# Unidad 3

## Bases de datos relacionales – aspectos avanzados – parte 3

Bases de Datos Aplicada

v1.0 – Agosto 2023

Universidad Nacional de La Matanza

DIIT
Departamento de Ingeniería e Investigaciones Tecnológicas

# Contenido

- Control de concurrencia.
- Transaction Control Language (TCL).
- Bloqueos y esperas.
- Transacciones distribuidas
- Monitoreo
- Metricas
- Performance

# Al finalizar deberías ser capaz de…

- Determinar que tipo de nivel de aislamiento corresponde según la transacción y el objeto donde se desea aplicar.
- Poder aplicar performance sobre SQL

# Transaction Control Language (TCL)

- Conjunto de operaciones que se realizan como **una sola unidad lógica** de trabajo.
- Dicha unidad debe cumplir con cuatro propiedades básicas ACID para poder ser clasificada como una transacción


Fuente:gravitar.biz

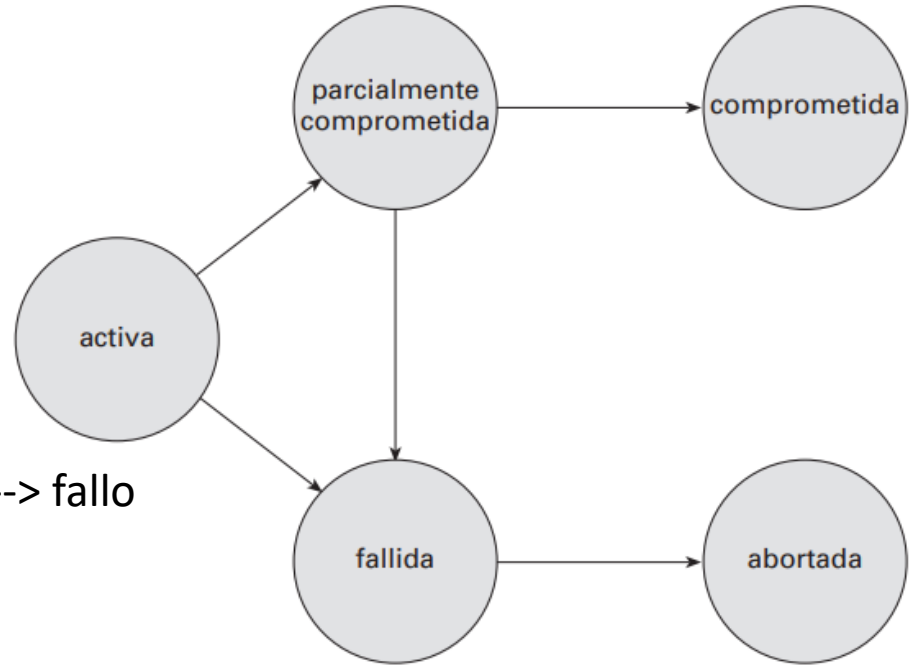
# Transaction Control Language (TCL)

- Ejemplo: Se tiene dos tablas
  - `creditCard_info` → contiene nro de tarjeta y el límite de crédito disponible
  - `Transaccion` → contiene el nro. de tarjeta, la fecha, el monto de la transacción y el tipo de transacción (Venta o Reversión)
- Por cada transacción de venta se decrementa el límite de crédito
- Por cada transacción de reversión se incrementa el límite de crédito

# Transaction Control Language (TCL)

- **Atomicidad**: para que una transacción se dé por «*completada*», deben haberse realizado todas sus partes o ninguna de ellas.





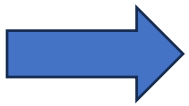

```
leer(Transaccion);
Transaccion := Transaccion + 50;
escribir(Transaccion);
leer(creditCard_info);
creditCard_info := creditCard_info-50; --> fallo
escribir(creditCard_info).
```

# Transaction Control Language (TCL)

- **Consistencia**: se basa en la premisa que afirma que una transacción debe llevar al sistema de un estado válido a otro que también lo sea.
- La responsabilidad de asegurar la consistencia de una transacción es del programador de la aplicación que codifica dicha transacción.



```
leer(Transaccion);
Transaccion := Transaccion + 50;
escribir(Transaccion);
leer(creditCard_info);
creditCard_info := creditCard_info-50;
escribir(creditCard_info)
```



La transacción se ve como **un todo**

# Transaction Control Language (TCL)

- **Aislamiento**: Asegura que el resultado obtenido al ejecutar concurrentemente las transacciones es un estado del sistema equivalente a uno obtenido al ejecutar una tras otra en algún orden.



