SQL Server開發進階

1

課程大綱

1. SQL Server架構
2. 設計資料表
3. 資料完整性(Data Integrity)
4. 進階資料庫設計
5. 索引(Index)
6. 檢視(View)
7. T-SQL流程控制語法
8. 交易(Transaction)
9. 預存程序(Stored Procedure)
10. 函數(Function)

2

1. SQL Server架構

3

SQL Server資料庫引擎

SQL Server資料庫引擎是由以下兩個部分所組成

1. 儲存引擎(Storage Engine)
2. 查詢處理器(Query Processor)

4

SQL Server企業級的平台

- 符合企業使用
  - 符合常規工業標準。在伺服器評定效能組織網站(Transaction Processing Performance Council:TPC)可以檢視SQL Server資料庫引擎的評定分數。
- 高可用性(High Availability)
- 安全性(Security)
- 可擴展性(Scalability)
- 低成本

5

SQL Server元件

元件	描述
SQL Server資料庫引擎(Database Engine)	基於結構化查詢語言(SQL)的關聯式資料庫語言。
Analysis Services(SSAS)	線上分析系統(OLAP)引擎，用以建立分析維度，以及資料採礦工具。
Reporting Services(SSRS)	提供網頁式介面，讓使用者可以設計報表的工具。
Integration Services(SSIS)	Integration Services 是一組圖形化工具和可程式化物件，可用於移動、複製和轉換資料。
Master Data Services(MDS)	Master Data Services (MDS) 是用於主要資料管理的 SQL Server 解決方案
Data Quality Services(DQS)	DQS 可讓使用參考資料提供者所提供的雲端式 Reference Data Services，執行資料清理。
Machine Learning 服務	Machine Learning 服務 (資料庫內) 支援使用企業資料來源之可調整的分散式 Machine Learning 解決方案。
使用 PolyBase 的資料虛擬化	從SQL Server查詢不同類型的資料來源資料類型。
Azure 連線服務	SQL Server 2022 (16.x)擴充Azure連線服務和功能。

6

SQL Server管理工具

工具	描述
SQL Server Management Studio(SSMS)	整合式管理環境，讓開發者以及資料庫管理者管理核心的資料庫引擎。
SQL Server Configuration Manager(SSCM)	管理SQL Server相關服務的基本設定、用戶端和伺服器端連線的通訊協定以及用戶端的別名。
SQL Server Profiler	圖形使用者介面，用以監控輔助資料庫引擎、Analysis Service元件效能調教及管理工作。
Database Engine Tuning Advisor	提供協助建立、優化索引、索引檢視(Indexed View)、分割資料表的輔助工具。
Data Quality Services Client	圖形使用者介面，用以連線DQS伺服器並提供資料清理工作(Data Cleansing Operations)並監控效能。
SQL Server Data Tools(SSDT)	整合式開發環境，用以提供商業智慧解決方案，應用於SSAS, SSRS, SSIS中。
Connectivity Components	用以溝通用戶端以及伺服器端之間的元件。例如: ODBC及OLE DB

7

SQL Server執行個體

- 有時候安裝多個SQL Server執行個體在同一台伺服器中，對於管理、開發SQL Server是有幫助的。
- 在一個伺服器中的每一個SQL Server服務稱之為執行個體(Instance)。
- 所有執行個體中，只能有一個執行個體為預設執行個體(Default Instance)，其它的執行個體都必須為其命名，稱之為具名執行個體(Named Instance)。
- 有些元件只能安裝一次，共用於多個執行個體之中，像是SSIS(SQL Server Integrated Service)。

8

多個執行個體

- 在以下的幾個狀況下，可以考慮安裝多個執行個體：
- 資料庫需要不同的管理者以及安全性環境。
- 應用程式需要不同SQL Server的設定，而這些設定本身互斥。
- 應用程式需要不同等級的服務，特別是和可用性有關。
- 必須支援不同版本(Version)的SQL Server。
- 應用程式需要伺服器等級的定序。

9

SQL Server 功能性版本(Edition)

- 在以下的幾個狀況下，可以考慮安裝多個執行個體：
- 資料庫需要不同的管理者以及安全性環境。
- 應用程式需要不同SQL Server的設定，而這些設定本身互斥。
- 應用程式需要不同等級的服務，特別是和可用性有關。
- 必須支援不同版本(Version)的SQL Server。
- 應用程式需要伺服器等級的定序。

10

SQL Server 功能性版本(Edition)

- 企業版(Enterprise) - SQL Server Enterprise Edition提供完整的高階資料中心功能。
- 標準版(Standard) - 針對部門和小型組織提供基本的資料管理資料庫。
- 開發版(Developer) - 可讓開發人員在 SQL Server上建立任何類型的應用程式。
- Express - SQL Server Express Edition是入門級免費伺服器。
- Azure® SQL Database - 提供雲端上的平台建立應用程式使用的資料庫。

11

SQL Server 歷年版本(Version)

- 早期的版本 - SQL Server最早的版本(1.0, 1.1版)是在OS/2作業系統上。直至4.2版才移植到Windows作業系統，最早是建立在Windows NT作業系統。
- 之後的版本

  - SQL Server 7.0、SQL Server 2000(8.0)、SQL Server 2005(9.0)、SQL Server 2008(10.0)、SQL Server 2008 R2(10.5)、SQL Server 2012(11.0)、SQL Server 2014(12.0)、
  SQL Server 2016(13.0)、SQL Server 2017(14.0)、 SQL Server 2022 (15.0)
- 目前最新的版本 - SQL Server 2022(16.0)

12

SQL Server 開發工作

- 熟悉常用的開發工具
- 儲存以及操作資料
- 程式化處理資料
- 確保及檢查資料品質
- 資料安全性

13

心得與討論

14

2. 設計資料表

15

什麼是資料表(Table)

- 關聯式資料庫中，資料表示儲存資料的地方。
- 資料表由欄位(Column)與資料列(Row)組合而成。
- 每一個資料列代表一個實體(Entity)，而欄位則是實體的屬性(Attribute)。
- 預設的情況下，資料列為最小安全性邊界。

16

命名規則(Naming Rules)

- **第一個字元必須是以下任一項：**
  - Unicode Standard 3.2 所定義的字母。
  - 底線 (_)、@ 符號或數字符號 (#)。
- **可包含的後續字元如下列：**
  - Unicode Standard 3.2 所定義的字母。
  - 其他基本拉丁文或其他國家 (地區) 字集中的十進位數字。
  - @ 記號 (@)、貨幣符號 ($)、數字記號 (#) 或底線 (_)。
- **不可以是 Transact-SQL 保留字**
  - 如果識別碼與上述規則不符，必須使用雙引號 ("") 或方括號 ([]) 加以分隔。
- **不允許內嵌的空格、特殊字元或補充字元。**

17

命名慣例(Naming Convention)

- **統一規範命名法則：**
  - 全大寫(例如:HRMS)
  - 全小寫(例如:employee)
  - Pascal(例如: EmployeeSalary)
  - Snake(例如: employee_id)
- **不使用前綴方式命名(例如: tblEmployee, vwEmployeeInfo, spCalculateSalary)**
- **避免使用簡稱**
- **不強調單複數**

18

資料型別

- 資料型別決定資料庫中可以放什麼資料，像是欄位、變數及參數都有資料型別。例如：tinyint欄位只能存放 0 到 255 的數值。
- 資料行也會決定運算式會回傳怎樣的值回來。
- 資料型別也是一種限制什麼值可以寫入資料庫的束制條件。

19

資料型別

常見的資料型

- 數值(bit, tinyint, smallint, int, bigint, real, float, decimal, numeric)
- 日期時間(smalldatetime, datetime, date, time, datetime2, datetimeoffset)
- 文字(char, varchar, nchar, nvarchar)
- 其它(binary, varbinary, uniqueidentifier, geometry, geography, xml…)

20

各種資料型別的限制-整數

- tinyint – 佔1 byte，數值範圍：0 至 255。
- smallint - 佔2 bytes，數值範圍：-32,768 至 32,767。
- int - 佔4 bytes - 數值範圍: -2,147,473,648 至 2,147,473,647。是最常用的資料型別。SQL Server使用integer為int的同義字。
- bigint - 8 bytes，數值範圍: -2^{63} 至 $2^{63}-1$(大約是 $\pm 9.22 \times 10^{18}$，18位數的整數)。

21

各種資料型別的限制-小數點(精確值)

- decimal – 是ANSI標準資料型別，可以指定精確值位數與小數位數。
  - 例如：decimal(5, 2) 代表精確值為數為5位，小數位數2位(也就是，3位整數, 2位小數範圍在±999.99)。精確值最多可以到38位數。
  - 系統金融貨幣相關的資料型別，多半會使用decimal。
  - numeric和decimal相同
- smallmoney - int型別, 固定4位小數
- money - bigint型別, 固定4位小數

22

各種資料型別的限制-小數點(浮點數)

- real - 佔4 bytes是ISO標準浮點數型別。
- float - SQL Server特有的資料型別，可以是4 bytes或8 bytes。

23

各種資料型別的限制-日期時間

- date -符合標準ANSI SQL中的陽曆(Gregorian calendar)定義。預設字串的格式為YYYY-MM-DD，這個格式是ISO 8601的DATE定義。範圍為 0001-01-01 至 9999-12-31。
- time -遵循SQL標準預設的格式為hh:mm:ss.nnnnnnn。範圍為 0:0:0 至 23:59:59.9999999。
- datetime2 – date + time
- datetimeoffset – datetime2 + 時區(例如: '2024-09-30 12:30+08:00')
- datetime – 是舊型的日期時間型別, 較datetime2的型別範圍較小，精度也較低。
- smalldatetime –是舊型的日期時間型別, yyyy-MM-dd hh:mm(精度到分鐘)

24

各種資料型別的限制-文字

- char、varchar、nchar、nvarchar

  - 傳統電腦系統中，字元會以1 byte儲存，一個字元只能有256種字元符號。這對很多語言是不夠的。因而SQL Server可以利用編碼(Code Page)將語言資料儲存到電腦中(Encoding)，也以相同的編碼將資料由電腦中取出顯示於畫面上(Decoding)。但是在此同時，資料的編碼不同，可能寫入，或讀出的資料無法正確解譯，就會變成?而造成資料的遺失。char、varchar就是以這種方式儲存資料。1980有學者研究將地球上的語言可以使用3個bytes儲存，但是這電腦儲存不是一個很理想的數字，於是就做了些簡化而有了 UTF-7、UTF-8、UTF-16以及UTF-32(UTF: Universal Text Format, 全球文字格式)。SQL Server使用UTF-16格式，資料型別nchar、nvarchar使用這種格式儲存。在字元常數值前加上N(National Character Set)表以Unicode方式解譯文字(例如：N’ 中文字’)。

