Vistas

¿Qué es una vista?

- Una vista es una tabla virtual que se basa en el resultado de una consulta.
- Sus atributos son atributos de tablas o de otras vistas.
- Pueden usarse en consultas como si fueran tablas.

Crear Vistas

- Para crear una vista debemos utilizar la sentencia **CREATE VIEW**, debiendo proporcionar un nombre a la vista y una sentencia SQL SELECT válida.

```
CREATE VIEW <nombre_vista>
AS
(<sentencia_select>);
```

Crear Vistas

- Queremos ofrecer una vista sobre la tabla de inscripciones en la que no aparezca el costo.

```
CREATE VIEW inscripciones_vista AS (
        SELECT CodEst, cursoCod, fechaInsc
        FROM INSCRIPCIONES
);
```

Actualizar / Borrar Vistas

```
ALTER VIEW inscripciones_vista AS (
        SELECT CodEst, cursoCod
        FROM INSCRIPCIONES
);
```

```
DROP VIEW <nombre_vista>
```

Algunas consideraciones

CREATE VIEW name [(alias1, alias2, . . . , aliasN)] AS <consulta>

Los alias son opcionales, permiten asignar nombres a las columnas de la vista.

Si se omiten, las columnas tendrán el mismo nombre que en las tablas originales (de donde se obtuvieron).

Las vistas son consultas a los datos que hay en las tablas, por lo que:

- si actualizamos los datos de una vista (UPDATE), estamos actualizando realmente la tabla.
- si actualizamos la tabla (UPDATE) estos cambios serán visibles desde la vista.

Razones por las cuales utilizar vistas

Seguridad: nos puede interesar que los usuarios tengan acceso a una parte de la información que hay en una tabla, pero no a toda la tabla.

Comodidad: el modelo relacional no es el más cómodo para visualizar los datos, lo que nos puede llevar a tener que escribir complejas sentencias SQL.
Tener una vista nos simplifica esta tarea.

Ejercicio 1

Crear una vista que permita mostrar para cada cliente su nombre, la cantidad de órdenes procesadas y el importe total de las mismas.

Ejercicio 2

Mediante una vista mostrar los nombres de los productos y la cantidad de órdenes en las que cada uno aparece.

Ejercicio 3

Definir una vista que permita visualizar el nombre de las ciudades y el nombre del país para todas las órdenes que tienen descuentos en alguna de sus líneas.

Ejercicio 4

Para un informe que se entrega a los vendedores en forma habitual, crear una vista que obtenga los datos de los productos cuyo mayor descuento aplicado supera el 0.15

Transacciones

Transacción

- Una transacción es una secuencia de operaciones que se ejecutan como una unidad indivisible. Las transacciones aseguran que todas las operaciones se realicen correctamente o que se deshagan completamente en caso de fallos.

- Partir en estado consistente y terminar en estado consistente

Propiedades de las transacciones

Siguen **cuatro propiedades básicas - ACID** (Atomicity, Consistency, Isolation, Durability):

- **Atomicidad**: aseguran que todas las operaciones dentro de la secuencia de trabajo **se completen satisfactoriamente**. Si no es así, la transacción se abandona en el punto del error y las operaciones previas retroceden a su estado inicial.
- **Consistencia**: aseguran que la base de datos **cambie estados** en una transacción exitosa.
- **Aislamiento**: permiten que las operaciones sean **aisladas** y **transparentes** unas de otras.
- **Durabilidad**: aseguran que el resultado o efecto de una transacción completada **permanezca en caso de error del sistema**.

Control de las transacciones

Existen tres comandos básicos de control en las transacciones SQL:

- **COMMIT**. Para **guardar los cambios**.
- **ROLLBACK**. Para **abandonar la transacción y deshacer los cambios** que se hubieran hecho en la transacción.
- **SAVEPOINT**. Crea **checkpoints**, puntos concretos en la transacción donde poder deshacer la transacción hasta esos puntos.

Los comandos de control de transacciones se pueden usar en scripting (**INSERT**, **DELETE** y **UPDATE)**. **No pueden utilizar en ddl** porque las operaciones se guardan automáticamente en la base de datos.

```
USE NorthWind
DECLARE @Error int
--Declaramos una variable que utilizaremos para almacenar un posible código de error

BEGIN TRAN --Iniciamos la transacción
	UPDATE Products SET UnitPrice=20 WHERE ProductName ='Chai' --Ejecutamos la primera sentencia
	SET @Error=@@ERROR --Si ocurre un error almacenamos su código en @Error
	--y saltamos al trozo de código que deshará la transacción. Si, eso de ahí es un
	--GOTO, el demonio de los programadores, pero no pasa nada por usarlo cuando es necesario
	IF (@Error<>0) GOTO TratarError --Si la primera sentencia se ejecuta con éxito, pasamos a la segunda
		UPDATE Products SET UnitPrice=20 WHERE ProductName='Chang'
		SET @Error=@@ERROR
		IF (@Error<>0) GOTO TratarError

		--Si llegamos hasta aquí es que los dos UPDATE se han completado con
		--éxito y podemos "guardar" la transacción en la base de datos
		COMMIT TRAN

TratarError:
--Si ha ocurrido algún error llegamos hasta aquí
If @@Error<>0 THEN
BEGIN
	PRINT 'Ha ocorrido un error. Abortamos la transacción'
	--Se lo comunicamos al usuario y deshacemos la transacción
	--todo volverá a estar como si nada hubiera ocurrido
	ROLLBACK TRAN
END
```