el intervalo.

Ejemplos con BETWEEN:

```
/* Lista todos los pedidos que la fecha del pedido sea mayor o igual que
   El 01 de julio de 1996 y menor igual que 31 de diciembre de 1996 */

SELECT  *
  FROM  Orders
 WHERE  OrderDate BETWEEN '1996-07-01' AND '1996-12-31'
```

```
/* Lista todos los productos en donde la categoría no esté en el
   Rango de 1 a 4 */

SELECT  *
  FROM  Products
 WHERE  NOT CategoryID BETWEEN 1 AND 4
```

2.2.7 Cláusula IN

Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de una lista. Su sintaxis es:

```
  EXPRESIÓN [NOT] IN (VALOR1, VALOR2,...)
```

Si se tiene la siguiente tabla **Purchasing.PurchaseOrderHeader** con la siguiente información.

ProductID	ProductName	UnitPrice	UnitsInStock
1	Chai	18.00	39
2	Chang	19.00	17
3	Aniseed Syrup	10.00	13
4	Chef Anton's Cajun Seasoning	22.00	53
5	Chef Anton's Gumbo Mix	21.35	0
6	Grandma's Boysenberry Spread	25.00	120
7	Uncle Bob's Organic Dried Pears	30.00	15

Se muestra solamente pedidos **1**, **2** y **3**.

```
SELECT  ProductID, ProductName, UnitPrice, UnitsInStock
  FROM  Products
 WHERE  ProductID IN (1, 2, 3)
```

El resultado obtenido sería el siguiente:

ProductID	ProductName	UnitPrice	UnitsInStock
1	Chai	18.00	39
2	Chang	19.00	17
3	Aniseed Syrup	10.00	13

El ejemplo anterior es completamente equivalente a:

```
SELECT  ProductID, ProductName, UnitPrice, UnitsInStock
  FROM  Products
 WHERE  (ProductID = 1) OR (ProductID = 2)
    OR  (ProductID = 3)
```

Si se desea mostrar a todos los empleados que no vivan en la ciudad de Tacoma, ni London; la consulta sería la siguiente.

```
SELECT  *
  FROM  Employees
 WHERE  City NOT IN ('Tacoma', 'London')
```

2.2.8 Cláusula LIKE

Se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis es:

```
EXPRESIÓN LIKE MODELO
```

En donde **modelo** es una cadena que es comparada con **expresión**.

Se puede utilizar el operador **LIKE** para encontrar valores en los campos que coincidan con el modelo especificado. Por **modelo** también se puede especificar un valor completo (Michael), o se pueden utilizar caracteres comodines, para encontrar un rango de valores de la siguiente forma.

```
SELECT  *
  FROM  Employees
 WHERE  FirstName LIKE 'Mi%'
```

A continuación, se presentan los distintos caracteres comodines para posteriormente, comprobar las expresiones con diferentes modelos.

Carácter comodín	Descripción
%	Cualquier cadena de cero o más caracteres.
_ (subrayado)	Cualquier carácter individual.
[ ]	Cualquier carácter individual de intervalo ([a-f]) o del conjunto ([abcdef]) especificado.
[^]	Cualquier carácter individual que no se encuentre en el intervalo ([^a-f]) o el conjunto ([^abcdef]) especificado.

El operador **LIKE** se puede utilizar en una expresión para comparar un valor de un campo con una expresión de cadena.

Por ejemplo, si se introduce **LIKE C%** en una consulta SQL, la consulta devuelve todos los valores de campo que comiencen por la letra **C**. En una consulta con parámetros, se puede hacer que el usuario escriba el modelo que se va a utilizar.