25

建立資料表

```
CREATE TABLE dbo.Employee
(
    employee_id int NOT NULL,
    employee_name nvarchar(20),
    birthdate date,
    PRIMARY KEY(employee_id)
)
```

26

修改資料表

```
ALTER TABLE dbo.Employees
ADD phone nvarchar(20), address nvarchar(20);
GO
ALTER TABLE dbo.Employees
ALTER COLUMN address nvarchar(50);
GO
ALTER TABLE dbo.Employees
DROP COLUMN phone, address;
GO
```

27

移除資料表

```
DROP TABLE IF EXISTS dbo.Employee
GO
```

28

暫存資料表

暫存資料表用於儲存使用者連線中的暫時資料。儲存於tempdb中，並且在連線中斷後自動刪除。暫存資料表和一般的資料表很相似，除了暫存資料表只能存在於建立者的連線中，並且在連線中斷時自動被刪除。但是在不需要使用時也可以自行將暫存資料表移除，以降低系統資源的使用。暫存資料表可以利用SELECT INTO語法建立。

29

心得與討論

30

3. 資料完整性(Data Integrity)

31

甚麼是資料完整性(Data Integrity)

- 資料完整性是指儲存在資料庫中資料的一致性以及正確性。
- 資料庫中資料的品質決定了應用程式的使用與效率。企業組織活動的決策好壞取決於這個資料品質。確保資料完整性是維護高品質資料很重要的步驟。
- 應用程式在每一個寫入資料的階段都必須確保資料的完整性。Microsoft SQL Server®資料管理軟體提供了一系列功能簡化了這個工作。

32

應用程式架構

應用程式通常採用三層式架構(three-tier architecture)。讓同類型的工作整合在同一層方便程式碼的管理，也讓程式碼有較多可以再利用的機會。通常應用程式會有以下這幾層：

- 使用者介面層(Presentation Tier)
- 中間層 – 或稱為商業邏輯層(Business Logic Tier)
- 資料層(Data Tier)

33

使用者介面層:展現層

在使用者介面層強制資料完整性檢查有幾個好處。

- 使用者可以直接面對操作上的問題，也避免其它層的錯誤。
- 錯誤訊息也可以很明確地傳達給使用者。

缺點是因為可能有不同的應用程式使用底層的資料，而各應用程式的商業邏輯各不相同，因此在資料層需要寫更多程式碼以檢查資料完整性。

34

商業邏輯層

許多資料完整性問題會在商業邏輯中處理，不同於非功能性資料完整性檢查(像是資料格式正確性)。商業邏輯層可以在多個應用程式中使用，讓程式可以提高再用性。

在商業邏輯層做資料完整性檢查，可以避免不同的使用者介面用不一樣的商業邏輯做資料檢查。商業邏輯層的邏輯和使用者的功能較容易理解，因此錯誤訊息也容易讓使用者了解。

35

資料層

資料層做資料檢查的好處是每一層都不能跳過這層檢查。特別是多個應用程式存取資料時無法保證資料的品質，如果不在資料層檢查資料完整性，必須在個別的程式中確保資料的正確。

缺點是在資料層處理資料完整性時，使用者的動作發生錯誤，也同時發生在資料層中，除非程式設計師有適當的處理錯誤的發生，一般人無法理解這個錯誤訊息。實際操作上，在資料層產生的錯誤必須經由上層的程式碼處理過才能傳達給使用者。

36

關聯式資料庫的資料完整性

資料完整性(Data Integrity)有三個基本形式：

- 定義域完整性(Domain Integrity)
- 實體完整性(Entity Integrity)
- 參考完整性(Referential Integrity)

37

定義域完整性(Domain Integrity)

SQL Server限制了定義域(或是欄位)可以輸入的資料以及是否允許空值(null)。

例如：一個欄位只能輸入數值，不能輸入英文字母，可以指定為int型別。相同地，tinyint型別限制欄位只能儲存0到255的數字。

check束制條件限制資料可以輸入的內容。

default可以在沒有輸入值時帶入預設值。

not null可以限制資料不得為null

38

實體完整性(Entity Integrity)

實體或資料表完整性確保資料表每一筆資料可以單獨的以欄位識別。

這個欄位(也包含複合鍵值的多個欄位)也就是所謂的主索引鍵(Primary Key)。主索引鍵可以作為和其它資料表關聯的基礎，這也是接下來要提的參考完整性。

PRIMARY KEY以及UNIQUE束制條件可以限制資料的唯一性。

39

參考完整性(Referential Integrity)

參考完整性確保資料關聯的主索引鍵和外部索引鍵之間的一致性。

建立關聯的資料表中，被關聯的資料表不存在的值，無法在關聯資料表中新增；關聯的資料表中已經存在的值，被參考的資料表中不允許刪除或修改，除非有設定關聯的動作。

參考完整性的實例，不能在訂單資料中新增一筆不存在的客戶。

40

建立束制條件-方法一

```
CREATE TABLE dbo.PK01
(
id int PRIMARY KEY,
data nvarchar(20)
);
```

41

建立束制條件-方法二

```
CREATE TABLE dbo.PK02
(
id int CONSTRAINT PK_PK02 PRIMARY KEY,
data nvarchar(20)
);
```

42

建立束制條件-方法三

```
CREATE TABLE dbo.PK03
(
id int,
data nvarchar(20),
PRIMARY KEY(id)
);
```

43

建立束制條件-方法四

```
CREATE TABLE dbo.PK04
(
id int,
data nvarchar(20),
CONSTRAINT PK_PK04 PRIMARY KEY(id)
);
```

44

建立束制條件-方法五

```
CREATE TABLE dbo.PK05
(
id int NOT NULL,
data nvarchar(20)
);
ALTER TABLE dbo.PK05
ADD CONSTRAINT PK_PK05 PRIMARY KEY(id);
GO
```

45

CHECK

- Check束制條件限制欄位的資料範圍。
- 決定了資料型別，是否允許NULL，可以進一步限制該欄位的資料範圍。例如：有個欄位為成績資料(Score)，型別為smallint，可以限定這個欄位的值只能介於0到100。
- CHECK束制條件是一個邏輯運算式，會回傳的值有TRUE、FALSE以及UNKNOWN。要特別注意CHECK束制條件並不會判斷NULL值，如果不允許NULL必須另外限制欄位為NOT NULL。

46

DEFAULT

- SQL語法的INSERT指令如果沒指定欄位，該欄位會寫入NULL值。透過DEFAULT值設定讓這些值可以帶入預設值。
- 有時候某個欄位是必須有值的欄位，可是在應用程式寫入資料時並沒有將資料帶入，這個時候設定DEFAULT可以確保欄位有值寫入。
- 欄位如果沒有設定DEFAULT，寫入資料時沒有指定該欄位，欄位的值會自動帶入NULL。如果欄位限制NOT NULL，寫入資料會發生錯誤。此時可藉由設定DEFAULT避免這樣的錯誤發生。

47

UNIQUE

- UNIQUE束制條件指定欄位不能重複(可以是NULL，不過只能有一筆)。設定UNIQUE束制條件的欄位SQL Server會自動建立索引。像是身份證字號、電子郵件信箱都有這種特性。

```
CREATE TABLE dbo.Student
(
   StudentID int CONSTRAINT PK_Student PRIMARY KEY,
   StudentName nvarchar(20) NOT NULL,
   EMail nvarchar(50) CONSTRAINT UQ_Student_Email UNIQUE
)
```

48

FOREIGN KEY

- FOREIGN KEY束制條件用於建立兩個資料表的關聯。像是，在訂單中的客戶編號是來自客戶基本資料的主索引鍵。訂單資料的客戶編號上設置FOREIGN KEY束制條件，參考客戶基本資料的主索引鍵。當外部索引鍵建立好之後，關聯就自動建立。
- 建立關聯時，被參考一方的欄位必須是主索引鍵或是有UNIQUE束制條件，並且不允許為NULL。

49

心得與討論

50

4. 進階資料庫設計

51

進階資料庫設計

這個單元中介紹以下幾個主題：

- 切割資料在不同檔案群組上。
- 資料壓縮技術減少對實體檔案空間的使用。
- 時態表(Temporal Table)是SQL 2016新增的功能，可以完整記錄一些重要資料表的完整異動歷程。
- 總帳資料表(Ledger Table)是SQL 2022引進區塊鍊(Block Chain) 防止竄改資料的技術，提供稽核或其他業務方，提供可昭公信的資料證明。

52

分割資料表

SQL Server可以對資料表以及索引切割。資料表或索引中的切割的Partition是依某個欄位的值將資料分割成更小的單元，這個欄位叫做Partition Key。可以將資料水平切割成幾個群組。這個功能只有在企業版以及開發版的SQL Server以及Azure SQL Server才有提供。

切割資料讓大型的資料表以及索引較容易管理。特別在要載入資料、搬移資料或是刪除同一區塊的資料。在切割的資料中可以利用一系列的語法合併、分割、搬移切割的資料。因為直接搬動內部資料而非利用INSERT、UPDATE、DELETE語法，可以取得較好的效能。

53

分割資料表範例

建立db01, 新增檔案群組 YR_BEFORE, YR_2006, YR_2007, YR_2008
分別以db01_before, db01_2006, db01_2007, db01_2008對應檔案
詳細設定如下：

Database files:

Logical Name	File Type	Filegroup	Size (MB)	Autogrowth / Maxsize		Path	File Name
db01	ROWS ...	PRIMARY	8	By 4 MB, Unlimited	...	C:\SQLDB	db01.mdf
db01_2006	ROWS ...	YR_2006	8	By 4 MB, Unlimited	...	C:\SQLDB	db01_2006.ndf
db01_2007	ROWS ...	YR_2007	8	By 4 MB, Unlimited	...	C:\SQLDB	db01_2007.ndf
db01_2008	ROWS ...	YR_2008	8	By 4 MB, Unlimited	...	C:\SQLDB	db01_2008.ndf
db01_before	ROWS ...	YR_BEFORE	8	By 4 MB, Unlimited	...	C:\SQLDB	db01_before.ndf
db01_log	LOG	Not Applicable	8	By 4 MB, Limited to 2...	...	C:\SQLDB	db01_log.ldf

