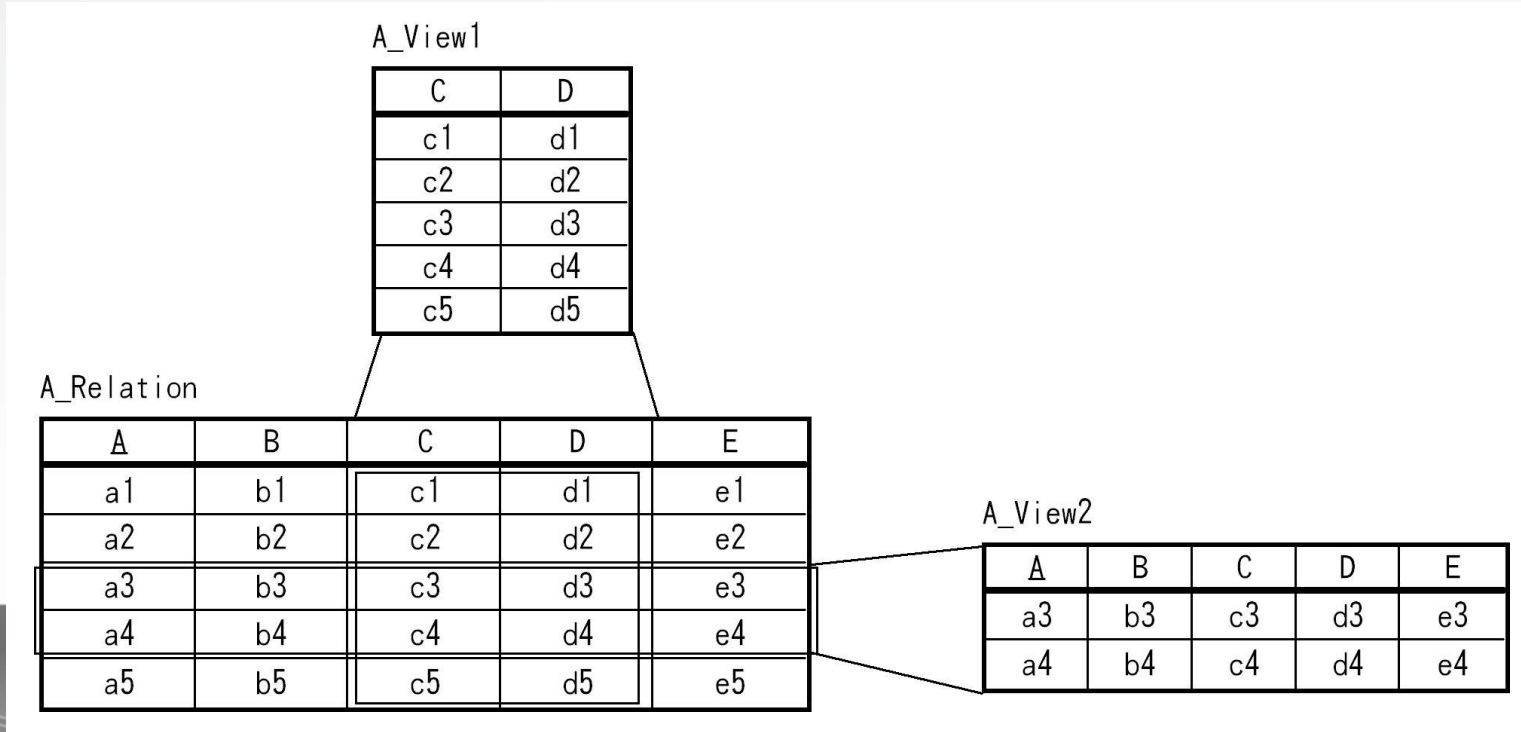
第11章 視界與資料庫程式設計

11-1 視界的基礎

- 11-1-1 視界的內容
- 11-1-2 視界的種類
- 11-1-3 視界的優缺點

11-1-1 視界的內容-說明

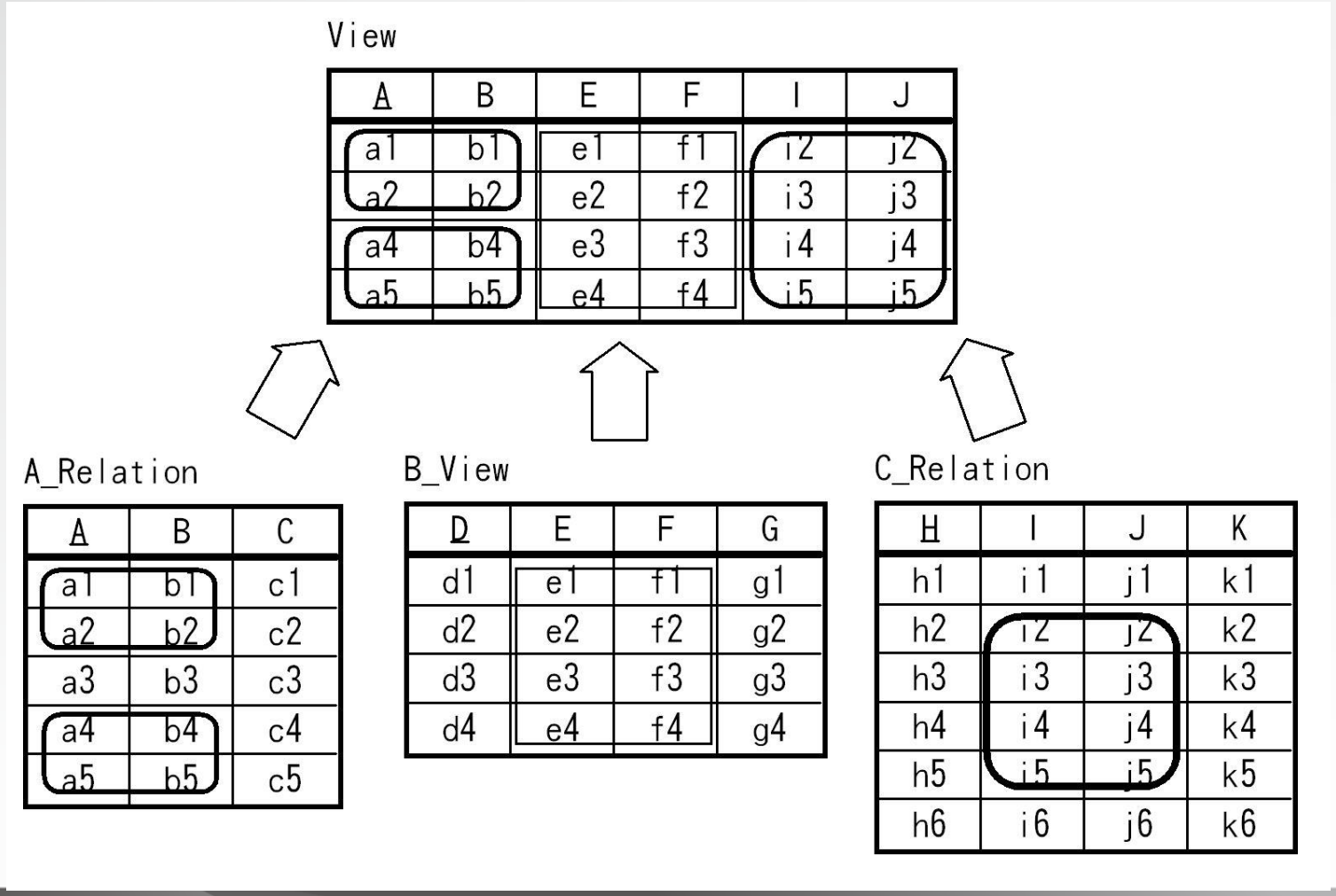
- 「視界」（Views）相當於ANSI/SPARC三層資料系統架構中外部層（External Level）顯示的資料，這是從基底關聯表（Base Relation）導出的虛擬關聯表（Virtual Relation），如下圖所示：



11-1-1 視界的內容-資料來源

- 視界之所以稱為虛擬關聯表，這是因為它並沒有真正將資料儲存在磁碟，而只是一些定義資料，定義從那些基底關聯表或視界挑出那些屬性或值組，SQL語言是使用CREATE VIEW指令來定義視界。
- 視界顯示的資料是從定義的基底關聯表導出，依照定義過濾掉不屬於視界的資料，如果視界是從其他視界導出，只是重複過濾一次，所以視界如同是一個從不同基底關聯表或視界抽出的資料積木，然後使用這些積木拼出所需的關聯表。

