



Universidad Nacional del Litoral  
*Facultad de Ingeniería y Ciencias Hídricas*  
Departamento de Informática

# Bases de Datos

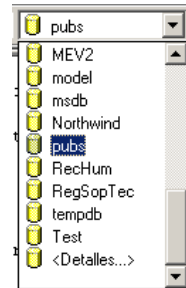
## *Guía de Trabajo Nro. 4*

### *T-SQL: Programación*

## Contexto de base de datos

---

- Cuando nos conectamos a un servidor de base de datos nos conectamos a un *contexto de base de datos*.
- En SQL Server, todo usuario se conecta identificándose con un *Login ID*. Este *Login ID* siempre es *database user* de al menos una base de datos y posee una base de datos definida por omisión. Esa base de datos será el *contexto de base de datos* de ese usuario.
- La lista de base de datos en el *Analizador de Consultas SQL* muestra gráficamente nuestro *contexto de base de datos actual*, y nos permite cambiarlo.



- Obtenemos el *contexto actual de base de datos* a través de la función T-SQL `db_name()`.
- Obtenemos información de las bases de datos disponibles en nuestra instalación a través del procedimiento almacenado del sistema `sp_helpdb`. Podemos obtener la misma información consultando la tabla `sysdatabases` de la base de datos del sistema `master`. La columna `name` posee el nombre de cada base de datos.
- Obtenemos información de un contexto de base de datos también a través del procedimiento almacenado del sistema `sp_helpdb`, pero pasándole como parámetro el nombre de la base de datos:

```
sp_helpdb pubs
```

- Si necesitamos cambiar el contexto de base de datos lo podemos hacer a través del comando T-SQL `USE`. La sintaxis es:

```
USE <base-de-datos>
```


- Cualquiera sea el contexto de base de datos actual siempre podemos acceder a **objetos** en otras bases de datos sin cambiar el contexto. Para eso debemos utilizar el *nombre completo del objeto*. El nombre completo de un objeto está compuesto por:
  - La *base de datos*  
Si no se especifica, SQL Server intenta localizar el objeto en el contexto actual.
  - El *owner*  
El usuario que la creó. Si no se especifica, SQL Server intenta localizar un objeto del usuario actual. Si no lo encuentra, intenta por el mismo nombre de objeto pero perteneciente al *dbo*.
  - El *nombre del objeto*

Por ejemplo, el nombre completo de la tabla `authors` en `Pubs` es `pubs.dbo.authors`.

1. Use T-SQL para obtener el contexto de base de datos actual. Averigüe que otros contextos disponibles existen en el sistema. Cambie el contexto a la base de datos `tempdb`. Desde ese contexto dispare un `SELECT` sobre la tabla `authors` en `pubs`. Retorne al contexto de la base de datos `pubs`.

## Batches y scripts

---

- Un batch es un conjunto de una o más sentencias SQL enviadas al servidor SQL a fin de que sean compiladas y ejecutadas como un grupo.
- Cuando trabajamos con el *Analizador de Consultas SQL*, ejecutamos un batch cada vez que hacemos click en . Dentro de un bloque de sentencias SQL se puede incluir la cláusula `GO` para indicar al programa cliente que envíe a procesar todas las sentencias anteriores al `GO` y continúe con el resto de las sentencias SQL luego de obtener los resultados del primer lote de sentencias.  
`GO` no es un comando T-SQL. Es una keyword utilizada por aplicaciones cliente para separar batches.
- Un *script SQL* es un conjunto de sentencias T-SQL compuesta de uno o más batches. Los scripts SQL se guardan en el lado cliente como un archivo `.sql`.
- Podemos agregar a un batch comentarios de simple línea con un doble guión `--`. Los comentarios de múltiple línea se demarcan con `/* y */`.

## Variables locales

---

- Las *variables locales* pueden utilizarse para conservar valores temporalmente dentro de un batch. Las utilizamos frecuentemente como:
  - Contadores en bucles
  - Medio de almacenamiento local de valores correspondientes a *variables del sistema*, antes de que el servidor modifique su valor.
  - Medio de recepción de *valores de columnas* para su tratamiento posterior (en un batch o procedimiento almacenado).
  - Medio de recepción de *valores de retorno* de procedimientos almacenados.
  - Medio de transferencia de parámetros hacia y desde procedimientos almacenados.

