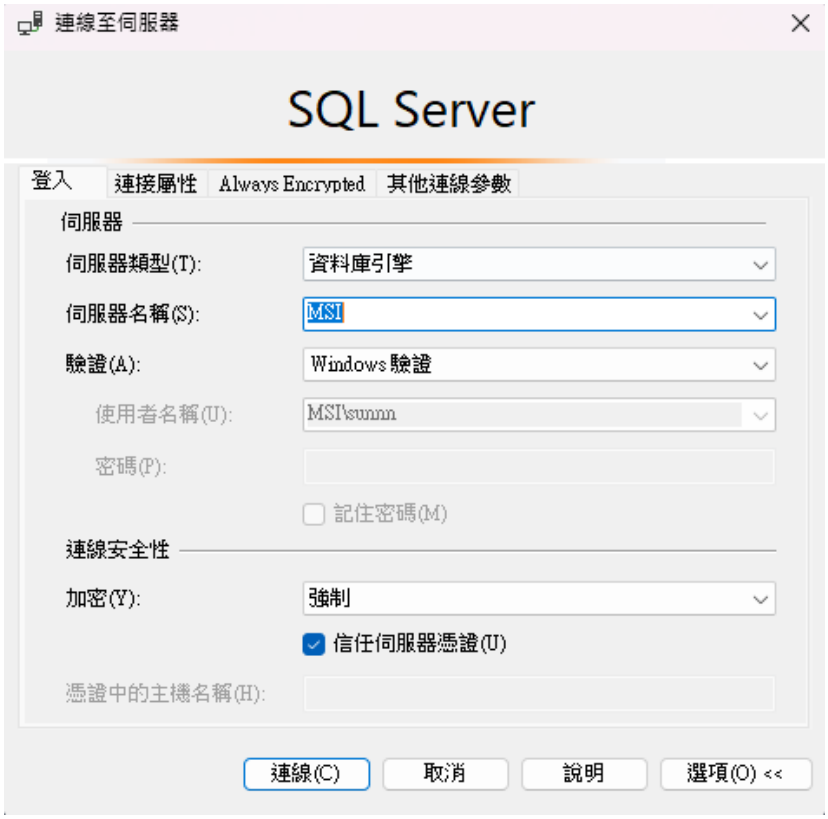
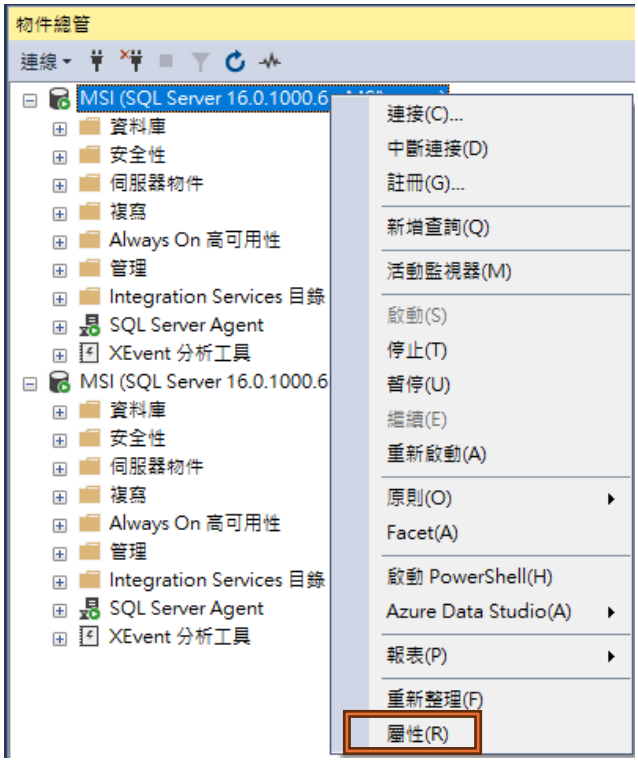
先確認資料庫驗證模式:

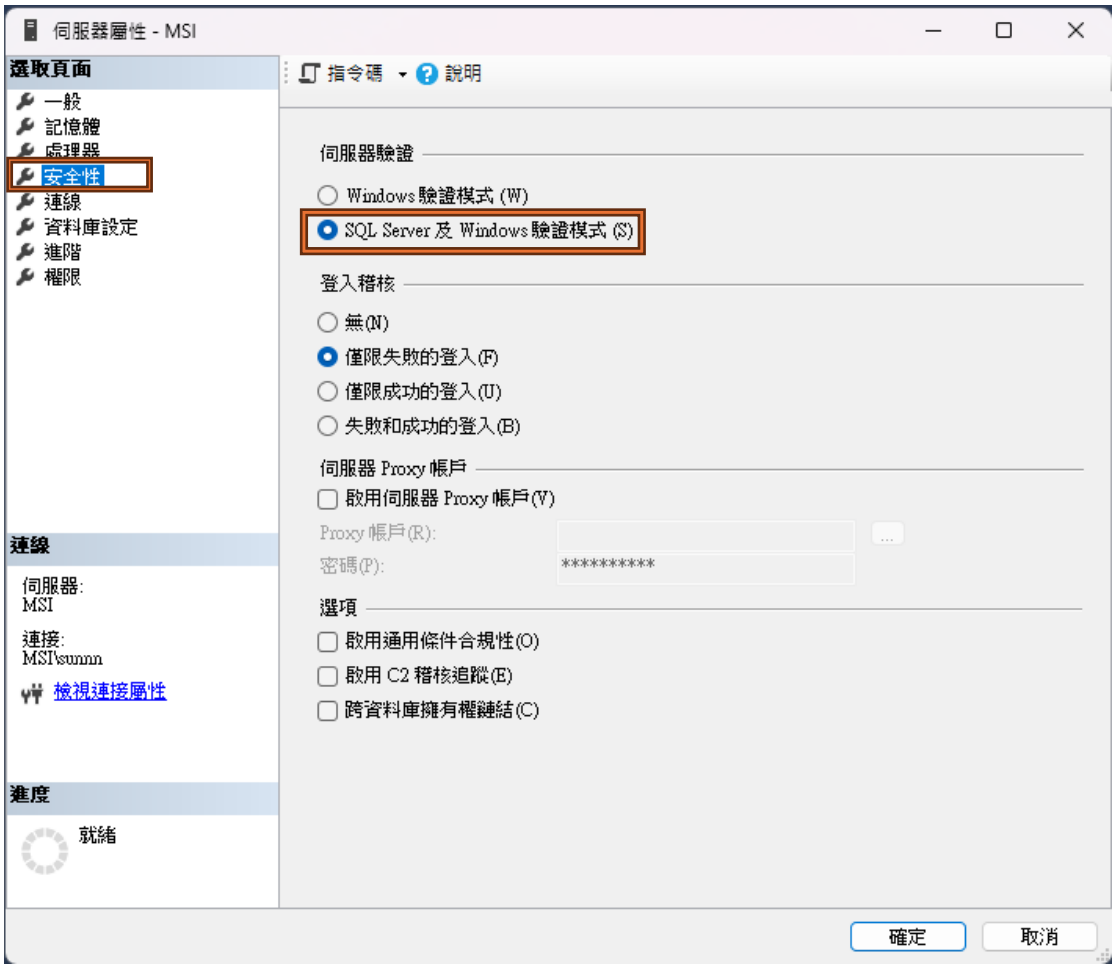
1. 先使用 Window 驗證進行登入



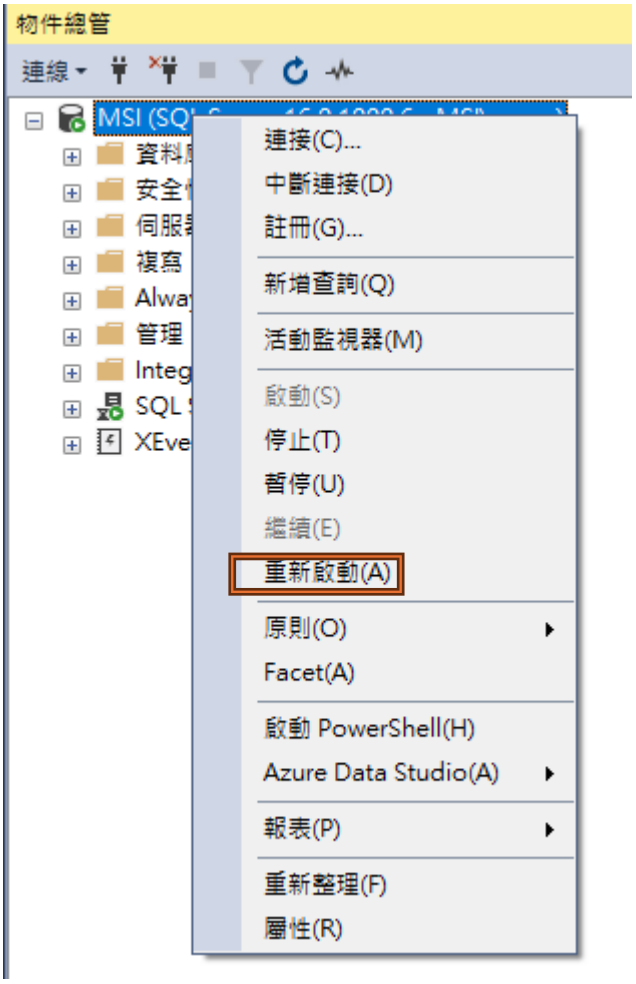
2. 右鍵點選資料庫，選擇屬性。



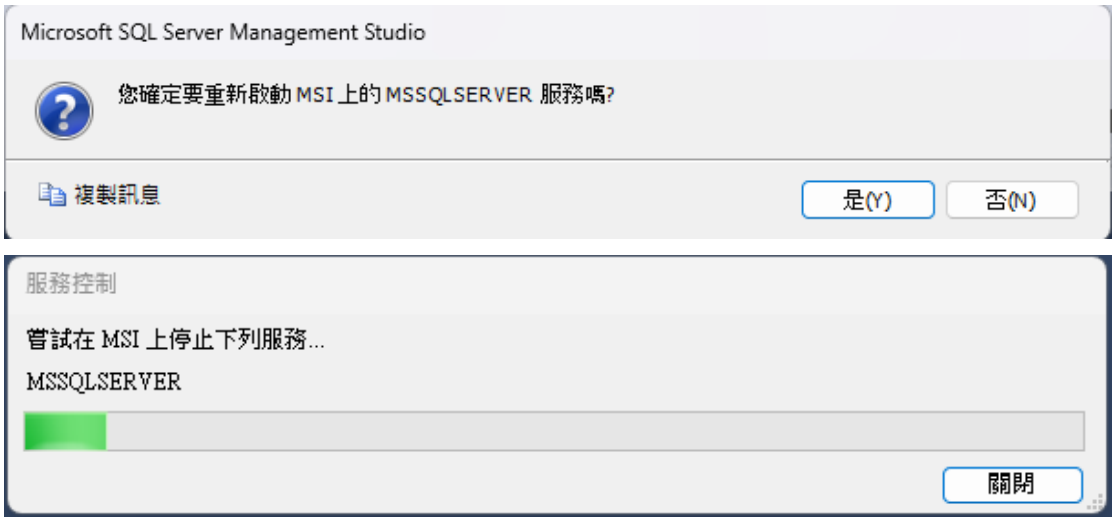
3. 選擇安全性 >> 勾選 SQL Server 及 Windows 驗證模式



