Escuela de Sistemas y Tecnologías

Transparencias de **ANALISTA DE SISTEMAS**

Edición 2017 – Materia: **Diseño de Aplicaciones Web**

TEMA: Base de Datos Avanzado

Plantel y Contactos

- **Bedelía:**
  - Mail: bedeliasistemas@bios.edu.uy

- **Encargado de Sucursal:**
  - Pablo Castaño
  - Mail: pablocasta@bios.edu.uy

Recursos

- **Recursos Imprescindibles:**
  - Sitio Web de material
    (comunicarse con Bedelía por usuario/contraseña).
  - Transparencias del Curso.
  - Contar con el software necesario

Agenda

- Transacciones a Nivel de Lógica
- Concurrencia de Datos
- Usuarios – Roles - Permisos
- Índices
- Vistas

Transacciones a Nivel Lógica

SqlTransaction (1)

- Representa una transacción de Transact-SQL que se realiza en una base de datos de SQL Server a nivel de código.
- Debe pre existir una conexión activa al servidor Sql Server.
- Se crea un objeto **SqlTransaction** mediante una llamada a la operación **BeginTransaction** del objeto **SqlConnection**.

SqlTransaction (2)

- Aquellas operaciones que se deseen realizar dentro de la misma conexión, deberán utilizar la misma transacción: serán asignadas al objeto **SqlTransaction** que generó dicha conexión.

- Para ello se utiliza la propiedad **Transaction** del objeto **SqlCommand** que se ejecutará.

- Después de comenzar una transacción a través de dicha conexión, no se podrán realizar tareas por fuera de dicha transacción.

SqlTransaction (3)

- *Operaciones*:
  - **Commit()** confirma la transacción en la base de datos
  - **RollBack()** deshace la transacción desde un estado pendiente

Concurrencia de Datos
(Ado .Net)

Introducción

- Cuando varios usuarios intentan modificar datos al mismo tiempo, es necesario establecer controles para impedir que las modificaciones de un usuario influyan negativamente en las de otros.
- El sistema mediante el cual se controla lo que sucede en esta situación se denomina control de concurrencia.
- Existen 3 formas de control de concurrencia:
  - **Pesimista**
  - **Optimista**
  - **El último gana**

Concurrencia: Pesimista (1)

- Una fila no está disponible para los usuarios desde el momento en que se obtiene el registro hasta que se actualiza en la base de datos.
- Normalmente, la concurrencia pesimista se utiliza por dos razones:
  - A veces existe una gran contienda por los mismos registros.
  - El costo de bloquear los datos es menor que el de deshacer los cambios cuando se producen conflictos de concurrencia.
- Es útil también en situaciones en las que no es conveniente que cambie el registro durante una transacción.

Concurrencia: Pesimista (2)

- No obstante, el control de concurrencia pesimista no es posible en una arquitectura desconectada.
- Las conexiones se abren sólo el tiempo suficiente para leer los datos o actualizarlos, por lo que los bloqueos no se pueden mantener durante períodos de tiempo prolongados.
- Además, una aplicación que mantiene los bloqueos durante períodos de tiempo prolongados no es escalable.

Concurrencia: Optimista (1)

- Una fila no está disponible para otros usuarios mientras los datos se estén actualizando.
- La actualización examina la fila de la base de datos y determina si se han realizado cambios.
- Si se intenta actualizar un registro que ya se ha modificado, se produce una infracción de concurrencia.
- Los bloqueos se establecen y mantienen sólo mientras se está teniendo acceso a la base de datos.
- Los bloqueos impiden que otros usuarios intenten actualizar registros en ese mismo instante.

Concurrencia: Optimista (2)

- Los datos están siempre disponibles, excepto durante el momento preciso en que está teniendo lugar una actualización.
- Cuando se intenta realizar una actualización, se compara la versión original de una fila modificada con la fila existente en la base de datos.
- Si las dos son diferentes, la actualización no se realiza debido a un error de concurrencia. En ese instante, es de su responsabilidad la reconciliación de las dos filas mediante su propia lógica.

Concurrencia: El Ultimo Gana

- Una fila no está disponible para otros usuarios mientras los datos se estén actualizando.
- Sin embargo, no se intenta comparar las actualizaciones con el registro original; simplemente el registro se escribe con la posibilidad de sobrescribir los cambios realizados por otros usuarios desde la última vez que se actualizaron los registros.
- Con la técnica de "el último gana“ no se realiza ninguna comprobación de los datos originales y la actualización simplemente se escribe en la base de datos.

