UNIVERSIDAD PERUANA DE LOS ANDES

FACULTAD DE INGENIERÍA

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

SEMANA 10

ESTUDIANTE: CARRILLO CÁRDENAS JOSÉ ESTEBAN

ASIGNATURA: BASE DE DATOS II

DOCENTE: RAUL FERNANDEZ, Bejarano

CICLO: V

HUANCAYO – 2025

Semana 10: Administración esencial

Objetivo: Dominar la administración diaria del servidor y la gestión de bases de datos

SEMANA 10

Temas:

1. Estructura de almacenamiento y archivos de datos

Desarrollo

La "estructura de almacenamiento y archivos de datos" se refiere a cómo se guardan físicamente los datos dentro de un sistema o base de datos —es decir, la forma en que la información se organiza, distribuye y administra en el disco duro o en otro medio de almacenamiento.

En administración de servidores o sistemas:

La **estructura de almacenamiento** describe **cómo el sistema organiza los archivos y directorios** (carpetas) donde guarda la información. Por ejemplo:

- En Linux o Windows, los archivos del sistema, los logs, las aplicaciones y los datos de usuarios se guardan en diferentes rutas o particiones.
- Saber cómo está organizada esta estructura ayuda a **hacer copias de seguridad, optimizar el rendimiento y evitar pérdida de datos**.

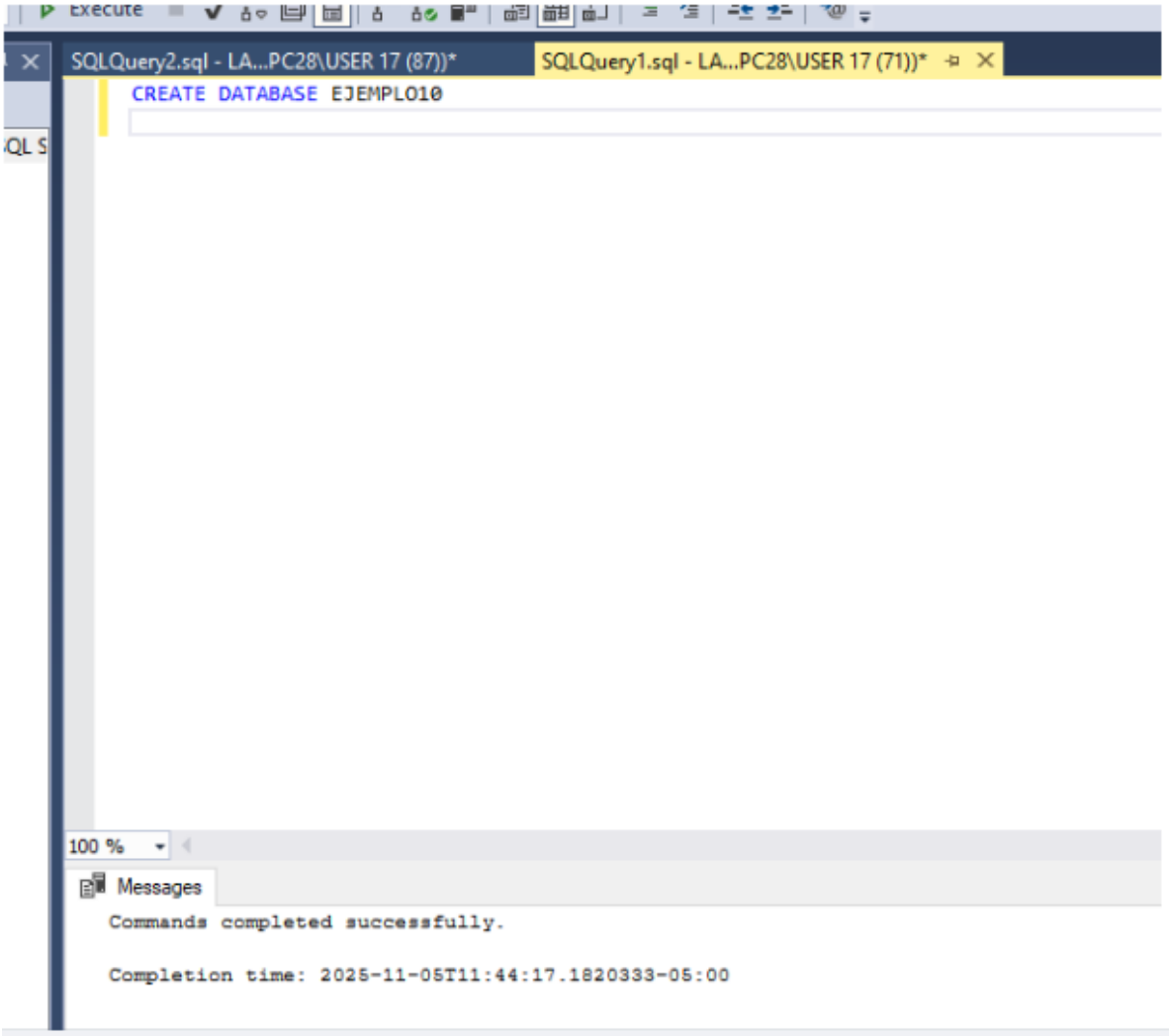
En bases de datos (como MySQL, SQL Server, PostgreSQL, Oracle, etc.):

La **estructura de almacenamiento y archivos de datos** se refiere a **los archivos físicos** donde el sistema de base de datos guarda la información. Por ejemplo:

- En **MySQL**, cada base de datos tiene archivos como:
  - .frm → define la estructura de las tablas.
  - .ibd o .myd → contienen los datos reales.
  - .ib_logfile → guarda los registros de transacciones.
- En **SQL Server**, hay:
  - Archivos .mdf (datos principales),
  - .ndf (archivos secundarios),
  - .ldf (logs de transacciones).