54

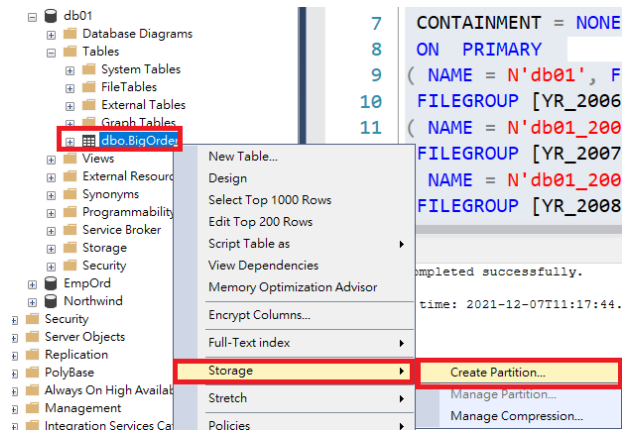
分割資料表範例(續)

在db01建立資料表 dbo.BigOrder

```
USE db01

CREATE TABLE dbo.BigOrder

(

order_id int IDENTITY(1, 1),

order_date date,

big_data nchar(4000) DEFAULT '',

CONSTRAINT PK_BigOrder PRIMARY KEY(order_id, order_date)

);
```

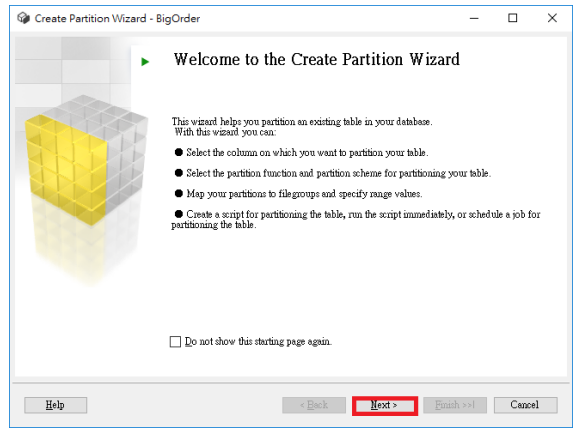
55

分割資料表範例(續)



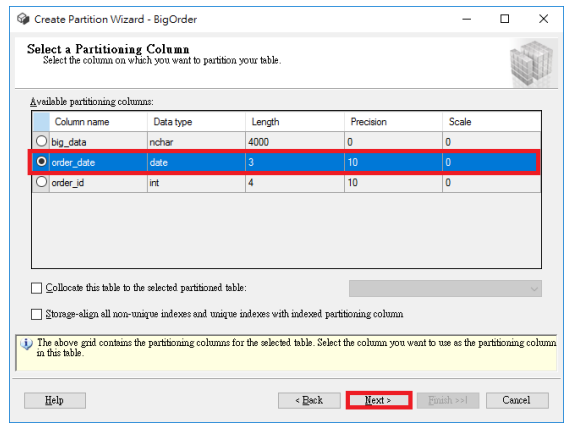
56

分割資料表範例(續)



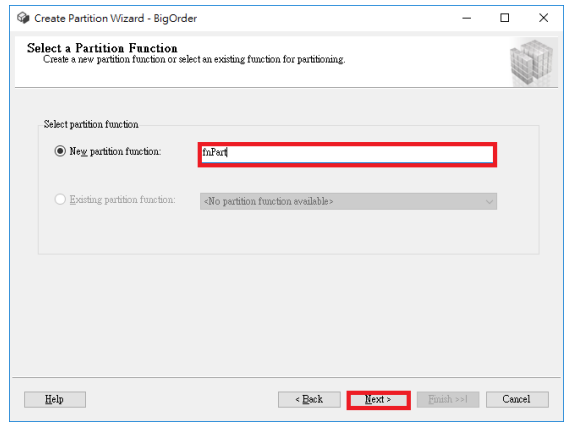
57

分割資料表範例(續)



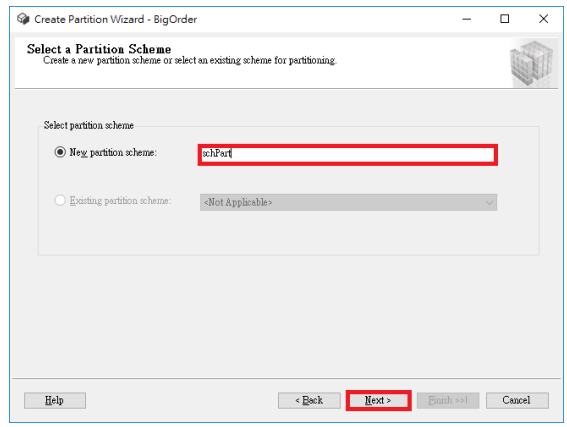
58

分割資料表範例(續)



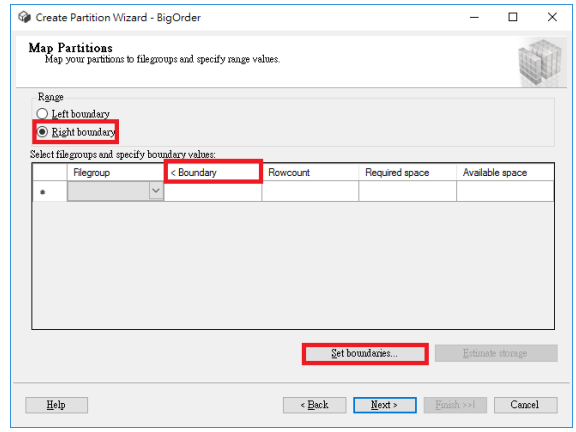
59

分割資料表範例(續)



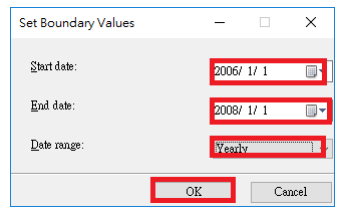
60

分割資料表範例(續)



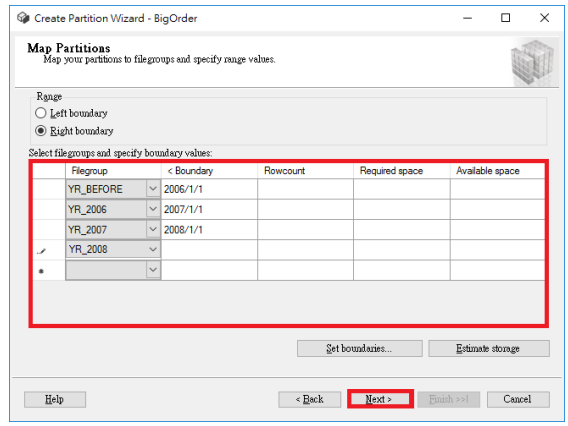
61

分割資料表範例(續)



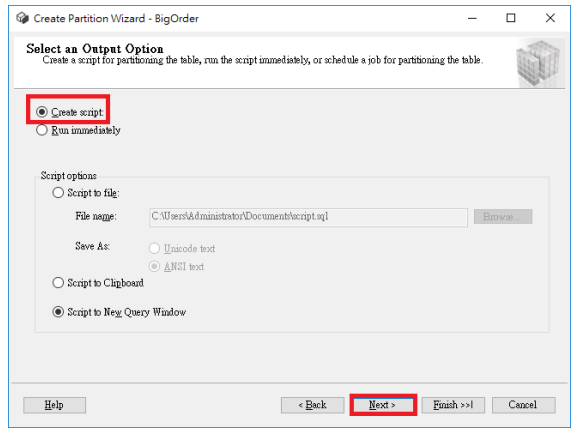
62

分割資料表範例(續)



63

分割資料表範例(續)



64

分割資料表範例-Split

```
USE [master]
GO
ALTER DATABASE [db01] ADD FILEGROUP [YR_2009]
ALTER DATABASE [db01] ADD FILE ( NAME = N'db01_2009', FILENAME = N'C:\SQLData\db01_2009.ndf' ,
SIZE = 8192KB , FILEGROWTH = 65536KB ) TO FILEGROUP [YR_2009]
GO

USE db01;
GO
ALTER PARTITION SCHEME schPart NEXT USED YR_2009;
ALTER PARTITION FUNCTION fnPart() SPLIT RANGE('2009-1-1');
GO
```

65

分割資料表範例-Merge

```
ALTER PARTITION FUNCTION fnPart() MERGE RANGE('2006-1-1');
GO

USE [db01]
ALTER DATABASE [db01]  REMOVE FILE [db01_2006]
ALTER DATABASE [db01] REMOVE FILEGROUP [YR_2006]
GO
```

66

壓縮資料表

SQL Server的資料壓縮可以減少資料的儲存空間。SQL Server在處理資料時，大多數的時間都是在資料讀取或寫入，當資料量大時所佔的分頁(Page)較多，相對必須花費較多的時間來處理這些分頁。而資料壓縮可以減少資料所佔的分頁，進而提升SQL Server的讀寫效率。

SQL Server可以對以下幾種物件壓縮：

- 資料表(Table)
- 非叢集索引(Non-Clustered Index)
- 索引檢視(Indexed View)
- 分割資料表中的各區塊(Partition)
- 空間集合索引(Spatial Index)

SQL Server上提供兩種資料壓縮方式：page compression，row compression。也可以對nchar、nvarchar使用Unicode壓縮。

67

Page壓縮

Page壓縮可以針對贅餘的資料重新取得資料空間。Page壓縮後，在Page表頭會有一組壓縮資訊(Compression Information: CI)結構。這個CI結構會儲存壓縮使用的中繼資料。

Page壓縮會以三種方式壓縮資料:

- Row壓縮：當設定Page壓縮時，SQL Server自動執行Row壓縮，也就是Page壓縮包含了Row壓縮。
- 前綴壓縮：壓縮資料後，SQL Server會掃描每個欄位，針對相同的前綴的資料會在CI結構中建立前綴的對應碼，資料以對應碼儲存，因為對應碼比前綴位元數來得小，可以藉此減少資料儲存空間。
- 字典壓縮：字典壓縮的處理方式和前綴壓縮方式很類似，不同點在於字典壓縮方式是找尋重複的值，紀錄在CI結構中。

68

Row壓縮

Row壓縮可以改變固定長度資料型別的資料儲存方式。壓縮功能把原來固定長度的儲存方法改為不固定長度方式儲存。例如：int型別佔4 bytes。可是在資料中有不同數字，像是6只需要1 byte，6000只需要2 bytes。Row壓縮只針對這類的資料。

