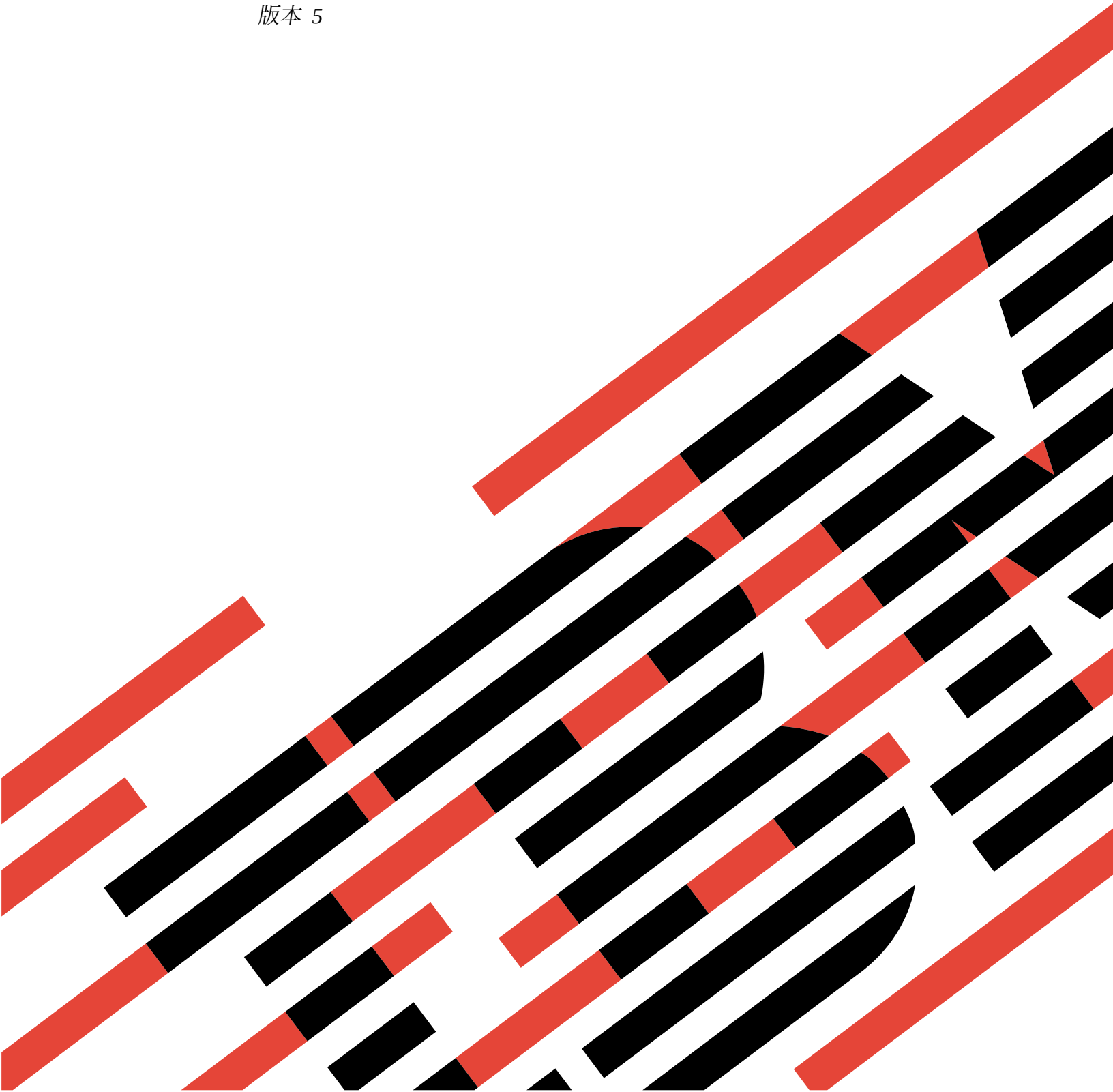
IBM

eserver

iSeries

DB2 Universal Database for iSeries SQL 程式設計概念

版本 5



IBM

@server

iSeries

DB2 Universal Database for iSeries SQL 程式設計概念

版本 5

目錄

關於 DB2 UDB for iSeries SQL 程式設計概念

本書說明程式設計師與資料庫管理者需要知道的一些基本 SQL 程式設計概念：

- 如何使用 DB2 UDB for iSeries 授權程式
- 如何存取資料庫中的資料
- 如何準備、執行及測試含有 SQL 陳述式的應用程式

有關在應用程式設計環境中施行的 DB2 UDB for iSeries SQL 指引及範例的詳細資訊，請參閱 iSeries 資訊中心中的下列書籍。

- SQL Reference
- DB2 UDB for iSeries SQL Programming for Host Languages
- DB2 UDB for iSeries Database Performance and Query Optimization
- SQL Call Level Interface (ODBC)
- SQL Messages and Codes

有關本手冊的詳細資訊，請參閱下列主題：

- 『本書的讀者』
- 『與 SQL 陳述式範例相關的假設』
- 第 x 頁的『程式碼不保事項聲明』
- 第 x 頁的『如何解譯本手冊中的語法圖』
- 第 xi 頁的『V5R2 版 SQL 程式設計概念新增資訊』

本書的讀者

本手冊適合熟悉及會使用 COBOL for iSeries、ILE COBOL for iSeries、iSeries PL/I、ILE C for iSeries、ILE C++、REXX、RPG III（RPG for iSeries 的組件）或 ILE RPG for iSeries 語言來設計程式，以及瞭解基本資料庫應用程式的應用程式設計師與資料庫管理者使用。

亦請參閱下列各節：

- 『與 SQL 陳述式範例相關的假設』
- 第 x 頁的『程式碼不保事項聲明』
- 第 x 頁的『如何解譯本手冊中的語法圖』

與 SQL 陳述式範例相關的假設

本手冊中的 SQL 陳述式範例是以附錄 A，DB2 UDB for iSeries 表格範例中的表格範例爲基礎，並做下列假設：

- 它們顯示於交談式 SQL 環境中，或以 ILE C 或 COBOL 撰寫。EXEC SQL 和 END-EXEC 用於 COBOL 程式中區隔 SQL 陳述式。有關如何在 COBOL 程式中使用 SQL 陳述式的說明，請參閱「在 COBOL 應用程式中撰寫 SQL 陳述式」。有關如何 ILE C 程式中使用 SQL 陳述式的說明，請參閱「在 C 應用程式中撰寫 SQL 陳述式」。

- 每一個 SQL 範例以數行顯示，而陳述式的每一個子句位於個別行上。
- SQL 關鍵字以高亮度顯示。
- 附錄 A，DB2 UDB for iSeries 表格範例提供的表格名稱使用綱目 CORPDATA。在「表格範例」中找不到的表格名稱應使用您所建立的綱目。
- 計算直欄以括弧 () 和方括弧 [] 括住。
- 使用 SQL 命名慣例。
- 假設 APOST 和 APOSTSQL 前置編譯器選項，即使它們不是 COBOL 中的預設選項也一樣。 SQL 與主電腦語言陳述式內的字串文字以單引號 (') 區隔。
- 除非特別指示，否則使用排序順序 *HEX。
- SQL 陳述式的完整語法通常不在任何一個範例中顯示。有關本手冊中描述的任何陳述式之完整說明和語法，請參閱 SQL Reference

當範例不同於這些假設時，會加以指明。

由於本手冊適用應用程式設計師，所以大部份範例會如同在應用程式中撰寫一般顯示。不過，許多範例可能稍有改變，而使用交談式 SQL 來交談式地執行。當使用交談式 SQL 時，SQL 陳述式的語法與嵌入程式中的相同陳述式格式稍有不同。

程式碼不保事項聲明

此文件包含程式設計範例。

IBM 授與您使用所有程式設計程式碼範例的非專屬授權，您可以依據這些範例，產生類似的函數，來符合您的需要。

IBM 提供的所有範例程式碼僅做爲說明用途。這些範例尚未徹底經過所有情況的測試。因此 IBM 不擔保或默示保証這些程式的可靠性、可用性或功能。

所有內含於此的程式是以「現況」提供給您，不具任何形式的擔保。IBM 明示排除有關這些程式的不侵權、可售性、符合特定使用目的之默示擔保。

如何解譯本手冊中的語法圖

在本書中，語法是以如下定義的結構來描述：

- 請遵循路線，由左而右，由上而下來閱讀語法圖。

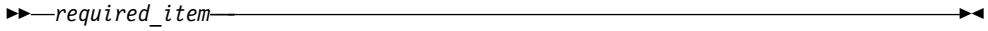
  ►►── 符號指示陳述式開頭。

  ──► 符號指示陳述式語法接續到下一行。

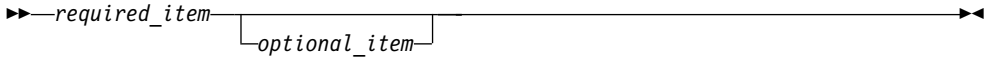
  ►── 符號指示陳述式接續自上一行。

  ──►◄ 符號指示陳述式結尾。

  語法單元圖不含以 ►── 符號開頭及以 ──► 符號結尾的完整陳述式。
- 必要的項目出現在水平線上 (主路徑)。



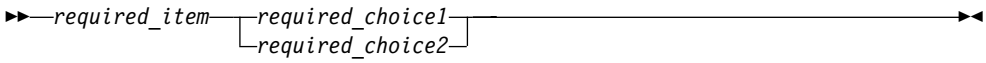
- 可選用的項目出現在主路徑下。



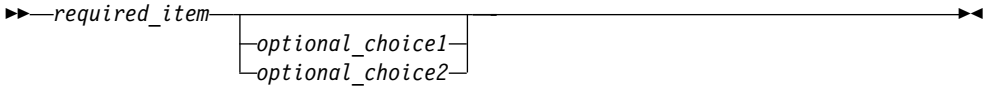
如果可選用的項目出現在主路徑上，則該項目對陳述式的執行沒有影響，並且只能供讀取。

- 如果您在兩個或多個項目中選擇，它們會在堆疊中垂直顯示。

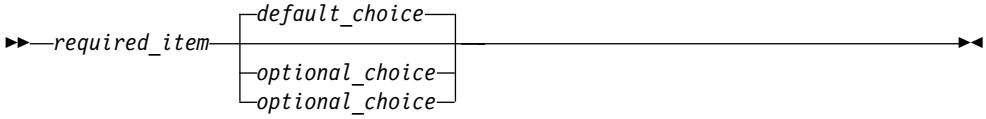
  如果您*必須*選擇其中一個項目，堆疊的其中一個項目會出現在主路徑上。



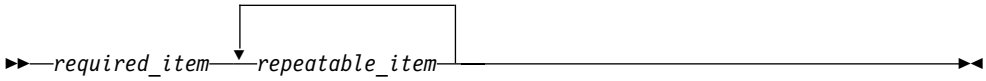
  如果選擇的其中一個項目是可選用的，則整個堆疊會出現在主路徑下。



  如果其中一個項目爲預設的，則它會出現在主路徑上，而剩餘選項在主路徑下顯示。



- 主要行上的向左箭頭指示項目可重複。



  如果重複箭頭含有逗點，您必須以逗點區隔重複的項目。

  堆疊上的重複箭頭指示您可重複堆疊中的項目。

- 關鍵字以大寫顯示 (例如，FROM)。它們必須如所示一般正確拼出。變數以全部小寫字母顯示 (例如，*column-name*)。它們代表使用者提供的名稱或值。
- 如果標點符號、括弧、算術運算子或其它符號有顯示，您必須將它們輸入爲語法的一部份。

V5R2 版 SQL 程式設計概念新增資訊

在本版次中，此資訊的主要變更如下：

- 使用者定義的表格函數。詳細資訊，請參閱第 185 頁的『使用者定義的函數 (UDF)』。
- 取消隔離的使用者定義的函數。詳細資訊，請參閱第 223 頁的『製作隔離或非隔離的函數』。
- 儲存點。詳細資訊，請參閱第 290 頁的『儲存點』。
- 暫時表格

- 識別直欄。詳細資訊，請參閱第 48 頁的『建立及變更識別直欄』。
- 純量子選取
- 對 SQL 程序進行除錯。詳細資訊，請參閱第 153 頁的『爲儲存程序除錯』。

第 1 章 DB2 UDB for iSeries 結構化查詢語言簡介

以下主題將說明 iSeries 伺服器如何使用 DB2 UDB for iSeries 與 DB2 UDB Query Manager and SQL Development Kit Version 5 授權程式來施行結構化查詢語言 (SQL)。SQL 是根據關聯式資料模型來管理資訊。 SQL 陳述式可內含在高階語言中，或以動態方式來準備並執行，或以交談方式來執行。

SQL 是由陳述式與子句所組成，它們可說明您要對資料庫中的資料執行何種作業，以及在何種狀況加以執行。

此主題將說明下列各項：

- 『SQL 概念』
- 第 5 頁的『SQL 物件』
- 第 9 頁的『應用程式物件』

SQL 採用「IBM 分散式連結資料庫架構* (DRDA*)」，可存取遠端關聯式資料庫中的資料。此功能說明於本手冊的第 21 章, 『分散式關聯資料庫功能』主題中。有關 DRDA 的進一步詳細資訊，請參閱 Distributed Database Programming 一書。

SQL 概念

DB2 UDB for iSeries SQL 包含下列主要組件：

- SQL 執行時間支援

  SQL 執行時間會剖析 SQL 陳述式並執行任何 SQL 陳述式。此種支援爲 Operating System/400* (OS/400) 授權程式的一部份，用來讓含有 SQL 陳述式的應用程式可在安裝了 DB2 UDB Query Manager and SQL Development Kit 授權程式的系統上執行。

- SQL 前置編譯器

  SQL 前置編譯器可預先編譯主語言中所內含的 SQL 陳述式。其所支援的語言如下：

  – ILE C
  – ILE C++ for iSeries
  – ILE COBOL
  – COBOL for iSeries
  – iSeries PL/I
  – RPG III (RPG for iSeries 的一部份)
  – ILE RPG

  SQL 主語言前置編譯器會先負責準備內含 SQL 陳述式的應用程式。經過前置編譯的主原始程式，再交由主語言編譯器進行編譯。有關前置編譯的詳細資訊，請參閱 *SQL Programming with Host Languages* 中的 Preparing and Running a Program with SQL Statements 主題。前置編譯器支援爲 DB2 UDB Query Manager and SQL Development Kit 授權程式的一部份。

- SQL 交談式介面

SQL 交談式介面可讓您建立及執行 SQL 陳述式。有關交談式 SQL 的詳細資訊，請參閱第 17 章, 『使用交談式 SQL』。交談式 SQL 是 DB2 UDB Query Manager and SQL Development Kit 授權程式的一部份。

- 執行 SQL Script

  iSeries 領航員中的「執行 SQL Script」視窗，可讓您建立、編輯、執行及疑難排解 SQL 陳述式的 script。「執行 SQL Script」是「iSeries 領航員」的一部份。相關詳細資訊，請參閱第 254 頁的『使用「執行 SQL Script」來查詢資料庫』。

- 執行 SQL 陳述式 CL 指令

  RUNSQLSTM 可讓您執行原始檔中所儲存一連串的 SQL 陳述式。有關「執行 SQL 陳述式」指令的詳細資訊，請參閱第 18 章, 『使用 SQL 陳述式處理器』。

- DB2 Query Manager for iSeries

  DB2 Query Manager for iSeries 具有以提示來驅動的交談式介面，可讓您在資料庫中建立資料、新增資料、維護資料及執行報表。 「Query Manager」是 DB2 UDB Query Manager and SQL Development Kit 授權程式的一部份。相關詳細資訊，請參閱 Query Manager Use 一書。

- SQL REXX 介面

  SQL REXX 介面可讓您在 REXX 程序中執行 SQL 陳述式。有關如何在 REXX 程序中使用 SQL 陳述式的詳細資訊，請參閱 *SQL Programming with Host Languages* 中的 Coding SQL Statements in REXX Applications 主題。

- SQL 呼叫層次介面

  DB2 UDB for iSeries 支援「SQL 呼叫層次介面」。它可讓任何 ILE 語言的使用者透過程序呼叫系統所提供的服務程式，來直接存取 SQL 功能。利用「SQL 呼叫層次介面」，您即可執行所有的 SQL 功能，而不需進行前置編譯。這是一種程序呼叫標準集，用於準備 SQL 陳述式、執行 SQL 陳述式、提取資料列，甚至執行進階功能，例如存取編目以及連結程式變數至輸出直欄。

  有關各類可用函數以及相關語法的完整說明，請參閱 SQL cCall Level Interface (ODBC) 一書。

- QSQPRCED API

  此種「應用程式介面 (API)」具備延伸的動態 SQL 功能。 SQL 陳述式可透過此 API 編譯成 SQL 資料包後再執行。透過此 API 而編譯成資料包的陳述式將持續存在，直到該資料包或陳述式遭捨棄爲止。 QSQPRCED 是 OS/400 授權程式的一部份。有關 QSQPRCED API 的詳細資訊，請參閱 iSeries 資訊中心其「程式設計」小節中的 QSQPRCED 主題。有關 API 的一般資訊，請參閱 iSeries 資訊中心中的 OS/400 API 主題。

- QSQCHKS API

  此 API 語法會檢查 SQL 陳述式。QSQCHKS 是 OS/400 授權程式的一部份。有關 QSQCHKS API 的詳細資訊，請參閱 iSeries 資訊中心其「程式設計」小節中的 QSQCHKS 主題。有關 API 的一般資訊，請參閱 iSeries 資訊中心中的 OS/400 API 主題。

- DB2 多系統

  此項作業系統特性可讓您的資料分散至多重伺服器。有關 DB2 多系統的詳細資訊，請參閱 DB2 Multisystem 一書。

- DB2 UDB 對稱多重程序 (SMP)

此項作業系統特性具備查詢最佳化工具，可提供額外方式來併入平行處理以擷取資料。對稱多重程序 (SMP) 是一種在單一系統上所達成的平行化，藉由讓共用記憶體與磁碟資源的多個處理器 (CPU 與 I/O 處理器) 同時運作，來實現單一端的結果。此種平行處理意味著資料庫管理程式可讓多個 (或全部) 系統處理器同時處理單一查詢。有關如何控制平行處理的詳細資訊，請參閱 *Database Performance and Query Optimization* 中的 Controlling Parallel Processing 主題。

相關詳細資訊，請參閱下列各節：

- 『SQL 關聯式資料庫與系統術語』
- 第 4 頁的『SQL 陳述式的類型』
- 第 5 頁的『SQL 通信區 (SQLCA)』

SQL 關聯式資料庫與系統術語

在關聯式資料模型下，所有資料皆會被視爲存在於表格中。 DB2 UDB for iSeries 物件的建立與維護完全比照系統物件。下表所示即爲系統術語與 SQL 關聯式資料庫術語的關係。有關如何透過傳統檔案介面來進行資料庫程式設計的詳細資訊，請參閱 Database Programming 一書。

表 1. 系統術語與 SQL 術語的關係

系統術語	SQL 術語
檔案庫。由相關物件所構成的群組，可讓您依據名稱來尋找物件。	**綱目**。由檔案庫、日誌、異動日誌接收器、SQL 編目以及資料字典 (選用) 所組成。綱目可將相關物件區分爲群組，讓您依據名稱來尋找物件。
實體檔案。記錄的集合。	**表格**。由直欄與列組成的集合。
記錄。欄位的集合。	**列**。表格的水平部份，內含一連串的直欄。
欄位。同一資料類型之一或多個字元的相關資訊。	**直欄**。相同的資料類型之表格的垂直部份。
邏輯檔案。由一或多個實體檔案的欄位與記錄所形成的子集。	**概略表**。由一或多個表格的直欄與列所形成的子集。
SQL 資料包。一種物件類型，用來執行 SQL 陳述式。	**資料包**。一種物件類型，用來執行 SQL 陳述式。
使用者設定檔	**授權名稱或授權 ID**。

另請參閱：

- 『SQL 與系統命名慣例』

SQL 與系統命名慣例

共有兩種命名慣例適用於 DB2 UDB for iSeries 程式設計：系統 (*SYS) 與 SQL (*SQL)。您所用的命名慣例將會影響檔案與表格名稱，以及交談式 SQL 顯示畫面所用術語的限定方式。您可以透過 SQL 指令的參數，或透過 SET OPTION 陳述式 (適用於 REXX)，來選取要用的命名慣例。相關的明細，請參閱 SQL Reference 中的未限定物件名稱的「限定方式」。

系統命名法 (*SYS): 在系統命名慣例中，表格與 SQL 陳述式中的其他 SQL 物件是按下述形式依綱目名稱來限定：

綱目/表格

SQL 命名法 (*SQL): 在 SQL 命名慣例中，表格與 SQL 陳述式中的其他 SQL 物件則是按下述形式依綱目名稱來限定：

```
schema.table
```

SQL 陳述式的類型

SQL 陳述式共有四種基本類型：

- 資料定義語言 (DDL) 陳述式
- 資料操作語言 (DML) 陳述式
- 動態 SQL 陳述式
- 雜項陳述式

SQL 陳述式可操作 SQL 所建立的物件，以及外部說明的實體檔案與單一格式的邏輯檔案，不論其是否常駐在 SQL 綱目中。它們並不會參照程式說明檔的 IDDU 字典定義。程式說明檔會以表格方式顯現，且僅擁有單一直欄。

SQL DDL 陳述式	SQL DML 陳述式
ALTER TABLE	CLOSE
COMMENT ON	COMMIT
CREATE ALIAS	DECLARE CURSOR
CREATE DISTINCT TYPE	DELETE
CREATE FUNCTION	FETCH
CREATE INDEX	INSERT
CREATE PROCEDURE	LOCK TABLE
CREATE SCHEMA	OPEN
CREATE TABLE	RELEASE SAVEPOINT
CREATE TRIGGER	ROLLBACK
CREATE VIEW	SAVEPOINT
DECLARE GLOBAL TEMPORARY TABLE	SELECT INTO
DROP ALIAS	SET 變數
DROP DISTINCT TYPE	UPDATE
DROP FUNCTION	VALUES INTO
DROP INDEX	
DROP PACKAGE	
DROP PROCEDURE	
DROP SCHEMA	
DROP TABLE	
DROP TRIGGER	
DROP VIEW	
GRANT DISTINCT TYPE	
GRANT FUNCTION	
GRANT PACKAGE	
GRANT PROCEDURE	
GRANT TABLE	
LABEL ON	
RENAME	
REVOKE DISTINCT TYPE	
REVOKE FUNCTION	
REVOKE PACKAGE	
REVOKE PROCEDURE	
REVOKE TABLE	

動態 SQL 陳述式	雜項陳述式
DESCRIBE	BEGIN DECLARE SECTION
EXECUTE	CALL
EXECUTE IMMEDIATE	CONNECT
PREPARE	DECLARE PROCEDURE
	DECLARE STATEMENT
	DECLARE VARIABLE
	DESCRIBE TABLE
	DISCONNECT
	END DECLARE SECTION
	FREE LOCATOR
	HOLD LOCATOR
	INCLUDE
	RELEASE
	SET CONNECTION
	SET OPTION
	SET PATH
	SET RESULT SETS
	SET SCHEMA
	SET TRANSACTION
	WHENEVER

SQL DDL 陳述式說明於第 45 頁的第 4 章, 『資料定義語言 (DDL)』中。 SQL DML 陳述式說明於第 59 頁的第 5 章, 『使用 SELECT 陳述式擷取資料』與第 87 頁的第 6 章, 『SQL 插入、更新及刪除』中。您可以在 SQL Reference 一書中，找到這些陳述式的完整說明。

SQL 通信區 (SQLCA)

SQLCA 是一組變數，它會在每一 SQL 陳述式執行結束時更新。含有可執行 SQL 陳述式的程式只能恰好提供一個 SQLCA (除非是使用獨立式 SQLCODE 或獨立式 SQLSTATE 變數)。相關詳細資訊，請參閱 iSeries 資訊中心其 *SQL Reference* 一書中的 SQL Communication Area 主題。

SQL 物件

SQL 物件包括了綱目、資料字典、日誌、編目、表格、別名、概略表、索引、限制、觸發、儲存程序、使用者定義的函數、使用者定義的類型和 SQL 資料包。 SQL 會比照系統物件的方式來建立及維護這些物件。以下是這些物件的簡短說明：

- 第 6 頁的『綱目』
- 第 6 頁的『資料字典』
- 第 6 頁的『日誌與異動日誌接收器』
- 第 6 頁的『編目』
- 第 6 頁的『表格、列和直欄』
- 第 7 頁的『別名』
- 第 7 頁的『概略表』
- 第 7 頁的『索引』
- 第 7 頁的『限制』

- 第 8 頁的『觸發』
- 第 8 頁的『儲存程序』
- 第 8 頁的『使用者定義的函數』
- 第 8 頁的『使用者定義的類型』
- 第 8 頁的『SQL 資料包』

綱目

綱目可提供 SQL 物件的邏輯分組方式。**綱目**的組成項包括檔案庫、日誌、異動日誌接收器、編目以及資料字典 (選用)。表格、概略表及系統物件 (例如程式) 可在任何系統檔案庫中建立、移動或復置。如果 SQL 綱目不含資料字典，所有系統檔案皆可在 SQL 綱目中建立或移動。但若 SQL 綱目含有資料字典，那麼：

- 來源實體檔或具有一個成員的非來源實體檔即可在 SQL 綱目中建立、移動或復置。
- 邏輯檔案不可置入 SQL 綱目中，因爲它們無法在資料字典中說明。

您可以建立及擁有許多綱目。*集合*一詞可視爲*綱目*的同義詞。

資料字典

在版本 3 版次 1 之前所建立的綱目，或已在 CREATE SCHEMA 陳述式指定了 WITH DATA DICTIONARY 子句，則綱目將含有資料字典。**資料字典**是一組含有物件定義的表格。如果 SQL 已建立字典，系統將自動加以維護。您可以透過交談式資料定義公用程式 (IDDU) 來使用資料字典，此公用程式爲 OS/400 程式的一部份。有關 IDDU 的詳細資訊，請參閱 IDDU Use  一書。

日誌與異動日誌接收器

日誌與**異動日誌接收器**是用來記錄資料庫中各類表格與概略表的變更內容。之後便可用來處理 SQL COMMIT、ROLLBACK、SAVEPOINT 及 RELEASE SAVEPOINT 陳述式。此外，它們還可作爲審核追蹤，或用於向前或向後回復。有關日誌登載的相關資訊，請參閱日誌登載主題。有關確定控制的詳細資訊，則請參閱確定控制主題。

編目

SQL **編目**是由一組表格與概略表所組成，他們可用來說明表格、概略表、索引、資料包、程序、函數、檔案、觸發及限制。此資訊內含於檔案庫 QSYS 與 QSYS2 的一組交互參照表中。在每一 SQL 綱目中，都有一組建置在編目表格上的概略表，其中含有關於綱目中各表格、概略表、索引、資料包、檔案及限制的資訊。

建立綱目時，編目即會自動建立。您不可捨棄或明確地變更編目。

有關 SQL 編目的詳細資訊，請參閱 SQL Reference 一書中的 Catalogs 主題。

表格、列和直欄

表格是一種二維的資料排列方式，由**列**與**直欄**組成。其中，列是指水平部份，內含一或多個直欄。而直欄則是垂直的部份，內含一或多個相同資料類型的資料列。直欄的所有資料必須是同一類型。SQL 中的表格是一種含鍵值或無鍵值的實體檔案。有關各種資料類型的說明，請參閱 SQL Reference 一書中的 Data Types 主題。

表格中的資料可分散在不同伺服器上。有關分散式表格的詳細資訊，請參閱 DB2 Multisystem 一書。

別名

別名是表格或概略表的替代名稱。若現有的表格或概略表可供參照，您即可用別名來加以參照。此外，別名還可用來結合表格成員。有關別名的詳細資訊，請參閱 SQL Reference 一書中的 Alias 主題。

概略表

概略表對應用程式而言就如同表格；不過，其中並不含任何資料。它是建立於一或多個表格。概略表可含有給定表格的所有直欄或其部份子集，而且可含有給定表格的所有列或其部份子集。取自表格的直欄，其在概略表中的排列方式可和表格者不同。 SQL 中的概略表是一種特殊形式的無鍵值邏輯檔案。

有關概略表的詳細資訊，請參閱 iSeries 資訊中心其 *SQL Reference* 一書中的 Views 主題。

索引

SQL **索引**是表格直欄中的資料子集，以升序或降序方式作邏輯排列。每一個索引皆含有個別的排列方式。這些排列方式可用來進行排序 (ORDER BY 子句)、分組 (GROUP BY 子句) 及結合。 SQL 索引是一種含鍵值的邏輯檔案。

索引可供系統用於加快資料的擷取。建立索引屬於選用作業。您可以建立任何數量的索引，亦可以隨時建立或捨棄索引。系統會自動維護索引。不過，也由於索引是由系統自動加以維護，因此大量的索引亦會影響用到該表格之應用程式的效能。

有關編寫有效率之索引的詳細資訊，請參閱 iSeries 資訊中心其 *Database Performance and Query Optimization* 一書中的 Using indexes to speed access to large tables 主題。

限制

限制是指由資料庫管理程式所強制採行的規則。 DB2 UDB for iSeries 支援下列限制：

- 唯一限制

  唯一限制是指索引鍵值只在具有唯一性時方始有效的規則。唯一限制可用 CREATE TABLE 與 ALTER TABLE 陳述式來建立。雖然 CREATE INDEX 亦可建立確具唯一性的唯一索引，但此種索引並非限制。

  唯一限制是在執行 INSERT 與 UPDATE 陳述式時所強制施行。 PRIMARY KEY 限制屬於一種 UNIQUE 限制。差別在於 PRIMARY KEY 不得含有任何可爲空值的直欄。

- 參照限制

  參照限制是指外來索引鍵只在下述情況方始有效的規則：

  – 顯現爲上層鍵的值，或

  – 外來索引鍵的部份元件爲空值。

  參照限制是在執行 INSERT、UPDATE 及 DELETE 等陳述式時所強制施行。

- 核對限制

  核對限制是指用於限制直欄或直欄群組中之容許值的規則。核對限制可用 CREATE TABLE 與 ALTER TABLE 陳述式來新增。核對限制是在執行 INSERT 與 UPDATE 陳述式時所強制施行。要滿足此限制，表格中所插入或更新的每一資料列須使指定條件爲 TRUE 或不明 (因空值所致)。

有關各類限制的詳細資訊，請參閱第 10 章, 『資料完整性』。

觸發

觸發是一組行動，每當指定的基本表格發生指定事件時，系統即會自動加以執行。所謂的事件可以是插入、更新、刪除或讀取作業。觸發可在事件之前或之後執行。 DB2 UDB for iSeries 支援 SQL 插入、更新及刪除等觸發，以及外部觸發。有關觸發的詳細資訊，請參閱本書的第 10 章, 『資料完整性』，或參閱 *Database Programming* 一書中的 Triggering automatic events in your database 主題。

儲存程序

儲存程序是一種程式，可用 SQL CALL 陳述式加以呼叫。DB2 UDB for iSeries 支援外部儲存程序以及 SQL 程序。外部儲存程序可以是任何系統程式或 REXX 程序。但不得爲 System/36 程式或程序，或服務程式。SQL 程序純粹是以 SQL 來定義，其中可含有 SQL 陳述式 (包括 SQL 控制陳述式)。有關儲存程序的詳細資訊，請參閱本書的第 11 章, 『儲存程序』 主題。

使用者定義的函數

使用者定義的函數是一種程式，其呼叫方式就如同任何內建函數。DB2 UDB for iSeries 可支援外部函數、SQL 函數及來源函數。外部函數可以是任何系統 ILE 程式或服務程式。SQL 函數純粹是以 SQL 來定義，其中可含有 SQL 陳述式 (包括 SQL 控制陳述式)。來源函數則是透過任何內建或現有之使用者定義的函數所建置。您可以建立純量函數或表格函數，來作爲 SQL 函數或外部函數。有關使用者定義的函數的詳細資訊，請參閱第 213 頁的第 13 章, 『撰寫使用者定義的函數 (UDF)』。

使用者定義的類型

使用者定義的類型是可由使用者在資料庫管理系統所提供的資料類型外，單獨定義的一種特殊資料類型。特殊資料類型會以一對一形式，對映至現有的資料庫類型。有關使用者定義的類型之詳細資訊，請參閱第 199 頁的『使用者定義的特殊類型 (UDT)』。

SQL 資料包

SQL 資料包是一種物件，內含當應用程式中之 SQL 陳述式連結至遠端關聯式資料庫管理系統 (DBMS) 時，所產生的控制結構。DBMS 在執行應用程式時，將會用此控制結構來處理所遇到的 SQL 陳述式。

在「建立 SQL (CRTSQLxxx)」指令中指定了關聯式資料庫名稱 (RDB 參數)，而且建立了程式物件時， SQL 資料包即會建立。您也可用 CRTSQLPKG 指令來建立資料包。有關資料包與分散式關連資料庫函數的詳細資訊，請參閱第 21 章, 『分散式關聯資料庫功能』。

SQL 資料包還可用 QSQPRCED API 來建立。本書所提及的 SQL 資料包純指「分散式程式」的 SQL 資料包。 QSQPRCED 即使用 SQL 資料包來提供「延伸動態 SQL」支援。有關 QSQPRCED 的詳細資訊，請參閱 iSeries 資訊中心中 OS/400 API 小節的 QSQPRCED 主題。

註: 此指令中的 xxx 代表主語言指示器： CI 爲 ILE C 語言，CPPI 爲 ILE C++ for iSeries 語言，CBL 爲 COBOL for iSeries 語言，CBLI 爲 ILE COBOL 語言，PLI 爲 iSeries PL/I 語言，RPG 爲 RPG for iSeries 語言，而 RPGI 爲 ILE RPG 語言。

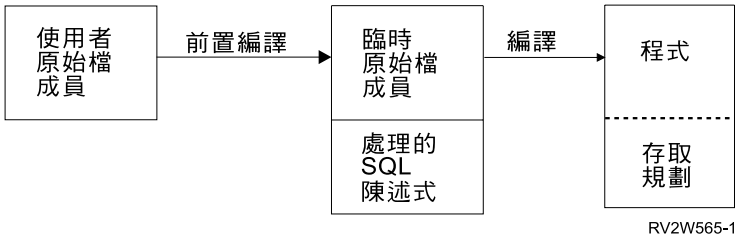
應用程式物件

建立 DB2 UDB for iSeries 應用程式的處理程序可能會導致數個物件的產生。本節將簡述建立 DB2 UDB for iSeries 應用程式的過程。DB2 UDB for iSeries 同時支援了非 ILE 與 ILE 前置編譯器。應用程式可以是分散式的或非分散式的。有關建立 DB2 UDB for iSeries 應用程式的詳細資訊，請參閱 *SQL Programming with Host Languages* 中的 Preparing and Running a Program with SQL Statements 主題。

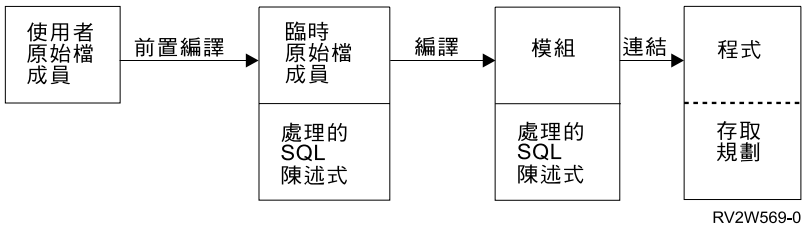
使用 DB2 UDB for iSeries 時，您可能會需要管理下列物件：

- 原始來源
- ILE 程式的模組物件 (選用)
- 程式或服務程式
- 分散式程式的 SQL 資料包

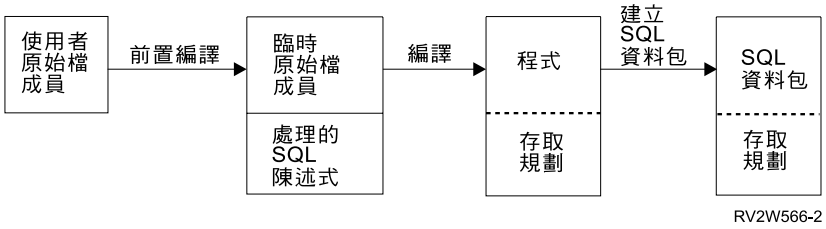
若是使用非分散式的非 ILE DB2 UDB for iSeries 程式，您就只需管理原始來源與結果程式。以下所示爲針對非分散式非 ILE DB2 UDB for iSeries 程式進行前置編譯與編譯處理時，所涉及的物件與步驟：



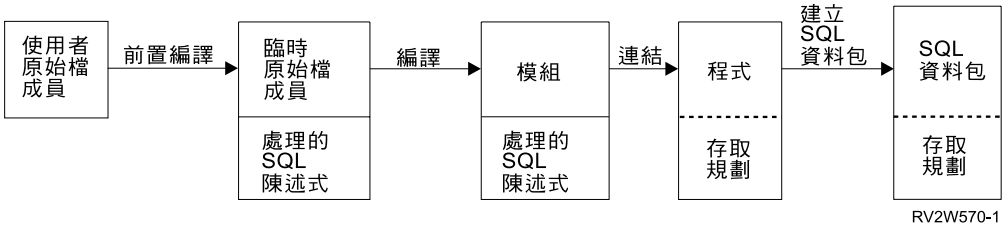
若是使用非分散式的 ILE DB2 UDB for iSeries 程式，您將須管理原始來源、模組及結果程式或服務程式。以下所示爲已在前置編譯指令中指定 OBJTYPE(*PGM) 時，針對非分散式 ILE DB2 UDB for iSeries 程式進行前置編譯與編譯處理期間，所涉及的物件與步驟：



若是使用分散式非 ILE DB2 UDB for iSeries 程式，您將須管理原始來源、結果程式及結果資料包。以下所示爲針對分散式非 ILE DB2 UDB for iSeries 程式進行前置編譯與編譯處理時，所涉及的物件與步驟：



若是使用分散式 ILE DB2 UDB for iSeries 程式，您將須管理原始來源、模組物件、結果程式或服務程式，以及結果資料包。您可爲分散式 ILE 程式或服務程式中的各個分散式模組建立 SQL 資料包。以下所示爲針對分散式 ILE DB2 UDB for iSeries 程式進行前置編譯與編譯處理時，所涉及的物件與步驟：



註: 與 DB2 UDB for iSeries 分散式程式物件相關的存取規劃須等到程式在本端執行後才會建立。

相關詳細資訊，請參閱下列各節：

- 『使用者原始檔成員』
- 『輸出原始檔成員』
- 第 11 頁的『程式』
- 第 11 頁的『模組』
- 第 11 頁的『服務程式』

使用者原始檔成員

原始檔成員含有程式設計師的應用程式語言與 SQL 陳述式。您可以透過原始檔登錄公用程式 (SEU) 來建立及維護原始檔成員，該公用程式爲 IBM WebSphere Development Studio for iSeries 授權程式的一部份。

輸出原始檔成員

SQL 前置編譯會建立一輸出原始檔成員。根據預設，前置編譯處理會在 QTEMP 中建立暫時原始檔 QSQLTxxxxx，或者由您在前置編譯指令中，將輸出原始檔指定爲永久檔。如果前置編譯處理使用了 QTEMP 檔案庫，系統會在工作完成時自動刪除該檔案。其間，會有一與程式同名的成員新增至輸出原始檔中。此成員含有下列項目：

- 對 SQL 執行時間支援的呼叫，其會置換內含的 SQL 陳述式
- 經過剖析與語法檢查的 SQL 陳述式

根據預設，前置編譯器會呼叫主語言編譯器。有關前置編譯器的詳細資訊，請參閱 *SQL Programming with Host Languages* 中的 Preparing and Running a Program with SQL Statements 主題。

程式

程式是一種可供您執行的物件，它是非 ILE 編譯處理的結果，或在 ILE 編譯過程中進行連結處理之結果。

存取規劃是一組內部結構與資訊，用來向 SQL 說明如何最有效地執行內含的 SQL 陳述式。它只有在程式已順利建立後才會建立。如果 SQL 陳述式有下列情形，則在針對陳述式建立程式期間，存取規劃將不會建立：

- 參照了找不到的表格或概略表
- 參照了您並無授權的表格或概略表

此類陳述式的存取規劃是在程式執行時才建立。若程式執行時仍找不到表格或概略表，或者您仍然沒有授權，系統即會傳回負值的 SQLCODE。存取規劃會儲存至非分散式 SQL 程式的程式物件中與分散式 SQL 程式的 SQL 資料包中並進行維護。

SQL 資料包

SQL 資料包中含有分散式 SQL 程式的存取規劃。

SQL 資料包是一種物件，其建立時機爲：

- 已在 CRTSQLxxx 指令中用 RDB 參數順利建立分散式 SQL 程式時。
- 執行「建立 SQL 資料包 (CRTSQLPKG)」指令時。

當分散式 SQL 程式建立後，SQL 資料包的名稱以及內部一致性記號將會存入該程式中。這些項目可在執行時間階段用來尋找 SQL 資料包，並驗證 SQL 資料包是否正確。由於 SQL 資料包的名稱對執行分散式 SQL 程式至關重要，因此 SQL 資料包不得：

- 移動
- 更名
- 重複
- 復置到不同的檔案庫

模組

模組是一種「整合語言環境 (ILE)」物件，可藉 CRTxxxMOD 指令 (或任何 CRTBNDxxx 指令，其中的 xxx 爲 C、CBL、CPP 或 RPG) 來編譯原始程式碼而加以建立。唯有用「建立程式 (CRTPGM)」指令將模組連結至程式中後，您才可加以執行。通常會將數個模組連結在一起，但您也可將模組連結至其本身。模組含有關於 SQL 陳述式的資訊；不過，SQL 存取規劃須等到模組已連結至程式或服務程式中後才會建立。有關「建立程式 (CRTPGM)」指令的詳細資訊，請參閱「指令語言」主題中的「建立程式 (CRTPGM)」。

服務程式

服務程式是一種「整合語言環境 (ILE)」物件，它可提供相關方法，用來將外部所支援之可呼叫常式 (函數或程序) 包裝成個別的物件。連結程式與其他服務程式可藉由解決其匯入對服務程式所提供的匯出，來存取這些常式。當呼叫程式建立時，對這些服務的連線也會完成。如此可增進此類常式的呼叫效能，而不需將程式碼併入呼叫程式中。

第 2 章 SQL 入門

本章說明如何使用「交談式 SQL」中的 SQL 陳述式來建立及使用綱目、表格與概略表。

SQL Reference 一書中詳細說明了本章中所使用的每一個 SQL 陳述式語法。如需如何在更複雜的狀況中使用 SQL 陳述式及子句的說明，請參閱第 45 頁的第 4 章, 『資料定義語言 (DDL)』、第 5 章, 『使用 SELECT 陳述式擷取資料』及第 6 章, 『SQL 插入、更新及刪除』。

在本章中，範例會使用交談式 SQL 介面來顯示 SQL 陳述式的執行。每一個 SQL 介面均提供使用 SQL 陳述式來定義表格、概略表及其它物件的方法、更新物件的方法及從物件中讀取資料的方法。

請參閱下列主題以取得明細：

- 『啓動交談式 SQL』
- 第 14 頁的『建立綱目』
- 第 14 頁的『建立及使用表格』
- 第 16 頁的『使用 LABEL ON 陳述式』
- 第 17 頁的『在表格中插入資訊』
- 第 20 頁的『從單一表格中取得資訊』
- 第 22 頁的『從一個以上的表格中取得資訊』
- 第 24 頁的『變更表格中的資訊』
- 第 27 頁的『從表格中刪除資訊』
- 第 27 頁的『建立及使用概略表』

註: 如需專屬於程式碼範例的相關資訊，請參閱第 x 頁的『程式碼不保事項聲明』資訊。

啓動交談式 SQL

若要在下列範例中開始使用交談式 SQL，請鍵入：

```
STRSQL NAMING(*SQL)
```

然後按下 Enter 鍵。當「輸入 SQL 陳述式」顯示畫面出現時，即可準備開始鍵入「SQL 陳述式」。如需交談式 SQL 及 STRSQL 指令的相關資訊，請參閱第 17 章, 『使用交談式 SQL』。

如果您要重覆使用現有的交談式 SQL 階段作業，請確定您已將命名模式設定爲 *SQL 命名*。您可以在 F13 (服務) 畫面中，使用選項 1 (變更階段作業屬性) 來設定此模式。

建立綱目

綱目是存放表格、概略表、索引及資料包的基本物件。如需建立綱目的相關資訊，請參閱 SQL CREATE SCHEMA 陳述式。

註: 術語*集合*可以當成*綱目*的同義字使用。

如需使用交談式 SQL 建立綱目的範例，請參閱『範例：建立綱目 (SAMPLECOLL)』。

範例：建立綱目 (SAMPLECOLL)

您可以在「輸入 SQL 陳述式」顯示畫面中鍵入下列 SQL 陳述式，然後按下 Enter 鍵，即可建立範例綱目，並命名爲 SAMPLECOLL：

```
                              輸入 SQL 陳述式

請鍵入 SQL 陳述式，然後按下 Enter 鍵。
     目前連線至關聯式資料庫 SYSTEM1
===> CREATE SCHEMA SAMPLECOLL_____________________________________________
     ____________________________________________________________________
     ____________________________________________________________________
     ____________________________________________________________________
                                                                    底端
F3=跳出    F4=提示      F6=插入行        F9=擷取        F10=複製行
F12=取消                F13=服務         F24=其餘鍵
```

註: 執行此陳述式即會建立數個物件，且需要幾秒鐘的時間。
在您順利建立綱目之後，即可以在其中建立表格、概略表及索引。也可以在檔案庫 (而非綱目) 上建立表格、概略表與索引。

建立及使用表格

您可以使用 SQL CREATE TABLE 陳述式建立表格。 CREATE TABLE 陳述式可以讓您建立表格、定義表格中直欄的實體屬性，以及定義限制以限定表格中容許的值。

如需使用交談式 SQL 建立表格的範例，請參閱
『範例：建立表格 (INVENTORY_LIST)』。

在建立表格時，您必須瞭解 NULL 值及預設值的概念。 NULL 值表示缺少列的直欄值。但和 0 的值或全部空白不同。它表示「不明」。且不等於任何值，甚至不等於其它 NULL 值。如果直欄不容許 NULL 值，則必須對直欄指定一個值，不論是預設值或使用者提供的值。

在表格中新增列，且沒有指定任何直欄值時，則會對該直欄指定預設值。如果沒有爲直欄定義特定的預設值，則將使用系統預設值。如需 INSERT 所使用的預設值之相關資訊，請參閱第 87 頁的『使用 INSERT 陳述式插入列』

範例：建立表格 (INVENTORY_LIST)

我們將建立表格，以維護目前業務庫存的相關資訊。表格將包含庫存中所保留的項目相關資訊，包括成本、目前現有的數量、前次訂購日期及前次訂購的數量。項目編號將爲必要的值。它不能是 NULL。項目名稱、現有數量及訂購數量將具有使用者提供的預設值。前次訂購日期及訂購的數量可容許 NULL 值。

在「輸入 SQL 陳述式」顯示畫面中，請鍵入 CREATE TABLE 並按下 F4 (提示)。即會顯示下列顯示畫面 (其中的輸入區尚未填寫)：

```
                        指定 CREATE TABLE 陳述式

請鍵入資訊，然後按下 Enter 鍵。

表格  . . . . . .  INVENTORY_LIST_____     名稱
  集合    . . . . .   SAMPLECOLL__         名稱，F4 以列示

NULL:  1=NULL, 2=NOT NULL, 3=NOT NULL WITH DEFAULT

直欄                FOR 直欄      類型            長度  小數位數 NULL
ITEM_NUMBER_______  ____________  CHAR____________ 6____ __     2
ITEM_NAME_________  ____________  VARCHAR_________ 20___ __     3
UNIT_COST_________  ____________  DECIMAL_________ 8____ 2_     3
QUANTITY_ON_HAND__  ____________  SMALLINT________ _____ __     1
LAST_ORDER_DATE___  ____________  DATE____________ _____ __     1
ORDER_QUANTITY____  ____________  SMALLINT________ _____ __     1
__________________  ____________  ________________ _____ __     3
                                                                 底端
  表格 CONSTRAINT  . . . . . .  N     Y=是，N=否
  分散式表格       . . . . . .  N     Y=是，N=否

F3=跳出   F4=提示          F5=重整     F6=插入行       F10=複製行
F11=顯示其餘屬性           F12=取消    F14=刪除行      F24=其餘鍵
```

鍵入您要建立的表格名稱及表格的綱目名稱，請在*表格*及*集合*提示中，在 SAMPLECOLL 中建立表格 INVENTORY_LIST。顯示畫面下半部的清單登錄代表您要爲表格定義的每一個直欄。對於每一個直欄，請鍵入直欄名稱、直欄的資料類型、其長度與小數位數，以及 NULL 屬性。

請按下 F11 以查看可以指定的其餘直欄屬性。您可以在此處指定預設值。

```
                        指定 CREATE TABLE 陳述式

請鍵入資訊，然後按下 Enter 鍵。

表格  . . . . . .  INVENTORY_LIST_____     名稱
  集合    . . . . .   SAMPLECOLL__         名稱，F4 以列示

資料:  1=BIT, 2=SBCS, 3=MIXED, 4=CCSID

直欄                資料 配置  CCSID  CONSTRAINT 預設
ITEM NUMBER_______   _   _____ _____    N       _________________
ITEM NAME_________   _   _____ _____    N      '***UNKNOWN***'___
UNIT_COST_________   _   _____ _____    N       _________________
QUANTITY_ON_HAND__   _   _____ _____    N      NULL______________
LAST_ORDER_DATE___   _   _____ _____    N       _________________
ORDER_QUANTITY____   _   _____ _____    N      20________________
__________________   _   _____ _____    -       _________________
                                                                 底端
  表格 CONSTRAINT  . . . . . .  N     Y=是，N=否
  分散式表格       . . . . . .  N     Y=是，N=否

F3=跳出   F4=提示          F5=重整     F6=插入行       F10=複製行
F11=顯示其餘屬性           F12=取消    F14=刪除行      F24=其餘鍵
```

註: 輸入直欄定義的另一種方式是將游標放在清單中的其中一個直欄登錄並按下 F4 (提示)。此即會出現一個顯示畫面，顯示所有屬性以定義單一直欄。

輸入所有值後，請按下 Enter 鍵以建立表格。將會重新顯示「輸入 SQL 陳述式」，並出現一則訊息表示表格已建立。

您可以在「輸入 SQL 陳述式」顯示畫面中直接鍵入此 CREATE TABLE 陳述式，如下所示：

```
CREATE TABLE SAMPLECOLL.INVENTORY_LIST
     (ITEM_NUMBER CHAR(6) NOT NULL,
      ITEM_NAME VARCHAR(20) NOT NULL WITH DEFAULT '***UNKNOWN***',
      UNIT_COST DECIMAL(8,2) NOT NULL WITH DEFAULT,
      QUANTITY_ON_HAND SMALLINT DEFAULT NULL,
      LAST_ORDER_DATE DATE,
      ORDER_QUANTITY SMALLINT DEFAULT 20)
```

建立供應商表格 (SUPPLIERS)

稍後在我們的範例中，將需要使用第二個表格。此表格將包含庫存項目供應商的相關資訊、他們提供的項目，以及從該供應商取得的項目成本。若要建立此表格，請在「輸入 SQL 陳述式」顯示畫面中直接鍵入，或按下 F4 (提示) 以使用交談式 SQL 顯示畫面來建立定義。

```
CREATE TABLE SAMPLECOLL.SUPPLIERS
     (SUPPLIER_NUMBER CHAR(4) NOT NULL,
     ITEM_NUMBER CHAR(6) NOT NULL,
     SUPPLIER_COST DECIMAL(8,2))
```

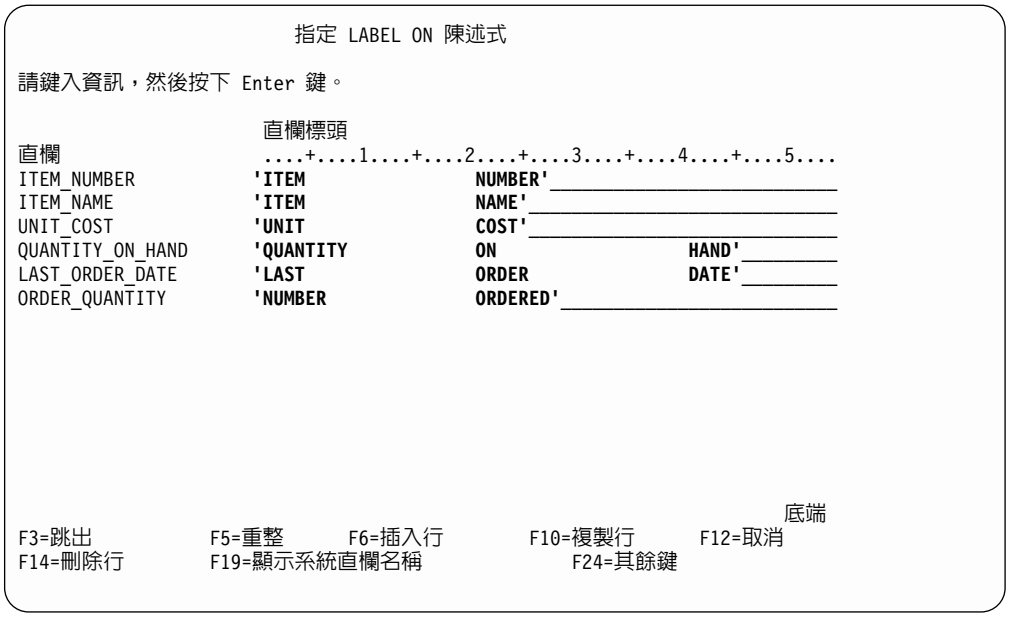
使用 LABEL ON 陳述式

在顯示以交談式 SQL 撰寫的 SELECT 陳述式輸出時，通常，會使用直欄名稱作爲直欄標頭。使用 LABEL ON 陳述式，您就可以建立直欄名稱的敘述性標籤。因爲我們將以交談式 SQL 來執行範例，所以使用 LABEL ON 陳述式來變更直欄標頭。即使直欄名稱是敘述性的，如果直欄標頭在單一行上顯示名稱的各部份，仍是容易讀取的。它也將使我們在單一顯示畫面上看到更多的資料直欄。

若要變更直欄的標籤，請在「輸入 SQL 陳述式」顯示畫面中鍵入 LABEL ON COLUMN，然後按下 F4 (提示)。即會出現下列顯示畫面：

```
                          指定 LABEL ON 陳述式

請鍵入選項，然後按 Enter 鍵。

  標籤      . . . .   2                          1=表格或概略表
                                                 2=直欄
                                                 3=套裝軟體
                                                 4=別名

  表格或概略表         INVENTORY_LIST______      名稱，F4 以列示
    集合         . .    SAMPLECOLL__            名稱，F4 以列示

  選項    . . . . .   1                          1=直欄標頭
                                                 2=本文









F3=跳出    F4=提示      F5=重整       F12=取消      F20=顯示完整名稱
F21=顯示陳述式
```

鍵入內含您要新增標籤之直欄的表格與綱目名稱，然後按下 Enter 鍵。即會顯示下列顯示畫面，提示您表格中的每一個直欄。



請鍵入每一個直欄的直欄標頭。您可以在 20 個字元的區段中定義直欄標頭。在顯示 SELECT 陳述式的輸出時，每一個區段均會顯示在不同行上。您可以使用直欄標頭輸入區頂端的尺規，以輕鬆、正確地保持標頭間格。鍵入標頭後，請按下 Enter 鍵。

下列訊息表示 LABEL ON 陳述式已順利完成。

```
SAMPLECOLL 中 INVEN00001 的 LABEL ON 已完成。
```

訊息中的表格名稱是此表格的系統表格名稱，而不是實際上在陳述式中指定的名稱。DB2 UDB for iSeries 可以維護名稱長度超出 10 個字元的兩個表格名稱。如需系統表格名稱的相關資訊，請參閱 SQL Reference 一書中的 CREATE TABLE 陳述式。

您亦可在「輸入 SQL 陳述式」顯示畫面中直接鍵入 LABEL ON 陳述式，如下所示：

```
LABEL ON SAMPLECOLL.INVENTORY_LIST
(ITEM_NUMBER      IS 'ITEM              NUMBER',
ITEM_NAME         IS 'ITEM              NAME',
UNIT_COST         IS 'UNIT              COST',
QUANTITY_ON_HAND  IS 'QUANTITY          ON                HAND',
LAST_ORDER_DATE   IS 'LAST              ORDER             DATE',
ORDER_QUANTITY    IS 'NUMBER            ORDERED')
```

在表格中插入資訊

在您建立表格後，您可以使用 SQL INSERT 陳述式，以在表格中插入或新增資訊 (資料)。

如需使用交談式 SQL 在表格中插入資料的範例，請參閱『範例：在表格 (INVENTORY_LIST) 中插入資訊』。

範例：在表格 (INVENTORY_LIST) 中插入資訊

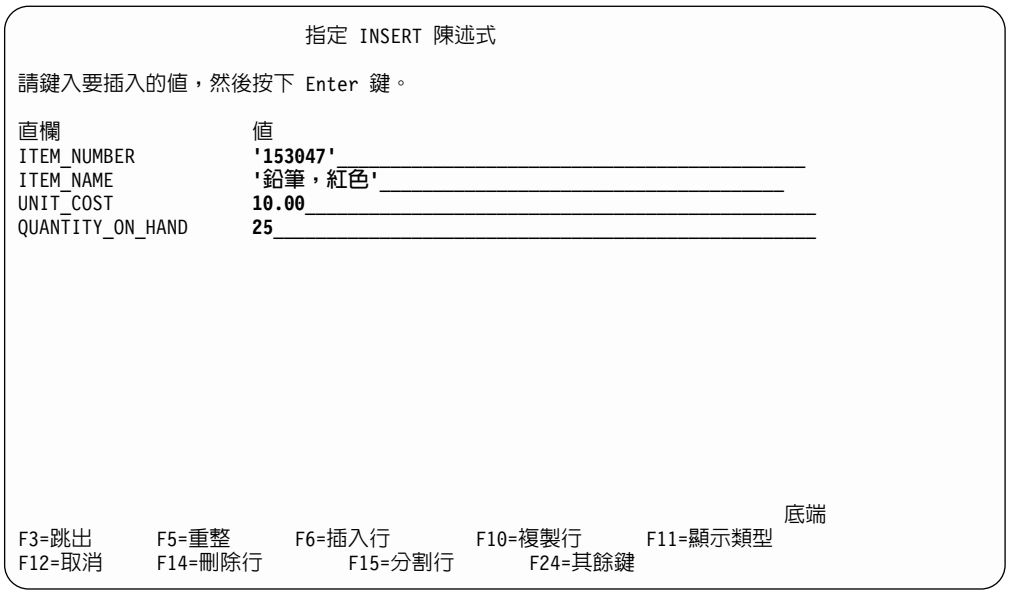
若要使用交談式 SQL，請在「輸入 SQL 陳述式」顯示畫面中鍵入 INSERT 並按下 F4 (提示)。將會顯示「指定 INSERT 陳述式」顯示畫面。

```
                        指定 INSERT 陳述式

請鍵入選項，然後按 Enter 鍵。

  INTO 表格  . . . . . .    INVENTORY_LIST_____      名稱，F4 以列示
    集合       . . . . . .      SAMPLECOLL__          名稱，F4 以列示

  選取插入的直欄
    INTO  . . . . . .  Y                             Y=是，N=否
  插入方法          . . . .  1                       1=輸入 VALUES
                                                     2=子選擇

請鍵入選項，然後按 Enter 鍵。

  WITH 隔離層次         . .  1              1=現行層次，2=NC (NONE)
                                            3=UR (CHG)，4=CS，5=RS (ALL)
                                            6=RR



F3=跳出    F4=提示      F5=重整       F12=取消      F20=顯示完整名稱
F21=顯示陳述式
```

在輸入欄位中鍵入表格名稱與綱目名稱，如下所示。將*選取直欄以插入 INTO* 提示變更爲「是」。按下 Enter 鍵，即會出現一個顯示畫面，您可以在其中選取要插入值的直欄。

```
                        指定 INSERT 陳述式

請鍵入序號 (1-999) 以進行選擇，然後按下 Enter 鍵。

順序 直欄                  類型              長度     小數位數
1__  ITEM_NUMBER           CHARACTER            6
2__  ITEM_NAME             VARCHAR             20
3__  UNIT_COST             DECIMAL              8    2
4__  QUANTITY_ON_HAND      SMALLINT             4
___  LAST_ORDER_DATE       DATE
___  ORDER_QUANTITY        SMALLINT             4

                                                                    底端
F3=跳出    F5=重整         F12=取消      F19=顯示系統直欄名稱
F20=顯示完整名稱           F21=顯示陳述式
```

在本例中，我們只想插入其中四個直欄。我們將在其它直欄中插入預設值。此顯示畫面上的序號表示直欄與值在 INSERT 陳述式中列出的次序。按下 Enter 鍵，即會出現顯示畫面，您可以在其中鍵入選取的直欄的值。



註: 若要查看插入清單中每一個直欄的資料類型與長度，請按下 F11 (顯示類型)。此將顯示插入值顯示畫面的不同概略表，並提供直欄定義的相關資訊。

鍵入所有要插入的直欄值，然後按下 Enter 鍵。即會將內含這些值的列新增到表格。未指定的直欄值將會以預設值插入。若爲 LAST_ORDER_DATE，它將是 NULL 值，因爲未提供任何預設值，且該直欄容許 NULL 值。若爲 ORDER_QUANTITY，其值爲 20，該值是 CREATE TABLE 陳述式中指定的預設值。

您可以在「輸入 SQL 陳述式」顯示畫面中鍵入 INSERT 陳述式，如下所示：

```
INSERT INTO SAMPLECOLL.INVENTORY_LIST
                  (ITEM_NUMBER,
                   ITEM_NAME,
                   UNIT_COST,
                   QUANTITY_ON_HAND)
           VALUES('153047',
                  'Pencils, red',
                  10.00,
                  25)
```

若要新增下一列到表格，請在「輸入 SQL 陳述式」顯示畫面中按下 F9 (擷取)。這會將之前的 INSERT 陳述式複製到鍵入區。您可以鍵入前一 INSERT 陳述式的值，或按下 F4 (提示)，以使用「交談式 SQL」顯示畫面來輸入資料。

繼續使用 INSERT 陳述式以新增下列橫列到表格。不應插入下面圖表中未顯示的值，所以將會使用預設值。在 INSERT 陳述式直欄清單中，只指定您要插入值的直欄名稱。例如，若要插入第三列，您只要指定 ITEM_NUMBER 及 UNIT_COST 作爲直欄名稱，且在 VALUES 清單中只指定這兩個直欄值。

ITEM_NUMBER	ITEM_NAME	UNIT_COST	QUANTITY_ON_HAND
153047	鉛筆，紅色	10.00	25
229740	畫線的筆記本	1.50	120
544931		5.00	
303476	紙夾	2.00	100

ITEM_NUMBER	ITEM_NAME	UNIT_COST	QUANTITY_ON_HAND
559343	信封，標準尺寸	3.00	500
291124	信封，標準		
775298	椅子，祕書	225.00	6
073956	筆，黑色	20.00	25

新增下列橫列到 SAMPLECOLL.SUPPLIERS 表格。

SUPPLIER_NUMBER	ITEM_NUMBER	SUPPLIER_COST
1234	153047	10.00
1234	229740	1.00
1234	303476	3.00
9988	153047	8.00
9988	559343	3.00
2424	153047	9.00
2424	303476	2.50
5546	775298	225.00
3366	303476	1.50
3366	073956	17.00

現在，範例綱目中已含有兩個表格，且每一個表格中各有數列資料。

從單一表格中取得資訊

現在，所有資訊已插入表格，我們必須能重新查看它們。在 SQL 中，可以使用 SELECT 陳述式來完成此動作。 SELECT 陳述式是所有 SQL 陳述式中最複雜的。此陳述式是由三個主要子句組成：

1. SELECT 子句，指定含有想要的資料的那些直欄。
2. FROM 子句，指定表格，表格中含有直欄與想要的資料。
3. WHERE 子句，提供條件以判定要擷取哪些資料列。

除了三個主要子句外，第 59 頁的第 5 章, 『使用 SELECT 陳述式擷取資料』及 SQL Reference 一書中另說明數種其它的子句，這些子句會影響最終的傳回資料格式。

若要查看插入於 INVENTORY_LIST 表格中的值，請鍵入 SELECT 並按下 F4 (提示)。將會顯示下列的顯示畫面：

```
                          指定 SELECT 陳述式

請鍵入 SELECT 陳述式資訊。按下 F4 以列示。

  FROM 表格    . . . . . .   SAMPLECOLL.INVENTORY_LIST_____________________
  SELECT 直欄. . . . . . .   *_____________________________________________
  WHERE 條件 . . . . . . .   _____________________________________________
  GROUP BY 直欄. . . . . .   _____________________________________________
  HAVING 條件. . . . . . .   _____________________________________________
  ORDER BY 直欄. . . . . .   _____________________________________________
  FOR UPDATE OF 直欄 . . . .  _____________________________________________

                                                                    底端
請鍵入選項，然後按 Enter 鍵。

  結果表格中的 DISTINCT 列   . . . . . .  N    Y=是，N=否
  UNION 與另一個 SELECT      . . . . . .  N    Y=是，N=否
  指定額外選項               . . . . . .  N    Y=是，N=否



F3=跳出          F4=提示      F5=重整       F6=插入行         F9=指定子查詢
F10=複製行       F12=取消     F14=刪除行         F15=分割行        F24=其餘鍵
```

請在顯示畫面的 *FROM 表格*欄位中鍵入表格名稱。若要從表格中選取所有直欄，請在顯示畫面的 *SELECT 直欄*欄位中鍵入 *。按下 Enter 鍵，陳述式即會執行以選取表格中所有直欄的所有資料。將會顯示下列輸出：

```
                              顯示資料
                                          資料寬度  . . . . . :     71
定位於行. . . . . .              移位至直欄 . . . . . .
....+....1....+....2....+....3....+....4....+....5....+....6....+....7.
ITEM    ITEM                      UNIT   QUANTITY  LAST      NUMBER
NUMBER  NAME                      COST   ON        ORDER     ORDERED
                                         HAND      DATE
153047  鉛筆，紅色                10.00        25   -              20
229740  畫線的筆記本               1.50       120   -              20
544931  ***UNKNOWN***             5.00         -   -              20
303476  紙夾                       2.00       100   -              20
559343  信封，標準尺寸             3.00       500   -              20
291124  信封，標準                  .00         -   -              20
775298  椅子，祕書               225.00         6   -              20
073956  筆，黑色                  20.00        25   -              20
********  資料結尾  ********

F3=跳出      F12=取消        F19=左        F20=右         F21=分割
```

會顯示使用 LABEL ON 陳述式定義的直欄標頭。第三個登錄的 ITEM_NAME 具有 CREATE TABLE 陳述式中所指定的預設值。若是有插入任何值的列，則 QUANTITY_ON_HAND 直欄具有 NULL 值。因爲 *LAST_ORDER_DATE* 直欄不在任何 INSERT 陳述式中，且未定義該直欄的任何預設值，所以該直欄含有所有 NULL 值。同樣地，ORDER_QUANTITY 直欄含有所有列的預設值。

您可以在「輸入 SQL 陳述式」顯示畫面中輸入此陳述式，如下所示：

```
SELECT *
  FROM SAMPLECOLL.INVENTORY_LIST
```

若要限制 SELECT 陳述式傳回的直欄數，您必須指定要查看的直欄。若要限制傳回的輸出列數，則使用 WHERE 子句。若只要查看成本在 10 元以上的項目，且只傳直欄 ITEM_NUMBER、UNIT_COST 及 ITEM_NAME 的值，請鍵入 SELECT 並按下 F4 (提示)。將顯示「指定 SELECT 陳述式」顯示畫面。

```
                          指定 SELECT 陳述式

請鍵入 SELECT 陳述式資訊。按下 F4 以列示。

  FROM 表格    . . . . . .   SAMPLECOLL.INVENTORY_LIST_____________________
  SELECT 直欄  . . . . . .   ITEM_NUMBER, UNIT_COST, ITEM_NAME_____________
  WHERE 條件   . . . . . .   UNIT_COST > 10.00_____________________________
  GROUP BY 直欄  . . . . .   _____________________________________________
  HAVING 條件    . . . . .   _____________________________________________
  ORDER BY 直欄  . . . . .   _____________________________________________
  FOR UPDATE OF 直欄 . . .   _____________________________________________

                                                                    底端
請鍵入選項，然後按 Enter 鍵。

  結果表格中的 DISTINCT 列   . . . . . .  N    Y=是，N=否
  UNION 與另一個 SELECT      . . . . . .  N    Y=是，N=否
  指定額外選項               . . . . . .  N    Y=是，N=否



F3=跳出          F4=提示      F5=重整       F6=插入行         F9=指定子查詢
F10=複製行       F12=取消     F14=刪除行         F15=分割行        F24=其餘鍵
```

雖然在「指定 SELECT 陳述式」顯示畫面中，每一個提示在起始時只顯示一行，您可以使用 F6 (插入行)，以在顯示畫面上半部的任何輸入區中新增更多行。如果在 *SELECT 直欄*清單中輸入更多直欄，或需要使用更長、更複雜的 WHERE 條件時，則可以使用此方法。

請填寫顯示畫面，如上所示。按下 Enter 鍵時，即會執行 SELECT 陳述式。您就會看到下列輸出：

```
                              顯示資料
                                          資料寬度  . . . . . :     41
定位於行. . . . . .                 移位至直欄 . . . . . .
....+....1....+....2....+....3....+....4.
ITEM          UNIT   ITEM
NUMBER         COST   NAME
775298       225.00   椅子，祕書
073956        20.00   筆，黑色
********  資料結尾  ********

F3=跳出       F12=取消        F19=左        F20=右         F21=分割
```

唯一傳回的是其值與 WHERE 子句指定的條件相比較的那些列。尤其，唯一傳回的資料值是來自您在 SELECT 子句中明確指定的直欄。不是那些明確識別的直欄資料值，則不會傳回。

此陳述式已輸入於「輸入 SQL 陳述式」顯示畫面中，如下所示：

```
SELECT ITEM_NUMBER, UNIT_COST, ITEM_NAME
  FROM SAMPLECOLL.INVENTORY_LIST
  WHERE UNIT_COST > 10.00
```

從一個以上的表格中取得資訊

SQL 可以讓您從一個以上表格所含的直欄中取得資訊。此作業稱爲*結合*運算。 (如需結合運算的詳細說明，請參閱第 74 頁的『結合多個表格的資料』)。在 SQL 中，您可以將想要結合在一起的表格名稱放入 SELECT 陳述式的同一個 FROM 子句中，即可指定結合運算。

假設您想要查看一份清單，其中含有所有供應商，以及他們所提供的項目之項目編號與項目名稱。項目名稱不在 SUPPLIERS 表格中。而是在 INVENTORY_LIST 表格中。利用共用直欄 ITEM_NUMBER，我們可以看到這三個直欄，就好像它們是來自單一表格一樣。

只要在要結合的兩個以上表格中有相同的直欄名稱，就必須以表格名稱來限定直欄名稱，以指定實際上是參照哪一個直欄。在此 SELECT 陳述式中，已在兩個表格中定義直欄名稱 ITEM_NUMBER，所以必須以表格名稱限定該直欄名稱。如果直欄有不同的名稱，就不會發生混淆，所以就不需要限定。

若要執行此結合，您可以使用下列 SELECT 陳述式。您可以在「輸入 SQL 陳述式」顯示畫面中直接鍵入，或利用提示來輸入它。如果使用提示，則必須在 *FROM 表格*輸入行中鍵入兩個表格名稱。

```
SELECT SUPPLIER_NUMBER, SAMPLECOLL.INVENTORY_LIST.ITEM_NUMBER, ITEM_NAME
  FROM SAMPLECOLL.SUPPLIERS, SAMPLECOLL.INVENTORY_LIST
  WHERE SAMPLECOLL.SUPPLIERS.ITEM_NUMBER
                    = SAMPLECOLL.INVENTORY_LIST.ITEM_NUMBER
```

輸入同一陳述式的另一種方式是使用相關名稱。相關名稱提供表格名稱的另一種名稱，以在陳述式中使用。當表格名稱相同時，必須使用相關名稱。您可以在 FROM 清單中的每一個表格名稱後指定其相關名稱。之前的陳述式可以重寫爲：

```
SELECT SUPPLIER_NUMBER, Y.ITEM_NUMBER, ITEM_NAME
  FROM SAMPLECOLL.SUPPLIERS X, SAMPLECOLL.INVENTORY_LIST Y
  WHERE X.ITEM_NUMBER = Y.ITEM_NUMBER
```

在本例中，SAMPLECOLL.SUPPLIERS 的指定相關名稱是 X，且 SAMPLECOLL.INVENTORY_LIST 的指定相關名稱是 Y。然後，使用 X 及 Y 來定義 ITEM_NUMBER 直欄名稱。

如需直欄及相關名稱的相關資訊，請參閱 iSeries 資訊中心中 *SQL Reference* 主題的相關名稱。

執行此範例會傳回下列輸出：

```
                              顯示資料
                                          資料寬度  . . . . . :     45
定位於行. . . . . .                 移位至直欄 . . . . . .
....+....1....+....2....+....3....+....4....+
SUPPLIER_NUMBER  ITEM    ITEM
                 NUMBER  NAME
     1234        153047  鉛筆，紅色
     1234        229740  畫線的筆記本
     1234        303476  紙夾
     9988        153047  鉛筆，紅色
     9988        559343  信封，標準尺寸
     2424        153047  鉛筆，紅色
     2424        303476  紙夾
     5546        775298  椅子，祕書
     3366        303476  紙夾
     3366        073956  筆，黑色
********  資料結尾  ********

F3=跳出      F12=取消        F19=左        F20=右         F21=分割
```

註: 因爲沒有指定查詢的任何 ORDER BY 子句，查詢所傳回的列次序可能會不相同。

結果表格中的資料值，代表兩個表格 INVENTORY_LIST 及 SUPPLIERS 中所含資料值的組合。此結果表格含有 SUPPLIER 表格中的供應商編號，及 INVENTORY_LIST 表

格中的項目編號與項目名稱。未出現在 SUPPLIER 表格中的任何項目編號，都不會顯示在此結果表格中。除非已指定 SELECT 陳述式的 ORDER BY 子句，否則不保證結果的次序。因爲我們沒有變更 SUPPLIER 表格的任何直欄標頭，所以會使用 SUPPLIER_NUMBER 直欄名稱作爲直欄標頭。

下列爲使用 ORDER BY 以保證列次序的範例。陳述式會先以 SUPPLIER_NUMBER 直欄來排序結果表格。如果與 SUPPLIER_NUMBER 值相同的列，則依照其 ITEM_NUMBER 排序。

```
SELECT SUPPLIER_NUMBER, Y.ITEM_NUMBER, ITEM_NAME
  FROM SAMPLECOLL.SUPPLIERS X, SAMPLECOLL.INVENTORY_LIST Y
  WHERE X.ITEM_NUMBER = Y.ITEM_NUMBER
  ORDER BY SUPPLIER_NUMBER, Y.ITEM_NUMBER
```

執行之前的陳述式會產生下列輸出。

```
                                顯示資料
                                          資料寬度  . . . . . :     45
定位於行. . . . . .                 移位至直欄 . . . . . .
....+....1....+....2....+....3....+....4....+
SUPPLIER_NUMBER  ITEM    ITEM
                 NUMBER  NAME
     1234        153047  鉛筆，紅色
     1234        229740  畫線的筆記本
     1234        303476  紙夾
     2424        153047  鉛筆，紅色
     2424        303476  紙夾
     3366        073956  筆，黑色
     3366        303476  紙夾
     5546        775298  椅子，祕書
     9988        153047  鉛筆，紅色
     9988        559343  信封，標準尺寸
********  資料結尾  ********

F3=跳出      F12=取消       F19=左       F20=右       F21=分割
```

變更表格中的資訊

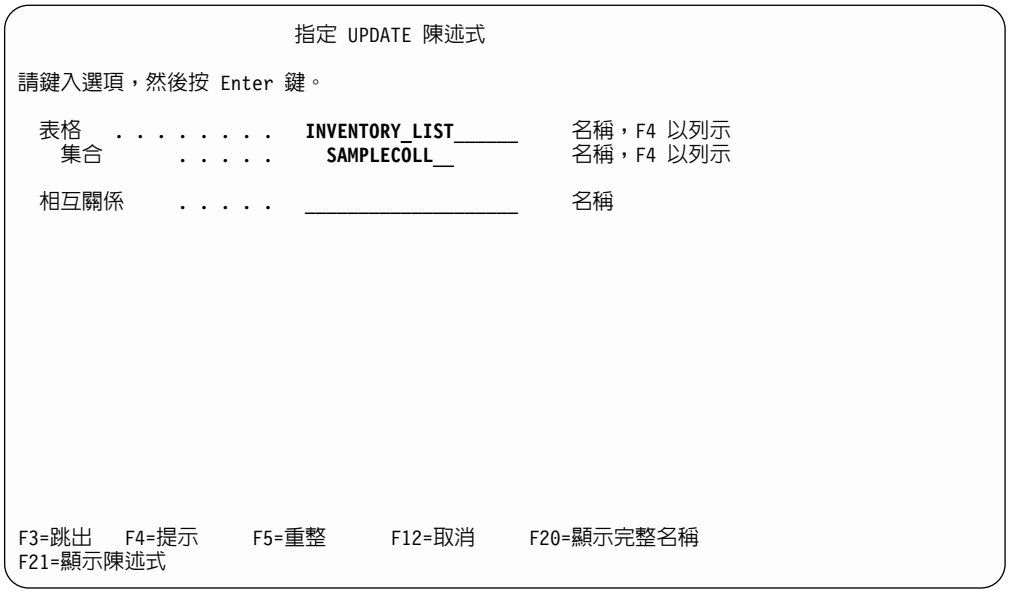
您可以使用 SQL UPDATE 陳述式，變更部份或所有表格直欄中的資料值。

如需使用交談式 SQL 變更表格資訊的範例，請參閱『範例：變更表格中的資訊』。

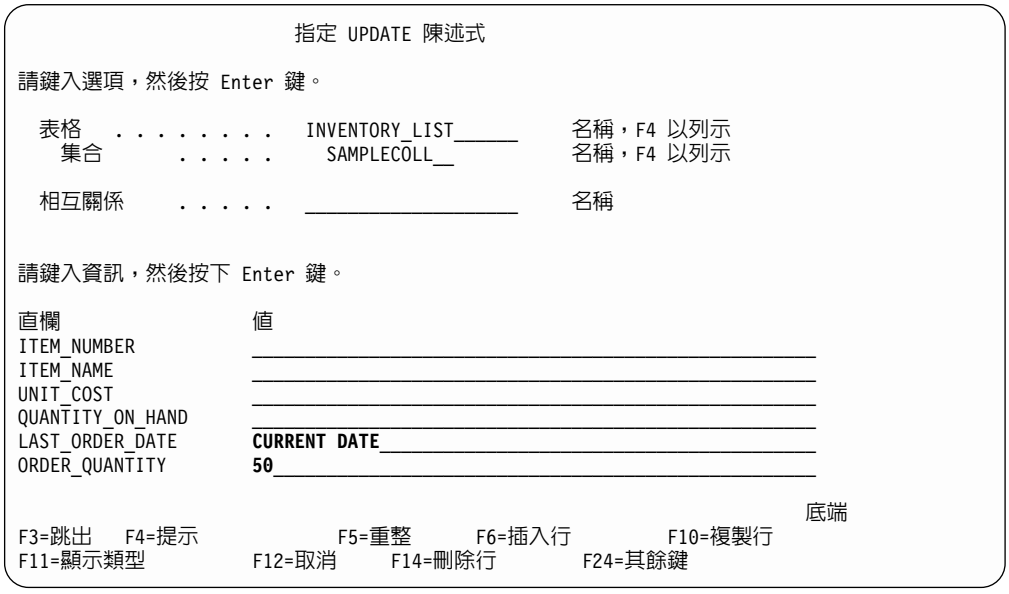
如果您想要限制單一陳述式執行期間變更的列數，請使用 WHERE 子句與 UPDATE 陳述式。如需相關資訊，請參閱第 91 頁的『使用 UPDATE 陳述式變更表格中的資料』。如果您沒有指定 WHERE 子句，則會變更所指定表格中的所有列。然而，如果您使用 WHERE 子句，則系統只會變更符合您指定條件的列。如需相關資訊，請參閱第 60 頁的『使用 WHERE 子句 指定搜尋條件』。

範例：變更表格中的資訊

假設我們想要使用交談式 SQL，並在今天訂購更多的紙夾。若要更新項目編號 303476 的 LAST_ORDER_DATE 及 ORDER_QUANTITY，請鍵入 UPDATE 並按下 F4 (提示)。此將顯示「指定 UPDATE 陳述式」顯示畫面。



鍵入表格名稱及綱目名稱後，請按下 Enter 鍵。此將重新出現該顯示畫面，並顯示表格中的直欄清單。



指定 CURRENT DATE 作爲值，會將所有選取列中的日期變更爲今天的日期。

鍵入要更新的表格值後，請按下 Enter 鍵，即會出現顯示畫面，您可以在其中指定 WHERE 條件。如果未指定 WHERE 條件，則會使用之前顯示畫面中的值來更新表格中的所有列。

```
                        指定 UPDATE 陳述式

請鍵入 WHERE 條件，然後按下 Enter 鍵。 按下 F4 以列示。
  ITEM_NUMBER = '303476'_____________________________________________
  ___________________________________________________________________



                                                                  底端
請鍵入選項，然後按 Enter 鍵。

  WITH 隔離層次        . . .   1              1=現行層次，2=NC (NONE)
                                              3=UR (CHG)，4=CS，5=RS (ALL)
                                              6=RR




F3=跳出          F4=提示      F5=重整       F6=插入行         F9=指定子查詢
F10=複製行       F12=取消     F14=刪除行        F15=分割行        F24=其餘鍵
```

鍵入條件之後，請按下 Enter 鍵以執行表格更新。將出現一則訊息表示函數已完成。

此陳述式已鍵入「輸入 SQL 陳述式」顯示畫面中，如下所示：

```
UPDATE SAMPLECOLL.INVENTORY_LIST
  SET LAST_ORDER_DATE = CURRENT DATE,
      ORDER_QUANTITY = 50
  WHERE ITEM_NUMBER = '303476'
```

執行 SELECT 陳述式可以從表格中取得所有列 (SELECT * FROM SAMPLECOLL.INVENTORY_LIST)，並傳回下列結果：

```
                              顯示資料
                                          資料寬度  . . . . . :     71
定位於行. . . . . .             移位至直欄 . . . . . .
....+....1....+....2....+....3....+....4....+....5....+....6....+....7.
ITEM    ITEM                      UNIT   QUANTITY  LAST      NUMBER
NUMBER  NAME                      COST   ON        ORDER     ORDERED
                                         HAND      DATE
153047  鉛筆，紅色                10.00       25   -              20
229740  畫線的筆記本               1.50      120   -              20
544931  ***UNKNOWN***              5.00        -   -              20
303476  紙夾                       2.00      100   05/30/94       50
559343  信封，標準尺寸             3.00      500   -              20
291124  信封，標準                  .00        -   -              20
775298  椅子，祕書               225.00        6   -              20
073956  筆，黑色                  20.00       25   -              20
********  資料結尾  ********
                                                                 底端
F3=跳出      F12=取消        F19=左        F20=右         F21=分割
```

只會變更紙夾的登錄。LAST_ORDER_DATE 已變更爲本日。此日期一律爲執行更新的日期。 NUMBER_ORDERED 會顯示其更新值。

從表格中刪除資訊

您可以使用 SQL DELETE 陳述式，由表格中刪除資料。當表格中的列不再含有必要資訊時，您可以刪除整列，或您可以使用 WHERE 子句與 DELETE 陳述式來識別在單一陳述式執行期間要刪除的列。如需相關資訊，請參閱第 95 頁的『使用 DELETE 陳述式移除表格中的列』。

如需使用交談式 SQL 刪除表格中資訊的範例，請參閱『範例：從表格 (INVENTORY_LIST) 中刪除資訊』。

範例：從表格 (INVENTORY_LIST) 中刪除資訊

如果想要移除表格中 QUANTITY_ON_HAND 直欄值爲 NULL 的所有列，則可以在「輸入 SQL 陳述式」顯示畫面中輸入下列陳述式：

```
DELETE
  FROM SAMPLECOLL.INVENTORY_LIST
  WHERE QUANTITY_ON_HAND IS NULL
```

若要檢查直欄是否有 NULL 值，請使用 IS NULL 比較。完成刪除後，執行另一個 SELECT 陳述式，將會傳回下列結果表格：

```
                                顯示資料
                                                資料寬度  . . . . . :     71
定位於行. . . . . .                          移位至直欄 . . . . . .
....+....1....+....2....+....3....+....4....+....5....+....6....+....7.
ITEM    ITEM                      UNIT   QUANTITY  LAST      NUMBER
NUMBER  NAME                      COST   ON        ORDER     ORDERED
                                         HAND      DATE
153047  鉛筆，紅色                10.00        25   -              20
229740  畫線的筆記本               1.50       120   -              20
303476  紙夾                       2.00       100   05/30/94       50
559343  信封，標準尺寸             3.00       500   -              20
775298  椅子，祕書               225.00         6   -              20
073956  筆，黑色                  20.00        25   -              20
********  資料結尾  ********
                                                                  底端
F3=跳出       F12=取消        F19=左        F20=右        F21=分割
```

即會刪除 QUANTITY_ON_HAND 值爲 NULL 的列。

建立及使用概略表

您可以發現沒有任何單一表格含有您所需的全部資訊。您可能也想提供使用者只存取表格中部份資料的存取權。概略表是提供表格子集的一種方式，使您可以只處理您需要的資料。概略表會減少複雜性，同時也會限制存取權。

您可以使用 SQL CREATE VIEW 陳述式來建立概略表。使用 CREATE VIEW 陳述式來定義表格上的概略表，就像是建立一個新表格，其中只含有您要的直欄和列。當應用程式使用概略表時，它無法存取概略表中沒有的表格列或直欄。然而，如果沒有使用 SQL WITH CHECK OPTION，仍然會在概略表中插入不符合選取準則的列。如需使用 WITH CHECK OPTION 的相關資訊，請參閱第 10 章, 『資料完整性』。

如需使用交談式 SQL 建立概略表的範例，請參閱下面：

- 第 28 頁的『範例：建立單一表格上的概略表』
- 第 28 頁的『範例：建立結合一個以上表格資料的概略表』

爲了建立概略表，您必須對概略表所依據的表格或實體檔案具有適當的權限。如需必要的權限清單，請參閱 *SQL Reference* 中的 CREATE VIEW 陳述式。

如果您沒有在概略表定義中指定直欄名稱，則直欄名稱會與概略表所依據的表格直欄名稱相同。

您可以透過概略表變更表格，即使概略表的直欄數或列數不同於表格。若爲 INSERT，不在概略表中的表格直欄必須具有預設值。

您可以使用概略表，就好像它是一個表格一樣，即使概略表是完全相依於一或多個表格上以取得資料。概略表沒有任何自己的資料，因此，不需要任何儲存體來儲存資料。因爲概略表是衍生自存在於儲存體的表格中，當您更新概略表資料時，實際上是更新表格中的資料。因此，當概略表相依的表格更新時，概略表會自動保有最新資料。

如需相關資訊，請參閱第 53 頁的『建立及使用概略表』。

範例：建立單一表格上的概略表

下列範例會顯示如何在單一表格上建立概略表。概略表是建立在 INVENTORY_LIST 表格上。表格有六個直欄，但概略表只使用其中三個直欄：ITEM_NUMBER、LAST_ORDER_DATE 及 QUANTITY_ON_HAND。 SELECT 子句中的直欄次序是依照它們出現在概略表中的次序。概略表只含有最近兩星期內訂購的項目列。 CREATE VIEW 陳述式類似下面範例：

```
CREATE VIEW SAMPLECOLL.RECENT_ORDERS AS
    SELECT ITEM_NUMBER, LAST_ORDER_DATE, QUANTITY_ON_HAND
         FROM SAMPLECOLL.INVENTORY_LIST
         WHERE LAST_ORDER_DATE > CURRENT DATE - 14 DAYS
```

在上述範例中，概略表中的直欄名稱與表格中的直欄相同，因爲沒有任何直欄清單接在概略表名稱後。建立概略表的綱目，不一定爲建立表格的同一綱目。可以使用任何綱目或檔案庫。下面顯示畫面爲執行 SQL 陳述式的結果：

```
SELECT * FROM SAMPLECOLL.RECENT_ORDERS
```

```
                              顯示資料
                                          資料寬度  . . . . . :     26
定位於行. . . . . .                 移位至直欄 . . . . . .
....+....1....+....2....+.
ITEM    LAST      QUANTITY
NUMBER  ORDER     ON
        DATE      HAND
303476  05/30/94      100
********  資料結尾  ********
                                                                 底端
F3=跳出      F12=取消        F19=左        F20=右        F21=分割
```

只有概略表選取的列會被更新爲具有本日日期的列。表格中的所有其它日期仍爲 NULL 值，所以不會將之傳回。

範例：建立結合一個以上表格資料的概略表

您可以在 FROM 子句中指定一個以上表格，以建立結合兩個以上表格資料的概略表。在下面的範例中，INVENTORY_LIST 表格含有項目編號直欄 *ITEM_NUMBER*，以及項目成本直欄 *UNIT_COST*。這些會與 SUPPLIERS 表格的 *ITEM_NUMBER* 直欄及

SUPPLIER_COST 直欄結合在一起。 WHERE 子句是用來限制傳回的列數。概略表中只含有項目編號，那些項目是來自能夠以比目前單位成本更低的成本來提供項目的供應商。

CREATE VIEW 陳述式類似下面範例：

```
CREATE VIEW SAMPLECOLL.LOWER_COST AS
  SELECT SUPPLIER_NUMBER, A.ITEM_NUMBER, UNIT_COST, SUPPLIER_COST
    FROM SAMPLECOLL.INVENTORY_LIST A, SAMPLECOLL.SUPPLIERS B
    WHERE A.ITEM_NUMBER = B.ITEM_NUMBER
    AND UNIT_COST > SUPPLIER_COST
```

下面表格爲執行 SQL 陳述式的結果：

```
SELECT * FROM SAMPLECOLL.LOWER_COST
```

```
                              顯示資料
                                          資料寬度  . . . . . :     51
定位於行. . . . . .            移位至直欄 . . . . . .
....+....1....+....2....+....3....+....4....+....5.
SUPPLIER_NUMBER  ITEM          UNIT   SUPPLIER_COST
                 NUMBER        COST
     1234        229740         1.50          1.00
     9988        153047        10.00          8.00
     2424        153047        10.00          9.00
     3366        303476         2.00          1.50
     3366        073956        20.00         17.00
********  資料結尾  ********
                                                          底端
F3=跳出      F12=取消        F19=左        F20=右         F21=分割
```

註: 因爲沒有指定查詢的任何 ORDER BY 子句，查詢所傳回的列次序可能會不相同。

透過此概略表所看到的列，只有供應商成本低於單位成本的那些列。

如需使用「交談式 SQL」的相關資訊，請參閱第 265 頁的第 17 章, 『使用交談式 SQL』。

第 3 章 iSeries 領航員資料庫入門

本章說明如何使用 iSeries 領航員來建立及使用檔案庫 (綱目或 SQL 集合)、表格及概略表。「iSeries 領航員資料庫」是一種圖形介面，您可以用來執行許多常用的管理資料庫作業。大部份的 iSeries 領航員作業是基於「結構化查詢語言 (SQL)」，但您不必完全瞭解 SQL 才執行作業。 第 251 頁的第 16 章, 『使用 iSeries 領航員的進階資料庫功能』說明使用 iSeries 領航員的部份進階資料庫功能。在本章中，範例會利用 iSeries 領航員來常用資料庫作業的執行。建立的物件，即是在第 13 頁的第 2 章, 『SQL 入門』中使用「交談式 SQL」範例所建立的相同物件。

請參閱下列主題以取得明細：

- 『啓動 iSeries 領航員』
- 『使用 iSeries 領航員建立檔案庫』
- 第 32 頁的『編輯 iSeries 領航員中顯示的檔案庫清單』
- 第 33 頁的『使用 iSeries 領航員建立及使用表格』
- 第 34 頁的『使用 iSeries 領航員定義表格中的直欄』
- 第 36 頁的『使用 iSeries 領航員複製直欄定義』
- 第 36 頁的『使用 iSeries 領航員在表格中插入資訊』
- 第 37 頁的『使用 iSeries 領航員檢視表格內容』
- 第 38 頁的『使用 iSeries 領航員複製及移動表格』
- 第 39 頁的『使用 iSeries 領航員建立及使用概略表』
- 第 43 頁的『使用 iSeries 領航員刪除資料庫物件』

註: 如需專屬於程式碼範例的相關資訊，請參閱第 x 頁的『程式碼不保事項聲明』資訊。

啓動 iSeries 領航員

啓動 iSeries 領航員，以取得下列範例：

1. 在 **iSeries 領航員**圖示上連按兩下
2. 展開您要使用的系統。

如需「設置 iSeries 領航員」的相關資訊，請參閱瞭解 iSeries 領航員。

使用 iSeries 領航員建立檔案庫

檔案庫是一種資料庫結構，其中含有您的表格、概略表與其它物件類型。您可以使用檔案庫來分組相關物件，並依照名稱尋找物件。您也可以建立檔案庫作爲綱目，並指定資料字典。

綱目 (SQL 集合) 也含有型錄概略表，其中包含在檔案庫中建立的所有表格、概略表、索引、檔案、套裝軟體及限制。建立於綱目中的所有表格，都會自動執行日誌登載。

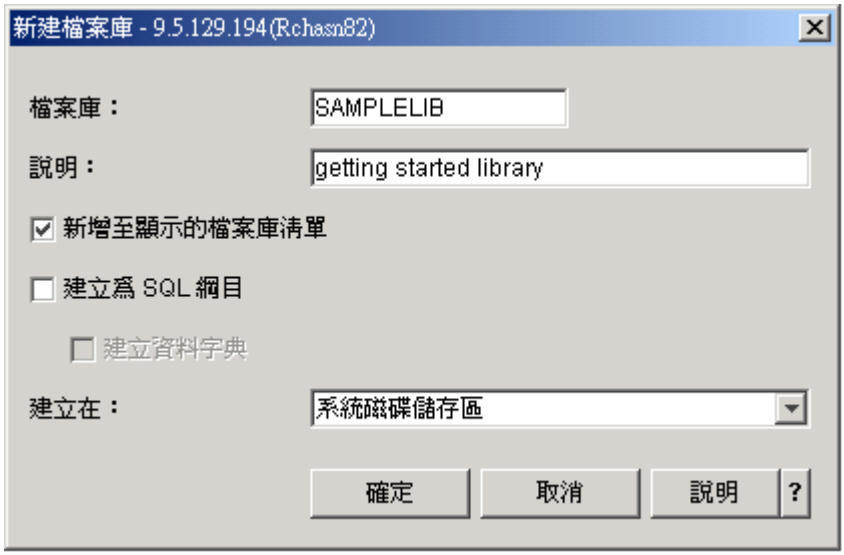
您也可以使用多重資料庫。如需相關明細，請參閱「使用多重資料庫」。

如需如何使用 iSeries 領航員以建立檔案庫 (綱目) 的範例，請參閱『範例：使用 iSeries 領航員建立檔案庫 (SAMPLELIB)』。

範例：使用 iSeries 領航員建立檔案庫 (SAMPLELIB)

您可以建立一個範例檔案庫，命名爲 SAMPLELIB，步驟如下：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器 → **資料庫** → 及您要使用的資料庫。
2. 以滑鼠右鍵按一下**檔案庫**，然後選取**新檔案庫**。
3. 在**新檔案庫**對話框的名稱欄位中，鍵入 SAMPLELIB。
4. 指定說明 (可選用的)。
5. 若要新增至檔案庫清單以便顯示，請選取顯示的**新增至檔案庫清單**。
6. 您可以選取**建立為綱目**以將檔案庫建立爲綱目，然後選取**建立資料字典**以建立資料字典。然而，只需建立此範例檔案庫作爲基本檔案庫即可。
7. 指定磁碟儲存區以包含檔案庫。選擇 1，表示檔案庫會建立在系統磁碟儲存區中。
8. 按一下「確定」。



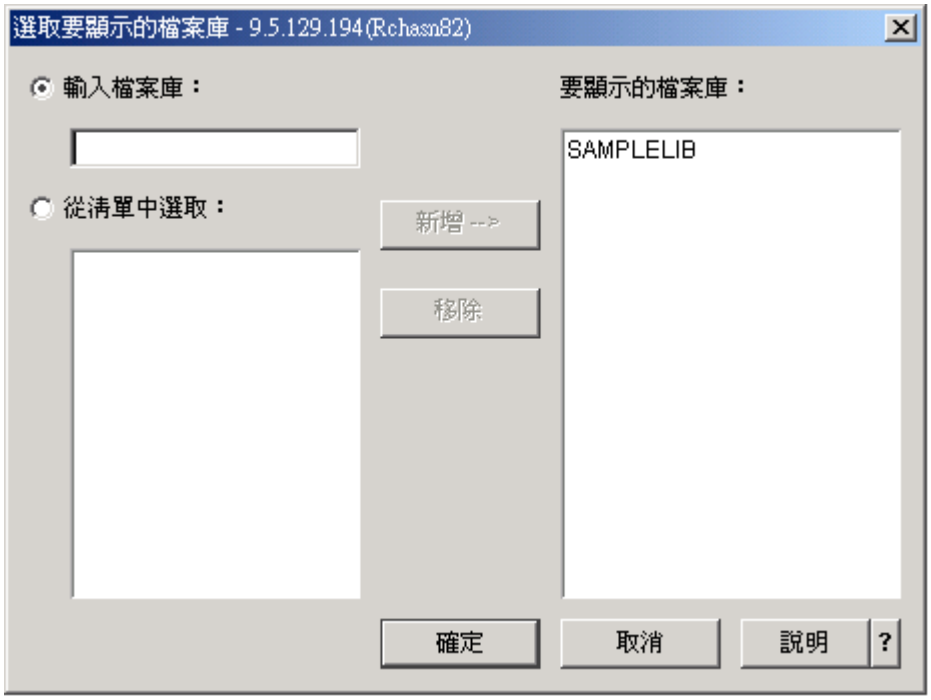
註:

1. 如果已建立 SAMPLELIB 作爲綱目，則會建立數個物件，且此處理可能需要幾秒鐘的時間。
2. 如需在使用者磁碟儲存區中建立檔案庫的相關資訊，請參閱「使用多重資料庫」。

編輯 iSeries 領航員中顯示的檔案庫清單

一旦您已順利建立檔案庫，就可以在其中建立表格、概略表、索引、儲存程序、使用者定義的函數及使用者定義的類型。當您按一下「檔案庫」後，若要編輯顯示的檔案庫清單，請：

1. 以滑鼠右鍵按一下**檔案庫**，然後選取**選取要顯示的檔案庫**。
2. 在**選取要顯示的檔案庫**對話框中，您可以選取檔案庫名稱，然後按一下「新增」，即可編輯它。
3. 您可以從要顯示的檔案庫清單中選取檔案庫，然後按一下**移除**，即可從要顯示的檔案庫清單中移除檔案庫。



立即顯示 SAMPLELIB 檔案庫。

使用 iSeries 領航員建立及使用表格

表格是用來儲存資訊的基本資料庫物件。一旦您已建立表格，就可以使用表格內容對話框來定義直欄、建立索引並新增觸發與限制。

如需使用 iSeries 領航員建立表格的範例，請參閱『範例：使用 iSeries 領航員建立表格 (INVENTORY_LIST)』。

當您要建立表格時，必須瞭解 NULL 值與預設值的概念。 NULL 值表示缺少列的直欄值。此與 0 的值或全部空白不同。它表示「不明」。且不等於任何值，甚至不等於其它 NULL 值。如果直欄不容許 NULL 值，則必須對直欄指定一個值。此值可以是預設值或使用者提供的值。

如果在表格中新增列時，沒有爲直欄指定任何值，則該列會被分派預設值。如果沒有對直欄指定特定的預設值，則直欄會使用系統預設值。如需 INSERT 所使用的預設值之相關資訊，請參閱第 87 頁的『使用 INSERT 陳述式插入列』

範例：使用 iSeries 領航員建立表格 (INVENTORY_LIST)

我們將建立表格，以維護目前業務庫存的相關資訊。表格將包含庫存中所保留的項目相關資訊，包括成本、目前現有的數量、前次訂購日期及前次訂購的數量。項目編號是必要的值。它不能是 NULL。項目名稱、現有數量及訂購數量將具有使用者提供的預設值。前次訂購日期及數量可容許 NULL 值。

若要建立表格：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器 →**資料庫**→ 您要使用的資料庫 → **檔案庫**。
2. 以滑鼠右鍵按一下 SAMPLELIB，然後選取**新增**。

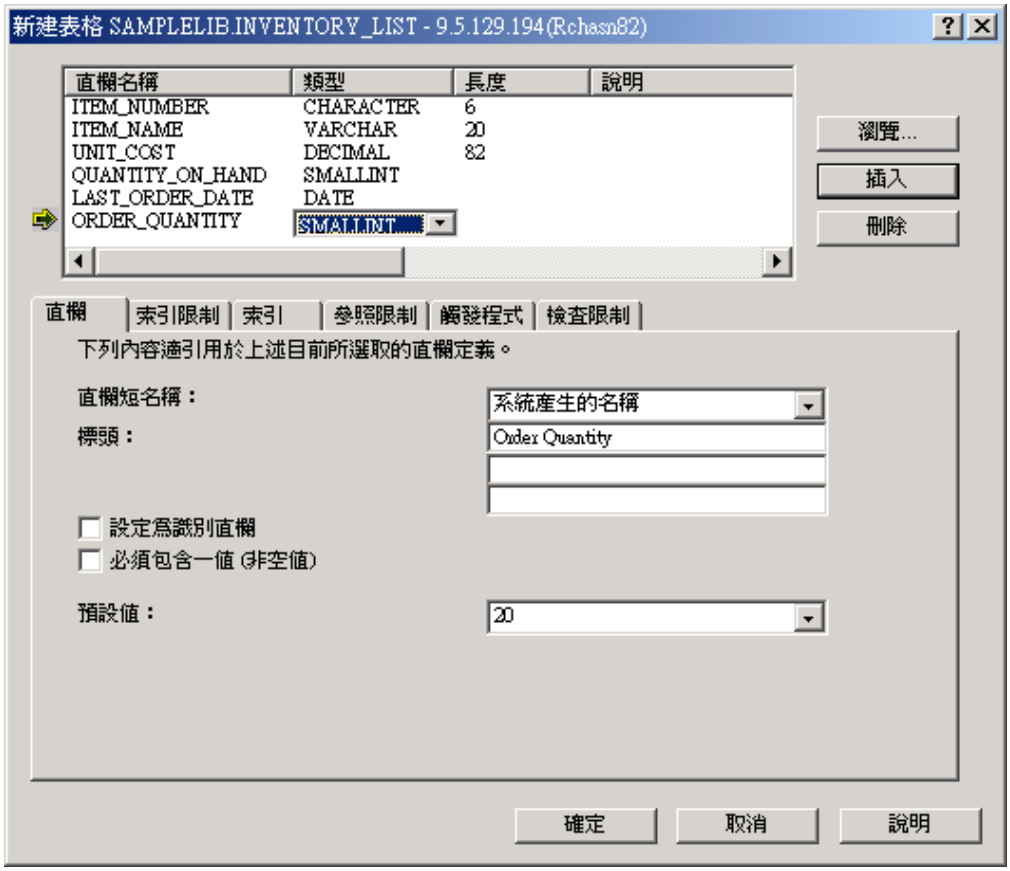
3. 選取**表格**。
4. 在**新增表格於**對話框中，鍵入 INVENTORY_LIST 作爲表格名稱。
5. 指定說明 (可選用的)。
6. 按一下**確定**。

即會出現「新表格 - INVENTORY_LIST」。請勿關閉此對話框；在下一個步驟中，我們會用到它。

使用 iSeries 領航員定義表格中的直欄

您可以定義新的或現有表格上的直欄。如果您要定義現有表格的直欄，請展開**資料庫**→**檔案庫**→*SAMPLELIB* (或適當的檔案庫名稱)，以導覽至表格。在明細窗格中，以滑鼠右鍵按一下表格 *INVENTORY_LIST*，然後選取**內容**。

1. 若要在**表格內容**或**新表格**對話框中定義直欄，請按一下**新增**或**插入**。即會出現**直欄定義格線**的新直行。
2. 在**直欄定義格線**中，輸入名稱 *ITEM_NUMBER*。
3. 請確定類型是 CHARACTER。您可以按一下目前列出的類型，按一下下移鍵，然後從提供的清單中選取新類型，即可變更類型。
4. 指定此直欄的長度爲 6。若爲大小已預先決定的資料類型，則已填入大小，且您不能變更該值。
5. 您可以指定直欄的說明。此步驟是選用的。
6. 在**直欄**欄標下，您可以在**短直欄名稱**文字框中指定簡稱。如果您沒有指定簡稱，則系統會自動產生一個名稱。如果直欄名稱的長度在 10 個字元以下，則「簡稱」與「直欄名稱」是相同的。您可以使用任一直欄名稱來執行查詢。目前，只需將此空間留白即可。
7. 輸入每一個直欄的直欄標頭。
8. 選取**必須內含值 (非 NULL)**。這可確保在此直欄中必須放入一個值，以便能順利完成列的插入。
9. 請確定是以**無預設值**來設定預設值。這是必須輸入值的直欄。



您現在已定義直欄 ITEM_NUMBER。新增下列直欄到「表格 INVENTORY_LIST」：

直欄名稱	類型	長度	小數位數	NULL	預設值
ITEM_NAME	VARCHAR	20		非 null	UNKNOWN
UNIT_COST	DECIMAL	8	2	非 null	(設定爲直欄資料類型預設值)
QUANTITY_ON_HAND	SMALLINT			NULL	NULL
LAST_ORDER_DATE	DATE			NULL	
ORDER_QUANTITY	SMALLINT			NULL	20

當您完成這些直欄的定義時，請按一下**確定**以建立表格。

使用 iSeries 領航員建立供應商表格 (SUPPLIERS)

稍後在我們的範例中，將需要使用第二個表格。此表格將包含庫存項目供應商的相關資訊、他們提供的項目，以及從該供應商取得的項目成本。請在 SAMPLELIB 中建立稱爲 SUPPLIERS 的表格。此表格將有三個直欄：SUPPLIER_NUMBER、ITEM_NUMBER 及 SUPPLIER_COST。請注意：此表格與表格 INVENTORY_LIST 有共用直欄：ITEM_NUMBER。與其建立新的 ITEM_NUMBER 直欄，我們可以複製用於 INVENTORY_LIST 表格中 ITEM_NUMBER 的直欄定義。

使用 iSeries 領航員複製直欄定義

若要複製直欄定義：

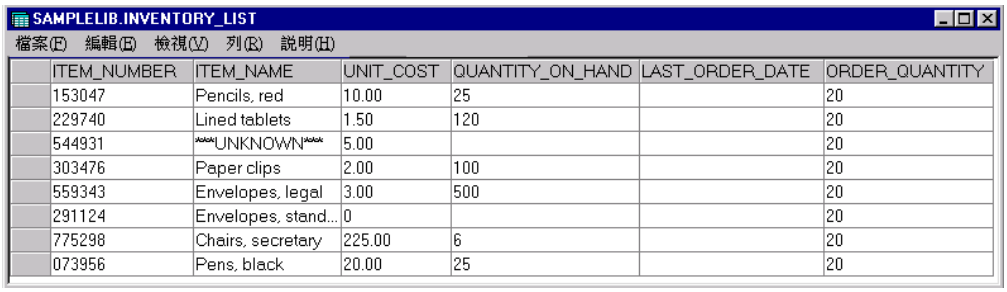
1. 在 SUPPLIER **表格內容**或**新表格**對話框中，按一下**瀏覽**。
2. 在**瀏覽表格**對話框中，展開 SAMPLELIB。
3. 按一下 INVENTORY_LIST。即會列出該表格中的直欄，以及其資料類型、大小與說明。
4. 選取 ITEM_NUMBER。
5. 按一下**確定**，以將此直欄定義複製到表格 SUPPLIERS。

請使用下列值來新增表格 SUPPLIERS 的最後兩個直欄：SUPPLIER_NUMBER、CHAR(4)、NOT NULL 及 SUPPLIER_COST、DECIMAL (8,2)

使用 iSeries 領航員在表格中插入資訊

若要在表格中插入、編輯或刪除資料，您必須具有使用該表格的權限。若要新增資料到表格 INVENTORY_LIST：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器 → **資料庫**→ 您要使用的資料庫 →**檔案庫**。
2. 按一下 SAMPLELIB。
3. 以滑鼠右鍵按一下 INVENTORY_LIST，然後選取**開啓**。
4. 從「列」功能表中，選取**插入**。即會出現新列。
5. 在適當的標頭下，輸入下列資訊。



	ITEM_NUMBER	ITEM_NAME	UNIT_COST	QUANTITY_ON_HAND	LAST_ORDER_DATE	ORDER_QUANTITY
	153047	Pencils, red	10.00	25		20
	229740	Lined tablets	1.50	120		20
	544931	***UNKNOWN***	5.00			20
	303476	Paper clips	2.00	100		20
	559343	Envelopes, legal	3.00	500		20
	291124	Envelopes, stand...	0			20
	775298	Chairs, secretary	225.00	6		20
	073956	Pens, black	20.00	25		20

註： 您輸入的值必須滿足所有限制，並滿足每一個直欄的類型。如果表格上有唯一限制或索引，則您輸入的值必須定義唯一索引鍵值。如果您沒有在直欄中輸入值，則將會輸入預設值，如果系統容許的話。就此練習而言，請勿插入下面圖表中未顯示的值，以便使用預設值。

ITEM_NUMBER	ITEM_NAME	UNIT_COST	QUANTITY_ON_HAND
153047	鉛筆，紅色	10.00	25
229740	畫線的筆記本	1.50	120
544931		5.00	
303476	紙夾	2.00	100
559343	信封，標準尺寸	3.00	500
291124	信封，標準		
775298	椅子，祕書	225.00	6
073956	筆，黑色	20.00	25

從**檔案**功能表中，選取**儲存**。

新增下列橫列到 SAMPLELIB.SUPPLIERS 表格。

SUPPLIER_NUMBER	ITEM_NUMBER	SUPPLIER_COST
1234	153047	10.00
1234	229740	1.00
1234	303476	3.00
9988	153047	8.00
9988	559343	3.00
2424	153047	9.00
2424	303476	2.50
5546	775298	225.00
3366	303476	1.50
3366	073956	17.00

從**檔案**功能表中，選取**儲存**。現在，範例檔案庫中已含有兩個表格，且每一個表格中各有數列資料。

使用 iSeries 領航員檢視表格內容

您可以使用「快速檢視」，以顯示表格及概略表的內容。您只能檢視內容；若要變更表格，就必須開啓表格。若要檢視 INVENTORY_LIST 的內容，請：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器→ **資料庫**→ 您要使用的資料庫 →**檔案庫**。
2. 按一下 SAMPLELIB。
3. 以滑鼠右鍵按一下 INVENTORY_LIST，然後選取**快速檢視**。

新建 SAMPLELIB.INVENTORY_LIST 的內容

	ITEM_NUMBER	ITEM_NAME	UNIT_COST	QUANTITY_ON_HAND	LAST_ORDER_DATE	ORDER_QUANTITY
1	153047	Pencils, red	10.00	25	-	20
2	229740	Lined tablets	1.50	-	-	20
3	559343	Envelopes, lined	3.00	500	-	20
4	153047	Pencils, red	10.00	25	-	20
5	229740	Lined tablets	1.50	120	-	20
6	544931	*UNKNOWN*	5.00	-	-	20
7	303476	Paper clips	2.00	100	-	20
8	291124	Envelopes, standard	0	-	-	20
9	775298	Chairs, secretary	225.00	6	-	20
10	073956	Pens, black	20.00	25	-	20

註：「快速檢視」可讓您存取 Datalink 直欄及其相關的全球資源定位器 (URL)，然後當您選取時即會啓動瀏覽器。

使用 iSeries 領航員變更表格中的資訊

您可以使用 iSeries 領航員來變更表格直欄中的資料值。假設我們想要使用 iSeries 領航員來更新直欄，以表示我們今天收到更多紙夾的訂單。請記住：您輸入的值，必須是該直欄的有效值。

1. 在表格 INVENTORY_LIST 上按兩下以開啓它。
2. 在「紙夾」列的 LAST_ORDER_DATE 直欄中輸入本日日期。務必使用系統的正確日期格式。

3. 將 ORDER_QUANTITY 變更爲 50。
4. 使用**快速檢視**以儲存及檢視表格內容。

紙夾列會反映您所做的變更。

使用 iSeries 領航員刪除表格中的資訊

您可以使用 iSeries 領航員，以刪除表格中的資料。您可以從列的單一直欄中刪除資訊，或刪除整列。請記住：如果直欄需要值，您將無法僅刪除它而沒有刪除整列。

1. 在表格 INVENTORY_LIST 上按兩下以開啓它。
2. 刪除「信封，標準」列的 ORDER_QUANTITY 直欄值。因爲這是容許 NULL 值的直欄，所以我們可以刪除值。
3. 刪除「畫線的筆記本」列的 UNIT_COST 直欄值。因爲此直欄不容許 NULL 值，所以無法刪除。

您也可以刪除整列，而不用一次移除一個直欄值。

1. 在表格 INVENTORY_LIST 上按兩下以開啓它。
2. 按一下 *UNKNOWN* 列左方的灰色資料格。這會以高亮度標示整列。
3. 從「列」功能表中選取**刪除**，或按下鍵盤中的「刪除」鍵。即會刪除 *UNKNOWN* 列。
4. 在表格 INVENTORY_LIST 中，刪除不需要 QUANTITY_ON_HAND 直欄中有值的所有列。
5. 儲存變更，並使用「快速檢視」來檢視內容。您應有一個含下列資料的表格：

ITEM_ NUMBER	ITEM_ NAME	UNIT_ COST	QUANTITY_ ON_ HAND	LAST_ ORDER_ DATE	ORDER_ QUANTITY
153047	鉛筆，紅色	10.00	25		20
229740	畫線的筆記本	1.50	120		20
303476	紙夾	2.00	100	2000-10-02	50
559343	信封，標準尺寸	3.00	500		20
775298	椅子，祕書	225.00	6		20
073956	筆，黑色	20.00	25		20

使用 iSeries 領航員複製及移動表格

iSeries 領航員可讓您將表格從某一檔案庫或系統中複製或移動到另一個檔案庫或系統。複製表格會建立一個以上的表格案例；移動會將表格轉送到它的新位置，同時會將案例從它先前的位置移除。

建立稱爲 LIBRARY1 的新檔案庫，並將它新增到顯示的檔案庫清單中。一旦您已建立此新檔案庫，請將 INVENTORY_LIST 複製到 LIBRARY1。若要複製表格，請：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器→ **資料庫**→ 您要使用的資料庫 →**檔案庫**。
2. 按一下 SAMPLELIB。
3. 以滑鼠右鍵按一下 INVENTORY_LIST，然後選取**複製**。
4. 以滑鼠右鍵按一下 LIBRARY，然後選取**貼上**。

現在，您已將表格 INVENTORY_LIST 複製到 LIBRARY1，請將表格 SUPPLIERS 移到 LIBRARY1。若要移動表格，請：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器→ **資料庫**→ 您要使用的資料庫 →**檔案庫**。
2. 按一下 SAMPLELIB。
3. 以滑鼠右鍵按一下 SUPPIERS，然後選取**剪下**。
4. 以滑鼠右鍵按一下 LIBRARY1，然後選取**貼上**。