Ejemplos con LIKE

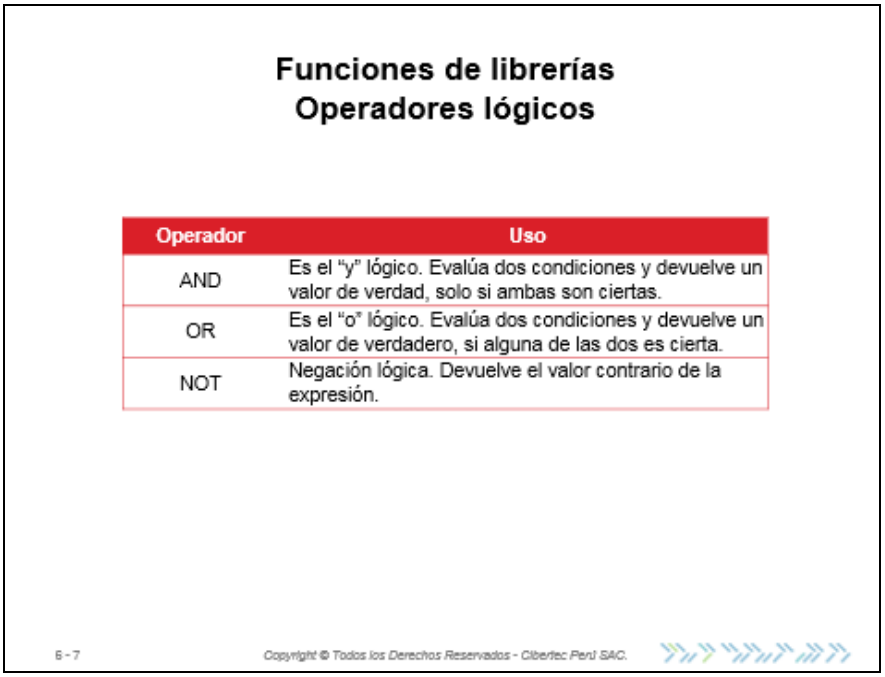
El siguiente ejemplo devuelve los nombres que comienzan con la letra **M** seguido de cualquier letra, entre **A** y **F**, y de tres caracteres cualesquiera.

```
SELECT  *
  FROM  Employees
 WHERE  FirstName LIKE 'M[A-F]___%'
```

Este otro ejemplo devuelve los nombres cuyo contenido empiece con una letra de la **A** hasta la **D**, seguidas de cualquier cadena.

```
SELECT  *
  FROM  Employees
 WHERE  FirstName LIKE '[A-D]%'
```

3. Funciones de librería



3.1 Funciones de cadena

A continuación, se muestra un resumen de las principales funciones de manejo de cadenas.

Función	Descripción	Ejemplo
LOWER()	Convierte las cadenas a minúsculas.	SELECT LOWER(Address) FROM Customers Convierte el contenido de la columna dirección (Address) a minúsculas.
LTRIM( )	Quita los espacios iniciales de una cadena.	SELECT LTRIM(Address) FROM Customers Muestra el contenido de la columna de dirección (Address) después de que se quiten delante los espacios iniciales.
SUBSTRING( )	Extrae uno o más caracteres de una cadena.	SELECT SUBSTRING(Phone, 1, 5) FROM Customers WHERE Country = 'UK' Muestra los cinco primeros caracteres (el código del área) de un número de teléfono.
UPPER( )	Convierte las cadenas a mayúsculas.	SELECT * FROM Customers WHERE UPPER(ContactName) = 'ANA TRUJILLO' Convierte el contenido de la columna **FirstName** a mayúsculas antes de compararlo con un valor específico (evita problemas de coincidencia si se distinguen mayúsculas de minúsculas). Para ver más detalles acerca de la distinción entre mayúsculas y

		minúsculas en SQL Server, consulte Consideraciones sobre el Diseñador de consultas.
LEFT(cadena, longitud)	Retorna la cantidad (longitud) de caracteres de la cadena comenzando desde la izquierda, primer carácter.	SELECT LEFT('buenos dias',8); Retorna "buenos d"
RIGHT(cadena, longitud)	Retorna la cantidad (longitud) de caracteres de la cadena comenzando desde la derecha, último carácter	SELECT RIGHT('buenos dias',8); Retorna "nos dias"

3.2 Funciones matemáticas

Del mismo modo, se presenta a continuación, un resumen de las funciones numéricas en Transact SQL.

Función	Descripción	Ejemplo
ROUND( )	Devuelve un valor numérico, redondeado a la longitud o precisión especificadas.	SELECT ROUND((Quantity * UnitPrice) - (UnitPrice * Discount), 2) FROM OrderDetails Muestra un precio total basado en un descuento, después redondea los resultados con dos números decimales.
ABS( )	Función matemática que devuelve el valor absoluto positivo de una expresión numérica específica.	SELECT ABS(-1.0), ABS(0.0), ABS(1.0); Se muestra el resultado de usar la función **ABS** en tres números distintos.
POWER( )	Devuelve el valor de la expresión especificada elevado a la potencia especificada.	SELECT POWER(2, 3) AS Result1, POWER(2.5, 3) AS Result2; Se muestra cómo elevar un número a la potencia 3 (el cubo del número).
SQRT( )	Devuelve la raíz cuadrada de un número	SELECT SQRT(25) Se muestra como obtener la raíz cuadrada de 25.

3.3 Funciones de fecha

A continuación, un resumen de las funciones de fecha en Transact SQL.

Función	Descripción	Ejemplo
DATEDIFF( )	Calcula un intervalo entre dos fechas.	SELECT DATEDIFF(DAY, OrderDate, ShippedDate) FROM Orders Calcula la diferencia de días entre la fecha de la orden y la fecha de envío.
DATEPART( )	Devuelve la porción especificada de una fecha o columna de fecha, incluidos el día, el mes o el año.	SELECT DATEPART(YEAR, HireDate) FROM Employees Muestra sólo el año en el que se contrató

		a un empleado (no la fecha entera).
GETDATE( )	Devuelve la fecha actual en formato de fecha y hora. Esta función es útil como entrada para muchas otras funciones de fecha, como calcular un intervalo hacia adelante o hacia atrás desde hoy.	SELECT * FROM Orders WHERE OrderDate = GETDATE() Muestra los pedidos realizados hoy.
DATENAME ()	Devuelve una cadena de caracteres que representa el parámetro datepart de una determinada fecha.	SELECT DATENAME(month, getdate()), DATENAME(weekday, getdate()) Muestra el nombre del mes y del día de la semana.

3.4 Funciones del sistema

A continuación, se presenta un resumen de las funciones del sistema del Transact SQL.

Función	Descripción	Ejemplo
DATALENGTH( )	Devuelve el número de bytes utilizados para la expresión especificada.	SELECT length = DATALENGTH(ProductName), ProductName FROM Products ORDER BY ProductName; Se busca la longitud de la columna **ProductName** de la tabla **Products**.
USER, USER_NAME( )	Devuelve el nombre de usuario actual.	SELECT name FROM sysusers WHERE name = USER_NAME(1); Busca el nombre del usuario actual sin especificar un identificador.

3.5 Otras funciones

A continuación, un resumen de otras funciones de interés.

Función	Descripción	Ejemplo
CONVERT( ) CAST ( )	Convierte los datos de un tipo de datos a otro. Es útil para dar formato a los datos o para utilizar el contenido de una columna de datos como argumento en una función que requiere un tipo de datos diferente.	SELECT ProductName, UnitPrice FROM Products WHERE CONVERT(int, UnitPrice) LIKE '3%' SELECT ProductName, UnitPrice FROM Products WHERE CAST(UnitPrice AS int) LIKE '3%'; Recupera el nombre de aquellos productos que tienen un 3 como primer dígito del precio y se convierte UnitPrice en int.
TRY_CONVERT() TRY_CAST()	Devuelve una conversión de valor a tipo de datos especificado si la conversión se realiza correctamente; de lo contrario, devuelve NULL.	SELECT TRY_CONVERT(float, 'test') SELECT TRY_CAST('test' as float)

STR( )	Convierte datos numéricos en una cadena de caracteres para que la pueda manipular con operadores de texto.	SELECT STR(EmployeeID) + ' ' + STR(ReportsTo) FROM Employees WHERE ReportsTo IS NOT NULL Muestra las columnas EmployeeID y ReportsTo (ambas numéricas) en una sola cadena.