Venta 1

```
leer(Transaccion);
Transaccion := Transaccion + 50;
escribir(Transaccion);
leer(creditCard_info);
creditCard_info := creditCard_info-50;
escribir(creditCard_info)
```

Venta 2

```
leer(Transaccion);
Transaccion := Transaccion + 50;
escribir(Transaccion);
leer(creditCard_info);
creditCard_info := creditCard_info-50;
escribir(creditCard_info)
```

# Transaction Control Language (TCL)

- **Durabilidad**: implica que los datos y cambios en una transacción que ya se ha realizado deben ser permanentes y no puede ocurrir una pérdida de los mismos en el sistema.
- Se puede garantizar la durabilidad si se asegura que:

1. Las modificaciones realizadas por la transacción se guardan en disco antes de que finalice la transacción.

2. La información de las modificaciones realizadas por la transacción guardada en disco es suficiente para permitir a la base de datos reconstruir dichas modificaciones cuando el sistema se reinicie después del fallo.

# Transaction Control Language (TCL)

En SQL Server

- Transacciones implícitas

Se inicia implícitamente una nueva transacción cuando se ha completado la anterior, pero **cada transacción se completa explícitamente** con una instrucción **`COMMIT`** o **`ROLLBACK`**.

Para activar Modo implícito: **`SET IMPLICIT_TRANSACTIONS ON`**

Para desactivar Modo implícito: **`SET IMPLICIT_TRANSACTIONS OFF`**

No es aconsejable este modo dado que puede olvidar de cerrar las transacciones, y esto puede ser origen de bloqueos.

# Transaction Control Language (TCL)

Para ver la configuración actual de **IMPLICIT_TRANSACTIONS**

```
DECLARE @IMPLICIT_TRANSACTIONS VARCHAR(3) = 'OFF';
IF ((2 & @@OPTIONS) = 2 )
   SET @IMPLICIT_TRANSACTIONS = 'ON';
SELECT @IMPLICIT_TRANSACTIONS AS ModoImplicitTranActual;
```

Ejecutar IMPLICIT_TRANSACTIONS.sql y analizar los resultados

# Transaction Control Language (TCL)

- IMPLICIT_TRANSACTIONS
- Ejecutar IMPLICIT_TRANSACTIONS.sql

# Transaction Control Language (TCL)

En SQL Server

- Transacciones explícitas

Cada transacción **se inicia explícitamente** con la instrucción BEGIN TRANSACTION y **se termina explícitamente** con una instrucción COMMIT o ROLLBACK.

Ejemplo de transacción
TCL_0.sql

Ejemplo 1 - READ UNCOMMITTED
TCL_1.sql y TCL_2.sql

Ejemplo 2- READ COMMITTED
TCL_3.sql y TCL_4.sql

Ejemplo 3- REPEATABLE READ
TCL_5.sql - TCL_6.sql - TCL_7.sql

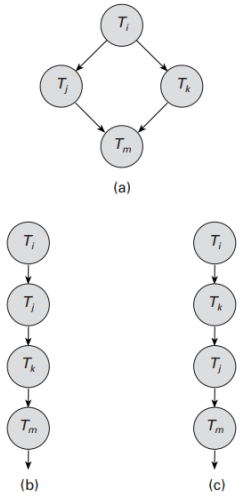
Ejemplo 4- SERIALIZABLE
TCL_8.sql y TCL_9.sql

# Control de concurrencia

Trata con los problemas de aislamiento y consistencia del procesamiento de transacciones

- Los esquemas de control de concurrencia que se describen a continuación se basan en la propiedad de secuencialidad.

- Todos los esquemas de ejecución que se presentan aseguran que las planificaciones son secuenciales si es equivalente a una planificación secuencial



# Control de concurrencia

**Bloqueos**

- Compartido. Si una transacción Ti obtiene un bloqueo en modo compartido (denotado por C) sobre el elemento Q, entonces Ti puede leer Q pero no lo puede escribir.