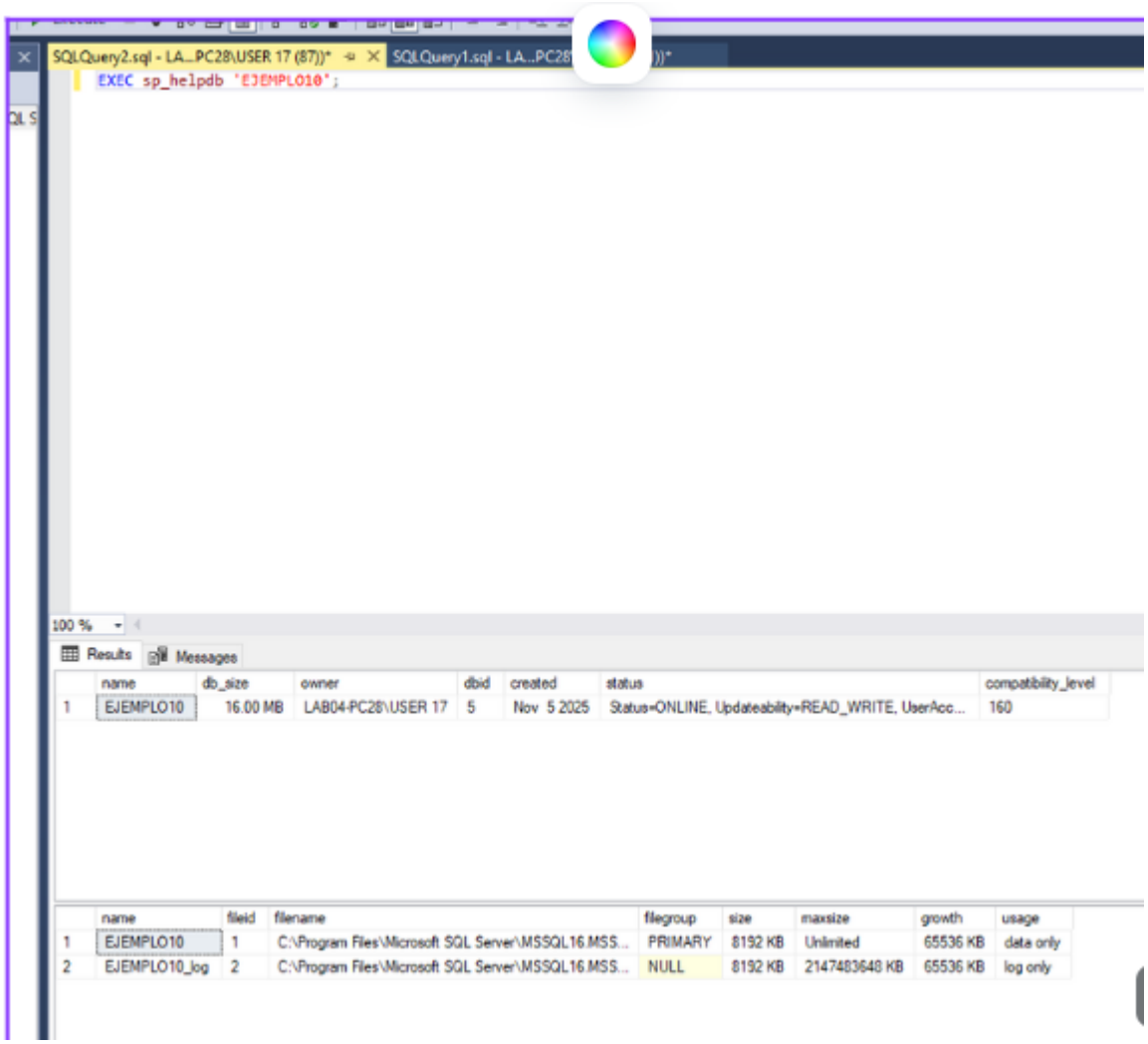
Estos archivos forman la **base física** de la base de datos. La “estructura” se refiere a **cómo están organizados, cómo crecen, y cómo se accede a ellos** para leer, escribir o respaldar datos.

Creamos una base de datos



luego buscamos en una nueva consulta escribimos el código de como saber si hemos creado una base de datos.

EXEC sp_helpdb 'EJEMPLO10';



2. Propiedades y configuraciones de bases de datos.

Cuando hablamos de “propiedades y configuraciones de bases de datos”, nos referimos a los parámetros y características que definen cómo funciona una base de datos: su nombre, ubicación, tamaño, codificación, modo de crecimiento, seguridad, entre otros.

En otras palabras, son los ajustes que controlan el comportamiento, el rendimiento y la administración de una base de datos dentro del sistema gestor (MySQL, SQL Server, PostgreSQL, etc.).

🧩 1. Propiedades de una base de datos

Son los **atributos básicos o informativos** que describen la base de datos.

Por ejemplo:

Propiedad	Descripción
Nombre	Identifica la base de datos (ej. `TiendaDB`)
Propietario	Usuario o rol que la creó y tiene control principal
Tamaño	Cuánto espacio ocupa en disco
Ubicación física	Ruta o carpeta donde se almacenan los archivos (`.mdf`, `.ibd`, etc.)
Fecha de creación	Cuándo fue creada
Estado	Si está activa, en recuperación, desconectada, etc.
Número de usuarios conectados	Cuántas conexiones tiene abiertas

⚙️ 2. Configuraciones de una base de datos

Son los **parámetros técnicos que afectan su funcionamiento.**
Se pueden ajustar para optimizar el rendimiento, la seguridad o la integridad de los datos.