11-1-1 視界的內容-資料來源圖例



11-1-2 視界的種類

- 列欄子集視界（Row-and-Column Subset Views）：從單一基底關聯表或視界導出的視界，只挑選關聯表或視界中所需的屬性和值組，換句話說。
- 合併視界（Join Views）：使用合併查詢從多重基底關聯表或視界所導出的視界，新視界的屬性和值組是來自多個基底關聯表或視界。
- 統計摘要視界（Statistical Summary Views）：一種列欄子集視界或合併視界，只是使用聚合函數產生指定欄位所需的統計資料。

11-1-3 視界的優缺點-優點1

- 達成邏輯資料獨立：視界相當於外部與概念對映（External/Conceptual Mapping），更改基底關聯表的綱要，只需同時更改視界的外部與概念對映的定義資料，就可以讓使用者檢視相同觀點的資料，而不會影響外部綱要。
- 增加資料安全性：視界可以隱藏和過濾資料，只讓使用者看到它允許看到的資料，增加資料的安全性。
  - 例如：在Students關聯表擁有生日欄位birthday，我們可以使用視界隱藏學生生日，只讓使用者看到其他部分的學生資料。

11-1-3 視界的優缺點-優點2

- 簡化資料查詢：將常用和複雜的查詢定義成視界，就可以簡化資料查詢，因為不再需要每次重複執行複雜的SQL查詢指令，直接使用現成的視界即可。
- 簡化使用者觀點：視界可以增加資料的可讀性，讓資料庫使用者專注於所需的資料。
  - 例如：替欄位更名成使用者觀點的欄位名稱。

11-1-3 視界的優缺點-缺點

- 執行效率差：視界並沒有真正儲存資料，資料是在使用時才從基底關聯表導出，因為經過轉換手續，執行效率一定比不過直接存取基底關聯表。
- 更多的操作限制：在新增、更新和刪除視界資料時，為了避免違反資料庫的完整性限制條件，在操作上有更多的限制。
- 增加管理的複雜度：視界可以一層一層的從其他視界導出。
  - 例如：View2和View4是從View1導出，View3是從View2導出，複雜的視界關聯將增加管理眾多關聯表和視界的複雜度，一不小心刪錯視界，就有可能造成嚴重後果。

11-2 建立與刪除視界

- 11-2-1 建立列欄子集視界
- 11-2-2 建立合併視界
- 11-2-3 建立統計摘要視界
- 11-2-4 從其他視界建立視界

11-2 建立與刪除視界-建立語法

- 在SQL語言建立視界是使用CREATE VIEW指令，其基本語法如下所示：

  CREATE VIEW view_name AS

  select_statement

- 上述語法建立名為view_name的視界，資料來源是AS之後的SELECT指令敘述。視界的欄位和資料型態都是對應SELECT指令敘述的欄位，如果使用聚合函數（Aggregate Function），請使用AS指令定義別名。

11-2 建立與刪除視界-刪除語法

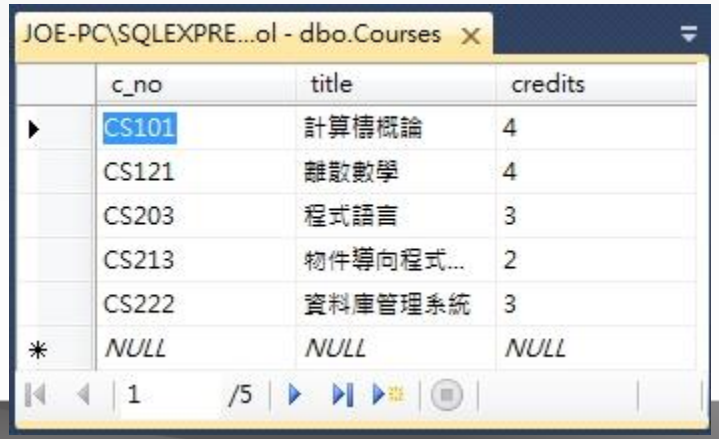
- 對於資料表中不再需要的視界，SQL語言可以使用DROP VIEW指令刪除視界，其基本語法如下所示：

  DROP VIEW view_name

- 刪除名為view_name的視界。

11-2-1 建立列欄子集視界-說明

- 列欄子集視界是指視界的內容是基底關聯表屬性集的子集合，也就是以資料表的欄位和記錄為單位，從這些欄位和記錄集合中，取出所需子集合的檢視表。
- 例如：在本節準備建立視界的基底關聯表Courses資料表，如下圖所示：



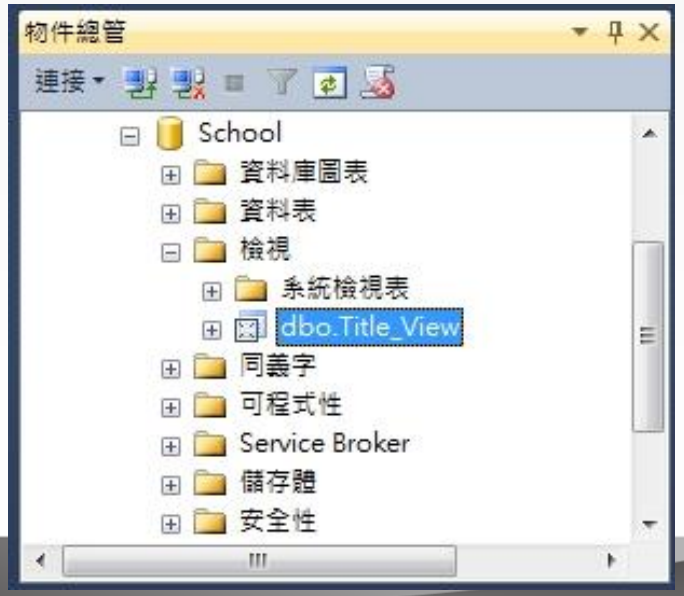
11-2-1 建立列欄子集視界-建立欄子集視界(範例)

- 欄子集視界（Column Subset Views）是指這個視界的屬性是基底關聯表屬性集合的子集合。

SQL查詢範例：Ch11_2_1_01.sql

- 從Courses基底關聯表建立課程名稱資料的Title_View視界，如下所示：

```
CREATE VIEW Title_View AS
SELECT c_no, title FROM Courses
```



11-2-1 建立列欄子集視界-建立欄子集視界(結果)

SQL查詢範例：Ch11_2_1_02.sql

- 使用SELECT指令查詢Title_View視界的內容，如下所示：

```
SELECT * FROM Title_View
```

	c_no	title
1	CS101	計算機概論
2	CS121	離散數學
3	CS203	程式語言
4	CS213	物件導向...
5	CS222	資料庫管...

11-2-1 建立列欄子集視界-建立列子集視界(範例)

- 列子集視界（Row Subset Views）是指這個視界的值組是基底關聯表值組集合的子集合。

SQL查詢範例：Ch11_2_1_04.sql

- 從Courses基底關聯表建立課程學分credits欄位大於等於3的Credits_View視界，如下所示：

```
CREATE VIEW Credits_View AS
SELECT * FROM Courses
WHERE credits >= 3
```

11-2-1 建立列欄子集視界-建立列子集視界(結果)

SQL查詢範例：Ch11_2_1_05.sql

- 查詢Credits_View視界的內容，如下所示：

```
SELECT * FROM Credits_View
```

	c_no	title	credits
1	CS101	計算機概論	4
2	CS121	離散數學	4
3	CS203	程式語言	3
4	CS222	資料庫管...	3

11-2-1 建立列欄子集視界-建立列欄子集視界(範例)

- 列欄子集視界（Row-and-Column Subset Views）是指視界的屬性和值組都是基底關聯表屬性和值組集合的子集合。

SQL查詢範例：Ch11_2_1_06.sql

- 從Courses基底關聯表建立學分credits欄位大於等於3，而且只有c_no和title二個欄位的Major_View視界，如下所示：

```
CREATE VIEW Major_View AS
SELECT c_no, title FROM Courses
WHERE credits >= 3
```

11-2-1 建立列欄子集視界-建立列欄子集視界(結果)

SQL查詢範例：Ch11_2_1_07.sql

- 查詢Major_View視界的內容，如下所示：

```
SELECT * FROM Major_View
```

	c_no	title
1	CS101	計算機概論
2	CS121	離散數學
3	CS203	程式語言
4	CS222	資料庫管理系統

11-2-2 建立合併視界-說明

- 合併視界（Join Views）是多個關聯表執行合併查詢所建立的視界。

11-2-2 建立合併視界-範例

SQL查詢範例：Ch11_2_2_01.sql

- 從Students、Courses、Instructors和Classes四個資料表建立合併視界Std_Class_View，可以顯示學生的選課資料，如下所示：

```
CREATE VIEW Std_Class_View AS
SELECT Classes.sid, Students.name, Classes.eid,
Instructors.name AS professor,
Classes.c_no, Courses.title, Classes.room
FROM Students, Courses, Instructors, Classes
WHERE Students.sid = Classes.sid
   and Courses.c_no = Classes.c_no
   and Instructors.eid = Classes.eid
```