Usuarios

Roles

Permisos

Introducción

- Para manejar la seguridad, Sql Server nos permite determinar que ***Usuario*** podrá acceder y/o modificar cual ***objeto*** de una o varias bases de datos.
- Para ello se basa en la definición de:
  - ***Usuarios***: entidad de seguridad.
  - ***Roles***: conjunto de permisos aplicables a uno o mas objetos del servidor. Son aplicados a los usuarios
  - ***Permisos***: determinan que acciones so podrán realizar sobre el servidor, o un objeto de la base de datos (por ejemplo: permiso de ejecución sobre un procedimiento almacenado)

Permisos (1)

- El Motor de base de datos del Sql Server administra un conjunto jerárquico de entidades que se pueden proteger mediante permisos.
- Esto elementos protegibles más importantes son los *Servidores* y las *Bases de Datos*. Pero También se pueden determinar permisos a nivel del ***Shema***.
- SQL Server regula las acciones en los elementos protegibles comprobando que se les han concedido los permisos adecuados para la acción que se desea realizar.

Permisos (2)

- Los permisos se pueden manipular mediante las siguientes sentencias de Transact-SQL :
  - ***GRANT***: Concede permisos sobre un elemento protegible, a una entidad de seguridad especifica.

    GRANT *permisos* [ON *objeto*] TO *usuarios*

  - ***DENY***: Quita un permiso a una entidad de seguridad.

    DENY *permisos* [ON objeto] TO usuarios

  - ***REVOKE***: Quita un permiso concedido o denegado previamente.

REVOKE permisos [ON objeto] TO | FROM usuario

Permisos (3)

- La información de permisos está disponible mediante vistas de catálogo. Las siguientes vistas, devuelven listas de permisos explícitos concedidos o denegados. Se encuentran incluidas dentro de todas las bases de datos y se pueden consultar mediante la sentencia ***Select***.
  - ***sys.database_permissions***: devuelve una fila por cada permiso en la base de datos. En las columnas, hay una fila por cada permiso que sea diferente del objeto correspondiente
  - ***sys.server_permissions***: devuelve una fila por cada permiso de nivel de servidor.

Roles: Nivel Servidor (1)

- Se denominan roles fijos de servidor porque no se pueden crear nuevos roles para este nivel. Estos se aplican a todo el servidor en lo que respecta a su ámbito de permisos.
- Se pueden agregar usuarios de inicios de sesión a los roles de nivel de servidor.
- Roles Principales :
  - ***sysadmin***: permite realizar cualquier actividad en el servidor.
  - ***dbcreator***: permite crear, modificar, quitar y restaurar cualquier base de datos.

Roles: Nivel Servidor (2)

- ***securityadmin***: permite administrar los inicios de sesión y sus propiedades; administrar los permisos de servidor GRANT, DENY y REVOKE; restablecer las contraseñas para los inicios de sesión de SQL Server.El rol ***securityadmin*** se debe tratar como equivalente del rol ***sysadmin***.

- ***public***: todo usuario de inicio de sesión pertenece a este rol. Cuando a un usuario no se le han concedido ni denegado permisos específicos para un objeto protegible, el usuario hereda los permisos concedidos al rol ***public*** en dicho objeto.

Roles: Nivel Servidor (3)

- Los siguientes procedimientos almacenados permiten manipular roles para usuarios de inicio de sesión:
  - ***sp_addsrvrolemember***: permite asignar roles a un usuario de logueo

```
EXEC sp_addsrvrolemember  [@loginame= ] 'login‘ , [@rolename = ] 'role'
```

  - ***sp_dropsrvrolemember***: permite eliminar un rol asignado a un usuario de logueo

```
EXEC sp_dropsrvrolemember  [@loginame= ] 'login‘ , [@rolename = ] 'role'
```

Roles: Nivel Base de Datos (1)

- Los roles de nivel de base de datos se aplican a toda la base de datos en lo que respecta a su ámbito de permisos.
- Existen dos tipos: los roles fijos (están predefinidos en la base de datos), y los roles flexibles (pueden crearse).
- Los roles fijos se definen en el nivel de base de datos y existen en cada una de ellas. Sólo los miembros del rol ***db_owner*** pueden agregar miembros al rol fijo *db_owner*. Los miembro de un rol fijo de base de datos puede agregar otros inicios de sesión a ese mismo rol.

Roles: Nivel Base de Datos (2)

- Roles Principales:
  - ***db_owner***: permite realizar todas las acciones de configuración, mantenimiento y eliminacion de la base de datos
  - ***db_securityadmin***: permite modificar la pertenencia a roles y administrar permisos.
  - ***db_ddladmin***: permite ejecutar cualquier comando del lenguaje de definición de datos (DDL) en la base de datos.
  - ***db_datawriter***: permite agregar, eliminar o cambiar datos en las tablas.
  - ***db_datareader***: permite leer los datos de tablas.