Algunos ejemplos comunes:

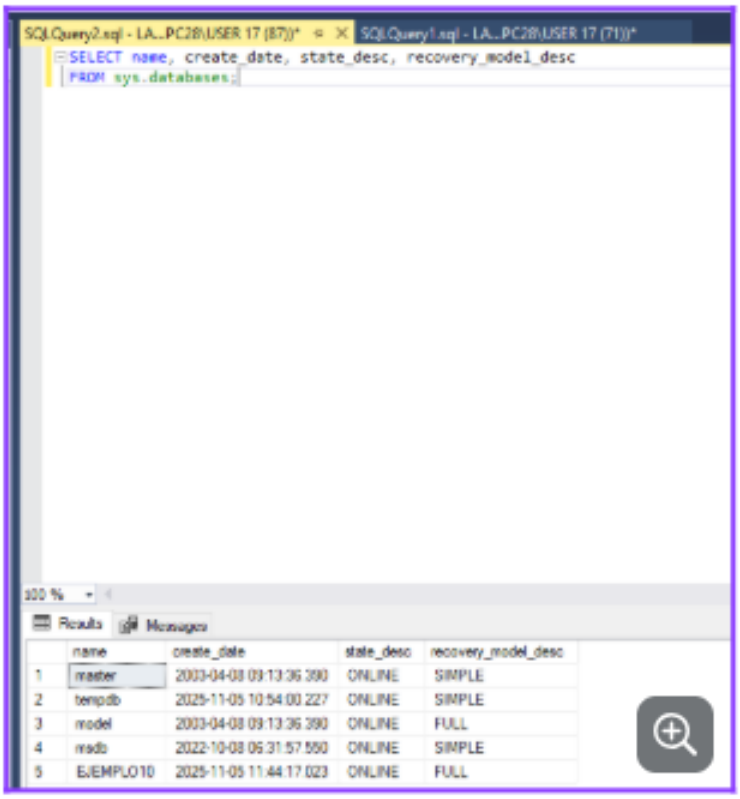
Configuración	Qué controla
Collation / Codificación	Cómo se almacenan y comparan los caracteres (importante para acentos y mayúsculas/minúsculas). Ej: `utf8_general_ci`
Modo de crecimiento automático (AUTO GROWTH)	Si los archivos de datos aumentan de tamaño automáticamente al llenarse.
Tamaño inicial y máximo	Cuánto espacio tiene la base de datos al crearse y hasta dónde puede crecer.
Modelo de recuperación (Recovery model)	Cómo se manejan los respaldos y los logs de transacciones (`FULL`, `SIMPLE`, `BULK_LOGGED` en SQL Server).
Opciones de conexión	Cuántos usuarios pueden conectarse, tiempo de espera, etc.
Permisos y roles	Qué usuarios tienen acceso y qué pueden hacer.

Ejemplo (en SQL Server):

```sql
SELECT name, create_date, state_desc, recovery_model_desc
FROM sys.databases;
```

SELECT name, create_date, state_desc, recovery_model_desc

FROM sys.databases



3. Tipos de recuperación (Simple, Full, Bulk-Logged).

Los **tipos de recuperación (Simple, Full y Bulk-Logged)** son **modos de administración del registro de transacciones** en **SQL Server** (y conceptos similares existen en otros gestores).

Estos **definen cómo se registran las operaciones** en la base de datos y **qué tan fácil o detallada puede ser la recuperación** en caso de un fallo o pérdida de datos.

¿Qué es el “modelo de recuperación”?

Cada base de datos en SQL Server tiene un **modelo de recuperación**, que controla:

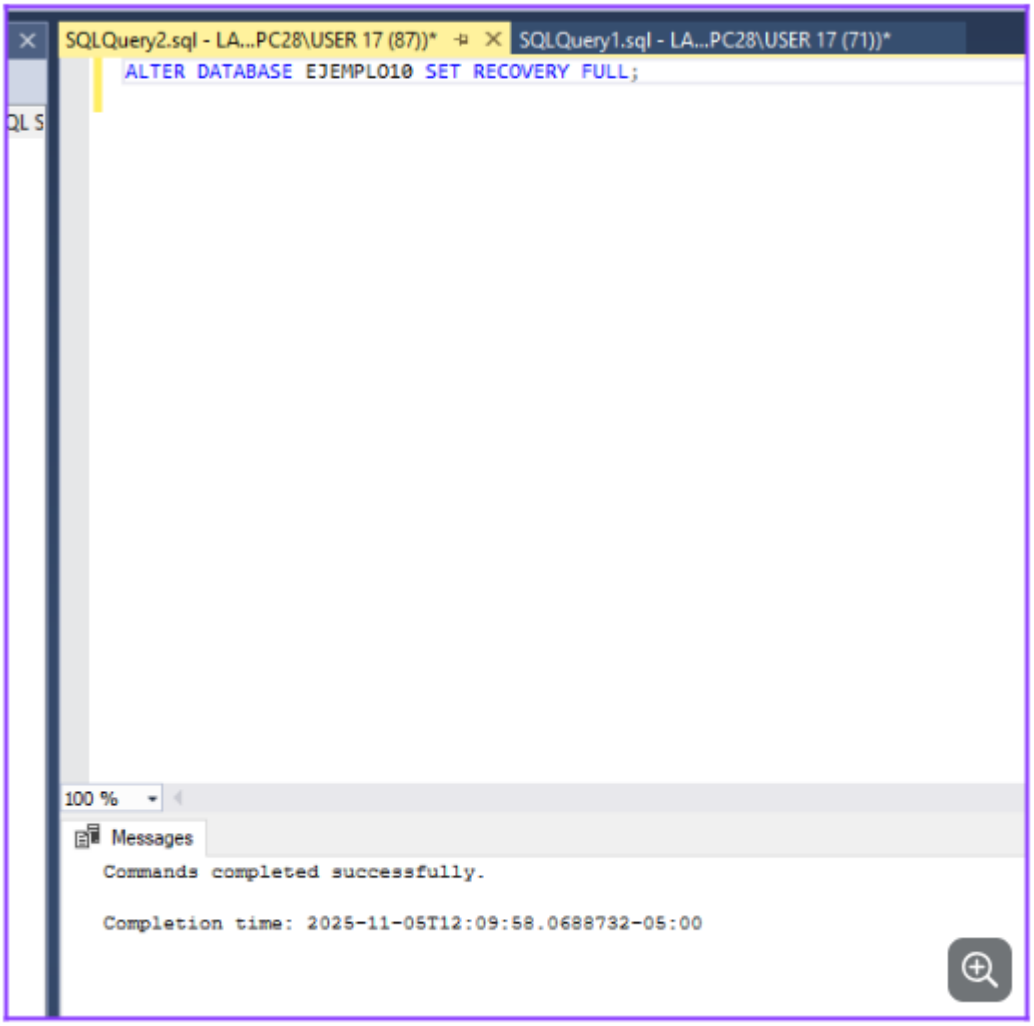
- Cómo se registra cada transacción (inserciones, actualizaciones, eliminaciones).
- Cuánto espacio usa el *log de transacciones* (archivo .ldf).
- Qué opciones de **respaldo y restauración** están disponibles.