11-2-2 建立合併視界-結果

SQL查詢範例：Ch11_2_2_02.sql

- 查詢Std_Class_View視界的內容，如下所示：

```
SELECT * FROM Std_Class_View
```

	sid	name	eid	professor	c_no	title	room
1	S001	陳會安	E001	陳慶新	CS101	計算機概論	180-M
2	S003	張三丰	E001	陳慶新	CS213	物件導向程式設計	622-G
3	S001	陳會安	E002	楊金檳	CS222	資料庫管理系統	100-M
4	S002	江小魚	E002	楊金檳	CS222	資料庫管理系統	100-M
5	S003	張三丰	E002	楊金檳	CS121	離散數學	221-S
6	S004	李四方	E002	楊金檳	CS222	資料庫管理系統	100-M
7	S001	陳會安	E003	李鴻章	CS203	程式語言	221-S
8	S001	陳會安	E003	李鴻章	CS213	物件導向程式設計	500-K
9	S002	江小魚	E003	李鴻章	CS203	程式語言	327-S

11-2-3 建立統計摘要視界-說明

- 統計摘要視界（Statistical Summary Views）屬於一種列欄子集視界或合併視界，只是使用聚合函數（Aggregate Function）產生指定欄位所需的統計資料。

11-2-3 建立統計摘要視界-範例1

SQL查詢範例：Ch11_2_3_01.sql

- 建立Students、Courses和Classes三個資料表的統計摘要視界Credits_View，這是一個合併視界，而且使用COUNT()和SUM()聚合函數（Aggregate Function）顯示每位學生的選課數和所修的總學分，如下所示：

```
CREATE VIEW Total_Credits_View AS
SELECT Students.sid, COUNT(*) AS numofcourses,
SUM(Courses.credits) AS credits
FROM Students, Courses, Classes
WHERE Students.sid = Classes.sid
   and Courses.c_no = Classes.c_no
GROUP BY Students.sid
```

11-2-3 建立統計摘要視界-結果1

SQL查詢範例：Ch11_2_3_02.sql

- 查詢Total_Credits_View視界的內容，如下所示：

```
SELECT * FROM Total_Credits_View
```

	sid	numofcourses	credits
1	S001	4	12
2	S002	2	6
3	S003	2	6
4	S004	1	3

11-2-3 建立統計摘要視界-範例2

SQL查詢範例：Ch11_2_3_03.sql

- 請修改統計摘要視界Total_Credits_View，建立只顯示學生所修總學分超過6個學分的學生選課總數，和學分數的合併視界Top_Creadits_View，如下所示：

```
CREATE VIEW Top_Credits_View AS
SELECT Students.sid, COUNT(*) AS numofcourses,
SUM(Courses.credits) AS credits
FROM Students, Courses, Classes
WHERE Students.sid = Classes.sid
   and Courses.c_no = Classes.c_no
GROUP BY Students.sid
HAVING SUM(Courses.credits) >= 6
```

11-2-3 建立統計摘要視界-結果2

SQL查詢範例：Ch11_2_3_04.sql

- 查詢Top_Credits_View視界的內容，如下所示：

```
SELECT * FROM Top_Credits_View
```

	sid	numofcourses	credits
1	S001	4	12
2	S002	2	6
3	S003	2	6

11-2-4 從其他視界建立視界-說明

- 視界不只可以從基底關聯表導出，如果有已經存在的視界，我們也可以從現有視界來建立新視界。

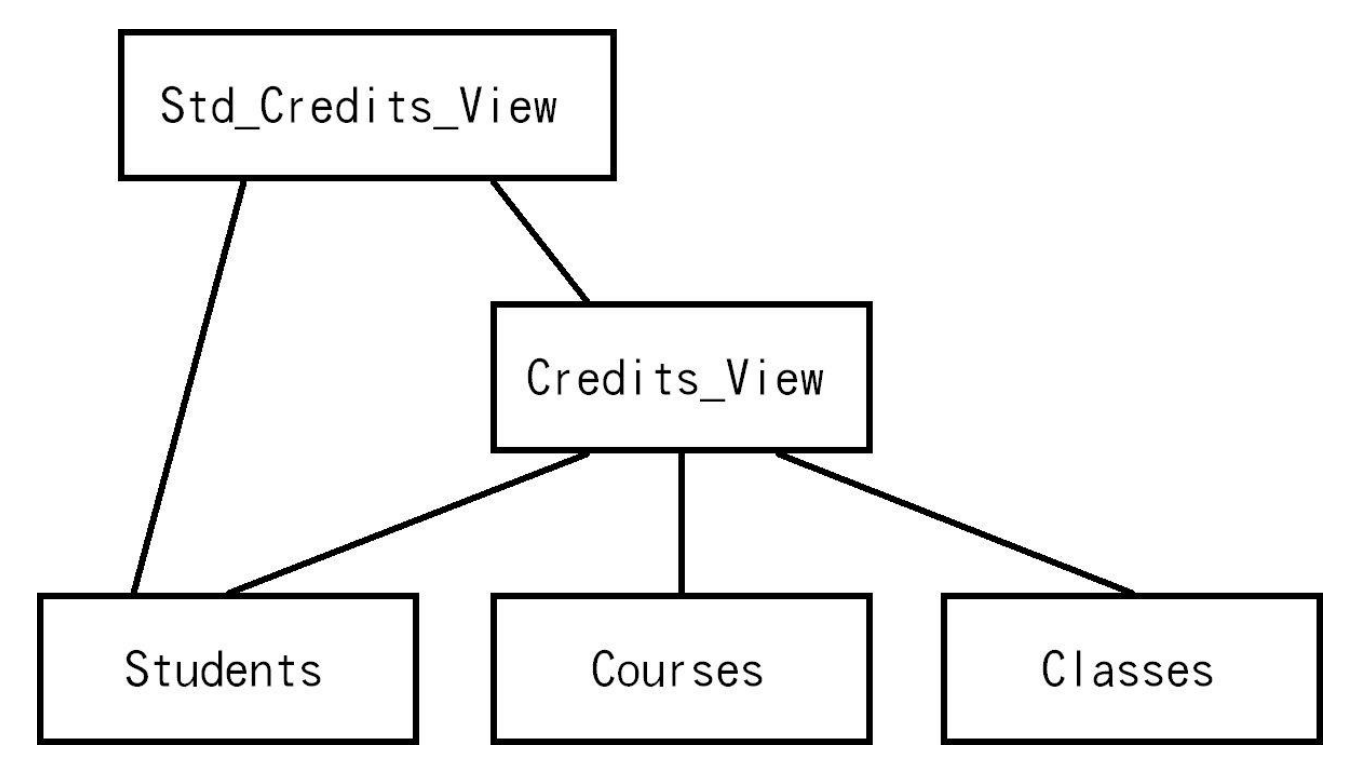
11-2-4 從其他視界建立視界-範例

SQL查詢範例：Ch11_2_4_01.sql

- 在上一節的Total_Credits_View視界只顯示學號，我們可以再次使用此視界和Students基底關聯表，建立合併視界Std_Credits_View顯示學生姓名name和電話tel欄位的詳細資料，如下所示：

```
CREATE VIEW Std_Credits_View AS
SELECT Total_Credits_View.*, Students.name, Students.tel
FROM Students, Total_Credits_View
WHERE Students.sid = Total_Credits_View.sid
```

11-2-4 從其他視界建立視界-圖例



11-2-4 從其他視界建立視界-結果

SQL查詢範例：Ch11_2_4_02.sql

- 查詢Std_Credits_View視界的內容，如下所示：

```
SELECT * FROM Std_Credits_View
```

	sid	numofcourses	credits	name	tel
1	S001	4	12	陳會安	02-22222222
2	S002	2	6	江小魚	03-33333333
3	S003	2	6	張三丰	04-44444444
4	S004	1	3	李四方	05-55555555

11-3 編輯視界的內容

- 11-3-1 視界編輯的基礎
- 11-3-2 從視界新增資料表的記錄
- 11-3-3 從視界更新資料表的記錄
- 11-3-4 從視界刪除資料表的記錄

11-3-1 視界編輯的基礎-限制條件

- 視界需要包含資料表的主鍵。
- 在CREATE VIEW指令的select_statement指令不可包含DISTINCT、聚合函數、GROUP BY和HAVING子句，如果有，視界就只能查詢，換句話說，統計摘要視界擁有聚合函數，所以只能查詢，而不能新增、更新和刪除記錄。
- 因為視界是從基底關聯表導出，所以新增、更新和刪除操作仍然需要遵守其來源基底關聯表的完整性限制條件。

11-3-1 視界編輯的基礎-WITH CHECK OPTION(語法)