- Exclusivo. Si una transacción Ti obtiene un bloqueo en modo exclusivo (denotado por X) sobre el elemento Q, entonces Ti puede tanto leer como escribir Q.

# Control de concurrencia

Los niveles de aislamiento definen el grado en que una transacción debe aislarse de las modificaciones de datos realizadas por cualquier otra transacción en el sistema de base de datos.
Cuando se accede a un mismo dato en dos transacciones distintas se pueden dar las siguientes situaciones:

- Lectura sucia (Dirty Read). Una transacción lee datos que han sido escritos por otra transacción que aún no se ha confirmado.

- Lectura no repetible (Non-repeateable Read). Una transacción vuelve a leer los datos que ha leído anteriormente y descubre que otra transacción confirmada ha modificado o eliminado los datos.

- Lectura fantasma (Phantom Read). Una transacción vuelve a ejecutar una consulta que devuelve un conjunto de filas que satisface una condición de búsqueda y descubre que otra transacción confirmada ha insertado filas adicionales que satisfacen la condición.

# Control de concurrencia

Para , el estándar SQL define cuatro niveles de aislamiento

| Nivel | Dirty Read (Lectura sucia) | Non-Repeatable Read (Lectura No Repetible) | Phantom Read (Lectura fantasma) |
|---|---|---|---|
| **Read Uncommitted** | Es posible | Es posible | Es posible |
| **Read Committed** | - | Es posible | Es posible |
| **Repeatable Read** | - | - | Es posible |
| **Serializable** | - | - | - |

# Control de concurrencia

SET TRANSACTION ISOLATION LEVEL

Controla el comportamiento del bloqueo y de las versiones de fila de las instrucciones Transact-SQL emitidas por una conexión a SQL Server

Tener en cuenta que la elección de un nivel de aislamiento de transacción no afecta a los bloqueos adquiridos para proteger la modificación de datos. Siempre se obtiene un bloqueo exclusivo en los datos modificados de una transacción, bloqueo que se mantiene hasta que se completa la transacción, independientemente del nivel de aislamiento seleccionado para la misma

# Control de concurrencia

READ UNCOMMITTED
Especifica que las instrucciones pueden leer filas que han sido modificadas por otras transacciones, pero todavía no se han confirmado.  Se trata del nivel de aislamiento menos restrictivo

READ COMMITTED
Especifica que las instrucciones no pueden leer datos que hayan sido modificados, pero no confirmados, por otras transacciones.  Esta opción es la predeterminada para SQL Server

# Control de concurrencia

**READ UNCOMMITTED**
Especifica que las instrucciones pueden leer filas que han sido modificadas por otras transacciones, pero todavía no se han confirmado.  Se trata del nivel de aislamiento menos restrictivo

**READ COMMITTED**
Especifica que las instrucciones no pueden leer datos que hayan sido modificados, pero no confirmados, por otras transacciones.  Esta opción es la predeterminada para SQL Server

# Control de concurrencia

El comportamiento de READ COMMITTED depende del valor de la opción de base de datos READ_COMMITTED_SNAPSHOT:

- Si se establece en OFF (el valor predeterminado de SQL Server), el motor de base de datos usa bloqueos compartidos para impedir que otras transacciones modifiquen las filas mientras la transacción actual esté ejecutando una operación de lectura.

- Si se establece en ON (el valor predeterminado de Azure SQL Database), el motor de base de datos usa versiones de fila para presentar a cada instrucción una instantánea coherente, desde el punto de vista transaccional, de los datos tal como se encontraban al comenzar la instrucción. No se emplean bloqueos para impedir que otras transacciones actualicen los datos.

# Control de concurrencia

**REPEATABLE READ**

Especifica que las instrucciones no pueden leer datos que han sido modificados, pero aún no confirmados por otras transacciones y que ninguna otra transacción puede modificar los datos leídos por la transacción actual hasta que ésta finalice.