- Definimos una variable local a través de la sentencia `DECLARE`:

```
DECLARE @<nombre-variable> <tipo-dato>
[,@<nombre-variable> <tipo-dato>...]
```

- Toda variable declarada y sin inicializar posee un valor `NULL`. Le asignamos un valor a través de la sentencia `SET`:

```
SET @<nombre-variable> = valor
```

- Podemos consultar el valor de cualquier variables a través de la sentencia `SELECT`:

```
SELECT @<nombre-variable>
```

- Debemos tener en cuenta que las *variables locales* **existen mientras existe el batch que las define**. Recordemos que un batch es un grupo de sentencias SQL enviadas al servidor a fin de que sean ejecutadas como un grupo.

### 2. Ejecute el siguiente lote de sentencias T-SQL y analice los resultados:

```
DECLARE @Mens varchar(40)
SET @Mens = 'Just testing...'
SELECT @Mens
GO

SELECT @Mens
GO
```

## Variables y SQL

---

- Los valores de la lista de salida de una sentencia `SELECT` pueden ser asignados a variables. Por ejemplo:

```
DECLARE @price int

SELECT @price = price
FROM titles
WHERE title_id = 'BU1111'
```

Los tipos de datos deben ser compatibles, de otra forma los datos pueden perder precisión o resultar truncados.

Si una consulta que es asignada a una variable retorna más de una fila, la variable asume el valor correspondiente a la *última fila* del *conjunto resultado*. Si la consulta no retorna filas, el valor de la variable permanece inalterado.

- Podemos asignar el valor de una variable en el contexto de una sentencia de actualización `UPDATE`. La variable almacena el valor previo a la actualización. La sintaxis es la siguiente:

```
UPDATE tabla
SET columna = nuevo-valor,
@variable = columna
WHERE condicion
```

3. Ejecute el siguiente batch que actualiza la cantidad vendida en una fila de la tabla `ventas`. Consulte el valor de la variable `@Cant`.

```
DECLARE @Cant smallint
UPDATE ventas
SET cant = Cant + 100,
@cant = cant
WHERE codvent = 1
```

## Variables del sistema

---

- T-SQL posee una serie de variables del sistema, que tienen la característica de ser globales. Sus valores son establecidos automáticamente por el RDBMS y son de solo lectura. Las variables del sistema se diferencian de las locales en que su nombre es precedido por dos símbolos `@`. Revisaremos las más utilizadas.

- La variable del sistema @@error retorna el código de éxito o error de la última sentencia SQL ejecutada. Un valor distinto de cero indica una condición de error.
- @@rowcount retorna la cantidad de filas afectada por la última sentencia ejecutada. Las sentencias de control de flujo como IF o WHILE restablecen esta variable a 0.
- Si estamos utilizando columnas de tipo *autoincremental*, @@identity nos proporciona - luego de realizada una operación INSERT- el valor insertado automáticamente por el RDBMS para la misma.

## Mensajes al usuario

---

- Podemos retornar mensajes informativos al cliente a través de la sentencia PRINT. Su sintaxis es:

```
PRINT 'Mensaje'
```

4. Escriba un batch que retorne al usuario una cadena informativa de la forma: *El último código de error registrado fue n*, donde *n* sea el último *código de error* retornado por SQL Server.

## Sentencias de flujo de control

---

- En T-SQL podemos definir una estructura condicional a través de la sentencia IF. Su sintaxis es la siguiente:

```
IF <condición>
BEGIN
    ...
END
[ELSE]
BEGIN
    ...
END
```

Las keywords *Begin* y *End* definen delimitan *bloques de código*. Si no los especificamos, T-SQL ejecuta sólo la primer sentencia del grupo.

- Establecemos un bucle a través de la sentencia `WHILE`. Su sintaxis es:

```
WHILE <condicion>
BEGIN
    ...
END
```

La sentencia `WHILE` posee dos cláusulas adicionales: `CONTINUE` salta el control al principio del bucle y vuelve a evaluar la condición del `WHILE`. `BREAK` efectúa una salida incondicional del bucle.

- La sentencia `RETURN` permite abandonar el batch de manera inmediata.

5. Escriba un batch que informe al cliente con un mensaje si el precio de la publicación BU1111 es menor, igual o mayor a \$10.

6. Cree una tabla `t1` con dos columnas: `ID` de tipo `identity`, `FechaHora` de tipo `datetime` no nulo con un valor por omisión ajustado a la fecha y hora actual. Implemente un bucle que inserte 100 filas a la tabla almacenando en la columna `FechaHora` la fecha y hora actual. Verifique los datos insertados.

## Transacciones

---

- Una transacción es un grupo de sentencias que se ejecutan exitosamente o fracasan como una unidad. Tienen como objetivo preservar la integridad de los datos en caso de fallas.
- En T-SQL indicamos el comienzo de una transacción con la sentencia `BEGIN TRANSACTION`.
- A lo largo del batch el desarrollador evaluará posibles condiciones de error o la no satisfacción de determinadas reglas de negocio relativas al contexto del problema. Si estas condiciones de error ponen en peligro la consecución de la transacción como una unidad el desarrollador solicitará que se deshagan las modificaciones pendientes de confirmación. Esto se lleva a cabo a través de la sentencia `ROLLBACK TRANSACTION`.
- Si no se produjo ninguna condición de error fueron satisfechas las reglas de negocios, el desarrollador solicita que se las modificaciones pendientes se hagan permanentes. Esto se lleva a cabo a través de la sentencia `COMMIT TRANSACTION`.

## Implicancias de las transacciones

- Toda *sentencia SQL aislada* disparada contra el RDBMS representa en sí lo que se denomina una *transacción implícita*. Es decir, el inicio (**BEGIN**) y confirmación (**COMMIT**) de la transacción están implícitamente definidos en toda sentencia SQL.
- La solicitud de deshacer (**ROLLBACK**) o confirmar (**COMMIT**) una transacción no interfiere en absoluto con el *flujo de control del batch*. Por ejemplo, después de una sentencia **ROLLBACK** el procesamiento continúa normalmente con la próxima sentencia del batch.
- Las transacciones persisten **abiertas** hasta que son confirmadas o deshechas. Debemos tener en cuenta que, cuando una transacción está pendiente de confirmación, las *páginas de datos* correspondientes a las filas afectadas por la transacción resultan bloqueadas, impidiendo a otras conexiones acceder a esos datos. Es por ello que las transacciones deben agrupar solo las sentencias relacionadas lógicamente y –como regla general– **deben involucrar la menor cantidad posible de operaciones**.
- Si ocurre un error grave dentro de una transacción (caída del servidor, por ejemplo), la próxima vez que arranca el RDBMS repasa el *registro de transacciones* y deshace todas las transacciones no confirmadas (**COMMIT**).

7. En base a las tablas `Productos` y `Detalle` creadas en la *Guía de Trabajo Nro. 2*, registre el pedido de cinco unidades del artículo con código 100. Para ello debe disminuir el stock en la tabla `Productos` e insertar los datos del pedido en la tabla `Detalle` (Código de detalle 1200, Número de pedido 1108). Transaccione las operaciones de manera tal que tengan éxito o fracasen como una unidad.



## SQL dinámico

---

- T-SQL permite definir sentencias SQL como una cadena de caracteres construida a partir de elementos variables, tales como variables locales o valores de retorno de procedimientos almacenados.
- Ejecutamos la cadena de caracteres como una sentencia SQL a través del comando `EXEC`. La siguiente es la sintaxis de `EXEC`:

```
EXEC (@Nombre-de-variable)
```

- Supongamos que necesitamos disparar un `SELECT` `name` sobre una *tabla del sistema*, cuyo valor se define en tiempo de ejecución. Por ejemplo, la tabla puede ser `sysobjects`, `syscolumns` o `sysusers`.

```
DECLARE @Cad1 Varchar(100)
DECLARE @Cad2 Varchar(100)
SET @Cad1 = 'SELECT name FROM'
SET @Cad2 = ' syscolumns'
EXEC (@Cad1 + @Cad2)
```