ÍNDICE

1. INTRODUCCIÓN

- Los datos son un recurso valioso para las organizaciones, por lo que se han de establecer políticas de seguridad para **garantizar su confidencialidad, su integridad y su disponibilidad**.
    - **Confidencialidad:** No desvelar datos a usuarios no autorizados. Comprende también la privacidad (protección de datos personales).
    - **Disponibilidad:** La información debe estar accesible.
    - **Integridad:** Permite asegurar que los datos no han sido falseados.
- El término seguridad en la BD engloba a cualquier mecanismo que proteja a la base de datos frente a amenazas intencionadas o accidentales.
- La seguridad no se aplica únicamente a los datos almacenados en las propias bases de datos, sino también a otras partes del sistema que pueden afectar directamente a la propia base de datos y al transporte de los datos. Por esa razón la seguridad en la base de datos es una técnica que abarca tanto el hardware, el software, las personas y los datos.
- Por ello hay que establecer **medidas de seguridad a varios niveles**:
    - **Físico:** Los equipos informáticos deben protegerse contra los fallos físicos (cortes de red, discos redundantes,..).
    - **Humano:** Todos los usuarios deben estar bien identificados y autorizados.
    - Sistema operativo: Un sistema operativo débil podría permitir un acceso no autorizado.
    - **Red:** Dado que muchas bases de datos permiten accesos remotos la seguridad a nivel de red es muy importante.
    - **Sistema de gestión de bases de datos:** Dado que sus usuarios pueden tener diferentes privilegios de acceso, el SGBD debe asegurarse de que éstos se cumplen.

El **control de acceso** representa una operación importante en la gestión de la seguridad sobre un servidor de BBDD.

La seguridad de los datos requiere una organización de los objetos de manera independiente de los usuarios, y esto es posible gracias a los **esquemas**, como veremos más adelante.

También es necesario el control de los **privilegios** que necesita cada usuario para poder trabajar.

1.1. PROBLEMAS DE SEGURIDAD EN BASES DE DATOS

- **El robo y el fraude:** Pérdida y alteración de datos.
- **Pérdida de confidencialidad y privacidad:** Confidencialidad hace referencia a la necesidad de mantener en secreto ciertos datos críticos para la organización, mientras que privacidad hace referencia a la necesidad de proteger datos acerca de las personas (LOPD).
- **Pérdida de integridad:** Se refiere a la aparición de datos inválidos o corrompidos.

1.2. MECANISMOS DE SEGURIDAD EN BBDD

- Los gestores de BD ofrecen:
  - **Protección de acceso al gestor.**
  - **Control de acceso a los objetos de la BD (gestión de usuarios, permisos, vistas…).**
  - **Recuperación ante fallos (checkpoints, backup).**
  - **Cifrado de datos.**

2. VISTAS

Una **vista (VIEW)** es ***una tabla virtual*** cuyo contenido está definido por una consulta. Al igual que una tabla real, una vista consta de un conjunto de columnas y filas de datos con un nombre. Sin embargo, la vista no existe como conjunto de valores almacenados en una bd, **los datos a los que hace referencia la vista no ocupan espacio en disco**.

La consulta que define la vista puede provenir de 1 ó más tablas, o bien de otras vistas, de la misma bd o de otras.

IMPORTANTE: Se trata de una consulta SQL que está permanentemente almacenada en la bd y a la que se le asigna un nombre, de modo que los resultados de la consulta almacenada son visibles a través de la vista, y SQL permite acceder a estos resultados como si fueran de hecho una tabla real en la bd.

2.1. VENTAJAS DEL USO DE LAS VISTAS

Entre las principales ventajas del uso de las vistas podemos mencionar:

- **Simplificación de las estructura de las tablas:** Algunas tablas tienen muchas columnas con nombres y tipos poco prácticos a la hora de manipularlos. La vista nos permite darle al usuario los datos de manera simplificada.
- **Reutilización de consultas:** Cuando existen consultas complicadas, con numerosos joins y cálculos, los usuarios invocan a las vistas en lugar de tener que entender y ejecutar complicadas consultas.
- **Seguridad de acceso:** Permite mostrar un subconjunto de filas (**vista horizontal**) y/o columnas (**vista vertical**) de una tabla. Podemos ocultar así filas y columnas dejando a disposición de los usuarios sólo determinada información.
- **Permite mostrar información de más de una tabla como si fuera una.**
- **Permite realizar uniones entre dos o más tablas y que para el usuario parezca una única.**

2.2. CREACIÓN DE VISTAS CON T-SQL

Sintaxis:

```
CREATE VIEW  [nombre_esquema.]nombre_vista_a_crear
                              [ ( lista_de_nombres_de_campos ) ]
[WITH  ENCRYPTION  |  WITH  SCHEMABINDING  |  WITH  VIEW_METADATA]
AS
   SENTENCIA_SELECT
  [WITH CHECK OPTION] ;
```

Si no indicamos los nuevos nombres de campos para nuestra vista, tomarán los mismos nombres que tenían en las tablas de origen.(**OJO** si en las tablas de origen se repiten los nombres de los campos, ya que estaríamos repitiendo nombres de campos de la vista).

Cláusula WITH ENCRYPTION: Permite cifrar el texto de la vista.

Cláusula WITH SCHEMABINDING: Permite vincular la vista al esquema.
Para hacerlo en la consulta de la vista debemos hacer referencia a cada objeto con su esquema delante (esquema.objeto).
Al incluir esta cláusula, los objetos usados en esta vista no podrán ser eliminados, y, si una operación ALTER afecta a la definición de la vista, dicha operación fallará.

Cláusula WITH VIEW_METADATA: Con esto indicamos al servidor que devuelva la Información de metadatos de la vista, y no de los objetos a los que hace referencia la vista. Esto es importante cuando la información se solicita desde una aplicación a través de por ejemplo ODBC.

SENTENCIA_SELECT: Es la instrucción que define la vista. En el FROM de esta consulta podemos incluir tablas, otras vistas, funciones y varias instrucciones SELECT separadas por UNION o UNION ALL.
La sentencia SELECT de una vista no puede contener:
- COMPUTE o COMPUTE BY
- la palabra clave INTO
- referencia a una tabla temporal o a una variable de tabla
- **ORDER BY,** a menos que se especifique la cláusula TOP n [percent].

Cláusula WITH CHECK OPTION:
Esta cláusula, si se añade en la creación de la vista, **exige que** todas las instrucciones de modificación de datos ejecutadas contra la vista **verifiquen las restricciones establecidas en la CONSULTA_SELECT**. Es decir, si la consulta sobre la que se define la vista tiene una restricción en la cláusula WHERE, los datos que se intenten actualizar en la vista deberán verificar esa restricción. Si no la verificasen la actualización se rechazaría.

OBSERVACIONES:
- Una vista solo se puede crear en la base de datos actual.
- Una vista puede tener un máximo de 1.024 columnas.
- Si una vista no se crea con la cláusula SCHEMABINDING, debe ejecutarse **sp_refreshview** cuando se realicen cambios en los objetos subyacentes de la vista que afecten a la definición de ésta. De lo contrario, la vista podría producir resultados inesperados en las consultas.
- Para poder ver el contenido de la vista haremos la consulta SELECT que nos interese.

Ejemplo: Crea una vista v_Emp_Primera_Mitad con la primera mitad de los nombres completos de los empleados obtenidos de una lista alfabética de la bd EMPLEADOS.

```
USE EMPLEADOS;
GO
CREATE VIEW v_Emp_Primera_Mitad AS
  SELECT TOP 50 percent ape1+' '+ isnull(ape2,'')+', '+ nombre as nombre_completo
  FROM repventas
    ORDER BY ape1, ape2, nombre;
```

2.3. ACTUALIZACIÓN DEL CONTENIDO DE UNA VISTA

Para que una vista sea actualizable, es decir que una tabla subyacente se pueda actualizar con INSERT, DELETE o UPDATE, debemos tener en cuenta las siguientes reglas a la hora de crearla**:**

- Cualquier modificación debe hacer referencia a las columnas de **una única tabla base**.
- Las columnas que se vayan a modificar en la vista no pueden ser ni expresiones ni columnas calculadas, deben ser **columnas simples.** No podrán contener:
  - Una función de agregado: AVG, COUNT, SUM, …
  - Un cálculo.
- Las columnas que se van a modificar no pueden verse afectadas por las cláusulas **GROUP BY**, **HAVING** o **DISTINCT**.

2.4. MODIFICACIÓN DE VISTAS

Sintaxis:

```
ALTER VIEW  [nombre_esquema.]nombre_vista_a_modificar
                        [ ( lista_de_nombres_de_campos ) ]
[WITH  ENCRYPTION  |  WITH SCHEMABINDING  |  WITH VIEW_METADATA]
AS
    SENTENCIA_SELECT
[WITH CHECK OPTION];
```

IMPORTANTE: Aunque sólo deseemos, por ejemplo, modificar un campo de la vista, siempre se deberá repetir la select completa. Hay que recordar que en realidad no estamos modificando la estructura de la tabla/s de origen, sino simplemente la vista a través de la que accedemos a ella/s.

Ejemplo: Si nos piden que cifremos el código de la vista v_Emp_Primera_Mitad creada anteriormente debemos hacerlo con la siguiente instrucción T-SQL:

```
USE EMPLEADOS;
GO
ALTER VIEW v_Emp_Primera_Mitad
WITH   ENCRYPTION
AS
  SELECT TOP 50 percent ape1+' '+ isnull(ape2,'')+', '+ nombre as nombre_completo
  FROM repventas
    ORDER BY ape1, ape2, nombre;
```

2.5. ELIMINACIÓN DE VISTAS

Sintaxis:

```
DROP VIEW  [nombre_esquema.]nombre_vista_a_borrar [ ..., n ];
```

OBSERVACIÓN: En una misma instrucción podemos borrar varias vistas.