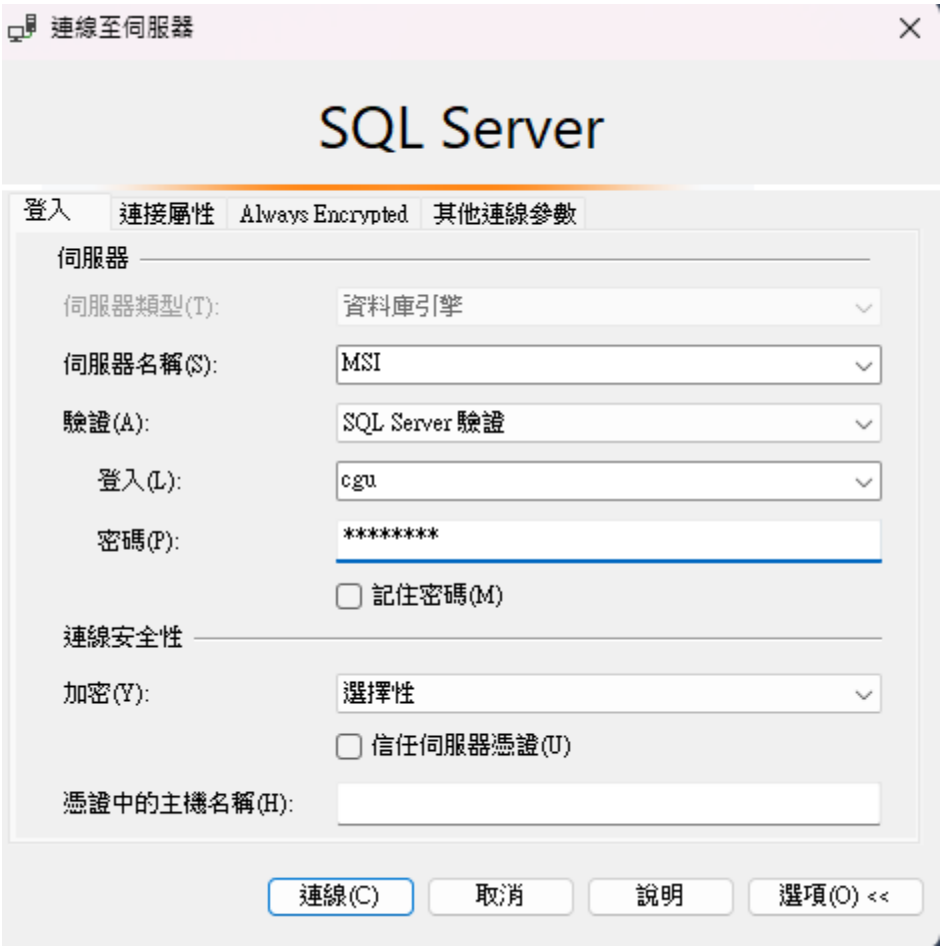
4. 右鍵選擇資料庫並選擇重新啟動，套用 SQL Server 及 Windows 驗證模式



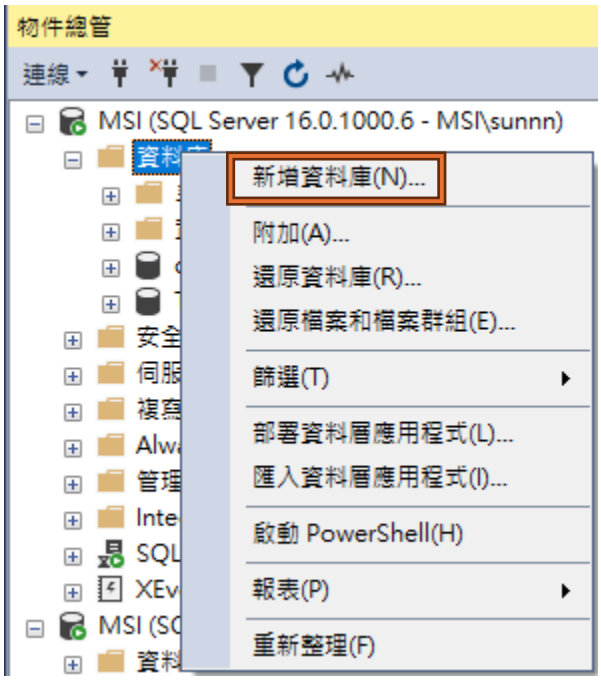
5. 點選「是」



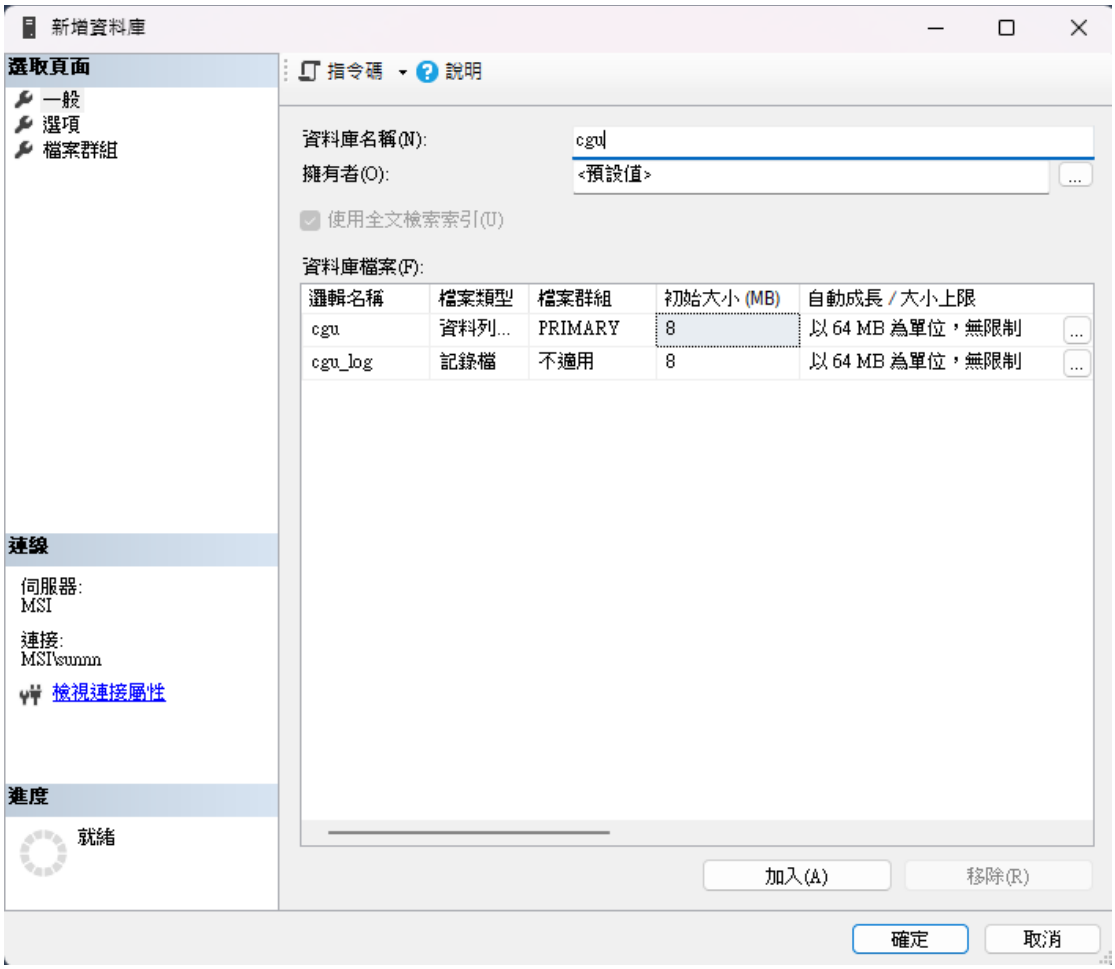
6. 重新登入 SQL Server



新增新資料庫 cgu



1. 輸入資料庫名稱 cgu

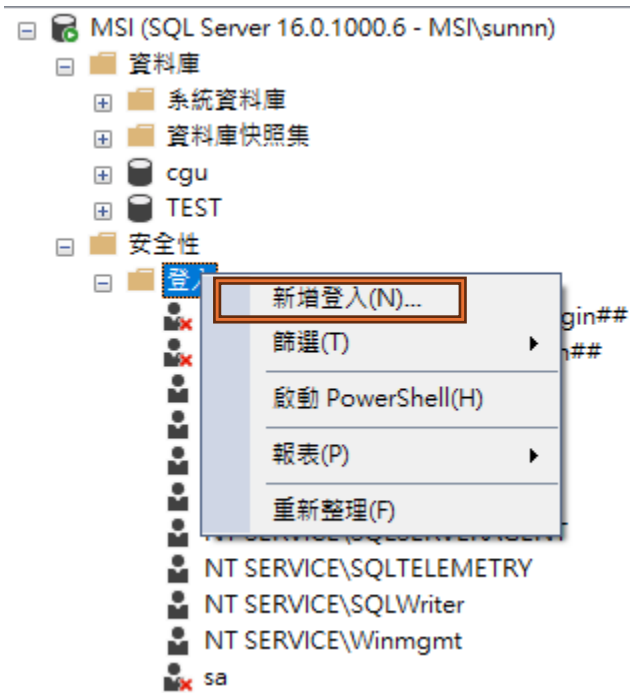


2. 按下確認後即建立成功

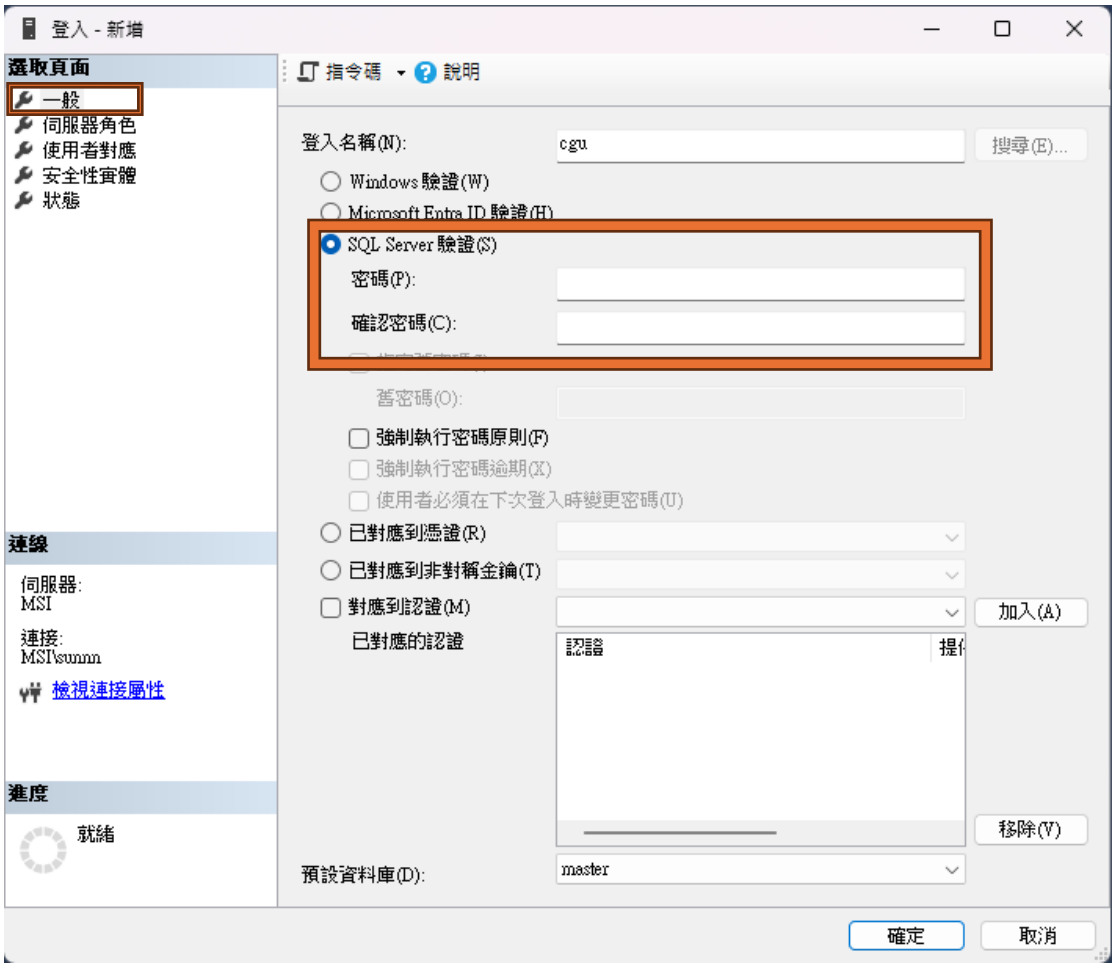
暫停 SA 避免安全性問題:

先透過登入 server 利用 SA 創立 cgu 的新登入帳號

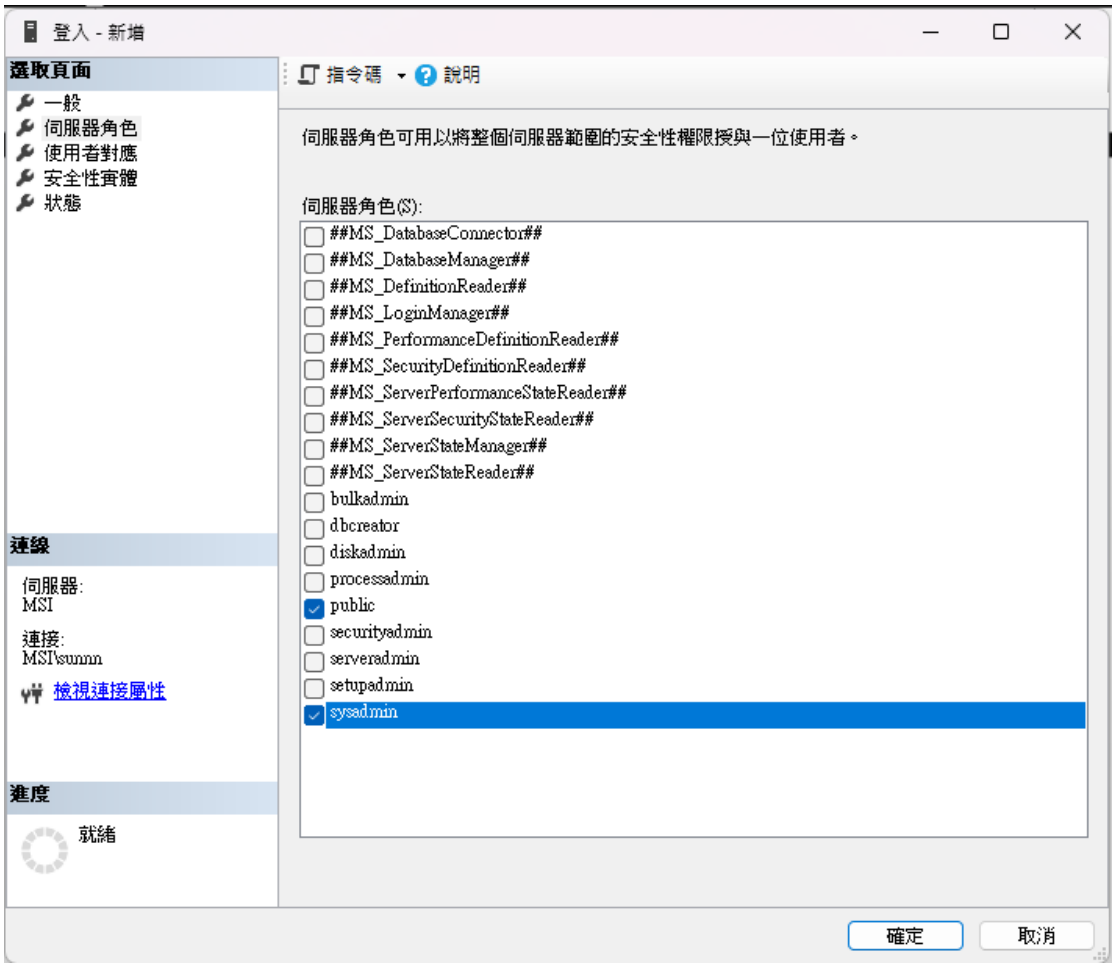
1. 點開安全性資料夾，再登入按下右鍵點選新增登入



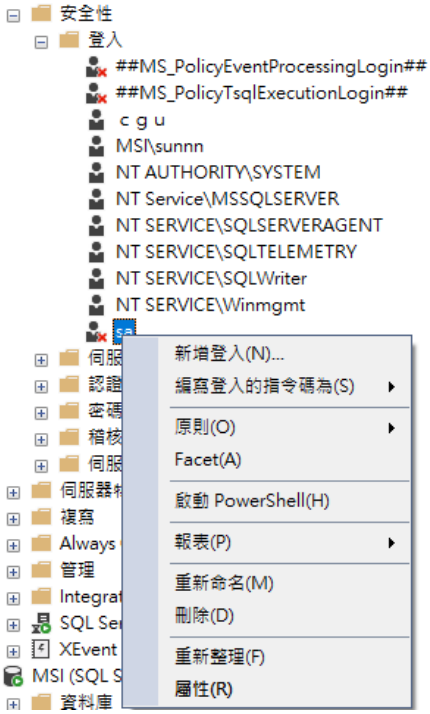
2. 輸入新登入身分的名稱，並勾選 SQL server 驗證，設立自己的密碼


登入 - 新增
選取頁面
一般
伺服器角色
使用者對應
安全性實體
狀態
指令碼
說明
登入名稱(N):
cgu
搜尋(E)...
Windows 驗證(W)
Microsoft Entra ID 驗證(H)
SQL Server 驗證(S)
密碼(P):
確認密碼(C):
舊密碼(O):
強制執行密碼原則(F)
強制執行密碼逾期(X)
使用者必須在下次登入時變更密碼(U)
已對應到憑證(R)
已對應到非對稱金鑰(T)
對應到認證(M)
加入(A)
已對應的認證
認證
移除(V)
預設資料庫(D):
master
連線
伺服器:
MSI
連接:
MSI\sunnn
檢視連接屬性
進度
就緒
確定
取消

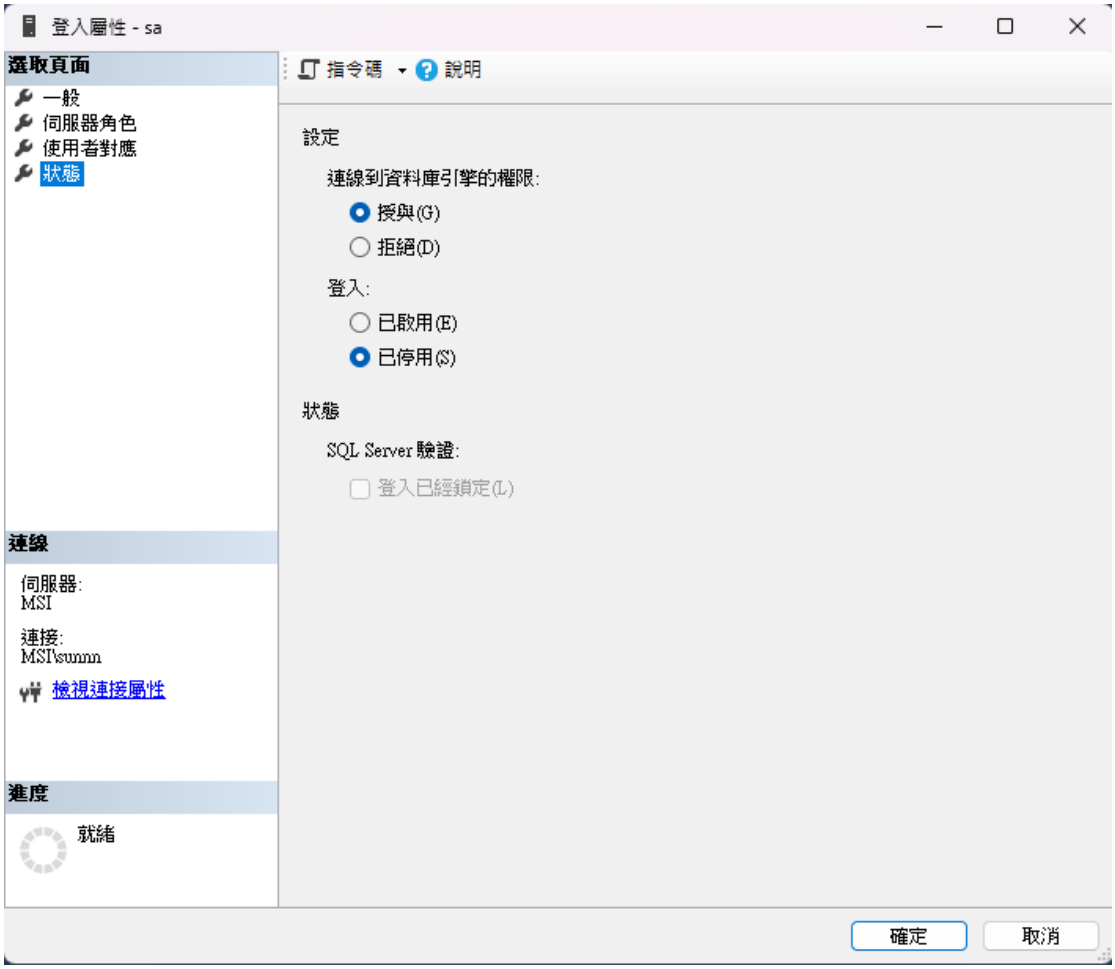
3. 勾選 sysadmin 給予最高權限



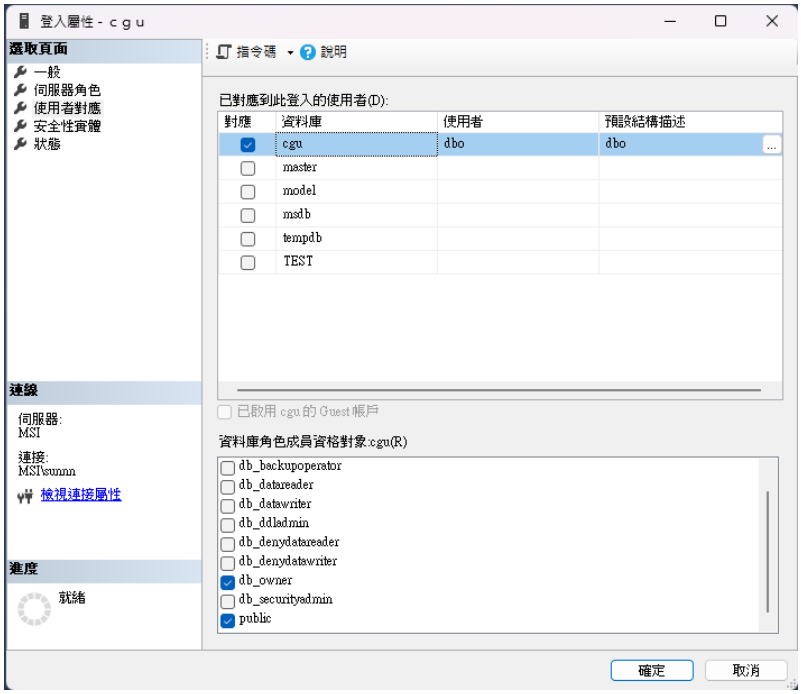
4. 選擇 sa 屬性



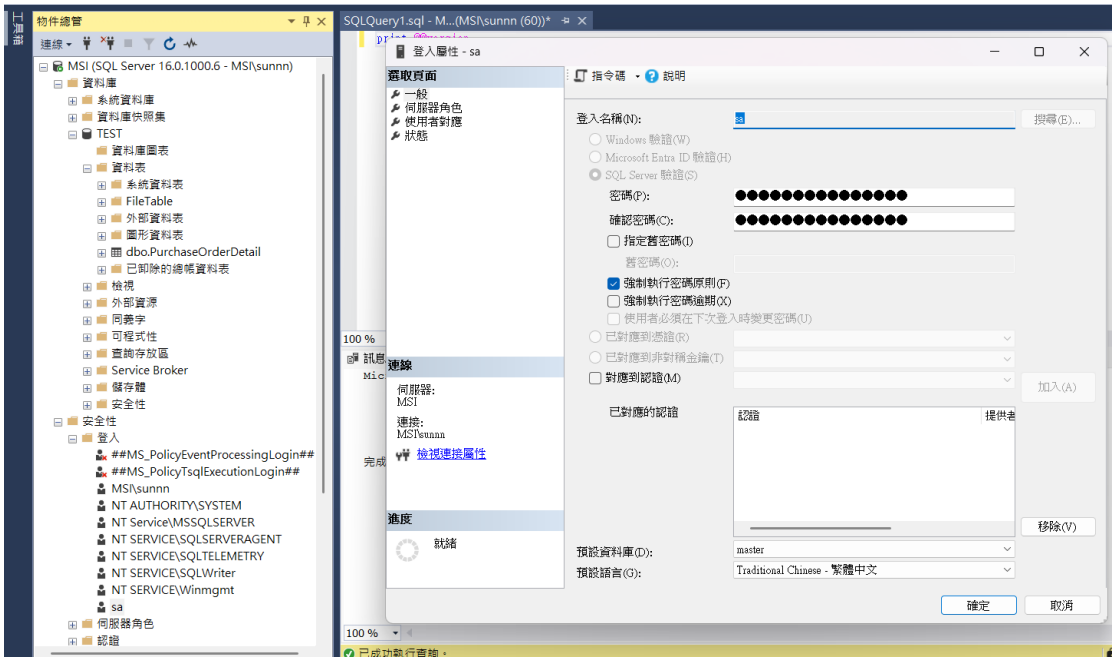
5. 選擇狀態將登入勾選已停用:



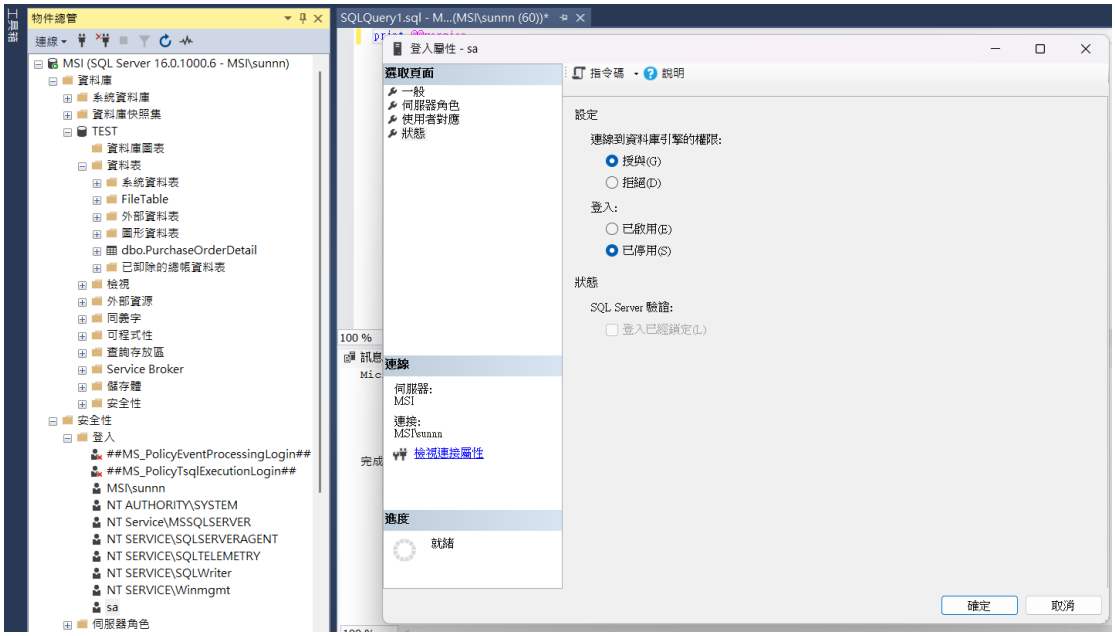
6. 選擇使用者對應選擇 db_owner:



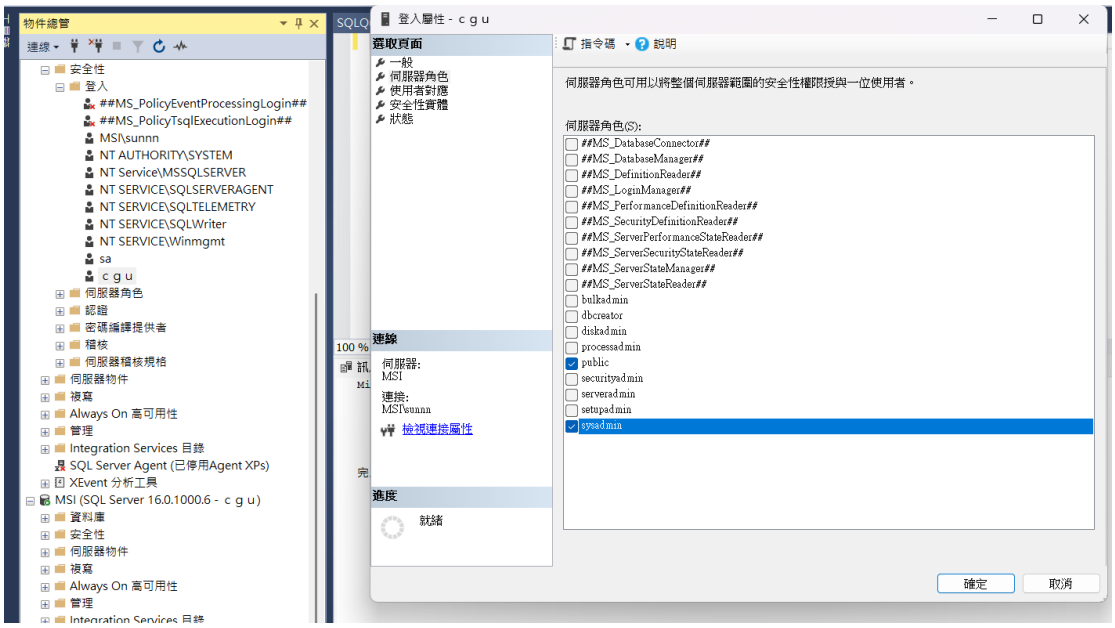
之後再右鍵於登入底下的 sa 點選屬性，會出現下列畫面。



再點選狀態，選擇已停用來停用 sa



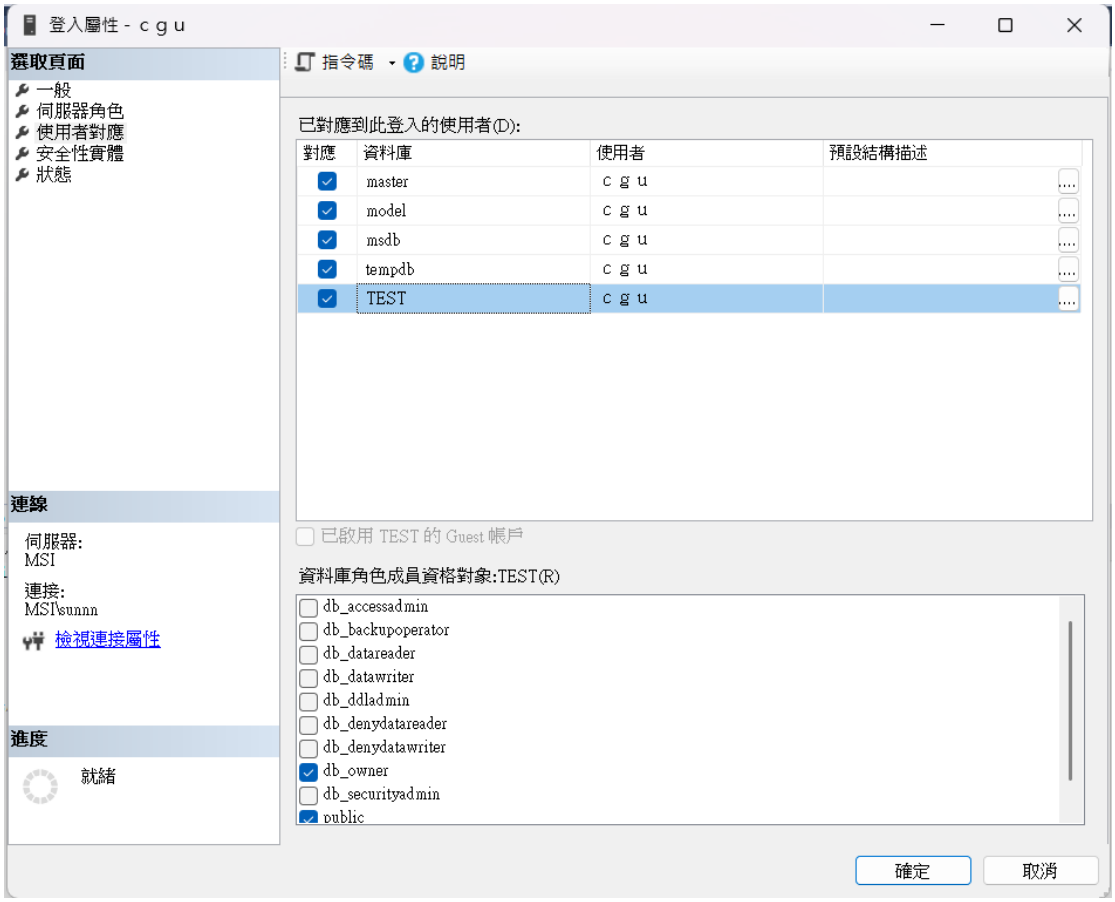
停用成功後再進入上面創建好的新身分 cgu 要進去屬性中，伺服器角色部分勾選最後的選項，給予最高權限。



其他角色權限簡單介紹:

角色名稱	中文名稱	主要權限／用途
bulkadmin	批次匯入管理員	可以執行大量資料匯入作業（BULK INSERT）。
dbcreator	資料庫建立者	可以建立、修改、刪除所有資料庫。
diskadmin	磁碟管理員	管理 SQL Server 使用的磁碟檔案 (filegroups、資料檔與日誌檔)。
processadmin	工作管理員	可以查看與終止目前所有使用者的執行緒 (session/process)。
public	公共角色	所有登入預設都屬於此角色，定義最基本的可用權限。
securityadmin	安全性管理員	管理登入與權限、設定伺服器層級的安全原則，也能修改資料庫使用者。
serveradmin	伺服器管理員	設定伺服器層級設定（如關閉 / 開啟伺服器），重新啟動伺服器。
setupadmin	安裝管理員	可以新增 / 刪除 linked servers，以及執行某些伺服器層級的設定作業。
sysadmin	系統管理員	擁有 SQL Server 實例所有權限，等同「神級」角色。可以做任何事。

點選使用者對應



資料庫層級角色（Database Roles）及其權限差異:

角色名稱	中文名稱	主要權限／用途
db_accessadmin	存取管理員	可以新增或移除資料庫內的使用者與角色對應，也就是管理誰能登入並存取此資料庫。
db_backupoperator	備份操作員	可以備份資料庫（BACKUP DATABASE），並且可以還原資料庫（RESTORE）時需要的基本權限。
db_datareader	資料讀取者	可以對所有資料表與檢視（VIEW）執行 SELECT，讀取資料。
db_datawriter	資料寫入者	可以對所有資料表與檢視執行 INSERT、UPDATE、DELETE，寫入或修改資料。
db_ddladmin	DDL 管理員	可以執行任何 DDL (Data Definition Language) 操作，如 CREATE、ALTER、DROP 資料表、索引、檢視、預存程序等。
db_denydatareader	拒絕資料讀取	拒絕對所有資料表與檢視執行

	者	SELECT。其 DENY 權限會覆蓋來自 db_datareader 或其他角色的 ALLOW。
db_denydatawriter	拒絕資料寫入者	拒絕對所有資料表與檢視執行 INSERT、UPDATE、DELETE。其 DENY 權限會覆蓋來自 db_datawriter 或其他角色的 ALLOW。
db_owner	資料庫所有者	擁有此資料庫的所有權限，等同「神級」，可以做任何操作（含刪除資料庫本身）。
db_securityadmin	安全性管理員	管理資料庫層級的許可權（GRANT、DENY、REVOKE），以及管理資料庫使用者登入對應的安全設定。
public	公共角色	所有使用者預設都屬於此角色，包含最基本的連線與檢視元資料（catalog）權限，無法讀寫資料表內容。

下面點擊右鍵開啟 Server Agent:

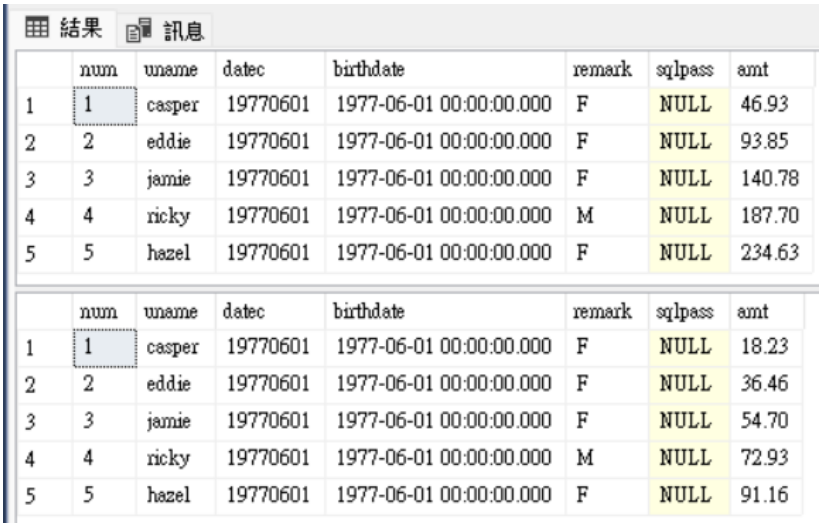
簡單 SQL 指令練習:

建立 cgu_h 資料表:

```
-- 切換到 cgu 資料庫

USE cgu;

GO

-- 建立修課學生主檔 (cgu_h)

CREATE TABLE cgu_h (

    num             BIGINT IDENTITY(1,1) NOT NULL,          -- 主流水號，自動遞增

    uname           NVARCHAR(20)         NOT NULL,          -- 用戶名稱

    datec           CHAR(8)              NULL,              -- 入學日期 (YYYYMMDD)

    birthdate       DATETIME             NULL,              -- 出生日期

    amt             DECIMAL(20,2)        NULL,              -- 可用資金

    remark          NVARCHAR(96)         NULL,              -- 備註說明

    sqlpass         VARCHAR(254)         NULL,              -- SQL 密碼或憑證

    CONSTRAINT pk_cgu_h_num PRIMARY KEY (num)               -- 設定 num 為主鍵

);

GO

-- (若需為 uname 建立索引，可取消下列註解)

-- CREATE INDEX in_cgu_h_uname ON cgu_h (uname);

-- GO
```

Insert 練習:

```
-- 切換到 cgu 資料庫

USE cgu;

GO  -- 以 GO 作為批次結束，SQL Server 會在此執行前面的語句

-- 查詢 cgu_h 資料表所有欄位與資料

SELECT *

FROM cgu_h;

GO

-- 更新 cgu_h 資料表：將 amt 欄位設為 num 欄位乘以隨機數 (0~1) 再乘以 100

UPDATE cgu_h

SET amt = num * RAND() * 100;

GO

-- 再次查詢 cgu_h 資料表，檢視更新後結果

SELECT *

FROM cgu_h;

GO
```