1. Modelo de recuperación SIMPLE

Características:

- El *log de transacciones* **se limpia automáticamente** después de cada punto de control (*checkpoint*).
- **No puedes restaurar hasta un punto exacto en el tiempo**, solo hasta el último respaldo completo o diferencial.
- Ideal para bases de datos donde **no es crítico perder los cambios más recientes**, o donde los datos se pueden regenerar fácilmente.

```
ALTER DATABASE EJEMPLO10 SET RECOVERY SIMPLE;
```



3. Modelo de recuperación BULK-LOGGED

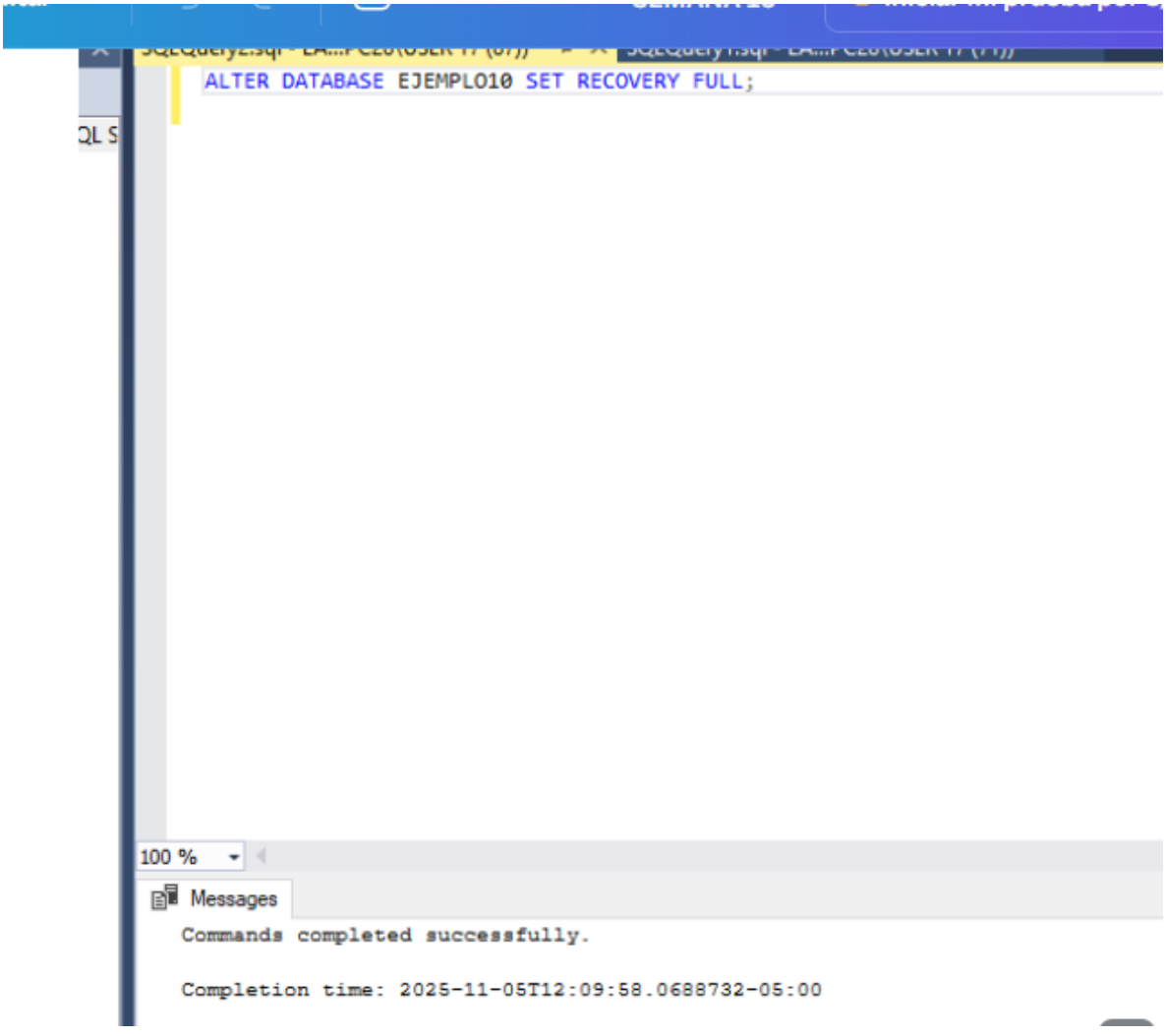
Características:

- Es similar al modo *Full*, pero **omite algunos detalles** durante operaciones masivas (*bulk operations*) para mejorar el rendimiento.
- Durante esas operaciones (como BULK INSERT, SELECT INTO, CREATE INDEX), **no se registran todos los detalles**, solo un resumen.
- Permite restauración hasta el punto del último respaldo del log, **siempre que no haya ocurrido una operación bulk sin respaldo posterior.**