- WITH CHECK OPTION指令是CREATE VIEW指令的選項，在建立視界時加上此選項，表示新增、更新和刪除記錄時，需要檢查完整性限制條件，如果不符合條件，就顯示錯誤訊息。
- WITH CHECK OPTION指令的基本語法如下所示：

```
CREATE VIEW view_name AS
select_statement
WITH CHECK OPTION
```

11-3-1 視界編輯的基礎-WITH CHECK OPTION(範例1)

SQL查詢範例：Ch11_3_1_01.sql

- 使用Students資料表建立學生生日資料的Birthday_View視界，而且使用WITH CHECK OPTION選項指令，如下所示：

```
CREATE VIEW Birthday_View AS
SELECT sid, name, birthday FROM Students
WITH CHECK OPTION
```

- 上述SQL指令建立名為Birthday_View的視界，此視界滿足前述限制條件。

11-3-1 視界編輯的基礎-WITH CHECK OPTION(範例2)

SQL查詢範例：Ch11_3_1_02.sql

- 建立學生生日資料的NP_Birthday_View視界，此視界不含資料表主鍵sid，同時新增一筆測試記錄S007，如下所示：

```
CREATE VIEW Birthday_View AS
SELECT sid, name, birthday FROM Students
WITH CHECK OPTION
GO
INSERT INTO Students VALUES
('S007','江峰','07-77777777','1965/05/23')
```

- 上述SQL指令建立的NP_Birthday_View視界因為沒有主鍵，所以不滿足前述限制條件。

11-3-2 從視界新增資料表的記錄-Birthday_View視界

SQL查詢範例：Ch11_3_2_01.sql

- 在Birthday_View視界新增一筆學生記錄，如下所示：

```
INSERT INTO Birthday_View
VALUES ('S006', '江峰' , '1966-10-01' )
```

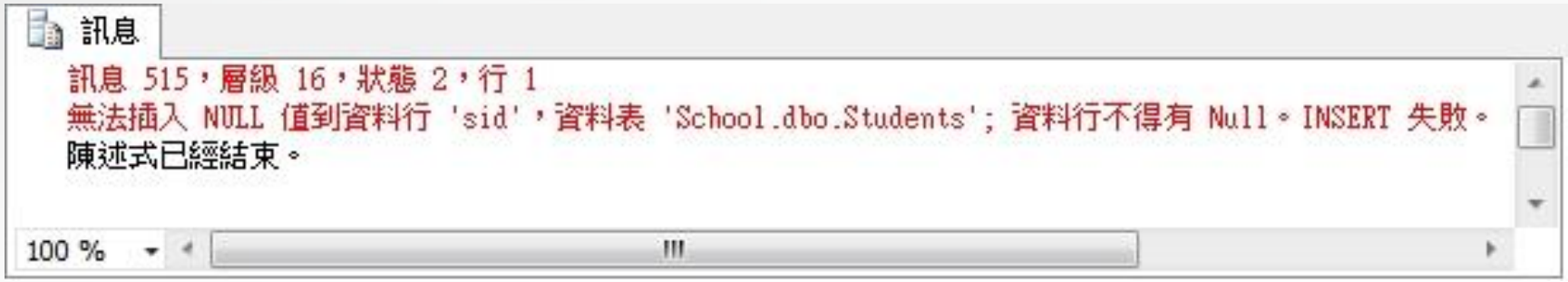
	sid	name	tel	birthday
1	S001	陳會安	02-22222222	1967-09-03
2	S002	江小魚	03-33333333	1978-02-02
3	S003	張三丰	04-44444444	1982-03-03
4	S004	李四方	05-55555555	1981-04-04
5	S005	陳允傑	02-22222222	1966-09-03
6	S006	江峰	NULL	1966-10-01
7	S007	江峰	07-77777777	1965-05-23

11-3-2 從視界新增資料表的記錄-No_Birthday_View視界

SQL查詢範例：Ch11_3_2_03.sql

- 在No_Birthday_View視界新增一筆學生記錄，如下所示：

```
INSERT INTO NP_Birthday_View
VALUES (‘江峰年’ , ‘1966-10-01’ )
```



11-3-3 從視界更新資料表的記錄-Birthday_View視界

SQL查詢範例：Ch11_3_3_01.sql

- 在Birthday_View視界將學號S006學生的birthday改為1966-02-01，如下所示：

```
UPDATE Birthday_View
SET birthday = '1966-02-01' WHERE sid = 'S006'
```

	sid	name	tel	birthday
1	S001	陳會安	02-22222222	1967-09-03
2	S002	江小魚	03-33333333	1978-02-02
3	S003	張三丰	04-44444444	1982-03-03
4	S004	李四方	05-55555555	1981-04-04
5	S005	陳允傑	02-22222222	1966-09-03
6	S006	江峰	NULL	1966-02-01
7	S007	江峰	07-77777777	1965-05-23

11-3-3 從視界更新資料表的記錄-No_Birthday_View視界

SQL查詢範例：Ch11_3_3_03.sql

- 在NP_Birthday_View視界將學生姓名'江峰'的birthday改為1966-02-01，如下所示：

```
UPDATE NP_Birthday_View
SET birthday = '1966-02-01' WHERE name = '江峰'
```

- 當執行上述SQL指令，SQL Server仍然會更新Students資料表的記錄S006和S007。理論上，資料庫管理系統應該避免在沒有主鍵的視界執行更新操作。

11-3-4 從視界刪除資料表的記錄-Birthday_View視界

SQL查詢範例：Ch11_3_4_01.sql

- 在Birthday_View視界刪除學號S006的學生資料，如下所示：

```
DELETE FROM Birthday_View
WHERE sid = 'S006'
```

	sid	name	tel	birthday
1	S001	陳會安	02-22222222	1967-09-03
2	S002	江小魚	03-33333333	1978-02-02
3	S003	張三丰	04-44444444	1982-03-03
4	S004	李四方	05-55555555	1981-04-04
5	S005	陳允傑	02-22222222	1966-09-03
6	S007	江峰	07-77777777	1966-02-01

11-3-4 從視界刪除資料表的記錄-No_Birthday_View視界

SQL查詢範例：Ch11_3_4_03.sql

- 在NP_Birthday_View視界刪除學生江峰，如下所示：

```
DELETE NP_Birthday_View
WHERE name = '江峰'
```

- 當執行上述SQL指令，SQL Server仍然會刪除Students資料表的記錄S007。理論上，資料庫管理系統應該避免在沒有主鍵的視界執行刪除操作。

11-4 資料庫程式設計的基礎

- 11-4-1 資料庫程式設計的程式語言
- 11-4-2 資料庫程式設計的實作

11-4 資料庫程式設計的基礎

- 資料庫程式設計（Database Programming）的目的是建立資料庫系統的應用程式，以主從架構的資料庫系統來說，就是指客戶端應用程式，因為應用程式需要存取資料庫的資料，所以，SQL結構化查詢語言扮演十分重要的角色。

11-4-1 資料庫程式設計的語言-說明

- SQL語言屬於一種宣告式的高階語言，可以很方便和容易存取關聯式資料庫的語言。不過，SQL語言並不適合處理一般程式邏輯，雖然Transact-SQL擴充SQL語言的功能，增加很多批次指令可以建立程式邏輯的條件和迴圈，不過功能仍然有限。
- 換句話說，單純使用SQL語言並不足以建立所需的應用程式，因為SQL語言並沒有通用用途的程式語言的功能，例如：VB、C/C++和Java語言。

11-4-1 資料庫程式設計的語言-分工

- SQL語言：負責資料庫查詢和操作的資料存取，分為兩種：「嵌入式SQL」（Embedded SQL）和「動態SQL」（Dynamic SQL）。
- 通用用途的程式語言：負責處理其他操作的商業邏輯和使用介面，稱為「宿主語言」（Host Languages）。

11-4-2 資料庫程式設計的實作-解決方案

- **特殊語法和預先編譯的嵌入式SQL**：嵌入式SQL是將SQL指令置於宿主語言的程式檔案，資料庫管理系統提供「前置處理程式」（Pre-processor）將程式檔案中的嵌入式SQL指令編譯成純宿主語言的程式碼檔案，然後使用宿主語言的編譯程式編譯成應用程式。
- **資料庫函數庫與動態SQL**：在宿主語言的程式碼檔案中，SQL指令是一個字串資料型態的變數，應用程式是在執行階段才使用資料庫函數庫的函數送出SQL指令字串執行資料庫存取，稱為動態SQL。

11-4-2 資料庫程式設計的實作-資料庫函數庫

- 資料庫程式設計可以在宿主語言直接使用「資料庫函數庫」（Database API），API的全名是（Application Programming Interface）。函數庫提供函數使用參數方式傳入SQL指令字串，然後執行資料庫存取。
- 資料庫函數庫可以分為兩種，如下所示：
  - **原生資料庫函數庫（Native Database API）**：資料庫管理系統提供的資料庫函數庫。
  - **中介層的資料庫函數庫（Middle Layer Database API）**：中介軟體提供的資料庫函數庫。