結果 訊息

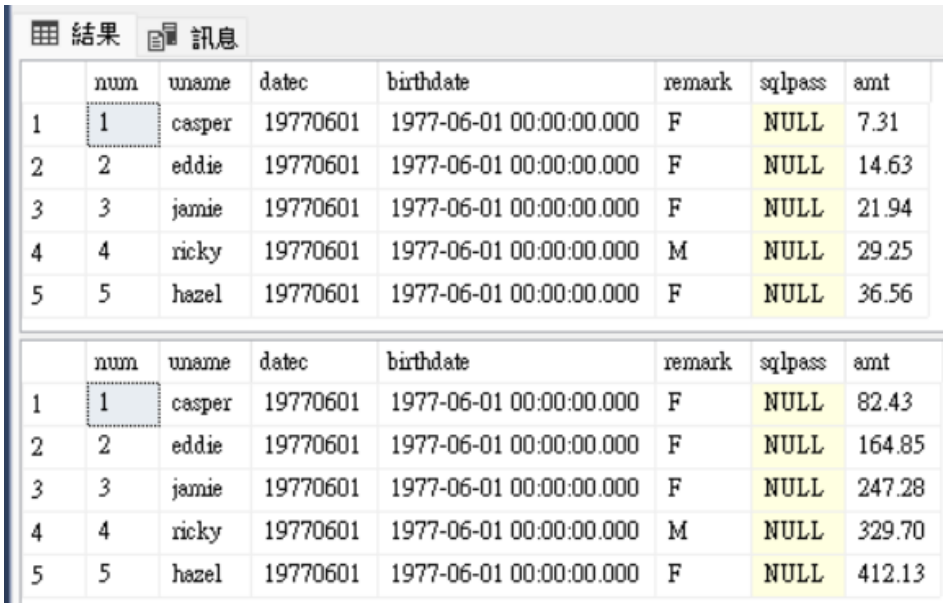
	num	uname	datec	birthdate	remark	sqlpass	amt
1	1	casper	19770601	1977-06-01 00:00:00.000	F	NULL	46.93
2	2	eddie	19770601	1977-06-01 00:00:00.000	F	NULL	93.85
3	3	jamie	19770601	1977-06-01 00:00:00.000	F	NULL	140.78
4	4	ricky	19770601	1977-06-01 00:00:00.000	M	NULL	187.70
5	5	hazel	19770601	1977-06-01 00:00:00.000	F	NULL	234.63

	num	uname	datec	birthdate	remark	sqlpass	amt
1	1	casper	19770601	1977-06-01 00:00:00.000	F	NULL	18.23
2	2	eddie	19770601	1977-06-01 00:00:00.000	F	NULL	36.46
3	3	jamie	19770601	1977-06-01 00:00:00.000	F	NULL	54.70
4	4	ricky	19770601	1977-06-01 00:00:00.000	M	NULL	72.93
5	5	hazel	19770601	1977-06-01 00:00:00.000	F	NULL	91.16

Update 練習:

```
-- 切換到 cgu 資料庫

USE cgu;

GO

-- 檢視更新前的所有資料

SELECT *

FROM cgu_h;

GO

-- 1) 更新 amt 欄位：將 num 乘上亂數(0~1)再乘以 100

UPDATE cgu_h

SET amt = num * RAND() * 100;

GO

-- 2) 根據 uname 內容設定 remark 欄位

--      - 如果 uname 中包含字母 'e'，則標記為 'F'

UPDATE cgu_h

SET remark = 'F'

WHERE uname LIKE '%e%';

--      - 其餘名稱標記為 'M'

UPDATE cgu_h

SET remark = 'M'

WHERE uname NOT LIKE '%e%';

GO

-- 再次檢視更新後的結果

SELECT *
```

```
FROM cgu_h;

GO
```



結果 訊息

	num	uname	datec	birthdate	remark	sqlpass	amt
1	1	casper	19770601	1977-06-01 00:00:00.000	F	NULL	7.31
2	2	eddie	19770601	1977-06-01 00:00:00.000	F	NULL	14.63
3	3	jamie	19770601	1977-06-01 00:00:00.000	F	NULL	21.94
4	4	ricky	19770601	1977-06-01 00:00:00.000	M	NULL	29.25
5	5	hazel	19770601	1977-06-01 00:00:00.000	F	NULL	36.56

	num	uname	datec	birthdate	remark	sqlpass	amt
1	1	casper	19770601	1977-06-01 00:00:00.000	F	NULL	82.43
2	2	eddie	19770601	1977-06-01 00:00:00.000	F	NULL	164.85
3	3	jamie	19770601	1977-06-01 00:00:00.000	F	NULL	247.28
4	4	ricky	19770601	1977-06-01 00:00:00.000	M	NULL	329.70
5	5	hazel	19770601	1977-06-01 00:00:00.000	F	NULL	412.13

Having 練習:

```
-- 1. 切換到 cgu 資料庫

USE cgu;

GO

-- 2. 查詢所有 uname 中包含字母 'e' 的記錄

SELECT *

FROM cgu_h

WHERE uname LIKE '%e%';

GO

-- 3. 查詢所有 uname 中不包含字母 'e' 的記錄

SELECT *

FROM cgu_h

WHERE uname NOT LIKE '%e%';
```

```
GO

-- 4. 依 remark 分組彙總：計算每組的 amt 加總與筆數，並只保留 amt 加總大於 200 的組別

SELECT
    remark,                         -- 標記欄位（如 'F' 或 'M'）
    SUM(amt)    AS total_amt,       -- amt 欄位加總，命名為 total_amt
    COUNT(*)    AS cnt              -- 該組別的筆數
FROM cgu_h
GROUP BY remark
HAVING SUM(amt) > 200;              -- 條件：加總後大於 200
GO
```

結果 訊息

	num	uname	datec	birthdate	remark	sqlpass	amt
1	1	casper	19770601	1977-06-01 00:00:00.000	F	NULL	82.43
2	2	eddie	19770601	1977-06-01 00:00:00.000	F	NULL	164.85
3	3	jamie	19770601	1977-06-01 00:00:00.000	F	NULL	247.28
4	5	hazel	19770601	1977-06-01 00:00:00.000	F	NULL	412.13

	num	uname	datec	birthdate	remark	sqlpass	amt
1	4	ricky	19770601	1977-06-01 00:00:00.000	M	NULL	329.70

	remark	total_amt	cnt
1	F	906.69	4
2	M	329.70	1

union 練習:

```
-- 切換到 cgu 資料庫
USE cgu;
GO
```

```
-- 合併兩次查詢結果，並確保欄位別名一致
SELECT
    uname    AS userid,     -- 使用者帳號
    datec    AS datefm,     -- 註冊日期
    remark AS sex           -- 性別標記
FROM cgu_h
UNION ALL
SELECT
    uname    AS userid,
    datec    AS datefm,
    remark AS sex
FROM cgu_h;
GO
```

結果 訊息

	userid	datefm	sex
1	casper	19770601	F
2	eddie	19770601	F
3	jamie	19770601	F
4	ricky	19770601	M
5	hazel	19770601	F
6	casper	19770601	F
7	eddie	19770601	F
8	jamie	19770601	F
9	ricky	19770601	M
10	hazel	19770601	F

TOP 練習:

```
-- 1) 切換到 cgu 資料庫

USE cgu;

GO

-- 2) 取出 cgu_h 中 amt 最低的 30% 記錄

SELECT TOP 30 PERCENT

    uname,    -- 使用者帳號

    datec,    -- 建檔日期

    remark,   -- 性別或其他標記

    amt       -- 金額欄位

FROM cgu_h

ORDER BY amt ASC;   -- 依 amt 由小到大排序，ASC 表示升冪；若要取最高的 30%，改為 DESC

GO
```

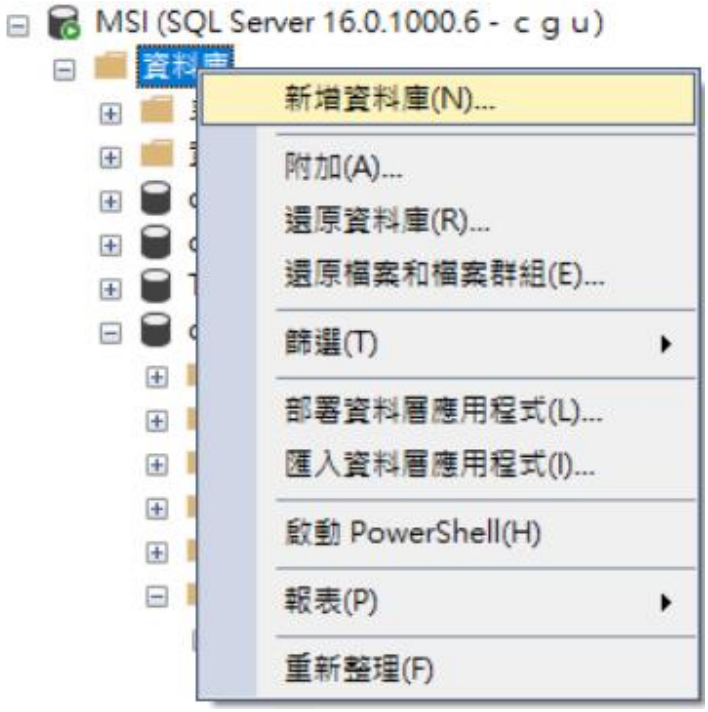
結果 訊息

	uname	datec	remark	amt
1	casper	19770601	F	82.43
2	eddie	19770601	F	164.85

建立實際例子:

實體	屬性 (Attribute)	PK	FK
區域	num	num	業務 → 業務.業務
	區域		
	業務		
	區域件數		
	可賣件數		
	成交件數		
	remark		
	sqlpass		
標的	nos	nos	num → 區域.num 屋主 → 屋主.屋主 業務 → 業務.業務
	地號		
	屋主		
	售價		
	remark		
	sqlpass		
	num		
	區域		
	業務		
業務	num	num	
	業務		
	sqlpass		
屋主	num	num	
	屋主		
	sqlpass		

先建造新的資料庫名叫 cgu_test:



根據上面屬性去建立各自資料表:

區域:

```
-- 1) 切換到 cgu_test 資料庫
USE cgu_test;
GO
-- 2) 如果已存在，先刪除舊的 區域 表
IF OBJECT_ID('dbo.區域', 'U') IS NOT NULL
    DROP TABLE dbo.區域;
GO
-- 3) 建立 區域 主階 表
CREATE TABLE dbo.區域 (
    num          BIGINT          IDENTITY(1,1) NOT NULL,   -- 實際的主鍵，自動遞增
    區域         NVARCHAR(20) NOT NULL,                    -- 場景用的自然鍵
    業務         NVARCHAR(20) NULL,                        -- 外來鍵，對應 業務.業務
    區域件數     INT             NULL,                     -- 該區域紀錄的件數
    可賣件數     INT             NULL,                     -- 該區域可賣件數
    成交件數     INT             NULL,                     -- 該區域成交件數
    remark       NVARCHAR(96)    NULL,                     -- 備註說明
    sqlpass      VARCHAR(254)    NULL,                     -- SQL 密碼或
```

憑證

    CONSTRAINT pk_區域_num PRIMARY KEY (num)   -- 設定 num 為主鍵

);

GO

-- 4) 為 業務 欄位建立索引，加速依業務查詢

CREATE INDEX in_區域_業務

    ON dbo.區域 (業務);

GO

-- 5) 插入測試資料

INSERT INTO dbo.區域 (區域, 業務)

VALUES

    ('A07', N'李'), -- 第一筆

    ('A08', N'陳'); -- 第二筆

GO

-- 6) 插入更多測試資料（一次多筆）

INSERT INTO dbo.區域 (區域, 業務)