註: 您可以在新檔案庫上拖放表格，以移動表格。將表格移到新位置，並不一定會將它從來源系統中移除。例如，如果您具有來源表格的讀取權限，但不具刪除權限，則您可以將表格移到目標系統。然而，您不能從來源系統中刪除表格，造成有兩個表格案例存在。

使用 iSeries 領航員建立及使用概略表

您可以發現沒有任何單一表格含有您所需的全部資訊。您可能也想提供使用者只存取表格中部份資料的存取權。概略表是提供表格子集的一種方式，使您可以只處理您需要的資料。概略表會減少複雜性，同時也會限制存取權。

爲了建立概略表，您必須對概略表所依據的表格或實體檔案具有適當的權限。如需必要的權限清單，請參閱 SQL Reference 中的 CREATE VIEW 陳述式。

如果您沒有在概略表定義中指定直欄名稱，則直欄名稱會與概略表所依據的表格直欄名稱相同。

您可以透過概略表變更表格，即使概略表的直欄數或列數不同於表格。若爲 INSERT，不在概略表中的表格直欄必須具有預設值。

您可以使用概略表，就好像它是一個表格一樣，即使概略表是完全相依於一或多個表格上以取得資料。概略表沒有任何自己的資料，因此，不需要任何儲存體來儲存資料。因爲概略表是衍生自存在於儲存體的表格中，當您更新概略表資料時，實際上是更新表格中的資料。因此，當概略表相依的表格更新時，概略表會自動保有最新資料。

如需相關資訊，請參閱第 53 頁的『建立及使用概略表』。

如需使用 iSeries 領航員建立概略表的範例，請參閱下列範例：

- 『使用 iSeries 領航員建立單一表格上的概略表』
- 第 41 頁的『使用 iSeries 領航員建立結合一個以上表格資料的概略表』

使用 iSeries 領航員建立單一表格上的概略表

下列範例會顯示如何在單一表格上建立概略表。概略表是建立在 INVENTORY_LIST 表格上。表格有六個直欄，但概略表只使用其中三個直欄：ITEM_NUMBER、LAST_ORDER_DATE 及 QUANTITY_ON_HAND。

若要在單一表格上建立概略表：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器→ **資料庫**→ 您要使用的資料庫 →**檔案庫**。
2. 以滑鼠右鍵按一下 SAMPLELIB，並選取**新增**，然後選取**概略表**。
3. 在**新概略表**對話框的**名稱**欄位中，鍵入 RECENT_ORDERS。

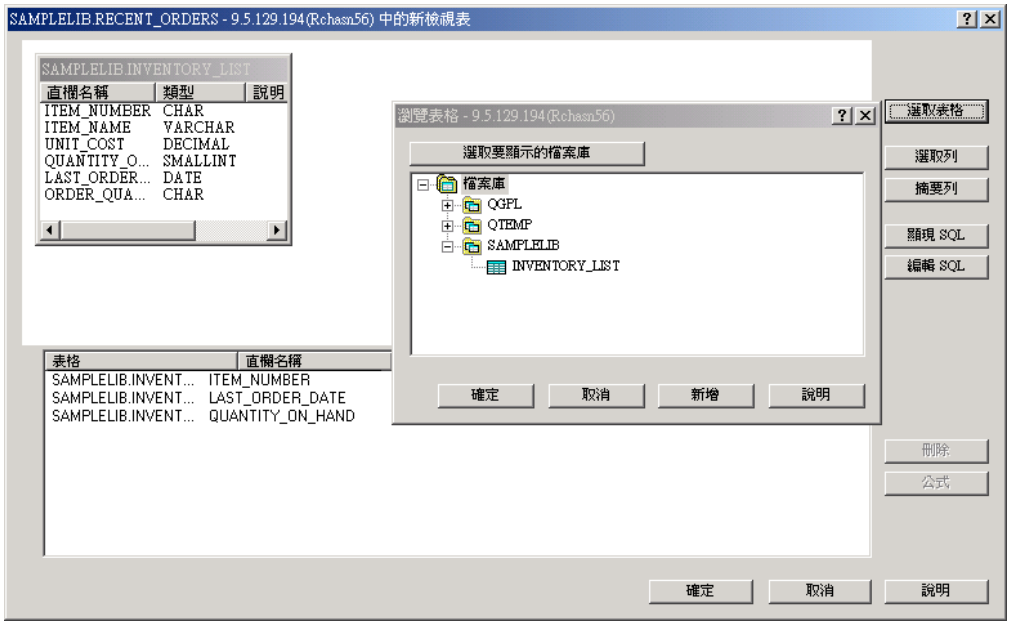
4. 您可以選擇性地指定說明。
5. 此外，請在此對話框中選取核對選項。概略表中的核對選項會指定在列中插入或更新的值，必須符合概略表的條件。如需核對選項的相關資訊，請參閱第 133 頁的『概略表上的 WITH CHECK OPTION』。針對此概略表，請選取**無**。
6. 按一下**確定**。

在**新概略表**對話框中：

1. 按一下**選取表格**。
2. 在**瀏覽表格**對話框中，展開 SAMPLELIB，然後選取 INVENTORY_LIST。
3. 按一下**新增**。
4. 按一下**確定**。 INVENTORY_LIST 現在應在「新概略表」對話框的工作區中。
5. 若要選擇您想要新概略表中使用的直欄，請在選定的表格中按一下那些直欄，然後將它們拖放到對話框下半部的選項格線中。選取 ITEM_NUMBER、LAST_ORDER_DATE 及 QUANTITY_ON_HAND。
6. 直欄在選項格線中出現的次序，就是它們出現在概略表中的次序。若要變更次序，請選取直欄，然後將它拖曳到新位置。請以下列次序放置直欄：ITEM_NUMBER LAST_ORDER_DATE QUANTITY_ON_HAND。

基本上，概略表現在已完成，但我們只想在概略表中顯示最後 14 天內訂購的那些項目。若要指定此資訊，我們必須建立 WHERE 子句：

1. 按一下**選取列**。
2. 在**選取列**對話框中，鍵入下列：WHERE LAST_ORDER_DATE > CURRENT DATE - 14 DAYS。您可以從顯示的選項中選取組成此 WHERE 子句的元素。
3. 按一下**確定**。
4. 若要檢視用來產生此概略表的 SQL，請按一下**顯示 SQL**。
5. 按一下**確定**以建立概略表。



若要顯示 RECENT_ORDERS 的內容，請以滑鼠右鍵按一下 RECENT_ORDERS，然後選取**快速檢視**。您應會看到下列顯示資訊：

ITEM_NUMBER	LAST_ORDER_DATE	QUANTITY_ON_HAND
303476	2000-10-02	100

在上述範例中，概略表中的直欄名稱與表格中的直欄名稱相同，因爲我們沒有指定新名稱。建立概略表的綱目，不一定是建立表格的同一綱目。您可以使用任何綱目或檔案庫。

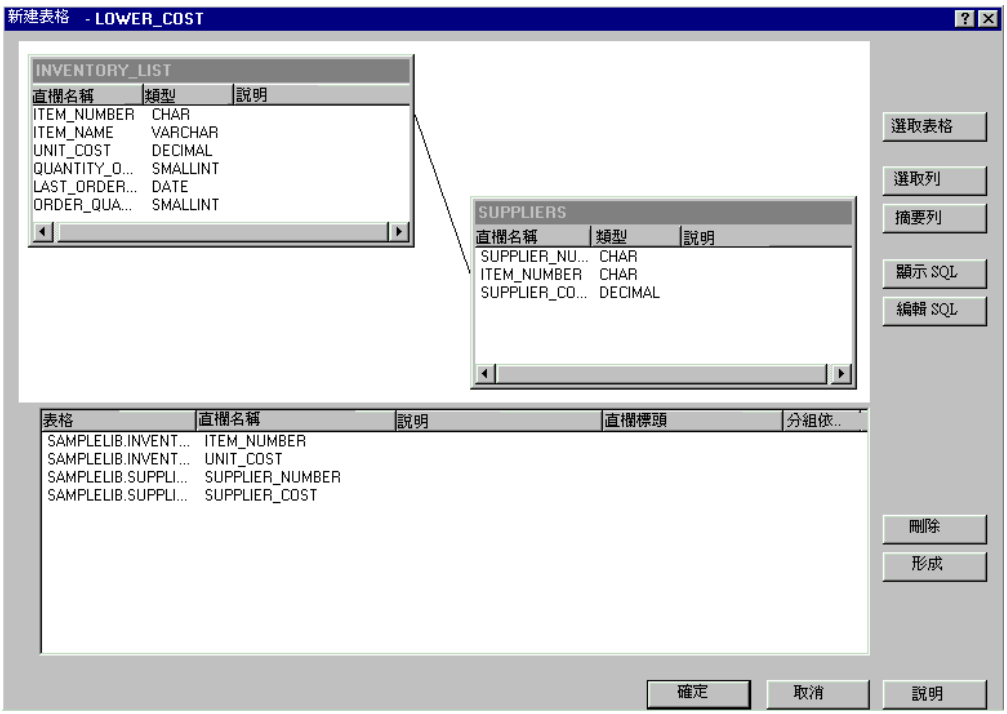
使用 iSeries 領航員建立結合一個以上表格資料的概略表

您可以在「新概略表」對話框的工作區中選取一個以上的表格，以建立結合一個以上表格資訊的概略表。您可以從不同的表格中選取要併入的直欄，然後按一下「確定」，即可從一個以上的表格中建立一個簡式概略表。然而，此範例會顯示如何建立概略表，其中結合了兩個不同表格的資訊且只傳回我們想看到的那些列，非常類似 WHERE 子句。

建立一個概略表，其中只含有項目編號，那些項目是來自能夠以比目前單位成本更低的成本來提供項目的供應商。這將需要從 INVENTORY_LIST 表格中選取 ITEM_NUMBER 及 UNIT_COST，並將它們與 SUPPLIERS 表格中的 SUPPLIER_NUMBER 及 SUPPLIER_COST 相結合。 WHERE 子句是用來限制傳回的列數。

1. 建立稱爲 LOWER_COST 的概略表。
2. 在**新概略表**對話框中，按一下**選取表格**。
3. 從 SAMPLELIB 中選取 INVENTORY_LIST，然後從 LIBRARY1 中選取 SUPPLIERS。
4. 按一下**確定**。兩個表格都將出現在對話框的工作區中。
5. 從 INVENTORY_LIST 中選取 ITEM_NUMBER 及 UNIT_COST。

6. 從 SUPPLIERS 中選取 SUPPLIER_NUMBER 及 SUPPLIER_COST。
7. 若要定義結合，請從 INVENTORY_LIST 中選取 ITEM_NUMBER，然後將它拖曳到 SUPPLIERS 中的 ITEM_NUMBER。即會從某一直欄到另一直欄間畫出一條線，且會開啓**結合**對話框。
8. 在**結合**對話框中，選取**內部結合**。如需「結合」的相關資訊，請參閱第 74 頁的『結合多個表格的資料』。
9. 按一下**確定**。
10. 再一次地，您可以選取**顯示 SQL** 以檢視用來建立此概略表的 SQL。您也可以選取**編輯 SQL** 以編輯 SQL。**編輯 SQL** 會啓動**執行 SQL Script**，您可以在其中編輯 SQL 陳述式。然而，請注意：如果您變更 SQL，則必須從**執行 SQL Script** 來執行陳述式，而不是返回**新概略表**對話框。如果您返回**新概略表**對話框，則不會儲存您的變更。
11. 按一下**選取列**，以建立概略表的 WHERE 子句。在 SUPPLIER_COST 上連按兩下，然後在 < 運算子上連按兩下，最後在 UNIT_COST 上連按兩下。當您按一下項目時，它們就會出現在對話框中。您也可以直接鍵入項目。
12. 按一下**確定**，以建立概略表 LOWER_COST。



若要顯示此新概略表的內容，請以滑鼠右鍵按一下 LOWER_COST，然後選取**快速檢視**。您透過此概略表所看到的列，只有供應商成本低於單位成本的那些列。

SUPPLIER_NUMBER	ITEM_NUMBER	UNIT_COST	SUPPLIER_COST
9988	153047	10.00	8.00
2424	153047	10.00	9.00
1234	229740	1.50	1.00
3366	303476	2.00	1.50
3366	073956	20.00	17.00

使用 iSeries 領航員刪除資料庫物件

一旦您已在系統中建立這些物件，也許想要捨棄它們以節省系統資源。您將需要「刪除」權限以執行這些作業。

註：如果您想要保留這些表格中的資訊，請建立第三個檔案庫，並將表格與概略表複製到該檔案庫中。首先，從 LIBRARY1 中捨棄 INVENTORY_LIST 表格：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器→ **資料庫**→ 您要使用的資料庫 →**檔案庫**。
2. 展開 LIBRARY1。
3. 以滑鼠右鍵按一下 INVENTORY_LIST，然後選取**刪除**，要不然就敲一下「刪除」鍵。
4. 在**物件刪除確認**對話框中，選取**刪除**。即會捨棄 INVENTORY_LIST 表格。

接下來，從 LIBRARY1 刪除 SUPPLIERS：

1. 以滑鼠右鍵按一下 SUPPLIERS，然後選取**刪除**，要不然就敲一下「刪除」鍵。
2. 在「物件刪除確認」對話框中，選取**是**。
3. 即會開啓新對話框，表示概略表 LOWER_COST 是相依於 SUPPLIERS，及是否也應刪除 SUPPLIERS。按一下**刪除**。

即會刪除 SUPPLIERS 及 LOWER_COST。現在，LIBRARY1 是空的，請在它上面按一下滑鼠右鍵，然後選取「刪除」以刪除它。在「物件刪除確認」對話框中，選取「是」。即會刪除 LIBRARY1。

最後，刪除 SAMPLELIB：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器→ **資料庫**→ 您要使用的資料庫 →**檔案庫**。
2. 以滑鼠右鍵按一下 SAMPLELIB，然後選取**刪除**。
3. 在「物件刪除確認」對話框中，選取**刪除**。
4. 即會開啓新對話框，表示表格 INVENTORY_LIST 及概略表 RECENT_ORDERS 是相依於 INVENTORY_LIST，是否應刪除它們。按一下**是**。

即會刪除 SAMPLELIB、INVENTORY_LIST 及 RECENT_ORDERS。

如需使用 iSeries 領航員的相關資訊，請參閱第 251 頁的第 16 章, 『使用 iSeries 領航員的進階資料庫功能』。

第 4 章 資料定義語言 (DDL)

資料定義語言 (DDL) 說明可讓您建立、變更及損毀資料庫物件的 SQL 部份。這些資料庫物件包括綱目、表格、概略表、型錄、索引及別名。

詳細資訊，請參閱下列各節：

- 『建立綱目』
- 第 46 頁的『建立表格』
- 第 46 頁的『使用 LIKE 建立表格』
- 第 47 頁的『使用 AS 建立表格』
- 第 47 頁的『宣告廣域暫時表格』
- 第 48 頁的『建立及變更識別直欄』
- 第 48 頁的『ROWID』
- 第 49 頁的『使用 LABEL ON 陳述式建立描述標籤』
- 第 50 頁的『使用 COMMENT ON 說明 SQL 物件』
- 第 50 頁的『變更表格定義』
- 第 52 頁的『建立及使用 ALIAS 名稱』
- 第 53 頁的『建立及使用概略表』
- 第 55 頁的『新增索引』
- 第 56 頁的『資料庫設計中的型錄』
- 第 57 頁的『捨棄資料庫物件』

建立綱目

綱目提供 SQL 物件的邏輯分組。綱目是由檔案庫、異動日誌、異動日誌接收器、型錄及資料字典(選用性)組成。可建立、移動或復置表格、概略表及系統物件 (例如程式) 至任何系統檔案庫中。如果 SQL 綱目不含資料字典，則所有系統檔可建立或移動至 SQL 綱目中。如果 SQL 綱目含有資料字典，則：

- 來源實體檔或具有一個成員的非來源實體檔可建立、移動或復置到 SQL 綱目中。
- 邏輯檔案無法放置到 SQL 綱目中，因爲它們無法描述於資料字典中。

您可以建立及擁有許多綱目。

使用 CREATE SCHEMA 陳述式來建立綱目。例如：

建立一個稱爲 DBTEMP 的綱目。

```
CREATE SCHEMA DBTEMP
```

您可以使用與*綱目*同義的術語*集合*。有關 CREATE SCHEMA 陳述式的詳細資訊，請參閱 *SQL Reference* 一書中的 CREATE SCHEMA。

建立表格

表格可視爲列與直欄組成的二維資料排列。列是含有一或多個直欄的橫向部分。直欄爲一種資料類型含有一或多列資料之垂直部分。直欄的所有資料類型必須相同。 SQL 中的表格爲索引或非索引實體檔案。有關資料類型的說明，請參閱 SQL Reference 一書中的 Data types 主題。

使用 CREATE TABLE 陳述式建立表格。此定義必須併入其名稱及其直欄的屬性和名稱。此定義可併入表格的其它屬性，例如主要索引鍵。

範例 1：授予您管理權限，建立一個具有下列直欄的表格 'INVENTORY'：

- 產品編號：介於 1 和 9999 之間的整數，不可以是空值
- 說明：長度 0 至 24 的字元
- 現有數量：介於 0 和 100000 之間的整數

主要索引鍵爲 PARTNO。

```
CREATE TABLE INVENTORY
                (PARTNO         SMALLINT     NOT NULL,
                 DESCR          VARCHAR(24 ),
                 QONHAND        INT,
                 PRIMARY KEY(PARTNO))
```

新增表格限制

您可以在新表格或現有的表格中加入一些限制。您可以使用 CREATE TABLE 或 ALTER TABLE 陳述式上的 *ADD constraint* 子句，來新增唯一或主要索引鍵、參照限制或核對限制。例如，將主要索引鍵加入新表格或現有表格中。下列範例說明使用 ALTER TABLE 陳述式將主要索引鍵加入現有表格中。

```
ALTER TABLE CORPDATA.DEPARTMENT
  ADD PRIMARY KEY (DEPTNO)
```

若要使此鍵成爲唯一鍵，只要將關鍵字 PRIMARY 替換成 UNIQUE 即可。詳細資訊，請參閱第 125 頁的第 10 章, 『資料完整性』。

使用 LIKE 建立表格

您可以建立像另一個表格的表格。亦即，您可以建立一個包含現有表格所有直欄定義的表格。複製的定義如下：

- 直欄名稱 (及系統直欄名稱)
- 資料類型、精準度、長度及比例
- CCSID
- 直欄文字 (LABEL ON)
- 直欄標頭 (LABEL ON)

如果 LIKE 子句後緊接表格名稱但未以括弧括住，則也會併入下列屬性：

- 預設值
- 可爲空值

如果指定的表格或概略表含有識別直欄，且您希望該識別直欄存在於新表格中，則必須在 CREATE TABLE 陳述式中指定 INCLUDING IDENTITY。CREATE TABLE 的

預設行爲是 EXCLUDING IDENTITY。如果指定的表格或概略表爲非 SQL 建立的實體檔案或邏輯檔案，則會移除任何非 SQL 屬性。

建立一個併入 EMPLOYEE 中所有直欄的表格 EMPLOYEE2。

```
CREATE TABLE EMPLOYEE2 LIKE EMPLOYEE
```

有關 CREATE TABLE LIKE 的詳細資訊，請參閱 *SQL Reference* 主題中的 CREATE TABLE。

使用 AS 建立表格

CREATE TABLE AS 會根據 SELECT 陳述式結果建立表格。可用於 SELECT 陳述式的所有表示式也可用於 CREATE TABLE AS 陳述式。您也可併入所選取一或多個表格中的所有資料。

例如，建立一個併入 EMPLOYEE (其中 DEPTNO = D11) 所有直欄定義的表格 EMPLOYEE3。

```
CREATE TABLE EMPLOYEE3 AS
   (SELECT PROJNO, PROJNAME, DEPTNO
    FROM EMPLOYEE
    WHERE DEPTNO = 'D11') WITH NO DATA
```

如果指定的表格或概略表含有識別直欄，且您希望該識別直欄存在於新表格中，則必須在 CREATE TABLE 陳述式指定 INCLUDING IDENTITY。 CREATE TABLE 的預設行爲是 EXCLUDING IDENTITY。WITH NO DATA 子句指出所要複製的直欄定義不含資料。如果您要將資料併入新表格 EMPLOYEE3 中，您必須加入 WITH DATA。有關使用 SELECT 的詳細資訊，請參閱第 59 頁的第 5 章, 『使用 SELECT 陳述式擷取資料』。如果指定的查詢併入了非 SQL 建立的實體檔案或邏輯檔案，則會移除任何非 SQL 結果屬性。有關 CREATE TABLE AS 的詳細資訊，請參閱 *SQL Reference* 主題中的 CREATE TABLE。

宣告廣域暫時表格

您可以使用 DECLARE GLOBAL TEMPORARY TABLE 陳述式建立搭配現行階段作業使用的暫時表格。此暫時表格不顯示於系統目錄中且無法爲其它階段作業共用。當您終止階段作業時，將刪除表格的列且表格將被捨棄。

此陳述式的語法類似於 CREATE TABLE，其併入 LIKE 和 AS 子句。

例如，建立暫時表格 ORDERS：

```
DECLARE GLOBAL TEMPORARY TABLE ORDERS
               (PARTNO   SMALLINT NOT NULL,
                DESCR    VARCHAR(24),
                QONHAND  INT)
      ON COMMIT DELETE ROWS
```

此表格以 QTEMP 建立。若要使用綱目名稱參照表格，請使用 SESSION 或 QTEMP。您可以對此表格發出 SELECT、INSERT、UPDATE 及 DELETE 陳述式，就像對其它任何表格一樣。您可以發出 DROP TABLE 陳述式來捨棄此表格：

```
DROP TABLE ORDERS
```

有關詳細資訊，請參閱 *SQL Reference* 主題中的 DECLARE GLOBAL TEMPORARY TABLE。

建立及變更識別直欄

每次加入新列至具有識別直欄的表格中時，系統便會增加 (或減少) 新列中的識別直欄值。僅類型爲 SMALLINT、INTEGER、BIGINT、DECIMAL 或 NUMERIC 的直欄可建立爲識別直欄。每一個表格只能有一個識別直欄。當變更表格定義時，只有您新增的直欄可指定爲識別直欄；現存直欄不行。

當建立表格時，您可以將表格中的直欄定義爲識別直欄。例如，建立一個具有 3 個直欄 ORDERNO、SHIPPED_TO 及 ORDER_DATE 的表格 ORDERS。將 ORDERNO 定義爲識別直欄。

```
CREATE TABLE ORDERS
   (ORDERNO SMALLINT NOT NULL
   GENERATED ALWAYS AS IDENTITY
   (START WITH 500
   INCREMENT BY 1
   CYCLE),
   SHIPPED_TO VARCHAR (36) ,
   ORDER_DATE DATE)
```

此直欄定義起始值爲 500，每插入一列便加 1，當達到最大值時將會重新循環。在本範例中，識別直欄的最大值爲資料類型的最大值。由於資料類型定義爲 SMALLINT，所以可指定給 ORDERNO 的值範圍是由 500 到 32767。當此直欄值達到 32767 時，便會再從 500 重新開始。如果仍指定 500 給某直欄，且對識別直欄指定了唯一鍵，則傳回重複鍵錯誤。下一個插入會嘗試使用 501。如果您沒有對識別直欄指定唯一鍵，則無論它在表格中出現幾次，都會再使用 500。

如需更大範圍的值，您可以將直欄指定爲 INTEGER，甚至爲 BIGINT。如果您要減少識別直欄值，可對 INCREMENT 選項指定負值。也可以使用 MINVALUE 和 MAXVALUE 來指定正確的數字範圍。

您可以使用 ALTER TABLE 陳述式來修改現有識別直欄的屬性。例如，如果您要以新值重新啓動識別直欄：

```
ALTER TABLE ORDER
  ALTER COLUMN ORDERNO
  RESTART WITH 1
```

您也可以由直欄捨棄識別屬性：

```
ALTER TABLE ORDER
  ALTER COLUMN ORDERNO
  DROP IDENTITY
```

直欄 ORDERNO 仍爲 SMALLINT 直欄，但捨棄了識別屬性。系統將不再產生此直欄的值。

ROWID

讓系統指定唯一值給表格中直欄的另一個方法是使用 ROWID。 ROWID 類似識別直欄，但不是數值直欄的屬性，它是一種個別的資料類型。建立一個類似識別直欄範例的表格：

```
CREATE TABLE ORDERS
   (ORDERNO ROWID
   GENERATED ALWAYS,
   SHIPPED_TO VARCHAR (36) ,
   ORDER_DATE DATE)
```

使用 LABEL ON 陳述式建立描述標籤

有時表格名稱、直欄名稱、概略表名稱、別名或 SQL 資料包名稱並無法明確定義表格交談式顯示畫面上顯示的資料。藉由使用 LABEL ON 陳述式，您可以對表格名稱、直欄名稱、概略表名稱、別名或 SQL 資料包名稱建立更詳細的描述性標籤。這些標籤可在 LABEL 直欄的 SQL 型錄中找到。

LABEL ON 陳述式外觀如下：

```
LABEL ON
  TABLE CORPDATA.DEPARTMENT IS 'Department Structure Table'

LABEL ON
  COLUMN CORPDATA.DEPARTMENT.ADMRDEPT IS 'Reports to Dept.'
```

執行這些陳述式後，表格 DEPARTMENT 的文字說明顯示爲*部門結構表格*，而直欄 *ADMRDEPT* 顯示標頭*報告部門*。表格、概略表、SQL 資料包及直欄文字的標籤不能超過 50 個字元，而直欄標頭的標籤不能超過 60 個字元 (包括空白)。下列爲直欄標頭的 LABEL ON 陳述式範例：

此 LABEL ON 陳述式提供直欄標頭 1 和直欄標頭 2。

```
*...+....1....+....2....+....3....+....4....+....5....+....6..*
LABEL ON COLUMN CORPDATA.EMPLOYEE.EMPNO IS
          'Employee            Number'
```

此 LABEL ON 陳述式提供 SALARY 直欄的 3 個直欄標頭層次。

```
*...+....1....+....2....+....3....+....4....+....5....+....6..*
LABEL ON COLUMN CORPDATA.EMPLOYEE.SALARY IS
           'Yearly              Salary              (in dollars)'
```

此 LABEL ON 陳述式移除 SALARY 的直欄標頭。

```
*...+....1....+....2....+....3....+....4....+....5....+....6..*
LABEL ON COLUMN CORPDATA.EMPLOYEE.SALARY IS ''
```

指定兩個層次的 DBCS 直欄標頭範例。

```
*...+....1....+....2....+....3....+....4....+....5....+....6..*
LABEL ON COLUMN CORPDATA.EMPLOYEE.SALARY IS
           '<AABBCCDD>          <EEFFGG>'
```

此 LABEL ON 陳述式提供 EDLEVEL 直欄的直欄文字。

```
*...+....1....+....2....+....3....+....4....+....5....+....6..*
LABEL ON COLUMN CORPDATA.EMPLOYEE.EDLEVEL TEXT IS
   'Number of years of formal education'
```

有關 LABEL ON 陳述式的詳細資訊，請參閱 SQL Reference 一書中的 LABEL ON。

使用 COMMENT ON 說明 SQL 物件

建立 SQL 物件 (例如表格、概略表、索引、資料包、程序、參數、使用者定義的類型、函數或觸發程式) 之後，您可以提供相關資訊來做進一步查核，例如物件用途、使用者及任何不常用或特殊的相關項目。您也可以併入有關表格或概略表每一個直欄的類似資訊。您的註解不可超過 2000 位元組。有關 COMMENT ON 陳述式的詳細資訊，請參閱 *SQL Reference* 一書中的 COMMENT ON。

如果您的名稱未明確指出直欄或物件的內容，註解就特別有用。在此情況下，使用註解來說明直欄或物件的特定內容。

COMMENT ON 的使用範例如下：

```
COMMENT ON TABLE CORPDATA.EMPLOYEE IS
   'Employee table.  Each row in this table represents
    one employee of the company.'
```

執行 COMMENT ON 陳述式後取得註解

對表格執行 COMMENT ON 陳述式後，註解會儲存到 SYSTABLES 的 *LONG_COMMENT* 直欄中。其它物件的註解則儲存於適當型錄表格的 LONG_COMMENT 直欄中。 (如果指示的列已有註解，則新註解會取代舊註解)。下列範例取得上一個範例中之 COMMENT ON 陳述式所新增的註解：

```
SELECT LONG_COMMENT
  FROM CORPDATA.SYSTABLES
  WHERE NAME = 'EMPLOYEE'
```

變更表格定義

變更表格定義可讓您新增直欄、變更現有直欄定義 (變更其長度、預設值等等)、捨棄現有直欄，以及新增和移除限制。使用 SQL ALTER TABLE 陳述式來變更表格定義。

您可以使用一個 ALTER TABLE 陳述式來新增、變更或捨棄直欄，及新增或移除所有限制。不過，單一直欄在 ADD COLUMN、ALTER COLUMN 及 DROP COLUMN 子句中僅被參照一次。亦即，您無法在相同 ALTER TABLE 陳述式中新增直欄後，又變更該直欄。

詳細資訊，請參閱下列主題：

- 『新增直欄』
- 第 51 頁的『變更直欄』
- 第 51 頁的『可允許轉換』
- 第 52 頁的『刪除直欄』
- 第 52 頁的『ALTER TABLE 陳述式的作業次序』

新增直欄

您可以使用 SQL ALTER TABLE 陳述式的 ADD COLUMN 子句，將直欄新增到表格中。

當新增新直欄到表格時，會以其所有現存列的預設值起始設定直欄。如果指定了 NOT NULL，則也必須指定預設值。

已變更的表格最多可以有 8000 個直欄。直欄的位元組數總和不可大於 32766，如果指定的是 VARCHAR 或 VARGRAPHIC 直欄，則不可大於 32740。如果指定一個 LOB 直欄，則直欄的記錄資料位元組數總和不可大於 15 728 640。

變更直欄

您可以使用 ALTER TABLE 陳述式的 ALTER COLUMN 子句，來變更表格中的直欄定義。當變更現有直欄的資料類型時，新舊屬性必須相容。 『可允許轉換』顯示相容屬性的轉換。

當轉換成長度較長的資料類型時，會以適當的填補字元填補資料。當轉換成長度較短的資料類型時，資料會因截斷而遺失。會出現查詢訊息來提示您確認要求。

如果您的直欄不允許空值而您想要變更爲可允許空值，請使用 DROP NOT NULL 子句。如果您的直欄允許空值而不想要使用空值，請使用 SET NOT NULL 子句。如果該直欄中有任何現存值爲空值，則不會執行 ALTER TABLE，且導致 -190 的 SQLCODE。

可允許轉換

表 2. 可允許轉換

FROM 資料類型	**TO 資料類型**
十進位	數值
十進位	Bigint、Integer、Smallint
十進位	浮點
數值	十進位
數值	Bigint、Integer、Smallint
數值	浮點
Bigint、Integer、Smallint	十進位
Bigint、Integer、Smallint	數值
Bigint、Integer、Smallint	浮點
浮點	數值
浮點	Bigint、Integer、Smallint
字元	開放式 DBCS
字元	UCS-2 圖形
開放式 DBCS	字元
開放式 DBCS	UCS-2 圖形
擇一式 DBCS	字元
擇一式 DBCS	開放式 DBCS
擇一式 DBCS	UCS-2 圖形
DBCS 專用	開放式 DBCS
DBCS 專用	DBCS 圖形
DBCS 專用	UCS-2 圖形
DBCS 圖形	UCS-2 圖形
UCS-2 圖形	字元
UCS-2 圖形	開放式 DBCS
UCS-2 圖形	DBCS 圖形

表 2. 可允許轉換 (繼續)

FROM 資料類型	TO 資料類型
特殊類型	來源類型
來源類型	特殊類型

當修改現有的直欄時，僅會變更所指定的屬性。所有其它屬性則不改變。例如，指定下列表格定義：

```
CREATE TABLE EX1 (COL1 CHAR(10) DEFAULT 'COL1',
                  COL2 VARCHAR(20) ALLOCATE(10) CCSID 937,
                  COL3 VARGRAPHIC(20) ALLOCATE(10)
                       NOT NULL WITH DEFAULT)
```

執行下列 ALTER TABLE 陳述式之後：

```
ALTER TABLE EX1 ALTER COLUMN COL2 SET DATA TYPE VARCHAR(30)
                ALTER COLUMN COL3 DROP NOT NULL
```

COL2 仍有配置長度 10 和 CCSID 937，而 COL3 仍有配置長度 10。

刪除直欄

您可以使用 ALTER TABLE 陳述式的 DROP COLUMN 子句來刪除直欄。

捨棄直欄會從表格定義中刪除該直欄。如果指定了 CASCADE，則也會捨棄與該直欄相依的任何概略表、索引及限制。如果指定了 RESTRICT，則任何概略表、索引或限制會與該直欄相依，該直欄不會被捨棄但會發出 -196 的 SQLCODE。

```
ALTER TABLE DEPT
  DROP COLUMN NUMDEPT
```

ALTER TABLE 陳述式的作業次序

ALTER TABLE 陳述式以下列一組步驟來執行：

1. 捨棄限制
2. 捨棄指定 RESTRICT 選項的直欄
3. 變更直欄定義 (這包括新增及捨棄指定 CASCADE 選項的直欄)
4. 新增限制

在每一個步驟內，指定的子句次序就是執行的子句次序，只有一個例外。如果已捨棄任何直欄，且記錄長度隨 ALTER TABLE 陳述式結果增加，則在新增或變更任何直欄定義之前，該作業以邏輯方式完成。

建立及使用 ALIAS 名稱

當您參照現有的表格或概略表，或參照由多個成員組成的實體檔案時，可藉由建立別名來避免使用檔案置換。您可以使用 SQL CREATE ALIAS 陳述式來執行這個操作。

您可以建立下列的別名：

- 表格或概略表
- 表格的*成員*

表格別名定義檔案的名稱，包括特定的成員名稱。您可以在 SQL 陳述式中以使用表格名稱的相同方式使用這個別名。與置換不同的是，別名在被捨棄之前是一直存在的物件。

例如，如果有一個具有成員 MBR1 和 DBR2 的多重成員檔案 MYLIB.MYFILE，則可爲第二個成員建立別名，以便 SQL 可輕易參照它。

```
CREATE ALIAS MYLIB.MYMBR2_ALIAS FOR MYLIB.MYFILE (MBR2)
```

當在下列 insert 陳述式指定別名 MYLIB.MYMBR2_ALIAS 時，值會被插入 MYLIB.MYFILE 的成員 MBR2 中。

```
INSERT INTO MYLIB.MYMBR2_ALIAS VALUES('ABC', 6)
```

在 DDL 陳述式上也可指定別名。假設別名 MYLIB.MYALIAS 存在且爲表格 MYLIB.MYTABLE 的別名。下列 DROP 陳述式會捨棄表格 MYLIB.MYTABLE。

```
DROP TABLE MYLIB.MYALIAS
```

如果您眞的要捨棄別名來替代，請在 DROP 陳述式上指定 ALIAS 關鍵字：

```
DROP ALIAS MYLIB.MYALIAS
```

建立及使用概略表

概略表可用來存取一或多個表格或概略表中的資料。做法是使用 SELECT 陳述式。有關使用 SELECT 子句的詳細資訊，請參閱第 59 頁的第 5 章, 『使用 SELECT 陳述式擷取資料』。以概略表而言，無法使用 ORDER BY 子句。

例如，若要建立一個只選取所有主管的姓氏和部門的概略表，請指定：

```
CREATE VIEW CORPDATA.EMP_MANAGERS AS
   SELECT LASTNAME, WORKDEPT FROM  CORPDATA.EMPLOYEE
   WHERE JOB = 'MANAGER'
```

如果選取清單包含直欄以外的元素 (例如表示式、函數、常數或特別暫存區)，而且未使用 AS 子句來命名直欄，則必須指定直欄清單給概略表。於下列範例中，概略表的直欄爲 LASTNAME 和 YEARSOFSERVICE。

```
CREATE VIEW CORPDATA.EMP_YEARSOFSERVICE
    (LASTNAME, YEARSOFSERVICE) AS
   SELECT LASTNAME, YEARS (CURRENT DATE - HIREDATE)
   FROM CORPDATA.EMPLOYEE
```

也可以定義上一個概略表，方法是使用選取清單中的 AS 子句來爲概略表中的直欄命名。例如：

```
CREATE VIEW CORPDATA.EMP_YEARSOFSERVICE AS
    SELECT LASTNAME,
           YEARS (CURRENT_DATE - HIREDATE) AS YEARSOFSERVICE
    FROM CORPDATA.EMPLOYEE
```

建立概略表後，您可以在 SQL 陳述式中使用它，就像表格名稱一樣。您可以變更基本表格中的資料。

建立概略表時必須考量下列限制：

- 您無法使用唯讀概略表來變更、插入或刪除資料。如果概略表包含下列之一，則爲唯讀概略表：
  - 第一個 FROM 子句定義多個表格 (結合)。

- 第一個 FROM 子句定義唯讀概略表。
- 第一個 FROM 子句定義使用者定義的表格函數。
- 第一個 SELECT 子句含有任何 SQL 直欄函數 (SUM、MAX、MIN、AVG、COUNT、STDDEV、COUNT_BIG 或 VAR)。
- 第一個 SELECT 子句指定關鍵字 DISTINCT。
- 外部子選擇含有 GROUP BY 或 HAVING 子句。
- 外部子選擇含有 UNION 子句。
- 最外部子選擇的基本物件和子查詢表格爲相同表格的子查詢

  在上面的情況中，您可以利用 SQL SELECT 陳述式來取得概略表資料，但無法使用 INSERT、UPDATE 或 DELETE 陳述式。

- 您無法將列插入概略表中，若：
  - 概略表所依據的表格無預設值、不允許空值及不在概略表中的直欄。
  - 概略表具有表示式、常數、函數或特別暫存區產生的直欄，該直欄已指定於 INSERT 直欄清單中。
  - 建立概略表時指定了 WITH CHECK OPTION，但列不符合選取準則。
- 您無法更新表示式、常數、函數或特別暫存區產生的概略表直欄。
- 您無法使用概略表定義中的 FOR UPDATE OF、FOR READ ONLY、FETCH FIRST *n* ROWS、ORDER BY、OPTIMIZE FOR n ROWS 或 isolation 子句。

概略表以執行 CREATE VIEW 陳述式時生效的排序順序建立。此排序順序套用到 CREATE VIEW 子選擇中所有字元和 UCS-2 圖形比較。有關排序順序的詳細資訊，請參閱第 107 頁的第 8 章, 『SQL 的排序順序』。

概略表可使用 UNION 運算子來建立。詳細資訊，請參閱『使用 UNION 建立概略表』。

當透過概略表插入或更新資料時，也可使用 WITH CHECK OPTION 建立概略表，來指定應執行的檢查層次。詳細資訊，請參閱第 133 頁的『概略表上的 WITH CHECK OPTION』。

使用 UNION 建立概略表

利用 UNION 關鍵字，您可以組合兩個或多個子選擇來形成單一概略表。例如：

```
CREATE VIEW D11_EMPS_PROJECTS AS
  (SELECT EMPNO
     FROM CORPDATA.EMPLOYEE
     WHERE WORKDEPT = 'D11'
   UNION
   SELECT EMPNO
     FROM CORPDATA.EMPPROJACT
     WHERE PROJNO = 'MA2112' OR
        PROJNO = 'MA2113' OR
        PROJNO = 'AD3111')
```

概略表中的結果具有下列資料：

表 3. 將概略表建立爲 UNION 結果

EMPNO
000060
000150

表 3. 將概略表建立爲 UNION 結果 (繼續)

EMPNO
000160
000170
000180
000190
000200
000210
000220
000230
000240
200170
200220

有關 UNION 的詳細資訊，請參閱第 81 頁的『使用 UNION 關鍵字結合子選擇』。

新增索引

您可以使用索引來排序及選取資料。此外，索引可協助系統加快資料擷取速度，以獲取最佳查詢效能。

使用 SQL CREATE INDEX 陳述式來建立索引。下列範例透過 CORPDATA.EMPLOYEE 表格中的直欄 *LASTNAME* 建立索引：

```
CREATE INDEX CORPDATA.INX1 ON CORPDATA.EMPLOYEE (LASTNAME)
```

您可以建立任何數目的索引。不過，由於索引是由系統來維護的，所以大量的索引會對效能產生不良影響。有關索引和查詢效能的詳細資訊，請參閱 *Database Performance and Query Optimization* 資訊中的 Effectively Using SQL Indexes。

已編碼的向量索引是一種索引類型，它加快掃描速度，以平行方式更容易處理。您可以使用 SQL CREATE INDX 陳述式來建立已編碼的向量索引。有關使用已編碼的向量索引來加速查詢的詳細資訊 ，請探訪 DB2 for iSeries 網頁。

如果建立的索引與現有索引的屬性完全相同，則新索引共用現有索引的二進位樹。否則，會建立另一個二進位樹。如果新索引屬性與另一索引完全相同 (除了新索引的直欄數較少之外)，則仍會建立另一個二進位樹。由於額外直欄會防止更新它們的 UPDATE 陳述式或游標使用索引，所以仍會建立另一個二進位樹。

索引以執行 CREATE INDEX 陳述式時生效的排序順序建立。此排序順序套用至索引的所有 SBCS 字元欄位和 UCS-2 圖形欄位。有關排序順序的詳細資訊，請參閱第 107 頁的第 8 章, 『SQL 的排序順序』。

資料庫設計中的型錄

型錄會在您建立綱目時自動建立。也有一個全系統型錄固定在 QSYS2 檔案庫中。當在綱目中建立 SQL 物件時，資訊會新增到系統型錄表格和綱目型錄表格中。當在檔案庫中建立 SQL 物件時，僅更新 QSYS2 型錄。使用 DECLARE GLOBAL TEMPORARY TABLE 所建立的表格不會新增至型錄中。有關型錄的詳細資訊，請參閱 SQL Reference 一書。

如下列範例所示，您可以顯示型錄資訊。您無法 INSERT、DELETE 或 UPDATE 型錄資訊。您必須具備編目概略表的 SELECT 專用權，才能執行下列範例。

取得有關表格的型錄資訊

SYSTABLES 含有 SQL 綱目中每個表格及概略表的一列。如果物件爲表格或概略表、物件名稱、物件擁有者、其中的 SQL 綱目等等，它會通知您。

下列範例陳述式顯示 CORPDATA.DEPARTMENT 表格資訊：

```
SELECT *
  FROM CORPDATA.SYSTABLES
  WHERE NAME = 'DEPARTMENT'
```

取得有關直欄的型錄資訊

SYSCOLUMNS 含有綱目中各表格及概略表之每一個直欄的一列。

下列範例陳述式顯示 CORPDATA.DEPARTMENT 表格中所有直欄名稱：

```
SELECT *
  FROM CORPDATA.SYSCOLUMNS
  WHERE TBNAME = 'DEPARTMENT'
```

上一個範例陳述式的結果爲表格中每一直欄的資訊列。某些資訊會因爲資訊寬度比顯示螢幕寬而看不到。

有關每一個直欄的詳細資訊，請指定如下的 SELECT 陳述式：

```
SELECT NAME, TBNAME, COLTYPE, LENGTH, DEFAULT
  FROM CORPDATA.SYSCOLUMNS
  WHERE TBNAME = 'DEPARTMENT'
```

除了每一個直欄的直欄名稱之外，SELECT 陳述式還顯示：

- 含有該直欄的表格名稱
- 直欄的資料類型
- 直欄的長度屬性
- 直欄是否允許預設值

結果如下：

NAME	TBNAME	COLTYPE	LENGTH	DEFAULT
DEPTNO	DEPARTMENT	CHAR	3	N
DEPTNAME	DEPARTMENT	VARCHAR	29	N
MGRNO	DEPARTMENT	CHAR	6	Y
ADMRDEPT	DEPARTMENT	CHAR	3	N

捨棄資料庫物件

DROP 陳述式可刪除物件。根據所要求的動作，直接或間接與該物件相依的任何物件也會被刪除，或可防止發生捨棄。例如，如果您捨棄表格，則與該表格相關的任何別名、限制、觸發、概略表或索引也會被捨棄。當刪除某物件時，其說明也會從型錄中刪除。

例如，若要捨棄表格 EMPLOYEE，可發出下列陳述式：

```
DROP TABLE EMPLOYEE RESTRICT
```

詳細資訊，請參閱 SQL Reference 一書中的 DROP 陳述式。

第 5 章 使用 SELECT 陳述式擷取資料

您可以使用 SELECT 陳述式從資料庫擷取資料。這一節會說明下列各節：

- 『使用 SELECT 陳述式查詢資料』

您可以使用一行或許多行來撰寫 SQL 陳述式。在經過前置編譯的程式中之 SQL 陳述式，其各行延續的規則和主語言 (撰寫程式所使用的語言) 的規則相同。

註:

1. 本節說明的 SQL 陳述式可以執行於 SQL 表格與概略表，以及資料庫實體和邏輯檔案之上。表格、概略表和檔案可以放在網目或檔案庫中。
2. SQL 陳述式中指定的字元字串 (例如與 WHERE 或 VALUES 子句一起使用的那些字串) 有區分大小寫；亦即，大寫字元必須以大寫輸入，小寫字元必須以小寫輸入。

   ```
   WHERE ADMRDEPT='a00'      (不會傳回結果)

   WHERE ADMRDEPT='A00'      (傳回有效的部門編號)
   ```

   如果使用共用權重排序順序，大小寫字元都當成相同的字元處理，比較時就不會區分大小寫。

使用 SELECT 陳述式查詢資料

您可以利用各種陳述式和子句查詢資料。方法之一是在程式中使用 SELECT 陳述式擷取特定的列 (例如員工列)。此外，在本範例中也會使用各種子句，以特定方法收集資料。SQL 提供許多修改查詢，以特定方式收集資料的方法。這些方法如下：

- 第 60 頁的『使用 WHERE 子句 指定搜尋條件』
- 第 63 頁的『GROUP BY 子句』
- 第 65 頁的『HAVING 子句』
- 第 66 頁的『ORDER BY 子句』

這裡所顯示的是非常基本的格式和語法。SELECT 陳述式可以比本章使用的範例有更多的變化。SELECT 陳述式可以包含下列：

1. 您要包括的每一個直欄名稱
2. 包含資料的表格或概略表之名稱
3. 能夠唯一識別包含您所要資訊之列的搜尋條件
4. 用於將您資料分組之每一直欄的名稱
5. 能夠唯一識別包含您所要資訊之群組的搜尋條件
6. 能夠傳回複製中特定列的結果次序

SELECT 陳述式的結構如下：

```
SELECT 直欄名稱
  FROM 表格或概略表名稱
  WHERE 搜尋條件
  GROUP BY 直欄名稱
  HAVING 搜尋條件
  ORDER BY 直欄名稱
```

必須要指定 SELECT 和 FROM 子句。其他子句則屬於可選用的。

您可以使用 SELECT 子句來指定要擷取的每一個直欄的名稱。例如：

```
SELECT EMPNO, LASTNAME, WORKDEPT
  ⋮
```

可以指定只擷取一個直欄，或者擷取多達 8000 個直欄。會以 SELECT 子句中指定的次序擷取您所指定的每一個直欄之值。

如果要擷取全部的直欄 (依照表格定義中顯示的次序擷取)，請使用星號 (*) 取代指定各直欄：

```
SELECT *
  ⋮
```

FROM 子句指定您要選取資料的*來源*表格。您可以選取多個表格中的直欄。發出 SELECT 時，必須指定 FROM 子句。發出下列陳述式：

```
SELECT *
       FROM EMPLOYEE
```

結果是 EMPLOYEE 表格中全部的直欄和列。

SELECT 清單也可以包含表示式，包括常數、特別暫存區及純量子選擇。也可以使用 AS 子句爲結果直欄命名。例如，發出下列陳述式：

```
SELECT LASTNAME, SALARY * .05 AS RAISE
       FROM EMPLOYEE
       WHERE EMPNO = '200140'
```

這個陳述式的結果如下：

表 4. 查詢結果

LASTNAME	RAISE
NATZ	1421

如果 SQL 無法找到可滿足搜尋條件的列，會傳回 +100 的 SQLCODE。

如果 SQL 在執行您的 SELECT 陳述式時發現錯誤，會傳回負的 SQLCODE。如果 SQL 找到比結果更多的主變數，則會傳回 +326。

如需定義傳回之資料的詳細資訊，請參閱『使用 WHERE 子句 指定搜尋條件』。

使用 WHERE 子句 指定搜尋條件

WHERE 子句會指定識別您要擷取、更新或刪除的一或多列之搜尋條件。您使用 SQL 陳述式處理的列數則視滿足 WHERE 子句**搜尋條件**的列數而定。搜尋條件包含一或多個**述語**。述語指定了您希望 SQL 套用至表格中之指定列的測試。如需述語的詳細資訊，請參閱第 71 頁的『執行複雜搜尋條件』。

在下列範例中，WORKDEPT = 'C01' 是一個述語， WORKDEPT 和 'C01' 是表示式，等號 (=)是比較運算子。請注意字元值是以單引號 (') 括住；數值則沒有。這適用於所有的常數值 (不論是加在 SQL 陳述式中任何位置)。例如，若您想要指定部門編號爲 C01 的列，可以這麼寫：

```
... WHERE WORKDEPT = 'C01'
```

在此例中，搜尋條件包含一個述語：WORKDEPT = 'C01'。

爲了進一步說明 WHERE，可將其放入 SELECT 陳述式中。假設 CORPDATA.DEPARTMENT 表格中所列的每一個部門都有唯一的部門編號。若您想要從 CORPDATA.DEPARTMENT 表格擷取 C01 部門的部門名稱和管理員編號。請發出下列陳述式：

```
SELECT DEPTNAME, MGRNO
       FROM CORPDATA.DEPARTMENT
       WHERE DEPTNO = 'C01'
```

執行這個陳述式的結果會產生一列：

表 5. 結果表格

DEPTNAME	MGRNO
INFORMATION CENTER	000030

如果搜尋條件包含字元或 UCS-2 圖形直欄述語，則在執行查詢時生效的排序順序會套用至這些述語。如需排序順序和選擇的詳細資訊，請參閱第 107 頁的第 8 章, 『SQL 的排序順序』。如果未使用排序順序，必須以大寫或小寫指定字元常數，使其與所比較的直欄或表示式相符。

如需使用 WHERE 子句的詳細資訊，請參閱下列各節：

- 『WHERE 子句中的表示式』
- 第 62 頁的『比較運算子』
- 第 63 頁的『NOT 關鍵字』

WHERE 子句中的表示式

WHERE 子句中的表示式指名或指定您要與其他內容比較的內容。每一個表示式在 SQL 的評估下，是一個字元字串、日期 / 時間 / 時間戳記或數值。您指定的表示式可以是：

- **直欄名稱**指定直欄的名稱。例如：

  ```
  ... WHERE EMPNO = '000200'
  ```

  EMPNO 指定一個定義爲 6 位元組字元值的直欄名稱。可以執行字元資料的相等比較 (就是 X = Y 或 X <> Y)。也可以對字元資料進行其他類型的比較。

  但是，您不能拿字元字串和數值做比較。也不能對字元資料執行運算作業 (雖然 EMPNO 是顯示爲數字的字元字串)。可以使用強制轉型函數將字元和數值資料轉換爲可以比較的值。您可以增加和扣除日期 / 時間值和持續時間。

- **表示式**會識別加 (+)、減 (-)、乘 (*)、除 (/)、乘冪 (**) 或連接 (CONCAT 或 ||) 至數值之結果的兩個值。表示式的運算元可爲：

  常數

  直欄

主變數
從函數傳回的值
特別暫存區
子查詢
另一個表示式

例如：

```
... WHERE INTEGER(PRENDATE - PRSTDATE) > 100
```

如果未以括弧指定評估次序，將會以下列次序評估表示式：

1. 字首運算子
2. 乘冪
3. 乘、除和連接
4. 加和減

同一優先順序層次的運算子會由左至右套用。

- **常數**指定表示式的文字值。例如：

  ```
  ... WHERE 40000 < SALARY
  ```

  SALARY 指定一個定義爲 9 位數壓縮十進位數值 (DECIMAL(9,2)) 的直欄指定名稱。以其與數值常數 40000 比較。
- **主變數**定義應用程式中的一個變數。例如：

  ```
  ... WHERE EMPNO = :EMP
  ```
- **特別暫存區**定義由資料庫管理程式定義的特殊值。例如：

  ```
  ... WHERE LASTNAME = USER
  ```
- **NULL** 值指定擁有不明值的狀況。

  ```
  ... WHERE DUE_DATE IS NULL
  ```
- 子查詢。如需使用子查詢的詳細資訊，請參閱第 97 頁的第 7 章, 『使用子查詢』。

搜尋條件不一定要侷限於使用運算或比較運算子分隔的兩個直欄名稱或常數。您可以開發複雜的搜尋條件，利用 AND 和 OR 分隔指定許多述語。不論搜尋條件如何複雜，針對列進行評估時都會提供 TRUE 或 FALSE 值。另外還有一個*不明*的眞值，實際上爲假。也就是，列的值若爲空值，不會將此空值當成搜尋結果傳回，因爲這並不是搜尋條件中指定的小於、等於或大於值。更複雜的搜尋條件和述語會於第 71 頁的『執行複雜搜尋條件』中說明。

若要徹底瞭解 WHERE 子句，必須知道 SQL 是如何評估搜尋條件和述語，以及比較表示式的值。這個主題在 SQL 參照一書中有詳細的討論。

比較運算子

SQL 支援下列比較運算子：

=	等於
<> 或 ¬= 或 !=	不等於
<	小於
>	大於

<= 或 ¬> 或 !>	小於或等於 (或不大於)
> = 或 ¬< 或 !<	大於或等於 (或不小於)

NOT 關鍵字

您可以在述語之前加上 NOT 關鍵字，指定要將述語值相反 (如果述語是 FALSE，就變成 TRUE，或 TRUE 變成 FALSE)。NOT 只會套用至其後面的述語，不會套用至 WHERE 子句中所有的述語。例如，若您想要指示部門 C01 以外的所有員工，可以寫成：

```
... WHERE NOT WORKDEPT = 'C01'
```

其效用等於：

```
... WHERE WORKDEPT <> 'C01'
```

GROUP BY 子句

不使用 GROUP BY 子句時，套用 SQL 直欄功能會傳回一列。使用 GROUP BY 時，功能會套用至*每一個*群組，因此會傳回群組中所有的列。

GROUP BY 子句可以讓您尋找整組列的性質而非個別列。您指定 GROUP BY 子句時，SQL 會將選取的列分成群組，使每一個群組中的列在一或多個直欄或表示式中都有符合值。接著，SQL 會處理每一個群組，以產生群組的單列結果。您可以在 GROUP BY 子句中指定一或多個直欄或表示式，將各列分組。您在 SELECT 陳述式中指定的項目是每一組列的內容，而非表格或概略表中個別列的內容。

例如，CORPDATA.EMPLOYEE 表格有許多組列，每一組都包含說明特定部門成員的列。若要尋找每一部門中人員的平均薪資，可以發出：

```
SELECT WORKDEPT, DECIMAL (AVG(SALARY),5,0)
       FROM CORPDATA.EMPLOYEE
       GROUP BY WORKDEPT
```

結果會有許多列，每一列代表一個部門。

WORKDEPT	AVG-SALARY
A00	40850
B01	41250
C01	29722
D11	25147
D21	25668
E01	40175
E11	21020
E21	24086

註:

1. 將列分組並不表示將其排序。分組會將每一選取的列置於群組中，SQL 再處理群組以取得群組性質。列的排序會以升序或降序對照順序將所有列放在結果表中 (第 66 頁的『ORDER BY 子句』說明其執行方式。) 根據資料庫管理員選取的施行，產生的群組可能會以排序顯示。

2. 如果您在 GROUP BY 子句中指定的直欄中有 NULL 值，會爲包含 NULL 值之列中的資料產生單列結果。
3. 如果是在字元或 UCS-2 圖形直欄分組，則在執行查詢時生效的排序順序會套用至分組。如需排序順序和選擇的詳細資訊，請參閱第 107 頁的第 8 章, 『SQL 的排序順序』。

您使用 GROUP BY 時，會列出希望 SQL 用於將列分組的直欄或表示式。例如，假設您想要一份參與 CORPDATA.PROJECT 表格中說明之每一主要專案的人數清單。您可以發出：

```
SELECT SUM(PRSTAFF), MAJPROJ
       FROM CORPDATA.PROJECT
       GROUP BY MAJPROJ
```

其結果是公司目前的主要專案和參與每一個專案之人數的清單。

SUM(PRSTAFF)	MAJPROJ
6	AD3100
5	AD3110
10	MA2100
8	MA2110
5	OP1000
4	OP2000
3	OP2010
32.5	?

您也可以指定希望依據多個直欄或表示式將列分組。例如，您可以發出一個 SELECT 陳述式，使用 CORPDATA.EMPLOYEE 表格尋找每一部門之男女性的平均薪資。其方式是發出：

```
SELECT WORKDEPT, SEX, DECIMAL(AVG(SALARY),5,0) AS AVG_WAGES
       FROM CORPDATA.EMPLOYEE
       GROUP BY WORKDEPT, SEX
```

結果爲：

WORKDEPT	SEX	AVG_WAGES
A00	F	49625
A00	M	35000
B01	M	41250
C01	F	29722
D11	F	25817
D11	M	24764
D21	F	26933
D21	M	24720
E01	M	40175
E11	F	22810
E11	M	16545
E21	F	25370

WORKDEPT	SEX	AVG_WAGES
E21	M	23830

由於您在這個範例中並未包含 WHERE 子句，因此 SQL 會檢查及處理 CORPDATA.EMPLOYEE 表格中全部的列。這些列會先依據部門編號，再依據 (每一部門中的) 性別分組，然後得到每一群組的平均 SALARY 值。

HAVING 子句

您可以使用 HAVING 子句指定根據 GROUP BY 子句選取之群組的搜尋條件。HAVING 子句表示您只要滿足該子句中條件的群組。因此，您在 HAVING 子句中指定的搜尋條件必須測試每一個群組的內容，而非測試群組中個別列的內容。

HAVING 接在 GROUP BY 子句之後，可以包含在 WHERE 子句中指定之同類搜尋條件。此外，您也可以在 HAVING 子句中指定直欄功能。例如，假設您希望擷取每一部門中女性的平均薪資。可以使用 AVG 直欄功能，並使用 WORKDEPT 將結果列分組，且指定 SEX = 'F' 的 WHERE 子句。

若要指定唯有在選取的部門中所有女性員工的教育程度等於或大於 16 (大學畢業) 時，才想要這份資料，請使用 HAVING 子句。HAVING 子句會測試群組的內容。在此例中是測試 MIN(EDLEVEL) 群組內容：

```
SELECT WORKDEPT, DECIMAL(AVG(SALARY),5,0) AS AVG_WAGES, MIN(EDLEVEL) AS MIN_EDUC
       FROM CORPDATA.EMPLOYEE
       WHERE SEX='F'
       GROUP BY WORKDEPT
       HAVING MIN(EDLEVEL)>=16
```

結果爲：

WORKDEPT	AVG_WAGES	MIN_EDUC
A00	49625	18
C01	29722	16
D11	25817	17

您可以在 HAVING 子句中使用多個述語，方法是利用 AND 和 OR 來連接，也可以在搜尋條件的任何述語使用 NOT。

註: 如果想要更新直欄或刪除列，不能在 DECLARE CURSOR 的 SELECT 陳述式中包含 GROUP BY 或 HAVING 子句 (DECLARE CURSOR 陳述式會在第 113 頁的第 9 章, 『使用游標』中討論)。這些子句會使其成爲唯讀游標。

具有非直欄功能之引數的述語可以放在 WHERE 或 HAVING 子句中。通常將選取準則加入 WHERE 子句中比較有效率，因爲於查詢處理程序中會先處理這個子句。 HAVING 選項執行於結果表格的後處理程序中。

如果搜尋條件包含有字元或 UCS-2 圖形直欄的述語，則在執行查詢時生效的排序順序會套用至這些述語。如需排序順序和選項的詳細資訊，請參閱第 107 頁的第 8 章, 『SQL 的排序順序』。

ORDER BY 子句

您可以使用 ORDER BY 子句，指定希望所選取的列以特定的次序傳回，依據直欄或表示式之值的升序或下降對照順序排序。可以像使用 GROUP BY 子句一樣使用 ORDER BY 子句：指定您希望 SQL 以對照順序擷取列時要使用的直欄或表示式。

例如，若要擷取以部門編號之英數次序列出之女性員工的姓名和部門編號，可以使用 SELECT 陳述式：

```
SELECT LASTNAME,WORKDEPT
       FROM CORPDATA.EMPLOYEE
       WHERE SEX='F'
       ORDER BY WORKDEPT
```

結果爲：

LASTNAME	WORKDEPT
HAAS	A00
HEMMINGER	A00
KWAN	C01
QUINTANA	C01
NICHOLLS	C01
NATZ	C01
PIANKA	D11
SCOUTTEN	D11
LUTZ	D11
JOHN	D11
PULASKI	D21
JOHNSON	D21
PEREZ	D21
HENDERSON	E11
SCHNEIDER	E11
SETRIGHT	D11
SCHWARTZ	E11
SPRINGER	E11
WONG	E21

註: NULL 值會排序爲最高的值。

ORDER BY 子句中指定的直欄不必包含在 SELECT 子句中。例如，以下陳述式會傳回全部的女性員工，薪資最高的排在前面：

```
SELECT LASTNAME,FIRSTNME
       FROM CORPDATA.EMPLOYEE
       WHERE SEX='F'
       ORDER BY SALARY DESC
```

如果指定 AS 子句爲選取清單中的結果直欄命名，可以在 ORDER BY 子句中指定這個名稱。AS 子句中指定的名稱在選取清單中必須是唯一的。例如，若要擷取以英數次序列出的員工全名，可以使用這個 SELECT 陳述式：

```
SELECT LASTNAME CONCAT FIRSTNME AS FULLNAME
  FROM CORPDATA.EMPLOYEE
  ORDER BY FULLNAME
```

這個 SELECT 陳述式也可以選用性地寫成：

```
SELECT LASTNAME CONCAT FIRSTNME
  FROM CORPDATA.EMPLOYEE
  ORDER BY LASTNAME CONCAT FIRSTNME
```

您可以使用編號代替指定直欄名稱來排序結果。例如，ORDER BY 3 指定您想要依據結果表的*第三個*直欄排序結果，如同選取清單所指定的。如果順序值不是指名的直欄，請使用編號排序結果表的列。

您也可以指定希望 SQL 以升序 (ASC) 或降序 (DESC) 順序理序各列。預設值爲升序對照順序。在前面的 SELECT 陳述式中，SQL 會先傳回 *FULLNAME* 表示式最低 (英數和數字順序) 的列，然後再傳回值較高的列。若要根據這個名稱以降序對照順序排序列，請指定：

```
... ORDER BY FULLNAME DESC
```

使用 GROUP BY 時，可以指定次要排序順序 (或許多層排序順序) 和主要排序順序。在前面的範例中，您可能希望列先依據部門編號排序，每一部門中再依據員工姓名排序。其方式是指定：

```
... ORDER BY WORKDEPT, FULLNAME
```

如果 ORDER BY 子句中使用了字元直欄或 UCS-2 圖形直欄，這些直欄會根據執行查詢時生效的排序順序排序。如需排序順序及其對排序影響的詳細資訊，請參閱第 107 頁的第 8 章, 『SQL 的排序順序』。

靜態 SELECT 陳述式

靜態 SELECT 陳述式 (SQL 程式中內含的) 中，必須在 FROM 子句之前指定 INTO 子句。INTO 子句會指定主變數 (您的程式中用於包含擷取之直欄值的變數) 的名稱。SELECT 子句中指定之第一個結果直欄的值會放入 INTO 子句中指名的第一個主變數中；第二個值會放入第二個主變數中，依此類推。

SELECT INTO 的結果表格應只包含一列。例如，CORPDATA.EMPLOYEE 表格中每一列都有一個唯一的 EMPNO (員工編號) 直欄。如果 WHERE 子句包含 EMPNO 直欄的等於比較，這個表格的 SELECT INTO 陳述式結果正好是一列 (或沒有列)。如果發現多列就是錯誤，不過還是會傳回一列。您可以指定 ORDER BY 子句，控制在此錯誤狀況下應傳回哪一列。如果使用 ORDER BY 子句，則傳回結果表格中的第一列。

如果希望 SELECT INTO 陳述式的結果是一列以上，請使用 DECLARE CURSOR 陳述式選取各列，後面加上 FETCH 陳述式，將直欄值移入主變數，一次一或多列。使用游標說明於第 113 頁的第 9 章, 『使用游標』中。

在應用程式中使用 SELECT 陳述式時，請列出直欄名稱，讓您的程式與資料較無關。這麼做有兩個理由：

1. 在您檢視原始程式碼陳述式時，可以輕易看到 SELECT 子句的直欄名稱和 INTO 子句中指定的主變數間的一對一對應。

2. 如果將直欄加入您存取的表格或概略表，而且您使用 『SELECT * ...,』，並且再次利用原始碼建立程式，INTO 子句並沒有與新直欄所指定的變數相符。額外的直欄會造成您在 SQLCA (SQLWARN4 將包含一個 『W』) 中得到警告 (不是錯誤)。

指示列中缺少直欄值的 NULL 值

NULL 值指示列中缺少直欄值。NULL 值不同於零或完全空白。NULL 值和『不明』相同。 NULL 值可以作爲 WHERE 和 HAVING 子句中的條件，也可以當成數學引數。例如，WHERE 子句可以指定某些列中包含 NULL 值的直欄。通常，使用包含 NULL 值之直欄的比較述語不會選取直欄中有 NULL 值的列。因爲 NULL 值即非小於、等於，也非大於條件中指定的值。若要選取管理員編號爲 NULL 值之所有列的值，可以指定：

```
SELECT DEPTNO, DEPTNAME, ADMRDEPT
  FROM CORPDATA.DEPARTMENT
  WHERE MGRNO IS NULL
```

結果爲：

DEPTNO	DEPTNAME	ADMRDEPT
D01	DEVELOPMENT CENTER	A00
F22	BRANCH OFFICE F2	E01
G22	BRANCH OFFICE G2	E01
H22	BRANCH OFFICE H2	E01
I22	BRANCH OFFICE I2	E01
J22	BRANCH OFFICE J2	E01

若要取得管理員編號並非 NULL 值的列，可以變更 WHERE 子句如下：

```
WHERE MGRNO IS NOT NULL
```

如需使用 NULL 值的詳細資訊，請參閱 SQL Reference 一書。

SQL 陳述式中的特別暫存區

您可以在 SQL 陳述式中指定某些「特別暫存區」。在本端執行的 SQL 陳述式，其特別暫存區和內容如以下表格所示：

特別暫存區	內容
CURRENT DATE CURRENT_DATE	本日。
CURRENT PATH CURRENT_PATH CURRENT FUNCTION PATH	用於解析動態準備的 SQL 陳述式中之未限定資料類型名稱、程序名稱及函數名稱的 SQL 路徑。
CURRENT SCHEMA	綱目名稱，用於限定在動態準備的 SQL 陳述式中適用之未限定資料庫物件參照。
CURRENT SERVER CURRENT_SERVER	目前使用的關聯式資料庫名稱。
CURRENT TIME CURRENT_TIME	目前的時間。

特別暫存區	內容
CURRENT TIMESTAMP CURRENT_TIMESTAMP	時間戳記格式的目前日期和時間。
CURRENT TIMEZONE CURRENT_TIMEZONE	鏈結區域時間和世界標準時間 (UTC) 的持續時間，使用如下公式： 區域時間 - CURRENT TIMEZONE = UTC 這是取自於系統值 QUTCOFFSET。
USER	工作的執行時間授權 ID (使用者設定檔)。

如果單一陳述式包含對任何 CURRENT DATE、CURRENT TIME 或 CURRENT TIMESTAMP 特別暫存區，或 CURDATE、CURTIME 或 NOW 純量函數的多個參照，所有的值都會以單一時鐘讀數爲基礎。

在*遠端*執行的 SQL 陳述式，其特別暫存區和內容如以下列表格所示：

特別暫存區	內容
CURRENT DATE CURRENT_DATE CURRENT TIME CURRENT_TIME CURRENT TIMESTAMP CURRENT_TIMESTAMP	遠端系統 (不是本端系統) 的目前時間和日期。
CURRENT TIMEZONE CURRENT_TIMEZONE	鏈結遠端系統時間至 UTC 的持續時間。
CURRENT SERVER CURRENT_SERVER	目前使用的關聯式資料庫名稱。
CURRENT SCHEMA	遠端系統上目前的綱目值。
USER	遠端系統上之伺服器工作的執行時間授權 ID。
CURRENT PATH CURRENT_PATH CURRENT FUNCTION PATH	遠端系統的目前路徑值。

對分散式表格的查詢若是參照到特別暫存區，則會使用要求查詢系統上特別暫存區的內容。如需分散式表格的詳細資訊，請參閱 DB2 Multisystem 一書。

強制轉型資料類型

有時您會遇到一些狀況，必須將資料類型的*類型*強制轉型 (或變更) 爲另一種資料類型，或是相同的資料類型，但是長度、精準度或小數位數不同。例如，您若是想要比較兩個不同類型的直欄 (如字元和整數)，可以將字元變更爲整數，或將整數變更爲字元，使其能夠比較。能夠變更爲另一個資料類型的資料類型可從來源資料類型*強制轉型*爲目標資料類型。

您可以使用強制轉型函數或 CAST 規格，明確地將某種資料類型強制轉型爲另一種資料類型。例如，您若是有一欄定義爲 DATE 的日期 (BIRTHDATE)，而您想將直欄資料類型強制轉型爲具有固定長度 10 的 CHARACTER，可以輸入如下的：

```
SELECT CHAR (BIRTHDATE,USA)
   FROM CORPDATA.EMPLOYEE
```

也可以使用 CAST 函數，直接強制轉型資料類型。

```
SELECT CAST(BIRTHDATE AS CHAR(10))
   FROM CORPDATA.EMPLOYEE
```

如需強制轉型資料類型的更多詳細資訊，請參閱 SQL Reference 主題中的 Casting between data types。

日期、時間和時間戳記資料類型

日期、時間和時間戳記是以 SQL 使用者看不到之內部格式表示的資料類型。日期、時間和時間戳記可以字元字串值代表，並指定至字元字串變數。資料庫管理程式會將以下各種值辨識爲日期、時間和時間戳記值：

- DATE、TIME 或 TIMESTAMP 純量函數傳回的值。
- CURRENT DATE、CURRENT TIME 或 CURRENT TIMESTAMP 特別暫存區傳回的值。
- 一個字元字串，這個字串是算術運算或比較的一個運算元，*而且*另一個運算元是日期、時間或時間戳記時。例如，在述語：

  ```
  ... WHERE HIREDATE < '1950-01-01'
  ```

  中，如果 HIREDATE 是日期直欄，字元字串 '1950-01-01' 會解譯爲日期。
- 一個字元字串變數或常數，用於 UPDATE 陳述式的 SET 子句中，或 INSERT 陳述式的 VALUES 子句中，設定日期、時間或時間戳記直欄。

如需日期、時間和時間戳記值之字元字串格式的詳細資訊，請參閱 SQL Reference 一書中的 Datetime Values。

另請參閱下列主題：

- 『指定目前日期和時間值』
- 第 71 頁的『日期 / 時間運算』

指定目前日期和時間值

您可在表示式中指定以下三種特別暫存區之一，來指定目前的日期、時間或時間戳記：CURRENT DATE、CURRENT TIME 或 CURRENT TIMESTAMP。每一種的值是基於陳述式執行時取得的日期時間時鐘讀數。同一個 SQL 陳述式中對 CURRENT DATE、CURRENT TIME 或 CURRENT TIMESTAMP 的多重參照會使用相同的值。下列陳述式會傳回 EMPLOYEE 表格中每一名員工在陳述式執行時的年齡 (以年數表示)：

```
SELECT YEAR(CURRENT DATE - BIRTHDATE)
  FROM CORPDATA.EMPLOYEE
```

CURRENT TIMEZONE 特別暫存區允許將區域時間轉換爲世界標準時間 (UTC)。例如，您若是有一個名爲 DATETIME 的表格，包含名爲 STARTT 的時間直欄類型，而您要將 STARTT 轉換爲 UTC，可以使用以下陳述式：

```
SELECT STARTT - CURRENT TIMEZONE
  FROM DATETIME
```

日期 / 時間運算

日期、時間和時間戳記值只能使用加和減算術運算子。您可以使用持續時間增量或減量日期、時間或時間戳記；或者從日期減掉日期、從時間減掉時間，或從時間戳記減掉時間戳記。如需日期和時間計算的詳細說明，請參閱 SQL Reference 一書中的 Datetime arithmetic。

防止重複的列

SQL 評估 SELECT 陳述式時，根據滿足 SELECT 陳述式之搜尋條件的列數，可能有許多列符合放入結果表格中的資格。結果表格中有些列可能會重複。您可以使用 DISTINCT 關鍵字，後面加上直欄名稱的清單，指定不希望有任何重複：

```
SELECT DISTINCT JOB, SEX
...
```

DISTINCT 表示您只要選取唯一的列。如果選取的列和結果表格中另一個列重複，則會忽略重複的列 (不會放入結果表格中)。例如，假設您想要一份員工工作碼的清單。您不必知道哪個員工的工作碼是什麼，因爲一個部門中可能有多名人員擁有相同的工作碼，您可以使用 DISTINCT 確保結果表格中只有唯一的值。

以下範例顯示執行方式：

```
SELECT DISTINCT JOB
    FROM CORPDATA.EMPLOYEE
    WHERE WORKDEPT = 'D11'
```

結果有兩列：

JOB
DESIGNER
MANAGER

如果您未在 SELECT 子句中包含 DISTINCT，可能會在結果中發現重複的列，因爲 SQL 會傳回滿足搜尋條件之每一列的 *JOB* 直欄值。 DISTINCT 會將 NULL 值當成重複的列處理。

如果您在 SELECT 子句中包含 DISTINCT，而且也包含了共用權重排序順序，會傳回較少的值。排序順序會使包含相同字元的值計算成相同的加權。如果 'MGR'、'Mgr' 或 'mgr' 都在同一個表格中，只會傳回其中一個值。如需排序順序和選項的詳細資訊，請參閱第 107 頁的第 8 章, 『SQL 的排序順序』。

執行複雜搜尋條件

除了基本的比較述語 (=、>、< 等) 之外，搜尋條件還可以包含 BETWEEN、IN、EXISTS、IS NULL 及 LIKE 等任何關鍵字。搜尋條件還可以包含子查詢。如需詳細資訊和範例，請參閱第 97 頁的第 7 章, 『使用子查詢』。

若是字元和 UCS-2 圖形直欄述語，在評估 BETWEEN、IN、EXISTS 及 LIKE 子句的述語之前，會先在運算元套用排序順序。如需配合選擇使用排序順序的詳細資訊，請參閱第 107 頁的第 8 章, 『SQL 的排序順序』。

您也可以執行多重搜尋條件。詳細資訊請參閱第 73 頁的『WHERE 子句中的多重搜尋條件』。

- **BETWEEN ... AND ...** 用於指定一種搜尋條件，任何值只要等於另外兩個值或介於這兩個值之間，就滿足此搜尋條件。例如，若要尋找在 1987 年雇用的所有員工，可以使用：

  ```
  ...  WHERE HIREDATE BETWEEN '1987-01-01' AND '1987-12-31'
  ```

  BETWEEN 關鍵字屬於包含性。以下的搜尋條件較複雜，但是很明確，會產生相同的結果：

  ```
  ... WHERE HIREDATE >= '1987-01-01' AND HIREDATE <= '1987-12-31'
  ```

- **IN** 表示您想要的列，其中指定之表示式的值位於您所列的值之中。例如，若要尋找部門 A00、C01 及 E21 中所有員工的姓名，可以指定：

  ```
  ... WHERE WORKDEPT IN ('A00', 'C01', 'E21')
  ```

- **EXISTS** 表示您要測試某些列是否存在。例如，若要查明是否有任何員工的薪資大於 60000，可以指定：

  ```
  EXISTS (SELECT * FROM EMPLOYEE WHERE SALARY > 60000)
  ```

- **IS NULL** 表示您要測試 NULL 值。例如，若要查明是否有任何員工沒有電話報表，可以指定：

  ```
  ... WHERE EMPLOYEE.PHONE IS NULL
  ```

- **LIKE** 表示您想要的列，其中某個欄位值類似您提供的值。您使用 LIKE 時，SQL 會搜尋與您指定者類似的字元字串。類似程度由您在搜尋條件中包含之字串裡使用的兩個特殊字元決定：

  _ 底線字元代表任何單一字元。

  % 百分比符號代表 0 或多個字元的不明字串。如果搜尋字串以百分比符號開頭，SQL 允許直欄中的符合值前面有 0 或多個字元。否則，搜尋字串必須從直欄的第一個位置開始。

  註: 如果是操作 MIXED 資料，適用以下區別：SBCS 底線字元代表一個 SBCS 字元。百分比符號則沒有這種限制；也就是，百分比符號可以代表任意數目的 SBCS 或 DBCS 字元。如需 LIKE 述語和 MIXED 資料的詳細資訊，請參閱 SQL Reference 一書。

  如果您不知道，或不在乎直欄值的所有字元，可以使用底線字元或百分比符號。例如，若要查明哪些員工居住在 Minneapolis，可以指定：

  ```
  ... WHERE ADDRESS LIKE '%MINNEAPOLIS%'
  ```

  SQL 會傳回 ADDRESS 直欄中有 MINNEAPOLIS 字串的任何列，至於字串的位置並不重要。

  在另一個範例中，若要列出名稱以 'SAN' 開頭的市鎮，可以指定：

  ```
  ... WHERE TOWN LIKE 'SAN%'
  ```

  如果想要尋找的街道名稱不在您主要街道名稱清單中的任何地址，可以在 LIKE 表示式中使用一個表示式。在此範例中，假設表格中的 STREET 直欄都是大寫字體。

  ```
  ... WHERE UCASE (:address_variable) NOT LIKE '%'||STREET||'%'
  ```

如果要搜尋包含底線或百分比字元的字元字串，請使用 ESCAPE 子句指定跳出字元。例如，若要查看名稱中有百分比的所有企業，可以指定：

```
... WHERE BUSINESS_NAME LIKE '%@%%' ESCAPE '@'
```

第一個和最後一個百分比字元會如常解譯。'@%' 組合則會當成實際的百分比字元。如需更多明細，請參閱『LIKE 的特殊注意事項』。

如需述語的完整列表，請參閱 SQL Reference 主題中的 Predicates。

LIKE 的特殊注意事項

- 使用主變數代替搜尋型樣中的字串常數時，應該考慮使用可變長度的主變數。這可讓您：
  - 將先前使用的字串常數原封不動地指派至主變數。
  - 取得與使用字串常數相同的選取準則和結果。
- 在搜尋型樣中使用固定長度主變數代替字串常數時，應該確定主變數中指定的值與字串常數先前使用的型樣相符。主變數中未指派值的所有字元都會以空白開頭。

  例如，您若是使用***可變長度***主變數中的字串型樣 'ABC%' 執行搜尋，以下是可能傳回的一些值：

  ```
  'ABCD      ' 'ABCDE'    'ABCxxx'      'ABC '
  ```

  但是，您若是使用*固定長度*爲 10 的主變數中包含的搜尋型樣 'ABC%' 執行搜尋，假設直欄的長度爲 12，以下爲可能傳回的一些值：

  ```
  'ABCDE       ' 'ABCD        ' 'ABCxxx      ' 'ABC         '
  ```

  請注意，傳回的值全都以 'ABC' 開頭，後面至少有六個空白。這是因爲主變數中最後六個字元並未指定特定的值，因此才使用空白。

  如果您要使用最後 7 個字元可爲任何內容的固定長度之主電腦執行搜尋，可搜尋 'ABC%%%%%%%'。以下爲可能傳回的一些值：

  ```
  'ABCDEFGHIJ'   'ABCXXXXXXX'  'ABCDE'      'ABCDD'
  ```

WHERE 子句中的多重搜尋條件

在第 60 頁的『使用 WHERE 子句 指定搜尋條件』中，您看到如何使用一個搜尋條件。您可以撰寫包含許多述語的搜尋條件，進一步定義您的要求。您指定的搜尋條件可以包含任何的比較運算子或 BETWEEN、IN、 LIKE、EXISTS、IS NULL 及 IS NOT NULL。

您可以使用連接詞 AND 和 OR 結合任兩個述語。此外，也可以使用 NOT 關鍵字，指定所要的搜尋條件爲指定之搜尋條件的負值。 WHERE 子句可以有任意多個述語。

- **AND** 表示，某一列必須同時滿足搜尋條件的兩個述語，才算是合格的列。例如，若要查明部門 D21 中有哪些員工是在 1987 年 12 月 31 日以後雇用，可以指定：

  ```
  ...
    WHERE WORKDEPT = 'D21' AND HIREDATE > '1987-12-31'
  ```

- **OR** 表示，某個列只要能滿足搜尋條件中任一個或兩個述語設定的條件，就是合格的列。例如，若要查明哪些員工是在部門 C01 或 D11，可以指定：

  ```
  ...
    WHERE WORKDEPT = 'C01' OR WORKDEPT = 'D11'
  ```

註: 也可以使用 IN 指定此一要求：WHERE WORKDEPT IN ('C01', 'D11')。

- **NOT** 表示，某個列必須不能夠滿足 NOT 之後的搜尋條件或述語設定的基準，才算合格。例如，若要找出部門 E11 中工作碼不等於分析師的所有員工，可以指定：

```
...
  WHERE WORKDEPT = 'E11' AND NOT JOB = 'ANALYST'
```

SQL 評估包含這些連接詞的搜尋條件時，會以特定的次序評估。 SQL 先評估 NOT 子句，接著評估 AND 子句，然後才是 OR 子句。

您可以使用括弧變更評估次序。會先評估使用括弧括住的搜尋條件。例如，若要選取部門 E11 和 E21 中教育程度高於 12 的所有員工，可以指定：

```
...
  WHERE EDLEVEL > 12 AND
     (WORKDEPT = 'E11' OR WORKDEPT = 'E21')
```

括弧決定搜尋條件的意義。在此範例中，您想要所有列有：

E11 或 E21 的 WORKDEPT 值，及

EDLEVEL 值大於 12

如果未使用括弧：

```
...
  WHERE EDLEVEL > 12 AND WORKDEPT = 'E11'
    OR WORKDEPT = 'E21'
```

結果將會不同。所選取的爲有以下條件的列：

WORKDEPT = E11 和 EDLEVEL > 12，或

WORKDEPT = E21，不論 EDLEVEL 值爲何

結合多個表格的資料

有時候，您要查看的資訊不只放在單一表格中。若要形成結果表格的列，可能要從某個表格擷取一些欄位值，並從另一個表格擷取一些欄位值。您可以擷取兩個以上表格的欄位值，結合到單一的列中。

DB2 UDB for iSeries 支援許多不同類型的結合：內部結合、外部結合、右外部結合、左異常結合、右異常結合及交叉結合。

- 第 75 頁的『內部結合』僅傳回每一個表格內結合直欄中有符合值的列。結果表格中不會顯示在表格間沒有相符結果的列。
- 第 76 頁的『左外部結合』會傳回第一個表格 (左邊的表格) 中所有的列之值，以及第二個表格中相符列的值。第二個表格中沒有符合值的列會傳回第二個表格所有直欄的 NULL 值。
- 第 76 頁的『右外部結合』會傳回第二個表格 (右邊的表格) 中所有列的值，以及第一個表格中符合列的值。第一個表格中沒有符合值的任何列會傳回第一個表格所有直欄的 NULL 值。
- 「左異常結合」只會傳回左邊表格中與右邊表格不相符的列。結果表格中取自右邊表格的直欄爲 NULL 值。
- 「右異常結合」只會傳回右邊表格中與左邊表格不相符的列。結果表格中取自左邊表格的直欄爲 NULL 值。

- 第 77 頁的『交叉結合』會將結合之兩表格 (Cartesian Product) 的每一個列組合傳回成爲結果表格中的一列。

您可以使用「左外部結合」和「右異常結合」模擬「完全外部結合」。詳細內容請參閱第 78 頁的『模擬完全外部結合』。此外，也可以在單一陳述式中使用多重結合類型。詳細內容請參閱第 78 頁的『一個陳述式中有多重結合類型』。

結合的附註

當您結合兩個以上表格時：

- 如果有共用的直欄名稱，必須使用每一個表格的名稱 (或相關名稱) 限定每一個直欄名稱。唯一的直欄名稱則不必限定。
- 如果未列出您要的直欄名稱，而是使用 SELECT *，SQL 傳回的列會包含第一個表格中所有的直欄，後接第二個表格的所有直欄，依此類推。
- 您必須擁有授權，才能選取 FROM 子句中指定的每一個表格或概略表之列。
- 排序順序會套用至要結合之所有字元和 UCS-2 圖形直欄。

內部結合

使用內部結合時，會將表格中某列的欄位值和另一個 (或同一個) 表格中另一列的欄位值結合，形成一列資料。SQL 會檢查指定要結合的兩個表格，從滿足結合搜尋條件的所有列中擷取資料。有兩種方法可以指定內部結合：使用 JOIN 語法，以及使用 WHERE 子句。

假設您要擷取負責專案之所有員工的員工編號、姓名及專案編號。換句話說，您想要 CORPDATA.EMPLOYEE 表格中的 *EMPNO* 和 *LASTNAME* 直欄，以及 CORPDATA.PROJECT 表格中的 *PROJNO* 直欄。只需考慮姓氏以 'S' 以後之字母開頭的員工。爲了尋找此一資訊，您必須結合兩個表格。

使用 JOIN 語法的內部結合

要使用內部結合語法，FROM 子句中會列出您要結合的兩個表格，以及套用至表格的結合條件。結合條件指定於 ON 關鍵字之後，會決定兩個表格應如何互相比較，以產生結合結果。條件可以是任何比較運算子，不一定要是等於運算子。 ON 子句中可以指定多個結合條件，使用 AND 關鍵字隔開。與實際結合無關的其他任何條件是指定於 WHERE 子句中，或者在 ON 子句中指定爲實際結合的一部分。

```
SELECT EMPNO, LASTNAME, PROJNO
  FROM CORPDATA.EMPLOYEE INNER JOIN CORPDATA.PROJECT
       ON EMPNO = RESPEMP
  WHERE LASTNAME > 'S'
```

在此範例中，使用兩個表格的 EMPNO 和 RESPEMP 直欄結合兩個表格。由於只要傳回姓氏以 'S' 之後字母開頭的員工，因此是在 WHERE 子句中提供此附加條件。

這個查詢會傳回下列輸出：

EMPNO	LASTNAME	PROJNO
000250	SMITH	AD3112
000060	STERN	MA2110
000100	SPENSER	OP2010
000020	THOMPSON	PL2100

使用 WHERE 子句的內部結合

使用 WHERE 子句執行相同的結合時，是在 WHERE 子句中同時撰寫結合條件和附加選擇條件。要結合的表格會列在 FROM 子句中，以逗點隔開。

```
SELECT EMPNO, LASTNAME, PROJNO
  FROM CORPDATA.EMPLOYEE, CORPDATA.PROJECT
  WHERE EMPNO = RESPEMP
  AND LASTNAME > 'S'
```

這個查詢會傳回與前一範例相同的輸出：

左外部結合

左外部結合會傳回內部結合傳回的所有列，再加上第一個表格中與第二個表格不符合的其他每一列。

假設您要尋找所有員工及其目前負責的所有專案。您也想要查看目前沒有負責專案的員工。以下查詢會傳回名字大於 'S' 之所有員工，以及其分派之專案編號的清單。

```
SELECT EMPNO, LASTNAME, PROJNO
  FROM CORPDATA.EMPLOYEE LEFT OUTER JOIN CORPDATA.PROJECT
       ON EMPNO = RESPEMP
  WHERE LASTNAME > 'S'
```

這個查詢的結果包含一些沒有專案編號的員工。這些員工會列在查詢中，但是傳回的專案編號是 NULL 值。

EMPNO	LASTNAME	PROJNO
000020	THOMPSON	PL2100
000060	STERN	MA2110
000100	SPENSER	OP2010
000170	YOSHIMURA	-
000180	SCOUTTEN	-
000190	WALKER	-
000250	SMITH	AD3112
000280	SCHNEIDER	-
000300	SMITH	-
000310	SETRIGHT	-
200170	YAMAMOTO	-
200280	SCHWARTZ	-
200310	SPRINGER	-
200330	WONG	-

附註

使用 RRN 純量函數傳回左外部結合或異常結合中右邊表格內某直欄的相對記錄號碼，不相符的列將會傳回 0 值。

右外部結合

右外部結合會傳回內部結合傳回的所有列，再加上第二個表格中與第一個表格不符合的其他每一列。這種結合和外部結合相同，不過指定的表格次序剛好相反。

可以將左外部結合範例使用的查詢寫成如下的右外部結合：

```
SELECT EMPNO, LASTNAME, PROJNO
  FROM CORPDATA.PROJECT RIGHT OUTER JOIN CORPDATA.EMPLOYEE
       ON EMPNO = RESPEMP
  WHERE LASTNAME > 'S'
```

這個查詢的結果和左外部結合查詢的結果相同。

異常結合

左異常結合只會傳回第一個表格中與第二個表格不相符的列。使用與前面相同的表格，傳回未負責任何專案的員工。

```
SELECT EMPNO, LASTNAME, PROJNO
  FROM CORPDATA.EMPLOYEE EXCEPTION JOIN CORPDATA.PROJECT
       ON EMPNO = RESPEMP
  WHERE LASTNAME > 'S'
```

這個結合會傳回以下輸出：

EMPNO	LASTNAME	PROJNO
000170	YOSHIMURA	-
000180	SCOUTTEN	-
000190	WALKER	-
000280	SCHNEIDER	-
000300	SMITH	-
000310	SETRIGHT	-
200170	YAMAMOTO	-
200280	SCHWARTZ	-
200310	SPRINGER	-
200330	WONG	-

也可以使用 NOT EXSITS 述語將異常結合寫成子查詢。前一查詢可以下列方法重寫：

```
SELECT EMPNO, LASTNAME
  FROM CORPDATA.EMPLOYEE
  WHERE LASTNAME > 'S'
    AND NOT EXISTS
       (SELECT * FROM CORPDATA.PROJECT
          WHERE EMPNO = RESPEMP)
```

這個查詢中唯一的差異是無法傳回 PROJECT 表格的值。

也有一個右異常結合，作用與左異常結合相同，不過表格剛好相反。

交叉結合

交叉結合 (或 Cartesian Product 結合) 傳回的結果表格中，會將第一個表格中每一列與第二個表格中每一列結合。結果表格中的列數等於兩邊表格的列數相加。如果相關表格很大，這種結合會花很長的時間。

交叉結合可以兩種方法指定：使用 JOIN 語法；或在 FROM 子句中列出表格，以逗點隔開，不使用 WHERE 子句提供結合基準。

假設有下列兩個表格。

表 6. 表格 A

ACOL1	ACOL2
A1	AA1
A2	AA2
A3	AA3

表 7. 表格 B

BCOL1	BCOL2
B1	BB1
B2	BB2

下列兩個 SELECT 陳述式會產生相同的結果。

```
SELECT * FROM A CROSS JOIN B
SELECT * FROM A, B
```

任一個 SELECT 陳述式的結果表格就像這樣：

ACOL1	ACOL2	BCOL1	BCOL2
A1	AA1	B1	BB1
A1	AA1	B2	BB2
A2	AA2	B1	BB1
A2	AA2	B2	BB2
A3	AA3	B1	BB1
A3	AA3	B2	BB2

模擬完全外部結合

完全外部結合和左與右外部結合一樣，會傳回兩個表格中符合的列。但是，完全符合外部結合也會傳回左和右兩個表格中不符合的列。DB2 UDB for iSeries 並不支援完全外部結合語法，但是您可以使用左外部結合和右異常結合模擬完全外部結合。假設您要尋找所有員工和所有專案，也想要查看目前沒有負責專案的員工。以下查詢會傳回姓名大於 'S' 的所有員工，以及其分派之專案編號的清單。

```
SELECT EMPNO, LASTNAME, PROJNO
     FROM CORPDATA.EMPLOYEE LEFT OUTER JOIN CORPDATA.PROJECT
           ON EMPNO = RESPEMP
     WHERE LASTNAME > 'S'
  UNION
  (SELECT EMPNO, LASTNAME, PROJNO
     FROM CORPDATA.PROJECT EXCEPTION JOIN CORPDATA.EMPLOYEE
           ON EMPNO = RESPEMP
     WHERE LASTNAME > 'S');
```

一個陳述式中有多重結合類型

有時候必須結合兩個以上的表格，以產生所要的結果。如果您要傳回所有員工、員工的部門名稱及其負責的專案 (如果有負責)，則 EMPLOYEE 表格、 DEPARTMENT 表格及 PROJECT 表格全都必須結合，才能得到資訊。以下範例顯示查詢和結果。

```
SELECT EMPNO, LASTNAME, DEPTNAME, PROJNO
  FROM CORPDATA.EMPLOYEE INNER JOIN CORPDATA.DEPARTMENT
       ON WORKDEPT = DEPTNO
     LEFT OUTER JOIN CORPDATA.PROJECT
       ON EMPNO = RESPEMP
  WHERE LASTNAME > 'S'
```

查詢的結果如下：

EMPNO	LASTNAME	DEPTNAME	PROJNO
000020	THOMPSON	PLANNING	PL2100
000060	STERN	MANUFACTURING SYSTEMS	MA2110
000100	SPENSER	SOFTWARE SUPPORT	OP2010
000170	YOSHIMURA	MANUFACTURING SYSTEMS	-
000180	SCOUTTEN	MANUFACTURING SYSTEMS	-
000190	WALKER	MANUFACTURING SYSTEMS	-
000250	SMITH	ADMINISTRATION SYSTEMS	AD3112
000280	SCHNEIDER	OPERATIONS	-
000300	SMITH	OPERATIONS	-
000310	SETRIGHT	OPERATIONS	-

如需結合的詳細資訊，請參閱 SQL Reference 一書。

使用表格表示式

您可以使用表格表示式指定中階結果表格。表格表示式可用來代替概略表，以避免在不需要一般用途概略表時建立概略表。表格表示式包含巢狀表格表示式 (也稱爲衍生表格) 和共用表格表示式。

巢狀表格表示式指定於 FROM 子句的括弧中。例如，假設您想要會顯示管理員編號、部門編號及每一部門最高薪資的結果表格。管理員編號在 DEPARTMENT 表格中、部門編號在 DEPARTMENT 和 EMPLOYEE 表格中、薪資在 EMPLOYEE 表格中。您可以在 FROM 子句中使用表格表示式選取每一部門的最高薪資。也可以在巢狀表格表示式之後加入相關名稱 T2 ，爲衍生表格命名。接著外部選取使用 T2 限定從衍生表格中選取的直欄，在此例中爲 MAXSAL 和 WORKDEPT。請注意，在巢狀表格表示式中選取的 MAX(SALARY) 直欄必須命名，才能在外部選取中參照。這個利用 AS 子句執行。

```
SELECT MGRNO, T1.DEPTNO, MAXSAL
FROM CORPDATA.DEPARTMENT T1,
      (SELECT MAX(SALARY) AS MAXSAL, WORKDEPT
       FROM CORPDATA.EMPLOYEE E1
       GROUP BY WORKDEPT) T2
WHERE T1.DEPTNO = T2.WORKDEPT
ORDER BY DEPTNO
```

查詢的結果如下：

MGRNO	DEPTNO	MAXSAL
000010	A00	52750.00
000020	B01	41250.00

MGRNO	DEPTNO	MAXSAL
000030	C01	38250.00
000060	D11	32250.00
000070	D21	36170.00
000050	E01	40175.00
000090	E11	29750.00
000100	E21	26150.00

可以在 SELECT 陳述式、INSERT 陳述式或 CREATE VIEW 陳述式中的全選之前指定共用表格表示式。如果必須在全選中共用同一個結果表格，就可以使用。共用表格表示式前接關鍵字 WITH。

例如，假設您想要一份表格，顯示某一組部門的最低和最高平均薪資。部門編號的第一個字元有其意義，而您想要取得以字母 'D' 開頭和以字母 'E' 開頭之部門的最小和最大值。您可以使用共用表格表示式選取每一部門的平均薪資。同樣地，您必須爲衍生表格命名：在此例中，其名稱爲 DT。接著您可以使用 WHERE 子句指定 SELECT 陳述式，限制只選取以特定字母開頭的部門。指定衍生表格 DT 中之 AVGSAL 直欄的最小和最大值。指定 UNION 以取得字母 'E' 的結果和字母 'D' 的結果。

```
WITH DT AS (SELECT E.WORKDEPT AS DEPTNO, AVG(SALARY) AS AVGSAL
            FROM CORPDATA.DEPARTMENT D , CORPDATA.EMPLOYEE E
            WHERE D.DEPTNO = E.WORKDEPT
            GROUP BY E.WORKDEPT)
 SELECT 'E', MAX(AVGSAL), MIN(AVGSAL) FROM DT
 WHERE DEPTNO LIKE 'E%'
 UNION
 SELECT 'D', MAX(AVGSAL), MIN(AVGSAL) FROM DT
 WHERE DEPTNO LIKE 'D%'
```

查詢的結果如下：

	MAX(AVGSAL)	MIN(AVGSAL)
E	40175.00	21020.00
D	25668.57	25147.27

假設您要針對訂購資料庫撰寫查詢，傳回訂購了 'XXX' 項目之客戶最後 1000 張訂單中的前 5 大項目 (訂購總數)。

```
WITH X AS (SELECT ORDER_ID, CUST_ID
                  FROM ORDERS
                  ORDER BY ORD_DATE DESC
                  FETCH FIRST 1000 ROWS ONLY),
     Y AS (SELECT CUST_ID, LINE_ID, ORDER_QTY
                  FROM X, ORDERLINE
                  WHERE X.ORDER_ID = ORDERLINE.ORDER_ID)
SELECT LINE_ID
     FROM (SELECT LINE_ID
                  FROM Y
                  WHERE Y.CUST_ID IN (SELECT DISTINCT CUST_ID
                                             FROM Y
                                             WHERE LINE.ID = 'XXX' )
                  GROUP BY LINE_ID
                  ORDER BY SUM(ORDER_QTY) DESC)
   FETCH FIRST 5 ROWS ONLY
```

第一個共用表格表示式 (X) 會傳回最新的 1000 個訂單編號。結果依據日期以降序排序，然後只傳回這些訂購列的前 1000 列做爲結果表格。

第二個共用表格表示式 (Y) 將最新的 1000 張訂單和行項目表格結合，傳回 (1000 張訂單的每一張) 客戶、行項目及該訂單的行項目數量。

主選取陳述式中的衍生表格會傳回客戶的行項目，這些客戶爲在於 1000 大訂單中訂購 XXX 項目者。接著再依據行項目將訂購 XXX 的所有客戶分組，然後依據行項目的總數排序群組。

最後，外部選取只會從衍生表格傳回的訂購清單中選取前 5 列。

使用 UNION 關鍵字結合子選擇

您可以使用 UNION 關鍵字結合兩個以上的子選擇，以形成全選。 SQL 遇到 UNION 關鍵字時，會處理每一個子選擇以形成中間結果表格，然後結合每一個子選擇的中間結果表格，再刪除重複的列以形成結合的結果表格。您使用 UNION 合併兩個以上表格的值清單。撰寫選擇陳述式時，可以使用您到目前爲止所學到的任何子句和技術。

您在合併取自許多表格的值清單時，可以使用 UNION 消除重複。例如，可以取得包含以下條件之員工編號的結合清單：

- 部門 D01 中的人員
- 其分派包含專案 MA2112、MA2113 及 AD3111 的人員

結合清單衍生自兩個表格，沒有包含重複。其方式爲指定：

```
SELECT EMPNO
   FROM CORPDATA.EMPLOYEE
   WHERE WORKDEPT = 'D11'
 UNION
 SELECT EMPNO
   FROM CORPDATA.EMPPROJACT
   WHERE PROJNO = 'MA2112' OR
      PROJNO = 'MA2113' OR
      PROJNO = 'AD3111'
 ORDER BY EMPNO
```

您也可以在共用表格表示式、巢狀表格表示式中，或在建立概略表時使用 UNION。詳細內容請參閱第 54 頁的『使用 UNION 建立概略表』。

若要更瞭解這些 SQL 陳述式的結果，可以想像 SQL 執行以下處理：

步驟 1，SQL 處理第一個 SELECT 陳述式：

```
SELECT EMPNO
     FROM CORPDATA.EMPLOYEE
     WHERE WORKDEPT = 'D11'
```

這會產生一個暫時的結果表格：

CORPDATA.EMPLOYEE 的 EMPNO
000060
000150
000160

CORPDATA.EMPLOYEE 的 EMPNO
000170
000180
000190
000200
000210
000220
200170
200220

步驟 2，SQL 處理第二個 SELECT 陳述式：

```
SELECT EMPNO
     FROM CORPDATA.EMPPROJACT
     WHERE PROJNO='MA2112' OR
                 PROJNO= 'MA2113' OR
                 PROJNO= 'AD3111'
```

這會產生另一個暫時的結果表格：

CORPDATA.EMPPROJACT 的 EMPNO
000230
000230
000240
000230
000230
000240
000230
000150
000170
000190
000170
000190
000150
000160
000180
000170
000210
000210

步驟 3，SQL 結合兩個暫時的結果表格，移除重複的列，再將結果排序：

```
SELECT EMPNO
     FROM CORPDATA.EMPLOYEE
     WHERE WORKDEPT = 'D11'
UNION
SELECT EMPNO
     FROM CORPDATA.EMPPROJACT
```

```
    WHERE PROJNO='MA2112' OR
                PROJNO= 'MA2113' OR
                PROJNO= 'AD3111'
ORDER BY EMPNO
```

這會產生以升序順序排序值的結合結果表格：

EMPNO
000060
000150
000160
000170
000180
000190
000200
000210
000220
000230
000240
200170
200220

您使用 UNION 時：

- 屬於聯集一部分的最後一個子選擇後面必須要有 ORDER BY 子句。在此範例中，結果會根據第一個所選取直欄 *EMPNO* 排序。ORDER BY 子句指定結合的結果表格必須採用對照順序。概略表中不能使用 ORDER BY。
- 如果結果直欄有命名，ORDER BY 子句上可能會指定名稱。如果每一個聯集的選取陳述式中之直欄的名稱相同，結果直欄就會以此爲名稱。可以使用 AS 子句爲選取清單中的直欄指定名稱。

  ```
  SELECT A + B AS X ...
  UNION
  SELECT X ... ORDER BY X
  ```

  如果結果直欄未命名，請使用正整數將結果排序。編號代表您的子選擇中所包含表示式清單中的表示式位置。

  ```
  SELECT A + B ...
  UNION
  SELECT X ... ORDER BY 1
  ```

若要識別每一列的來源子選擇，可以在聯集中每一子選擇的選取清單尾端包含一個常數。SQL 傳回您的結果時，最後一個直欄會包含該列來源次選擇的常數。例如，您可以指定：

```
SELECT A, B, 'A1' ...
UNION
SELECT X, Y, 'B2'...
```

傳回列時，列中會包含一個值 (A1 或 B2)，指示該列之值的來源表格。如果聯集中的直欄名稱不同，SQL 會使用交談式 SQL 顯示或列印結果時指定於第一個子選擇中，或者處理 SQL DESCRIBE 陳述式所產生之 SQLDA 中指定的直欄名稱組。

如需 UNION 中直欄長度與資料類型相容性的資訊，請參閱 SQL Reference 一書中的 Rules for result data type 主題。

註: 使 UNION 兩端的欄位相容之後，就會套用排序順序。在 UNION 處理程序期間隱含發生的不同處理程序會使用排序順序。如需排序順序的詳細內容，請參閱第 107 頁的第 8 章, 『SQL 的排序順序』。

指定 UNION ALL

如果您要在 UNION 結果中保留重複內容，請指定 UNION ALL 代替單獨使用 UNION。使用與 UNION 相同的步驟和範例：

步驟 3，SQL 會結合兩個暫時的結果表格：

```
SELECT EMPNO
     FROM CORPDATA.EMPLOYEE
     WHERE WORKDEPT = 'D11'
 UNION ALL
SELECT EMPNO
     FROM CORPDATA.EMPPROJACT
     WHERE PROJNO='MA2112' OR
                 PROJNO= 'MA2113' OR
                 PROJNO= 'AD3111'
   ORDER BY EMPNO
```

會產生一個包含重複內容的已排序結果表格：

EMPNO
000060
000150
000150
000150
000160
000160
000170
000170
000170
000170
000180
000180
000190
000190
000190
000200
000210
000210
000210
000220
000230
000230

EMPNO
000230
000230
000230
000240
000240
200170
200220

UNION ALL 運算爲聯合的，例如：

```
(SELECT PROJNO FROM CORPDATA.PROJECT
UNION ALL
SELECT PROJNO FROM CORPDATA.PROJECT)
UNION ALL
SELECT PROJNO FROM CORPDATA.EMPPROJACT
```

這個陳述式也可以寫成：

```
SELECT PROJNO FROM CORPDATA.PROJECT
UNION ALL
(SELECT PROJNO FROM CORPDATA.PROJECT
UNION ALL
SELECT PROJNO FROM CORPDATA.EMPPROJACT)
```

但是，若在與 UNION 運算子相同的 SQL 陳述式中包含 UNION ALL，運算結果依評估的次序而定。如果沒有括弧，會由左至右評估。如果有括弧，會先評估以括弧括住的子選擇，然後再由左至右評估陳述式的其他部分。

資料擷取錯誤

如果 SQL 發覺擷取的字元或圖形直欄太長，無法放進主變數中，SQL 會執行下列動作：

- 將值指派至主變數時會截斷資料。
- 將 SQLCA 中的 SQLWARN0 和 SQLWARN1 設定爲 'W' 值。
- 將指示器變數 (如果有提供) 設定爲值截斷之前的長度。

如果 SQL 在執行陳述式時發現資料對映錯誤，會發生下列兩種狀況之一：

- 如果是在 SELECT 清單中的表示式上發生錯誤，而且有錯誤表示式的指示器變數：
  - SQL 會爲與錯誤表示式對應的指示器變數傳回 -2。
  - SQL 會傳回該列的一切有效資料。
  - SQL 會傳回一個正 SQLCODE。
- 如果沒有指示器變數，SQL 會在 SQLCA 中傳回對應的負 SQLCODE。

資料對映錯誤包括：

- +138 - 子字串函數的引數無效。
- +180 - 代表日期、時間或時間戳記之字串的語法無效。
- +181 - 日期、時間或時間戳記的字串呈現不是有效值。

- +183 - 日期 / 時間表示式的結果無效。產生的日期或時間戳記不在有效的日期或時間戳記範圍內。
- +191 - MIXED 資料的格式不正確。
- +304 - 數字轉換錯誤 (例如溢位、下溢或除以零)。
- +331 - 無法轉換字元。
- +420 - CAST 引數中的字元無效。
- +802 - 資料轉換或資料對映錯誤。

若是資料對映錯誤，SQLCA 只會報告偵測到的最後一個錯誤。對應至發生錯誤之每一個結果直欄的指示器變數會設定爲 -2。

如果全選的選取清單中包含 DISTINCT，而且選取清單中的直欄包含無效的數值資料，則在完成排序查詢後，會將該資料視爲等於空值。如果使用現有的索引，則不會將資料視爲空值。

資料對映錯誤對 ORDER BY 子句的影響視狀況而定：

- 如果是在指派資料至 SELECT INTO 或 FETCH 陳述式中的主變數時發生資料對映錯誤，而且該表示式也用於 ORDER BY 子句中，則結果記錄會根據表式示的值排序。不會當成空值 (高於其他一切值) 排序。這是因爲會在嘗試指派給主變數之前，就先評估表示式。
- 如果是在評估選取清單中之表示式之前發生資料對映錯誤，而且同一個表示式也用於 ORDER BY 子句中，則結果直欄會當成如空值一樣 (高於其他一切值) 正常排序。如果是使用排序實作 ORDER BY 子句，結果直欄會當成如空值一樣排序。如果是使用現有的索引實作 ORDER BY 子句，在下列情況下，結果直欄會根據索引中表示式的實際值排序：
  - 表示式是日期格式爲 *MDY、*DMY、*YMD 或 *JUL 的日期直欄，並且因爲日期不是在有效的日期範圍內而發生日期轉換錯誤。
  - 表示式是字元直欄，而字元無法轉換。
  - 表示式是十進位直欄，且偵測到無效的數值。

第 6 章 SQL 插入、更新及刪除

本節顯示在表格和概略表中更新、刪除及插入資料的一些基本 SQL 陳述式和子句。使用的 SQL 陳述式爲 UPDATE、DELETE 及 INSERT。提供使用這些 SQL 陳述式的範例來協助您開發 SQL 應用程式。在 SQL Reference 一書中，提供了 SQL 陳述式的詳細語法及參數說明。

註:

1. 本節所述的 SQL 陳述式可在 SQL 表格和概略表，以及資料庫實體和邏輯檔案上執行。表格、概略表及檔案可位於網目或檔案庫中。
2. SQL 陳述式中指定的字串 (例如與 WHERE 或 VALUES 子句一起使用的那些字串) 須區分大小寫；亦即，大寫字元必須以大寫輸入，而小寫字元則必須以小寫輸入。

   ```
   WHERE ADMRDEPT='a00'      (不傳回結果)

   WHERE ADMRDEPT='A00'      (傳回有效的部門編碼)
   ```

   如果所使用的共用權重排序順序中的大寫和小寫字元被視爲相同字元，則字串不區分大小寫。有關排序順序的詳細資訊，請參閱第 107 頁的第 8 章, 『SQL 的排序順序』。

本章各節如下：

使用 INSERT 陳述式插入列

透過下列其中一個方式，您可以使用 INSERT 陳述式將列新增到表格或概略表中：

- 在 INSERT 陳述式中，對所要新增單一列的直欄指定一些值。
- 將 SELECT 陳述式併入 INSERT 陳述式中，告知 SQL 新列的哪些資料包含在另一表格或概略表中。第 89 頁的『使用 SELECT 陳述式將列插入表格中』說明如何使用 INSERT 陳述式內的 SELECT 陳述式來新增零、一或多列到表格中。
- 指定 INSERT 陳述式的區塊化形式來新增多列。 第 90 頁的『使用區塊化 INSERT 陳述式將多列插入表格中』說明如何使用 INSERT 陳述式的區塊化形式來新增多列到表格中。

註: 由於在表格上建立的概略表實際上不含資料，所以在使用概略表時可能產生混淆。有關使用概略表插入資料的詳細資訊與限制，請參閱第 53 頁的『建立及使用概略表』。

有關 INSERT 的完整說明，請參閱 SQL Reference 中的 INSERT 陳述式。

對於您所插入的每一列，如果直欄沒有預設值，您必須以 NOT NULL 屬性爲每一個定義的直欄提供一值。將列新增至表格或概略表的 INSERT 陳述式如下：

```
INSERT INTO table-name
   (column1, column2, ... )
VALUES (value-for-column1, value-for-column2, ... )
```

INTO 子句可爲您指定值的直欄命名。VALUES 子句爲使用 INTO 子句命名的每個直欄指定一值。指定的值可以是：

常數。插入 VALUES 子句中提供的值。

空值。使用關鍵字 NULL 插入空值。直欄必須定義爲能夠包含空值，否則會發生錯誤。

主變數。插入主變數內容。

特別暫存區。插入特別暫存區值；例如 USER。

表示式。插入表示式產生的值。

子查詢，插入 SELECT 陳述式執行結果的值。

DEFAULT 關鍵字。插入直欄預設值。直欄必須有爲其定義的預設值或允許 NULL 值，否則發生錯誤。

您必須在 VALUES 子句中爲 INSERT 陳述式直欄清單中指名的每個直欄提供一值。如果表格中所有直欄都有 VALUES 子句提供的值，則可省略直欄名稱清單。如果直欄有預設值，則可使用關鍵字 DEFAULT 作爲 VALUES 子句中的值。這使得直欄的預設值可置於直欄中。

建議您爲插入值到其中的所有直欄命名，原因如下：

- 您的 INSERT 陳述式描述更詳細。
- 您可以根據直欄名稱來驗證提供的值次序是否適當。
- 您有更佳的資料獨立性。表格中所定義之直欄次序不影響您的 INSERT 陳述式。

如果直欄定義爲允許空值或具有預設值，您不需要在直欄名稱中爲它命名或指定值給它。因爲它會使用預設值。如果直欄定義爲使用預設值，則直欄中有預設值。如果對直欄定義指定了 DEFAULT，但沒有明確的預設值，SQL 會將該資料類型的預設值置於直欄。如果直欄沒有定義給它的預設值，但定義爲允許空值 (未在直欄定義中指定 NOT NULL)，SQL 會將空值置於直欄中。

- 若是數值直欄，預設值爲 0。
- 若是固定長度字元或圖形直欄，預設值爲空白。
- 若是可變長度字元或圖形直欄，或 LOB 直欄，預設值爲長度零的字串。
- 若是日期、時間及時間戳記直欄，預設值爲現行日期、時間或時間戳記。當插入一組記錄時，會在撰寫該組記錄後從系統取出預設日期/時間值。這表示直欄會被指定該組記錄中每一列相同的預設值。
- 若是 DataLink 直欄，預設值對應到 DLVALUE(’’,’URL’,’’)。
- 若是特殊類型直欄，預設值就是對應來源類型的預設值。
- 若是 ROWID 直欄或定義 AS IDENTITY 的直欄，資料庫管理程式會產生預設值。請參閱第 90 頁的『插入識別直欄』。

當您的程式嘗試插入與表格中另一列重複的列時，可能會發生錯誤。根據建立索引時使用的選項，多重空值不一定就是重複值。

- 如果表格具有主要索引鍵、唯一鍵或唯一索引，則不插入列。反之，SQL 傳回 -803 的 SQLCODE。
- 如果表格沒有主要索引鍵、唯一鍵或唯一索引，則可插入列而不會有錯誤。

如果 SQL 在執行 INSERT 陳述式時發現錯誤，便會停止插入資料。如果您指定 COMMIT(*ALL)、COMMIT(*CS)、COMMIT(*CHG) 或 COMMIT(*RR)，則不插入任何列。在 INSERT 具有 SELECT 陳述式或區塊化插入的情況下，會刪除此陳述式已插入的列。如果指定的是 COMMIT(*NONE)，則*不會*刪除任何已插入的列。

SQL 建立的表格是以「重覆使用刪除的記錄」參數 *YES 建立。這可讓資料庫管理程式重覆使用表格中標示爲已刪除的任何列。CHGPF 指令可用來將屬性變更爲 *NO。這使得 INSERT 恆將列新增到表格結尾。

列的插入次序並不保證就是列的擷取次序。

如果插入的列沒有錯誤，則 SQLCA 的 SQLERRD(3) 欄位具有值 1。

註: 若區塊化 INSERT 或含 SELECT 陳述式之 INSERT，則可插入多列。插入的列數反映在 SQLERRD(3) 中。

使用 SELECT 陳述式將列插入表格中

您可以使用 INSERT 陳述式內的 SELECT 陳述式，將指定表格或概略表中所選取的零、一或多列插入另一表格中。您選取列的來源表格可以是插入列的相同表格。如果它們是相同表格，SQL 會建立一個內含所選取列的暫時結果表格，然後從暫時表格插入目標表格中。

這種 INSERT 陳述式的用法就是將資料移到您爲摘要資料所建立的表格中。例如，假設您想要一個顯示每位員工對專案所使用時間的表格。您可以建立一個稱爲 EMPTIME 的表格，其中有直欄 *EMPNUMBER*、*PROJNUMBER*、 *STARTDATE* 及 *ENDDATE*，然後使用下列 INSERT 陳述式填入表格：

```
INSERT INTO CORPDATA.EMPTIME
   (EMPNUMBER, PROJNUMBER, STARTDATE, ENDDATE)
SELECT EMPNO, PROJNO, EMSTDATE, EMENDATE
  FROM CORPDATA.EMPPROJACT
```

INSERT 陳述式內含的 SELECT 陳述式與您用來擷取資料的 SELECT 陳述式並無不同。除了 FOR READ ONLY、FOR UPDATE 或 OPTIMIZE 子句之外，您可以使用所有關鍵字、直欄函數及技術來擷取資料。SQL 會將所有符合搜尋條件的列插入指定的表格中。將列從一個表格插入另一個表格中，並不會影響來源表格或目標表格中現有的列。

多列插入的注意事項

在表格中插入多列時，您應該考量下列情況：

- INSERT 陳述式中明確或不明確列示的直欄數必須等於 SELECT 陳述式中列示的直欄數。
- 當搭配 SELECT 陳述式使用 INSERT 時，所選取直欄中的資料必須與插入直欄中的資料相容。
- 在 INSERT 內含的 SELECT 陳述式沒有傳回列的事件中，會傳回 100 的 SQLCODE 來警示您沒有插入任何列。如果您順利插入一些列，SQLCA 的 SQLERRD(3) 欄位中有一個代表 SQL 實際插入列數的整數。
- 如果 SQL 在執行 INSERT 陳述式時發現錯誤，SQL 便會停止作業。如果您指定 COMMIT (*CHG)、COMMIT(*CS)、COMMIT (*ALL) 或 COMMIT(*RR)，則不插入任何項目到表格中且傳回負值 SQLCODE。如果指定的是 COMMIT(*NONE)，則發生錯誤前插入的任何列仍留在表格中。

- 您可以將兩個或更多表格與 INSERT 陳述式中的 SELECT 陳述式結合。若以此方式載入，則表格可使用 UPDATE、DELETE 及 INSERT 陳述式來操作，因爲存在的列就是表格中實際儲存的列。

使用區塊化 INSERT 陳述式將多列插入表格中

區塊化 INSERT 可透過單一陳述式將多列插入表格中。除 REXX 以外的所有語言都支援區塊化 INSERT 陳述式。插入表格的資料必須在主電腦結構陣列中。如果指示器變數搭配區塊化 INSERT 使用，它們也必須在主電腦結構陣列中。有關特定語言的主電腦結構陣列資訊，請參閱 SQL Programming with Host Languages 一書中有關該語言的章節。

例如，將 10 個員工新增至 CORPDATA.EMPLOYEE 表格中：

```
INSERT INTO CORPDATA.EMPLOYEE
                  (EMPNO,FIRSTNME,MIDINIT,LASTNAME,WORKDEPT)
            10 ROWS VALUES(:DSTRUCT:ISTRUCT)
```

DSTRUCT 是程式所宣告具有 5 個元素的主電腦結構陣列。這 5 個元素對應到 EMPNO、FIRSTNME、MIDINIT、LASTNAME 及 WORKDEPT。 DSTRUCT 至少有 10 個維度來容納插入的 10 列。ISTRUCT 是宣告於程式的主電腦結構陣列。 ISTRUCT 至少有用於指示器的 10 個小整數欄位的維度。

非分散式 SQL 應用程式和分散式應用程式皆支援區塊化 INSERT 陳述式，這兩個應用程式的應用程式伺服器和應用要求程式都是 iSeries 系統。

插入識別直欄

您可以將值插入識別直欄或讓系統爲您插入值。例如，以第 48 頁的『建立及變更識別直欄』建立的表格具有直欄 ORDERNO (識別直欄)、SHIPPED_TO (varchar(36)) 和 ORDER_DATE (日期)。您可以發出下列陳述式來將列插入這個表格中：

```
INSERT INTO ORDERS (SHIPPED_TO, ORDER_DATE)
    VALUES ('BME TOOL', 2002-02-04)
```

在這個案例中，系統會自動爲識別直欄產生一值。您也可以使用 DEFAULT 關鍵字來撰寫此陳述式：

```
INSERT INTO ORDERS (SHIPPED_TO, ORDER_DATE, ORDERNO)
    VALUES ('BME TOOL', 2002-02-04, DEFAULT)
```

插入之後，您可以使用 IDENTITY_VAL_LOCAL 函數來決定系統指定給直欄的值。有關詳細資訊與範例，請參閱 SQL Reference 中的 IDENTITY_VAL_LOCAL 函數。

有時，識別直欄的值由使用者指定，例如在這個 INSERT 陳述式中使用 SELECT：

```
INSERT INTO ORDERS OVERRIDING USER VALUE
   (SELECT * FROM TODAYS_ORDER)
```

在這個案例中，OVERRIDING USER VALUE 指示系統根據 SELECT 忽略提供給識別直欄的值，並爲識別直欄產生新值。如果識別直欄使用 GENERATED ALWAYS 子句建立，則必須使用 OVERRIDING USER VALUE；對 GENERATED BY DEFAULT 而言，它是選用的。如果未指定 OVERRIDING USER VALUE 給 GENERATED BY DEFAULT 識別直欄，則會插入在 SELECT 中提供給直欄的值。

您可以指定 OVERRIDING SYSTEM VALUE 來強制系統對 GENERATED ALWAYS 識別直欄使用 SELECT 中的值。例如，發出下列陳述式：

```
INSERT INTO ORDERS OVERRIDING SYSTEM VALUE
   (SELECT * FROM TODAYS_ORDER)
```

這個 INSERT 陳述式使用 SELECT 中的值；它不爲識別直欄產生新值。如果沒有使用 OVERRIDING SYSTEM VALUE 子句，則您無法對使用 GENERATED ALWAYS 建立的識別直欄提供值。

使用 UPDATE 陳述式變更表格中的資料

若要變更表格中的資料，請使用 UPDATE 陳述式。透過 UPDATE 陳述式，您可以在符合 WHERE 子句搜尋條件的每一列中，變更一或多個直欄的值。 UPDATE 陳述式的結果是表格的零或多列中一或多個變更的直欄值 (根據符合 WHERE 子句所指定搜尋條件的列數)。UPDATE 陳述式如下：

```
UPDATE table-name
  SET column-1 = value-1,
      column-2 = value-2, ...
  WHERE search-condition ...
```

例如，假設某員工已重新安置。若要更新 CORPDATA.EMPLOYEE 表格中員工資料的數個項目來反映更動，可指定：

```
UPDATE CORPDATA.EMPLOYEE
  SET JOB = :PGM-CODE,
      PHONENO = :PGM-PHONE
  WHERE EMPNO = :PGM-SERIAL
```

使用 SET 子句來爲想要更新的每一個直欄指定新值。 SET 子句可爲您要更新的直欄命名，以及提供您要變更的直欄值。您指定的值可爲：

直欄名稱。將直欄的現行值置換爲同一列中另一直欄的內容。

常數。將直欄的現行值置換爲 SET 子句提供的值。

NULL 值。使用關鍵字 NULL，將直欄的現行值置換爲 NULL 值。建立表格後，直欄必須定義爲能夠包含 NULL 值，否則會發生錯誤。

主變數。將直欄的現行值置換爲主變數內容。

特別暫存區。將直欄的現行值置換爲特別暫存區值；例如 USER。

表示式。將直欄的現行值置換爲表示式產生的值。

純量子選擇。將直欄的現行值置換爲子查詢傳回的值。

DEFAULT 關鍵字。將直欄的現行值置換爲直欄的預設值。直欄必須有爲其定義的預設值或允許 NULL 值，否則發生錯誤。

下列爲使用許多不同值的陳述式範例：

```
UPDATE WORKTABLE
   SET COL1 = 'ASC',
       COL2 = NULL,
       COL3 = :FIELD3,
       COL4 = CURRENT TIME,
       COL5 = AMT - 6.00,
       COL6 = COL7
    WHERE EMPNO = :PGM-SERIAL
```

若要識別所要更新的列，請使用 WHERE 子句：

- 若要更新單一列，請使用僅選取一列的 WHERE 子句。
- 若要更新多列，請使用僅選取您所要更新列的 WHERE 子句。

您可以省略 WHERE 子句。如果這麼做的話，SQL 會以您提供的值來更新表格或概略表中每一列。

如果資料庫管理程式在執行您的 UPDATE 陳述式時發現錯誤，它會停止更新並傳回負值 SQLCODE。如果您指定 COMMIT(*ALL)、COMMIT(*CS)、COMMIT(*CHG) 或 COMMIT(*RR)，則不變更表格中任何列 (此陳述式已變更的列 (若有的話) 則復置回先前值)。如果指定 COMMIT(*NONE)，則已變更的任何列不復置回先前值。

如果資料庫管理程式找不到符合搜尋條件的任何列，則傳回 +100 的 SQLCODE。

註: UPDATE 陳述式可能更新了不止一列。更新的列數反映在 SQLCA 的 SQLERRD(3) 中。

UPDATE 陳述式的 SET 子句有許多方式可用來決定要在所更新每一列中設定的實際值。下列範例列出每一個直欄及其對應值：

```
UPDATE EMPLOYEE
  SET WORKDEPT = 'D11',
      PHONENO = '7213',
      JOB = 'DESIGNER'
  WHERE EMPNO = '000270'
```

也可指定所有欄位及所有值來撰寫先前的更新：

```
UPDATE EMPLOYEE
  SET (WORKDEPT, PHONENO, JOB)
      = ('D11', '7213', 'DESIGNER')
  WHERE EMPNO = '000270'
```

有關更新表格中資料的其它方法，請參閱下列各節：

- 『使用純量子選擇來更新表格』
- 『使用另一表格中的列來更新表格』
- 第 93 頁的『更新擷取自表格的資料』

有關 UPDATE 陳述式的完整說明，請參閱 SQL Reference 中的 UPDATE。

使用純量子選擇來更新表格

選取一個更新值 (或多個值) 的另一個方法是使用純量子選擇。純量子選擇可讓您更新一或多個直欄，方法是將它們設爲從另一表格中選取的一或多個值。在下列範例中，員工調到另一個部門，且繼續負責相同的專案。員工表格已更新成包含新部門號碼。現在，需要更新專案表格來反映此員工 (員工號碼爲 '000030') 的新部門號碼。

```
UPDATE PROJECT
  SET DEPTNO =
        (SELECT WORKDEPT FROM EMPLOYEE
           WHERE PROJECT.RESPEMP = EMPLOYEE.EMPNO)
  WHERE RESPEMP='000030'
```

此相同技術可用來更新具有單一 SELECT 傳回的多重值之直欄清單。

使用另一表格中的列來更新表格

若要更新一個表格中的整列，也可使用另一表格的列值來更新。假設有一個主要類別排程需要以表格備份中所做的變更來更新。所做的工作變更會在每晚複製及合併到主要表格中。兩表格有完全相同的直欄，其中一個直欄 CLASS_CODE 是唯一鍵直欄。

```
UPDATE CL_SCHED
  SET ROW =
       (SELECT * FROM MYCOPY
          WHERE CL_SCHED.CLASS_CODE = MYCOPY.CLASS_CODE)
```

此更新會將 CL_SCHED (具有 MYCOPY 中的值) 中所有的列更新。

更新識別直欄

您可以將識別直欄中的值更新成指定值，或讓系統產生新值。例如，透過第 48 頁的『建立及變更識別直欄』建立的表格，其中有直欄 ORDERNO (識別直欄)、SHIPPED_TO (varchar(36)) 及 ORDER_DATE (日期)，您可以發出下列陳述式來更新識別直欄中的值。

```
UPDATE ORDERS
  SET (ORDERNO, ORDER_DATE)=
       (DEFAULT, 2002-02-05)
  WHERE SHIPPED_TO = 'BME TOOL'
```

系統會自動爲識別直欄產生一值。您可以使用 OVERRIDING SYSTEM VALUE 子句來置換系統產生的值：

```
UPDATE ORDERS OVERRIDING SYSTEM VALUE
  SET (ORDERNO, ORDER_DATE)=
       (553, '2002-02-05')
  WHERE SHIPPED_TO = 'BME TOOL'
```

更新擷取自表格的資料

您可以使用游標來更新擷取的資料列。有關游標的詳細資訊，請參閱 第 113 頁的第 9 章, 『使用游標』。在 SELECT 陳述式上，使用 FOR UPDATE OF，後接可更新的直欄清單。然後，使用游標控制的 UPDATE 陳述式。WHERE CURRENT OF 子句指定游標來指向您要更新的列。如果指定 FOR UPDATE OF、ORDER BY、FOR READ ONLY 或 SCROLL 子句，但沒指定 DYNAMIC 子句，則會更新所有直欄。

如果已指定並執行多列 FETCH 陳述式，則游標定位到區塊最後一列。因此，如果 UPDATE 陳述式上指定了 WHERE CURRENT OF 子句，則會更新區塊最後一列。如果必須更新區塊內某列，程式必須先將游標定位到該列。然後，才能指定 UPDATE WHERE CURRENT OF。請參考下列範例：

表 8. 更新表格

可捲動的游標 SQL 陳述式	說明
EXEC SQL **DECLARE** THISEMP **DYNAMIC SCROLL CURSOR FOR** **SELECT** EMPNO, WORKDEPT, BONUS **FROM** CORPDATA.EMPLOYEE **WHERE** WORKDEPT = 'D11' **FOR UPDATE OF** BONUS END-EXEC.	
EXEC SQL **OPEN** THISEMP END-EXEC.	