Resumen comparativo

Característica	SIMPLE	FULL	BULK-LOGGED
Registro detallado de transacciones	❌ No	✅ Sí	⚙ Parcial
Tamaño del log	Pequeño	Grande	Moderado
Permite restaurar a un punto exacto	❌ No	✅ Sí	⚙ Depende
Requiere respaldo de log	❌ No	✅ Sí	✅ Sí
Ideal para	Pruebas, desarrollo	Producción crítica	Cargas masivas de

ALTER DATABASE EJEMPLO10 SET RECOVERY BULK_LOGGED;



3. Modelo de recuperación BULK-LOGGED

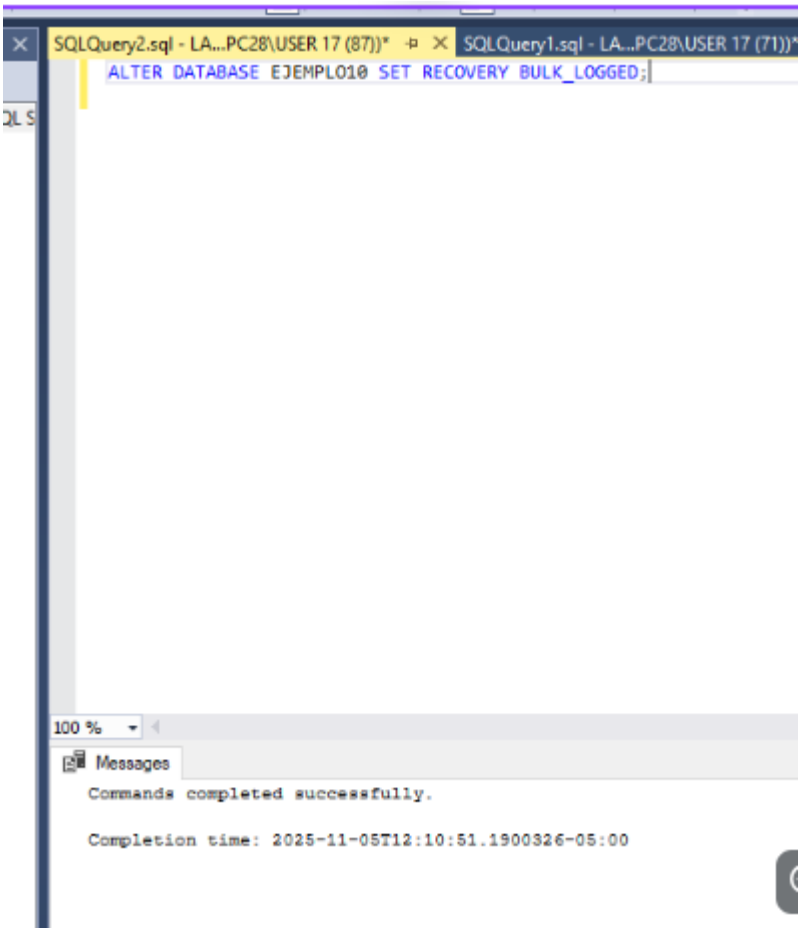
Características:

- Es similar al modo *Full*, pero **omite algunos detalles** durante operaciones masivas (*bulk operations*) para mejorar el rendimiento.
- Durante esas operaciones (como BULK INSERT, SELECT INTO, CREATE INDEX), **no se registran todos los detalles**, solo un resumen.
- Permite restauración hasta el punto del último respaldo del log, **siempre que no haya ocurrido una operación bulk sin respaldo posterior.**

📊 Resumen comparativo

Característica	SIMPLE	FULL	BULK-LOGGED
Registro detallado de transacciones	❌ No	✅ Sí	⚙️ Parcial
Tamaño del log	Pequeño	Grande	Moderado
Permite restaurar a un punto exacto	❌ No	✅ Sí	⚙️ Depende
Requiere respaldo de log	❌ No	✅ Sí	✅ Sí
Ideal para	Pruebas, desarrollo	Producción crítica	Cargas masivas de

ALTER DATABASE EJEMPLO10 SET RECOVERY BULK_LOGGED;



4. Administración de usuarios y roles de seguridad.

El tema “Administración de usuarios y roles de seguridad” es una parte fundamental de la gestión de bases de datos, ya que se trata de controlar quién puede acceder a la base de datos y qué puede hacer dentro de ella.

¿Qué significa?

La **administración de usuarios y roles de seguridad** consiste en:

1. **Crear usuarios** que puedan conectarse al sistema de base de datos.
2. **Asignar permisos** (lectura, escritura, modificación, administración, etc.).
3. **Organizar a los usuarios en roles** para simplificar la gestión de privilegios.

Esto asegura que **solo las personas autorizadas** puedan acceder a los datos y **solo realizar las acciones necesarias**, lo que protege la base de datos de errores o accesos indebidos.