不定長度的資料型別Row壓縮就沒有實質效益，像是xml, image, text, ntext。使用Row壓縮時，SQL Server在每個欄位會增加4 bits。這4 bits 可以換得省下的空間。NULL值就只會佔4 bits。

69

壓縮注意要點

- 對於固定長度欄位，Row壓縮可以節省一定的空間。像是smallint佔2 bytes，如果數值小於256只需要1 byte，就可以省下1 byte，但是如果大多數資料都超過256，省下的空間就比較少。
- 如果有大量重複的資料，Page壓縮可以取得較高的壓縮比。
- Unicode壓縮所節省的空間要視所使用的語言而定

70

時態表(Temporal Table)

時態表(Temporal Table)是在SQL Server 2016中新增的功能，解決了長久以來紀錄儲存資料異動的問題。

開發者會希望將資料的異動記錄下來以作為稽核或是報表。資料倉儲系統常需要從線上交易系統(OLTP)取得異動資料，緩時變換維度(SCD)是常用的一種方式。

而在線上交易系統中一個簡單的方式就是建立時態表(Temporal Table)。

71

建立時態表

```
CREATE TABLE dbo.Employee
(
EmployeeID int PRIMARY KEY,
EmployeeName nvarchar(20),
STime datetime2 GENERATED ALWAYS AS ROW START,
ETime datetime2 GENERATED ALWAYS AS ROW END,
PERIOD FOR SYSTEM_TIME(STime, ETime)
)
WITH(SYSTEM_VERSIONING=ON(HISTORY_TABLE=dbo.EmployeeHistory));
```

72

時態表的查詢語法

查詢時態表的語法中，在FOR SYSTEM_TIM子句中可以使用以下條件式：

AS OF <date_time>：指定時間點，當時資料表的內容

FROM <start_date_time> **TO** <end_date_time>：排除異動時間在起始時間之前以及結束時間之後的資料。

BETWEEN <start_date_time> **AND** <end_date_time>：和**FROM** … **TO** … 條件相同，增加起始時間等於<end_date_time>的資料

CONTAINED IN(<start_date_time> , <end_date_time>)：起始時間和結束時間包含在條件的區間中。

ALL：所有的資料以及歷史異動紀錄

73

時態表的注意事項

- 系統時間欄位(SysStartTime, SysEndTime)必須是datetime2資料型別
- 資料表必須有主索引鍵，歷史資料表不能使用束制條件
- 資料表和歷史資料表必須指定結構描述名稱(Schema Name)
- 預設狀態下，歷史資料表會啟動Page壓縮
- 歷史資料表和資料表必須在相同資料庫之內
- 時態表不能使用在FILETABLE或是FILESTREAM功能
- 使用BLOB欄位(像是varbinary(max), varchar(max), nvarchar(max))會造成大量資料儲存
- 無法直接利用INSERT, UPDATE寫入系統時間(sStartTime, SysEndTime)
- 無法直接修改歷史資料表，必須先將SYSTEM_VERSIONING設定為OFF
- 無法直接使用Truncate Table，必須先將SYSTEM_VERSIONING設定為OFF
- 不支援MERGE複寫

74

總帳資料表(Ledger Table)

總帳資料表是由系統建立版本的資料表，使用者可以在這些資料表中執行UPDATE和DELETE，同時也提供防竄改功能。

發生UPDATE或DELETE時，資料列的所有舊版都會保留在次要資料表中，稱為歷程記錄資料表(History Table)。歷程記錄資料表會反映可更新總帳資料表的結構描述。

更新資料列時，資料列的最新版本會保留在總帳資料表中，而系統會以對應用程式而言透明的方式將其較早的版本新增至歷程記錄資料表中。

75

建立總帳資料表

```
CREATE TABLE dbo.Balance(
    CustomerID int PRIMARY KEY,
    LastName varchar(50),
    FirstName varchar(50),
    Balance decimal(10, 2)
)
WITH (
 SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.BalanceHistory),
 LEDGER = ON
);
```

76

查詢總帳資料表

```
SELECT ts.name + '.' + t.name AS ledger_table_name,
hs.name + '.' + h.name AS history_table_name, vs.name + '.' + v.name AS
edger_view_name
FROM sys.tables AS t
JOIN sys.tables AS h ON (h.object_id = t.history_table_id)
JOIN sys.views v ON (v.object_id = t.ledger_view_id)
JOIN sys.schemas ts ON (ts.schema_id = t.schema_id)
JOIN sys.schemas hs ON (hs.schema_id = h.schema_id)
JOIN sys.schemas vs ON (vs.schema_id = v.schema_id)
WHERE t.object_id = OBJECT_ID('dbo.Balance');
```

77

查詢總帳資料表(續)

Results | Messages

	ledger_table_name	history_table_name	ledger_view_name
1	Account.Balance	Account.MSSQL_LedgerHistoryFor_466100701	Account.Balance_Ledger

```
INSERT INTO dbo.Balance
VALUES (1, 'Jones', 'Nick', 5000);

INSERT INTO dbo.Balance
VALUES (2, 'Smith', 'John', 500),
(3, 'Smith', 'Joe', 30),
(4, 'Michaels', 'Mary', 200);
```

78

查詢總帳資料表(續)

```
SELECT CustomerID
   ,LastName
   ,FirstName
   ,Balance
   ,ledger_start_transaction_id
   ,ledger_end_transaction_id
   ,ledger_start_sequence_number
   ,ledger_end_sequence_number
 FROM dbo.Balance;
```

79

查詢總帳資料表(續)

```
UPDATE dbo.Balance SET Balance = 10000
WHERE CustomerID = 1;


SELECT
 t.commit_time AS CommitTime
 , t.principal_name AS UserName
 , l.CustomerID, l.LastName, l.FirstName,
l.Balance
 , l.ledger_operation_type_desc AS Operation
 FROM Account.Balance_Ledger l
 JOIN sys.database_ledger_transactions t
 ON t.transaction_id = l.ledger_transaction_id
 ORDER BY t.commit_time DESC;
```

80

總帳資料表

- 總帳有助於保護資料不為任何攻擊者，也包括資料庫系統管理員 (DBA)、系統管理員和雲端系統管理員在內的高權限使用者所變更。
- 此功能和傳統的總帳一樣，會保留歷程記錄資料。
- 資料庫只要更新資料列，就會將前一個值留存在記錄資料表中並予以保護。總帳會提供資料庫一段時間內的所有變更紀事輯。

81

心得與討論

82

5. 索引(Index)

83

SQL Server的索引設計

SQL Server存取資料時可以讀取資料表中的所有資料分頁(Page)，這個稱之為資料表掃描(Table Scan)，或者僅讀取需要的索引分頁(index pages)。每個分頁為8KB。

SQL Serve存取資料時，會在各個資料表以及索引上掃描(Scan)或搜尋(Seek)。SQL Server會選擇使用最少的動作完成整個動作。查詢資料時可以將資料表以及相關資料全部讀到記憶體中再處理，但是這個方式比起使用適當的索引相對比較慢。

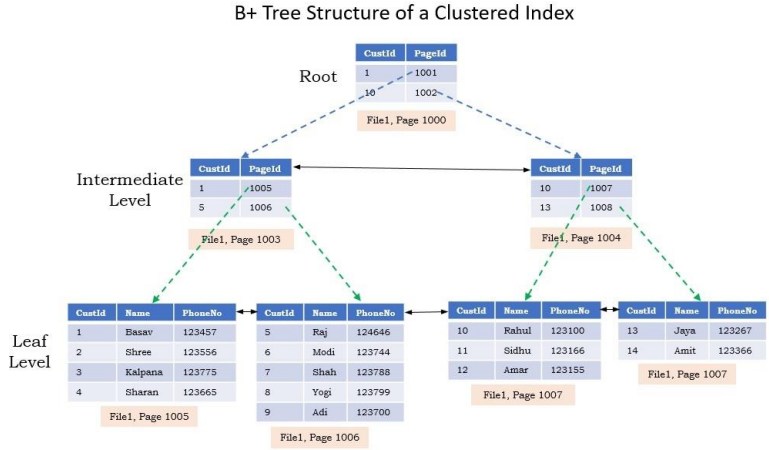
索引並不是ANSI SQL的標準語法。索引可以視為資料庫供應商提供的功能。

如前所述，SQL Server查詢資料時可以將整個資料表讀到記憶體中，再回傳結果。但是這樣的做法比較沒效率。

索引可以讓查詢的工作變得更有效率，前提是索引有做適當的設計。

84

SQL Server B+ Tree



85

以圖書館為例

以一個實體的圖書館為例，圖書館中的書通當是以一定的順序排列，例如是類別的字母順序。無論以哪一種順序，其它的排序方式都比較困難查詢。

例如：圖書館中的每一本書是以類別的順序排列放置在各個區域的書架上，現在要找某一位作者的書，只能從第一本書找到最後一本，耗時又費力。

此時如果可以另外建立作者的索引卡片，以作者的姓名排列，列出作者的每一本書在哪一區、那個書架哪一排的哪一個位置，圖書館管理員就可以根據作者索引卡片上的區域、第幾排、第幾本，就可以快速找到要找的書。

86

選擇性、密度以及索引深度

- 選擇性(Selectivity)
  - 選擇性是指一個索引值有多少對應到的資料筆數
- 密度(Density)
  - 密度是一個資料表中資料唯一值的量測值。這個值介於0到1.0之間，計算公式為：密度 = 1 / 欄位不重複值的筆數。密度愈低，索引的查詢效能愈高。
- 索引深度(Index Depth)
  - 索引深度是指從根節點到葉節點的層數。常有個迷思認為SQL Server的深度會有很多層，實際上並非如此。每個節點可以有很多個子節點，索引結構可以是很扁平的。一個只有三、四層的索引結構是很常見的。

87

索引的碎裂(Fragmentation)

- 索引的資料碎裂(Fragmentation)會造成使用索引的效率降低。
  - 資料表的資料經過修改之後會出現很多零散的空間。查詢資料時，資料分頁緊密的結合可以讓查詢效率較高。索引在一開始建立時，資料分頁是緊密排列的，但是修改資料會造成索引資料Page Split，這時候會產生索引的碎裂(Fragmentation)。
- 內部碎裂(Internal Fragmentation) vs.外部碎裂(External Fragmentation)
  - 資料庫引擎向作業系統請求空間時，是以固定大小(1Page = 8KB)，需實際使用的大小並未完全用完，因此留下一些無法使用的空間，這樣的情況會產生內部碎裂(Internal Fragmentation)。
  - 當資料異動或新增時，需要向作業系統請求更多的空間，此時連續的空間已被佔用，資料會跨到不連續的資料分頁(Page)，而產生外部碎裂(External Fragmentation)。