Ejemplo: Para borrar la vista creada en ejemplos anteriores v_Emp_Primera_Mitad escribiremos la siguiente sentencia:

```
USE EMPLEADOS;
GO
DROP VIEW v_Emp_Primera_Mitad ;
```

3. GESTIÓN DE USUARIOS EN SQL Server

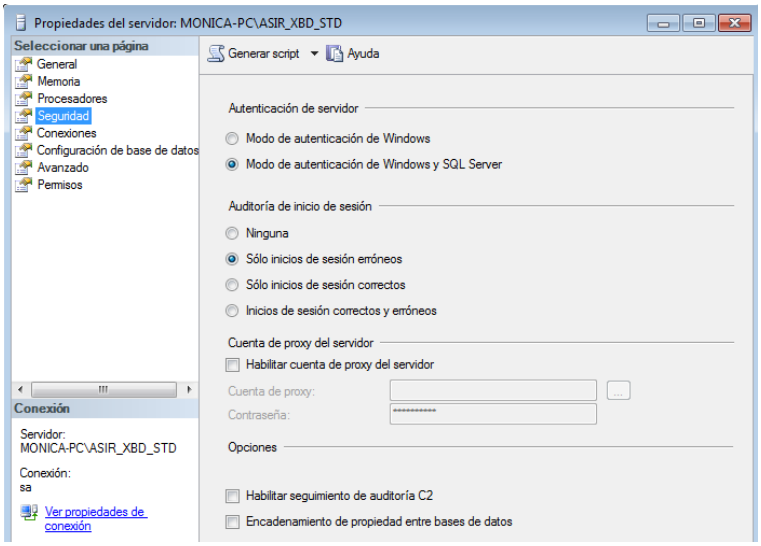
3.1. MECANISMOS DE AUTENTICACIÓN

Cada instancia de MS SQL Server® se ejecuta, entre otros, en uno de los **dos modos de autenticación**:

1) **Modo de autenticación de Windows** (*conocido como seguridad integrada en versiones anteriores a la 2000*). En este modo, sólo permite conexiones que utilicen Autenticación de MS Windows®. Cualquier usuario del sistema que tenga los privilegios para acceder al gestor de BBDD, podrá hacerlo.
2) **Modo mixto**. En este modo, las conexiones pueden realizarse con Autenticación de Windows o con Autenticación de SQL Server.

Para comprobar o para modificar el modo de autenticación debemos situarnos en el Management Studio sobre el icono de la instancia de nuestro servidor. Desplegamos el menú contextual con el botón derecho del ratón y seleccionamos la opción **Propiedades.**

Una vez abierta la ventana de propiedades del servidor debemos seleccionar la opción **Seguridad**. En esta página encontramos información sobre el modo de autenticación de nuestra instancia.



En el momento de la instalación del servidor, se predefinen dos conexiones:

- En modo Windows se autoriza a conectarse al grupo local de los Administradores de Windows,
- en modo de seguridad de SQL Server será el usuario sa (system administrator) el que puede conectarse al servidor.

Tanto los administradores de Windows como el usuario sa tienen privilegios de administrador del servidor SQL.

Se pueden crear cuentas de inicio de sesión a partir de usuarios y grupos ya existentes en MS Windows®, o se pueden crear cuentas de inicio de sesión en MS SQLServer®.

Se recomienda usar la seguridad Windows siempre que sea posible. Por ejemplo, debemos utilizar el modo de autenticación mixto con clientes que no usan Windows NT/2000/XP/7/8, clientes de Internet, con acceso a la BBDD desde aplicaciones.

3.2. INICIOS DE SESIÓN

Los inicios de sesión o cuentas de conexión permiten gestionar las conexiones al servidor SQL Server. Estos inicios de sesión, sean de tipo SQL Server o Windows, deben definirse en la instancia del servidor a la que queremos que se conecten.

IMPORTANTE: Los inicios de sesión son necesarios para acceder a la base de datos pero además hay que asociarles una base de datos predeterminada que será la bd de trabajo por defecto, es decir en la que se posiciona el servidor después del inicio de la conexión. Además para acceder a la bd por defecto es necesario definir una cuenta de usuario de base de datos para ese inicio de sesión.

Si queremos consultar la información de los inicios de sesión podemos consultar las vistas del sistema:

- **sys.server_principals:** Contiene una fila por cada entidad de seguridad del servidor.
- **sys.sql_logins:** Devuelve una fila por cada inicio de sesión de SQL.

3.2.1. Tipos de inicios de sesión

En SQL Server 2008 R2 existen los siguientes tipos de inicio de sesión:

1) Windows:
   a. Usuario
   b. Grupo de usuarios
2) SQL Server
3) Certificado
4) Clave asimétrica
5) Asociados a credenciales: Para acceso a recursos externos.

IMPORTANTE: Nosotros nos centraremos en los inicios de sesión de Windows y de SQL Server.

3.2.2. Inicios de sesión predefinidos

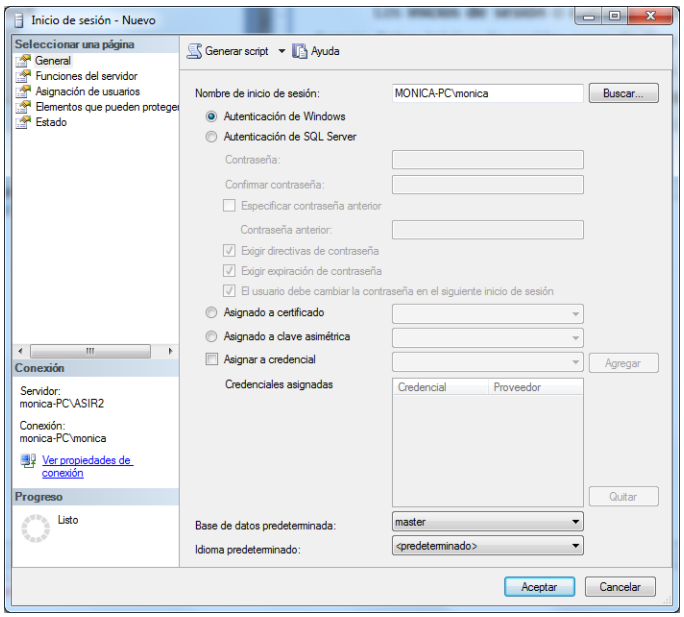
Al instalarse SQL Server se crean 2 inicios de sesión:

1) La cuenta de servicio que se utiliza para iniciar el servicio SQLServer. Puede cambiarse sus privilegios.
2) El usuario sa. Este no puede eliminarse ni modificarse. No estará disponible si solo está configurada la autentificación de Windows.

Ambos pueden realizar cualquier tarea en SQL Server por pertenecer al rol de servidor sysadmin.

3.2.3. Creación de un inicio de sesión en modo de seguridad Windows

A) Gráficamente: Desde el explorador de objetos, a nivel de la instancia del servidor, accedemos a la carpeta **Seguridad->Inicios de sesión** y en el menú contextual seleccionamos **Nuevo inicio de sesión…** Se muestra la pantalla siguiente:



B) Con Transact-SQL:

Sintaxis:

```
CREATE LOGIN nombre_inicio_de_sesión
FROM WINDOWS
[WITH DEFAULT DATABASE=nombre_de_la_bd | DEFAULT_LANGUAGE=idioma];
```

nombre_inicio_de_sesión: Nombre del usuario o grupo de Windows a añadir. Debe indicarse con la forma **dominio\usuario**.
nombre_de_la_bd: BD que será usada por defecto, si no se indica nada la bd por defecto es master.
idioma: es el idioma de trabajo por defecto para esta conexión. Sólo es necesario indicarlo si el idioma es diferente al definido en el servidor.

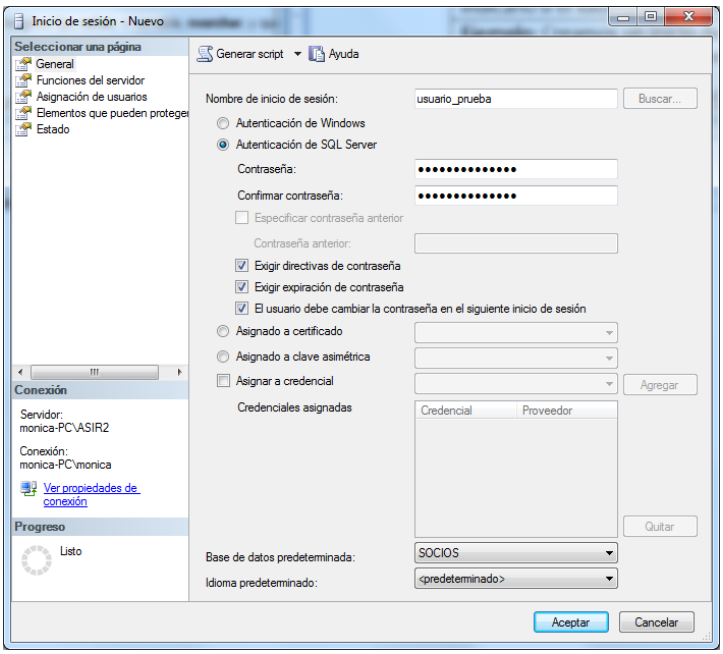
Ejemplo: Creamos un inicio de sesión para el usuario de Windows usu_pruebas.

```
CREATE LOGIN "MONICA-PC\usu_pruebas"
FROM WINDOWS
WITH DEFAULT_DATABASE=SOCIOS;
```

***Observa** como ponemos entre comillas dobles el nombre del inicio de sesión porque el nombre del dominio (el equipo local en este caso) contiene un carácter especial (-). También podríamos usar corchetes [] en lugar de las comillas dobles.*

3.2.4. Creación de un inicio de sesión en modo de seguridad SQL Server

A) Gráficamente: Desde el explorador de objetos, a nivel de la instancia del servidor, accedemos a la carpeta **Seguridad->Inicios de sesión** y en el menú contextual seleccionamos **Nuevo inicio de sesión…** Se muestra la pantalla siguiente:



B) Con Transact-SQL:

Sintaxis:

```
CREATE LOGIN nombre_inicio_de_sesión
WITH PASSWORD='contraseña'  [MUST_CHANGE]
[
, DEFAULT_DATABASE= nombre_de_la_bd  |
, DEFAULT_LANGUAGE=idioma  |
, CHECK_EXPIRATION= { ON | OFF}
, CHECK_POLICY= { ON | OFF}
, [ CREDENTIAL= nombre_credencial ]
];
```

nombre_inicio_de_sesión: Nombre del nuevo inicio de sesión.
Contraseña: contraseña asociada al inicio de sesión. Es obligatoria para cada login.
MUST_CHANGE: La primera vez que se inicie sesión en el servidor se solicitará al usuario introducir una nueva contraseña. Sólo es posible si CHECK_EXPIRATION y CHECK_POLICY están a ON.
nombre_de_la_bd: BD que será usada por defecto, si no se indica nada la bd por defecto es master.
idioma: es el idioma de trabajo por defecto para esta conexión. Sólo es necesario indicarlo si el idioma es diferente al definido en el servidor.
CHECK_EXPIRATION: Especifica si debe aplicarse la directiva de expiración de contraseñas en este inicio de sesión. El valor predeterminado es OFF.
CHECK_POLICY: Especifica que se deben aplicar las directivas de contraseñas de Windows en el equipo que ejecuta SQL Server para este inicio de sesión. El valor predeterminado es ON.
CREDENTIAL: Se asigna al inicio de sesión una credencial creada previamente. Permite a las conexiones SQL Server acceder a los recursos externos al servidor. Una credencial contiene un nombre de cuenta de Windows y una contraseña.

Ejemplo: Creamos un inicio de sesión de SQL Server de nombre usu_pruebas_sql.

```
CREATE LOGIN usu_pruebas_sql
WITH PASSWORD='usu_pruebas_sql' MUST_CHANGE,
DEFAULT_DATABASE=SOCIOS,
CHECK_EXPIRATION=ON;
```

3.2.5. Borrado de un inicio de sesión

Sintaxis:

```
DROP LOGIN nombre_inicio_de_sesión;
```

IMPORTANTE: Sí se pueden eliminar inicios de sesión que tengan usuarios de base de datos asignados, pero esto creará usuarios huérfanos, sin login.
Para asignar de nuevo un login a un usuario huérfano usaremos el procedimiento almacenado del sistema sp_change_users_login

3.2.6. Modificación de las características de un inicio de sesión

Sintaxis:

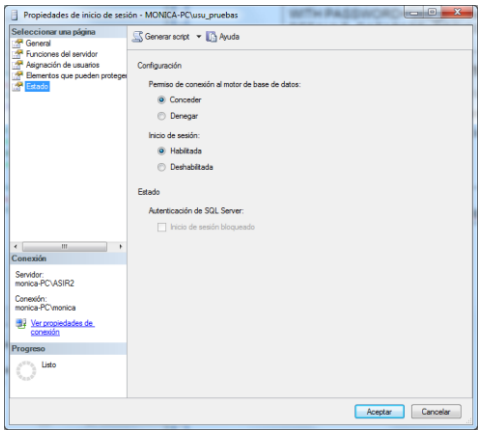
```
ALTER LOGIN nombre_inicio_de_sesión
CARACTERÍSTICAS A MODIFICAR
```

Ejemplo: Cambiamos el nombre de inicio de sesión Maria por MariJose.

```
ALTER LOGIN Maria WITH NAME = MariJose;
```

3.2.7. Activar/desactivar un inicio de sesión

A) Gráficamente: Desde el explorador de objetos, a nivel de la instancia del servidor, accedemos a la carpeta **Seguridad->Inicios de sesión** y en el menú contextual del inicio de sesión que nos interese seleccionamos la opción **Propiedades** y aparece la siguiente ventana:



B) Con Transact-SQL:

Sintaxis:

```
ALTER LOGIN nombre_inicio_de_sesión {ENABLE | DISABLE } ;
```

3.3. USUARIOS DE BASE DE DATOS

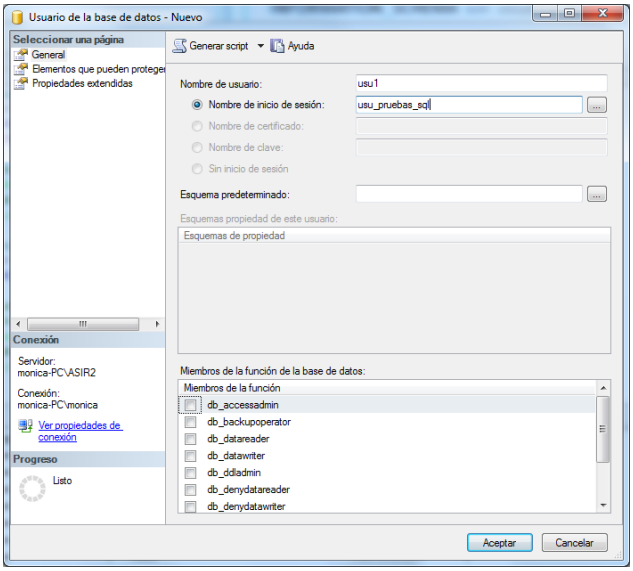
- Después de definir los inicios de sesión a nivel de instancia del servidor, es necesario definir los usuarios en las bases de datos. Es un paso obligatorio, excepto si el inicio de sesión tiene permisos de administración.
- Los usuarios de base de datos se asocian a una conexión del servidor.
- Los usuarios especiales **guest**, **sys** e **INFORMATION_SCHEMA** no se asignan a ninguna conexión. Las cuentas sys e INFORMATION_SCHEMA son usuarios de las vistas del sistema, imprescindibles para SQL Server y no se pueden quitar.
- Además podemos crear un usuario sin asociar a un login (WITHOUT LOGIN).
- Si queremos **consultar la información de las cuentas de usuario** de base de datos podemos consultar la vista del sistema **sys.database_principals**
- El procedimiento almacenado del sistema **sp_who** nos permite ver **qué usuarios están actualmente conectados y los procesos en curso.**

3.3.1. Usuarios por defecto en una base de datos

1) **dbo:** Es el usuario propietario. No puede ser borrado de la BD.
2) **Guest:** Permite a usuarios que no tienen cuenta en la BD, acceder a ella.
3) **INFORMATION_SCHEMA:** Permite ver los metadatos de SQL Server
4) **sys:** Permite consultar las tablas y vistas del sistema, procedimientos extendidos (xp_) y otros objetos del catálogo del sistema.

3.3.2. Creación de un usuario de base de datos

A) Gráficamente: Desde el explorador de objetos, en la carpeta (nodo) de la bd en la que queremos crear el usuario nos situamos en la carpeta **Seguridad->Usuarios** y en el menú contextual y escogemos la opción **Nuevo usuario…** Aparece la siguiente ventana:



B) Con Transact-SQL:

Sintaxis:

```
CREATE USER nombre_usuario
[ FOR   LOGIN nombre_conexión
  | WITHOUT LOGIN  ]
[ WITH DEFAULT_SCHEMA = nombre_esquema];
```

nombre_usuario: Nombre del nuevo usuario de la base de datos.
nombre_conexión: Nombre del inicio de sesión al que se asocia el usuario.
nombre_esquema: Nombre del esquema asociado al usuario nuevo. Es posible asociar varios usuarios al mismo esquema.

Ejemplo: Creación del usuario usu1.

```
CREATE USER usu1
FOR LOGIN usu_pruebas_sql
WITH DEFAULT_SCHEMA=usu1;
```

NOTAS:

- Si especificamos WITHOUT LOGIN (*en lugar de FOR LOGIN)*: Especifica que el usuario no se debe asignar a un inicio de sesión existente. Este usuario puede conectarse a otras bases de datos como guest.
- Si se omite FOR LOGIN: El usuario se asociará al inicio de sesión de SQL Server con su mismo nombre.
- Si se deja sin definir DEFAULT_SCHEMA: El esquema predeterminado del usuario será dbo.

3.3.3. Borrado de un usuario de BD

Sintaxis:

```
DROP USER nombre_usuario;
```

IMPORTANTE: No se puede eliminar un usuario si es propietario de objetos.

3.3.4. Modificación de las características de un usuario de BD

Sintaxis:

```
ALTER USER nombre_usuario
CARACTERÍSTICAS A MODIFICAR
```

Ejemplo: Cambiamos el nombre de usuario de Pepe por JoseMari.

```
ALTER USER Pepe WITH NAME = JoseMari;
```

3.3.5. Creación de un usuario INVITADO de base de datos

No podemos crear un usuario de tipo invitado en la base de datos porque ya existe uno creado en cada bd de nombre GUEST. Por defecto está deshabilitado y para habilitarlo se haría como indicamos a continuación:

Sintaxis:
```
GRANT CONNECT TO guest;
```

NOTAS:
- Se concede usando GRANT el permiso CONNECT al usuario guest.

3.4. ESQUEMAS

Para entender el concepto de esquema (*schema*) vamos a compararlo con el de propietario (*object owner*).

Todo objeto de base de datos tiene un nombre, que permite identificarlo de forma única en la base de datos en la que existe. Si tenemos una tabla llamada ALUMNADO como los objetos en la mayoría de los casos se crean con propietario o esquema dbo, sería posible acceder a ésta tabla denominándola ALUMNADO o dbo.ALUMNADO. ¿Por qué ocurre esto?