1. Usuarios

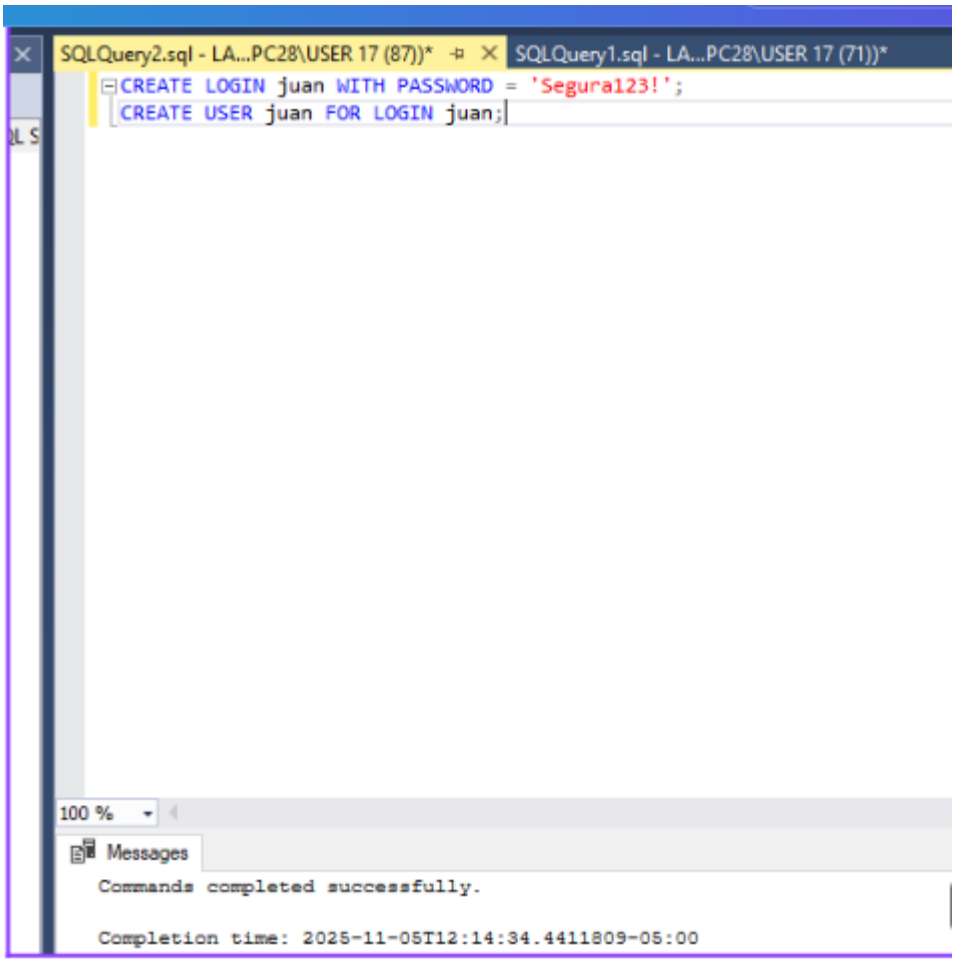
Un **usuario** es una cuenta con la que alguien (una persona o aplicación) accede a la base de datos.

Cada usuario puede tener:

- Un **nombre de usuario** (user),
- Una **contraseña**,
- Y **permisos** específicos.

Ejemplo en SQL Server:

```
cREATE LOGIN juan000 WITH PASSWORD = 'Juan123!';

CREATE LOGIN maria000 WITH PASSWORD = 'Maria123!';

-- Crear usuarios en la base de datos

CREATE USER juan000 FOR LOGIN juan000;

CREATE USER maria000 FOR LOGIN maria000;

GO
```



2. Permisos

Los **permisos** definen qué puede hacer cada usuario.

Ejemplos de permisos:

- SELECT → Leer datos.
- INSERT → Agregar datos.
- UPDATE → Modificar datos.
- DELETE → Eliminar datos.
- CREATE, ALTER, DROP → Administrar tablas o bases de datos.

```
CREATE LOGIN juan WITH PASSWORD = 'Segura123!';

-- Usar la base de datos donde se va a crear el usuario

USE EJEMPLO10;

-- Crear el usuario dentro de la base de datos
```

CREATE USER juan FOR LOGIN juan;

3. Roles

Un **rol** es un conjunto de permisos que puede asignarse a varios usuarios. Así se simplifica la administración: en lugar de dar permisos uno por uno a cada usuario, se asigna un **rol**.

CREATE ROLE Vendedor;

GRANT SELECT, INSERT ON Ventas TO Vendedor;

EXEC sp_addrolemember 'Vendedor', 'juan'

5. Asignación de permisos y políticas de acceso.

La **asignación de permisos y políticas de acceso** consiste en **definir qué usuarios o roles pueden hacer qué acciones** sobre la base de datos, y de esta forma controlar la seguridad de los datos.

Te daré un **ejemplo completo en SQL Server**, usando la base de datos TiendaDB y las tablas Clientes y Ventas que creamos antes.

Crear roles

Primero, creamos roles según el tipo de acceso que queramos dar:

USE EJEMPLO10;

GO

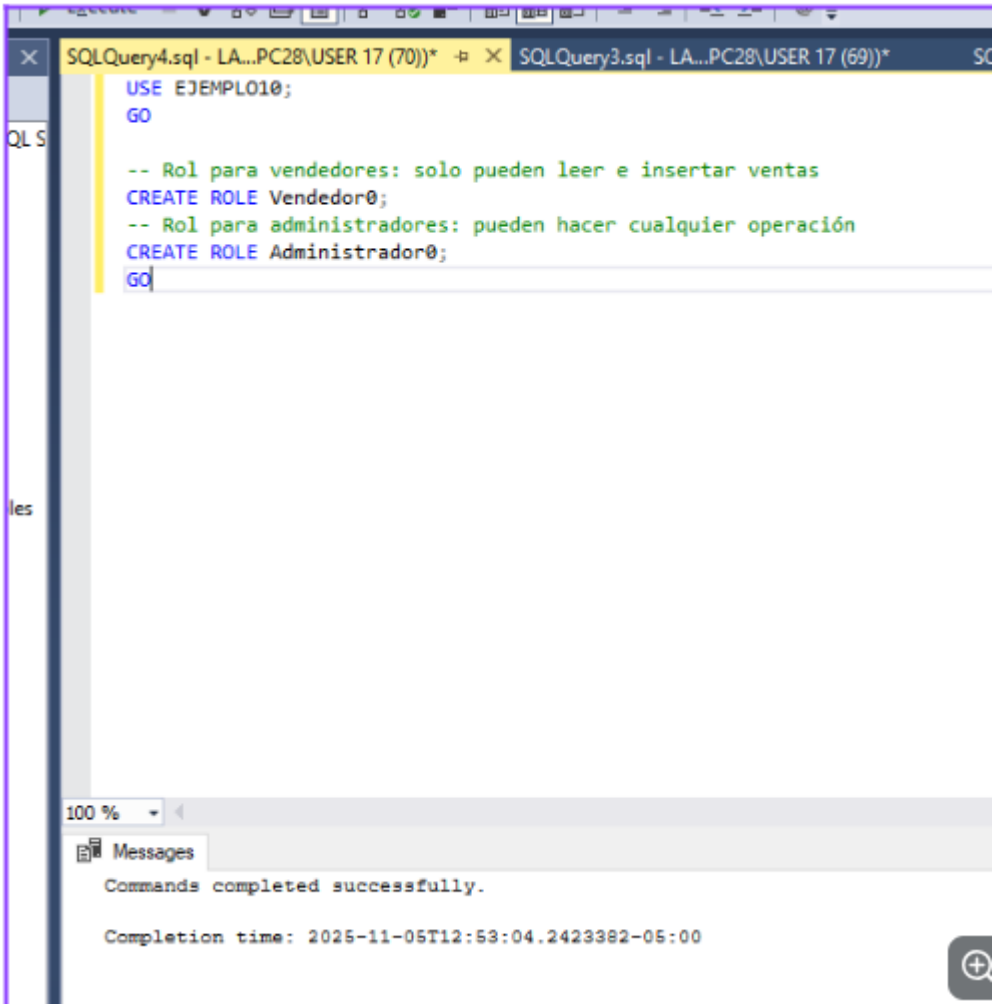
-- Rol para vendedores: solo pueden leer e insertar ventas