88

叢集索引(Clustered Index)

資料表可以具有內部邏輯順序，而不是將資料的儲存行作為堆積方式存儲。這種資料表稱為叢集索引(Clustered Index)。

常見的誤解是，叢集索引中的資料分頁是「按順序在記憶體中儲存」。儘管這在極少數情況下是可能的，但情況並非如此。如果叢集索引中的頁面是按順序儲存，則將不存在叢集索引的索引碎裂。SQL Server在建立索引時會嘗試對齊實體和邏輯順序，但是在修改資料時會出現混亂。

索引和資料分頁在邏輯層次結構中鏈接，並且在該層次結構的同一級別上的所有分頁之間都進行了雙向鏈接(Double Linked List)，在掃描索引時提供幫助。

一個資料表中只能有一個叢集索引。

89

非叢集索引(Non-Clustered Index)

資料表除了可以建立為堆積或叢集索引，另一種選擇是，可以在這些資料表上建立其它索引，快速找到所需的資料。這些附加索引稱為非叢集索引。

一個資料表最多可以有999個非叢集索引。可以在資料表上定義非叢集索引(無論該資料表使用叢集索引還是堆積)，改善查詢的效能。

有非叢集索引的資料表上更新鍵值資料時，非叢集索引也需要更新。這會影響系統的資料修改效能。新增到資料表中的每個其它索引都會增加SQL Server在修改表中的資料行時可能需要執行的工作。新增非叢集索引時，必須考慮到提升的查詢效能和修改資料時耗用的成本，從中找出平衡點。

90

建立叢集索引

```
-- Create a new table with three columns.
CREATE TABLE dbo.TestTable (
    TestCol1 INT NOT NULL,
    TestCol2 NCHAR(10) NULL,
    TestCol3 NVARCHAR(50) NULL
);
GO

-- Create a clustered index called IX_TestTable_TestCol1
-- on the dbo.TestTable table using the TestCol1 column.
CREATE CLUSTERED INDEX IX_TestTable_TestCol1 ON dbo.TestTable (TestCol1);
GO
```

91

建立非叢集索引

```
USE AdventureWorks;

-- Find an existing index named IX_ProductVendor_VendorID and delete it if found.
IF EXISTS (SELECT name FROM sys.indexes
            WHERE name = N'IX_ProductVendor_VendorID')
    DROP INDEX IX_ProductVendor_VendorID ON Purchasing.ProductVendor;
GO
-- Create a nonclustered index called IX_ProductVendor_VendorID
-- on the Purchasing.ProductVendor table using the BusinessEntityID column.
CREATE NONCLUSTERED INDEX IX_ProductVendor_VendorID
    ON Purchasing.ProductVendor (BusinessEntityID);
GO
```

92

索引的進階設定-複合鍵

索引可以建立在多個欄位上，這類的索引稱作複合鍵索引(Composite Index)。在應用程式裏複合鍵索引常常比單一鍵值索引效用更大。複合鍵索引的優點有：

- 高選擇性
- 可以避免輸出資料的排序

在圖書館的例子中，如果必須查詢某出版社在特定的發行年度的書籍，雖然建立出版社的索引有助於縮小查詢範圍，對發行年度建立索引則無助於查詢的效能提昇，如果建立出版社加上發行年度的複合鍵索引，將可更精確找到資料。

相同地，索引建立在書名上可以限縮查詢結果的資料筆數。找到書名後要確定是哪位作者，這時候複合鍵的使用就相當方便。

建立由書名加上作者的複合鍵就可以直接找到特定的資料。設計複合鍵時，索引鍵的欄位順序非常重要。第一個欄位會時選擇性最高的欄位。不僅是因為效能的關係也影響到優化處理器優化語法的演算。例如：複合鍵的順序是作者、發行日期，當WHERE子句只有使用發行日期時，這個索引就無法被選用。

93

索引的進階設定-複合鍵

在選擇複合鍵的欄位時要注意以下幾點事項：

- 欄位是否具選擇性，只使用具選擇性的欄位
- 欄位的值有多常被變動？如果鍵值欄位的值常常變更，很容易造成索引的重鍵，儘可能選用較不常變動資料的欄位。
- 欄位常常做為查詢條件嗎？
- 選擇性最高的欄位做為複合鍵的第一個欄位
- 複合鍵的欄位數儘可能減少，欄位數愈多，索引佔用空間愈大

94

索引的進階設定-Include

SQL Server對索引的大小有限制。在SQL Server 2022中的索引：

- 最多32個欄位。
- 叢集索引鍵值不得超過900 bytes。
- 非叢集索引鍵值不得超過1700 bytes。
- 索引不能包含LOB資料型別。LOB資料類型包括ntext、text、varchar(max)、nvarchar(max)、varbinary(max)、xml和image。
- 所有欄位都必須來自同一資料表或檢視。

索引中的欄位稱為鍵值欄位，而使用INCLUDE子句新增的的欄位稱為非鍵值欄位。使用INCLUDE子句可以將非鍵欄位新增到任何索引中。

95

索引的進階設定-Include

非鍵值欄位在葉節點

- 使用INCLUDE子句新增的欄位將附加到索引的葉分頁(Leaf Pages)，而不是更高的節點中。這使它們適用於SELECT語句列，而不是用於連結Join或排序的欄位。

何時使用INCLUDE子句

- 使用INCLUDE子句的最有效方法是將其用於僅在SELECT語句中使用的具有較大資料類型的列。將WHERE或GROUP BY子句中使用的列設置為鍵值欄位，以使索引較小且取得較好的效能。

96

索引的進階設定-覆蓋索引

- 覆蓋索引(Coverying Index) - Include所有欄位
  - 覆蓋索引是一種非叢集索引，其中包含特定查詢所需的所有列。因為查詢具有非叢集索引中的所有欄位，SQL Server無需從叢集索引中檢索資料。這可以大大提高查詢的性能。當索引包括查詢所需的所有欄位時，則稱該索引覆蓋了查詢。
- 什麼時候應該使用覆蓋索引？
  - 使用覆蓋索引可以提高任何運行太慢的查詢的性能。儘管索引可以提高查詢性能，但也必須付出相對的代價。輸入或修改資料時，必須更新相關索引。因此，增加新索引必須考慮，新增或修改資料所增加的時間，與減少了查詢資料所花費的時間之間的平衡。

97

索引的進階設定-篩選索引

- 篩選索引的限制
  - 只能在資料表上建立篩選索引，而不能在檢視上建立。
  - 篩選條件僅能使用簡單比較語法(只能使用基本的等式不等式、AND以及IN語法，不能使用OR、BETWEEN、LIKE或子查詢等語法)。
  - 如需執行資料運算(+, -, *, /)或是函式運算，只能在條件式中等號的右方進行。

98

索引的進階設定-Fill Factor

■什麼是填滿因子(Fill Factor)？

- 填滿因子是一個參數，用於確定索引的每個葉級資料分頁上剩餘多少剩餘空間。它用於減少索引碎裂。在建立和重建索引時，葉級資料分頁被填充到某個特定百分比(由填滿因子確定)，剩餘的空間留給以後的增長。伺服器或每個單獨的索引都可以各別設定填滿因子。

■填滿因子設定

- 填滿因子設定在每個層級資料分頁上填充資料時，將保留的剩餘空間量。填滿因子表示為頁面中已填充資料的百分比-填滿因子75表示將有75%的頁面被資料填充，而25%的空間將被保留。填滿因子0和100被視為相同，葉級資料分頁完全填充了資料。

■填滿因子如何提高性能

- 新增新的資料後，會將新增到適當的葉層級資料分頁。但是當資料分頁已滿時，必須先分裂資料分頁(Page Split)，然後才能新增新資料。這意味著將建立一個附加的葉級資料分頁，並且在新資料分頁和舊資料分頁之間拆分資料。分裂資料分頁會減慢資料新增的速度，應將其最小化。

99

索引的進階設定-Pad Index

甚麼是填充因子

- 填充索引(PAD_INDEX)是與填滿因子結合使用的參數。確定在索引的中間節點上是否還留有空間。
- 填充索引與填滿因子一起確定在索引的所有級別上都留有空間。選項WITH PAD_INDEX指定應該在中間節點處保留相同的空間量索引，保留在索引的葉節點上。填充索引使用與填滿因子指定的百分比相同的百分比。但是，SQL Server總是在中間索引頁上至少保留兩行，並確保有足夠的空間將一筆資新增到中間頁。

100

索引的進階設定-Statistics

■**什麼是統計物件(Statistics)？**

- SQL Server優化處理器(Query Optimizer)使用統計物件來建立執行計劃。SQL Server使用內部物件來保存有關資料庫中保存的資料的資訊。具體來說，如何在資料列中分配值。過時或遺失的統計資訊可能會導致查詢性能下降。
- SQL Server通過了解一資料表中包含多少個不同的值來計算統計資訊。然後使用統計資訊來估計在查詢的不同階段可能回傳的資料列筆數。

■**SQL Server中如何使用統計資訊？**

- 優化處理器是SQL Server資料庫引擎的一部分，安排資料庫引擎如何執行查詢。優化處理器使用統計資訊在執行查詢的不同方式之間進行選擇。例如，使用索引掃描(Index Scan)還是資料表掃描(Table Scan)。為了讓統計資訊有用，統計資訊必須是最新的並反映資料的當前狀態。

■統計資訊可以根據需要自動或手動更新。

101

索引的進階設定-自動更新統計資訊

以下三個方式可以確定如何在資料庫中建立和更新統計資訊。這些特定於每個資料庫，並使用ALTER DATABASE語句進行了更改：

- AUTO_CREATE_STATISTICS。優化處理器根據需要建立統計資訊，以提高查詢性能。預設情況下，此功能處於開啟狀態，通常應保持開啟狀態。
- AUTO_UPDATE_STATISTICS。優化處理器根據需要更新過時的統計資訊。預設情況下，此功能處於開啟狀態，通常應保持開啟狀態。
- AUTO_UPDATE_STATISTICS_ASYNC。僅在AUTO_UPDATE_STATISTICS啟用時使用。允許優化處理器在更新統計資訊之前選擇執行計劃。預設情況下是關閉的。

102

索引的維護-Rebuild, Reorganize

重建索引(REBUILD Index)

```
ALTER INDEX ALL ON HumanResources.Employee
REBUILD ;
```