- Hasta SQL Server 2000 el nombre completo de un objeto toma la forma de instancia_del_servidor.base_de_datos.*propietario*.objeto, mientras que,
- a partir de SQL Server 2005 el nombre completo de un objeto pasa a ser instancia_del_servidor.base_de_datos.*esquema*.objeto

Se cambia al Propietario por el Esquema ya que hasta la versión 2005 no existía el concepto de Esquema, de tal modo que un usuario al crear un objeto, si no lo calificaba de forma explícita (no ponía al crearlo en nombre del propietario delante), se creaba con el nombre del usuario creador como propietario. Por ejemplo, si el usuario usu1 crea la tabla con nombre ALUMNADO sin calificar (CREATE TABLE ALUMNADO …) la tabla se denominará usu1.ALUMNADO

A partir de SQL Server 2005 aparecen los esquemas que son objetos de base de datos que nos proporcionan un espacio de nombres (*namespace*).

IMPORTANTE: Al usar esquemas podemos eliminar un usuario sin que impacte en las aplicaciones y consultas existentes, ya que eliminar un usuario no implica la eliminación del esquema.

- Podemos decir que un esquema es un contenedor de objetos de base de datos.

3.4.1. Uso de los esquemas

- Todo objeto de base de datos (por ejemplo: tabla, vista, procedimiento almacenado,…) tiene asociado uno y sólo un esquema.
- Todo usuario tiene asignado un esquema por defecto (*default schema*), en su defecto, el esquema dbo. Múltiples usuarios pueden tener asignado el mismo esquema por defecto, si nos resulta de utilidad.
- Los esquemas también nos permitirán organizar nuestros objetos, de tal manera que podemos crear múltiples esquemas como si fueran carpetas y en cada carpeta tener los objetos que nos interesen.

- También facilitan la configuración de seguridad al poder conceder permisos sobre todos los objetos contenidos en un esquema mediante una única instrucción.
- Los esquemas, como objetos de base de datos que son, tienen un propietario o creador. Es decir, un usuario con suficientes permisos puede crear un esquema, siendo él el propietario. Después podrá crear múltiples objetos y asignarlos a ese esquema. Es posible cambiar el propietario de un esquema utilizando ALTER AUTHORIZATION, como se muestra posteriormente.
- Es posible mover objetos de un esquema a otro.

IMPORTANTE: Debemos diferenciar entre esquema y usuario claramente, porque nos vamos a encontrar bases de datos que tienen esquemas con los mismos nombres que los usuarios, (*por ejemplo, como resultado de una migración de SQL Server 2000 a SQL Server 2022*) y si no estamos atentos, podemos confundirnos.

3.4.2. Crear esquemas

Sintaxis: Instrucción de creación de un esquema:

```
CREATE SCHEMA nombre_del_esquema AUTHORIZATION nombre_del_propietario
[    {
        definición_de_tabla | definición_de_vista | instrucción_grant |
        instrucción_revoke | instrucción_deny
    }
    [ ...n ]
]
```

IMPORTANTE: […n] Esto último indica que en la creación del esquema pueden incluirse más de una instrucción de definición de tabla, de definición de vista o instrucción `grant/revoke/deny`.

Ejemplo1: En la BD SOCIOS vamos a crear el esquema *Maestras* propiedad del usuario existente USU1, y en la misma instrucción creamos la tabla PAIS. Además también le concedemos permisos de consulta a los usuarios existentes USU2 y USU3:

```
USE SOCIOS;
GO
CREATE SCHEMA Maestras AUTHORIZATION USU1
    CREATE TABLE PAIS
    (codigo char(3) not null,
     descripcion varchar(30) not null,
     CONSTRAINT PK_pais PRIMARY KEY(codigo))
    GRANT SELECT TO USU2, USU3;
GO
```

3.4.3. Mover un elemento de un esquema a otro

Sintaxis:

```
ALTER SCHEMA nombre_del_esquema
TRANSFER [ { object |  type } :: ] nombre_del_elemento;
```

Ejemplo: Vamos a cambiar el objeto PERSONA del esquema *Ventas* al esquema *Personal*:

```
 ALTER SCHEMA Personal
TRANSFER Ventas.PERSONA;
```

3.4.4. Asignar un esquema por defecto a un usuario

Sintaxis:

```
ALTER USER nombre_del_usuario
WITH DEFAULT_SCHEMA= nombre_del_esquema;
```

Ejemplo: Vamos a asignar el esquema *Ventas* por defecto al usuario existente USU2:

```
ALTER USER USU2
WITH DEFAULT_SCHEMA=Ventas;
```

3.4.5. Cambiar el propietario de un esquema

Sintaxis:

```
ALTER AUTHORIZATION
ON SCHEMA::nombre_del_esquema
TO nombre_del_nuevo_propietario;
```

IMPORTANTE: ALTER AUTHORIZATION permite cambiar la propiedad de cualquier entidad que tenga propietario, no sólo esquemas (bases de datos, tablas, vistas, funciones…)

Ejemplo: Suponiendo que el usuario USU1 sea el propietario del esquema Ventas, vamos a cambiarlo para que el propietario sea USU3.

```
ALTER AUTHORIZATION
ON SCHEMA::Ventas
TO USU3;
```

IMPORTANTE: Al tener este usuario un esquema en propiedad, ya no podremos eliminarlo, salvo que se transfiera la propiedad del esquema a otro usuario.

3.4.6. Borrar un esquema

Sintaxis:

```
DROP SCHEMA nombre_del_esquema;
```

IMPORTANTE: El esquema que se va a eliminar no puede contener objetos.

Ejemplo: En la BD EMPLEADOS vamos a crear el esquema *Pruebas* propiedad del usuario existente USU1, y en la misma instrucción creamos la tabla *prueba_borrado_esquema*.

```
USE EMPLEADOS;
GO
CREATE SCHEMA Pruebas AUTHORIZATION USU1
    CREATE TABLE prueba_borrado_esquema
    (numero int  not null,
     nombre varchar(40) not null,
     CONSTRAINT PK_prueba_borrado_esquema PRIMARY KEY(numero));
GO
--Para eliminar el esquema primero eliminamos la tabla
DROP TABLE Pruebas.prueba_borrado_esquema;
DROP SCHEMA Pruebas;
GO
```

3.5. ROLES O FUNCIONES

Un **rol** (o **función** en versiones anteriores de SQL Server) es un tipo de objeto de BD que facilita la creación de grupos de usuarios, ya que en lugar de concederle los permisos a cada usuario, se les asigna un rol, y es al rol al que se le asignan los permisos. Aunque creemos roles podremos seguir dando permisos directamente a cada usuario. Los roles son como los grupos del SO MS Windows.

Se pueden crear roles de BD cuando un grupo de usuarios necesite realizar el mismo conjunto de actividades. Así en lugar de otorgar los permisos a cada usuario, se otorgan al rol, y lo que se hace es asignar el usuario al rol.

IMPORTANTE: Los roles existen en cada BD y no pueden abarcar más de una BD.

SQL Server proporciona roles o funciones de servidor y de BD, pero también se pueden crear roles o funciones de BD definidos por el usuario.

Pueden ser:

- **Rol estándar**
- **Rol de aplicación:** para establecer los permisos de una aplicación sobre una BD.

3.5.1. Roles de nivel de servidor

Permiten agrupar los privilegios administrativos en el **nivel del servidor** (y no a nivel de cada BBDD). Se administran de forma independiente de las BBDD de usuario. No se pueden crear, ni modificar ni eliminar este tipo de funciones.

Roles (funciones) de nivel DE SERVIDOR Roles de nivel de servidor y permisos que tienen los usuarios miembros de estos roles.	
Nombre del Rol *(o función)*	**Permisos**
sysadmin	Cualquier actividad en el servidor.
dbcreator	Crear, modificar, quitar y restaurar bases de datos.
diskadmin	Administrar archivos de disco.
processadmin	Administrar procesos que se ejecutan en una instancia del servidor.
serveradmin	Cambiar las opciones de configuración del servidor y apagarlo.
setupadmin	Agregar y quitar servidores vinculados.
securityadmin	Administrar los inicios de sesión y sus usuarios. Administran los permisos GRANT, REVOKE y DENY a nivel de servidor y de bd y restablecer las contraseñas para los inicios de sesión de SQL Server.
bulkadmin	Ejecutar instrucciones BULK INSERT *Copia un archivo de datos a una tabla o vista de base de datos en un formato especificado por el usuario.*
public	Cada inicio de sesión de SQL Server pertenece al rol public de servidor. Los permisos asignados a public estarán disponibles para todos los usuarios.

IMPORTANTE: Microsoft recomienda no asignar permisos tan generales sino concretar los permisos en la medida de lo posible. Microsoft los sigue proporcionando por compatibilidad con versiones anteriores y con fines de comodidad para el administrador.

TRABAJAR CON ROLES DE NIVEL DE SERVIDOR Comandos, vistas y funciones para trabajar con los roles de nivel de servidor.	
Comando/Vista/Función	**Descripción**
sp_helpsrvrole	Devuelve una lista de roles de nivel de servidor.
sp_helpsrvrolemember	Devuelve información acerca de los miembros de un rol de nivel de servidor.
sp_srvrolepermission	Muestra los permisos de un rol de nivel de servidor.
IS_SRVROLEMEMBER	Indica si un inicio de sesión de SQL Server es miembro del rol de nivel de servidor especificado.
sys.server_role_members	Devuelve una fila por cada miembro de cada rol de nivel de servidor.
sp_addsrvrolemember	Agrega un inicio de sesión como miembro de un rol de nivel de servidor.
sp_dropsrvrolemember	Quita un inicio de sesión de SQL Server o un usuario o grupo de Windows de un rol de nivel de servidor.

3.5.2. Roles de nivel de base de datos

Permiten agrupar los privilegios administrativos en el **nivel de cada bd**. Se administran de forma independiente de las BBDD de usuario. Este tipo de roles no se pueden crear, ni modificar ni eliminar. Existen 2 tipos:

- **Roles fijos de bd:** son los que están predefinidos en la bd.
- **Roles flexibles de bd:** pueden crearse.

IMPORTANTE: Microsoft recomienda no agregar roles flexibles de bd (creados por el usuario) como miembros de roles fijos, ya que esto podría habilitar más privilegios de los deseados.