11-5 嵌入式SQL

- 11-5-1 嵌入式SQL的基礎
- 11-5-2 嵌入式SQL的程式架構
- 11-5-3 空值的處理
- 11-5-4 指標的使用

11-5 嵌入式SQL

- 「嵌入式SQL」（Embedded SQL）是ANSI-SQL 92的標準，可以使用特殊語法將SQL指令包含在通用用途程式語言的程式碼檔案，例如：VB、C/C++和Java語言等，這些語言稱為宿主語言。

11-5-1 嵌入式SQL的基礎-語法

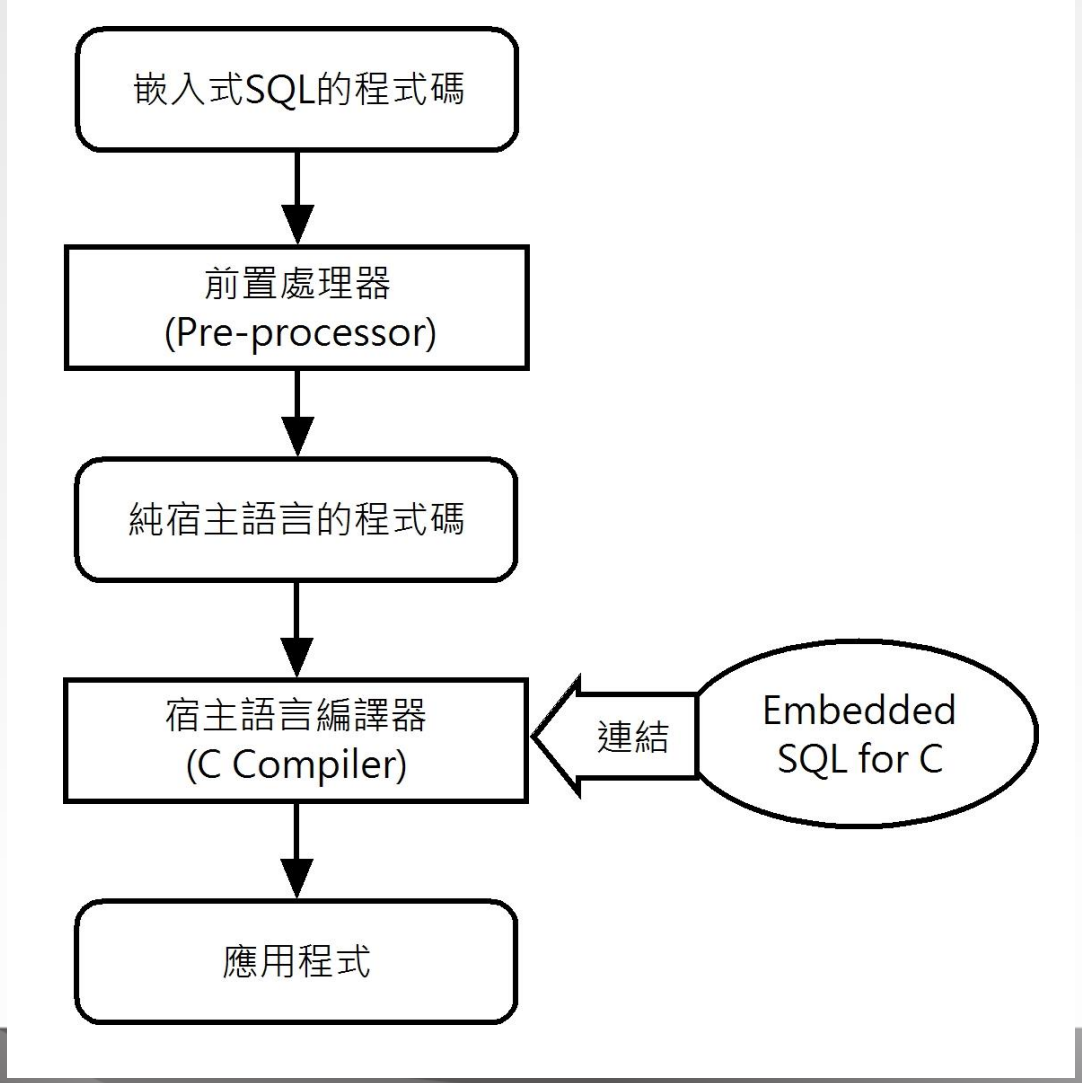
- 嵌入式SQL是將SQL指令置於宿主語言的程式碼中，為了分辨那些部分是嵌入式SQL，嵌入式SQL的指令是以EXEC SQL開頭的指令敘述，其基本語法如下所示：

```
EXEC SQL embedded_sql_statement;
```

11-5-1 嵌入式SQL的基礎-編譯過程

- 嵌入式SQL程式碼需要兩個步驟的編譯過程，如下所示：
  - 使用前置處理器轉換成一系列的函數呼叫（Function Call），以SQL Server為例，前置處理器名為nsqlprep.exe，可以將嵌入式SQL轉換成C語言的Embedded SQL for C（簡稱ESQL/C）函數呼叫。
  - 嵌入式SQL程式轉換成宿主語言程式碼後，就可以使用宿主語言的編譯器連結ESQL/C函數庫檔案建立應用程式。

11-5-1 嵌入式SQL的基礎-編譯圖例



11-5-1 嵌入式SQL的基礎-優點

- 嵌入式SQL的優點，如下所示：
  - 嵌入式SQL可以處理SQL指令和宿主語言變數間的資料交換，這些變數稱為「宿主變數」（Host Variables），也就是將SQL查詢結果的資料傳入宿主變數，反過來說，SQL查詢指令也可以直接使用宿主變數來建立查詢條件。
  - 嵌入式SQL的SQL指令敘述在編譯階段就會進行語法檢查。
  - 嵌入式SQL是ANSI-SQL 92的標準，所以很多資料庫管理系統都支援標準的嵌入式SQL。

11-5-2 嵌入式SQL的程式架構-架構1

```
#include <stdio.h>
#include <string.h>
/* 宣告宿主變數 */
EXEC SQL BEGIN DECLARE SECTION;
    char stdID[4];
    char stdname[10];
    char stdtel[12];
    char stdbirthday[12];
EXEC SQL END DECLARE SECTION;
……
```

11-5-2 嵌入式SQL的程式架構-架構2

```
int main() {
……
    strcpy(stdID, “S010”);
    strcpy(stdname, “陳蘭皋” );
    strcpy(stdtel, “04-12345678”);
    strcpy(stdIbirthday, “1956/10/23”);
    /* 執行新增操作, 新增學號S010 */
    EXEC SQL INSERT INTO Students
         VALUES (:stdID, :stdname, :stdtel, :stdbirthday);
    /* 執行更新操作, 更新學號S010 */
    EXEC SQL UPDATE Students
         SET birthday = '1966-02-01' WHERE sid = :stdID;
```

11-5-2 嵌入式SQL的程式架構-架構3

```
        /* 執行刪除操作, 刪除學號S010 */
    EXEC SQL DELETE FROM Students
        WHERE sid = :stdID;
    strcpy(stdID, “S001”);
    /* 執行SQL查詢，查詢學號S001的學生姓名 */
    EXEC SQL SELECT name INTO :stdname
        FROM Students
        WHERE sid = :stdID;
    /* 顯示學生姓名 */
    printf("學生姓名: %s\n%", stdname);
……
    return (0);
}
```

11-5-2 嵌入式SQL的程式架構-說明1

宣告宿主變數

- 在嵌入式SQL指令和C程式語言的變數交換機制是宿主變數（Host Variables），這是使用DECLAR SECTION宣告的C語言變數，如下：

```
EXEC SQL BEGIN DECLARE SECTION;
    char stdID[4];
    char stdname[10];
    char stdtel[12];
    char stdbirthday[12];
EXEC SQL END DECLARE SECTION;
```

11-5-2 嵌入式SQL的程式架構-說明2

在嵌入式SQL指令使用宿主變數

- 在嵌入式SQL指令使用宿主變數，就是使用C語言的變數值作為SQL指令條件，或用來建立SQL指令敘述，這是使用’ :’開頭的宿主變數。
- 更新記錄的嵌入式SQL指令UPDATE，如下所示：

```
EXEC SQL UPDATE Students
  SET birthday = '1966-02-01' WHERE sid = :stdID;
```

- 上述UPDATE指令可以更新資料表的記錄，WHERE子句的條件是使用宿主變數:stdID。

11-5-2 嵌入式SQL的程式架構-說明3

使用宿主變數取得查詢結果

- 我們也可以使用使用宿主變數取得查詢結果，查詢記錄的嵌入式SQL指令SELECT，如下所示：

```
EXEC SQL SELECT name INTO :stdname
    FROM Students
    WHERE sid = :stdID;
```