CREATE ROLE Vendedor0;

-- Rol para administradores: pueden hacer cualquier operación

CREATE ROLE Administrador0;

GO



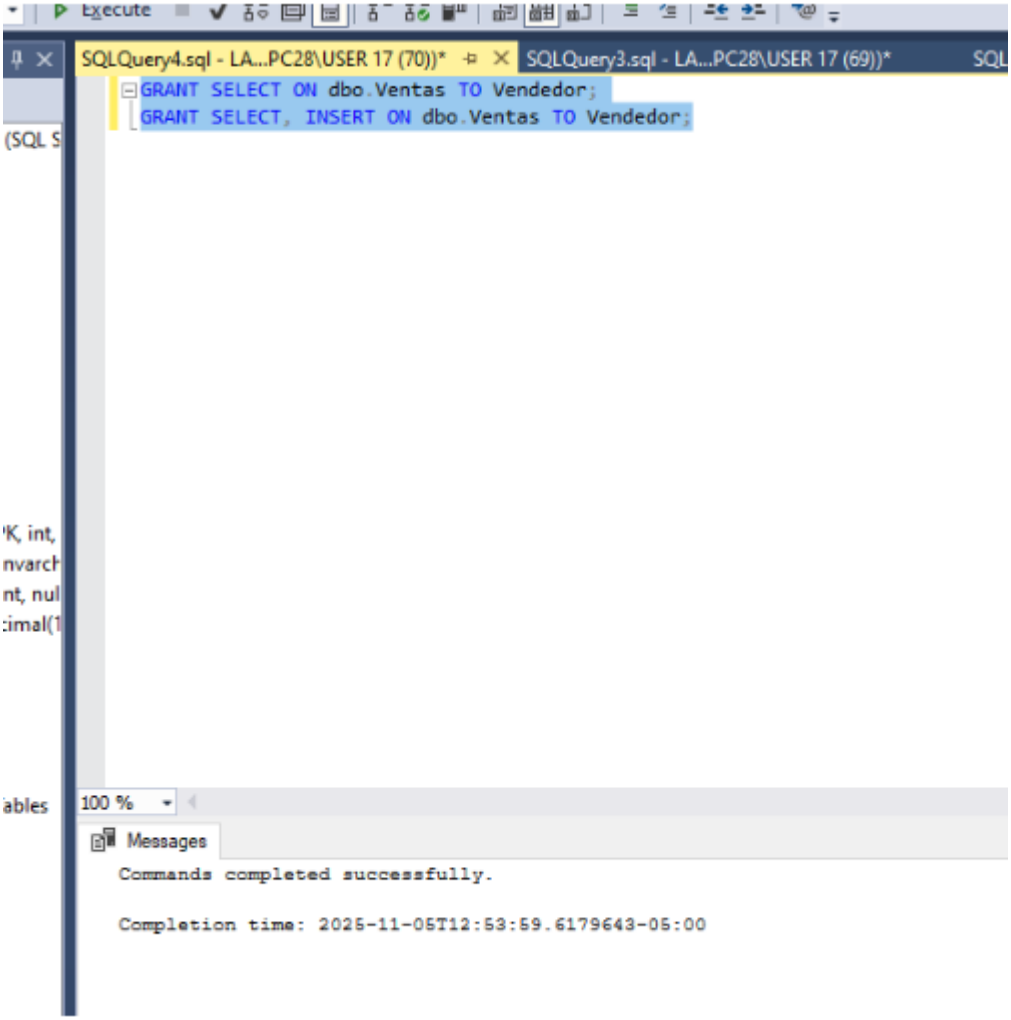
Asignar permisos a los roles

permisos para el rol Vendedor

-- Permiso para leer clientes y ventas

GRANT SELECT ON dbo.Ventas TO Vendedor

GRANT SELECT, INSERT ON dbo.Ventas TO Vendedor



Permisos para el rol Administrador

-- Permisos completos sobre todas las tablas

GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER, DROP ON dbo.Clientes TO Administrador;

GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER, DROP ON dbo.Ventas TO Administrador;

Crear usuarios

-- Crear usuario Juan (vendedor)

CREATE LOGIN juan WITH PASSWORD = 'Segura123!';

CREATE USER juan FOR LOGIN juan;

-- Crear usuario Ana (administrador)

CREATE LOGIN ana WITH PASSWORD = 'Admin123!';