3.5.3. Roles fijos de nivel de BD

Roles (funciones) FIJOS de nivel DE BASE DE DATOS	
Nombre del Rol *(o función)*	**Permiso**
public	Mantiene todos los permisos predeterminados de una BD. **IMPORTANTE:** La función public es una función de BD especial a la que pertenecen todos los usuarios de la BD y que no se puede eliminar.
db_owner	Realizar cualquier actividad en la BD (configurar, mantener y quitar).
db_accessadmin	Agregar o quitar el acceso a la BD para inicios de sesión de Windows, grupos de Windows e inicios de sesión de SQL Server.
db_ddladmin	Ejecutar cualquier comando DDL en la BD.
db_securityadmin	Administrar permisos y modificar la pertenencia a roles. **IMPORTANTE:** Asignar miembros a esta función puede implicar un aumento de privilegios no deseado.
db_backupoperator	Realizar copias de seguridad y restauraciones.
db_datareader	Leer datos de cualquier tabla de usuario.
db_datawriter	Agregar, modificar o eliminar datos de cualquier tabla de usuario.
db_denydatareader	No puede leer datos de las tablas de usuario de la BD.
db_denydatawriter	No puede agregar, modificar o eliminar datos de las tablas de usuario.

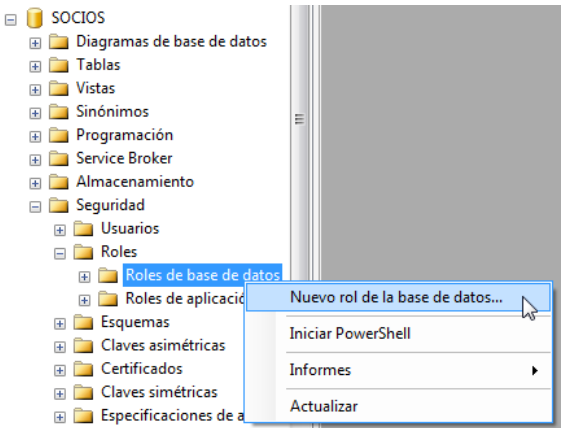
TRABAJAR CON ROLES DE NIVEL DE BASE DE DATOS Comandos, vistas y funciones para trabajar con los roles de nivel de BD.	
Comando/Vista/Función	**Descripción**
sp_helpdbfixedrole	Devuelve una lista de roles **fijos** de nivel de base de datos.
sp_dbfixedrolepermission	Muestra los permisos de un rol **fijo** de base de datos.
sp_helprole	Devuelve información acerca de los roles de la base de datos actual.
sp_helprolemember	Devuelve información acerca de los miembros de un rol de la base de datos actual.
sys.database_role_members	Devuelve una fila por cada miembro de cada rol de base de datos.
IS_MEMBER	Indica si el usuario actual es miembro del grupo de Microsoft Windows o del rol de base de datos de SQL Server especificados.
CREATE ROLE	Crea un nuevo rol de base de datos en la base de datos actual.
ALTER ROLE	Cambia el nombre de un rol de base de datos.
DROP ROLE	Quita un rol de la base de datos.
sp_addrole	Crea un nuevo rol de base de datos en la base de datos actual.
sp_droprole	Quita un rol de base de datos de la base de datos actual.
sp_addrolemember	Agrega un usuario de base de datos, un rol de base de datos, un inicio de sesión de Windows o un grupo de Windows a un rol de base de datos en la base de datos actual.
sp_droprolemember	Quita una cuenta de seguridad de un rol de SQL Server de la base de datos actual.

3.5.4. Roles flexibles de nivel de BD

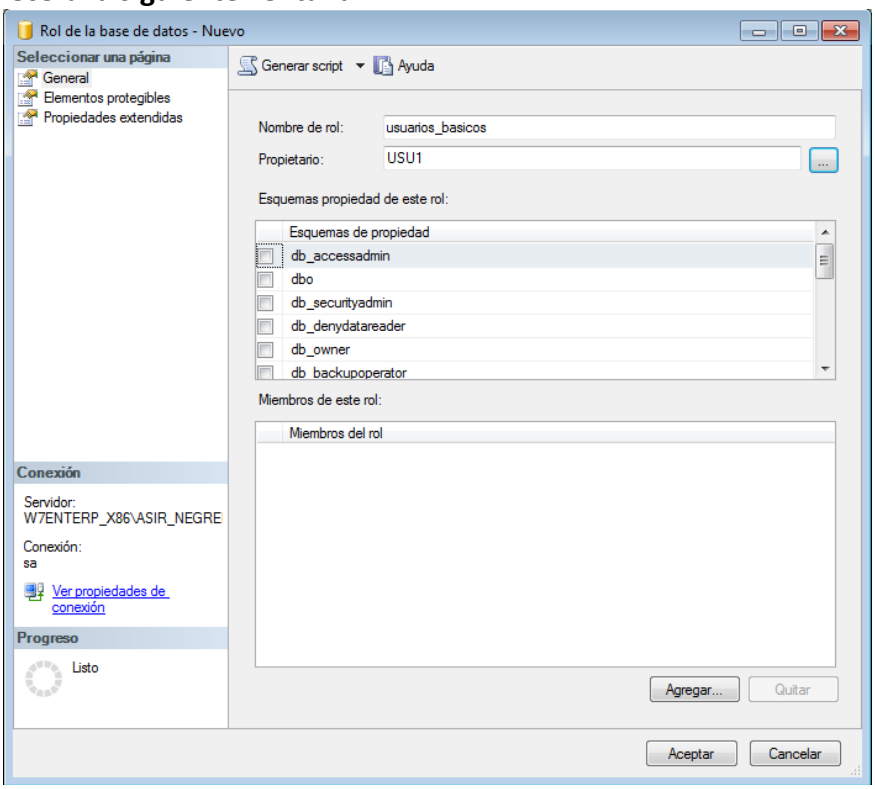
Los roles flexibles de BD son **roles definidos por el usuario** que permiten crear un grupo de usuarios con un conjunto de permisos comunes. Sería de utilidad, por ejemplo, cuando tenemos usuarios que tienen permiso sólo de consulta en determinadas tablas. Crearíamos una función que permitiese sólo leer (SELECT) esas tablas y después asignaríamos a los usuarios como miembros de la función creada.

3.5.5. Crear roles de BD

A) **Gráficamente:** Desde el explorador de objetos, en la bd donde queremos crear el rol en la carpeta **Seguridad,** accedemos a la carpeta **Roles** (en versiones anteriores a 2008R2 funciones), y en el menú contextual del nodo Roles de bases de datos escogemos **Nuevo rol de base de datos…**



Aparecerá la siguiente ventana



B) Con Transact-SQL:

Sintaxis:

```
CREATE ROLE nombre_del_rol_a_crear [ AUTHORIZATION nombre_propietario ];
```

`AUTHORIZATION nombre_propietario`: Usuario o rol de bd propietario del nuevo rol. Si no se especifica, el rol será propiedad del usuario que ejecute la instrucción CREATE ROLE.

Ejemplo: En la bd SOCIOS creamos el rol *usuarios_basicos* y le asignamos como propietario al usuario:

```
CREATE ROLE usuarios_basicos
AUTHORIZATION USU1;
```

3.5.6. Cambiar el nombre de un rol de BD

Sintaxis:

```
ALTER ROLE nombre_del_rol WITH NAME= nuevo_nombre_de_rol;
```

Ejemplo: Cambiamos el nombre del rol creado anteriormente de *usuarios_basicos* a *usuarios_ingenuos*:

```
ALTER ROLE usuarios_basicos
WITH NAME= usuarios_ingenuos;
```

3.5.7. Borrar un rol de bd

Sintaxis:

```
DROP ROLE nombre_del_rol_a_borrar;
```

Ejemplo: Eliminamos el rol de la bd SOCIOS *usuarios_ingenuos*:

```
DROP ROLE usuarios_ingenuos;
```

IMPORTANTE: Para poder eliminar un rol de base de datos no puede tener miembros.

3.5.8. Asignar un usuario a un rol

Sintaxis:

```
sp_addrolemember nombre_del_rol, usuario_a_asignar_al_rol;
```

Ejemplo: Asignamos al usuario Maria al rol *usuarios_ingenuos* de la bd SOCIOS:

```
USE SOCIOS;
GO
sp_addrolemember 'usuarios_ingenuos', 'Maria';
```

3.6. GESTIÓN DE PERMISOS

El usuario debe tener los permisos adecuados para realizar las actividades que el administrador desee permitirle. La administración de permisos incluye la concesión o revocación de los mismos.

3.6.1. Conceder permisos. La sentencia GRANT

La sentencia **GRANT** crea una entrada en el sistema de seguridad que permite a un usuario de la base de datos actual trabajar con datos de la bd actual o ejecutar instrucciones SQL específicas.