重整索引(REORGANIZE Index)

```
ALTER INDEX ALL ON HumanResources.Employee
REORGANIZE ;
```

103

心得與討論

104

6. 檢視(View)

105

甚麼是檢視(View)

檢視可以篩選資料表的資料，讓資料可以更適合在特定的場合上使用。例如報表功能或是使用者的權限設計。檢視是儲存資料庫中的查詢運算式。雖然行為模式類似資料表，但是不會儲存實際的資料。檢視物件在資料庫中只會佔非常小的空間，資料內容會在原資料表中。

檢視是以SELECT語法建立。必須給定名稱，儲存定義的語法。可以在SELECT語法中使用檢視，也可以定義一個檢視使用其它檢視。

106

檢視的類型

資料庫中有兩大類的檢視：

- 使用者自訂檢視：資料庫中由使用者建立以及管理。
- 系統檢視：SQL Server提供的檢視。

107

使用者自訂檢視

自訂檢視大致有三種

- 一般檢視(Standard View)：由基底的資料表或檢視所組成。檢視所定義的運算語法，像是JOIN、彙總運算，在查詢當下才會執行，不會在建立時執行。這一類的檢視為一般檢視。
- 索引檢視(Indexed View)：利用建立叢集索引產生。建立索引檢視會將資料儲存在檢視中，之後查詢檢視時可以取得較佳的效能。對於JOIN多個資料表或是匯總運算的的檢視，索引檢視可以取得較佳的效能。如果底層資料表的異動頻繁，索引檢視較不適合使用。
- 分割檢視(Partitioned View)：利用UNION語法連結多個資料表。各資料表的資料列合併為一個檢視。在個別資料表上，利用CHECK語法限制資料範圍，在查詢特定範圍的資料可以產生較佳效能的執行計畫。檢視只包含一個SQL Server執行個體中的資料表為區域分割檢視，跨多個SQL Server之行個體的檢視為分散式分割檢視。

108

系統檢視

- 動態管理檢視(Dynamic Management View-DMV)

  提供動態狀態資訊，像是工作階段狀態或是目前執行中的查詢。

- 系統檢視

  提供SQL Server資料庫引擎的狀態資訊。

- 相容性檢視

  具備向前相容性，取代舊版本SQL Server的系統資料表。

- 結構描述檢視(Information Schema View)

  INFORMATION_SCHEMA結構描述符合ISO標準定義的系統檢視。

109

檢視的優點

- 簡化複雜的關聯，僅顯示對應的資料

  檢視可以幫使用者專注於需要處理的資料，而不需一一檢視所有的欄位，就如同處理單一資料表一般。

- 安全性限定使用者只能根據權限讀取所需資料

  將權限設定在檢視上，限制底層的資料表權限，簡化權限設計

- 提供應用程式的介面

  有些外部的程式無法執行TRANSACT-SQL語法或是預存程序，將查詢語法建立成檢視，可以抽離程式碼和SQL語法。

- 檢視可以作為應用程式的格式化資料的工具

  報表類型的應用程式，常需要建立複雜的查詢語法，連結多個資料表，彙總運算，檢視的使用可以簡化這些複雜的查詢語法，在應用程式中可以避免這些複雜的運算邏輯。

110

建立檢視

```
CREATE OR ALTER VIEW vwEmployeeList AS
        SELECT Title, FirstName, MiddleName, LastName
        FROM person.person
        INNER JOIN HumanResources.Employee
            ON Person.BusinessEntityID = Employee.BusinessEntityID
        WHERE Employee.CurrentFlag = 1;
GO
```

111

可變動資料的檢視

利用可變動資料的檢視可以修改底層資料表中的資料。這表示，檢視除了可以利用SELECT查詢資料外，也可以對檢視使用INSERT、UPDATE、DELETE語法修改資料。

可變動資料的檢視的限制

- 可變動資料的檢視中的欄位
- 必須從同一個資料表中查詢
- 直接參考底層的資料表欄位
- 不能使用彙總函數: AVG, COUNT, SUM, MIN, MAX, GROUPING, STDEV, STDEVP, VAR, VARP
- 不能使用DISTINCT、GROUP BY、HAVING子句
- 不能使用計算欄位
- 不能使用TOP語法

112

分割檢視(Partitioned View)

分割檢視是指在一個檢視切割成多個資料表。這個檢視像一個資表，即使是多個資料表組成的。

使用分割檢視修改資料

分割檢視是多個資料表給成一個檢視。這個檢視允許就像一個資料表的使用。對分割資料表比較容易管理。也可以對檢視執行新增、修改及刪除的動作，寫到底層資料表。

分割檢視效能上的提昇

大型資料表分割資料表可以讓個別資料表的速度提昇。查詢速度較快，索引也較小較快。分割檢視在建立上較分割資料表簡單，也可以簡化應用程式的邏輯。

113

索引檢視(Indexed View)

甚麼是索引檢視(Indexed View)

索引檢視上有一個叢集索引。藉由建立叢集索引讓檢視「實體化」(materialized)並讓資料儲存在磁碟中。複雜的Join以及彙總函的檢視上建立索引可以讓查詢速度變快，因為彙總的計算不必在執行時計算。

114

索引檢視(Indexed View)-續

■建立索引檢視(Indexed View)

➢索引檢視可以TRANSACT-SQL語法建立。檢視必須是固定式(deterministic)，確保資料儲存都是固定一致。

➢以CREATE VIEW WITH SCHEMABINDING 語法建立，之後使用CREATE UNIQUE CLUSTERED INDEX建立叢集索引。

■使用索引檢視

兩種方法可以使用索引檢視

- 直接在SQL的FROM子句查詢索引檢視。索引檢視的查詢速度會遠快過一般的檢視。
- 索引檢視也可以在執行計劃中使用，而不會去存取底層的資料表，也能取得在效能上的改善。

115

心得與討論

116

7. T-SQL流程控制語法

117

T-SQL流程控制語法

T-SQL流程控制語法主要有

- 變數(Variable)
- IF
- WHILE
- GOTO
- BEGIN TRY … END TRY

118

變數

Transact-SQL 區域變數是一種物件，可保存特定類型的單一資料值。 批次和指令碼中的變數通常的用途為：

- 做為計數器，可計算執行循環的次數，或控制執行循環的次數。
- 容納由流程控制陳述式測試的資料值。
- 若要儲存預存程序傳回碼或函數傳回值所傳回的資料值。

119

變數-設定常數值

```
USE Northwind;

DECLARE @n AS int=0;
SET @n = 50;
SELECT @n AS num;
PRINT @n;
GO
```

120

變數-以SQL語法傳回值(SELECT)

```
DECLARE @f_name nvarchar(30), @l_name nvarchar(30);

SELECT @f_name = FirstName, @l_name = LastName
FROM dbo.Employees
WHERE EmployeeID = 5;
SELECT @f_name + ' ' + @l_name AS emp_name
GO
```

121

變數-以SQL語法傳回值(SET)

```
DECLARE @rec_count int;
SET @rec_count =
(
SELECT COUNT(*)
FROM dbo.Orders
WHERE YEAR(OrderDate) = 1997 AND MONTH(OrderDate) = 1
);
SELECT @rec_count AS Order_Count;
```

122

IF

在 Transact-SQL 陳述式的執行上強制加上條件。

如果符合條件，則會執行後面 IF 關鍵詞及其條件的 Transact-SQL 指令

選擇性 ELSE 關鍵字當條件式不為True(False 或是Unknown)時執行

語法：

```
IF boolean_expression
    { sql_statement | statement_block }
[ ELSE
    { sql_statement | statement_block } ]
```

123

IF-範例-1

```
USE Northwind;

DECLARE @n int;
SET @n = 50;
IF @n < 60
    PRINT 'small';
ELSE IF @n >= 60
    PRINT 'big';
ELSE
    PRINT 'unknown';
GO
```

124

IF-範例-2

```
USE Northwind;

DECLARE @name nvarchar(20);
SET @name= N'Nancy';
IF EXISTS(SELECT * FROM dbo.Employees WHERE FirstName=@name)
    PRINT CONCAT('Yes,', @name, ' is here!');
ELSE
    PRINT CONCAT('No,', @name, ' is not here!');
GO
```

125

IF-範例-3

```
DECLARE @n int = 8, @result nvarchar(20);
IF @n < 5
    SET @result = N'Less than 5';
ELSE IF @n BETWEEN 5 AND 10
    SET @result = N'Between 5 and 10';
ELSE IF @n > 10
    SET @result = N'Greater than 10';
ELSE
    SET @result = N'Unknown';
SELECT @n AS num, @result AS result;
GO
```

126

IF-範例-3(比較CASE)

```
DECLARE @n int, @result nvarchar(20);
SET @n = 8;
SET @result = CASE
    WHEN @n < 5 THEN N'Less than 5’
    WHEN @n BETWEEN 5 AND 10 THEN N'Between 5 and 10’
    WHEN @n > 10 THEN N'Greater than 10’
    ELSE N'Unknown'
END;
SELECT @n AS num, @result AS result;
GO
```

127

WHILE

設定重複執行 SQL 陳述式或陳述式區塊的條件。 只要符合指定的條件，就會重複執行這些陳述式。 迴圈中的 WHILE 語句執行可以從 迴圈內使用 BREAK 和 CONTINUE 關鍵詞來控制。

語法：

```
WHILE boolean_expression
    { sql_statement | statement_block | BREAK | CONTINUE }
```

128

WHILE-範例-1

```
USE Northwind;

DECLARE @n int = 1;
WHILE @n <= 10
BEGIN
    PRINT @n;
    SET @n = @n + 1;
END;
GO
```

129

WHILE-範例-2

```
DECLARE @d date, @d1 date , @d2 date;
DECLARE @tbl AS TABLE(dt date);
SET @d1 = DATEFROMPARTS(1997, 1, 1);
SET @d2 = EOMONTH(@d1);
SET @d = @d1;
SET NOCOUNT ON;
WHILE @d <= @d2
BEGIN
    INSERT INTO @tbl(dt) VALUES(@d);
    SET @d = DATEADD(DAY, 1, @d);
END;
SELECT * FROM @tbl;
```

130

TRY … CATCH

實作 Transact-SQL 的錯誤處理，類似於 C# 和 Visual C++ 語言中的例外狀況處理。 Transact-SQL 語句的群組可以封入區塊中 TRY 。 如果區塊中 TRY 發生錯誤，控件通常會傳遞至區塊中所 CATCH 封入的另一組語句。