- SELECT INTO指令可以取得資料表的一筆記錄，條件是宿主變數:stdID，取得查詢結果的學生姓名name欄位值是存入INTO指令後的:stdname宿主變數。

11-5-3 空值的處理

- 在嵌入式SQL新增「指示變數」（Indicators）的旗標變數，以判斷宿主變數是否是空值，如下所示：

```
EXEC SQL BEGIN DECLARE SECTION;
   char stdID[4];
   char stdname[10];
   char stdtel[12];
   int name_ind;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT name, tel
     INTO :stdname:name_ind, :stdtel
     FROM Students
     WHERE sid = :stdID;
/* 不是空值 */
If ( name_ind == 0 ) {   }
```

11-5-4 指標的使用-說明

- 嵌入式SQL提供「指標」（Cursor），可以取得記錄集合中的每一筆記錄，我們可以將它視為是一個資料列標籤（Row Marker），記錄在記錄集合中，目前存取的是那一筆記錄。

11-5-4 指標的使用-宣告語法

宣告指標（Cursor）

- 嵌入式SQL使用指標（Cursor）需要事先宣告，其宣告語法，如下所示：

```
EXEC SQL DECLARE cursor_name CURSOR FOR
    select_statement;
```

- 語法宣告建立名為cursor_name的指標，在FOR指令之後是取得記錄集合的SELECT查詢指令。

11-5-4 指標的使用-宣告範例

- 例如：在嵌入式SQL程式碼檔案宣告名為std_cursor的指標，如下所示：

```
EXEC SQL DECLARE std_cursor CURSOR FOR
    SELECT sid, name, birthday, tel
    FROM Students
    WHERE birthday<='1978-02-02';
```

11-5-4 指標的使用-開啟指標

開啟指標

- 在宣告指標之後，我們需要使用OPEN指令開啟指標，如下所示：

```
EXEC SQL OPEN std_cursor;
```

11-5-4 指標的使用-FETCH語法

取得查詢結果

- 在開啟指標後，就可以使用FETCH指令從指標位置取得指定列的記錄，其語法如下所示：

```
EXEC SQL FETCH [ [ NEXT | PRIOR | FIRST | LAST ]
    FROM ] cursor_name
INTO :host_var1 [, host_var2...]
```

- 語法是將目前cursor_name位置的記錄資料存入INTO指令後的宿主變數。

11-5-4 指標的使用-FETCH參數

- 在FETCH指令後可以指定指標的移動方式，如下所示：
  - **NEXT**：這是預設移動方式，如果是第一次執行FETCH指令，就是取得記錄集合中的第一筆記錄，否則就是移到記錄集合目位置的下一筆記錄。
  - **PRIOR**：取得上一筆記錄。
  - **FIRST**：將指標移到第一筆，取得第一筆記錄。
  - **LAST**：將指標移到最後一筆，取得最後一筆記錄。

11-5-4 指標的使用-FETCH範例

- FETCH指令一次可以取得一筆記錄，如果需要取得記錄集合的每一筆記錄，就需要配合C語言的while迴圈指令，如下所示：

```
while (SQLCODE == 0) {
  EXEC SQL
    FETCH std_cursor
   INTO :stdID, :stdname, :stdbirthday, :stdtel;
  if (SQLCODE == 0)
    printf("%4s %12s %s %f\n", stdID, stdname, stdbirthday,
   stdtel);
}
```

11-5-4 指標的使用-SQLCODE變數說明

- SQLCODE變數是C語言SQLCA（SQL Communications Area）結構的欄位，SQL Server前置處理程式會自動在嵌入式SQL應用程式加入SQLCA結構，這是嵌入式SQL的錯誤處理機制。
- 我們也可以使用INCLUDE指令在程式檔案開頭加入此結構，如下所示：

```
EXEC SQL INCLUDE SQLCA;
```

11-5-4 指標的使用-SQLCODE變數種類

- 嵌入式SQL程式碼可以檢查此結構的欄位，以了解嵌入式SQL指令的執行狀態，主要的欄位說明，如下所示：
  - **SQLCODE**：負值的SQL Server錯誤碼，0表示執行成功。
  - **SQLWARN**：旗標變數，如果設定，表示有異常的例外情況發生。
  - **SQLERRM**：錯誤訊息的說明字串。
  - **SQLERRD1**：錯誤碼。
  - **SQLERRD3**：這是一個陣列，指出受影響的記錄編號。
  - **SQLSTATE**：ANSI-SQL 92標準執行階段錯誤碼。

11-5-4 指標的使用-關閉指標

關閉指標

- 當指標不再需要時，請使用CLOSE指令關閉指標，釋放查詢結果記錄集合所佔用的記憶體空間，如下所示：

```
EXEC SQL CLOSE std_cursor;
```

11-6 動態SQL與ORM

- 11-6-1 嵌入式SQL的動態SQL指令
- 11-6-2 使用資料庫函數庫執行動態SQL指令
- 11-6-3 Object-relational Mapping(ORM)

11-6 動態SQL與ORM-說明

- 動態SQL（Dynamic SQL）是對比「靜態SQL」（Static SQL），其說明如下所示：
  - **靜態SQL**：SQL指令是在編譯階段就決定，預先編譯程式可以執行資料型態的檢查，嵌入式SQL就是一種靜態SQL。
  - **動態SQL**：SQL指令是動態在執行階段才產生，這些指令是儲存成宿主語言的字串變數，在執行階段才送到資料庫管理系統執行。

11-6-1 嵌入式SQL的動態SQL指令-說明

- 動態SQL的重點是在執行階後才建立SQL指令，在嵌入式SQL也提供2個動態SQL指令（Transact-SQL支援），如下所示：
  - **PREPARE指令**：建立SQL查詢指令字串，通常是宿主語言的字串變數，字串可以內含’ ?’號的參數。
  - **EXECUTE指令**：執行PREPARE指令的SQL查詢指令。

11-6-1 嵌入式SQL的動態SQL指令-範例1

- 首先使用建立儲存SQL指令字串的宿主變數sqlstring，其程式碼如下所示：

```
EXEC SQL BEGIN DECLARE SECTION;
    char sqlstring[255];
    char desiredID[20];
EXEC SQL END DECLARE SECTION;
```

- 接著指定sqlstring字串內容的SQL指令，條件sid欄位是一個參數，如下所示：

```
sqlstring = “DELETE FROM Students WHERE sid= ?”;
```

11-6-1 嵌入式SQL的動態SQL指令-範例2

- 然後使用PREPARE指令以宿主變數建立SQL查詢，如下所示：

```
EXEC SQL PREPARE sqlDelete FROM :sqlstring;
```

- 最後就是以使用者輸入的學號變數desiredID，使用EXECUTE執行查詢，如下所示：

```
EXEC SQL EXECUTE sqlDelete USING :desiredID;
```

11-6-2 使用資料庫函數庫執行動態SQL指令

- 我們可以在程式碼呼叫資料庫函數庫的函數將完整SQL指令字串，在執行時以參數方式送給資料庫管理系統執行。例如：ASP.NET技術使用ADO.NET元件的Connection物件，透過OLE DB中介軟體來執行動態SQL指令，如下所示：

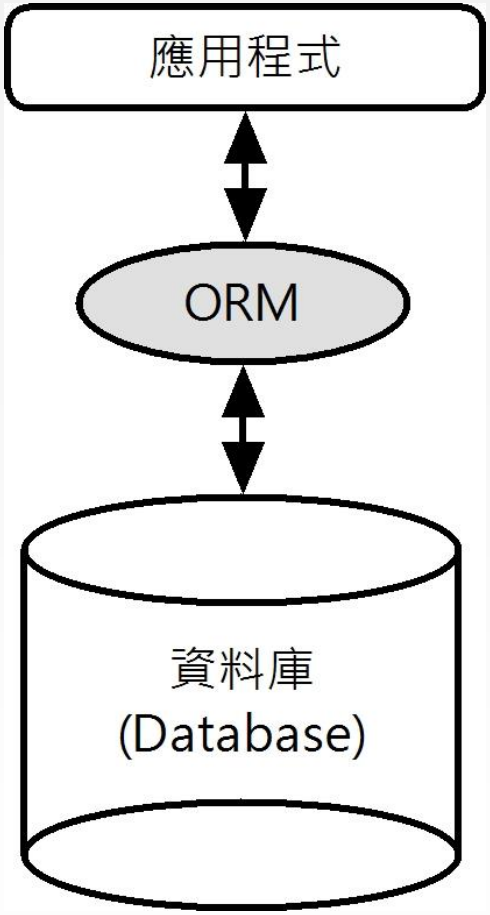
```
strSQL = "INSERT INTO Students (sid, name"  & _
        ",tel, birthday) "
……
strSQL &= "'" & birthday.Text & "')"
objCon = New SqlConnection(strDbCon)
objCon.Open()
objCmd = New SqlCommand(strSQL, objCon)
count = objCmd.ExecuteNonQuery()
```