VALUES

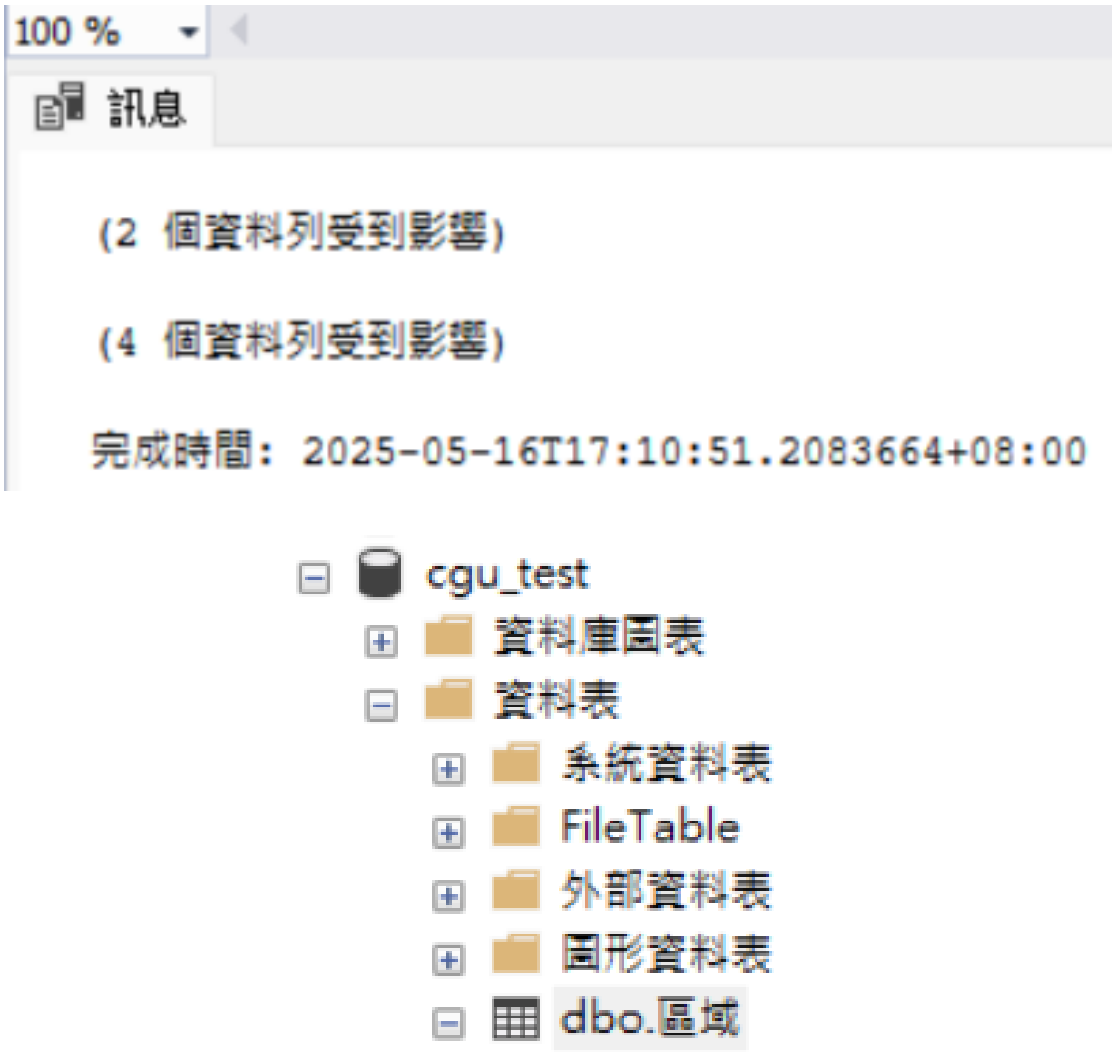
    ('A09', N'李'),

    ('A10', N'陳'),

    ('A11', N'李'),

    ('A12', N'陳');

GO


100 %
訊息
(2 個資料列受到影響)
(4 個資料列受到影響)
完成時間: 2025-05-16T17:10:51.2083664+08:00
cgu_test
資料庫圖表
資料表
系統資料表
FileTable
外部資料表
圖形資料表
dbo.區域

標的:

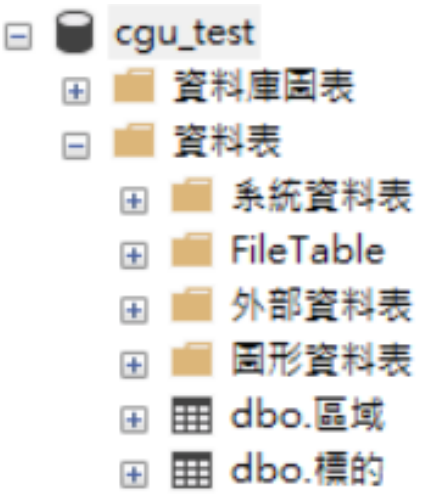
```
-- 1) 切換到 cgu_test 資料庫
USE cgu_test;
GO
-- 2) 若「標的」表已存在，則先刪除以免衝突
IF OBJECT_ID('dbo.標的', 'U') IS NOT NULL
    DROP TABLE dbo.標的;
GO

-- 3) 建立「標的」子階表
CREATE TABLE dbo.標的 (
    nos         BIGINT          IDENTITY(1,1) NOT NULL,   -- 主鍵，自動遞增
    地號        NVARCHAR(20) NULL,                        -- 地號
    屋主        NVARCHAR(20) NULL,                        -- 屋主名稱（語意上的外來鍵）
    售價        INT             NULL,                     -- 售價
    remark      NVARCHAR(96)    NULL,                     -- 備註
    sqlpass     VARCHAR(254)    NULL,                     -- SQL 密碼或憑證
    num         BIGINT          NOT NULL,                 -- 對應 區域.num 的外來鍵
    區域        NVARCHAR(20)    NULL,                     -- 區域碼（冗餘欄位，可同步自 區域 表）
    業務        NVARCHAR(20)    NULL,                     -- 業務人員（外來鍵，同步自 區域 表）
```

```
    CONSTRAINT pk_標的_nos PRIMARY KEY (nos)              -- 主鍵約束
);
GO
-- 4) 為外來鍵欄位建立索引，加速查詢
CREATE INDEX in_標的_num   ON dbo.標的(num);
CREATE INDEX in_標的_區域  ON dbo.標的(區域);
GO

-- 5) 插入測試資料
INSERT INTO dbo.標的 (地號, 屋主, 售價, num)
VALUES
    ('A071', N'林', 777, 1),
    ('A072', N'莊', 888, 1),
    ('A081', N'林', 999, 2),
    ('A082', N'林', 777, 2),
    ('A091', N'林', 888, 3),
    ('A092', N'莊', 999, 3),
    ('A101', N'莊', 777, 4),
    ('A102', N'莊', 888, 4),
    ('A111', N'林', 999, 5),
    ('A112', N'林', 777, 5),
    ('A121', N'林', 888, 6),
    ('A122', N'莊', 999, 6);
GO
```

訊息

(12 個資料列受到影響)

完成時間: 2025-05-16T18:26:24.2006861+08:00



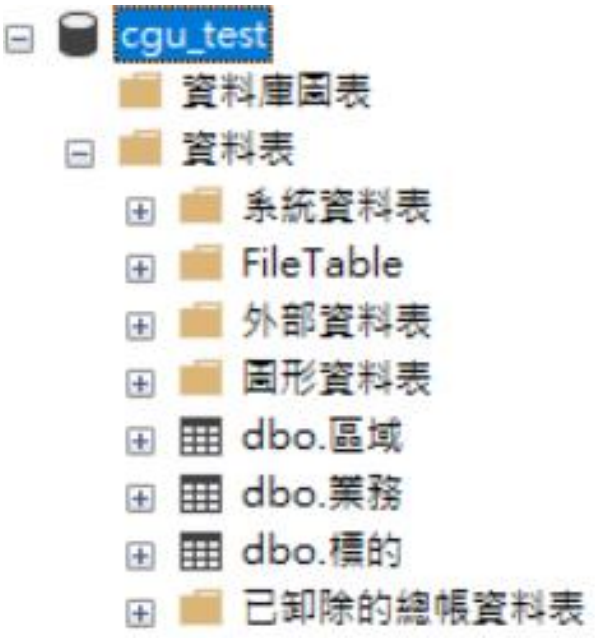
業務:

```
-- 1) 切換到 cgu_test 資料庫
USE cgu_test;
GO
-- 2) 如果已存在「業務」表，先刪除避免衝突
IF OBJECT_ID('dbo.業務', 'U') IS NOT NULL
    DROP TABLE dbo.業務;
GO
-- 3) 建立「業務」主階表
CREATE TABLE dbo.業務 (
    num         BIGINT          IDENTITY(1,1) NOT NULL,   -- 自增主鍵
    業務        NVARCHAR(29) NULL,                        -- 業務人員名稱（自然鍵）
    sqlpass   VARCHAR(254) NULL,                          -- SQL 密碼或憑證
    CONSTRAINT pk_業務_num PRIMARY KEY (num)              -- 指定 num 為主鍵
);
GO
-- 4) 插入測試資料
INSERT INTO dbo.業務 (業務)
VALUES
    (N'李'),   -- 第一筆業務
    (N'陳');   -- 第二筆業務
GO
```

訊息

(2 個資料列受到影響)

完成時間: 2025-05-16T18:30:08.5227853+08:00



屋主:

```
-- 1) 切換到 cgu_test 資料庫
USE cgu_test;
GO
-- 2) 如果已存在「屋主」表，先刪除以免衝突
IF OBJECT_ID('dbo.屋主', 'U') IS NOT NULL
    DROP TABLE dbo.屋主;
GO
-- 3) 建立「屋主」主階表
CREATE TABLE dbo.屋主 (
    num         BIGINT          IDENTITY(1,1) NOT NULL,   -- 自增主鍵：唯一識別每位屋主
    屋主        NVARCHAR(29) NULL,                        -- 屋主名稱（自然鍵，可改 NOT NULL）
    sqlpass   VARCHAR(254) NULL,                        -- SQL 密碼或憑證
    CONSTRAINT pk_屋主_num PRIMARY KEY (num)            -- 指定 num 為主鍵
);
GO
-- 4) 插入測試資料
INSERT INTO dbo.屋主 (屋主)
VALUES
    (N'林'),  -- 第一筆屋主
```

(N'莊'); -- 第二筆屋主

GO


訊息
(2 個資料列受到影響)
完成時間: 2025-05-16T18:42:39.8372910+08:00


cgu_test
資料庫圖表
資料表
系統資料表
FileTable
外部資料表
圖形資料表
dbo.區域
dbo.屋主
dbo.業務
dbo.標的
已卸除的總帳資料表

再對上面設計好的程式做些新增欄位和資料的操作:

```
SELECT * FROM 業務              -- 查詢「業務」表所有記錄
GO

ALTER TABLE 業務
ADD 姓名 NVARCHAR(20)        -- 在「業務」表新增「姓名」欄位，長度 20
GO

SELECT * FROM 區域              -- 查詢「區域」表所有記錄，確認「姓名」欄位已新增

-- 一般都會遵守正規化。
-- 但有時用空間換取時間是值得的。
-- 而且有些情境是應該保留當時的記錄的。
-- 舉例，業務的電話，可能會隨著時間而改變。
-- 若只是遵守正規化的話，將只能顯示最新的電話。


---- 如果想要指定 num 值，需先開啟 IDENTITY_INSERT
--SET IDENTITY_INSERT dbo.業務 ON;

--INSERT INTO dbo.業務 (num, 業務, sqlpass, 姓名)
--VALUES
```

```
--      (3, N'kyujin', NULL, NULL),  -- 第三筆：業務 = kyujin，sqlpass/姓名 為 NULL
--      (4, N'123',      NULL, NULL);  -- 第四筆：業務 = 123，sqlpass/姓名 為 NULL

--SET IDENTITY_INSERT dbo.業務 OFF;
--SELECT * FROM 業務              -- 查詢「區域」表所有記錄，確認加入完成


-- 更新業務代碼與對應的姓名
UPDATE 業務
SET 業務 = N'李', 姓名 = 'ARIAL'
WHERE num = 1                   -- 將 num=1 的記錄設為 業務='李'、姓名 ='ARIAL'
UPDATE 業務
SET 業務 = N'陳', 姓名 = 'BABY'
WHERE num = 2                   -- 將 num=2 的記錄設為 業務='陳'、姓名 ='BABY'
UPDATE 業務
SET 業務 = 'C', 姓名 = 'CAT'
WHERE num = 3                   -- 將 num=3 的記錄設為 業務='C'、姓名='CAT'
UPDATE 業務
SET 業務 = 'M', 姓名 = 'MARY'
WHERE num = 4                   -- 將 num=4 的記錄設為 業務='M'、姓名 ='MARY'
```

GO

結果 訊息

	num	業務	sqlpass	姓名
1	1	李	NULL	ARIAL
2	2	陳	NULL	BABY
3	3	C	NULL	CAT
4	4	M	NULL	MARY

Index(索引):

定義:

Index（索引） 是資料庫為了加速查詢或排序，依照一或多個欄位值建立的資料結構（通常是 B-Tree）。它像書本的目錄，能快速定位到符合條件的資料列，而無須掃描整張表。

執行時機:

- CREATE INDEX：在表建立後、或資料量增大、查詢需求改變時，可手動或自動（維護腳本）新增索引。
- DROP INDEX：當索引不再被查詢使用、對寫入造成過度負擔，或重建過程中需要先行移除。
- ALTER INDEX … REBUILD / REORGANIZE：在索引分散（fragmented）過高時，定期重組或重建，以維持最佳效能。

注意：索引並不會「自動」隨資料變動而被觸發執行，而是在每次查詢（SELECT）、更新（UPDATE）、刪除（DELETE）、插入（INSERT）時，底層引擎會自動維護它們──更新索引頁、插入新鍵或清除舊鍵。

Index 帶來的好處:

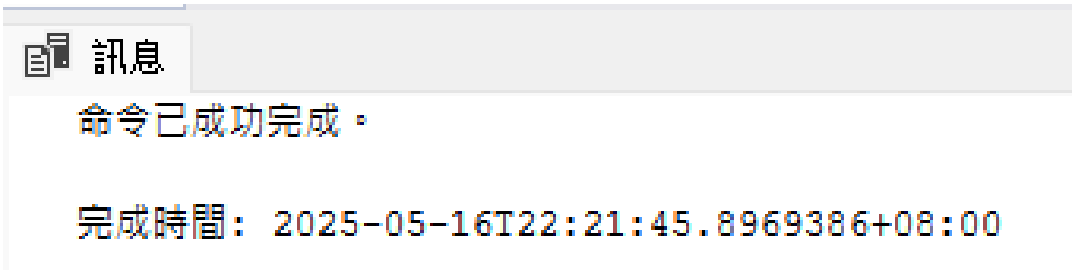
好處	說明
查詢加速	透過索引的 B-Tree 結構，快速定位符合 WHERE、JOIN、ORDER BY、GROUP BY 條件的資料列，避免全表掃描。
排序優化	在需要 ORDER BY 或分頁（OFFSET/FETCH）時，可利用索引的排序順序，省去額外的排序開銷。
唯一性保障	建立唯一索引（UNIQUE INDEX）可強制欄位值不重複，與主鍵／唯一約束相同，進一步確保資料完整性。
篩選效率	在大量資料表中，針對常用的查詢條件建立適當的覆蓋索引（Covering Index），可讓查詢只從索引就取得所有所需欄位，減少回表。

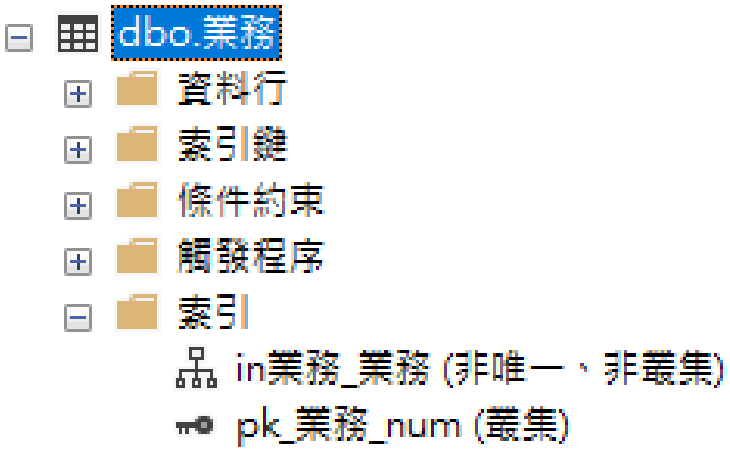
統計資訊	資料庫會針對索引自動收集統計資訊（Statistics），有助優化器選擇最佳執行計畫。
可維護性	索引定義集中於資料庫層，所有查詢共享同一份索引定義；定期 Maintenance（重建/重組）可保持效能穩定。

程式範例:

```
USE [cgu_test]

GO

-- 切換至 cgu_test 資料庫，確保後續動作在此庫中執行


/****** Object:  Index [in 業務_業務]    Script Date: 2025/4/26 上午 09:42:10 ******/

/*

    此區塊為系統自動產生的註解，指出物件類型（Index）、名稱（in 業務_業務）及產生時間

*/


DROP INDEX [in 業務_業務] ON [dbo].[業務]

GO

-- 刪除「業務」表上已存在的非叢集索引 in 業務_業務，以便重新建立或更新索引設定


CREATE NONCLUSTERED INDEX [in 業務_業務]

ON [dbo].[業務]

(
```

```
    [業務] ASC      -- 依「業務」欄位進行升冪排序
)
WITH
(
    PAD_INDEX = OFF,                    -- 不在頁面填滿索引列
    STATISTICS_NORECOMPUTE = OFF,       -- 建立索引時同時更新統計資訊
    SORT_IN_TEMPDB = OFF,               -- 建立過程中不使用 tempdb 進行排序
    DROP_EXISTING = OFF,                -- 不允許同名索引已存在時直接覆蓋
    ONLINE = OFF,                       -- 建立索引時鎖定表，不支援線上重建
    ALLOW_ROW_LOCKS = ON,               -- 允許列鎖
    ALLOW_PAGE_LOCKS = ON,              -- 允許頁鎖
    OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF -- 不針對遞增鍵優化
)
ON [PRIMARY]    -- 將索引放在 PRIMARY 檔案群組
GO
-- 建立完成後，該索引將加速針對「業務」欄位的查詢與排序效能
```




dbo.業務
資料行
索引鍵
條件約束
觸發程序
索引
in業務_業務 (非唯一、非叢集)
pk_業務_num (叢集)

UDT(user design data type):

定義:

UDT（使用者定義資料型別）是由資料庫管理員或開發者基於內建型別，定義出專案或企業層面頻繁重複使用的資料型別別名。它本質上包裹了一個或多個內建型別，並可加上預設值、允許 NULL／NOT NULL 等屬性。

執行時機:

- **欄位宣告**：在 CREATE TABLE 或 ALTER TABLE … ADD COLUMN 時，用 UDT 替代內建型別，讓資料表欄位宣告更具可讀性與一致性。
- **參數／變數宣告**：在儲存程序、函式或腳本中，用 UDT 宣告變數或輸入參數，使程式維護與跨專案重用更方便。
- **預設與規範**：若多個欄位都要共享相同的長度、NULL 規則或預設值，可透過 UDT 一次定義，全域套用。

UDT 帶來的好處:

好處	說明
一致性	全庫／跨專案使用同一份型別定義，保證所有相同語意的欄位都遵循相同長度、精度與 NULL 規則。
集中管理	型別修改時，只需在 UDT 定義上變更一次，各資料表與程式自動繼承，降低分散改動的維護成本。
可讀性	以具業務意義的別名命名（如 udt_PhoneNumber、udt_Email），程式碼或資料模型結構更易理解。
DRY 原則	不必在每張表、每個參數都重複寫相同的內建型別與屬性，減少重複程式碼（Don't Repeat Yourself）。
規範化擴充	日後若要改變欄位長度或預設值，只要更新 UDT，所有使用該型別的欄位自動生效。
減少錯誤	手動為多張表同步改動欄位型別容易漏掉某些表，UDT 能避免此類版本不一致導致的資料錯誤或跑版問題。

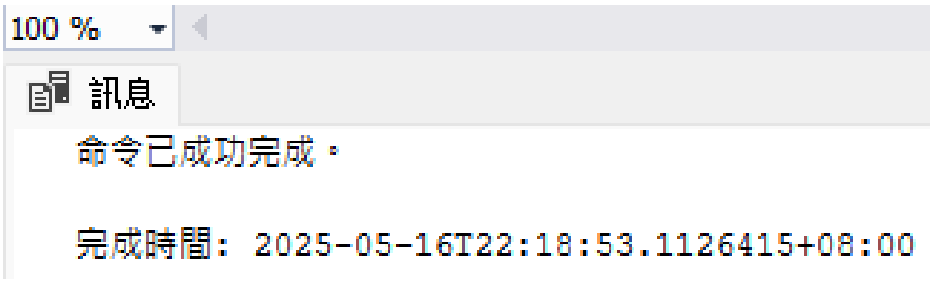
程式範例:

```
use cgu_test

go

DROP TYPE udt_業務

GO

CREATE TYPE udt_業務  FROM nvarchar(20) null;--

go
```

```
命令已成功完成。

完成時間: 2025-05-16T22:16:44.0786918+08:00
```

```
ALTER TABLE  業務

ALTER COLUMN  業務  udt_業務       -- 將「業務」表的「業務」欄位資料型別改為自訂型別  udt_業務

GO


ALTER TABLE  區域

ALTER COLUMN  業務  udt_業務       -- 將「區域」表的「業務」欄位資料型別改為自訂型別  udt_業務

GO


ALTER TABLE  標的

ALTER COLUMN  業務  udt_業務       -- 將「標的」表的「業務」欄位資料型別改為自訂型別  udt_業務
```

GO



Function:

定義:

Function（函式） 是一段可重複呼叫的 T-SQL 程式片段，它接受參數、執行運算或查詢，並回傳單一值或表格結果。與儲存程序（Stored Procedure）不同，Function 可以在 SQL 句中直接當作表達式或資料來源使用。

執行時機:

- Scalar Function（純量函式）
  - 在任何可放置表達式的地方執行：SELECT uf_函式(a, b), …、WHERE uf_函式(x) > 10 等。
- able-valued Function（表值函式）
  - 類似一張表，可在 FROM、JOIN 中使用：

    SELECT *

    FROM dbo.uf_取房價(‘A07’) AS t

注意：Function 只能透過顯式呼叫執行，無法自動「觸發」。

Function 帶來的好處:

好處	說明
邏輯封裝	將常用計算或查詢封裝成函式，呼叫端只要一行，就能複用相同邏輯，降低重複程式碼。
可組合性	可以在 SQL 查詢中當作表達式（純量函式）或資料來源（表值函式）直接串接，靈活打造複雜報表。
參數化	支援輸入參數，依不同參數執行不同邏輯；回傳值也可做為下一層 SQL 運算的依據。
效能快取	資料庫會編譯並快取函式執行計畫，重複呼叫時可加速，但須留意勿在批次裡做大量 row-by-row 呼叫，以免效能下降。
安全性	可設定執行權限，呼叫者只要有執行函式的權限，不需直接存取底層表。
易於維護	變更邏輯只要修改函式定義，所有呼叫端立即同步受益，不需分散到各應用程式或查詢中。

程式範例:

```
USE cgu_test                    -- 切換到 cgu_test 資料庫
GO

-- 刪除已存在的函式 uf_業務_姓名，以避免重複建立時出錯
DROP FUNCTION [dbo].[uf_業務_姓名]
GO

-- 建立自訂函式 uf_業務_姓名
CREATE FUNCTION [dbo].[uf_業務_姓名]
(
    @業務 udt_業務          -- 輸入參數：業務代碼，使用自訂資料型別 udt_業務
)
RETURNS NVARCHAR(36)        -- 回傳值：業務完整姓名，最長 36 個字元

AS
BEGIN
    DECLARE @姓名 NVARCHAR(36);    -- 宣告區域變數，用以暫存查到的姓名

    -- 從 業務 表撈出對應 @業務 代碼的 姓名，並賦值給 @姓名
    SELECT @姓名 = 姓名
```

```
    FROM 業務
    WHERE 業務 = @業務;

    -- 如果未找到對應姓名（NULL 或空字串），則預設為「無名氏」
    IF ISNULL(@姓名, '') = ''
        SET @姓名 = '無名氏';

    RETURN @姓名;  -- 回傳最終的姓名字串
END
GO
```

Trigger:

定義:

Trigger（觸發器）是一段與表格綁定的程式，會在指定的 DML 事件（INSERT、UPDATE、DELETE）發生後自動執行，不需由應用程式顯式呼叫。

執行時機:

- AFTER UPDATE / INSERT / DELETE：在資料變更真正寫入表後觸發。
- INSTEAD OF …：取代原本的變更動作，先執行 Trigger 內容，再決定是否要回寫表。

Trigger 帶來的好處:

好處	說明
集中管理	把跨表更新或驗證邏輯都寫在 Trigger，避免散落在各種應用程式或批次中，降低維護成本。
即時性	當資料一改動，Trigger 立即生效，不須等待排程或手動執行。
一致性	不管是用哪個工具（程式、報表、管理介面）修改資料，都會觸發相同的邏輯，避免資料「語意」不一致。
防呆	禁止未經授權或未符合條件的更新。可以在 Trigger 內檢查並回報錯誤，甚至 Rollback 整個交易。
減少客戶端程式碼	由資料庫層自動完成後續處理，前端／中介層程式只需專注在發出 DML 指令，不用再額外撰寫同步邏輯。