Sintaxis para permisos de la instrucción:

```
GRANT { ALL | instrucc1 [, instrucc2,...instruccN ] }
TO entidad_de_seguridad1 [,entidad_de_seguridad12,... entidad_de_seguridadN];
```

Sintaxis para permisos del objeto:

```
GRANT { ALL [ PRIVILEGES ] | permiso1 [, permiso2,...permisoN ] [(col1
[,...] ) ] }
{ ON { tabla | vista }
| ON { tabla | vista } [(col1 [,...] ) ]
| ON { proced_almacenado | proced_almacenado_extendido }
| ON { funcion_definida_por_el_usuario }
| ON SCHEMA::esquema
}
TO entidad_de_seguridad1 [,entidad_de_seguridad12,... entidad_de_seguridadN]|
PUBLIC
[ WITH GRANT OPTION ]
```

- **ALL:** Se conceden todos los permisos aplicables. NO concede todos los permisos posibles. *(Está en desuso y sólo se mantiene por compatibilidad).*
- **Instrucc:** es la instrucción para la que se concede el permiso:
  - Si el elemento protegible es una **base de datos**: **CREATE DATABASE**, **CREATE FUNCTION**, **CREATE PROCEDURE**, **CREATE RULE**, **CREATE TABLE**, **CREATE VIEW**, **BACKUP DATABASE** y **BACKUP LOG.**
- **permiso1 [,...n]:** Es el permiso de objeto que se concede.
  - Si es una **función escalar: EXECUTE y REFERENCES.**
  - Si es una **función de tabla: DELETE, INSERT, REFERENCES, SELECT y UPDATE.**
  - Si es un **procedimiento almacenado: EXECUTE.**
  - Si es una **tabla/vista: DELETE, INSERT, REFERENCES, SELECT, UPDATE, ALTER e INDEX.**
  - Si se están concediendo **permisos sobre una columna o una lista de** columnas, podemos usar los permisos **SELECT, UPDATE ó REFERENCES**, de tal manera que el permiso sólo afectará a esas columnas.
- **TO *entidad_de_seguridad1 [,entidad_de_seguridad12,... entidad_de_seguridadN]*:** Especifica la cuenta o lista de cuentas de usuario a la que se le conceden el/los permisos. También puede especificar un rol de SQL Server, o bien un usuario o grupo de usuarios de MS Windows®.
- **WITH GRANT OPTION:** Esta cláusula opcional especifica que se concede a la cuenta o cuentas indicadas en TO, la capacidad de conceder el/los permiso/s de objeto especificado a otras cuentas.

Ejemplo: Conceder al usuario USU1 de la bd SOCIOS el permiso de consulta y borrado sobre la tabla CUOTA, con permiso para concederlos a otros.

```
GRANT  SELECT, DELETE
ON CUOTA
TO USU1
WITH GRANT OPTION;
```

IMPORTANTE: Si en lugar de una cuenta de usuario especificamos la función/rol PUBLIC, estaremos concediendo el/los permiso/s a todos los usuarios de la BD.

3.6.2. Denegar permisos. La sentencia DENY

Deniega un permiso previamente concedido, de tal manera que aunque otro usuario de su grupo (de Windows) o función/rol (de SQL Server) intente otorgárselo de nuevo no podrá tenerlo.

Evita que la entidad de seguridad herede permisos por su pertenencia a grupos o roles.

Sintaxis para permisos de la instrucción:

```
DENY { ALL |  instrucc1 [, instrucc2,...instruccN ] }
TO entidad_de_seguridad1 [,entidad_de_seguridad12,... entidad_de_seguridadN];
```

Sintaxis para permisos del objeto:

```
DENY   { ALL [ PRIVILEGES ]    |   permiso1 [, permiso2,...permisoN ] [(col1
[,...] ) ]  }
{  ON { tabla | vista }
|  ON { tabla | vista } [(col1 [,...] ) ]
|  ON { proced_almacenado  | proced_almacenado_extendido }
|  ON  { funcion_definida_por_el_usuario }
|  ON SCHEMA::esquema

}
TO entidad_de_seguridad1 [,entidad_de_seguridad12,... entidad_de_seguridadN] |
PUBLIC
[ CASCADE  ]
```

- **CASCADE:** Indica que el permiso se deniega para la entidad de seguridad especificada y para el resto de entidades de seguridad a las que ésta le concedió el permiso. Es obligatorio cuando la entidad de seguridad tiene el permiso GRANT OPTION.

Ejemplo: Denegar al usuario USU1 de la bd SOCIOS el permiso de consulta sobre la tabla CUOTA.

```
DENY  SELECT
ON CUOTA
TO USU1
CASCADE;
```

IMPORTANTE: Un permiso DENY de nivel de tabla no tiene prioridad sobre uno GRANT de nivel de columna. Esta incoherencia en la jerarquía de permisos se ha mantenido por motivos de compatibilidad con versiones anteriores. Se quitará en una versión futura.

3.6.3. Revocar permisos: La sentencia REVOKE

Permite revocar un permiso que previamente se ha otorgado o denegado. A diferencia de **DENY**, aunque **REVOKE** también retira un permiso concedido, **no impide que el usuario lo herede de un nivel superior** (de su grupo o rol).

Sintaxis para permisos de la instrucción:

```
REVOKE [ GRANT OPTION FOR ]
 { ALL |  instrucc1 [, instrucc2,...instruccN ] }
{TO    |    FROM    }    entidad_de_seguridad1    [,entidad_de_seguridad12,...
entidad_de_seguridadN];
```

Sintaxis para permisos del objeto:

```
REVOKE {   ALL [ PRIVILEGES ]    |   permiso1 [, permiso2,...permisoN ] [(col1
[,...] ) ] }
{  ON { tabla | vista }
|  ON { tabla | vista } [(col1 [,...] ) ]
|  ON { proced_almacenado  | proced_almacenado_extendido }
|  ON  { funcion_definida_por_el_usuario }
|  ON SCHEMA::esquema
}
{TO    |    FROM    }    entidad_de_seguridad1    [,entidad_de_seguridad12,...
entidad_de_seguridadN] | PUBLIC
[ CASCADE  ];
```

- **GRANT OPTION FOR:** Se le quita al usuario la capacidad de conceder el permiso especificado. Se requiere CASCADE.
- **CASCADE:** Indica que el permiso se deniega para la entidad de seguridad especificada y para el resto de entidades de seguridad a las que ésta le concedió el permiso. Es obligatorio cuando la entidad de seguridad tiene el permiso GRANT OPTION.

Ejemplo: Revoca al usuario USU1 de la bd SOCIOS el permiso de borrado sobre la tabla CUOTA.

```
REVOKE  DELETE
ON CUOTA
TO USU1
CASCADE;
```

IMPORTANTE: Un permiso DENY de nivel de tabla no tiene prioridad sobre uno GRANT de nivel de columna. Esta incoherencia en la jerarquía de permisos se ha mantenido por motivos de compatibilidad con versiones anteriores. Se quitará en una versión futura.

En el Management Studio podemos administrar permisos de usuario con la opción **Elementos que pueden protegerse** de la ventana que aparece cuando hacemos doble clic sobre el nombre del usuario.

En el caso de los roles (o funciones) haremos los mismo pero seleccionando el rol en el que deseemos consultar o cambiar los permisos. En este caso la opción del menú contextual a elegir es la de Propiedades.

Instrucción DCL	Descripción
GRANT	*Puede ejecutar una acción*
DENY	*No puede ejecutar una acción* y **NO puede ser sobrescrito** por la pertenencia a una función (NO puede heredarlo)
REVOKE	*No puede ejecutar una acción* pero **SÍ puede ser sobrescrito** por la pertenencia a una función (SÍ puede heredarlo)

4. COPIAS DE SEGURIDAD

Como acabamos de ver, uno de los mecanismos de seguridad para poder recuperar la información de las bases de datos ante un fallo son las **copias de seguridad**.

En las BBDD hay que considerar **dos tipos de copias**:

- **Copias de seguridad (BACKUP):** Periódicamente se deben hacer copias y guardarlas en lugar seguro. Estas deben basarse en **copias completas** (por ej. cada semana) y diferenciales (cada día o fracción) para facilitar la recuperación y no hacer caer el rendimiento del gestor con copias frecuentes.
- **Registro histórico (LOG):**
  - El **log se debe almacenar en un disco distinto a los datos** de forma que éste no se pierda a no ser que el fallo sea catastrófico.
  - También **se debe realizar copia de seguridad del log**, de forma que se pueda restaurar la base de datos desde su último backup hasta la última situación estable antes del fallo.

4.1. MODELOS DE RECUPERACIÓN

El **modelo de recuperación** es una propiedad de la base de datos:

```
Alter DataBase nombre_BD
Set Recovery     Full | Simple| Bulk_Logged
```

Para comprobar el modelo de recuperación de las bases de datos del sistema:

```
SELECT  Name as BD, Recovery_Model_Desc As Modelo_de_Recuperación
FROM  sys.databases
```

En MS SQLServer® existen 3 modelos de recuperación:

- **Modelo de recuperación simple (Simple):** Permite recuperar la BD hasta la copia de seguridad más reciente (mezcla copias completas y copias diferenciales). En este modelo, cada transacción que se copia a disco se elimina del log.
- **Modelo de recuperación completa (Full):** Permite recuperar la BD hasta el momento del error.
- **Modelo de recuperación de registro masivo (Bulk_Logged):** Es similar a la completa pero tiene la desventaja de que no permite restaurar la copia de seguridad sólo hasta una marca. Se usa sobre todo en entornos con bajo nivel de actualizaciones. *NO ES OBJETO DE ESTE TEMA y por lo tanto sólo estudiaremos los modelos Simple y Completo*.

MODELOS DE RECUPERACIÓN	
Modelo de recuperación	**Copias de seguridad para recuperar la BD**
SIMPLE	– **Completa** – **Diferencial**
COMPLETA	– **Completa** – **Diferencial** – **De LOG**
DE REGISTRO MASIVO	NO ES OBJETO DE ESTE TEMA

4.2. MODELO DE RECUPERACIÓN SIMPLE

Como ya hemos indicado, este modelo para recuperar una base de datos utiliza:

- las copias de seguridad **completas, y,**
- las copias de seguridad **diferenciales**.
- Vamos a ver esos dos tipos de **copia** de seguridad de BD así como su **restauración**.

4.2.1. Copia de seguridad de BD COMPLETA

Una copia de seguridad completa copia todos los datos de la BD.

