Unidad 1 y Unidad 2

Fecha: 07/04/2016

Parte 1: Respaldo de BD

1) Definir una estrategia de respaldo y recuperación para las bases de datos relacionales de los siguientes casos de estudio:

Caso 1:

Un supermercado mayorista está dotado de un sistema Cliente/Servidor de dos capas. Cada caja registradora tiene una aplicación WinForm que accede a una base de datos relacional central. La aplicación se encarga del registro y emisión de los ticket fiscales. Tiene más de 20 cajas registradoras que operan durante el horario comercial. Es muy importante mantener estable el desempeño del sistema durante el horario de trabajo porque el cúmulo de transacciones diarias es muy grande.
La base de datos tiene un tamaño importante (750 GB), con una tasa de crecimiento mensual de 30 GB. Por otro lado, es fundamental garantizar la consistencia, integridad y respaldo de los datos en todo momento, por lo que ante una falla del sistema, el mismo debe recuperarse lo más rápido posible, siendo admisible solamente una pérdida de información o de transacciones de hasta 30 minutos atrás.

Caso 2:

Una empresa que da servicio de CallCenter posee un sistema CRM (Administración de Clientes) que tiene una arquitectura Cliente/Servidor de tres capas, dotado de un servidor web, y otro de base de datos.
El sistema debe funcionar las 24 horas, los 365 días del año. Mas del 90% de las operaciones son solamente de lectura sobre los datos. La base de datos tiene un tamaño del órden de los 45 GB, con una tasa de crecimiento mensual de 1.2 GB.
Según las estadísticas que mensualmente se extraen del servidor web, el pico de utilización del sistema es en los horarios de 10:00 a 12:00 hs., y el de menor utilización es de 1:00 a 4:00 hs.

2) Investigar las posibilidades que posee SQL Server y MySQL en materia de Backup y Restore, e indicar cómo se implementarían las estrategias definidas anteriormente para ambos casos.

Parte 2: Integridad de BD

Caso 3:

Un supermercado tiene una aplicación Cliente/Servidor encargada de la registración la impresión de los tickets de las ventas que se realizan en todas las cajas.
La arquitectura consta de las siguientes partes:

- Servidor de base de datos: Posee la base de datos corporativas en donde se realizan todas las registraciones y consultas de las ventas realizadas.
- Cliente de la caja registradora: realiza la consulta del precio de cada producto, arma el ticket y lo imprime.
- Cliente administrativo: permite la administración de los datos del producto. Tiene formularios para crear productos, modificar su descripción, código de barra, precio unitario, stock, etc.

1) ¿Cómo podría el DBA asegurarse que en los datos no existan nunca las siguientes inconsistencias?:

a. Productos finales con el mismo código.
b. Detalles de tickes con cabeceras que no existen.
c. Productos con el dato de precio en blanco o nulo.
d. Valores alfabéticos en el campo que representa la cantidad comprada de un producto.
e. El borrado de clientes que tengan tickes a su nombre.
f. La actualización del nombre de un proveedor que provea más de 10 productos.
g. Fecha de emisión de un ticket anteriores al año 2010.

2) Investigar las posibilidades que posee SQL Server y MySQL para implementar los diferentes tipos de controles de integridad de forma automatizada.

Parte 3: Concurrencia

1) Sobre el enunciado del ejercicio anterior, se plantea simular el funcionamiento concurrente de los dos tipos de cliente contra la misma base de datos corporativa. Se requieren las siguientes condiciones:

- Tener una base de datos SQL Server con una tabla "Producto" en donde el "PrecioUnitario" sea uno de sus atributos.
- Tener cargados un par de productos en la tabla.
- Tener dos conexiones abiertas desde el ManagementStudio a la base de datos anterior. En una simularíamos las sentencias SQL que llegarían a la base de dato desde el cliente que

corre en cada caja registradora, y en la otra simularíamos las sentencias que le llegan desde el cliente administrativo.
- Tener una conexión abierta adicional que es usada por el DBA para chequear el estado de la base de datos.