表 8. 更新表格 (繼續)

可捲動的游標 SQL 陳述式	說明
EXEC SQL **WHENEVER NOT FOUND** GO TO CLOSE-THISEMP END-EXEC.	
EXEC SQL **FETCH NEXT FROM** THISEMP **FOR** 5 **ROWS** **INTO** :DEPTINFO :IND-ARRAY END-EXEC.	DEPTINFO 和 IND-ARRAY 在程式中宣告爲主電腦結構陣列和指示器陣列。
... 判斷部門 D11 中是否有任何員工獲取 $500.00 以下的紅利。 如果有的話，請將記錄更新爲新的最小值 $500.00。	
EXEC SQL **FETCH** RELATIVE :NUMBACK **FROM** THISEMP END-EXEC.	... 定位於區塊中的記錄，藉由反向次序提取來更新。
EXEC SQL **UPDATE** CORPDATA.EMPLOYEE **SET** BONUS = 500 **WHERE CURRENT OF** THISEMP END-EXEC.	... 更新部門 D11 中的紅利，其新最小值 $500.00 以下的員工紅利。
EXEC SQL **FETCH** RELATIVE :NUMBACK **FROM** THISEMP **FOR** 5 **ROWS** **INTO** :DEPTINFO :IND-ARRAY END-EXEC.	... 定位於已提取的相同區塊開頭，再重新提取區塊。 (NUMBACK -(5 - NUMBACK - 1))
... 返回以決定區塊中是否還有員工的紅利在 $500.00 以下。 ... 返回以提取及處理下一區塊的列。	
CLOSE-THISEMP. EXEC SQL **CLOSE** THISEMP END-EXEC.	

限制

您無法將 FOR UPDATE OF 與併入下列任何元素的 SELECT 陳述式一起使用：

- 第一個 FROM 子句定義多個表格或概略表。
- 第一個 FROM 子句定義唯讀概略表。
- 第一個 SELECT 子句指定關鍵字 DISTINCT。
- 第一個 FROM 子句定義使用者定義的表格函數。
- 外部子選擇含有 GROUP BY 子句。
- 外部子選擇含有 HAVING 子句。
- 第一個 SELECT 子句含有直欄函數。
- SELECT 陳述式含有 UNION 或 UNION ALL 運算子。
- SELECT 陳述式含有 ORDER BY 子句，但未指定 FOR UPDATE OF 子句和 DYNAMIC SCROLL。

- SELECT 陳述式併入 FOR FETCH ONLY 子句。
- 指定的 SCROLL 關鍵字不含 DYNAMIC。
- SELECT 清單併入 DATALINK 直欄，但未指定 FOR UPDATE OF 子句。
- 第一個子選擇需要暫時結果表格。
- SELECT 陳述式併入 FETCH FIRST *n* ROWS ONLY。

如果指定了 FOR UPDATE OF 子句，則您無法更新未在 FOR UPDATE OF 子句中指定的直欄。但可以爲 FOR UPDATE OF 子句中不在 SELECT 清單的直欄命名，如這個範例所示：

```
SELECT A, B, C FROM TABLE
FOR UPDATE OF A,E
```

請勿在 FOR UPDATE OF 子句中命名超過您需求的直欄數；當您存取表格時，並不會使用到那些直欄的索引。

使用 DELETE 陳述式移除表格中的列

若要移除表格中的列，請使用 DELETE 陳述式。當您將某列 DELETE 時，表示您移除了整列。DELETE 不移除列中特定的直欄。DELETE 陳述式的結果是移除表格的零或多列 (根據符合 WHERE 子句所指定搜尋條件的列數)。如果您在 DELETE 陳述式中省略了 WHERE 子句，SQL 會移除表格的所有列。 DELETE 陳述式如下：

```
DELETE FROM table-name
  WHERE search-condition ...
```

例如，假設部門 D11 已遷往另一處。您要使用 D11 的 WORKDEPT 值來刪除 CORPDATA.EMPLOYEE 表格中每一列，如下所示：

```
DELETE FROM CORPDATA.EMPLOYEE
  WHERE WORKDEPT = 'D11'
```

WHERE 子句指示 SQL 您要刪除表格中哪些列。SQL 會刪除基本表格中符合搜尋條件的所有列。刪除概略表中的列也會刪除基本表格中的列。您可以省略 WHERE 子句，但建議您最好加入它，因爲 DELETE 陳述式若不含 WHERE 子句，則會刪除表格或概略表中所有列。若要刪除表格定義及表格內容，請發出 DROP 陳述式。有關 DROP 陳述式的詳細資訊，請參閱 *SQL Reference* 一書中的 DROP 陳述式主題。

如果 SQL 在執行您的 DELETE 陳述式時發現錯誤，它會停止刪除資料並傳回負值 SQLCODE。如果您指定 COMMIT(*ALL)、COMMIT(*CS)、COMMIT(*CHG) 或 COMMIT(*RR)，則不刪除表格中任何列 (此陳述式已刪除的列 (若有的話) 則復置回先前值)。如果指定 COMMIT(*NONE)，則已刪除的任何列不復置回先前值。

如果 SQL 找不到符合搜尋條件的任何列，則傳回 +100 的 SQLCODE。

註: DELETE 陳述式可能刪除了多列。刪除的列數反映在 SQLERRD(3) 中。

有關 DELETE 陳述式的詳細資訊，請參閱 *SQL Reference* 一書中的 DELETE 陳述式主題。

第 7 章 使用子查詢

您可以在搜尋條件中使用子查詢，作爲選取資料的另一種方式。子查詢可以用於表示式、選取清單，以及 ORDER BY 與 GROUP BY 子句中。如需相關明細，請參閱下面各節：

SELECT 陳述式中的子查詢

在簡式 WHERE 及 HAVING 子句中，您可以使用文字值、直欄名稱、表示式或特別暫存區來指定搜尋條件。在那些搜尋條件中，您可以知道您正在搜尋某一特定值。然而，有時在您從表格中擷取其它資料前，無法提供該值。例如，假設您想要一份清單，其中含有員工編號、姓名，以及從事特定專案 (假設專案號碼是 MA2100) 的所有員工的工作代碼。陳述式第一個部份的撰寫很簡單：

```
SELECT EMPNO, LASTNAME, JOB
FROM CORPDATA.EMPLOYEE
WHERE EMPNO ...
```

但您無法更進一步，因爲 CORPDATA.EMPLOYEE 表格不含專案號碼資料。如果不對 CORPDATA.EMP_ACT 表格發出另一個 SELECT 陳述式。您就不知道哪些員工是從事專案 MA2100。

使用 SQL，您可以在一個陳述式中加上另一個 SELECT 陳述式以形成巢狀，即可解決此問題。內部 SELECT 陳述式稱爲**子查詢**。環繞在此子查詢外的 SELECT 陳述式稱爲**外部層次 SELECT**。使用子查詢，您只要發出一個 SQL 陳述式，就可以擷取員工編號、姓名及從事專案 MA2100 的員工工作代碼：

```
SELECT EMPNO, LASTNAME, JOB
FROM CORPDATA.EMPLOYEE
WHERE EMPNO IN
   (SELECT EMPNO
      FROM CORPDATA.EMPPROJACT
      WHERE PROJNO = 'MA2100')
```

若要更瞭解此 SQL 陳述式的結果，請想像 SQL 完成下列處理：

步驟 1：SQL 會評估子查詢以取得 EMPNO 值的清單：

```
(SELECT EMPNO
     FROM CORPDATA.EMPPROJACT
     WHERE PROJNO= 'MA2100')
```

則會產生暫時的結果表：

EMPNO from CORPDATA.EMPPROJACT
000010
000110

步驟 2：然後，暫時的結果表可以當成外部層次 SELECT 搜尋條件中的清單使用。基本上，這才是要執行的陳述式。

```
SELECT EMPNO, LASTNAME, JOB
     FROM CORPDATA.EMPLOYEE
     WHERE EMPNO IN
            ('000010', '000110')
```

最終的結果表格就會像這個表格：

EMPNO	LASTNAME	JOB
000010	HAAS	PRES
000110	LUCCHESSI	SALESREP

如需相關明細，請參閱下面各節：

- 『相互關係』
- 『子查詢與搜尋條件』
- 第 99 頁的『如何使用子查詢』
- 第 100 頁的『使用子查詢的注意事項』
- 第 101 頁的『相關子查詢』
- 第 104 頁的『在 UPDATE 陳述式中使用相關子查詢』
- 第 104 頁的『在 DELETE 陳述式中使用相關子查詢』

相互關係

子查詢的目的在於提供必要資訊，以評估橫列 (WHERE 子句) 或橫列群組 (HAVING 子句) 的述詞。此動作可以透過子查詢產生的結果表格完成。在概念上，只要必須處理新橫列或橫列群組時，即會評估子查詢。事實上，如果每一列或群組的子查詢都相同，則只會評估一次。這類的子查詢表示是**不相關的**。

部份子查詢會以從列到列，或從群組到群組的方式傳回不同的值。容許此方式的機制稱爲**相互關係**，且子查詢則表示是**相關的**。如需相關子查詢的其餘特定資訊，請參閱第 101 頁的『相關子查詢』。

子查詢與搜尋條件

子查詢可以是搜尋條件的一部份。搜尋條件的格式爲*運算元 運算子 運算元*。任一運算元都可以是一個子查詢。在下面範例中，第一個**運算元**是 EMPNO，且**運算子**是 IN。搜尋條件可以是 WHERE 或 HAVING 子句的一部份。子句可以含有一個以上的搜尋條件，而該搜尋條件可以有一個子查詢。就像任何其它搜尋條件一樣，內含子查詢的搜尋條件可以使用括弧括住，之前可以有關鍵字 NOT，且可以透過關鍵字 AND 及 OR 鏈結到其它搜尋條件。例如，查詢的 WHERE 子句會類似下面這個：

```
WHERE  (subquery1) = X AND (Y > SOME (subquery2) OR Z = 100)
```

子查詢也可以出現在其它子查詢的搜尋條件中。這類子查詢被稱爲部份巢狀層次上的**巢狀**。例如，外部層次 SELECT 中的子查詢中的子查詢是兩層巢狀層次上的巢狀。 SQL 可以向下形成有 32 個巢狀層次的巢狀。

如何使用子查詢

有數種方式可以在 WHERE 或 HAVING 子句中併入子查詢：

- 『基本比較』
- 『數量化比較 (ALL、ANY 及 SOME)』
- 第 100 頁的『IN 關鍵字』
- 第 100 頁的『EXISTS 關鍵字』

基本比較

您可以在任何比較運算子之前或之後使用子查詢。子查詢最多可以傳回一個值。該值可以是直欄功能或算術運算的結果。然後，SQL 會將子查詢產生的值與比較運算子另一端的值相比較。例如，假設您想要找出員工編號，姓名及員工薪資，且該員工的教育水準高於全公司的平均教育水準。

```
SELECT EMPNO, LASTNAME, SALARY
FROM CORPDATA.EMPLOYEE
WHERE EDLEVEL >
   (SELECT AVG(EDLEVEL)
      FROM CORPDATA.EMPLOYEE)
```

SQL 會先評估子查詢，然後在 SELECT 陳述式的 WHERE 子句中替換結果。在本例中，結果是全公司平均教育水準。除了傳回單一值外，子查詢也可以完全不傳回任何值。如果這麼做，則比較的結果不明。

數量化比較 (ALL、ANY 及 SOME)

您可以在後面接有關鍵字 ALL、ANY 或 SOME 的比較運算子後使用子查詢。使用此方法時，子查詢會傳回零個、一個或許多個值，包括 NULL 值。您可以在下列方式中使用 ALL、ANY 及 SOME：

- 使用 ALL，表示必須以指示的方式來比較您提供的值與子查詢傳回的**所有**值。例如，假設您使用大於比較運算子與 ALL：

  ```
  ... WHERE 表示式 > ALL (子查詢)
  ```

  爲了滿足此 WHERE 子句，表示式中的值必須大於子查詢傳回的所有值 (亦即，大於最高的值)。如果子查詢傳回空集合 (亦即，未選取任何值)，則滿足條件。

- 使用 ANY 或 SOME，表示必須以指定的方式來比較您提供的值與*至少一個*子查詢傳回值。例如，假設您使用大於比較運算子與 **ANY**：

  ```
  ... WHERE 表示式 > ANY (子查詢)
  ```

  爲了滿足此 WHERE 子句，表示式中的值至少必須大於一個子查詢傳回的值 (亦即，大於最低的值)。如果子查詢傳回空集合，則不滿足此條件。

註: 當子查詢傳回一或多個 NULL 值時，您可能會對此結果感到驚訝，除非您熟悉形式邏輯。如需相關明細，請參閱 SQL Reference 中有關數量化述詞的說明。

IN 關鍵字

您可以使用 IN 來表示：表示式中的值必須是在子查詢傳回的值中。使用 IN 就等於使用 *=ANY* 或 *=SOME*。ANY 及 SOME 的使用方法已在前面說明。您也可以使用 IN 關鍵字與 NOT 關鍵字，以便在該值不是子查詢傳回的值時，可以選取列。例如，您可以使用：

```
... WHERE WORKDEPT NOT IN (SELECT ...)
```

EXISTS 關鍵字

在這類的子查詢中，SQL 會評估子查詢並使用其結果作爲外部層次 SELECT 中 WHERE 子句的一部份。相對的，當您使用關鍵字 EXISTS 時， SQL 只會檢查子查詢是否傳回一或多列。如果是的話，則滿足條件。如果未傳回任何列，則不滿足條件。例如：

```
SELECT EMPNO,LASTNAME
FROM CORPDATA.EMPLOYEE
WHERE EXISTS
   (SELECT *
      FROM CORPDATA.PROJECT
      WHERE PRSTDATE > '1982-01-01');
```

在範例中，如果 CORPDATA.PROJECT 表格中有任何專案預估開始日期比 1982 年 1 月 1 日晚，則搜尋條件爲眞。請注意：本範例並未顯示 EXISTS 的完整功能，因爲針對外部層次 SELECT 檢查的每一列結果永遠都相同。結果，不是每一列都出現在結果中，就是未出現任何列。在功能更強大的範例中，子查詢本身是相關的，且會從列到列變更。如需相關子查詢的其餘相關資訊，請參閱第 101 頁的『相關子查詢』。

如範例中所顯示，您不需要在 EXISTS 子句的子查詢選取清單中指定直欄名稱。相反的，您應撰寫 SELECT * 的程式碼。

您也可以使用 EXISTS 關鍵字與 NOT 關鍵字，以便在您指定的資料或條件不存在時，可以選取列。 您可以使用下列方式：

```
... WHERE NOT EXISTS (SELECT ...)
```

使用子查詢的注意事項

1. 在巢狀化 SELECT 陳述式時，您可以依照您的需求來使用陳述式 (1 到 31 個子查詢)，雖然每多一個子查詢，效能就會變得更慢。
2. 當外部陳述式是 SELECT 陳述式 (在任何巢狀層次) 時：
   - 作爲外部陳述式的子查詢可以相同的表格或概略表爲基礎，或以不同的表格或概略表爲基礎。
   - 即使當外部層次 SELECT 是 DECLARE CURSOR、CREATE TABLE、 CREATE VIEW 或 INSERT 陳述式的一部份時，您仍然可以在外部層次 SELECT 的 WHERE 子句中使用子查詢。
   - 您可以在 SELECT 陳述式的 HAVING 子句中使用子查詢。當您這麼做時，SQL 會評估子查詢並使用它來定義每一個群組。
3. 當陳述式是 UPDATE 或 DELETE 陳述式時，您可以在 UPDATE 或 DELETE 陳述式的 WHERE 子句中使用子查詢。您也可以在 UPDATE 陳述式的 SET 子句中使用子查詢。
4. 在 UPDATE 陳述式的 SET 子句中使用子查詢時，次選擇的結果表格所含值的數目可以與要更新的對應直欄清單相同。在所有其它的情況下，子查詢的結果表格必須是由單一直欄組成，除非子查詢是與 EXISTS 關鍵字一起使用。若爲使用關鍵字

ALL、 ANY、SOME 或 EXISTS 的述詞，從子查詢傳回的列數可能會是零到許多列。對於所有其它的子查詢，傳回的列數必須是零或一。

5. 子查詢不能含有 ORDER BY、UNION、UNION ALL、FOR READ ONLY、 FETCH FIRST *n* ROWS、UPDATE 或 OPTIMIZE 子句。

相關子查詢

在之前說明的子查詢中，SQL 會評估子查詢一次、替換搜尋條件中的子查詢結果，以及依據搜尋條件的值來評估外部層次 SELECT。您也可以撰寫一個子查詢，讓 SQL 在檢查外部層次 SELECT 中的每一新列 (WHERE 子句) 或橫列群組 (HAVING 子句) 時，必須重新評估子查詢。這稱爲**相關子查詢**。

相關名稱與參照

在子查詢的搜尋條件中可以出現相關參照。參照的格式恆爲 X.C，其中 X 是相關名稱，且 C 是出現 X 的表格中的直欄名稱。

您可以爲 FROM 子句中出現的任何表格定義相關名稱。相關名稱提供查詢中表格的唯一名稱。在查詢及其巢狀子選擇中可以使用多次相同的表格名稱。爲每一個表格參照選取不同的相關名稱，可能可以唯一指定直欄參照的表格。

相關名稱定義於查詢的 FROM 子句。此查詢可以是外部層次 SELECT，或內含參照查詢的任何子查詢。例如，假設查詢含有子查詢 A、B 及 C，且 A 包含 B 且 B 包含 C。則 C 中使用的相關名稱會定義於 B、A 或外部層次 SELECT。若要定義相關名稱，只要在表格名稱後併入相關名稱。請在表格名稱及其相關名稱之間留下一個或多個空格，且如果後面接著其它表格名稱，則請在相關名稱後加入一個逗號。下列 FROM 子句會定義表格 TABLEA 及 TABLEB 的相關名稱 TA 及 TB，且表格 TABLEC 沒有任何相關名稱。

```
FROM TABLEA TA, TABLEC, TABLEB TB
```

在子查詢中可以出現任何出量的相關參考。例如，搜尋條件中的一個相關名稱可以定義於外部層次 SELECT 中，同時另一個名稱可以定義於內含的子查詢中。

執行子查詢之前，參照直欄中的值恆會被替換爲相關參照。

範例：WHERE 子句中的相關子查詢

假設您想要一份員工清單，其中所有員工的教育水準均高於各自部門的平均教育水準。若要取得此資訊，SQL 必須搜尋 CORPDATA.EMPLOYEE 表格。對於表格中的每一個員工， SQL 必須比較員工的教育水準與該員工部門的平均教育水準。在子查詢中，您告知 SQL 要在現行列中計算部門編號的平均教育水準。例如：

```
SELECT EMPNO, LASTNAME, WORKDEPT, EDLEVEL
 FROM CORPDATA.EMPLOYEE X
 WHERE EDLEVEL >
    (SELECT AVG(EDLEVEL)
       FROM CORPDATA.EMPLOYEE
       WHERE WORKDEPT = X.WORKDEPT)
```

相關子查詢看起來像是不相關的，除非出現一或多個相關參考。在本例中，於子選擇 FROM 子句的 X.WORKDEPT 中只出現單一一個相關參考。此處的限定元 *X*，爲定義於外部 SELECT 陳述式的 FROM 子句中的相關名稱。在該子句中，*X* 被引用爲表格 CORPDATA.EMPLOYEE 的相關名稱。

現在，請考慮針對 CORPDATA.EMPLOYEE 的指定列執行子查詢時會發生什麼事。在執行子查詢之前，會以該列的 WORKDEPT 直欄值置換 X.WORKDEPT。例如，假設該列是代表 CHRISTINE I HAAS。她的工作部門是 A00，那就是此列的 WORKDEPT 值。此列執行的子查詢爲：

```
(SELECT AVG(EDLEVEL)
   FROM CORPDATA.EMPLOYEE
   WHERE WORKDEPT = 'A00')
```

因此，針對考量的列，子查詢會產生 Christine 部門的平均教育水準。然後，會在外部陳述式中比較此值與 Christine 自己的教育水準。部份其它列的 WORKDEPT 會有不同的值，該值會代替 A00 出現在子查詢中。例如，針對代表 MICHAEL L THOMPSON 的列，此值是 B01，且該列的子查詢會遞送部門 B01 的平均教育水準。

查詢產生的結果表格會具有下列值：

表 9. 之前查詢的結果集

EMPNO	LASTNAME	WORKDEPT	EDLEVEL
000010	HAAS	A00	18
000030	KWAN	C01	20
000070	PULASKI	D21	16
000090	HENDERSON	E11	16
000110	LUCCHESSI	A00	19
000160	PIANKA	D11	17
000180	SCOUTTEN	D11	17
000210	JONES	D11	17
000220	LUTZ	D11	18
000240	MARINO	D21	17
000260	JOHNSON	D21	16
000280	SCHNEIDER	E11	17
000320	MEHTA	E21	16
000340	GOUNOT	E21	16
200010	HEMMINGER	A00	18
200220	JOHN	D11	18
200240	MONTEVERDE	D21	17
200280	SCHWARTZ	E11	17
200340	ALONZO	E21	16

範例：HAVING 子句中的相關子查詢

假設您想要一份部門清單，其中所有部門的平均薪資高於該區域 (WORKDEPT 是以相同字母起首的所有部門都屬於相同區域) 的平均薪資。若要取得此資訊，SQL 必須搜尋 CORPDATA.EMPLOYEE 表格。針對表格中的每一個部門， SQL 會比較部門的平均薪資與區域的平均薪資。在子查詢中，SQL 會計算現行群組中部門區域的平均薪資。例如：

```
SELECT WORKDEPT, DECIMAL(AVG(SALARY),8,2)
 FROM CORPDATA.EMPLOYEE X
 GROUP BY WORKDEPT
```

```
HAVING AVG(SALARY) >
   (SELECT AVG(SALARY)
      FROM CORPDATA.EMPLOYEE
      WHERE SUBSTR(X.WORKDEPT,1,1) = SUBSTR(WORKDEPT,1,1))
```

請考慮針對 CORPDATA.EMPLOYEE 的指定部門執行子查詢時會發生什麼事。執行子查詢之前，會以該群組的 WORKDEPT 直欄值置換 X.WORKDEPT。例如，假設選取的第一個群組中 WORKDEPT 的值是 A00。此群組執行的子查詢爲：

```
(SELECT AVG(SALARY)
   FROM CORPDATA.EMPLOYEE
   WHERE SUBSTR('A00',1,1) = SUBSTR(WORKDEPT,1,1))
```

因此，針對考量的群組，子查詢會產生區域的平均薪資。然後，會在外部陳述式中比較此值與部門 'A00' 的平均薪資。對於 WORKDEPT 是 'B01' 的部份其它群組，子查詢會產生部門 B01 所屬區域的平均薪資。

查詢產生的結果表格會具有下列值：

WORKDEPT	AVG SALARY
D21	25668.57
E01	40175.00
E21	24086.66

範例：選取清單中的相關子查詢

假設您想要一份所有部門的清單，包括部門名稱、編號與經理姓名。您可以在 CORPDATA.DEPARTMENT 表格中取得部門名稱與編號。然而，DEPARTMENT 只有經理編號，而沒有經理姓名。若要找出每一個部門的經理姓名，必須從 EMPLOYEE 表格中找出符合 DEPARTMENT 表格中經理編號的員工姓名，然後傳回相符的橫列名稱。只會傳回目前已被分派經理的部門。請執行下列：

```
SELECT DEPTNO, DEPTNAME,
    (SELECT FIRSTNME CONCAT ' ' CONCAT
         MIDINIT CONCAT ' ' CONCAT LASTNAME
       FROM EMPLOYEE X
       WHERE X.EMPNO = Y.MGRNO) AS MANAGER_NAME
  FROM DEPARTMENT Y
  WHERE MGRNO IS NOT NULL
```

針對每一個傳回的 DEPTNO 及 DEPTNAME 列，系統會在其中尋找 EMPNO = MGRNO，然後傳回經理的姓名。查詢產生的結果表格會具有下列值：

表 10.

DEPTNO	DEPTNAME	MANAGER_NAME
A00	SPIFFY COMPUTER SERVICE DIV.	CHRISTINE I HAAS
B01	PLANNING	MICHAEL L THOMPSON
C01	INFORMATION CENTER	SALLY A KWAN
D11	MANUFACTURING SYSTEMS	IRVING F STERN
D21	ADMINISTRATION SYSTEMS	EVA D PULASKI
E01	SUPPORT SERVICES	JOHN B GEYER
E11	OPERATIONS	EILEEN W HENDERSON

表 10. (繼續)

DEPTNO	DEPTNAME	MANAGER_NAME
E21	SOFTWARE SUPPORT	THEODORE Q SPENSER

在 UPDATE 陳述式中使用相關子查詢

當您在 UPDATE 陳述式中使用相關子查詢時，相關名稱會參照您想要更新的列。例如，當專案的所有活動必須在 1983 年 9 月之前完成時，您的部門會將該專案視爲優先專案。您可以使用下面的 SQL 陳述式來評估 CORPDATA.PROJECT 表格中的專案，並在每一個優先專案的 PRIORITY 直欄 (您爲了此目的而新增至 CORPDATA.PROJECT 的直欄) 中寫入 1 (表示「優先順序」的旗號)。

```
UPDATE CORPDATA.PROJECT X
  SET PRIORITY = 1
  WHERE '1983-09-01' >
     (SELECT MAX(EMENDATE)
         FROM CORPDATA.EMPPROJACT
         WHERE PROJNO = X.PROJNO)
```

當 SQL 檢查 CORPDATA.EMPPROJACT 表格中的每一列時，它會決定專案所有活動 (從 CORPDATA.PROJECT 表格) 的活動結束日期最大值 (EMENDATE)。如果與專案相關的每個活動結束日期都在 1983 年 9 月之前，則會定義 CORPDATA.PROJECT 表格中的現行列並加以更新。

以變更的訂購數量更新主要訂單表格。如果訂單表格中的數量未設定 (NULL 值)，請保留主要訂單表格中的值。

```
UPDATE MASTER_ORDERS X
   SET QTY=(SELECT COALESCE (Y.QTY, X.QTY)
               FROM ORDERS Y
               WHERE X.ORDER_NUM = Y.ORDER_NUM)
   WHERE X.ORDER_NUM IN (SELECT ORDER_NUM
               FROM ORDERS)
```

在本例中，會檢查 MASTER_ORDERS 表格的每一列，以查看是否在 ORDERS 表格中具有相對應的列。如果在 ORDERS 表格中沒有相符列，則會使用 COALESCE 函數以傳回 QTY 直欄的值。如果 ORDERS 表格中的 QTY 具有非 NULL 值，則會使用該值來更新 MASTER_ORDERS 表格中的 QTY 直欄。如果 ORDERS 表格中的 QTY 值是 NULL，則會以它自己的值來更新 MASTER_ORDERS QTY 直欄。

在 DELETE 陳述式中使用相關子查詢

當您在 DELETE 陳述式中使用相關子查詢時，相關名稱代表您刪除的列。SQL 會針對在 DELETE 陳述式中具名的表格的每一列評估一次相關子查詢，以決定是否要刪除該列。

假設已刪除 CORPDATA.PROJECT 表格中的某一列。則必須同時在 CORPDATA.EMPPROJACT 表格中，刪除與被刪除的專案相關的橫列。若要執行此動作，您可以使用：

```
DELETE FROM CORPDATA.EMPPROJACT X
  WHERE NOT EXISTS
     (SELECT *
        FROM CORPDATA.PROJECT
        WHERE PROJNO = X.PROJNO)
```

SQL 會針對 CORPDATA.EMP_ACT 表格中的每一列，決定在 CORPDATA.PROJECT 表格中是否存在具有相同專案號碼的列。如果沒有，則會刪除 CORPDATA.EMP_ACT 列。

第 8 章 SQL 的排序順序

排序順序定義了比較或排序字集中的不同字元時，字元間的相互關係。如需排序順序的完整討論，請參閱 SQL Reference 一書中的 Sort Sequence 一節。

排序順序用於 SQL 陳述式中執行的所有字元與 UCS-2 圖形比較。單位元組和雙位元組字元資料都有排序順序表。每一個單位元組排序順序表有一個相關的雙位元組排序順序表，反過來也一樣。必須實施查詢時，就會執行這兩種順序表之間的轉換。此外，CREATE INDEX 陳述式會在索引中參照到字元直欄套用排序順序 (實際上是在陳述式執行時)。

如需詳細資訊，請參閱下列主題：

- 『ORDER BY 和列選項使用的排序順序』
- 第 108 頁的『排序順序和 ORDER BY』
- 第 109 頁的『列選項』
- 第 110 頁的『排序順序和概略表』
- 第 110 頁的『排序順序和 CREATE INDEX 陳述式』
- 第 110 頁的『排序順序和限制』

ORDER BY 和列選項使用的排序順序

若要瞭解如何使用排序順序，請以本節的範例執行下表中顯示的 STAFF 表格。請注意，JOB 直欄中的值是大小寫混用。您可以看到 'Mgr'、'MGR' 和 'mgr' 等值。

表 11. STAFF 表格

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
10	Sanders	20	Mgr	7	18357.50	0
20	Pernal	20	Sales	8	18171.25	612.45
30	Merenghi	38	MGR	5	17506.75	0
40	OBrien	38	Sales	6	18006.00	846.55
50	Hanes	15	Mgr	10	20659.80	0
60	Quigley	38	SALES	0	16808.30	650.25
70	Rothman	15	Sales	7	16502.83	1152.00
80	James	20	Clerk	0	13504.60	128.20
90	Koonitz	42	sales	6	18001.75	1386.70
100	Plotz	42	mgr	6	18352.80	0

在以下各範例中，會使用下列各項顯示每一個陳述式的結果：

- *HEX 排序順序
- 使用語言 ID ENU 的共用權重排序順序
- 使用語言 ID ENU 的唯一權重排序順序

註: 在 CRTSQLxxx、STRSQL 或 RUNSQLSTM 指令上使用 SRTSEQ(*LANGIDUNQ)，或 SRTSEQ(*LANGIDSHR) 和 LANGID(ENU)，或者使用 SET OPTION 陳述式，選擇 ENU 做爲語言 ID。

排序順序和 ORDER BY

以下 SQL 陳述式會使結果表格利用 JOB 直欄中的值排序：

```
SELECT * FROM STAFF ORDER BY JOB
```

表 12 顯示使用 *HEX 排序順序的結果表格。各列會根據 JOB 直欄中的 EBCDIC 值排序。此時，所有的小寫字母會在大寫字母之前排序。

*表 12. 使用 *HEX 排序順序的 "SELECT * FROM STAFF ORDER BY JOB"。*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
100	Plotz	42	mgr	6	18352.80	0
90	Koonitz	42	sales	6	18001.75	1386.70
80	James	20	Clerk	0	13504.60	128.20
10	Sanders	20	Mgr	7	18357.50	0
50	Hanes	15	Mgr	10	20659.80	0
30	Merenghi	38	MGR	5	17506.75	0
20	Pernal	20	Sales	8	18171.25	612.45
40	OBrien	38	Sales	6	18006.00	846.55
70	Rothman	15	Sales	7	16502.83	1152.00
60	Quigley	38	SALES	0	16808.30	650.25

表 13 顯示如何執行唯一權重排序順序的排序。將排序順序套用至 JOB 直欄中的值之後，各列就會排序。請注意，排序之後，小寫字母會排列在相同的大寫字母之前，而 'mgr'、'Mgr' 及 'MGR' 等值則會相鄰。

*表 13. 使用 ENU 語言 ID 唯一權重排序順序的 "SELECT * FROM STAFF ORDER BY JOB"。*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
80	James	20	Clerk	0	13504.60	128.20
100	Plotz	42	mgr	6	18352.80	0
10	Sanders	20	Mgr	7	18357.50	0
50	Hanes	15	Mgr	10	20659.80	0
30	Merenghi	38	MGR	5	17506.75	0
90	Koonitz	42	sales	6	18001.75	1386.70
20	Pernal	20	Sales	8	18171.25	612.45
40	OBrien	38	Sales	6	18006.00	846.55
70	Rothman	15	Sales	7	16502.83	1152.00
60	Quigley	38	SALES	0	16808.30	650.25

第 109 頁的表 14 顯示如何執行共用權重排序順序的排序。將排序順序套用至 JOB 直欄中的值之後，各列就會排序。在排序比較時，每一個小寫字母都當成對應的大寫字母一樣處理。請注意在第 109 頁的表 14 中，所有的 'MGR'、'mgr' 和 'Mgr' 值都混在一起。

*表 14. 使用 ENU 語言 ID 共用權重排序順序的 "SELECT * FROM STAFF ORDER BY JOB"。*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
80	James	20	Clerk	0	13504.60	128.20
10	Sanders	20	Mgr	7	18357.50	0
30	Merenghi	38	MGR	5	17506.75	0
50	Hanes	15	Mgr	10	20659.80	0
100	Plotz	42	mgr	6	18352.80	0
20	Pernal	20	Sales	8	18171.25	612.45
40	OBrien	38	Sales	6	18006.00	846.55
60	Quigley	38	SALES	0	16808.30	650.25
70	Rothman	15	Sales	7	16502.83	1152.00
90	Koonitz	42	sales	6	18001.75	1386.70

列選項

以下 SQL 陳述式會從 JOB 直欄中選取值爲 'MGR' 的列：

```
SELECT * FROM STAFF WHERE JOB='MGR'
```

表 15 顯示如何以 *HEX 排序順序執行列選項。在表 15 中，完全依照 SELECT 陳述式中的指定選取符合直欄 'JOB' 之列選取準則的列。只選取了大寫的 'MGR'。

*表 15. 使用 *HEX 排序順序的 "SELECT * FROM STAFF WHERE JOB='MGR'"。*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
30	Merenghi	38	MGR	5	17506.75	0

表 16 顯示如何使用唯一權重排序順序執行選項。在表 16 中，小寫和大寫字母都當成唯一來處理。小寫的 'mgr' 不會當成和大寫的 'MGR' 一樣。因此，不會選取小寫的 'mgr'。

*表 16. 使用 ENU 語言 ID 唯一權重排序順序的 "SELECT * FROM STAFF WHERE JOB = 'MGR'"。*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
30	Merenghi	38	MGR	5	17506.75	0

表 17 顯示如何使用共用權重排序順序執行選項。在表 17 中，大寫字母視爲和小寫字母一樣，選取符合直欄 'JOB' 之列選取準則的列。請注意，表 17 中選取了 'mgr'、 'Mgr' 和 'MGR' 所有值。

*表 17. 使用 ENU 語言 ID 共用權重排序順序的 "SELECT * FROM STAFF WHERE JOB = 'MGR'"。*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
10	Sanders	20	Mgr	7	18357.50	0
30	Merenghi	38	MGR	5	17506.75	0
50	Hanes	15	Mgr	10	20659.80	0
100	Plotz	42	mgr	6	18352.80	0

排序順序和概略表

概略表是利用 CREATE VIEW 陳述式執行時生效的排序順序建立。 FROM 子句中參照到概略表時，會使用該排序順序進行 CREATE VIEW 中子選擇的任何字元比較。這時候，會利用概略表子選擇產生中階結果表格。接著將查詢執行時生效的排序順序套用至查詢中指定的所有字元和 UCS-2 圖形比較 (包括牽涉到隱含轉換至字元或 UCS-2 圖形的比較)。

下列 SQL 陳述式和表格顯示概略表和排序順序如何工作。以下範例中使用的概略表 V1 是以 SRTSEQ(*LANGIDSHR) 和 LANGID(ENU) 的共用權重排序順序建立的。CREATE VIEW 陳述式如下：

```
CREATE VIEW V1 AS SELECT *
  FROM STAFF
  WHERE JOB = 'MGR' AND ID < 100
```

表 18 顯示概略表的結果表格。

*表 18. "SELECT * FROM V1"*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
10	Sanders	20	Mgr	7	18357.50	0
30	Merenghi	38	MGR	5	17506.75	0
50	Hanes	15	Mgr	10	20659.80	0

針對概略表 V1 執行的任何查詢都會針對表 18 中顯示的結果表格執行。以下所顯示的查詢是以 SRTSEQ(*LANGIDUNQ) 和 LANGID(ENU) 的排序順序執行。

*表 19. 使用 ENU 語言 ID 唯一權重排序順序的 "SELECT * FROM V1 WHERE JOB = 'MGR'"*

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
30	Merenghi	38	MGR	5	17506.75	0

排序順序和 CREATE INDEX 陳述式

索引是使用 CREATE INDEX 陳述式執行時生效的排序順序建立。每次在定義索引的表格中執行插入時，就會在索引中新增一個登錄。索引登錄包含字元鍵和 UCS-2 圖形鍵直欄的加權值。系統會根據索引的排序順序轉換索引鍵值，以取得加權值。

使用該排序順序和該索引進行選擇時，不必在比較前轉換字元或 UCS-2 圖形鍵。這增進了查詢的效能。如需建立有效索引和排序順序的詳細資訊，請參閱 *Database Performance and Query Optimization* 一書中的 Using indexesto speed access to large tables。

排序順序和限制

唯一限制是利用索引實作。如果加入唯一限制的表格是以排序順序定義，將會以相同的排序順序建立索引。

如果是定義參照限制，上層與相依表格之間的排序順序必須相符。如需排序順序和限制的詳細資訊，請參閱 iSeries 資訊中心裡 Database Programming 一書中的 Ensuring data integrity with referential constraints 主題。

定義核對限制時使用的排序順序，和系統在 INSERT 或 UPDATE 時驗證是否符合限制時使用的排序順序相同。

第 9 章 使用游標

當 SQL 執行 SELECT 陳述式時，由結果列組成結果表格。游標提供存取結果表格的方式。它在 SQL 程式內用來維護結果表格中的位置。 SQL 透過游標來使用結果表格中的列，且讓您的程式可以使用這些列。您的程式可以有多個游標，但每一個游標必須有唯一名稱。

與使用游標相關的陳述式如下：

- DECLARE CURSOR 陳述式，可定義及爲游標命名，以及指定要以內含的 SELECT 陳述式擷取列。
- OPEN 和 CLOSE 陳述式，可開啓及關閉在程式內使用的游標。游標必須在擷取任何列之前開啓。
- FETCH 陳述式，可擷取游標結果表格中的列或將游標定位在另一列。
- UPDATE ... WHERE CURRENT OF 陳述式，可更新游標的現行列。
- DELETE ... WHERE CURRENT OF 陳述式，可刪除游標的現行列。

有關這些陳述式的完整討論，請參閱 SQL Reference 一書。

有關游標的詳細資訊，請參閱下列主題：

- 『游標類型』
- 第 114 頁的『游標使用範例』
- 第 120 頁的『使用多列 FETCH 陳述式』
- 第 124 頁的『工作單元與開啓游標』

註: 有關程式碼範例的資訊，請參閱第 x 頁的『程式碼不保事項聲明』資訊。

游標類型

SQL 支援序列及可捲動的游標。游標類型決定游標使用的定位方法。詳細資訊，請參閱：

- 『序列游標』
- 第 114 頁的『可捲動的游標』

序列游標

序列游標是不以 SCROLL 關鍵字定義的游標。

以序列游標而言，游標每次 OPEN 時，只能一次提取結果表格一列。當開啓游標時，游標定位於結果表格中第一列之前。當發出 FETCH 後，游標移到結果表格的下一列。之後，該列就是現行列。如果指定了主變數 (使用 FETCH 陳述式的 INTO 子句)，SQL 會將現行列內容移到您程式的主變數中。

每次發出 FETCH 陳述式都會重複此順序，直到達到資料結尾 (SQLCODE = 100) 爲止。當達到資料結尾時，請關閉游標。達到資料結尾後，您便無法存取結果表格中任何列。若要重新使用序列游標，您必須先關閉游標，再重新發出 OPEN 陳述式。您絕對無法使用序列游標來備份。

可捲動的游標

對可捲動的游標而言，結果表格的列可重複提取。游標根據 FETCH 陳述式指定的定位選項在結果表格中移動。當開啓游標後，游標定位於結果表格中第一列之前。於發出 FETCH 時，游標移到結果表格中定位選項所指定的列。之後，該列就是現行列。如果指定了主變數 (使用 FETCH 陳述式的 INTO 子句)， SQL 會將現行列內容移到您程式的主變數中。 BEFORE 和 AFTER 定位選項無法指定主變數。

每次發出 FETCH 陳述式都會重複此順序。當資料結尾或資料開始發生狀況時，不需要關閉游標。定位選項可讓程式繼續提取表格中的列。

發出 FECCH 陳述式時，下列捲動選項用來定位選項。這些定位相對於結果表格中的現行游標位置。

NEXT	將游標定位在下一列。如果未指定任何定位，此爲預設值。
PRIOR	將游標定位在上一列。
FIRST	將游標定位在第一列。
LAST	將游標定位在最後一列。
BEFORE	將游標定位在第一列之前。
AFTER	將游標定位於最後一列之後。
CURRENT	不變更游標定位。
RELATIVE n	評估與游標現行位置相關的主變數或整數 *n*。例如，若 *n* 爲 -1，游標便會定位在結果表格的上一列。若 *n* 爲 +3，游標便會定位在現行列之後三列。

對可捲動的游標而言，表格結尾取決於下列所示：

```
FETCH AFTER FROM C1
```

一旦游標定位在表格結尾，程式便可使用 PRIOR 或 RELATIVE 捲動選項，從表格結尾處開始定位及提取資料。

游標使用範例

假設您的程式檢查部門 D11 中人員的相關資料。下列範例顯示您要併入程式中來定義和使用序列及可捲動的游標之 SQL 陳述式。這些游標可用來取得 CORPDATA.EMPLOYEE 表格中部門的相關資訊。

以序列游標範例而言，程式會處理表格中所有列、更新部門 D11 所有成員的工作，以及刪除其它部門的員工記錄。

表 20. 序列游標範例

序列游標 SQL 陳述式	說明於章節中
EXEC SQL **DECLARE** THISEMP **CURSOR FOR** WORKDEPT, JOB **FROM** CORPDATA.EMPLOYEE **FOR UPDATE OF** JOB END-EXEC.	第 116 頁的『步驟 1：定義游標』. **SELECT** EMPNO, LASTNAME,

表 20. 序列游標範例 (繼續)

序列游標 SQL 陳述式	說明於章節中
EXEC SQL **OPEN** THISEMP END-EXEC.	第 117 頁的『步驟 2：開啓游標』.
EXEC SQL **WHENEVER NOT FOUND** **GO TO** CLOSE-THISEMP END-EXEC.	第 118 頁的『步驟 3：指定到達資料結尾時要採取的動作』.
EXEC SQL **FETCH** THISEMP **INTO** :EMP-NUM, :NAME2, :DEPT, :JOB-CODE END-EXEC.	第 118 頁的『步驟 4：使用游標來擷取列』.
... 所有 D11 部門的員工， 更新 JOB 值： EXEC SQL **UPDATE** CORPDATA.EMPLOYEE **SET** JOB = :NEW-CODE **WHERE CURRENT OF** THISEMP END-EXEC. ... 接著，列印列。	第 119 頁的『步驟 5a：更新現行列』.
... 其他員工， 刪除列： EXEC SQL **DELETE FROM** CORPDATA.EMPLOYEE **WHERE CURRENT OF** THISEMP END-EXEC.	第 119 頁的『步驟 5b：刪除現行列』.
分支返回提取並處理下一列。	
CLOSE-THISEMP. EXEC SQL **CLOSE** THISEMP END-EXEC.	第 119 頁的『步驟 6:：關閉游標』.

對可捲動的游標範例而言，程式使用 RELATIVE 定位選項來取得部門 D11 的薪資代表範例。

表 21. 可捲動的游標範例

可捲動的游標 SQL 陳述式	說明於章節中
EXEC SQL **DECLARE** THISEMP **DYNAMIC SCROLL CURSOR FOR** **SELECT** EMPNO, LASTNAME, SALARY **FROM** CORPDATA.EMPLOYEE **WHERE** WORKDEPT = 'D11' END-EXEC.	『步驟 1：定義游標』.
EXEC SQL **OPEN** THISEMP END-EXEC.	第 117 頁的『步驟 2：開啓游標』.
EXEC SQL **WHENEVER NOT FOUND** **GO TO** CLOSE-THISEMP END-EXEC.	第 118 頁的『步驟 3：指定到達資料結尾時要採取的動作』.
...起始設定程式合計 薪資變數 EXEC SQL **FETCH RELATIVE** 3 **FROM** THISEMP **INTO** :EMP-NUM, :NAME2, :JOB-CODE END-EXEC. ...新增目前的薪資至 程式合計薪資 ...分支返回取得並 處理下一列。	第 118 頁的『步驟 4：使用游標來擷取列』.
...計算平均 薪資	
CLOSE-THISEMP. EXEC SQL **CLOSE** THISEMP END-EXEC.	第 119 頁的『步驟 6:：關閉游標』.

步驟 1：定義游標

若要定義結果表格使用游標來存取，請使用 DECLARE CURSOR 陳述式。

DECLARE CURSOR 陳述式可命名游標及指定 SELECT 陳述式。 SELECT 陳述式定義一組以概念方式組成結果表格的列。對序列游標而言，此陳述式看起來如下 (FOR UPDATE OF 子句爲可選用的)：

```
EXEC SQL
 DECLARE cursor-name CURSOR FOR
 SELECT column-1, column-2 ,...
   FROM table-name , ...
   FOR UPDATE OF column-2 ,...
END-EXEC.
```

對可捲動的游標而言，此陳述式看起來如下 (WHERE 子句爲可選用的)：

```
EXEC SQL
 DECLARE cursor-name DYNAMIC SCROLL CURSOR FOR
 SELECT column-1, column-2 ,...
   FROM table-name ,...
   WHERE column-1 = expression ...
END-EXEC.
```

此處所示的 SELECT 陳述式相當簡單。不過，您可以針對序列及可捲動的游標，在 DECLARE CURSOR 陳述式的 SELECT 陳述式中撰寫其它幾個不同類型的游標。

如果您打算更新識別表格 (指 FROM 子句中命名的表格) 任何或全部列中的任何直欄，請加入 FOR UPDATE OF 子句。它會命名您打算更新的每一個直欄。如果您沒有指定直欄名稱，但指定了 ORDER BY 或 FOR READ ONLY 子句，則在嘗試更新時會傳回負值 SQLCODE。如果您沒有指定 FOR UPDATE OF、FOR READ ONLY 或 ORDER BY 子句，而且結果表格不是唯讀的，則可更新指定表格的任何直欄。

您可以更新識別表格的直欄，即使它不是結果表格的一部份。在這個情況下，您不需要於 SELECT 陳述式中命名該直欄。當游標擷取內含您要更新的直欄值之列 (使用 FETCH) 時，您可以使用 UPDATE ... WHERE CURRENT OF 來更新該列。

例如，假設結果表格的每一列併入 CORPDATA.EMPLOYEE 表格中的 *EMPNO*、*LASTNAME* 及 *WORKDEPT* 直欄。如果您要更新 *JOB* 直欄 (CORPDATA.EMPLOYEE 表格每一列的其中一個直欄)，則即使 SELECT 陳述式中省略了 JOB，DECLARE CURSOR 陳述式還是應併入 FOR UPDATE OF JOB ...。

如果下列任何一項屬實，結果表格和游標是*唯讀*：

- 第一個 FROM 子句定義多個表格或概略表。
- 第一個 FROM 子句定義唯讀概略表。
- 第一個 FROM 子句定義使用者定義的表格函數。
- 第一個 SELECT 子句指定關鍵字 DISTINCT。
- 外部子選擇含有 GROUP BY 子句。
- 外部子選擇含有 HAVING 子句。
- 第一個 SELECT 子句含有直欄函數。
- SELECT 陳述式含有子查詢，例如外部子選擇和子查詢的基本物件爲相同表格。
- SELECT 陳述式含有 UNION 或 UNION ALL 運算子。
- SELECT 陳述式含有 ORDER BY 子句，但未指定 FOR UPDATE OF 子句和 DYNAMIC SCROLL。
- SELECT 陳述式併入 FOR READ ONLY 子句。
- 不以 DYNAMIC 來指定 SCROLL 關鍵字。
- SELECT 清單併入 DataLink 直欄，且未指定 FOR UPDATE OF 子句。
- 第一個子選擇需要暫時結果表格。
- SELECT 陳述式併入 FETCH FIRST *n* ROWS ONLY。

步驟 2：開啓游標

若要開始處理結果表格的列，請發出 OPEN 陳述式。當您的程式發出 OPEN 陳述式時，SQL 會處理 DECLARE CURSOR 陳述式內的 SELECT 陳述式，使用 SELECT 陳述式中指定的任何主變數現行值來識別一組列，稱爲結果表格。根據搜尋條件適合的延伸範圍，結果表格可包含零、一或多列。 OPEN 陳述式如下：

```
EXEC SQL
 OPEN cursor-name
END-EXEC.
```

步驟 3：指定到達資料結尾時要採取的動作

若要知道何時會到達結果表格結尾，可用值 100 來測試 SQLCODE 欄位，或用值 '02000' (亦即資料結尾) 來測試 SQLSTATE 欄位。當 FETCH 陳述式擷取到結果表格中最後一列且您的程式發出後續的 FETCH 時，便會發生此狀況。例如：

```
...
IF SQLCODE =100 GO TO DATA-NOT-FOUND.

or

IF SQLSTATE ='02000' GO TO DATA-NOT-FOUND.
```

此技術的選擇方案是撰寫 WHENEVER 陳述式。使用 WHENEVER NOT FOUND 會導致您程式 (發出 CLOSE 陳述式所在的位置) 另一部份的分支。WHENEVER 陳述式如下：

```
EXEC SQL
 WHENEVER NOT FOUND GO TO symbolic-address
END-EXEC.
```

每當游標用來提取列時，您的程式應預期到資料結尾狀況，而且應準備好處理發生的狀況。

當您使用序列游標且到達資料結尾時，每一個後續的 FETCH 陳述式傳回資料結尾狀況。您不能將游標定位在已處理過的列上。CLOSE 陳述式是可執行於游標上的唯一作業。

當您使用可捲動的游標且到達資料結尾時，結果表格仍可處理其它資料。您可以使用定位選項組合，將游標定位在結果表格的任何地方。您不需要在到達資料結尾時將游標 CLOSE。

步驟 4：使用游標來擷取列

若要將所選取列的內容移到您程式的主變數中，請使用 FETCH 陳述式。 DECLARE CURSOR 陳述式內的 SELECT 陳述式定義一些含有您程式所要直欄值的列。不過，SQL 不會爲您的應用程式擷取任何資料，除非發出 FETCH 陳述式。

當您的程式發出 FETCH 陳述式時，SQL 使用現行游標位置作爲起點，來尋找結果表格中所要求的列。這會將該列變更爲**現行列**。如果指定了 INTO 子句， SQL 會將現行列內容移到您程式的主變數中。每次發出 FETCH 陳述式都會重複此順序。

SQL 會維護現行列的位置 (亦即，游標指向現行列)，直到對游標發出下一個 FETCH 陳述式爲止。即使有 DELETE 陳述式，UPDATE 陳述式還是不會變更結果表格內現行列的位置。

序列游標 FETCH 陳述式如下：

```
EXEC SQL
 FETCH cursor-name
   INTO :host variable-1[, :host variable-2] ...
END-EXEC.
```

可捲動的游標 FETCH 陳述式如下：

```
EXEC SQL
 FETCH RELATIVE integer
   FROM cursor-name
   INTO :host variable-1[, :host variable-2] ...
END-EXEC.
```

步驟 5a：更新現行列

當您的程式將游標定位在某列時，您可以使用含有 WHERE CURRENT OF 子句的 UPDATE 陳述式來更新其資料。WHERE CURRENT OF 子句指定一個游標指向您要更新的列。UPDATE ... WHERE CURRENT OF 陳述式如下：

```
EXEC SQL
 UPDATE table-name
   SET column-1 = value [, column-2 = value] ...
   WHERE CURRENT OF cursor-name
END-EXEC.
```

當 UPDATE 陳述式與游標一起使用時，它可以：

- 只能更新一列，就是現行列
- 定義指向要更新的列之游標
- 如果也指定了 ORDER BY 子句，則需要先在 DECLARE CURSOR 陳述式的 FOR UPDATE OF 子句中指定更新的直欄

更新某列之後，除非對下一列發出 FETCH 陳述式，否則游標的位置仍在該列上 (亦即，游標未改變的現行列)。

步驟 5b：刪除現行列

當您的程式已擷取現行列時，您可以使用 DELETE 陳述式來刪除該列。做法是，發出設計來與游標一起使用的 DELETE 陳述式；WHERE CURRENT OF 子句指定游標指向您要刪除的列。 DELETE ... WHERE CURRENT OF 陳述式如下：

```
EXEC SQL
 DELETE FROM table-name
   WHERE CURRENT OF cursor-name
END-EXEC.
```

當 DELETE 陳述式與游標一起使用時：

- 只能刪除一列，就是現行列
- 使用 WHERE CURRENT OF 子句來辨識游標所指向要刪除的列

刪除某列之後，除非發出 FETCH 陳述式來定位游標，否則您無法使用該游標來更新或刪除另一列。

第 95 頁的『使用 DELETE 陳述式移除表格中的列』顯示如何使用 DELETE 陳述式來刪除符合特定搜尋條件的所有列。當您想要取得列的副本、檢查它，然後刪除它，也可以使用 FETCH 和 DELETE ... WHERE CURRENT OF 陳述式。

步驟 6:：關閉游標

如果您針對序列游標處理結果表格的列，而且想要重新使用該游標，請在重新開啓該游標之前發出 CLOSE 陳述式來關閉它。

```
EXEC SQL
 CLOSE cursor-name
END-EXEC.
```

如果您處理了結果表格的列但不想要重新使用該游標，您可以讓系統關閉游標。在下列情況下，系統會自動關閉游標：

- 發出了不含 HOLD 陳述式的 COMMIT，且游標不是使用 WITH HOLD 子句來宣告。
- 發出了不含 HOLD 陳述式的 ROLLBACK。
- 工作結束。
- 啓動群組結束且在前置編譯上指定了 CLOSQLCSR(*ENDACTGRP)。
- 呼叫堆疊中第一個 SQL 程式結束，且在程式經過前置編譯後未指定 CLOSQLCSR(*ENDJOB) 或 CLOSQLCSR(*ENDACTGRP)。
- 使用 DISCONNECT 陳述式結束了與應用程式伺服器的連線。
- 釋放了與應用程式伺服器的連線且 COMMIT 順利完成。
- 發生了 *RUW CONNECT。

  由於開啓游標仍在參照的表格或概略表上保留鎖定，所以您應該關閉任何不再需要的開啓游標。

使用多列 FETCH 陳述式

多列 FETCH 陳述式可利用單一 FETCH 來擷取表格或概略表中的許多列。程式藉由對 FETCH 陳述式要求的列數來控制列的區塊 (OVRDBF 沒有影響)。對單一提取呼叫可要求的最大列數爲 32767。一旦擷取資料，游標便會定位在所擷取的最後一列。

定義被提取列所在的儲存體有兩個方法：具相關描述子的列儲存區或主電腦結構陣列。這兩個方法可利用 SQL 前置編譯器支援的所有語言來撰寫，但使用 REXX 的主電腦結構陣列除外。有關程式設計語言的詳細資訊，請參閱 SQL Programming with Host Languages 資訊。多列 FETCH 陳述式的兩個格式允許應用程式撰寫個別的指示器陣列。對可爲空值的每個主變數而言，指示器陣列應包含一個指示器。

多列 FETCH 陳述式可與序列及可捲動的游標一起使用。用來定義、開啓及關閉多列 FETCH 的游標之作業仍相同。僅變更 FETCH 陳述式來指定擷取列數及存放這些列的儲存體。

在每個多列 FETCH 之後，會透過 SQLCA 將資訊傳回程式。除 SQLCODE 和 SQLSTATE 欄位之外，SQLERRD 提供了下列資訊：

- SQLERRD3 含有對多列 FETCH 陳述式擷取的列數。如果 SQLERRD3 小於所要求的列數，則會發生錯誤或資料結尾狀況。
- SQLERRD4 含有所擷取每一列的長度。
- SQLERRD5 含有已提取表格中最後一列的指示。當游標未立即感受到更新時，它可用來偵測所提取表格中的資料結尾狀況。確實立即感受到更新的游標應繼續提取，直到接收了 SQLCODE +100 來偵測資料結尾狀況爲止。

使用主電腦結構陣列的多列 FETCH

若要搭配主電腦結構陣列來使用多列 FETCH，應用程式必須定義可供 SQL 使用的主電腦結構陣列。每一種語言都有自己定義主電腦結構陣列的慣例和規則。主電腦結構陣列可透過變數宣告來定義，或藉由使用編譯器指引擷取「外部檔案說明」 (例如 COBOL COPY 指引) 來定義。

主電腦結構陣列由結構陣列組成。每一個結構對應到結果表格的一列。陣列中第一個結構對應列至第一列，陣列中第二個結構對應至第二列...等等，以此類推。 SQL 根據主

電腦結構陣列宣告來決定主電腦結構陣列中基本項目的屬性。若要擁有最大效能，組成主電腦結構陣列的項目屬性應符合擷取的直欄屬性。

考慮下列 COBOL 範例：

```
      EXEC SQL INCLUDE SQLCA
      END-EXEC.

...

      01 TABLE-1.
          02 DEPT  OCCURS 10 TIMES.
             05  EMPNO   PIC   X(6).
             05 LASTNAME.
                49 LASTNAME-LEN  PIC S9(4) BINARY.
                49 LASTNAME-TEXT PIC X(15).
             05  WORKDEPT PIC  X(3).
             05  JOB      PIC  X(8).
      01 TABLE-2.
         02 IND-ARRAY   OCCURS 10 TIMES.
            05 INDS PIC  S9(4) BINARY OCCURS 4 TIMES.

...
      EXEC SQL
      DECLARE D11 CURSOR FOR
      SELECT EMPNO, LASTNAME, WORKDEPT, JOB
       FROM CORPDATA.EMPLOYEE
        WHERE WORKDEPT = "D11"
      END-EXEC.

...

      EXEC SQL
       OPEN D11
      END-EXEC.
      PERFORM FETCH-PARA UNTIL SQLCODE NOT EQUAL TO ZERO.
  ALL-DONE.
        EXEC SQL CLOSE D11 END-EXEC.

...

      FETCH-PARA.
         EXEC SQL WHENEVER NOT FOUND GO TO ALL-DONE END-EXEC.
      EXEC SQL FETCH D11 FOR 10 ROWS INTO :DEPT :IND-ARRAY
      END-EXEC.

...
```

在本範例中，爲 CORPDATA.EMPLOYEE 表格定義游標以選取 WORKDEPT 直欄等於 'D11' 的所有列。結果表格含有 8 列。 DECLARE CURSOR 和 OPEN 陳述式與多列 FETCH 陳述式一起使用時，沒有任何特殊語法。可在程式其他地方撰寫針對相同游標傳回單一列的另一個 FETCH 陳述式。多列 FETCH 陳述式用來擷取結果表格中所有的列。遵循 FETCH 之後，游標位置仍在擷取的最後一列。

主電腦結構陣列 DEPT 及相關的指示器陣列 IND-ARRAY 皆定義於應用程式中。這兩個陣列的維度爲 10。指示器陣列具有結果表格中每一直欄的登錄。

DEPT 主電腦結構陣列基本項目的類型和長度屬性符合擷取的直欄。

當多列 FETCH 陳述式順利完成後，主電腦結構陣列包含全部 8 列的資料。指示器陣列 IND_ARRAY 包含代表每一列中每一直欄的 0，因爲沒有傳回 NULL 值。

傳回應用程式的 SQLCA 含有下列資訊：

- SQLCODE 含有 0
- SQLSTATE 含有 '00000'
- SQLERRD3 含有 8 (指提取的列數)
- SQLERRD4 含有 34 (指每一列的長度)
- SQLERRD5 含有 +100 (指結果表格中的最後一列位於區塊中)

有關 SQLCA 的說明，請參閱 SQL Reference 一書的 Appendix B。

使用列儲存區的多列 FETCH

應用程式必須先定義列儲存區及相關的說明區，才能搭配列儲存區使用多列 FETCH。列儲存區是應用程式中定義的主變數。列儲存區含有多列 FETCH 的結果。列儲存區可以是具有足夠位元組數來保留多列 FETCH 上要求的所有列之字元變數。

SQLDA 含有每個所傳回直欄的 SQLTYPE 和 SQLLEN，它由多列 FETCH 的列儲存區套表上使用的相關描述子來定義。描述子中提供的資訊可決定從資料庫對映到列儲存區的資料。若要擁有最大效能，描述子中的資訊應符合擷取的直欄屬性。

有關 SQLDA 的說明，請參閱 SQL Reference 一書的 Appendix C。

參考下列 PL/I 範例：

```
      *....+....1....+....2....+....3....+....4....+....5....+....6....+....7...*
     EXEC SQL INCLUDE SQLCA;
     EXEC SQL INCLUDE SQLDA;

...

    DCL DEPTPTR PTR;
    DCL 1 DEPT(20) BASED(DEPTPTR),
        3 EMPNO CHAR(6),
        3 LASTNAME CHAR(15) VARYING,
        3 WORKDEPT CHAR(3),
        3 JOB CHAR(8);
    DCL I BIN(31) FIXED;
    DEC J BIN(31) FIXED;
    DCL ROWAREA CHAR(2000);

...

    ALLOCATE SQLDA SET(SQLDAPTR);
    EXEC SQL
     DECLARE D11 CURSOR FOR
     SELECT EMPNO, LASTNAME, WORKDEPT, JOB
     FROM CORPDATA.EMPLOYEE
     WHERE WORKDEPT = 'D11';
```

圖 1. 使用列儲存區的多列 FETCH 範例 (1/2)

```
...
    EXEC SQL
      OPEN D11;
    /* SET UP THE DESCRIPTOR FOR THE MULTIPLE-ROW FETCH */
    /* 4 COLUMNS ARE BEING FETCHED                 */
    SQLD = 4;
    SQLN = 4;
    SQLDABC = 366;
    SQLTYPE(1) = 452; /* FIXED LENGTH CHARACTER -  */
                      /* NOT NULLABLE              */
    SQLLEN(1) = 6;
    SQLTYPE(2) = 456; /*VARYING LENGTH CHARACTER   */
                      /* NOT NULLABLE              */
    SQLLEN(2) = 15;
    SQLTYPE(3) = 452; /* FIXED LENGTH CHARACTER -  */
    SQLLEN(3) = 3;
    SQLTYPE(4) = 452; /* FIXED LENGTH CHARACTER -  */
                      /* NOT NULLABLE              */
    SQLLEN(4) = 8;
    /*ISSUE THE MULTIPLE-ROW FETCH STATEMENT TO RETRIEVE*/
    /*THE DATA INTO THE DEPT ROW STORAGE AREA      */
    /*USE A HOST VARIABLE TO CONTAIN THE COUNT OF  */
    /*ROWS TO BE RETURNED ON THE MULTIPLE-ROW FETCH     */

    J = 20;        /*REQUESTS 20 ROWS ON THE FETCH */
   ...
    EXEC SQL
      WHENEVER NOT FOUND
      GOTO FINISHED;
    EXEC SQL
      WHENEVER SQLERROR
      GOTO FINISHED;
    EXEC SQL
      FETCH D11 FOR :J ROWS
      USING DESCRIPTOR :SQLDA INTO :ROWAREA;
    /* ADDRESS THE ROWS RETURNED                   */
    DEPTPTR = ADDR(ROWAREA);
    /*PROCESS EACH ROW RETURNED IN THE ROW STORAGE */
    /*AREA BASED ON THE COUNT OF RECORDS RETURNED  */
    /*IN SQLERRD3.                                 */
    DO I = 1 TO SQLERRD(3);
     IF EMPNO(I) = '000170'  THEN
       DO;
      :
       END;
    END;
    IF SQLERRD(5) = 100 THEN
      DO;
        /* PROCESS END OF FILE  */
      END;
    FINISHED:
```

圖 1. 使用列儲存區的多列 FETCH 範例 (2/2)

在本範例中，爲 CORPDATA.EMPLOYEE 表格定義游標來選取 WORKDEPT 直欄等於 'D11' 的所有列。附錄 A, 『DB2 UDB for iSeries 表格範例』中的範例 EMPLOYEE 表格顯示含有多列的結果表格。DECLARE CURSOR 和 OPEN 陳述式與多列 FETCH 陳述式一起使用時，沒有特殊語法。可在程式其他地方撰寫針對相同游標傳回單一列的另一個 FETCH 陳述式。多列 FETCH 陳述式用來擷取結果表格中所有的列。遵循 FETCH 之後，游標位置仍在區塊中最後一列。

列區域 ROWAREA 定義爲字元陣列。結果表格中的資料存放在主變數中。在本範例中，指標變數指定給 ROWAREA 的位址。檢查所傳回列中每一個項目，並與基本結構 DEPT 一起使用。

描述子中項目的屬性 (類型和長度) 符合擷取的直欄。在這個情況下，不提供指示器區域。

完成 FETCH 陳述式之後，ROWAREA 含有等於 'D11' 的所有列，這裡是指 11 列。傳回應用程式的 SQLCA 含有下列資訊：

- SQLCODE 含有 0
- SQLSTATE 含有 '00000'
- SQLERRD3 含有 11 (指傳回的列數)
- SQLERRD4 含有 34 (指提取的列長度)
- SQLERRD5 含有 +100 (指提取結果表格中之最後一列)

在本範例中，應用程式利用 SQLERRD5 含有到達檔案結尾的指示之事實。因此，應用程式不需要重新呼叫 SQL 來嘗試擷取其它列。如果游標能立即感受到插入，萬一有新增任何記錄，您應該呼叫 SQL。當確定控制層次是 *RR 以外的項目，游標會立即感受到。

工作單元與開啓游標

當程式完成工作單元時，應確定或回轉所做的變更。除非您對 COMMIT 或 ROLLBACK 陳述式指定了 HOLD，否則 SQL 會自動關閉所有開啓游標。以 WITH HOLD 子句宣告的游標不會對 COMMIT 自動關閉。它們會自動對 ROLLBACK 關閉 (忽略 DECLARE CURSOR 陳述式上指定的 WITH HOLD 子句)。

在 COMMIT 或 ROLLBACK 之後，如果您要繼續從現行游標位置處理，則必須指定 COMMIT HOLD 或 ROLLBACK HOLD。當指定了 HOLD 後，任何開啓游標保持開啓，並且保留游標位置，以便處理程序可回復。在 COMMIT 陳述式上，游標位置受到維護。在 ROLLBACK 陳述式上，游標位置復置到先前工作單元中擷取的最後一列之後。所有記錄鎖定仍被釋放。

發出不含 HOLD 的 COMMIT 或 ROLLBACK 陳述式後，會釋放所有鎖定並關閉所有游標。您可以重新開啓游標，但要從結果表格的第一列開始處理。

註: CRTSQLxxx 指令的 ALWBLK(*ALLREAD) 參數規格可變更唯讀游標的游標位置還原。有關 ALWBLK 參數用法及 CRTSQLxxx 指令上其它效能相關選項的資訊，請參閱第 14 章, 『動態 SQL 應用程式』。

有關確定控制及工作單元的詳細資訊，請參閱「確定控制」主題。

第 10 章 資料完整性

資料完整性旨在確保綱目中各表格間的資料值，使之處於合理的狀態下。例如，若一銀行在表格 A 中存有客戶清單，並於表格 B 中存放了客戶帳戶清單，設若表格 B 中新增了帳戶但表格 A 中卻無相關的客戶存在，此情況即不合理。

本章將說明系統在自動強制要求此類關係時所採用的不同方式。參照完整性、核對限制及觸發正是達成資料完整性的主要憑藉。此外，CREATE VIEW 的 WITH CHECK OPTION 子句亦可限制概略表的資料插入或更新方式。相關詳細資訊，請參閱以下主題：

- 『新增與使用核對限制』
- 第 126 頁的『參照完整性』
- 第 133 頁的『概略表上的 WITH CHECK OPTION』
- 第 136 頁的『DB2 UDB for iSeries 觸發支援』

如需關於資料完整性的完整說明，請參閱 Database Programming 一書。

新增與使用核對限制

*核對限制*會藉由限制直欄或直欄群組中的容許值，來確保進行讀取和插入及更新時的資料有效性。您可以用 SQL CREATE TABLE 與 ALTER TABLE 陳述式來新增或捨棄核對限制。

本例中，下列陳述式將建立含有三個直欄的表格，並對 COL2 設置核對限制以使該欄的容許值只限正整數：

```
CREATE TABLE T1 (COL1 INT, COL2 INT CHECK (COL2>0), COL3 INT)
```

對此表格而言，下列陳述式：

```
INSERT INTO T1 VALUES (-1, -1, -1)
```

將會失敗，因爲要插入到 COL2 的值不符核對限制的要求；亦即，-1 不大於 0。

但下列陳述式則會成功：

```
INSERT INTO T1 VALUES (1, 1, 1)
```

一旦插入該列後，下列陳述式將失敗：

```
ALTER TABLE T1 ADD CONSTRAINT C1 CHECK (COL1=1 AND COL1<COL2)
```

此 ALTER TABLE 陳述式嘗試新增第二個核對限制，以便將 COL1 的容許值限定爲 1，同時有效規定 COL2 的值須大於 1。但此限制將不被允許，因爲其中的第二部份與現有資料 (COL2 中的 '1' 並未小於 COL1 中的 '1') 不符。

參照完整性

參照完整性是指資料庫中一組表格的狀況，在此狀況下各表格彼此間的參照皆屬有效。

試考慮下列範例：(這些範例表格列於附錄 A, 『DB2 UDB for iSeries 表格範例』中：

- CORPDATA.EMPLOYEE 爲員工的主要清單。
- CORPDATA.DEPARTMENT 爲所有部門編號的主要清單。
- CORPDATA.EMP_ACT 提供相關專案活動的主要清單。

其他表格則參考這些表格中所說明的實體。當某一表格所含的資料存有主要清單時，該資料應實際出現在主要清單中，否則參照即無效。內含主要清單的表格爲*上層表格*，而參照此類表格者則爲*相依表格*。若相依表格對上層表格的參照爲有效，此表格集合的狀況即稱爲*參照完整性*。

換言之，參照完整性乃指資料庫中所有外來索引鍵值皆有效的狀態。而每一個外來索引鍵值亦必須存在於上層鍵中，或必須是空值。要瞭解此種參照完整性的定義，須先瞭解下列術語：

- *唯一鍵*是表格中的一個直欄或一組直欄，用以作爲某列的唯一識別。表格雖可擁有數個唯一鍵，但任兩列不得擁有相同的唯一鍵值。
- *主要索引鍵*是不允許爲空值的唯一鍵。表格不得擁有一個以上的主要索引鍵。
- *上層鍵*是指某一參照限制中被參考到的唯一鍵或主要索引鍵。
- *外來索引鍵*是某一直欄或一組直欄，且其值與上層鍵的相符。若用來建立外來索引鍵的任何直欄值爲空值，則此規則即不適用。
- *上層表格*是含有上層鍵的表格。
- *相依表格*是含有外來索引鍵的表格。
- *衍生表*是相依表格或相依表格的相依項。

強制要求參照完整性可避免違反每一非空值外來索引鍵須具有相符上層鍵的規則。

有關參照完整性的詳情，請參閱下列主題：

- 『新增或捨棄參照限制』
- 第 128 頁的『移除參照限制』
- 第 128 頁的『插入至內含參照限制的表格』
- 第 129 頁的『更新含有參照限制的表格』
- 第 130 頁的『從含有參照限制的表格中進行刪除』
- 第 133 頁的『核對擱置』

SQL 係以 CREATE TABLE 與 ALTER TABLE 等陳述式來支援參照完整性的概念。有關此類指令的詳細說明，請參閱 SQL Reference 一書。您也可利用 iSeries 領航員，在建立表格時新增限制，或將限制新增至現有表格中。

新增或捨棄參照限制

限制爲用來確保表格、相依表格、對另一表格之資料、上層表格的參照皆屬有效的規則。您可以使用參照限制來確保「參照完整性」。

使用 SQL CREATE TABLE 與 ALTER TABLE 陳述式來新增或變更參照限制。

透過參照限制，非空值的外來索引鍵只有在同時亦爲上層鍵的值時才屬有效。定義參照限制時，您必須指定：

- 主要索引鍵或唯一鍵
- 外來索引鍵
- 刪除及更新在上層列遭刪除或更新時，用來指定須就相依列所採取之動作的相關規則。

另外，您也可爲限制指定名稱。若未指定名稱，系統會自動代爲產生。

一旦定義了參照限制後，系統即會對透過 SQL 或任何其他介面 (包括「iSeries 領航員」、CL 指令、公用程式或高階語言陳述式) 所執行的每一 INSERT、DELETE 及 UPDATE 作業，來強制施行此限制。

範例：新增參照限制

要求範例員工表格所示的每一個部門編號亦須出現在部門表格中的規則，即爲一種參照限制。此限制可確保每位員工皆隸屬某一現有部門。下列 SQL 陳述式將會以所定義的這些限制關係來建立 CORPDATA.DEPARTMENT 與 CORPDATA.EMPLOYEE 表格。

```
CREATE TABLE CORPDATA.DEPARTMENT
      (DEPTNO    CHAR(3)     NOT NULL PRIMARY KEY,
       DEPTNAME  VARCHAR(29) NOT NULL,
       MGRNO     CHAR(6),
       ADMRDEPT  CHAR(3)     NOT NULL
                             CONSTRAINT REPORTS_TO_EXISTS
                               REFERENCES CORPDATA.DEPARTMENT (DEPTNO)
                               ON DELETE CASCADE)

CREATE TABLE CORPDATA.EMPLOYEE
      (EMPNO     CHAR(6)     NOT NULL PRIMARY KEY,
       FIRSTNAME VARCHAR(12) NOT NULL,
       MIDINIT   CHAR(1)     NOT NULL,
       LASTNAME  VARCHAR(15) NOT NULL,
       WORKDEPT  CHAR(3)     CONSTRAINT WORKDEPT_EXISTS
                               REFERENCES CORPDATA.DEPARTMENT (DEPTNO)
                               ON DELETE SET NULL ON UPDATE RESTRICT,
       PHONENO   CHAR(4),
       HIREDATE  DATE,
       JOB       CHAR(8),
       EDLEVEL   SMALLINT    NOT NULL,
       SEX       CHAR(1),
       BIRTHDATE DATE,
       SALARY    DECIMAL(9,2),
       BONUS     DECIMAL(9,2),
       COMM      DECIMAL(9,2),
       CONSTRAINT  UNIQUE_LNAME_IN_DEPT UNIQUE (WORKDEPT, LASTNAME))
```

此時，DEPARTMENT 表格將擁有作爲主要索引鍵的唯一部門編號 (DEPTNO) 的直欄，且該表格爲下列兩項限制關係中的上層表格：

REPORTS_TO_EXISTS

一種自行參照限制，其間 DEPARTMENT 表格將是同一關係中的上層與相依項。ADMRDEPT 的每個非空值須與 DEPTNO 的值相符。部門須隸屬於資料庫中的現有部門。DELETE CASCADE 規則指出若刪除 DEPTNO 值爲 *n* 的列，則表格中 ADMRDEPT 爲 *n* 的每一列亦會遭刪除。

WORKDEPT_EXISTS

會建立 EMPLOYEE 表格來作爲相依表格，且以員工部門分派 (WORKDEPT) 直欄作爲外來索引鍵。因此，每一 WORKDEPT 值須與 DEPTNO 值相符。

DELETE SET NULL 規則指出若從 DEPTNO 爲 *n* 的 DEPARTMENT 中刪除某一列，則 EMPLOYEE 中具有 *n* 值的每一列，其 WORKDEPT 值將設定爲空值。而 UPDATE RESTRICT 規則規定若 EMPLOYEE 中存有與現行 DEPTNO 值相符的 WORKDEPT 值時，即不得更新 DEPARTMENT 中的 DEPTNO 值。

EMPLOYEE 表格中的 UNIQUE_LNAME_IN_DEPT 限制會使姓氏在部門中具唯一性。但此限制則不同，它呈現了如何在表格層次定義由數個直欄組成的限制。

移除參照限制

ALTER TABLE 陳述式可用來針對表格一次新增或捨棄一個限制。如果所捨棄的限制爲某些參照限制關係中的上層鍵，則此上層檔案與任何相依檔間的限制亦會被移除。

DROP TABLE 與 DROP SCHEMA 陳述式也可針對遭移除的表格或綱目，移除其上的限制。

範例：移除限制

下列範例會在表格 DEPARTMENT 中移除 DEPTNO 直欄的主要索引鍵。而分別在表格 DEPARTMENT 與 EMPLOYEE 中所定義的限制 REPORTS_TO_EXISTS 與 WORKDEPT_EXISTS 也會遭移除，因爲所移除的主要索引鍵是這些限制關係中的上層鍵。

```
ALTER TABLE CORPDATA.EMPLOYEE DROP PRIMARY KEY
```

您還可依名稱來移除限制，如下述範例所示：

```
ALTER TABLE CORPDATA.DEPARTMENT
            DROP CONSTRAINT UNIQUE_LNAME_IN_DEPT
```

插入至內含參照限制的表格

將資料插入至含有參照限制的表格時，有一些重要事項務必要牢記。如果是將資料插入至含有上層鍵的上層表格，SQL 將不允許：

- 重複的上層鍵值
- 若上層鍵爲主要索引鍵，任何主要索引鍵的直欄存有空值

如果是將資料插入至含有外來索引鍵的相依表格：

- 插入至外來索引鍵直欄中的每個非空值，須等於上層表格之對應上層鍵中的部份值。
- 若外來索引鍵中有任何直欄爲空值，整個外來索引鍵即被視爲空值。如果含有該直欄的所有外來索引鍵皆爲空值，INSERT 即會成功 (只要不違反鍵值唯一的索引)。

範例：插入有限制的資料

請變更範例應用程式專案表格 (PROJECT)，以定義下列兩種外來索引鍵：

- 部門編號的外來索引鍵 (DEPTNO)，其參照部門表格
- 員工編號的外來索引鍵 (RESPEMP)，其參照員工表格。

```
ALTER TABLE CORPDATA.PROJECT ADD CONSTRAINT RESP_DEPT_EXISTS
            FOREIGN KEY (DEPTNO)
            REFERENCES CORPDATA.DEPARTMENT
            ON DELETE RESTRICT
```

```
ALTER TABLE CORPDATA.PROJECT ADD CONSTRAINT RESP_EMP_EXISTS
            FOREIGN KEY (RESPEMP)
            REFERENCES CORPDATA.EMPLOYEE
            ON DELETE RESTRICT
```

請注意，REFERENCES 子句中並未指定上層表格直欄。只要被參考到的表格具有主要索引鍵或者可作爲上層鍵的適當唯一鍵，此類直欄即不須指定。

插入至 PROJECT 表格的每一列都須具有一 DEPTNO 值，且此值須相等於部門表格中的部份 DEPTNO 值。 (不允許爲空值，因爲專案表格中的 DEPTNO 已定義爲 NOT NULL。) 該列還須具有 RESPEMP 值，且此值須等於員工表格中的部份 EMPNO 值或爲空值。

附錄 A, 『DB2 UDB for iSeries 表格範例』 中所示帶有範例資料的表格即符合此類限制。下述 INSERT 陳述式則會失敗，因爲 DEPARTMENT 表格中並無相符的 DEPTNO 值 ('A01')。

```
INSERT INTO  CORPDATA.PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
       VALUES  ('AD3120', 'BENEFITS ADMIN', 'A01', '000010')
```

同理，下述 INSERT 陳述式也不會成功，因爲 EMPLOYEE 表格中並無 '000011' 的 EMPNO 值。

```
INSERT INTO CORPDATA.PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
       VALUES ('AD3130', 'BILLING', 'D21', '000011')
```

但下述 INSERT 陳述式則會順利完成，因爲 DEPARTMENT 表格中有相符的 DEPTNO 值 'E01'，且 EMPLOYEE 表格中也有相符的 EMPNO 值 '000010'。

```
INSERT INTO CORPDATA.PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
       VALUES ('AD3120', 'BENEFITS ADMIN', 'E01', '000010')
```

更新含有參照限制的表格

如果正在更新*上層*表格，您不得修改存有對應相依列的主要索引鍵。變更此索引鍵將會違反相依表格的參照限制，進而使得某些列沒有上層。此外，您亦不得將任何部份的主要索引鍵訂爲空值。

更新規則

對上層表格執行 UPDATE 時，相依表格的相應動作將取決於針對參照限制所指定的更新規則。若未針對參照限制定義任何更新規則，系統即會使用 UPDATE NO ACTION 規則。

UPDATE NO ACTION

指定若無其他相依列時，上層表格中的列即可更新。如果在關係中存有相依列，UPDATE 將會失敗。相依列的檢查執行於陳述式結尾處。

UPDATE RESTRICT

指定若無其他相依列時，上層表格中的列即可更新。如果在關係中存有相依列，UPDATE 將會失敗。立即執行相依列的檢查。

要瞭解 RESTRICT 規則與 NO ACTION 規則之間的些微差異，最快的方法是觀察觸發與參照限制的互動。觸發可定義在某項作業 (此處即爲 UPDATE 陳述式) 之前或之後啓動。*觸發之前*是在執行 UPDATE 之前即啓動，因此會早於任何限制檢查。*觸發之後*則是在執行 UPDATE 後才啓動，因此會晚於 RESTRICT 限制規則 (立即執行檢查) 但早於 NO ACTION 限制規則 (於陳述式末尾處執行檢查)。觸發與規則的發生順序如下所述：

1. *觸發之前*會在 UPDATE 之前啓動，且會早於 RESTRICT 或 NO ACTION 限制規則。
2. *觸發之後*會在 RESTRICT 限制規則之後啓動，但早於 NO ACTION 規則。

如果是更新*相依表格*，您所變更的任何非空值之外來索引鍵值，須與各個關係 (該表格爲其中的相依項者) 的主要索引鍵相符。例如，員工表格中的部門編號即相依於部門表格中的部門編號。您可將某一員工指派至無部門 (空值)，但不能指派至不存在的部門。

若針對內含參照限制的表格執行 UPDATE 失敗時，更新作業期間所作的全部變更將會還原。有關在處理限制時其確定控制與日誌登載之含義的詳情，請參閱第 285 頁的『日誌登載』 與第 286 頁的『確定控制』。

範例：UPDATE 規則

例如，若部門表格中的某一部門編號仍負責一些專案您即不得加以變更，而其間的關係是由專案表格中的相依列來說明。

下述 UPDATE 將會失敗，因爲 PROJECT 表格具有相依於 DEPARTMENT.DEPTNO 且值爲 'D01' 的列 (由 WHERE 陳述式所描述的列)。如果此 UPDATE 被允許，則 PROJECT 與 DEPARTMENT 表格之間的參照限制將會被打破。

```
UPDATE CORPDATA.DEPARTMENT
        SET DEPTNO = 'D99'
        WHERE DEPTNAME = 'DEVELOPMENT CENTER'
```

下述陳述式同樣也會失敗，因爲它違反了主要索引鍵 DEPTNO (於 DEPARTMENT 中) 與外來索引鍵 DEPTNO (於 PROJECT 中) 之間的參照限制：

```
UPDATE CORPDATA.PROJECT
        SET DEPTNO = 'D00'
        WHERE DEPTNO = 'D01';
```

此陳述式嘗試將所有部門編號 D01 變更爲部門編號 D00。由於 D00 並非主要索引鍵 DEPTNO 在 DEPARTMENT 中的值，所以此陳述式將失敗。

從含有參照限制的表格中進行刪除

若一表格具有主要索引鍵但並無相依項時，DELETE 的操作方式即如同其確無參照限制。若表格只有外來索引鍵而無主要索引鍵時情況亦同。若表格具有主要索引鍵與相依表格，DELETE 則會根據指定的刪除規則來進行刪除或更新。要使刪除作業成功，必須滿足所有受影響關係的全部刪除規則。若違反了參照限制，DELETE 即會失敗。

對上層表格執行 DELETE 時，相依表格的相應動作將取決於針對參照限制所指定的刪除規則。若未定義任何刪除規則，系統將會使用 DELETE NO ACTION 規則。

DELETE NO ACTION

指定若無其他相依列時，上層表格中的列即可更新。如果在關係中存有相依列，DELETE 將會失敗。相依列的檢查是在陳述式結尾處執行。

DELETE RESTRICT

指定若無其他相依列時，上層表格中的列即可更新。如果在關係中存有相依列，DELETE 將會失敗。立即執行相依列的檢查。

例如，若部門表格中的某一部門編號仍負責一些專案 (由專案表格中的相依列來說明) 時即不得加以刪除。

DELETE CASCADE

指定首先刪除上層表格中的指定列。然後，再刪除相依列。

例如，您可以藉由刪除部門表格中的相關列，來刪除某一部門。從部門表格中刪除相關的列也會一併刪除：

- 隸屬於其下之所有部門的列
- 隸屬此類部門下之所有部門，依此類推。

DELETE SET NULL

指定各相依列中可爲空值的外來索引鍵直欄將會設成其預設值。此即表示該直欄唯有在屬於外來索引鍵 (參照所刪除之列者) 之成員時，才會設成預設值。其間，唯有立即衍生的相依列會受影響。

DELETE SET DEFAULT

指定各相依列中的各個外來索引鍵直欄將會設成其預設值。此即表示該直欄唯有在屬於外來索引鍵 (參照所刪除之列者) 之成員時，才會設成預設值。其間，唯有立即衍生的相依列會受影響。

例如，即使某一員工管理數個部門，您仍可從員工表格 (EMPLOYEE) 中加以刪除。在此情況下，隸屬於經理下的各個員工之 MGRNO 值會在部門表格 (DEPARTMENT) 中被設成空白。若在建立表格時指定了其他預設值，該值將被使用。

這是由針對部門表格所定義的 REPORTS_TO_EXISTS 限制所導致。

若衍生表具有 RESTRICT 或 NO ACTION 刪除規則且找到某一列，衍生列將無法刪除，整個 DELETE 會失敗。

透過程式執行此陳述式時，所刪除的列數會傳回到 SQLCA 中的 SQLERRD(3)。此數字僅包含 DELETE 陳述式中所指定的表格刪除列數。而根據 CASCADE 規則所刪除的列數則不屬包含範圍。 SQLCA 中的 SQLERRD(5) 則含有各表格中受參照限制所影響的列數。

要瞭解 RESTRICT 規則與 NO ACTION 規則之間的微妙差異，最快的方法是觀察觸發與參照限制的互動。觸發可定義在某項作業 (此處即爲 DELETE 陳述式) 之前或之後啓動。*觸發之前*是在執行 DELETE 之前即啓動，因此會早於任何限制檢查。*觸發之後*則是在執行 DELETE 後才啓動，因此會晚於 RESTRICT 限制規則 (立即執行檢查)，但早於 NO ACTION 限制規則 (於陳述式末尾處執行檢查)。觸發與規則的發生順序如下所述：

1. *觸發之前*會在 DELETE 之前啓動，且會早於 RESTRICT 或 NO ACTION 限制規則。
2. *觸發之後*會在 RESTRICT 限制規則之後啓動，但早於 NO ACTION 規則。

Example: DELETE 重疊規則

從 DEPARTMENT 表格中刪除部門，將使指派至該部門的每位員工之 WORKDEPT (於 EMPLOYEE 表格中) 皆設爲空值。試考慮下列 DELETE 陳述式：

```
DELETE FROM CORPDATA.DEPARTMENT
       WHERE DEPTNO = 'E11'
```

以附錄 A, 『DB2 UDB for iSeries 表格範例』 中所列的表格與資料爲例，假設某一列已從表格 DEPARTMENT 中刪除，且表格 EMPLOYEE 已更新而將 WORKDEPT 值設爲其預設值 (每當 WORKDEPT 值爲 'E11' 時)。下列範例資料中的問號 ('?') 代表

空值。其結果如下：

表 22. DEPARTMENT 表格. 此表格在完成 DELETE 陳述式後的內容。

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00
B01	PLANNING	000020	A00
C01	INFORMATION CENTER	000030	A00
D01	DEVELOPMENT CENTER	?	A00
D11	MANUFACTURING SYSTEMS	000060	D01
D21	ADMINISTRATION SYSTEMS	000070	D01
E01	SUPPORT SERVICES	000050	A00
E21	SOFTWARE SUPPORT	000100	E01
F22	BRANCH OFFICE F2	?	E01
G22	BRANCH OFFICE G2	?	E01
H22	BRANCH OFFICE H2	?	E01
I22	BRANCH OFFICE I2	?	E01
J22	BRANCH OFFICE J2	?	E01

請注意，DEPARTMENT 表格中並無重疊的刪除，因爲沒有任何部門隸屬於部門 'E11'。

以下所示爲完成 DELETE 之前及之後，EMPLOYEE 表格受影響的部份。

表 23. 部份 EMPLOYEE 表格. 完成 DELETE 陳述式之前的部份內容。

EMPNO	FIRSTNME	MI	LASTNAME	WORKDEPT	PHONENO	HIREDATE
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05
000250	DANIEL	S	SMITH	D21	0961	1960-10-30
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11
000270	MARIA	L	PEREZ	D21	9001	1980-09-30
000280	ETHEL	R	SCHNEIDER	E11	0997	1967-03-24
000290	JOHN	R	PARKER	E11	4502	1980-05-30
000300	PHILIP	X	SMITH	E11	2095	1972-06-19
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07
000330	WING		LEE	E21	2103	1976-02-23
000340	JASON	R	GOUNOT	E21	5696	1947-05-05

表 24. 部份 EMPLOYEE 表格. 完成 DELETE 陳述式之後的部份內容。

EMPNO	FIRSTNME	MI	LASTNAME	WORKDEPT	PHONENO	HIREDATE
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05
000250	DANIEL	S	SMITH	D21	0961	1960-10-30
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11
000270	MARIA	L	PEREZ	D21	9001	1980-09-30

表 24. 部份 EMPLOYEE 表格 (繼續). 完成 DELETE 陳述式之後的部份內容。

EMPNO	FIRSTNME	MI	LASTNAME	WORKDEPT	PHONENO	HIREDATE
000280	ETHEL	R	SCHNEIDER	?	0997	1967-03-24
000290	JOHN	R	PARKER	?	4502	1980-05-30
000300	PHILIP	X	SMITH	?	2095	1972-06-19
000310	MAUDE	F	SETRIGHT	?	3332	1964-09-12
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07
000330	WING		LEE	E21	2103	1976-02-23
000340	JASON	R	GOUNOT	E21	5696	1947-05-05

核對擱置

參照限制與核對限制可處於稱爲核對擱置的狀態下，在此狀態下存有違反限制的可能。對參照限制而言，當上層鍵與外來索引鍵間存有不相符情況時即會發生違規。對核對限制而言，當受到核對限制所規範的直欄中存有潛伏值時即會發生違規。當系統研判限制可能已遭違反時 (例如在復置作業之後)，此限制即會被標示爲核對擱置。發生此情況時，與該限制相關的表格在使用上將被受限。對參照限制而言，下列限制將會套用：

- 不許對相依檔進行輸入或輸出作業。
- 只允許對上層檔案進行插入作業。

當核對限制處於核對擱置狀態下時，下列限制將會套用：

- 不許對檔案進行讀取作業。
- 允許插入及更新，且強制施行限制。

若要將核對限制解除核對擱置，您必須：

1. 停用對「變更實體檔案限制 (CHGPFCST)」CL 指令的關係。
2. 更正參照限制的索引鍵 (外部索引、上層鍵或兩者) 資料，或核對限制的直欄資料。
3. 以 CHGPFCST CL 指令再次啓用該限制。

您可以用「顯示核對擱置限制 (DSPCPCST)」CL 指令，來識別違反限制的相關列。

有關使用處於核對擱置狀態之表格的詳細資訊，請參閱 Database Programming 一書。

概略表上的 WITH CHECK OPTION

WITH CHECK OPTION 是 CREATE VIEW 陳述式的選用子句，可在透過概略表來插入或更新資料時，用來指定應執行的核對層次。若指定了此選項，概略表中所插入或更新的每一列皆須符合此概略表的定義。

若概略表是唯讀的，即不可指定 WITH CHECK OPTION。概略表的定義不得含有子查詢。

建立概略表時若未使用 WITH CHECK OPTION 子句，系統將不會核對此概略表的插入與更新作業是否符合概略表定義。如果概略表是直接或間接相依於另一含有 WITH CHECK OPTION 的概略表，某些核對仍會進行。由於並未使用概略表的定義，因此資料列可能是透過不符定義的概略表所插入或更新。此即表示無法再以概略表來選取資料列。

核對方式可爲 『WITH CASCADED CHECK OPTION』 或 『WITH LOCAL CHECK OPTION』。有關 WITH CHECK OPTION 的其他討論，請參閱 SQL Reference 一書中的 CREATE VIEW 主題。

WITH CASCADED CHECK OPTION

WITH CASCADED CHECK OPTION 可用來指定透過概略表所插入或更新的每一列皆須符合概略表的定義。此外，當已插入或更新各列時，所有相依概略表的搜尋條件將被核對。若某列不符概略表的定義，該列即不得使用概略表來擷取。

例如，試考慮下列可更新概略表：

```
CREATE VIEW V1 AS SELECT COL1
       FROM T1 WHERE COL1 > 10
```

由於未指定 WITH CHECK OPTION，因此即便所插入的值不符概略表的搜尋條件，下述 INSERT 陳述式仍會成功。

```
INSERT INTO V1 VALUES (5)
```

若透過 V1 建立另一概略表，且指定 WITH CASCADED CHECK OPTION：

```
CREATE VIEW V2 AS SELECT COL1
         FROM V1 WITH CASCADED CHECK OPTION
```

則下述 INSERT 陳述式將失敗，因爲它會產生不符 V2 定義的列：

```
INSERT INTO V2 VALUES (5)
```

再考慮另一透過 V2 所建立的概略表：

```
CREATE VIEW V3 AS SELECT COL1
         FROM V2 WHERE COL1 < 100
```

下述 INSERT 陳述式將失敗，因爲 V3 相依於 V2，而 V2 具有 WITH CASCADED CHECK OPTION。

```
INSERT INTO V3 VALUES (5)
```

但下述 INSERT 陳述式則會成功，因爲它符合 V2 的定義。而由於 V3 不具備 WITH CASCADED CHECK OPTION，所以不論陳述式是否符合 V3 的定義皆無關係。

```
INSERT INTO V3 VALUES (200)
```

WITH LOCAL CHECK OPTION

除了您可以更新某列使其無法再透過概略表來擷取外，WITH LOCAL CHECK OPTION 即相同於 WITH CASCADED CHECK OPTION。該情況唯有在概略表直接或間接相依於未定義 WITH CHECK OPTION 子句的概略表時才會發生。

就以前述範例中的可更新概略表爲例：

```
CREATE VIEW V1 AS SELECT COL1
         FROM T1 WHERE COL1 > 10
```

透過 V1 建立第二個概略表，不過此次指定了 WITH LOCAL CHECK OPTION：

```
CREATE VIEW V2 AS SELECT COL1
         FROM V1 WITH LOCAL CHECK OPTION
```

在先前 CASCADED CHECK OPTION 範例中失敗的同一 INSERT 此時將會成功，因爲 V2 不帶任何搜尋條件，而 V1 的搜尋條件則不需核對 (因 V1 未指定核對選項)。

```
INSERT INTO V2 VALUES (5)
```

若再考慮另一個透過 V2 所建立的概略表：

```
CREATE VIEW V3 AS SELECT COL1
     FROM V2 WHERE COL1 < 100
```

下述 INSERT 亦會再次成功，因爲 V1 的搜尋條件受 V2 的 WITH LOCAL CHECK OPTION 所影響而不會被核對，不同於前述範例的 WITH CASCADED CHECK OPTION。

```
INSERT INTO V3 VALUES (5)
```

LOCAL 與 CASCADED CHECK OPTION 之間的差異，取決於插入或更新某列時，相依概略表被核對的搜尋條件之多寡。

- WITH LOCAL CHECK OPTION 會指定在插入或更新某列時，只針對具有 WITH LOCAL CHECK OPTION 或 WITH CASCADED CHECK OPTION 的相依概略表來核對其搜尋條件。
- WITH CASCADED CHECK OPTION 則會指定在插入或更新某列時，所有相依概略表的搜尋條件皆須核對。

範例：階式排列核對選項

試以下述表格與概略表爲例：

```
CREATE TABLE T1 (COL1 CHAR(10))

CREATE VIEW  V1 AS SELECT COL1
      FROM T1 WHERE COL1 LIKE 'A%'

CREATE VIEW V2 AS SELECT COL1
      FROM V1 WHERE COL1 LIKE '%Z'
                           WITH LOCAL CHECK OPTION

CREATE VIEW V3 AS SELECT COL1
      FROM V2 WHERE COL1 LIKE 'AB%'

CREATE VIEW V4 AS SELECT COL1
      FROM V3 WHERE COL1 LIKE '%YZ'
                            WITH CASCADED CHECK OPTION

CREATE VIEW V5 AS SELECT COL1
      FROM V4 WHERE COL1 LIKE 'ABC%'
```

不同的搜尋條件是否接受核對，將取決於 INSERT 或 UPDATE 所操作的概略表。

- 若操作於 V1，將無任何條件受到核對，因爲 V1 未指定 WITH CHECK OPTION。
- 若操作於 V2，
  - COL1 須以字母 Z 爲結尾，但不須以字母 A 爲開頭。這是因爲核對選項爲 LOCAL，而概略表 V1 並未指定任何核對選項。
- 若操作於 V3，
  - COL1 須以字母 Z 爲結尾，但不須以字母 A 爲開頭。 V3 未指定任何核對選項，因此其本身的搜尋條件必不得相符。不過，V2 的搜尋條件則須核對，因爲 V3 是定義於 V2 中，而 V2 具有核對選項。
- 若操作於 V4，

– COL1 須以 'AB' 爲開頭，並須以 'YZ' 爲結尾。由於 V4 已指定 WITH CASCADED CHECK OPTION，因此 V4 所相依之每個概略表的所有搜尋條件皆須核對。

- 若操作於 V5，

– COL1 須以 'AB' 爲開頭，但不須爲 'ABC' 形式。這是因爲 V5 未指定核對選項，因此其本身的搜尋條件不須核對。不過，由於 V5 是在 V4 中定義，而 V4 具有階式排列核對選項，因此 V4、V3、V2 及 V1 的每項搜尋條件都須核對。亦即，COL1 須以 'AB' 爲開頭，並須以 'YZ' 爲結尾。

如果 V5 是以 WITH LOCAL CHECK OPTION 來建立，操作於 V5 將意味著 COL1 須以 'ABC' 爲開頭，並須以 'YZ' 爲結尾。 LOCAL CHECK OPTION 則會新增一附加規範，即第三個字元須爲 'C'。

DB2 UDB for iSeries 觸發支援

*觸發*是一組行動，當對指定表格執行指定的變更作業時，系統即會自動加以執行。所謂變更作業可以是一 SQL INSERT、UPDATE 或 DELETE 陳述式，或是應用程式中的插入、更新或刪除等高階語言陳述式。觸發對諸如施行商業規則、驗證輸入資料及保存審核追蹤極爲有用。

觸發可用下列兩種不同方式來定義：

- 第 137 頁的『SQL 觸發程式』
- 第 141 頁的『外部觸發』

若是外部觸發，可使用 CRTPFTRG CL 指令。含有觸發動作集合的程式可用任何受支援的高階語言來定義。外部觸發可爲插入、更新、刪除或讀取等類觸發。

若是 SQL 觸發程式，則使用 CREATE TRIGGER 陳述式。此時觸發程式完全是以 SQL 來定義。SQL 觸發程式可爲插入、更新或刪除等類觸發。

一旦將觸發與表格結合後，每當對表格 (或透過此表格所建立的任何邏輯檔案及概略表) 進行變更作業時，觸發支援即會呼叫觸發程式。您可爲同一表格定義 SQL 觸發與外部觸發。單一表格最多可定義 200 個觸發。

每一個變更作業都可在作業發生之前或之後來呼叫觸發。此外，您還可以新增*讀取*觸發，以便在每次表格被存取時加以呼叫。因此，表格可和許多類型的觸發相結合。

- 刪除觸發前
- 插入觸發前
- 更新觸發前
- 刪除觸發後
- 插入觸發後
- 更新觸發後
- 唯讀觸發 (僅限外部觸發)

有關觸發限制的詳細資訊，包括可爲 SQL 表格定義多少觸發及最大的觸發巢狀層次，以及有關撰寫觸發時的建議與預防措施，請參閱 Database Programming 一書中的 Triggering automatic events in the your database 章節。

SQL 觸發程式

SQL CREATE TRIGGER 陳述式可提供有效的方式，讓資料庫管理系統每在執行插入、更新或刪除作業時，能主動控制、監督及管理眾多表格。每當執行 SQL 插入、更新或刪除作業時，SQL 觸發程式中所指定的陳述式即會執行。而當執行觸發程式時，SQL 觸發程式可以呼叫儲存程序或使用者定義的函數來執行額外處理。

有別於儲存程序的是，SQL 觸發程式不得從應用程式中直接呼叫。反之，SQL 觸發程式是由資料庫管理系統於執行觸發插入、更新或刪除作業時所呼叫。 SQL 觸發程式的相關定義是儲存在資料庫管理系統中，當用於定義觸發的 SQL 表格遭變更時，資料庫管理系統即會加以呼叫。

有關建立 SQL 觸發程式的詳細資訊，請參閱 『建立 SQL 觸發程式』。

如需關於使用 CREATE TRIGGER 陳述式的完整說明，請參閱 SQL Reference 主題中的 CREATE TRIGGER 陳述式。

建立 SQL 觸發程式

SQL 觸發程式可藉由指定 CREATE TRIGGER SQL 陳述式來建立。 SQL 會將 SQL 觸發程式的常式主體中之陳述式轉換爲程式 (*PGM) 物件。此程式將在觸發程式名稱限定元所指定的綱目中建立。而指定的觸發程式則會登記於 SYSTRIGGERS、SYSTRIGDEP、SYSTRIGCOL 及 SYSTRIGUPD 等 SQL 編目中。有關如何在 SQL 觸發程式中使用各類控制陳述式，以及如何在 SQL 陳述式層次進行 SQL 觸發程式除錯的詳細資訊，請參閱 SQL Reference 中的 SQL procedures, functions, and triggers 章節。

如需相關範例以及有關建立 SQL 觸發程式時的注意事項，請參閱：

- 『BEFORE SQL 觸發』
- 第 138 頁的『AFTER SQL 觸發』
- 第 139 頁的『SQL 觸發程式中的處理程式』
- 第 140 頁的『SQL 觸發程式轉移表格』

BEFORE SQL 觸發

BEFORE 觸發雖無法修改表格，但可用來驗證輸入欄位值，以及修改表格中所插入或更新的欄位值。下述範例中，觸發先被用來設定公司的會計季度，之後才將相關列插入至目標表格。

```
CREATE TABLE TransactionTable (DateOfTransaction DATE, FiscalQuarter SMALLINT)

CREATE TRIGGER TransactionBeforeTrigger BEFORE INSERT ON TransactionTable
REFERENCING NEW AS new_row
FOR EACH ROW MODE DB2ROW
BEGIN
  DECLARE newmonth SMALLINT;
SET newmonth = MONTH(new_row.DateOfTransaction);
  IF newmonth < 4 THEN
    SET new_row.FiscalQuarter=3;
  ELSEIF newmonth < 7 THEN
    SET new_row.FiscalQuarter=4;
  ELSEIF newmonth < 10 THEN
    SET new_row.FiscalQuarter=1;
```

```
  ELSE
    SET new_row.FiscalQuarter=2;
  END IF;
END
```