CREATE USER ana FOR LOGIN ana;

Asignar usuarios a roles

-- Juan será Vendedor

EXEC sp_addrolemember 'Vendedor', 'juan';

-- Ana será Administrador

EXEC sp_addrolemember 'Administrador', 'ana';

Verificar permisos

Para comprobar qué permisos tiene un usuario:

-- Ver permisos de Juan

EXEC sp_helprotect 'juan';

-- Ver miembros de un rol

EXEC sp_helprolemember 'Vendedor';

6. Monitoreo básico con el SQL Server Activity Monitor.

Qué es SQL Server Activity Monitor

El **Activity Monitor** es una herramienta gráfica incluida en **SQL Server Management Studio (SSMS)** que te permite:

- Ver los **procesos activos**.
- Monitorear el **uso de CPU, memoria y I/O**.
- Detectar **bloqueos y transacciones largas**.
- Revisar el **estado de los recursos de tu servidor**.

Se usa principalmente para **diagnóstico rápido** del rendimiento.

Cómo abrir Activity Monitor

1. Abre **SQL Server Management Studio (SSMS)**.
2. Conéctate a tu servidor.
3. Haz clic derecho sobre el servidor en **Object Explorer**.
4. Selecciona **Activity Monitor** (o presiona Ctrl+Alt+A).

Pestañas principales del Activity Monitor

Pestaña	Qué muestra
Processes	Todas las conexiones activas y sus consultas en ejecución
Resource Waits	Esperas por recursos (CPU, I/O, bloqueos)
Data File I/O	Actividad de lectura/escritura de archivos de la base de datos
Recent Expensive Queries	Consultas que consumen más recursos
Blocked Processes	Transacciones bloqueadas y sus causas

a) Ver procesos activos (similar a la pestaña Processes)

```
SELECT

session_id,

login_name,

status,

blocking_session_id,

wait_type,

cpu_time,

total_elapsed_time,

text AS query_text

FROM sys.dm_exec_requests
```

```
CROSS APPLY sys.dm_exec_sql_text(sql_handle);
```

b) Ver consultas más costosas (CPU/IO) (similar a Recent Expensive Queries)

```
SELECT TOP 10
qs.total_worker_time AS CPU,
qs.total_elapsed_time AS Tiempo,
qs.execution_count,
SUBSTRING(qt.text, (qs.statement_start_offset/2)+1,
((CASE qs.statement_end_offset
WHEN -1 THEN DATALENGTH(qt.text)
ELSE qs.statement_end_offset
END - qs.statement_start_offset)/2)+1) AS query_text
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) qt
ORDER BY qs.total_worker_time DESC;
```

c) Ver bloqueos (similar a Blocked Processes)

```
SELECT
blocking_session_id AS bloqueador,
session_id AS bloqueado,
wait_type,
wait_time,
text AS query_text
FROM sys.dm_exec_requests
CROSS APPLY sys.dm_exec_sql_text(sql_handle)
```

```
WHERE blocking_session_id <> 0;
```

d) Uso de recursos por base de datos (CPU, I/O)

```
SELECT

DB_NAME(database_id) AS BaseDatos,

SUM(num_of_reads) AS Lecturas,

SUM(num_of_writes) AS Escrituras,

SUM(cpu_time) AS CPU

FROM sys.dm_exec_query_stats qs

CROSS APPLY sys.dm_exec_query_plan(qs.plan_handle)

GROUP BY database_id;
```

7. Introducción al uso de SQL Server Agent (tareas automáticas).

Qué es SQL Server Agent

- Es un **servicio de SQL Server** que permite programar y automatizar tareas (jobs).
- Las tareas pueden incluir:
  - Ejecutar **scripts T-SQL**.
  - Hacer **copias de seguridad**.
  - Ejecutar **paquetes SSIS**.
  - Enviar **alertas por correo**.
- Muy útil para **mantenimiento diario**, por ejemplo: respaldos, limpieza de datos o informes automáticos.

Nota: SQL Server Agent debe estar **activado**. En SSMS lo ves en **Object Explorer → SQL Server Agent**. Si tiene un ícono rojo, dale clic derecho → **Start**.

Crear un Job básico usando SQL Server Agent

Paso 1: Crear el Job

```
USE msdb;

GO

EXEC sp_add_job
```

@job_name = N'RespaldarTiendaDB', -- Nombre del job

@enabled = 1, -- Activo

@description = N'Job diario para respaldar TiendaDB';

GO

Paso 2: Agregar un paso al Job

Este paso ejecutará un **script T-SQL**:

EXEC sp_add_jobstep

@job_name = N'RespaldarTiendaDB',

@step_name = N'BackupPaso',

@subsystem = N'TSQL',

@command = N'BACKUP DATABASE TiendaDB TO DISK = "C:\Backups\TiendaDB.bak" WITH INIT;',

@retry_attempts = 3,

@retry_interval = 5;

GO

Paso 3: Programar el Job

Podemos hacer que se ejecute **diariamente a las 2 AM**:

EXEC sp_add_schedule

@schedule_name = N'Diario2AM',

@freq_type = 4, -- Diario

@freq_interval = 1, -- Todos los días

@active_start_time = 020000; -- 02:00 AM

GO

EXEC sp_attach_schedule

@job_name = N'RespaldarTiendaDB',

@schedule_name = N'Diario2AM';

GO

Paso 4: Asignar un operador (opcional)

Para enviar alertas por correo si falla el Job:

EXEC sp_add_operator

@name = N'DBA_Operador',

@email_address = N'dba@empresa.com';

GO

EXEC sp_add_notification

@job_name = N'RespaldarTiendaDB',

@operator_name = N'DBA_Operador',

@notification_method = 1; -- 1 = Email

GO

Paso 5: Ejecutar el Job manualmente (opcional)

EXEC sp_start_job @job_name = 'RespaldarTiendaDB';