Situación de Concurrencia 1:

SQL Cliente Administrativo: consulta de todos los datos de un artículo para llenar el formulario que ve el usuario por pantalla:

```
SELECT * FROM Producto WHERE Id_Producto = 5
```

SQL Caja Registradora: consulta de precio de un artículo

```
SELECT PrecioUnitario FROM Producto WHERE Id_Producto = 5
```

¿Se produjo algún inconveniente en alguno de los clientes?

Situación de Concurrencia 2:

SQL Cliente Administrativo: modificación del precio del artículo. Aún el cambio no fue confirmado, ya que falta que el usuario precione el botón "Confirmar" en un cuadro de diálogo emergente. El usuario se levantó de su estación de trabajo y quedó el diálogo pendiente en la pantalla.

```
BEGIN TRANSACTION
    UPDATE Producto SET PrecioUnitario = PrecioUnitario * 1.2 WHERE Id_Producto = 5
```

SQL Caja Registradora: consulta de precio de un artículo

```
SELECT PrecioUnitario FROM Producto WHERE Id_Producto = 5
```

¿Se produjo algún inconveniente en alguno de los clientes?

Ante el reclamo de que el sistema de todas las cajas registradoras del supermercado está trabado, el DBA ejecuta el comando para verificar los procesos corriendo en el servidor de base de datos:

```
EXEC sp_who2
```

El DBA revisa que no existan procesos que hace mucho tiempo que se estén ejecutando o que estén consumiendo mucha CPU, memoria o disco.
Si descarta problemas de performance en el servidor (pero el sistema sigue sin responder), revisa el campo "BlkBy", que significa "bloqueado por". Si el proceso está bloqueado, ese campo debe indicar qué SPID (Id de proceso) es el que lo está bloqueando.
Si el DBA detecta un bloqueo, ejecuta el comando para ver la tabla de bloqueo de cada proceso:

EXEC sp_lock <<SPID Proc.Bloqueado>>
EXEC sp_lock <<SPID Proc.Bloqueador>>

Teniendo en cuenta el mecanismo que usan los motores de base de datos para prevenir problemas de concurrencia, ¿cuál es el recurso en donde se está trabando el sistema?

Cuando se detecta el proceso iniciador del bloqueo, el DBA ejecuta el comando para cortar el proceso y forzar el rollbak.

KILL <<SPID Proc.Bloqueador>>

El sistema del supermercado vuelve a la normalidad. El usuario administrativo cuando vuelve a su estación de trabajo encuentra en mensaje de error del motor de base de datos (perdió todos los cambios que había realizado).

Situación de Concurrencia 3:

Como la situación anterior se repitió en varias oportunidades, se tomó la decisión de bajar el nivel de aislamiento de la sentencia de consulta de precios del cliente de las cajas registradora.

SELECT PrecioUnitario FROM Producto (NOLOCK) WHERE Id_Producto = 5

El problema del sistema bloqueado no se volvió a repetir. Pero en las auditorías se encontraron tickets que salieron con precios unitarios incorrectos.

Repetir la situación de concurrencia 2 teniendo en cuenta el cambio realizado. Con la premisa de que los usuarios administrativos el 10% de las veces que realiza un cambio de precios se da cuenta que cometió un error y en lugar de presionar el botón "Confirmar" (que ejecuta un COMMIT en la base de datos), presiona un "Cancelar" (que ejecuta un ROLLBACK).

¿Cómo se explicaría la situación detectada en la auditoría? ¿Cuál hubiese sido la decisión correcta para atacar el problema del bloqueo del sistema?

2) Teniendo en cuenta el siguiente conjunto de transacciones:

T1: Aumenta un 20% el precio unitario del producto 5.
T2: Actualiza el precio del producto 5 en $25.
T3: Duplica el precio del producto 5.

Teniendo en cuenta que las tres transacciones son concurrentes en un momento dado en la base de datos. ¿Cuántos son los resultados correctos posibles (sin realizar cálculos)? ¿Cuáles son los resultados correctos en los cuales puede quedar el precio del producto luego de su ejecución si inicialmente, antes de iniciarse estas transacciones, el precio era de 55$? Justifique su respuesta.