11-6-3 Object-relational Mapping(ORM)-說明

- ORM（Object-relational Mapping）也稱為ORM或O/R mapping，這是一種應用程式開發技術的中介軟體，可以轉換不同型態資料成為物件導向程式語言（Object-oriented Language）的物件（Object），讓程式設計者專注於物件的存取和操作，而不用考量物件背後連接的資料。

11-6-3 Object-relational Mapping(ORM)-ORM與關聯式資料庫

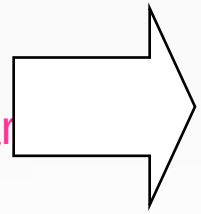
- 以關聯式資料庫來說，ORM是將資料庫眾多資料表一一轉換成物件導向程式語言的物件，也就是建立資料庫各資料表與程式語言物件之間的對應（Mapping），可以自動提供資料表與物件之間的資料轉換，例如：Java、C++、PHP、Visual Basic和C#等程式語言，如右圖所示：



11-6-3 Object-relational Mapping(ORM)-為什麼使用ORM

- 當我們使用物件導向程式語言建立資料庫應用程式來存取資料庫時，馬上面對的問題是程式語言的物件模型（Object Model）和資料庫的關聯式資料庫模型並不相容。

```
public class Employee {
   private int id;
   private String name;
   private int salary;
   public Employee() {}
   public Employee(String name, int salar
      this.name = name;
      this.salary = salary;
   }
   public int getId() { return id; }
   public String getName() { return name; }
   public int getSalary() { return salary; }
}
```



```
create table Employee (
   id INT NOT NULL auto_increment,
   name VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

11-6-3 Object-relational Mapping(ORM)-ORM的優點

- 增加生產力：因為資料存取程式碼通常都是應用程式很重要和主要部分，使用ORM可以自動產生對應資料模型的存取程式碼，大幅減少應用程式的開發時程。
- 建立更佳的應用程式設計：ORM可以分割資料存取和商業邏輯的程式碼，幫助我們建立更佳的應用程式設計。
- 建立可重複使用的程式碼：ORM自動產生的資料存取程式碼可以重複使用在其他應用程式開發。
- 幫助我們維護應用程式：因為ORM自動產生的程式碼都是經過良好測試的程式碼，並不用特別維護，而且當更改資料來源時，我們只需重新使用ORM產生資料存取程式碼，即可以快速建立適用在不同資料的應用程式，而不用動到商業邏輯的程式碼。

11-6-3 Object-relational Mapping(ORM)-ORM的缺點

- ORM的主要缺點是在效能（Performance），因為資料需要對應和轉換，一般來說，ORM自動建立的物件程式碼通常比直接撰寫程式碼存取資料庫來的複雜，而複雜的程式碼會降低程式的執行效能。

11-7 Transact-SQL的預存程序

- 11-7-1 預存程序的基礎
- 11-7-2 建立預存程序
- 11-7-3 執行預存程序

11-7 Transact-SQL的預存程序

- Transact-SQL支援程式化功能，可以撰寫SQL程式檔案的批次指令、預存程序和觸發程序，其說明如下所示：
  - **批次指令（Batch）**：提供IF/ELSE、GOTO、WHILE、BREAK、CONTINUE等條件或迴圈指令。
  - **預存程序（Stored Procedure）**：將例行、常用和複雜的資料庫操作預先建立成SQL指令的程式檔，可以簡化相關的資料庫操作。
  - **觸發程序（Trigger）**：一種特殊用途的預存程序，不過是主動執行的程序，當資料表操作符合指定的條件時，就會自動執行觸發程序。

11-7-1 預存程序的基礎-說明

- 「預存程序」（Stored Procedure）是一個個程序，每一個程序可以執行所需的資料庫操作，它一樣可以使用Transact-SQL的流程控制指令，撰寫出複雜的資料庫操作功能。
- 換句話說，我們可以將目前的資料庫相關的查詢指令轉換成預存程序。

11-7-1 預存程序的基礎-範例1

- 例如：在Management Studio輸入SELECT查詢指令，如下所示：

```
SELECT column1, column2 FROM table
```

- SQL指令取出資料表table的2個欄位，將這個SQL指令轉換成預存程序，程序內容如下所示：

```
CREATE PROCEDURE MyStoredProcedure
AS
SELECT column1, column2 FROM table
GO
```

11-7-1 預存程序的基礎-範例2

- 預存程序也可以傳遞參數，value值是傳入的參數，如下所示：

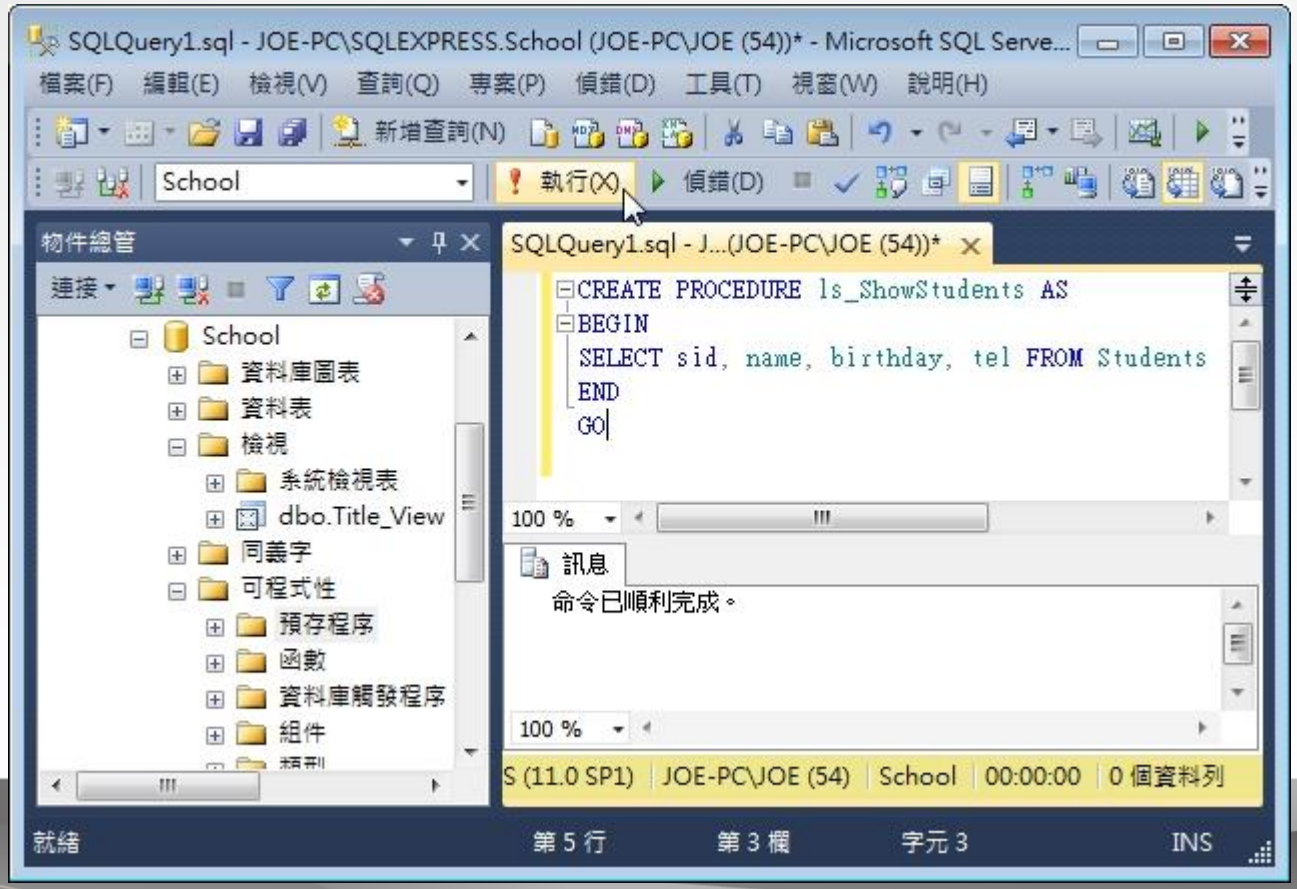
```
CREATE PROCEDURE MyStoredProcedure
    @MyPara
AS
SELECT column1, column2 FROM table WHERE column1 =
    @MyPara
GO
```

- 執行預存程序就需要加上傳入的參數MyPara，如下所示：