Copia de seguridad de BD COMPLETA USANDO TRANSACT-SQL	
A DISCO	`BACKUP DATABASE nombre_bd TO DISK = 'ruta_disco\nombre_del_archivo';`
A CINTA	`BACKUP DATABASE nombre_bd TO TAPE = 'ruta_cinta\nombre_cinta';`
A DISPOSITIVO de copia de seguridad	`BACKUP DATABASE nombre_bd TO nombre_dispositivo`

***Algunas* opciones que se pueden indicar en la copia (son opcionales)**	`WITH FORMAT,` *`(para que formatee el dispositivo)`*
	`NAME='Nombre que le queremos dar a la copia',`
	`DESCRIPTION='Descripción de la copia',`
	`NOINIT | INIT` *`(para que la copia se anexe o para que sobrescriba la existente)`*

Ejemplo:

```
BACKUP DATABASE EMPLEADOS TO DISK='C:\BACKUPS\backup_empleados.bak'
WITH FORMAT,
NAME='copia_emple_diaria',
DESCRIPTION='es un ejemplo',
INIT;
```

4.2.2. Copia de seguridad de BD DIFERENCIAL

Se usa en bases de datos que se modifican frecuentemente.

- **Requiere que exista una copia de seguridad completa**.
- Hace **copia de seguridad de los cambios de la base de datos desde la última copia de seguridad completa**. *Esta copia completa se denomina* base de copia de seguridad diferencial o **BASE DIFERENCIAL**.
- Usándolas **se disminuye el tiempo** tanto del proceso de copia como del de restauración.
- Debe establecerse una **convención de nombres de los archivos de copia de seguridad**, que permita distinguir cuáles hacen referencia a copias de seguridad completas y cuáles a diferenciales (o incrementales).

Copia de seguridad de BD DIFERENCIAL USANDO TRANSACT-SQL	
A DISCO	`BACKUP DATABASE nombre_bd TO DISK = 'ruta_disco\nombre_del_archivo'` `WITH DIFFERENTIAL;`
A CINTA	`BACKUP DATABASE nombre_bd TO TAPE = 'ruta_cinta\nombre_cinta'` `WITH DIFFERENTIAL;`
A DISPOSITIVO de copia de seguridad	`BACKUP DATABASE nombre_bd TO nombre_dispositivo` `WITH DIFFERENTIAL;`

***Algunas* opciones que se pueden indicar en la copia (son opcionales)**	`WITH FORMAT, (para que formatee el dispositivo)`
	`NAME='Nombre que le queremos dar a la copia',`
	`DESCRIPTION='Descripción de la copia',`
	`NOINIT | INIT (para que la copia se anexe o para que sobrescriba la existente)`

Ejemplo:

```
BACKUP DATABASE SOCIOS TO DISK='C:\BACKUPS\backup_socios.bak'
WITH DIFFERENTIAL,
FORMAT,
NAME='copia_socios_diferencial',
DESCRIPTION='es un ejemplo de una copia diferencial de la BD SOCIOS',
INIT;
```

4.2.3. RESTAURACIÓN de copia de seguridad de BD COMPLETA

Antes de restaurar debemos hacer lo siguiente:

- **Restringir el acceso** a la BD.
- Hacer una **copia de seguridad del registro de transacciones** (del LOG).

Restauración de copia de seguridad de BD COMPLETA USANDO TRANSACT-SQL	
DESDE DISCO	**RESTORE** DATABASE nombre_bd **FROM** DISK ='ruta_disco\nombre_del_archivo';
DESDE CINTA	**RESTORE** DATABASE nombre_bd **FROM** TAPE = 'ruta_cinta\nombre_cinta';
DESDE DISPOSITIVO de copia de seguridad	**RESTORE** DATABASE nombre_bd **FROM** nombre_dispositivo;
***Algunas* opciones que se pueden indicar en la restauración (son opcionales)**	**WITH RECOVERY** Indica a la operación de restauración que **deshaga las transacciones no confirmadas.** Después del proceso de recuperación, la base de datos está preparada para ser utilizada. Si las siguientes operaciones RESTORE (RESTORE LOG o RESTORE DATABASE a partir de una copia de seguridad diferencial) están planeadas, se debe especificar en su lugar **WITH NORECOVERY** *Hay muchas más opciones, de hecho cualquiera de las opciones que se pueden marcar en las ventanas de restauración del Management Studio, pueden indicarse en T-SQL. Si quieres conocerlas consulta la sintaxis de RESTORE DATABASE en los libros de ayuda de SQLServer.*
Ejemplo: **RESTORE DATABASE** EMPLEADOS FROM DISK='C:\BACKUPS\backup_empleados.bak' **WITH RECOVERY;**	

4.2.4. RESTAURACIÓN de copia de seguridad de BD DIFERENCIAL

Actuaremos de igual modo que en la restauración de una copia de seguridad completa.

Restauración de copia de seguridad de BD DIFERENCIAL USANDO TRANSACT-SQL
Igual que la restauración de una copia de seguridad completa

4.3. MODELO DE RECUPERACIÓN COMPLETA

Este modelo para recuperar una base de datos utiliza:

- las **copias de seguridad completas,**
- las **copias de seguridad diferenciales, y,**
- **las copias del registro de transacciones**.

4.3.1. Copia de seguridad de REGISTRO DE TRANSACCIONES (LOG)

Vamos a ver el tipo de copia de seguridad de BD que nos falta de los tres, el de registro de transacciones o de log, pero antes vamos a definir el registro de transacciones:

- **REGISTRO DE TRANSACCIONES o de LOG: Es un registro que** guarda en serie (unas después de otras) todas las transacciones (operaciones) **que se han realizado en la BD desde la última copia de seguridad del registro de transacciones.**
- **Al copiar el registro de transacciones** podemos recuperar la BD hasta **un momento determinado, justo** antes de que se haya producido un error**.**
- En SQL Server el LOG tiene dos funciones:
  - **Almacenar el histórico de todas las transacciones de la base de datos,** principalmente para que los cambios de dichas transacciones se hagan en memoria y no en disco y funcione mucho más rápido (hay un proceso posterior llamado Ahead Logging que se encarga de guardar las páginas de memoria sucias (las que han cambiado), a través de una transacción a fichero de datos físico de la base de datos.
  - **Servir como recuperación de la base de datos hasta el momento justo en que se cayó la base de datos**, que es el caso que nos ocupa. Para que la recuperación sea efectiva, el fichero de log debe estar en un disco duro distinto del del fichero de datos de la base de datos. Eso hace que si falla el disco de datos, el fichero de log aún sea accesible, además de mejorar el rendimiento de E/S general de la base de datos ya que no compiten el log y el fichero de datos por el acceso al mismo disco.

Copia de seguridad de registro de transacciones USANDO TRANSACT-SQL	
A DISCO	`BACKUP LOG nombre_bd TO DISK = 'ruta_disco\nombre_del_archivo';`
A CINTA	`BACKUP LOG nombre_bd TO TAPE = 'ruta_cinta\nombre_cinta';`
A DISPOSITIVO de copia de seguridad	`BACKUP LOG nombre_bd TO nombre_dispositivo`
***Algunas* opciones que se pueden indicar en la copia (son opcionales)**	**`IMPORTANTE: La parte inactiva del log es aquella formada por las transacciones que ya han sido escritas a disco.`**

La palabra WITH sólo se pone una vez	**WITH** NO_TRUNCATE, *(para no truncar la parte inactiva del registro de transacciones)* TRUNCATE_ONLY ó NO_LOG, *(limpia el registro de transacciones sin copiarlo)* FORMAT, *(para que formatee el dispositivo)* NAME='Nombre que le queremos dar a la copia', DESCRIPTION='Descripción de la copia', NOINIT | INIT *(para que la copia se anexe o para que sobrescriba la existente)*

Ejemplo:

```
BACKUP LOG EMPLEADOS TO [Copia LOG EMPLEADOS]
WITH NAME='copia_log_emple',
DESCRIPTION='es un ejemplo de copia de log';
```

*Los **corchetes** son necesarios porque el nombre del dispositivo de copia tiene espacios en blanco. También es necesario si el nombre tiene caracteres especiales*

4.3.2. RESTAURACIÓN de copia de seguridad de REGISTRO DE TRANSACCIONES (LOG)

Restauración del registro de transacciones USANDO TRANSACT-SQL	
DESDE DISCO	RESTORE **LOG** nombre_bd FROM DISK ='ruta_disco\nombre_del_archivo';
DESDE CINTA	RESTORE **LOG** nombre_bd FROM TAPE = 'ruta_cinta\nombre_cinta';
DESDE DISPOSITIVO de copia de seguridad	RESTORE **LOG** nombre_bd FROM nombre_dispositivo;
***Algunas* opciones que se pueden indicar en la restauración (son opcionales)**	**WITH RECOVERY** Indica a la operación de restauración que **deshaga las transacciones no confirmadas**. Después del proceso de recuperación, la base de datos está preparada para ser utilizada. Si las siguientes operaciones RESTORE (RESTORE LOG o RESTORE DATABASE a partir de una copia de seguridad diferencial) están planeadas, se debe especificar en su lugar **WITH NORECOVERY** *Hay muchas más opciones, de hecho cualquiera de las opciones que se pueden marcar en las ventanas de restauración del Management Studio, pueden indicarse en T-SQL. Si quieres conocerlas consulta la sintaxis de RESTORE DATABASE en los libros de ayuda de SQLServer.*

Ejemplo:

```
RESTORE LOG SOCIOS
FROM DISK='C:\BACKUPS\backup_log_socios.bak'
WITH RECOVERY;
```

4.4. DISPOSITIVOS PERMANENTES DE COPIAS DE SEGURIDAD

- Son **dispositivos físicos donde se almacenan una o más copias de seguridad**, de la misma BD o de distintas y del mismo tipo (completas sólo, por ejemplo) o de distinto tipo (diferenciales y completas, por ejemplo).
- Realmente **consiste en asociar un nombre a la ruta/nombre del fichero donde se almacena la copia** de seguridad.
- Se denominan también **MEDIOS de copia de seguridad**.