對下述 SQL 插入陳述式而言，若目前日期爲西元 2000 年 11 月 14 日，″FiscalQuarter″直欄將被設爲 2。

```
INSERT INTO TransactionTable(DateOfTransaction)
           VALUES(CURRENT DATE)
```

設若 SQL 觸發程式已存取並能使用「使用者定義的特殊類型 (UDT)」與儲存程序。下述範例中，SQL 觸發程式將呼叫儲存程序來執行某些預先定義的業務邏輯，此處是將某一直欄設爲業務上的預定值。

```
CREATE DISTINCT TYPE enginesize AS DECIMAL(5,2) WITH COMPARISONS

CREATE DISTINCT TYPE engineclass AS VARCHAR(25) WITH COMPARISONS

CREATE PROCEDURE SetEngineClass(IN SizeInLiters enginesize,
                                OUT CLASS engineclass)
LANGUAGE SQL CONTAINS SQL
BEGIN
  IF SizeInLiters<2.0 THEN
    SET CLASS = 'Mouse';
  ELSEIF SizeInLiters<3.1 THEN
    SET CLASS ='Economy Class';
  ELSEIF SizeInLiters<4.0 THEN
    SET CLASS ='Most Common Class';
  ELSEIF SizeInLiters<4.6 THEN
    SET CLASS = 'Getting Expensive';
  ELSE
    SET CLASS ='Stop Often for Fillups';
  END IF;
END

CREATE TABLE EngineRatings (VariousSizes enginesize, ClassRating engineclass)

CREATE TRIGGER SetEngineClassTrigger BEFORE INSERT ON EngineRatings
REFERENCING NEW AS new_row
FOR EACH ROW MODE DB2ROW
  CALL SetEngineClass(new_row.VariousSizes, new_row.ClassRating)
```

對下述 SQL 插入陳述式而言，若 ″VariousSizes″ 直欄的值爲 3.0，″ClassRating″ 直欄將會設爲 ″Economy Class″。

```
INSERT INTO EngineRatings(VariousSizes) VALUES(3.0)
```

SQL 規定在建立 SQL 觸發程式之前，所有表格、使用者定義的函數、程序及使用者定義的類型必須已經存在。上例中，所有的表格、儲存程序及使用者定義的類型，都是在建立觸發之前便已定義。

AFTER SQL 觸發

WHEN 條件可用於 SQL 觸發程式中來指定條件。如果條件經評估爲眞，則 SQL 觸發程式常式主體中的 SQL 陳述式即會執行。若條件經評估爲假，SQL 觸發程式常式主體中的 SQL 陳述式將不會執行，且控制權會交還給資料庫系統。下述範例中，SQL 會就查詢進行評估，以決定當啓動觸發後是否應執行觸發常式主體中的陳述式。

```
CREATE TABLE TodaysRecords(TodaysMaxBarometricPressure FLOAT,
  TodaysMinBarometricPressure FLOAT)

CREATE TABLE OurCitysRecords(RecordMaxBarometricPressure FLOAT,
```

```
  RecordMinBarometricPressure FLOAT)

CREATE TRIGGER UpdateMaxPressureTrigger
AFTER UPDATE OF TodaysMaxBarometricPressure ON TodaysRecords
REFERENCING NEW AS new_row
FOR EACH ROW MODE DB2ROW
WHEN (new_row.TodaysMaxBarometricPressure>
     (SELECT MAX(RecordMaxBarometricPressure) FROM
     OurCitysRecords))
  UPDATE OurCitysRecords
         SET RecordMaxBarometricPressure =
             new_row.TodaysMaxBarometricPressure

CREATE TRIGGER UpdateMinPressureTrigger
AFTER UPDATE OF TodaysMinBarometricPressure
ON TodaysRecords
REFERENCING NEW AS new_row
FOR EACH ROW MODE DB2ROW
WHEN(new_row.TodaysMinBarometricPressure<
     (SELECT MIN(RecordMinBarometricPressure) FROM
     OurCitysRecords))
  UPDATE OurCitysRecords
         SET RecordMinBarometricPressure =
             new_row.TodaysMinBarometricPressure
```

首先是起始設定表格的現行值。

```
INSERT INTO TodaysRecords VALUES(0.0,0.0)
INSERT INTO OurCitysRecords VALUES(0.0,0.0)
```

對下述 SQL 更新陳述式而言，OurCitysRecords 中的 RecordMaxBarometricPressure 是由 UpdateMaxPressureTrigger 所更新。

```
UPDATE TodaysRecords SET TodaysMaxBarometricPressure = 29.95
```

但到了明天，如果 TodaysMaxBarometricPressure 僅爲 29.91，則 RecordMaxBarometricPressure 即不會更新。

```
UPDATE TodaysRecords SET TodaysMaxBarometricPressure = 29.91
```

SQL 允許爲單一觸發動作定義多個觸發。在上述範例中，共有兩個 AFTER UPDATE 觸發：UpdateMaxPressureTrigger 與 UpdateMinPressureTrigger。這些觸發都只在表格 TodaysRecords 的特定直欄遭更新時才會啓動。

AFTER 觸發可修改表格。上例中，UPDATE 作業會被套用至第二個表格。注意應避免遞迴性的插入與更新作業。若達到觸發巢狀層次的上限，資料庫管理系統即會終止該作業。您可以藉由新增條件邏輯來避免遞迴情況，使插入或更新作業在未達觸發巢狀層次的上限前即結束。在遞迴性重疊顯示的觸發網路中，亦須避免此情況發生。

SQL 觸發程式中的處理程式

SQL 觸發程式中的處理程式可讓 SQL 觸發程式有能力透過錯誤或日誌資訊 (關於執行觸發常式主體中的 SQL 陳述式時所發生的錯誤者) 來回復。

下例中，已定義了兩項處理程式：一個負責處理溢位狀況，另一個則處理 SQL 異常。

```
CREATE TABLE ExcessInventory(Description VARCHAR(50), ItemWeight SMALLINT)

CREATE TABLE YearToDateTotals(TotalWeight SMALLINT)

CREATE TABLE FailureLog(Item VARCHAR(50), ErrorMessage VARCHAR(50), ErrorCode INT)

CREATE TRIGGER InventoryDeleteTrigger
```

```
AFTER DELETE ON ExcessInventory
REFERENCING OLD AS old_row
FOR EACH ROW MODE DB2ROW
BEGIN
  DECLARE sqlcode INT;
  DECLARE invalid_number condition FOR '22003';
  DECLARE exit handler FOR invalid_number
  INSERT INTO FailureLog VALUES(old_row.Description,
        'Overflow occurred in YearToDateTotals', sqlcode);
  DECLARE exit handler FOR sqlexception
  INSERT INTO FailureLog VALUES(old_row.Description,
        'SQL Error occurred in InventoryDeleteTrigger', sqlcode);
  UPDATE YearToDateTotals SET TotalWeight=TotalWeight +
        old_row.itemWeight;
END
```

首先，是起始設定表格的現行值。

```
INSERT INTO ExcessInventory VALUES('Desks',32500)
INSERT INTO ExcessInventory VALUES('Chairs',500)
INSERT INTO YearToDateTotals VALUES(0)
```

執行下述第一個 SQL 刪除陳述式時，項目 ″Desks″ 的 ItemWeight 會加總到表格 YearToDateTotals 的 TotalWeight 直欄總計中。執行第二個 SQL 刪除陳述式時，當項目 ″Chairs″ 的 ItemWeight 加總至 TotalWeight 的直欄總計時發生溢位，因爲該直欄最多只能處理到 32767 的值。發生溢位時，invalid_number 結束處理程式將會執行，而且會有一資料列寫入 FailureLog 表格。若 YearToDateTotals 表格遭意外刪除，sqlexception 結束處理程式將會執行。本例中，處理程式是用來寫入日誌，以便日後可就問題進行診斷。

```
DELETE FROM ExcessInventory WHERE Description='Desks'
DELETE FROM ExcessInventory WHERE Description='Chairs'
```

SQL 觸發程式轉移表格

SQL 觸發程式可能會需要參考所有受到 SQL 插入、更新或刪除作業影響的列。例如，若觸發須對受影響列的特定直欄套用聚集函數 (例如 MIN 或 MAX) 時。 OLD_TABLE 與 NEW_TABLE 轉移表格即可用於此情況。在下述範例中，觸發即對表格 StudentProfiles 中所有受影響的列，套用了聚集函數 MAX。

```
CREATE TABLE StudentProfiles(StudentsName VARCHAR(125),
      StudentsYearInSchool SMALLINT, StudentsGPA DECIMAL(5,2))

CREATE TABLE CollegeBoundStudentsProfile
      (YearInSchoolMin SMALLINT, YearInSchoolMax SMALLINT, StudentGPAMin
      DECIMAL(5,2), StudentGPAMax DECIMAL(5,2))

CREATE TRIGGER UpdateCollegeBoundStudentsProfileTrigger
AFTER UPDATE ON StudentProfiles
REFERENCING NEW_TABLE AS ntable
FOR EACH STATEMENT MODE DB2SQL
BEGIN
  DECLARE maxStudentYearInSchool SMALLINT;
  SET maxStudentYearInSchool =
       (SELECT MAX(StudentsYearInSchool) FROM ntable);
  IF maxStudentYearInSchool >
       (SELECT MAX (YearInSchoolMax) FROM
          CollegeBoundStudentsProfile) THEN
    UPDATE CollegeBoundStudentsProfile SET YearInSchoolMax =
         maxStudentYearInSchool;
  END IF;
END
```

上述範例中，執行觸發更新陳述式後，僅執行了一次觸發，因爲它是定義爲 FOR EACH STATEMENT 觸發。當您定義會參照轉移表格的觸發時，須考慮資料庫管理系統在大量輸入資料至轉移表格時所需的額外處理時間。

外部觸發

對外部觸發而言，含有觸發動作集合的程式可用任何受支援的高階語言 (建立 *PGM 物件者) 來定義。觸發程式中可內含 SQL。若要定義外部觸發，您必須先建立觸發程式，然後用 ADDPFTRG CL 指令將其新增至表格中，或者用 iSeries 領航員來加以新增。若要在表格中新增觸發，您必須：

- 界定表格
- 界定作業的種類
- 界定會執行所需動作的程式。

如需外部觸發的範例，請參閱『外部觸發範例程式』。

外部觸發範例程式

以下是外部觸發程式的範例。此範例是以 ILE C 所撰寫，且具有內含的 SQL。

如需完整的說明及更多有關 DB2 UDB for iSeries 中的外部觸發範例，請參閱 Database Programming 一書中的 Triggering automatic events in the your database 章節。

註: 如須相關的程式碼範例，請參閱第 x 頁的『程式碼不保事項聲明』。

```
#include "string.h"
#include "stdlib.h"
#include "stdio.h"
#include <recio.h>
#include <xxcvt.h>
#include "qsysinc/h/trgbuf"       /* Trigger input parameter         */
#include "lib1/csrc/msghand1"     /* User defined message handler    */
/*********************************************************************/
/* This is a trigger program which is called whenever there is an    */
/* update to the EMPLOYEE table. If the employee's commission is     */
/* greater than the maximum commission, this trigger program will    */
/* increase the employee's salary by 1.04 percent and insert into    */
/* the RAISE table.                                                  */
/*                                                                   */
/* The EMPLOYEE record information is passed from the input parameter*/
/* to this trigger program.                                          */
/*********************************************************************/

Qdb_Trigger_Buffer_t *hstruct;
char *datapt;

   /*****************************************************/
   /* Structure of the EMPLOYEE record which is used to   */
   /* store the old or the new record that is passed to   */
   /* this trigger program.                               */
   /*                                                     */
   /* Note : You must ensure that all the numeric fields  */
   /*        are aligned at 4 byte boundary in C.         */
   /*        Used either Packed struct or filler to reach */
   /*        the byte boundary alignment.                 */
   /*****************************************************/

_Packed struct rec{
         char  empn[6];
  _Packed struct { short fstlen ;
                   char  fstnam[12];
                 } fstname;
         char  minit[1];
_Packed struct { short lstlen;
                 char lstnam[15];
               } lstname;
       char  dept[3];
       char  phone[4];
       char  hdate[10];
       char  jobn[8];
       short edclvl;
       char  sex1[1];
       char  bdate[10];
       decimal(9,2) salary1;
       decimal(9,2) bonus1;
       decimal(9,2) comm1;
       } oldbuf, newbuf;
EXEC SQL INCLUDE SQLCA;
```

圖 2. 觸發程式範例 (1/5)

```
main(int argc, char **argv)
{
int i;
int obufoff;                        /* old buffer offset            */
int nuloff;                         /* old null byte map offset     */
int nbufoff;                        /* new buffer offset            */
int nul2off;                        /* new null byte map offset     */
short work_days  = 253;             /* work days during in one year */
decimal(9,2) commission = 2000.00; /* cutoff to qualify for        */
decimal(9,2) percentage = 1.04;    /* raised salary as percentage  */
char raise_date[12] = "1982-06-01";/* effective raise date         */


struct {
       char  empno[6];
       char  name[30];
       decimal(9,2) salary;
       decimal(9,2) new_salary;
       } rpt1;

   /*******************************************************/
   /* Start to monitor any exception.                     */
   /*******************************************************/

  _FEEDBACK fc;
  _HDLR_ENTRY hdlr = main_handler;
                            /*****************************************/
                            /* Make the exception handler active.    */
                            /*****************************************/
  CEEHDLR(&hdlr, NULL, &fc);
                            /*****************************************/
                            /* Ensure exception handler OK           */
                            /*****************************************/
  if (fc.MsgNo != CEE0000)
    {
      printf("Failed to register exception handler.\n");
      exit(99);
    };

   /*******************************************************/
   /* Move the data from the trigger buffer to the local  */
   /* structure for reference.                            */
   /*******************************************************/

hstruct = (Qdb_Trigger_Buffer_t *)argv[1];
datapt  = (char *) hstruct;

obufoff = hstruct ->Old_Record_Offset;       /* old buffer   */
memcpy(&oldbuf,datapt+obufoff,; hstruct->Old_Record_Len);

nbufoff = hstruct ->New_Record_Offset;       /* new buffer   */
memcpy(&newbuf,datapt+nbufoff,; hstruct->New_Record_Len);
```

圖 2. 觸發程式範例 (2/5)

```
EXEC SQL WHENEVER SQLERROR  GO TO ERR_EXIT;

   /*******************************************************/
   /* Set the transaction isolation level to the same as  */
   /* the application based on the input parameter in the */
   /* trigger buffer.                                     */
   /*******************************************************/

if(strcmp(hstruct->Commit_Lock_Level,"0") == 0)
     EXEC SQL SET TRANSACTION ISOLATION LEVEL NONE;
else{
  if(strcmp(hstruct->Commit_Lock_Level,"1") == 0)
       EXEC SQL SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED, READ
                 WRITE;
  else {
     if(strcmp(hstruct->Commit_Lock_Level,"2") == 0)
       EXEC SQL SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
  else
     if(strcmp(hstruct->Commit_Lock_Level,"3") == 0)
       EXEC SQL SET TRANSACTION ISOLATION LEVEL ALL;
  }
}

 /*******************************************************/
 /* If the employee's commission is greater than maximum */
 /* commission, then increase the employee's salary      */
 /* by 1.04 percent and insert into the RAISE table.     */
 /*******************************************************/

if (newbuf.comm1 >= commission)
{
  EXEC SQL  SELECT EMPNO, EMPNAME, SALARY
         INTO :rpt1.empno, :rpt1.name, :rpt1.salary
         FROM TRGPERF/EMP_ACT
         WHERE EMP_ACT.EMPNO=:newbuf.empn ;

  if (sqlca.sqlcode == 0) then
  {
     rpt1.new_salary = salary * percentage;
     EXEC SQL INSERT INTO TRGPERF/RAISE VALUES(:rpt1);
  }
  goto finished;
 }
 err_exit:
   exit(1);

  /* All done  */
 finished:
  return;
}  /* end of main line  */
```

圖 2. 觸發程式範例 (3/5)

```
/*****************************************************************/
/*   INCLUDE NAME : MSGHAND1                                     */
/*                                                               */
/*   DESCRIPTION  : Message handler to signal an exception to    */
/*                  the application to inform that an            */
/*                  error occured in the trigger program.        */
/*                                                               */
/*   NOTE : This message handler is a user defined routine.      */
/*                                                               */
/*****************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <recio.h>
#include <leawi.h>

#pragma linkage (QMHSNDPM, OS)
void QMHSNDPM(char *,             /* Message identifier             */
              void *,             /* Qualified message file name    */
              void *,             /* Message data or text           */
              int,                /* Length of message data or text */
              char *,             /* Message type                   */
              char *,             /* Call message queue             */
              int,                /* Call stack counter             */
              void *,             /* Message key                    */
              void *,             /* Error code                     */
              ...);               /* Optionals:
                                       length of call message queue
                                        name
                                       Call stack entry qualification
                                       display external messages
                                        screen wait time            */
/*********************************************************************/
/********  This is the start of the exception handler function.      */
/*********************************************************************/
  void main_handler(_FEEDBACK *cond, _POINTER *token, _INT4 *rc,
                    _FEEDBACK *new)
   {
                           /*************************************/
                           /* Initialize variables for call to  */
                           /* QMHSNDPM.                         */
                           /* User must create a message file and */
                           /* define a message ID to match the  */
                           /* following data.                   */
                           /*************************************/
  char      message_id[7] = "TRG9999";
  char      message_file[20] = "MSGF      LIB1      ";
  char      message_data[50] = "Trigger error                " ;
  int       message_len = 30;
  char      message_type[10] = "*ESCAPE   ";
  char      message_q[10] = "_C_pep    ";
  int       pgm_stack_cnt = 1;
  char      message_key[4];
```

圖 2. 觸發程式範例 (4/5)

```
                          /****************************************/
                          /* Declare error code structure for     */
                          /* QMHSNDPM.                            */
                          /****************************************/
struct error_code {
  int bytes_provided;
  int bytes_available;
  char message_id[7];
} error_code;

error_code.bytes_provided = 15;
                          /****************************************/
                          /* Set the error handler to resume and  */
                          /* mark the last escape message as      */
                          /* handled.                             */
                          /****************************************/
*rc = CEE_HDLR_RESUME;
                          /****************************************/
                          /* Send my own *ESCAPE message.         */
                          /****************************************/
QMHSNDPM(message_id,
        &message_file,
        &message_data,
         message_len,
         message_type,
         message_q,
         pgm_stack_cnt,
        &message_key,
        &error_code );
                          /****************************************/
                          /* Check that the call to QMHSNDPM      */
                          /* finished correctly.                  */
                          /****************************************/
if (error_code.bytes_available != 0)
    {
      printf("Error in QMHOVPM : %s\n", error_code.message_id);
    }
}
```

圖 2. 觸發程式範例 (5/5)

第 11 章 儲存程序

程序 (通常稱爲儲存程序) 是一個程式，可以在呼叫後執行包含主語言陳述式和 SQL 陳述式的作業。 SQL 中的程序提供與主語言中程序相同的優點。

DB2 SQL for iSeries 儲存程序支援爲 SQL 應用程式提供一種方法，可以透過 SQL 陳述式定義然後再呼叫程序。儲存程序可用於分散式和非分散式 DB2 SQL for iSeries 應用程式。使用儲存程序的優點之一，是對分散式應用程式而言，在應用程式要求程式 (或從屬站) 上執行一次 CALL 陳述式，可以在應用程式伺服器上執行任意數量的工作。

您可以將程序定義爲 SQL 程序或外部程序。外部程序可以是所支援的任何高階語言程式 (System/36* 程式和程序除外) 或 REXX 程序。程序不一定要包含 SQL 陳述式，但是可以包含 SQL 陳述式。 SQL 程序完全定義於 SQL 中，可以併入包含 SQL 控制陳述式的 SQL 陳述式。

使用者若要撰寫儲存程序，必須瞭解下列各項：

- 透過 CREATE PROCEDURE 陳述式的儲存程序定義
- 透過 CALL 陳述式的儲存程序呼叫
- 參數傳遞慣例
- 將完成狀態傳回程式以呼叫程序的方法。

您可以使用 CREATE PROCEDURE 陳述式定義儲存程序。 CREATE PROCEDURE 陳述式會將程序和參數定義加入目錄表格 SYSROUTINES 和 SYSPARMS。然後系統中任何的 SQL CALL 陳述式都可以存取這些定義。

若要建立外部程序或 SQL 程序，可以使用 SQL CREATE PROCEDURE 陳述式。或者，可以使用「iSeries 領航員」定義程序。

以下各節會說明用於定義和呼叫儲存程序的 SQL 陳述式、傳遞參數至儲存程序的資訊，以及儲存程序用法的範例。

- 第 148 頁的『定義外部程序』
- 第 148 頁的『定義 SQL 程序』
- 第 153 頁的『爲儲存程序除錯』
- 第 154 頁的『呼叫儲存程序』
- 第 158 頁的『儲存程序和 UDF 的參數傳遞慣例』
- 第 162 頁的『指示器變數和儲存程序』
- 第 164 頁的『傳回完成狀態至呼叫程式』
- 第 165 頁的『CALL 陳述式的範例』

如需以 Java 撰寫之儲存程序的說明，請參閱 IBM Developer Kit for Java 主題中的 Java SQL Routines。

如需以 DRDA 使用儲存程序的資訊，請參閱第 321 頁的『DRDA 儲存程序注意事項』。

註: 如需程式碼範例的相關資訊，請參閱第 x 頁的『程式碼不保事項聲明』的資訊。

定義外部程序

外部程序的 CREATE PROCEDURE 陳述式：

- 爲程序命名
- 定義參數及其屬性
- 提供系統在呼叫程序時使用的程序其他相關資訊。

請考量以下範例：

```
CREATE PROCEDURE P1
         (INOUT PARM1 CHAR(10))
         EXTERNAL NAME MYLIB.PROC1
         LANGUAGE C
         GENERAL WITH NULLS;
```

此 CREATE PROCEDURE 陳述式：

- 將程序命名爲 P1。
- 定義一個同時做爲輸入參數和輸出參數的參數。參數爲長度十的字元欄位。參數可以定義爲 IN、OUT 或 INOUT 等類型。參數類型決定參數值傳遞至程序，或從程序傳來的時機。
- 定義對應至程序的程式名稱，是 MYLIB 中的 PROC1。 MYLIB.PROC1 爲程序於 CALL 陳述式呼叫時所呼叫的程式。
- 指示程序 P1 (程式 MYLIB.PROC1) 是以 C 寫成。語言會影響可以傳遞之參數類型，因此很重要。它也會影響參數傳遞至程序的方式 (例如，ILE C 程序的 NUL 終止字元會在圖形、日期、時間及時間戳記參數上傳遞)。
- 定義 CALL 類型爲 GENERAL WITH NULLS。這指示程序的參數可能包含 NULL 值，因此要有一個附加的引數傳遞至 CALL 陳述式的程序。附加的引數是 N 短整數的陣列，N 是在 CREATE PROCEDURE 陳述式中宣告的參數數目。在此範例中，陣列只包含一個元素，因爲只有一個參數。

特別要注意的是，不一定要定義程序才能呼叫程序。不過，若是在此程式中前一個 CREATE PROCEDURE 或 DECLARE PROCEDURE 找不到程序定義，會在 CALL 陳述式上呼叫程序時加上一些限制和假設。例如，無法傳遞 NULL 指示器引數。如需不使用對應程序定義的 CALL 陳述式範例，請參閱第 155 頁的『使用沒有程序定義存在的內含 CALL 陳述式』。

定義 SQL 程序

SQL 程序的 CREATE PROCEDURE 陳述式：

- 爲程序命名
- 定義參數及其屬性
- 提供於呼叫程序時所使用程序之其他相關資訊。
- 定義程序主體。程序主體爲程序可執行的部分，是單一的 SQL 陳述式。

請考量以下簡單範例，此範例以員工編號和等級爲輸入，並且更新員工的薪資：

```
CREATE PROCEDURE UPDATE_SALARY_1
  (IN EMPLOYEE_NUMBER CHAR(10),
   IN RATE DECIMAL(6,2))
   LANGUAGE SQL MODIFIES SQL DATA
```

```
  UPDATE CORPDATA.EMPLOYEE
    SET SALARY = SALARY * RATE
    WHERE EMPNO = EMPLOYEE_NUMBER;
```

此 CREATE PROCEDURE 陳述式：

- 將程序命名爲 UPDATE_SALARY_1。
- 定義字元資料類型的輸入參數 EMPLOYEE_NUMBER，長度爲 6，以及十進位資料類型輸入參數 RATE。
- 指示程序是修改 SQL 資料的 SQL 程序。
- 將程序主體定義爲單一 UPDATE 陳述式。呼叫程序時，會使用爲 EMPLOYEE_NUMBER 和 RATE 傳遞的值執行 UPDATE 陳述式。

也可以使用 SQL 控制陳述式在 SQL 程序中加入邏輯，以代替單一 UPDATE 陳述式。SQL 控制陳述式包含下列項目：

- 分派陳述式
- CALL 陳述式
- CASE 陳述式
- 複合陳述式
- FOR 陳述式
- GET DIAGNOSTICS 陳述式
- GOTO 陳述式
- IF 陳述式
- ITERATE 陳述式
- LEAVE 陳述式
- LOOP 陳述式
- REPEAT 陳述式
- RESIGNAL 陳述式
- RETURN 陳述式
- SIGNAL 陳述式
- WHILE 陳述式

下列範例會以員工編號和上次評估時接收的等級做爲輸入。程序使用 CASE 陳述式決定更新的適當增加和紅利：

```
CREATE PROCEDURE UPDATE_SALARY_2
  (IN EMPLOYEE_NUMBER CHAR(6),
  IN RATING INT)
  LANGUAGE SQL MODIFIES SQL DATA
     CASE RATING
        WHEN 1 THEN
          UPDATE CORPDATA.EMPLOYEE
            SET SALARY = SALARY * 1.10,
             BONUS = 1000
            WHERE EMPNO = EMPLOYEE_NUMBER;
        WHEN 2 THEN
          UPDATE CORPDATA.EMPLOYEE
            SET SALARY = SALARY * 1.05,
            BONUS = 500
            WHERE EMPNO = EMPLOYEE_NUMBER;
       ELSE
           UPDATE CORPDATA.EMPLOYEE
```

```
            SET SALARY = SALARY * 1.03,
            BONUS = 0
            WHERE EMPNO = EMPLOYEE_NUMBER;
      END CASE;
```

此 CREATE PROCEDURE 陳述式：

- 將程序命名爲 UPDATE_SALARY_2。
- 定義字元資料類型的輸入參數 EMPLOYEE_NUMBER，長度爲 6，以及整數資料類型的輸入參數 RATING。
- 指示程序爲修改 SQL 資料的 SQL 程序。
- 定義程序主體。呼叫程序時，會檢查輸入參數 RATING，並執行適當的更新陳述式。

可以新增複合陳述式，將多個陳述式加入程序主體。複合陳述式中可以指定任意多個 SQL 陳述式。此外，也可以宣告 SQL 變數、游標及處理程式。

下列範例會以部門編號做爲輸入。它會傳回該部門中全部員工的總薪資，以及該部門中領到紅利的員工數目。

```
CREATE PROCEDURERETURN_DEPT_SALARY
         (IN DEPT_NUMBER CHAR(3),
          OUT DEPT_SALARY DECIMAL(15,2),
          OUT DEPT_BONUS_CNT INT)
         LANGUAGE SQL READS SQL DATA
         P1: BEGIN
          DECLARE EMPLOYEE_SALARY DECIMAL(9,2);
          DECLARE EMPLOYEE_BONUS DECIMAL(9,2);
          DECLARE TOTAL_SALARY DECIMAL(15,2)DEFAULT 0
          DECLARE BONUS_CNT INT DEFAULT 0;
          DECLARE END_TABLE INT DEFAULT 0;
          DECLARE C1 CURSOR FOR
           SELECT SALARY, BONUS FROM CORPDATA.EMPLOYEE
            WHERE WORKDEPT = DEPT_NUMBER;
          DECLARE CONTINUE HANDLER FOR NOT FOUND
            SET END_TABLE = 1;
          DECLARE EXIT HANDLER FOR SQLEXCEPTION
            SET DEPT_SALARY = NULL;
          OPEN C1;
          FETCH C1 INTO EMPLOYEE_SALARY, EMPLOYEE_BONUS;
          WHILE END_TABLE = 0 DO
             SET TOTAL_SALARY = TOTAL_SALARY + EMPLOYEE_SALARY + EMPLOYEE_BONUS;
             IF EMPLOYEE_BONUS > 0 THEN
               SET BONUS_CNT = BONUS_CNT + 1;
             END IF;
             FETCH C1 INTO EMPLOYEE_SALARY, EMPLOYEE_BONUS;
           END WHILE;
           CLOSE C1;
           SET DEPT_SALARY = TOTAL_SALARY;
           SET DEPT_BONUS_CNT = BONUS_CNT;
         END P1;
```

此 CREATE PROCEDURE 陳述式：

- 將程序命名爲 RETURN_DEPT_SALARY。
- 定義字元資料類型的輸入參數 DEPT_NUMBER，長度爲 3、十進位資料類型輸出參數 DEPT_SALARY 和整數資料類型輸出參數 DEPT_BONUS_CNT。
- 指示程序是讀取 SQL 資料的 SQL 程序。
- 定義程序主體。
  - 將 SQL 變數 EMPLOYEE_SALARY 和 TOTAL_SALARY 宣告爲小數欄位。

- 宣告起始設定爲 0 的整數 SQL 變數 BONUS_CNT 和 END_TABLE。
- 宣告從員工表格選取直欄的游標 C1。
- 宣告 NOT FOUND 的繼續處理程式，於被呼叫時將變數 END_TABLE 設定爲 1。FETCH 沒有可以再傳回的列時，就會呼叫這個處理程式。呼叫處理程式後，SQLCODE 和 SQLSTATE 會重新起始設定爲 0。
- 宣告 SQLEXCEPTION 的結束處理程式。若呼叫， DEPT_SALARY 會設定爲 NULL，而且複合陳述式的處理程序會被終止。如果發生任何錯誤 (SQLSTATE 類別不是 '00'、'01' 或 '02')，就呼叫這個處理程式。由於指示器一律會傳遞至 SQL 程序，因此在程序返回時，DEPT_SALARY 的指示器值是 -1。如果呼叫這個處理程式，SQLCODE 和 SQLSTATE 會重新起始設定爲 0。

  如果未指定 SQLEXCEPTION 的處理程式，而且發生未在其他處理程式中處理的錯誤，複合陳述式將會終止執行，同時 SQLCA 中會傳回錯誤。和指示器一樣，一律會從 SQL 程序傳回 SQLCA。
- 包含游標 C1 的 OPEN、FETCH 及 CLOSE。如果未指定游標的 CLOSE，則由於未在 CREATE PROCEDURE 陳述式中指定 SET RESULT SETS，游標會在複合陳述式結束時結束。
- 包含一個 WHILE 陳述式，會一直執行迴圈，直到提取最後一筆記錄爲止。擷取的每一列，TOTAL_SALARY 都會累加，而且員工的紅利若是大於 0，BONUS_CNT 也會累加。
- 傳回 DEPT_SALARY 和 DEPT_BONUS_CNT 做爲輸出參數。

可以使複合陳述式成爲原子，這樣一旦發生未預期的錯誤，可以回轉原子陳述式中的陳述式。原子複合陳述式是利用 SAVEPOINTS 實作。如果順利完成複合陳述式，就會確定異動。如需使用 SAVEPOINTS 的詳細資訊，請參閱第 290 頁的『儲存點』。

下列範例會以部門編號做爲輸入。它會確定 EMPLOYEE_BONUS 表格存在，並且插入部門中領到紅利之所有員工的姓名。程序會傳回領到紅利之所有員工的總計數。

```
CREATE PROCEDURE CREATE_BONUS_TABLE
        (IN DEPT_NUMBER CHAR(3),
         INOUT CNT INT)
         LANGUAGE SQL MODIFIES SQL DATA
          CS1: BEGIN ATOMIC
          DECLARE NAME VARCHAR(30) DEFAULT NULL;
          DECLARE CONTINUE HANDLER FOR SQLSTATE '42710'
             SELECT COUNT(*) INTO CNT
             FROM DATALIB.EMPLOYEE_BONUS;
          DECLARE CONTINUE HANDLER FOR SQLSTATE '23505'
             SET  CNT = CNT - 1;
          DECLARE UNDO HANDLER FOR SQLEXCEPTION
             SET CNT = NULL;
          IF DEPT_NUMBER IS NOT NULL THEN
            CREATE TABLE DATALIB.EMPLOYEE_BONUS
                 (FULLNAME VARCHAR(30),
                  BONUS DECIMAL(10,2),
                 PRIMARY KEY (FULLNAME));
          FOR_1:FOR V1 AS C1 CURSOR FOR
              SELECT FIRSTNME, MIDINIT, LASTNAME, BONUS
                 FROM CORPDATA.EMPLOYEE
                 WHERE WORKDEPT = CREATE_BONUS_TABLE.DEPT_NUMBER
             DO
             IF BONUS > 0 THEN
               SET NAME = FIRSTNME CONCAT ' ' CONCAT
                          MIDINIT  CONCAT ' 'CONCAT LASTNAME;
               INSERT INTO DATALIB.EMPLOYEE_BONUS
                  VALUES(CS1.NAME, FOR_1.BONUS);
```

```
        SET CNT = CNT + 1;
      END IF;
    END FOR FOR_1;
  END IF;
  END CS1
```

此 CREATE PROCEDURE 陳述式：

- 將程序命名爲 CREATE_BONUS_TABLE。
- 定義字元資料類型的輸入參數 DEPT_NUMBER，長度爲 3，以及整數資料類型的輸入 / 輸出參數 CNT。
- 指示程序爲修改 SQL 資料的 SQL 程序。
- 定義程序主體。
  - 宣告 SQL 變數 NAME 爲可變字元。
  - 宣告 SQLSTATE 42710 之繼續處理程式，表格已經存在。如果 EMPLOYEE_BONUS 表格已存在，會呼叫處理程式並擷取表格中的記錄數目。SQLCODE 和 SQLSTATE 會重設爲 0，處理程序繼續執行 FOR 陳述式。
  - 宣告 SQLSTATE 23505 的繼續處理程式，重複鍵。如果程序嘗試插入表格中已有的名稱，會呼叫處理程式，同時將 CNT 減量。處理程序會繼續在 INSERT 陳述式之後執行 SET 陳述式。
  - 宣告 SQLEXCEPTION 的 UNDO 處理程式。呼叫時會回轉前一個陳述式、CNT 設定爲 0，而且處理程序在複合陳述式之後繼續。在此例中，由於複合陳述式之後沒有陳述式，因此會傳回程序。
  - 使用 FOR 陳述式宣告游標 C1 從 EMPLOYEE 表格讀取記錄。 FOR 陳述式中使用選取清單的直欄名稱，做爲包含提取之列中之資料的 SQL 變數。每一個列中，FIRSTNME、MIDINIT 和 LASTNAME 直欄中的資料會以空格連接在一起，結果則放入 SQL 變數 NAME。 SQL 變數 NAME 和 BONUS 會插入 EMPLOYEE_BONUS 表格中。由於建立程序時必須知道選取清單項目的資料類型，因此在建立程序時， FOR 陳述式中指定的表格必須已存在。

    SQL 變數名稱可以使用定義此變數之 FOR 陳述式或複合陳述式的標籤名稱來限定。在此例中，FOR_1.BONUS 指的是包含每一選取列之 BONUS 直欄之值的 SQL 變數。 CS1.NAME 是在開頭標籤爲 CS1 的複合陳述式中定義的變數 NAME。也可以使用程序名稱限定參數名稱。CREATE_BONUS_TABLE.DEPT_NUMBER 是程序 CREATE_BONUS_TABLE 的 DEPT_NUMBER 參數。如果在也允許直欄名稱的 SQL 陳述式中使用未限定 SQL 變數名稱，而且變數名稱和直欄名稱相同，將會使用此名稱參照直欄。

您也可以在 SQL 程序中使用動態 SQL。下列範例會建立包含特定部門中全部員工的表格。部門編號會當成輸入傳遞至程序，並且連接至表格名稱。

```
CREATE PROCEDURE CREATE_DEPT_TABLE (IN P_DEPT CHAR(3))
        LANGUAGE SQL
   BEGIN
     DECLARE STMT CHAR(1000);
     DECLARE MESSAGE CHAR(20);
     DECLARE TABLE_NAME CHAR(30);
     DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
        SET MESSAGE = 'ok';
     SET TABLE_NAME = 'CORPDATA.DEPT_' CONCAT P_DEPT CONCAT '_T';
     SET STMT = 'DROP TABLE ' CONCAT TABLE_NAME;
     PREPARE S1 FROM STMT;
     EXECUTE S1;
        SET STMT = 'CREATE TABLE ' CONCAT TABLE_NAME CONCAT
```

```
      '( EMPNO CHAR(6) NOT NULL,
         FIRSTNME VARCHAR(12) NOT NULL,
         MIDINIT CHAR(1) NOT NULL,
         LASTNAME CHAR(15) NOT NULL,
         SALARY DECIMAL(9,2))';
    PREPARE S2 FROM STMT;
    EXECUTE S2;
    SET STMT = 'INSERT INTO ' CONCAT TABLE_NAME CONCAT
      'SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, SALARY
        FROM CORPDATA.EMPLOYEE
        WHERE  WORKDEPT = ?';
    PREPARE S3 FROM STMT;
    EXECUTE S3 USING P_DEPT;
END;
```

此 CREATE PROCEDURE 陳述式：

- 將程序命名爲 CREATE_DEPT_TABLE
- 定義字元資料類型的輸入參數 P_DEPT，長度爲 3。
- 指示程序是 SQL 程序。
- 定義程序主體。
  - 宣告 SQL 變數 STMT 和 SQL 變數 TABLE_NAME 爲字元。
  - 宣告 CONTINUE 處理程式。如果表格已經存在，程序會嘗試 DROP 表格。如果表格不存在，第一個 EXECUTE 會失敗。處理程式部分的處理程序將會繼續。
  - 將變數 TABLE_NAME 設定爲 'DEPT_'，後接參數 P_DEPT 中傳遞的字元，再接 '_T'。
  - 將變數 STMT 設定至 DROP 陳述式，並且準備及執行陳述式。
  - 將變數 STMT 設定至 CREATE 陳述式，並且準備及執行陳述式。
  - 將變數 STMT 設定至 INSERT 陳述式，並且準備及執行陳述式。 WHERE 子句中指定了參數記號。執行陳述式時，USING 子句會傳遞變數 P_DEPT。

如果呼叫程序，傳遞部門值 'D21'，將會建立表格 DEPT_D21_T，並且以部門 'D21' 中所有員工起始設定表格。

爲儲存程序除錯

在您建立 SQL 程序、建立 SQL 函數或建立觸發陳述式中指定 SET OPTION DBGVIEW = *SOURCE，就可在 SQL 陳述式層次爲產生的程式或模組除錯。也可以指定 DBGVIEW(*SOURCE) 爲 RUNSQLSTM 指令的參數，它將會套用至 RUNSQLSTM 中所有的常式。

系統會利用您的原始常式主體，在 QTEMP/QSQDSRC 中建立來源概略表。來源概略表不會和程式或服務程式一起儲存。它會分割成一些程式碼行，這些程式碼行對應至可在除錯時停止的位置。文字 (包括參數和變數名稱) 會變爲大寫。

所有的變數和參數都當作結構的一部分產生。在除錯中評估變數時，必須使用結構名稱。變數由目前的標籤名稱限定。參數則由程序或函數名稱限定。觸發程式中的轉移變數由適當的相關名稱限定。強烈建議您爲每一個複合陳述式或 FOR 陳述式指定一個標籤名稱。如果未指定，系統爲替您產生。這樣一來幾乎不可能去評估變數。請記得，所有變數和參數都必須以大寫名稱評估。您也可以評估結構的名稱，這會顯示結構中所有的變數。如果變數或參數可爲空值，該變數或或參數的指示器在結構中會緊接在其後面。

由於 SQL 常式是以 C 產生，因此 C 當中的一些限制也會影響 SQL 原始程式除錯。在 SQL 常式主體中指定的分隔名稱不能在 C 中指定。系統會爲這些名稱產生名稱，這一來同樣很難除錯或評估。爲了評估任何字元變數的內容，請在變數名稱的前面指定一個 *。

由於系統會爲大部分的變數和參數名稱產生指示器，因此無法直接檢查變數是否有 SQL 空值。即使指示器設定爲指示空值，不過評估變數時一定會顯示值。

爲了判斷處理程式是否被呼叫，請在處理程式中第一個陳述式設定一個岔斷點。在複合陳述式或處理程式的 FOR 陳述式中宣告的變數都可以評估。

呼叫儲存程序

SQL CALL 陳述式會呼叫儲存程序。CALL 陳述式上會指定儲存程序和任何引數的名稱。引數可以是常數、特別暫存區或主電腦。CALL 陳述式中指定的外部儲存程序不需要有對應的 CREATE PROCEDURE 陳述式。SQL 程序建立的程式，只能利用呼叫 CREATE PROCEDURE 陳述式中指定之程序名稱來呼叫。

雖然程序是系統程式物件，但是使用 CALL CL 指令通常不能呼叫程序。 CALL CL 指令並不使用程序定義以對映輸入和輸出參數，也不會使用程序的參數樣式傳遞參數給程式。

DB2 SQL for iSeries 對每一種類型有不同的規則，因此有三種 CALL 陳述式必須處理。分別是：

- 有程序定義存在的內含或動態 CALL 陳述式
- 沒有程序定義存在的內含 CALL 陳述式
- 沒有 CREATE PROCEDURE 存在的動態 CALL 陳述式

註: 這裡的動態指的是：

- 動態準備及執行的 CALL 陳述式
- 在交談式環境中 (例如透過 STRSQL 或查詢管理程式) 發出的 CALL 陳述式
- 在 EXECUTE IMMEDIATE 陳述式中執行的 CALL 陳述式。

以下是每一種類型的討論。

- 『使用有程序定義存在的 CALL 陳述式』
- 第 155 頁的『使用沒有程序定義存在的內含 CALL 陳述式』
- 第 156 頁的『使用具有 SQLDA 的內含 CALL 陳述式』
- 第 157 頁的『使用沒有 CREATE PROCEDURE 存在的動態 CALL 陳述式』

使用有程序定義存在的 CALL 陳述式

這一類 CALL 陳述式會從 CREATE PROCEDURE 型錄定義讀取程序和引數屬性相關的一切資訊。以下 PL/I 範例顯示對應至所示 CREATE PROCEDURE 陳述式的 CALL 陳述式。

```
DCL HV1 CHAR(10);
DCL IND1 FIXED BIN(15);
   :
EXEC SQL CREATE P1 PROCEDURE
          (INOUT PARM1 CHAR(10))
          EXTERNAL NAME MYLIB.PROC1
```

```
          LANGUAGE C
          GENERAL WITH NULLS;
   :
EXEC SQL CALL P1 (:HV1 :IND1);
   :
```

呼叫這個 CALL 陳述式時，會進行程式 MYLIB/PROC1 的呼叫，並傳遞兩個引數。由於程式語言是 ILE C，因此第一個引數是十一個字元長之 C 的 NUL 終止字串，包含主變數 HV1 的內容。請注意，在對 ILE C 程序的呼叫上，如果參數宣告爲字元、圖形、日期、時間或時間戳記變數，DB2 SQL for iSeries 會在參數宣告增加一個字元。第二個引數是指示器陣列。由於此例中的 CREATE PROCEDURE 陳述式只有一個參數，因此這是一個短整數。這個引數包含程序登錄之指示器變數 IND1 的內容。

因爲第一個參數宣告爲 INOUT，所以 SQL 會先以 MYLIB.PROC1 傳回的值更新主變數 HV1 和指示器變數 IND1，再返回使用者程式。

註:

1. CREATE PROCEDURE 和 CALL 陳述式上指定的程序名稱必須完全相符，才能在 SQL 前置編譯程式期間進行兩者之間的連結。
2. 若是 CREATE PROCEDURE 和 DECLARE PROCEDURE 陳述式同時存在的內含 CALL 陳述式，將會使用 DECLARE PROCEDURE 陳述式。

使用沒有程序定義存在的內含 CALL 陳述式

沒有對應 CREATE PROCEDURE 陳述式的靜態 CALL 陳述式會以下列規則處理：

- 所有的主變數引數都當成 INOUT 類型參數處理。
- CALL 類型是 GENERAL (不會傳遞指示器引數)。
- 根據 CALL 上指定的程序名稱 (必要時再加上命名慣例) 決定要呼叫的程式。
- 根據從系統擷取的程式相關資訊決定要呼叫的程式語言。

範例：沒有程序定義存在的內含 CALL 陳述式

註: 如需程式碼範例的相關資訊，請參閱第 x 頁的『程式碼不保事項聲明』的資訊。

以下是沒有程序定義存在之內含 CALL 陳述式的 PL/I 範例：

```
DCL HV2 CHAR(10);
   :
EXEC SQL CALL P2 (:HV2);
   :
```

呼叫 CALL 陳述式時，DB2 SQL for iSeries 會根據標準 SQL 命名慣例嘗試尋找程式。以上面的範例而言，假設使用 *SYS (系統命名) 命名選項，而且未在 CRTSQLPLI 指令上指定 DFTRDBCOL 參數。在此例中，會搜尋名爲 P2 之程式的程式庫清單。由於呼叫類型是 GENERAL，因此不會傳遞指示器變數的其他引數至程式。

註: 如果在 CALL 陳述式上指定了指示器變數，而且在執行 CALL 陳述式時其值小於零，會因爲無法傳遞指示器至程序而造成錯誤。

假設在程式庫清單中找到程式 P2，主變數 HV2 的內容會傳遞至 CALL 的程式，而且在 P2 執行完畢後，從 P2 傳回的引數會對映回主變數。

使用具有 SQLDA 的內含 CALL 陳述式

不論是哪一種內含 CALL (程序定義存在或不存在)，都可以傳遞 SQLDA，而非傳遞參數清單，如以下 C 範例的說明。假設儲存程序需要 2 個參數，第一個是 SHORT INT 類型，第二個是長度爲 4 的 CHAR 類型。

```
#define SQLDA_HV_ENTRIES 2
#define SHORTINT 500
#define NUL_TERM_CHAR 460

exec sql include sqlca;
exec sql include sqlda;
...
typedef struct sqlda Sqlda;
typedef struct sqlda* Sqldap;
...
main()
{
 Sqldap dap;
 short col1;
 char col2[4];
 int bc;
 dap = (Sqldap) malloc(bc=SQLDASIZE(SQLDA_HV_ENTRIES));
        /* SQLDASIZE 為定義於 sqlda include 中的巨集 */
 col1 = 431;
 strcpy(col2,"abc");
 strncpy(dap->sqldaid,"SQLDA   ",8);
 dap->sqldabc = bc;         /* bc 設定於上述之 malloc 陳述式中 */
 dap->sqln = SQLDA_HV_ENTRIES;
 dap->sqld = SQLDA_HV_ENTRIES;
 dap->sqlvar[0].sqltype = SHORTINT;
 dap->sqlvar[0].sqllen  = 2;
 dap->sqlvar[0].sqldata = (char*) &col1;
 dap->sqlvar[0].sqlname.length = 0;
 dap->sqlvar[1].sqltype = NUL_TERM_CHAR;
 dap->sqlvar[1].sqllen  = 4;
 dap->sqlvar[1].sqldata = col2;
 ...
 EXEC SQL CALL P1 USING DESCRIPTOR :*dap;
 ...
}
```

呼叫之程序的名稱也可以儲存在主變數中，主變數用於 CALL 陳述式中，代替寫死在程式中的程序名稱。例如：

```
...
main()
{
 char proc_name[15];
 ...
 strcpy (proc_name, "MYLIB.P3");
 ...
 EXEC SQL CALL :proc_name ...;
 ...
}
```

上面的範例中，如果 MYLIB.P3 是預期的參數，則可以使用參數清單或以 USING DESCRIPTOR 子句傳遞的 SQLDA，如前面的範例所示。

如果 CALL 陳述式中使用包含程序名稱的主變數，而且 CREATE PROCEDURE 型錄定義存在，將會使用該定義。不能將程序名稱指定爲參數記號。

本章後面會有呼叫儲存程序的更多範例。

使用沒有 CREATE PROCEDURE 存在的動態 CALL 陳述式

下列規則專屬於沒有 CREATE PROCEDURE 定義時的動態 CALL 陳述式處理程序：

- 所有引數都當成 IN 類型參數。
- CALL 類型是 GENERAL (不會傳遞指示器引數)。
- 根據在 CALL 指定的程序名稱和命名慣例決定要呼叫的程式。
- 根據從系統擷取的程式相關資訊決定要呼叫的程式之語言。

範例：沒有 CREATE PROCEDURE 存在的動態 CALL 呼叫

註: 如需程式碼範例的相關資訊，請參閱第 x 頁的『程式碼不保事項聲明』的資訊。

以下是動態 CALL 陳述式的 C 範例：

```
char hv3[10],string[100];
    :
strcpy(string,"CALL MYLIB.P3 ('P3 TEST')");
EXEC SQL EXECUTE IMMEDIATE :string;
    :
```

這個範例顯示透過 EXECUTE IMMEDIATE 陳述式執行的動態 CALL 陳述式。以當成字串變數 (包含 'P3 TEST') 傳遞的一個參數呼叫程式 MYLIB.P3。

執行 CALL 陳述式並傳遞常數時 (如前一個範例所示)，必須記住程式中預期的引數之長度。如果程式 MYLIB.P3 預期的是只有 5 個字元的引數，程式會遺失範例中指定之常數的最後 2 個字元。

註: 因此，較保險的方法是在 CALL 陳述式使用主變數，使程序的屬性能夠完全相符，字元也不至於遺失。如果使用 PREPARE 和 EXECUTE 陳述式處理動態 SQL，可以爲 CALL 陳述式引數指定主變數。

CALL 陳述式傳遞的數值常數適用下列規則：

- 所有整數常數都傳遞爲全字組二進位整數。
- 所有十進位常數都傳遞爲壓縮十進位數值。精準度和小數位數則根據常數值決定。例如，123.45 的值會傳遞爲壓縮十進位數 (5,2)。同樣地，001.01 的值也分別傳遞爲精準度 5 和小數位數 2。
- 所有浮點常數都以倍精準度浮點傳遞。

動態 CALL 陳述式上指定的特別暫存區傳遞方式如下：

CURRENT DATE
以 ISO 格式的 10 位元組字串傳遞。

CURRENT TIME
以 ISO 格式的 8 位元組字串傳遞。

CURRENT TIMESTAMP
以 IBM SQL 格式的 26 位元組字串傳遞。

CURRENT TIMEZONE
以精確度 6 小數位數及 0 的壓縮十進位數傳遞。

CURRENT SCHEMA
以 128 位元組的可變長度字串傳遞。

CURRENT SERVER
以 18 位元組的可變長度字串傳遞。

USER
以 18 位元組的可變長度字串傳遞。

CURRENT PATH
以 3483 位元組的可變長度字串傳遞。

儲存程序和 UDF 的參數傳遞慣例

CALL 陳述式可以傳遞引數至以所有支援主語言撰寫的程式和 REXX 程序。每一種語言支援適合該語言的不同資料類型。SQL 資料類型包含在以下表格的最左直欄。該列的其他直欄包含一個指示，指示是否支援該資料類型做爲特定語言的參數類型。如果直欄包含破折號 (-)，則不支援該資料類型做爲該語言的參數類型。主變數宣告指示 DB2 SQL for iSeries 支援這種資料類型做爲此語言的參數。宣告指示應如何宣告主變數，才能讓程序正確接收到及設定。呼叫 SQL 程序時支援所有的 SQL 資料類型，因此表格中未提供任何直欄。

如需詳細資料，請參閱 SQL Programming for Host Languages 一書和 IBM Developer's Kit for Java 中之 Java SQL routines 一節。

表 25. 參數的資料類型

SQL 資料類型	C 和 C++	CL	COBOL for iSeries 和 ILE COBOL for iSeries
SMALLINT	short	-	PIC S9(4) BINARY
INTEGER	long	-	PIC S9(9) BINARY
BIGINT	long long	-	PIC S9(18) BINARY **註**：只支援 ILE COBOL for iSeries。
DECIMAL(p,s)	decimal(p,s)	TYPE(*DEC) LEN(p s)	PIC S9(p-s)V9(s) PACKED-DECIMAL **註**：精準度不能大於 18。
NUMERIC(p,s)	-	-	PIC S9(p-s)V9(s) DISPLAY SIGN LEADING SEPARATE **註**：精準度不能大於 18。
REAL 或 FLOAT(p)	float	-	COMP-1 **註**：只支援 ILE COBOL for iSeries。
DOUBLE PRECISION 或 FLOAT 或 FLOAT(p)	double	-	COMP-2 **註**：只支援 ILE COBOL for iSeries。
CHARACTER(n)	char ... [n+1]	TYPE(*CHAR) LEN(n)	PIC X(n)
VARCHAR(n)	char ... [n+1]	-	可變長度字元字串 (請參閱 SQL Programming with Host Languages 中介紹 COBOL 的一章)。
VARCHAR(n) FOR BIT DATA	VARCHAR 結構化形式 (請參閱 SQL Programming for Host Languages 一書中介紹 C 的一章)。	-	可變長度字元字串 (請參閱 SQL Programming with Host Languages 中介紹 COBOL 的一章)。

表 25. 參數的資料類型 (繼續)

SQL 資料類型	C 和 C++	CL	COBOL for iSeries 和 ILE COBOL for iSeries
GRAPHIC(n)	wchar_t ... [n+1]	-	PIC G(n) DISPLAY-1 或 PIC N(n) **註**：只支援 ILE COBOL for iSeries。
VARGRAPHIC(n)	VARGRAPHIC 結構化形式 (請參閱 C 的一章)	-	可變長度圖形字串 (請參閱 SQL Programming with Host Languages 中介紹 COBOL 的一章)。 **註**：只支援 ILE COBOL for iSeries。
DATE	char ... [11]	TYPE(*CHAR) LEN(10)	PIC X(10) 只限 ILE COBOL for iSeries，FORMAT DATE。
TIME	char ... [9]	TYPE(*CHAR) LEN(8)	PIC X(8) 只限 ILE COBOL for iSeries，FORMAT TIME。
TIMESTAMP	char ... [27]	TYPE(*CHAR) LEN(26)	PIC X(26) 只限 ILE COBOL for iSeries，FORMAT TIMESTAMP。
CLOB	CLOB 結構化形式 (請參閱 SQL Programming with Host Languages 中介紹 C 的一章)。	-	CLOB 結構化形式 (請參閱 SQL Programming with Host Languages 中介紹 COBOL 的一章)。**註**：只支援 ILE COBOL for iSeries。
BLOB	BLOB 結構化形式 (請參閱 SQL Programming with Host Languages 中介紹 C 的一章)。	-	BLOB 結構化形式 (請參閱 SQL Programming with Host Languages 中介紹 COBOL 的一章)。**註**：只支援 ILE COBOL for iSeries。
DBCLOB	DBCLOB 結構化形式 (請參閱 SQL Programming with Host Languages 中介紹 C 的一章)。	-	DBCLOB 結構化形式 (請參閱 SQL Programming with Host Languages 中介紹 COBOL 的一章)。**註**：只支援 ILE COBOL for iSeries。
ROWID	ROWID 結構化形式 (請參閱 SQL Programming with Host Languages 中介紹 C 的一章)。	-	ROWID 結構化形式 (請參閱 SQL Programming with Host Languages 中介紹 COBOL 的一章)。
DataLink	-	-	-
指示器變數	short	-	PIC S9(4) BINARY

表 26. 參數的資料類型

SQL 資料類型	FORTRAN	Java 參數樣式 JAVA	Java 參數樣式 DB2GENERAL	PL/I
SMALLINT	INTEGER*2	short	short	FIXED BIN(15)

表 26. 參數的資料類型 (繼續)

SQL 資料類型	FORTRAN	Java 參數樣式 JAVA	Java 參數樣式 DB2GENERAL	PL/I
INTEGER	INTEGER*4	int	int	FIXED BIN(31)
BIGINT	-	long	long	-
DECIMAL(p,s)	-	BigDecimal	BigDecimal	FIXED DEC(p,s)
NUMERIC(p,s)	-	BigDecimal	BigDecimal	-
REAL 或 FLOAT(p)	REAL*4	float	float	FLOAT BIN(p)
DOUBLE PRECISION 或 FLOAT 或 FLOAT(p)	REAL*8	double	double	FLOAT BIN(p)
CHARACTER(n)	CHARACTER*n	String	String	CHAR(n)
VARCHAR(n)	-	String	String	CHAR(n) VAR
VARCHAR(n) FOR BIT DATA	-	-	com.ibm.db2.app.Blob	CHAR(n) VAR
GRAPHIC(n)	-	String	String	-
VARGRAPHIC(n)	-	String	String	-
DATE	CHARACTER*10	Date	String	CHAR(10)
TIME	CHARACTER*8	Time	String	CHAR(8)
TIMESTAMP	CHARACTER*26	Timestamp	String	CHAR(26)
CLOB	-	-	com.ibm.db2.app.Clob	CLOB 結構化形式 (請參閱 SQL Programming with Host Languages 中介紹 PL/I 的一章)。
BLOB	-	-	com.ibm.db2.app.Blob	BLOB 結構化形式 (請參閱 SQL Programming with Host Languages 中介紹 PL/I 的一章)。
DBCLOB	-	-	com.ibm.db2.app.Clob	DBCLOB 結構化形式 (請參閱 SQL Programming with Host Languages 中介紹 PL/I 的一章)。
ROWID	-	-	-	ROWID 結構化形式 (請參閱 SQL Programming with Host Languages 中介紹 PL/I 的一章)。
DataLink	-	-	-	-
Indicator Variable	INTEGER*2	-	-	FIXED BIN(15)

表 27. 參數的資料類型

SQL 資料類型	REXX	RPG	ILE RPG
SMALLINT	-	包含單一次欄位的資料結構。次欄位規格的位置 43 爲 ***B***，長度必須是 2，而位置 52 則爲 0。	資料規格。次欄位規格的位置 40 爲 ***B***，長度必須 <= 4，而位置 41-42 則爲 00。 或 資料規格。次欄位規格的位置 40 爲 ***I***，長度必須爲 5，而位置 41-42 則爲 00。
INTEGER	沒有小數的數字字串 (和選用的前置符號)	包含單一次欄位的資料結構。次欄位規格的位置 43 爲 ***B***，長度必須爲 4，而位置 52 則爲 0。	資料規格。次欄位規格的位置 40 爲 ***B***，長度必須爲 <=09 和 >=05，而位置 41-42 則爲 00。 或 資料規格。次欄位規格的位置 40 爲 ***I***，長度必須爲 10，而位置 41-42 中則爲 00。
BIGINT	-	-	資料規格。次欄位規格的位置 40 爲 ***I***，長度必須爲 20，而位置 41-42 中則爲 00。
DECIMAL(p,s)	有小數的數字字串 (和選用的前置符號)	包含單一次欄位的資料結構。次欄位規格的位置 43 爲 ***P***，而在位置 52 則爲 0 至 9。或數字輸入欄位或計算結果欄位。	資料規格。次欄位規格的位置 40 爲 ***P***，而在位置 41-42 則爲 00 至 31。
NUMERIC(p,s)	-	包含單一次欄位的資料結構。次欄位規格的位置 43 爲***空白***，而在位置 52 則爲 0 至 9。	資料規格。次欄位規格的位置 40 爲 ***S***，或在位置 40 爲***空白***，而在位置 41-42 則爲 00 至 31。
REAL 或 FLOAT(p)	先是數字，再來是 E，(接著是選用的符號)，再來是數字的字串	-	資料規格。位置 40 爲 ***F***，長度必須是 4。
DOUBLE PRECISION 或 FLOAT 或 FLOAT(p)	先是數字，再來是 E，(接著是選用的符號)，再來是數字的字串	-	資料規格。位置 40 爲 ***F***，長度必須是 8。
CHARACTER(n)	以兩個單引號括住的 n 個字元的字串	沒有次欄位的資料結構欄位，或包含單一次欄位的資料結構。在次欄位規格的位置 43 和 52 爲***空白***。或 A 字元輸入欄位或計算結果欄位。	資料規格。次欄位規格的位置 40 爲 ***A***，或在位置 40 和 41-42 爲 ***空白***。
VARCHAR(n)	以兩個單引號括住的 n 個字元的字串	-	資料規格。次欄位規格的位置 40 爲 ***A***，或在位置 40 和 41-42 爲 ***空白***，在位置 44-80 爲關鍵字 VARYING。
VARCHAR(n) FOR BIT DATA	以兩個單引號括住的 n 個字元的字串	-	資料規格。次欄位規格的位置 40 爲 ***A***，或位置 40 和 41-42 爲***空白***，而位置 44-80 爲關鍵字 VARYING。

表 27. 參數的資料類型 (繼續)

SQL 資料類型	REXX	RPG	ILE RPG
GRAPHIC(n)	以 G' 開始，然後是 n 個雙位元組字元，然後是 ' 的字串	-	資料規格。次欄位規格的位置 40 爲 ***G***。
VARGRAPHIC(n)	以 G' 開始，然後是 n 個雙位元組字元，然後是 ' 的字串	-	資料規格。次欄位規格的位置 40 爲 ***G***，位置 44-80 爲關鍵字 VARYING。
DATE	以兩個單引號括住的 10 個字元的字串	沒有次欄位的資料結構欄位，或包含單一次欄位的資料結構。在次欄位規格的位置 43 和 52 爲***空白***。長度是 10。或 A 字元輸入欄位或計算結果欄位。	資料規格。次欄位規格的位置 40 爲 ***D***。位置 44-80 爲 DATFMT(*ISO)。
TIME	以兩個單引號括住的 8 個字元的字串	沒有次欄位的資料結構欄位，或包含單一次欄位的資料結構。在次欄位規格的位置 43 和 52 爲***空白***。長度是 8。或 A 字元輸入欄位或計算結果欄位。	資料規格。次欄位規格的位置 40 爲 ***T***。位置 44-80 爲 TIMFMT(*ISO)。
TIMESTAMP	以兩個單引號括住的 26 個字元字串	沒有次欄位的資料結構欄位，或包含單一次欄位的資料結構。在次欄位規格的位置 43 和 52 爲***空白***。長度是 26。或字元輸入欄位或計算結果欄位。	資料規格。次欄位規格的位置 40 爲 ***Z***。
CLOB	-	-	CLOB 結構化形式 (請參閱 SQL Programming with Host Languages 中介紹 RPG 的一章)
BLOB	-	-	BLOB 結構化形式 (請參閱 SQL Programming with Host Languages 中介紹 RPG 的一章)
DBCLOB	-	-	DBCLOB 結構化形式 (請參閱 SQL Programming with Host Languages 中介紹 RPG 的一章)
ROWID	-	-	ROWID 結構化形式 (請參閱 SQL Programming with Host Languages 中介紹 RPG 的一章)
DataLink	-	-	-
指示器變數	沒有小數的數字字串 (和選取的前置符號)。	包含單一次欄位的資料結構。次欄位規格的位置 43 爲 ***B***，長度必須是 2，而位置 52 則爲 0。	資料規格。次欄位規格的位置 40 爲 ***B***，長度必須 <=4，而位置 41-42 則爲 00。

指示器變數和儲存程序

假如參數使用主變數，則指示器變數可配合 CALL 陳述式使用，以傳遞附加資訊至程序 (和從程序傳遞資訊過來)。指示器變數是 SQL 表示應將相關的主變數解譯爲包含 NULL 值的標準方法，這也是其主要用途。

若要指示相關的主變數包含 NULL 值，主變數 (雙位元組的整數) 會設定爲負值。含有指示器變數之 CALL 陳述式的處理方式如下：

- 如果指示器變數是負的，這表示 NULL 值。CALL 會傳遞相關主變數的預設值，而指示器變數則保持不變傳遞。
- 如果指示器變數不是負的，這表示主變數包含非 NULL 值。於此例中，主變數和指示器變數都保持不變傳遞。

這些處理程序規則與輸入到程序的輸入參數和從程序傳回的輸出參數相同。指示器變數配合儲存程序使用時，撰寫其處理程式的正確方法是先檢查指示器變數的值，再使用相關的主變數。

以下範例說明在 CALL 陳述式中處理指示器變數。請注意，邏輯會在使用相關變數之前檢查指示器變數的值。另外也請注意主變數傳遞至程序 PROC1 的方法 (當成構成雙位元組值之陣列的第三個引數)。

假設程序定義如下：

```
CREATE PROCEDURE PROC1
   (INOUT DECIMALOUT DECIMAL(7,2), INOUT DECOUT2 DECIMAL(7,2))
   EXTERNAL NAME LIB1.PROC1 LANGUAGE RPGLE
   GENERAL WITH NULLS)
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Program CRPG
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    D INOUT1          S              7P 2
    D INOUT1IND       S              4B 0
    D INOUT2          S              7P 2
    D INOUT2IND       S              4B 0
    C                   EVAL      INOUT1 = 1
    C                   EVAL      INOUT1IND  = 0
    C                   EVAL      INOUT2 = 1
    C                   EVAL      INOUT2IND = -2
    C/EXEC SQL CALL PROC1 (:INOUT1 :INOUT1IND , :INOUT2
    C+                        :INOUT2IND)
    C/END-EXEC
    C                   EVAL      INOUT1 = 1
    C                   EVAL      INOUT1IND  = 0
    C                   EVAL      INOUT2 = 1
    C                   EVAL      INOUT2IND = -2
    C/EXEC SQL CALL PROC1 (:INOUT1 :INOUT1IND , :INOUT2
    C+                        :INOUT2IND)
    C/END-EXEC
    C     INOUT1IND     IFLT      0
    C*                   :
    C*                  HANDLE NULL INDICATOR
    C*                   :
    C                   ELSE
    C*                   :
    C*                  INOUT1 CONTAINS VALID DATA
    C*                   :
    C                   ENDIF
    C*                   :
    C*                  HANDLE ALL OTHER PARAMETERS
    C*                  IN A SIMILAR FASHION
    C*                   :
    C                   RETURN
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
End of PROGRAM CRPG
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

圖 3. 處理 CALL 陳述式中的指示器變數 (1/2)

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Program PROC1
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    D INOUTP          S              7P 2
    D INOUTP2         S              7P 2
    D NULLARRAY       S              4B 0 DIM(2)
    C     *ENTRY        PLIST
    C                   PARM                    INOUTP
    C                   PARM                    INOUTP2
    C                   PARM                    NULLARRAY
    C     NULLARRAY(1)  IFLT      0
    C*                   :
    C*                  INOUTP DOES NOT CONTAIN MEANINGFUL DATA
    C*
    C                   ELSE
    C*                   :
    C*                  INOUTP CONTAINS MEANINGFUL DATA
    C*                   :
    C                   ENDIF
    C*                  PROCESS ALL REMAINING VARIABLES
    C*
    C*                  BEFORE RETURNING, SET OUTPUT VALUE FOR FIRST
    C*                  PARAMETER AND SET THE INDICATOR TO A NON-NEGATIV
    C*                  VALUE SO THAT THE DATA IS RETURNED TO THE CALLING
    C*                  PROGRAM
    C*
    C                   EVAL      INOUTP2 = 20.5
    C                   EVAL      NULLARRAY(2) = 0
    C*
    C*                  INDICATE THAT THE SECOND PARAMETER IS TO CONTAIN
    C*                  THE NULL VALUE UPON RETURN. THERE IS NO POINT
    C*                  IN SETTING THE VALUE IN INOUTP SINCE IT WON'T BE
    C*                  PASSED BACK TO THE CALLER.
    C                   EVAL      NULLARRAY(1) = -5
    C                   RETURN
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
End of PROGRAM PROC1
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

圖 3. 處理 CALL 陳述式中的指示器變數 (2/2)

傳回完成狀態至呼叫程式

SQL 程序中任何未處理的錯誤，都會傳回 SQLCA 中的呼叫程式。 SIGNAL 和 RESIGNAL 控制陳述式也可用於傳送錯誤資訊。如需詳細資訊，請參閱 SQL Reference 中的 SQL Procedures, Functions, and Triggers 主題。

若是外部程序，有兩個方法可以傳回狀態資訊。傳回狀態給發出 CALL 陳述式之 SQL 程式的一種方法，是加入額外的一個 INOUT 類型參數，並在從程序傳回之前設定好。接受呼叫的程序若是現有程式，這不一定可行。

傳回狀態給發出 CALL 陳述式之 SQL 程式的另一種方法，是傳送跳離訊息到呼叫程序的呼叫程式 (作業系統程式 QSQCALL)。呼叫程序的呼叫程式是 QSQCALL。每一種語言都有通知條件和傳送訊息的一些方法。請參照各種語言的參照，以決定通知訊息的正確方法。通知訊息之後，QSQCALL 會將錯誤變成 SQLCODE/SQLSTATE -443/38501。

CALL 陳述式的範例

這些範例顯示如何將 CALL 陳述式的引數傳遞至許多種語言的程序。同時還顯示如何將引數接收至程序中的區域變數。

第一個範例顯示使用 CREATE PROCEDURE 定義呼叫 P1 和 P2 程序的 ILE C 程式。程序 P1 是以 C 寫成，有 10 個參數。 程序 P2 是以 PL/I 寫成，也有 10 個參數。

假設兩個程序定義如下：

```
EXEC SQL CREATE PROCEDURE P1 (INOUT PARM1 CHAR(10),
                              INOUT PARM2 INTEGER,
                              INOUT PARM3 SMALLINT,
                              INOUT PARM4 FLOAT(22),
                              INOUT PARM5 FLOAT(53),
                              INOUT PARM6 DECIMAL(10,5),
                              INOUT PARM7 VARCHAR(10),
                              INOUT PARM8 DATE,
                              INOUT PARM9 TIME,
                              INOUT PARM10 TIMESTAMP)
          EXTERNAL NAME  TEST12.CALLPROC2
          LANGUAGE C GENERAL WITH NULLS

EXEC SQL CREATE PROCEDURE P2 (INOUT PARM1 CHAR(10),
                              INOUT PARM2 INTEGER,
                              INOUT PARM3 SMALLINT,
                              INOUT PARM4 FLOAT(22),
                              INOUT PARM5 FLOAT(53),
                              INOUT PARM6 DECIMAL(10,5),
                              INOUT PARM7 VARCHAR(10),
                              INOUT PARM8 DATE,
                              INOUT PARM9 TIME,
                              INOUT PARM10 TIMESTAMP)
          EXTERNAL NAME  TEST12.CALLPROC
          LANGUAGE PLI GENERAL WITH NULLS
```

範例 1：從 ILE C 應用程式呼叫的 ILE C 和 PL/I 程序

註: 如需程式碼範例的相關資訊，請參閱第 x 頁的『程式碼不保事項聲明』的資訊。

```
/*************************************************************/
/*********** START OF SQL C Application **********************/

#include <stdio.h>
#include <string.h>
#include <decimal.h>
main()
{
 EXEC SQL INCLUDE SQLCA;
 char PARM1[10];
 signed long int PARM2;
 signed short int PARM3;
 float PARM4;
 double PARM5;
 decimal(10,5) PARM6;
 struct { signed short int parm7l;
          char parm7c[10];
        } PARM7;
char PARM8[10];        /* FOR DATE */
char PARM9[8];         /* FOR TIME */
char PARM10[26];       /* FOR TIMESTAMP */
```

圖 4. CREATE PROCEDURE 和 CALL 的範例 (1/2)

```
/*****************************************************/
/* Initialize variables for the call to the procedures */
/*****************************************************/
strcpy(PARM1,"PARM1");
PARM2 = 7000;
PARM3 = -1;
PARM4 = 1.2;
PARM5 = 1.0;
PARM6 = 10.555;
PARM7.parm7l = 5;
strcpy(PARM7.parm7c,"PARM7");
strncpy(PARM8,"1994-12-31",10);         /* FOR DATE      */
strncpy(PARM9,"12.00.00",8);            /* FOR TIME      */
strncpy(PARM10,"1994-12-31-12.00.00.000000",26);
                                        /* FOR TIMESTAMP */
/*********************************************/
/* Call the C procedure                      */
/*                                           */
/*                                           */
/*********************************************/
EXEC SQL CALL P1 (:PARM1, :PARM2, :PARM3,
                  :PARM4, :PARM5, :PARM6,
                  :PARM7, :PARM8, :PARM9,
                  :PARM10 );
if (strncmp(SQLSTATE,"00000",5))
 {
 /* Handle error or warning returned on CALL statement */
 }

/* Process return values from the CALL.                */
:


/*********************************************/
/* Call the PLI procedure                    */
/*                                           */
/*                                           */
/*********************************************/
/* Reset the host variables prior to making the CALL    */
/*                                                      */
:
EXEC SQL CALL P2 (:PARM1, :PARM2, :PARM3,
                  :PARM4, :PARM5, :PARM6,
                  :PARM7, :PARM8, :PARM9,
                  :PARM10 );
if (strncmp(SQLSTATE,"00000",5))
 {
  /* Handle error or warning returned on CALL statement */
}
/* Process return values from the CALL.                */
:
}

/******** END OF C APPLICATION *********************************/
/***************************************************************/
```

圖 4. CREATE PROCEDURE 和 CALL 的範例 (2/2)

```
/******** START OF C PROCEDURE P1 ******************************/
/*        PROGRAM TEST12/CALLPROC2                              */
/************************************************************/

#include <stdio.h>
#include <string.h>
#include <decimal.h>
main(argc,argv)
  int argc;
  char *argv[];
  {
    char parm1[11];
    long int parm2;
    short int parm3,i,j,*ind,ind1,ind2,ind3,ind4,ind5,ind6,ind7,
              ind8,ind9,ind10;
    float parm4;
    double parm5;
    decimal(10,5) parm6;
    char parm7[11];
    char parm8[10];
    char parm9[8];
    char parm10[26];
    /* ********************************************************/
    /* Receive the parameters into the local variables -       */
    /* Character, date, time, and timestamp are passed as      */
    /* NUL terminated strings - cast the argument vector to    */
    /* the proper data type for each variable. Note that       */
    /* the argument vector could be used directly instead of   */
    /* copying the parameters into local variables - the copy  */
    /* is done here just to illustrate the method.             */
    /* ********************************************************/

    /* Copy 10 byte character string into local variable      */
    strcpy(parm1,argv[1]);

    /* Copy 4 byte integer into local variable                */
    parm2 = *(int *) argv[2];

    /* Copy 2 byte integer into local variable                */
    parm3 = *(short int *) argv[3];

    /* Copy floating point number into local variable         */
    parm4 = *(float *) argv[4];

    /* Copy double precision number into local variable       */
    parm5 = *(double *) argv[5];

    /* Copy decimal number into local variable                */
    parm6 = *(decimal(10,5) *) argv[6];
```

圖 5. 範例程序 P1 (1/2)

```
/*********************************************************/
/* Copy NUL terminated string into local variable.         */
/* Note that the parameter in the CREATE PROCEDURE was   */
/* declared as varying length character. For C, varying   */
/* length are passed as NUL terminated strings unless     */
/* FOR BIT DATA is specified in the CREATE PROCEDURE     */
/*********************************************************/
strcpy(parm7,argv[7]);

/*********************************************************/
/* Copy date into local variable.                        */
/* Note that date and time variables are always passed in */
/* ISO format so that the lengths of the strings are      */
/* known. strcpy would work here just as well.            */
/*********************************************************/
strncpy(parm8,argv[8],10);

/* Copy time into local variable                         */
strncpy(parm9,argv[9],8);

/*********************************************************/
/* Copy timestamp into local variable.                   */
/* IBM SQL timestamp format is always passed so the length*/
/* of the string is known.                               */
/*********************************************************/
strncpy(parm10,argv[10],26);

/*********************************************************/
/* The indicator array is passed as an array of short     */
/* integers. There is one entry for each parameter passed */
/* on the CREATE PROCEDURE (10 for this example).        */
/* Below is one way to set each indicator into separate   */
/* variables.                                            */
/*********************************************************/
   ind = (short int *) argv[11];
   ind1 = *(ind++);
   ind2 = *(ind++);
   ind3 = *(ind++);
   ind4 = *(ind++);
   ind5 = *(ind++);
   ind6 = *(ind++);
   ind7 = *(ind++);
   ind8 = *(ind++);
   ind9 = *(ind++);
   ind10 = *(ind++);
  :
/* Perform any additional processing here                */
  :
return;
}
/******** END OF C PROCEDURE P1 *****************************/
```

圖 5. 範例程序 P1 (2/2)

```
/******** START OF PL/I PROCEDURE P2 *************************/
/******** PROGRAM TEST12/CALLPROC ****************************/
/************************************************************/


CALLPROC :PROC( PARM1,PARM2,PARM3,PARM4,PARM5,PARM6,PARM7,
                       PARM8,PARM9,PARM10,PARM11);
 DCL  SYSPRINT FILE STREAM OUTPUT EXTERNAL;
 OPEN FILE(SYSPRINT);
 DCL PARM1 CHAR(10);
 DCL PARM2 FIXED BIN(31);
 DCL PARM3 FIXED BIN(15);
 DCL PARM4 BIN FLOAT(22);
 DCL PARM5 BIN FLOAT(53);
 DCL PARM6 FIXED DEC(10,5);
 DCL PARM7 CHARACTER(10) VARYING;
 DCL PARM8 CHAR(10);      /* FOR DATE */
 DCL PARM9 CHAR(8);       /* FOR TIME */
 DCL PARM10 CHAR(26);     /* FOR TIMESTAMP */
 DCL PARM11(10) FIXED BIN(15);   /* Indicators */

 /* PERFORM LOGIC - Variables can be set to other values for  */
 /* return to the calling program.                            */

 :

 END CALLPROC;
```

圖 6. 範例程序 P2

下一個範例顯示從 ILE C 程式呼叫的 REXX 程序。

假設程序定義如下：

```
EXEC SQL CREATE PROCEDURE REXXPROC
              (IN PARM1 CHARACTER(20),
               IN PARM2 INTEGER,
               IN PARM3 DECIMAL(10,5),
               IN PARM4 DOUBLE PRECISION,
               IN PARM5 VARCHAR(10),
               IN PARM6 GRAPHIC(4),
               IN PARM7 VARGRAPHIC(10),
               IN PARM8 DATE,
               IN PARM9 TIME,
               IN PARM10 TIMESTAMP)
         EXTERNAL NAME 'TEST.CALLSRC(CALLREXX)'
         LANGUAGE REXX GENERAL WITH NULLS
```

範例 2. 從 C 應用程式呼叫的範例 REXX 程序

註: 如需程式碼範例的相關資訊，請參閱第 x 頁的『程式碼不保事項聲明』的資訊。

```
/************************************************************/
/*********** START OF SQL C Application **********************/

        #include <decimal.h>
        #include <stdio.h>
        #include <string.h>
        #include <wcstr.h>
        /*-----------------------------------------------------------*/
        exec sql include sqlca;
        exec sql include sqlda;
        /* ***********************************************************/
        /* Declare host variable for the CALL statement              */
        /* ***********************************************************/
        char parm1[20];
        signed long int parm2;
        decimal(10,5) parm3;
        double parm4;
        struct {  short dlen;
                  char dat[10];
                } parm5;
        wchar_t parm6[4] = { 0xC1C1, 0xC2C2, 0xC3C3, 0x0000 };
        struct {  short  dlen;
                  wchar_t dat[10];
                } parm7 = {0x0009, 0xE2E2,0xE3E3,0xE4E4, 0xE5E5, 0xE6E6,
                           0xE7E7, 0xE8E8, 0xE9E9, 0xC1C1, 0x0000 };

        char parm8[10];
        char parm9[8];
        char parm10[26];
        main()
        {
```

圖 7. 從 C 應用程式呼叫的範例 REXX 程序 (1/4)

```
        /* ************************************************************/
        /* Call the procedure - on return from the CALL statement the  */
        /* SQLCODE should be 0. If the SQLCODE is non-zero,            */
        /* the procedure detected an error.                            */
        /* ************************************************************/
        strcpy(parm1,"TestingREXX");
        parm2 = 12345;
        parm3 = 5.5;
        parm4 = 3e3;
        parm5.dlen = 5;
        strcpy(parm5.dat,"parm6");
        strcpy(parm8,"1994-01-01");
        strcpy(parm9,"13.01.00");
        strcpy(parm10,"1994-01-01-13.01.00.000000");

         EXEC SQL CALL REXXPROC (:parm1, :parm2,
                                 :parm3,:parm4,
                                 :parm5, :parm6,
                                 :parm7,
                                 :parm8, :parm9,
                                 :parm10);

         if (strncpy(SQLSTATE,"00000",5))
           {
           /* handle error or warning returned on CALL   */
           :
           }
                :
       }

/****** END OF SQL C APPLICATION   ***********************************/
/*********************************************************************/
```

圖 7. 從 C 應用程式呼叫的範例 REXX 程序 (2/4)

```
/*********************************************************************/
/****** START OF REXX MEMBER TEST/CALLSRC CALLREXX *******************/
/*********************************************************************/
       /* REXX source member TEST/CALLSRC CALLREXX              */
       /* Note the extra parameter being passed for the indicator*/
       /* array.                                                */
       /*                                                       */
       /* ACCEPT THE FOLLOWING INPUT VARIABLES SET TO THE       */
       /* SPECIFIED VALUES :                                    */
       /* AR1        CHAR(20)          =  'TestingREXX'         */
       /* AR2        INTEGER           = 12345                  */
       /* AR3        DECIMAL(10,5)     = 5.5                    */
       /* AR4        DOUBLE PRECISION  = 3e3                    */
       /* AR5        VARCHAR(10)       = 'parm6'                */
       /* AR6        GRAPHIC           = G'C1C1C2C2C3C3'        */
       /* AR7        VARGRAPHIC        =                        */
       /*             G'E2E2E3E3E4E4E5E5E6E6E7E7E8E8E9E9EAEA'   */
       /* AR8        DATE              = '1994-01-01'           */
       /* AR9        TIME              = '13.01.00'             */
       /* AR10       TIMESTAMP         =                        */
       /*                      '1994-01-01-13.01.00.000000'     */
       /* AR11       INDICATOR ARRAY   = +0+0+0+0+0+0+0+0+0+0    */

       /*********************************************************/
       /* Parse the arguments into individual parameters        */
       /*********************************************************/
       parse arg ar1 ar2 ar3 ar4 ar5 ar6 ar7 ar8 ar9 ar10 ar11

       /*********************************************************/
       /* Verify that the values are as expected                */
       /*********************************************************/
       if ar1<>"'TestingREXX'" then signal ar1tag
       if ar2<>12345 then signal ar2tag
       if ar3<>5.5 then signal ar3tag
       if ar4<>3e3 then signal ar4tag
       if ar5<>"'parm6'" then signal ar5tag
       if ar6 <>"G'AABBCC'" then signal ar6tag
       if ar7 <>"G'SSTTUUVVWWXXYYZZAA'" then ,
        signal ar7tag
       if ar8 <> "'1994-01-01'" then signal ar8tag
       if ar9 <> "'13.01.00'" then signal ar9tag
       if ar10 <> "'1994-01-01-13.01.00.000000'" then signal ar10tag
       if ar11 <> "+0+0+0+0+0+0+0+0+0+0" then signal ar11tag
```

圖 7. 從 C 應用程式呼叫的範例 REXX 程序 (3/4)

```
       /************************************************************/
       /* Perform other processing as necessary ..                 */
       /************************************************************/
          :
       /************************************************************/
       /* Indicate the call was successful by exiting with a       */
       /*                                                          */
       /************************************************************/
       exit(0)

       ar1tag:
       say "ar1 did not match" ar1
       exit(1)
       ar2tag:
       say "ar2 did not match" ar2
       exit(1)
       :
       :

/************ END OF REXX MEMBER   *********************************/
```

圖 7. 從 C 應用程式呼叫的範例 REXX 程序 (4/4)

第 12 章 使用物件關聯功能

本章將討論 DB2 的物件導向功能。
- 爲何使用 DB2 物件延伸套件？
- DB2 對物件的支援方式
- 使用大型物件 (LOB)
- 使用者定義的函數 (UDF)
- 使用者定義的特殊類型 (UDT)
- UDT、UDF 及 LOB 之間的協同作用

現代程式設計語言技術近來所發生的最重要事項之一就是*物件導向*。所謂物件導向是指應用程式領域中的實體可塑造成獨立物件，再藉由分類使其彼此相關。物件導向可讓您攫取應用程式領域中各類物件的異同之處，或將其集結成相關類型。相同類型的物件其行爲模式亦相同，因爲它們共用了相同類型的函數組合。而這些函數將會反映您的物件在應用程式領域中的行爲。

為何使用 DB2 物件延伸套件？

透過 DB2 的物件延伸套件，您可將物件導向 (OO) 的概念與方法導入您的關聯式資料庫中。要達成此目的，您可將其擴展，以納入更多的類型與函數。有了這些延伸套件，您便可將物件導向式資料類型案例存入表格欄位中，再藉由 SQL 陳述式中的函數加以操作。此外，您可將儲存物件的語意行爲作爲重要資源，供所有應用程式透過資料庫來加以共用。

要將物件導向納入到您的關聯式資料庫系統中，您可以先定義本身的新類型與函數。而這些新類型與函數應能反映應用程式領域中的物件語意。但有些您想塑造的資料物件可能相當龐大且複雜 (例如，文字、語音、影像及財務資料)。因此，您還需要可供儲存及處理大型物件的機制。使用者定義的特殊類型 (UDT)、使用者定義的函數 (UDF) 及大型物件 (LOB) 正是 DB2 所提供的此種機制。透過 DB2，您即可定義本身的新類型與函數，用來儲存及處理資料庫中的物件。

如後續各節將提及的，這些物件導向特性之間存有重大的協同作用。您可以將應用程式領域中的複雜物件塑造成 UDT。而接著 UDT 可在內部作爲 LOB。之後，UDT 的行爲可用 UDF 來建置。本節將示範如何透過 LOB 以及相關步驟來定義 UDT 與 UDF。您還將學到如何在應用程式中善用 UDT、UDF 及 LOB 來代表物件，進而使其搭配運作。

註: DB2 物件導向機制 (UDT、UDF 及 LOB) 的用途並不限於對物件導向應用程式的支援。就如同 C++ 程式設計語言會採用不同類型的非物件導向應用程式，DB2 的物件導向機制亦支援各類的非物件導向應用程式。 UDT、UDF 及 LOB 係屬於一般性的機制，可用來打造任何資料庫應用程式。因此，這些 DB2 物件延伸套件除可提升對傳統應用程式的支援外，更能全面支援非傳統 (亦即，物件導向) 的應用程式外。

DB2 對物件的支援方式

DB2 的物件延伸套件可讓您在享有關聯技術的長處外，同時感受物件技術的優點。在關聯式系統中，是以資料類型來描述表格欄位中所儲存的資料，繼而用這些表格來儲存此類資料類型的案例 (或物件)。至於這些案例的相關作業，則是透過在表示式中呼叫運算子或函數來加以執行。

DB2 在支援物件延伸套件上的作法，完全相容於關聯式的模式。 UDT 就是您所定義的資料類型。UDT (好比內建類型) 可用於描述儲存在表格欄位中的資料。 UDF 則是您定義的函數。 UDF (好比內建函數或運算子) 可用於支援 UDT 案例的操作。因此，UDT 案例將被儲存在表格欄位中，然後再透過 SQL 查詢以 UDF 加以操作。 UDT 在內部可用不同的方式來呈現。 LOB 不過是其中的範例之一。

使用大型物件 (LOB)

VARCHAR 與 VARGRAPHIC 資料類型對儲存量有 32KB 的限制。雖然這對中型大小的文字資料已經足夠，但應用程式通常需要儲存大型的文字文件。此外還可能必須儲存各類的資料類型，例如音效、視訊、圖表、混合圖文以及影像。 DB2 提供三種資料類型，將資料物件存成字串，其大小可達 2 GB。這三種資料類型分別是：「二進位大型物件 (BLOB)」、「單位元組字元大型物件 (CLOB)」及「雙位元組字元大型物件 (DBCLOB)」。

儲存大型物件 (LOB) 時，您還需要一些相關方法來參考、使用及修改資料庫中的各個 LOB。由於每個 DB2 表格都可能擁有大量的 LOB 相關資料。雖然只含有一或多個 LOB 值的單一列不會超過 3.5 GB，但一份表格則可能含有幾近 256 GB 的 LOB 資料。而且特定列的 LOB 欄位內容隨時都含有*大型物件值*。

此時您可以用主變數來參考及操作 LOB，就如同任何其他資料類型的情形一樣。不過，主變數常會用到從屬站的記憶體緩衝區，而其容量可能不足以存放 LOB 值。因此若要處理這些大型值，便需藉用其他方式。*定位器*就是相當方便的工具，它可以識別及處理資料庫伺服器的大型物件，並擷取 LOB 值的細瑣部份。*檔案參考變數*也是十分有用，它可用來在主從兩端之間實際移動大型物件值 (或其中大部份)。

以下各小節將就以上所介紹的主體進行深入探討：

- 『瞭解大型物件資料類型 (BLOB、CLOB、DBCLOB)』
- 第 177 頁的『瞭解大型物件定位器』
- 第 177 頁的『範例：使用定位器來處理 CLOB 值』
- 第 181 頁的『指示器變數與 LOB 定位器』
- 第 181 頁的『LOB 檔案參考變數』
- 第 182 頁的『範例：提取文件至檔案』
- 第 184 頁的『範例：將資料插入於 CLOB 直欄中』
- 第 184 頁的『顯示 LOB 直欄的佈置方式』
- 第 184 頁的『LOB 直欄的異動日誌登錄佈置方式』

瞭解大型物件資料類型 (BLOB、CLOB、DBCLOB)

大型物件資料類型所儲存的資料大小可從 0 位元組到 2GB。

以下是三種大型物件資料類型的定義：

- 字元大型物件 (CLOB) — 一種由單位元組字元與相關字碼頁所組成的字串。此種資料類型最適合用來保存文字導向的資訊，其間的資訊量會隨著成長而超越一般 VARCHAR 資料類型的限制 (上限爲 32KB)。 CLOB 可支援字碼頁轉換，並且相容於其他字元類型。
- 雙位元組字元大型物件 (DBCLOB) — 一種由雙位元組字元與相關字碼頁組成的字串。此資料類型適合用來保存會用到雙位元組字元集的文字導向資訊。同樣的，它也支援字碼頁轉換，而且相容於其他雙位元組字元類型。
- 二進位大型物件 (BLOB) — 一種由位元組與相關字碼頁所組成的二進位字串。此種資料類型可能是最有用的類型，因爲它可以儲存二進位資料。所以它是「使用者定義的特殊類型 (UDT)」的最佳來源類型。以 BLOB 作爲來源類型的 UDT 可供儲存影像、聲音、圖形及其他類型的商業或應用程式資料。有關 UDT 的詳情，請參閱第 199 頁的『使用者定義的特殊類型 (UDT)』。

瞭解大型物件定位器

就概念而言，LOB 定位器代表了一個存在頗久的簡單觀念；以較小且較易管理的值來參照較大的值。明確地說，LOB 定位器是一個儲存在主變數中長 4 個位元組的值，供程式用來參照資料庫系統中的 LOB 值 (或 LOB 表示式)。透過 LOB 定位器，程式就能將 LOB 值視同存放在一般主變數中來加以處理。當您使用 LOB 定位器時，將不須把 LOB 值從伺服器傳輸到應用程式 (或者再次傳回)。

LOB 定位器係關聯至 LOB 值或 LOB 表示式，而非某資料列或資料庫中的實體儲存體位置。因此，選取 LOB 值到定位器後，您將無法對原始列或表格執行作業，即使其對定位器所參照的值有任何作用。關聯至定位器的值將保持有效，直到在工作單元結束或定位器被明確釋放 (以先發生者爲主)。 FREE LOCATOR 陳述式即可用來釋放定位器與其相關的值。同理，確定或回轉作業也可釋放所有與異動相關的 LOB 定位器。

LOB 定位器還可在 DB2 與 UDF 之間負責傳遞。在 UDF 中，會用及 LOB 資料的函數亦可透過 LOB 定位器來處理 LOB 值。

要選取 LOB 值時，您有三種選項可用。

- 選取整個 LOB 值到主變數中。整個 LOB 值將被複製到主變數。
- 選取 LOB 值到 LOB 定位器中。LOB 值仍留在伺服器上；並不會複製到主變數。
- 選取整個 LOB 值到檔案參考變數中。LOB 值會移至「整合檔案系統 (IFS)」檔案中。

在程式中使用 LOB 值可協助程式設計師決定最適用的方法。如果 LOB 值相當龐大而且只是作爲一或多個後續 SQL 陳述式的輸入值，可將此值保存在定位器中。

如果程式需要整個 LOB 值而不管其大小，則除了轉送 LOB 外即無其他選擇。但即使是在此種情況下，您仍然可有其他選擇。您可以將整個值選取到一般主變數或檔案參照主變數中。也可以選取 LOB 值到定位器中，然後從定位器逐段加以讀取至主變數，如下述範例 『範例：使用定位器來處理 CLOB 值』 中的建議方式。

範例：使用定位器來處理 CLOB 值

此例中，應用程式會擷取某一 LOB 值的定位器；然後以此定位器來提取 LOB 值中的資料。透過此方法，程式可以僅配置足量的儲存體供一段 LOB 資料使用 (大小由程式決定)。甚者，該程式只須發出一個用到游標的提取呼叫。

註: 如需相關的程式碼範例，請參閱 第 x 頁的『程式碼不保事項聲明』。

範例 LOBLOC 程式的運作方式

1. **宣告主變數**。 BEGIN DECLARE SECTION 與 END DECLARE SECTION 陳述式會界定主變數的宣告內容。在 SQL 陳述式中被參照時，主變數是以冒號 (:) 爲字首。 CLOB LOCATOR 被宣告爲主變數。
2. **提取 LOB 值到定位器主變數中**。 以 CURSOR 與 FETCH 常式來取得 LOB 欄位在資料庫中的位置，並將其傳給定位器主變數。
3. **釋放 LOB LOCATORS**。本例中所用的 LOB LOCATORS 將被釋放，使定位器從原先所相關的值釋放出來。

CHECKERR 巨集/函數是一支在該程式外部的錯誤檢查公用程式。此公用程式的位置將取決於所用的程式設計語言：

C check_error 會被重新定義爲 CHECKERR，且位於 util.c 檔案中。

C 範例：LOBLOC.SQC

```
 #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "util.h"

EXEC SQL INCLUDE SQLCA;

#define  CHECKERR(CE_STR)   if (check_error (CE_STR, &sqlca) != 0) return 1;

int main(int argc, char *argv[]) {

#ifdef DB2MAC
   char * bufptr;
#endif

   EXEC SQL BEGIN DECLARE SECTION; 1
      char number[7];
      long deptInfoBeginLoc;
      long deptInfoEndLoc;
      SQL TYPE IS CLOB_LOCATOR resume;
      SQL TYPE IS CLOB_LOCATOR deptBuffer;
      short lobind;
      char buffer[1000]="";
      char userid[9];
      char passwd[19];
   EXEC SQL END DECLARE SECTION;

   printf( "Sample C program: LOBLOC\n" );

   if (argc == 1) {
      EXEC SQL CONNECT TO sample;
  CHECKERR ("CONNECT TO SAMPLE");
   }
   else if (argc == 3) {
      strcpy (userid, argv[1]);
      strcpy (passwd, argv[2]);
      EXEC SQL CONNECT TO sample USER :userid USING :passwd;
      CHECKERR ("CONNECT TO SAMPLE");
   }
   else {
      printf ("\nUSAGE: lobloc [userid passwd]\n\n");
      return 1;
   } /* endif */

   /* Employee A10030 is not included in the following select, because
      the lobeval program manipulates the record for A10030 so that it is
      not compatible with lobloc */

   EXEC SQL DECLARE c1 CURSOR FOR
            SELECT empno, resume FROM emp_resume WHERE resume_format='ascii'
            AND empno <> 'A00130';

   EXEC SQL OPEN c1;
   CHECKERR ("OPEN CURSOR");

   do {
      EXEC SQL FETCH c1 INTO :number, :resume :lobind;  2
      if (SQLCODE != 0) break;
   if (lobind < 0) {
         printf ("NULL LOB indicated\n");
   } else {
```

```
        /* EVALUATE the LOB LOCATOR */
        /* Locate the beginning of "Department Information" section */
        EXEC SQL VALUES (POSSTR(:resume, 'Department Information'))
           INTO :deptInfoBeginLoc;
        CHECKERR ("VALUES1");

        /* Locate the beginning of "Education" section (end of "Dept.Info" */
        EXEC SQL VALUES (POSSTR(:resume, 'Education'))
           INTO :deptInfoEndLoc;
        CHECKERR ("VALUES2");

        /* Obtain ONLY the "Department Information" section by using SUBSTR */
        EXEC SQL VALUES(SUBSTR(:resume, :deptInfoBeginLoc,
           :deptInfoEndLoc - :deptInfoBeginLoc)) INTO :deptBuffer;
        CHECKERR ("VALUES3");

        /* Append the "Department Information" section to the :buffer var. */
        EXEC SQL VALUES(:buffer || :deptBuffer) INTO :buffer;
        CHECKERR ("VALUES4");
   } /* endif */
   } while ( 1 );

#ifdef DB2MAC
   /* Need to convert the newline character for the Mac */
   bufptr = &(buffer[0]);
   while ( *bufptr != '\0' ) {
      if ( *bufptr == 0x0A ) *bufptr = 0x0D;
  bufptr++;
   }
#endif

   printf ("%s\n",buffer);

   EXEC SQL FREE LOCATOR :resume, :deptBuffer; 3
   CHECKERR ("FREE LOCATOR");

   EXEC SQL CLOSE c1;
   CHECKERR ("CLOSE CURSOR");

   EXEC SQL CONNECT RESET;
   CHECKERR ("CONNECT RESET");
   return 0;
}
/* end of program : LOBLOC.SQC */
```

COBOL 範例：LOBLOC.SQB

```
 Identification Division.
 Program-ID. "lobloc".

 Data Division.
 Working-Storage Section.
     copy "sqlenv.cbl".
     copy "sql.cbl".
     copy "sqlca.cbl".

     EXEC SQL BEGIN DECLARE SECTION END-EXEC. 1
 01 userid            pic x(8).
 01 passwd.
   49 passwd-length   pic s9(4) comp-5 value 0.
   49 passwd-name     pic x(18).
 01 empnum            pic x(6).
 01 di-begin-loc      pic s9(9) comp-5.
 01 di-end-loc        pic s9(9) comp-5.
 01 resume            USAGE IS SQL TYPE IS CLOB-LOCATOR.
 01 di-buffer         USAGE IS SQL TYPE IS CLOB-LOCATOR.
 01 lobind            pic s9(4) comp-5.
 01 buffer            USAGE IS SQL TYPE IS CLOB(1K).
     EXEC SQL END DECLARE SECTION END-EXEC.

 77 errloc          pic x(80).

 Procedure Division.
 Main Section.
     display "Sample COBOL program: LOBLOC".

* Get database connection information.
     display "Enter your user id (default none): "
          with no advancing.
     accept userid.

     if userid = spaces
       EXEC SQL CONNECT TO sample END-EXEC
     else
       display "Enter your password : " with no advancing
       accept passwd-name.
```

```
* Passwords in a CONNECT statement must be entered in a VARCHAR
* format with the length of the input string.
     inspect passwd-name tallying passwd-length for characters
        before initial " ".

     EXEC SQL CONNECT TO sample USER :userid USING :passwd
         END-EXEC.
     move "CONNECT TO" to errloc.
     call "checkerr" using SQLCA errloc.

* Employee A10030 is not included in the following select, because
* the lobeval program manipulates the record for A10030 so that it is
* not compatible with lobloc

     EXEC SQL DECLARE c1 CURSOR FOR
              SELECT empno, resume FROM emp_resume
              WHERE resume_format = 'ascii'
              AND empno <> 'A00130' END-EXEC.

     EXEC SQL OPEN c1 END-EXEC.
     move "OPEN CURSOR" to errloc.
     call "checkerr" using SQLCA errloc.

     Move 0 to buffer-length.

     perform Fetch-Loop thru End-Fetch-Loop
        until SQLCODE not equal 0.

* display contents of the buffer.
     display buffer-data(1:buffer-length).

     EXEC SQL FREE LOCATOR :resume, :di-buffer END-EXEC.  3
     move "FREE LOCATOR" to errloc.
     call "checkerr" using SQLCA errloc.

     EXEC SQL CLOSE c1 END-EXEC.
     move "CLOSE CURSOR" to errloc.
     call "checkerr" using SQLCA errloc.

     EXEC SQL CONNECT RESET END-EXEC.
     move "CONNECT RESET" to errloc.
     call "checkerr" using SQLCA errloc.
 End-Main.
        go to End-Prog.

 Fetch-Loop Section.
     EXEC SQL FETCH c1 INTO :empnum, :resume :lobind  2
        END-EXEC.

     if SQLCODE not equal 0
        go to End-Fetch-Loop.

* check to see if the host variable indicator returns NULL.
     if lobind less than 0 go to NULL-lob-indicated.

* Value exists.  Evaluate the LOB locator.
* Locate the beginning of "Department Information" section.
     EXEC SQL VALUES (POSSTR(:resume, 'Department Information'))
              INTO :di-begin-loc END-EXEC.
     move "VALUES1" to errloc.
     call "checkerr" using SQLCA errloc.

* Locate the beginning of "Education" section (end of Dept.Info)
     EXEC SQL VALUES (POSSTR(:resume, 'Education'))
               INTO :di-end-loc END-EXEC.
     move "VALUES2" to errloc.
     call "checkerr" using SQLCA errloc.

     subtract di-begin-loc from di-end-loc.

* Obtain ONLY the "Department Information" section by using SUBSTR
     EXEC SQL VALUES (SUBSTR(:resume, :di-begin-loc,
              :di-end-loc))
              INTO :di-buffer END-EXEC.
     move "VALUES3" to errloc.
     call "checkerr" using SQLCA errloc.

* Append the "Department Information" section to the :buffer var
     EXEC SQL VALUES (:buffer || :di-buffer) INTO :buffer
              END-EXEC.
     move "VALUES4" to errloc.
     call "checkerr" using SQLCA errloc.

     go to End-Fetch-Loop.

 NULL-lob-indicated.
     display "NULL LOB indicated".
```

```
End-Fetch-Loop. exit.

End-Prog.
          stop run.
```

指示器變數與 LOB 定位器

對應用程式中的一般主變數而言，選取 NULL 值到主變數中時，指示器變數會被指派負值以代表該值爲 NULL。但就 LOB 定位器而言，指示器變數的意義稍有不同。由於定位器主變數本身絕不得爲 NULL，因此負的指示器變數值係指 LOB 定位器所代表的 LOB 值爲 NULL。 NULL 資訊會被保存在使用該指示器變數值的從屬站本端 — 伺服器並不會追蹤具備有效指示器的 NULL 值。

LOB 檔案參考變數

檔案參考變數與主變數極爲類似，不過它是用來對 IFS 檔案轉送資料 (而非對記憶體緩衝區)。檔案參考變數代表 (而非含有) 著檔案，就如同 LOB 定位器代表 (而非含有) 著 LOB 值。資料庫的查詢、更新及插入，都可能會用到檔案參考變數來儲存或擷取單一 LOB 值。

對極龐大的物件而言，檔案恰如其天然的儲存區。大部份的 LOB 很可能開始時都是儲存在從屬站之檔案中的資料，之後才被移入伺服器上的資料庫。而使用檔案參考變數將可協助您來移動 LOB 資料。程式則可透過檔案參考變數，將 LOB 資料從 IFS 檔案直接轉送至資料庫引擎。若要進行 LOB 資料的移動，應用程式將不需撰寫公用程式，以便透過主變數來讀寫檔案。

註: 被檔案參考變數所參考的檔案須可從執行程式的系統上 (但不須常駐於其上) 加以存取。對儲存程序而言，則須可從伺服器存取。

檔案參考變數的資料類型有 BLOB、CLOB 或 DBCLOB 三種。它可作爲資料來源 (輸入) 或資料目標 (輸出)。檔案參考變數可使用相對檔名或完整的檔案路徑名稱 (建議採用後者)。其檔名長度指定於應用程式中。在進行輸入時不會使用檔案參考變數的資料長度部份。但在進行輸出時，應用程式要求器程式碼會將資料長度設爲寫入該檔案之新資料的長度。

使用檔案參考變數時，在輸入與輸出方面都有不同的選項可供使用。您必須藉由設定檔案參考變數結構中的 `file_options` 欄位，爲檔案選擇某一動作。用以對此欄位同時指派輸入與輸出值的各種選項如下所示。

使用輸入檔案參考變數時的相關值 (以 C 爲例) 與選項如下：

- **SQL_FILE_READ** (一般檔案) — 此選項的值爲 2。這是一個可開啓、讀取及關閉的檔案。開啓此檔案時，DB2 會先決定其中的資料長度 (以位元組爲單位)。然後 DB2 會透過檔案參考變數結構的 `data_length` 欄位傳回長度。(COBOL 的值爲 SQL-FILE-READ。)

使用輸出檔案參考變數時的值與選項如下：

- **SQL_FILE_CREATE** (新檔案) — 此選項的值爲 8。此選項會建立新檔案。若檔案已存在，即會傳回錯誤訊息。 (COBOL 的值爲 SQL-FILE-CREATE。)
- **SQL_FILE_OVERWRITE** (改寫檔案) — 此選項的值爲 16。若無任何檔案存在時，這個選項會建立新檔案。若該檔案已存在，則用新資料改寫檔案中的資料。(COBOL 的值爲 SQL-FILE-OVERWRITE。)

- **SQL_FILE_APPEND** (附加檔案) — 此選項的值爲 32。這個選項會將輸出附加至檔案 (若檔案已存在)。否則，即建立新檔案。(COBOL 的值爲 SQL-FILE-APPEND。)

註: 若在 OPEN 陳述式中使用了 LOB 檔案參考變數，在游標關閉前請勿刪除關聯至該變數的檔案。

有關整合檔案系統的詳情，請參閱整合檔案系統。

範例：提取文件至檔案

此程式範例將示範如何把 CLOB 元素從某一表格中提取到外部檔案。

範例 LOBFILE 程式的運作方式

1. **宣告主變數**。BEGIN DECLARE SECTION 與 END DECLARE SECTION 陳述式會界定主變數的宣告內容。在 SQL 陳述式中被參照時，主變數是以冒號 (:) 爲字首。宣告了 CLOB FILE REFERENCE 主變數。
2. **設定 CLOB FILE REFERENCE 主變數**。設定 FILE REFERENCE 的屬性。根據預設，未含完整宣告路徑的檔名被置於使用者的現行目錄中。如果路徑名稱並非以正斜線 (/) 爲開頭，即表示其未經限定。
3. **選取到 CLOB FILE REFERENCE 主變數中**。 來自 resume 欄位的資料被選取到主變數所參考的檔名中。

CHECKERR 巨集/函數是一支在該程式外部的錯誤檢查公用程式。此公用程式的位置將取決於所用的程式設計語言：

C 將重新定義 check_error 爲 CHECKERR，且位於 util.c 檔案中。

COBOL CHECKERR 是一名爲 checkerr.cbl 的外部程式

註: 如需相關的程式碼範例，請參閱 第 x 頁的『程式碼不保事項聲明』。

C 範例：LOBFILE.SQC

```
 #include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sql.h>
#include "util.h"

EXEC SQL INCLUDE SQLCA;

#define  CHECKERR(CE_STR)   if (check_error (CE_STR, &sqlca) != 0) return 1;

int main(int argc, char *argv[]) {

   EXEC SQL BEGIN DECLARE SECTION; 1
      SQL TYPE IS CLOB_FILE resume;
      short lobind;
      char userid[9];
      char passwd[19];
   EXEC SQL END DECLARE SECTION;

   printf( "Sample C program: LOBFILE\n" );

   if (argc == 1) {
      EXEC SQL CONNECT TO sample;
  CHECKERR ("CONNECT TO SAMPLE");
   }
   else if (argc == 3) {
      strcpy (userid, argv[1]);
      strcpy (passwd, argv[2]);
      EXEC SQL CONNECT TO sample USER :userid USING :passwd;
      CHECKERR ("CONNECT TO SAMPLE");
   }
   else {
      printf ("\nUSAGE: lobfile [userid passwd]\n\n");
      return 1;
```

```
   } /* endif */

   strcpy (resume.name, "RESUME.TXT");   2
   resume.name_length = strlen("RESUME.TXT");
   resume.file_options = SQL_FILE_OVERWRITE;

   EXEC SQL SELECT resume INTO :resume :lobind FROM emp_resume   3
      WHERE resume_format='ascii' AND empno='000130';

   if (lobind < 0) {
      printf ("NULL LOB indicated \n");
   } else {
      printf ("Resume for EMPNO 000130 is in file : RESUME.TXT\n");
   } /* endif */

   EXEC SQL CONNECT RESET;
   CHECKERR ("CONNECT RESET");
   return 0;
}
/* end of program : LOBFILE.SQC */
```

COBOL 範例：LOBFILE.SQB

```
 Identification Division.
 Program-ID. "lobfile".

 Data Division.
 Working-Storage Section.
     copy "sqlenv.cbl".
     copy "sql.cbl".
     copy "sqlca.cbl".

     EXEC SQL BEGIN DECLARE SECTION END-EXEC.  1
 01 userid            pic x(8).
 01 passwd.
   49 passwd-length   pic s9(4) comp-5 value 0.
   49 passwd-name     pic x(18).
 01 resume            USAGE IS SQL TYPE IS CLOB-FILE.
 01 lobind            pic s9(4) comp-5.
     EXEC SQL END DECLARE SECTION END-EXEC.

 77 errloc          pic x(80).

 Procedure Division.
 Main Section.
     display "Sample COBOL program: LOBFILE".

* Get database connection information.
     display "Enter your user id (default none): "
          with no advancing.
     accept userid.

     if userid = spaces
       EXEC SQL CONNECT TO sample END-EXEC
     else
       display "Enter your password : " with no advancing
       accept passwd-name.

* Passwords in a CONNECT statement must be entered in a VARCHAR
* format with the length of the input string.
     inspect passwd-name tallying passwd-length for characters
        before initial " ".

     EXEC SQL CONNECT TO sample USER :userid USING :passwd
         END-EXEC.
     move "CONNECT TO" to errloc.
     call "checkerr" using SQLCA errloc.

     move "RESUME.TXT" to resume-NAME.                          2
     move 10 to resume-NAME-LENGTH.
     move SQL-FILE-OVERWRITE to resume-FILE-OPTIONS.

     EXEC SQL SELECT resume INTO :resume :lobind  3
              FROM emp_resume
              WHERE resume_format = 'ascii'
              AND empno = '000130' END-EXEC.
     if lobind less than 0 go to NULL-LOB-indicated.

     display "Resume for EMPNO 000130 is in file : RESUME.TXT".
     go to End-Main.

 NULL-LOB-indicated.
     display "NULL LOB indicated".

 End-Main.
     EXEC SQL CONNECT RESET END-EXEC.
```

```
    move "CONNECT RESET" to errloc.
    call "checkerr" using SQLCA errloc.
End-Prog.
          stop run.
```

範例：將資料插入於 CLOB 直欄中

在下列 C 程式區段的路徑說明中：

- `userid` 代表您的使用者之一的目錄。
- `dirname` 代表『userid』的子目錄名稱。
- `filnam.1` 可成爲您要插入至表格中的文件之一的名稱。
- `clobtab` 爲具有 CLOB 資料類型之表格的名稱。

下述範例將示範如何將資料從 `:hv_text_file` 所參考的一般檔案插入至 CLOB 直欄：

```
strcpy(hv_text_file.name, "/home/userid/dirname/filnam.1");
hv_text_file.name_length = strlen("/home/userid/dirname/filnam.1");
hv_text_file.file_options = SQL_FILE_READ; /* this is a 'regular' file */

EXEC SQL INSERT INTO CLOBTAB
  VALUES(:hv_text_file);
```

顯示 LOB 直欄的佈置方式

以 CL 指令 (例如「顯示實體檔案成員 (DSPPFM)」) 來顯示保有 LOB 直欄之表格中的資料列時，此列中所儲存的 LOB 資料將不會顯現。反之，「資料庫」會針對 LOB 直欄顯示一特殊值。此特殊值的佈置方式如下：

- 13 到 28 個位元組的十六進位零值。
- 16 個位元組以 *POINTER 開頭並後接空白。

此值第一部份的位元組數會設成所需的數目 (上限爲 16 個位元組)，以對齊此值的第二部份。

例如，假設您的表格具有三個直欄：ColumnOne Char(10)、ColumnTwo CLOB(40K) 及 ColumnThree BLOB(10M)。如果您對此表格發出 DSPPFM，各資料列的外觀將如下所示。

- ColumnOne： 10 個位元組填入字元資料。
- ColumnTwo： 22 個位元組填入十六進位零值，且 16 個位元組填入 '*POINTER '。
- ColumnThree： 16 個位元組填入十六進位零值，且 16 個位元組填入 '*POINTER '。

在此方式下用於顯示 LOB 直欄的完整指令集爲：

- 顯示「實體檔案成員 (DSPPFM)」
- 當指定 *PRINT 值給 TOFILE 關鍵字時「複製檔案 (CPYF)」
- 顯示日誌 (DSPJRN)
- 擷取異動日誌登錄 (RTVJRNE)
- 當指定 *TYPE1、*TYPE2、*TYPE3 及 *TYPE4 等值給 ENTFMT 關鍵字時「接收異動日誌登錄 (RCVJRNE)」。

LOB 直欄的異動日誌登錄佈置方式

有兩種指令會傳回一緩衝區，爲使用者提供已登載之 LOB 資料的定址能力：

- 「接收異動日誌登錄 (RCVJRNE)」CL 指令，爲 ENTFMT 關鍵字指定 *TYPEPTR 值時
- 擷取異動日誌登錄 (QjoRetrieveJournalEntries) API

此類登錄中的 LOB 直欄其佈置方式如下：

- 0 到 15 個位元組的十六進位零值
- 1 個位元組的系統資訊被設爲 '00'x
- 4 個位元組用於保存指標所訂長度的 LOB 資料 (以此長度爲上限)
- 8 個位元組的十六進位零值
- 16 個位元組用於保存指標，此指標指至「異動日誌登錄」中所儲存的 LOB 資料。

此佈置的第一部份不得超過 16 個位元組，以對齊 LOB 資料的指標。此區域的位元組數主要取決於 LOB 直欄之後續直欄的長度。如需如何計算第一部份長度的範例，請參閱前述「顯示 LOB 直欄的佈置方式」一節。

有關 LOB 直欄的日誌登載方式之詳情，請參閱日誌登載主題。

使用者定義的函數 (UDF)

使用者定義的函數是一種可讓您撰寫本身 SQL 延伸套件的機制。DB2 所附的內建函數誠然是一組十分有用的函數，但它們可能無法滿足您的所有需求。因此，您可能會因爲下列理由而需要擴展 SQL：

- **自訂**。

  DB2 中並沒有適合您特定用途的函數。不論是簡單的轉換、瑣碎的計算或是複雜的統計分析，您或許可以透過 UDF 來進行處理。

- **彈性**。

  DB2 內建函數並不太允許您對應用程式作許多變化。

- **標準化**。

  您在工作場合中所用的程式多數會引用相同的基礎函數集，但各家的引用方式存有微小的差異。因此，您將無法確定所獲的結果是否一致。如果您用 UDF 將這些函數一次建立齊全，所有的程式就可在 SQL 中直接加以引用，進而提供一致的結果。

- **物件關聯支援**。

  就如第 199 頁的『使用者定義的特殊類型 (UDT)』 中所述， UDT 對擴展相關功能及提升 DB2 的安全性極爲有用。 UDF 就如同 UDT 的*憑藉*，提供其行爲並封裝相關類型。

此外，SQL UDF 還能支援「大型物件」與 DataLink 等類型的處理。雖然資料庫所提供的數種內建函數在處理這些資料類型上已相當好用，不過 SQL UDF 可讓使用者進一步操作及增強資料庫在此方面的功能 (以達成特殊需求)。

相關資訊，請參閱以下各節：

- 第 186 頁的『爲何使用 UDF？』
- 第 188 頁的『UDF 概念』
- 第 189 頁的『實施 UDF』
- 第 190 頁的『登記 UDF』
- 第 190 頁的『儲存與復置方面的注意事項』
- 第 190 頁的『範例：登記 UDF』
- 第 194 頁的『使用 UDF』

為何使用 UDF？

撰寫 DB2 應用程式時，您可以選擇將所要的動作或作業實施爲：

- UDF
- 應用程式中的次常式或函數。

雖然將新作業實施爲次常式或函數似乎較爲容易，但您仍應考慮下列情況：

- **重覆使用**。

  若新作業可供其他使用者或程式加以使用，則 UDF 有助於加強其重覆使用的能力。此外，只要一表示式可供資料庫的任何使用者使用，函數即可在 SQL 中直接呼叫。資料庫會自動處理各類資料類型的函數引數。例如，透過 DECIMAL to DOUBLE，資料庫即允許您將函數套用到不同但相容的資料類型上。

  將您的新函數實施爲一般函數看起來似乎較爲容易。 (您即不須在 DB2 中定義此函數。) 但如果採行此作法，您將必須知會所有相關的應用程式開發者，並有效封裝該函數以方便他們使用。可是，此處理過程會忽略交談式使用者，例如通常是使用「指令行處理器 (CLP)」來存取資料庫者。再者，專爲程式內部使用所撰寫的函數會遺略掉並無相關程式的 (交談式) 使用者。其中包括諸如 STRSQL、STRQM 及 RUNSQLSTM 等指令，此外還有許多從屬站，如 ODBC、JDBC 等。 CLP 使用者將無法使用您的函數，除非它是資料庫中的 UDF。此情形也會發生在任何使用了未重新編譯之 SQL (例如 Visualizer) 的其他工具上。

- **效能**。

  在某些情況下，直接從資料庫引擎 (而非您的應用程式) 呼叫 UDF 會對效能提供莫大助益。若函數是被用來進一步處理資料限定，其效果將十分明顯。尤其當函數被用於處理列的選取時，這些情況都會發生。

  試以您在處理某些資料時的過程爲例。其間您用函數 SELECTION_CRITERIA() 的表示式來符合某些選取準則。假設您的應用程式可發出下列 SELECT 陳述式：

  ```
  SELECT A, B, C FROM T
  ```

  當它收到每一列時，會針對資料來執行 SELECTION_CRITERIA，以決定是否要加以進一步處理。此處，表格 T 的每一列皆須傳回給應用程式。但若 SELECTION_CRITERIA() 是建置爲 UDF，您的應用程式即可發出下列陳述式：

  ```
  SELECT C FROM T WHERE SELECTION_CRITERIA(A,B)=1
  ```

  此時，只有相關的列與直欄會透過應用程式與資料庫間的介面進行傳遞。

  處理大型物件 (LOB) 是 UDF 具有效能優點的另一個情況。假設您的函數可從某一 LOB 類型的值中提取某些資訊。您即可直接對資料庫伺服器執行此提取作業，而且只將提取出來的值傳回應用程式。這比傳回整個 LOB 值給應用程式然後再執行提取來得更有效率。將此函數包裝爲 UDF 時其效能可能會相當顯著，不過須視其特定用途而定。 (請注意，您還須藉由 LOB 定位器來提取部份的 LOB。有關類似情節的範例，請參閱第 181 頁的『指示器變數與 LOB 定位器』。)

- **物件導向**。

  您可以透過 UDF，來對使用者定義的特殊類型 (UDT，又稱*特殊類型*) 實施其行爲。有關 UDT 的詳細資訊，請參閱第 199 頁的『使用者定義的特殊類型 (UDT)』。有關 UDT 的其它明細及*強制轉型*的重要概念，請參閱 SQL Reference 中的 CREATE DISTINCT TYPE 陳述式。當您在建立特殊類型時，系統會自動爲您提供此特殊類型及其來源類型之間的強制轉型函數。此外還可能會提供比較運算子，例如 =、>、<

等，視來源類型而定。但此外的任何附加行爲，則須由您自行提供。建議最好是將特殊類型的行爲保存在資料庫中，以方便其所有使用者加以存取。因此，您可以利用 UDF 來作爲施行機制。

例如，假設您針對 1 MB 大小的 BLOB 定義了 BOAT 特殊類型。 type create 陳述式爲：

```
CREATE DISTINCT TYPE BOAT AS BLOB(1M)
```

此 BLOB 中含有各類的船舶規格與一些設計圖。設若您想要比較船隻的大小。可是光靠您在 BLOB 來源類型上所定義的特殊類型，系統並不會自動產生比較作業。此時您可以建置 BOAT_COMPARE 函數，根據您所選擇的度量，來比較兩艘船隻的大小。相關的度量可以是：排水量、總長度、噸數或其他根據 BOAT 物件所作的計算。您所建立的 BOAT_COMPARE 函數如下：

```
CREATE SQL FUNCTION BOAT_COMPARE (BOAT, BOAT) RETURNS INTEGER ...
```

如果您的函數傳回：

- 1 表第一個 BOAT 較大
- 2 表第二個較大
- 0 表兩者相等。

您可在 SQL 程式中以此函數來比較不同船隻。假設您建立了下列表格：

```
CREATE TABLE BOATS_INVENTORY (
  BOAT_ID        CHAR(5),
  BOAT_TYPE      VARCHAR(25),
  DESIGNER       VARCHAR(40),
  OWNER          VARCHAR(40),
  DESIGN_DATE    DATE,
  SPEC           BOAT,
  ...                 )

CREATE TABLE MY_BOATS (
  BOAT_ID        CHAR(5),
  BOAT_TYPE      VARCHAR(25),
  DESIGNER       VARCHAR(40),
  DESIGN_DATE    DATE,
  ACQUIRE_DATE   DATE,
  ACQUIRE_PRICE  CANADIAN_DOLLAR,
  CURR_APPRAISL  CANADIAN_DOLLAR,
  SPEC           BOAT,
  ...                 )
```

您可以執行下列 SQL SELECT 陳述式：

```
SELECT INV.BOAT_ID, INV.BOAT_TYPE, INV.DESIGNER,
  INV.OWNER, INV.DESIGN_DATE
  FROM BOATS_INVENTORY INV, MY_BOATS MY
  WHERE MY.BOAT_ID = '19GCC'
  AND BOAT_COMPARE(INV.SPEC, MY.SPEC) = 1
  AND INV.DESIGNER = MY.DESIGNER
```

此一簡單範例會從相同設計師的 BOATS_INVENTORY 中，傳回所有大於 MY_BOATS 中特定船隻的所有船隻。請注意，此範例只能傳回相關資料列給應用程式，因爲比較作業是在資料庫伺服器中進行。事實上，它會完全避免傳遞資料類型爲 BOAT 的任何值。而這將可提供大幅提升儲存容量與效能，因爲 BOAT 是以 1 MB 的 BLOB 資料類型爲基礎。

UDF 概念

下面將討論一些您在撰寫 UDF 程式碼之前須知道的重要概念：

函數名稱

- 函數的完整名稱。

  採用 *SQL 命名時，函數的完整名稱爲 <schema-name>.<function-name>。

  採用 *SYS 命名時，函數的完整名稱則爲 <schema-name>/<function-name>。 DML 陳述式中的函數名稱不得用 *SYS 命名來限定。

  您可以在任意位置用此完整名稱來參照某一函數。 例如：

  ```
  QGPL.SNOWBLOWER_SIZE    SMITH.FOO    QSYS2.SUBSTR    QSYS2.FLOOR
  ```

  不過，您可能會省略掉 <schema-name>，此時 DB2 須決定您所參照的函數爲何。例如：

  ```
  SNOWBLOWER_SIZE    FOO    SUBSTR    FLOOR
  ```

- 路徑

  *路徑*的概念對 DB2' 在解析因未指定 schema-name 而發生的*未限定*參照時相當重要。有關如何在 DDL 陳述式中以路徑參照函數的詳情，請參閱 SQL Reference 中其對應的 CREATE FUNCTION 陳述式之說明。路徑本身是一個經過排序的綱目名稱清單。其中列有一組綱目，用以解析未經限定的 UDF 與 UDT 參照。倘若某一函數參照符合路徑中多個綱目下的函數，該路徑的綱目次序將被用來解析此相符情況。路徑是在靜態 SQL 的前置編譯及連結指令中以 SQLPATH 選項來建立。並以動態 SQL 的 SET PATH 陳述式來設定。當啓動群組中的第一個 SQL 陳述式執行 (採 SQL 命名方式) 時，路徑將具有下列預設值：

  ```
  "QSYS","QSYS2","<ID>"
  ```

  此值同時適用於靜態與動態 SQL，其中的 <ID> 代表現行陳述式授權 ID。

  當啓動群組中的第一個 SQL 陳述式執行 (採系統命名方式) 時，預設值爲 *LIBL。

- 超載函數名稱。

  函數名稱可以*超載*。所謂超載是指多個函數 (即使是在相同綱目下) 可使用相同名稱。不過，兩個函數不得具有相同的*簽章*。函數簽章可定義爲：限定函數名稱再依序接附已定妥資料類型的各個函數參數，而參數的接附順序是比照其定義時的順序。如需超載函數的範例，請參閱 第 191 頁的『範例：BLOB 字串搜尋』。 有關簽章與函數解析的詳情，請參閱 SQL Reference 書籍中的 Function 主題。

- 函數解析。

  *函數解析演算法*會負責就超載與函數路徑的考量，從各種函數參照中選出*最適當*者，不論其是否爲限定參照或未限定參照。所有函數都是透過函數選取演算法來處理，即使是內建函數也不例外。函數解析演算法並不會考慮函數的類型。所以表格函數也可能會被解析爲是*最適當*的函數，即使是用到參照而需要純量函數。

  路徑的概念 (SET PATH 陳述式) 與函數解析演算法會在 SQL Reference 中深入探討。 SQLPATH 前置編譯選項則會在指令附錄中詳加說明。

- 函數的類型。

  函數有以下數種類型：

  – *內建的*。 此類函數是隨附在資料庫中。SUBSTR() 即爲一例。

– *系統產生的*。 此類函數是由資料庫引擎於建立 DISTINCT TYPE 時暗中產生的。它們可在 DISTINCT TYPE 及其基本類型間提供強制轉型作業。
– *使用者定義的*。 此類函數是由使用者所建立並向資料庫登記的。

此外，每一函數可再進一步分類爲*純量*、*直欄*或*表格*函數。

*純量函數*每經呼叫後會傳回單一值回答。例如，內建函數 SUBSTR() 即爲純量函數，而許多內建函數同樣也是。系統產生的函數一律爲純量函數。純量 UDF 可以是外部的 (以諸如 C 等程式設計語言，或以 SQL— 一種 SQL 函數來撰寫程式碼)，或取自相關來源的 (利用現有的函數建置)。

*直欄函數*會接收一組類值 (一欄資料)，並透過此值集來傳回單一值回答。其在 DB2 中又稱爲*聚集函數*。部份的內建函數屬於直欄函數。像內建函數 AVG() 就是直欄函數。外部 UDF 不可定義爲直欄函數。不過，來源 UDF 若是取自內建直欄函數之一則會被定義爲直欄函數。後者對特殊類型非常有用。例如，若存在以基本類型 INTEGER 所定義的特殊類型 SHOESIZE，您就可用現有的內建直欄功能 AVG(INTEGER) 作爲來源，將 UDF AVG(SHOESIZE) 定義爲直欄函數。

*表格函數*會將表格傳回至參照它的 SQL 陳述式。它必須在 SELECT 的 FROM 子句中被參考到。表格函數可用來將 SQL 語言的處理能力應用到非屬 DB2 的資料上，或將此類資料轉換成 DB2 表格。例如，它可以將檔案轉成表格；從全球資訊網取得資料樣本再加以製表；或存取 Lotus Notes 資料庫並傳回有關郵件訊息的資訊，例如日期、寄件人及訊息本文。而此類資訊可和資料庫中的其他表格相結合。表格函數可被定義爲外部函數或 SQL 函數；但不得定義爲來源函數。

實施 UDF

UDF 的類型共有三種：來源、外部及 SQL。而各種類型的施行方式差異頗大。

- 來源 UDF。 就是資料庫中所登記的函數，且其本身會參照另一函數。實際上，它們會對映到來源函數。因此，在實施此類函數時，只須以 CREATE FUNCTION 陳述式將其登記到資料庫即可。
- 外部函數。 此類函數會參照以高階語言(例如 C、COBOL 或 RPG) 所寫成之程式與服務程式。一旦其向資料庫登記後，每當 DML 陳述式參照到此類函數時，資料庫將會呼叫程式或服務程式。因此，外部 UDF 的撰寫者除了要嫻熟高階語言及如何用其來編寫此類函數外，還須瞭解程式與資料庫之間的介面。有關如何撰寫外部函數的詳情，請參閱第 213 頁的第 13 章, 『撰寫使用者定義的函數 (UDF)』。
- SQL UDF。 SQL UDF 是完全以 SQL 語言撰寫而成的函數。其「程式碼」實際爲內含在 CREATE FUNCTION 陳述式自身中的陳述式。 SQL UDF 具有以下多種優點：
  – 完全以 SQL 撰寫，因此具備極佳可攜性。
  – 藉 SQL 宣告來定義資料庫與函數間的介面，無需憂慮參數的實際傳遞細節。
  – 可以傳遞大型物件、DataLink 及 UDT 作爲參數，繼而在函數中加以操作。有關 SQL 函數的詳細資訊，請參閱第 213 頁的第 13 章, 『撰寫使用者定義的函數 (UDF)』。

若要使用 UDF，請執行下列步驟：

1. 在 DB2 中登記 UDF。不論建立哪種類型的 UDF，都須用 CREATE FUNCTION 陳述式來向資料庫登記。若是要建立來源函數，此登記步驟會執行向資料庫定義此函數時所需的全部作業。對 SQL UDF 而言，CREATE FUNCTION 陳述式亦含有定義函數時所需的全部事項，但 CREATE 陳述式的語法更爲複雜 (含有實際的 SQL

可執行碼)。對外部 UDF 而言，CREATE FUNCTION 陳述式**只會**對資料庫登記函數；實際實施該函數的程式碼須單獨撰寫。相關資訊，請參閱 『登記 UDF』。

2. 對 UDF 除錯。 請參閱第 213 頁的第 13 章, 『撰寫使用者定義的函數 (UDF)』。

順利完成這些步驟後，您的 UDF 即可用於資料操作語言 (DML) 或資料定義語言 (DDL) 陳述式中，例如 CREATE VIEW。

登記 UDF

UDF 須先在資料庫中完成登記後，資料庫才能加以辨識及使用。您可以用 CREATE FUNCTION 陳述式來登記 UDF。

此陳述式可讓您指定程式的語言與名稱，以及諸如 DETERMINISTIC、ALLOW PARALLEL 及 RETURNS NULL ON NULL INPUT 等選項。這些選項有助於讓資料庫進一步界定函數的用途，以及如何讓資料庫呼叫最佳化。

在撰寫外部 UDF 的實際程式碼及完成測試後，您即應向 DB2 進行登記。雖然您可以先定義 UDF 然後才實際加以撰寫。不過，爲避免在執行 UDF 時發生任何問題，您最好是在進行登記前先完成撰寫與測試的工作。有關測試 UDF 的詳細資訊，請參閱第 213 頁的第 13 章, 『撰寫使用者定義的函數 (UDF)』。

如需有關登記 UDF 的範例，請參閱 『範例：登記 UDF』。

儲存與復置方面的注意事項

建立了與 ILE 外部程式或服務程式相關的外部函數後，系統會嘗試將此函數的屬性存入其相關程式或服務程式物件中。若 *PGM 或 *SRVPGM 物件經儲存後再復置到此系統或另一系統時，編目將會自動更新此類屬性。如果函數的屬性無法儲存，編目將不會自動更新，此時使用者須在新的系統上建立外部函數。外部函數的屬性是否能夠儲存，主要取決於下列限制：

- 外部程式檔案庫不得爲 QSYS 或 QSYS2。
- 發出 CREATE FUNCTION 陳述式時，外部程式須存在。
- 外部程式須爲 ILE *PGM 或 *SRVPGM 物件。
- 外部程式或服務程式至少須含一個 SQL 陳述式。

如果物件無法更新，函數仍會建立。

範例：登記 UDF

下列範例將說明可登記 UDF 時的各種典型情況。這些範例包括：

- 範例：乘冪
- 範例：字串搜尋
- 範例：透過 UDT 搜尋字串
- 範例：具有 UDT 參數的外部函數
- 範例：UDT 上的 AVG
- 範例：計數
- 範例：傳回文件 ID 的表格函數

註: 如需相關的程式碼範例，請參閱 第 x 頁的『程式碼不保事項聲明』。

範例：乘冪

假設您撰寫了外部 UDF 來執行浮點數值的乘冪，並想將其登記在 MATH 綱目中。

```
CREATE FUNCTION MATH.EXPON (DOUBLE, DOUBLE)
  RETURNS DOUBLE
  EXTERNAL NAME 'MYLIB/MYPGM(MYENTRY)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
  RETURNS NULL ON NULL INPUT
  ALLOW PARALLEL
```

本例中，系統將會使用 RETURNS NULL ON NULL INPUT 預設值。而此值相當適用，因爲您希望當引數之一爲 NULL 時其結果須爲 NULL。由於您並未硬性要求高速暫存器與最終呼叫，因此將使用 NO SCRATCHPAD 與 NO FINAL CALL 預設值。而基於並無理由要求 EXPON 不得爲平行，所以指定了ALLOW PARALLEL 值。

範例：字串搜尋

您的同事 Willie 寫了一個 UDF，以便在給定的 CLOB 值中尋找是否有給定的短字串，而其兩者皆爲引數。如果找到該字串，此 UDF 會傳回該字串在 CLOB 中的位置，否則即傳回零。

此外，Willie 還撰寫了用來傳回 FLOAT 結果的函數。假設您知道當其用於 SQL 中時，應一律傳回 INTEGER。 您即可建立下列函數：

```
CREATE FUNCTION FINDSTRING (CLOB(500K), VARCHAR(200))
  RETURNS INTEGER
  CAST FROM FLOAT
  SPECIFIC "willie_find_feb95"
  EXTERNAL NAME 'MYLIB/MYPGM(FINDSTR)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
  RETURNS NULL ON NULL INPUT
```

請注意，CAST FROM 子句是用來指定 UDF 主體眞的傳回 FLOAT 值，但您希望先將此強制轉型爲 INTEGER，然後才將該值傳回到使用 UDF 的陳述式。就如 SQL Reference 中所述，INTEGER 內建函數可爲您執行此強制轉型。另外，您還想爲函數提供您自己的特定名稱，然後再於 DDL (請參閱第 192 頁的『範例：透過 UDT 搜尋字串』) 中加以參考。由於此 UDF 並非用於處理 NULL 值，所以您應使用 RETURNS NULL ON NULL INPUT。同時由於沒有高速暫存器，您應使用 NO SCRATCHPAD 與 NO FINAL CALL 預設值。而基於並無理由要求 FINDSTRING 不得平行，所以用了 ALLOW PARALLEL 預設值。

範例：BLOB 字串搜尋

由於您希望此函數可用於 BLOB 與 CLOB，因此您定義了另一個 FINDSTRING 並將 BLOB 作爲第一個參數：

```
CREATE FUNCTION FINDSTRING (BLOB(500K), VARCHAR(200))
  RETURNS INTEGER
  CAST FROM FLOAT
  SPECIFIC "willie_fblob_feb95"
  EXTERNAL NAME 'MYLIB/MYPGM(FINDSTR)'
  LANGUAGE C
```

```
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
```

此例會說明 UDF 名稱的超載，並示範多個 UDF 如何共用相同主體。請注意，雖然 BLOB 不得指派給 CLOB，但其原始程式碼仍可使用。上述範例中並無程式設計上的問題，因爲在 DB2 與 UDF 之間，BLOB 與 CLOB 的程式設計介面是相同的：長度是隨資料而變。 DB2 並不會檢查使用特定函數主體的 UDF，是否與使用相同主體的任何其他 UDF 相一致。

範例：透過 UDT 搜尋字串

本例是前一範例的延續。假設您對第 191 頁的『範例：BLOB 字串搜尋』 的 FINDSTRING 函數甚感滿意，但現在您多了一個特殊類型 BOAT 且其來源類型爲 BLOB。您希望 FINDSTRING 也能處理資料類型爲 BOAT 的值，因此便建立了另一個 FINDSTRING 函數。此函數是以 FINDSTRING 作爲來源，後者會操作第 191 頁的『範例：BLOB 字串搜尋』 中的 BLOB 值。請注意本例中 FINDSTRING 的進一步超載：

```
CREATE FUNCTION FINDSTRING (BOAT, VARCHAR(200))
  RETURNS INT
  SPECIFIC "slick_fboat_mar95"
  SOURCE SPECIFIC "willie_fblob_feb95"
```

其中，此 FINDSTRING 函數具有來自第 191 頁的『範例：BLOB 字串搜尋』 中之 FINDSTRING 函數的不同簽章，因此在超載名稱上並無問題。您想提供本身的特定名稱，以便日後可在 DDL 中參照。由於您正在使用 SOURCE 子句，因此無法用 EXTERNAL NAME 子句或任何可指定函數屬性的相關關鍵字。這些屬性皆是取自來源函數。最後，在識別來源函數時，您是使用 第 191 頁的『範例：BLOB 字串搜尋』 中所明確提供的特定函數名稱。由於其爲未限定的參考，此來源函數所常駐的綱目須在函數路徑中，否則該參照將無法解析。

範例：具有 UDT 參數的外部函數

您寫了另一個 UDF 來取得 BOAT 並檢查其設計屬性，同時產生以加拿大幣計算的造船成本。即使在內部可能是以德國馬克、日圓或美元來計算勞力成本，此函數在產生造船成本時仍須使用所需幣別：此即表示函數須從 exchange_rate 檔案 (位於 DB2 之外) 取得現行的匯率資訊，而其回答須視該函數在此檔案中找到的內容而定。因此使得該函數變爲 NOT DETERMINISTIC。

```
CREATE FUNCTION BOAT_COST (BOAT)
  RETURNS INTEGER
  EXTERNAL NAME 'MYLIB/COSTS(BOATCOST)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  NOT DETERMINISTIC
  NO EXTERNAL ACTION
```

其間並未指定 CAST FROM 與 SPECIFIC，但指定了 NOT DETERMINISTIC。

範例：UDT 上的 AVG

本例會透過 CANADIAN_DOLLAR 特殊類型來實施 AVG 直欄函數。有關 CANADIAN_DOLLAR 的定義，請參閱第 200 頁的『範例：貨幣』。強勢類型會防止您對特殊類型使用內建的 AVG 函數。結果，CANADIAN_DOLLAR 的來源類型爲 DECIMAL，因此您藉由以 AVG(DECIMAL) 內建函數爲來源來實施 AVG。執行此動作

的能力主要取決於是否可從 DECIMAL 強制轉型爲 CANADIAN_DOLLAR (反之亦然)，但由於 DECIMAL 是 CANADIAN_DOLLAR 的來源類型，因此您知道這些強制轉型將是可行的。

```
CREATE FUNCTION AVG (CANADIAN_DOLLAR)
  RETURNS CANADIAN_DOLLAR
  SOURCE "QSYS2".AVG(DECIMAL(9,2))
```

請注意，您在 SOURCE 子句中限定了該函數的名稱，以免有某些其他 AVG 函數潛伏在您的 SQL 路徑中。

範例：計數

您的簡易計數函數最初會傳回 1，其後每當被呼叫時即會逐次增量 1。此函數並無 SQL 引數，且其根據定義爲 NOT DETERMINISTIC 函數，因爲其回答在每次呼叫時皆不同。它會使用 SCRATCHPAD 來儲存最後傳回的值。每當被呼叫時，該函數會將此值增量並加以傳回。

```
CREATE FUNCTION COUNTER ()
  RETURNS INT
  EXTERNAL NAME 'MYLIB/MYFUNCS(CTR)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  NOT DETERMINISTIC
  NOT FENCED
  SCRATCHPAD 4
  DISALLOW PARALLEL
```

您將注意到其中並未提供任何參數定義，只有空括弧。上述函數會指定 SCRATCHPAD 並使用預設規格 NO FINAL CALL。在此情況下，高速暫存器的大小僅設成 4 個位元組，已足供計數器使用。由於 COUNTER 函數需要用到單一高速暫存器以適當運作，所以加入 DISALLOW PARALLEL 以防止 DB2 與其平行運作。

範例：傳回文件 ID 的表格函數

您寫了一個表格函數，它所傳回的資料列中含有您的文字管理系統中各已知文件的單一文件 ID 直欄，而此 ID 直欄須符合給定主旨區域 (第一個參數) 並含有給定字串 (第二個參數)。此 UDF 會利用文字管理系統的函數來快速識別各文件：

```
CREATE FUNCTION DOCMATCH (VARCHAR(30), VARCHAR(255))
  RETURNS TABLE (DOC_ID CHAR(16))
  EXTERNAL NAME 'DOCFUNCS/UDFMATCH(udfmatch)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
  NOT FENCED
  SCRATCHPAD
  NO FINAL CALL
  DISALLOW PARALLEL
  CARDINALITY 20
```

在單一階段作業的環境中，它將一律傳回相同的表格，因此它被定義爲 DETERMINISTIC。請注意，RETURNS 子句會定義 DOCMATCH 的輸出，包括直欄名稱 DOC_ID。其間並不需要爲各個表格函數指定 FINAL CALL。此外，還加入了 DISALLOW PARALLEL 關鍵字，因爲表格函數無法平行運作。雖然 DOCMATCH 的輸出大小變化極鉅，CARDINALITY 20 爲其代表值，此處指定它來協助 DB2 最佳化工具作出良好決策。

一般而言，此表格函數將被用於和內含文件文字的表格相結合，如下所示：

```
SELECT T.AUTHOR, T.DOCTEXT
  FROM DOCS AS T, TABLE(DOCMATCH('MATHEMATICS', 'ZORN''S LEMMA')) AS F
  WHERE T.DOCID = F.DOC_ID
```

請注意用於 FROM 子句中指定表格函數的特殊語法 (TABLE 關鍵字)。在此呼叫中，DOCMATCH() 表格函數會傳回一資料列，其中含有各個參照 ZORN'S LEMMA 之 MATHEMATICS 文件的單一直欄 DOC_ID。這些 DOC_ID 值會被結合到主要文件表格，用以擷取作者的名稱與文件文字。

使用 UDF

純量與直欄 UDF 都可在 SQL 陳述式中呼叫，幾乎只要表示式有效之處皆可使用。表格 UDF 則可在 SELECT 的 FROM 子句中被呼叫。但在 UDF 的使用上仍有一些限制：

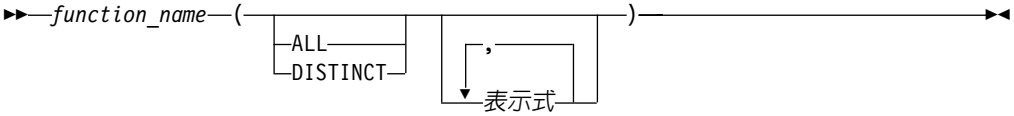
- 核對限制中不得指定 UDF 函數與系統產生的函數。核對限制亦不得含有對 DLVALUE、DLURLPATH、DLURLPATHONLY、DLURLSCHEME、DLURLCOMPLETE、DLURLSERVER、ATAN2、DIFFERENCE、RADIANS、RAND、SOUNDEX、NOW、CURDATE 及 CURTIME 等內建函數的參照。
- ORDER BY 或 GROUP BY 子句中不得參照外部 UDF、SQL UDF 及內建函數 DLVALUE、DLURLPATH、DLURLPATHONLY、DLURLSCHEME、DLURLCOMPLETE 及 DLURLSERVER，除非 SQL 陳述式是唯讀且允許暫時處理程序 (ALWCPYDTA(*YES) 或 (*OPTIMIZE))。

SQL Reference 列有關於所有此類環境的詳細說明。本節的內容與範例著重於相當簡單的 SELECT 陳述式環境，不過其用途並不以此環境爲限。

有關*路徑*與*函數解析*演算法的使用方式與重要性摘要，請參閱第 188 頁的『UDF 概念』。至於此兩者的概念，可參閱 SQL Reference。任何資料操作語言 (DML) 的函數參照解析作業，都須用到函數解析演算法，因此務必要瞭解其運作方式。

參照函數

每一種函數參照，不論其是 UDF 或內建函數，都應遵照下列語法：



其中，function_name 可以是限定的或未限定的函數名稱。請注意，採用 *SYS 命名慣例時，函數不得爲限定的。引數的數量可從 0 到 90，而表示式可含：

- 直欄名稱，限定的或未限定的
- 常數
- 表示式
- 函數
- 主變數
- 含 CAST 函數的參數記號
- 純量子選擇
- 特別暫存區

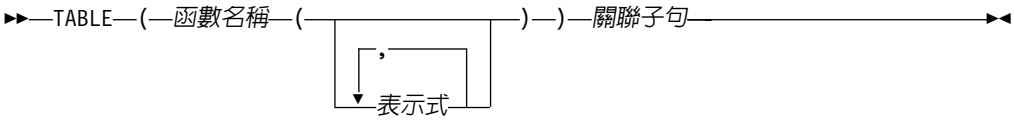
引數的位置相當重要，必須符合函數定義以使語意正確無誤。而引數與函數定義兩者的位置皆須符合函數主體本身。DB2 並不會嘗試打亂引數來使其更符合函數定義，而且 DB2 也不會嘗試決定個別函數參數的語意。

函數呼叫範例

以下是一些有效的函數呼叫範例：

```
AVG(FLOAT_COLUMN)
BLOOP(COLUMN1)
BLOOP(FLOAT_COLUMN + CAST(? AS INTEGER))
BLOOP(:hostvar :indicvar)
BRIAN.PARSE(CONCAT(CHAR_COLUMN,USER), 1, 0, 0, 1)
CTR()
FLOOR(FLOAT_COLUMN)
PABLO.BLOOP(A+B)
PABLO.BLOOP(:hostvar)
"search_schema"(CURRENT PATH, 'GENE')
SUBSTR(COLUMN2,8,3)
QSYS2.FLOOR(AVG(EMP.SALARY))
QSYS2.AVG(QSYS2.FLOOR(EMP.SALARY))
QSYS2.SUBSTR(COLUMN2,11,LENGTH(COLUMN3))
```

參照表格函數



上圖中，function_name 可爲未限定的或限定的函數名稱。請注意，若是採用 *SYS 或 *SQL 命名慣例，使用者定義的表格函數可以是限定的。

以下是一些有效的表格函數呼叫範例：

```
TABLE(TABFUNC()) X
TABLE(BLOOP_TAB(:hostvar, COL1+COL2))
NEW_TABLE TABLE(SCHEMA1.BLOOP_TAB2()) X
```

在函數中使用參數標記或 NULL

對參數標記與 NULL 值兩者而言，有一項重要的限制；您不可只撰寫下述程式碼：

```
BLOOP(?)
```

或

```
BLOOP(NULL)
```

由於函數解析並不知道引數究竟爲何種資料類型，因此它將無法解析參照。您可以用 CAST 規格來提供類型給參數記號或 NULL 值，例如 INTEGER，以供函數解析使用：

```
BLOOP(CAST(? AS INTEGER))
```

或

```
BLOOP(CAST(NULL AS INTEGER))
```

使用限定函數參照

如果您使用了限定函數參照，您即限制了 DB2 搜尋符合該綱目的函數。例如，您具有下列陳述式：

```
SELECT PABLO.BLOOP(COLUMN1) FROM T
```

僅考慮綱目 PABLO 中的 BLOOP 函數。此將無關於使用者 SERGE 是否定義了 BLOOP 函數，或是否有內件的 BLOOP 函數。現在假設使用者 PABLO 已在其綱目中定義了兩個 BLOOP 函數：

```
CREATE FUNCTION BLOOP (INTEGER) RETURNS ...
CREATE FUNCTION BLOOP (DOUBLE) RETURNS ...
```

BLOOP 因此會在 PABLO 綱目中超載，而函數選取演算法將根據引數 column1 的資料類型，選取最佳的 BLOOP。此時，兩個 PABLO.BLOOP 皆會採用數值引數，而倘若 column1 並非數值類型之一，陳述式將失敗。另一方面，若 column1 爲 SMALLINT 或 INTEGER，函數選取作業將會進行解析而選取第一個 BLOOP，但若 column1 爲 DECIMAL 或 DOUBLE，所選的則是第二個 BLOOP。

以下是有關此範例的數個要點：

1. 它說明了引數的晉用。第一個 BLOOP 雖是以 INTEGER 參數所定義，但您仍可將其作爲 SMALLINT 引數來傳遞。其間，函數選取演算法會支援內建資料類型的晉用 (相關詳情，請參閱 SQL Reference)， DB2 則會執行適當的資料值轉換。
2. 如果您基於某些原因而想用 SMALLINT 或 INTEGER 引數來呼叫第二個 BLOOP，您即必須在陳述式中明確採行下述動作：

   ```
   SELECT PABLO.BLOOP(DOUBLE(COLUMN1)) FROM T
   ```

3. 另外，若您想用 DECIMAL 或 DOUBLE 引數來呼叫第一個 BLOOP，您可以根據自己的實際用途來選擇明確的動作：

   ```
   SELECT PABLO.BLOOP(INTEGER(COLUMN1)) FROM T
   SELECT PABLO.BLOOP(FLOOR(COLUMN1)) FROM T
   ```

   您應在 SQL Reference 中詳細查閱此類的其他函數。 INTEGER 函數是 QSYS2 綱目中的內建函數。

使用未限定函數參照

若是使用未限定的函數參照 (而非限定函數參照)，DB2 在搜尋相符函數時，通常會使用函數路徑來限定參照。倘若是 DROP FUNCTION 或 COMMENT ON FUNCTION 函數時，如果它們未針對 *SQL 命名慣例、或 *SQL 命名慣例的*LIBL 進行限定，則其參照是用現行授權 ID 來限定。因此，*您務必要知道您的函數路徑爲何，以及您現行函數路徑的綱目中存有哪些衝突的函數 (如果有)*。例如，假設您是 PABLO 而且您的靜態 SQL 陳述式如下，其中 COLUMN1 的資料類型爲 INTEGER：

```
SELECT BLOOP(COLUMN1) FROM T
```

您建立了兩個在 第 195 頁的『使用限定函數參照』 中被引用的 BLOOP 函數，而且您希望並預期其中之一會被選取。若使用下列預設函數路徑時，如果 QSYS 或 QSYS2 中並無衝突的 BLOOP，則第一個 BLOOP 將被選取 (因爲 column1 爲 INTEGER)：

```
"QSYS","QSYS2","PABLO"
```

不過，假設您忘記了正在使用一個先前爲其他用途所撰寫的 script，來進行前置編譯及連結。在此 script 中，您因爲其某一不適用於現行工作的原因，明確寫下要 SQLPATH 參數來指定下列函數：

```
"KATHY","QSYS","QSYS2","PABLO"
```

如果 Kathy 爲其本身的用途寫了一個 BLOOP 函數，且函數選取作業可以充份解析 Kathy' 的函數，而您的陳述式也可以正確執行。其間您將不會接獲通知，因爲 DB2 會假設您知道自己在作什麼。此時您即必須自行查明您的陳述式有哪些不正確的輸出，並進行必要的更正。

函數參照摘要

不論是對限定的或未限定的參照而言，函數選取演算法都會觀察所有適用函數 (包含內建的與使用者定義的) 是否具有下列項目：

- 給定的名稱
- 於函數參照中，已定義函數與引數有相同數量
- 每一個相同於或晉用自對應引數類型的參數。

(*適用函數*對限定參照而言係指*指定綱目中的函數*，對未限定參照而言則指*函數路徑之綱目中的函數*)。演算法會從這些函數中尋找完全相符者，如果失敗則挑出最佳者。如果僅有未限定參照時，倘若其在不同的綱目中找到兩個一樣好的函數，則是以現行函數路徑來作爲決定因素。有關此演算法的明細，請參閱 SQL Reference。

在 第 195 頁的『使用限定函數參照』 末尾的範例中呈現了一項有趣的特性，也就是*函數參照可以是巢狀的*，即使是參照了相同函數。這對內建函數以及 UDF 而言大致是如此；不過，當涉及直欄函數時便有些限制。

我們修正稍早前的範例：

```
CREATE FUNCTION BLOOP (INTEGER) RETURNS INTEGER ...
CREATE FUNCTION BLOOP (DOUBLE) RETURNS INTEGER ...
```

現在請考慮下列 DML 陳述式：

```
SELECT BLOOP( BLOOP(COLUMN1)) FROM T
```

如果 column1 是 DECIMAL 或 DOUBLE 直欄，內層的 BLOOP 參照將會解析爲上面所定義的第二個 BLOOP。由於此 BLOOP 是傳回 INTEGER，因此外層 BLOOP 會解析至第一個 BLOOP。

另外，如果 column1 是 SMALLINT 或 INTEGER 直欄，內層 bloop 參照將解析爲上面所定義的第一個 BLOOP。由於此 BLOOP 是傳回 INTEGER，因此外層 BLOOP 仍會解析至第一個 BLOOP。此時，您所看到的就是對相同函數的巢狀參照。

以下是一些關於函數參照的附加要點：

- 您可以用 SQL 運算子的名稱之一來定義函數。例如，假設您可以爲 "+" 運算子附加某些意義，來代表具有特殊類型 BOAT 的值。您可以定義下列 UDF：

  ```
  CREATE FUNCTION "+" (BOAT, BOAT) RETURNS ...
  ```

  那麼您可以撰寫下述有效的 SQL 陳述式：

  ```
  SELECT "+"(BOAT_COL1, BOAT_COL2)
  FROM BIG_BOATS
  WHERE BOAT_OWNER = 'Nelson Mattos'
  ```

  請注意，您不可用此方式來超載內建的條件運算子，例如 >、=、LIKE、IN 等。

- 函數選取演算法在解析特定函數的參照時，並不會考慮參照的環境。試以下列略經修改的 BLOOP 函數爲例：

  ```
  CREATE FUNCTION BLOOP (INTEGER) RETURNS INTEGER ...
  CREATE FUNCTION BLOOP (DOUBLE) RETURNS CHAR(10)...
  ```

現在假設您撰寫了下列 SELECT 陳述式：

```
SELECT 'ABCDEFG' CONCAT BLOOP(SMALLINT_COL) FROM T
```

由於 SMALLINT 引數所解析的最佳相符者爲第一個 BLOOP， CONCAT 第二個運算元將解析爲資料類型 INTEGER。此陳述式將會失敗，因爲 CONCAT 所要求的是字串引數。如果未出現第一個 BLOOP，另一 BLOOP 將會中選，且將順利執行陳述式。

- UDF 可用參數或帶有任何 LOB 類型的結果來定義：如 BLOB、CLOB 或 DBCLOB。 DB2 會先在儲存體中將整個 LOB 值具體化然後才呼叫此類函數，即使值的來源是 *LOB 定位器*主變數。例如，試考慮下列 C 語言應用程式的片段：

  ```
  EXEC SQL BEGIN DECLARE SECTION;
    SQL TYPE IS CLOB(150K) clob150K ;        /* LOB host var */
    SQL TYPE IS CLOB_LOCATOR clob_locator1;  /* LOB locator host var */
    char                  string[40];        /* string host var */
  EXEC SQL END DECLARE SECTION;
  ```

  當具有對應參數之函數其某一引數被定義爲 CLOB(500K) 時，主變數 :clob150K 或 :clob_locator1 爲有效。因此，參考第 191 頁的『範例：字串搜尋』 中所定義的 FINDSTRING 時，下列兩者在程式中皆屬有效：

  ```
  ... SELECT FINDSTRING (:clob150K, :string) FROM ...
  ... SELECT FINDSTRING (:clob_locator1, :string) FROM ...
  ```

- 非 SQL UDF 參數或具有 LOB 類型之一的結果，可用 LOCATOR 修飾元來建立。此時，整個 LOB 值將不會在呼叫前被具體化。反之，系統會傳遞一 LOB LOCATOR 給 UDF。

  您也可以對 UDF 參數或具有特殊類型 (以 LOB 爲基礎者) 之結果，使用此功能。此功能僅限於非 SQL UDF 。請注意，此類函數的引數可以是已定義類型的任何 LOB 值；不一定要是定義爲 LOCATOR 類型之一的主變數。以主變數定位器作爲引數，與在 UDF 參數及結果定義中使用 AS LOCATOR，兩者完全無關。

- 定義 UDF 時，可用特殊類型來作爲參數或作爲結果。(之前的範例已作了示範。) DB2 在傳遞相關值給 UDF 時，會採用特殊類型其來源資料類型的格式。

  源自主變數或作爲 UDF (已將對應參數定義爲某一特殊類型者) 引數的特殊類型值，**必須由使用者明確強制轉型為該特殊類型**。特殊類型並無任何主語言類型可供代表。 DB2 的強勢類型會強制作此要求。否則您的結果可能會曖昧不明。因此，請考慮採行透過 BLOB 所定義的 BOAT 特殊類型，並使用來自 第 192 頁的『範例：具有 UDT 參數的外部函數』 的 BOAT_COST UDFCOST (以 BOAT 類型的物作爲其引數)。在下述 C 語言應用程式的片段中，主變數 :ship 保有將傳給 BOAT_COST 函數的 BLOB 值：

  ```
  EXEC SQL BEGIN DECLARE SECTION;
    SQL TYPE IS BLOB(150K) ship;
  EXEC SQL END DECLARE SECTION;
  ```

  下列兩個陳述式皆可正確解析爲 BOAT_COST 函數，因爲兩者都會將 :ship 主變數強制轉型爲類型 BOAT：

  ```
  ... SELECT BOAT_COST (BOAT(:ship)) FROM ...
  ... SELECT BOAT_COST (CAST(:ship AS BOAT)) FROM ...
  ```

  如果資料庫中存有多個 BOAT 特殊類型，或者其他綱目亦有 BOAT UDF，您必須小心處理您的函數路徑。否則您的結果可能會曖昧不明。

使用者定義的特殊類型 (UDT)

使用者定義的特殊類型是一種系統機制，可讓您擴展 DB2 的功能，使之超越現有的內建資料類型。它可讓您在 DB2 中定義新的資料類型進而擁有更大的力量，而不再受限於非得使用系統提供的內建資料類型來打造您的業務或攫取資料意涵。特殊資料類型可讓您以一對一的形式，對映至現有的資料庫類型。

以下主題將進一步詳述各類 UDT：

- 『爲何使用 UDT？』
- 『定義 UDT』
- 第 200 頁的『定義具有 UDT 的表格』
- 第 201 頁的『操作 UDT』
- 第 205 頁的『UDT、UDF 及 LOB 之間的協同作用』

為何使用 UDT？

UDT 的相關優點如下：

1. **擴充性**。

   藉由定義新的類型，您可以無限增加 DB2 所提供的類型集，來支援您的應用。

2. **彈性**。

   您可以透過使用者定義的函數 (UDF)，爲您的新類型指定任何語意和行爲，來增大系統中可用的類型範圍。

3. **行為一致性**。

   強勢類型可確保您的 UDT 作出適當行爲。它可保證只有透過 UDT 所定義的函數才會被應用至 UDT 的案例上。

4. **封裝性**。

   您的 UDT 行爲會受限於其所能套用的函數與運算子。此作法可提供施行上的彈性，因爲執行應用程式時，將不再取決於您爲類型所選擇的內部表示法。

5. **可延伸的行為**。

   使用者定義的函數可增加既有的功能，以便隨時處理您的 UDT。(請參閱第 185 頁的『使用者定義的函數 (UDF)』)

6. **物件導向延伸套件的基礎**。

   UDT 是大部份物件導向特性的基礎。它們是邁向物件導向延伸套件的最重要步驟。

定義 UDT

就如同表格、索引及 UDF 一般，UDT 須用 CREATE 陳述式來加以定義。

請透過 CREATE DISTINCT TYPE 陳述式來定義新的 UDT。有關陳述式語法及其所有選項的詳細資訊，請參閱 SQL Reference 中的 CREATE DISTINCT TYPE。

對 CREATE DISTINCT TYPE 陳述式而言，請注意：

1. 新的 UDT 名稱可以是已限定或未限定的名稱。
2. UDT 的來源類型是供 DB2 用於內部代表 UDT。因此，它必須是內建的資料類型。先前所定義的 UDT 不得作爲其他 UDT 的來源類型。

基於 UDT 的定義，DB2 一律會產生強制轉型函數來：

- 將 UDT 強制轉型爲來源類型 (使用來源類型的標準名稱)。例如，倘若您根據 FLOAT 建立了某一特殊類型，名爲 DOUBLE 的強制轉型函數亦會同時建立。
- 將來源類型強制轉型爲 UDT。有關何時會對 UDT 產生額外強制轉型的詳情，請參閱 SQL Reference。

此類函數對於操作查詢中的 UDT 十分重要。

解析未限定 UDT

函數路徑是用來解析任何對未限定類型名稱或函數的參照，除非類型名稱或函數已
- 建立
- 捨棄
- 加註。

有關如何解析未限定函數參照的詳情，請參閱第 195 頁的『使用限定函數參照』。

範例：使用 CREATE DISTINCT TYPE

以下是使用 CREATE DISTINCT TYPE 的範例：
- 範例：貨幣
- 範例：履歷表

註: 如需相關的程式碼範例，請參閱第 x 頁的『程式碼不保事項聲明』。

範例：貨幣

假設您正在撰寫的應用程式須處理不同的貨幣，而您希望確保 DB2 不允許此類貨幣直接與另一貨幣相比較或進行處理。請記得，每當您想要比較不同貨幣的價值時，便須執行換算。因此您定義了所需數目的 UDT；每種需要用到的貨幣各有一種 UDT：

```
CREATE DISTINCT TYPE US_DOLLAR AS DECIMAL (9,2)
CREATE DISTINCT TYPE CANADIAN_DOLLAR AS DECIMAL (9,2)
CREATE DISTINCT TYPE GERMAN_MARK AS DECIMAL (9,2)
```

範例：履歷表

假設您想將貴公司應徵者所填寫的求職表保存至 DB2 表格中，而且您將使用函數來提取此類求職表的資訊。由於前述函數無法適用於一般字串 (其將無法找到應該傳回的資訊)，因此您定義了 UDT 來代表所填寫的求職表：

```
CREATE DISTINCT TYPE PERSONAL.APPLICATION_FORM AS CLOB(32K)
```

定義具有 UDT 的表格

定義了數個 UDT 後，您即可開始定義直欄類型爲 UDT 的表格。以下是使用 CREATE TABLE 的範例：
- 範例：營業額
- 範例：求職表

註: 如需相關的程式碼範例，請參閱 第 x 頁的『程式碼不保事項聲明』。

範例：營業額

假設您想定義相關表格，以便用不同幣別來保存公司的營業額：

```
CREATE TABLE US_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1985),
   TOTAL         US_DOLLAR)
```

```
CREATE TABLE CANADIAN_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1985),
   TOTAL         CANADIAN_DOLLAR)

CREATE TABLE GERMAN_SALES
  (PRODUCT_ITEM  INTEGER,
   MONTH         INTEGER CHECK (MONTH BETWEEN 1 AND 12),
   YEAR          INTEGER CHECK (YEAR > 1985),
   TOTAL         GERMAN_MARK)
```

上述範例中的 UDT 是以第 200 頁的『範例：貨幣』 中的同一 CREATE DISTINCT TYPE 陳述式所建立。請注意，前述範例皆使用了核對限制。有關核對限制的詳情，請參閱第 125 頁的『新增與使用核對限制』。

範例：求職表

假設您必須定義一表格，以便用於保存應徵者所填寫的求職表，如下所示：

```
CREATE TABLE APPLICATIONS
  (ID               INTEGER,
   NAME             VARCHAR (30),
   APPLICATION_DATE DATE,
   FORM             PERSONAL.APPLICATION_FORM)
```

您已充份限定了 UDT 名稱，因爲其限定元與您的授權 ID 不同，而且您並未變更預設函數路徑。請記得，每當類型與函數名稱未經充份限定時，DB2 將搜尋現行函數路徑中所列的各個綱目，尋找相符於給定之未限定名稱的類型或函數名稱。

操作 UDT

有關 UDT 的最重要概念之一是*強勢類型*。強勢類型可保證唯有已透過 UDT 來定義的函數及運算子可套用至其案例上。

強勢類型對確保 UDT 案例的正確性而言十分重要。例如，若您定義了一個函數來根據現行匯率將美元轉換爲加拿大幣，您即不該使用此函數來將德國馬克轉換爲加拿大幣，因爲所傳回的必定是錯誤的金額。

施行強勢類型後，DB2 將不許您撰寫查詢，進行諸如 UDT 案例與 UDT 來源類型案例的比較。基於相同的理由，DB2 不許您對 UDT 套用定義於其他類型上的函數。如果您想比較 UDT 案例與另一類型的案例，必須將其中一種比較強制轉型。同理，若您想對 UDT 案例套用未對 UDT 定義的函數時，即必須將 UDT 案例強制轉型爲該函數之參數的類型。

如需有關操作 UDT 的範例，請參閱『操作 UDT 的範例』。

操作 UDT 的範例

以下是有關操作 UDT 的範例：

- 範例：UDT 與常數之間的比較
- 範例：UDT 之間的強制轉型
- 範例：涉及 UDT 的比較
- 範例：涉及 UDT 的來源 UDF
- 範例：涉及 UDT 的分派
- 範例：動態 SQL 中的分派

- 範例：涉及不同 UDT 的分派
- 範例：在 UNION 中使用 UDT

註: 如需相關的程式碼範例，請參閱 第 x 頁的『程式碼不保事項聲明』。

範例：UDT 與常數之間的比較

假設您想知道在 1992 年 7 月份 (7/92) 時，哪些產品的銷售量超過 US $100 000.00。

```
SELECT PRODUCT_ITEM
  FROM   US_SALES
  WHERE  TOTAL > US_DOLLAR (100000)
  AND    month = 7
  AND    year  = 1992
```

由於您不可直接比較美元與美元來源類型的案例 (即，DECIMAL)，因此您使用 DB2 所提供的強制轉型函數，將 DECIMAL 強制轉型爲美元。您還可使用 DB2 提供的其他強制轉型函數 (亦即，可將美元強制轉型爲 DECIMAL 者)，將直欄總計強制轉型爲 DECIMAL。不論是哪種強制轉型方式，您都可使用強制轉型規格表示法或函數表示法，來執行強制轉型。亦即，您可以將上述查詢寫成：

```
SELECT PRODUCT_ITEM
  FROM   US_SALES
  WHERE  TOTAL > CAST (100000 AS us_dollar)
  AND    MONTH = 7
  AND    YEAR  = 1992
```

範例：UDT 之間的強制轉型

假設您想定義一 UDF 用來將加拿大幣轉換爲美元。設若您可以從位於 DB2 外部的檔案中取得現行匯率。那麼您可以定義一 UDF 先取得加拿大幣價值、存取匯率檔案並傳回對應的美元金額。

乍看之下，此種 UDF 似乎寫起來頗爲容易。不過，並非所有的 C 編譯器皆可支援 DECIMAL 值。之前您已將代表不同貨幣的 UDT 皆定義爲 DECIMAL。而您的 UDF 將須接收並傳回 DOUBLE 值，因爲這是 C 所提供唯一可用來表示 DECIMAL 值而不致流失小數位數的資料類型。因此，您的 UDF 應定義如下：

```
CREATE FUNCTION CDN_TO_US_DOUBLE(DOUBLE) RETURNS DOUBLE
  EXTERNAL NAME 'MYLIB/CURRENCIES(C_CDN_US)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  NOT DETERMINISTIC
```

由於每次呼叫 UDF 時，加拿大幣與美元之間的匯率有可能改變，所以您加以宣告爲 NOT DETERMINISTIC。

現在的問題是，如何將加拿大幣傳給此 UDF 並從中取得美元？此時加拿大幣必須強制轉型爲 DECIMAL 值。然後再將 DECIMAL 值強制轉型爲 DOUBLE。您還須將傳回的 DOUBLE 值強制轉型爲 DECIMAL，再將 DECIMAL 值強制轉型爲美元。

這些強制轉型都可在您定義來源 UDF 時由 DB2 自動執行，但來源 UDF 的參數與傳回類型並不會完全符合來源函數的參數與傳回類型。因此，您必須定義兩個來源 UDF。第一個可將 DOUBLE 值變爲 DECIMAL 表示法。第二個則將 DECIMAL 值傳給 UDT。其定義方式如下：

```
CREATE FUNCTION CDN_TO_US_DEC (DECIMAL(9,2)) RETURNS DECIMAL(9,2)
  SOURCE CDN_TO_US_DOUBLE (DOUBLE)

CREATE FUNCTION US_DOLLAR (CANADIAN_DOLLAR) RETURNS US_DOLLAR
  SOURCE CDN_TO_US_DEC (DECIMAL())
```

請注意，當以 US_DOLLAR(C1) 呼叫 US_DOLLAR 函數時 (其中的 C1 是類型爲加拿大幣的直欄)，其效果將與下列呼叫相同：

```
US_DOLLAR (DECIMAL(CDN_TO_US_DOUBLE (DOUBLE (DECIMAL (C1)))))
```

亦即，C1 (以加拿大幣爲單位) 會強制轉型爲十進位，然後再強制轉型爲倍整數值以傳遞給 CDN_TO_US_DOUBLE 函數。此函數會存取匯率檔案並傳回倍整數值 (代表美元金額)，此值將強制轉型爲十進位，然後再轉成美元。

用於將德國馬克轉換成美元的函數，其作法將與上例相似：

```
CREATE FUNCTION GERMAN_TO_US_DOUBLE(DOUBLE)
  RETURNS DOUBLE
  EXTERNAL NAME 'MYLIB/CURRENCIES(C_GER_US)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  NOT DETERMINISTIC

CREATE FUNCTION GERMAN_TO_US_DEC (DECIMAL(9,2))
  RETURNS DECIMAL(9,2)
  SOURCE GERMAN_TO_US_DOUBLE(DOUBLE)

CREATE FUNCTION US_DOLLAR(GERMAN_MARK) RETURNS US_DOLLAR
  SOURCE GERMAN_TO_US_DEC (DECIMAL())
```

範例：涉及 UDT 的比較

假設您想知道在 1989 年 3 月 (3/89) 時，哪些產品於美國的銷售量是超過加拿大與德國的：

```
SELECT US.PRODUCT_ITEM, US.TOTAL
  FROM US_SALES AS US, CANADIAN_SALES AS CDN, GERMAN_SALES AS GERMAN
  WHERE US.PRODUCT_ITEM = CDN.PRODUCT_ITEM
  AND US.PRODUCT_ITEM = GERMAN.PRODUCT_ITEM
  AND US.TOTAL > US_DOLLAR (CDN.TOTAL)
  AND US.TOTAL > US_DOLLAR (GERMAN.TOTAL)
  AND US.MONTH = 3
  AND US.YEAR  = 1989
  AND CDN.MONTH = 3
  AND CDN.YEAR  = 1989
  AND GERMAN.MONTH = 3
  AND GERMAN.YEAR  = 1989
```

由於您無法直接比較美元與加幣 (或德國馬克)，因此便透過 UDF 將加幣金額強制轉型爲美元，並用另一 UDF 將德國馬克金額強制轉型爲美元。您無法將其全部強制轉型爲 DECIMAL 並比較轉換後的 DECIMAL 值，因爲各金額採不同幣別所以無法比較。

範例：涉及 UDT 的來源 UDF

假設您已根據內建的 SUM 函數定義了來源 UDF，以支援對「德國馬克」的總和：

```
CREATE FUNCTION SUM (GERMAN_MARKS)
  RETURNS GERMAN_MARKS
  SOURCE SYSIBM.SUM (DECIMAL())
```

您想知道各產品於 1994 年度在德國的銷售總額。而您想用美元來取得銷售總額：

```
SELECT PRODUCT_ITEM, US_DOLLAR (SUM (TOTAL))
  FROM GERMAN_SALES
  WHERE YEAR = 1994
  GROUP BY PRODUCT_ITEM
```

您不可撰寫 **SUM** (us_dollar (total))，除非您已比照前述方式針對美元定義了 SUM 函數。

範例：涉及 UDT 的分派

假設您想將新應徵者所填的求職表存入資料庫。您定義了一個主變數，其中含有用來代表求職表的字串值：

```
EXEC SQL BEGIN DECLARE SECTION;
  SQL TYPE IS CLOB(32K) hv_form;
EXEC SQL END DECLARE SECTION;

/* Code to fill hv_form */

INSERT INTO APPLICATIONS
  VALUES (134523, 'Peter Holland', CURRENT DATE, :hv_form)
```

您並未明確呼叫強制轉型函數來將字串轉換至 UDT personal.application_form。這是因爲 DB2 允許您將 UDT 之來源類型的案例，指派給具有此 UDT 的目標。

範例：動態 SQL 中的分派

如果您想將 『範例：涉及 UDT 的分派』 中所給定的同一陳述式用於動態 SQL 中，可按下述方法使用參數標記：

```
EXEC SQL BEGIN DECLARE SECTION;
  long id;
  char  name[30];
  SQL TYPE IS CLOB(32K) form;
  char command[80];
EXEC SQL END DECLARE SECTION;

/* Code to fill host variables */

strcpy(command,"INSERT INTO APPLICATIONS VALUES");
strcat(command,"(?, ?, CURRENT DATE, ?)");

EXEC SQL PREPARE APP_INSERT FROM :command;
EXEC SQL EXECUTE APP_INSERT USING :id, :name, :form;
```

您用了 DB2 的強制轉型規格來告知 DB2 參數記號的類型爲 CLOB(32K)，而此類型可指派給 UDT 直欄。請記得，您不可宣告 UDT 類型的主變數，因爲主語言不支援 UDT。因此，您不得將參數記號的類型指定爲 UDT。

範例：涉及不同 UDT 的分派

假設您已根據內建的 SUM 函數定義了兩個來源 UDF，用以支援美元與加幣的 SUM，其作法就類似於第 203 頁的『範例：涉及 UDT 的來源 UDF』 中以德國馬克爲來源的 UDF：

```
CREATE FUNCTION SUM (CANADIAN_DOLLAR)
  RETURNS CANADIAN_DOLLAR
  SOURCE SYSIBM.SUM (DECIMAL())

CREATE FUNCTION SUM (US_DOLLAR)
  RETURNS US_DOLLAR
  SOURCE SYSIBM.SUM (DECIMAL())
```

現在假設您的上司要求您以美元來維護各產品在不同國家的年度銷售總額，而且使用不同的表格：

```
CREATE TABLE US_SALES_94
  (PRODUCT_ITEM  INTEGER,
   TOTAL         US_DOLLAR)

CREATE TABLE GERMAN_SALES_94
  (PRODUCT_ITEM  INTEGER,
   TOTAL         US_DOLLAR)

CREATE TABLE CANADIAN_SALES_94
  (PRODUCT_ITEM  INTEGER,
   TOTAL         US_DOLLAR)

INSERT INTO US_SALES_94
  SELECT PRODUCT_ITEM, SUM (TOTAL)
  FROM US_SALES
  WHERE YEAR = 1994
  GROUP BY PRODUCT_ITEM

INSERT INTO GERMAN_SALES_94
  SELECT PRODUCT_ITEM, US_DOLLAR (SUM (TOTAL))
  FROM GERMAN_SALES
  WHERE YEAR = 1994
  GROUP BY PRODUCT_ITEM

INSERT INTO CANADIAN_SALES_94
  SELECT PRODUCT_ITEM, US_DOLLAR (SUM (TOTAL))
  FROM CANADIAN_SALES
  WHERE YEAR = 1994
  GROUP BY PRODUCT_ITEM
```

由於不同的 UDT 彼此無法直接指派，所以您明確地將加幣與德國馬克的金額強制轉型爲美元。您不可使用強制轉型規格語法，因爲 UDT 只能強制轉型爲其本身的來源類型。

範例：在 UNION 中使用 UDT

假設您想對您的美國使用者提供查詢畫面，以顯示貴公司各項產品的所有銷售額：

```
SELECT PRODUCT_ITEM, MONTH, YEAR, TOTAL
FROM US_SALES
UNION
SELECT PRODUCT_ITEM, MONTH, YEAR, US_DOLLAR (TOTAL)
FROM CANADIAN_SALES
UNION
SELECT PRODUCT_ITEM, MONTH, YEAR, US_DOLLAR (TOTAL)
FROM GERMAN_SALES
```

您將加幣強制轉型爲美元，並將德國馬克強制轉型爲美元，因爲 UDT 只能和相同的 UDT 比較。您必須使用函數表示法來強制轉型不同的 UDT，因爲強制轉型規格只允許您在 UDT 及其來源類型間進行強制轉型。

UDT、UDF 及 LOB 之間的協同作用

在前面各節中，您學到了如何定義及使用各種 DB2 物件延伸套件 (UDT、UDF 及 LOB)。不過在本節中，您將會瞭解在此三種物件延伸套件之間存有許多協同作用。

相關明細，請參閱以下各節，

- 第 206 頁的『結合 UDT、UDF 與 LOB』
- 第 206 頁的『複雜應用程式的範例』

結合 UDT、UDF 與 LOB

根據物件導向的概念，應用程式領域中的相似物件會群組成相關類型。這些類型都有其名稱 (內部表示法) 與行爲。藉由使用 UDT，您就可將新類型的名稱及其內部表示法告知 DB2。LOB 即可能是您的新類型在內部的表示法之一，而且它最適合用來表示龐大、複雜的結構。而藉由使用 UDF，您則可定義新類型的行爲。因此，在 UDT、UDF 及 LOB 間實可謂存在著重要的協同作用。其間，具有複雜資料結構與行爲的應用程式類型是以 UDT 來打造，此 UDT 在內部則以 LOB 形式存在，另外再以 UDF 來建置應用程式類型的行爲。至於負責支配應用程式類型其語意完整性的規則，則是以限制與觸發來呈現。爲能有效控制及組織您的相關 UDT 與 UDF，您應該將其保存在相同綱目下。

複雜應用程式的範例

下述範例將示範如何在複雜的應用程式中搭配使用 UDT、UDF 及 LOB：

範例：定義 UDT 與 UDF
範例：使用 LOB 函數來大量輸入資料至資料庫
範例：使用 UDF 來查詢 UDT 案例
範例：使用 LOB 定位器來操作 UDT 案例

註: 如需相關的程式碼範例，請參閱第 x 頁的『程式碼不保事項聲明』。

範例：定義 UDT 與 UDF

假設您想將寄給貴公司的電子郵件 (e-mail) 保存在 DB2 表格中。撇開任何隱私問題，您打算就此種電子郵件撰寫查詢，以查明其主旨、您的電子郵件服務被用於接收客戶訂單的頻率等資訊。電子郵件可以是相當龐大的，而且它具有複雜的內部結構 (寄件人、收件人、主旨、日期及郵件內容)。因此，您決定藉助來源類型爲大型物件的 UDT 來代表電子郵件。您對您的電子郵件類型定義了一組 UDF，例如用來提取郵件主旨、寄件人、日期等資料的函數。您還定義了一些其他函數，用以搜尋電子郵件的內容。而您是使用下述 CREATE 陳述式來進行作業：

```
CREATE DISTINCT TYPE E_MAIL AS BLOB (1M)

CREATE FUNCTION SUBJECT (E_MAIL)
  RETURNS VARCHAR (200)
  EXTERNAL NAME 'LIB/PGM(SUBJECT)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION

CREATE FUNCTION SENDER (E_MAIL)
  RETURNS VARCHAR (200)
  EXTERNAL NAME 'LIB/PGM(SENDER)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION

CREATE FUNCTION RECEIVER (E_MAIL)
  RETURNS VARCHAR (200)
  EXTERNAL NAME 'LIB/PGM(RECEIVER)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION
```

```
CREATE FUNCTION SENDING_DATE (E_MAIL)
  RETURNS DATE CAST FROM VARCHAR(10)
  EXTERNAL NAME 'LIB/PGM(SENDING_DATE)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION

CREATE FUNCTION CONTENTS (E_MAIL)
  RETURNS BLOB (1M)
  EXTERNAL NAME 'LIB/PGM(CONTENTS)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION

CREATE FUNCTION CONTAINS (E_MAIL, VARCHAR (200))
  RETURNS INTEGER
  EXTERNAL NAME 'LIB/PGM(CONTAINS)'
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  NO SQL
  DETERMINISTIC
  NO EXTERNAL ACTION

CREATE TABLE ELECTRONIC_MAIL
  (ARRIVAL_TIMESTAMP TIMESTAMP,
   MESSAGE E_MAIL)
```

範例：使用 LOB 函數來大量輸入資料至資料庫

假設您藉由將保存在檔案中的電子郵件轉送給 DB2，來爲表格大量輸入資料。您將使用不同的 HV_EMAIL_FILE 值多次執行下述 INSERT 陳述式，直到將所有電子郵件存入 DB2：

```
EXEC SQL BEGIN DECLARE SECTION
  SQL TYPE IS BLOB_FILE HV_EMAIL_FILE;

EXEC SQL END DECLARE SECTION
  strcpy (HV_EMAIL_FILE.NAME, "/u/mail/email/mbox");
  HV_EMAIL_FILE.NAME_LENGTH = strlen(HV_EMAIL_FILE.NAME);
  HV_EMAIL_FILE.FILE_OPTIONS = 2;

EXEC SQL INSERT INTO ELECTRONIC_MAIL
  VALUES (CURRENT TIMESTAMP, :hv_email_file);
```

DB2 LOB 支援能力所提供的全部函數皆適用於來源類型爲 LOB 的 UDT。因此，您使用了 LOB 檔案參考變數，將檔案內容指派到 UDT 直欄中。您並未使用強制轉型函數，來將 BLOB 類型的值轉換爲您的電子郵件類型。這是因爲 DB2 允許您將來源類型爲特殊類型的值，指派給特殊類型的目標。

範例：使用 UDF 來查詢 UDT 案例

假設您需要知道特定客戶對客戶訂單寄出了幾封電子郵件，而且您在客戶表格中存有客戶的電子郵件地址。

```
SELECT COUNT (*)
  FROM ELECTRONIC_MAIL AS EMAIL, CUSTOMERS
  WHERE SUBJECT (EMAIL.MESSAGE) = 'customer order'
  AND CUSTOMERS.EMAIL_ADDRESS = SENDER (EMAIL.MESSAGE)
  AND CUSTOMERS.NAME = 'Customer X'
```

您在此 SQL 查詢中使用了在 UDT 上所定義的 UDF，因爲這是處理 UDT 的唯一方法。因此，您的 UDT 電子郵件已經完全封裝。其內部表示法與結構已被隱藏，而且只能由經過定義的 UDF 來加以處理。這些 UDF 知道如何解譯資料，而不需曝露其表示法。

假設您需要知道貴公司在 1994 年所收到且與產品在市場上的表現有關的全部電子郵件明細。

```
SELECT SENDER (MESSAGE), SENDING_DATE (MESSAGE), SUBJECT (MESSAGE)
  FROM ELECTRONIC_MAIL
  WHERE CONTAINS (MESSAGE,
  '"performance" AND "products" AND "marketplace"') = 1
```

其中您使用了 contains UDF，它可以分析訊息內容並搜尋相關關鍵字與同義字。

範例：使用 LOB 定位器來操作 UDT 案例

假設您想要取得特定電子郵件的相關資訊，而不想將整個電子郵件轉送到您應用程式的主變數中。 (請記得，電子郵件可以是相當龐大的。) 由於您的 UDT 是定義於 LOB 上的，因此可用 LOB 定位器來達成目的：

```
EXEC SQL BEGIN DECLARE SECTION
  long hv_len;
  char hv_subject[200];
  char hv_sender[200];
  char hv_buf[4096];
  char hv_current_time[26];
  SQL TYPE IS BLOB_LOCATOR hv_email_locator;
EXEC SQL END DECLARE SECTION

EXEC SQL SELECT MESSAGE
  INTO :hv_email_locator
  FROM ELECTRONIC MAIL
  WHERE ARRIVAL_TIMESTAMP = :hv_current_time;

EXEC SQL VALUES (SUBJECT (E_MAIL(:hv_email_locator))
  INTO :hv_subject;
.... code that checks if the subject of the e_mail is relevant ....
.... if the e_mail is relevant, then.............................

EXEC SQL VALUES (SENDER (CAST (:hv_email_locator AS E_MAIL)))
  INTO :hv_sender;
```

由於您的主變數其類型爲 BLOB 定位器 (UDT 的來源類型)，每當其被作爲 UDF (定義於 UDT 上者)，您都會明確地將 BLOB 定位器轉換爲 UDT。

使用 DataLink

DataLink 資料類型是擴展可儲存至資料庫檔案之資料類型時，不可或缺的基本構件之一。DataLink 的理念係指實際儲存在直欄中的資料純粹是物件的指標。此物件可以是任何東西，如影像檔、錄音、文字檔等。而用來將其解析爲物件的方法，就是儲存全球資源定位器 (URL)。此即表示表格中的列可用來收納傳統資料類型之物件的相關資訊，而物件本身可用 DataLink 資料類型來加以參照。使用者可藉新的 SQL 純量函數來取得指至物件以及存放物件之伺服器的路徑。透過 DataLink 資料類型，資料列與物件之間將存有一相當鬆散的關係。例如，刪除某列將會切斷對 DataLink 所參照之物件的關係，但物件本身可能不會遭刪除。

以 DataLink 直欄所建立的 SQL 表格，可用來保存物件的相關資訊，而不需實際含有物件本身。此概念讓使用者擁有更多彈性，使其可藉由 SQL 表格來管理資料類型。例

如，若使用者在其伺服器的整合檔案系統中存有數千影像片段，可能會希望用 SQL 表格來收納關於這些影像片段的資訊。但由於使用者已將此類物件存入目錄中，他們只想用 SQL 表格來收納對物件的參照，而非收納實際的物件內容。此時的良好解決方案就是使用 DataLink。 SQL 表格會用傳統的 SQL 資料類型來收納關於各影像片段的資訊，例如標題、長度、日期等。但影像片段本身則是用 DataLink 直欄來加以參照。表格中的每一列將儲存物件的 URL 以及選用的註解。當應用程式要使用這些影像片段時，即可藉 SQL 介面擷取 URL，然後以瀏覽器或其他播放軟體來處理 URL 進而顯示影像片段。

使用此技巧的優點有：

- 整合檔案系統可儲存任何類型的串流檔。
- 整合檔案系統可儲存相當大的物件，而這些物件可能無法放入字元直欄甚至 LOB 直欄中。
- 整合檔案系統的階層性相當適合用來組織及處理串流檔物件。
- 藉由將物件保留在資料庫外部的整合檔案系統中，應用程式將可獲致更高的效能，因爲 SQL 執行時間引擎將只需處理查詢與報表，而檔案系統則專司處理影像的串流、顯示等。

使用 DataLink 還可在物件處於「已鏈結」狀態下時提供物件的控制能力。建立 DataLink 直欄後，若 SQL 表格中有任何列參照了物件，受參照物件將無法刪除、移動或更名。此物件將被視爲已鏈結。一旦含有參照的列遭刪除後，該物件即會解除鏈結。若要充份瞭解此概念，您應先熟悉在建立 DataLink 直欄時可供指定的控制層次。如需有關建立 DataLink 直欄時所用的確實語法，請參閱 SQL Reference。

有關 DataLink 的明細，請參閱以下各節：

- 『NO LINK CONTROL』
- 『FILE LINK CONTROL (含檔案系統許可權)』
- 第 210 頁的『FILE LINK CONTROL (含資料庫許可權)』
- 第 210 頁的『用於處理 DataLink 的指令』

NO LINK CONTROL

當直欄是以 NO LINK CONTROL 建立時，若資料列被新增至 SQL 表格中，此時將不會有鏈結發生。系統雖會驗證 URL 的語法是否正確，但不會檢查伺服器是否可以存取，或者檔案是否存在。

FILE LINK CONTROL (含檔案系統許可權)

以具有檔案系統 (FS) 許可權的 FILE LINK CONTROL 來建立 DataLink 直欄時，系統會驗證任何 DataLink 值是否爲有效的 URL，其必須具備有效的伺服器名稱及檔名。當該列被插入至 SQL 表格中時，此檔案必須存在。若找到物件時，它即會被標示爲已鏈結。此即表示其間該物件不得移動、刪除或更名。此外，物件亦不得被鏈結超過一次以上。如果 URL 的伺服器名稱部份指向某一遠端系統，該系統必須是可存取的。若含有 DataLink 的列遭到刪除，該物件即會解除鏈結。若 DataLink 值被更新爲不同的值，舊的物件會解除鏈結，新的解除鏈結則被鏈結。

受鏈結物件的管理同樣是由整合檔案系統負責。進行鏈結或解除鏈結之處理時，許可權並不會修改。此選項可用來在物件受鏈結時，控制物件的存在性。

FILE LINK CONTROL (含資料庫許可權)

以具有資料庫許可權的 FILE LINK CONTROL 來建立 DataLink 直欄時，URL 會接受驗證，且所有現有的物件許可權皆會遭到移除。物件的所有權會變更爲系統所提供的特殊使用者設定檔。在物件受鏈結期間，要存取物件的唯一方式是向其所屬的 SQL 表格取得 URL。此動作是以附加至 SQL 所傳回 URL 的特殊存取記號來處理。若無此存取記號，所有對物件的存取嘗試皆會因違反權限而遭致失敗。如果已藉正常方式 (FETCH、SELECT INTO 等) 從 SQL 表格擷取帶有存取記號的 URL，檔案系統過濾器會先驗證此存取記號，進而允許存取該物件。

此選項可提供相關控制能力，防止受鏈結物件因使用者嘗試直接存取該物件而遭致更新。由於存取物件的唯一方式是向 SQL 作業取得存取記號，因此管理者可透過 SQL 表格(含有 DataLink 直欄) 的資料庫許可權，來有效控制受鏈結物件的存取。

用於處理 DataLink 的指令

對 DataLink 資料類型的支援可再細分爲 3 個不同的元件：

1. 具有名爲 DATALINK 之資料類型的 DB2 資料庫支援。此支援可在諸如 CREATE TABLE 與 ALTER TABLE 等 SQL 陳述式中指定。其間，直欄不得具有 NULL 以外的任何預設值。存取資料時必須使用 SQL 介面。這是因爲 DATALINK 本身不相容於任何主變數類型。 SQL 純量函數可用來擷取字元形式的 DATALINK 值。若要在直欄中 INSERT 及 UPDATE 相關值，須在 SQL 中使用 DLVALUE 純量函數。
2. 資料鏈結檔案管理程式 (DLFM)，此元件可用於維護伺服器檔案的鏈結狀態，並追蹤各檔案的中繼資料。它會負責處理鏈結、解除鏈結和確定控制方面的問題。 DataLink 的重要事項之一是含有 DataLink 直欄之 SQL 表格所在的相同實體系統上不得有 DLFM。如此 SQL 表格才可鏈結常駐於同一系統之整合檔案系統中，或遠端伺服器之整合檔案系統中的物件。
3. DataLink 過濾器，當檔案系統嘗試對含有受鏈結物件之目錄中的檔案進行作業時，須先執行此過濾器。此元件會決定檔案是否受到鏈結，以及使用者是否經過授權來存取此檔案 (選用)。若檔名中含有存取記號，此記號亦會接受驗證。由於此種過濾處理需要額外執行時間，因此只有在所存取的物件存在於「字首」中的目錄之一時才會執行。有關字首的詳情，請參閱下文。

使用 DataLink 時，必須採行一些相關步驟來適當地配置系統：

- 與使用 DataLink 相關的所有系統上皆須配置 TCP/IP。其中包括將建立 DataLink 直欄之 SQL 表格所在的系統，以及將含有待鏈結物件的系統。而在多數情況下，此兩者會是同一系統。由於用來參照物件的 URL 中含有 TCP/IP 伺服器名稱，將含有 DataLink 的系統須能辨識此名稱。 CFGTCP 指令可用來配置 TCP/IP 名稱，或登記 TCP/IP 名稱伺服器。
- 含有 SQL 表格的系統須更新「關聯式資料庫目錄」，以反映本端資料庫系統及任何選用的遠端系統。 WRKRDBDIRE 指令則可用來新增或修改此目錄下的資訊。爲求一致性，TCP/IP 伺服器名稱與「關聯式資料庫」名稱兩者最好相同。
- 在任何將含待鏈結物件的系統上，須先啓動 DLFM 伺服器。 STRTCPSVR *DLFM 指令可用來啓動 DLFM 伺服器。若要結束 DLFM 伺服器，則可使用 CL 指令 ENDTCPSVR *DLFM。

一旦 DLFM 啓動後，還需執行一些相關步驟來配置 DLFM。此類 DLFM 函數可經由 script 來提供，script 則可從 QShell 介面輸入。若要存取此交談式 shell 介面，請使用 CL 指令 QSH。它會帶出一個指令輸入螢幕，供您輸入 DLFM script 指令。 script 指

令 dfmadmin -help 可用來顯示說明本文與語法圖。對於最常用的函數而言，它也提供有 CL 指令。透過 CL 指令，即可完成大部份或全部的 DLFM 配置作業，而不需用到 script 介面。您可以依據偏好，選擇使用 QSH 指令輸入螢幕的 script 指令，或是 CL 指令輸入螢幕的的 CL 指令。

由於這些函數對系統管理者或資料庫管理者而言都很重要，因此都需要 *IOSYSCFG 特殊權限才可使用。

新增字首 - 字首是一路徑或目錄，其中將含有待鏈結的物件。於系統上設定 DLFM 時，管理者須新增任何將用於 DataLink 的字首。 script 指令 dfmadmin -add_prefix 可用來新增字首。可用於新增字首的 CL 指令則是 ADDPFXDLFM。

例如，在伺服器 TESTSYS1 上，有一個名爲 /mydir/datalinks/ 的目錄，其中含有將被鏈結的物件。 管理者使用了指令 ADDPFXDLFM PREFIX(('/mydir/datalinks/')) 來新增字首。此時，下列 URL 鏈結：

```
http://TESTSYS1/mydir/datalinks/videos/file1.mpg
```

或

```
file://TESTSYS1/mydir/datalinks/text/story1.txt
```

都是有效的，因爲其路徑皆具備有效的字首。

script 指令 dfmadmin -del_prefix 也可用來來移除字首。但此功能不常使用，因爲只有當字首名稱所含的目錄結構中沒有任何受鏈結物件時才能加以執行。

新增主資料庫 - 主資料庫是一關聯式資料庫系統，也就是發出鏈結要求的所在。如果 DLFM 與將含有 DataLink 之 SQL 表格同在一個系統上時，只有本端資料庫名稱需要新增。如果 DLFM 將具有來自遠端系統的鏈結要求，則其所有名稱皆須登記至 DLFM。用於新增主資料庫的 script 指令是 dfmadmin -add_db，而 CL 指令則是 ADDHDBDLFM。此函數還會要求必須登記內含 SQL 表格的檔案庫。

例如，若您已在伺服器 TESTSYS1 上新增了 /mydir/datalinks/ 字首，而您想將檔案庫 TESTDB 或 PRODDB 中的本端系統 SQL 表格鏈結到此伺服器上的物件。此時可使用下列指令：**ADDHDBDLFM HOSTDBLIB((TESTDB) (PRODDB)) HOSTDB(TESTSYS1)**

一旦 DLFM 已啓動，且字首與主資料庫名稱皆已登記後，您就可開始鏈結檔案系統中的物件。

第 13 章 撰寫使用者定義的函數 (UDF)

使用者定義的函數 (UDF) 是由三種類型組成：來源、外部及 SQL。來源函數 UDF 會呼叫其函數以執行作業。SQL 及外部函數 UDF 需要您撰寫及執行個別的程式碼。本章是有關撰寫 SQL 及外部函數。若要撰寫外部及 SQL 函數，您必須執行下列各項：

- 瞭解『UDF 執行時間環境』
- 登錄 UDF，讓資料庫知道它
- 撰寫函數碼，以執行函數並傳送適當的參數
- 除錯及測試函數。

如需撰寫函數的範例，請參閱

- 第 223 頁的『範例：UDF 的平方數』
- 第 225 頁的『範例：計數器』

UDF 執行時間環境

有關執行 UDF 的環境及該環境的限制，您必須考慮數項因素。如果您打算爲 UDF 撰寫複雜的函數碼，應仔細考慮這些因素。

因數如下：

- 『UDF 執行的時間長度』
- 第 214 頁的『緒注意事項』
- 第 214 頁的『平行處理』

UDF 執行的時間長度

UDF 是從 SQL 陳述式執行中呼叫，通常是一種查詢作業且會以潛在方式對表格中的數千列執行。因爲此原因，必須從低層次的資料庫呼叫 UDF。

因爲是從這類低層次中呼叫，因此在呼叫 UDF 時及執行 UDF 期間內，將保留某些資源 (鎖定及扣留)。這些資源主要是鎖定在呼叫 UDF 的 SQL 陳述式中所含的任何表格與索引上。由於這些保留資源，因此 UDF 不得執行任何可能會延長期限 (分鐘或小時) 的作業。由於長期保留資源的重要性質，資料庫只能等候某段時間使 UDF 完成。如果於配置的時間內尚未完成 UDF，則 SQL 陳述式將失敗，而這會讓一般使用者非常生氣。

資料庫使用的預設 UDF 等待時間應多於足以讓正常 UDF 執行完成的時間。然而，如果您有一個要長時間執行的 UDF 且希望增加等待時間，則可以利用查詢 INI 檔中的 UDF_TIME_OUT 選項來完成。如需 INI 檔案的相關明細，請參閱 *Database Performance and Query Optimization* 資訊中的 Query Options File QAQQINI。然而，請注意：不論 UDF_TIME_OUT 的指定值爲何，資料庫不能超出最大時間限制。

因爲執行 UDF 時會保留資源，所以 UDF 不得在爲原始 SQL 陳述式配置的相同表格或索引上運作，若確爲如此，則不可執行與 SQL 陳述式中執行的作業相衝突之作業。特別是，UDF 不應在那些表格上嘗試執行任何插入、更新或刪除列作業。

緒注意事項

定義爲 FENCED 的 UDF 會執行於與呼叫它之 SQL 陳述式的相同工作中。然而，UDF 執行於系統緒中，不同於執行 SQL 陳述式的緒。如需緒的相關資訊，請參閱「資訊中心」中「程式設計」種類的多緒程式設計的資料庫注意事項。

因爲 UDF 會在與 SQL 陳述式相同的工作中執行，它會與 SQL 陳述式共用幾乎相同的環境。然而，因爲它是在個別緒下執行，適用下列緒注意事項：

- UDF 會與 SQL 陳述式的緒所保留的緒層次資源相衝突。基本上，這些都是前面討論過的表格資源。
- UDF 未繼承在呼叫 SQL 陳述式時可能已作用的任何程式採用權限。UDF 權限是來自與 UDF 程式本身相關的權限，或執行 SQL 陳述式的使用者權限。
- UDF 無法執行任何在次要緒中受到阻礙而無法執行的作業。
- 必須建立 UDF 程式，使它能夠在已命名的啓動群組下或在其呼叫程式 (ACTGRP 參數) 的啓動群組中執行。不容許指定 ACTGRP(*NEW) 的程式當成 UDF 執行。

如需定義函數爲 UNFENCED 的相關資訊，請參閱第 223 頁的『製作隔離或非隔離的函數』。

平行處理

可定義 UDF 以容許平行處理。這表示可以同時在多個緒上執行相同的 UDF 程式。因此，如果爲 UDF 指定 ALLOW PARALLEL，請確定它是安全緒。如需緒的相關資訊，請參閱 iSeries 資訊中心中「程式設計」種類的多緒程式設計的資料庫注意事項。

使用者定義的表格函數不能平行執行；因此，在建立函數時，必須指定 DISALLOW PARALLEL

撰寫函數碼

撰寫函數碼牽涉到知道如何撰寫 SQL 或外部函數，以執行函數。它也涉及瞭解資料庫與函數碼間的介面以正確地定義之，並在建立可執行程式時決定包裝選項。

您可以撰寫 UDF 作爲 SQL 函數或作爲外部函數。如需相關明細，請參閱

- 『撰寫 UDF 作爲 SQL 函數』
- 第 215 頁的『撰寫 UDF 作爲外部函數』

另請參閱第 222 頁的『表格函數注意事項』。

撰寫 UDF 作為 SQL 函數

SQL 函數是您使用 CREATE FUNCTION 陳述式定義、撰寫及登錄的 UDF。它們本身是只使用 SQL 語言撰寫的，且其定義是完全內含於一個 (可能是大型的) CREATE FUNCTION 陳述式中。建立 SQL 函數會造成登錄 UDF、產生函數的可執行碼、以及對資料庫定義如何實際傳送參數的明細。因此，撰寫這些函數是相當安全的，較少有機會在函數中造成錯誤。

純量 UDF

SQL 純量函數的 CREATE FUNCTION 陳述式遵循一般流程：

```
CREATE FUNCTION function-name(parameters) RETURNS return-value
LANGUAGE SQL
BEGIN
   sql-statements
END
```

例如，依據日期傳回優先順序的函數：

```
CREATE FUNCTION PRIORITY(indate DATE) RETURNS CHAR(7)
LANGUAGE SQL
BEGIN
RETURN(
      CASE   WHEN indate>CURRENT DATE-3 DAYS THEN 'HIGH'
         WHEN indate>CURRENT DATE-7 DAYS THEN 'MEDIUM'
         ELSE 'LOW'
      END
      );
END
```

然後會呼叫函數作爲：

```
SELECT ORDERNBR, PRIORITY(ORDERDUEDATE) FROM ORDERS
```

表格 UDF

SQL 表格函數的 CREATE FUNCTION 陳述式遵循一般流程：

```
CREATE FUNCTION function-name(parameters) RETURNS TABLE return-columns
LANGUAGE SQL
BEGIN
   sql-statements
RETURN
   select-statement
END
```

例如，依據日期傳回資料的函數：

```
CREATE FUNCTION PROJFUNC(indate DATE)
  RETURNS TABLE (PROJNO CHAR(6), ACTNO SMALLINT, ACTSTAFF DECIMAL(5,2),
           ACSTDATE DATE, ACENDATE DATE)
  LANGUAGE SQL
  BEGIN
   RETURN SELECT * FROM PROJACT
     WHERE ACSTDATE<=indate;
  END
```

然後會呼叫函數作爲：

```
SELECT * FROM TABLE(PROJFUNC(:datehv)) X
```

SQL 表格函數需要有一個 (且是唯一的一個) RETURN 陳述式。

撰寫 UDF 作為外部函數

您也可以使用 SQL 以外的語言撰寫 UDF 的可執行碼。雖然此方法比 SQL 函數稍爲麻煩些，但它提供彈性，讓您可以使用對您而言最有效率的語言。可執行碼可以內含於程式或服務程式。

也可以使用 Java 撰寫外部函數。如需參數的說明，請參閱 IBM Developer Kit for Java 主題中的 Java SQL Routines。

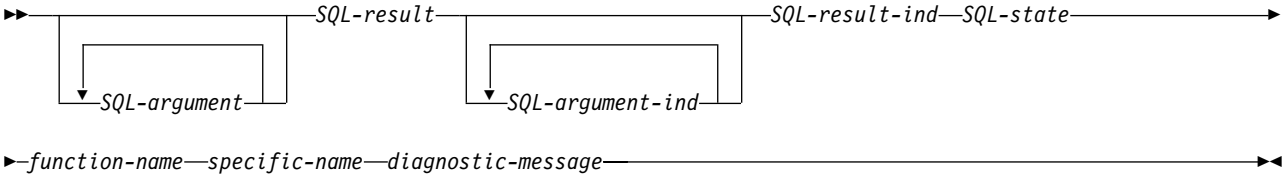
從 DB2 傳送引數到外部函數

DB2 提供儲存體以儲存傳送到 UDF 的所有參數。因此，可以利用位址將參數傳送到外部函數。這是程式的正常參數傳送方法。若爲服務程式，請確定參數已正確地定義於函數碼中。

在定義及使用 UDF 中的參數時，應小心謹慎，以確保針對給定參數所參照的儲存體不會比爲該參數所定義的多。參數全部儲存在相同的空間中，且超出給定參數的儲存體空間會改寫另一個參數的值。相反的，這會造成函數查看無效的輸入資料，或造成傳回資料庫的值無效。

外部 UDF 可以使用數種支援的參數樣式。就大部份的樣式而言，樣式會依傳送到外部程式或服務程式的參數數目而有所區別。

參數樣式 SQL: 參數樣式 SQL 符合工業標準「結構化查詢語言 (SQL)」。此參數樣式只能與純量 UDF 一起使用。利用參數樣式 SQL，參數會傳送到外部程式，如下所示 (依照指定的次序)：



SQL-argument

此引數是在呼叫 UDF 之前由 DB2 設定的。此值會重複 *n* 次，其中 *n* 是函數參照中指定的引數數目。每一個引數的值都是採自函數呼叫中指定的表示式。在 CREATE FUNCTION 陳述式中，已定義參數的資料類型即會表示該值。註：這些參數只會被視爲輸入；DB2 會忽略 UDF 對參數值所做的變更。

SQL-result

此引數是在傳回到 DB2 前，由 UDF 設定的。資料庫提供儲存體，以儲存回覆值。因爲參數是以位址傳送，位址是指應放置回覆值的儲存體位址。資料庫會依照 CREATE FUNCTION 陳述式中的定義，提供回覆值所需的所有儲存體。如果在 CREATE FUNCTION 陳述式中使用 CAST FROM 子句，DB2 會假設 UDF 依照 CAST FROM 子句中的定義傳回值，否則，DB2 會假設 UDF 依照 RETURNS 子句中的定義傳回值。

SQL-argument-ind

此引數是在呼叫 UDF 之前，由 DB2 設定的。UDF 可以使用它來判定相對應的 *SQL-argument* 是否爲 NULL。第 *n* 個 *SQL-argument-ind* 對應於第 *n* 個 *SQL-argument*，如之前所述。每一個指示符均定義爲 2 位元組帶正負號的的整數。它設定爲下列其中一個值：

0 出現引數且不是 NULL。

-1 引數是 NULL。

如果使用 RETURNS NULL ON NULL INPUT 定義函數，則 UDF 不需要檢查是否有 NULL 值。然而，如果是使用 CALLS ON NULL INPUT 定義函數，則任何引數都可以是 NULL，且 UDF 應檢查是否有 NULL 輸入。註：這些參數只會被視爲輸入；DB2 會忽略 UDF 對參數值所做的變更。

SQL-result-ind

此引數是在傳回到 DB2 前，由 UDF 設定的。資料庫提供儲存體，以儲存回覆值。引數定義爲 2 位元組帶正負號的整數。如果設定爲負值，則資料庫會將函數結果解譯爲 NULL。如果設定爲 0 或正值，資料庫會使用 *SQL-result* 中傳回的值。資料庫提供儲存體，以儲存回覆值指示符。因爲參數是以位址傳送，位址是指應放置指示符值的儲存體位址。

SQL-state

此引數是 CHAR(5) 值，代表 SQLSTATE。

此參數由設定爲 '00000' 的資料庫傳送，且可以由函數設定爲函數的結果狀態。然而，通常 SQLSTATE 不是由函數設定，它可以用來對資料庫發出錯誤或警告信號，如下所示：

01Hxx 函數碼偵測到警告狀況。這會造成 SQL 警告，此處的 *xx* 可以是數種可能字串之一。

38xxx 函數碼偵測到錯誤狀況。它會造成 SQL 錯誤。此處的 *xxx* 可以是數種可能字串之一。

如需函數可以使用的有效 SQLSTATE 的相關資訊，請參閱SQL 訊息與訊息碼。

function-name

此引數是在呼叫 UDF 之前，由 DB2 設定的。它是一個 VARCHAR(139) 值，其中含有呼叫的函數碼所代表的函數名稱。

傳送的函數名稱格式爲：

```
<schema-name>.<function-name>
```

當多個 UDF 定義使用函數碼時，此參數是非常有用的，這樣函數碼即可區分正在呼叫哪一個定義。註：此參數只會被視爲輸入；DB2 會忽略 UDF 對參數值所做的變更。

specific-name

此引數是在呼叫 UDF 之前，由 DB2 設定的。它是一個 VARCHAR(128) 值，其中含有呼叫的函數碼所代表的函數特定名稱。

就像函數名稱一樣，當多個 UDF 定義使用函數碼時，此參數是非常有用的，這樣函數碼即可區分正在呼叫哪一個定義。如需 *specific-name* 的相關資訊，請參閱 CREATE FUNCTION 陳述式。註：此參數只會被視爲輸入；DB2 會忽略 UDF 對參數值所做的變更。

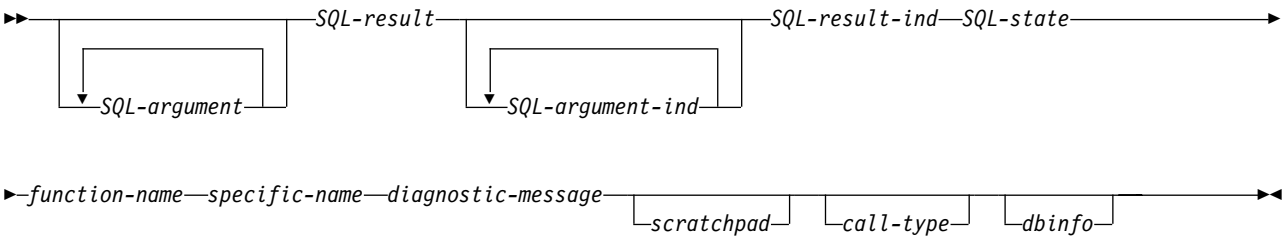
diagnostic-message

此引數是在呼叫 UDF 之前，由 DB2 設定的。它是一個 VARCHAR(70) 值，當 UDF 發出 SQLSTATE 警告或錯誤信號時，UDF 可以用它來傳回訊息文字。

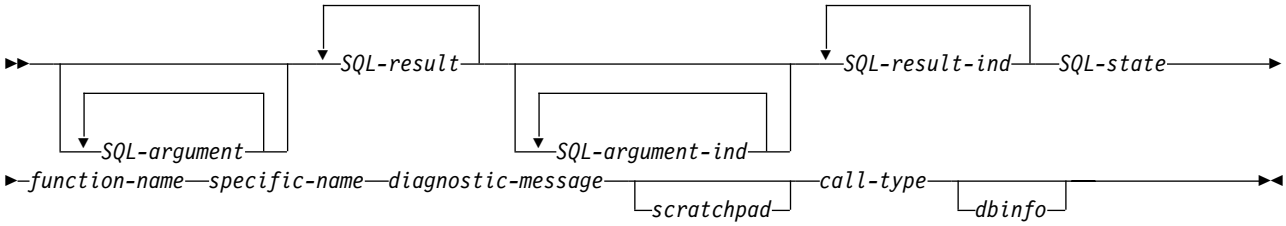
它是由資料庫在輸入至 UDF 時起始設定，且可以由 UDF 以敘述資訊加以設定。DB2 會忽略訊息文字，除非 UDF 已設定 SQL-state 參數。

參數樣式 DB2SQL: 使用 DB2SQL 參數樣式，相同的參數及相同的參數次序會被傳送至外部程式或服務程式，就像參數樣式 SQL 的傳送一樣。然而，DB2SQL 容許一起傳送附加的選用參數。如果在 UDF 定義中指定一個以上的選用參數 (如下所示)，則參數會依照下面定義的次序傳送至 UDF。請參照參數樣式 SQL，以取得一般參數。此參數樣式可以用於純量及表格 UDF。

若爲純量函數：


SQL-result
SQL-argument
SQL-result-ind
SQL-state
SQL-argument-ind
function-name
specific-name
diagnostic-message
scratchpad
call-type
dbinfo

若爲表格函數：



scratchpad

此引數是在第一次呼叫 UDF 之前，由 DB2 設定的。只有在 UDF 的 CREATE FUNCTION 陳述式指定 SCRATCHPAD 關鍵字時，它才會出現。此引數的結構包含下列元素：

- INTEGER，內含 scratchpad 的長度。
- 實際的 scratchpad，在呼叫 UDF 之前，已由 DB2 起始設定爲全部二進位 0'。

UDF 可以使用 scratchpad 作爲工作儲存區或作爲持久性儲存體，因爲它可以跨 UDF 呼叫維護。

若爲表格函數，如果在 CREATE FUNCTION 中指定了 FINAL CALL，則在 UDF 的 FIRST 呼叫之前，即會起始設定 scratchpad (如前面所述)。在此呼叫之後，表格函數即可完全控制 scratchpad 內容。在這之後，DB2 不會檢查或變更 scratchpad 的內容。 scratchpad 會在每一次呼叫中傳送至函數。函數可以是重入，且 DB2 會將它的狀態資訊保留在 scratchpad。

如果表格函數指定或預設 NO FINAL CALL，則會在每一次 OPEN 呼叫時，起始設定 scratchpad (如前面所述)，且 OPEN 呼叫間的表格函數會完全控制 scratchpad 內容。對於結合或子查詢中使用的表格函數而言，這是非常重要的。如果有必要跨 OPEN 呼叫來維護 scratchpad 的內容，則必須在您的 CREATE FUNCTION 陳述式中指定 FINAL CALL。使用指定的 FINAL CALL，除了正常的 OPEN、FETCH 及 CLOSE 呼叫外，基於 scratchpad 維護及資源釋放的目的，表格函數也會接收 FIRST 及 FINAL 呼叫。

call-type

此引數是在呼叫 UDF 之前，由 DB2 設定的。若爲純量函數，只有在 UDF 的 CREATE FUNCTION 陳述式指定 FINAL CALL 關鍵字時，它才會出現。然而，若爲表格函數，它是一定會出現的。它接在 *scratchpad* 引數之後；或如果 scratchpad 引數不存在，則會接在 *diagnostic-message* 引數之後。此引數會採用 INTEGER 值的格式。

若爲純量函數：

-1 這是*首次呼叫*此陳述式的 UDF。第一次呼叫是*正常呼叫*，在此呼叫中會傳送所有 SQL 引數值。

0 這是*正常呼叫*。(傳送所有正常輸入引數值)。

1 這是*最終呼叫*。不會傳送任何 *SQL-argument* 或 *SQL-argument-ind* 值。UDF 不應使用 SQL-result、SQL-result-ind 引數、SQL-state 或 diagnostic-message 引數傳回任何回答。一旦 UDF 傳回這些引數，即會被 DB2 忽略。

若爲表格函數：

-2 這是*首次呼叫*此陳述式的 UDF。首次呼叫是*正常呼叫*，在此呼叫中會傳送所有 SQL 引數值。

-1 這是*開放呼叫*此陳述式的 UDF。如果已指定 NO FINAL CALL，則會起始設定 scratchpad，否則就不需要。傳送所有 SQL 引數值。

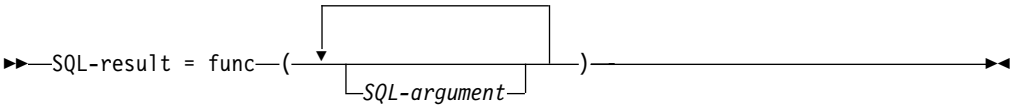
0 這是*提取呼叫*。DB2 預期表格函數會傳回含有一組回覆值的列，或 SQLSTATE 值 '02000' 所指示的表格結束狀況。

1 這是*關閉呼叫*。此呼叫可平衡 OPEN 呼叫，且可以用來執行任何外部 CLOSE 處理程序及資源釋放。

2 這是*最終呼叫*。不會傳送任何 *SQL-argument* 或 *SQL-argument-ind* 值。UDF 不應使用 SQL-result、SQL-result-ind 引數、SQL-state 或 diagnostic-message 引數傳回任何回答。一旦 UDF 傳回這些引數，即會被 DB2 忽略。

dbinfo 此引數是在呼叫 UDF 之前，由 DB2 設定的。只有在 UDF 的 CREATE FUNCTION 陳述式指定 DBINFO 關鍵字時，它才會出現。引數是一種結構，其定義內含於 `sqludf` include。

參數樣式 GENERAL (或 SIMPLE CALL): 使用參數樣式 GENERAL，參數會傳送到外部服務程式，就像在 CREATE FUNCTION 陳述式中的指定一樣。此參數樣式只能與純量 UDF 一起使用。格式如下：



SQL-argument

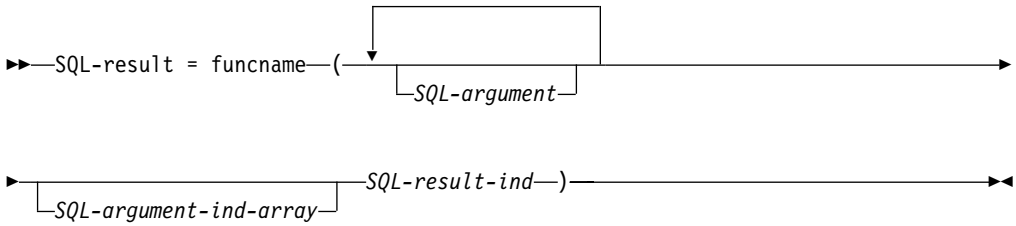
此引數是在呼叫 UDF 之前，由 DB2 設定的。此值會重複 *n* 次，其中 *n* 爲函數參照中指定的引數數目。每一個引數的值都是採自函數呼叫中指定的表示式。在 CREATE FUNCTION 陳述式中，已定義參數的資料類型即會表示該值。註：這些參數僅被視爲輸入；DB2 會忽略 UDF 對參數值所做的變更。

SQL-result

此值是由 UDF 傳回。DB2 會將值複製到資料庫儲存體。爲了正確地傳回值，函數碼必須是值傳回的函數。資料庫只會依照 CREATE FUNCTION 陳述式上指定回覆值的定義來複製值。如果在 CREATE FUNCTION 陳述式中使用 CAST FROM 子句， DB2 會假設 UDF 依照 CAST FROM 子句中的定義傳回值，否則，DB2 會假設 UDF 依照 RETURNS 子句中的定義傳回值。

因爲基本要求是函數碼必須是值傳回的函數，所以必須在服務程式中建立用於參數樣式 GENERAL 的所有函數碼。

參數樣式 GENERAL WITH NULLS: 參數樣式 GENERAL WITH NULLS 只能與純量 UDF 一起使用。使用此參數樣式，參數會傳送到服務程式，如下所示 (依照指定的次序)：



SQL-argument

此引數是在呼叫 UDF 之前，由 DB2 設定的。此值會重複 *n* 次，其中 *n* 是函數參照中指定的引數數目。每一個引數的值都是採自函數呼叫中指定的表示式。在 CREATE FUNCTION 陳述式中，已定義參數的資料類型即會表示該值。註：這些參數僅被視爲輸入；DB2 會忽略 UDF 對參數值所做的變更。

SQL-argument-ind-array

此引數是在呼叫 UDF 之前，由 DB2 設定的。UDF 可以使用它來判定一或多個 *SQL-arguments* 是否爲 NULL。它是一個由 2 位元組帶正負號的整數 (指示符) 組成的陣列。第 *n* 個陣列引數對應於第 *n* 個 *SQL-argument*。每一個陣列登錄已設定爲下列其中一個值：

0 出現引數且不是 NULL。

-1 引數是 NULL。

UDF 應檢查 NULL 輸入。註：此參數僅被視爲輸入；DB2 會忽略 UDF 對參數值所做的變更。

SQL-result-ind

此引數是在傳回到 DB2 前，由 UDF 設定的。資料庫提供儲存體，以儲存回覆值。引數定義爲 2 位元組帶正負號的整數。如果設定爲負值，則資料庫會將函數結果解譯爲 NULL。如果設定爲 0 或正值，資料庫會使用 *SQL-result* 中傳回的值。資料庫提供儲存體，以儲存回覆值指示符。因爲參數是以位址傳送，位址是指應放置指示符值的儲存體位址。

SQL-result

此值是由 UDF 傳回。DB2 會將值複製到資料庫儲存體。爲了正確地傳回值，函數碼必須是值傳回的函數。資料庫只會依照 CREATE FUNCTION 陳述式上指定回覆值的定義來複製值。如果在 CREATE FUNCTION 陳述式中使用 CAST FROM 子句， DB2 會假設 UDF 依照 CAST FROM 子句中的定義傳回值，否則，DB2 會假設 UDF 依照 RETURNS 子句中的定義傳回值。

因爲基本要求是函數碼必須爲值傳回的函數，所以必須在服務程式中建立用於參數樣式 GENERAL WITH NULLS 的所有函數碼。

註:

1. 可以使用引號或不使用引號來指定 CREATE FUNCTION 陳述式上指定的外部名稱。如果名稱未以引號括住，則在它儲存之前是大寫；如果以引號括住，則會依照指定來儲存。在命名實際程式時，這會變得非常重要，因爲資料庫會搜尋程式，此程式名稱必須完全符合以函數定義儲存的名稱。例如，若函數建立爲：

```
CREATE FUNCTION X(INT) RETURNS INT
  LANGUAGE C
  EXTERNAL NAME 'MYLIB/MYPGM(MYENTRY)'
```

且程式的來源爲：

```
void myentry(
       int*in
       int*out,
       .
       .
       . .
```

資料庫找不到登錄，因爲它是小寫 *myentry*，而系統是指示資料庫去尋找大寫的 *MYENTRY*。

2. 若爲具有 C++ 模組的服務程式，請確定是在 C++ 原始程式碼中繼續處理具有 *extern "C"* 的程式函數定義。否則，C++ 編譯器將執行函數名稱的「名稱識別編碼」，且資料庫將找不到該名稱。

參數樣式 DB2GENERAL: 參數樣式 DB2GENERAL 是由 Java UDF 使用。如需此參數樣式的說明，請參閱 IBM Developer Kit for Java 主題中的 Java SQL routine。

參數樣式 Java: Java 參數樣式是 SQLJ Part 1: SQL Routines 標準指定的樣式。如需此參數樣式的說明，請參閱 IBM Developer Kit for Java 主題中的 Java SQL Routines。

表格函數注意事項

外部表格函數是一個 UDF，可以遞送表格到參照該表格的 SQL 中。表格函數參照只有在 SELECT 的 FROM 子句中有效。在使用表格函數時，請觀察下列各項：

- 即使表格函數遞送表格，在 DB2 及 UDF 之間的實體介面仍是一次一列。可以對表格函數發出五種類型的呼叫： OPEN、FETCH、CLOSE、FIRST 及 FINAL。FIRST 及 FINAL 呼叫的存在取決於您如何定義 UDF。可用於純量函數的相同 *call-type* 機制是用來區別這些呼叫。
- 擴充 DB2 與使用者定義純量函數間所使用的標準介面，以配合表格函數。重複表格函數的 *SQL-result* 引數；會依照 CREATE FUNCTION 陳述式中 RETURNS TABLE 子句的定義，傳回對應於直欄的每一個案例。同樣地，*SQL-result-ind* 引數也會重複，每一個案例會關聯到相對應的 *SQL-result* 案例。
- 不是在 CREATE FUNCTION 陳述式的 RETURNS 子句中定義的每一個表格函數結果直欄都必須傳回。 CREATE FUNCTION 的 DBINFO 關鍵字及相對應的 *dbinfo* 引數會啓用最佳化，只需要傳回特定表格函數參照所需的那些直欄。
- 傳回的個別欄位值格式符合純量函數傳回的值。
- 如果表格函數的 CREATE FUNCTION 陳述式有 CARDINALITY *n* 規格。此規格可以讓定義者通知 DB2 最佳化工具有關結果的大約大小，使最佳化工具可以在函數被參照時作出更好的決策。不論指定爲表格函數之 CARDINALITY 的值爲何，在撰寫具有無限列數的函數時，請小心注意；亦即，函數一定會在 FETCH 呼叫中傳回列。DB2 預期表格結束狀況，就像是查詢處理程序中的觸媒一樣。所以從未傳回表格結束狀況 (SQL-state 值 '02000') 的表格函數將會造成無限處理迴圈。

表格函數錯誤處理程序

表格函數呼叫的錯誤處理程序模型如下：

1. 如果 FIRST 呼叫失敗，不會有進一步的呼叫。
2. 如果 FIRST 呼叫成功，則會進行巢狀 OPEN、FETCH 及 CLOSE 呼叫，且一定會進行 FINAL 呼叫。
3. 如果 OPEN 呼叫失敗，則不會進行 FETCH 或 CLOSE 呼叫。

4. 如果 OPEN 呼叫成功，則會進行 FETCH 及 CLOSE 呼叫。
5. 如果 FETCH 呼叫失敗，不會進行進一步的 FETCH 呼叫，但會進行 CLOSE 呼叫。

註: 此模型說明表格 UDF 的正常錯誤處理程序。在系統失敗或通信問題的事件中，可能不會執行錯誤處理程序模型所指示的呼叫。

純量函數錯誤處理程序

以 FINAL CALL 規格定義的純量 UDF 之錯誤處理程序模型如下：

1. 如果 FIRST 呼叫失敗，不會有進一步的呼叫。
2. 如果 FIRST 呼叫成功，則會依照陳述式處理程序的保證進行進一步的 NORMAL 呼叫，且一定會進行 FINAL 呼叫。
3. 如果 NORMAL 呼叫失敗，不會進行進一步的 NORMAL 呼叫，但會進行 FINAL 呼叫 (如果您已指定 FINAL CALL)。這表示如果在 FIRST 呼叫中傳回錯誤，必須在傳回前先清除 UDF，因爲不會進行任何 FINAL 呼叫。

註: 此模型說明純量 UDF 的正常錯誤處理程序。在系統失敗或通信問題的事件中，可能不會執行錯誤處理程序模型所指示的呼叫。

製作隔離或非隔離的函數

在建立「使用者定義函數 (UDF)」時，請考慮是否要讓 UDF 成爲「非隔離的 UDF」。根據預設值，UDF 會建立爲「隔離的 UDF」。隔離表示資料庫應在個別緒中執行 UDF。若爲複雜的 UDF，此分隔是很重要的，因爲它會避免潛在的問題，如產生唯一的 SQL 游標名稱。不需要擔心資源衝突，是使用預設值及建立 UDF 爲隔離的 UDF 的原因。使用 'NOT FENCED' 選項所建立的 UDF 會向資料庫表示，使用者要求能在起始 UDF 的相同緒中執行 UDF。非隔離爲對資料庫的建議，但資料庫仍可決定是否要以相同的方式，將 UDF 當成「隔離的 UDF」來執行。

```
CREATE FUNCTION QGPL.FENCED (parameter1 INTEGER)
RETURNS INTEGER LANGUAGE SQL
BEGIN
RETURN parameter1 * 3;
END;

CREATE FUNCTION QGPL.UNFENCED1 (parameter1 INTEGER)
RETURNS INTEGER LANGUAGE SQL NOT FENCED
-- 使用 NOT FENCED 選項建置 UDF 以要求執行速度更快
BEGIN
RETURN parameter1 * 3;
END;
```

UDF 程式碼的範例

這些範例會顯示如何使用 SQL 函數及外部函數來執行 UDF 程式碼：

- 『範例：UDF 的平方數』
- 第 225 頁的『範例：計數器』
- 第 226 頁的『範例：天氣表格函數』

範例：UDF 的平方數

註: 如需專屬於程式碼範例的相關資訊，請參閱第 x 頁的『程式碼不保事項聲明』資訊。

假設您想要一個傳回平方數的函數。查詢陳述式是：

```
SELECT SQUARE(myint) FROM mytable
```

下列範例會顯示如何定義 UDF 的數種不同方法。

- **使用 SQL 函數**

```
CREATE FUNCTION SQUARE( inval INT) RETURNS INT
LANGUAGE SQL
BEGIN
RETURN(inval*inval);
END
```

- **使用外部函數、參數樣式 SQL：**

  CREATE FUNCTION 陳述式：

```
CREATE FUNCTION SQUARE(INT) RETURNS INT CAST FROM FLOAT
LANGUAGE C
EXTERNAL NAME 'MYLIB/MATH(SQUARE)'
DETERMINISTIC
NO SQL
NO EXTERNAL ACTION
PARAMETER STYLE SQL
ALLOW PARALLEL
```

  程式碼：

```
void SQUARE(int *inval,
double *outval,
short *inind,
short *outind,
char *sqlstate,
char *funcname,
char *specname,
char *msgtext)
{
if (*inind<0)
  *outind=-1;
else
  {
   *outval=*inval;
   *outval=(*outval)*(*outval);
   *outind=0;
  }
return;
}
```

  若要建立外部服務程式，以便能進行除錯：

```
CRTCMOD MODULE(mylib/square) DBGVIEW(*SOURCE)
CRTSRVPGM SRVPGM(mylib/math) MODULE(mylib/square)
EXPORT(*ALL) ACTGRP(*CALLER)
```

- **使用外部函數、參數樣式 GENERAL：**

  CREATE FUNCTION 陳述式：

```
CREATE FUNCTION SQUARE(INT) RETURNS INT CAST FROM FLOAT
LANGUAGE C
EXTERNAL NAME 'MYLIB/MATH(SQUARE)'
DETERMINISTIC
NO SQL
NO EXTERNAL ACTION
PARAMETER STYLE GENERAL
ALLOW PARALLEL
```

  程式碼：

```
double SQUARE(int *inval)
{
  double outval;
  outval=*inval;
  outval=outval*outval;
return(outval);
}
```

若要建立外部服務程式，以便能進行除錯：

```
CRTCMOD MODULE(mylib/square) DBGVIEW(*SOURCE)

   CRTSRVPGM SRVPGM(mylib/math) MODULE(mylib/square)
EXPORT(*ALL) ACTGRP(*CALLER)
```

範例：計數器

註: 如需專屬於程式碼範例的相關資訊，請參閱第 x 頁的『程式碼不保事項聲明』資訊。

假設您只是想要計算 SELECT 陳述式中的列數。您可以撰寫一個會增量並傳回計數器的 UDF。此範例使用具有 DB2 SQL 參數樣式的外部函數及 scratchpad。

```
CREATE FUNCTION COUNTER()
       RETURNS INT
       SCRATCHPAD
       NOT DETERMINISTIC
       NO SQL
       NO EXTERNAL ACTION
       LANGUAGE C
       PARAMETER STYLE DB2SQL
       EXTERNAL NAME 'MYLIB/MATH(ctr)'
       DISALLOW PARALLEL

/* structure scr defines the passed scratchpad for the function "ctr" */
struct scr {
   long len;
   long countr;
   char not_used[96];
};

void ctr (
   long *out,                          /* output answer (counter) */
   short *outnull,                     /* output NULL indicator */
   char *sqlstate,                     /* SQL STATE */
   char *funcname,                     /* function name */
   char *specname,                     /* specific function name */
   char *mesgtext,                     /* message text insert */
   struct scr *scratchptr) {           /* scratch pad */

   *out = ++scratchptr->countr;      /* increment counter & copy out */
   *outnull = 0;
   return;
}
/* end of UDF : ctr */
```

針對此 UDF，請觀察：

- 它沒有定義任何輸入 SQL 引數，但傳回值。
- 它在四個標準尾隨引數 (亦即 *SQL-state*、*function-name*、*specific-name* 及 *message-text*) 後，附加了 scratchpad 輸入引數。
- 其包含結構定義以對映傳送的 scratchpad。
- 未定義任何輸入參數。這與程式碼一致。
- 已撰寫 SCRATCHPAD，造成 DB2 配置、適當地起始設定及傳送 scratchpad 引數。
- 您已將它指定爲 NOT DETERMINISTIC，因爲它不只是取決於 SQL 輸入引數 (在此情況下是不依據任何輸入引數)。

- 您已正確地指定 DISALLOW PARALLEL，因爲 UDF 的更正函數取決於單一 scratchpad。

範例：天氣表格函數

註: 如需專屬於程式碼範例的相關資訊，請參閱第 x 頁的『程式碼不保事項聲明』資訊。

下面是範例表格函數，它會傳回美國各城市的天氣資訊。這些城市的天氣日期是讀取自外部檔，如範例程式中的說明所指示。資料包含城市名稱，其後接著當地天氣資訊。對其它城市也重複此型樣。

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <sqludf.h> /* for use in compiling User Defined Function */


#define   SQL_NOTNULL   0   /* Nulls Allowed - Value is not Null */
#define   SQL_ISNULL   -1   /* Nulls Allowed - Value is Null */
#define   SQL_TYP_VARCHAR 448
#define   SQL_TYP_INTEGER 496
#define   SQL_TYP_FLOAT 480

/* Short and long city name structure */
typedef struct {
  char * city_short ;
  char * city_long ;
} city_area ;

/* Scratchpad data */ 1
/* Preserve information from one function call to the next call */
typedef struct {
  /* FILE * file_ptr; if you use weather data text file */
  int file_pos ;  /* if you use a weather data buffer */
} scratch_area ;

/* Field descriptor structure */
typedef struct {
  char fld_field[31] ;                    /* Field data */
  int  fld_ind ;            /* Field null indicator data */
  int  fld_type ;                          /* Field type */
  int  fld_length ;  /* Field length in the weather data */
  int  fld_offset ;  /* Field offset in the weather data */
} fld_desc ;

/* Short and long city name data */
city_area cities[] = {
  { "alb", "Albany, NY"                 },
  { "atl", "Atlanta, GA"                },
  .
  .
  .
  { "wbc", "Washington DC, DC"          },
  /* You may want to add more cities here */

  /* Do not forget a null termination */
  { ( char * ) 0, ( char * ) 0          }
} ;

/* Field descriptor data */
fld_desc fields[] = {
  { "", SQL_ISNULL, SQL_TYP_VARCHAR, 30,  0 }, /* city         */
```

```
  { "", SQL_ISNULL, SQL_TYP_INTEGER,  3,  2 }, /* temp_in_f     */
  { "", SQL_ISNULL, SQL_TYP_INTEGER,  3,  7 }, /* humidity      */
  { "", SQL_ISNULL, SQL_TYP_VARCHAR,  5, 13 }, /* wind          */
  { "", SQL_ISNULL, SQL_TYP_INTEGER,  3, 19 }, /* wind_velocity */
  { "", SQL_ISNULL, SQL_TYP_FLOAT,    5, 24 }, /* barometer     */
  { "", SQL_ISNULL, SQL_TYP_VARCHAR, 25, 30 }, /* forecast      */
  /* You may want to add more fields here */

  /* Do not forget a null termination */
  { ( char ) 0, 0, 0, 0, 0 }
} ;

/* Following is the weather data buffer for this example. You */
/* may want to keep the weather data in a separate text file. */
/* Uncomment the following fopen() statement.  Note that you  */
/* have to specify the full path name for this file.          */
char * weather_data[] = {
   "alb.forecast",
   "   34   28%    wnw   3  30.53 clear",
   "atl.forecast",
   "   46   89%   east  11  30.03 fog",
   .
   .
   .
   "wbc.forecast",
   "   38   96%    ene  16  30.31 light rain",
   /* You may want to add more weather data here */

   /* Do not forget a null termination */
   ( char * ) 0
} ;

#ifdef __cplusplus
extern "C"
#endif
/* This is a subroutine. */
/* Find a full city name using a short name */
int get_name( char * short_name, char * long_name ) {

    int name_pos = 0 ;

    while ( cities[name_pos].city_short != ( char * ) 0 ) {
       if (strcmp(short_name, cities[name_pos].city_short) == 0) {
          strcpy( long_name, cities[name_pos].city_long ) ;
          /* A full city name found */
          return( 0 ) ;
       }
       name_pos++ ;
    }
    /* Could not find such city in the city data */
    strcpy( long_name, "Unknown City" ) ;
    return( -1 ) ;

}

#ifdef __cplusplus
extern "C"
#endif
/* This is a subroutine. */
/* Clean all field data and field null indicator data */
int clean_fields( int field_pos ) {

    while (fields[field_pos].fld_length !=0 ) {
       memset( fields[field_pos].fld_field, '\0', 31 ) ;
       fields[field_pos].fld_ind = SQL_ISNULL ;
       field_pos++ ;
    }
```

```
    return( 0 ) ;

}

#ifdef __cplusplus
extern "C"
#endif
/* This is a subroutine. */
/* Fills all field data and field null indicator data ... */
/* ... from text weather data */
int get_value( char * value, int field_pos ) {

    fld_desc * field ;
    char field_buf[31] ;
    double * double_ptr ;
    int * int_ptr, buf_pos ;

    while ( fields[field_pos].fld_length != 0 ) {
       field = &fields[field_pos] ;
       memset( field_buf, '\0', 31 ) ;
       memcpy( field_buf,
               ( value + field->fld_offset ),
               field->fld_length ) ;
       buf_pos = field->fld_length ;
       while ( ( buf_pos > 0 ) &&
               ( field_buf[buf_pos] == ' ' ) )
          field_buf[buf_pos--] = '\0' ;
       buf_pos = 0 ;
       while ( ( buf_pos < field->fld_length ) &&
               ( field_buf[buf_pos] == ' ' ) )
          buf_pos++ ;
       if ( strlen( ( char * ) ( field_buf + buf_pos ) ) > 0 ||
            strcmp( ( char * ) ( field_buf + buf_pos ), "n/a") != 0 ) {
          field->fld_ind = SQL_NOTNULL ;

          /* Text to SQL type conversion */
          switch( field->fld_type ) {
            case SQL_TYP_VARCHAR:
                 strcpy( field->fld_field,
                         ( char * ) ( field_buf + buf_pos ) ) ;
                 break ;
            case SQL_TYP_INTEGER:
                 int_ptr = ( int * ) field->fld_field ;
                 *int_ptr = atoi( ( char * ) ( field_buf + buf_pos ) ) ;
                 break ;
            case SQL_TYP_FLOAT:
                 double_ptr = ( double * ) field->fld_field ;
                 *double_ptr = atof( ( char * ) ( field_buf + buf_pos ) ) ;
                 break ;
            /* You may want to add more text to SQL type conversion here */
          }

       }
       field_pos++ ;
    }
    return( 0 ) ;

}

#ifdef __cplusplus
extern "C"
#endif
void SQL_API_FN weather( /* Return row fields */
              SQLUDF_VARCHAR * city,
              SQLUDF_INTEGER * temp_in_f,
              SQLUDF_INTEGER * humidity,
              SQLUDF_VARCHAR * wind,
```

```
                    SQLUDF_INTEGER * wind_velocity,
                    SQLUDF_DOUBLE  * barometer,
                    SQLUDF_VARCHAR * forecast,
                    /* You may want to add more fields here */

                    /* Return row field null indicators */
                    SQLUDF_NULLIND * city_ind,
                    SQLUDF_NULLIND * temp_in_f_ind,
                    SQLUDF_NULLIND * humidity_ind,
                    SQLUDF_NULLIND * wind_ind,
                    SQLUDF_NULLIND * wind_velocity_ind,
                    SQLUDF_NULLIND * barometer_ind,
                    SQLUDF_NULLIND * forecast_ind,
                    /* You may want to add more field indicators here */

                    /* UDF always-present (trailing) input arguments */
                    SQLUDF_TRAIL_ARGS_ALL
                  ) {

scratch_area * save_area ;
char line_buf[81] ;
int line_buf_pos ;

/* SQLUDF_SCRAT is part of SQLUDF_TRAIL_ARGS_ALL */
/* Preserve information from one function call to the next call */
save_area = ( scratch_area * ) ( SQLUDF_SCRAT->data ) ;

/* SQLUDF_CALLT is part of SQLUDF_TRAIL_ARGS_ALL */
switch( SQLUDF_CALLT ) {

  /* First call UDF: Open table and fetch first row */
  case SQL_TF_OPEN:
       /* If you use a weather data text file specify full path */
       /* save_area->file_ptr = fopen("tblsrv.dat","r"); */
       save_area->file_pos = 0 ;
       break ;

  /* Normal call UDF: Fetch next row */ 2
  case SQL_TF_FETCH:
       /* If you use a weather data text file */
       /* memset(line_buf, '\0', 81); */
       /* if (fgets(line_buf, 80, save_area->file_ptr) == NULL) { */
       if ( weather_data[save_area->file_pos] == ( char * ) 0 ) {

          /* SQLUDF_STATE is part of SQLUDF_TRAIL_ARGS_ALL */
          strcpy( SQLUDF_STATE, "02000" ) ;

          break ;
       }
       memset( line_buf, '\0', 81 ) ;
       strcpy( line_buf, weather_data[save_area->file_pos] ) ;
       line_buf[3] = '\0' ;

       /* Clean all field data and field null indicator data */
       clean_fields( 0 ) ;

       /* Fills city field null indicator data */
       fields[0].fld_ind = SQL_NOTNULL ;

       /* Find a full city name using a short name */
       /* Fills city field data */
       if ( get_name( line_buf, fields[0].fld_field ) == 0 ) {
          save_area->file_pos++ ;
          /* If you use a weather data text file */
          /* memset(line_buf, '\0', 81); */
          /* if (fgets(line_buf, 80, save_area->file_ptr) == NULL) { */
          if ( weather_data[save_area->file_pos] == ( char * ) 0 ) {
```

```
            /* SQLUDF_STATE is part of SQLUDF_TRAIL_ARGS_ALL */
            strcpy( SQLUDF_STATE, "02000" ) ;
            break ;
         }
         memset( line_buf, '\0', 81 ) ;
         strcpy( line_buf, weather_data[save_area->file_pos] ) ;
         line_buf_pos = strlen( line_buf ) ;
         while ( line_buf_pos > 0 ) {
            if ( line_buf[line_buf_pos] >= ' ' )
               line_buf_pos = 0 ;
            else {
               line_buf[line_buf_pos] = '\0' ;
               line_buf_pos-- ;
            }
         }
      }

      /* Fills field data and field null indicator data ... */
      /* ... for selected city from text weather data */
      get_value( line_buf, 1 ) ;  /* Skips city field */

      /* Builds return row fields */
      strcpy( city, fields[0].fld_field ) ;
      memcpy( (void *) temp_in_f,
              fields[1].fld_field,
              sizeof( SQLUDF_INTEGER ) ) ;
      memcpy( (void *) humidity,
              fields[2].fld_field,
              sizeof( SQLUDF_INTEGER ) ) ;
      strcpy( wind, fields[3].fld_field ) ;
      memcpy( (void *) wind_velocity,
              fields[4].fld_field,
              sizeof( SQLUDF_INTEGER ) ) ;
      memcpy( (void *) barometer,
              fields[5].fld_field,
              sizeof( SQLUDF_DOUBLE ) ) ;
      strcpy( forecast, fields[6].fld_field ) ;

      /* Builds return row field null indicators */
      memcpy( (void *) city_ind,
              &(fields[0].fld_ind),
              sizeof( SQLUDF_NULLIND ) ) ;
      memcpy( (void *) temp_in_f_ind,
              &(fields[1].fld_ind),
              sizeof( SQLUDF_NULLIND ) ) ;
      memcpy( (void *) humidity_ind,
              &(fields[2].fld_ind),
              sizeof( SQLUDF_NULLIND ) ) ;
      memcpy( (void *) wind_ind,
              &(fields[3].fld_ind),
              sizeof( SQLUDF_NULLIND ) ) ;
      memcpy( (void *) wind_velocity_ind,
              &(fields[4].fld_ind),
              sizeof( SQLUDF_NULLIND ) ) ;
      memcpy( (void *) barometer_ind,
              &(fields[5].fld_ind),
              sizeof( SQLUDF_NULLIND ) ) ;
      memcpy( (void *) forecast_ind,
              &(fields[6].fld_ind),
              sizeof( SQLUDF_NULLIND ) ) ;

      /* Next city weather data */
      save_area->file_pos++ ;

      break ;

   /* Special last call UDF for cleanup (no real args!): Close table */ 3
```

```
    case SQL_TF_CLOSE:
         /* If you use a weather data text file */
         /* fclose(save_area->file_ptr); */
         /* save_area->file_ptr = NULL; */
         save_area->file_pos = 0 ;
         break ;

  }

}
```

參照此 UDF 程式碼中的內含數字，觀察：

1. 已定義 scratchpad。row 變數已起始設定於 OPEN 呼叫中，且 iptr 陣列及 nbr_rows 變數已在開啓時由 *mystery* 函數填寫。
2. FETCH 會使用列作爲索引以遍訪 iptr 陣列，並將有興趣的值從 iptr 的現行元素移到 out_c1、out_c2 及 out_c3 結果值指標所指向的位置。
3. 最後，CLOSE 會釋放由 OPEN 所取得且固定在 scratchpad 中的儲存體。

下列是此 UDF 的 CREATE FUNCTION 陳述式：

```
CREATE FUNCTION tfweather_u()
  RETURNS TABLE (CITY VARCHAR(25),
                 TEMP_IN_F INTEGER,
                 HUMIDITY INTEGER,
                 WIND VARCHAR(5),
                 WIND_VELOCITY INTEGER,
                 BAROMETER FLOAT,
                 FORECAST VARCHAR(25))
  SPECIFIC tfweather_u
  DISALLOW PARALLEL
  NOT FENCED
  DETERMINISTIC
  NO SQL
  NO EXTERNAL ACTION
  SCRATCHPAD
  NO FINAL CALL
  LANGUAGE C
  PARAMETER STYLE DB2SQL
  EXTERNAL NAME 'LIB1/WEATHER(weather)';
```

參照此陳述式，請觀察：

- 它未採用任何輸入，且傳回 7 個輸出直欄。
- 已指定 SCRATCHPAD，所以 DB2 會配置、適當地起始設定並傳送 scratchpad 引數。
- 已指定 NO FINAL CALL。
- 函數已指定爲 NOT DETERMINISTIC，因爲它不只是取決於 SQL 輸入引數。亦即，它取決於 mystery 函數，且我們假設內容會隨著每一次的執行而改變。
- 表格函數需要 DISALLOW PARALLEL。
- CARDINALITY 100 爲預估會傳回的預期列數，且此值會提供給 DB2 最佳化工具。
- 未使用 DBINFO，且未執行最佳化只傳回參照此函數之特定陳述式所需的直欄。
- 已指定 NOT NULL CALL，所以若有任何輸入 SQL 引數爲 NULL，就不會呼叫 UDF，且不必檢查此條件。

若要選取此表格函數所產生的全部橫列，請使用下列查詢：

```
SELECT *
    FROM TABLE (tfweather_u())x
```

第 14 章 動態 SQL 應用程式

動態 SQL 可讓應用程式在程式執行時間定義和執行 SQL 陳述式。爲動態 SQL 提供的應用程式接受字串形式的 SQL 陳述式作爲輸入(或建置)。應用程式不必知道將要執行的 SQL 陳述式類型。應用程式：

- 建置或接受 SQL 陳述式爲輸入
- 準備要執行的 SQL 陳述式
- 執行陳述式
- 處理 SQL 回覆碼

交談式 SQL (說明於第 17 章, 『使用交談式 SQL』) 是動態 SQL 程式的一個例子。由交談式 SQL 來動態地處理和執行 SQL 陳述式。

註:

1. 動態 SQL 處理陳述式的執行時期之額外執行時間比靜態 SQL 陳述式更大。額外的處理類似於前置編譯、連結及執行一個程式，而不只是執行而已。因此，只有需要動態 SQL 彈性的應用程式才有此需要。其他應用程式應該使用正常 (靜態) SQL 陳述式來存取資料庫的資料。
2. 含有 EXECUTE 或 EXECUTE IMMEDIATE 陳述式及使用 FOR READ ONLY 子句來產生唯讀游標的程式，效能會較好，因爲它們使用區塊化來擷取游標的列。

   對於未明確編寫 FOR UPDATE OF 或已經定位參照該游標的 deletes 或 updates 的所有游標， ALWBLK(*ALLREAD) CRTSQLxxx 選項會暗示 FOR READ ONLY 宣告。具暗示 FOR READ ONLY 的游標將受惠於此清單的第二個項目。

部份動態 SQL 陳述式需要使用位址變數。RPG for iSeries 程式需要借助 PL/I、COBOL、C 或 ILE RPG for iSeries 程式來管理位址變數。

下列表格顯示 DB2 UDB for iSeries 支援的全部陳述式，並指出是否可用於動態應用程式中。

註: 下表中，*動態 SQL* 直欄的數字對應到下一頁的附註。

表 28. 動態應用程式中容許的 SQL 陳述式清單

SQL 陳述式	靜態 SQL	動態 SQL
ALTER TABLE	Y	Y
BEGIN DECLARE SECTION	Y	N
CALL	Y	Y
CLOSE	Y	N
COMMENT ON	Y	Y
COMMIT	Y	Y
CONNECT	Y	N
CREATE ALIAS	Y	Y
CREATE DISTINCT TYPE	Y	Y
CREATE FUNCTION	Y	Y

表 28. 動態應用程式中容許的 SQL 陳述式清單 (繼續)

SQL 陳述式	靜態 SQL	動態 SQL
CREATE INDEX	Y	Y
CREATE PROCEDURE	Y	Y
CREATE SCHEMA	Y	Y
CREATE TABLE	Y	Y
CREATE TRIGGER	Y	Y
CREATE VIEW	Y	Y
DECLARE CURSOR	Y	請參閱附註 1。
DECLARE GLOBAL TEMPORARY TABLE	Y	Y
DECLARE PROCEDURE	Y	N
DECLARE STATEMENT	Y	N
DECLARE VARIABLE	Y	N
DELETE	Y	Y
DESCRIBE	Y	請參閱附註 2。
DESCRIBE TABLE	Y	N
DISCONNECT	Y	N
DROP	Y	Y
END DECLARE SECTION	Y	N
EXECUTE	Y	請參閱附註 3。
EXECUTE IMMEDIATE	Y	請參閱附註 4。
FETCH	Y	N
FREE LOCATOR	Y	Y
GRANT	Y	Y
HOLD LOCATOR	Y	Y
INCLUDE	Y	N
INSERT	Y	Y
LABEL ON	Y	Y
LOCK TABLE	Y	Y
OPEN	Y	N
PREPARE	Y	請參閱附註 5。
RELEASE	Y	N
RELEASE SAVEPOINT	Y	Y
RENAME	Y	Y
REVOKE	Y	Y
ROLLBACK	Y	Y
SAVEPOINT	Y	Y
SELECT INTO	Y	請參閱附註 6。
SELECT 陳述式	Y	請參閱附註 7。
SET CONNECTION	Y	N
SET OPTION	Y	請參閱附註 8。
SET PATH	Y	Y

表 28. 動態應用程式中容許的 SQL 陳述式清單 (繼續)

SQL 陳述式	靜態 SQL	動態 SQL
SET RESULT SETS	Y	N
SET SCHEMA	Y	Y
SET TRANSACTION	Y	Y
SET 變數	Y	N
UPDATE	Y	Y
VALUES INTO	Y	N
WHENEVER	Y	N

註:

1. 無法準備，但可在執行之前用來定義相關動態 SELECT 陳述式的游標。
2. 無法準備，但可用來傳回準備的陳述式說明。
3. 無法準備，但可用來執行準備的 SQL 陳述式。在使用 EXECUTE 陳述式之前，必須先以 PREPARE 陳述式來準備 SQL 陳述式。請參閱第 236 頁的『使用 PREPARE 和 EXECUTE 陳述式』下的 PREPARE 範例。
4. 無法準備，但可搭配動態陳述式字串使用，該字串不含任何 ? 參數標記。EXECUTE IMMEDIATE 陳述式可在程式執行時間動態地準備和執行陳述式字串。請參閱『處理非 SELECT 陳述式』下的 EXECUTE IMMEDIATE 範例。
5. 無法準備，但可用於在執行之前剖析、最佳化及設定動態 SELECT 陳述式。請參閱『處理非 SELECT 陳述式』下的 PREPARE 範例。
6. 無法準備 SELECT INTO 陳述式，或使用於 EXECUTE IMMEDIATE 中。
7. 無法搭配 EXECUTE 或 EXECUTE IMMEDIATE 使用，但可以 OPEN 來準備和執行。
8. 只有當執行 REXX 程序或在經過前置編譯的程式中時才可以使用。

註: 請參閱第 x 頁的『程式碼不保事項聲明』資訊，以取得關於程式碼範例的資訊。

設計和執行動態 SQL 應用程式

若要發出動態 SQL 陳述式，您必須搭配 EXECUTE 陳述式或 EXECUTE IMMEDIATE 陳述式來使用此陳述式，因爲動態 SQL 陳述式不是在前置編譯時準備的，因此必須在執行時間準備。 EXECUTE IMMEDIATE 陳述式可在程式執行時間動態地準備和執行 SQL 陳述式。

有兩種基本的動態 SQL 陳述式類型：SELECT 陳述式和非 SELECT 陳述式。非 SELECT 陳述式包括 DELETE、INSERT 及 UPDATE 等陳述式。

使用諸如 ODBC 介面的主從應用程式使用動態 SQL 來存取資料庫。關於開發 iSeries Access 主從應用程式的詳細資訊，請參閱 Programming for iSeries Access Express。

處理非 SELECT 陳述式

若要建置動態 SQL 非 SELECT 陳述式：

1. 驗證您要建置的 SQL 陳述式是可以動態執行的陳述式 (請參閱第 233 頁的表 28)。

2. 建置 SQL 陳述式。(使用「交談式 SQL」這種簡單的方式來建置、驗證及執行 SQL 陳述式。詳細資訊請參閱第 17 章, 『使用交談式 SQL』)。

若要執行動態 SQL 非 SELECT 陳述式：

1. 使用 EXECUTE IMMEDIATE 來執行 SQL 陳述式，或使用 PREPARE 來準備 SQL 陳述式，再使用 EXECUTE 來執行已準備的陳述式。
2. 處理任何可能傳回的 SQL 回覆碼。

以下是執行動態 SQL 非 SELECT 陳述式的一個應用程式範例 (stmtstrg)：

```
EXEC SQL
 EXECUTE IMMEDIATE :stmtstrg;
```

動態 SQL 陳述式的 CCSID

此 SQL 陳述式通常是一個主變數。主變數的 CCSID 用來做爲陳述式文字的 CCSID。在 PL/I 中，此陳述式也可以是字串表示式。在本例中，工作 CCSID 用來當做陳述式文字的 CCSID。

動態 SQL 陳述式是使用陳述式文字的 CCSID 來處理的。這對變體字元影響最大。例如，not 符號 (¬) 位於 CCSID 500 中的 'BA'X。這表示如果陳述式文字的 CCSID 是 500，則 SQL 會預期 not 符號 (¬) 位於 'BA'X。

如果陳述式文字 CCSID 是 65535，則 SQL 會將變體字元當做具有 CCSID 37 來處理。這表示 SQL 會在 '5F'X 上尋找 not 符號 (¬)。

使用 PREPARE 和 EXECUTE 陳述式

如果非 SELECT 陳述式不包含參數標記，則可以使用 EXECUTE IMMEDIATE 陳述式來動態執行。不過，如果非 SELECT 陳述式含有參數標記，則必須使用 PREPARE 和 EXECUTE 來執行。

PREPARE 陳述式可準備非 SELECT 陳述式(例如，DELETE 陳述式)，並且提供一個名稱讓您選擇。如果在 CRTSQLxxx 指令上指定 DLYPRP (*YES)，除非在 PREPARE 陳述式中指定 USING 子句，否則會延遲到 EXECUTE 或 DESCRIBE 陳述式中第一次使用此陳述式時才會準備。在本例中，我們稱它爲 S1。在準備好陳述式後，即可在參數標記上使用不同的值，在相同程式中執行許多次。下列範例是一個執行多次的已備妥陳述式：

```
DSTRING = 'DELETE FROM CORPDATA.EMPLOYEE WHERE EMPNO = ?';

/*The ? is a parameter marker which denotes
  that this value is a host variable that is
  to be substituted each time the statement is run.*/

EXEC SQL PREPARE S1 FROM :DSTRING;

/*DSTRING is the delete statement that the PREPARE statement is
  naming S1.*/

 DO UNTIL (EMP =0);
/*The application program reads a value for EMP from the
  display station.*/
   EXEC SQL
     EXECUTE S1 USING :EMP;


END;
```

在類似上述範例的常式中，您必須知道參數標記的數目及其資料類型，因爲用來提供輸入資料的主變數是在撰寫程式時就宣告。

註: 每當對於應用程式伺服器的連線終止時，便會銷毀與應用程式伺服器相關的所有準備的陳述式。終止連線是以一個 CONNECT (Type 1) 陳述式、一個 DISCONNECT 陳述式、或一個 RELEASE 再接一個成功的 COMMIT 來完成的。

處理 SELECT 陳述式和使用 SQLDA

有兩種基本的 SELECT 陳述式類型：**固定清單**和**變動清單**。

處理固定清單 SELECT 陳述式並不需要 SQLDA。

若要處理變動清單 SELECT 陳述式，首先您必須宣告一個 SQLDA 結構。SQLDA 是一個控制區塊，用以將主變數輸入值從應用程式傳送到 SQL，以及從 SQL 接收輸出值。此外，關於 SELECT 清單表示式的資訊可在 PREPARE 或 DESCRIBE 陳述式中傳回。

固定清單 SELECT 陳述式

在動態 SQL 中，固定清單 SELECT 陳述式是用來擷取可預期數目和類型的資料。使用這些陳述式時，您可以預期和定義主變數來接受擷取的資料，所以不需要 SQLDA。每一個接續的 FETCH 會傳回與上一個 FETCH 相同數目的值，這些值的資料格式與上一個 FETCH 傳回的值相同。您可以按照在任何 SQL 應用程式中的相同方式來指定主變數。

您可以在任何 SQL 支援的應用程式中使用固定清單動態 SELECT 陳述式。

若要動態執行固定清單 SELECT 陳述式，您的應用程式必須：

1. 將輸入 SQL 陳述式放入主變數中。
2. 發出 PREPARE 陳述式來驗證動態 SQL 陳述式，並做成可以執行的形式。如果在 CRTSQLxxx 指令上指定 DLYPRP (*YES)，除非在 PREPARE 陳述式中指定 USING 子句，否則會延遲到 EXECUTE 或 DESCRIBE 陳述式中第一次使用此陳述式時才會準備。
3. 爲陳述式名稱宣告一個游標。
4. 開啓游標。
5. FETCH 一列到一個固定的變數清單中(並不是像您使用變動清單 SELECT 陳述式時放到記述子區域中，請參閱下一節變動清單 Select 陳述式)。
6. 到了資料結尾時，請關閉游標。
7. 處理任何傳回的 SQL 回覆碼。

例如：

```
MOVE 'SELECT EMPNO, LASTNAME FROM CORPDATA.EMPLOYEE WHERE EMPNO>?'
TO DSTRING.
EXEC SQL
 PREPARE S2 FROM :DSTRING END-EXEC.

EXEC SQL
 DECLARE C2 CURSOR FOR S2 END-EXEC.

EXEC SQL
 OPEN C2 USING :EMP END-EXEC.
```

```
PERFORM FETCH-ROW UNTIL SQLCODE NOT=0.

EXEC SQL
 CLOSE C2 END-EXEC.
STOP-RUN.
FETCH-ROW.
EXEC SQL
 FETCH C2 INTO :EMP, :EMPNAME END-EXEC.
```

註: 請記得，因爲本例中的 SELECT 陳述式總是傳回與先前執行的固定清單 SELECT 陳述式相同的資料項目的數目和類型，所以您不必使用 SQL 記述子區域 (SQLDA)。

變動清單 Select 陳述式

在動態 SQL 中，變動清單 SELECT 陳述式傳回的結果直欄的數目和格式是無法預期的；亦即，您不知道需要多少變數，或是什麼資料類型。因此，您無法事前定義主變數來接受傳回的結果直欄。

註: 在 REXX 中，步驟 5.b、6 及 7 不適用。

如果應用程式接受變動清單 SELECT 陳述式，則您的程式必須：

1. 將輸入 SQL 陳述式放入主變數中。
2. 發出 PREPARE 陳述式來驗證動態 SQL 陳述式，並做成可以執行的形式。如果在 CRTSQLxxx 指令上指定 DLYPRP (*YES)，除非在 PREPARE 陳述式中指定 USING 子句，否則會延遲到 EXECUTE 或 DESCRIBE 陳述式中第一次使用此陳述式時才會準備。
3. 爲陳述式名稱宣告一個游標。
4. 開啓含有動態 SELECT 陳述式名稱的游標 (宣告於步驟 3 中)。
5. 發出 DESCRIBE 陳述式，向 SQL 要求有關結果表格中每一個直欄的類型和大小的資訊。

   註:

   a. 您亦可撰寫含 INTO 子句的 PREPARE 陳述式，以單一陳述式來執行 PREPARE 和 DESCRIBE 的功能。

   b. 如果 SQLDA 的大小不足以包含每一個擷取直欄的直欄說明，則程式必須決定需要多少空間、取得該空間大小的儲存體、建置新的 SQLDA，並重新發出 DESCRIBE 陳述式。
6. 配置包含一列擷取的資料所需的儲存體數量。
7. 將儲存體位址置於 SQLDA（SQL 記述子區域），以告知 SQL 將每一個擷取的資料項目置於何處。
8. FETCH 一列。
9. 到了資料結尾時，請關閉游標。
10. 處理任何可能傳回的 SQL 回覆碼。

請參閱第 243 頁的『範例：爲 SQLDA 配置儲存體的 Select 陳述式』，以取得如何執行這些步驟的詳細資訊。

SQL 記述子區域 (SQLDA)

動態 SQL 使用一個稱爲 SQL 記述子區域 (SQLDA) 的變數結構，以傳遞 SQL 和您的應用程式之間有關 SQL 陳述式的資訊。需要 SQLDA 來執行 DESCRIBE 和 DESCRIBE TABLE 陳述式，且亦可用在 PREPARE、OPEN、FETCH、CALL 及 EXECUTE 陳述式。

SQLDA 中資訊的意義視其用途而定。在 PREPARE 和 DESCRIBE 中，SQLDA 提供關於準備的陳述式的資訊給應用程式。在 DESCRIBE TABLE 中，SQLDA 提供關於表格或概略表中直欄的資訊給應用程式。在 OPEN、EXECUTE、CALL 及 FETCH 中，SQLDA 提供關於主變數的資訊。例如，您可以使用 DESCRIBE 陳述式在 SQLDA 中讀入一些值，將值變更爲主變數的位址，然後在 FETCH 陳述式中重覆使用這些值。

如果應用程式可讓您同時開啓數個游標，則您可以撰寫數個 SQLDA，每一個動態 SELECT 陳述式各使用一個。詳細資訊，請參閱 SQL Reference 書中的 SQLDA 和 SQLCA。

SQLDA 可使用於 C、C++、COBOL、PL/I、REXX 及 RPG。因爲 RPG for iSeries 未提供設定指標的方法，所以必須以 PL/I、C、C++、COBOL 或 ILE RPG for iSeries 程式在 RPG for iSeries 程式的外部設定 SQLDA。然後該程式必須呼叫 RPG for iSeries 程式。

SQLDA 格式

SQLDA 由 4 個變數組成，變數後面接著一個任意數字，此數字代表一連串稱爲 SQLVAR 的 6 個變數出現的次數。

註: REXX 中的 SQLDA 有所不同。詳細資訊，請參閱 *SQL Programming with Host Languages*中的 Coding SQL Statements in REXX Applications。

在 OPEN、FETCH、CALL 及 EXECUTE 中使用 SQLDA 時，每一個 SQLVAR 都說明了一個主變數。

SQLDA 的欄位如下：

SQLDAID

SQLDAID 用來當做傾出儲存體的「醒目位元組」。它是一個 8 個字元的字串，在 PREPARE 或 DESCRIBE 陳述式中使用 SQLDA 之後，它具有 'SQLDA' 這個值。此變數不使用於 FETCH、OPEN、CALL 或 EXECUTE。

位元組 7 可用來決定每一個直欄是否需要一個以上的 SQLVAR 登錄。如果有任何 LOB 或特殊類型直欄，則可能需要多重 SQLVAR 登錄。如果沒有任何 LOB 或特殊類型，則此旗號設爲空白。

SQLDAID 不適用於 REXX。

SQLDABC

SQLDABC 指出 SQLDA 的長度。這是一個 4 位元組的整數，在 PREPARE 或 DESCRIBE 陳述式中使用 SQLDA 之後，具有 SQLN*LENGTH(SQLVAR) + 16 的值。在 FETCH、OPEN、CALL 或 EXECUTE 使用 SQLDABC 之前，SQLDABC 的值必須等於或大於 SQLN*LENGTH(SQLVAR) + 16。

SQLABC 不適用於 REXX。

SQLN SQLN 是一個 2 位元組整數，指定 SQLVAR 的總出現次數。在被任何 SQL 陳述式使用之前，必須設定成一個大於或等於 0 的值。

SQLN 不適用於 REXX。

SQLD SQLD 是一個 2 位元組整數，指定 SQLVAR 的出現次數，換言之，指 SQLDA 所說明的主變數或直欄的數目。此欄位由 SQL 設定於 DESCRIBE 或 PREPARE 陳述式中。在其他陳述式中，此欄位在使用之前必須先設定成大於或等於 0 且小於或等於 SQLN 的值。

SQLVAR

每一個主變數或直欄會重複一次這一組值。這些變數由 SQL 設定於 DESCRIBE 或 PREPARE 陳述式中。在其他陳述式中，必須事先設定才能使用。這些變數定義如下：

SQLTYPE

SQLTYPE 是一個 2 位元組整數，指定主變數或直欄的資料類型，如第 241 頁的表 29所示。SQLTYPE 的奇數值顯示主變數有一個相關的指示器變數，由 SQLIND 來定址。

SQLLEN

SQLLEN 是一個 2 位元組整數變數，指定主變數或直欄的長度屬性。

SQLRES

SQLRES 是一個 12 位元組保留區域，有邊界對齊的用途。請注意，在 OS/400 中，指標*必須*在一個 Quad-word 界限上。

SQLRES 不適用於 REXX。

SQLDATA

SQLDATA 是一個 16 位元組指標變數，當 OPEN、FETCH、CALL 及 EXECUTE 上使用 SQLDA 時，可指定主變數的位址。

在 PREPARE 和 DESCRIBE 上使用 SQLDA 時，下列資訊會覆加在此區域上：

字元或圖形欄位的 CCSID 儲存在 SQLDATA 的第三個和第四個位元組中。以 BIT 資料爲例，CCSID 是 65535。在 REXX，CCSID 傳回於變數 SQLCCSID 中。

SQLIND

SQLIND 是一個 16 位元組指標，指定小整數主變數的位址，在 OPEN、FETCH、CALL 及 EXECUTE 上使用 SQLDA 時，用來當做空值或非空值的指示。負值表示空值，非負值表示非空值。此指標只有在 SQLTYPE 包含奇數值時才使用。

在 PREPARE 和 DESCRIBE 上使用 SQLDA 時，此區域保留供未來使用。

SQLNAME

SQLNAME 是一個可變長度字元變數，最大長度爲 30。在 PREPARE 或 DESCRIBE 後面，此變數包含選取的直欄、標籤的名稱或系統直欄名稱。在 OPEN、FETCH、EXECUTE 或 CALL 中，此變數可用來傳遞字串的 CCSID。字元和圖形主變數的 CCSID 可以傳遞。

可以設定輸入 SQLDA 的 SQLVAR 陣列登錄中的 SQLNAME 欄位來指定 CCSID：

資料類型	子類型	SQLNAME 的長度	SQLNAME 位元組 1 & 2	SQLNAME 位元組 3 & 4
字元	SBCS	8	X'0000'	CCSID
字元	MIXED	8	X'0000'	CCSID
字元	BIT	8	X'0000'	X'FFFF'
GRAPHIC	不適用	8	X'0000'	CCSID
任何其他資料類型	不適用	不適用	不適用	不適用

註: 切記，SQLNAME 欄位只用來置換 CCSID。使用預設值的應用程式不需要傳遞 CCSID 資訊。如果未傳遞 CCSID，則使用工作的預設 CCSID。

圖形主變數的預設值爲工作 CCSID 的相關雙位元組 CCSID。如果相關的雙位元組 CCSID 不存在，則使用 65535。

表 29. PREPARE、DESCRIBE、FETCH、OPEN、CALL 或 EXECUTE 的 SQLTYPE 和 SQLLEN 值

	PREPARE 和 DESCRIBE		FETCH、 OPEN、CALL 及 EXECUTE	
SQLTYPE	COLUMN DATA TYPE	SQLLEN	HOST VARIABLE DATA TYPE	SQLLEN
384/385	Date	10	日期的固定長度字串表示法	主變數的長度屬性
388/389	Time	8	時間的固定長度字串表示法	主變數的長度屬性
392/393	Timestamp	26	時間戳記的固定長度字串表示法	主變數的長度屬性
396/397	DataLink 附註 #1	直欄的長度屬性	N/A	N/A
400/401	N/A	N/A	NUL 結尾的圖形字串	主變數的長度屬性
392/393	Timestamp	26	時間戳記的固定長度字串表示法	主變數的長度屬性
404/405	BLOB	0 (請參閱附註 #2)	BLOB	未使用。(請參閱附註 #2)
408/409	CLOB	0 (請參閱附註 #2)	CLOB	未使用。(請參閱附註 #2)
412/413	DBCLOB	0 (請參閱附註 #2)	DBCLOB	未使用。(請參閱附註 #2)
452/453	固定長度字串	直欄的長度屬性	固定長度字串	主變數的長度屬性
456/457	長可變長度字串	直欄的長度屬性	長可變長度字串	主變數的長度屬性
460/461	N/A	N/A	NUL 結尾的字串	主變數的長度屬性
464/465	可變長度圖形字串	直欄的長度屬性	可變長度圖形字串	主變數的長度屬性
468/469	固定長度圖形字串	直欄的長度屬性	固定長度圖形字串	主變數的長度屬性
472/473	長可變長度圖形字串	直欄的長度屬性	長圖形字串	主變數的長度屬性
476/477	N/A	N/A	PASCAL L 字串	主變數的長度屬性
480/481	浮點	4 代表單一精準度，8 代表倍精準度	浮點	4 代表單一精準度，8 代表倍精準度

表 29. PREPARE、DESCRIBE、FETCH、OPEN、CALL 或 EXECUTE 的 SQLTYPE 和 SQLLEN 值 (繼續)

SQLTYPE	PREPARE 和 DESCRIBE		FETCH、 OPEN、CALL 及 EXECUTE	
	COLUMN DATA TYPE	SQLLEN	HOST VARIABLE DATA TYPE	SQLLEN
484/485	壓縮十進位數	精準度在位元組 1 中；小數位數在位元組 2 中	壓縮十進位數	精準度在位元組 1 中；小數位數在位元組 2 中
488/489	區域十進位數	精準度在位元組 1 中；小數位數在位元組 2 中	區域十進位數	精準度在位元組 1 中；小數位數在位元組 2 中
492/493	大整數	8	大整數	8
496/497	大整數	4 (請參閱附註 #3)	大整數	4
500/501	小整數	2 (請參閱附註 #3)	小整數	2
504/505	N/A	N/A	DISPLAY SIGN LEADING SEPARATE	精準度在位元組 1 中；小數位數在位元組 2 中
904/905	N/A	N/A	ROWID	40
916/917	N/A	N/A	BLOB 檔案參考變數	267
920/921	N/A	N/A	CLOB 檔案參考變數	267
924/925	N/A	N/A	DBCLOB 檔案參考變數	267
960/961	N/A	N/A	BLOB 定位器	4
964/965	N/A	N/A	CLOB 定位器	4
968/969	N/A	N/A	DBCLOB 定位器	4

註:

1. DataLink 資料類型只在 DESCRIBE TABLE 上傳回。
2. 次要 SQLVAR 中的 len.sqllonglen 欄位包含直欄的長度屬性。
3. 大二進位數字和小二進位數字可以在 SQL 記述子區域 (SQLDA) 中表示成長度 2 或 4。亦可以使用位元組 1 的精準度和位元組 2 的小數位數來表示。如果第一個位元組大於 X'00'，則表示精準度和小數位數。大整數數字不容許精準度和小數位數。SQLDA 將它們定義成長度 8。

SQLVAR2

此爲含有 3 個欄位的「延伸 SQLVAR」結構。如果結果包含特殊類型或 LOB 直欄，則結果的所有直欄需要「延伸 SQLVAR」。對於特殊類型，則包含特殊類型名稱。對於 LOB，則包含主變數的長度屬性和一個含有實際長度的緩衝區的指標。如果使用定位器來代表 LOB，則不需要這些登錄。需要「延伸 SQLVAR」的次數是根據用到 SQLDA 的陳述式及所說明的直欄或參數的資料類型而定。 SQLDAID 的位元組 7 一律設定爲必需的 SQLVAR 組的數目。

如果 SQLD 未設定成足夠的 SQLVAR 出現次數：

- SQLD 會設定爲所有 SQLVAR 的出現總數。
- 如果在 Base SQLVAR 登錄上指定剛好足夠的數目，則 SQLCA 的 SQLCODE 欄位中會傳回 +237 警告。傳回 Base SQLVAR 登錄，但不會傳回「延伸 SQLVAR」。

- 如果在 Base SQLVAR 登錄上未指定足夠的 SQLVAR，則 SQLCA 的 SQLCODE 欄位中會傳回 +239 警告。不傳回任何 SQLVAR 登錄。

SQLLONGLEN

SQLLONGLEN 是一個 4 位元組整數變數，指定 LOB (BLOB、CLOB 或 DBCLOB) 主變數或直欄的長度屬性。

SQLDATALEN

SQLDATALEN 是一個 16 位元組指標變數，指定主變數的長度的位址。此變數只使用於 LOB (BLOB、CLOB 及 DBCLOB) 主變數。不適用於 DESCRIBE 或 PREPARE。

如果此欄位是 NULL，則資料的實際長度會儲存在資料開頭的前 4 個位元組中， SQLDATA 會指向欄位長度的第一個位元組。長度指出一個 BLOB 或 CLOB 的位元組數和一個 DBCLOB 的字元數。

如果此欄位不是 NULL，則包含一個指向 4 位元組緩衝區的指標，此緩衝區的內容是相符的基本 SQLVAR 中 SQLDATA 欄位所指向的緩衝區中資料的實際位元組長度 (即使爲 DBCLOB)。

SQLDATATYPE_NAME

SQLDATATYPE 是一個可變長度字元變數，最大長度爲 30。只使用於 DESCRIBE 或 PREPARE。此變數設定爲下列其中一項：

- 若是特殊類型直欄，資料庫管理程式將此變數設定爲完整的特殊類型名稱。如果完整名稱超過 30 個位元組，則會被截斷。
- 若是標籤，則資料庫管理程式會將此變數設定爲標籤的前 20 個位元組。
- 若是直欄名稱，資料庫管理程式會將此變數設定爲直欄名稱。

範例：為 SQLDA 配置儲存體的 Select 陳述式

假設您的應用程式必須能夠處理動態 SELECT 陳述式；即每次使用時都不一樣。此陳述式可從顯示器讀取、從另一個應用程式傳入，或由您的應用程式即時建立。換言之，您完全不知道此陳述式每次會傳回什麼。您的應用程式必須能夠處理事前不知道資料類型的任意數目的結果直欄。

例如，需要處理下列陳述式：

```
SELECT WORKDEPT, PHONENO
  FROM CORPDATA.EMPLOYEE
  WHERE LASTNAME = 'PARKER'
```

註: 此 SELECT 陳述式沒有 INTO 子句。動態 SELECT 陳述式即使只傳回一列，也*不可以*有 INTO 子句。

此陳述式被分派到主變數。然後，本例中的主變數 DSTRING 再經由下列的 PREPARE 陳述式來處理：

```
EXEC SQL
PREPARE S1 FROM :DSTRING;
```

接下來，您需要決定結果直欄的數目及其資料類型。您需要 SQLDA 來配合。

定義 SQLDA 的第一個步驟是配置儲存體。(REXX 中不需要配置儲存體。) 取得儲存體的技術視語言而定。 SQLDA 必須配置在一個 16 位元組界限上。SQLDA 由一個長度爲 16 位元組的固定長度標頭組成。標頭後面接著一個可變長度陣列區段 (SQLVAR)，其中每一個元素的長度是 80 位元組。

您需要配置的儲存體數量視您在 SQLVAR 陣列中需要多少個元素而定。您選取的每一個直欄必須有一個對應的 SQLVAR 陣列元素。因此，SELECT 陳述式中列示的直欄數，決定您應該配置多少 SQLVAR 陣列元素。由於此 SELECT 陳述式是指定於執行時間，所以不可能知道將存取多少個直欄。因此，您必須預估直欄數。在本範例中，假設單一 SELECT 陳述式不會存取 20 個以上的直欄。在此情況下，SQLVAR 陣列應該有一個 20 的維度，確定 SELECT 清單中的每一個項目在 SQLVAR 中有一個對應的登錄。總計 SQLDA 大小爲 20 x 80 或 1600，加上 16，總共是 1616 位元組

在爲您的 SQLDA 配置足夠的預估空間之後，您需要將 SQLDA 的 SQLN 欄位設定成等於 SQLVAR 陣列元素的數目，在本例中是 20。

在配置儲存體和起始設定大小之後，您現在可以發出 DESCRIBE 陳述式。

```
EXEC SQL
DESCRIBE S1 INTO :SQLDA;
```

當執行 DESCRIBE 陳述式時，SQL 會在 SQLDA 中放入值來爲您的陳述式提供有關 SELECT 清單的資訊。下表顯示 DESCRIBE 執行之後的 SQLDA 內容。只顯示在此環境中有意義的登錄。

SQLDA 標頭包含：

表 30. SQLDA 標頭

說明	值
SQLAID	'SQLDA'
SQLDABC	1616
SQLN	20
SQLD	2

SQLDAID 是 DESCRIBE 執行時由 SQL 所起始設定的 ID 欄位。 SQLDABC 是 SQLDA 的位元組計數或大小。SQLDA 標頭後面接著 2 個 SQLVAR 結構，各使用於所說明的 SELECT 陳述式的結果表格中的每一個直欄：

表 31. SQLVAR 元素 1

說明	值
SQLTYPE	453
SQLLEN	3
SQLDATA (3:4)	37
SQLNAME	8 WORKDEPT

表 32. SQLVAR 元素 2

說明	值
SQLTYPE	453
SQLLEN	4

表 32. SQLVAR 元素 2 (繼續)

說明	值
SQLDATA(3:4)	37
SQLNAME	7 PHONENO

如果 SQLDA 的大小不足以包含所說明的 SQLVAR 元素，則您的程式可能需要改變 SQLN 值。例如，假設 SELECT 陳述式實際上傳回 27，而不是預估的最大直欄數 20。SQL 無法說明此 SELECT 清單，因爲 SQLVAR 需要的元素超過配置的空間所容許的數量。取而代之，SQL 會將 SQLD 設定爲 SELECT 陳述式指定的實際直欄數，忽略剩餘的結構。因此，在 DESCRIBE 之後，您應該比較 SQLN 值和 SQLD 值。如果 SQLD 的值大於 SQLN 的值，請根據 SQLD 的值來配置較大的 SQLDA，如下所示，並重新執行 DESCRIBE：

```
EXEC SQL
    DESCRIBE S1 INTO :SQLDA;
IF SQLN <= SQLD THEN
DO;

/*Allocate a larger SQLDA using the value of SQLD.*/
/*Reset SQLN to the larger value.*/

EXEC SQL
    DESCRIBE S1 INTO :SQLDA;
  END;
```

如果您在非 SELECT 陳述式上使用 DESCRIBE，SQL 會將 SQLD 設爲 0。因此，如果您的程式被設計來處理 SELECT 和非 SELECT 陳述式，您可以在每一個陳述式備妥後進行說明，判斷是否爲 SELECT 陳述式。本範例設計爲僅處理 SELECT 陳述式；不檢查 SQLD 值。

您的程式現在必須分析成功的 DESCRIBE 所傳回的 SQLVAR 的元素。SELECT 清單中的第一個項目是 WORKDEPT。在 SQLTYPE 欄位中， DESCRIBE 傳回一個值來指出表示式的資料類型，以及是否可使用空值 (請參閱第 241 頁的表 29)。

在本範例中，SQL 在 SQLVAR 元素 1 中將 SQLTYPE 設爲 453。這指定 WORKDEPT 是固定長度字串結果直欄，且直欄中允許空值。

SQL 將 SQLLEN 設爲直欄的長度。因爲 WORKDEPT 的資料類型是 CHAR，所以 SQL 將 SQLLEN 設爲等於字元直欄的長度。以 WORKDEPT 而言，長度是 3。因此，稍後執行 SELECT 陳述式時，將需要一個足夠保存一個 CHAR(3) 字串的儲存區。

因爲 WORKDEPT 的資料類型是 CHAR FOR SBCS DATA， SQLDATA 的前 4 個位元組已設定爲字元直欄的 CCSID。

SQLVAR 元素的最後一個欄位是可變長度字串，稱爲 SQLNAME。SQLNAME 的前 2 個位元組包含字元資料的長度。字元資料本身通常是 SELECT 陳述式中使用的直欄名稱，在本例中是 WORKDEPT。未命名的 SELECT 清單項目則屬例外，例如函數 (如 SUM(SALARY))、表示式 (如 A+B-C) 及常數。在這些例子中，SQLNAME 是空字串。SQLNAME 亦可包含標籤而不是名稱。與 PREPARE 和 DESCRIBE 陳述式相關的其中一個參數是 USING 子句。您可以使用下列方式來指定：

```
EXEC SQL
    DESCRIBE S1 INTO:SQLDA
        USING LABELS;
```

如果您指定：

NAMES (或完全省略 USING 參數)
SQLNAME 欄位中只會放入直欄名稱。

SYSTEM NAMES
SQLNAME 欄位中只會放入系統直欄名稱。

LABELS
此處只會輸入 SQL 陳述式中列示的直欄的相關標籤。

ANY 只針對有標籤的欄位，才於 SQLNAME 欄位中放入標籤；否則輸入直欄名稱。

BOTH 名稱和標籤及其對應的長度會放入欄位中。請記得將 SQLVAR 陣列的大小加倍，因爲您要併入兩倍的元素數目。

ALL 直欄名稱、標籤及系統直欄名稱加上其對應的長度會放入欄位中。請記得將 SQLVAR 陣列的大小調整爲 3 倍。

關於 USING 選項的詳細資訊，請參閱 *SQL Reference* 書中的 DESCRIBE 陳述式和 SQLDA 這一節。

在本例中，第二個 SQLVAR 元素包含 SELECT 中使用的第二個直欄的資訊：PHONENO。SQLTYPE 中的 453 碼指定 PHONENO 是 CHAR 直欄。SQLLEN 設爲 4。

現在，您需要設定當執行 SELECT 陳述式時使用 SQLDA 去擷取值。

在分析 DESCRIBE 的結果之後，您可以配置變數的儲存體來包含 SELECT 陳述式的結果。對於 WORKDEPT，必須配置長度 3 的字元欄位；對於 PHONENO，必須配置長度 4 的字元欄位。因爲這兩個結果可以爲 NULL 值，所以也必須爲每一個欄位配置一個指示符變數。

在配置儲存體之後，您必須設定 SQLDATA 和 SQLIND 來指向已配置的儲存區。對於 SQLVAR 陣列的每一個元素，SQLDATA 會指向要放入結果值的位置。 SQLIND 指向要放入空值指示符值的位置。下列表格顯示目前的結構外貌。只顯示在此環境中有意義的登錄：

表 33. SQLDA 標頭

說明	值
SQLAID	'SQLDA'
SQLDABC	1616
SQLN	20
SQLD	2

表 34. SQLVAR 元素 1

說明	值
SQLTYPE	453
SQLLEN	3
SQLDATA	指向 CHAR(3) 結果區域的指標
SQLIND	指向結果直欄的 2 位元組整數指示符的指標

表 35. SQLVAR 元素 2

說明	值
SQLTYPE	453
SQLLEN	4
SQLDATA	指向 CHAR(4) 結果區域的指標
SQLIND	指向結果直欄的 2 位元組整數指示符的指標

您現在可以開始擷取 SELECT 陳述式結果。動態定義的 SELECT 陳述式不可以有 INTO 陳述式。因此，所有動態定義的 SELECT 陳述式必須使用游標。動態定義的 SELECT 陳述式使用特殊格式的 DECLARE、OPEN 及 FETCH。

範例陳述式的 DECLARE 陳述式：

```
EXEC SQL DECLARE C1 CURSOR FOR S1;
```

如您所見，唯一的差別在於使用已準備的 SELECT 陳述式 (S1) 的名稱，而不使用 SELECT 陳述式本身的名稱。實際擷取結果列的方法如下：

```
EXEC SQL
    OPEN C1;
EXEC SQL
    FETCH C1 USING DESCRIPTOR :SQLDA;
DO WHILE (SQLCODE = 0);
/*Process the results pointed to by SQLDATA*/
EXEC SQL
    FETCH C1 USING DESCRIPTOR :SQLDA;
  END;
EXEC SQL
    CLOSE C1;
```

開啓游標。SELECT 的結果列會利用 FETCH 陳述式一次傳回一列。在 FETCH 陳述式上，沒有輸出主變數的清單。相反地，FETCH 陳述式會告訴 SQL 將結果傳到 SQLDA 所說明的區域。結果會傳到 SQLVAR 元素的 SQLDATA 和 SQLIND 欄位所指向的儲存區。在處理 FETCH 陳述式之後，WORKDEPT 的 SQLDATA 指標會將其被參照值設定爲 'E11'。其對應的指示符值是 0，因爲傳回一個非空值。PHONENO 的 SQLDATA 指標將其被參照值設定爲 '4502'。其對應的指示符值也是 0，因爲傳回一個非空值。

參數標記

在我們使用的範例中，動態執行的 SELECT 陳述式在 WHERE 子句中有一個常數值。在範例中是：

```
    WHERE LASTNAME = 'PARKER'
```

如果您要使用不同的 LASTNAME 值來執行數次相同的 SELECT 陳述式，您可以使用如下的 SQL 陳述式：

```
    SELECT WORKDEPT, PHONENO
     FROM CORPDATA.EMPLOYEE
        WHERE LASTNAME = ?
```

無法預期您的參數時，則在執行時間之前，應用程式將無從得知參數的數目和類型。您可以安排在應用程式執行時才接收此資訊，透過 OPEN 陳述式上的 USING DESCRIPTOR，您可以用特定主變數中的值來取代 SELECT 陳述式的 WHERE 子句中的參數標記。

若要撰寫這種程式，您需要使用 OPEN 陳述式和 USING DESCRIPTOR 子句。此 SQL 陳述式不只用來開啓游標，也可以將每一個參數記號取代爲對應主變數的值。您以此陳述式所指定的記述子名稱必須識別一個 SQLDA，其中包含那些主變數的有效說明。此 SQLDA 與前述的有所不同，它不是用來傳回 SELECT 清單中部份資料項目的資訊。亦即，不是用來當做 DESCRIBE 陳述式的輸出，而是 OPEN 陳述式的輸入。它提供關於主變數的資訊，這些主變數是用來取代 SELECT 陳述式的 WHERE 子句中的參數標記。此資訊是來自應用程式，應用程式必須設計將適當的值放入 SQLDA 的必要欄位中。然後，當 SQL 將參數標記取代成主變數資料時，SQLDA 就可以用來當做資訊的來源。

當您在含有 USING DESCRIPTOR 子句的 OPEN 陳述式中使用 SQLDA 當做輸入時，不一定要填入所有的欄位。尤其是 SQLDAID、 SQLRES 及 SQLNAME 三者，可以保持空白 (如果需要特定的 CCSID，則可以設定 SQLNAME)。因此，當您使用此方法將參數標記置換成主變數值時，您需要決定：

- 有多少參數標記
- 這些參數標記 (SQLTYPE、SQLLEN 及 SQLNAME) 的資料類型和屬性
- 是否要指示器變數

此外，如果常式要處理 SELECT 和非 SELECT 陳述式，則您可能要決定陳述式的種類。(另外，您可以撰寫程式碼來尋找 SELECT 關鍵字。)

如果應用程式使用參數標記，則您的程式必須：

1. 將陳述式讀入 DSTRING 可變長度字串主變數中。
2. 決定參數標記的數目。
3. 配置此大小的 SQLDA。

   這不適用於 REXX。
4. 將 SQLN 和 SQLD 設定成 ? 參數標記的數目。

   SQLN 不適用於 REXX。
5. 設定 SQLDABC 等於 SQLN*LENGTH(SQLVAR) + 16。

   這不適用於 REXX。
6. 對於每一個參數記號：
   a. 決定資料類型、長度及指示器。
   b. 設定 SQLTYPE 和 SQLLEN。
   c. 配置儲存體來保留輸入值 (? 值)。
   d. 設定這些值。
   e. 爲每一個參數記號設定 SQLDATA 和 SQLIND (如果適當的話)。
   f. 如果使用字元變數，且是 CCSID 而不是工作預設 CCSID，請適當地設定 SQLNAME (REXX 中是 SQLCCSID)。
   g. 如果使用圖形變數，且其 CCSID 不是工作 CCSID 相關的 DBCS CCSID，請將 SQLNAME (REXX 中是 SQLCCSID) 設定爲該 CCSID。
   h. 發出含有 USING DESCRIPTOR 子句的 OPEN 陳述式來開啓您的游標，並且將每一個參數標記置換成主變數。

然後就可以正常處理陳述式。

第 15 章 透過從屬站介面使用動態 SQL

您可以透過伺服器上的從屬站介面來存取 DB2 UDB for iSeries 資料。下列主題協助您開始進行必要的作業：

- 『以 Java 存取資料』
- 『以 Domino 存取資料』
- 『以 ODBC 存取資料』
- 『以「可攜式應用程式解決方案環境 (PASE)」存取資料』

以 Java 存取資料

您可以以 Developer Kit for Java Database Connectivity (JDBC) 驅動程式來存取 Java 程式中的 DB2 UDB for iSeries 資料。此驅動程式可讓您執行下列作業。

- 存取資料庫檔案
- 以內嵌 Structured Query Language (SQL) for Java 存取 JDBC 資料庫功能
- 執行 SQL 陳述式並處理結果。

有關如何使用 JDBC 驅動程式的詳細資訊，請參閱 iSeries 資訊中心中的主題 「設定使用 IBM Developer Kit for Java JDBC 驅動程式」。

以 Domino 存取資料

Domino for iSeries 是一種 Domino 伺服器產品，可讓您在 DB2 UDB for iSeries 資料庫和 Domino 資料庫之間雙向整合資料。若要使用這個整合功能，您必須瞭解如何管理兩資料庫類型之間的授權工作。詳細資訊，請參閱 iSeries 資訊中心的 Domino for iSeries 種類。

以 ODBC 存取資料

您可以使用 iSeries Access for Windows ODBC 驅動程式來讓您的 ODBC 從屬站應用程式彼此有效地共用資料，以及與伺服器共用資料。請參閱 iSeries 資訊中心的 iSeries Access for Windows 種類中的「 ODBC 管理」。

您可以在 iSeries ODBC Driver for Linux 主題中找到使用 Linux 連接的相關資訊。此主題討論如何在 iSeries 邏輯分割區上安裝 Linux，以及如何使用 iSeries ODBC Driver for Linux 來存取 iSeries 資料庫。

以「可攜式應用程式解決方案環境 (PASE)」存取資料

「可攜式應用程式解決方案環境 (PASE)」是執行於 iSeries 系統上之 AIX (或其它 UNIX 型) 應用程式的一種整合的執行時間環境。詳細資訊，請參閱 iSeries 資訊中心的整合作業環境種類中的 「OS/400 PASE」。

第 16 章 使用 iSeries 領航員的進階資料庫功能

本章涵蓋資料庫中可以使用 iSeries 領航員來執行的一些進階功能。提供了概觀資訊和秘訣。 第 31 頁的第 3 章, 『iSeries 領航員資料庫入門』包含部份基本功能。主題如下：

- 『使用「資料庫領航員」來對映資料庫』
- 第 254 頁的『使用「執行 SQL Script」來查詢資料庫』
- 第 256 頁的『使用「產生 SQL」來重建 SQL 陳述式』
- 第 257 頁的『使用 iSeries 領航員的進階表格功能』
- 第 261 頁的『使用 iSeries 領航員來定義 SQL 物件』
- 第 263 頁的『建立 SQL 資料包』

關於您在 iSeries 領航員可執行的其他非基於 SQL 的功能，請參閱以下在 Database Programming 中的主題：

- 顯示表格、概略表及索引說明
- 重組表格
- 顯示已鎖定列

效能相關的作業，請參閱 Database Performance and Query Optimization。

使用「資料庫領航員」來對映資料庫

「資料庫領航員」可讓您以視覺化的方式描述系統上資料庫物件的關係。您爲資料庫所建立的視覺化描述稱爲「資料庫領航員對映」。「資料庫領航員對映」在本質上是資料庫及對映的所有物件之間關係的一個瞬像。

使用「資料庫領航員」，您可以利用圖形表示方式來探勘資料庫物件的複雜關係，圖形表示方式可代表資料庫中的表格、表格之間的關係，以及加諸於表格的索引和限制。連接到系統後，您可以使用「資料庫領航員」：

- 建立「資料庫領航員」對映
- 新增新的物件到對映中
- 變更物件以併入於對映中
- 建立使用者定義關係

「資料庫領航員」的主要工作區是一個分割爲數個主區域的視窗。這些區域可讓您尋找要併入對映中的物件、顯示和隱藏對映中的項目、檢視對映以及檢查在對映中處於擱置中的變更狀態。下列說明「資料庫領航員」視窗的主要區域。

定位器窗格

「資料庫領航員」視窗左邊的「定位器窗格」是用來尋找您要併入新對映中的物件，或尋找已開啓對映中的部份物件。上半部的「定位器窗格」是一個搜尋機能，可用來指定要併入對映中的物件的「名稱」、「類型」及「檔案庫」。搜尋結果顯示於「檔案庫樹」和「檔案庫表格」標籤下方之下半部「定位器窗格」中。當結果

顯示於這些標籤下方時，您可以在物件上按一下右鍵選取「新增到對映」或按兩下物件名稱，將物件新增到對映中。然後，於建立對映後，您可以按一下「對映中的物件」標籤來查看對映中的物件清單。

對映窗格

「資料庫領航員」視窗右邊的「對映窗格」以圖形方式顯示資料庫物件及其關係。在「對映窗格」中，您可以：

- 新增已存在於系統中但不包含在現行對映案例中的表格和概略表。
- 從對映中移除物件
- 變更物件位置
- 放大或縮小物件
- 變更對映中的物件
- 產生對映中所有物件的 SQL

狀態列

物件狀態列

「資料庫領航員」視窗左下方的「物件狀態列」可顯示對映中可見的和可選取的物件數目。

動作狀態列

「資料庫領航員」視窗正中央下方的「動作狀態列」清楚說明對映中發生的情況及是否有擱置中的修改。

修改狀態列

「修改狀態列」指出一項修改已完成或處於擱置中。

使用「資料庫領航員」的秘訣

- 若要變更視窗中任一方的大小，請拖曳兩方中間的條欄 (分割器)。
- 請記得在視窗左邊和右邊的物件上按一下右鍵。按一下右鍵功能表可讓您快速存取常用功能。
- 若要快速開啓檔案庫並顯示其中的物件，請連按兩下檔案庫。
- 若要存取各種「資料庫領航員」指令，請使用**功能表**列或**工具列**。

建立「資料庫領航員」對映

您爲資料庫所建立的視覺化描述稱爲「資料庫領航員對映」。實際上，對映是當您選擇建立對映或重新整理現有的對映時，資料庫資料的一個瞬像。這種資料庫圖形式表示法主要是讓您能夠瞭解現有的複雜資料庫、建立新的資料庫、與您的資料庫通信以及管理資料庫中的物件。若要建立「資料庫領航員對映」：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器 → **資料庫** → 您要使用的資料庫 → **資料庫領航員**。
2. 在**資料庫領航員**上按一下右鍵，選取**新建**。
3. 在**資料庫領航員定位器窗格**中按一下**檔案庫樹**標籤，然後選取檔案庫，其中含有您要用來建立對映的表格、概略表或索引。
4. 展開物件類型 (表格、索引或概略表)。
5. 在您要建立對映的物件上按一下右鍵，選取**新增到對映**或連按兩下物件。

註: 您亦可按一下 **iSeries 領航員**視窗底端的作業工作台上之**對映您的資料庫**作業。

您可以從**檔案**功能表中選取**結束**來儲存此對映。然後，如果有擱置中的變更，請在**另存變更**對話框中選取**是**。此對映可於稍後重新開啓。

一旦您已爲資料庫建立此對映，您可以執行下列動作：

- 新增新的物件到對映中
- 變更物件以來併入於對映中
- 建立使用者定義關係

新增新的物件到對映

透過「資料庫領航員」，您可以建立新的 SQL 物件來新增到對映中。可以建立的物件如下：

- 表格
- 日誌
- 概略表

若要建立新的 SQL 物件來顯示於對映中：

1. 建立或開啓「資料庫領航員」對映。
2. 在對映窗格中按一下右鍵，選取**建立**。
3. 選取您要建立的物件類型。

變更物件以併入於對映中

依據預設，「資料庫領航員」會搜尋及併入對映中的所有物件。若要限制搜尋的物件數，您可以變更使用者喜好設定。

若要變更要併入對映中的物件，請執行下列動作：

1. 建立或開啓「資料庫領航員對映」。
2. 從**選項**功能表中選取**使用者喜好設定**。
3. 在**使用者喜好設定**對話框的**當新增物件到對映時，尋找這些相關的物件**群組框中，選取您要併入的物件，或取消選取您不要併入的物件。
4. 按一下**確定**。
5. 如果您要重新整理對映來套用新的喜好設定，請在「資訊框」中按一下**是**。

建立使用者定義關係

如果您已有程式定義的關係，您可以在「資料庫領航員」中建立使用者定義關係，讓您的關係顯示於對映中。例如，建立一個使用者定義關係來提醒程式設計師兩個表格之間的重要結合。

若要新增使用者定義關係到對映中，請執行下列動作：

1. 建立或開啓「資料庫領航員對映」。
2. 在對映上按一下右鍵，選取**建立**。
3. 選取**使用者定義關係**。
4. 指定使用者定義關係的**名稱**和**說明**。和某些 iSeries 領航員功能的說明是可選用的不一樣，對使用者定義關係提供一個有意義的說明是非常重要的，因爲這是指出使用者定義關係所代表的意義的唯一方式。
5. 從物件清單中選取您要併入關係中的物件。

6. 選擇您要的物件形狀和顏色。

使用「執行 SQL Script」來查詢資料庫

iSeries 領航員的**執行 SQL Script** 視窗，可讓您建立、編輯、執行及疑難排解 SQL 陳述式的 Script。當您處理完 Script 時，就可將它們儲存到 PC 上。您可以使用「執行 SQL Script」來：

- 建立 SQL Script
- 執行 SQL Script
- 檢視儲存程序的結果集
- 檢視工作日誌
- 變更執行 SQL Script 的選項
- 產生資料庫物件的 SQL

執行 SQL Script 視窗有幾個主要區域：

輸入

執行 SQL Script 視窗的上半部爲**輸入**窗格。此區域用來建立和編輯您要執行的 SQL 陳述式。您可手動建立陳述式或從**範例**清單中選取。您亦可使用**產生 SQL** 功能在現行游標位置上插入產生的 SQL。

範例

範例清單框會列出 SQL 陳述式和「控制語言 (CL)」指令的範例。選取一個陳述式，按一下「插入」，將範例置於「輸入」窗格中的現行游標位置。

註: 每一個陳述式必須以一個分號來分開。

輸出

執行 SQL Script 視窗的下半部爲**輸出**窗格，由**訊息**標籤及其他顯示執行的 SQL 陳述式的輸出標籤所組成。**訊息**標籤基於執行的 SQL 陳述式來提供意見。

「執行 SQL Script」視窗的秘訣

- 您可以不必啓動 iSeries 領航員而使用 SQL Script。一旦您已儲存 Script 檔，您可以連按兩下 Script 檔來直接使用，而不必啓動 iSeries 領航員。
- 使用分號 (;) 來分隔陳述式。
- 使用兩個破折號 (--) 使行的剩餘部份成爲註解。
- 若要製作較長的註解，請以 ″/*″ 做爲註解開頭，以 ″*/″ 做爲結尾。這種方式的註解在長度上沒有限制。
- 將游標置於陳述式上，執行時會自動選取此陳述式。
- 在陳述式開頭加上 CL: 即可執行「控制語言 (CL)」指令。您可以使用可在批次作業中提出的任何 CL 指令。

建立 SQL Script

若要建立 SQL Script：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器 → **資料庫** → 您要使用的資料庫 →。
2. 在您要使用的資料庫上按一下右鍵，選取**執行 SQL Script**。
3. 由**執行 SQL Script** 視窗中選取**新建**。

4. 手動建立陳述式，從**範例**清單中插入範例，或使用產生 SQL 功能來擷取現有物件的 SQL。
5. 在您完成建立陳述式後，您可以從**執行**功能表中選取**語法檢查**來檢查陳述式的語法。

完成語法檢查時，您可以從**檔案**功能表中選取**儲存**來儲存 Script。此時會提示您指定 Script 的位置和名稱。

一旦建立 Script，您就可以：

- 執行 SQL Script
- 檢視工作日誌
- 變更執行 SQL Script 的選項
- 產生資料庫物件的 SQL

執行 SQL Script

若要執行 SQL Script，請從**執行**功能表中選取下列其中一個選項：

- **全部** - 從頭到尾執行一次 SQL Script。如果發生錯誤，且已開啓**錯誤時停止**選項，則程式會停止，發生錯誤的陳述式會保持選取狀態。
- **由已選取的位置開始** - 從第一個選取的陳述式或從現行游標位置開始來啓動 SQL Script，一直到 Script 的結尾。
- **已選取的** - 執行已選取的陳述式。

結果會加到**訊息**標籤的尾端。如果未選取**選項**功能表的**智慧型陳述式選取**選項，則選取的文字將視爲單一 SQL 陳述式來執行。

變更執行 SQL Script 的選項

若要變更 SQL Script 的執行選項，請從**選項**功能表中選取下列其中一個選項：

錯誤時停止

開啓或關閉錯誤時停止。如果選取此選項，則發生錯誤時，SQL Script 會停止執行，導致錯誤的陳述式會保持選取狀態。

智慧型陳述式選取

開啓或關閉「智慧型陳述式選取」。如果選取此選項，則當選取**執行**功能表之**已選取的**指令時，所有高亮度顯示的陳述式將依序執行。如果未選取此選項，則**已選取的**指令會將高亮度顯示的文字視爲單一 SQL 陳述式來執行。當高亮度顯示一或多個陳述式的一部份時，選取**智慧型陳述式選項**亦可確保會執行完整的陳述式。

在不同視窗中顯示結果

將 SELECT 陳述式的結果顯示在不同的視窗中，不顯示於**輸出**窗格中。

在工作日誌中併入除錯訊息

開啓或關閉於**執行 SQL Script** 視窗中執行的陳述式除錯。您可以開啓此選項、執行陳述式、然後重新整理**工作日誌**視窗來查看查詢最佳化工具及其他資料庫除錯訊息。

檢視儲存程序的結果集

您可以在「執行 SQL Script」中鍵入 CALL 陳述式做爲 SQL 陳述式來檢視儲存程序的結果集。結果會在結果視窗中出現，就像正常查詢一樣。如果有多個結果集，則以

附加的結果標籤來呈現每一個結果集。此外，您可以在**訊息**標籤中檢視輸出參數。關於使用 CALL 的詳細資訊，請參閱 SQL Reference 一書的 CALL。

檢視工作日誌

工作日誌可顯示與工作相關的訊息。若要查看查詢最佳化工具及其他資料庫除錯訊息，請從**選項**功能表中選取**在工作日誌中併入除錯訊息**，重新執行陳述式。如果執行這項動作時已開啓**工作日誌**對話框，請重新整理概略表來查看新的訊息。若要檢視「工作日誌」：

從**檢視**功能表中選取**工作日誌**。

註: 使用**清除執行歷程**不會清除「工作日誌」，所以您可以使用「工作日誌」來查看已不存在於**輸出**窗格中的訊息。

若要停止或取消執行 SQL Script，請從「執行」功能表中選取下列其中一個選項：

於現行陳述式之後停止
在目前執行的陳述式結束之後，停止執行 SQL Script。

取消要求
要求系統取消現行的 SQL 陳述式。不過，因爲不是所有 SQL 陳述式都可以取消，所以即使在使用此選項之後， SQL 陳述式仍可能繼續完成。在按下**取消要求**之前已完成主要處理程序的 SQL 陳述式，也將繼續完成。例如，已完成查詢處理程序但尚未將結果傳回從屬站的 SELECT 陳述式，通常無法取消。

使用「產生 SQL」來重建 SQL 陳述式

「產生 SQL」可讓您重建用來建立現有資料庫物件的 SQL。此處理程序通常稱爲反推。您可以產生「綱目」、「表格」、「類型」、「概略表」、「程序」、「函數」、「別名」及「索引」的 SQL。此外，如果您要產生 SQL 的表格有相關的限制或觸發程式，則也會一併產生這些相關的 SQL。您可以一次產生一個或多個物件的 SQL。您亦可選擇將產生的 SQL 傳送到「執行 SQL Script」視窗來執行或編輯，或直接將產生的 SQL 寫入資料庫或 PC 檔案中。

關於使用「產生 SQL」的詳細資訊，請參閱下列主題：

- 產生資料庫物件的 SQL
- 編輯物件清單

產生資料庫物件的 SQL

若要產生用來建立現有資料庫物件的 SQL：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器 → **資料庫** → 您要使用的資料庫 → **檔案庫**。
2. 若要產生檔案庫的 SQL，請在檔案庫上按一下右鍵，選取**產生 SQL**。
3. 若要爲檔案庫中的物件產生 SQL，請按一下您要產生 SQL 的物件所在的檔案庫。
4. 在您要產生 SQL 的物件上按一下右鍵，選取**產生 SQL**。

您可以變更標籤上的值來變更產生 SQL 時的預設選項。在**輸出**標籤上，您可以選擇目的地來存放產生的 SQL。您可以將產生的 SQL 傳送到**執行 SQL Script**，或將其直接儲存到 PC 檔或系統檔案。在**選項**標籤上，您可以選擇要讓產生的 SQL 遵循的**標準**，

此外，還可以選擇併入參考訊息、併入捨棄訊息、產生標籤以及格式化以方便閱讀。在**格式**標籤上，您可以選取格式選項。這些選項必須符合**標準**選項。

編輯要產生 SQL 的物件清單

您可以編輯要產生 SQL 的物件清單。若要新增物件：

1. 按一下**新增**。
2. 在**編輯物件清單**對話框上，導覽至您要併入的物件，並選取**新增**。
3. 按一下**確定**來回到主要的產生 SQL 對話框。

若要從清單中移除物件：

1. 從**要產生 SQL 的物件**中選取您要移除的物件。
2. 按一下**移除**。

使用 iSeries 領航員的進階表格功能

第 31 頁的第 3 章, 『iSeries 領航員資料庫入門』解釋您可以執行的一些基本表格功能。不過，您亦可使用 iSeries 領航員：

- 建立別名
- 新增索引
- 新增索引鍵限制
- 新增核對限制
- 新增參照限制
- 新增觸發程式
- 啓用和停用觸發程式
- 移除限制或觸發程式

使用 iSeries 領航員來建立別名

別名是表格或概略表的替代名稱。當有現存的表格或概略表可供參照的情況下，您可以使用別名來參照表格或概略表。您可以爲表格或概略表建立別名，或爲表格或概略表的成員建立別名。

1. 在 **iSeries 領航員**視窗中，展開您的伺服器 → **資料庫** → 您要使用的資料庫 → **檔案庫**。
2. 在您要其中建立新別名的檔案庫上按一下右鍵。
3. 從蹦現功能表中，選取**新建**，再選取**別名**。
4. 在**新的別名**對話框的**別名**欄位中，指定您要建立的別名。名稱不可以與伺服器上已存在的任何索引、表格、概略表、檔案或別名相同。
5. 在**說明**欄位中，指定新別名的說明。此說明的長度最多可以包含 50 個字元。此欄位是可選用的。
6. 在**表格/概略表**欄位中，指定您要讓別名指向的表格或概略表。
7. 在**檔案庫**欄位中，指定您要讓別名指向的表格或概略表所在的檔案庫。
8. 按一下**進階**。
9. 在**進階**對話框上，如果您要爲表格或概略表建立別名，請按一下**為表格或概略表建立別名**。此爲預設選項。

10. 若要爲表格或概略表的成員建立別名，請按一下**為表格或概略表的成員建立別名**。在組合框中，輸入或選取您要讓別名指向的成員。
11. 按一下**確定**來返回**新的別名**對話框。
12. 按一下**確定**來建立別名。

註: 您亦可在表格或概略表上按一下右鍵並選取**建立別名**來建立別名。
關於別名的詳細資訊，請參閱第 52 頁的『建立及使用 ALIAS 名稱』。

使用 iSeries 領航員來新增索引

您可以使用索引來排序和選取資料。此外，索引可協助系統更快速擷取資料，提高查詢效能。

您可以建立許多索引。不過，因爲是由系統來維護索引，太多的索引反而會降低效能。關於索引和查詢效能的詳細資訊，請參閱 *Database Performance and Query Optimization* 資訊中的 Effectively Using SQL Indexes。

有一種稱爲編碼向量索引的索引類型，可以更快速地掃描，可以更容易地平行處理。

關於使用編碼向量索引的詳細資訊，請瀏覽 DB2 for iSeries 網頁。

您可以在新的或現有的表格上建立索引。您可以使用 iSeries 領航員來建立基數索引或編碼向量索引：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器 → **資料庫** → 您要使用的資料庫 → **檔案庫**。
2. 按一下您要新增索引之表格所在的檔案庫。
3. 在明細窗格中，在您要新增索引的表格上按一下右鍵，選取**內容**。
4. 在**表格內容**或**新的表格**對話框上，選取「索引」標籤。
5. 在**索引**標籤上，按一下**新建**。
6. 在**索引**欄位中，指定新索引的名稱。
7. 在**檔案庫**欄位中，選取索引所在的檔案庫。
8. 選取將組成索引的表格直欄。若要新增直欄，請按一下直欄，左邊會出現一個數字。數字決定索引中直欄的鍵位置。若要從索引中移除直欄，請再按一下直欄。
9. 若要將鍵值欄位的次序從升序順序變更爲遞減順序 (或遞減順序變更爲升序順序)，請按一下第二個直欄。
10. 選取**索引類型**。
11. 如果您建立編碼向量索引，請選取特殊值的數字。
12. 按一下**確定**來建立索引。

註: 您亦可在**新的表格**對話框中新增索引到新的表格中。

您只能修改在現行表格編輯階段作業中已定義的限制。如果您新增限制，然後在**新的表格**對話框或**表格內容**對話框上按一下**確定**，則您只具備此限制的唯讀存取權。如果您要變更限制內容，則必須捨棄限制，再以適當的變更來重建。

關於建立索引的詳細資訊，請參閱第 55 頁的『新增索引』。

使用 iSeries 領航員來新增索引鍵限制

限制是資料庫管理程式所實施的規則。DB2 UDB for iSeries 支援兩種索引鍵限制類型：

- 唯一索引鍵限制是指索引鍵的值必須是唯一的才有效。唯一限制實施於執行 INSERT 和 UPDATE 陳述式期間。
- 主要索引鍵限制是一種唯一限制。差別在於主要索引鍵不可包含任何可爲空值直欄。

若要建立索引鍵限制，請執行下列動作：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器 → **資料庫** → 您要使用的資料庫 → **檔案庫**。
2. 連按兩下您要新增核對限制之表格所在的檔案庫。
3. 在您要新增索引鍵的表格上按一下右鍵，選取**內容**。
4. 在**表格內容**對話框上，選取**索引鍵限制**標籤。
5. 在**索引鍵限制**標籤上，按一下**新建**。
6. 在**新的索引鍵限制**對話框上，指定名稱於名稱文字框中。如果未指定名稱，系統會自動產生一個名稱。
7. 選取您要新增索引鍵的直欄。
8. 選取**主要的**來建立主要索引鍵，或選取**唯一的**來建立唯一鍵。
9. 按一下**確定**以返回**表格內容**對話框。
10. 按一下**確定**以建立索引鍵。

註: 您亦可在**新的表格**對話框中新增索引鍵限制到新的表格中。

您只能修改在現行表格編輯階段作業中已定義的限制。如果您新增限制，然後在**新的表格**對話框或**表格內容**對話框上按一下**確定**，則您只具備此限制的唯讀存取權。如果您要變更限制內容，則必須捨棄限制，再以適當的變更來重建。

關於新增索引鍵限制的詳細資訊，請參閱第 125 頁的第 10 章, 『資料完整性』。

使用 iSeries 領航員來新增核對限制

核對限制藉由限制一個直欄或一組直欄中容許的值，來確保插入和更新期間的資料有效性。

若要建立核對限制，請執行下列動作：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器 → **資料庫** → 您要使用的資料庫 → **檔案庫**。
2. 按一下您要新增索引鍵限制之表格所在的檔案庫。
3. 在您要新增核對限制的表格上按一下右鍵，選取**內容**。
4. 在**表格內容**對話框上，按一下**核對限制**標籤。
5. 在**核對限制**標籤上，按一下**新建**。
6. 在**核對限制搜尋條件**對話框上，於名稱文字框中指定名稱。如果未指定名稱，系統會自動產生一個名稱。
7. 在**直欄**清單上，連按兩下您要新增限制的直欄。直欄會出現在子句方框中。
8. 若要從清單中插入運算子，請連按兩下運算子。運算子會出現在子句方框中。
9. 若要從清單中插入函數，請連按兩下函數。您可以從下拉清單選取函數類型來修改清單。一旦您按兩下函數，函數將出現在子句方框中。

10. 修改表示式，直到正確爲止。
11. 按一下**確定**以返回**表格內容**對話框。
12. 按一下**確定**以建立核對限制。

註: 您亦可在**新的表格**對話框中新增核對限制到新的表格中。

您只能修改在現行表格編輯階段作業中已定義的限制。如果您新增限制，然後在**新的表格**對話框或**表格內容**對話框上按一下**確定**，則您只具備此限制的唯讀存取權。如果您要變更限制內容，則必須捨棄限制，再以適當的變更來重建。

關於新增核對限制的詳細資訊，請參閱第 125 頁的『新增與使用核對限制』。

使用 iSeries 領航員來新增參照限制

參照完整性是指資料庫中一組表格的條件，其中表格之間的所有參照關係皆有效。您可以新增參照限制來確定資料庫中的參照完整性。

若要新增參照限制，請執行下列動作：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器 → **資料庫** → 您要使用的資料庫 → **檔案庫**。
2. 連按兩下含有相依表格的檔案庫。
3. 在相依表格上按一下右鍵，選取**內容**。
4. 在**表格內容**對話框上，選取**參照限制**標籤。
5. 在**參照限制**標籤上，按一下**新建**。
6. 在**新的參照限制**對話框上，指定限制的名稱。如果未指定名稱，系統會自動產生一個名稱。
7. 選取您要相依於上層表格的上層鍵值的直欄。
8. 選取含有上層表格的檔案庫。
9. 選取含有上層鍵的表格。
10. 選取要參照的上層鍵。
11. 選取刪除動作。
12. 選取更新動作 (預設爲插入動作)。
13. 按一下**確定**以返回**表格內容**對話框。
14. 按一下**確定**以建立參照限制。

註: 您亦可在**新的表格**對話框中新增參照限制到新的表格中。

您只能修改在現行表格編輯階段作業中已定義的限制。如果您新增限制，然後在**新的表格**對話框或**表格內容**對話框上按一下**確定**，則您只具備此限制的唯讀存取權。如果您要變更限制內容，則必須捨棄限制，再以適當的變更來重建。

關於新增參照限制的詳細資訊，請參閱第 126 頁的『新增或捨棄參照限制』。

使用 iSeries 領航員來新增觸發程式

觸發程式是指當指定的實體資料庫檔案上執行指定的變更作業時會自動執行的一組動作。在此主題下，表格爲一個實體檔案。變更作業可以是應用程式中的插入、更新或

刪除等高階語言陳述式，或 SQL INSERT、UPDATE 或 DELETE 陳述式。對於實施商業規則、驗證輸入資料及保存審核追蹤等作業，觸發程式非常有用。

您可以使用 iSeries 領航員來定義系統觸發程式和 SQL 觸發程式。此外，您可以啓用或停用觸發程式。

若要新增觸發程式，請執行下列動作：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器 → **資料庫**→ 您要使用的資料庫 → **檔案庫**。
2. 按一下您要新增觸發程式之表格所在的檔案庫。
3. 在您要新增觸發程式的表格上按一下右鍵，選取**內容**。在**表格內容**對話框上，按一下**觸發程式**標籤。
4. 選取**新增系統觸發程式**來新增系統觸發程式。
5. 選取**新增 SQL 觸發程式**來新增 SQL 觸發程式。

關於系統觸發程式的詳細資訊，請參閱*Database Programming* 中的 Triggering automatic events in your database。

關於 SQL 觸發程式的詳細資訊，請參閱第 137 頁的『SQL 觸發程式』。

啓用和停用觸發程式

必須啓用觸發程式才能執行之。不過，停用觸發程式可讓您在使用表格時不讓觸發程式執行。

若要啓用/停用觸發程式，請執行下列動作：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器 → **資料庫** → 您要使用的資料庫 → **檔案庫**。
2. 按一下您要啓用/停用觸發程式之表格所在的檔案庫。
3. 在表格上按一下右鍵，選取**內容**。
4. 在**表格內容**對話框上，按一下**觸發程式**標籤。
5. 選取您要啓用/停用的觸發程式，按一下**啓用**來啓用觸發程式，或按一下**停用**來停用觸發程式。

關於使用 CHGPFTRG CL 指令來啓用和停用觸發程式的相關資訊，請參閱 *Database Programming* 中的 Enabling and disabling a trigger。

移除限制和觸發程式

1. 在 **iSeries 領航員**視窗中，展開您的伺服器 → **資料庫**→ 您要使用的資料庫 → **檔案庫**。
2. 按一下您要移除限制或觸發程式之表格所在的檔案庫。
3. 在您要移除限制或觸發程式的表格上按一下右鍵，選取**內容**。
4. 在**表格內容**對話框上，按一下您要移除的限制或觸發程式的類型標籤。
5. 選取您要移除的限制，按一下**刪除**。

使用 iSeries 領航員來定義 SQL 物件

iSeries 領航員提供在系統上定義一些 SQL 物件的簡單方法。例如，您可以定義：

- 程序：程序 (通常稱爲儲存程序) 是一個程式，可呼叫來執行含有主語言陳述式和 SQL 陳述式的作業。 SQL 中的程序提供與主語言中的程序相同的好處。亦即，通用的程式碼片段只需要撰寫和維護一次，就可以從數個程式中呼叫。關於程序的詳細資訊，請參閱第 147 頁的第 11 章, 『儲存程序』。
- 使用者定義的函數：使用者定義的函數 (UDF) 由三種類型組成：來源、外部及 SQL。來源函數 UDF 會呼叫其他函數來執行作業。 SQL 和外部函數 UDF 需要撰寫和執行個別的程式碼。您可以建立純量、直欄及表格函數。關於「函數」的詳細資訊，請參閱第 185 頁的『使用者定義的函數 (UDF)』。
- 使用者定義的類型：使用者定義的特殊類型是一種機制，可讓您延伸 DB2 功能來超越可用的內建資料類型。使用者定義的特殊類型可讓您在 DB2 中定義新的資料類型，大幅提升您的能力，因爲您不再受限於只能使用系統提供的內建資料類型來模塑化您的業務和攫取資料的語意。特殊資料類型可讓您一對一地對映到現有的資料庫類型。關於類型的詳細資訊，請參閱第 199 頁的『使用者定義的特殊類型 (UDT)』。

關於使用 iSeries 領航員來定義這些物件的詳細資訊，請參閱下列主題：

- 『使用 iSeries 領航員來定義儲存程序』
- 『使用 iSeries 領航員來定義使用者定義的函數』
- 第 263 頁的『使用 iSeries 領航員來定義使用者定義的類型』

使用 iSeries 領航員來定義儲存程序

如果您要在 SQL 程式中像呼叫程序一樣地來呼叫另一個程式，首先您必須將程式定義成一個外部程序。要定義程序的程式，在定義程序時並不需存在。

若要將程式定義成程序，請執行下列動作：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器 → **資料庫** → 您要使用的資料庫 → **檔案庫**。
2. 在您要定義函數的檔案庫上按一下右鍵，選取**新建**。
3. 選取**程序**。
4. 選取**外部**來建立外部儲存程序。
5. 選取 **SQL** 來建立 SQL 儲存程序。

關於建立和定義外部儲存程序的詳細資訊，請參閱第 148 頁的『定義外部程序』。

關於建立和定義 SQL 儲存程序的詳細資訊，請參閱第 148 頁的『定義 SQL 程序』。

使用 iSeries 領航員來定義使用者定義的函數

若要將程式定義爲使用者定義函數，請執行下列動作：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器 → **資料庫** → 您要使用的資料庫 →**檔案庫**。
2. 在您要定義函數的檔案庫上按一下右鍵，選取**新建**。
3. 選取**函數**。
4. 選取**外部**以建立外部使用者定義的函數。
5. 選取 **SQL** 以建立 SQL 使用者定義的函數。
6. 選取**來源**以根據另一個函數建立函數。

關於建立和定義外部函數的詳細資訊，請參閱第 185 頁的『使用者定義的函數 (UDF)』。

使用 iSeries 領航員來定義使用者定義的類型

基於現有的資料類型來建立新的使用者定義資料類型可讓您更能夠控制資料。

若要建立使用者定義的類型，請執行下列動作：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器 → **資料庫** → 您要使用的資料庫 → **檔案庫**。
2. 在您要定義類型的檔案庫上按一下右鍵，選取**新建**。
3. 選取**類型**。
4. 在**新的類型**對話框中，在「類型」欄位中指定您爲新類型命名的名稱。
5. 在**來源資料類型**區段的**類型**欄位中，指定此新類型的基本資料類型。
6. 按一下**確定**。

關於建立和定義使用者定義類型的詳細資訊，請參閱第 199 頁的『使用者定義的特殊類型 (UDT)』。

建立 SQL 資料包

SQL 資料包是永久物件，用來儲存準備的 SQL 陳述式的相關資訊。在資料來源上選取「延伸動態」方框時，ODBC 支援會使用它們。

若要建立 SQL 資料包：

1. 在 **iSeries 領航員**視窗中，展開您的伺服器→ **資料庫**
2. 在您要使用的資料庫上按一下右鍵，選取**新的 SQL 資料包**。
3. 在**建立 SQL 資料包**對話框上，視需要指定參數。
4. 按一下**確定**。

關於完整的參數清單，請參閱第 345 頁的『CRTSQLPKG (建立結構化查詢語言資料包) 指令』。

第 17 章 使用交談式 SQL

本章將說明如何使用交談式 SQL 來執行 SQL 陳述式，以及提示功能的使用方法。其間還將提供概觀資訊，以及交談式 SQL 的相關使用秘訣。如果您想進一步瞭解如何使用 SQL，請參閱第 2 章, 『SQL 入門』。有關如何透過遠端連接來使用交談式 SQL 的特殊注意事項，則請參閱第 274 頁的『以交談式 SQL 存取遠端資料庫』。

相關明細，請參閱 『交談式 SQL 的基本功能』。

註:

1. *集合*一詞與*綱目*同義。
2. 如需相關的程式碼範例，請參閱第 x 頁的『程式碼不保事項聲明』 的資訊。

交談式 SQL 的基本功能

交談式 SQL 可讓程式設計師或資料庫管理者輕鬆且快速地定義、更新、刪除或查看資料，以便進行測試、問題分析及資料庫維護。程式設計師若使用交談式 SQL，即可先在表格中插入列並測試 SQL 陳述式，然後才在應用程式中加以執行。資料庫管理者則可使用交談式 SQL 來授予或撤回專用權；建立或捨棄綱目、表格或概略表；或者從系統目錄表格中選取資訊。

執行交談式 SQL 陳述式後，系統會顯示完成訊息或錯誤訊息。此外，對於長時間執行的陳述式，系統通常會在其間顯示狀態訊息。

您可以將游標移到訊息上並按下「F1=說明」，來顯示相關說明。

交談式 SQL 所提供的基本功能有：

- **陳述式登錄**功能可讓您：
  - 鍵入交談式 SQL 陳述式並加以執行。
  - 擷取和編輯陳述式。
  - 提示輸入 SQL 陳述式。
  - 翻看先前的陳述式與訊息。
  - 呼叫階段作業服務。
  - 呼叫清單選項功能。
  - 跳出交談式 SQL。
- **提示**功能可讓您先鍵入完整的或部份的 SQL 陳述式，然後按「F4=提示」，此時該陳述式的語法提示便會出現。它還可讓您藉由按下 F4 來取得所有 SQL 陳述式的功能表。透過此功能表，您可以選取陳述式並提示其語法。
- **清單選項功能**可讓您從清單中選取已授權的關聯式資料庫、綱目、表格、概略表、直欄、限制或 SQL 資料包。

  您在清單中所作的選項，可插入至 SQL 陳述式中的游標所在位置。
- **階段作業服務**功能可讓您：
  - 變更階段作業屬性。
  - 列印現行階段作業。

– 移除現行階段作業中的所有登錄。
– 將階段作業存入原始檔。

相關明細，請參閱下列各節：

- 『啓動交談式 SQL』
- 第 267 頁的『使用陳述式登錄功能』
- 第 267 頁的『提示』
- 第 270 頁的『使用清單選項功能』
- 第 272 頁的『階段作業服務說明』
- 第 273 頁的『跳出交談式 SQL』
- 第 274 頁的『使用現有的 SQL 階段作業』
- 第 274 頁的『回復 SQL 階段作業』
- 第 274 頁的『以交談式 SQL 存取遠端資料庫』

啓動交談式 SQL

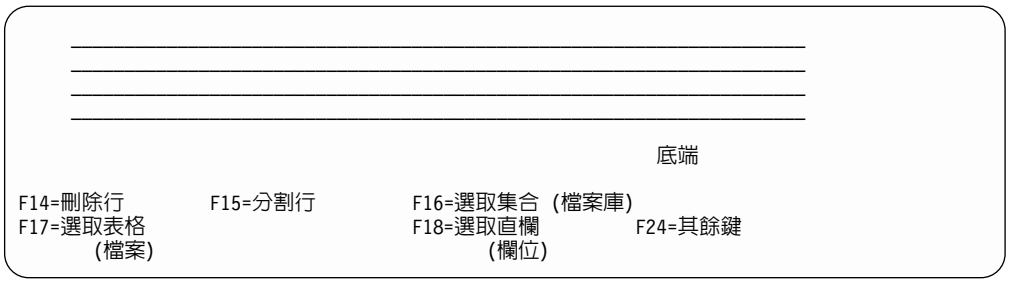
您可以藉著在 OS/400 指令行中鍵入 STRSQL 來開始使用交談式 SQL。有關此指令及其參數的完整說明，請參閱附錄 B, 『DB2 UDB for iSeries CL 指令說明』。

首先出現的是「輸入 SQL 陳述式」顯示畫面。這就是交談式 SQL 的主畫面。透過此畫面，您可以輸入 SQL 陳述式及使用：

- F4=提示
- F13=階段作業服務
- F16=選取集合
- F17=選取表格
- F18=選取直欄

```
                              輸入 SQL 陳述式

請鍵入 SQL 陳述式，然後按 Enter 鍵。
    現行連線是連接關聯式資料庫 rdjacque。
===>_____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
    _____________________________________________________________________
                                                                      底端
F3=跳出    F4=提示      F6=插入行        F9=擷取        F10=複製行
F12=取消               F13=服務         F24=其餘鍵
```

按F24=其餘鍵以檢視其餘功能鍵。



註: 如果您是採用系統命名慣例，系統將顯示括弧中的名稱，而非上述畫面中的原本名稱。

交談式階段作業的組成項目包含：

- 您爲 STRSQL 指令所指定的參數值
- 您在階段作業中所輸入的 SQL 陳述式，以及接在各 SQL 陳述式之後的對應訊息
- 您以階段作業服務功能所變更的任何參數值
- 您所作的清單選項

交談式 SQL 會提供唯一的階段作業 ID，它是由您的使用者 ID 與現行工作站 ID 所組成。此種階段作業 ID 概念可讓多位具有相同使用者 ID 的使用者，透過多個工作站同時使用交談式 SQL。另外，同一使用者 ID 也可從相同工作站同時執行多個交談式 SQL 階段作業。

若某一 SQL 階段作業已存在且正在重新進入，在 STRSQL 指令上所指定的任何參數皆會被忽略。系統將會使用現有 SQL 階段作業的參數。

使用陳述式登錄功能

陳述式登錄功能是您選取交談式 SQL 之後，所進入的第一項功能。處理完每一個交談式 SQL 陳述式後，您仍回到陳述式登錄功能。

在陳述式登錄功能中，您可以鍵入或要求提示整個 SQL 陳述式，然後按 Enter 鍵加以提交處理。

鍵入陳述式

您在指令行所鍵入的陳述式其長度可以是一或數行。但在交談式 SQL 中，您無法爲 SQL 陳述式鍵入備註說明。待陳述式處理完畢後，陳述式與結果訊息將一併送往顯示畫面。此時您可繼續輸入另一陳述式。

若一陳述式已通過 SQL 的辨識但含有語法錯誤，此陳述式與結果文字訊息 (語法錯誤) 將會被送往顯示畫面。系統會在輸入區中顯示同樣的陳述式並將游標放在語法錯誤之處。您可以將游標移至訊息上，然後按「F1=說明」以取得有關該錯誤的進一步資訊。

您可以翻看先前的陳述式、指令及訊息。若將游標移至前一陳述式上並按下「F9=擷取」，即可在輸入區中複製該陳述式。如果您需要更多空間來鍵入 SQL 陳述式，可在顯示畫面上向下翻頁。

提示

提示功能可協助您針對所要使用的陳述式，來提供必需的語法資訊。提示功能對下列三種陳述式處理模式皆適用：*RUN、*VLD 及 *SYN。

當使用提示程式時，您有兩種選項：

- 先鍵入陳述式的動詞，然後再按「F4=提示」。

  陳述式即會進行剖析，而已完成的子句將被填入到提示畫面。

  如果您鍵入 SELECT 並按下「F4=提示」，下列畫面將會出現：

```
                          指定 SELECT 陳述式

請鍵入 SELECT 陳述式資訊。 然後按 F4 以列示。

  FROM 表格 . . . . . .  _____________________________________________
  SELECT 直欄 . . . . . .  _____________________________________________
  WHERE 條件 . . . . . .  _____________________________________________
  GROUP BY 直欄 . . . . . .  _____________________________________________
  HAVING 條件 . . . . . .  _____________________________________________
  ORDER BY 直欄 . . . . . .  _____________________________________________
  FOR UPDATE OF 直欄 . . . .  _____________________________________________

                                                                  底端
請鍵入選項，然後按 Enter 鍵。

  結果表格中的 DISTINCT 列 . . . . . .  N   Y=是, N=否
  UNION 另一個 SELECT . . . . . .  N   Y=是, N=否
  指定額外選項 . . . . . .  N   Y=是, N=否



F3=跳出          F4=提示     F5=重整      F6=插入行        F9=指定子查詢
F10=複製行       F12=取消    F14=刪除行   F15=分割行       F24=其餘鍵
```

- 先按「F4=提示」，然後在「輸入 SQL 陳述式」畫面中鍵入相關內容。您將會看到列有各類陳述式的清單。此陳述式清單的內容會隨著現行交談式 SQL 陳述式處理模式而變。對 *NONE 以外之語言的語法檢查模式而言，該清單會列出所有的 SQL 陳述式。對執行與驗證模式而言，只有能在交談式 SQL 中執行的陳述式會被列出。您可以從中選取所要使用的陳述式編號。系統則會針對您所選取的陳述式提供提示。

  如果您按下「F4=提示」且未鍵入任何內容，下列畫面將會出現：

```
                          選取 SQL 陳述式

請選取下列其中一項：

     1. ALTER TABLE
     2. CALL
     3. COMMENT ON
     4. COMMIT
     5. CONNECT
     6. CREATE ALIAS
     7. CREATE COLLECTION
     8. CREATE INDEX
     9. CREATE PROCEDURE
    10. CREATE TABLE
    11. CREATE VIEW
    12. DELETE
    13. DISCONNECT
    14. DROP ALIAS
                                                        尚有資訊...
選項
     __

F3=跳出    F12=取消
```

若在提示畫面上按下「F21=顯示陳述式」，提示程式會就迄今已填寫的內容，顯示經格式化的 SQL 陳述式。

若在提示中按下 Enter 鍵，透過提示螢幕所建立的陳述式將會插入至階段作業中。如果陳述式處理模式爲 *RUN，陳述式將會執行。如果發生錯誤，提示程式會持續掌控。

語法檢查

當 SQL 陳述式進入提示程式時，系統即會檢查其語法。提示程式將不會接受語法錯誤的陳述式。您必須更正語法或移除錯誤的部份，否則提示將無法進行。

陳述式處理模式

陳述式處理模式可在「變更階段作業屬性」畫面中選取。在 *RUN (執行) 或 *VLD (驗證) 模式下，系統只會提示被允許在交談式 SQL 中執行的陳述式。在 *SYN (語法檢查) 模式下，所有 SQL 陳述式皆被允許。陳述式在 *SYN 或 *VLD 模式下實際上並未執行；而只是檢查語法與物件的存在性。

子查詢

您可以在具有 WHERE 或 HAVING 子句的任何顯示畫面上選取子查詢。若要顯示子查詢畫面，可在游標位於 WHERE 或 HAVING 輸入行之上時，按下「F9=指定子查詢」。此時即會出現顯示畫面供您鍵入子選擇資訊。按 F9 時，如果游標是位於子查詢的括弧中，子查詢資訊將會填入下一顯示畫面。若游標是位於括弧之外，下一顯示畫面即爲空白。有關子查詢的詳情，請參閱 第 97 頁的『SELECT 陳述式中的子查詢』。

CREATE TABLE 提示

爲 CREATE TABLE 進行提示時，系統可提供相關支援供您輸入個別的直欄定義。請將游標移至顯示畫面的直欄定義區段中，然後按「F4=提示」。此時即會出現另一畫面，提供空間用以輸入某一直欄定義的所有資訊。

若要輸入超過 18 個字元的直欄名稱，請按「F20=顯示完整名稱」。一視窗即會出現，且具有足以容納 30 個字元長之名稱的空間。

「F6=插入行」、「F10=複製行」及「F14=刪除行」等編輯鍵，可用來在直欄定義清單中新增及刪除登錄。

輸入 DBCS 資料

不論是「輸入 SQL 陳述式」畫面上與 SQL 提示程式中，兩者在處理多行 DBCS 資料時的規則都相同。每一行所含的移入字元與移出字元數目必須相同。在處理需要多行才夠輸入的 DBCS 資料字串時，額外的移入與移出字元會被移除。若某行的最後一個直欄含有移入字元，而下一行的第一個直欄則含移出字元，提示程式在組合此兩行時會移除所述的移入與移出字元。若某行的最後兩個直欄含有一個移入字元再接一個單位元組空白，而下一行的第一個直欄則含移出字元，此移入、空白、移出之組合會在併行時被移除。此種移除動作可讓 DBCS 資訊在讀取時形同連續的字串。

試舉一例說明，假設您已輸入下列 WHERE 條件。在兩行字串區段的頭尾處皆會顯示移位字元。

```
                          指定 SELECT 陳述式

請鍵入 SELECT 陳述式資訊。 然後按 F4 以列示。

  FROM 表格 . . . . . .  TABLE1_____________________________________
  SELECT 直欄 . . . . . .  *_____________________________________
  WHERE 條件  . . . . . .  COL1 = '<AABBCCDDEEFFGGHHIIJJKKLLMMNNOOPPQQ>
                           <RRSS>'_____________________________________
  GROUP BY 直欄 . . . . .  _____________________________________
  HAVING 條件 . . . . . .  _____________________________________
  ORDER BY 直欄 . . . . .  _____________________________________
  FOR UPDATE OF 直欄  . .  _____________________________________
```

按下 Enter 鍵時，字串將合併於一處，移除額外的移位字元。此時，「輸入 SQL 陳述式」畫面所顯示的陳述式如下：

```
SELECT * FROM TABLE1 WHERE COL1 = '<AABBCCDDEEFFGGHHIIJJKKLLMMNNOOPPQQRRSS>'
```

使用清單選項功能

在特定的提示畫面上按下 F4，或在「輸入 SQL 陳述式」畫面上按下 F16、F17 或 F18，即可存取清單選項功能。按下功能鍵後，您就會看到一份清單，供您選擇已授權的關聯式資料庫、綱目、表格、概略表、別名、直欄、限制、程序、參數或資料包。如果是要求表格清單，但先前並未選取綱目，系統會請您先選取綱目。

在清單上，您可以選取一或多個項目，並以數字來指定其在陳述式中的出現順序。跳出清單功能後，您所作的選項即會插入到您原來畫面上的游標所在位置。

請一律選取您最想使用的清單。例如，假設您想要一份直欄清單，不過您認爲所要的直欄是在目前未選定的表格中，此時可按下「F18=選取直欄」。然後，在直欄清單中按下 F17 來變更表格。如果最先選取的是表格清單，表格名稱將插入至您的陳述式中。而您將無法選取直欄。

在「輸入 SQL 陳述式」畫面上鍵入 SQL 陳述式的同時，您可以隨時要求清單。您在清單中所作的選項會被插入「輸入 SQL 陳述式」畫面上。它們會按照您在清單畫面中指定的數字順序，依次插入到游標所在位置。雖然系統會爲您新增所選定的清單資訊，不過您仍必須鍵入陳述式的關鍵字。

清單功能雖會儘量爲選定的直欄、表格及 SQL 資料包，提供必需的限定資格。不過，有時它會無法決定 SQL 陳述式的用途。因此您必須複查 SQL 陳述式，並驗證所選取的直欄、表格及 SQL 資料包皆已適當地限定。

範例：使用清單選項功能

下述範例將示範如何使用清單功能來建置 SELECT 陳述式。

假設您已：

- 在 OS/400 指令行上鍵入 STRSQL 而進入交談式 SQL。
- 尚未進行任何清單選項或登錄。
- 選取 *SQL 命名慣例。

註: 範例中所示的清單並不存在於您的伺服器上。他們只是作爲範例之用。

開始使用 SQL 陳述式：

1. 在第一個陳述式登錄行鍵入 SELECT。

2. 在第二個陳述式登錄行鍵入 FROM。
3. 將游標留在 FROM 之後。

```
                              輸入 SQL 陳述式

請鍵入 SQL 陳述式，然後按 Enter 鍵。
===> SELECT
     FROM _
```

4. 由於您想將表格名稱接在 FROM 之後，所以請先按下「F17=選取表格」來取得表格清單。

   出現的不是您所預期的表格清單，而是集合清單 (「選取及排序集合」顯示畫面)。此時您已進入 SQL 階段作業，且尚未選取要處理的綱目。
5. 在 YOURCOLL2 綱目旁的*順序*直欄中鍵入 1。

```
                          選取及排序集合

請鍵入序號 (1-999) 來選取集合，然後按 Enter 鍵。

順序  集合            類型   本文
      YOURCOLL1       SYS    公司福利
1     YOURCOLL2       SYS    員工個人資料
      YOURCOLL3       SYS    工作分類/需求
      YOURCOLL4       SYS    公司保險
```

6. 按 Enter 鍵。

   「選取及排序表格」畫面即會出現，顯示出 YOURCOLL2 綱目中的現有表格。
7. 在 PEOPLE 表格旁的*順序*直欄中鍵入 1。

```
                          選取及排序表格

請鍵入序號 (1-999) 來選取表格，然後按 Enter 鍵。

順序 表格                  集合          類型  本文
     EMPLCO                YOURCOLL2     TAB   員工公司資料
1    PEOPLE                YOURCOLL2     TAB   員工個人資料
     EMPLEXP               YOURCOLL2     TAB   員工經歷
     EMPLEVL               YOURCOLL2     TAB   員工評量報告
     EMPLBEN               YOURCOLL2     TAB   員工福利記錄
     EMPLMED               YOURCOLL2     TAB   員工醫療記錄
     EMPLINVST             YOURCOLL2     TAB   員工投資記錄
```

8. 按 Enter 鍵。

   「輸入 SQL 陳述式」畫面再次出現，且表格名稱 YOURCOLL2.PEOPLE 插放於 FROM 之後。此表格名稱是按 *SQL 命名慣例由綱目名稱來限定。

```
                              輸入 SQL 陳述式

請鍵入 SQL 陳述式，然後按 Enter 鍵。
===> SELECT
     FROM YOURCOLL2.PEOPLE _
```

9. 將游標移到 SELECT 之後。
10. 由於您想將直欄名稱接在 SELECT 之後，所以請按下「F18=選取直欄」以取得直欄清單。

    「選取及排序直欄」畫面即會出現，顯示出 PEOPLE 表格中的直欄。
11. 在 *NAME* 直欄旁的*順序*直欄中鍵入 2。
12. 在 *SOCSEC* 直欄旁的*順序*直欄中鍵入 1。

```
                          選取及排序直欄

請鍵入序號 (1-999) 來選取直欄，然後按 Enter 鍵。

順序 直欄                 表格                 類型                 位數長度
2    NAME                 PEOPLE               CHARACTER                 6
     EMPLNO               PEOPLE               CHARACTER                30
1    SOCSEC               PEOPLE               CHARACTER                11
     STRADDR              PEOPLE               CHARACTER                30
     CITY                 PEOPLE               CHARACTER                20
     ZIP                  PEOPLE               CHARACTER                 9
     PHONE                PEOPLE               CHARACTER                20
```

13. 按 Enter 鍵。

    「輸入 SQL 陳述式」畫面再次出現，且 SOCSEC、NAME 出現在 SELECT 之後。

```
                          輸入 SQL 陳述式

請鍵入 SQL 陳述式，然後按 Enter 鍵。
===> SELECT SOCSEC, NAME
     FROM YOURCOLL2.PEOPLE
```

14. 按 Enter 鍵。

    所建立的陳述式就會開始執行。

一旦使用了清單功能後，您所選取的值將保持有效，除非您加以變更或在「變更階段作業屬性」畫面上變更了綱目清單。

階段作業服務說明

交談式 SQL 的「階段作業服務」畫面，可藉由在「輸入 SQL 陳述式」畫面中按下 F13 來要求。

透過此顯示畫面，您可以變更階段作業屬性，以及列印、清除或儲存階段作業 (存至原始檔中)。

「選項 1 (變更階段作業屬性)」會顯示「變更階段作業屬性」畫面，供您爲交談式 SQL 階段作業選取有效的現行值。此畫面中的選項會隨著所選取的陳述式處理選項而變。

可供變更的階段作業屬性如下：

- 確定控制屬性。
- 陳述式處理控制。
- SELECT 輸出裝置。
- 綱目清單。
- 清單類型，用來選取所有的系統與 SQL 物件，或僅選取您的 SQL 物件。

- 顯示資料時的重新整理資料選項。
- 允許複製資料選項。
- 命名選項。
- 程式設計語言。
- 日期格式。
- 時間格式。
- 日期分隔字元。
- 時間分隔字元。
- 小數點表示法。
- SQL 字串定界符號。
- 排序順序。
- 語言 ID。

「選項 2 (列印現行階段作業)」可存取「變更印表機」顯示畫面，供您立即列印現行階段作業，然後繼續工作。其間系統會提示印表機資訊。您輸入的所有 SQL 陳述式，以及系統顯示的所有訊息，都會比照其在「輸入 SQL 陳述式」畫面的樣式來列印。

「選項 3 (移除現行階段作業中的所有登錄)」可供您移除「輸入 SQL 陳述式」畫面中以及階段作業歷程中的所有 SQL 陳述式與訊息。系統會提示您是否確定要刪除資訊。

「選項 4 (將階段作業存入原始檔)」可存取「變更原始檔」畫面，供您將階段作業存入原始檔中。系統會提示您輸入原始檔檔名。此功能可藉由使用原始檔登錄公用程式 (SEU)，讓您將原始檔內含在主語言程式中。

註: 選項 4 可讓您將原型化 SQL 陳述式內含在使用 SQL 的高階語言 (HLL) 程式中。以選項 4 所建立的原始檔經編輯後，可作爲「執行 SQL 陳述式 (RUNSQLSTM)」指令的輸入原始檔。

跳出交談式 SQL

在「輸入 SQL 陳述式」畫面中按下「F3=跳出」，即可讓您離開交談式 SQL 環境並執行下列其中一項：

1. 儲存並跳出階段作業。 離開交談式 SQL。 您的現行階段作業將被儲存，並留待您下次啓動交談式 SQL 時使用。
2. 跳出但不儲存階段作業。 離開交談式 SQL 而不儲存您的階段作業。
3. 回復階段作業。 留在交談式 SQL 中並返回「輸入 SQL 陳述式」畫面。現行的階段作業參數仍然有效。
4. 將階段作業存入原始檔。 將現行的階段作業存入原始檔中。「變更原始檔」畫面將會出現，供您選取階段作業所要儲存的位置。您將無法在交談式 SQL 中回復並再次使用此階段作業。

註:

1. 選項 4 可讓您將原型化 SQL 陳述式內含在使用 SQL 的高階語言 (HLL) 程式中。請用原始檔登錄公用程式 (SEU) 將陳述式複製到您的程式中。原始檔還可在編輯後，作爲「執行 SQL 陳述式 (RUNSQLSTM)」指令的輸入原始檔。
2. 如果相關列已遭變更，且此工作單元目前已保留鎖定，而您嘗試跳出交談式 SQL，即會出現警告訊息。

使用現有的 SQL 階段作業

若您在「跳出交談式 SQL」畫面中，以選項 1 (儲存並跳出階段作業) 只儲存了一個交談式 SQL 階段作業，您即可在任何工作站回復該階段作業。不過，若您以選項 1 在不同工作站上儲存了兩個以上的階段作業時，交談式 SQL 會先嘗試回復與您工作站相符的階段作業。如果找不到相符的階段作業，交談式 SQL 將擴大搜尋範圍，併入屬於您使用者 ID 的所有階段作業。如果您的使用者 ID 並無任何階段作業，系統將會爲您的使用者 ID 與現行工作站建立新的階段作業。

例如，您在工作站 1 儲存了一個階段作業並在工作站 2 儲存了另一階段作業，而您目前正在使用工作站 1。交談式 SQL 會先嘗試回復針對工作站 1 所儲存的階段作業。如果該階段作業目前正在使用中，交談式 SQL 便會嘗試回復針對工作站 2 所儲存的階段作業。如果該階段作業也在使用，系統將會爲工作站 1 建立第二個階段作業。

不過，假設您正在使用工作站 3，而想要使用相關於工作站 2 的 ISQL 階段作業。您需先用「跳出交談式 SQL」畫面的選項 2 (跳出而不儲存階段作業) 來刪除工作站 1 的階段作業。

回復 SQL 階段作業

如果前一 SQL 階段作業異常結束，交談式 SQL 會在開始下一階段作業 (輸入下一個 STRSQL 指令) 時，帶出「回復 SQL 階段作業」畫面。透過此顯示畫面，您將可以：

- 藉由選取選項 1 (嘗試回復現有的 SQL 階段作業) 來回復舊的階段作業。
- 藉由選取選項 2 (刪除現有 SQL 階段作業並開始新的階段作業)，來刪除舊階段作業並開始新階段作業。

如果您選擇刪除舊的階段作業並繼續新的階段作業，系統將使用您在輸入 STRSQL 時所指定的參數。如果選擇回復舊的階段作業，或者您正在進入先前儲存的階段作業，您在輸入 STRSQL 時所指定的參數將被忽略而改爲使用舊階段作業的參數。此時，系統會傳回訊息，指出哪些參數已從指定值變更爲舊階段作業值。

以交談式 SQL 存取遠端資料庫

在交談式 SQL 中，您可以透過 SQL CONNECT 陳述式來與遠端的關聯式資料庫通信。交談式 SQL 會採用 CONNECT 的陳述式 CONNECT (類型 2) 語意 (分散式工作單元)。在開始 SQL 階段作業時，交談式 SQL 會對本端 RDB 進行隱含連接。當 CONNECT 陳述式完成時，會有訊息出現並顯示已建立的關聯式資料庫連線。若正在開始新的階段作業但並未指定 COMMIT(*NONE)，或者正在復置所儲存的階段作業但其確定層次並非 *NONE，該連線將會登記確定控制。此種隱含連接與可能的確定控制登記作業可能會影響對遠端資料庫的後續連線。相關詳細資訊，請參閱 第 312 頁的『決定連線類型』。建議您在連接遠端系統之前：

- 若是連接不支援分散式工作單元的應用程式伺服器，應發出 RELEASE ALL 再加上 COMMIT 來結束先前的連線，包括本端隱含連線。
- 連接非 DB2 UDB for iSeries 應用程式伺服器時，應發出 RELEASE ALL 再加上 COMMIT 來結束先前的連線 (包括本端隱含連線)，並將確定控制層次至少變更爲 *CHG。

當您正與非 DB2 UDB for iSeries 應用程式伺服器連線時，可將某些階段作業屬性變更爲該應用程式伺服器所支援者。下表所示即爲變更的屬性。

表 36. 值表格

階段作業屬性	原始值	新的值
日期格式	*YMD *DMY *MDY *JUL	*ISO *EUR *USA *USA
時間格式	含 : 分隔字元之 *HMS 含任何其他分隔字元之 *HMS	*JIS *EUR
確定控制	*CHG, *NONE *ALL	*CS 可重複讀取
命名慣例	*SYS	*SQL
允許複製資料	*NO, *YES	*OPTIMIZE
重新整理資料	*ALWAYS	*FORWARD
小數點	*SYSVAL	*PERIOD
排序順序	任何 *HEX 以外的值	*HEX

註:

1. 如果連接正執行版本 2 版次 3 之前版本的伺服器，排序順序值將變爲 *HEX。
2. 連接 DB2/2 或 DB2/6000 應用程式伺服器時，指定的日期格式與時間格式兩者須相同。

連線完成後，系統會發出訊息指出階段作業屬性已變更。變更的階段作業屬性可用階段作業服務畫面來顯示。當正在執行交談式 SQL 時，您將無法爲預設啓動群組建立其他連線。

若以交談式 SQL 連接遠端系統時，純語法的陳述式處理模式將會針對本端系統而非遠端系統所支援的陳述式進行語法檢查。同樣地，SQL 提示程式與清單支援亦會採用本端系統所支援的陳述式語法與命名慣例。不過，陳述式是在遠端系統上執行。由於兩種系統在 SQL 支援層次上可能會有差異，因此遠端系統上的陳述式在執行時間亦可能出現語法錯誤。

當您連接本端關聯式資料庫時，系統會提供綱目與表格的清單。直欄清單則唯有在您連接至支援 DESCRIBE TABLE 陳述式的關聯式資料庫管理程式後才會提供。

跳出具有連線的交談式 SQL 時，若連線帶具擱置中的變更或是使用受保護的交談，此連線將會保留。若未在連線上執行任何額外的作業，連線會在下一 COMMIT 或 ROLLBACK 作業期間結束。您也可藉由在跳出交談式 SQL 前執行 RELEASE ALL 與 COMMIT，來結束連線。

透過交談式 SQL 對非 DB2 UDB for iSeries 的應用程式伺服器進行遠端存取時將需要一些設定步驟。相關詳細資訊，請參閱 Distributed Database Programming 一書。

註: 在通信追蹤的輸出中，可能會存有對 CREATE TABLE XXX 陳述式的參照。此參照是用來決定資料包的存在性；它屬於正常處理的一部份，可加以忽略。

第 18 章 使用 SQL 陳述式處理器

這一節會說明 SQL 陳述式處理器。您使用執行 SQL 陳述式 (RUNSQLSTM) 指令時，就能使用處理器。

SQL 陳述式處理器允許從原始成員執行 SQL 陳述式。原始成員中的陳述式可以重複執行或變更，不必編譯原始程式碼。這可以更容易地設定資料庫環境。可以配合 SQL 陳述式處理器使用的陳述式有：

- ALTER TABLE
- CALL
- COMMENT ON
- COMMIT
- CREATE ALIAS
- CREATE DISTINCT TYPE
- CREATE FUNCTION
- CREATE INDEX
- CREATE PROCEDURE
- CREATE SCHEMA
- CREATE TABLE
- CREATE TRIGGER
- CREATE VIEW
- DECLARE GLOBAL TEMPORARY TABLE
- DELETE
- DROP
- GRANT (函數或程序專用權)
- GRANT (資料包專用權)
- GRANT (表格專用權)
- GRANT (使用者定義的類型專用權)
- INSERT
- LABEL ON
- LOCK TABLE
- RELEASE SAVEPOINT
- RENAME
- REVOKE (函數或程序專用權)
- REVOKE (資料包專用權)
- REVOKE (表格專用權)
- REVOKE (使用者定義的類型專用權)
- ROLLBACK
- SAVEPOINT

- SET PATH
- SET SCHEMA
- SET TRANSACTION
- UPDATE

原始成員中的陳述式以分號結束，而且不是以 **EXEC SQL** 開始。如果原始成員的記錄長度大於 80，只會讀取前 80 個字元。原始成員中的註解可以是行註解或區塊註解。行註解以雙連字號 (--) 開頭，在行尾結束。區塊註解以 /* 開頭，可以跨越許多行，直到下一個 */ 爲止。區塊註解可以爲巢狀。原始檔中只允許有 SQL 陳述式和註解。 SQL 陳述式的輸出報表和產生的訊息會傳送至列印檔。預設的列印檔是 QSYSPRT。

如果只要執行原始成員中所有陳述式的語法檢查，請在 RUNSQLSTM 指令上指定 PROCESS(*SYN) 參數。

如需詳細資訊，請參閱以下各節：

- 『發生錯誤後的陳述式執行』
- 『SQL 陳述式處理器中的確定控制』
- 『SQL 陳述式處理器中的綱目』
- 第 279 頁的『SQL 陳述式處理器的原始成員報表』

發生錯誤後的陳述式執行

陳述式傳回嚴重性高於爲 RUNSQLSTM 指令之錯誤層次 (ERRLVL) 參數指定之值的錯誤時，陳述式就會失敗。將會剖析來源中的其餘陳述式，檢查是否有語法錯誤，但是不會執行這些陳述式。大部份 SQL 錯誤的嚴重性爲 30。如果您要在 SQL 陳述式失敗後繼續處理程序，請將 RUNSQLSTM 指令的 ERRLVL 參數設定在 30 或 30 以上。

SQL 陳述式處理器中的確定控制

確定控制層次是指定於 RUNSQLSTM 指令上。如果指定 *NONE 以外的確定控制層次，SQL 陳述式會在確定控制下執行。如果所有的陳述式都順利執行，會在 SQL 陳述式處理器完成時執行 COMMIT。否則，將會執行 ROLLBACK。如果陳述式的回覆碼嚴重性小於或等於 RUNSQLSTM 指令之 ERRLVL 參數指定的值，陳述式就視爲成功。

可以在原始成員中使用 SET TRANSACTION 陳述式，以置換 RUNSQLSTM 指令所指定的確定控制層次。

註: 工作必須在工作單元界限內，才能使用 SQL 陳述式處理器配合確定控制。

SQL 陳述式處理器中的綱目

SQL 陳述式處理器支援 CREATE SCHEMA 陳述式。這是一個複雜的陳述式，可視爲擁有兩個不同的部分。第一部分定義綱目的集合。第二部分包含定義集合中之物件的 DDL 陳述式。

第一部分可利用兩種方法之一撰寫：

- CREATE SCHEMA *集合名稱*

  集合是使用指定的集合名稱所建立的。

- CREATE SCHEMA AUTHORIZATION 授權名稱

  集合是使用授權名稱當成集合名稱建立。執行綱目時，使用者必須擁有名爲**授權名稱**之使用者設定檔的權限。

  陳述式之授權名稱擁有的專用權必須包括：

  – 執行 CREATE COLLECTION 陳述式的權限

  – 執行 CREATE SCHEMA 中每一個 SQL 陳述式的權限

CREATE SCHEMA 陳述式的第二部分可以包含零到任意個下列陳述式：

- COMMENT ON
- CREATE ALIAS
- CREATE DISTINCT TYPE
- CREATE INDEX
- CREATE TABLE
- CREATE VIEW
- GRANT (表格專用權)
- GRANT (使用者定義的類型專用權)
- LABEL ON

這些陳述式緊接在陳述式的第一部分之後。陳述式和各部分**不**使用分號隔開。如果這個綱目定義之後有其他 SQL 陳述式，綱目中的最後一個陳述式必須以分號結束。

綱目陳述式第二部分中建立或參照的所有物件必須位於爲綱目建立的集合中。所有未限定的參照都會由建立的集合隱含地限定。所有限定的參照必須由建立的集合限定。

SQL 陳述式處理器的原始成員報表

如需程式碼範例的相關資訊，請參閱第 x 頁的『程式碼不保事項聲明』資訊。

```
5722ST1 V5R2M0 020719              執行 SQL 陳述式            綱目          08/06/02 15:35:18   頁   1
原始檔.....................CORPDATA/SRC
成員.......................SCHEMA
確定.......................*NONE
名稱.......................*SYS
產生層次...................10
日期格式...................*JOB
日期分隔字元...............*JOB
時間格式...................*HMS
時間分隔符號...............*JOB
預設集合...................*NONE
IBM SQL 旗號...............*NOFLAG
ANS 旗號...................*NONE
小數點.....................*JOB
排序順序...................*JOB
語言 ID....................*JOB
印表機檔案.................*LIBL/QSYSPRT
原始檔 CCSID...............65535
工作 CCSID.................0
陳述式處理程序.............*RUN
允許複製資料...............*OPTIMIZE
允許區塊...................*READ
原始成員變更日期 04/01/98  11:54:10
```

圖 8. SQL 陳述式處理器 QSYSPRT 報表 (1/3)

```
5722ST1 V5R2M0 020719              執行 SQL 陳述式            綱目                      08/06/02 15:35:18   頁   2
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  上次變更
    1
    2     DROP COLLECTION DEPT;
    3     DROP COLLECTION MANAGER;
    4
    5     CREATE SCHEMA DEPT
    6          CREATE TABLE EMP (EMPNAME CHAR(50), EMPNBR INT)
    7                           -- 會在集合 DEPT 中建立 EMP
    8          CREATE INDEX EMPIND ON EMP(EMPNBR)
    9                           -- 會在 DEPT 中建立 EMPIND
   10          GRANT SELECT ON EMP TO PUBLIC;  -- 授予權限
   11
   12     INSERT INTO DEPT/EMP VALUES('JOHN SMITH', 1234);
   13                           /* 表格已不在綱目中，
   14                              因此必須限定 */
   15
   16     CREATE SCHEMA AUTHORIZATION MANAGER
   17                           -- 這個綱目會使用 MANAGER 的
   18                           -- 使用者設定檔
   19           CREATE TABLE EMP_SALARY (EMPNBR INT, SALARY DECIMAL(7,2),
   20                                    LEVEL CHAR(10))
   21           CREATE VIEW LEVEL AS SELECT EMPNBR, LEVEL
   22                FROM EMP_SALARY
   23           CREATE INDEX SALARYIND ON EMP_SALARY(EMPNBR,SALARY)
   24
   25           GRANT ALL ON LEVEL TO JONES GRANT SELECT ON EMP_SALARY TO CLERK
   26                     -- 兩個陳述式可以放在同一行
* * * * *  原 始 程 式 碼 結 束  * * * * *
```

圖 8. SQL 陳述式處理器 QSYSPRT 報表 (2/3)

```
5722ST1 V5R2M0 020719              執行 SQL 陳述式            綱目                  08/06/02 15:35:18   頁     3
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  上次變更
MSG ID  SEV  RECORD  TEXT
SQL7953   0       1  位置 1 將 DEPT 放入 QSYS 完成。
SQL7953   0       3  位置 3 將 MANAGER 放入 QSYS 完成。
SQL7952   0       5  位置 3 建立集合 DEPT。
SQL7950   0       6  位置 8 在 DEPT 中建立表格 EMP。
SQL7954   0       8  位置 8 在 DEPT 中表格 EMP 的 DEPT 建立
                     索引 EMPIND。
SQL7966   0      10  位置 10 GRANT 權限給 DEPT 中的 EMP 已完成。
SQL7956   0      10  位置 40 將 1 列插入 DEPT 中的 EMP。
SQL7952   0      13  位置 28 建立集合 MANAGER。
SQL7950   0      19  位置 9 在售合 MANAGER 中建立表格
                     EMP_SALARY。
SQL7951   0      21  位置 9 在 MANAGER 中建立概略表 LEVEL。
SQL7954   0      23  位置 9 在 MANAGER 中表格 EMP_SALARY 的
                     MANAGER 建立索引 SALARYIND。
SQL7966   0      25  位置 9 GRANT 權限給 MANAGER 中的 LEVEL
                     已完成。
SQL7966   0      25  位置 37 GRANT 權限給 MANAGER 中的
                     EMP_SALARY 已完成。

訊息摘要
總計      資訊       警告       錯誤      嚴重性       終端機
   13      13         0          0          0           0
在來源中發現 00 個層次嚴重性錯誤
* * * * *  報 表 結 束  * * * * *
```

圖 8. SQL 陳述式處理器 QSYSPRT 報表 (3/3)

第 19 章 DB2 UDB for iSeries 資料保護

本章將說明用於保護 SQL 資料不受未授權使用者擅用的安全計劃，以及用來確保資料完整性的相關方法。相關詳細資訊，請參閱以下主題：

- 『SQL 物件的安全』
- 第 282 頁的『使用 iSeries 領航員保護資料』
- 第 284 頁的『資料完整性』

如需相關的程式碼範例，請參閱 第 x 頁的『程式碼不保事項聲明』 的資訊。

SQL 物件的安全

伺服器上的所有物件，包括 SQL 物件，都是由系統安全功能負責管理。使用者可透過 SQL GRANT 與 REVOKE 陳述式，或是「編輯物件權限 (EDTOBJAUT)」、「授予物件權限 (GRTOBJAUT)」及「撤回物件權限 (RVKOBJAUT)」等 CL 指令，來授權 SQL 物件。有關系統安全以及 GRTOBJAUT 與 RVKOBJAUT 指令的詳細資訊，請參閱 iSeries Security Reference 一書。

SQL 的 GRANT 與 REVOKE 陳述式可處理 SQL 資料包、SQL 程序、表格、概略表、以及表格與概略表上的個別直欄。此外，SQL GRANT 與 REVOKE 陳述式僅授予專用與公用權限。在某些情形下，則必須使用 EDTOBJAUT、GRTOBJAUT 及 RVKOBJAUT 來授權使用者使用其他物件，如指令與程式。

有關 GRANT 與 REVOKE 陳述式的詳細資訊，請參閱 SQL Reference 一書。

SQL 陳述式的權限檢查取決於陳述式究竟爲靜態、動態或是否以交談方式來執行。

對靜態 SQL 陳述式而言：

- 若 USRPRF 值爲 *USER，系統會透過執行程式之使用者的使用者設定檔，來檢查其是否具有在本端執行 SQL 陳述式的權限。至於在遠端執行 SQL 陳述式的權限，則是利用位於應用程式伺服器的使用者設定檔來加以檢查。 *USER 是系統 (*SYS) 名稱的預設值。
- 若 USRPRF 值爲 *OWNER，其在本端執行 SQL 陳述式的權限，是透過執行該程式之使用者以及該程式之擁有者的使用者設定檔來檢查。而在遠端執行 SQL 陳述式的權限，則是利用應用程式伺服器工作以及 SQL 資料包之擁有者兩者的使用者設定檔來檢查。較高的權限即爲所用的權限。*OWNER 是 SQL (*SQL) 名稱的預設值。

對動態 SQL 陳述式而言：

- 若 USRPRF 值爲 *USER，其在本端執行 SQL 陳述式的權限，是透過執行程式之人員的使用者設定檔來檢查。至於在遠端執行 SQL 陳述式的權限，則是利用應用程式伺服器工作的使用者設定檔來檢查。
- 若 USRPRF 值爲 *OWNER 且 DYNUSRPRF 爲 *USER，其在本端執行 SQL 陳述式的權限，是透過執行程式之人員的使用者設定檔來檢查。至於在遠端執行 SQL 陳述式的權限，則是利用應用程式伺服器工作的使用者設定檔來檢查。

- 若 USRPRF 值爲 *OWNER 且 DYNUSRPRF 爲 *OWNER，其在本端執行 SQL 陳述式的權限，是透過執行程式之人員以及該程式之擁有者的使用者設定檔來檢查。而在遠端執行 SQL 陳述式的權限，則是利用應用程式伺服器工作以及 SQL 資料包之擁有者兩者的使用者設定檔來檢查。最高的權限即爲所用的權限。基於安全考量，您應謹慎使用 DYNUSRPRF 的 *OWNER 參數值。此選項會將擁有者之程式或資訊包的存取權限，授予執行該程式之人。

對於交談式 SQL 陳述式而言，系統是根據處理該陳述式之人員的權限來進行權限檢查。所採用的權限不會被用於交談式 SQL 陳述式。

授權 ID

授權 ID 可定義唯一的使用者，其本身是伺服器上的一種使用者設定檔物件。授權 ID 可透過系統的「建立使用者設定檔 (CRTUSRPRF)」指令來建立。

概略表

概略表可防止未經授權的使用者存取敏感資料。它可讓應用程式在表格中直接存取其所需的資料，而不必碰觸敏感或受限制的資料。概略表可藉由不在 SELECT 清單中指定相關直欄 (例如，員工薪資)，進而限制對特定直欄的存取。它還可藉由指定 WHERE 子句，來限制對特定表格列的存取 (例如，只允許存取與特定部門編號相關的列)。

審核

DB2 UDB for iSeries 在設計上完全合乎美國政府機關 C2 安全層次。此層次的主要特性是具有審核系統上相關動作的能力。 DB2 UDB for iSeries 所用的審核機能是由系統安全功能來負責管理。其審核作業可在物件層次、使用者或系統層次上執行。系統值 QAUDCTL 會控制審核作業是在物件層次或使用者層次上執行。「變更使用者審核 (CHGUSRAUD)」指令與「變更物件審核 (CHGOBJAUD)」指令可指定哪些使用者與物件須受審核。系統值 QAUDLVL 則控制哪種類型的動作會受到審核 (例如，授權失敗、建立、刪除、授予、撤回等)。有關審核的進一步資訊，請參閱 iSeries Security Reference


一書。
DB2 UDB for iSeries 亦可藉由使用 DB2 UDB for iSeries 日誌登載支援來審核列的變更。

在某些情況下，審核日誌中的登錄其順序可能會與記載時不同。以某一在確定控制下執行的工作爲例，若其先刪除某表格，接著建立與所刪表格同名的新表格，然後執行確定。但此過程在審核日誌中會被記錄爲先是建立然後才是刪除動作。這是因爲新建立的物件會立即記錄到日誌中。而在確定控制下遭刪除的物件，則是先被隱藏起來，直到確定完成才實際加以刪除。一旦完成確定，此動作即會記錄到日誌中。

使用 iSeries 領航員保護資料

您可以藉由下列方式透過 iSeries 領航員來保護您的資料：

- 第 283 頁的『定義物件的公用權限』
- 第 283 頁的『設置新物件的預設公用權限』
- 第 283 頁的『授權使用者或群組使用物件』

定義物件的公用權限

系統會爲其上的每個物件定義公用權限，用以說明使用者對特定物件所擁有的存取權限類型。若要定義公用權限：

1. 在 **iSeries 領航員** 視窗中，展開您的伺服器 → **資料庫** → 您要處理的資料庫 → **檔案庫**。
2. 進行導覽，直至出現您要編輯其許可權的物件。
3. 在想要新增許可權的物件上按一下右鍵，然後選取**許可權**。
4. 在**許可權**對話框上，從群組清單中選取**公用**。
5. 按一下**明細**按鈕，以實施詳細的許可權。
6. 藉由選取適當的勾選框，來套用所要的公用許可權。
7. 按一下**確定**。

設置新物件的預設公用權限

設定預設公用權限可讓您針對檔案庫中建立的所有新物件指定通用的權限。之後，您可以針對需要不同安全層次的個別物件來編輯許可權。若要設定預設公用權限：

1. 在 **iSeries 領航員** 視窗中，展開您的伺服器 → **資料庫** → 您要處理的資料庫 → **檔案庫**。
2. 在想要設定公用權限的檔案庫上按一下右鍵，然後選取**許可權**。
3. 在**許可權**對話框中，按一下**新物件**。
4. 在**新物件**對話框中，選取預設公用權限。若要指派「授權清單」，您可以輸入或**瀏覽**授權清單的名稱。若要檢視「授權清單」的內容，請選取**開啓**。
5. 按一下**確定**。

授權使用者或群組使用物件

部份使用者可能會需要不同於「公用權限」所允許的物件權限。若要將物件權限授予使用者或群組：

1. 在 **iSeries 領航員** 視窗中，展開您的伺服器 → **資料庫** → 您要處理的資料庫 → **檔案庫**。
2. 進行導覽，直至出現您要編輯其許可權的物件。
3. 在想要新增許可權的物件上按一下右鍵，然後選取**許可權**。
4. 在**許可權**對話框中，按一下**新增**。
5. 在**新增**對話框中選取一或多個使用者與群組，或是在使用者或群組欄位中輸入其名稱。
6. 按一下**確定**。

如此即會將這些使用者或群組新增至清單的頂端。

註: 使用者或群組最初是獲得物件的預設權限。您可以將其權限變更爲系統所定義的權限類型之一，或者自訂相關權限。

若要移除使用者對物件的權限：

1. 選取要移除權限的使用者或群組。
2. 按一下**移除**。

資料完整性

資料完整性可保護資料不受未授權人員、系統作業或硬體故障 (如磁碟的實體損壞)、程式設計錯誤、工作完成前的岔斷 (如斷電) 或同時執行多支應用程式造成的干擾 (如序列化問題) 所破壞或變更。資料完整性可藉下列功能來確保：

- 『並行』
- 第 285 頁的『日誌登載』
- 第 286 頁的『確定控制』
- 第 291 頁的『原子作業』
- 第 293 頁的『限制』
- 第 293 頁的『儲存/復置』
- 第 294 頁的『損壞容差』
- 第 294 頁的『索引回復』
- 第 295 頁的『編目完整性』
- 第 295 頁的『使用者輔助儲存體儲存區 (ASP)』
- 第 295 頁的『獨立式輔助儲存體儲存區 (IASP)』

有關此類功能的進一步詳情，請參閱確定控制主題、日誌管理主題、 Database Programming 一書及備份及回復 一書。

並行

並行是一種讓多重使用者同時存取及變更相同表格或概略表之資料而不會喪失資料完整性的能力。此能力是由 DB2 UDB for iSeries 資料庫管理程式自動提供。它會將鎖定功用隱含地加諸於表格與列上，藉以防止並行使用者同時變更相同的資料。

一般而言，DB2 UDB for iSeries 會對資料列設下鎖定以確保完整性。不過，有些情況會使 DB2 UDB for iSeries 採行更專用的表格層次鎖定，而不僅是列鎖定。相關詳細資訊，請參閱第 286 頁的『確定控制』。

例如，某一游標目前對某列所保有的更新 (專用) 鎖定，可由同一程式中 (或在未與此游標相關的 DELETE 或 UPDATE 陳述式中) 的另一游標加以取得。如此可防止已定位 UPDATE 或已定位 DELETE 陳述式參照第一個游標，直到執行另一 FETCH 爲止。但某一游標目前對某列保有的讀取 (共用且不更新) 鎖定，將無法防止同一程式中 (或 DELETE 或 UPDATE 陳述式) 的另一游標取得該列的鎖定。

預設的與使用者可指定的鎖定等待逾時值都已獲支援。DB2 UDB for iSeries 所建立的表格、概略表及索引可具有預設記錄等待時間 (60 秒)，以及預設檔案等待時間 (*IMMED)。此種鎖定等待時間是供 DML 陳述式使用。您可以透過「變更實體檔案 (CHGPF)」、「變更邏輯檔案 (CHGLF)」及「置換資料庫檔案 (OVRDBF)」等 CL 指令來變更這些值。

DDL 陳述式與 LOCK TABLE 陳述式所用的的鎖定等待時間，即爲工作預設等待時間 (DFTWAIT)。您可以利用 CL 指令「變更工作 (CHGJOB)」或「變更類別 (CHGCLS)」來變更此值。

若已指定大型記錄等待時間，死結偵測作業即會展開。例如，假設某一工作對第 1 列具有專用鎖定，而另一工作則在第 2 列具有專用鎖定。若第一個工作嘗試鎖定第 2 列就

必須等待，因爲第二個工作仍保有鎖定。若第二個工作接著嘗試鎖定第 1 列，DB2 UDB for iSeries 將偵測到這兩個工作已呈死結狀態，因此會傳回錯誤給第二個工作。

您可以透過 SQL LOCK TABLE 陳述式 (詳見 SQL Reference 一書)，來明確防止其他使用者同時使用某一表格。使用 COMMIT(*RR) 亦可防止其他使用者在進行工作單元時使用表格。

爲了增進效能，DB2 UDB for iSeries 會經常將開啓的資料路徑 (ODP) 保持開啓 (相關詳情，請參閱 Database Performance and Query Optimization 一節)。此種效能特性雖也會對 ODP 所參照的表格設置鎖定，但不會對任何列設下鎖定。表格鎖定可防止另一工作對此表格執行作業。但在多數情形下，DB2 UDB for iSeries 會偵測到其他工作亦保有鎖定，因此會對這些工作發出事件。此事件將使 DB2 UDB for iSeries 關閉任何與該表格相關且目前僅因效能原因而開啓的 ODP (並釋放表格鎖定)。請注意，鎖定等待逾時長度應足夠，以使事件可以送出並讓其他工作得以關閉 ODP，否則將會傳回錯誤。

除非是使用 LOCK TABLE 陳述式來取得表格鎖定，或是使用 COMMIT(*ALL) 或 COMMIT(*RR)，否則某一工作所讀取的資料可能會立即由另一工作所變更。一般而言，資料多是在執行 SQL 陳述式時才會讀取，因此相當具即時性 (例如，進行 FETCH 時)。但在下述情形中，資料是在執行 SQL 陳述式之前所讀取，因此可能不具有即時性 (例如，進行 OPEN 時)。

- 指定了 ALWCPYDTA(*OPTIMIZE) 且最佳化工具研判了複製資料的作法其效能較高。
- 有些查詢要求資料庫管理程式建立暫時的結果表格。但暫時結果表格中的資料無法反應游標開啓後所作的變更。下述情況將需要暫時結果表格：
  - ORDER BY 子句中所指定的直欄儲存體總長度 (以位元組爲單位) 超過 2000 個位元組。
  - ORDER BY 與 GROUP BY 子句指定了不同直欄或順序有異的直欄。
  - 指定了 UNION 或 DISTINCT 子句。
  - ORDER BY 或 GROUP BY 子句指定了並非來自相同表格的直欄。
  - 將 JOINDFT 資料定義規格 (DDS) 關鍵字所定義的邏輯檔案結合至另一檔案。
  - 結合了以多重資料庫檔案成員爲依據的邏輯檔案或對其指定了 GROUP BY。
  - 查詢所含的結合中至少有一檔案爲內含 GROUP BY 子句的概略表。
  - 查詢中含有 GROUP BY 子句，且此子句參照了內含 GROUP BY 子句的概略表。
- 開啓查詢時，評估了基本子查詢。

日誌登載

DB2 UDB for iSeries 日誌支援提供審核追蹤與向前及向後回復。向前回復可供您取用舊版本的表格，並將登載於日誌中的變更內容套用至該表格。向後回復則可用來在表格中移除日誌所登載的變更內容。

建立 SQL 綱目時，綱目中亦會建立一日誌及異動日誌接收器。當 SQL 建立日誌與異動日誌接收器時，若已在 CREATE COLLECTION 或 CREATE SCHEMA 陳述式指定了 ASP 子句，此日誌與異動日誌接收器將只在使用者輔助儲存體儲存區 (ASP) 中建立。不過，由於在本身的 ASP 上設置異動日誌接收器可提升效能，因此日誌管理人員可能會想花個別的 ASP 上建立未來所有的異動日誌接收器。

當表格建立於綱目中時，DB2 UDB for iSeries 在此綱目中所建立的日誌 (QSQJRN) 即會自動加以登載。若檔案庫中存有名爲 QSQJRN 的日誌，於非綱目中所建立的表格亦

會進行日誌登載。除上述者外，您應負責透過日誌功能來管理日誌、異動日誌接收器及表格日誌登載作業。例如，若一表格移入到綱目中，其日誌登載狀態將不會自動變更。若復置了表格，所套用的將是正常的日誌登載規則。也就是說，若表格是在儲存時進行日誌登載，其在復置時也是登載到相同日誌中。若表格未在儲存時進行日誌登載，其在復置時將不會被登載。

於 SQL 集合中所建立的日誌，通常即爲用於登載所有 SQL 表格變更內容的日誌。不過，您可以用系統日誌功能，將 SQL 表格登載到不同的日誌中。若某一綱目中的表格是另一綱目中之表格的上層，即可能需要採用此作法。這是因爲 DB2 UDB for iSeries 規定參照限制中若對上層表格執行更新或刪除時，上層與相依檔須登載到相同的日誌中。

使用者可藉日誌功能來停止任何表格的日誌登載，但此作法將使應用程式無法在確定控制下執行。若對含有 NO ACTION、CASCADE、SET NULL 或 SET DEFAULT 等刪除規則之參照限制的上層表格停止了日誌登載，所有的更新與刪除作業將被阻止。但若您指定了 COMMIT(*NONE)，應用程式仍可繼續作用；不過，此情況下的完整性層次將不及日誌登載與確定控制所提供者。

有關日誌登載的進一步資訊，請參閱日誌登載主題。

確定控制

DB2 UDB for iSeries 確定控制支援可提供有效的手段，將成群的資料庫變更 (更新、插入、DDL 作業或刪除) 視爲單一工作單元來處理。確定作業可保證作業群組可以順利完成。回轉作業則可保證作業群組可順利倒退。儲存點可用來將異動分成較小的單位，以利於回轉。確定作業可透過數種不同介面來發出。例如，

- SQL COMMIT 陳述式
- CL COMMIT 指令
- 語言確定陳述式 (例如 RPG COMMIT 陳述式)

回轉作業亦可透過數種不同介面來發出。例如，

- SQL ROLLBACK 陳述式
- CL ROLLBACK 指令
- 語言回轉陳述式 (例如 RPG ROLBK 陳述式)

唯一無法確定或回轉的 SQL 陳述式是：

- DROP COLLECTION
- GRANT 或 REVOKE，若指定物件的權限儲存器存在

用 COMMIT(*NONE) 以外的隔離層次執行 SQL 陳述式，或執行 RELEASE 陳述式時，若確定控制尚未啓動， DB2 UDB for iSeries 會藉由隱含地呼叫 CL 指令「啓動確定控制 (STRCMTCTL)」來設定確定控制環境。 DB2 UDB for iSeries 將在 STRCMTCTL 指令中，指定 NFYOBJ(*NONE) 與 CMTSCOPE(*ACTGRP) 參數以及 LCKLVL。所指定的 LCKLVL 即爲 CRTSQLxxx、STRSQL 或 RUNSQLSTM 等指令的 COMMIT 參數鎖定層次。在 REXX 中，所指定的 LCKLVL 則爲 SET OPTION 陳述式中的鎖定層次。您可以用 STRCMTCTL 指令來指定不同的 CMTSCOPE、NFYOBJ 或 LCKLVL。若是指定 CMTSCOPE(*JOB) 來啓動工作層次確定定義， DB2 UDB for iSeries 會將此工作層次確定定義用於該啓動群組中的程式。

註:

1. 使用確定控制時，「資料操作語言」陳述式在應用程式中所參考的表格必須作日誌登載。
2. 請注意，所指定的 LCKLVL 僅爲預設鎖定層次。待啓動確定控制後，SET TRANSACTION SQL 陳述式以及在 CRTSQLxxx、STRSQL 或 RUNSQLSTM 指令之 COMMIT 參數中指定的鎖定層次將置換預設的鎖定層次。

對使用 GROUP BY 或 HAVING 等直欄功能的以及在確定控制下執行的游標而言，ROLLBACK HOLD 對游標位置並無影響。此外，下述情況亦會發生在進行確定控制時：

- 若已針對這些游標之一指定了 COMMIT(*CHG) 與 (ALWBLK(*NO) 或 (ALWBLK(*READ))，系統即會送出訊息 (CPI430B) 指出已要求 COMMIT(*CHG) 但未獲准。
- 若已針對游標之一指定了帶有 KEEP LOCKS 子句的 COMMIT(*ALL)、COMMIT(*RR) 或 COMMIT(*CS)， DB2 UDB for iSeries 將鎖定所有在共用模式 (*SHRNUP) 下被參考的表格。此鎖定可防止並行應用程式處理對指定表格執行唯讀以外的任何作業。其間會有一訊息 (SQL7902 或 CPI430A) 傳送，指出已針對所要求的游標之一指定了帶有 KEEP LOCKS 子句的 COMMIT(*ALL)、COMMIT(*RR) 或 COMMIT(*CS) 但不被允許。也可能傳送出訊息 SQL0595。

對於已指定了帶有 KEEP LOCKS 子句之 COMMIT(*ALL)、COMMIT(*RR) 或 COMMIT(*CS) 的游標，以及已使用編目檔或需要暫時結果表格的游標，DB2 UDB for iSeries 會鎖定在共用模式 (*SHRNUP) 下被參考到的所有表格。如此將可防止並行處理對表格執行唯讀以外的任何作業。其間會傳送出一訊息 (SQL7902 或 CPI430A)，指出已要求了 COMMIT(*ALL) 但不被允許。也可能傳送出訊息 SQL0595。

若已指定 ALWBLK(*ALLREAD) 與 COMMIT(*CHG)，當程式經過前置編譯後，所有唯讀游標將允許進行列鎖定，且 ROLLBACK HOLD 將無法回轉游標定位。

若已要求 COMMIT(*RR)，表格將被持續鎖定，直到查詢關閉爲止。若游標是唯讀，表格將被鎖定 (*SHRNUP)。若游標是在更新模式下，表格將被鎖定 (*EXCLRD)。由於其他使用者將無法使用該表格，此時執行可重複讀取將能防止並行存取該表格。

若已指定 COMMIT(*NONE) 以外的隔離層次，且應用程式發出了 ROLLBACK 或啓動群組異常結束 (且確定定義定義並非 *JOB)，在工作單元中所作的任何更新、插入、刪除及 DDL 作業都會被倒退。若應用程式發出了 COMMIT 或啓動群組正常結束，在工作單元中所作的任何更新、插入、刪除及 DDL 作業都會被確定。

DB2 UDB for iSeries 會藉由列鎖定來阻止其他工作在工作單元完成前存取已變更的資料。若已指定 COMMIT(*ALL)，對提取列所設的讀取鎖定，亦可用來在工作單元完成前防止其他工作變更已讀取的資料。但此作法無法防止其他工作讀取未變更的列。如此可確保當相同工作單元重新讀取某列時，所得的結果亦相同。讀取鎖定並無法防止其他工作提取相同的列。

確定控制最多可在一個工作單元中處理 4 百萬個獨特的列變更。若已指定 COMMIT(*ALL) 或 COMMIT(*RR)，此上限將會計入所有列讀取作業。 (若某列在一工作單元中被變更或讀取一次以上，其在限額中將只計列一次。) 保留大量的鎖定將會

影響系統效能，而且在工作單元結束前，將不允許並行使用者存取工作單元中已鎖定的列。因此在工作單元中僅處理少量的列將最符效益。

確定控制最多允許每一日誌可有 512 個檔案在確定控制下開啓，或最可關閉 512 個檔案 (但工作單元中仍含有擱置中的變更)。

COMMIT HOLD 與 ROLLBACK HOLD 可讓您將游標保持開啓，並啓動另一工作單元而不會再次發出 OPEN。當您連接不在 iSeries 系統上的遠端資料庫時，將無 HOLD 值可用。不過，DECLARE CURSOR 的 WITH HOLD 選項可用來在 COMMIT 之後保持游標開啓。當您連接不在 iSeries 系統上的遠端資料庫時，即會支援此類型的游標。此種游標會在回轉時關閉。

表 37. 列鎖定持續時間

SQL 陳述式	COMMIT 參數 (請參閱附註 6)	列鎖定的持續時間	鎖定類型
SELECT INTO	*NONE	未鎖定	
SET 變數	*CHG	未鎖定	
VALUES INTO	*CS (請參閱附註 8)	讀取與釋放時列遭鎖定	READ
	*ALL (請參閱附註 2)	從讀取直到 ROLLBACK 或 COMMIT	READ
FETCH (唯讀游標)	*NONE	未鎖定	
	*CHG	未鎖定	
	*CS (請參閱附註 8)	從讀取直到下一 FETCH	READ
	*ALL (請參閱附註 2)	從讀取直到 ROLLBACK 或 COMMIT	READ
FETCH (更新或刪除可用游標) (請參閱附註 1)	*NONE	當未更新或刪除列時 從讀取直到下一 FETCH 當已更新或刪除列時 從讀取直到 UPDATE 或 DELETE	UPDATE
	*CHG	當未更新或刪除列時 從讀取直到下一 FETCH 當已更新或刪除列時 從讀取直到 COMMIT 或 ROLLBACK	UPDATE
	*CS	當未更新或刪除列時 從讀取直到下一 FETCH 當已更新或刪除列時 從讀取直到 COMMIT 或 ROLLBACK	UPDATE
	*ALL	從讀取直到 ROLLBACK 或 COMMIT	UPDATE[3]
INSERT (目標表格)	*NONE	未鎖定	
	*CHG	從插入直到 ROLLBACK 或 COMMIT	UPDATE
	*CS	從插入直到 ROLLBACK 或 COMMIT	UPDATE
	*ALL	從插入直到 ROLLBACK 或 COMMIT	UPDATE[4]
INSERT (子選擇中的表格)	*NONE	未鎖定	
	*CHG	未鎖定	
	*CS	每一列在進行讀取時遭鎖定	READ
	*ALL	從讀取直到 ROLLBACK 或 COMMIT	READ
UPDATE (無游標)	*NONE	每一列在進行更新時遭鎖定	UPDATE
	*CHG	從讀取直到 ROLLBACK 或 COMMIT	UPDATE
	*CS	從讀取直到 ROLLBACK 或 COMMIT	UPDATE
	*ALL	從讀取直到 ROLLBACK 或 COMMIT	UPDATE

表 37. 列鎖定持續時間 (繼續)

SQL 陳述式	COMMIT 參數 (請參閱附註 6)	列鎖定的持續時間	鎖定類型
DELETE (無游標)	*NONE	每一列在進行刪除時遭鎖定	UPDATE
	*CHG	從讀取直到 ROLLBACK 或 COMMIT	UPDATE
	*CS	從讀取直到 ROLLBACK 或 COMMIT	UPDATE
	*ALL	從讀取直到 ROLLBACK 或 COMMIT	UPDATE
UPDATE (含游標)	*NONE	更新列時即釋放鎖定	UPDATE
	*CHG	從讀取直到 ROLLBACK 或 COMMIT	UPDATE
	*CS	從讀取直到 ROLLBACK 或 COMMIT	UPDATE
	*ALL	從讀取直到 ROLLBACK 或 COMMIT	UPDATE
DELETE (含游標)	*NONE	刪除列時即釋放鎖定	UPDATE
	*CHG	從讀取直到 ROLLBACK 或 COMMIT	UPDATE
	*CS	從讀取直到 ROLLBACK 或 COMMIT	UPDATE
	*ALL	從讀取直到 ROLLBACK 或 COMMIT	UPDATE
子查詢 (更新或刪除可用游標或 UPDATE 或 DELETE 無游標)	*NONE	從讀取直到下一 FETCH	READ
	*CHG	從讀取直到下一 FETCH	READ
	*CS	從讀取直到下一 FETCH	READ
	*ALL (請參閱附註 2)	從讀取直到 ROLLBACK 或 COMMIT	READ
子查詢 (唯讀游標或 SELECT INTO)	*NONE	未鎖定	
	*CHG	未鎖定	
	*CS	每一列在進行讀取時遭鎖定	READ
	*ALL	從讀取直到 ROLLBACK 或 COMMIT	READ

附註:

1. 若結果表格並非唯讀 (請參閱 SQL Reference 一書中的 DECLARE CURSOR 說明) 且以下其中一項爲眞時，游標即會以 UPDATE 或 DELETE 功能開啓：
   - 已使用 FOR UPDATE 子句定義了游標。
   - 定義游標時未用 FOR UPDATE、FOR READ ONLY 或 ORDER BY 子句，且程式至少含有下列其中一項：
     - 參考相同游標名稱的 UPDATE 游標
     - 參考相同游標名稱的 DELETE 游標
     - 於 CRTSQLxxx 指令中指定了 EXECUTE 或 EXECUTE IMMEDIATE 陳述式與 ALWBLK(*READ) 或 ALWBLK(*NONE)。
2. 可專門鎖定表格或概略表以滿足 COMMIT(*ALL)。如果所處理的子選擇含有 UNION，或在查詢的處理過程中須用到暫時結果，此時即會設置專用鎖定來防止您看到未經確定的變更。
3. 若未更新或刪除該列，鎖定即降爲 *READ。
4. 對目標表格的列設下 UPDATE 鎖定，且對子選擇表格的列設下 READ 鎖定。
5. 可專門鎖定表格或概略表以滿足可重複讀取。列鎖定仍在可重複讀取下完成。所取得的鎖定及其持續時間與 *ALL 相同。
6. 可重複讀取 (*RR) 列鎖定將相同於針對 *ALL 所指明的鎖定。
7. 有關隔離層次與鎖定的詳情，請參閱 SQL Reference 一書中的 Isolation Level。
8. 若以 *CS 指定了 KEEP LOCKS 子句，任何讀取鎖定皆會保留，直到游標關閉或完成 COMMIT 或 ROLLBACK 爲止。若無游標與隔離子句相關，則鎖定將會保留，直到 SQL 陳述式完成爲止。

相關明細，請參閱確定控制主題。

儲存點

儲存點可讓您在異動中建立管制點。若異動進行回轉，變更將還原到指定的儲存點，而非還原到異動的開頭。儲存點可用 SAVEPOINT SQL 陳述式來設定。例如，建立一名爲 STOP_HERE 的儲存點：

```
SAVEPOINT STOP_HERE
ON ROLLBACK RETAIN CURSORS
```

應用程式的程式邏輯會指明應用程式在進行過程中是否可重覆使用儲存點名稱，或儲存點名稱是否代表不得重覆使用的唯一管制點。

若要讓儲存點是不得用另一 SAVEPOINT 陳述式來移動的唯一的管制點，可指定 UNIQUE 關鍵字。如此可防止因呼叫了會使用與 SAVEPOINT 陳述式中相同儲存點名稱之儲存程序，而可能意外發生的名稱重覆使用情形。不過，若已在迴圈中使用了 SAVEPOINT 陳述式，則不應使用 UNIQUE 關鍵字。下列 SQL 陳述式會設定名爲 START_OVER 的唯一儲存點。

```
SAVEPOINT START_OVER UNIQUE
       ON ROLLBACK RETAIN CURSORS
```

若要回轉儲存點，可使用 ROLLBACK 陳述式加上 TO SAVEPOINT 子句。下述範例將示範如何使用 SAVEPOINT 與 ROLLBACK TO SAVEPOINT 陳述式：

此應用程式邏輯會在所要的日期預訂航空公司機位，然後預訂旅館房間。如果旅館沒有空房，它會回轉預訂機位，然後對另一日期重覆預約處理。其間最多可嘗試 3 個日期。

```
got_reservations =0;
EXEC SQL SAVEPOINT START_OVER UNIQUE ON ROLLBACK RETAIN CURSORS;

if (SQLCODE != 0) return;

for (i=0; i<3 & got_reservations == 0; ++i)
{
  Book_Air(dates(i), ok);
  if (ok)
  {
    Book_Hotel(dates(i), ok);
    if (ok) got_reservations = 1;
    else
    {
      EXEC SQL ROLLBACK TO SAVEPOINT START_OVER;
      if (SQLCODE != 0) return;
    }
  }
}

EXEC SQL RELEASE SAVEPOINT START_OVER;
```

儲存點可用 RELEASE SAVEPOINT 陳述式來釋放。若未使用 RELEASE SAVEPOINT 陳述式來明確釋放儲存點，它會在現行儲存點層次的末尾或異動末尾處釋放。下列陳述式將會釋放儲存點 START_OVER。

```
RELEASE SAVEPOINT START_OVER
```

儲存點是在異動已確定或回轉時釋放。一旦儲存點名稱被釋放後，即不可能再回轉到此儲存點名稱。 COMMIT 或 ROLLBACK 陳述式可釋放異動中所建立的全部儲存點名稱。由於所有儲存點名稱已在異動中釋放，因此所有儲存點名稱在確定或回轉作業後可重覆使用。

儲存點的層次

單一陳述式可隱含地或明確地呼叫使用者定義的函數、觸發程式或儲存程序。此即稱爲巢狀化。在某些情形下，若啓用新的巢狀層次，新的儲存點層次亦會同時啓用。新的儲存點層次會將呼叫方應用程式，與較低層常式或觸發程式的任何儲存點活動相隔離。

儲存點只能在其被定義的同一儲存點層次 (或範圍) 中被參考。 ROLLBACK TO SAVEPOINT 陳述式無法用來回轉至建立於現行儲存點層次外的儲存點。同理，RELEASE SAVEPOINT 陳述式亦無法用於釋放建立於現行儲存點層次外的儲存點。下表將彙總儲存點層次的啓用與終止時機：

何時啓用新的儲存點層次：	何時終止此儲存點層次：
啓動新的工作單元	發出 COMMIT 或 ROLLBACK
已呼叫觸發程式	觸發程式完成
已呼叫使用者定義的函數	使用者定義的函數回報呼叫程式
已呼叫儲存程序，且該儲存程序是用 NEW SAVEPOINT LEVEL 子句來建立	儲存程序回報呼叫程式
ATOMIC 複合 SQL 陳述式帶有 BEGIN	ATOMIC 複合陳述式帶有 END

於某一儲存點層次中建立的儲存點，會在該儲存點層次終止時暗中釋放。

關於分散式資料庫環境中之儲存點的注意事項

儲存點只限在單一連線中使用。一旦儲存點建立後，它並不會分散至應用程式所連接的各個遠端資料庫。儲存點僅適用於當其建立之時應用程式所連接的現行資料庫。

原子作業

所有作業於 COMMIT(*CHG)、COMMIT(*CS) 或 COMMIT(*ALL) 下執行時，保證如原子一般。亦即，當其完成時或進行中都有如並未啓動，而是默默動作。而不論函數是何時或如何結束或岔斷 (例如斷電、工作異常結束或工作取消)，情況都是如此。

不過，若已指定 COMMIT (*NONE)，某些基礎資料庫的資料定義函數則無法形同原子。下述 SQL 資料定義陳述式保證具原子特性：

ALTER TABLE (請參閱附註 1)

COMMENT ON (請參閱附註 2)

LABEL ON (請參閱附註 2)

GRANT (請參閱附註 3)

REVOKE (請參閱附註 3)

DROP TABLE (請參閱附註 4)

DROP VIEW (請參閱附註 4)

DROP INDEX

DROP PACKAGE

註:

1. 若需新增或移除限制，以及變更直欄定義，各項作業將會逐一處理，因此整個 SQL 陳述式即不同於原子。相關作業的順序如下：

- 移除限制
- 捨棄已指定 RESTRICT 選項的直欄
- 所有其他直欄定義的變更 (DROP COLUMN CASCADE、ALTER COLUMN、ADD COLUMN)
- 新增限制

2. 若對 COMMENT ON 或 LABEL ON 陳述式指定了多重直欄，各直欄將逐一處理，因此整個 SQL 陳述式即不具原子特性，但若是對 COMMENT ON 或 LABEL ON 指定個別直欄或物件則如同原子。
3. 若對 GRANT 或 REVOKE 陳述式指定了多重表格、SQL 資料包或使用者，由於各表格將逐一處理，因此整個 SQL 陳述式即不具原子特性，但若是對 GRANT 或 REVOKE 指定了個別表格則如同原子。
4. 進行 DROP TABLE 或 DROP VIEW 時若須捨棄相依概略表，各相依概略表將會逐一處理，因此整個陳述式即不具原子特性。

下述資料定義陳述式並不具原子特性，因爲它們涉及一個以上的 DB2 UDB for iSeries 資料庫作業：

CREATE ALIAS

CREATE DISTINCT TYPE

CREATE FUNCTION

CREATE INDEX

CREATE PROCEDURE

CREATE SCHEMA

CREATE TABLE

CREATE TRIGGER

CREATE VIEW

DROP ALIAS

DROP DISTINCT TYPE

DROP FUNCTION

DROP PROCEDURE

DROP SCHEMA

DROP TRIGGER

RENAME (請參閱附註 1)

註:

1. RENAME 唯有在名稱或系統名稱變更時才具原子性。當兩者皆變更時，RENAME 即不具原子性質。

例如，CREATE TABLE 可在 DB2 UDB for iSeries 實體檔案建立後予以岔斷，但必須是在新增成員之前。因此，遇到建立陳述式時，若作業異常結束，您可能須先捨棄物件，然後再次加以建立。若是 DROP COLLECTION 陳述式，您將須再次捨棄集合，或用 CL 指令「刪除資料庫 (DLTLIB)」來移除該集合的其餘部份。

限制

DB2 UDB for iSeries 支援有唯一限制、參照限制及核對限制。唯一限制是用來保證索引鍵值具有唯一性的規則。參照限制是用於規範相依表格中所有非空值之外來索引鍵，須在上層表格中具有對應上層鍵。核對限制則是用於限制直欄或直欄群組中之容許值的規則。

DB2 UDB for iSeries 會在執行任何 DML (資料操作語言) 陳述式期間，強制施行限制的效力。但某些作業 (例如復置相依表格) 會使限制的效力變成不明。此時，DML 陳述式可能會無法執行，除非 DB2 UDB for iSeries 已驗證了限制的效力。

- 唯一限制是以索引來建置。若用於建置唯一限制的索引無效，可用「編輯存取路徑的重新建置 (EDTRBDAP)」指令來顯示任何目前須重建的索引。
- 若 DB2 UDB for iSeries 目前並不曉得某一參照限制或核對限制是否有效，此限制將被視爲處在核對擱置之狀態。「編輯核對擱置限制 (EDTCPCST)」指令可用來顯示任何目前須重建的索引。

有關各類限制的詳情，請參閱第 125 頁的第 10 章, 『資料完整性』 與 Database Programming 一書。

儲存/復置

OS/400 的儲存/復置功能是用來將表格、概略表、索引、日誌、異動日誌接收器、SQL 資料包、SQL 程序、 SQL 觸發程式、使用者定義的函數、使用者定義的類型及集合儲存至磁碟上 (儲存檔案) 或某些外部媒體上 (磁帶或磁片)。而儲存的版本可在日後復置到任何 iSeries 系統上。儲存/復置功能可儲存整個集合、所選取的物件或給定日期與時間後受變更的物件。其間，將物件復置爲原有狀態時所需的全部資訊也會一併儲存。此功能可藉由復置先前版本的表格資料或整個集合，而將受損部份回復成個別表格。

當復置了爲 SQL 程序或服務程式 (爲 SQL 函數或來源函數所建立者) 所建立的程式時，只要該程序或函數尚不存在相同的簽章，它會自動新增至 SYSROUTINES 與 SYSPARMS 編目中。於 QSYS 中建立的 SQL 程式在復置時，將不會被建立爲 SQL 程序。此外，被 CREATE PROCEDURE 或 CREATE FUNCTION 陳述式所參考的外部程式或服務程式，可能會含有在 SYSROUTINES 中登記常式時所需的資訊。若此種資訊存在且簽章是唯一的，進行復置時各函數或程序亦會被新增至 SYSROUTINES 與 SYSPARMS 中。

當 SQL 表格被復置時，針對該表格所定義的 SQL 觸發程式定義也會復置。 SQL 觸發程式定義會自動新增到 SYSTRIGGERS、SYSTRIGDEP、SYSTRIGCOL 及 SYSTRIGUPD 等編目中。當儲存及復置 SQL 表格時，透過 SQL CREATE TRIGGER 陳述式所建立的程式物件亦須儲存及復置。儲存及復置程式物件並非由資料庫管理程式自動進行。將 SQL 表格復置到新的檔案庫時，應複查自我參考觸發程式的預防措施。請參閱 SQL Reference 一書中「CREATE TRIGGER 陳述式附註」小節的Inoperative triggers。

當針對某一使用者定義的類型來復置 *SQLUDT 物件時，此使用者定義的類型會自動新增至 SYSTYPES 編目中。要在使用者定義的類型與來源類型間強制轉型時所需的適當函數亦會建立，只要該類型與函數尚未存在。

分散式 SQL 程式或其相關 SQL 資料包可儲存及復置到任何數目的系統上。如此可讓不同系統上的任何數目 SQL 程式副本存取同一應用程式伺服器上的相同的 SQL 資料

包。這也可以讓單一分散式 SQL 程式連接已復置 (CRTSQLPKG 亦可使用) 該 SQL 資料包的任何數目的應用程式伺服器。SQL 資料包不可復置到不同的檔案庫。

警告: 將綱目復置到現有的檔案庫或名稱不同的綱目中，並不會復置日誌、異動日誌接收器或 IDDU 字典 (如果有話)。如果綱目是復置到不同名稱的綱目中，此綱目中的編目概略表將只反映舊綱目中的物件。不過，QSYS2 中的編目概略表會適當反映所有物件。

損壞容差

伺服器可提供數種機制來減少或消弭磁碟錯誤所造成的損壞。例如，鏡映、檢查和及 RAID 磁碟等都可降低發生磁碟問題的可能性。 DB2 UDB for iSeries 函數亦對磁碟錯誤或系統錯誤造成的損壞具有某種程度的容差。

不論損壞爲何，DROP 作業皆一律成功。這可確保在發生損壞時，至少表格、概略表、SQL 資料包、索引、程序、函數或特殊類型可先刪除，然後進行復置或再次建立。

倘若磁碟錯誤損壞了表格中的部份列，DB2 UDB for iSeries 資料庫管理程式可讓您讀取仍能存取的列。

索引回復

DB2 UDB for iSeries 提供有數種功能來處理索引回復。

- 系統管理的索引保護

  EDTRCYAP CL 指令可讓使用者指示 DB2 UDB for iSeries 保證在發生系統故障或斷電時，將回復所有系統索引所需的時間量保持在指定時間內。系統會自動將足夠的資訊登載至系統日誌中，以便將回復時間限制在指定量之內。

- 索引的日誌登載

  DB2 UDB for iSeries 提供有索引日誌登載功能，可免除因斷電或系統故障以致得重建整個索引的需要。如果索引已登載至日誌，系統資料庫支援會自動確保索引同步於表格中的資料，而不須在損壞時加以重建。SQL 索引*不會*自動登載至日誌中。不過，您可用 CL 指令「啓動日誌存取路徑 (STRJRNAP)」來登載 DB2 UDB for iSeries 所建立的任何索引。

- 索引重新建置

  系統上的全部索引皆具有維護選項，可指定何時要維護索引。SQL 索引是以 *IMMED 維護的屬性來建立。

  發生斷電或異常的系統故障時，若索引不具有前述技術之一的保護，這些索引在變更過程中可能須由資料庫管理程式加以重建，以確保其與實際資料相一致。系統上的全部索引亦具有回復選項，可供指定必要時應在什麼時候重建索引。屬性爲 UNIQUE 的所有 SQL 索引都是以 *IPL 回復屬性來建立 (亦即，這些索引是在 OS/400 啓動前即已重建)。而其他所有 SQL 索引則是以 *AFTIPL 回復選項所建立 (亦即，在作業系統啓動後，索引才非同步地重建)。進行起始程式載入(IPL) 時，操作員會看到一顯示畫面列出需重建的索引及其回復選項。操作員可置換這些回復選項。

- 索引的儲存與復置

  當表格是以「儲存物件 (SAVOBJ)」或「儲存檔案庫 (SAVLIB)」等 CL 指令中的 ACCPTH(*YES) 來儲存時，儲存/復置功能可讓您儲存索引。進行復置且索引已儲存時，即不須重建索引。資料庫管理程式會自動將先前未經儲存及復置的任何索引以非同步方式來重建。

編目完整性

編目內含關於綱目中表格、概略表、SQL 資料包、索引、程序、函數、觸發程式及綱目中參數的資訊。資料庫管理程式應負責確保編目中的資訊始終正確無誤。其中包括防止一般使用者明確變更編目中的任何資訊，以及當編目所說明的表格、概略表、SQL 資料包、索引、類型、程序、函數、觸發程式及參數發生變更時，暗中維護編目中的資訊。

不論綱目中的物件是否被 SQL 陳述式、OS/400 CL 指令、System/38 環境 CL 指令、System/36 環境函數或 iSeries 系統上的任何其他產品或公用程式所變更，編目的完整性皆會保持。例如，要刪除表格時，可藉由執行 SQL DROP 陳述式、發出 OS/400 DLTF CL 指令、發出 System/38 DLTF CL 指令或在 WRKF 或 WRKOBJ 顯示畫面中輸入選項 4 來達成。但不論是用何種介面來刪除表格，資料庫管理程式皆會在執行刪除時，將該表格的說明從編目中移除。下表所列是各類函數以及其對編目的相關作用：

表 38. 各類函數對編目的作用

函數	對編目的作用
新增限制至表格	將資訊新增至編目中
由表格移除限制	移除編目中的相關資訊
在綱目中建立物件	將資訊新增至編目中
刪除綱目中的物件	移除編目中的相關資訊
將物件復置到綱目中	將資訊新增至編目中
變更物件的長註解	更新編目中的註解
變更物件標籤 (文字)	更新編目中的標籤
變更物件擁有者	更新編目中的擁有者
將物件移出綱目	移除編目中的相關資訊
將物件移入綱目	將資訊新增至編目中
更名物件	更新編目中的物件名稱

使用者輔助儲存體儲存區 (ASP)

您可以透過 CREATE COLLECTION 與 CREATE SCHEMA 陳述式的 ASP 子句，在使用者 ASP 中建立綱目。還可用 CRTLIB 指令在使用者 ASP 中建立檔案庫。然後再藉由此檔案庫來接收 SQL 表格、概略表及索引。有關輔助儲存體儲存區的詳情，請參閱 備份及回復 一書。

獨立式輔助儲存體儲存區 (IASP)

獨立式磁碟儲存區是用於在 iSeries 伺服器上設定使用者資料庫。獨立式磁碟儲存區共有三種類型：主要、次要及使用者定義的檔案系統 (UDFS)。而資料庫的設定是藉由主要獨立式磁碟儲存區來進行。

在 iSeries 伺服器上，您可以使用多重資料庫。iSeries 伺服器具備系統資料庫 (通常稱爲 SYSBAS)，以及使用一或多個使用者資料庫的能力。使用者資料庫是透過獨立式磁碟儲存區而建置在 iSeries 伺服器上，此儲存區則是以「iSeries 領航員」的「磁碟管理」功能來設定。一旦設定好獨立式磁碟儲存區後，它就形同「iSeries 領航員」之「資料庫」功能下的另一資料庫。

第 20 章 測試應用程式中的 SQL 陳述式

本章說明如何建立應用程式中 SQL 陳述式的測試環境，及如何除錯此程式。

如需相關明細，請參閱下面各區段：

- 『建立測試環境』
- 第 298 頁的『測試您的 SQL 應用程式』

建立測試環境

測試程式的需求條件如下：

- **授權**。您必須取得建立表格及概略表、存取 SQL 資料、建立及執行程式的授權。
- **測試資料結構**。如果您的程式在表格及概略表中更新、插入或刪除資料，*您應使用測試資料*來驗證程式的執行。如果您的程式只從表格及概略表中擷取資料，您可能要考慮在測試程式時使用生產層次的資料。然而，建議您使用 CL 指令「啓動除錯 (STRDBG)」與 UPDPROD(*NO)，以確保生產層次資料不會被意外變更。如需除錯的相關資訊，請參閱 CL Programming  一書中有關測試的章節。
- **測試輸入資料**。程式在測試期間使用的輸入資料應是有效的資料，且該資料代表您所能想到的各種可能輸入狀況。除非您使用有效的輸入資料，否則無法確定您的輸出資料是否有效。

  如果程式驗證輸入資料是有效的，請併入有效及無效的資料，以驗證是否會處理有效資料並偵測到無效資料。

  您可能必須重新整理資料，以供後續的測試使用。

若要完整地測試程式，請盡可能地透過程式測試各種路徑。例如：

- 使用輸入資料，強迫程式執行每一個分支。
- 檢查結果。例如，如果程式更新某一列，請選取該列以查看是否正確地更新。
- 務必測試程式錯誤常式。再次使用輸入資料，強迫程式盡可能地發生各種預期的錯誤狀況。
- 測試程式使用的編輯與驗證常式。盡可能地提供程式各種不同的輸入資料組合，以驗證它是否正確地編輯或驗證該資料。

如需相關明細，請參閱『設計測試資料結構』。

設計測試資料結構

若要測試存取 SQL 資料的應用程式，您可能要建立測試表格與概略表：

- **測試現有表格的概略表**。如果應用程式沒有變更資料，且資料存在於一或多個生產層次表格，您可以考慮使用現有表格的概略表。也建議您使用 STRDBG 指令與 UPDPROD(*NO)，以確保生產層次資料不會被意外變更。如需除錯的相關資訊，請參閱 CL Programming  。

- **測試表格**。當應用程式建立、變更或刪除資料時，您大概會想要使用含有測試資料的表格來測試應用程式。如需如何建立表格與概略表的說明，請參閱第 2 章, 『SQL 入門』。

  同時，您可能想要使用 CL 指令「建立重複物件 (CRTDUPOBJ)」來建立重複的測試表格、概略表或索引。

授權

您必須先取得建立表格及使用表格所在綱目的授權，然後才能建立表格。此外，您必須具有權限以建立及執行您要測試的程式。

如果您想要使用現有的表格及概略表 (不論是直接使用或作爲概略表的基礎)，必須取得存取那些表格及概略表的授權。

如果您想要建立概略表，必須取得建立概略表的授權，且必須取得使用概略表依據的每一個表格及概略表的授權。如需任何特定 SQL 陳述式所需特定權限的相關資訊，請參閱 SQL Reference 一書。

測試您的 SQL 應用程式

測試 DB2 UDB for iSeries SQL 應用程式分爲兩個階段：程式除錯階段與效能驗證階段。它們是：『程式除錯階段』 及『效能驗證階段』。

程式除錯階段

執行此測試階段，可以確保 SQL 查詢的指定正確，且程式會產生正確的結果。

除錯具有 SQL 陳述式的程式和除錯沒有 SQL 陳述式的程式是非常相似的。然而，在除錯模式的工作中執行 SQL 陳述式時，資料庫管理程式會在工作日誌中放入每一個 SQL 陳述式的執行相關訊息。此訊息是 SQL 陳述式 SQLCODE 的一種指示。如果陳述式順利執行，則 SQLCODE 值是 0，且會發出完成訊息。負值的 SQLCODE 會產生診斷訊息。正值的 SQLCODE 會產生參考訊息。

訊息是以 **SQL** 爲字首的 4 位數代碼，或是以 **SQ** 位字首的 5 位數代碼。例如，-204 的 SQLCODE 會產生 SQL0204 訊息， 30000 的 SQLCODE 會產生訊息 SQ30000。

與 SQLCODE 相關的是 SQLSTATE。SQLSTATE 是在 SQLCA 中提供的一種附加回覆碼，可識別不同 IBM 關聯式資料庫產品中的錯誤狀況。不同關聯式資料庫產品上的相同錯誤狀況，會產生相同的 SQLSTATE。相同的錯誤狀況不會產生相同的 SQLCODE。在判定從非 DB2 UDB for iSeries 系統上執行的關聯式資料庫作業中傳回的錯誤原因時，此回覆碼特別有用。

若爲非 ILE 程式除錯，除錯模式中高階語言陳述式號碼的參照必須採自編譯報表中。若爲 ILE 程式除錯，請前置編譯指定 DBGVIEW(*SOURCE) 的程式，然後使用來源層次除錯器。

SQL 一定會在工作日誌中放入負值 SQLCODE 及 +100 以外正值代碼的訊息，不論它是否處於除錯模式中。

效能驗證階段

此測試階段會驗證是否可以使用適當的索引，及查詢的撰寫方式是否可以讓資料庫管理程式在預期的回應時間內解析查詢。 SQL 應用程式的效能取決於要存取的表格屬性。

如果您使用小型表格，則查詢的回應時間不受索引的可用性影響。然而，當您在具有大型表格的資料庫上執行相同的查詢，且該資料庫上沒有適當的索引時，則查詢的回應時間可能會非常長。

測試環境應盡可能地類似生產環境。測試綱目的表格名稱與組合應與生產綱目相同。在兩個綱目的表格上，必須可以使用相同的索引。表格中的列數與值的分送應該類似。

如需可以用來驗證效能的工作及指令的相關說明，請參閱 Database Performance and Query Optimization 一書。

第 21 章 分散式關聯資料庫功能

*分散式關聯資料庫*由一組分散在交互連接電腦系統間的 SQL 物件組成。這些關聯式資料庫可以是相同類型（例如，DB2 UDB for iSeries）或不同類型 （DB2 Universal Database for OS/390、DB2 for VSE and VM、 DB2 Universal Database (UDB)，或支援 DRDA 的非 IBM 資料庫管理系統）。每一個關聯式資料庫都有一個關聯式資料庫管理程式來管理其環境中的表格。藉由提供資料庫管理程式存取權，在另一系統的關聯式資料庫上執行 SQL 陳述式，資料庫管理程式可彼此通信及合作。

應用要求程式支援連線的應用程式端。應用程式伺服器爲應用要求程式連接的本端或遠端資料庫。DB2 UDB for iSeries 提供「分散式關連資料庫架構 (DRDA)」支援，讓應用要求程式與應用程式伺服器通信。此外，DB2 UDB for iSeries 可呼叫跳出程式來存取不支援 DRDA 的其他資料庫管理系統上的資料。這些跳出程式稱爲應用要求驅動程式 (ARD)。

DB2 UDB for iSeries 支援兩層次的分散式關聯資料庫：

- 遠端工作單元 (RUW)

  遠端工作單元是在某工作單元期間在唯一應用程式伺服器上準備及執行 SQL 陳述式的所在。 DB2 UDB for iSeries 透過 APPC 或 TCP/IP 支援 RUW。
- 分散式工作單元 (DUW)

  分散式工作單元是在某工作單元期間可於多重應用程式伺服器準備及執行 SQL 陳述式的所在。不過，單一 SQL 陳述式只能參照位於單一應用程式伺服器上的物件。DB2 UDB for iSeries 透過 APPC 支援 DUW，而從 V5R1 開始透過 TCP/IP 支援 DUW。

有關分散式關聯資料庫的綜合資訊，請參閱 Distributed Database Programming 一書。

詳細資訊，請參閱

- 第 302 頁的『DB2 UDB for iSeries 分散式關聯資料庫支援』
- 第 302 頁的『DB2 UDB for iSeries 分散式關聯資料庫程式範例』
- 第 303 頁的『SQL 資料包支援』
- 第 307 頁的『SQL 的 CCSID 注意事項』
- 第 307 頁的『連線管理和啓動群組』
- 第 311 頁的『分散式支援』
- 第 317 頁的『分散式工作單元』
- 第 321 頁的『應用要求驅動程式』
- 第 321 頁的『問題處理』
- 第 321 頁的『DRDA 儲存程序注意事項』

DB2 UDB for iSeries 分散式關聯資料庫支援

DB2 UDB Query Manager and SQL Development Kit 授權程式支援以下列 SQL 陳述式來與分散式資料庫做交談式存取：

- CONNECT
- SET CONNECTION
- DISCONNECT
- RELEASE
- DROP PACKAGE
- GRANT PACKAGE
- REVOKE PACKAGE

有關這些陳述式的詳細說明，請參閱 SQL Reference 一書。

開發套件透過 SQL 前置編譯器指令的參數提供其他支援：

Create SQL ILE C Object (CRTSQLCI) 指令

Create SQL ILE C++ Object (CRTSQLCPPI) 指令

Create SQL COBOL Program (CRTSQLCBL) 指令

Create SQL ILE COBOL Object (CRTSQLCBLI) 指令

Create SQL PL/I Program (CRTSQLPLI) 指令

Create SQL RPG Program (CRTSQLRPG) 指令

Create SQL ILE RPG Object (CRTSQLRPGI) 指令

有關 SQL 前置編譯器指令的詳細資訊，請參閱 *SQL Programming with Host Languages* 資訊中的主題 Preparing and Running a Program with SQL Statements。 Create SQL Package (CRTSQLPKG) 指令可讓您從建立爲分散式程式的 SQL 程式中建立 SQL 資料包。有關 CRTSQLPKG 和 CRTSQLxxx 指令的語法及參數定義，請參閱 附錄 B, 『DB2 UDB for iSeries CL 指令說明』。

有關程式碼範例的資訊，請參閱第 x 頁的『程式碼不保事項聲明』資訊。

DB2 UDB for iSeries 分散式關聯資料庫程式範例

遠端工作單元關聯式資料庫程式範例隨附於 SQL 產品中。QSQL 檔案庫內有一些檔案和成員可協助您設定執行分散式 DB2 UDB for iSeries 程式範例的環境。

若要使用這些檔案和成員，您需要執行位於檔案 QSQL/QSQSAMP 中的 SETUP 批次工作。 SETUP 批次工作可讓您自訂範例來執行下列作業：

- 在本端及遠端位置建立 QSQSAMP 檔案庫。
- 在本端及遠端位置設定關聯式資料庫目錄登錄。
- 在本端位置建立應用程式畫面。
- 前置編譯、編譯及執行程式來建立分散式範例應用程式綱目、表格、索引及概略表。
- 將資料載入本端及遠端位置上的表格。
- 前置編譯及編譯程式。
- 在遠端位置爲應用程式建立 SQL 資料包。

- 前置編譯、編譯及執行程式來更新部門表格中的位置直欄。

執行 SETUP 之前，您需要編輯 QSQL/QSQSAMP 檔案的 SETUP 成員。成員中併入一些指示作爲說明。若要執行 SETUP，請在系統指令行指定下列指令：

```
========> SBMDBJOB QSQL/QSQSAMP SETUP
```

等待批次工作完成。

若要使用程式範例，請在指令行指定下列指令：

```
========> ADDLIBLE QSQSAMP
```

若要呼叫供您自訂程式範例的第一個顯示畫面，請在指令行指定下列指令。

```
========> CALL QSQ8HC3
```

即出現下列顯示畫面。您可以在這個顯示畫面中自訂資料庫程式範例。

```
DB2 for OS/400 ORGANIZATION APPLICATION


ACTION...........:   _      A  (ADD)                  E (ERASE)
D  (DISPLAY)           U (UPDATE)

OBJECT...........:   __     DE (DEPARTMENT)           EM (EMPLOYEE)
DS (DEPT STRUCTURE)

SEARCH CRITERIA..:   __     DI (DEPARTMENT ID)     MN (MANAGER NAME)
DN (DEPARTMENT NAME)   EI (EMPLOYEE ID)
MI (MANAGER ID)        EN (EMPLOYEE NAME)

LOCATION.........:   ________________  (BLANK IMPLIES LOCAL LOCATION)

DATA.............:   ________________________________




                                                          底端

F3=跳出
                                    (C) COPYRIGHT IBM CORP. 1982, 1991
```

SQL 資料包支援

OS/400 程式支援稱爲 SQL 資料包的物件。(OS/400 物件類型爲 *SQLPKG。) SQL 資料包含有執行分散式程式時在應用程式伺服器上處理 SQL 陳述式所需的控制結構和存取規劃。在下列情況可建立 SQL 資料包：

- CRTSQLxxx 指令上已指定 RDB 參數且順利建立了程式物件。將於 RDB 參數指定的系統上建立 SQL 資料包。

  如果編譯失敗或編譯僅建立模組物件，則不建立 SQL 資料包。
- 使用 CRTSQLPKG 指令。當前置編譯時未建立資料包或 RDB 需要的資料包不是前置編譯指令上指定的資料包時，CRTSQLPKG 可用來建立資料包。

Delete SQL Package (DLTSQLPKG) 指令可讓您刪除本端系統上的 SQL 資料包。

除非與建立 SQL 資料包相關的授權 ID 所保留的專用權在遠端系統 (應用程式伺服器) 上有建立資料包的適當權限，否則不建立 SQL 資料包。若要執行程式，授權 ID 必須有 SQL 資料包的 EXECUTE 專用權。在 iSeries 系統上，EXECUTE 專用權包含系統權限 *OBJOPR 和 *EXECUTE。

有關 Create SQL Package (CRTSQLPKG) 指令的語法，請參閱 附錄 B, 『DB2 UDB for iSeries CL 指令說明』。

SQL 資料包中有效的 SQL 陳述式

連接另一個伺服器的程式可使用 SQL Reference 一書中描述的任何 SQL 陳述式，SET TRANSACTION 陳述式除外。使用 DB2 UDB for iSeries（參照非 DB2 UDB for iSeries 的系統）編譯的程式可使用遠端系統支援的可執行 SQL 陳述式。前置編譯器會繼續對它不瞭解的陳述式發出診斷訊息。這些陳述式在 SQL 資料包建立期間傳送到遠端系統。當陳述式無法在現行應用程式伺服器上執行時，執行時間支援會傳回 -84 或 -525 的 SQLCODE。例如，唯有當分散程式中的應用要求程式和應用程式伺服器都是版本 2 版次 2 或以上的 OS/400，才允許多列 FETCH、區塊化 INSERT 及可捲動的游標支援。詳細資訊，請參閱 SQL Reference 一書中的 Considerations for Using Distributed Relational Database。

建立 SQL 資料包的注意事項

建立 SQL 資料包有許多需要考量的注意事項。下面列出部份注意事項。

CRTSQLPKG 授權

在 iSeries 系統上建立 SQL 資料包時，使用的授權 ID 必須有 CRTSQLPKG 指令的 *USE 權限。

在非 DB2 UDB for iSeries 上建立資料包

當您對非 DB2 UDB for iSeries 建立程式及 SQL 資料包，並且試圖使用該關聯式資料庫唯一的 SQL 陳述式時，CRTSQLxxx GENLVL 參數應設爲 30。除非出現嚴重性層次超過 30 的訊息，否則會建立程式。如果出現的訊息含有嚴重性層次超過 30，陳述式可能不適用任何關聯式資料庫。例如，未定義或無法使用的主變數或無效的常數會產生嚴重性層次超過 30 的訊息。

當以大於 10 的 GENLVL 執行時，應檢查前置編譯器報表中是否有非預期的訊息。對 DB2 Universal Database 建立資料包時，您必須將 GENLVL 參數設爲小於 20 的值。

如果 RDB 參數指定的是非 DB2 UDB for iSeries 系統，則下列選項不應使用於 CRTSQLxxx 指令：

- COMMIT(*NONE)
- OPTION(*SYS)
- DATFMT(*MDY)
- DATFMT(*DMY)
- DATFMT(*JUL)
- DATFMT(*YMD)
- DATFMT(*JOB)
- DYNUSRPRF(*OWNER)
- TIMFMT(*HMS)(如果指定了 TIMSEP(*BLANK) 或 TIMSEP(’,’))

- SRTSEQ(*JOBRUN)
- SRTSEQ(*LANGIDUNQ)
- SRTSEQ(*LANGIDSHR)
- SRTSEQ(檔案庫名稱/表格名稱)

註: 連接 DB2 Universal Database 伺服器時，套用下列附加規則：

- 指定的日期與時間格式必須相同
- *BLANK 值必須用於 TEXT 參數
- 不支援預設綱目 (DFTRDBCOL)
- 其中建立了資料包的原始程式之 CCSID 不可以是 65535；如果使用 65535，則會建立空的資料包。

目標版次 (TGTRLS)

建立資料包時，會檢查 SQL 陳述式來判定哪一個版次可支援此函數。此版次設爲資料包的復置層次。例如，如果資料包含有在表格中新增 FOREIGN KEY 限制的 CREATE TABLE 陳述式，則資料包的復置層次將爲版本 3 版次 1，因爲此版次之前的版次不支援 FOREIGN KEY 限制。當 TGTRLS 參數爲 *CURRENT 時，TGTRLS 訊息會被抑制。

SQL 陳述式大小

Create SQL package 函數可能無法處理前置編譯器可處理的相同大小 SQL 陳述式。在 SQL 程式前置編譯期間，SQL 陳述式存放在程式的相關位置中。在這個情形下，每一個記號以一個空格區隔。此外，當指定了 RDB 參數，來源陳述式的主變數置換爲 'H'。Create SQL package 函數會將陳述式以及該陳述式的主變數清單傳遞至應用程式伺服器。記號與置換的主變數之間增加的空格可能使陳述式超過最大 SQL 陳述式大小（SQL0101 理由 5）。

不需要資料包的陳述式

有時候，您想要建立 SQL 資料包，但無法建立它，且程式仍在執行。當程式僅含不需要執行 SQL 資料包的 SQL 陳述式時，會發生此狀況。例如，僅含 SQL 陳述式 DESCRIBE TABLE 的程式，在 SQL 資料包建立期間會產生訊息 SQL5041。不需要 SQL 資料包的 SQL 陳述式如下：

- COMMIT
- CONNECT
- DESCRIBE TABLE
- DISCONNECT
- RELEASE
- RELEASE SAVEPOINT
- ROLLBACK
- SAVEPOINT
- SET CONNECTION

資料包物件類型

SQL 資料包一律建立爲非 ILE 物件且一律在預設啓動群組中執行。

ILE 程式和服務程式

連結內含 SQL 陳述式的若干模組之 ILE 程式和服務程式，必須有每一個模組個別的 SQL 資料包。

資料包建立連線

資料包建立完成的連線類型視使用 RDBCNNMTH 參數所要求的連接類型而定。如果指定了 RDBCNNMTH(*DUW)，則會使用確定控制，而且連線可能是唯讀連線。如果連線是唯讀的，則資料包建立失敗。

工作單元

由於資料包建立未明確執行確定或回轉，所以在嘗試資料包建立之前，確定定義必須位於工作單元界限上。下列狀況必須完全爲 TRUE，確定定義才能在工作單元界限上：

- SQL 在工作單元界限上。
- 本端或 DDM 檔案不是使用確定控制開啓，而且關閉的本端或 DDM 檔案不含擱置中變更。
- 沒有登錄任何 API 資源。
- 沒有登錄任何與 DRDA 或 DDM 無關的 LU 6.2 資源。

在本端建立資料包

指定於 RDB 參數的名稱可以是本端系統名稱。如果是本端系統名稱，則會在本端系統上建立 SQL 資料包。您可以儲存 (SAVOBJ 指令) SQL 資料包，再復置 (RSTOBJ 指令) 到另一個伺服器。當執行具有本端系統連線的程式時，不使用 SQL 資料包。如果您對 RDB 參數指定 *LOCAL，則不建立 *SQLPKG 物件，但會在 *PGM 物件中儲存資料包資訊。

標籤

您可以使用 LABEL ON 陳述式來建立 SQL 資料包的說明。

一致性記號

程式及其相關的 SQL 資料包含有呼叫 SQL 資料包時檢查的一致性記號。一致性記號必須相符，否則無法使用資料包。程式與 SQL 資料包可能出現不協調的狀況。假設程式位於 iSeries 系統上，而應用程式伺服器是另一個 iSeries 系統。程式正執行於階段作業 A 中，但重建於階段作業 B 中 (在此也會重建 SQL 資料包)。下次在階段作業 A 呼叫該程式時，可能導致一致性記號的錯誤發生。爲免除每次呼叫都要尋找 SQL 資料包，SQL 會對每個階段作業使用的 SQL 資料包位址清單進行維護。當階段作業 B 重建 SQL 資料包時，舊的 SQL 資料包會移到 QRPLOBJ 檔案庫中。階段作業 A 中的 SQL 資料包位址仍然有效。(從執行程式的階段作業中建立程式及 SQL 資料包，或在建立程式之前提出遠端指令來刪除舊的 SQL 資料包，可避免發生此狀況。)

若要使用新的 SQL 資料包，您應該結束與遠端系統的連線。您可以登出階段作業後再重新登入，或是使用交談式 SQL (STRSQL) 指令，對不受保護的網路連線發出 DISCONNECT，或對受保護的連線發出 RELEASE，再發出 COMMIT。然後，使用 RCLDDMCNV 來結束網路連線。重新呼叫程式。

SQL 與遞迴

如果您在前置編譯的同時對岔斷鍵程式呼叫 SQL，則會出現無法預期的結果。

CRTSQLxxx、CRTSQLPKG、STRSQL 指令及 SQL 執行時間環境不遞迴。如果嘗試遞迴，它們會產生無法預期的結果。如果執行其中一個指令時 (或執行內含 SQL 陳述式的程式)，在完成指令前工作被岔斷，並且啓動了另一個 SQL 函數，則會發生遞迴。

SQL 的 CCSID 注意事項

如果您在執行分散式應用程式，而其中一個系統不是 iSeries 系統，則 iSeries 伺服器上的 CCSID 值不能設爲 65535。

要求遠端系統建立 SQL 資料包之前，應用要求程式恆將 RDB 參數上指定的名稱、SQL 資料包名稱、檔案庫名稱及 SQL 資料包文字，從工作的 CCSID 轉換成 CCSID 500。這對於 DRDA 是必要的。當遠端關聯式資料庫爲 iSeries 系統時，名稱則不從 CCSID 500 轉換成工作 CCSID。

建議不要將有定界符號的 ID 用於表格、概略表、索引、綱目、檔案庫或 SQL 資料包名稱。具有不同 CCSID 的系統之間不會發生名稱轉換。請參考下列範例，其中的系統 A 以 CCSID 37 執行，而系統 B 以 CCSID 500 執行。

- 在系統 A 建立一個使用 ″a¬b|c″ 名稱建立表格的程式。
- 在系統 A 儲存程式 ″a¬b|c″，然後將它復置到系統 B。
- 在 CCSID 37 中，¬ 的字碼點爲 x’5F’，而在 CCSID 500 中是 x’BA’。
- 在系統 B 上，名稱應顯示爲 ″a[b]c″。如果您建立的程式參照名稱爲 ″a¬b|c″ 的表格，則該程式找不到此表格。

SQL 物件名稱中不應使用 @ 符號 (@)、# 字符號 (#) 和錢幣符號 ($) 字元。它們的字碼點視使用的 CCSID 而定。如果您使用有定界符號的名稱或這三個擴充元，則可能無法在未來版次中使用名稱解析功能。

連線管理和啓動群組

詳細資訊，請參閱下列主題：

- 『連線與交談』
- 第 308 頁的『PGM1 的原始程式碼：』
- 第 308 頁的『PGM2 的原始程式碼：』
- 第 309 頁的『PGM3 的原始程式碼：』
- 第 310 頁的『與相同關聯式資料庫的多重連線』
- 第 311 頁的『預設啓動群組的隱含連線管理』
- 第 311 頁的『非預設啓動群組的隱含連線管理』

連線與交談

在 DRDA 使用 TCP/IP 之前，「連線」一詞即已明確。從 SQL 的觀點來看，它指的是連線。亦即，在其中一個執行 CONNECT TO 某個 RDB 時啓動連線，而在執行了 DISCONNECT，或發出 RELEASE ALL，再發出成功的 COMMIT 後結束。根據工作的 DDMCNV 屬性值，以及交談對象是 iSeries 或其他系統類型，不一定保留 APPC 交談。

TCP/IP 術語中並未併入「交談」一詞。不過，類似的概念存在。隨著 DRDA 所支援 TCP/IP 的出現，本書中使用的「交談」一詞將以更通用的術語「連線」來取代，除非有

特別討論到 APPC 交談。因此，現在讀者必須知道兩個不同類型的連線：上面描述的 SQL 連線類型，以及取代「交談」一詞的「網路」連線。

兩連線類型之間可能會產生混淆，所以要藉由 SQL 或「網路」的定義來讓讀者瞭解其意謂的含意。

SQL 連線是在啓動群組層次上管理。工作內每一個啓動群組管理其本身的連線，而這些連線不在啓動群組之間共用。若程式在預設啓動群組中執行，則連線的管理仍如同在版本 2 版次 3 之前一樣。

下列爲執行於多重啓動群組中的應用程式範例。本範例用來說明啓動群組間的交談、連線管理及確定控制。它**不是**建議的撰寫程式碼樣式。

PGM1 的原始程式碼：

```
....
EXEC SQL
  CONNECT TO SYSB
END-EXEC.
EXEC SQL
  SELECT ....
END-EXEC.
CALL PGM2.
....
```

圖 9. PGM1 的原始程式碼

對 PGM1 建立程式和 SQL 資料包的指令：

```
CRTSQLCBL PGM(PGM1) COMMIT(*NONE) RDB(SYSB)
```

PGM2 的原始程式碼：

```
...
        EXEC SQL
          CONNECT TO SYSC;
        EXEC SQL
          DECLARE C1 CURSOR FOR
              SELECT ....;
        EXEC SQL
          OPEN C1;
        do {
             EXEC SQL
               FETCH C1 INTO :st1;
             EXEC SQL
               UPDATE ...
                   SET COL1 = COL1+10
                   WHERE CURRENT OF C1;
             PGM3(st1);
        } while SQLCODE == 0;
        EXEC SQL
         CLOSE C1;
        EXEC SQL COMMIT;
....
```

圖 10. PGM2 的原始程式碼

對 PGM2 建立程式和 SQL 資料包的指令：

```
CRTSQLCI OBJ(PGM2) COMMIT(*CHG) RDB(SYSC) OBJTYPE(*PGM)
```

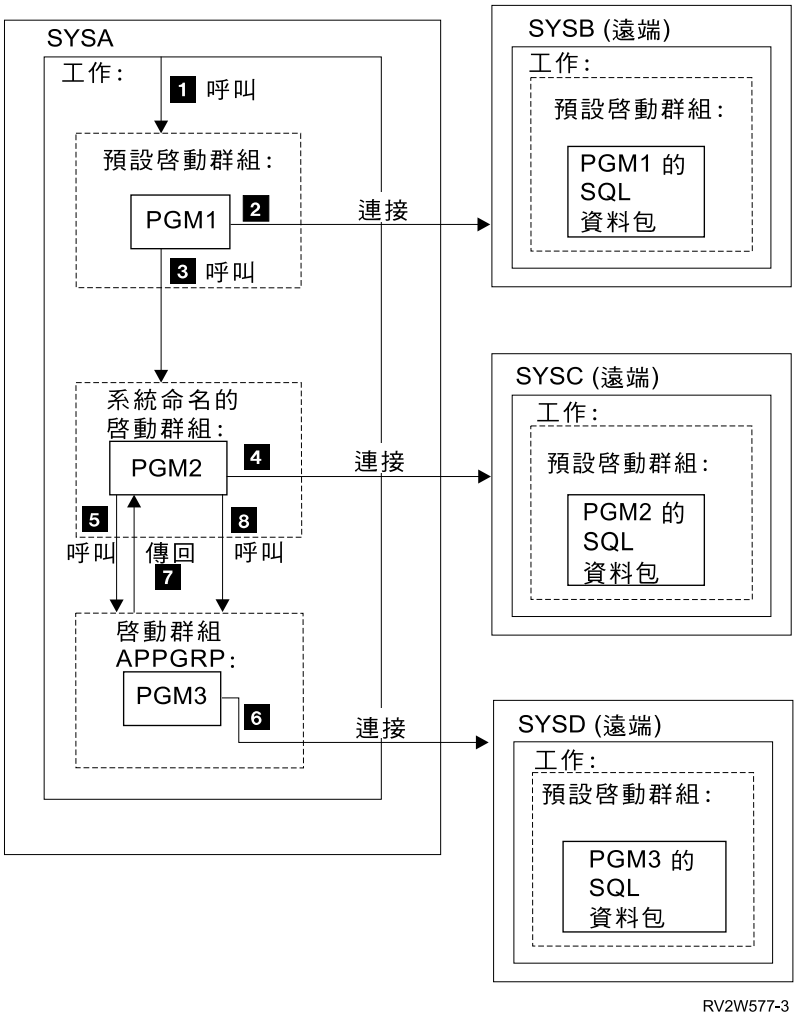
PGM3 的原始程式碼：

```
...
 EXEC SQL
   INSERT INTO TAB VALUES(:st1);
 EXEC SQL COMMIT;
....
```

圖 11. PGM3 的原始程式碼

對 PGM3 建立程式和 SQL 資料包的指令：

```
CRTSQLCI OBJ(PGM3) COMMIT(*CHG) RDB(SYSD) OBJTYPE(*MODULE)
CRTPGM PGM(PGM3) ACTGRP(APPGRP)
CRTSQLPKG PGM(PGM3) RDB(SYSD)
```



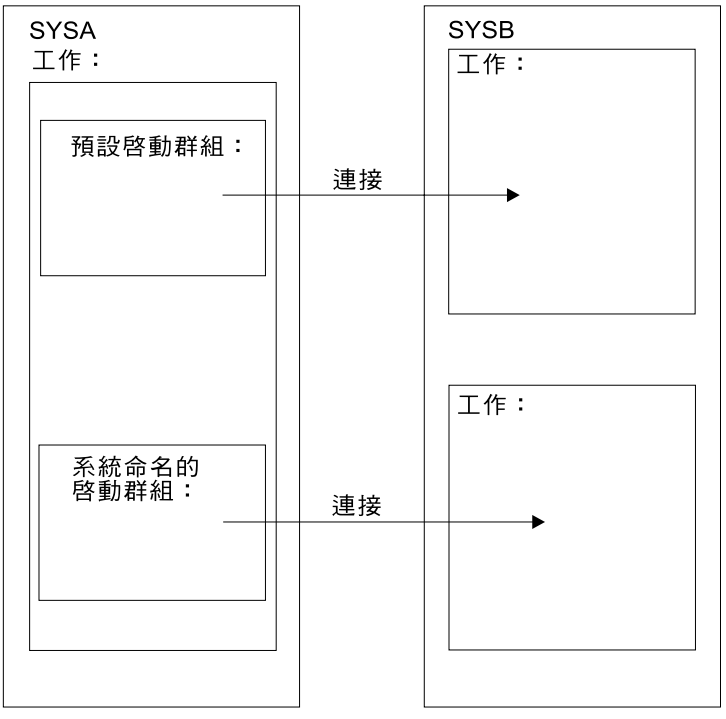
在這個範例中，PGM1 是使用 CRTSQLCBL 指令建立的非 ILE 程式。此程式在預設啓動群組中執行。PGM2 是使用 CRTSQLCI 指令所建立的，它在系統指定的啓動群組中執行。 PGM3 也是使用 CRTSQLCI 指令建立的，但它在名爲 APPGRP 的啓動群組中執行。由於 APPGRP 不是 ACTGRP 參數的預設值，所以個別發出 CRTPGM 指令。

CRTPGM 指令後面接著是 CRTSQLPKG 指令，它可在 SYSD 關聯式資料庫上建立 SQL 資料包物件。在這個範例中，使用者未明確啓動工作層次確定定義。 SQL 隱含地啓動確定控制。

1. 呼叫 PGM1 並執行於預設啓動群組中。
2. PGM1 連接到關聯式資料庫 SYSB 並執行 SELECT 陳述式。
3. 然後，PGM1 呼叫 PGM2，後者執行於系統指定的啓動群組中。
4. PGM2 連接到關聯式資料庫 SYSC。由於 PGM1 和 PGM2 位於不同的啓動群組中，所以系統指定的啓動群組中之 PGM2 所啓動的連線不切斷預設啓動群組中 PGM1 所啓動的連線。這兩個連線都在作用中。PGM2 開啓游標，並且提取及更新列。在確定控制下執行的 PGM2 位於工作單元中間，不是處於可連接狀態。
5. PGM2 呼叫 PGM3，後者執行於啓動群組 APPGRP 中。
6. 第一個陳述式中的 INSERT 陳述式執行於啓動群組 APPGRP 中。第一個 SQL 陳述式造成與關聯式資料庫 SYSD 的隱含連線。插入列至關聯式資料庫 SYSD 的表格 TAB 中。然後，確定插入。未確定系統命名的啓動群組中之擱置變更，因爲 SQL 啓動的確定控制具有啓動群組的確定範圍。
7. 然後，PGM3 跳出，控制權傳回 PGM2。PGM2 提取及更新另一列。
8. 再次呼叫 PGM3 以插入列。在第一次呼叫 PGM3 時會執行隱含的連接。不會在後續的呼叫上執行，因爲啓動群組沒有在 PGM3 呼叫之間結束。最後，由 PGM2 處理所有列，然後確定與系統指定的啓動群組相關的工作單元。

與相同關聯式資料庫的多重連線

如果不同的啓動群組連接到相同的關聯式資料庫，則每一個 SQL 連線都有本身的網路連線及本身的應用程式伺服器工作。如果啓動群組以確定控制執行，則一個啓動群組中確定的變更，不會確定另一個啓動群組中的變更，除非使用工作層次確定定義。



預設啓動群組的隱含連線管理

應用要求程式可隱含地連接應用程式伺服器。當應用要求程式偵測到預設啓動群組的第一個作用中 SQL 程式發出第一個 SQL 陳述式，而且下列項目屬實時，會發生隱含的 SQL 連線：

- 發出的 SQL 陳述式不是具有參數的 CONNECT 陳述式。
- SQL 未在預設啓動群組中作用。

以分散式程式而言，隱含的 SQL 連線是連接到 RDB 參數上指定的關聯式資料庫。若是非分散式程式，隱含的 SQL 連線是連接到本端關聯式資料庫。

當 SQL 變成非作用中時，它會結束預設啓動群組中任何作用中的連線。當下列情況發生，SQL 變成非作用中：

- 應用要求程式偵測到處理程序的第一個作用中 SQL 程式已結束，而且下列全部屬實：
  - 沒有擱置中 SQL 變更
  - 沒有使用受保護連線的連線
  - SET TRANSACTION 陳述式非作用中
  - 沒有執行以 CLOSQLCSR(*ENDJOB) 前置編譯的程式。

  如果有擱置中變更、受保護的連線或作用中 SET TRANSACTION 陳述式， SQL 處於跳出狀態。如果執行了以 CLOSQLCSR(*ENDJOB) 前置編譯的程式，則在工作結束前，SQL 對預設啓動群組而言仍作用中。
- 在工作單元結束時，SQL 處於跳出態。當您在 SQL 程式外部發出 COMMIT 或 ROLLBACK 指令時，會發生這個情況。
- 於工作結束時。

非預設啓動群組的隱含連線管理

應用要求程式可隱含地連接應用程式伺服器。當應用要求程式偵測到對啓動群組發出的第一個 SQL 陳述式不是具有參數的 CONNECT 陳述式時，會發生隱含的 SQL 連線。

以分散式程式而言，隱含的 SQL 連線是連接到 RDB 參數上指定的關聯式資料庫。若是非分散式程式，隱含的 SQL 連線是連接到本端關聯式資料庫。

在處理程序的下列時機上會發生隱含的斷線：

- 當啓動群組結束時，如果確定控制爲非作用中、啓動群組層次確定控制爲作用中，或工作層次確定定義處於工作單元界限。

  如果工作層次確定定義爲作用中且不是在工作單元界限上，SQL 處於跳出狀態。
- 如果 SQL 處於跳出狀態，當確定或回轉工作層次確定定義。
- 於工作結束時。

分散式支援

DB2 UDB for iSeries 支援兩個層次的分散式關聯資料庫：

- 遠端工作單元 (RUW)

遠端工作單元是在某工作單元期間於唯一應用程式伺服器上準備及執行 SQL 陳述式的所在。在應用要求程式上具有應用程式處理的啓動群組可連接到應用程式伺服器，而在一或多個工作單元內，可執行參照應用程式伺服器上的物件的任何靜態或動態 SQL 陳述式。遠端工作單元也稱爲 DRDA 層次 1。

- 分散式工作單元 (DUW)

  分散式工作單元是在某工作單元期間可在多重應用程式伺服器上準備及執行 SQL 陳述式的所在。不過，單一 SQL 陳述式只能參照位於單一應用程式伺服器上的物件。分散式工作單元也稱爲 DRDA 層次 2。

  分散式工作單元允許下列操作：

  – 更新一個邏輯工作單元中多重應用程式伺服器的存取權

    或

  – 更新一個邏輯工作單元中具有多重應用程式伺服器讀取權的單一應用程式伺服器存取權。

  多重應用程式伺服器能否在工作單元中更新，取決於應用要求程式上是否有同步點管理程式、應用程式伺服器上是否有同步點管理程式，以及應用要求程式與應用程式伺服器之間是否有兩階段確定通信協定支援。

  同步點管理程式是一種系統元件，用來協調兩階段確定通信協定中參與者之間的確定和回轉操作。當執行分散式更新時，不同系統上的同步點管理程式會共同確保資源達到一致的狀態。同步點管理程式使用的通信協定和流程也稱爲兩階段確定通信協定。

  如果使用兩階段確定通信協定，則連線爲受保護的資源；否則，連線爲未受保護的資源。

  系統間使用的資料傳輸通信協定類型會影響網路連線爲受保護的或未受保護的。在 V5R1 之前，TCP/IP 連線恆爲未受保護的；所以它們只能以受限制的方式參與分散式工作單元。在 V5R1 中，加入了具有 TCP/IP 的 DUW 完整支援。

  例如，如果從程式建立的第一個連線是透過 TCP/IP 連接到 V5R1 之前的伺服器，則可透過它來執行更新，但即使是透過 APPC 的任何後續連線將爲唯讀連線。

  請注意，當使用交談式 SQL 時，第一個 SQL 連線是連接到本端系統。因此，在 V5R1 之前的環境中，爲透過 TCP/IP 來更新遠端系統，在執行 CONNECT TO 遠端 TCP 系統之前，您必須執行 RELEASE ALL，再執行 COMMIT，以結束所有 SQL 連線。

決定連線類型

當建立遠端連線時，它將使用未受保護或受保護的網路連線。關於可確定的更新，無論建立連線時此 SQL 連線是否爲可更新，都可能是唯讀、可更新或不明。可確定的更新是在確定控制下執行的任何 insert、delete、update 或 DDL 陳述式。如果連線是唯讀的，則仍可執行使用 COMMIT(*NONE) 的變更。在 CONNECT 或 SET CONNECTION 後，SQLCA 的 SQLERRD(4) 指示連線類型。 SQLERRD(4) 也將指出連線使用的是受保護的或未受保護的網路連線。特定的值如下：

1. 可確定的更新可在連線上執行。此連線未受保護。當下列出現時，會發生此情況：
   - 連線是使用遠端工作單元 (RDBCNNMTH(*RUW)) 建立。這也包括使用遠端工作單元的應用要求驅動程式 (ARD) 連線和本端連線。
   - 若使用分散式工作單元 (RDBCNNMTH(*DUW)) 來建立連線，則下列全部屬實：
     – 連線不是本端連線。

- 應用程式伺服器不支援分散式工作單元。例如，DB2 UDB for iSeries 應用程式伺服器具有版本 3 版次 1 之前的 OS/400 版次。
- 所發出的連線程式之確定控制層次不是 *NONE。
- 沒有與可執行可確定的更新之其他應用程式伺服器 (包括本端) 的連線存在，或所有與未支援分散式工作單元的應用程式伺服器之連線爲唯讀連線。
- 在確定定義的確定控制下沒有開啓的可更新本端檔案。
- 在確定定義的確定控制下沒有使用不同連線之開啓的可更新 DDM 檔案。
- 沒有確定定義的 API 確定控制資源。
- 沒有對確定定義登錄的受保護連線。

如果以確定控制執行，SQL 會對遠端連線登錄一階段可更新 DRDA 資源，或對本端和 ARD 連線登錄兩階段可更新 DRDA 資源。

2. 無法在連線上執行可確定的更新。連線爲唯讀連線。網路連線未受保護。

   對於使用遠端工作單元連線管理 (*RUW) 編譯的應用程式而言，絕不會發生這種情況。

   對於分散式工作單元應用程式而言，只有在建立連線而下列實屬時，才會發生這種情況：

   - 連線不是本端連線。
   - 應用程式伺服器不支援分散式工作單元
   - 至少下列其中一項屬實：
     - 發出連線的程式之確定控制層次爲 *NONE。
     - 與不支援分散式工作單元的應用程式伺服器之另一個連線存在，而該應用程式伺服器可執行可確定的更新
     - 與支援分散式工作單元的應用程式伺服器 (包括本端) 之另一個連線存在。
     - 在確定定義的確定控制下有開啓的可更新本端檔案。
     - 在確定定義的確定控制下有使用不同連線之開啓的可更新 DDM 檔案。
     - 沒有確定定義的一階段 API 確定控制資源。
     - 有對確定定義登錄的受保護連線。

   如果以確定控制執行，SQL 會登錄一階段唯讀資源。

3. 是否可執行可確定的更新，情況不明。此連線受保護。

   對於使用遠端工作單元連線管理 (*RUW) 編譯的應用程式而言，絕不會發生這種情況。

   對於分散式工作單元應用程式而言，只有在建立連線而下列全部屬實時，才會發生這種情況：

   - 連線不是本端連線。
   - 發出連線的程式之確定控制層次不是 *NONE。
   - 應用程式伺服器支援分散式工作單元和兩階段確定通信協定 (受保護的連線)。

   如果以確定控制執行，SQL 會登錄兩階段不確定資源。

4. 是否可執行可確定的更新，情況不明。此連線未受保護。

   對於使用遠端工作單元連線管理 (*RUW) 編譯的應用程式而言，絕不會發生這種情況。

對於分散式工作單元而言，只有在建立連線而下列全部屬實時，才會發生這種情況：

- 連線不是本端連線。
- 應用程式伺服器支援分散式工作單元
- 應用程式伺服器不支援兩階段確定通信協定 (受保護的連線)，或發出連線的程式之確定控制層次爲 *NONE。

如果以確定控制執行，SQL 會登錄一階段 DRDA 不確定資源。

5. 是否可執行可確定的更新，且連線爲使用分散式工作單元的本端連線或使用分散式工作單元的 ARD 連線，情況不明。

   如果以確定控制執行，SQL 會登錄兩階段 DRDA 不確定資源。

有關兩階段及一階段資源的詳細資訊，請參閱確定控制主題。備份及回復 一書。

下表彙總針對遠端分散式工作單元連線產生的連線類型。SQLERRD(4) 設定於成功的 CONNECT 和 SET CONNECTION 陳述式上。

表 39. 連線類型摘要

確定控制下的連線	應用程式伺服器支援兩階段確定	應用程式伺服器支援分散式工作單元	登錄其他可更新一階段資源	SQLERRD(4)
否	否	否	否	2
否	否	否	是	2
否	否	是	否	4
否	否	是	是	4
否	是	否	否	2
否	是	否	是	2
否	是	是	否	4
否	是	是	是	4
是	否	否	否	1
是	否	否	是	2
是	否	是	否	4
是	否	是	是	4
是	是	否	否	N/A *
是	是	否	是	N/A *
是	是	是	否	3
是	是	是	是	3

*DRDA 不允許受保護的連線用於僅支援遠端工作單元 (DRDA1) 的應用程式伺服器。這包括所有 DB2 for iSeries TCP/IP 連線。

連線與確定控制限制

您在使用確定控制來連線時有一些限制。當您嘗試使用確定控制執行陳述式，但連線是使用 COMMIT(*NONE) 建立，這些限制也適用。

如果已登錄兩階段不確定或可更新資源，或已登錄一階段可更新資源，則無法登錄另一個一階段可更新資源。

此外，當受保護的連線非作用中且 DDMCNV 工作屬性爲 *KEEP 時，這些未用的 DDM 連線也會導致程式中以 RUW 連線管理編譯的 CONNECT 陳述式失敗。

如果以 RUW 連線管理執行且使用工作層次確定定義，則會有一些限制。

- 如果有多個啓動群組使用工作層次確定定義，則所有 RUW 連線必須連接到本端關聯式資料庫。
- 如果連線爲遠端，則僅一個啓動群組可對 RUW 連線使用工作層次確定定義。

決定連線狀態

不含參數的 CONNECT 陳述式可用來決定現行連線對現行工作單元而言是可更新或唯讀。於 SQLERRD(3) 中將傳回值 1 或 2。SQLERRD(3) 中的值決定如下：

1. 可在工作單元連線上執行可確定的更新。

   當下列其中一項屬實時，會發生此情況：

   - SQLERRD(4) 具有值 1。
   - 下列全部屬實：
     - SQLERRD(4) 具有值 3 或 5。
     - 與不支援分散式工作單元的應用程式伺服器 (可執行可確定的更新) 連線不存在。
     - 下列其中一項屬實：
       - 第一個可確定的更新執行於使用受保護連線的連線上、執行於本端資料庫上，或執行於 ARD 程式連線上。
       - 在確定控制下有開啓的可更新本端檔案。
       - 有使用受保護的連線之開啓的可更新 DDM 檔案。
       - 有兩階段 API 確定控制資源。
       - 沒有執行可確定的更新。
   - 下列全部屬實：
     - SQLERRD(4) 具有值 4
     - 與不支援分散式工作單元的應用程式伺服器 (可執行可確定的更新) 之其他連線不存在。
     - 第一個可確定的更新執行於此連線上，或沒有執行任何可確定的更新。
     - 沒有使用受保護的連線之開啓的可更新 DDM 檔案。
     - 在確定控制下沒有開啓的可更新本端檔案。
     - 沒有兩階段 API 確定控制資源。
2. 無法在此工作單元的連線上執行可確定的更新。

   當下列其中一項屬實時，會發生此情況：

   - SQLERRD(4) 具有值 2。
   - SQLERRD(4) 具有值 3 或 5，而且下列其中一項屬實：
     - 與僅支援遠端工作單元的可更新應用程式伺服器之連線存在。
     - 第一個可確定的更新執行於使用不受保護連線的連線上。

- SQLERRD(4) 具有值 4，而且下列其中一項屬實：
  - 與僅支援遠端工作單元的可更新應用程式伺服器之連線存在。
  - 第一個可確定的更新未執行於此連線上。
  - 使用受保護連線之開啓的可更新 DDM 檔案。
  - 在確定控制下有開啓的可更新本端檔案。
  - 有兩階段 API 確定控制資源。

下表彙總如果僅支援遠端工作單元的應用程式伺服器有可更新連線，且在此發生了第一個可確定的更新，SQLERRD(3) 如何取決於 SQLERRD(4) 值。

表 40. 決定 SQLERRD(3) 值的摘要

SQLERRD(4)	與可更新遠端工作單元應用程式伺服器的連線存在	發生第一個可確定的更新之所在 *	SQLERRD(3)
1	--	--	1
2	--	--	2
3	是	--	2
3	否	無更新	1
3	否	一階段	2
3	否	此連線	1
3	否	兩階段	1
4	是	--	2
4	否	無更新	1
4	否	一階段	2
4	否	此連線	1
4	否	兩階段	2
5	是	--	2
5	否	無更新	1
5	否	一階段	2
5	否	此連線	1
5	否	兩階段	1

* 此直欄中的術語定義如下：

- *無更新*，指示沒有執行可確定的更新、沒有開啓 DDM 檔案來使用受保護的連線更新、沒有開啓本端檔案來更新，以及沒有登錄確定控制 API。
- *一階段*，指示使用不受保護的連線執行了第一個可確定的更新，或開啓 DDM 檔案來使用不受保護的連線更新。
- *兩階段*，指示在兩階段分散式工作單元應用程式伺服器上執行了可確定的更新、開啓 DDM 檔案來使用受保護的連線更新、登錄確定控制 API，或開啓本端檔案於確定控制下更新。

當 SQLERRD(4) 的值爲 3、4 或 5 (由於 ARD 程式) 且 SQLERRD(3) 的值爲 2 時，如果嘗試透過連線執行可確定的更新，工作單元將處於需要回轉的狀態中。如果工作單元處於需要回轉的狀態，可允許的唯一陳述式是 ROLLBACK 陳述式；其他所有陳述式會導致發生 SQLCODE -918。

分散式工作單元連線注意事項

在分散式工作單元應用程式中連接時，有許多注意事項。本節列出一些預定注意事項。

- 如果工作單元將在多個應用程式伺服器上執行更新並使用確定控制，則完成更新的所有連線應使用確定控制來建立。如果完成的連線未使用確定控制且稍後執行了可確定的更新，則可能產生工作單元的唯讀連線。
- 其它非 SQL 確定資源 (例如本端檔案、DDM 檔案和確定控制 API 資源) 會影響連線的可更新及唯讀狀態。
- 如果使用確定控制連接不支援分散式工作單元 (例如，使用 TCP/IP 的 V4R5 iSeries) 的應用程式伺服器，則該連線爲可更新或唯讀。如果連線爲可更新，則它是唯一可更新連線。

結束連線

由於遠端連線使用到一些資源，所以不再使用的連線應儘快結束。連線可以隱含或明確地結束。有關隱含地結束連線的說明，請參閱第 311 頁的『預設啓動群組的隱含連線管理』 和第 311 頁的『非預設啓動群組的隱含連線管理』。在 DISCONNECT 陳述式或 RELEASE 陳述式後面接著成功的 COMMIT，即可明確地結束連線。DISCONNECT 陳述式只能搭配使用不受支援的連線之連線使用，或搭配本端連線使用。執行 DISCONNECT 陳述式可結束連線。RELEASE 陳述式可搭配受保護或未受保護的連線使用。當執行 RELEASE 陳述式時，不會結束連線而是進入釋放狀態。處於釋放狀態中的連線仍可使用。除非執行成功的 COMMIT，否則不會結束連線。ROLLBACK 或失敗的 COMMIT 不會結束釋放狀態中的連線。

當建立遠端 SQL 連線時，會使用 DDM 網路連線 (APPC 交談或 TCP/IP 連線)。當結束 SQL 連線時，網路連線可能處於未使用狀態或被捨棄。網路連線是否被捨棄或處於未使用狀態，視 DDMCNV 工作屬性而定。如果工作屬性值爲 *KEEP 且連線到另一個 iSeries 伺服器，連線成爲未使用的。如果工作屬性值爲 *DROP 且連線到另一個 iSeries 伺服器，則捨棄此連線。如果連線到非 iSeries 伺服器，則一律捨棄連線。在下列狀況中，*DROP 是可行的：

- 維護未用連線的成本高且近期內將不使用連線。
- 當執行混合的程式時，部份程式以 RUW 連線管理編譯，且部份程式以 DUW 連線管理編譯。當受保護的連線存在時，嘗試對遠端位置執行以 RUW 連線管理編譯的程式將失敗。
- 使用 DDM 或 DRDA 執行受保護的連線。未用受保護的連線之確定和回轉需要額外執行時間。

Reclaim DDM connections (RCLDDMCNV) 指令可用來結束所有未用連線，如果它們是在確定界限上。

分散式工作單元

分散式工作單元 (DUW) 允許存取相同工作單元內多個應用程式伺服器。每一個 SQL 陳述式僅能存取一個應用程式伺服器。使用分散式工作單元可確定或回轉單一工作單元內多個應用程式伺服器的變更。

管理分散式工作單元連線

使用 CONNECT、SET CONNECTION、DISCONNECT 及 RELEASE 陳述式，來管理 DUW 環境中的連線。當程式使用 RDBCNNMTH(*DUW) 來前置編譯後，會執行分散式工作單元 CONNECT，此爲預設值。此形式的 CONNECT 陳述式不切斷現有的連線，而是將上一個連線置於靜止狀態中。CONNECT 陳述式上指定的關聯式資料庫變成現行連線。 CONNECT 陳述式只能用來啓動新連線；如果您要在現有的連線之間切換，則必須使用 SET CONNECTION 陳述式。由於連線會使用到系統資源，所以應結束不再使用的連線。 RELEASE 或 DISCONNECT 陳述式可用來結束連線。RELEASE 陳述式後必須是成功的確定，如此才能結束連線。

下列爲執行於 DUW 環境中使用確定控制的 C 程式範例。

```
....
 EXEC SQL WHENEVER SQLERROR GO TO done;
 EXEC SQL WHENEVER NOT FOUND GO TO done;
....
 EXEC SQL
   DECLARE C1 CURSOR WITH HOLD FOR
       SELECT PARTNO, PRICE
            FROM PARTS
            WHERE SITES_UPDATED = 'N'
            FOR UPDATE OF SITES_UPDATED;
/*    Connect to the systems      */
EXEC SQL CONNECT TO LOCALSYS;
EXEC SQL CONNECT TO SYSB;
EXEC SQL CONNECT TO SYSC;
/* Make the local system the current connection */
EXEC SQL SET CONNECTION LOCALSYS;
/* Open the cursor */
EXEC SQL OPEN C1;
```

圖 12. 分散式工作單元程式範例 (1/4)

```
while (SQLCODE==0)
   {
    /* Fetch the first row */
    EXEC SQL FETCH C1 INTO :partnumber,:price;
    /* Update the row which indicates that the updates have been
       propagated to the other sites */
    EXEC SQL UPDATE PARTS SET SITES_UPDATED='Y'
                WHERE CURRENT OF C1;
    /* Check if the part data is on SYSB */
    if ((partnumber > 10) && (partnumber < 100))
       {
        /* Make SYSB the current connection and update the price */
        EXEC SQL SET CONNECTION SYSB;
        EXEC SQL UPDATE PARTS
                    SET PRICE=:price
                    WHERE PARTNO=:partnumber;
        }
```

圖 12. 分散式工作單元程式範例 (2/4)

```
        /* Check if the part data is on SYSC */
        if ((partnumber > 50) && (partnumber < 200))
           {
            /* Make SYSC the current connection and update the price */
            EXEC SQL SET CONNECTION SYSC;
            EXEC SQL UPDATE PARTS
                        SET PRICE=:price
                        WHERE PARTNO=:partnumber;
           }
        /* Commit the changes made at all 3 sites */
        EXEC SQL COMMIT;
        /* Set the current connection to local so the next row
           can be fetched */
        EXEC SQL SET CONNECTION LOCALSYS;
       }
    done:
```

圖 12. 分散式工作單元程式範例 (3/4)

```
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    /* Release the connections that are no longer being used */
    EXEC SQL RELEASE SYSB;
    EXEC SQL RELEASE SYSC;
    /* Close the cursor */
    EXEC SQL CLOSE C1;
    /* Do another commit which will end the released connections.
       The local connection is still active because it was not
       released. */
    EXEC SQL COMMIT;
   ...
```

圖 12. 分散式工作單元程式範例 (4/4)

在這個程式中，有 3 個應用程式伺服器在作用中：一個本端系統 LOCALSYS 和兩個遠端系統 SYSB 和 SYSC。SYSB 及 SYSC 也支援分散式工作單元和兩階段確定。一開始，涉及異動的每一個應用程式伺服器之所有連線是使用 CONNECT 陳述式而成爲作用中連線。當使用 DUW 時，CONNECT 陳述式不切斷上一個連線，而是將它置於靜止狀態。連接所有應用程式伺服器後，本端連線使用 SET CONNECTION 陳述式建立現行連線。然後，開啓游標並提取第一列資料。接著決定哪些應用程式伺服器需要更新資料。如果需要更新 SYSB，則 SYSB 使用 SET CONNECTION 陳述式建立現行連線並執行更新。對 SYSC 採取相同步驟。然後確定變更。由於使用了兩階段確定，所以保證會在本端系統及兩個遠端系統上確定變更。由於宣告游標 WITH HOLD，所以確定後仍爲開啓。現行連線則變更爲本端系統，以便提取下一列資料。重複這組提取、更新及確定，直到所有資料處理完畢爲止。提取所有資料之後，兩遠端系統的連線會被釋放。它們無法被斷線，因爲它們是使用受保護的連線。釋放連線後，便發出確定來結束連線。仍連接著本端系統並繼續進行處理。

檢查連線狀態

如果執行於可能有唯讀連線的環境中，則在執行可確定的更新前，應檢查連線狀態。這可防止工作單元進入需要回轉的狀態。下列 COBOL 範例顯示如何檢查連線狀態。

```
   ...
    EXEC SQL
     SET CONNECTION SYS5
    END-EXEC.
    ...
* Check if the connection is updateable.
     EXEC SQL CONNECT END-EXEC.
* If connection is updateable, update sales information otherwise
* inform the user.
     IF SQLERRD(3) = 1 THEN
       EXEC SQL
         INSERT INTO SALES_TABLE
             VALUES(:SALES-DATA)
        END-EXEC
      ELSE
        DISPLAY 'Unable to update sales information at this time'.
...
```

圖 13. 檢查連線狀態的範例

游標與備妥的陳述式

游標和備妥的陳述式被限定在編譯單元和連線的範圍內。限定編譯單元的範圍，意指從另一個獨立編譯的程式呼叫的程式無法使用呼叫程式所開啓或準備的游標或備妥的陳述式。限定連線範圍，意指程式內每一個連線可擁有自己的個別游標或備妥的陳述式案例。

下列分散式工作單元範例顯示如何在兩個游標 C1 案例產生的兩個不同連線中，開啓相同游標名稱。

```
.....
 EXEC SQL DECLARE C1 CURSOR FOR
            SELECT * FROM CORPDATA.EMPLOYEE;
 /* Connect to local and open C1 */
 EXEC SQL CONNECT TO LOCALSYS;
 EXEC SQL OPEN C1;
 /* Connect to the remote system and open C1 */
 EXEC SQL CONNECT TO SYSA;
 EXEC SQL OPEN C1;
 /* Keep processing until done */
 while (NOT_DONE) {
    /* Fetch a row of data from the local system */
    EXEC SQL SET CONNECTION LOCALSYS;
    EXEC SQL FETCH C1 INTO :local_emp_struct;
    /* Fetch a row of data from the remote system */
    EXEC SQL SET CONNECTION SYSA;
    EXEC SQL FETCH C1 INTO :rmt_emp_struct;
    /* Process the data */
    .....
 }
 /* Close the cursor on the remote system */
 EXEC SQL CLOSE C1;
 /* Close the cursor on the local system */
 EXEC SQL SET CONNECTION LOCALSYS;
 EXEC SQL CLOSE C1;
 .....
```

圖 14. DUW 程式中游標的範例

應用要求驅動程式

若要補充實作 DRDA 的產品所提供的資料庫存取權，DB2 UDB for iSeries 提供了一個介面，供您在 DB2 UDB for iSeries 應用要求程式上撰寫跳出程式來處理 SQL 要求。這種跳出程式稱爲**應用要求驅動程式**。在下列作業期間，伺服器呼叫 ARD 程式：

- 在使用 CRTSQLPKG 或 CRTSQLxxx 指令執行資料包建立期間，關聯式資料庫 (RDB) 參數符合對應到 ARD 程式的 RDB 名稱。
- 現行連線連接對應到 ARD 程式的 RDB 名稱時，SQL 陳述式的處理程序。

這些呼叫允許 ARD 程式將 SQL 陳述式及其相關資訊傳遞至遠端關聯式資料庫，並傳回結果給系統。系統則傳回結果給應用程式或使用者。存取 ARD 程式所存取的關聯式資料庫就像存取不同環境中的 DRDA 應用程式伺服器一樣。不過，ARD 環境並不支援所有 DRDA 功能。不受支援的功能範例爲大型物件 (LOB) 和長密碼（通行碼）。

有關應用要求驅動程式的詳細資訊，請參閱 OS/400 File API。

問題處理

攫取及報告分散式資料庫功能錯誤資訊的主要策略稱爲**第一個失敗資料攫取 (FFDC)**。FFDC 支援的目的是提供於 OS/400 系統的 DDM 元件中所偵測到錯誤的正確資訊，從該系統中可建立 APAR (授權程式分析報告)。透過此功能，金鑰結構及 DDM 資料串流會自動傾出至排存檔。錯誤資訊的前 1024 位元組也會記錄在系統錯誤日誌中。自動傾出第一個出現錯誤的錯誤資訊，表示客戶所報告的錯誤不應該再出現。 FFDC 同時在 OS/400 DDM 元件的應用要求程式和應用程式伺服器功能中作用。不過，若是要記載的 FFDC 資料，系統值 QSFWERRLOG 必須設爲 *LOG。

註: 並非所有負值 SQLCODE 都會被傾出；僅傾出用來產生 ARAR 的那些 SQLCODE。有關處理分散式關聯資料庫作業問題的詳細資訊，請參閱 *Distributed Database Problem Determination Guide*

當偵測到 SQL 錯誤時，會傳回具有對應 SQLSTATE 的 SQLCODE 到 SQLCA 中。有關這些程式碼的詳細資訊，請參閱 iSeries 資訊中心中的 SQL 訊息與程式碼主題。

DRDA 儲存程序注意事項

iSeries DRDA 伺服器支援從儲存程序傳回一或多個結果集。不過，請注意：於 V5R1 中，僅提供伺服器賦能，以便只從支援儲存程序結果集的非 iSeries 從屬站使用該功能。

於 V5R2 中，針對使用 SQL 的 CLI 介面的應用程式加入了從屬端支援。不過，您必須套用 PTF 來啓用 V5R1 iSeries 伺服器，才能將儲存程序結果集傳回 V5R2 iSeries 從屬站。請參閱 PTF 說明函資料庫  ，以查閱按版次、日期或修訂碼排序的說明函清單。

藉由開啓與 SQL SELECT 陳述式相關的一或多個 SQL 游標，可在儲存程序中產生結果集。此外，也可傳回一個陣列結果集的最大值。有關撰寫傳回結果集的儲存程序詳細資訊，請參閱 SQL Reference 一書中 SET RESULT SETS、 CREATE PROCEDURE (SQL) 及 CREATE PROCEDURE (External) 陳述式的說明。有關搭配使用儲存程序與

DRDA 的一般資訊，請參閱 Distributed Database Programming 一書。

附錄 A. DB2 UDB for iSeries 表格範例

本附錄包含本手冊和 SQL Reference 一書中提到和用到的表格範例。表格旁附有建立表格的 SQL 陳述式。關於建立表格的詳細資訊，請參閱第 14 頁的『建立及使用表格』。

這組表格包含的資訊說明員工、部門、專案及活動。此資訊組成一個範例應用程式，示範 DB2 UDB Query Manager and SQL Development Kit 授權程式的部份特性。所有範例皆假設表格是在 CORPDATA 綱目中 (表示公司資料)。

儲存程序隨系統附上，包含建立這些表格的 DDL 陳述式，以及大量輸入資料的 INSERT 陳述式。此程序將建立呼叫程序時指定的綱目。因爲這是一個 SQL 外部儲存程序，所以可從任何 SQL 介面來呼叫，包括交談式 SQL 和「iSeries 領航員」。若要在呼叫程序中建立您要的綱目 *SAMPLE*，請發出下列陳述式：

```
CALL QSYS.CREATE_SQL_SAMPLE ('SAMPLE')
```

綱目名稱必須指定爲大寫。綱目必須不存在。

表格如下：

- 『部門表格 (DEPARTMENT)』
- 第 325 頁的『員工表格 (EMPLOYEE)』
- 第 327 頁的『員工照片表格 (EMP_PHOTO)』
- 第 328 頁的『員工履歷表格 (EMP_RESUME)』
- 第 329 頁的『員工專案活動表格 (EMPPROJACT)』
- 第 332 頁的『專案表格 (PROJECT)』
- 第 334 頁的『專案活動表格 (PROJACT)』
- 第 336 頁的『活動表格 (ACT)』
- 第 337 頁的『班別排程表格 (CL_SCHED)』
- 第 338 頁的『收件匣表格 (IN_TRAY)』

對於其中的許多表格會建立索引、別名及概略表。此處不含概略表定義。

也會另外建立三個與第一組無關的表格。

- 第 339 頁的『組織表格 (ORG)』
- 第 340 頁的『職員表格 (STAFF)』
- 第 341 頁的『銷售表格 (SALES)』

註:

1. 在這些表格範例中，問號 (?) 表示 NULL 值。

部門表格 (DEPARTMENT)

部門表格說明企業中的每一個部門，並定義其主管及其向上報告的部門。部門表格是使用下列 CREATE TABLE 和 ALTER TABLE 陳述式來建立：

```
CREATE TABLE DEPARTMENT
      (DEPTNO    CHAR(3)           NOT NULL,
       DEPTNAME  VARCHAR(36)       NOT NULL,
       MGRNO     CHAR(6)                    ,
       ADMRDEPT  CHAR(3)           NOT NULL
       LOCATION  CHAR(16),
       PRIMARY KEY (DEPTNO))

ALTER TABLE DEPARTMENT
      ADD FOREIGN KEY ROD (ADMRDEPT)
          REFERENCES DEPARTMENT
          ON DELETE CASCADE
```

稍後新增下列外來索引鍵

```
ALTER TABLE DEPARTMENT
      ADD FOREIGN KEY RDE (MGRNO)
          REFERENCES EMPLOYEE
          ON DELETE SET NULL
```

建立下列索引：

```
CREATE UNIQUE INDEX XDEPT1
       ON DEPARTMENT (DEPTNO)

CREATE INDEX XDEPT2
       ON DEPARTMENT (MGRNO)

CREATE INDEX XDEPT3
       ON DEPARTMENT (ADMRDEPT)
```

爲表格建立下列別名：

```
CREATE ALIAS DEPT FOR DEPARTMENT
```

下列表格顯示直欄的內容：

表 41. 部門表格的直欄

直欄名稱	說明
DEPTNO	部門編號或 ID。
DEPTNAME	名稱，說明部門的一般活動。
MGRNO	部門主管的員工編號 (EMPNO)。
ADMRDEPT	此部門要向上報告的部門 (DEPTNO)；最上層的部門向自己報告。
LOCATION	部門的位置。

關於完整的 DEPARTMENT 清單，請參閱『DEPARTMENT』。

DEPARTMENT

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	?
B01	PLANNING	000020	A00	?
C01	INFORMATION CENTER	000030	A00	?
D01	DEVELOPMENT CENTER	?	A00	?
D11	MANUFACTURING SYSTEMS	000060	D01	?

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
D21	ADMINISTRATION SYSTEMS	000070	D01	?
E01	SUPPORT SERVICES	000050	A00	?
E11	OPERATIONS	000090	E01	?
E21	SOFTWARE SUPPORT	000100	E01	?
F22	BRANCH OFFICE F2	?	E01	?
G22	BRANCH OFFICE G2	?	E01	?
H22	BRANCH OFFICE H2	?	E01	?
I22	BRANCH OFFICE I2	?	E01	?
J22	BRANCH OFFICE J2	?	E01	?

員工表格 (EMPLOYEE)

員工表格使用員工編號來定義所有員工，並列出基本人事資訊。員工表格是使用下列 CREATE TABLE 和 ALTER TABLE 陳述式來建立：

```
CREATE TABLE EMPLOYEE
      (EMPNO       CHAR(6)         NOT NULL,
       FIRSTNME    VARCHAR(12)     NOT NULL,
       MIDINIT     CHAR(1)         NOT NULL,
       LASTNAME    VARCHAR(15)     NOT NULL,
       WORKDEPT    CHAR(3)                 ,
       PHONENO     CHAR(4)                 ,
       HIREDATE    DATE                    ,
       JOB         CHAR(8)                 ,
       EDLEVEL     SMALLINT        NOT NULL,
       SEX         CHAR(1)                 ,
       BIRTHDATE   DATE                    ,
       SALARY      DECIMAL(9,2)            ,
       BONUS       DECIMAL(9,2)            ,
       COMM        DECIMAL(9,2)
       PRIMARY KEY (EMPNO))

ALTER TABLE EMPLOYEE
      ADD FOREIGN KEY RED (WORKDEPT)
      REFERENCES DEPARTMENT
      ON DELETE SET NULL

ALTER TABLE EMPLOYEE
      ADD CONSTRAINT NUMBER
      CHECK (PHONENO >= '0000' AND PHONENO <= '9999')
```

建立下列索引：

```
CREATE UNIQUE INDEX XEMP1
       ON EMPLOYEE (EMPNO)

CREATE INDEX XEMP2
       ON EMPLOYEE (WORKDEPT)
```

爲表格建立下列別名：

```
CREATE ALIAS EMP FOR EMPLOYEE
```

下列表格顯示直欄的內容。

直欄名稱	說明
EMPNO	員工編號
FIRSTNME	員工的姓名
MIDINIT	員工的中間名字之首字母
LASTNAME	員工的姓氏
WORKDEPT	員工的部門 ID
PHONENO	員工電話號碼
HIREDATE	受僱日期
JOB	員工從事的工作
EDLEVEL	正規教育年數
SEX	員工的性別 (M 或 F)
BIRTHDATE	出生日期
SALARY	年薪 (美元)
BONUS	年度紅利 (美元)
COMM	年度佣金 (美元)

關於完整的 EMPLOYEE 清單，請參閱第 327 頁的『EMPLOYEE』。

EMPLOYEE

EMP NO	FIRST NAME	MID INIT	LASTNAME	WORK DEPT	PHONE NO	HIRE DATE	JOB	ED LEVEL	SEX	BIRTH DATE	SAL-ARY	BONUS	COMM
000010	CHRISTINE	I	HAAS	A00	3978	1965-01-01	PRES	18	F	1933-08-24	52750	1000	4220
000020	MICHAEL	L	THOMPSON	B01	3476	1973-10-10	MANAGER	18	M	1948-02-02	41250	800	3300
000030	SALLY	A	KWAN	C01	4738	1975-04-05	MANAGER	20	F	1941-05-11	38250	800	3060
000050	JOHN	B	GEYER	E01	6789	1949-08-17	MANAGER	16	M	1925-09-15	40175	800	3214
000060	IRVING	F	STERN	D11	6423	1973-09-14	MANAGER	16	M	1945-07-07	32250	500	2580
000070	EVA	D	PULASKI	D21	7831	1980-09-30	MANAGER	16	F	1953-05-26	36170	700	2893
000090	EILEEN	W	HENDERSON	E11	5498	1970-08-15	MANAGER	16	F	1941-05-15	29750	600	2380
000100	THEODORE	Q	SPENSER	E21	0972	1980-06-19	MANAGER	14	M	1956-12-18	26150	500	2092
000110	VINCENZO	G	LUCCHESSI	A00	3490	1958-05-16	SALESREP	19	M	1929-11-05	46500	900	3720
000120	SEAN		O'CONNELL	A00	2167	1963-12-05	CLERK	14	M	1942-10-18	29250	600	2340
000130	DOLORES	M	QUINTANA	C01	4578	1971-07-28	ANALYST	16	F	1925-09-15	23800	500	1904
000140	HEATHER	A	NICHOLLS	C01	1793	1976-12-15	ANALYST	18	F	1946-01-19	28420	600	2274
000150	BRUCE		ADAMSON	D11	4510	1972-02-12	DESIGNER	16	M	1947-05-17	25280	500	2022
000160	ELIZABETH	R	PIANKA	D11	3782	1977-10-11	DESIGNER	17	F	1955-04-12	22250	400	1780
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1978-09-15	DESIGNER	16	M	1951-01-05	24680	500	1974
000180	MARILYN	S	SCOUTTEN	D11	1682	1973-07-07	DESIGNER	17	F	1949-02-21	21340	500	1707
000190	JAMES	H	WALKER	D11	2986	1974-07-26	DESIGNER	16	M	1952-06-25	20450	400	1636
000200	DAVID		BROWN	D11	4501	1966-03-03	DESIGNER	16	M	1941-05-29	27740	600	2217
000210	WILLIAM	T	JONES	D11	0942	1979-04-11	DESIGNER	17	M	1953-02-23	18270	400	1462
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29	DESIGNER	18	F	1948-03-19	29840	600	2387
000230	JAMES	J	JEFFERSON	D21	2094	1966-11-21	CLERK	14	M	1935-05-30	22180	400	1774
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05	CLERK	17	M	1954-03-31	28760	600	2301
000250	DANIEL	S	SMITH	D21	0961	1969-10-30	CLERK	15	M	1939-11-12	19180	400	1534
000260	SYBIL	P	JOHNSON	D21	8953	1975-09-11	CLERK	16	F	1936-10-05	17250	300	1380
000270	MARIA	L	PEREZ	D21	9001	1980-09-30	CLERK	15	F	1953-05-26	27380	500	2190
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24	OPERATOR	17	F	1936-03-28	26250	500	2100
000290	JOHN	R	PARKER	E11	4502	1980-05-30	OPERATOR	12	M	1946-07-09	15340	300	1227
000300	PHILIP	X	SMITH	E11	2095	1972-06-19	OPERATOR	14	M	1936-10-27	17750	400	1420
000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12	OPERATOR	12	F	1931-04-21	15900	300	1272
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07	FILEREP	16	M	1932-08-11	19950	400	1596
000330	WING		LEE	E21	2103	1976-02-23	FILEREP	14	M	1941-07-18	25370	500	2030
000340	JASON	R	GOUNOT	E21	5698	1947-05-05	FILEREP	16	M	1926-05-17	23840	500	1907
200010	DIAN	J	HEMMINGER	A00	3978	1965-01-01	SALESREP	18	F	1933-08-14	46500	1000	4220
200120	GREG		ORLANDO	A00	2167	1972-05-05	CLERK	14	M	1942-10-18	29250	600	2340
200140	KIM	N	NATZ	C01	1793	1976-12-15	ANALYST	18	F	1946-01-19	28420	600	2274
200170	KIYOSHI		YAMAMOTO	D11	2890	1978-09-15	DESIGNER	16	M	1951-01-05	24680	500	1974
200220	REBA	K	JOHN	D11	0672	1968-08-29	DESIGNER	18	F	1948-03-19	29840	600	2387
200240	ROBERT	M	MONTEVERDE	D21	3780	1979-12-05	CLERK	17	M	1954-03-31	28760	600	2301
200280	EILEEN	R	SCHWARTZ	E11	8997	1967-03-24	OPERATOR	17	F	1936-03-28	26250	500	2100
200310	MICHELLE	F	SPRINGER	E11	3332	1964-09-12	OPERATOR	12	F	1931-04-21	15900	300	1272
200330	HELENA		WONG	E21	2103	1976-02-23	FIELDREP	14	F	1941-07-18	25370	500	2030
200340	ROY	R	ALONZO	E21	5698	1947-05-05	FIELDREP	16	M	1926-05-17	23840	500	1907

員工照片表格 (EMP_PHOTO)

員工照片表格包含一張以員工編號儲存的員工照片。員工照片表格是使用下列 CREATE TABLE 和 ALTER TABLE 陳述式來建立：

```
CREATE TABLE EMP_PHOTO
            (EMPNO CHAR(6) NOT NULL,
             PHOTO_FORMAT VARCHAR(10) NOT NULL,
             PICTURE BLOB(100K),
             EMP_ROWID CHAR(40) NOT NULL DEFAULT '',
             PRIMARY KEY (EMPNO,PHOTO_FORMAT))

ALTER TABLE EMP_PHOTO
           ADD COLUMN DL_PICTURE DATALINK(1000)
               LINKTYPE URL NO LINK CONTROL

ALTER TABLE EMP_PHOTO
             ADD FOREIGN KEY (EMPNO)
             REFERENCES EMPLOYEE
             ON DELETE RESTRICT
```

建立下列索引：

```
CREATE UNIQUE INDEX XEMP_PHOTO
            ON EMP_PHOTO (EMPNO,PHOTO_FORMAT)
```

下列表格顯示直欄的內容。

直欄名稱	說明
EMPNO	員工編號
PHOTO_FORMAT	PICTURE 儲存的影像格式
PICTURE	照片影像
EMP_ROWID	唯一的列 ID，目前不使用

關於完整的 EMP_PHOTO 清單，請參閱『EMP_PHOTO』。

EMP_PHOTO

EMPNO	PHOTO_FORMAT	PICTURE	EMP_ROWID
000130	bitmap	?	
000130	gif	?	
000140	bitmap	?	
000140	gif	?	
000150	bitmap	?	
000150	gif	?	
000190	bitmap	?	
000190	gif	?	

員工履歷表格 (EMP_RESUME)

員工照片表格包含一份以員工編號儲存的履歷表。員工履歷表格是使用下列 CREATE TABLE 和 ALTER TABLE 陳述式來建立：

```
CREATE TABLE EMP_RESUME
            (EMPNO CHAR(6) NOT NULL,
             RESUME_FORMAT VARCHAR(10) NOT NULL,
             RESUME CLOB(5K),
             EMP_ROWID CHAR(40) NOT NULL DEFAULT '',
             PRIMARY KEY (EMPNO,RESUME_FORMAT))

ALTER TABLE EMP_RESUME
           ADD COLUMN DL_RESUME DATALINK(1000)
               LINKTYPE URL NO LINK CONTROL

ALTER TABLE EMP_RESUME
             ADD FOREIGN KEY (EMPNO)
             REFERENCES EMPLOYEE
             ON DELETE RESTRICT
```

建立下列索引：

```
CREATE UNIQUE INDEX XEMP_RESUME
             ON EMP_RESUME (EMPNO,RESUME_FORMAT)
```

下列表格顯示直欄的內容。

直欄名稱	說明
EMPNO	員工編號
RESUME_FORMAT	RESUME 儲存的文字格式
RESUME	履歷表
EMP_ROWID	唯一的列 ID，目前不使用

關於完整的 EMP_RESUME 清單，請參閱『EMP_RESUME』。

EMP_RESUME

EMPNO	RESUME_FORMAT	RESUME	EMP_ROWID
000130	ascii	?	
000130	html	?	
000140	ascii	?	
000140	html	?	
000150	ascii	?	
000150	html	?	
000190	ascii	?	
000190	html	?	

員工專案活動表格 (EMPPROJACT)

員工專案活動表格定義負責執行每一個專案的每一個活動的員工。表格中亦包含員工的參與等級 (全職或兼職) 和活動的時程表。員工專案活動表格是使用下列 CREATE TABLE 和 ALTER TABLE 陳述式來建立：

```
CREATE TABLE EMPPROJACT
      (EMPNO     CHAR(6)          NOT NULL,
       PROJNO    CHAR(6)          NOT NULL,
       ACTNO     SMALLINT         NOT NULL,
       EMPTIME   DECIMAL(5,2)             ,
       EMSTDATE  DATE                     ,
       EMENDATE  DATE                     )

ALTER TABLE EMPPROJACT
             ADD FOREIGN KEY REPAPA (PROJNO, ACTNO, EMSTDATE)
             REFERENCES PROJACT
             ON DELETE RESTRICT
```

爲表格建立下列別名：

```
CREATE ALIAS EMPACT FOR EMPPROJACT

CREATE ALIAS EMP_ACT FOR EMPPROJACT
```

下列表格顯示直欄的內容。

表 42. 員工專案活動表格的直欄

直欄名稱	說明
EMPNO	員工 ID 號碼

表 42. 員工專案活動表格的直欄 (繼續)

直欄名稱	說明
PROJNO	員工被分派到的專案的 PROJNO
ACTNO	員工在一個專案中被分派到的一個活動的 ID
EMPTIME	員工的全部時間 (0.00 和 1.00 之間) 花在專案上的比例，從 EMSTDATE 到 EMENDATE
EMSTDATE	活動的開始日期
EMENDATE	活動的完成日期

關於完整的 EMPPROJACT 清單，請參閱『EMPPROJACT』。

EMPPROJACT

EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
000010	AD3100	10	.50	1982-01-01	1982-07-01
000070	AD3110	10	1.00	1982-01-01	1983-02-01
000230	AD3111	60	1.00	1982-01-01	1982-03-15
000230	AD3111	60	.50	1982-03-15	1982-04-15
000230	AD3111	70	.50	1982-03-15	1982-10-15
000230	AD3111	80	.50	1982-04-15	1982-10-15
000230	AD3111	180	.50	1982-10-15	1983-01-01
000240	AD3111	70	1.00	1982-02-15	1982-09-15
000240	AD3111	80	1.00	1982-09-15	1983-01-01
000250	AD3112	60	1.00	1982-01-01	1982-02-01
000250	AD3112	60	.50	1982-02-01	1982-03-15
000250	AD3112	60	1.00	1983-01-01	1983-02-01
000250	AD3112	70	.50	1982-02-01	1982-03-15
000250	AD3112	70	1.00	1982-03-15	1982-08-15
000250	AD3112	70	.25	1982-08-15	1982-10-15
000250	AD3112	80	.25	1982-08-15	1982-10-15
000250	AD3112	80	.50	1982-10-15	1982-12-01
000250	AD3112	180	.50	1982-08-15	1983-01-01
000260	AD3113	70	.50	1982-06-15	1982-07-01
000260	AD3113	70	1.00	1982-07-01	1983-02-01
000260	AD3113	80	1.00	1982-01-01	1982-03-01
000260	AD3113	80	.50	1982-03-01	1982-04-15
000260	AD3113	180	.50	1982-03-01	1982-04-15
000260	AD3113	180	1.00	1982-04-15	1982-06-01
000260	AD3113	180	1.00	1982-06-01	1982-07-01
000270	AD3113	60	.50	1982-03-01	1982-04-01
000270	AD3113	60	1.00	1982-04-01	1982-09-01
000270	AD3113	60	.25	1982-09-01	1982-10-15
000270	AD3113	70	.75	1982-09-01	1982-10-15

EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
000270	AD3113	70	1.00	1982-10-15	1983-02-01
000270	AD3113	80	1.00	1982-01-01	1982-03-01
000270	AD3113	80	.50	1982-03-01	1982-04-01
000030	IF1000	10	.50	1982-06-01	1983-01-01
000130	IF1000	90	1.00	1982-10-01	1983-01-01
000130	IF1000	100	.50	1982-10-01	1983-01-01
000140	IF1000	90	.50	1982-10-01	1983-01-01
000030	IF2000	10	.50	1982-01-01	1983-01-01
000140	IF2000	100	1.00	1982-01-01	1982-03-01
000140	IF2000	100	.50	1982-03-01	1982-07-01
000140	IF2000	110	.50	1982-03-01	1982-07-01
000140	IF2000	110	.50	1982-10-01	1983-01-01
000010	MA2100	10	.50	1982-01-01	1982-11-01
000110	MA2100	20	1.00	1982-01-01	1983-03-01
000010	MA2110	10	1.00	1982-01-01	1983-02-01
000200	MA2111	50	1.00	1982-01-01	1982-06-15
000200	MA2111	60	1.00	1982-06-15	1983-02-01
000220	MA2111	40	1.00	1982-01-01	1983-02-01
000150	MA2112	60	1.00	1982-01-01	1982-07-15
000150	MA2112	180	1.00	1982-07-15	1983-02-01
000170	MA2112	60	1.00	1982-01-01	1983-06-01
000170	MA2112	70	1.00	1982-06-01	1983-02-01
000190	MA2112	70	1.00	1982-01-01	1982-10-01
000190	MA2112	80	1.00	1982-10-01	1983-10-01
000160	MA2113	60	1.00	1982-07-15	1983-02-01
000170	MA2113	80	1.00	1982-01-01	1983-02-01
000180	MA2113	70	1.00	1982-04-01	1982-06-15
000210	MA2113	80	.50	1982-10-01	1983-02-01
000210	MA2113	180	.50	1982-10-01	1983-02-01
000050	OP1000	10	.25	1982-01-01	1983-02-01
000090	OP1010	10	1.00	1982-01-01	1983-02-01
000280	OP1010	130	1.00	1982-01-01	1983-02-01
000290	OP1010	130	1.00	1982-01-01	1983-02-01
000300	OP1010	130	1.00	1982-01-01	1983-02-01
000310	OP1010	130	1.00	1982-01-01	1983-02-01
000050	OP2010	10	.75	1982-01-01	1983-02-01
000100	OP2010	10	1.00	1982-01-01	1983-02-01
000320	OP2011	140	.75	1982-01-01	1983-02-01
000320	OP2011	150	.25	1982-01-01	1983-02-01
000330	OP2012	140	.25	1982-01-01	1983-02-01
000330	OP2012	160	.75	1982-01-01	1983-02-01

EMPNO	PROJNO	ACTNO	EMPTIME	EMSTDATE	EMENDATE
000340	OP2013	140	.50	1982-01-01	1983-02-01
000340	OP2013	170	.50	1982-01-01	1983-02-01
000020	PL2100	30	1.00	1982-01-01	1982-09-15

專案表格 (PROJECT)

專案表格說明該公司目前執行的每一個專案。每一列包含的資料包括專案編號、名稱、負責人及排定日期。專案表格是使用下列 CREATE TABLE 和 ALTER TABLE 陳述式來建立：

```
CREATE TABLE PROJECT
      (PROJNO    CHAR(6)         NOT NULL,
       PROJNAME  VARCHAR(24)     NOT NULL DEFAULT,
       DEPTNO    CHAR(3)         NOT NULL,
       RESPEMP   CHAR(6)         NOT NULL,
       PRSTAFF   DECIMAL(5,2)            ,
       PRSTDATE  DATE                    ,
       PRENDATE  DATE                    ,
       MAJPROJ   CHAR(6)                 ,
       PRIMARY KEY (PROJNO))

ALTER TABLE PROJECT
              ADD FOREIGN KEY (DEPTNO)
              REFERENCES DEPARTMENT
              ON DELETE RESTRICT
ALTER TABLE PROJECT
              ADD FOREIGN KEY (RESPEMP)
              REFERENCES EMPLOYEE
              ON DELETE RESTRICT
ALTER TABLE PROJECT
              ADD FOREIGN KEY  RPP (MAJPROJ)
              REFERENCES PROJECT
          ON DELETE CASCADE
```

建立下列索引：

```
CREATE UNIQUE INDEX XPROJ1
              ON PROJECT (PROJNO)

CREATE INDEX XPROJ2
              ON PROJECT (RESPEMP)
```

爲表格建立下列別名：

```
CREATE ALIAS PROJ FOR PROJECT
```

下列表格顯示直欄的內容：

直欄名稱	說明
PROJNO	專案編號
PROJNAME	專案名稱
DEPTNO	負責專案的部門之部門編號
RESPEMP	負責專案的人員之員工編號
PRSTAFF	預估平均人力
PRSTDATE	預估專案開始日期
PRENDATE	預估專案結束日期

直欄名稱	說明
MAJPROJ	控制子專案的專案編號

關於完整的 PROJECT 清單，請參閱『PROJECT』。

PROJECT

PROJNO	PROJNAME	DEPTNO	RESPEMP	PRSTAFF	PRSTDATE	PRENDATE	MAJPROJ
AD3100	ADMIN SERVICES	D01	000010	6.5	1982-01-01	1983-02-01	?
AD3110	GENERAL ADMIN SYSTEMS	D21	000070	6	1982-01-01	1983-02-01	AD3100
AD3111	PAYROLL PROGRAMMING	D21	000230	2	1982-01-01	1983-02-01	AD3110
AD3112	PERSONNEL PROGRAMMING	D21	000250	1	1982-01-01	1983-02-01	AD3110
AD3113	ACCOUNT PROGRAMMING	D21	000270	2	1982-01-01	1983-02-01	AD3110
IF1000	QUERY SERVICES	C01	000030	2	1982-01-01	1983-02-01	?
IF2000	USER EDUCATION	C01	000030	1	1982-01-01	1983-02-01	?
MA2100	WELD LINE AUTOMATION	D01	000010	12	1982-01-01	1983-02-01	?
MA2110	W L PROGRAMMING	D11	000060	9	1982-01-01	1983-02-01	MA2100
MA2111	W L PROGRAM DESIGN	D11	000220	2	1982-01-01	1982-12-01	MA2110
MA2112	W L ROBOT DESIGN	D11	000150	3	1982-01-01	1982-12-01	MA2110
MA2113	W L PROD CONT PROGS	D11	000160	3	1982-02-15	1982-12-01	MA2110
OP1000	OPERATION SUPPORT	E01	000050	6	1982-01-01	1983-02-01	?
OP1010	OPERATION	E11	000090	5	1982-01-01	1983-02-01	OP1000
OP2000	GEN SYSTEMS SERVICES	E01	000050	5	1982-01-01	1983-02-01	?
OP2010	SYSTEMS SUPPORT	E21	000100	4	1982-01-01	1983-02-01	OP2000
OP2011	SCP SYSTEMS SUPPORT	E21	000320	1	1982-01-01	1983-02-01	OP2010
OP2012	APPLICATIONS SUPPORT	E21	000330	1	1982-01-01	1983-02-01	OP2010
OP2013	DB/DC SUPPORT	E21	000340	1	1982-01-01	1983-02-01	OP2010
PL2100	WELD LINE PLANNING	B01	000020	1	1982-01-01	1982-09-15	MA2100

專案活動表格 (PROJACT)

專案活動表格說明該公司目前執行的每一個專案。每一列包含的資料包括專案編號、活動編號及排定日期。專案活動表格是使用下列 CREATE TABLE 和 ALTER TABLE 陳述式來建立：

```
CREATE TABLE PROJACT
            (PROJNO CHAR(6) NOT NULL,
             ACTNO SMALLINT NOT NULL,
             ACSTAFF DECIMAL(5,2),
             ACSTDATE DATE NOT NULL,
             ACENDATE DATE ,
             PRIMARY KEY (PROJNO, ACTNO, ACSTDATE))

ALTER TABLE PROJACT
             ADD FOREIGN KEY RPAP (PROJNO)
             REFERENCES PROJECT
             ON DELETE RESTRICT
```

稍後新增下列外來索引鍵：

```
ALTER TABLE PROJACT
           ADD FOREIGN KEY RPAA (ACTNO)
               REFERENCES ACT
               ON DELETE RESTRICT
```

建立下列索引：

```
CREATE UNIQUE INDEX XPROJAC1
             ON PROJACT (PROJNO, ACTNO, ACSTDATE)
```

下列表格顯示直欄的內容：

直欄名稱	說明
PROJNO	專案編號
ACTNO	活動編號
ACSTAFF	預估平均人力
ACSTDATE	活動開始日期
ACENDATE	活動結束日期

關於完整的 PROJACT 清單，請參閱『PROJACT』。

PROJACT

PROJNO	ACTNO	ACSTAFF	ACSTDATE	ACENDATE
AD3100	10	?	1982-01-01	?
AD3110	10	?	1982-01-01	?
AD3111	60	?	1982-01-01	?
AD3111	60	?	1982-03-15	?
AD3111	70	?	1982-03-15	?
AD3111	80	?	1982-04-15	?
AD3111	180	?	1982-10-15	?
AD3111	70	?	1982-02-15	?
AD3111	80	?	1982-09-15	?

PROJNO	ACTNO	ACSTAFF	ACSTDATE	ACENDATE
AD3112	60	?	1982-01-01	?
AD3112	60	?	1982-02-01	?
AD3112	60	?	1983-01-01	?
AD3112	70	?	1982-02-01	?
AD3112	70	?	1982-03-15	?
AD3112	70	?	1982-08-15	?
AD3112	80	?	1982-08-15	?
AD3112	80	?	1982-10-15	?
AD3112	180	?	1982-08-15	?
AD3113	70	?	1982-06-15	?
AD3113	70	?	1982-07-01	?
AD3113	80	?	1982-01-01	?
AD3113	80	?	1982-03-01	?
AD3113	180	?	1982-03-01	?
AD3113	180	?	1982-04-15	?
AD3113	180	?	1982-06-01	?
AD3113	60	?	1982-03-01	?
AD3113	60	?	1982-04-01	?
AD3113	60	?	1982-09-01	?
AD3113	70	?	1982-09-01	?
AD3113	70	?	1982-10-15	?
IF1000	10	?	1982-06-01	?
IF1000	90	?	1982-10-01	?
IF1000	100	?	1982-10-01	?
IF2000	10	?	1982-01-01	?
IF2000	100	?	1982-01-01	?
IF2000	100	?	1982-03-01	?
IF2000	110	?	1982-03-01	?
IF2000	110	?	1982-10-01	?
MA2100	10	?	1982-01-01	?
MA2100	20	?	1982-01-01	?
MA2110	10	?	1982-01-01	?
MA2111	50	?	1982-01-01	?
MA2111	60	?	1982-06-15	?
MA2111	40	?	1982-01-01	?
MA2112	60	?	1982-01-01	?
MA2112	180	?	1982-07-15	?
MA2112	70	?	1982-06-01	?
MA2112	70	?	1982-01-01	?
MA2112	80	?	1982-10-01	?
MA2113	60	?	1982-07-15	?

PROJNO	ACTNO	ACSTAFF	ACSTDATE	ACENDATE
MA2113	80	?	1982-01-01	?
MA2113	70	?	1982-04-01	?
MA2113	80	?	1982-10-01	?
MA2113	180	?	1982-10-01	?
OP1000	10	?	1982-01-01	?
OP1010	10	?	1982-01-01	?
OP1010	130	?	1982-01-01	?
OP2010	10	?	1982-01-01	?
OP2011	140	?	1982-01-01	?
OP2011	150	?	1982-01-01	?
OP2012	140	?	1982-01-01	?
OP2012	160	?	1982-01-01	?
OP2013	140	?	1982-01-01	?
OP2013	170	?	1982-01-01	?
PL2100	30	?	1982-01-01	?

活動表格 (ACT)

活動表格說明每一個活動。活動表格是使用下列 CREATE TABLE 陳述式來建立：

```
CREATE TABLE ACT
             (ACTNO SMALLINT NOT NULL,
              ACTKWD CHAR(6) NOT NULL,
              ACTDESC VARCHAR(20) NOT NULL,
              PRIMARY KEY (ACTNO))
```

建立下列索引：

```
CREATE UNIQUE INDEX XACT1
              ON ACT (ACTNO)

CREATE UNIQUE INDEX XACT2
              ON ACT (ACTKWD)
```

下列表格顯示直欄的內容。

直欄名稱	說明
ACTNO	活動編號
ACTKWD	活動的關鍵字
ACTDESC	活動的說明

關於完整的 ACT 清單，請參閱『ACT』。

ACT

ACTNO	ACTKWD	ACTDESC
10	MANAGE	MANAGE/ADVISE
20	ECOST	ESTIMATE COST

30	DEFINE	DEFINE SPECS
40	LEADPR	LEAD PROGRAM/DESIGN
50	SPECS	WRITE SPECS
60	LOGIC	DESCRIBE LOGIC
70	CODE	CODE PROGRAMS
80	TEST	TEST PROGRAMS
90	ADMQS	ADM QUERY SYSTEM
100	TEACH	TEACH CLASSES
110	COURSE	DEVELOP COURSES
120	STAFF	PERS AND STAFFING
130	OPERAT	OPER COMPUTER SYS
140	MAINT	MAINT SOFTWARE SYS
150	ADMSYS	ADM OPERATING SYS
160	ADMDB	ADM DATA BASES
170	ADMDC	ADM DATA COMM
180	DOC	DOCUMENT

班別排程表格 (CL_SCHED)

班別排程表格說明：每一班別、班別的開始時間、班別的結束時間及班別代碼。班別排程表格是使用下列 CREATE TABLE 陳述式來建立：

```
CREATE TABLE CL_SCHED
      (CLASS_CODE          CHAR(7),
       "DAY"               SMALLINT,
       STARTING            TIME,
       ENDING              TIME)
```

下列表格顯示直欄的內容。

直欄名稱	說明
CLASS_CODE	班別代碼 (教室：講師)
DAY	4 天時程表的日期編號
STARTING	班別開始時間
ENDING	班別結束時間

關於完整的 CL_SCHED 清單，請參閱『CL_SCHED』。

CL_SCHED

CLASS_CODE	DAY	STARTING	ENDING
042:BF	4	12:10:00	14:00:00
553:MJA	1	10:30:00	11:00:00
543:CWM	3	09:10:00	10:30:00
778:RES	2	12:10:00	14:00:00
044:HD	3	17:12:30	18:00:00

收件匣表格 (IN_TRAY)

收件匣表格說明電子收件箱，包括：收到訊息時的時間戳記、傳送訊息的人員的使用者 ID 以及訊息本身。收件匣表格是使用下列 CREATE TABLE 陳述式來建立：

```
CREATE TABLE IN_TRAY
    (RECEIVED           TIMESTAMP,
     SOURCE             CHAR(8),
     SUBJECT            CHAR(64),
     NOTE_TEXT          VARCHAR(3000))
```

下列表格顯示直欄的內容。

直欄名稱	說明
RECEIVED	收件日期和時間
SOURCE	傳送短文的人員之使用者 ID
SUBJECT	短文的簡短說明
NOTE_TEXT	短文

關於完整的 IN_TRAY 清單，請參閱『IN_TRAY』。

IN_TRAY

RECEIVED	SOURCE	SUBJECT	NOTE_TEXT
1988-12-25-17.12.30.000000	BADAMSON	FWD: Fantastic year! 4th Quarter Bonus.	To: JWALKER Cc: QUINTANA, NICHOLLS Jim, Looks like our hard work has paid off. I have some good beer in the fridge if you want to come over to celebrate a bit. Delores and Heather, are you interested as well? Bruce <Forwarding from ISTERN> Subject: FWD: Fantastic year! 4th Quarter Bonus. To: Dept_D11 Congratulations on a job well done. Enjoy this year's bonus. Irv <Forwarding from CHAAS> Subject: Fantastic year! 4th Quarter Bonus. To: All_Managers Our 4th quarter results are in. We pulled together as a team and exceeded our plan! I am pleased to announce a bonus this year of 18%. Enjoy the holidays. Christine Haas

RECEIVED	SOURCE	SUBJECT	NOTE_TEXT
1988-12-23-08.53.58.000000	ISTERN	FWD: Fantastic year! 4th Quarter Bonus.	To: Dept_D11 Congratulations on a job well done. Enjoy this year's bonus. Irv <Forwarding from CHAAS> Subject: Fantastic year! 4th Quarter Bonus. To: All_Managers Our 4th quarter results are in. We pulled together as a team and exceeded our plan! I am pleased to announce a bonus this year of 18%. Enjoy the holidays. Christine Haas
1988-12-22-14.07.21.136421	CHAAS	Fantastic year! 4th Quarter Bonus.	To: All_Managers Our 4th quarter results are in. We pulled together as a team and exceeded our plan! I am pleased to announce a bonus this year of 18%. Enjoy the holidays. Christine Haas

組織表格 (ORG)

組織表格說明公司的組織。組織表格是使用下列 CREATE TABLE 陳述式來建立：

```
CREATE TABLE ORG
             (DEPTNUMB SMALLINT NOT NULL,
              DEPTNAME VARCHAR(14),
              MANAGER SMALLINT,
              DIVISION VARCHAR(10),
              LOCATION VARCHAR(13))
```

下列表格顯示直欄的內容。

直欄名稱	說明
DEPTNUMB	部門編號
DEPTNAME	部門名稱
MANAGER	部門的主管編號
DIVISION	部門的分區
LOCATION	部門的位置

關於完整的 ORG 清單，請參閱『ORG』。

ORG

DEPTNUMB	DEPTNAME	MANAGER	DIVISION	LOCATION
10	Head Office	160	Corporate	New York
15	New England	50	Eastern	Boston
20	Mid Atlantic	10	Eastern	Washington

DEPTNUMB	DEPTNAME	MANAGER	DIVISION	LOCATION
38	South Atlantic	30	Eastern	Atlanta
42	Great Lakes	100	Midwest	Chicago
51	Plains	140	Midwest	Dallas
66	Pacific	270	Western	San Francisco
84	Mountain	290	Western	Denver

職員表格 (STAFF)

職員表格說明員工。職員表格是使用下列 CREATE TABLE 陳述式來建立：

```
CREATE TABLE STAFF
            (ID  SMALLINT NOT NULL,
             NAME VARCHAR(9),
             DEPT SMALLINT,
             JOB CHAR(5),
             YEARS SMALLINT,
             SALARY DECIMAL(7,2),
             COMM DECIMAL(7,2))
```

下列表格顯示直欄的內容。

直欄名稱	說明
ID	員工編號
NAME	員工名稱
DEPT	部門編號
JOB	工作職稱
YEARS	年資
SALARY	員工的年薪
COMM	員工的佣金

關於完整的 STAFF 清單，請參閱『STAFF』。

STAFF

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
10	Sanders	20	Mgr	7	18357.50	?
20	Pernal	20	Sales	8	18171.25	612.45
30	Marenghi	38	Mgr	5	17506.75	?
40	O’Brien	38	Sales	6	18006.00	846.55
50	Hanes	15	Mgr	10	20659.80	?
60	Quigley	38	Sales	7	16508.30	650.25
70	Rothman	15	Sales	7	16502.83	1152.00
80	James	20	Clerk	?	13504.60	128.20
90	Koonitz	42	Sales	6	18001.75	1386.70
100	Plotz	42	Mgr	7	18352.80	?
110	Ngan	15	Clerk	5	12508.20	206.60

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
120	Naughton	38	Clerk	?	12954.75	180.00
130	Yamaguchi	42	Clerk	6	10505.90	75.60
140	Fraye	51	Mgr	6	21150.00	?
150	Williams	51	Sales	6	19456.50	637.65
160	Molinare	10	Mgr	7	22959.20	?
170	Kermisch	15	Clerk	4	12258.50	110.10
180	Abrahams	38	Clerk	3	12009.75	236.50
190	Sneider	20	Clerk	8	14252.75	126.50
200	Scoutten	42	Clerk	?	11508.60	84.20
210	Lu	10	Mgr	10	20010.00	?
220	Smith	51	Sales	7	17654.50	992.80
230	Lundquist	51	Clerk	3	13369.80	189.65
240	Daniels	10	Mgr	5	19260.25	?
250	Wheeler	51	Clerk	6	14460.00	513.30
260	Jones	10	Mgr	12	21234.00	?
270	Lea	66	Mgr	9	18555.50	?
280	Wilson	66	Sales	9	18674.50	811.50
290	Quill	84	Mgr	10	19818.00	?
300	Davis	84	Sales	5	15454.50	806.10
310	Graham	66	Sales	13	21000.00	200.30
320	Gonzales	66	Sales	4	16858.20	844.00
330	Burke	66	Clerk	1	10988.00	55.50
340	Edwards	84	Sales	7	17844.00	1285.00
3650	Gafney	84	Clerk	5	13030.50	188.00

銷售表格 (SALES)

銷售表格說明：每一位銷售人員的每一筆銷售記錄。銷售表格是使用下列 CREATE TABLE 陳述式來建立：

```
CREATE TABLE SALES
             (SALES_DATE DATE,
              SALES_PERSON VARCHAR(15),
              REGION VARCHAR(15),
              SALES INTEGER)
```

下列表格顯示直欄的內容。

直欄名稱	說明
SALES_DATE	達成銷售交易的日期
SALES_PERSON	完成銷售的人員
REGION	達成銷售交易的區域
SALES	銷售數量

關於完整的 SALES 清單，請參閱第 342 頁的『SALES』。

SALES

SALES_DATE	SALES_PERSON	REGION	SALES
12/31/1995	LUCCHESSI	Ontario-South	1
12/31/1995	LEE	Ontario-South	3
12/31/1995	LEE	Quebec	1
12/31/1995	LEE	Manitoba	2
12/31/1995	GOUNOT	Quebec	1
03/29/1996	LUCCHESSI	Ontario-South	3
03/29/1996	LUCCHESSI	Quebec	1
03/29/1996	LEE	Ontario-South	2
03/29/1996	LEE	Ontario-North	2
03/29/1996	LEE	Quebec	3
03/29/1996	LEE	Manitoba	5
03/29/1996	GOUNOT	Ontario-South	3
03/29/1996	GOUNOT	Quebec	1
03/29/1996	GOUNOT	Manitoba	7
03/30/1996	LUCCHESSI	Ontario-South	1
03/30/1996	LUCCHESSI	Quebec	2
03/30/1996	LUCCHESSI	Manitoba	1
03/30/1996	LEE	Ontario-South	7
03/30/1996	LEE	Ontario-North	3
03/30/1996	LEE	Quebec	7
03/30/1996	LEE	Manitoba	4
03/30/1996	GOUNOT	Ontario-South	2
03/30/1996	GOUNOT	Quebec	18
03/30/1996	GOUNOT	Manitoba	1
03/31/1996	LUCCHESSI	Manitoba	1
03/31/1996	LEE	Ontario-South	14
03/31/1996	LEE	Ontario-North	3
03/31/1996	LEE	Quebec	7
03/31/1996	LEE	Manitoba	3
03/31/1996	GOUNOT	Ontario-South	2
03/31/1996	GOUNOT	Quebec	1
04/01/1996	LUCCHESSI	Ontario-South	3
04/01/1996	LUCCHESSI	Manitoba	1
04/01/1996	LEE	Ontario-South	8
04/01/1996	LEE	Ontario-North	?
04/01/1996	LEE	Quebec	8
04/01/1996	LEE	Manitoba	9
04/01/1996	GOUNOT	Ontario-South	3
04/01/1996	GOUNOT	Ontario-North	1

SALES_DATE	SALES_PERSON	REGION	SALES
04/01/1996	GOUNOT	Quebec	3
04/01/1996	GOUNOT	Manitoba	7

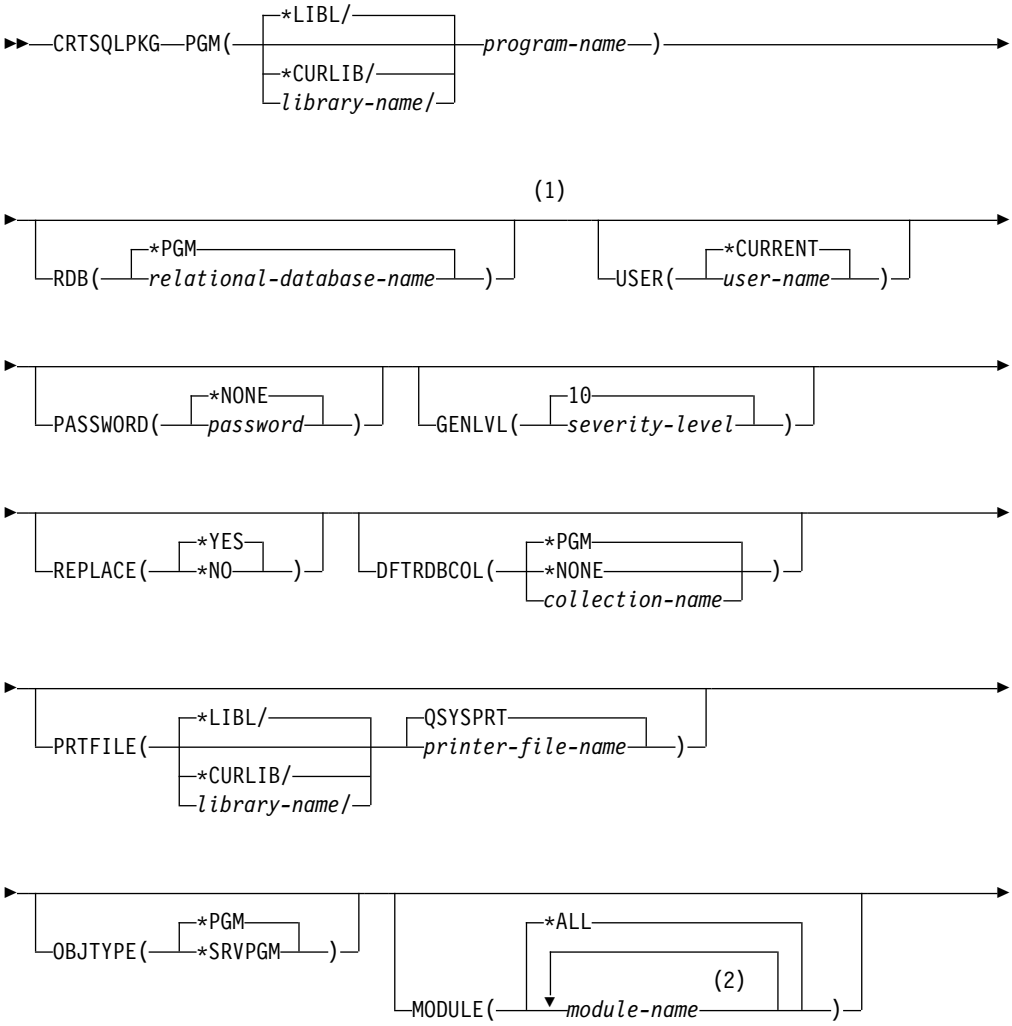
附錄 B. DB2 UDB for iSeries CL 指令說明

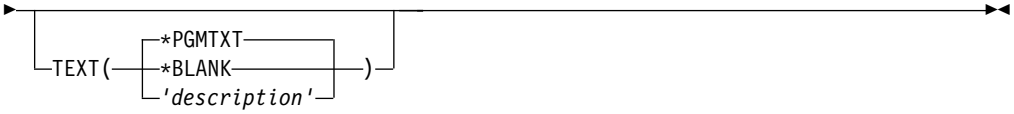
此附錄含有在本書及 SQL Reference 書籍中參照及使用的語法圖。

有關指令說明，請參閱下列主題：

CRTSQLPKG (建立結構化查詢語言資料包) 指令

Job: B,I Pgm: B,I REXX: B,I Exec





註:

1 在此點之前的所有參數，都可以依照位置格式加以指定。

2 最多可以指定 256 個模組。

目的：

「建立結構化查詢語言資料包」(CRTSQLPKG) 指令可從現有的分散式 SQL 程式中建立 (或重建) 關聯式資料庫上的 SQL 資料包。分散式 SQL 程式是在 CRTSQLxxx (其中 xxx = C、CI、CBL、CBLI、 FTN、PLI 或 RPG、RPGI) 指令上指定 RDB 參數來建立的一個程式。

參數：

PGM

指定要建立 SQL 資料包程式的完整名稱。程式必須是分散式 SQL 程式。

程式的名稱可經由下列其中一個檔案庫值來限定：

***LIBL：** 搜尋工作檔案庫清單中的所有檔案庫，直到找到第一個符合的項目爲止。

***CURLIB：** 搜尋工作的現行檔案庫。如果未指定工作的現行檔案庫，則使用 QGPL 檔案庫。

library-name： 指定要搜尋的檔案庫名稱。

*program-name：*指定要建立資料包程式的名稱。

RDB

指定要建立 SQL 資料包的關聯式資料庫的名稱。

***PGM：** 使用爲 SQL 程式所指定的關聯式資料庫名稱。關聯式資料庫名稱是指定於分散式 SQL 程式的 RDB 參數上。

relational-database-name： 指定要建立 SQL 資料包的關聯式資料庫之名稱。使用「使用關聯式資料庫目錄登錄」(WRKRDBDIRE) 指令來顯示在此參數上有效的關聯式資料庫名稱。

USER

指定當開始交談時要傳送到遠端系統的使用者名稱。

***CURRENT：** 使用與現行工作相關的使用者名稱。

*user-name：*指定在應用程式伺服器工作上使用的使用者名稱。

PASSWORD

指定使用於遠端系統上的密碼。

***NONE：**不傳送密碼。如果指定此值，則亦必須指定 USER(*CURRENT)。

*password：*指定在 USER 參數上指定之使用者名稱的密碼。

GENLVL

指定在建立 SQL 資料包期間，偵測錯誤時所容許的最大嚴重性層次。如果錯誤發生的層次超過指定的層次，則不建立 SQL 資料包。

10：預設 severity-level 是 10。

severity-level：指定最大的嚴重性層次。有效值的範圍從 0 到 40。

REPLACE

指定現有的資料包是否要置換成新的資料包。關於此參數的詳細資訊，請參閱 *CL Reference* 一書中的「Appendix A，Expanded Parameter Descriptions」。

***YES**：將現有相同名稱的 SQL 資料包置換成新的 SQL 資料包。

***NO**：不置換現有相同名稱的 SQL 資料包；如果資料包已存在於指定的檔案庫中，則不建立新的 SQL 資料包。

DFTRDBCOL

指定在表格、概略表、索引及 SQL 資料包的不完整名稱上使用的集合名稱。此參數僅適用於資料包中的靜態 SQL 陳述式。

***PGM**：使用爲 SQL 程式所指定的集合名稱。預設關聯式資料庫集合名稱是在分散式 SQL 程式的 DFTRDBCOL 參數上指定的。

***NONE**：表格、概略表、索引及 SQL 資料包的不完整名稱，使用建立程式時的 CRTSQLxxx 指令之 OPTION 參數所指定的搜尋慣例。

collection-name： 指定在不完整表格、概略表、索引及 SQL 資料包上使用的集合名稱。

PRTFILE

指定建立 SQL 資料包錯誤報表要傳送到的印表機裝置檔案之完整名稱。如果建立 SQL 資料包期間未偵測到錯誤，則不產生報表。

印表機檔案的名稱可經由下列其中一個檔案庫值來限定：

***LIBL**：搜尋工作檔案庫清單中的所有檔案庫，直到找到第一個符合的項目爲止。

***CURLIB**：搜尋工作的現行檔案庫。如果未指定工作的現行檔案庫，則使用 QGPL 檔案庫。

library-name：指定要搜尋的檔案庫名稱。

QSYSPRT：如果未指定檔名，則建立 SQL 資料包錯誤報表會傳送到 IBM 提供的印表機檔案 QSYSPRT。

printer-file-name: 指定建立 SQL 資料包錯誤報表要傳送到的印表機裝置檔案的名稱。

OBJTYPE

指定要建立 SQL 資料包的程式類型。

***PGM**： 使用 PGM 參數上指定的程式來建立 SQL 資料包。

***SRVPGM**：使用 PGM 參數上指定的服務程式來建立 SQL 資料包。

MODULE

指定連結程式中的模組清單。

***ALL**：爲程式中的每一個模組建立一個 SQL 資料包。如果程式中沒有任何模組含有 SQL 陳述式或沒有分散式模組，則送出錯誤訊息。

註: CRTSQLPKG 可處理不超過 1024 個模組的程式。

module-name：指定程式中最多 256 個模組的名稱，以建立 SQL 資料包。如果需要建立 SQL 資料包的模組超過 256 個，則必須使用多重 CRTSQLPKG 指令。

相同程式中容許重複的模組名稱。此指令會查詢程式中的每一個模組，如果在 MODULE 參數上指定 *ALL 或模組名稱，則會繼續處理來決定是否要建立 SQL 資料包。如果使用 SQL 來建立模組，且在前置編譯指令上指定 RDB 參數，則會爲模組建立 SQL 資料包。 SQL 資料包會與連結程式的模組結合。

TEXT

指定 SQL 資料包及其功能的簡述文字。

***PGMTXT**：使用要建立 SQL 資料包的程式中的文字。

***BLANK**：不指定文字。

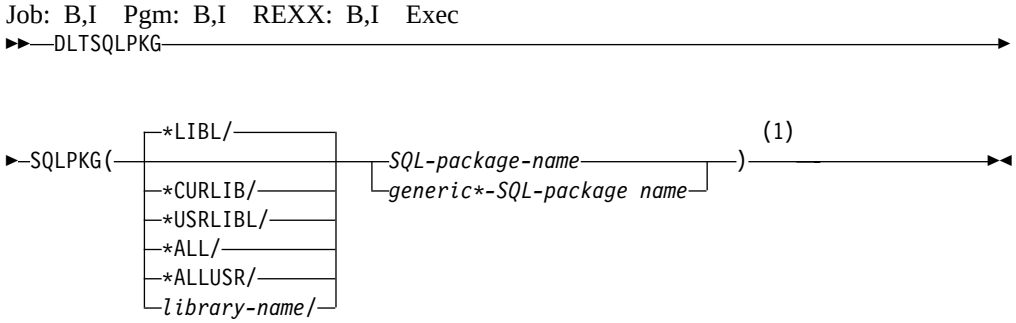
'description'：以單引號含括，指定最多 50 個字元的文字。

範例：

```
CRTSQLPKG   PAYROLL  RDB(SYSTEMA)
  TEXT('Payroll Program')
```

此指令使用關聯式資料庫 SYSTEMA 上的分散式 SQL 程式 PAYROLL 來建立 SQL 資料包。

DLTSQLPKG (刪除結構化查詢語言資料包) 指令



註:

1 在此點之前的所有參數，都可以依照位置格式加以指定。

目的：

「刪除結構化查詢語言資料包」(DLTSQLPKG) 指令可用來刪除一或多個 SQL 資料包。

DLTSQLPKG 是本端指令，必須使用在要刪除 SQL 資料包的 iSeries 系統上。

若要刪除的 SQL 資料包也是位於 iSeries 系統上的遠端系統，請使用「提出遠端指令 (SBMRMTCMD)」指令在遠端系統上執行 DLTSQLPKG 指令。

使用者可執行下列動作來刪除不是 iSeries 系統的遠端系統上的 SQL 資料包：

- 使用交談式 SQL 來執行 CONNECT 和 DROP PACKAGE 作業。
- 登入遠端系統，使用該系統上的本端指令。
- 建立及執行一個含有 DROP PACKAGE SQL 陳述式的 SQL 陳述式。

參數：

SQLPKG

指定要刪除的 SQL 資料包的完整名稱。可以指定特定的或同屬的 SQL 資料包名稱。

SQL 資料包的名稱可經由下列其中一個檔案庫值來限定：

***LIBL**：搜尋工作檔案庫清單中的所有檔案庫，直到找到第一個符合的項目爲止。

***CURLIB**：搜尋工作的現行檔案庫。如果未指定工作的現行檔案庫，則使用 QGPL 檔案庫。

***USRLIBL**：僅搜尋工作檔案庫中使用者部份的檔案庫。

***ALL:**搜尋系統中的所有檔案庫，包括 QSYS。

***ALLUSR**：搜尋所有使用者檔案庫。搜尋不以字母 Q 開頭的所有檔案庫，除了下列檔案庫：

```
#CGULIB     #DFULIB     #RPGLIB     #SEULIB
#COBLIB     #DSULIB     #SDALIB
```

雖然下列 Qxxx 檔案庫由 IBM 提供，但通常包含時常變更的使用者資料。因此，會將這些檔案庫視爲使用者檔案庫來一併搜尋：

```
QDSNX          QRCL          QUSRBRM          QUSRSYS
QGPL           QS36F         QUSRIJS          QUSRVxRxMx
QGPL38         QUSER38       QUSRINFSKR
QPFRDATA       QUSRADSM      QUSRRDARS
```

附註：使用者可對 IBM 支援的每一個版次建立不同的檔案庫名稱，格式爲 QUSRVxRxMx。 VxRxMx 是檔案庫的版本、版次及修正層次。

library-name：指定要搜尋的檔案庫名稱。

SQL-package-name：指定要刪除的 SQL 資料包的名稱。

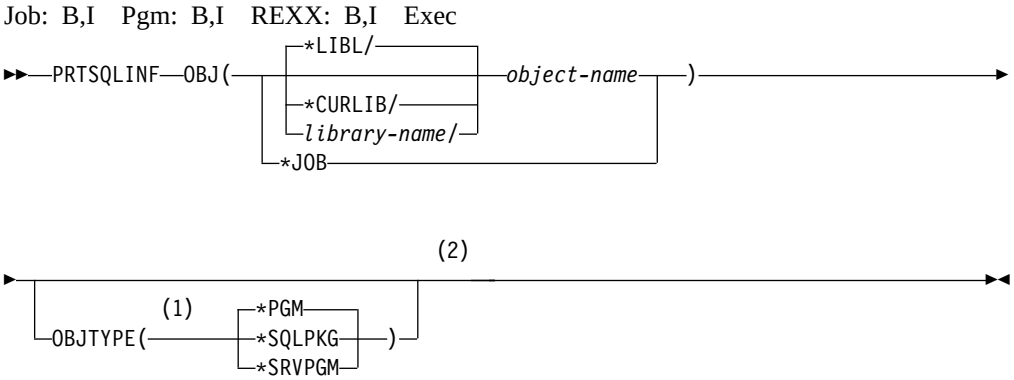
generic-SQL-package-name*：指定要刪除的 SQL 資料包的同屬名稱。同屬名稱是一或多個字元再緊接著一個星號 (*) 的字串；例如，ABC*。如果指定同屬名稱，則會刪除以此同屬名稱開頭且使用者具備權限的所有 SQL 資料包。如果同屬(前置)名稱不含星號，則系統會認爲是完整的 SQL 資料包名稱。

範例：

```
DLTSQLPKG  SQLPKG(JONES)
```

此指令會刪除 SQL 資料包 JONES。

PRTSQLINF (列印結構化查詢語言資訊) 指令



註:

1 指定 OBJ(*JOB) 時，不容許 OBJTYPE 參數。

2 在此點之前的所有參數，都可以依照位置格式加以指定。

目的：

「列印結構化查詢語言資訊 (PRTSQLINF)」指令可列印程式、SQL 資料包、服務程式或工作中關於 SQL 陳述式的資訊。此資訊包含 SQL 陳述式、執行陳述式期間使用的存取規劃、以及前置編譯物件的原始成員期間或執行 SQL 陳述式時所定義的指令參數。

參數：

OBJ

指定您要列印 SQL 資訊的物件的名稱，或指定 *OBJ 來表示要列印工作的 SQL 資訊。指名的物件可以是程式、SQL 資料包或服務程式。

物件的名稱可經由下列其中一個檔案庫值來限定：

***LIBL**：搜尋工作檔案庫清單中的所有檔案庫，直到找到第一個符合的項目爲止。

***CURLIB**：搜尋工作的現行檔案庫。如果未指定工作的現行檔案庫，則使用 QGPL 檔案庫。

library-name：指定要搜尋的檔案庫名稱。

object-name：指定您要列印資訊的程式、SQL 資料包或服務程式的名稱。

- ***JOB**：指出要列印現行工作的 SQL 資訊。

可選用的參數

OBJTYPE

指定物件的類型。

- ***PGM**：物件是一個程式。
- ***SQLPKG**：物件是一個 SQL 資料包。
- ***SRVPGM**：物件是一個服務程式。

範例：

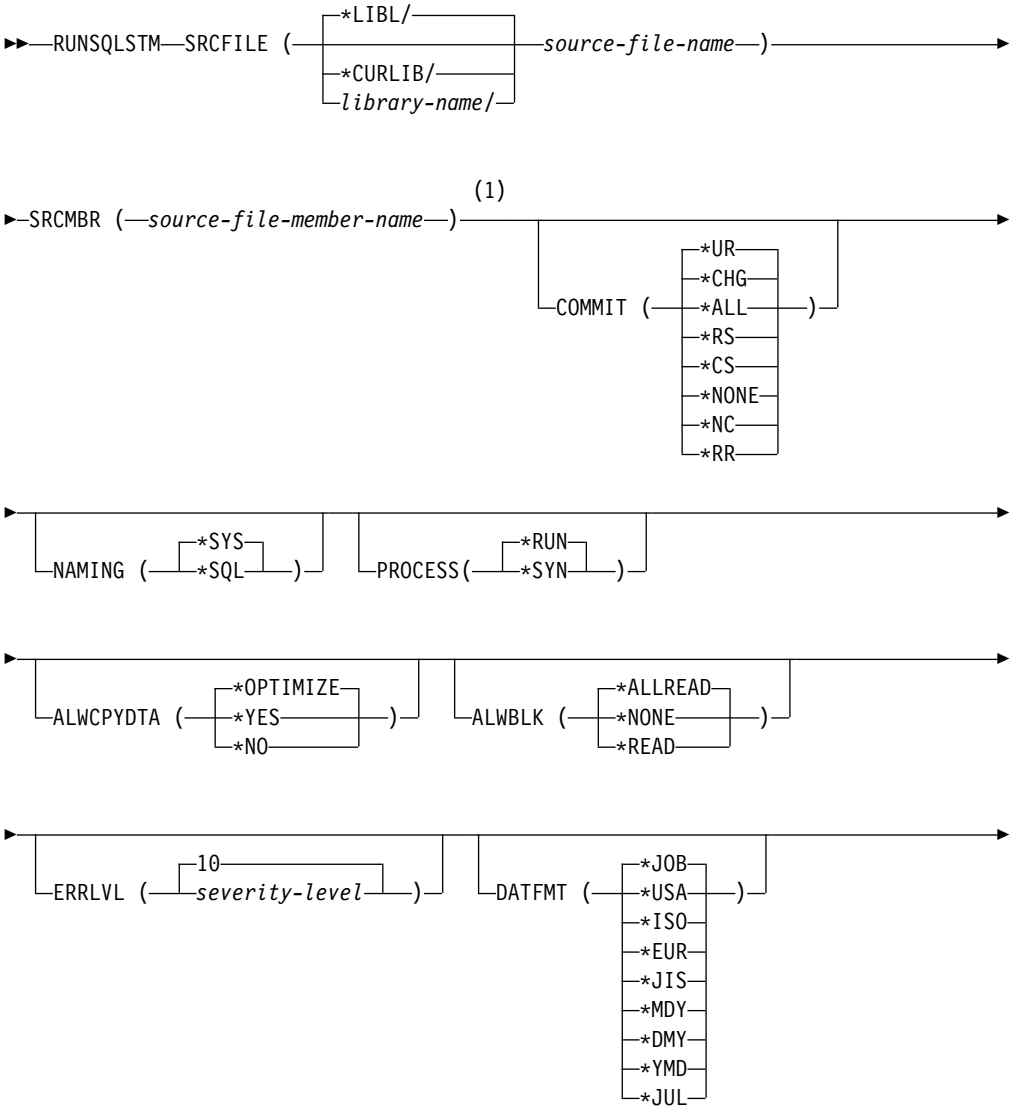
```
PRTSQLINF PAYROLL
```

此指令會列印 PAYROLL 程式中 SQL 陳述式的相關資訊。

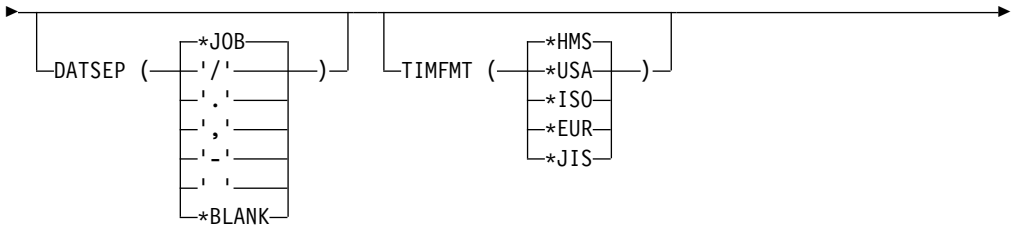
請注意，OBJTYPE 現在是一個相依關鍵字，只有當 OBJ 參數不是 *JOB 時才會出現提示。

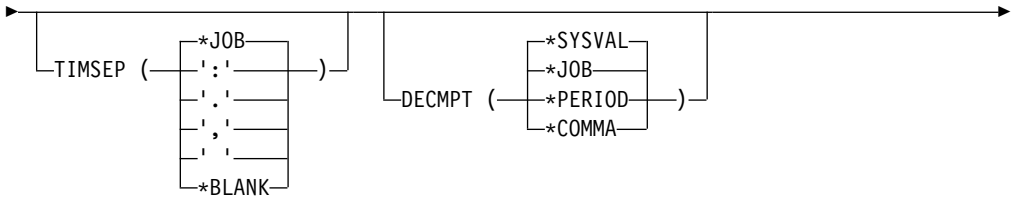
RUNSQLSTM (執行結構化查詢語言陳述式) 指令

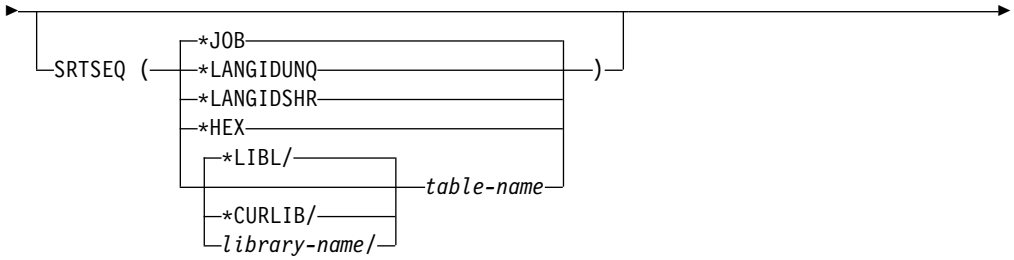
Job: B,I Pgm: B,I REXX: B,I Exec

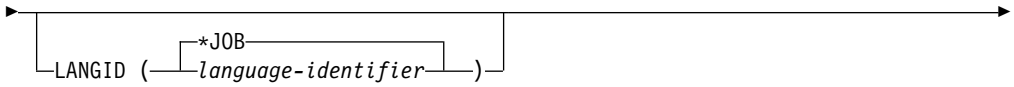


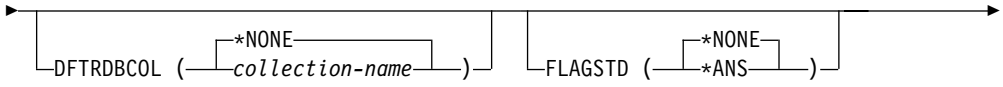
RUNSQLSTM

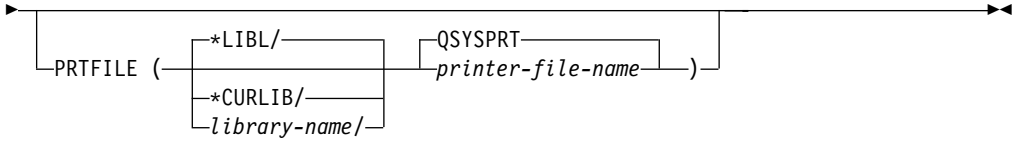




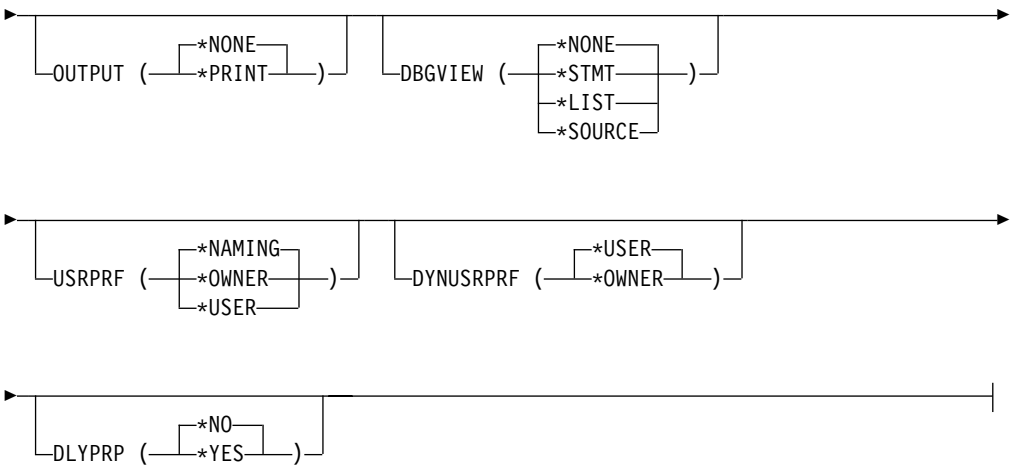








SQL-routine-parameters:



註:

1 在此點之前的所有參數，都可以依照位置格式加以指定。

目的：

「執行結構化查詢語言陳述式 (RUNSQLSTM)」指令可以處理 SQL 陳述式的原始檔。

參數：

SRCFILE

指定原始檔的完整名稱，檔案中含有要執行的 SQL 陳述式。

您可以使用下列其中一個檔案庫值來限定原始檔名稱：

***LIBL：** 搜尋工作檔案庫清單中的所有檔案庫，直到找到第一個相符的檔案庫爲止。

***CURLIB：** 搜尋工作的現行檔案庫。如果沒有指定任何檔案庫作爲工作的現行檔案庫，則會使用 QGPL 檔案庫。

library-name： 指定要搜尋的檔案庫名稱。

source-file-name： 指定原始檔名稱，檔案中含有要執行的 SQL 陳述式。原始檔可以是資料庫檔案或列入資料檔。

SRCMBR

指定原始檔成員的名稱，其中含有要執行的 SQL 陳述式。

COMMIT

指定原始檔中的 SQL 陳述式是否要在確定控制下執行。

***CHG 或 *UR：** 指定在 SQL ALTER、CALL、 COMMENT ON、CREATE、DROP、GRANT、LABEL ON、RENAME 及 REVOKE 陳述式中參照的物件，且會鎖定已更新、刪除及插入的列，直到工作單元 (異動) 結束。您可以看見其他工作中未確定的變更。

***ALL 或 *RS：** 指定在 SQL ALTER、CALL、COMMENT ON、CREATE、DROP、GRANT、LABEL ON、RENAME 及 REVOKE 陳述式中參照的物件，且會鎖定已選取、更新、刪除及插入的列，直到工作單元 (異動) 結束。您無法看見其他工作中未確定的變更。

***CS:** 指定在 SQL ALTER、 CALL、COMMENT ON、CREATE、DROP、GRANT、LABEL ON、RENAME 及 REVOKE 陳述式中參照的物件，且會鎖定已更新、刪除及插入的列，直到工作單元 (異動) 結束。已選取，但未更新的列會被鎖定，直到選取下一列。您無法看見其他工作中未確定的變更。

***NONE 或 *NC：** 指定不使用確定控制。您可以看見其他工作中未確定的變更。如果程式中已併入 SQL DROP COLLECTION 陳述式，則無法指定 *NONE 或 *NC。如果已在 RDB 參數中指定關聯式資料庫，且關聯式資料庫不是在 AS/400 系統上，則無法使用 *NONE 或 *NC。

***RR：** 指定在 SQL ALTER、 CALL、COMMENT ON、CREATE、DROP、GRANT、LABEL ON、RENAME 及 REVOKE 陳述式中參照的物件，且會鎖定已選取、更新、刪除及插入的列，直到工作單元 (異動) 結束。您無法看見其他工作中未確定的變更。 在 SELECT、UPDATE、DELETE 及 INSERT 陳述式中參照的所有表格都會被鎖定，直到工作單元 (異動) 結束。

NAMING

指定用於 SQL 陳述式中命名物件的命名慣例。

***SYS：** 使用系統命名慣例 (檔案庫名稱/檔名)。

***SQL：** 使用 SQL 命名慣例 (集合名稱.表格名稱)。

PROCESS

指定是否要執行原始檔成員中的 SQL 陳述式或只進行語法檢查。

***RUN：** 檢查陳述式的語法，然後執行。

***SYN：** 只檢查陳述式的語法。

ALWCPYDTA

指定是否可以在 SELECT 陳述式中使用資料副本。

***OPTIMIZE：** 系統會決定是否使用直接從資料庫中擷取的資料，或使用資料的副本。決策是取決於哪一個方法可以提供最佳效能。如果 COMMIT 是 *CHG 或 *CS 且 ALWBLK 不是 *ALLREAD，或如果 COMMIT 是 *ALL 或 *RR，則只有在必須執行查詢時才能使用資料副本。

***YES：** 只有在必要時，才能使用資料副本。

***NO：** 不使用資料副本。如果需要資料的暫時副本來執行查詢，則會傳回錯誤訊息。

ALWBLK

指定資料庫管理程式是否可以使用記錄阻礙，及哪一個阻礙的延伸範圍可以用於唯讀游標。

***ALLREAD：** 如果在 COMMIT 參數中指定 *NONE 或 *CHG，則唯讀游標的列會暫停執行。開啓程式中無法明確更新的所有游標以進行唯讀處理，即使程式中可能有 EXECUTE 或 EXECUTE IMMEDIATE 陳述式。

指定 *ALLREAD：

- 除了 *READ 容許的阻礙外，也容許在確定控制層次 *CHG 下的記錄阻礙。
- 可以增進程式中幾乎所有唯讀游標的效能，但以下列方式限制查詢：
  - 指定 *ALLREAD 時，「反轉 (ROLLBACK)」指令、主語言的 ROLLBACK 陳述式或 ROLLBACK HOLD SQL 陳述式不會重新定位唯讀游標。

– 動態執行已定位的 UPDATE 或 DELETE 陳述式 (例如，使用 EXECUTE IMMEDIATE)，不能用來更新游標中的列，除非游標的 DECLARE 陳述式含有 FOR UPDATE 子句。

***NONE**： 未暫停執行列以擷取游標資料。

指定 *NONE：

- 保證擷取的資料是最新的。
- 可以減少擷取查詢的第一列資料時所需的時間量。
- 在結束查詢前，只擷取了前幾列查詢時，停止資料庫管理程式，以避免擷取程式不使用的資料列區塊。
- 可以降低擷取大量列的查詢整體效能。

***READ**： 發生下列情況時，暫停執行唯讀資料擷取記錄：

- 已在 COMMIT 參數中指定 *NONE，它表示不使用確定控制。
- 使用 FOR FETCH ONLY 子句宣告游標，或沒有任何動態陳述式可以執行游標的已定位 UPDATE 或 DELETE 陳述式。

指定 *READ 可以增進符合上述條件且擷取大量記錄的查詢整體效能。

ERRLVL

依據 SQL 陳述式處理程序所產生的訊息嚴重性，指定處理程序是否順利完成。如果在處理程序期間，發生大於此參數指定值的錯誤，即不會再處理任何陳述式，且如果陳述式是在確定控制下執行，則會回復。

10： 當收到的錯誤訊息嚴重性層次大於 10 時，即會停止陳述式處理程序。

嚴重性層次： 指定要使用的嚴重性層次。

DATFMT

指定在存取日期結果直欄時使用的格式。若爲輸入日期字串，則使用指定的值來判定指定的日期格式是否有效。

註: 使用格式 *USA、*ISO、*EUR 或 *JIS 的輸入日期字串一律是有效的。

***JOB**： 使用工作的指定格式。使用「顯示工作 (DSPJOB)」指令以確定工作的現行日期格式。

***USA**： 使用美國日期格式 (mm/dd/yyyy)。

***ISO:** 使用「國際標準組織 (ISO)」日期格式 (yyyy-mm-dd)。

***EUR**： 使用歐洲日期格式 (dd.mm.yyyy)。

***JIS**： 使用「日本工業標準」日期格式 (yyyy-mm-dd)。

***MDY**： 使用日期格式 (mm/dd/yy)。

***DMY**： 使用日期格式 (dd/mm/yy)。

***YMD**： 使用日期格式 (yy/mm/dd)。

***JUL**： 使用羅馬曆日期格式 (yy/ddd)。

DATSEP

指定存取日期結果直欄時使用的分隔字元。

註: 只有在 DATFMT 參數中已指定 *JOB、*MDY、*DMY、*YMD 或 *JUL 時，此參數才適用。

***JOB：** 使用工作的指定日期分隔字元。使用「顯示工作 (DSPJOB)」指令，以確定工作的現行值。

'/'： 使用斜線 (/)。

'.'： 使用句點 (.) 。

','： 使用逗點 (,)。

'-'： 使用破折號 (-)。

' '： 使用空格 ( )。

***BLANK：** 使用空白 ( )。

TIMFMT

指定存取時間結果直欄時使用的格式。若爲輸入時間字串，則使用指定的值來判定指定的時間格式是否有效。

註: 使用格式 *USA、*ISO、*EUR 或 *JIS 的輸入日期字串一律是有效的。

***HMS：** 使用 hh:mm:ss 格式。

***USA：** 使用美國時間格式 hh:mm xx，其中 xx 是 AM 或 PM。

***ISO：** 使用「國際標準組織 (ISO)」時間格式 hh.mm.ss。

***EUR:** 使用「歐洲」時間格式 hh.mm.ss。

***JIS：** 使用「日本工業標準」時間格式 hh:mm:ss。

TIMSEP

指定存取時間結果直欄時使用的分隔字元。

註: 只有在 TIMFMT 參數中已指定 *HMS 時，此參數才適用。

***JOB：** 使用工作的指定時間分隔符號。使用「顯示工作 (DSPJOB)」指令，以確定工作的現行值。

':'： 使用冒號 (:)。

'.'： 使用句點 (.) 。

','： 使用逗點 (,)。

' '： 使用空格 ( )。

***BLANK：** 使用空白 ( )。

DECMPT

指定 SQL 陳述式中數值常數所使用的小數點值。

***JOB**： 當成 SQL 中數值常數的小數點使用的值，代表執行陳述式的工作所指定的小數點。

***SYSVAL**： QDECFMT 系統值是當成小數點使用。

***PERIOD**： 句點代表小數點。

***COMMA**： 逗點代表小數點。

SRTSEQ

指定 SQL 陳述式中字串比較所使用的排序順序表。

***JOB**： 擷取工作的 LANGID 值。

***LANGIDSHR**： 排序順序表使用多個字元的相同權重，且是與 LANGID 參數中指定語言相關的共用權重排序順序表。

***LANGIDUNQ**： 使用 LANGID 參數中指定語言的唯一權重排序表。

***HEX**： 不使用排序順序表。使用字元的十六進位值以決定排序順序。

您可以使用下列其中一個檔案庫值來限定表格名稱：

***LIBL**： 搜尋工作檔案庫清單中的所有檔案庫，直到找到第一個相符的檔案庫爲止。

***CURLIB**： 搜尋工作的現行檔案庫。如果沒有指定任何檔案庫作爲工作的現行檔案庫，則會使用 QGPL 檔案庫。

library-name： 指定要搜尋的檔案庫名稱。

table-name： 指定要使用的排序順序表名稱。

LANGID

指定在已指定 SRTSEQ(*LANGIDUNQ) 或 SRTSEQ(*LANGIDSHR) 時使用的語言 ID。

***JOB**： 前置編譯期間，擷取工作的 LANGID 值。

language-identifier： 指定語言 ID。

DFTRDBCOL

指定表格、概略表、索引及 SQL 資料包的不完整名稱所使用的集合名稱。

***NONE**： 使用 OPTION 參數中所定義的命名慣例。

collection-name： 指定集合 ID 的名稱。使用此值以代替 OPTION 參數中指定的命名慣例。

FLAGSTD

指定「美國國家標準局 (ANSI)」旗號函數。此參數會在 SQL 陳述式上加上旗號，以驗證它們是否符合下列標準。

```
ANSI X3.135-1992 登錄
ISO 9075-1992 登錄
FIPS 127.2 登錄
```

***NONE**： 不檢查 SQL 陳述式以決定它們是否符合 ANSI 標準。

***ANS**： 檢查 SQL 陳述式以決定是否符合 ANSI 標準。

SAAFLAG

指定 IBM SQL 旗號函數。此參數會在 SQL 陳述式上標示旗號，以驗證是否符合 IBM SQL 語法。如需 IBM 資料庫產品 IBM SQL 語法的相關資訊，請參閱 *DRDA IBM SQL Reference* (SC26-3255-00)。

***NOFLAG**： 不檢查 SQL 陳述式以決定是否符合 IBM SQL 語法。

***FLAG**： 檢查 SQL 陳述式以決定是否符合 IBM SQL 語法。

PRTFILE

指定導入 RUNSQLSTM 輸出報表的印表機裝置檔案完整名稱。檔案的最小長度必須爲 132 位元組。如果指定的檔案記錄長度小於 132 位元組，則會遺失資訊。

您可以使用下列其中一個檔案庫值來限定印表機檔案名稱：

***LIBL**： 搜尋工作檔案庫清單中的所有檔案庫，直到找到第一個相符的檔案庫爲止。

***CURLIB**： 搜尋工作的現行檔案庫。如果沒有指定任何檔案庫作爲工作的現行檔案庫，則會使用 QGPL 檔案庫。

library-name： 指定要搜尋的檔案庫名稱。

QSYSPRT： 如果沒有指定檔名，則 RUNSQLSTM 輸出報表會導入 IBM 提供的印表機檔案 QSYSPRT。

printer-file-name：指定導入 RUNSQLSTM 輸出報表的印表機裝置檔案名稱。

SQL 常式的參數：

下面列出的參數僅適用於原始檔中的陳述式，該檔案會建立 SQL 程序、SQL 函數及 SQL 觸發程式。在建立與 SQL 程序、SQL 函數及 SQL 觸發程式相關的程式物件期間，會使用這些參數。

TGTRLS

指定使用者意圖在其中使用正在建立的物件的作業系統版次。

在 *CURRENT 值的給定範例中，及在指定 *release-level* 值時，會使用格式 VxRxMx 來指定版次，其中 Vx 是版本、 Rx 是版次及 Mx 是修正層次。例如，V2R3M0 是版本 2、版次 3、修正層次 0。

***CURRENT** 物件是用於目前在使用者系統上執行的作業系統版次中。例如，如果系統上執行的是 V2R3M5，則 *CURRENT 表示使用者意圖在安裝 V2R3M5 的系統上使用物件。使用者也可以在安裝後續作業系統版次的系統上使用物件。

註: 如果系統上執行的是 V2R3M5，且物件是用於安裝了 V2R3M0 的系統上，則指定 TGTRLS(V2R3M0) 而非 TGRRLS(*CURRENT)。

release-level： 以格式 VxRxMx 指定版次。物件可以用於安裝了指定的作業系統版次，或安裝任何後續版次的系統上。

有效值取決於現行版本、版次及修正層次，且會隨著每一個新版次而變更。如果您指定的版次比此指令支援的最早版次還早，則會傳送一則錯誤訊息，指出最早支援的版次。

CLOSQLCSR

指定何時隱含關閉 SQL 游標、隱含捨棄 SQL 準備的陳述式及釋放 LOCK TABLE 鎖定。當您發出 CLOSE、COMMIT 或 ROLLBACK (沒有 HOLD) SQL 陳述式時，即會明確地關閉 SQL 游標。

***ENDACTGRP**： 關閉 SQL 游標，且會隱含捨棄 SQL 準備的陳述式。

ENDMOD： 關閉 SQL 游標，且在模組跳出時會隱含捨棄 SQL 準備的陳述式。當呼叫堆疊中的第一個 SQL 程式結束時，即會釋放 LOCK TABLE 鎖定。

OUTPUT

指定是否產生前置編譯器報表。

***NONE**：不產生前置編譯器報表。

***PRINT**：產生前置編譯器報表。

DBGVIEW

指定 SQL 前置編譯器提供的來源除錯資訊類型。

***NONE**： 不產生來源概略表。

***STMT**： 容許使用程式陳述式號碼及符號 ID 來除錯已編譯的模組。

***LIST**：產生除錯已編譯模組物件的報表概略表。

***SOURCE**： 產生 SQL 程序、函數、觸發程式的來源概略表。

USRPRF

指定在執行已編譯的程式物件時使用的使用者設定檔，包括靜態 SQL 陳述式中每一個物件的程式物件權限。您可以使用程式擁有者或程式使用者的設定檔，以控制程式物件可以使用哪些物件。

***NAMING**：使用者設定檔是由命名慣例決定。如果命名慣例是 *SQL，則使用 USRPRF(*OWNER)。如果命名慣例是 *SYS，則使用 USRPRF(*USER)。

***USER**： 使用執行程式物件的使用者設定檔。

***OWNER**：執行程式時，使用程式擁有者及程式使用者的使用者設定檔。

DYNUSRPRF

指定用於動態 SQL 陳述式的使用者設定檔。

***USER**： 若爲本端，則動態 SQL 陳述式會在程式使用者的使用者設定檔下執行。若爲分散式，則動態 SQL 陳述式會在 SQL 資料包使用者的設定檔下執行。

***OWNER**： 若爲本端，動態 SQL 陳述式會在程式擁有者的設定檔下執行。若爲分散式，動態 SQL 陳述式會在 SQL 資料包擁有者的設定檔下執行。

DLYPRP

指定是否要延遲 PREPARE 陳述式的動態陳述式驗證，直到執行 OPEN、EXECUTE 或 DESCRIBE 陳述式。延遲驗證會消除冗餘的驗證，以增進效能。

***NO**： 不延遲動態陳述式驗證。準備動態陳述式時，即會驗證存取規劃。在 OPEN 或 EXECUTE 陳述式中使用動態陳述式時，會重新驗證存取規劃。因爲動態陳述式參照的物件權限或存在可能會變更，您仍須在發出 OPEN 或 EXECUTE 陳述式後檢查 SQLCODE 或 SQLSTATE，以確定動態陳述式仍然有效。

***YES**： 延遲動態陳述式驗證，直到在 OPEN、EXECUTE 或 DESCRIBE SQL 陳述式中使用動態陳述式。使用動態陳述式時，即會完成驗證且會建置存取規劃。如

果您在此參數中指定 *YES，則應在執行 OPEN、EXECUTE 或 DESCRIBE 陳述式後檢查 SQLCODE 及 SQLSTATE，以確定動態陳述式是有效的。

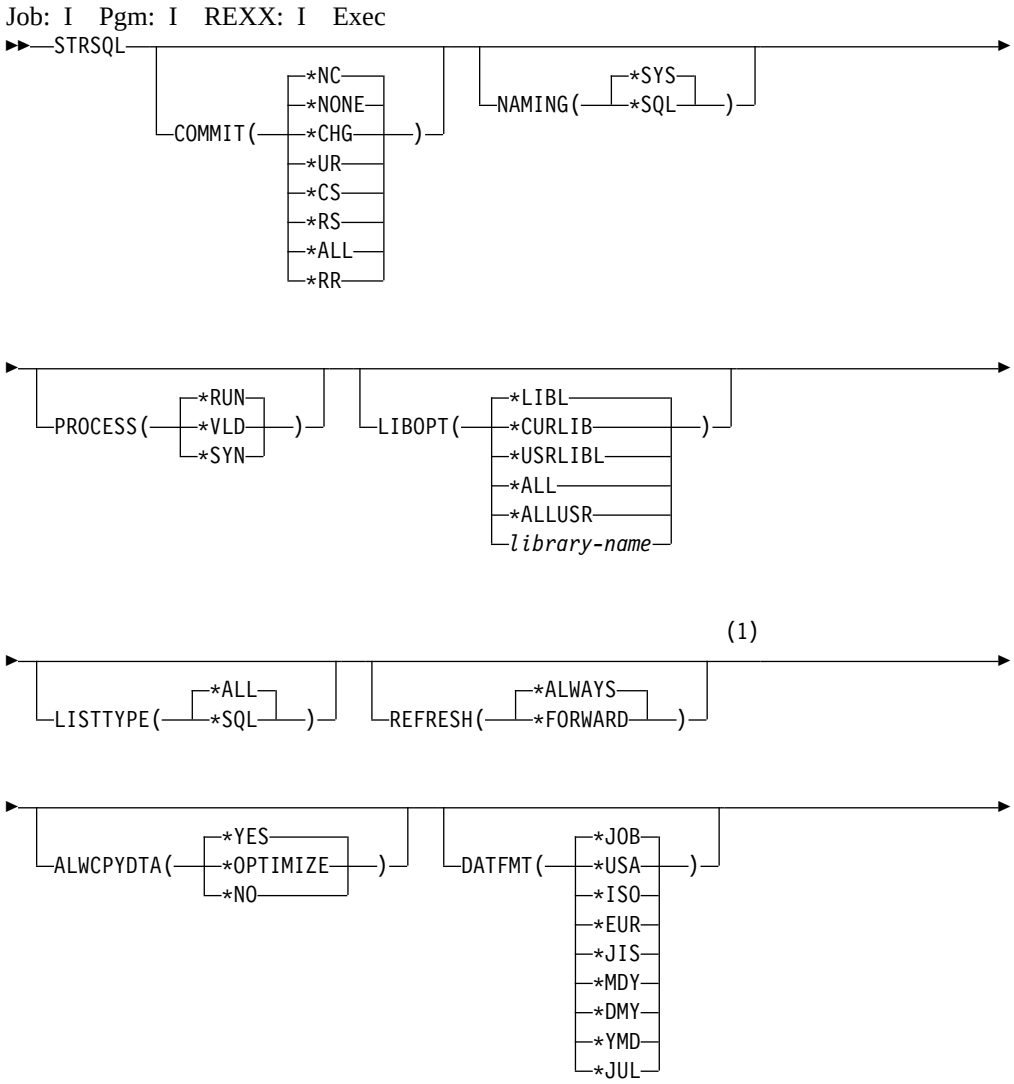
註: 如果您指定 *YES，則若在 PREPARE 陳述式中使用 INTO 子句，或若在發出陳述式的 OPEN 前，DESCRIBE 陳述式使用動態陳述式，效能並不會增進。

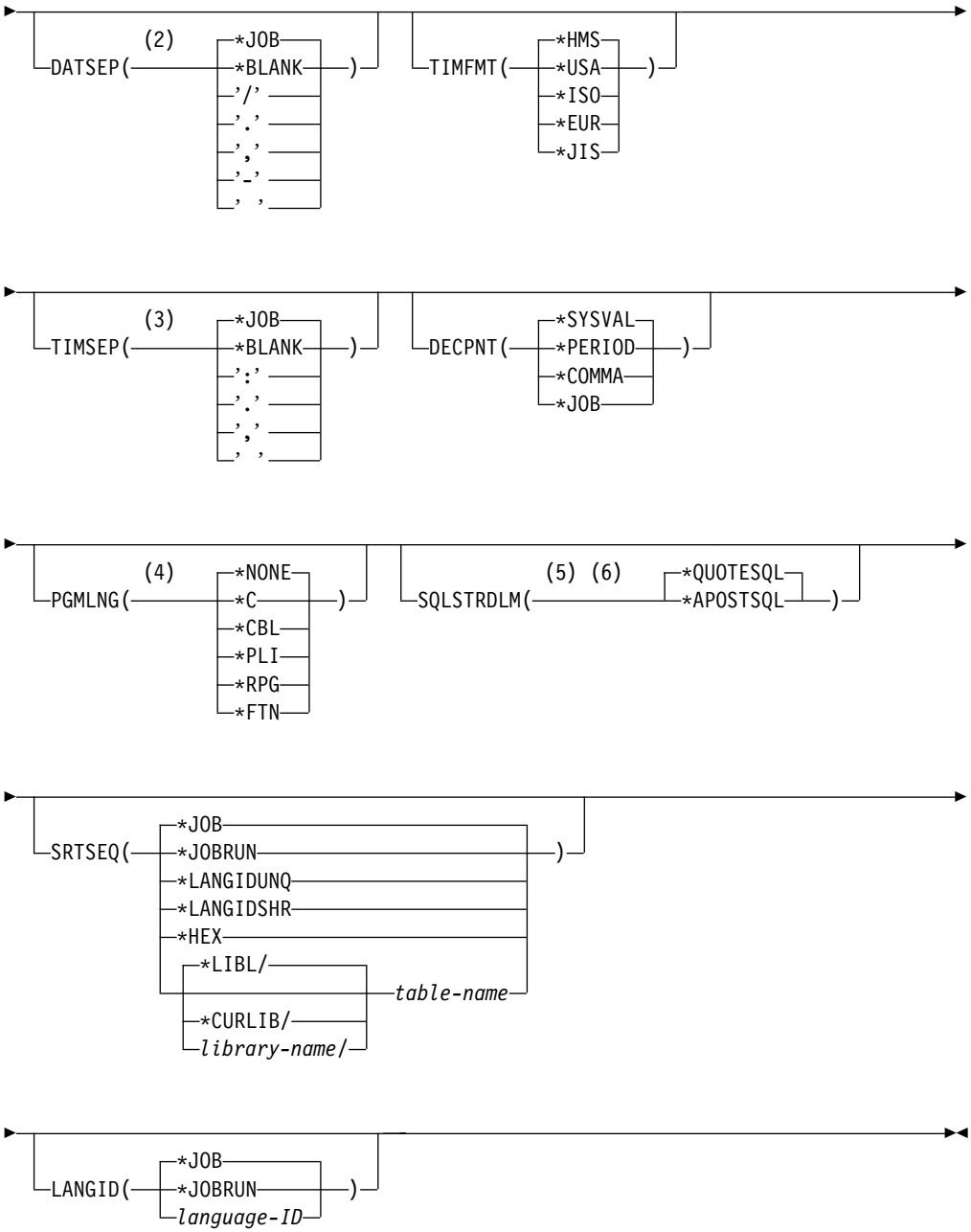
範例：

```
RUNSQLSTM  SRCFILE(MYLIB/MYFILE)  SRCMBR(MYMBR)
```

此指令會處理檔案庫 MYLIB、檔案 MYFILE、成員 MYMBR 中的 SQL 陳述式。

STRSQL (啓動結構化查詢語言) 指令





註:

1 在此點之前的所有參數，都可以依照位置格式加以指定。

2 只有在 DATFMT 參數中指定 *MDY、*DMY、*YMD 或 *JUL 時，DATSEP 才有效。

3 只有在指定 TIMFMT(*HMS) 時，TIMSEP 才有效。

4 只有在指定 PROCESS(*SYN) 時，PGMLNG 及 SQLSTRDLM 才有效。

5 只有在指定 PROCESS(*SYN) 時，PGMLNG 及 SQLSTRDLM 才有效。

6 只有在指定 PGMLNG(*CBL) 時，SQLSTRDLM 才有效。

目的：

STRSQL

「啓動結構化查詢語言 (STRSQL)」指令會啓動交談式「結構化查詢語言 (SQL)」程式。程式會啓動交談式 SQL 的陳述式登錄，它會立即顯示「輸入 SQL 陳述式」顯示畫面。此顯示畫面可讓使用者在交談式環境中建置、編輯、輸入及執行 SQL 陳述式。執行程式期間所收到的訊息會顯示在此顯示畫面中。

參數：

COMMIT

指定 SQL 陳述式是否執行於確定控制下。

***NONE 或 *NC：** 指定不使用確定控制。您可以看見其他工作中未確定的變更。如果程式中已併入 SQL DROP COLLECTION 陳述式，則必須使用 *NONE 或 *NC。如果已在 RDB 參數中指定關聯式資料庫，且關聯式資料庫不在 iSeries 系統上，則必須使用 *NONE 或 *NC。

***CHG 或 *UR：** 指定在 SQL ALTER、CALL、 COMMENT ON、CREATE、DROP、GRANT、LABEL ON、RENAME 及 REVOKE 陳述式中參照的物件，且會鎖定已更新、刪除及插入的列，直到工作單元 (異動) 結束。您可以看見其他工作中未確定的變更。

***CS:** 指定在 SQL ALTER、 CALL、COMMENT ON、CREATE、DROP、GRANT、LABEL ON、RENAME 及 REVOKE 陳述式中參照的物件，且會鎖定已更新、刪除及插入的列，直到工作單元 (異動) 結束。已選取，但未更新的列會被鎖定，直到選取下一列。您無法看見其他工作中未確定的變更。

***ALL 或 *RS：** 指定在 SQL ALTER、CALL、COMMENT ON、CREATE、DROP、GRANT、LABEL ON、RENAME 及 REVOKE 陳述式中參照的物件，且會鎖定已選取、更新、刪除及插入的列，直到工作單元 (異動) 結束。您無法看見其他工作中未確定的變更。

***RR：** 指定在 SQL ALTER、CALL、COMMENT ON、CREATE、DROP、GRANT、LABEL ON、RENAME 及 REVOKE 陳述式中參照的物件，且會鎖定已選取、更新、刪除及插入的列，直到工作單元 (異動) 結束。您無法看見其他工作中未確定的變更。 在 SELECT、UPDATE、DELETE 及 INSERT 陳述式中參照的所有表格都會被鎖定，直到工作單元 (異動) 結束。

註: CRTSQLXXX 指令 (當 XXX=CI、CPPI、 CBL、FTN、PLI、CBLI、RPG 或 RPGI) 中此參數的預設值是 *CHG。

NAMING

指定用於 SQL 陳述式中命名物件的命名慣例。

***SYS：** 使用系統命名慣例 (檔案庫名稱/檔名)。

***SQL：** 使用 SQL 命名慣例 (集合名稱.表格名稱)。

PROCESS

指定用來處理 SQL 陳述式的值。

***RUN：** 陳述式語法檢查、資料檢查，然後執行。

***VLD：** 陳述式語法檢查及資料檢查，但不執行。

***SYN：** 僅陳述式語法檢查。

LIBOPT

按下 F4、F16、F17 或 F18 功能鍵時，指定要使用哪些集合及檔案庫作爲建置集合清單的基礎。

您可以使用下列其中一個檔案庫值來限定集合清單名稱：

***LIBL**： 搜尋工作檔案庫清單中的所有檔案庫，直到找到第一個相符的檔案庫爲止。

***CURLIB**： 搜尋工作的現行檔案庫。如果沒有指定任何檔案庫作爲工作的現行檔案庫，則會使用 QGPL 檔案庫。

***USRLIBL**： 只搜尋工作檔案庫清單中使用者部份的檔案庫。

***ALL**： 搜尋系統中的所有檔案庫，包括 QSYS。

***ALLUSR**： 搜尋所有使用者檔案庫。搜尋名稱不是以字母 Q 起首的所有檔案庫，但下列除外：

```
#CGULIB     #DFULIB     #RPGLIB     #SEULIB
#COBLIB     #DSULIB     #SDALIB
```

雖然下列 Qxxx 檔案庫是由 IBM 提供，但通常含有經常變更的使用者資料。因此，這些檔案庫被視爲使用者檔案庫，且也會被搜尋：

```
QDSNX          QRCL          QUSRBRM          QUSRSYS
QGPL           QS36F         QUSRIJS          QUSRVxRxMx
QGPL38         QUSER38       QUSRINFSKR
QPFRDATA       QUSRADSM      QUSRRDARS
```

註: 使用者可以針對 IBM 支援的每一個版次，建立格式爲 QUSRVxRxMx 的不同檔案庫名稱。 VxRxMx 是檔案庫的版本、版次及修正層次。

檔案庫名稱：指定要搜尋的檔案庫名稱。

LISTTYPE

按下 F4、F16、F17 或 F18 功能鍵，指定清單支援顯示的物件類型。

***ALL**：顯示全部物件。

***SQL**：僅顯示 SQL 建立的物件。

REFRESH

指定何時重新整理顯示選取輸出資料。

***ALWAYS**：通常是在向前或向後捲動時重新整理資料。

***FORWARD**：只有在第一次向前捲動至資料結尾期間，重新整理資料。向後捲動時，則會顯示已經檢視過的資料副本。

ALWCPYDTA

指定是否可以在 SELECT 陳述式中使用資料副本。如果已指定 COMMIT(*ALL)，則 SQL 執行時間會忽略 ALWCPYDTA 值並使用現行資料。

***YES**：必要時，使用資料的副本。

***OPTIMIZE**：系統會決定是否使用從資料庫中擷取的資料，或使用資料的副本。決定是取決於哪一種方法以提供最佳效能。

***NO**：不容許資料副本。如果需要資料的暫時副本來執行查詢，則會傳回錯誤訊息。

STRSQL

DATFMT

指定 SQL 陳述式中所使用的日期格式。

***JOB**：使用工作屬性 DATFMT 中指定的格式。

***USA**：使用美國日期格式 (mm/dd/yyyy)。

***ISO**：使用「國際標準組織」日期格式 (yyyy-mm-dd)。

***EUR**：使用歐洲日期格式 (dd.mm.yyyy)。

***JIS**：使用「日本工業標準紀元」日期格式 (yyyy-mm-dd)。

***MDY**：使用月、日、年日期格式 (mm/dd/yy)。

***DMY**：使用日、月、年日期格式 (dd/mm/yy)。

***YMD**：使用年、月、日日期格式 (yy/mm/dd)。

***JUL**：使用羅馬曆日期格式 (yy/ddd)。

DATSEP

指定 SQL 陳述式中所使用的日期分隔字元。

***JOB**：使用工作屬性中指定的日期分隔字元。如果使用者在新交談式 SQL 階段作業中指定 *JOB，則會儲存及使用現行值。交談式 SQL 不會偵測到稍後對工作日期分隔字元所做的變更。

***BLANK**：使用空白 ( )。

'/'：使用斜線 (/)。

'.'：使用句點 (.) 。

','：使用逗點 (,)。

'-'：使用破折號 (-)。

' '：使用空格 ( )。

TIMFMT

指定 SQL 陳述式中使用的時間格式。

***HMS**：使用「時-分-秒」時間格式 (hh:mm:ss)。

***USA**：使用美國時間格式 (hh:mm xx，其中 xx 是 AM 或 PM)。

***ISO**：使用「國際標準組織」時間格式 (hh.mm.ss)。

***EUR**：使用歐洲時間格式 (hh.mm.ss)。

***JIS**：使用「日本工業標準紀元」時間格式 (hh:mm:ss)。

TIMSEP

指定 SQL 陳述式中所使用的時間分隔符號。

***JOB**：使用工作屬性中指定的時間分隔符號。如果使用者在新交談式 SQL 階段作業中指定 *JOB，則會儲存及使用現行值。交談式 SQL 不會偵測到稍後對工作時間分隔符號所做的變更。

***BLANK**：使用空白 ( )。

':': 使用冒號 (:)。

'.'：使用句點 (.) 。

','：使用逗點 (,)。

' '：使用空格 ( )。

DECPNT

指定使用的小數點種類。

***JOB**：當成 SQL 中數值常數的小數點使用的值，代表執行陳述式的工作所指定的小數點。

***SYSVAL**：小數點是擷取自系統值。如果使用者在新交談式 SQL 階段作業中指定 *SYSVAL，則會儲存及使用現行值。交談式 SQL 不會偵測到稍後對系統時間分隔符號所做的變更。

***PERIOD**：句點代表小數點。

***COMMA**：逗點代表小數點。

PGMLNG

指定要使用哪一個程式語言語法規則。若要使用此參數，必須在 PROCESS 參數中選取 *SYN。

***NONE**：不使用任何特定的語言語法檢查規則。

所支援的語言如下：

***C**：依據 C 語言語法規則來執行語法檢查。

***CBL**：依據 COBOL 語言語法規則來執行語法檢查。

***PLI**：依據 PL/I 語言語法規則來執行語法檢查。

***RPG**：依據 RPG 語言語法規則來執行語法檢查。

***FTN**：依據 FORTRAN 語言語法規則來執行語法檢查。

SQLSTRDLM

指定 SQL 字串區隔字元。使用此參數時需要使用 COBOL (*CBL) 字集。

***QUOTESQL**：引號代表 SQL 字串區隔字元。

***APOSTSQL**：撇號 (´)代表 SQL 字串區隔字元。

SRTSEQ

指定在「輸入 SQL 陳述式」顯示畫面的 SQL 陳述式中，字串比較所使用的排序順序表。

***JOB**：擷取工作的 SRTSEQ 值。

***JOBRUN**：每一次使用者啓動交談式 SQL 時，即會擷取工作的 SRTSEQ 值。

***LANGIDUNQ**：使用 LANGID 參數中指定語言的唯一權重排序表。

***LANGIDSHR**：使用 LANGID 參數中指定語言的共用權重排序表。

***HEX**：不使用排序順序表。使用字元的十六進位值以決定排序順序。

您可以使用下列其中一個檔案庫值來限定表格名稱：

***LIBL**：搜尋工作檔案庫清單中的所有檔案庫，直到找到第一個相符的檔案庫爲止。

***CURLIB**：搜尋工作的現行檔案庫。如果沒有指定任何檔案庫作爲工作的現行檔案庫，則會使用 QGPL 檔案庫。

library-name：指定要搜尋的檔案庫名稱。

table-name：指定要與交談式 SQL 階段作業一起使用的排序順序表名稱。

LANGID

指定在已指定 SRTSEQ(*LANGIDUNQ) 或 SRTSEQ(*LANGIDSHR) 時所使用的語言 ID。

***JOB**：擷取工作的 LANGID 值。

***JOBRUN**：每一次啓動交談式 SQL 時，即會擷取工作的 LANGID 值。

language-ID：指定要使用的語言 ID。

範例：

```
STRSQL   PROCESS(*SYN)  NAMING(*SQL)
  DECPNT(*COMMA)  PGMLNG(*CBL)
  SQLSTRDLM(*APOSTSQL)
```

此指令會啓動交談式 SQL 階段作業，只檢查 SQL 陳述式的語法。語法檢查程式所使用的字集會使用 COBOL 語言語法規則。此階段作業使用 SQL 命名慣例。 小數點是以逗點代表，且 SQL 字串區隔字元是以單引號代表。

參考書目

本手冊列示一些出版品，內容是關於本手冊中所參照或描述的主題的其餘資訊。本節所列的各手冊含有它們的完整標題及訂單號碼，但以文字參照時，則使用標題的縮寫版本。

- 備份及回復

  本手冊含有「備份及回復」一書中的資訊子集。內容是關於規劃備份及回復策略、儲存與復置程序可用的不同媒體類型，以及磁碟回復程序的資訊。它也說明如何使用備份來重新安裝系統。

- File Management

  此手冊提供使用應用程式中檔案的相關資訊。

- DB2 UDB for iSeries Database Programming

  此手冊提供 DB2 UDB for iSeries 資料庫組織的詳細說明，包括如何建立、說明及更新系統上資料庫檔案的相關資訊。

- CL Programming

  此手冊提供 DB2 UDB for iSeries 程式設計主題的廣泛討論，包括物件和檔案庫、CL 程式設計、控制流程和程式間通信、在 CL 程式中使用物件，以及建立 CL 程式等的一般討論。其他主題包括預先定義和即發訊息、處理、定義和建立使用者定義的指令及功能表、應用程式測試，包括除錯模式、岔斷點、追蹤及顯示功能。

- Control Language (CL)

  此手冊提供 DB2 UDB for iSeries 控制語言 (CL) 及其 OS/400 指令的說明。(非 OS/400 指令說明於相對的授權程式出版品中)。它也提供伺服器使用的所有 CL 指令概觀，以及說明撰寫它們所需的語法規則。

- iSeries Security Reference

  此手冊提供系統安全概念、安全性規劃及在系統上設定安全性的相關資訊。它也提供有關保護系統和資料免於受到無適當授權者使用、保護資料免於受到有意或無意的損壞、將安全性保持最新狀態，以及在系統上設定安全性的資訊。

- DB2 UDB for iSeries SQL Reference

  此手冊提供 DB2 UDB for iSeries 陳述式及其參數的相關資訊。它也提供說明 SQL 通信區 (SQLCA) 及 SQL 說明區 (SQLDA) 的附錄。

- IDDU 使用

  此手冊說明如何使用 DB2 UDB for iSeries 交談式資料定義公用程式 (IDDU) 來對系統描述資料字典、檔案及記錄。

- WebSphere Development Studio: ILE COBOL Programmer's Guide

  此手冊提供在 iSeries 系統上設計、撰寫、測試及維護 COBOL for iSeries 程式所需的資訊。

- WebSphere Development Studio: ILE RPG Programmer's Guide

  此手冊提供在 iSeries 系統上設計、撰寫、測試及維護 ILE RPG for iSeries 程式所需的資訊。

- ILE C for AS/400 Language Reference

  此手冊提供在 iSeries 系統上設計、撰寫、測試及維護 ILE C for iSeries 程式所需的資訊。

- WebSphere Development Studio: ILE C/C++ Programmer's Guide

  此手冊提供在 iSeries 系統上設計、撰寫、測試及維護 ILE C for iSeries 程式所需的資訊。

- WebSphere Development Studio: ILE COBOL Reference

  此手冊提供在 iSeriesCOBOL 系統上設計、撰寫、測試及維護 COBOL for iSeries 程式所需的資訊。

- REXX/400 Programmer's Guide

  此手冊提供在 iSeries 系統上設計、撰寫、測試及維護 REXX/400 程式所需的資訊。

- DB2 多系統

  此手冊說明分散式關聯資料庫檔案、節點群組及分割區的基本概念。本書提供建立及使用跨多重 iSeries 系統分割的資料庫檔案所需之資訊。所提供的資訊是關於如何配置系統、如何建立檔案，以及如何在應用程式中使用檔案。

- Performance Tools for iSeries 

  此手冊提供程式設計師收集系統、工作或程式效能相關資料所需的資訊。此書也提供列印及分析效能資料來識別及更正可能存在的低效能狀況之秘訣。包含了管理員及代理程式特性的相關資訊。

- DB2 UDB for iSeries SQL Call Level Interface (ODBC)

  此手冊提供應用程式設計師使用 DB2 呼叫層次介面撰寫應用程式所需的資訊。

- IBM Developer Kit for Java

  此手冊提供在 iSeries 系統上設計、撰寫、測試及維護 Java 程式所需的資訊。其亦包含「iSeries Developer Kit for Java JDBC driver」一章，提供使用 JDBC 或 SQLj 從 Java 程式存取資料庫檔案的相關資訊。

索引

索引順序以中文字，英文字，及特殊符號之次序排列。

〔一劃〕

〔二劃〕

〔三劃〕

〔四劃〕

〔五劃〕

〔六劃〕

〔七劃〕

〔八劃〕

〔九劃〕

〔十劃〕

〔十一劃〕

〔十二劃〕

〔十三劃〕

〔十四劃〕

〔十五劃〕

〔十六劃〕

〔十七劃〕

〔十八劃〕

〔十九劃〕

〔二十劃〕

〔二十一劃〕

〔二十三劃〕

A

B

C

D

E

F

G

H

I

J

L

U

W

X