**SERIALIZABLE**, especifica lo siguiente:

- Las instrucciones no pueden leer datos que hayan sido modificados, pero aún no confirmados, por otras transacciones.
- Ninguna otra transacción puede modificar los datos leídos por la transacción actual hasta que la transacción actual finalice.
- Otras transacciones no pueden insertar filas nuevas con valores de clave que pudieran estar incluidos en el intervalo de claves leído por las instrucciones de la transacción actual hasta que ésta finalice.

# Control de concurrencia

**SNAPSHOT**

Especifica que los datos leídos por cualquier instrucción de una transacción serán la versión coherente, desde el punto de vista transaccional, de los datos existentes al inicio de la transacción.
La transacción únicamente puede reconocer las modificaciones de datos confirmadas antes del comienzo de la misma. Las instrucciones que se ejecuten en la transacción actual no verán las modificaciones de datos efectuadas por otras transacciones después del inicio de la transacción actual.
El efecto es como si las instrucciones de una transacción obtienen una instantánea de los datos confirmados tal como se encontraban al inicio de la transacción

# Control de concurrencia

**Observaciones**

SET TRANSACTION ISOLATION LEVEL se aplica en tiempo de ejecución, no en tiempo de análisis

Solo es posible establecer una de las opciones de nivel de aislamiento cada vez, y permanecerá activa para la conexión hasta que se cambie explícitamente. Todas las operaciones de lectura realizadas dentro de la transacción se rigen por las reglas del nivel de aislamiento especificado, a menos que se utilice una sugerencia de tabla en la cláusula FROM de una instrucción para especificar un comportamiento de bloqueo o versiones diferente para una tabla.

# Control de concurrencia

Cuando se cambia el nivel de aislamiento de una transacción por otro, los recursos leídos después del cambio se protegen de acuerdo con las reglas del nuevo nivel. Los recursos leídos antes del cambio siguen estando protegidos en función de las reglas del nivel anterior. Por ejemplo, si una transacción ha cambiado de READ COMMITTED a SERIALIZABLE, los bloqueos compartidos adquiridos después del cambio se mantienen hasta el final de la transacción.

Si se ejecuta SET TRANSACTION ISOLATION LEVEL en un procedimiento almacenado o un desencadenador, cuando el objeto devuelve el control, el nivel de aislamiento se restablece en el nivel en efecto cuando se invocó el objeto. Por ejemplo, si se establece REPEATABLE READ en un lote y, después, este lote llama a un procedimiento almacenado que establece el nivel de aislamiento en SERIALIZABLE, el valor del nivel de aislamiento vuelve a REPEATABLE READ cuando el procedimiento almacenado devuelve el control al lote

# Control de concurrencia

```
- - Syntax for SQL Server and Azure SQL Database
SET TRANSACTION ISOLATION LEVEL
      { READ UNCOMMITTED
      | READ COMMITTED
      | REPEATABLE READ
      | SNAPSHOT
      | SERIALIZABLE }
```

```
USE AdventureWorks2022;
GO
 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
GO
BEGIN TRANSACTION;
GO
 SELECT * FROM HumanResources.EmployeePayHistory;
GO
SELECT * FROM HumanResources.Department;
GO
COMMIT TRANSACTION;
GO
```

# Transacciones distribuidas

Es una transacción que afecta a varios recursos.

- Para que se confirme **todos los participantes** deben **garantizar** que los cambios en los datos serán **permanentes**. Los cambios deben conservarse a pesar de bloqueos del sistema u otros eventos imprevistos.
- Si **alguno** de los participantes no cumple esta garantía, toda la transacción da error y **se revertirán** los cambios en los datos en el ámbito de la transacción.

# Transacciones distribuidas

```
BEGIN DISTRIBUTED { TRAN | TRANSACTION }
    [ transaction_name | @tran_name_variable ]
  [ ; ]
```

Argumentos

**transaction_name**

Nombre de transacción definida por el usuario que se utiliza para realizar el seguimiento de la transacción distribuida en las utilidades de MS DTC. `transaction_name` debe seguir las reglas de los identificadores y ser <= 32 caracteres.

**@tran_name_variable**

Nombre de una variable definida por el usuario que contiene el nombre de una transacción utilizada para realizar el seguimiento de la transacción distribuida en las utilidades de MS DTC. La variable debe declararse con un tipo de datos char, varchar, nchar o nvarchar.