Creación de un dispositivo permanente de copia de seguridad de BD USANDO TRANSACT-SQL

```
[EXECUTE] sp_addumpdevice 'tipo', 'nombre', 'ruta\fichero.bak'
```

- Para crear un dispositivo de copia de seguridad se usa este procedimiento almacenado del sistema
- EXEC se pone entre corchetes porque es opcional por ser un procedimiento almacenado del sistema
- Tipo, nombre y la ruta deben ir siempre entre **comillas simples**
- **'tipo':** hace referencia a si está en un disco o en una cinta, por lo que se pondrá **DISK o TAPE**
- **'nombre':** nombre del dispositivo que se está creando
- **'ruta\fichero.bak':** Se especifica la ruta completa del fichero asociado al dispositivo

4.5. COPIA DE SEGURIDAD DE ARCHIVOS Y GRUPOS DE ARCHIVOS

- Cuando el tamaño y los requisitos de rendimiento de la base de datos hagan que no sea práctico realizar una copia de seguridad completa de la base de datos, se puede crear una copia de seguridad de archivo en su lugar.

- Una copia de seguridad de archivo contiene todos los datos de uno o varios archivos (o grupos de archivos).

Restauración del registro de transacciones USANDO TRANSACT-SQL

A DISCO	BACKUP DATABASE *nombre_bd* { **FILE** = *nombre_lógico_del_fichero* | **FILEGROUP** = *nombre_lógico_grupo_ficheros* } [,...*f*] TO DISK ='ruta_disco\nombre_del_archivo' [WITH *opción* [,...*o*]] ;
A CINTA	BACKUP DATABASE *nombre_bd* { **FILE** = *nombre_lógico_del_fichero* | **FILEGROUP** = *nombre_lógico_grupo_ficheros* } [,...*f*] TO TAPE = 'ruta_cinta\nombre_cinta' [WITH *opción* [,...*o*]] ;
A DISPOSITIVO de copia de seguridad	BACKUP DATABASE *nombre_bd* { **FILE** = *nombre_lógico_del_fichero* | **FILEGROUP** = *nombre_lógico_grupo_ficheros* } [,...*f*] TO nombre_dispositivo [WITH *opción* [,...*o*]] ;

Ejemplo1: Creamos una copia de seguridad de 2 archivos:

```
BACKUP DATABASE Sales
FILE = 'SalesGrupo1Fich2',
FILE = 'SalesGrupo2Fich2'
TO DISK = 'D:\ Backups\BD_Sales\SalesFich2.bak';
```

Ejemplo2: Creamos una copia de seguridad de todos los archivos de 2 grupos:

```
BACKUP DATABASE Sales
FILEGROUP = 'SalesGrupo1',
FILEGROUP = 'SalesGrupo2'
TO DISK = 'D:\ Backups\BD_Sales\SalesGrupos1y2.bak';
```

4.6. COPIA DE SEGURIDAD DE SÓLO COPIA

- Una **copia de seguridad de solo copia** es una copia de seguridad de SQL Server independiente de la secuencia de copias de seguridad convencionales de SQL Server. Normalmente, la realización de una copia de seguridad cambia la base de datos y afecta a la forma de restaurar las copias de seguridad posteriores. Sin embargo, a veces es útil realizar una copia de seguridad con un fin específico sin afectar a los procedimientos generales de copias de seguridad y restauración de la base de datos.

- Los **tipos de copias de seguridad de solo copia** son los siguientes:
  - **Copias de seguridad completas de solo copia (todos los modelos de recuperación):** Una copia de seguridad completa de solo copia no puede servir como base diferencial ni copia de seguridad diferencial y no afecta a la base diferencial.
  - **Copias de seguridad de log de solo copia.**

COPIA DE SEGURIDAD DE SÓLO COPIA	
A DISCO	`BACKUP DATABASE|LOG nombre_bd TO DISK = 'ruta_disco\nombre_del_archivo' WITH COPY_ONLY;`
A CINTA	`BACKUP DATABASE|LOG nombre_bd TO TAPE = 'ruta_cinta\nombre_cinta' WITH COPY_ONLY;`
A DISPOSITIVO de copia de seguridad	`BACKUP DATABASE|LOG nombre_bd TO nombre_dispositivo WITH COPY_ONLY;`

Ejemplo:

```
BACKUP LOG EMPLEADOS TO [Copia LOG EMPLEADOS]
WITH NAME='copia_log_emple',
DESCRIPTION='es un ejemplo de copia de log',
COPY_ONLY;
```

*Los **corchetes** son necesarios porque el nombre del dispositivo de copia tiene espacios en blanco. También es necesario si el nombre tiene caracteres especiales*

4.7. COPIA DE SEGURIDAD EN MÁS DE UN DISPOSITIVO

- Se usa la cláusula **MIRROR TO**
- Especifica un conjunto de hasta tres dispositivos de copia de seguridad, cada uno de los cuales reflejará los dispositivos de copia de seguridad especificados en la cláusula TO.
- La cláusula MIRROR TO debe incluir el mismo número y tipo de dispositivos de copia de seguridad que la cláusula TO.
- El número máximo de cláusulas MIRROR TO es tres.

COPIA DE SEGURIDAD EN MÁS DE UN DISPOSITIVO

Se añade la cláusula **MIRROR TO**

Ejemplo:

```
BACKUP DATABASE  LIGA TO
    DISK = 'D:\ Backups\BD_LIGA\Liga1a.bak',
    DISK = 'D:\ Backups\BD_LIGA\Liga2a.bak',
    DISK = 'D:\ Backups\BD_LIGA\Liga3a.bak'
MIRROR TO
    DISK = 'D:\ Backups\BD_LIGA\Liga1b.bak',
    DISK = 'D:\ Backups\BD_LIGA\Liga2b.bak',
    DISK = 'D:\ Backups\BD_LIGA\Liga3b.bak';
```

4.8. CASO PRÁCTICO

Somos los DBA de una empresa.

- **Copia de seguridad completa a las 2:00hh:**

Imaginemos que hemos hecho la copia de seguridad de nuestra base de datos de contabilidad (de nombre CONTABILIDAD) a las 2:00 de la mañana con una sentencia similar a esta:

```
BACKUP   DATABASE CONTABILIDAD
TO   DISK = 'D:\backup\contabilidad.bak'
WITH INIT;
```

Esta sentencia nos permite hacer una copia de seguridad completa de la base de datos CONTABILIDAD al fichero contabilidad.bak.

- Se produce un **FALLO a las 19:00hh**:

Supongamos que la base de datos está activa al día siguiente, el personal de la empresa trabaja sin problemas con ella y justo antes de finalizar la jornada laboral, a las 7 de la tarde, el **disco de datos del servidor falla** y por tanto perdemos la base de datos.

- **¿Cómo resolvemos el problema?:**

Tenemos discos de repuesto y en poco tiempo tenemos el servidor online otra vez, pero al arrancar SQL Server, la base de datos de contabilidad no aparece… (normal porque hemos perdido el fichero de datos).

Lo primero que podemos pensar es en restaurar la copia de seguridad de la noche anterior (la de las 2 de la mañana que tenemos automatizada). Pero si hacemos eso perdemos todo lo que hemos hecho durante ese día. Si por ejemplo ese día se hubiesen cargado 1000 facturas sería un problemón tener que perder tiempo y dinero en meterlas todas…. Hay entornos críticos en los que perder esa información es inadmisible…

■ **SOLUCIÓN para evitar perder el trabajo realizado desde la copia:**

Una solución es utilizar el fichero de log. Si este fichero está en otro disco duro físico, entonces no habrá fallado y por tanto será accesible. Además este fichero, tendrá almacenados todo el histórico de todas las transacciones que se han hecho durante ese día, justo hasta el momento en que se cayó el sistema.

1º. Con SQL Server, podemos **recuperar** dicho **log** con esta sentencia:

```
BACKUP LOG    CONTABILIDAD
TO  DISK = 'H:\backup\final.bak'
WITH  INIT,
NO_TRUNCATE;
```

Fijémonos que **aquí estamos haciendo copia del fichero log, no del de los datos** y sobre todo, **fijémonos en el parámetro NO_TRUNCATE**. Este parámetro le dice a SQL Server **que no trunque el log** y que por tanto copie la parte final del log desde la última copia de seguridad, o sea desde las 2:00 de la mañana, hasta las 19:00 horas del día siguiente. Esto es lo que se denomina en SQL Server el **"Tail log"**, o sea la parte final del log que contiene el histórico de transacciones desde la última copia de seguridad que se hizo (o desde la última copia de log si ha habido otras intermedias).

2º. Una vez tenemos la parte final del log salvada, ya podemos hacer el restore de la base datos, osea la recuperación de la copia de la noche anterior con esta sentencia:

```
RESTORE DATABASE CONTABILIDAD
FROM DISK = 'D:\backup\contabilidad.bak'
WITH NORECOVERY;
```

Fijémonos que **esto permitirá recuperar la base de datos hasta las 2 de la mañana** que es cuando se hizo la última copia de seguridad. Fijémonos en algo aún más importante el parámetro **NORECOVERY**. Este parámetro le dice a SQL Server que restaure los ficheros desde la copia de seguridad (el de datos y el log) pero **que no haga la recuperación**, o sea que los deje tal y como están. De esta forma, le estamos diciendo a SQL Server que deje intacto el log, porque vamos a seguir restaurando parte de log, para volver hasta el momento en el que se cayó el sistema.

3º. Finalmente nos faltará **restaurar la parte final del log (Tail-log)** que es la que contiene el histórico de transacciones desde las 2:00 de la mañana hasta las 19:00 horas que es cuando cayó el sistema. Usaremos una sentencia parecida a esta:

```
RESTORE LOG CONTABILIDAD
FROM DISK = 'H:\backup\final.bak'
WITH RECOVERY;
```

Fijémonos que en esta sentencia, **estamos recuperando el log y no el fichero de datos** y además al final le indicamos con el parámetro **RECOVERY**, que precisamente haga la recuperación. Es decir **que vuelva a rehacer todas las transacciones que tiene en el log** (desde las 2:00 de la mañana hasta las 19:00 de la tarde) y que deje la base de datos online.

De esta forma, el problema, se vuelve un problema menos grave, nadie pierde nada y volvemos a tener la base de datos online sin perder información.