語法：

```
BEGIN TRY
    { sql_statement | statement_block }
END TRY
BEGIN CATCH
    [ { sql_statement | statement_block } ]
END CATCH
[ ; ]
```

131

TRY … CATCH-範例-1

```
SELECT 1;
IF @@ERROR <> 0 GOTO errHandler;
SELECT 2;
IF @@ERROR <> 0 GOTO errHandler;
SELECT 3/0;
IF @@ERROR <> 0 GOTO errHandler;
SELECT 4;
IF @@ERROR <> 0 GOTO errHandler;
```

132

TRY … CATCH-範例-1(續)

```
SELECT 5;
IF @@ERROR <> 0 GOTO errHandler;
RETURN;
errHandler:
SELECT 'Error';
GO
```

133

TRY … CATCH-範例-2

```
BEGIN TRY
    SELECT 1;
    SELECT 2;
    SELECT 3/0;
    SELECT 4;
    SELECT 5;
END TRY
```

134

TRY … CATCH-範例-2(續)

```
BEGIN CATCH
    SELECT ERROR_NUMBER() AS err_no,
        ERROR_LINE() AS err_line,
        ERROR_MESSAGE() AS err_msg,
        ERROR_PROCEDURE() AS err_proc,
        ERROR_SEVERITY() AS err_level,
        RROR_STATE() AS err_state
END CATCH
```

135

心得與討論

136

8. 交易(Transaction)

137

關聯式資料庫中的交易

交易是將多個TRANSACT-SQL語法變成一個工作。所有對資料的變動必須在交易成功才能寫入。在T-SQL語法中，每一個執行命令(以分號做結尾的語法)，都被視作一筆交易，交易必須符合ACID原則：

- **Atomicity**：不可分割性。
- **Consistency**：資料一致性
- **Isolation**：隔離性
- **Durability**：持久性

138

併行作業處理

資料庫中多人同時變更並用的資料時，並行作業是相當重要的系統功能。在分享的資料上，愈多人使用，並行作業的等級就愈高。並行作業的等級提高後，發生資料衝突(同時間兩個以上的使用者要存取相同的資料)的可能性也會跟著提高。

在並行作業時，處理資料衝突有兩種方式，樂觀鎖定以及悲觀鎖定。

139

悲觀鎖定(Pessimistic Concurrency)

悲觀鎖定有以下的特性：

- 資料讀取時鎖定，其它使用者不能修改資料。
- 資料修改時鎖定，其它使用者不能修改以及讀取資料。
- 資料讀取或修改時都會鎖定，資料鎖定的數目較多。
- 寫入時其它使用者不能讀取和寫入，讀取時其它使用者不能寫入。

悲觀鎖定適合以下類型的系統

- 資料競爭(Data Contention)較常發生
- 鎖定時間較短
- 防止資料衝突的鎖定成本低於復原寫入資料的成本

140

樂觀鎖定(Optimistic Concurrency)

樂觀鎖定有以下的特性：

- 資料讀取時不鎖定，其它使用者可以讀取資料、修改資料。
- 資料修改時不鎖定，其它使用者可以讀取資料、修改資料。
- 資料寫入前檢查資料在異動期間是否變動過，如果沒有異動就寫入資料。
- 資料鎖定數目較少

樂觀鎖定適合以下類型的系統

- 資料競爭不常發生
- 資料異動時間較長
- 復原寫入資料的成本低於防止資料衝突的鎖定成本
- 讀取資料時可以同時寫入資料

141

鎖定範圍

資料庫物件及資源有不同層級的鎖定範圍。SQL Server的鎖定列表(由最小範圍到最大範圍)，RID以及Key是同級的。

- RID 資料列識別碼：用來鎖定堆積內單一資料列。
- KEY： 索引中的資料列鎖定，用來保護可序列化交易中的索引鍵範圍。
- PAGE： 資料庫中的 8 KB 頁面，例如資料或索引頁面。
- EXTENT： 連續八個頁面的群組，例如資料頁或索引頁面。
- HoBT： 堆積或B+ Tree目錄。 針對資料表中沒有叢集索引的 B+ Tree結構 (索引) 或堆積資料頁面進行保護鎖定。

142

鎖定範圍(續)

- TABLE： 一整個資料表，包含所有資料和索引。
- FILE： 資料庫檔案
- APPLICATION： 應用程式指定資源。
- METADATA： 中繼資料鎖定。
- ALLOCATION_UNIT： 配置單位。
- DATABASE： 整個資料庫。

143

鎖定模式

- **共用鎖定-Sared Lock(S)**：鎖定允許並行交易在(Pessimistic)並行控制之下讀取 (SELECT) 資源。
- **更新鎖定-Update Lock(U)**：鎖定可防止常見的死結。 在可重複讀取或可序列化交易中，交易在讀取資料時取得資源 (頁面或資料列) 的共用 (S) 鎖定，然後修改資料，此過程需要將鎖定轉換為獨佔 (X) 鎖定。
- **獨佔鎖定-Exclusice Lock(X)**：鎖定防止並行交易存取某個資源。運用獨佔 (X) 鎖定，沒有其它交易可修改資料。
- **意圖鎖定-Intent Lock(I)**：SQL Server Database Engine 使用意圖鎖定來保護，它把共用 (S) 鎖定或獨佔 (X) 鎖定放在鎖定階層中較低的資源上。

144

鎖定模式的相容性。

現有已授與的模式 / 要求的模式	IS	S	U	IX	SIX	X
意圖共用 (IS)	Y	Y	Y	Y	Y	N
共用 (S)	Y	Y	Y	N	N	N
更新 (U)	Y	Y	N	N	N	N
意圖獨佔 (IX)	Y	N	Y	N	N	N
與意圖獨佔共用 (SIX)	Y	N	N	N	N	N
獨占 (X)	N	N	N	N	N	N

145

交易隔離性等級

- **READ UNCOMMITTED**

  可以讀取其它交易已修改，但尚未認可的資料列。

- **READ COMMITTED**

  不能讀取其它交易已修改而尚未認可的資料。

- **REPEATABLE READ**

  不能讀取其它交易已修改而尚未認可的資料，且在目前交易完成之前，任何其它交易都不能修改目前交易已讀取的資料。

146

交易隔離性等級(續)

- **SERIALIZABLE**
  - 無法讀取其它交易已修改但尚未認可的資料。
  - 在目前交易完成之前，其它交易不能修改目前交易已讀取的資料。
  - 在目前交易完成之前，其它交易所插入的新資料列，其索引鍵值不能在目前交易的任何陳述式所讀取的索引鍵範圍中。
- **SNAPSHOT**

  指定交易中任何陳述式所讀取的資料，都是交易開始時就存在之資料的交易一致性版本。

147

Dirty Read, Phantom Read

- **Dirty Read**

  讀取到尚未被認可(Uncommitted)的資料。
- **Phantom Read**

  一次交易中，兩個SELECT語法不具連續性(Serializable)。

148

死結(Deadlock)-發生原因

- **互斥(Mutual Exclusion)**
  同時間一項資源(Resource)只能被一個程序(Process)所佔用
- **不可強佔(No Preemption)**
  資源被占用時，其他程序必須等待資源被釋放才能占用這個資源
- **持有並等待(Hold and Wait)**
  一個程序佔用一項資源，也等待另一項資源被釋放
- **循環式等待(Circular Wait)**
  A程序等待B程序釋放資源
  B程序等待C程序釋放資源
  C程序等待D程序釋放資源
  ....
  X程序等待A程序釋放資源

149

死結(Deadlock)-處理方式

SQL Server中的鎖定監控程序(Lock Monitor Process)負責偵測死結。依據以下方式週期式執行：

- **預設狀態下，每5秒鐘偵測死結的發生**
- **發現死結時，立刻進行偵測程序**
- **確定死結的發生，偵測的頻率改為每0.1秒執行一次**

偵測到死結時，鎖定監控程序會選定一個程序為犧牲者(Victim)，終止這個程序並發送錯誤邊碼為1205的錯誤訊息。程序終止後，鎖定被解除，其他程序就可以繼續工作。死結犧牲者會依下列方式進行選定

- **如果發生死結的交易deadlock priority相同，會優先選定復原成本較低的交易。**
- **deadlock priority設定不相同時，會優先選定deadlock priority較低者**

150

死結的偵測

- SQL Server Profiler
- Extended Event

151

心得與討論

152

9. 預存程序(Stored Procedure)

153

甚麼是預存程序(Stored Procedure)

預存程序有如高階語言中的程序、方法或是函數。可以有輸入參數、輸入參數以及回傳值。

預存程序會將查詢結果傳回。如果預存程序中有多個查詢語法，會回傳所有的查詢結果。

預存程序可以用TRANSACT-SQL語法建立或是.NET程式碼建立，然後以**EXECUTE**語法執行。

154

預存程序的優點

- 安全性的邊界

  預存程式可以視為結構描述的一部份，可以提高應用程式的安全性。

- 模組化程式設計

  考量程式碼管理時再用性很重要。預存程序可以讓重複執行的程式邏輯寫在一個模組裏，這有助於模組化設計的概念。

- 延遲綁定(Delayed Binding)

  在建立預程序時，要存取的資料庫物件有可能還沒生成。預存程序在執行時期才編譯為可執行程式，這有助於資料庫的建置順序。

- 效能

  比起從應用程式中執行多行的TRANSACT-SQL語法到SQL Server中，預存程序只需傳送預存程序名稱，可以大量降低網路流量。

155

預存程序不能使用的語法

並非所有的TRANSACT-SQL語法都能在存程序中使用。以下的語法是不能使用的：

- USE databasename
- CREATE AGGREGATE
- CREATE DEFAULT
- CREATE RULE
- CREATE SCHEMA
- CREATE or ALTER FUNCTION

156

預存程序不能使用的語法(續)

- CREATE or ALTER PROCEDURE
- CREATE or ALTER TRIGGER
- CREATE or ALTER VIEW
- SET PARSEONLY
- SET SHOWPLAN_ALL
- SET SHOWPLAN_TEXT
- SET SHOWPLAN_XML

157

建立預存程序

```
USE Northwind;
GO
CREATE PROC dbo.P1
AS
SELECT EmployeeID,
FirstName + ' ' + LastName
FROM dbo.Employees
ORDER BY FirstName;
GO
EXEC dbo.P1;
GO
```

158

建立預存程序-範例-1

```
USE Northwind;
GO
CREATE OR ALTER PROC dbo.P1
AS
SELECT EmployeeID,
FirstName + ' ' + LastName
FROM dbo.Employees
ORDER BY FirstName;
GO
EXEC dbo.P1;
GO
```