```
MyStoredProcedure 10
```

11-7-2 建立預存程序-在Management Studio建立預存程序

- 在SQL Sever可以使用Management Studio建立和測試預存程序。



11-7-2 建立預存程序-新增查詢建立預存程序

- 因為Management Studio指令建立預存程序是編輯和執行SQL指令碼檔案，我們可以直接按【新增查詢】鈕新增查詢編輯視窗後，自行輸入建立預存程序的SQL指令敘述。

SQL指令碼檔：ls_ShowStudents_p.sql

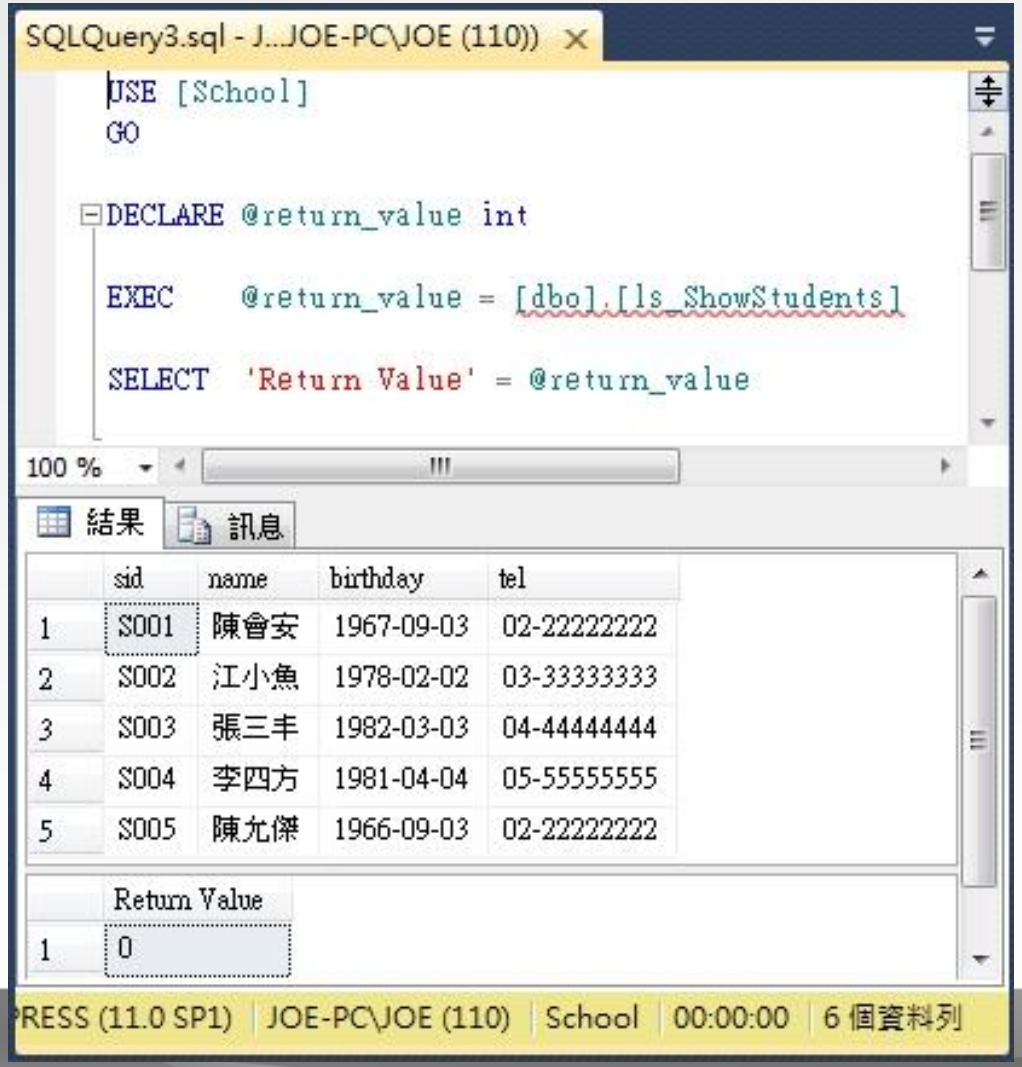
- 建立顯示學生個人資料的ls_ShowStudents_p預存程序，擁有參數的學號來顯示學生資料，如下所示：

```
CREATE PROCEDURE ls_ShowStudents_p
      @sid VARCHAR(4)
AS
BEGIN
SELECT sid, name, birthday, tel FROM Students
WHERE sid = @sid
END
```

11-7-3 測試預存程序-Management Studio(步驟)

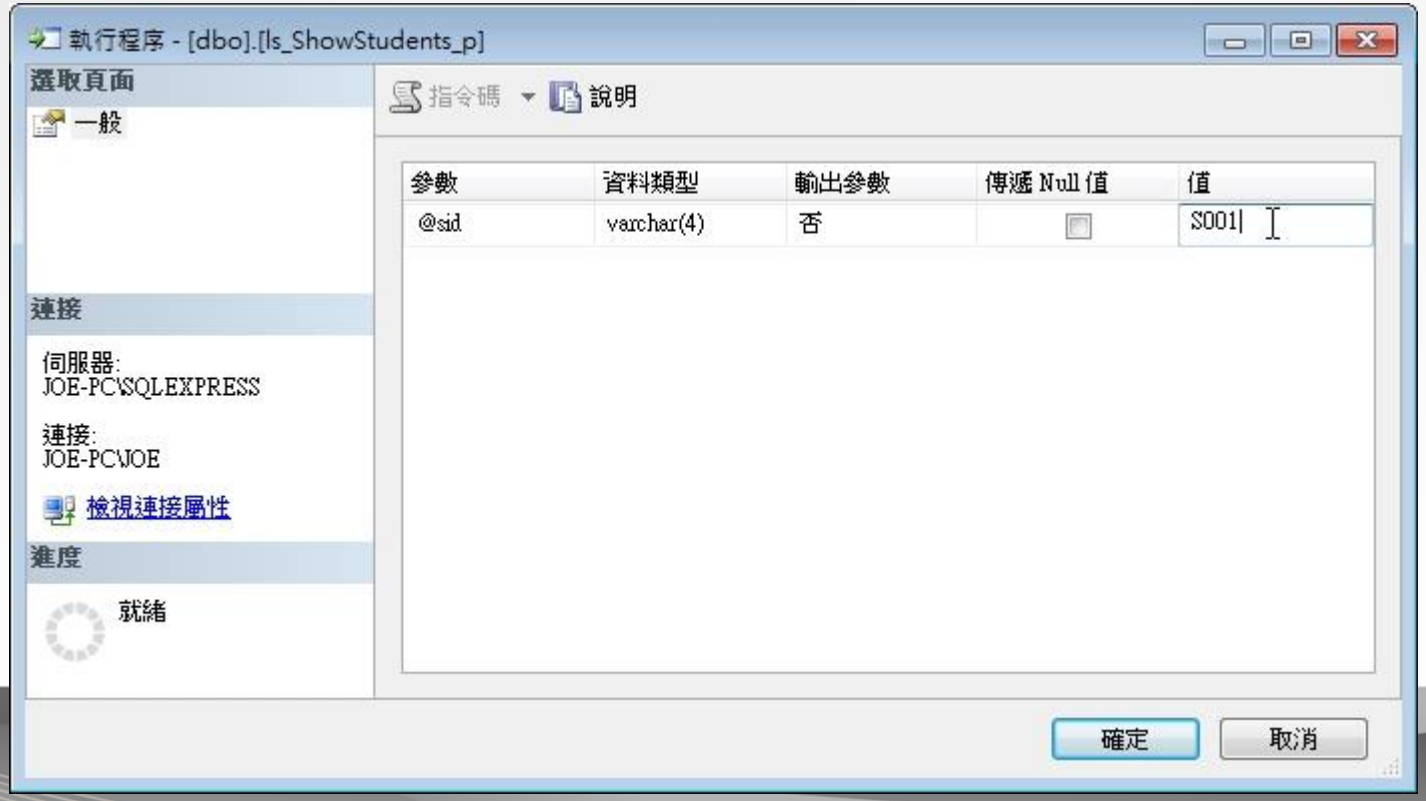
- 現在我們已經成功建立2個預存程序，可以分別顯示Students資料表的記錄和使用WHERE子句的條件。
- 首先測試名為ls_ShowStudents的預存程序，請在Management Studio展開【School】資料庫的預存程序。
- 在ls_ShowStudents預存程序上，執行【右】鍵快顯功能表的【執行預存程序】指令後，可以看到「執行程序」對話方塊。按【確定】鈕，稍等一下，可以看到執行結果。

11-7-3 測試預存程序-Management Studio(結果)



11-7-3 測試預存程序-輸入參數

- 如果是執行ls_ShowStudents_p預存程序，因為擁有參數，在「執行程序」對話方塊的【值】欄可以輸入參數值，如下圖所示：




End