程式範例:

```
-- 1) 切換到 cgu_test 資料庫

USE cgu_test;
```

```
GO

-- 2) 刪除舊的觸發器（如果已存在）
IF OBJECT_ID('dbo.tr010 標的', 'TR') IS NOT NULL
    DROP TRIGGER dbo.tr010 標的;
GO

-- 3) 建立觸發器：當「標的」表的屋主欄位被更新時，同步更新「區域」表的統計欄位
CREATE TRIGGER dbo.tr010 標的
ON dbo.標的
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    -- 只對單筆更新生效
    IF @@ROWCOUNT > 1
        RETURN;

    -- 只有在「屋主」欄位被更新時才繼續
    IF NOT UPDATE(屋主)
        RETURN;
```

```
DECLARE @num BIGINT;
DECLARE @可賣件數 INT, @成交件數 INT, @區域件數 INT;

-- 從 INSERTED 暫存表取得剛更新那筆記錄的區域 num
SELECT TOP 1 @num = num
FROM inserted;

-- 計算「可賣件數」：同一區域、屋主為空字串或 NULL
SELECT @可賣件數 = COUNT(*)
FROM dbo.標的
WHERE num = @num
  AND ISNULL(屋主, '') = '';

-- 計算「成交件數」：同一區域、屋主非空
SELECT @成交件數 = COUNT(*)
FROM dbo.標的
WHERE num = @num
  AND ISNULL(屋主, '') <> '';

-- 計算「區域件數」＝可賣件數 + 成交件數
SET @區域件數 = ISNULL(@可賣件數, 0) + ISNULL(@成交件數, 0);

-- 更新「區域」表對應記錄
UPDATE dbo.區域
```

```
    SET
        區域件數 =@區域件數,
        可賣件數 =@可賣件數,
        成交件數 =@成交件數
    WHERE num = @num;

    SET NOCOUNT OFF;
END;
GO
```

```
-- 4) 測試觸發器生效前後：
--      （以下示範查詢與更新，可取消註解執行）
/*
SELECT * FROM dbo.區域  WHERE num = 6;
SELECT * FROM dbo.標的  WHERE num = 6;
-- 將同區域中某兩筆標的的屋主設為空，觸發 AFTER UPDATE
UPDATE dbo.標的  SET  屋主  = '' WHERE nos IN (11, 12);
-- 再次檢視結果
SELECT * FROM dbo.區域  WHERE num = 6;
SELECT * FROM dbo.標的  WHERE num = 6;
*/
```

100 %

結果 訊息

	num	區域	業務	區域件數	可賣件數	成交件數	remark	sqlpass
1	6	A12	陳	NULL	NULL	NULL	NULL	NULL

	nos	地號	屋主	售價	remark	sqlpass	num	區域	業務
1	11	A121	林	888	NULL	NULL	6	NULL	NULL
2	12	A122	莊	999	NULL	NULL	6	NULL	NULL

	num	區域	業務	區域件數	可賣件數	成交件數	remark	sqlpass
1	6	A12	陳	2	2	0	NULL	NULL

	nos	地號	屋主	售價	remark	sqlpass	num	區域	業務
1	11	A121		888	NULL	NULL	6	NULL	NULL
2	12	A122		999	NULL	NULL	6	NULL	NULL

程式範例 2:

用途:

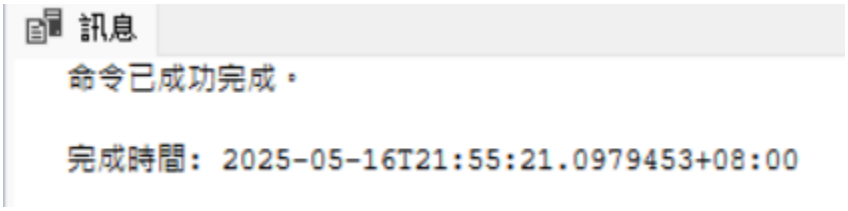
自動同步：每次有人新增或修改 區域.業務 欄位，系統都會自動幫你把對應的 姓名 更新好，不用手動再去跑一支 UPDATE。

集中維護：把「業務代碼 → 業務姓名」的映射邏輯寫在資料庫層（uf_業務_姓名 函式＋觸發器），前端程式或其他使用者都不必重複實作。

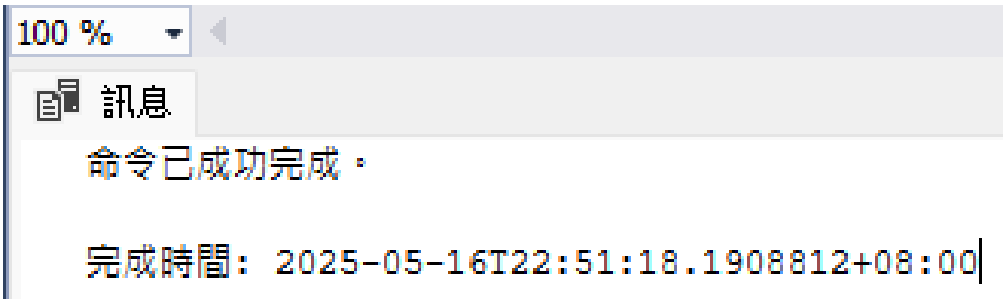
資料一致性：確保 區域 表裡的 姓名 欄位永遠與最新的 業務 代碼對應結果一致，避免因為忘了更新而出現錯誤資料。

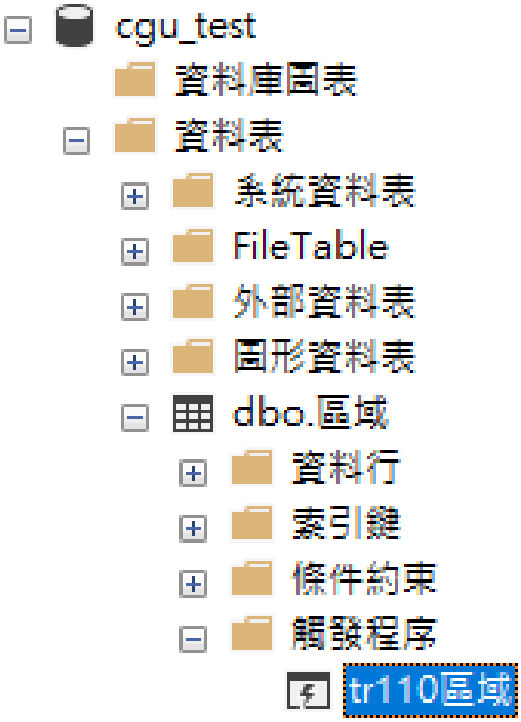
範例程式碼:

```
USE [cgu_test]                                    -- 切換到 cgu_test 資料庫
GO

-- 以下示範如何批次更新「業務」欄位的值：
-- update 區域 set 業務 = 'M' WHERE 業務 = N'D'    -- 將業務名稱為「D」的記錄改為「M」
-- GO
```



```
-- 刪除已存在的觸發器 tr110 區域，避免重複建立
DROP TRIGGER [dbo].[tr110 區域]
GO

CREATE TRIGGER [dbo].[tr110 區域]
ON [dbo].[區域]
AFTER INSERT, UPDATE                    -- 指定觸發時機：插入或更新完成後
AS
BEGIN
    SET NOCOUNT ON;                     -- 關閉「N rows affected」訊息，以提升效能

    -- 使用 inserted 暫存表，將剛新增或更新的每筆記錄，
    -- 呼叫函式 dbo.uf_業務_姓名，取得業務欄位對應的姓名，並寫回「姓名」欄位
    UPDATE 區域
    SET 姓名 = dbo.uf_業務_姓名(inserted.業務)
    FROM inserted
    WHERE 區域.num = inserted.num;      -- 以主鍵 num 連結原表與 inserted 暫存表
END
GO
```





-- 以下為測試用查詢與更新，可取消註解檢視或修改區域表

-- select * from 區域 -- 查詢所有區域記錄

-- UPDATE 區域 set sqlpass = '123' -- 將所有區域記錄的 sqlpass 欄位設為 '123'

-- 建立觸發器：在 dbo.區域 表執行 INSERT 或 UPDATE 後，自動呼叫函式 uf_業務_姓名 更新「姓名」欄位

Stored Procedure :

主要用意:

1. 邏輯集中化
   把複雜的查詢、計算、資料操作流程集中在資料庫層，不必散落在各種應用程式或報表中。
2. 重複使用性
   定義一次，就能由多個應用程式或使用者重複呼叫，無需重複撰寫相同邏輯。
3. 參數化控制
   支援輸入／輸出參數，可根據不同需求傳入不同值動態運算，靈活性高。
4. 效能最佳化
   資料庫會將程序編譯成執行計畫並快取，下次呼叫即重複使用相同計畫，減少重複解析、編譯的成本。
5. 安全性管理
   可設定授權權限，只開放執行權而不必暴露底層表結構；也能在程序內做權限檢查、錯誤處理。

Stored Procedure 常見應用場景:

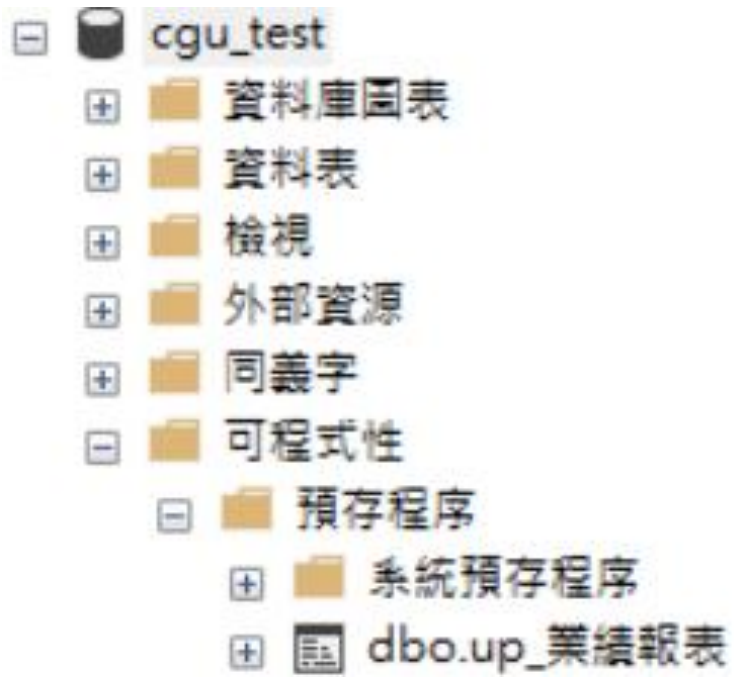
1. 批次處理：定期執行資料匯總、歸檔、報表產生等作業。
2. 資料檢核：集中寫入驗證邏輯，確保欄位值或關聯一致性。
3. 複雜運算：多表 JOIN、迴圈、游標處理等需逐筆計算的邏輯，常以程序實現。
4. API 後端：以資料庫程序為核心，外層應用程式只負責傳遞參數與呈現結果。

程式範例:

```
-- 1) 切換到 cgu_test 資料庫

USE cgu_test;

GO
```

```
-- 2) 如果已存在同名儲存程序，則先刪除
IF OBJECT_ID('dbo.up_業績報表', 'P') IS NOT NULL
    DROP PROCEDURE dbo.up_業績報表;
GO
-- 3) 建立「業績報表」儲存程序
CREATE PROCEDURE dbo.up_業績報表
AS
BEGIN
    SET NOCOUNT ON;  -- 關閉「已影響列數」訊息，提高效能

    -- 4) 定義暫存表：存放業務、地號與計算後的獎金
    DECLARE @業績 TABLE (
        業務   NVARCHAR(20),
        地號   NVARCHAR(20),
        獎金   INT
    );

    -- 5) 將「標的」表中已成交（屋主不為空）的記錄匯入暫存表
    INSERT INTO @業績 (業務, 地號)
    SELECT 業務, 地號
    FROM dbo.標的
    WHERE ISNULL(屋主, '') <> '';

    -- 6) 根據地號進行獎金計算：取地號第 2~4 字元轉為整數後乘以 100
```

```
    UPDATE @業績
    SET  獎金  = CONVERT(INT, SUBSTRING(地號, 2, 3)) * 100;


    -- 7) 回傳結果
    SELECT  業務, 地號, 獎金
    FROM @業績;
END;
GO
-- 8) 測試儲存程序
--  查看原始「標的」資料
SELECT * FROM dbo.標的;
GO
--  執行業績報表
EXEC dbo.up_業績報表;
GO
```



結果 訊息

	nos	地號	屋主	售價	remark	sqlpass	num	區域	業務
1	1	A071	林	777	NULL	NULL	1	NULL	NULL
2	2	A072	莊	888	NULL	NULL	1	NULL	NULL
3	3	A081	林	999	NULL	NULL	2	NULL	NULL
4	4	A082	林	777	NULL	NULL	2	NULL	NULL
5	5	A091	林	888	NULL	NULL	3	NULL	NULL
6	6	A092	莊	999	NULL	NULL	3	NULL	NULL
7	7	A101	莊	777	NULL	NULL	4	NULL	NULL
8	8	A102	莊	888	NULL	NULL	4	NULL	NULL
9	9	A111	林	999	NULL	NULL	5	NULL	NULL
10	10	A112	林	777	NULL	NULL	5	NULL	NULL
11	11	A121		888	NULL	NULL	6	NULL	NULL
12	12	A122		999	NULL	NULL	6	NULL	NULL

	業務	地號	獎金
1	NULL	A071	7100
2	NULL	A072	7200
3	NULL	A081	8100
4	NULL	A082	8200
5	NULL	A091	9100
6	NULL	A092	9200
7	NULL	A101	10...
8	NULL	A102	10...
9	NULL	A111	11...
10	NULL	A112	11...

點擊資料庫圖表右鍵，可以新增關係圖: