
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

2015

**Información para la realización de las Guías de Trabajo**

Las guías de trabajo están basadas en los motores de base de datos **SQL Server 2008** y **PostgreSQL 9.1.** Para cada ejercicio se indica cuál es el motor de base de datos a utilizar para la resolución del mismo.

En esta primer guía de trabajo las diferencias no son sustanciales. Los ejercicios se realizarán en clase sobre SQL Server. En tiempo extra-clase el alumno deberá realizar la Guía de trabajo sobre PostgreSQL.

La mayoría de los ejercicios de práctica se aplican al modelo físico de la base de datos `Pubs` provista por la instalación de SQL Server. `Pubs` modeliza un sistema de gestión de publicaciones.
En el sitio FTP y en el aula virtual se proveen los scripts para la creación de la misma base de datos sobre el motor de base de datos **PostgreSQL**.

Podemos ver el esquema de la base de datos `pubs` en la ayuda de SQL Server, que se denomina **Books Online**: Debemos ir a *Inicio/SQL Server/Books Online* y luego, en la vista de árbol, seleccionar *Building SQL Server Applications/Transact-SQL Reference/pubs Sample Database*.

# 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.

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.
Puede conservar la consulta anterior en el editor, pero deberá comentarla a fin de que no vuelva a ejecutarse. Comentamos una o varias consultas encerrándolas en un bloque `/*` y `*/`.
También podemos comentar una única línea con un doble guión (`--`).

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. Que observa respecto a los valores `NULL` de la columna ordenada?

**4.** Muchos DBMSs permiten 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.

**Constantes en la lista de salida del `SELECT`**

Podemos especificar constantes 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.

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

Qué sucede en cada caso?

| | (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 |

**Conversión de datos numéricos a caracter**

Podemos convertir datos numéricos a caracter 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 de conversión de tipos.

# 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`.

**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.

**El predicado** `[NOT] IN`

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

**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]` |
| [^] | Que no coincida con el conjunto o rango especificado. |

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

**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 pare ello debemos utilizar la cláusula IS (o su negación IS NOT)
Que sucede si explícitamente compara la columna contra el valor NULL?.

**Funciones de agregación.**
**La cláusula DISTINCT**

Recordemos que las funciones de agregación ANSI SQL son COUNT, COUNT(*), SUM, MAX, MIN y AVG.

La cláusula DISTINCT se puede utilizar asociada a las funciones de agregación COUNT, SUM, y AVG cuando necesitamos que la función se aplique **sólo** a valores diferentes. O sea, no deseamos que cuenten, sumen o promedien valores repetidos en diferentes filas:

```
SELECT COUNT (DISTINCT type)
   FROM titles
```

Recordemos también que las funciones COUNT, SUM, y AVG no consideran a los valores NULL de las columnas a las que se aplican.

Debemos tener en cuenta que si tenemos una función de agregación en la lista de salida de un SELECT, solo podrán existir otras funciones de agregación en tal lista de salida. (La excepción ocurre en combinación con la cláusula GROUP BY).

Las funciones de agregación se pueden aplicar también a expresiones

**13.** Obtenga la cantidad de publicaciones con precio en contraposición a la cantidad total de publicaciones.

**14.** Obtenga la cantidad de precios diferentes en la tabla de publicaciones.

**15.** Algunas funciones de agregación pueden aplicarse a fechas. Obtenga el apellido y nombre del empleado contratado más recientemente. No es necesario que obtenga la solución con consultas anidadas (las veremos en la Guía de Trabajo Nro. 3). Por ahora resuelva la consulta en dos pasos.

**16.** La columna `ytd-sales` de la tabla `titles` posee la cantidad vendida de cada título. Obtenga la cantidad de dinero recaudada en ventas.

# 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.

**17.** La información de publicaciones vendidas reside en la tabla `Sales`. Calcule la cantidad de publicaciones vendidas (columna `qty`) para el mes de Junio.

## 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.