# Transacciones distribuidas

La instancia del DBMS SQL Server que ejecuta la instrucción **`BEGIN DISTRIBUTED TRANSACTION`** es el **originador** de la transacción y **controla su realización**. Posteriormente, cuando en la sesión se ejecuta una instrucción **`COMMIT TRANSACTION`** o **`ROLLBACK TRANSACTION`**, la instancia que controla la transacción solicita a MS DTC que administre la realización de la transacción distribuida entre todas las instancias participantes.

El aislamiento de instantáneas de nivel de instantánea no admite transacciones distribuidas.

La principal manera en que las instancias remotas del Motor de base de datos se dan de alta en una transacción distribuida es cuando una sesión ya dada de alta en la transacción distribuida ejecuta una consulta distribuida que hace referencia a un servidor vinculado.

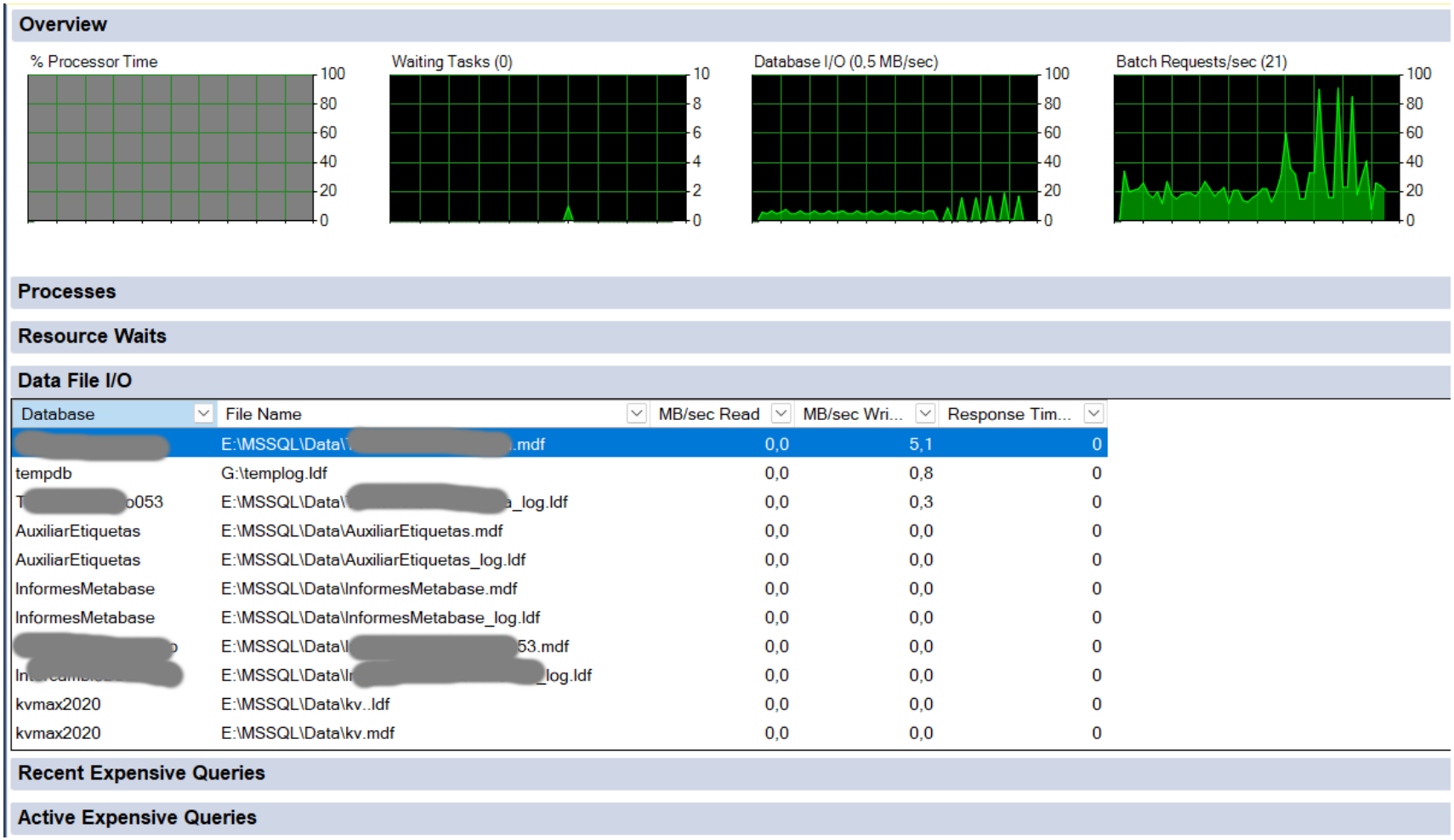
# Transacciones distribuidas

Este ejemplo elimina un candidato de la base de datos AdventureWorks2022 tanto en la instancia local del Motor de base de datos como en la instancia de un servidor remoto. **Ambas bases de datos**, local y remota, confirmarán o revertirán la transacción.

```
USE AdventureWorks2022;
GO
BEGIN DISTRIBUTED TRANSACTION;
 -- Borramos un registro local
DELETE AdventureWorks2022.HumanResources.JobCandidate
WHERE JobCandidateID = 13;
-- Borramos el correspondiente en un srv remoto
DELETE
RemoteServer.AdventureWorks2022.HumanResources.JobCand
idate
WHERE JobCandidateID = 13;
COMMIT TRANSACTION;
```

# Monitoreo

*El ECG de SQL*



# Métricas de performance

Nos ayudan a entender la **utilización** de la DB y el **consumo** de recursos.

Pueden incluir:

- Estadísticas de HW (consumo de CPU, memoria, lecturas, escrituras).
- Cuenta de filas.
- Esperas (*waits*).
- *Deadlocks*.

Son de gran utilidad para determinar si un servidor tiene **pocos recursos**, en el desarrollo, resolución de **problemas** y detección de bugs.

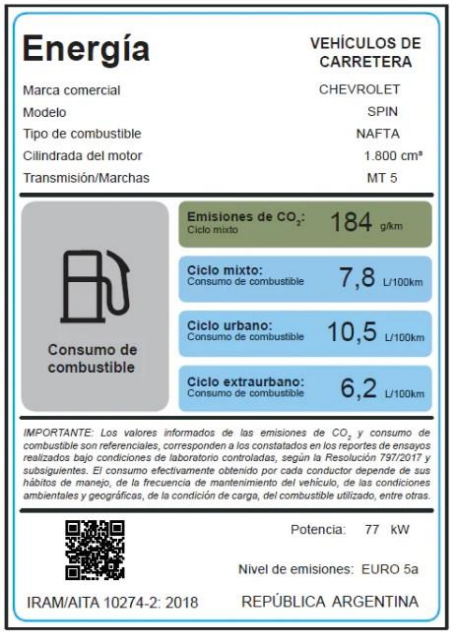
# Métricas de performance

Concepto importante: ***baseline***

Necesitamos una **referencia** para determinar el **comportamiento anómalo.**

Debemos conocer el sistema y su uso por hora/día/etc.

- ¿Cuántas transacciones se realizan?
- ¿Cuántas filas se agregan en X tabla?
- ¿Qué ritmo de crecimiento tiene la DB?
- ¿Cuál es la duración estándar de algunas operaciones críticas?



# Métricas de performance

Recuento de filas (**row counts**)

- Fácil de medir y efectiva para aplicaciones *data-driven*.
- Puede controlarse por valores absolutos o en términos de porcentajes, o ambos.
- Limitación: no provee información sobre actualizaciones o lecturas.

Supongamos que normalmente se registran 5 mil registros en una tabla de ventas en un sistema dado. Si un día solo se cargan 20 y otro día hay 200 mil, tal vez haya que investigar la causa. (Tal vez feriado o *hot sale*).

# Métricas de performance

Entrada/salida de archivos (**database file IO**)

- Permite conocer cuántos datos se leen y escriben en un archivo dado.
- Puede medirse en datos y log, incluso en bases con varias particiones.
- Limitación: Sirven para detectar un problema pero no para determinar una tabla u objeto particularmente afectado.
- Permiten predecir si el crecimiento de la DB será acompañado por los recursos de almacenamiento, red, backup a lo largo del tiempo.