Roles: Nivel Base de Datos (3)

- Los siguientes procedimientos almacenados permiten asignar roles de la base de datos actual a usuarios:
  - ***sp_addrolemember***: permite asignar roles a un usuario de la base de datos

```
EXEC sp_addrolemember [@ rolename = ] "role‘ , [@ membername = ] ‘usuario'
```

  - ***sp_droprolemember*** : permite eliminar un rol asignado a un usuario de la base de datos

```
EXEC sp_droprolemember [@ rolename = ] "role‘ , [@ membername = ] ‘usuario'
```

Usuarios (1)

- Los usuarios son las entidades en las cuales se basa el Sql Server para:
  - Loguearse al servidor
  - Dar o denegar acceso a objetos
  - Permitir o no la ejecución de acciones
- Existen dos niveles de usuarios
  - ***Inicio de Sesión***: son creados con permisos para poder autenticar y autorizar el ingreso a SqlServer. Cuando se instala el servidor, ya se crean usuarios por defecto.
  - ***Base de Datos***: son creados a nivel de una base de datos especifica, sobre la cual tendrán permisos específicos.

Usuarios (2)

- Usuarios creados por defecto en la instalación:
  - ***sa***: usuario de logueo. Entidad de seguridad del servidor que por defecto se asigna a la base de datos ***master***.
  - ***dbo***: por defecto se crea en todas las bases de datos. Es utilizado por defecto por el rol ***Sysadmin*** y el usuario de logueo ***sa***.
  - ***guest***: por defecto existe en todas las bases de datos, y se encuentra deshabilitado. Permite logueos sin cuenta de usuario.

Usuarios (3)

- Mediante la sentencia ***Create Login*** se pueden agregar nuevos usuarios para el inicio de sesión.

```
CREATE LOGIN [NombreUsuarioWin] FROM WINDOWS
                [WITH DEFAULT_DATABASE = NombreBd ]

CREATE LOGIN [NombreUsuarioWin] WITH PASSWORD = 'pass'
                [WITH DEFAULT_DATABASE = NombreBd ]
```

- Mediante la sentencia ***Drop Login*** se pueden quitar usuarios de logueo

```
DROP LOGIN [NombreUsuarioDeLogueo]
```

Usuarios (4)

- Mediante la sentencia ***Create User*** se pueden agregar nuevos usuarios a la base de datos actual.

```
CREATE USER [NombreUsuario]
        FROM LOGIN [NombreUsuarioLogueo]
        [WITH DEFAULT_SCHEMA = nombre ]
```

- Mediante la sentencia ***Drop User*** se pueden quitar usuarios de la base de datos actual

```
DROP USER [NombreUsuario]
```

Índices

Introducción

- Un índice es una estructura de disco asociada con una tabla o vista que acelera la recuperación de filas.
- Un índice contiene claves generadas a partir de una o varias columnas de la tabla o vista. Dichas claves están almacenadas en una estructura (árbol) que permite que SQL Server busque de forma rápida y eficiente la fila o filas asociadas a los valores de cada clave.
- Los índices se crean automáticamente cuando las restricciones PRIMARY KEY y UNIQUE se definen en las columnas de tabla.

Tipos (1)

- *Agrupado (Clustered)*: ordena y almacena las filas de datos de la tabla o vista por orden en función de la clave del índice. El índice agrupado se implementa como una estructura de árbol b que admite la recuperación rápida de las filas a partir de los valores de las claves del índice agrupado. Las columnas están incluidas en la definición del índice. Sólo puede haber un índice clúster por cada tabla, porque las filas de datos sólo pueden estar ordenadas de una forma. Por lo tanto, la única ocasión en la que las filas de datos de una tabla están ordenadas, es cuando contiene un índice clúster. Si una tabla no tiene un índice clúster, sus filas de datos están almacenadas en una estructura sin ordenar denominada montón.

Tipos (2)

- ***No agrupado (Non Clustered)***: tienen una estructura separada de las filas de datos. Contiene los valores de clave y cada entrada de valor de clave tiene un puntero a la fila de datos que contiene el valor clave (localizador de fila). La estructura del localizador de filas depende de si las páginas de datos están almacenadas en un montón (puntero hacia la fila) o en una tabla agrupada (clave de índice agrupada).
- ***Único***: garantiza que la clave de índice no contenga valores duplicados y, por tanto, cada fila de la tabla o vista es en cierta forma única. Tanto los índices clúster como los no clúster pueden ser únicos.

Sentencias (1)

- Definir una restricción ***Primary Key*** o ***Unique*** a nivel de creación de tabla, implica la creación automática de un índice por dicho campo.
- La sentencia ***Create Index*** permite crear indices sobre una tabla o vista ya existentes

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX nombre
    ON objeto (NombreCampo [ ASC | DESC ] ……. )
```

- Los índices se pueden deshabilitar manualmente en cualquier momento mediante la instrucción: ALTER INDEX DISABLE.

```
ALTER INDEX nombre | ALL ON objeto DISABLE
```

Sentencias (2)

- Cuando ya no se necesita de un índice, se puede quitar de la base de datos y recuperar el espacio en disco que utilizaba. No puede quitar un índice utilizado por una restricción ***Primary Key*** o ***Unique***, excepto si quita la restricción. Se puede usar la siguiente sentencia para eliminar a un índice:

```
DROP   INDEX  NombreIndice ON objeto
```

Vistas

Introducción (1)

- Una vista es una tabla virtual cuyo contenido está definido por una consulta.
- Consta de un conjunto de campos y filas, con un nombre único en la base de datos. Sin embargo, a menos que esté indizada, una vista no existe como conjunto de valores de datos almacenados en la base de datos. Sus campos y filas proceden de tablas a las que se hace referencia en la consulta que define la vista y se producen de forma dinámica cuando se hace referencia a la vista.
- La consulta que define la vista puede usar una o varias tablas, otras vistas de la base de datos actual u otras bases de datos.

Introducción (2)

- No existe ninguna restricción a la hora de consultar vistas y muy pocas restricciones a la hora de modificar los datos de éstas.
- A tomar en cuenta en el diseño de una vista:
  - Los nombres de las vistas deben seguir las reglas que se aplican a las tablas y ser únicos para cada esquema: su nombre debe ser distinto del de las tablas incluidas en ese esquema.
  - SQLServer permite anidar vistas.
  - No se pueden asociar a las vistas, ni reglas ni definiciones DEFAULT.
  - No se pueden crear vistas temporales, ni vistas dentro de tablas temporales.

Introducción (3)

- La consulta que define la vista no puede incluir la cláusula ORDER BY, a menos que también haya una cláusula TOP en la lista de selección de la instrucción SELECT.
- Se deben especificar el nombre de todas los campos de la vista en el caso de que:
- alguna de las columnas de la vista derive de una expresión aritmética, una función integrada o una constante.
- dos o más campos de la vista tuviesen el mismo nombre (dos o más tablas diferentes tienen el mismo nombre de campo).

Introducción (4)

- De forma predeterminada, a medida que se agregan o se actualizan filas mediante una vista, éstas desaparecen del ámbito de la vista cuando dejan de cumplir los criterios de la consulta que define la vista.

Sentencias

- La sentencia ***Create View*** permite crear vistas sobre una o varias tablas o vistas ya existentes.

```
CREATE  VIEW    nombre   [NombreCampo, ……..]   AS
SentenciaConsultaSelect
```

- La sentencia ***Alter View*** permite modifica una vista creada anteriormente.

```
ALTER  VIEW    nombre   [NombreCampo, ……..]   AS
SentenciaConsultaSelect
```

- La sentencia ***Drop View*** quita una o más vistas de la base de datos actual.

```
DROP   VIEW    NombreVista, ……..
```

Modificar datos (1)

- Se pueden modificar los datos de una tabla base subyacente a través de una vista, de la misma forma que se modifican los datos de una tabla mediante las instrucciones ***Insert***, ***Update*** y ***Delete***.
- Restricciones que se aplican a la actualización de vistas, pero no a las tablas:
  - Cualquier modificación, incluidas las instrucciones ***Insert***, ***Update*** y ***Delete***, debe hacer referencia a las columnas de una única tabla base.
  - Los campos que se vayan a modificar en la vista deben hacer referencia directa a los datos subyacentes de las columnas de la tabla. No se pueden obtener de: una función agregada o un cálculo

Modificar datos (1)

- Los campos que se modifican no pueden verse afectadas por cláusulas ***Group By***, ***Having*** ni ***Distinct***.
- Las instrucciones ***Insert*** deben especificar valores para las columnas de la tabla subyacente que no permiten valores ***Null*** y no tienen definiciones ***Default***.
- Si se elimina una fila, todas las restricciones ***Foreign Key*** de las tablas relacionadas deben cumplirse para que pueda llevarse a cabo la eliminación.