159

建立預存程序-範例-2

```
CREATE OR ALTER PROC dbo.P2
@yr int, @mn int
AS
SELECT OrderID, OrderDate
FROM dbo.Orders
WHERE YEAR(OrderDate) = @yr
AND MONTH(OrderDate) = @mn
GO
```

160

建立預存程序-範例-2(續)

```
EXEC dbo.P2 1997, 1;
EXEC dbo.P2 @yr=1997, @mn=2;
EXEC dbo.P2 @mn=3, @yr=1997;
EXEC dbo.P2 @yr=1997;
GO
```

161

建立預存程序-範例-2(續)

```
CREATE OR ALTER PROC dbo.P2
@yr int, @mn int=NULL
AS
SELECT OrderID, OrderDate
FROM dbo.Orders
WHERE YEAR(OrderDate) = @yr
AND (MONTH(OrderDate) = @mn OR @mn IS NULL)
GO
```

162

建立預存程序-範例-2(續)

```
EXEC dbo.P2 1997, 1;
EXEC dbo.P2 @yr=1997, @mn=2;
EXEC dbo.P2 @mn=3, @yr=1997;
EXEC dbo.P2 @yr=1997;
GO
```

163

建立預存程序-範例-3

```
CREATE OR ALTER PROC dbo.P3
@yr int, @mn int = NULL, @eid int = NULL
AS
SELECT e.EmployeeID, e.FirstName,
MONTH(o.OrderDate) AS Mn,
SUM(od.Quantity*od.UnitPrice) AS Total
FROM dbo.Employees e
INNER JOIN dbo.Orders o
ON e.EmployeeID = o.EmployeeID
```

164

建立預存程序-範例-3(續)

```
INNER JOIN dbo.[Order Details] od
ON o.OrderID =od.OrderID
WHERE YEAR(o.OrderDate) = @yr
AND (MONTH(o.OrderDate) = @mn OR @mn IS NULL)
AND (e.EmployeeID = @eid OR @eid IS NULL)
GROUP BY e.EmployeeID, e.FirstName, MONTH(o.OrderDate)
ORDER BY Mn, e.EmployeeID
GO
```

165

建立預存程序-範例-3(續)

```
EXEC dbo.P3 @yr=1997, @mn=1, @eid=1;
EXEC dbo.P3 @yr=1997, @mn=1;
EXEC dbo.P3 @yr=1997, @eid=1;
EXEC dbo.P3 @yr=1997;
```

166

使用輸出參數

輸出參數的宣告與使用和輸入參數類似，但是輸出參數有一些特殊要求。

輸出參數的要求：

- 在預存程序中宣告輸出參數必須加上OUTPUT關鍵字。
- 執行預存程序時，EXEC的語法中必須在輸出參數加上OUTPUT關鍵字。

167

使用輸出參數-範例-4

```
CREATE PROC dbo.P4
AS
SELECT * FROM dbo.Orders;
GO

DECLARE @n int;
EXEC @n=dbo.P4;
SELECT @n AS ret_value; --0
GO
```

168

使用輸出參數-範例-4(續)

```
CREATE OR ALTER PROC dbo.P4
AS
SELECT * FROM dbo.Orders;
RETURN 100;
GO

DECLARE @n int;
EXEC @n=dbo.P4;
SELECT @n AS ret_value; --100
GO
```

169

使用輸出參數-範例-4(續)

```
CREATE OR ALTER PROC dbo.P4
AS
SELECT * FROM dbo.Orders;
RETURN 100;
GO

DECLARE @n int;
EXEC @n=dbo.P4;
SELECT @n AS ret_value; --100
GO
```

170

使用輸出參數-範例-5

```
CREATE PROC dbo.P5
@yr int, @mn int, @record_count int OUTPUT
AS
SELECT @record_count = COUNT(*)
FROM dbo.Orders
WHERE YEAR(OrderDate) = @yr AND MONTH(OrderDate) = @mn;
GO

DECLARE @n int;
EXEC dbo.P5 @yr=1997, @mn=1, @record_count=@n OUTPUT;
SELECT @n AS RecordCount;
GO
```

171

參數探測(Parameter Sniffering)

預存程序執行時，重複使用先前執行過同樣的執行計劃通常不會有太大的問題。但在使用參數時，重複的執行計劃可能在效能並不理想，各而需要重新產生執行計劃。

這個問題通常稱之為參數探測(Parmeter Sniffing)問題，SQL Server有幾個解這個問題的方法。注意到參數探測僅適用於預存程序的參數，對於批次執行的變數是沒有作用的。

172

參數探測-範例-6

```
CREATE PROC dbo.P6
@d1 date, @d2 date
AS
SELECT * FROM dbo.Orders
WHERE OrderDate BETWEEN @d1 AND @d2
GO
EXEC dbo.P6 @d1 = '1997-1-1', @d2='1997-1-5';
EXEC dbo.P6 @d1 = '1997-1-1', @d2='1997-1-31';
EXEC dbo.P6 @d1 = '1997-1-1', @d2='1997-1-31' WITH RECOMPILE;
GO
```

173

參數探測-範例-6(續)

```
CREATE OR ALTER PROC dbo.P6
@d1 date, @d2 date
WITH RECOMPILE
AS
SELECT * FROM dbo.Orders
WHERE OrderDate BETWEEN @d1 AND @d2
GO
EXEC dbo.P6 @d1 = '1997-1-1', @d2='1997-1-5';
EXEC dbo.P6 @d1 = '1997-1-1', @d2='1997-1-31';
```

174

參數探測-範例-6(續)

```
CREATE OR ALTER PROC dbo.P6
@d1 date, @d2 date
AS
SELECT * FROM dbo.Orders
WHERE OrderDate BETWEEN @d1 AND @d2
OPTION (OPTIMIZE FOR(@d1='1997-1-1', @d2='1997-1-31'))
GO
EXEC dbo.P6 @d1 = '1997-1-1', @d2='1997-1-5';
EXEC dbo.P6 @d1 = '1997-1-1', @d2='1997-1-31';
```

175

心得與討論

176

10. 函數(Function)

177

SQL Server的函數

函數是包含一個或多個Transact-SQL陳述式的程序可以將程式碼封裝以重複使用。函數可以輸入零個到多個參數，然後輸出一個純量值(scalar)或是表格(table)。函數並不支援輸出參數而是直接將結果以純量或是表格的形式回傳。

178

函數的類型

Microsoft® SQL Server®提供三種函數：

- 純量值函數(scalar functions)
- 資料表值函數(TVFs, table-value functions)
- 系統函數(system functions)

使用者可以創建：

- 純量值函數(scalar functions)
- 內嵌資料表值函數(inline TVFs)
- 多重陳述式資料表值函數(multistatement TVFs)。

179

純量值函數

純量值函數會回傳單一的資料值，資料值的型別由RETURN子句所定義。純量值函數主體定義於BEGIN...END區塊中，由一系列具有回傳值的Transact-SQL陳述句所組成。

注意事項：

- 針對函數與其參考的所有資料庫中物件都使用兩部分名稱(two-part naming)
- 避免Transact-SQL 造成陳述式取消並且以模組中下一個陳述式繼續 (例如觸發程序或預存程序)的錯誤會在函式內部以不同方式處理。 在函數中，這樣的錯誤會造成函數停止執行。進而導致叫用該函數的陳述式取消。
- SQL Server不允許函數修改底層資料庫。

180

純量值函數

```
CREATE FUNCTION dbo.Fact(@n tinyint)
RETURNS decimal(38, 0) AS
BEGIN
    DECLARE @i tinyint =1, @result decimal(38, 0) = 1;
    WHILE @i <= @n
    BEGIN
        SET @result = @result * @i;
        SET @i = @i + 1;
    END
    RETURN @result
END
GO
```

181

純量值函數(續)

```
DECLARE @i tinyint = 1;
WHILE @i <= 33
BEGIN
PRINT CONCAT(@i, '! = ', dbo.Fact(@i));
SET @i = @i + 1;
END
```

182

內嵌資料表值函數

```
CREATE dbo.Top3Order(@yr int, @mn int = NULL, @eid int = NULL)
RETURNS TABLE AS RETURN
(
    SELECT TOP(3)
      OrderID, OrderDate, CustomerID, EmployeeID
    FROM dbo.Orders
    WHERE YEAR(OrderDate) = @yr
      AND (MONTH(OrderDate) = @mn OR @mn IS NULL)
      AND (EmployeeID = @eid OR @eid IS NULL)
    ORDER BY OrderID DESC
)
GO
```

183

內嵌資料表值函數(續)

```
SELECT * FROM dbo.Top3Order(1997, 1, 1);
SELECT * FROM dbo.Top3Order(1997, DEFAULT, 1);
SELECT * FROM dbo.Top3Order(1997, 1, DEFAULT);
SELECT * FROM dbo.Top3Order(1997, DEFAULT, DEFAULT);
```

184

多重陳述式資料表值函數

```
CREATE FUNCTION dbo.StringTable(@s nvarchar(max))
RETURNS @tbl TABLE(ch nchar(1)) AS
BEGIN
    DECLARE @i int = 1;
    WHILE @i <= LEN(@s)
    BEGIN
        INSERT INTO @tbl(ch) VALUES(SUBSTRING(@s, @i, 1));
        SET @i = @i + 1;
    END
    RETURN
END
GO
```

185

多重陳述式資料表值函數(續)

```
EXEC sp_helptext 'dbo.StringTable';

SELECT * FROM dbo.StringTable('Hello world!');
```

186

建立函數的注意事項

建立使用者定義函數時應該要考慮以下幾點注意事項：

- 使用者定義函數效能：在許多案例中，內嵌式函數會比多重陳述式函數快許多，因此可能的話，應該盡量試著將函數實作為內嵌式函數。
- 使用者定義函數大小：避免建立大型、廣泛用途的函數，函數應保持相對輕巧且針對特定目的用途設計。這樣可以避免程式碼複雜且提高函數被重複使用的機會。
- 使用者定義函數名稱：使用二部分名稱來限定資料庫中被函數參考到的物件名稱，而對於函數的名稱，也應使用二部分名稱。
- 使用者定義函數與例外處理：避免會產生Transacts-SQL錯誤的陳述句，因為函數中並沒有例外處理的語法。
- 使用者定義函數索引：注意函數與索引一起使用所造成的影響，特別是在WHERE子句所使用的條件判斷式，如以下範例，就可以將使用在**CustomerID**上的索引移除。

187

心得與討論

188