# Métricas de performance

El **modelo de recuperación** es una propiedad de la DB que controla cómo se mantiene el log de transacciones.

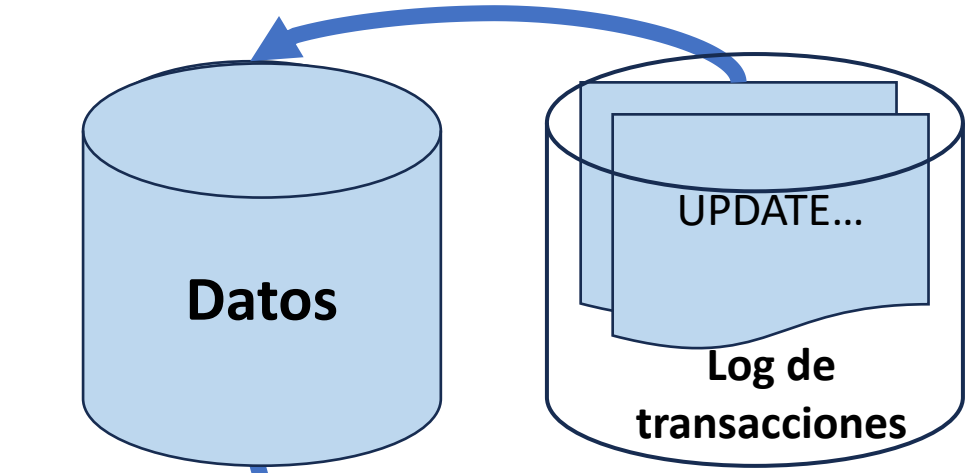
- Cuándo se registran
- Si se require que se respalden (backup)
- Qué tipos de operaciones de restauración están disponibles.

Son tres: Simple, full, and bulk-logged.

Puede cambiarse en cualquier momento.

https://learn.microsoft.com/en-us/sql/relational-databases/backup-restore/recovery-models-sql-server?view=sql-server-ver16

# Modelo de recuperación SIMPLE

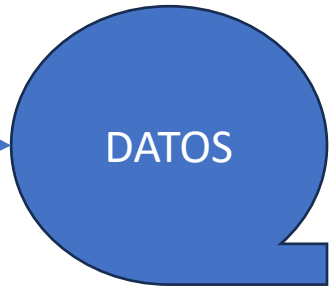



**Hay varios procesos que generan un checkpoint**

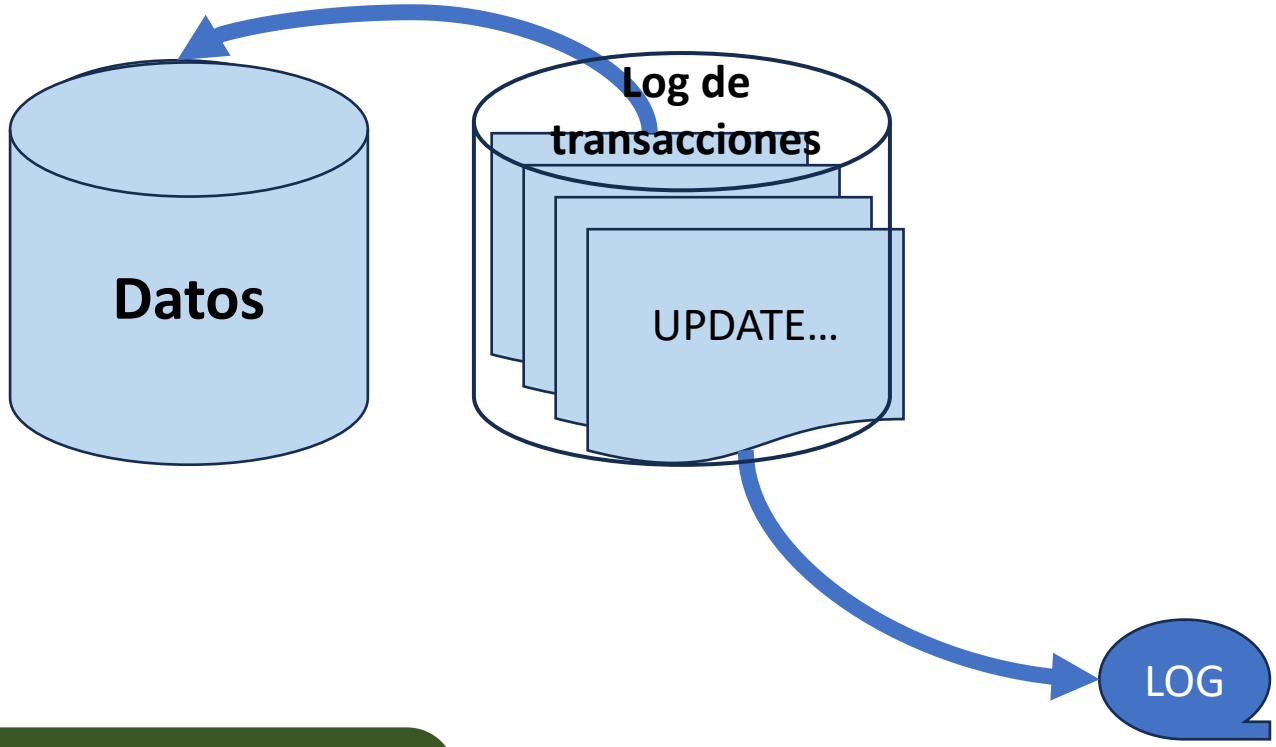
**CHECKPOINT**

**No hay backup del log de transacciones**

**Generar backup**



# Modelo de recuperación FULL




**Generar backup del log de transacciones**

*Veremos más sobre backups en la unidad 5*

# Métricas de performance

| Simple | Full |
|---|---|
| El log se libera al completarse las transacciones. | El log se libera cuando se respalda. Debe realizarse para que no crezca en forma desmedida. |
| No se respalda el log. | Requiere backups del log de transacciones. |
| Solo se puede recuperar a estados respaldados en un backup. | Puede recuperarse a un punto específico en el tiempo (por ejemplo antes de un error de una App). |
| No admite funcionalidades tales como AlwaysOn, Mirroring, log shipping, restauración a un punto en el tiempo. | |

https://www.mssqltips.com/sqlservertutorial/4/sql-server-simple-recovery-model/

# Métricas de performance

Tamaño del backup del log de transacciones (**transaction log backup size**)

- En una DB en modo de recuperación completo (*full*) o *bulk-logged* los cambios sobre ella se reflejarán en el tamaño del log.
- Si los backups del log de transacciones se toman cada hora y tienen un tamaño dado, un crecimiento anormal de ese tamaño indica una situación a investigar.
- Ventaja: puede determinarse sin acceder a la DB, sino solo a los backups.

# Métricas de performance

Bloqueos/esperas

- Una consulta que demora mucho por **lentitud** o por **esperar** que otros procesos liberen recursos pueden apuntar a una **App a depurar**.
- Las esperas pueden detectarse haciendo muestreo repetidamente.
- Las *wait stats* son un recurso de mucha utilidad cuando se requiere investigación adicional, porque pueden asociarse a SPID, texto de la consulta, plan de ejecución en XML, etc.

# Métricas de performance

**Obtención y análisis de las métricas**

- Ejecución manual de scripts.
- Trabajo de ejecución automática.
- Envío de email, *dashboard*, etc.
- Software especializado (*3rd party*).


Los valores fuera de lo <u>normal</u> pudieran indicar problemas a resolver.

Imagen: https://www.virtualmetric.com/microsoft-sql-server-performance-monitoring/

# ¿Dudas?

# Bibliografia

TCL
Silberschatz, A., Korth, F., Sudarshan, S.: Fundamentos de Base de Datos, cuarta edicion
capitulo 15-16-17
SET TRANSACTION ISOLATION LEVEL (Transact-SQL)
https://learn.microsoft.com/es-es/sql/t-sql/statements/set-transaction-isolation-level-transact-sql?view=sql-server-ver16
Transacciones Distribuidas
https://learn.microsoft.com/es-es/sql/t-sql/statements/set-transaction-isolation-level-transact-sql?view=sql-server-ver16