目录

第一章　AS/400 DB2 结构化查询语言介绍

本书介绍 AS/400 系统中使用的 DB2 结构化查询语言(SQL)，DB2 查询管理和 SQL 开发工具第四版特许程序的有关内容，SQL 管理关系型数据的信息，SQL 语句能嵌套在高级语言中，动态地准备和运行或交互地运行。

SQL 由描述用数据库中的数据做什么的语句和子句组成，以及所需条件的说明。

SQL 通过 DDM 访问远程数据库。这个功能将在本书第二十四章给出说明，详细内容请看分布式数据库程序设计一书。

1.1　SQL 概念

AS/400 SQL 由下列主要部分组成：

SQL 运行时支持

SQL 运行时从语法上解释 SQL 语句及运行任何一条 SQL 语句。这种支持是操作系统/400 特许程序的一部分，即在系统没有安装 DB2 查询管理和 SQL 开发工具特许程序的情况下，含 SQL 语句的应用程序也能在系统上运行。

SQL 预编译器

SQL 预编译支持在主语言中预编译嵌套的 SQL 语句。以下是所支持的语言：

—ILE C/400*

—COBOL/400*

—ILE COBOL/400*

—AS/400 PL/I*

—RPG Ⅲ (RPG/400*的一部分)

—ILE RPG/400*

SQL 主语言预编译准备的包括 SQL 语句的应用程序.然后主语言编译器编译已预编译过的主源程序。详细内容，请看第十六章。预编译器的支持是 DB2 查询管理和 SQL 开发工具特许程序的一部分。

SQL 交互接口

SQL 交互接口允许建立并执行 SQL 语句。关于交互 SQL 的详细信息可在第十七章中查到。交互 SQL 是 DB2 查询管理器和 SQL 开发工具特许程序的一部分。

运行 SQL 语句的 CL 命令

RUNSQLSTM 允许运行一系列 SQL 语句，它们存在源码文件中。RUNSQLSTM 是 DB2 查询管理器和 SQL 开发工具特许程序的一部分。第十八章中有 SQL 语句运行的详细说明。

AS/400 DB2 查询管理

AS/400 DB2 查询管理提供提示驱动的交互接口。这个接口允许生成数据、添加数据、维护数据，及在数据库上运行报表。查询管理是 DB2 查询管理器和 SQL 开发工具特许程序的一部分。详细内容请看《AS/400 DB2 查询管理》。

SQL REXX 接口

SQL REXX 接口允许在 REXX 过程中运行 SQL 语句。这个接口是 DB2 查询管理器和 SQL 开发工具特许程序的一部分。要了解在 REXX 过程中使用 SQL 语句的详细信息，请

看第十五章。

SQL 调用级接口

AS/400 DB2 支持 SQL 调用级接口。它允许使用任何 ILE 语言的用户通过过程调用由系统提供的服务程序直接访问 SQL 功能。使用 SQL 调用级接口，不必预编译就可以实现所有的 SQL 功能。这是一个标准的过程调用；准备 SQL 语句，执行 SQL 语句，获取数据甚至可以实现访问目录、为输出列连接程序变量等高级功能。

在 AS/400 DB2 SQL 调用接口(ODBL)一书中，对所有可用功能函数及其语法有详细的描述。

QSQPRCED API

这个应用程序接口(API)提供了扩展的动态 SQL 能力。由 SQL 语句形成 SQL 软件包，然后由 API 执行。软件包中的语句通过 API 存留，直到明显的删除软件包或语句。QSQPRCED 是 OS/400 特许程序的一部分，在《系统 API 参考》中有更多的 QSQPRCED API 的信息。

QSQCHKS API

这个 API 语法检查 SQL 语句。QSQCHKS 是 OS/400 特许程序的一部分。在《系统 API 参考》中有更多的 QSQCHKS API 的信息。

DB2 多系统

这个操作系统的特性是允许数据跨越 AS/400 多个系统。在《AS/400 的 DB2 多系统》中有更多的说明。

DB2 对称多处理

这个系统的特性是为取出那些包含并行处理的数据给出附加方法提供查询优化。对称多处理(SMP)是在一个单系统中得到并行处理模式，这个系统有多处理器(CPU 和 I/O 处理器)，这些处理器为了一个共同的最终目标而共享内存和磁盘资源。并行处理意味着数据库管理器能使多个(或全部)系统处理器为一个查询同时工作。在 22.27 中阐述了怎样进行并行处理。

1.1.1 关系数据库和术语

在关系数据模型中，所有的数据放在表中。DB2 目标做为 AS/400 系统的目标来生成和维护。下表给出 AS/400 系统术语和 SQL 关系数据库术语间的相互关系。关于数据库的详细信息在《AS/400 DB2 数据库程序设计》一书中可了解到。

表 1-1　系统术语和 SQL 术语的关系

系统术语	SQL 术语
库：一组相关目标并允许通过名字查找目标。	集合：由库、日志、日志接收器、SQL 集合和可选的数据字典组成。集合把相关目标组合在一起，并允许由名字查找目标。
物理文件：一系列记录。	表：一系列行和列。
记录：　　一系列字段。	行：由一些列组成表中的水平部分。
字段：同一数据类型的有关信息的一个或多个字符	列：一个数据类型表当中的垂直部分。
逻辑文件：一个或多个物理文件中字段和记录的子集	视图：一个或多个表的列和行的子集。
SQL 包：用来运行 SQL 语句的目标类型	包：用来运行 SQL 语句的目标类型。
用户配置文件	授权名或授权 ID

1.1.1.1 SQL 术语

在 AS/400 DB2 程序设计中有两个命名约定：系统(*SYS)和 SQL(SQL)。命名约定影响限定文件名和表名及在交互 SQL 显示项的方法。对 REXX 是通过 SET OPTION 语句来选择的，所用的命名约定是由 SQL 命令参数选择。

系统命名(*SYS)：在系统命名约定中，文件由下列格式用库名限定：

库/文件

如果没有明显规定限定表名且在 CRTSQLxxx (1)或 CRTSQLPKG 命令的 DFTRDBCOL 参数规定了缺省的集合名，则使用缺省的集合名。若表名未明显限定且没规定缺省集合名，限定的规则为：

下面的 CREATE 语句处理没有限定的目标：

—CREATE TABLE—在当前库中生成表(*CURLIB)

—CREATE VIEW—在子集中第一个引用的库中生成视图

—CREATE INDEX—在建索引的表所在库或集合里生成索引

—CREATE PROCEDURE—在当前库中生成过程

所有其它的 SQL 语句会使 SQL 在库列表中检索未限定表。

缺省关系数据库集合(DFTRDBCOL)参数仅对静态 SQL 语句适用。

SQL 命名(*SQL)：在 SQL 命名约定中，表由下面格式的集合名限定：

集合.表

若表名没有明显限定且在CRTSQLxxx命令中规定了DFTRDBCOL参数,则使用缺省集合名。如果没有明显限定表名且没规定缺省集合名，那么规则为：

—对静态 SQL，缺省的限定名是程序主人的用户配置文件。

—对动态 SQL 或交互 SQL，缺省的限定名是运行语句作业的用户配置文件。

(1)这里的 xxx 表示嵌入 SQL 的主语言，它们可以是 CI、CBL、CBLI、PLI、RPG、RPGI。

1.1.2 SQL 语句类型

SQL 有四种基本语句类型：

数据定义语句（DDL)、数据操作语句（DML)、动态 SQL 语句和混杂语句。

SQL 能处理用 SQL 生成的目标，也能处理 AS/400 外部说明的物理文件和单格式逻辑文件，且不管它们是否在 SQL 集合中，但不能引用程序说明文件的 IDDU 字典定义，程序说明文件在表中只做一列出现。

SQL 数据定义语句	SQL数据操作语句
ALTER TABLE	CLOSE
COMMENT ON	COMMIT
CREATE COLLECTION	DECLARE CURSOR
CREATE INDEX	DELETE
CREATE PROCEDURE	FETCH
CREATE SCHEMA	INSERT
CREATE TABLE	LOCK TABLE
CREATE VIEW	OPEN
DROP COLLECTION	ROLLBACK
DROP INDEX	SELECT INTO
DROP PACKAGE	UPDATE
DROP PROCEDURE	

DROP SCHEMA DROP TABLE DROP VIEW GRANT PACKAGE GRANT PROCEDURE GRANT TABLE LABEL ON RENAME REVOKE PACKAGE REVOKE PROCEDURE REVOKE TABLE	

动态 SQL 语句	其他 SQL 语句
DESCRIBE EXECUTE EXECUTE IMMEDIATE PREPARE	BEGIN DECLARE SECTION CALL CONNECT DECLARE PROCEDURE DECLARE STATEMENT DECLARE VARIABLE DESCRIBE TABLE DISCONNECT END DECLARE SECTION INCLUDE RELEASE SET CONNECTION SET OPTION SET RESULT SETS SET TRANSACTION WHENEVER

1.2 SQL 目标

在 AS/400 系统中 SQL 的目标有：集合、表、视图、SQL 包、索引和目录。SQL 把这些目标做为 AS/400 数据库目标来生成和管理。下面是对这些目标简短描述。

1.2.1 集合

集合提供对 SQL 目标的逻辑分类。集合由库、日志、日志接收器、目录和可选的数据字典组成。表、视图、系统目标(例如程序)能在任何 AS/400 库中建立、移动、重存。

如果 SQL 集合中不包括数据字典，所存 AS/400 文件也能在 SQL 集合中生成或移动。如果 SQL 集合包括数据字典，则 AS/400 源物理文件或一个成员的非源物理文件能在 SQL 集合中生成、移动或重存，AS/400 逻辑文件不能放在 SQL 集合中，因为它们不能用数据字典描述。

可生成并拥有多个集合。

1.2.1.1 数据字典

若一个集合是在版本 3.1 前建立，或在 CREATE COLLECTION，CREATE SCHEMA 语句规定

了数据字典子句，那么集合中包含数据字典。数据字典就是包含目标定义的一组表。如 SQL 建立了字典，那么系统就会自动管理它。可用交互数据定义实用工具(IDDU)来处理数据字典。IDDU 是 OS/400 程序的一部分，在《IDDU 使用》一书中有对 IDDU 的详细介绍。

1.2.1.2 日志和日志接收器

日志和日志接收器用来记录对表和视图的修改。可在 SQL COMMIT 和 ROLLBACK 语句处理时使用。它们还可用在审计跟踪和或向前向后恢复。有关日志的详细内容，请看备份和恢复一书。

1.2.1.3 目录

SQL 目录由一系列表和视图组成，它描述了表、视图、索引、包、过程、文件及约束。这个信息放在库 QSYS 和 QSYS2 中的一组交叉引用表中。QSYS 目录表描绘了关于所有表、视图、索引、包、过程、文件及系统约束的信息，库 QSYS2 也包括建立在 QSYS 目录表上的一组目录视图，每个 SQL 集合里都有一组建立在目录表上的视图，它们包括集合中的表、视图、索引、包、文件和约束的信息。

当生成一个集合时，会自动生成目录。你不能释放或明显地更改目录。

更详细的 SQL 目录信息，请看《AS/400 DB2 SQL 参考》。

1.2.2 表、行和列

表是由行和列构成的数据的二维排列。行是一列或多列组成的水平部分。列是一个数据类型的一行或多行数据的垂直部分。一列的所有数据必须为同一个类型。SQL 表是一个键字或非键字的物理文件。关于数据类型的详细说明，请看《AS/400 SQL 参考》。

表中数据能分布在多个 AS/400 中。在《AS/400 DB2 多系统》中详细介绍了分布式表。

下面是 SQL 表的例子：

PIC2

1.2.3 视图

对于应用程序来说，视图看起来就象一个表，但视图不包含数据。用一个或多个表生成的视图可以包括给定表的全部列或它们中的一些子集，也能包括给定表的所有行或某些子集，视图中列的排序可以不同于它们所在表中列的顺序。SQL 视图是非键字逻辑文件的特殊形式。

下图给出由前面 SQL 表例子生成的视图。视图用表的 PROJNO 和 PROJNAME 列及 MA2110 和 MA2100 行建立的。

PIC3。

1.2.4 索引

SQL 索引是按逻辑升序或降序安排的表中列数据的子集。每个索引都有独立的安排。这些安排用来排序(由 ORDER 子句)、分组(由 GROUP 子句)、及连接。SQL 索引是键字逻辑文件。

索引可以使系统更快地进行数据检索。索引的建立是可选的，你可以建立任意数目的索引，也可在任何时候生成或删除索引。索引由系统自动维护，然而，因为索引是由系统维护的，大量的索引相反会影响修改表应用的性能。

1.2.5 限制

限制是由数据库管理实施的规则。AS/400 DB2 支持以下限制：

唯一性限制

唯一性限制就是做键字的值要唯一。用 CREATE TABLE 和 ALTER TABLE 语句可以生成唯一性限制。(2)

唯一性限制在执行 INSERT 和 UPDATE 语句时实施。PRIMARY KEY 限制是唯一限制的形式，它们间的区别是 PRIMARY KEY 不能有任何空列。

引用限制

引用限制就是在下列情况下外键值才有效的规则：

—它们做为父键的值出现，或

—外键的一些元素是空的。

引用限制是执行 INSERT、UPDATE、DELETE 语句时实施的。

检查限制

检查限制是对一列或一组列中的值限定的规则。可用 CREATE TABLE 和 ALTER TABLE 语句添加检查限制。检查限制在执行 INSERT 和 UPDATE 语句时实施。为满足限制，对每一行数据的插入或更新必须规定或真或未知（由于一个空值)的条件。在第六章《数据完整性》详细论述了限制。

(2)尽管 CREATE INDEX 能建立唯一索引且也能保证唯一性，这样的索引不是一个限制。

1.2.6 触发器

触发器就是一组动作，它给出对一个指定的基本表发生特定事件时自动执行的操作。这些事件可以是插入、更新、删除操作。触发器可在事件前或事件后运行。在本书第六章《数据完整性》或在《AS/400 数据库程序设计》中可以得到更多的关于触发器的信息。

1.2.7 存储过程

存储过程是用 SQL CALL 语句时被调用的程序。DB2 支持外部的存储过程和 SQL 过程，外部存储过程可以是任何 AS/400 程序和 REXX 过程，它们不能是系统/36 程序或过程。SQL 过程可完全用 SQL 语句定义，也可由包括 SQL 控制语句的 SQL 语句组成。若想了解详细的存储过程，请看第七章。

1.2.8 包

应用程序的 SQL 语句和远程关系数据库管理系统(DBMS)联编时，包含在控制结构过程中的目标就是 SQL 包。DBMS 使用控制结构来处理程序运行时遇到的 SQL 语句。

在 CRESQLxxx 命令或生成程序目标时，如果规定了关系数据库名(RDB 参数)，就建立 SQL 包。也可用 CRTSQLPKG 建立包。关于包和关系数据库的功能分布的详细信息，请看第二十四章。

也可使用 QSQPRCED API 建立包。在《系统 API 参考》中有关于 QSQPRCED 的介绍。

1.3 应用程序目标

生成 AS/400 DB2 应用程序会导致生成几个目标。这部分简短介绍建立 AS/400 DB2 应用程序的过程，AS/400 DB2 支持非 ILE 和 ILE 预编译，应用程序可以是分布式的也可以是非分布式的。关于生成 AS/400 DB2 应用程序的其它信息，请看第十六章。

使用 AS/400 DB2 需要管理以下目标：

初始源文件

可选的，ILE 程序的模块目标

程序和服务程序

应用于分布式程序的 SQL 包

对于非分布式非 ILE AS/400 DB2 程序，仅须管理初始源文件和结果程序。下面给出用于非分布式，非 ILE AS/400 DB2 程序的预编译和编译的目标和步骤。

PIC4

对于非分布式 ILE AS/400 DB2 程序，可以管理初始源文件、模块、结果程序或服务程序。下图给出在预编译命令中规定 OBJTYPE(*PGM)时，非分布式 ILE AS/400 DB2 预编译和编译的目标和步骤。

PIC5

对于分布式非 ILE AS/400 DB2 程序，必须管理初始源文件、结果程序、包。下图给出对分布式非 ILE AS/400 DB2 程序编译和预编译的步骤和目标。

PIC6

对于分布式 ILE AS/400 DB2 程序，必须管理初始源文件、模块目标、结果程序或服务程序和结果包。SQL 包能对分布式 ILE 程序或服务程序的每一个分布式模块生成。下图给出对分布式 ILE AS/400 DB2 程序的预编译和编译过程的目标和步骤。

PIC7

注：与 AS/400 DB2 分布式程序目标相关的访问计划直到程序在本地运行才会建立。

1.3.1 用户源文件成员

源文件成员包括程序员的应用语言和 SQL 语句。可用 SEU 建立和维护成员，它是 AS/400 开发工具特许程序的一部分。

1.3.2 临时源文件成员

名为 QSQLTEMP(CRTSQLRPGI 生成的名为 QSQLTEMP1)的临时源文件放在 QTEMP 库中，它是由 CRTSQLxxx 预编译生成的，在作业完成时由系统自动删除，与程序同名的一个成员也加到临时源文件中。

这个成员包括：

调用 SQL 运行时支持，其中有替代嵌入的 SQL 语句。

SQL 语句分析和语法检查。

由预编译调用主语言编译是缺省情况。第十六章详细介绍了预编译的有关问题。

1.3.3 程序

程序是可运行的目标，它是非 ILE 编译过程的结果或是 ILE 编译过程联编的结果。

访问计划是一系列内部结构和信息，能告诉你怎样最有效地运行嵌套的 SQL 语句。仅在成功生成程序时才创建它。访问计划不能在程序用下列 SQL 语句生成时建立：

访问的表或视图找不到

要去访问还未授权的表和视图

当运行程序时，这些语句的访问计划才可以建立。这时如果上面两种情况仍存在，将返回负的 SQLCODE。对非分布 SQL 程序访问计划存储在程序目标中且由它管理，而分布式 SQL 程序，它放在 SQL 包中存储和维护。

1.3.4 包

SQL 包包括分布 SQL 程序的访问计划。SQL 包是在下面情况下建立的目标：

在 CRTSQLxxx 命令中使用 RDB 参数，成功生成分布的 SQL 程序。

运行 CRTSQLPKG 命令。

当生成一个分布 SQL 程序时,SQL 包的名字和一个内部一致的令牌就保存在这个程序里。它用来在运行时寻找 SQL 包和验证 SQL 包对这个程序是否正确。因为 SQL 包名字对于分布 SQL 程序是很关键的，所以对 SQL 包不能：

移动

改名

复制

存储到不同的库

1.3.5 模块

模块是一个 ILE 目标,它是用 CRTxxxMOD 命令(或任何的 CRTBNDxxx 命令)编译源代码生成的，模块用 CRTPGM 联编成程序时才能运行。通常是几个模块联编到一起，但有时还可以和自身联编。模块包括 SQL 语句信息，但只有在模块同程序或服务程序联编时，才建立 SQL 访问计划。

1.3.6 服务程序

服务程序是 ILE 目标，它提供把外部支持的可调用的例程(功能或过程)封装到一个目标中的方法。联编程序和其他服务程序能访问由服务程序提供的解决它们输入到输出的例程。当生成调用程序时建立与服务的连接。由于不必在调用程序中编码，这提高了调用这些例程的性能。

第二章　启动 SQL

这章介绍如何生成和处理 SQL 集合、表和视图。

这一章中每一个 SQL 语句语法的详细说明请看《AS/400 DB2 参考》。在第三章及第五章介绍在复杂情况下怎样使用 SQL 语句和子句。

在这一章，交互 SQL 接口的例子给出怎样执行 SQL 语句。每个 SQL 接口提供使用 SQL 语句定义表、视图和其他目标的方法、更新目标的方法及从这些目标中读数据的方法。

2.1　启动交互 SQL

要启动交互 SQL，输入下列命令：

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

按 Enter 键。当出现“Enter SQL statements”显现时，就可以输入 SQL 语句了。交互式 SQL 和 STRSQL 命令的详细介绍请看第十七章。

2.2　建立 SQL 集合

SQL 集合是放置表、索引、视图、包的基本目标。

输入下面的 SQL 语句，并按 Enter 键，一个简单的名为 SAMPLECOLL 的集合就建立了。

```
                              Enter SQL Statements
Type SQL statement, press Enter.
     Current connection is to relational database SYSTEM1
===> CREATE COLLECTION SAMPLECOLL
                                                                        Bottom
F3=Exit   F4=Prompt   F6=Insert line   F9=Retrieve   F10=Copy line
F12=Cancel            F13=Services     F24=More keys
```

注：运行这条语句能生成几个目标，所以用的时间稍长。

在成功地生成集合后，可以在其中生成表、视图和索引，表、视图和索引也可在库中生成不放在集合中。

2.3 建立和使用表

用 CREATE TABLE 语句建立一个表，定义表中列的物理属性，定义一些条件来限制表中允许的值。

2.3.1 建立 INVENORY_LIST 表

我们要建一个表来维护一个商户的当前库存。它们包括库存内的物品号、物品名、价格、数量、最后定货日期和数量，物品号是必须的值，它不能为空，物品名、数量和定货数可有用户提供的缺省值，最后定货日期和数量允许为空值。

空值指出一行的一个空列值，它不同于零值和空格值。它意味着‘未知’，它不等于任何值，空值之间也不相等。如果一个列不允许空值，那么这列必须有值，可以是缺省值或用户提供的值。

缺省值是在往表中加一行时对某列没有规定值分配给此列的值，如果对某列没规定缺省值，就用系统缺省值，详细内容请看 3.1.1。

在‘enter SQL statement’显示中，写 CREATE BABLE 并按 F4 键出现下面的显示：

```
                        Specify CREATE TABLE Statement
Type information, press Enter.
Table . . . . . .   INVENTORY_LIST______     Name
  Collection . . . . . .     SAMPLECOLL__          Name, F4 for list
Nulls:  1=NULL, 2=NOT NULL, 3=NOT NULL WITH DEFAULT
Column               FOR Column    Type              Length  Scale  Nulls
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
                                                                  Bottom
  Table CONSTRAINT . . . . . .   N      Y=Yes, N=No
  Distributed Table  . . . . . .   N      Y=Yes, N=No
F3=Exit   F4=Prompt          F5=Refresh   F6=Insert line    F10=Copy line
F11=Display more attributes   F12=Cancel   F14=Delete line   F24=More keys
```

在表和集合的提示中，输入表名和集合名（如图）。要在表中定义的每一列，在显示的下部重复给出。对每一列，输入列名、类型、范围和长度以及空属性。

按 F11 键可以看到能为这些列规定的一些属性，这里可以规定缺省值。

```
  *                                                                    *
* *                        Specify CREATE TABLE Statement              *
  *                                                                    *
* * Type information, press Enter.                                    *
  *                                                                    *
* * Table  . . . . . .   INVENTORY_LIST______     Name                *
* *   Collection . . . . .     SAMPLECOLL__         Name, F4 for list   *
  *                                                                    *
* * Data:  1=BIT, 2=SBCS, 3=MIXED, 4=CCSID                            *
  *                                                                    *
* * Column               Data  Allocate  CCSID  CONSTRAINT  Default    *
* * ITEM NUMBER_______     _     _____    _____      N      ___________ *
* * ITEM NAME_________     _     _____    _____      N      '***UNKNOWN***'___ *
* * UNIT_COST_________     _     _____    _____      N      ___________ *
* * QUANTITY_ON_HAND__     _     _____    _____      N      NULL_______ *
* * LAST_ORDER_DATE___     _     _____    _____      N      ___________ *
* * ORDER_QUANTITY____     _     _____    _____      N      20_________ *
* * ___________________     _     _____    _____      _      ___________ *
* *                                                          Bottom    *
  *                                                                    *
* *   Table CONSTRAINT . . . . . .   N      Y=Yes, N=No               *
* *   Distributed Table  . . . . . .   N      Y=Yes, N=No               *
  *                                                                    *
* * F3=Exit   F4=Prompt          F5=Refresh   F6=Insert line   F10=Copy line  *
* * F11=Display more attributes   F12=Cancel   F14=Delete line   F24=More keys  *
  *                                                                    *
```

注：另一种输入列定义的方法是，把光标放在表中其中一列，按 F4 键(提示)，这会显示定义列的所有属性。

输入所有值后，按 Enter 键生成表。会再次出现“Enter SQL statement”显示，指出表已建好。

也可直接在“Enter SQL statements”显示中输入 CREATE TABLE 语句，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)
```

2.3.2 建立 SUPPLIERS 表

在以后的例子中，我们还需第二个表。这个表包含库存项的供货商，他们提供货物，及费用。要生成此表，可在“Enter SQL statements”上直接输入值或使用交互 SQL 显示按 F4 键：

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

2.4 使用 LABLE ON 语句

在交互 SQL 中，当显示 SELECT 语句输出值时，通常用列名做列标题，用 LABLE ON 语句，可为列名建立描述性的标号。既然我们用交互 SQL 运行例子，就使用 LABLE ON 语句改变列标题。尽管列名也是描述性，但如果列标题能在一行上显示名字各个部分，那么它更容易读懂，也可在一个显示中看到更多列的数据。

要修改列标号，在“Enter SQL statements”显示中输入“LABLE ON COLUMN”然后按 F4 键，显示如下：

```
  *                                                                    *
* *                        Specify LABEL ON Statement                  *
  *                                                                    *
* * Type choices, press Enter.                                         *
  *                                                                    *
* *  Label on . . . .   2                      1=Table or view         *
* *                                             2=Column               *
* *                                             3=Package              *
  *                                                                    *
* *  Table or view      INVENTORY_LIST_____     Name, F4 for list       *
* *    Collection . .     SAMPLECOLL__          Name, F4 for list       *
  *                                                                    *
* *  Option . . . . .   1                      1=Column heading        *
* *                                             2=Text                 *
  *                                                                    *
  *                                                                    *
  *                                                                    *
  *                                                                    *
  *                                                                    *
```

```
  *                                                                    *
  *                                                                    *
  *                                                                    *
* * F3=Exit    F4=Prompt    F5=Refresh    F12=Cancel    F20=Display full names  *
* * F21=Display statement                                              *
  *                                                                    *
```

输入表的名字和要加列标号的列集合的名字，再按 Enter 键，出现下面的显示，它提示表中每一列：

```
  *                                                                    *
  *                          Specify LABEL ON Statement                *
  *                                                                    *
  * Type information, press Enter.                                     *
  *                                                                    *
  *                       Column Heading                               *
  * Column                ....+....1....+....2....+....3....+....4....+....5....  *
  * ITEM_NUMBER           'ITEM               NUMBER'_____________________  *
  * ITEM_NAME             'ITEM               NAME'_____________________  *
  * UNIT_COST             'UNIT               COST'_____________________  *
  * QUANTITY_ON_HAND      'QUANTITY           ON                  HAND'_____  *
  * LAST_ORDER_DATE       'LAST               ORDER               DATE'_____  *
  * ORDER_QUANTITY        'NUMBER             ORDERED'_____________________  *
  *                                                                    *
  *                                                                    *
  *                                                                    *
  *                                                                    *
  *                                                                    *
  *                                                                Bottom  *
  * F3=Exit          F5=Refresh   F6=Insert line   F10=Copy line   F12=Cancel  *
  * F14=Delete line  F19=Display system column names   F24=More keys   *
  *                                                                    *
```

对每列键入标题，列标题可由 20 个字符段定义。当显示 SELECT 语句时，每段将在不同行显示。列标题区顶部的标尺，可以很容易的移动。当键入列标题后，按 Enter 键。

下面信息表示 LABEL ON 语句成功执行了：

```
LABEL ON for INVEN0001 in SAMPLECOLL Completed。
```

这条信息中的表名是这个表的系统表名，而不是在语句中实际指定的名字。AS/400 DB2 维护两个长于 10 个字符的表名的系统表名详细介绍请看《AS/400 DB2 参考》

LABEL ON 语句也能在 enter SQL statement 显示中直接输入：

```
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')
```

2.5 往表中插入信息

表建立后，可用 INSERT 加入信息。(这个语句的详细介绍在 3.1.1 中)。

在“Enter SQL Statement”显示中，键入 INSERT 再按 F4 键，出现下面的显示：

```
*                                                                    *
*                          Specify INSERT Statement                  *
*                                                                    *
* Type choices, press Enter.                                         *
*                                                                    *
*   INTO table . . . . . .    INVENTORY_LIST_____    Name, F4 for list  *
*     Collection . . . . . .    SAMPLECOLL__         Name, F4 for list  *
*                                                                    *
*   Select columns to insert                                         *
*     INTO  . . . . . .  .    Y                    Y=Yes, N=No        *
*   Insertion method  . . . .  1                   1=Input VALUES     *
*                                                  2=Subselect        *
*                                                                    *
* Type choices, press Enter.                                         *
*                                                                    *
*   WITH isolation level  . .  1          1=Current level, 2=NC (NONE) *
*                                         3=UR (CHG), 4=CS, 5=RS (ALL) *
*                                         6=RR                       *
*                                                                    *
*                                                                    *
*                                                                    *
* F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F20=Display full names *
* F21=Display statement                                              *
*                                                                    *
```

在显示的输入字段里键入表名和集合名，把’selet colunms to insert INTO’提示改为 Y，按 Enter 键后显示能选择做插入值的列。

```
*                                                                          *
*                          Specify INSERT Statement                        *
*                                                                          *
* Type sequence numbers (1-999) to make selections, press Enter.           *
*                                                                          *
* Seq  Column              Type         Digits  Length                     *
* 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                                *
*                                                                          *
*                                                                          *
*                                                                          *
*                                                                          *
*                                                                          *
*                                                                          *
*                                                                          *
*                                                                          *
*                                                                          *
*                                                                   Bottom *
* F3=Exit   F5=Refresh      F12=Cancel   F19=Display system column names   *
* F20=Display entire name   F21=Display statement                          *
*                                                                          *
```

在此例中我们只想插入四列，其他列按缺省值插入，显示的顺序号表明在 INSERT 语句中列和值的顺序。按 Enter 键，显示能输入值的列。

```
*                                                                          *
*                          Specify INSERT Statement                        *
*                                                                          *
* Type values to insert, press Enter.                                      *
*                                                                          *
* Column              Value                                                *
* ITEM_NUMBER         '153047'_____________________________________        *
* ITEM_NAME           'Pencils, red'_______________________________        *
* UNIT_COST           10.00________________________________________        *
* QUANTITY_ON_HAND    25___________________________________________        *
*                                                                          *
*                                                                          *
*                                                                          *
```

```
*                                                                              *
*                                                                              *
*                                                                              *
*                                                                        Bottom *
* F3=Exit      F5=Refresh   F6=Insert line   F10=Copy line   F11=Display type  *
* F12=Cancel   F14=Delete line   F15=Split line   F24=More keys                *
*                                                                              *
```

注：按 F11 键，显示每一列的数据类型和长度。还会显示不同的插入值视图，提供列定义的有关信息。

键入为所有列插入的值然后按 Enter 键，包含这些值的行就加到表中。没有规定值的列用缺省值插入。因 LAST_ORDER_DATE 没有缺省值又允许为空，所以它为空值。因在 CREATE TABLE 中，规定了 ORER_QUANTITY 的缺省值，所以它的值为 20。

INSERT 语句可在 Enter SQL Stutement 显示上直接输入：

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

要在表中添下一行，在 Enter SQL Stutement 显示上按 F9 键。这个操作将以前的 INSERT 语句复制到键入区，你可在前面 INSERT 语句上输入，也可用 F4 键来使用交互 SQL 显示来输入数据。

继续使用 INSERT 语句向表中添加其余的行。在下表中未给出的值，使用缺省值。在 INSERT 列表中，只指定想向其中插入值的列名。例如，在插入第三行时，可只规定 ITEM_NUMBER 和 UNIT_COST 两个列名及在 VALUES 列表中给出这些列的二个值。

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

下面这些行加到表 SAMPLECOLL.SUPPLIERS 中：

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

在样本集合中已有两个分别带有几行数据的表。

2.6 获取单表信息

现在已在表中插入所有信息,还需再查看一下。在 SQL 中,可用 SELECT 语句处理。SELECT 语句是所有 SQL 语句中最复杂的，这个语句由 3 个子句构成：

1. SELECT 子句：规定包括要求数据的那些列。

2. FROM 子句：规定包括要求数据列的表。

3. WHERE 子句：规定提供要求数据行的条件。

除这三个主要子句外，还有几个其它的子句，请看 3.1，它们会影响返回数据的格式。

键入 SELECT 并按 F4 键(提示)，会看到我们在 INENTORY_LIST 表中插入的数据。下面是显示结果：

```
*                                                                    *
*                          Specify SELECT Statement                  *
*                                                                    *
* Type SELECT statement information.  Press F4 for a list.           *
*                                                                    *
*   FROM tables  . . . . . .   SAMPLECOLL.INVENTORY_LIST_____       *
*   SELECT columns . . . . . .   *_____________________________       *
*   WHERE conditions . . . . .   _____________________________       *
*   GROUP BY columns . . . . .   _____________________________       *
*   HAVING conditions  . . . .   _____________________________       *
*   ORDER BY columns . . . . .   _____________________________       *
*   FOR UPDATE OF columns  . .   _____________________________       *
*                                                                    *
*                                                          Bottom    *
* Type choices, press Enter.                                         *
*                                                                    *
*   DISTINCT rows in result table  . . . . . .   N    Y=Yes, N=No    *
*   UNION with another SELECT  . . . . . . . .   N    Y=Yes, N=No    *
*   Specify additional options . . . . . . . .   N    Y=Yes, N=No    *
*                                                                    *
*                                                                    *
*                                                                    *
```

```
* F3=Exit         F4=Prompt   F5=Refresh   F6=Insert line   F9=Specify subquery    *
* F10=Copy line   F12=Cancel   F14=Delete line   F15=Split line   F24=More keys    *
*                                                                                 *
```

在显示的“FROM table”字段中键入表名。要选择表中所有列，在 SELECT 列键入*，按 Enter 键，语句将运行来选择表中所有列中的全部数据。下面显示输出结果：

```
*                                                                                 *
*                                  Display Data                                   *
*                                                 Data width . . . . . :      71  *
* Position to line  . . . . .              Shift to column  . . . . . .           *
* ....+....1....+....2....+....3....+....4....+....5....+....6....+....7.         *
* ITEM    ITEM                        UNIT   QUANTITY  LAST      NUMBER           *
* NUMBER  NAME                        COST   ON        ORDER     ORDERED          *
*                                            HAND      DATE                       *
* 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, standard          .00        -   -              20          *
* 775298  Chairs, secretary         225.00        6   -              20          *
* 073956  Pens, black                20.00       25   -              20          *
* ********  End of data  ********                                                 *
*                                                                                 *
* F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split             *
*                                                                                 *
```

显示用 LABLE ON 语句定义的列标题内容显示，第三项的“ITEM_NAME”在 CREATE TABLE 语中指定了缺省值。这行的 QUANTITY_ON_HAND 列没有插入任何值而有空值。LAST_ORDER_DATE 没在 INSERT 语句中指定值也没有指定缺省值，所以 LAST_ORDER_DATE 列全部为空值。同样，ORDER_QUANTITY 列全部为缺省值。

可在“Enter SQL Statements”显示中输入这条语句：

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

为约束由 SELECT 语句返回列的数目，必须指定想看到的列。为约束返回输出行的数量，必须使用 WHERE 子句。要只看到多于 10 美元的项，且只想要返回 ITEM_NUMBER，UNIT_COST 和 ITEM_NAME，键入 SELECT 并按 F4 键，将显示规定的 SELECT 语句：

```
*                                                                          *
*                          Specify SELECT Statement                        *
*                                                                          *
* Type SELECT statement information.  Press F4 for a list.                 *
*                                                                          *
*   FROM tables  . . . . . .   SAMPLECOLL.INVENTORY_LIST_____________      *
*   SELECT columns . . . . . .   ITEM_NUMBER, UNIT_COST, ITEM_NAME_____      *
*   WHERE conditions . . . . . .   UNIT_COST > 10.00_________________      *
*   GROUP BY columns . . . . . .   _________________________________      *
*   HAVING conditions  . . . . .   _________________________________      *
*   ORDER BY columns . . . . . .   _________________________________      *
*   FOR UPDATE OF columns  . . .   _________________________________      *
*                                                                          *
*                                                                Bottom    *
* Type choices, press Enter.                                               *
*                                                                          *
*   DISTINCT rows in result table  . . . . . .   N    Y=Yes, N=No         *
*   UNION with another SELECT  . . . . . . . .   N    Y=Yes, N=No         *
*   Specify additional options . . . . . . . .   N    Y=Yes, N=No         *
*                                                                          *
*                                                                          *
*                                                                          *
* F3=Exit         F4=Prompt   F5=Refresh   F6=Insert line   F9=Specify subquery   *
* F10=Copy line   F12=Cancel   F14=Delete line   F15=Split line   F24=More keys   *
*                                                                          *
```

最初在 Specify SELECT statements 显示中，对于每一个提示只显示一行，可用 F6 键在显示顶部的插入区域加更多行。这在 SELECT 列表中要输入很多列，或在需要更长更复杂的 WHERE 条件时，这一功能很有用。

在上面显示中填上所示内容，当按下 Enter 键时，将运行 SELECT 语句。可以看到下面的输出：

```
*                                                                          *
*                                Display Data                              *
*                                          Data width . . . . . :     41   *
* Position to line  . . . . .            Shift to column  . . . . . .      *
* ....+....1....+....2....+....3....+....4.                                *
* ITEM          UNIT   ITEM                                                *
* NUMBER        COST   NAME                                                *
* 775298      225.00   Chairs, secretary                                   *
* 073956       20.00   Pens, black                                         *
```

```
* ********  End of data  ********                                          *
*                                                                         *
* F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split     *
*                                                                         *
```

只有那些数据和 WHERE 子句中规定的条件比较后符合条件的行才能返回，且返回的值只取自那些在 SELECT 子句中明显规定的列，没有明显规定列的数据不能返回。

上面这些条件可在 Enter SQL statements 显示中输入下面的语句来实现：

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

2.7 获取多表信息

SQL 允许从多个表的列中获取信息。这个操作叫连接操作。(要详细了解，请看 5.4)。用 SQL，连接操作就是把要连在一起的表名写到 SELECT 语句的 FROM 子句中。

假定想看所有供货商及货物数量和他们供应的货物名称，货物名称不在 SUPPLIER 表中，它在 INVENTORY_LIST 表中，使用公共列 ITEM_NUMBER，我们会看到其它三列列好像从一个表中取出的一样。

在要连接的多个表中有相同列名时，必须用表名来限定实际要引用的列。在 SELECT 语句中，列名 ITEM_NUMBER 在两个表中都有定义，所以需要用表名限定要引用的列。若列名不同，则不存在混淆现象，当然也不用限定了。

为形成连接，在“Enter SQL statement”屏上直接输入或由提示输入下面的 SELECT 语句，如使用提示，在 FROM tables 输入行上需要键入两个表名：

```
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
```

输入相同语句的另一种方法是使用相关名。相关名提供在一个语句中使用的另一个名称。相关名必须用在表名相同时，它能在 FROM 列表中指定每个表名。前面的语句可写为：

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

在这个例子中，SAMPLECOLL.SUPPLIES 的相关名为 X，SAMPLCOLL.INVENTORY_LIST 相关名为 Y，用 X 和 Y 限定 ITEM_NUMBER 列名。

相关名的详细内容，请看《AS/400 DB2 SQL 参考》。

运行上面的语句后得到下面输出：

```
*                                                                    *
*                              Display Data                          *
*                                        Data width . . . . . :  45  *
* Position to line  . . . . .          Shift to column  . . . . .    *
* ....+....1....+....2....+....3....+....4....+                      *
* SUPPLIER_NUMBER  ITEM    ITEM                                       *
*                  NUMBER  NAME                                       *
*      1234        153047  Pencils, red                              *
*      9988        153047  Pencils, red                              *
*      2424        153047  Pencils, red                              *
*      1234        229740  Lined tablets                             *
*      1234        303476  Paper clips                               *
*      2424        303476  Paper clips                               *
*      3366        303476  Paper clips                               *
*      9988        559343  Envelopes, legal                          *
*      5546        775298  Chairs, secretary                         *
*      3366        073956  Pens, black                               *
* ********  End of data  ********                                     *
*                                                                    *
* F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split  *
*                                                                    *
```

结果表中的数值表示了包含在 INVENTORY_LIST 和 SUPPLIERS 两个表中的组合数据，它包含了 SUPLLIER 表中的供货商号码及 INVENTORY_LIST 表中的货物号码和货物名称。SUPPLIER 中没有的货物号码不出现在结果表中，除非在 SELECT 语句中的 ORDER BY 子句中有规定，否则结果不会按任何顺序排序。因为我们没改变 SUPPLIER 的列标题，所以用 SUPPLIER_NUMBER 列名做为列标题。

下面是使用 ORDER BY 子句保证行顺序的一个例子。语句首先按 SUPPLIER_NUMBER 列排序，在有相同 SUPPLIER_NUMBER 值的行，按 ITEM_NUMBTER 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
```

运行前面语句将产生以下输出：

```
*                                                                    *
*                              Display Data                          *
*                                        Data width . . . . . :  45  *
* Position to line  . . . . .          Shift to column  . . . . .    *
```

```
* ....+....1....+....2....+....3....+....4....+                    *
* SUPPLIER_NUMBER  ITEM    ITEM                                     *
*                  NUMBER  NAME                                     *
*      1234        153047  Pencils, red                             *
*      1234        229740  Lined tablets                            *
*      1234        303476  Paper clips                              *
*      2424        153047  Pencils, red                             *
*      2424        303476  Paper clips                              *
*      3366        073956  Pens, black                              *
*      3366        303476  Paper clips                              *
*      5546        775298  Chairs, secretary                        *
*      9988        153047  Pencils, red                             *
*      9988        559343  Envelopes, legal                         *
* ********  End of data  ********                                   *
*                                                                   *
* F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split *
*                                                                   *
```

2.8 更改表信息

使用 UPDATE 语句，可修改表中一些列或全部列的数值。可用 UPDATE 语句中 WHERE 子句在执行单个语句时约束可修改的行数。如没有 WHERE 子句，那么表中的所有行都改变。若用了 WHERE 子句，那么只有满足条件的行才能改变。(更详细了解 UPDATE 语句和 WHERE 子句，请参阅 3.1.2 和 3.1.7)。

假设要为多个 Paper dips 排序，要修改货物号 303476 的 LAST_ORDER_DATE 和 ORDER_QUANTITY，键入 UPDATE 并按 F4 键，规定的 UPDATE 将显示为：

```
*                                                                   *
*                          Specify UPDATE Statement                 *
*                                                                   *
* Type choices, press Enter.                                        *
*                                                                   *
*   Table  . . . . . .   INVENTORY_LIST______    Name, F4 for list  *
*     Collection . . . .     SAMPLECOLL__        Name, F4 for list  *
*                                                                   *
*   Correlation  . . . .   ____________________  Name               *
*                                                                   *
*                                                                   *
*                                                                   *
*                                                                   *
*                                                                   *
*                                                                   *
```

```
*                                                                          *
*                                                                          *
*                                                                          *
*                                                                          *
*                                                                          *
*                                                                          *
*                                                                          *
* F3=Exit   F4=Prompt   F5=Refresh   F12=Cancel   F20=Display full names  *
* F21=Display statement                                                    *
*                                                                          *
```

键入表名和集合名后，按 Enter 键，显示下列内容：

```
*                                                                          *
*                          Specify UPDATE Statement                        *
*                                                                          *
* Type choices, press Enter.                                               *
*                                                                          *
*   Table  . . . . . .   INVENTORY_LIST_____     Name, F4 for list         *
*     Collection . . . . .    SAMPLECOLL__       Name, F4 for list         *
*                                                                          *
*   Correlation  . . . . .   ____________________     Name                  *
*                                                                          *
*                                                                          *
* Type information, press Enter.                                           *
*                                                                          *
* Column               Value                                               *
* ITEM_NUMBER          ____________________________________________        *
* ITEM_NAME            ____________________________________________        *
* UNIT_COST            ____________________________________________        *
* QUANTITY_ON_HAND     ____________________________________________        *
* LAST_ORDER_DATE      CURRENT DATE________________________________        *
* ORDER_QUANTITY       50__________________________________________        *
*                                                                          *
*                                                                   Bottom *
* F3=Exit   F4=Prompt   F5=Refresh   F6=Insert line    F10=Copy line      *
* F11=Display type      F12=Cancel   F14=Delete line   F24=More keys      *
*                                                                          *
```

将所有选择行的 CURRENT DATE 的值改变为今天的日期。

键入表中要更新的值以后，按 Enter 键，会看到规定 WHERE 条件的显示。如 WHERE 条件

没规定，那么表中所有行将按前面显示的值更新。

```
  *                                                                    *
* *                          Specify UPDATE Statement                  *
  *                                                                    *
* * Type WHERE conditions, press Enter.  Press F4 for a list.          *
* *   ITEM_NUMBER = '303476'_____________________________________      *
* *   ___________________________________________________________      *
  *                                                                    *
  *                                                                    *
  *                                                                    *
* *                                                          Bottom    *
* * Type choices, press Enter.                                         *
  *                                                                    *
* *   WITH isolation level . . .   1              1=Current level, 2=NC (NONE)   *
* *                                               3=UR (CHG), 4=CS, 5=RS (ALL)   *
* *                                               6=RR                           *
  *                                                                    *
  *                                                                    *
  *                                                                    *
  *                                                                    *
  *                                                                    *
  *                                                                    *
  *                                                                    *
* * F3=Exit         F4=Prompt   F5=Refresh   F6=Insert line   F9=Specify subquery   *
* * F10=Copy line   F12=Cancel   F14=Delete line   F15=Split line   F24=More keys   *
  *                                                                    *
```

键入条件后按 Enter 键将执行表的更新，也显示功能已完成的信息。

也可在‘Enter SQL statements’显示中输入下列语句：

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

运行 SELECT 语句，从表中得到全部行(SELECT * FROM SAMPLE.INVENTORY_LIST)，返回结果如下：

```
  *                                                                    *
* *                              Display Data                          *
```

```
* *                                          Data width . . . . . :      71    *
* * Position to line  . . . . .             Shift to column  . . . . .          *
* * ....+....1....+....2....+....3....+....4....+....5....+....6....+....7.     *
* * ITEM    ITEM                        UNIT   QUANTITY  LAST      NUMBER       *
* * NUMBER  NAME                        COST   ON        ORDER     ORDERED      *
* *                                           HAND      DATE                   *
* * 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  05/30/94      50       *
* * 559343  Envelopes, legal             3.00      500  -             20       *
* * 291124  Envelopes, standard           .00        -  -             20       *
* * 775298  Chairs, secretary          225.00        6  -             20       *
* * 073956  Pens, black                 20.00       25  -             20       *
* * ********  End of data  ********                                            *
* *                                                                  Bottom    *
* * F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split          *
  *                                                                           *
```

只有 Paper dips 这项改变了，LAST_ORDER_DATE 更改为当前日期，这个日期总是运行 UPDATE 的日期。NUMBER_ORDERED 显示更新后的值。

2.9 删除表信息

DELETE 语句删除表中整个一行，因为这些行不再有用。DELETE 允许在执行单条语句时用 WHERE 子句约束要删除的行。本书 3.1.3 中有详细介绍。

若我们想删除表中 QUANTITY_ON_HAND 列有空值的行，用下面的语句：

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

为检查空值，要用 IS NULL 比较，完成 DELETE 后运行另一条 SELECT 语句，将返回下面内容：

```
  *                                                                           *
  *                                 Display Data                              *
  *                                          Data width . . . . . :      71    *
  * Position to line  . . . . .             Shift to column  . . . . .          *
  * ....+....1....+....2....+....3....+....4....+....5....+....6....+....7.     *
  * ITEM    ITEM                        UNIT   QUANTITY  LAST      NUMBER       *
  * NUMBER  NAME                        COST   ON        ORDER     ORDERED      *
```

```
*                                        HAND      DATE                    *
* 153047  Pencils, red              10.00    25    -             20        *
* 229740  Lined tablets              1.50   120    -             20        *
* 303476  Paper clips                2.00   100    05/30/94      50        *
* 559343  Envelopes, legal           3.00   500    -             20        *
* 775298  Chairs, secretary        225.00     6    -             20        *
* 073956  Pens, black               20.00    25    -             20        *
* ********  End of data  ********                                          *
*                                                                 Bottom   *
* F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split       *
*                                                                         *
```

QUANTITY_ON_HAND 列有空值的行被删除了。

2.10 建立和使用视图

你可能会发现没有一个单表能包含想要的全部信息，也可能只想让用户访问表中的部分数据。要实现这些功能，可使用 CREATE VIEW 语句来做一个表的子集。

视图让你只处理需要的数据，这样就变得简单且能约束存取某些数据。

在应用程序使用视图时，它不能访问视图中没有包括的列或行。如果不用 WITH CHECK OPTION，通过视图仍可插入不符合选择条件的行，详细内容请看第六章。

生成视图的方法很类似生成表，它是由 CREATE VIEW 语句定义的，定义一个视图类似生成一个就包含你想要的那些列和行的一个新表。要生成一个视图，必须对它依赖的表或物理文件有相应的权限，详细内容请看 AS/400 DB2 SQL 参考。

如果在视图定义中没有规定列名，那么视图中的列名与表中一致。即使视图与其表有不同的行数或列数，也可以通过视图来修改表。对于 INSERT 语句，没在视图中的列必须有缺省值。可把视图就看做一个表，但它的数据是依赖于一个或多个表的，视图本身没有数据，这样就不必用放数据的存储空间。由于视图是由存在存储中的表而得到的，在修改视图数据时，实际上修改的是表中数据。这样，视图自动的与其所用的表保持数据的一致，详细内容请看 5.8。

2.10.1 建立单表视图

下例给出建立单表视图，此表有六列，但视图仅用其中三列。在 SELECT 语句中的列的顺序做为它们出现在视图中的顺序，视图仅包括在最后二周定货的那些项，使用的语句如下：

```
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
```

在上面的例子中，视图的列名同表中列名。放视图的集合名不必同它依赖的表的集合名

相同，可以使用任何一个集合或库。下面是运行 SQL 语句的输出结果：

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

```
*                                                                    *
*                              Display Data                          *
*                                          Data width . . . . . :    26    *
* Position to line  . . . . .          Shift to column  . . . . . .          *
* ....+....1....+....2....+.                                         *
* ITEM    LAST      QUANTITY                                         *
* NUMBER  ORDER     ON                                               *
*         DATE      HAND                                             *
* 303476  05/30/94      100                                          *
* ********  End of data  ********                                    *
*                                                              Bottom    *
* F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split          *
*                                                                    *
```

仅选择了把日期更新为当前日期的那一行，表中其他日期值仍为空值，所以不出现。

2.10.2　建立多表数据合并的视图

在 FROM 子句中规定多个表名，能从多表生成一个合并数据的视图。在下例中，表 INVENTORY_LIST 中包含名为 ITEM_NUMBER 和 UNIT_COST，它们与表 SUPPLIERS 中的 ITEM_NUMBER 和 SUPPLIER_COST 列合并。WHERE 子句约束返回的行数，视图仅包括成本低于当前单位成本的货物号码：

```
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
```

下面是运行 SQL 语句的结果：

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

```
*                                                                    *
*                              Display Data                          *
*                                          Data width . . . . . :    51    *
* Position to line  . . . . .          Shift to column  . . . . . .          *
* ....+....1....+....2....+....3....+....4....+....5.                *
* SUPPLIER_NUMBER  ITEM          UNIT   SUPPLIER_COST                *
```

```
*                NUMBER        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                         *
* ********  End of data  ********                                          *
*                                                                 Bottom    *
* F3=Exit      F12=Cancel      F19=Left      F20=Right      F21=Split       *
*                                                                         *
```

从这个视图只能看到供应成本低于单位成本的那些行。

第三章　基本概念和技术

这一章解释 SQL 语句中的一些概念，介绍 SQL 公共语句和子句。这一章引用本书原文附录 A 中的表。

3.1　使用基本的 SQL 语句和子句

这部分介绍基本 SQL 语句和检索、修改、删除以及把数据插入表和视图中的子句。SQL 语句中常用的是 SELECT、UPDATE、DELETE 和 INSERT 四个语句。FETCH 语句也用在应用程序里来存取数据，这个语句在第四章介绍。用 SQL 语句的例子将有助于开发 SQL 应用程序。关于 SQL 语句语法和参数的详尽描述在 AS/400 DB2 SQL 参考中给出。

SQL 语句可在一行或多行上写，续行的规则与主语言(写程序的语言)相一致。

注：

1.这里介绍的 SQL 语句可在 SQL 表和视图中运行，也可在数据库物理文件、逻辑文件上运行，表、视图和文件可在 SQL 集合里，也可在库里。

2.SQL 语句里的字符串(例如 WHERE 或 VALUES 子句)是区分大小写的，也就是说，大写字母必须以大写体输入，小写字母必须以小写体输入。

```
WHERE ADMRDEPT='a00'      (无返回结果)

WHERE ADMRDEPT='A00'      (返回有效的部门号)
```

如果使用共享权分类排序，它不区分大小写母，那么比较时不区别大小写。

3.1.1　INSERT 语句

按以下方式之一，用 INSERT 语句在表或视图中加入新行：

在 INSERT 语句里规定加一行的各列的值。

规定加入多行的 INSERT 语句的块格式，详细内容请看 5.1.2。

在 INSERT 语句里包含一个选择语句，它告诉 SQL，新一行数据来自另一个表或视图。详细内容请看 5.1.1。

注：

由于视图是由表建立的，实际上不包括数据，因此将涉及与视图相关的工作，详细内容请看 5.8。

对插入的每一行,如果某列没规定缺省值,那么必须给定义为非空属性的列提供一个值。用 INSERT 语句添加一行到表或视图中的格式为：

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

INTO 子句规定列的名字，VALUES 子句给在 INTO 子句里规定的列指定值。

在 INSERT 语句列表里命名的列，都必须在 VALUES 子句里提供一个值。如果表中各列都有在 VALUES 子句里提供的值,那么列名可以省略。如果某一列有一个缺省值,则键字 DEFAULT 可用做 VALUES 子句中的值。

列出要插入值的各列名是一个好办法，这是因为：

INSERT 语句更具有描述性。

可以按列名顺序核对给定值。

有较好的数据独立性，表中定义的列序并不影响 INSERT 语句。

如果列定义为允许空值或有缺省值，就不需要在列名表中给名或为该列设定值，直接使用缺省值。如果这列有缺省值，将缺省值放置到列中。如果对列定义了 DEFAULT 但没有明显给出缺省值，SQL 根据数据类型放置缺省值。如果列设有定义缺省值，但定义允许空值(在列定义中没规定 NOT NULL)，SQL 把空值放在此列中。

对数字列，缺省值为零。

对定长字符或图形列，缺省值为空格。

对变长字符或图形列，缺省值为零长度的串。

对日期、时间和时间标记列，缺省值为当前日期、时间或时间标记。当插入记录块时，缺省的日期/时间值在块写入时从系统中提取，这意味着对块里的每行分配相同的缺省值。

当程序试图插入表中已存在的行时，可能出现错误。根据建立索引时所使用的可选项，多空值可以认为是重复值，也可不认为是重复值：

如果表中有一个主键、唯一键或唯一索引,不能插入行,SQL 返回的 SQLCODE 为-803。

如果表中没有主键、唯一键或唯一索引，可以插入行且无错。

如果在运行 INSERT 语句时 SQL 发现错误,则停止数据插入。如果规定了 COMMIT(*ALL)，COMMIT(*CS)，COMMIT(*CHG)或 COMMIT(*RR)，则不会插入行。由该语句已插入的行在用带有选择语句或块插入的 INSERT 语句时，删除这些行，如果规定了 COMMIT(*NONE)，则不删除已插入的行。

由 SQL 生成的表是用可重用删除记录参数为*YES 来生成的，这就允许数据库管理重用表中做删除标志的行。可用 CHGPF 命令修改其属性为*NO，这将导致 INSERT 语句总在表末添加行。

插入行的顺序不能保证就是其检索的顺序。

如果插入行无错，那么 SQLCA 的 SQLERRD(3)字段的值为 1。

注：

对成块 INSERT 或有选择语句的 INSERT，能插入多行，插入的行数也反映在 SQLERRD(3)中。

3.1.2 UPDATE 语句

若要改变表中的数据，可以使用 UPDATE 语句。通过该语句，可以改变每行中的一列或多列值，而这列要满足 WHERE 子句的查询条件。(根据有多少行满足 WHERE 子句指定的查询条件而定)，UPDATE 语句的结果是表的零或多行的一列或多列改变。UPDATE 语句格式如下：

```
UPDATE 表名
  SET 列 - 1 = 值 - 1,
```

```
        列 - 2 = 值 - 2, ...
    WHERE 检索条件 ...
```

例如，假设一个雇员进行重定位，为了修改 CORPDATA.EMPLOYEE 表中雇员数据的几项来反映这种变化，规定如下：

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

使用 SET 子句为要修改的列设定新值。SET 子句给出要修改的列名以及要修改的值，规定值可以是：

列名。用同一行中另一列的内容替换列的当前值。

常量。用 SET 子句提供的值替换列的当前值。

空值。使用键字 NULL，以空值替换列的当前值。当建立表时，列必须定义为可包括空值的属性，否则出错。

主变量。用主变量的内容替换列的当前值。

专用寄存器。用专用寄存器值替换列的当前值；例如，USER。

表达式。用表达式结果值替换列的当前值。表达式可以包括列表中的任何值。

键字 DEFAULT。用列的缺省值替换列的当前值。列必须定义过缺省值，或者允许空值，否则出错。

下面是使用多个不同值的语句例子：

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

为了规定需要修改的行，使用 WHERE 子句：

要修改一行，使用仅选择一行的 WHERE 子句。

要修改多行，使用选择需要修改那些行的 WHERE 子句。

可以省略 WHERE 子句，如果这样，SQL 将修改表或视图中提供值的每一行。

如果数据库管理在运行 UPDATE 语句时发现错误，它将停止修改并返回负的 SQLCODE。如果规定 COMMIT(*ALL)，COMMIT(*CS)，COMMIT(*CHG)或 COMMIT(*RR)，那么表中无行发生改变(如果有由这个语句发生改变的行，将变回以前值)。如果规定 COMMIT(*NONE)，已发生改变的行不变回以前的值。

如果数据库管理没有找到满足查询条件的行，返回的 SQLCODE 为+100。

注：

带有 WHERE 子句的 UPDATE 语句可修改多行。修改的行数也反映在 SQLERRD(3)中。

3.1.3 DELETE 语句

若要从表中取消几行，使用 DELETE 语句。当删除一行时，就取消了整行。DELETE 不能取消行中的某些列。DELETE 语句的结果是删除表中的零行或多行(根据有多少行满足 WHERE 子句设定的查询条件而定)。如果在 DELETE 语句中没有 WHERE 子句，SQL 将取消表中的全部行。DELETE 语句格式为：

```
DELETE  FROM  表名
WHERE  检索条件
```

例如，假设部门 D11 被移到另一个地方，可以按下面方法删除 CORPDATA.EMPLOYEE 表里 WORKDEPT 值为 D11 的每一行：

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

WHERE 子句告诉 SQL 要从表中删除哪些行。SQL 删除所有符合基本表查询条件的行。可以省略 WHERE 子句，但最好还是包括它，因为没有 WHERE 子句的 DELETE 语句将从表或视图中删除所有行。若要删除表定义和表内容，要用 DROP 语句(请参看 AS/400 DB2 SQL 参考)。

SQL 在运行 DELETE 语句时如发现错误，将停止删除数据并返回负的 SQLCODE。如果规定 COMMIT(*ALL)，COMMIT(*CS)，COMMIT(*CHG)，COMMIT(*RR)，表中设有行被删除(如果有由该语句已删除的行，将保留从前值)。如果规定 COMMIT(*NONE)，则已删除的行不保留从前值。

如果 SQL 未找到满足查询条件的行，SQLCODE 返回+100。

注：带 WHERE 子句的 DELETE 语句可删除多行。删除行的数目也反映在 SQLERRD(3)中。

3.1.4 SELECT　INTO 语句

可以在程序中用 SELECT　INTO 语句检索指定行(例如，一个雇员行)。这里给出的格式和语法都是最基本的。实际上 SELECT　INTO 语句可比本章中所举的例子变化更多。SELECT INTO 语句包括以下几部分：

1、需要的每列的名称。
2、包含检索数据的每个主变量的名称。
3、包含数据的表或视图的名称。
4、包含需要信息的唯一标识行的查询条件。
5、用于数据分组的每列的名称。
6、包含需要信息的唯一标识组的查询条件。
7、返回多行中某行的结果顺序。

SELECT　INTO 语句格式为：

```
SELECT          列名
  INTO          主变量
  FROM          表或视图名
  WHERE         检索条件
  GROUP BY      列名
```

```
HAVING        检索条件
ORDER BY      列名
```

必须指定 SELECT，INTO 和 FROM 子句，其它子句都是可选的。

INTO 子句命名主变量(程序中使用的放列值的变量)。SELECT 子句规定的第一列值放到 INTO 子句的第一个主变量里。第二个值放到第二个主变量中，以此类推。

SELECT INTO 语句的结果表只包括一行。例如，CORPDATA.EMPLOYEE 表中的每一行有唯一的 EMPNO(雇员号)列，如果 WHERE 子句在 EMPNO 列上有一个相等的比较，这个表的 SELECT INTO 语句的结果确切为一行(或没有行)。找到多行是错误的，只能返回一行。规定 ORDER BY 子句，能控制在这种错误条件下返回哪行。如果用 ORDER BY 子句，返回结果表的第一行。

如果需要多行的选择语句结果，使用 DECLARE CURSOR 语句来选择行，接着用 FETCH 语句一次性的把列值移入主变量的一或多行。CURSORS 的使用在第四章讲述。

FROM 子句规定包含数据的表(或视图)。

假定 CORPDATA.DEPARTMENT 表里的每个部门都有唯一的部门号，要想从此表中检索部门 C01 的部门名和经理号，为此，程序设置 PGM-DEPT 值为 C01 并执行下列语句：

```
SELECT DEPTNAME, MGRNO
  INTO :PGM-DEPTNAME, :PGM-MGRNO
  FROM CORPDATA.DEPARTMENT
  WHERE DEPTNO = :PGM-DEPT
```

当运行这个语句时，结果为一行：

PGM-DEPTNAME	PGM-MGRNO
INFORMATION CENTER	000030

这些值都分配给主变量 PGM-DEPTNAME 和 PGM-MGRNO。

如果 SQL 找不到满足查询条件的行，返回 SQLCODE+100。

如果 SQL 在运行选择语句时发现错误，将返回负的 SQLCODE。如果 SQL 发现主变量多于结果，返回+326。

检索视图中的数据与检索表中的数据方法一样，但用视图做修改、插入或删除数据时，有一些限制。这些限制将在 5.8 中讲述。

3.1.5 数据检索错误

如果 SQL 发现检索字符或图形列过长不能装入主变量中，SQL 做以下操作：

当把值分配给主变量时，将截断数据尾部。

设置 SQLCA 中的 SQLWARN0 和 SQLWARN1 值为 ‘W’。

如果提供指示器变量，则设置为截断前值的长度。

如果 SQL 在运行语句时发现数据映象错误，将发生下面两件事之一：

如果错误发生在 SELECT 列表的表达中且提供指示器变量表示错误的表达式：

—对应这个出错表达式，SQL 返回-2 给指示器变量。

—SQL 返回那行的所有有效数据。

—SQL 返回正的 SQLCODE。

如果不提供指示器变量，SQL 在 SQLCA 中将返回相应的负 SQLCODE。

数据映象错误包括：

+138——完成子串功能的自变量无效。

+180——日期、时间或时间标记的串表达式语法无效。

+181——日期、时间或时间标记的串表达式值无效。

+183——日期/时间表达式结果无效。结果日期或时间标记不在日期或时间标记的有效范围之内。

+191——MIXED 数据不符合正确的形式。

+304——数字转换错误。(例如，溢出，下溢或零做除数错)。

+331——字符无法转换。

+420——在 CAST 自变量中的字符无效。

+802——数据转换或数据映象错。

对数据映象错误，SQLCA 仅报告检测到的最后错误。对应于每列错误结果，相应的指示器变量为-2。

如果全选择包括在选择列表中的 DISTINCT 且选择列表中的某列包括无效的数字数据，如果查询按一种分类完成，则认为数据等于空值。如果使用已有的索引，则认为数据不为空。

依照不同情况，ORDER BY 子句的数据映象错误的影响为：

当在 SELECT INTO 或 FETCH 语句里分配数据给主变量时，如果发生数据映象错误，且在 ORDER BY 子句里使用同一表达式时，结果记录按表达式值排序。没有排序就认为它为空(高于所有其它值)。这是因为在分配主变量之前计算表达式的值。

如果在选择列表计算表达式值且在 ORDER BY 子句里使用此表达式时发生数据映象错误，那么结果列正常排序，就好像它为空值(高于所有其它值)。如果用分类实现 ORDER BY 子句，则排序的结果列好像是空值。如果用索引实现 ORDER BY 子句，在下面的情况里，结果列按索引中表达式的实际值排序：

—表达式为日期列，其格式为*MDY，*DMY，*YMD，或*JUL，由于日期不在有效范围内，将出现日期转换错误。

—表达式为字符列，字符不转换。

—表达式为十进制列，检查无效的数字值。

3.1.6 SELECT 子句

用 SELECT 子句(选择语句的第一部分)，设定要检索的每列名称，例如：

```
SELECT  EMPNO，LASTNAME，WORKDEPT
.
.
.
```

可以指定检索一列，或者至多 8000 列。检索的列值按 SELECT 子句指定的顺序进行。

如果想检索所有的列(按出现在行中的相同顺序)，那么用星号(*)不必写列名：

```
SELECT  *
.
.
.
```

在应用程序里用选择语句时，给出列名会给程序更多的数据独立性。这是因为：

1、看源码语句时，很容易发现 SELECT 子句命名的列和 INTO 子句命名的主变量之间的

对应关系。

2、如果在存取的表或视图中添加一列，并使用“SELECT *…”，同时由源码生成程序，那么 INTO 子句中没有与新列相匹配的主变量。这列使你在 SQLCA(SQLWARN4 包括一个“W”)里得到一个警告(而不是错误)。

3.1.7 WHERE 子句

WHERE 子句设定一个检索条件，用来识别需要检索、修改、或删除的行，用 SQL 语句处理的行数由满足 WHERE 子句查询条件的行数决定。检索条件由一个或多个谓词组成，一个谓词指定一个检测，SQL 对表的规定行进行操作。

在下例中，WORKDEPT = ‘C01’ 是一个谓词，WORKDEPT 和 ‘C01’ 都是表达式，等号(=)是比较运算符。注意，字符值放在引号(’)内，数字值不用放在引号内，这种规定适合 SQL 中的所有常量值。例如，为了指定想要的部门号为 C01 的行，可以规定：

…WHERE WORKDEPT = ‘C01’

在这个例子里，检索条件由一个谓词；WORKDEPT = ‘C01’ 组成。

如果检索条件包含字符或 UCS-2 图形列谓词 ，那么当查询运行时有效的分类顺序应适用于这些谓词。详细信息请看 3.7。

3.1.7.1 在 WHERE 子句中使用表达式

WHERE 子句中的表达式命名或规定要与之相比较的内容。SQL 语言求值的表达式可以是一个字符串，日期/时间/时间标记，或数值。表达式可以是：

一个列名。例如：

…WHERE EMPNO = ‘000200’

EMPNO 命名一列，该列定义为 6 字节的字符值。相等比较(也就是 X=Y 或 X< >Y)可以在字符数据中实现，其它比较类型也可以由字符数据评估。

不能把字符串和数据进行比较，也不能在字符数据中执行算术运算(即使 EMPNO 是可出现数字的字符串)。可以增加/减少日期/时间值。

表达式可识别进行加(+)、减(-)、乘(*)、除(/)、乘方(**)或合并(CONCAT 或 ¦¦)运算的两个值，结果为一个值。表达式的操作数可能是：

一个常量(也就是，一个常数值)。

一列。

一个主变量。

函数返回值。

专用寄存器。

另一个表达式。

例如：

… WHERE INTEGER(PRENDATE-PRSTDATE) > 100

当未用圆括号指定求值顺序时，表达式按以下顺序求值：

1、前缀运算符　　2、幂运算

3、乘、除和合并运算　　4、加减运算

同一级的运算符按由左到右的顺序计算。

常量规定表达式为文字值。例如：

…WHERE 40000 < SALARY

SALARY 是列名，定义为 9 位数字压缩十进制值(DECIMAL(9,2))。它与数字常量 40000 比较。

主变量标识应用程序中的变量。例如：

…WHERE EMPNO = :EMP

专用寄存器标识由数据库管理生成的特殊值。例如：

…WHERE LASTNAME = USER

空值规定未知值的条件。

…WHERE DUE-DATE IS NULL

检索条件不必限制算术或比较运算符分开的两个列名或常量，可以指定由 AND 或 OR 来规定复杂的检索条件。无论多么复杂，当对一行求值时，都会提供一个 TRUE 或 FALSE 值。也有一个未知的真值，实际上是假的，也就是说，如果一行值为空，这个空值不作为检索结果返回。因为它不小于，等于或大于检索条件指定的值。更复杂的查询条件和谓词将在 5.3 中讲述。

全面理解 WHERE 子句，需要了解 SQL 怎样计算检索条件和谓词，怎样比较表达式值。这方面内容在 AS/400 DB2 SQL 参考中介绍。

3.1.7.2 比较运算符

SQL 支持下面列出的比较运算符：

=	等于
< >	不等于
<	小于
>	大于
< =	小于等于(或不大于)
> =	大于等于(或不小于)

3.1.7.3 NOT 键字

可在谓词前加 NOT 键字，用来指定谓词的相反值。(也就是说，如谓词是 FALSE，它为 TRUE；反之也可)。NOT 仅管其后的谓词，并不管 WHERE 子句的所有谓词。例如，想要除了在部门 C01 工作的所有雇员，可以规定：

…WHERE NOT WORKDEPT = ‘C01’

它等价于：

…WHERE WORKDEPT < > ‘C01’

3.1.8 GROUP BY 子句

不用 GROUP BY 子句，SQL 列功能的应用只返回一行。用 GROUP BY 时，功能适用于一组，组中有多少行就返回多少行。

GROUP BY 子句允许找到多行的一组特性，而不是一行的特性。当规定 GROUP BY 子句时，SQL 把选择的行分组，使每组的行在一列或多列有相匹配的值。接下来，SQL 处理每组

产生的单行结果。可以在 GROUP BY 子句中规定一列或多列给行分组。在 SELECT 语句中规定的项是行的每组属性，而不是表或视图各行的属性。

例如，CORPDATA.EMPLOYEE 表中有几组行，每组都由描述专门部门成员的行组成，为了找出每个部门人员的平均工资，可以规定：

PIC8

结果为几行，每个部门对应一行。

注：

1、给行分组并不意味给它们排序。分组是选择的行放在一组，SQL 处理组中取得的特性，排序是按升序或降序把所有行放到结果表中。(3.1.10 详细介绍如何做)。

2、如果在 GROUP BY 子句中规定的列有空值，有空值行的数据产生一行结果。

3、如果分组发生在字符或 UCS-2 图形列上，当对组运行查询时分类序列起作用。详情请看 3.7。

当用 GROUP BY 时，要命名 SQL 用来分组的列。例如，假定你需要一个从事主要项目工作的人数表，它取自该表 CORPDATA.PROJECT 表。可以规定：

PIC9

结果是公司当前主要项目和从事每一项目的人数的列表。

可以按多列指定需要分组的行。例如，能用选择语句来查找每个部门男职工和女职工的平均工资，它们来自 CORPDATA.EMPLOYEE 表。可在规定：

PIC10

由于这个例子没用 WHERE 子句，SQL 将检查并处理 CORPDATA.EMPLOYEE 表里的所有行。这些行首先按部门号分组，然后(在每个部门内)按性别分组，这些工作将在 SQL 获取每组平均工资值之前完成。

3.1.9 HAVING 子句

对用 GROUP BY 子句选择的分组，可以使用 HAVING 子句规定检索条件。HAVING 子句表明只要满足子句条件的那些组。由 HAVING 子句规定的检索条件测试每组的属性，而不是组里每行的属性。

HAVING 子句在 GROUP BY 子句之后，可包括由 WHERE 子句规定的同种检索条件。另外，可以用 HAVING 子句规定列函数。例如，假定想检索每个部门妇女的平均工资，可以使用 AVG 列函数，按 WORKDEPT 分组，同时规定 WHERE 子句的条件为 SEX = ‘F’。

为了仅要选取部门中教育水平大于等于 16(大学毕业)的全部女职员时，使用 HAVING 子句。HAVING 子句可以检测组的属性。此时，测试是 MIN(DELEVEL)：

PIC11

可以在 HAVING 子句里用 AND 或 OR 连接多个谓词，对于检索条件的任何谓词都可以使用 NOT 健字。

注：如果打算修改一列或删除一行,不能在 DECLARE CURSOR 语句中的 SELECT 中包含 GROUP BY 子句或 HAVING 子句。

没有列函数的自变量的谓词能用在 WHERE 或 HAVING 子句中，用 WHERE 子句给选择准则编码通常是更有效的，它在查询处理的初始段进行处理，HAVING 选择在结果表的录入处理时完成。

如果检索条件包括字符或 UCS-2 图形列谓词,查询运行时有效的分类排序也适合于这些谓词。详情请见 3.7。

3.1.10 ORDER BY 子句

用 ORDER BY 子句可以指定按某个顺序进行检索来选择行，按列值的升序或降序进行分类排序。ORDER BY 子句的使用与 GROUP BY 子句类似：当按分类顺序检索行时，规定列名或 SQL 使用的一些列。

例如，按部门号的字母顺序，检索女职员的名字和部门号，可用下面的选择语句：

PIC12

注：

1、所有在 ORDER BY 子句中命名的列也必须在 SELECT 列表中命名。

2、空值为最高值排序。

用表达式，列函数，或不是列名的其它内容排序，可在选择列表里规定 AS 子句。AS 子句可命名结果列，这个名字可在 ORDER BY 子句中规定。要用在 AS 子句指定的名字排序：

这个名字在选择列表中必须是唯一的。

这个名字必需是不限定的。

例如，为检索按字母顺序列出的全体职员名字，可以使用下面的选择语句：

```
SELECT LASTNAME CONCAT FIRSTNAME AS FULLNAME ...
ORDER BY FULLNAME
```

代替用列名对结果排序，可以使用一个数字。例如，ORDER BY 3 规定按结果表的第 3 列给结果定序。当排序值不是一个命名列时，使用一个数字给结果表的各行排序。

也可以指定是否希望 SQL 按升序(ASC)或降序(DESC)对行分类，升序为缺省值。在上面的选择语句里，SQL 首先返回最低部门号的行(字母的和数字的部门号)，接着返回较高部门号的行，若按降序对行排序，规定：

…ORDER BY WORKDEPT DESC

用 GROOP BY，可以规定辅助排序顺序(或几级排序)。在上例中，可能希望首先按部门号给行定序，而在每个部门内，按雇员名排序。为此，规定：

…ORDER BY WORKDEPT，LASTNAME

如果 ORDER BY 子句里用字符列或 UCS-2 图形列，这些列的排序是基于查询运行时有效的分类顺序。详情请看 3.7。

3.2 使用空值

空值表示一行里没有列值。空值不等于零或空白。空值是“未知的”。空值可作为 WHERE 和 HAVING 子句的一个条件，作为一个数学自变量。例如，WHERE 子句能规定某些行包括空值的列。通常，使用含有空值列的比较谓词，并不选择含有空值列的行。这是因为空值既不小于，等于，也不大于条件中的指定值。要选择有空值经理号的所有行，可以规定：

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

结果为：

DEPTNO	DEPTNAME	ADMRDEPT
D01	DEVELOPMENT CENTER	A00

要选经理号为非空值的行，改变 WHERE 子句为：

WHERE MGRNO IS NOT NULL

关于空值应用方面的详细内容，请看 AS/400 DB2 SQL 参考一书。

3.3 使用专用寄存器

可以在 SQL 语句里规定某种“专用寄存器”。对本地运行的 SQL 语句，专用寄存器及其内容如下表所示：

专用寄存器	内容
CURRENT DATE CURRENT_DATE	当前日期
CURRENT TIME CURRENT_TIME	当前时间
CURRENT TIMESTAMP CURRENT_TIMESTAMP	用时间标记格式的当前日期和时间
CURRENT TIMEZONE CURRENT_TIMEZONE	使用下列格式连接本地时间和 UTC 的时间间隔： local time – CURRENT TIMEZONE = UTC 它取自系统值 QUTCOFFSET.
CURRENT SERVER CURRENT_SERVER	相关数据库的名字作为在相关数据库目录中的相关数据库目录表名。
USER	作业的运行时授权标识(用户配置文件).

如果一条语句包括多个引用 CURRENT DATE，CURRENT TIME 或 CURRENT TIMESTAMP 专用寄存器，或者 CURDATE，CURTIME 或 NOW 标量函数，这些值都基于简单的时钟读。

对于远程运行的 SQL 语句，专用寄存器及其内容如下表所示：

专用寄存器	内容
CURRENT DATE CURRENT_DATE CURRENT TIME CURRENT_TIME CURRENT TIMESTAMP CURRENT_TIMESTAMP	远程系统的当前日期和时间，不是本地系统。
CURRENT TIMEZONE CURRENT_TIMEZONE	连接远程系统时间和 UTC 的时间间隔。
CURRENT SERVER CURRENT_SERVER	相关数据库的名字作为在相关数据库目录中的相关数据库目录表名。
USER	在远程系统中几个作业的运行时授权标识。

在分配表上的查询引用专用寄存器时，使用请求查询系统中的专用寄存器内容。有关分配表的详细信息，请看参见 AS/400 DB2 多系统一书。

3.4 使用日期，时间和时间标记

日期、时间和时间标记数据类型的内部格式，SQL 用户看不到。日期、时间和时间标记是字符串，分配给字符串变量。数据库管理认识下面的值为时间、日期和时间标记：

由 DATE，TIME 或 TIMESTAMP 标量函数返回的值。

由 CURRENT DATE，CURRENT TIME 或 CURRENT TIMESTAMP 专用寄存器返回的值。

算术表达式、比较式中与日期，时间或时间标记操作数、运算或比较的另一字符串。例如，在下面谓词里：

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

如果 HIREDATE 是日期列，则字符串‘1950-01-01’解释为日期。

在 UPDATE 语句的 SET 子句里，或者在 INSERT 语句的 VALUES 子句里，用来设置日期、时间或时间标记列的字符串变量或常量

关于日期、时间和时间标记的详细内容请看 AS/400 DB2 SQL 参考的第三章。

3.4.1 规定当前日期和时间值

可在表达式中用三个专用寄存器：CURRENT DATE，CURRENT TIME，CURRENT TIMESTAMP，指定当前日期、时间或时间标记。每个值都是以语句运行时间得到的日历读时钟为基础的。在同一 SQL 语句中对 CURRENT DATE，CURRENT TIME 或 CURRENT TIMESTAMP 的多次引用，使用同一值。下面语句在运行时返回 EMPLOYEE 表里每个职员的年龄(按年计算)：

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

专用寄存器 CURRENT TIMEZONE 允许本地时间转变为通用时间坐标系(UTC)。例如，一个表 DATETIME 中，包括一个名为 STARTT 的时间列，要把 STARTT 变换为 UTC，可以使用下面的语句：

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

3.4.2 日期/时间运算

加法和减法是用于日期，时间和时间标记值的唯一算术运算。可用时间间隔来增加和减少日期、时间或时间标记；或者从日期减去一个日期，从时间减去一个时间，从时间标记减去一个时间标记。关于日期和时间运算的详细介绍，请看 AS/400 DB2 SQL 参考中的第二章。

3.5 使用 LABEL ON

有时，在交互显示表时，表名，列名，视图名或 SQL 包名不是很清楚地显示，用 LABEL ON 语句，能为表名，列名，视图名或 SQL 包名建立更具描述性的标号，这些标号可在 SQL 目录里的 LABEL 列中看到。

LABEL ON 语句格式如下：

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

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

      Department Structure Table

        Reports to Dept
```

运行这些语句后，表名 DEPARTMENT 显示为 'Department Structure Table'，列名 ADMRDEPT 显示为 'heading Reports to Dept'。表，视图，SQL 包和列说明的标号不能超过 50 个字符，而列标题标号不能多于 60 字符（包括空格）。下面是 LABEL ON 语句的例子：

LABEL ON 语句给出列标题 1 和列标题 2：

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

LABEL ON 语句给 SALARY 提供 3 个标题：

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

LABEL ON 语句取消 SALARY 的列标题：

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

指定两个的 DBCS 列标题的例子：

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

这个 LABEL ON 语句给 EDLEVEL 列提供列说明：

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

LABEL ON 语句的详细信息，请看 AS/400 DB2 SQL 参考一书。

3.6 使用 COMMENT ON

在生成 SQL 目标，如表，视图，索引，包，过程或参数后，可以进一步提供有关的信息，例如，目标用途，谁使用它，还有关于它的限制及特殊的一些内容，也可以包括关于表或视图每一列的类似信息，注释不能超过 2000 字节。

如果名字不能很清楚地指出列或目标的内容，那么注释是有用的，使用注释来描述列或目标的特定内容。

下面是使用 COMMENT ON 的一个例子：

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

3.6.1 获得注释

当运行 COMMENT ON 语句之后,注释存在 SYSTABLES 或 SYSCOLUMNS 的 REMARKS 列里。(如果此行已包含一个注释，旧注释被新注释替代)。为前面的例子加 COMMENT ON 语句获得注解，语句如下：

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

3.7 在 SQL 中使用分类顺序

分类顺序定义在字符进行比较或排序时，怎样用字符集中的字符互相转换。分类顺序的详细介绍，请看 AS/400 DB2 SQL 参考一书的第一章。

分类顺序用在 SQL 语句里的所有字符和 UCS-2 图形比较中,单字节和双字节字符数据都有分类顺序表。每个单字节分类顺序表都有一个相应的双字节分类排序表，反之亦然。两个表之间的转换在执行必要的查询时完成。另外，CREATE INDEX 语句也有分类顺序(在语句运行时有效的)，它提供在索引中引用的字符列。

3.7.1 用在 ORDER BY 和记录选择的分类顺序

为了理解怎样使用分类顺序，运行在表 3-1 里的 STAFF 表的几个例子。注意，JOB 列的值为大小写混合。可以看到值'Mgr','MGR','mgr'。

表 3-1. STAFF 表

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

在下面的例子里，每个语句的显示结果使用：

*HEX 分类顺序

使用语言标识 ENU 的共享权分类顺序。

使用语言标识 ENU 的唯一权分类顺序。

注：

选定 ENU 做语言标识，这是在 CRTSQLxxx，STRSQL 或 RUNSQLSTM 命令中指定 SRTSEQ(*LANGIDUNQ)，或 SRTSEQ(*LANGIDSHR)和 LANGID(ENU)，或通过使用 SET OPTION 语句实现的。

3.7.2 ORDER BY

下面的 SQL 语句使结果表按 JOB 列值进行分类：

SELECT * FROM STAFF ORDER BY JOB

表 3-2 是使用*HEX 分类顺序的结果表，这些行按 JOB 列的 EBCDIC 值进行分类。在这里，所有小写字母排在大写字母之前。

表 3-2 使用*HEX 分类顺序的"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	17808.30	650.25

表 3-3 给出如何用唯一权分类排序。在用于 JOB 列里的值排序之后，再对行进行分类。

注意，分类后，小写字母排在相同大写字母前面，‘mgr’, ‘Mgr’和 ‘MGR’彼此相邻。

表 3-3 对 ENU 语言标识便用唯一权分类顺序的"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

表 3-4 给出如何用共享权分类排序。用 JOB 列值分类之后，再对行进行分类。对这种分类比较，小写字母和相应的大写字母同样看待。在表 3-4 里，所有‘MGR’, ‘mgr’和 ‘Mgr’都混合在一起。

表 3-4 对 ENU 语言标识用共享权分类顺序的"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

3.7.3 记录选择

下面的 SQL 语句选择 JOB 列中值为‘MGR’的记录：

SELECT * FROM STAFF WHERE JOB=‘MGR’

表 3-5 给出如何用*HEX 分类顺序选择记录。在表 3-5 里，与记录选择准则 JOB 列匹配的行，准确地按指定的选择语句被选择，仅选择大写字母‘MGR’。

表 3-5 用*HEX 分类顺序的"SELECT * FROM STAFF WHERE JOB='MGR'

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
30	Merenghi	38	MGR	5	17506.75	0

表 3-6 给出如何用唯一权分类排序选择记录。在表 3-6 里，区分大小写字母。小写字母‘mgr’不同于大写字母‘MGR’。因此，小写字母‘mgr’不入选。

表 3-6 对 ENU 语言标识使用唯一权分类顺序的"SELECT * FROM STAFF WHERE JOB = 'MGR' "

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
30	Merenghi	38	MGR	5	17506.75	0

表 3-7 给出如何用共享权分类排序选择记录。在表 3-7 里，与记录选择准则 JOB 列中匹配的行，用大小写字母同样对待来选择。所以，‘mgr’ ,‘Mgr’ 和‘MGR’都入选。

表 3-7 对 ENU 语言标识使用共享权的"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	20659.80	0
50	Hanes	15	Mgr	10	20659.80	0
100	Plotz	42	Mgr	6	18352.80	0

3.7.4 分类顺序和视图

由 CREATE VIEW 语句运行时建立有效的分类顺序视图。在 FROM 子句里引用视图时，使用在 CREATE VIEW 子选择里的字符比较的分类顺序。此时，从视图子选择中产生中间结果表。在查询运行时有效的分类顺序提供给查询规定的全部字符和 UCS-2 图形比较(包括字符或 UCS-2 图形隐式转换的比较)。

下面的 SQL 语句和表给出视图和分类顺序如何工作。下例里的视图 V1，是用共享权分类顺序 SRTSEQ(*LANGIDSHR)和 LANGID(ENU)生成的。CREATE VIEW 语句格式如下：

```
CREATE  VIEW  V1  AS  SELECT  *
FROM  STAFF
WHERE  JOB  =  ‘MGR’  AND  ID < 100
```

表 3-8 给出视图的结果表

表 3-8 “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

表 3-8 里，用视图 V1 运行查询表。以下显示的是用分类顺序 SRTSEQ(*LANGIDUNQ)和 LANGID(ENU)。

表 3-9 对语言标识 ENU 用唯一权分类顺序的"SELECT * FROM V1 WHERE JOB = 'MGR'"						
ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
30	Merenghi	38	MGR	5	17506.75	0

3.7.5 分类顺序和 CREATE INDEX 语句

用 CREATE INDEX 语句生成有效的分类顺序索引。每次对表进行插入时都有入口要加到定义索引的表中。索引入口包括字符键和 UCS-2 图形键列的权值。系统根据索引的分类顺序转换键值得到权值。

用分类顺序和索引进行选择时，字符或 UCS-2 图形键不需要在比较前转换，这点改善了

查询的性能。

3.7.6 分类顺序和约束

唯一约束由索引实现，如果加唯一约束的表定义了分类顺序，那么用同一分类顺序生成索引。

如果定义一个引用约束，那么双亲表之间的分类顺序必须匹配。详细内容请看 AS/400 DB2 数据库程序设计一书。

定义检查约束时使用的分类顺序与系统用来确认 INSERT 或UPDATE 约束的分类顺序是相同的。

第四章　使用游标

当 SQL 运行选择语句时，结果行组成结果表。游标提供存取结果表的方法，它用在 SQL 程序中管理结果表的位置。SQL 使用游标处理结果表中各行，让程序使用它们。程序可能有几个游标，但每一个都必须有唯一的名字。

与使用游标有关的语句包括下面几个：

DECLARE　CURSOR 语句，定义并给游标命名，规定在嵌套的选择语句中检索的行。

OPEN 和 CLOSE 语句，在程序里用来打开和关闭游标，游标必须在检索行前打开。

FETCH 语句，从游标结果表或在游标另一行的位置处检索行。

UPDATE…WHERE　CURRENT　OF 语句，用来修改游标的当前行。

DELETE…WHERE　CURRENT　OF 语句，用来删除游标的当前行。

4.1　游标类型

SQL 支持连续或滚动游标，游标类型确定游标使用的定位方式。

4.1.1　连续游标

连续游标是一个没有规定 SCROLL 键字的游标。

对于连续游标，在游标每次打开时仅能取一次结果表的行。当游标打开时，它定位在结果表的第一行前，当执行 FETCH 时，游标移到结果表的下一行，那行即为当前行。如果设定主变量(使用 FETCH 语句里的 INTO 子句)，SQL 把当前行的内容移到程序的主变量里。

执行每个 FETCH 语句时重复这个过程，直到数据结束(SQLCODE=100)。当达到数据结束时，关闭游标。在到达数据结束后就不能存取结果表里的任何行。要再次使用游标，首先必须关闭游标，然后重新执行 OPEN 语句，不可以返回去。

4.1.2　滚动游标

对于滚动游标，结果表中的行可多次取出。用 FETCH 语句指定的位置选项为基础，游标在结果表中移动。当打开游标时，游标定位在结果表第一行前，当执行 FETCH 语句时，游标定位在由位置选项规定的结果表行上，该行即为当前行。如果规定主变量(用 FETCH 语句中的 INTO 子句)，SQL 把当前行的内容移入程序的主变量里。主变量不能规定 BEFORE 和 AFTER 位置选项。

每次执行 FETCH 语句时，重复这个顺序。当数据结束或数据开始条件出现时，不需要关闭游标，位置选项使程序继续从表中取出行。

当执行 FETCH 语句时，下面的滚动选项用来定位游标，这些位置与结果表里当前游标的位置是相关的。

NEXT	游标定位在下一行。如果未指定位置，它为缺省值。
PRIOR	游标定位在前一行。

FIRST	游标定位在第一行。
LAST	游标定位在最后一行。
BEFORE	游标定位在第一行之前。
AFTER	游标定位在最后行之后。
CURRENT	不改变游标位置。
RELATIVE n	计算与游标当前位置相关的主变量或整数 n 位置。例如如果 n= -1，光标定位在结果表的前一行。如果 n=+3，游标定位在当前行之后的第三行。

4.2 使用游标举例

假定程序检查部门 D11 的人员数据，下面的例子给出在程序里定义和使用连续和滚动游标的 SQL 语句，这些游标能从 CORPDATA.EMPLOYEE 表中获取关于部门的信息。

对于连续游标，程序处理表中的所有行，修改部门 D11 全部人员的工作，还从其它部门删除这些人员的记录。

表 4-1　游标例子
连续游标 SQL 语句　　该部分的描述
在 4.2.1 里“第一步：定义游标”
在 4.2.2 里“第二步：打开游标”
在 4.2.3 里“第三步：指定到达数据结束时做什么。”
在 4.2.4 里“第四步：使用游标检索行”。
在 4.2.5 里“第五步 a：修改当前行”。
在 4.2.6 里“第五步 b：删除当前行”。
分支返回取出并处理下一行。
在 4.2.7 里“第六步：关闭游标”。

对于滚动游标例子，程序使用 RELATIVE 位置选项获取部门 D11 相应人员的工资。

表 4-2　滚动游标例子
滚动游标 SQL 语句　　该部分的描述
在 4.2.1 中“第一步：定义游标”
在 4.2.2 中“第二步：打开游标”
在 4.2.3 中“第三步：指定到达数据结束时做什么。”
…初始化程序总和　　在 4.2.4 中“第四步：使用游标检索行”。 工资变量 …把当前工资加到程序总和工资中。 …分支返回到取出并处理下一行。
…计算平均工资
在 4.2.7 中“第六步：关闭游标”。

4.2.1 第一步：定义游标

要定义存取结果表的游标，使用 DECLARE　CURSOR 语句。

DECLARE CURSOR 语句命名游标并规定选择语句。选择语句定义一些行，它们组成结果表。对于连续游标，语句格式为：（FOR UPDATE 子句是可选的）

```
EXEC SQL
 DECLARE 游标名 CURSOR FOR
 SELECT 列 - 1, 列 - 2 ,...
   FROM 表名 , ...
   FOR UPDATE OF 列 - 2 ,...
END-EXEC.
```

对于滚动游标，语句格式为：（WHERE 子句是可选的）

```
EXEC SQL
 DECLARE 游标名 DYNAMIC SCROLL CURSOR FOR
 SELECT 列 - 1, 列 - 2 ,...
   FROM 表名 ,...
   WHERE 列 - 1 = 表达式expression ...
END-EXEC.
```

这里给出的选择语句是相当简单的，在 DECLARE CURSOR 语句中，可以写几个其它类型的选择语句给连续游标和滚动游标。

如果打算修改规定表的任一列或所有列（表由 FROM 子句给名），用 FOR UPDATE OF 子句，它命名要修改的每一列。如果未指定列名，那么要指定 ORDER BY 子句或 FOR READ ONLY 子句，如果试图修改，那么返回负的 SQLCODE。如果没有指定 FOR UPDATE OF 子句、FOR READ ONLY 子句或 ORDER BY 子句，且结果表不是只读，那么可以修改指定表的任何列。

可以修改指定表中的列，虽然它可不是结果表的一部分。在这种情况下，不必用 SELECT 语句命名列。当游标检索到包括要修改列值的行时（使用 FETCH），可用 UPDATE…WHERE CURRENT OF 来修改行。

例如，假设结果表的每一行都包括来自 CORPDATA.EMPLOYEE 表的 EMPNO，LASTNAME 和 WORKDEPT 列，如果想修改 JOB 列（在 CORPDATA.EMPLOYEE 表的一列），DECLARE CURSOR 语句要包括 FOR UPDATE OF JOB…，这时 JOB 并不在 SELECT 语句中。

如果下面任何一个为真，那么结果表和游标是只读的：

第一个 FROM 子句标识多个表或视图。

第一个 FROM 子句标识一个只读视图。

第一个 SELECT 子句规定键字 DISTINCT。

外部子选择包括 GROUP BY 子句。

外部子选择包括 HAVING 子句。

第一个 SELECT 子句包括列函数。

选择语句包括子查询，这样外部子选择和子查询的基本目标是同一个表。

选择语句包括 UNION 或 UNION ALL 操作符。

选择语句包括 ORDER BY 子句，且未规定 FOR UPDATE OF 子句和 DYNAMIC SCROLL。

选择语句包括 FOR READ ONLY 子句。

规定没有 DYNAMIC 的键字 SCROLL。

4.2.2 第二步：打开游标

若着手处理结果表的行，使用 OPEN 语句。当程序使用这个语句时，SQL 处理在 DECLARE CURSOR 语句中的选择语句来标识一些行，叫做结果表(1)，SQL 使用在选择语句中规定的主变量的当前值。OPEN 语句格式为：

```
EXEC SQL
    OPEN 游标名
END—EXEC.
```

(1)根据满足查询条件情况，结果表中可以没有行、一行或多行。

4.2.3 第三步：到达数据结束时做什么

当到达结果表末尾时，检查 SQLCODE 字段的值是否为 100 或检测 SQLSTATE 字段是否为‘02000’（也就是，数据结束）。这个条件发生在 FETCH 语句已检索到结果表的最后一行且程序执行后续的 FETCH 语句。例如：

```
...
IF SQLCODE =100 GO TO DATA-NOT-FOUND.

or

IF SQLSTATE =’02000’ GO TO DATA-NOT-FOUND.
```

处理这种情况的另一方法为用 WHENEVER 语句。使用 WHENEVER NOT FOUND 能转到程序另一部分，此处使用 CLOSE 语句。WHENEVER 语句格式为：

```
EXEC SQL
    WHENEVER  NOT  FOUND  GO  TO 符号地址
END—EXEC
```

无论何时使用游标取出一行，程序要测试数据结束条件，当条件发生时，程序要准备处理这种情况。

当使用连续游标且到达数据结束时，每一后续的 FETCH 语句返回数据结束条件，不能把游标定位在处理过的行上。CLOSE 语句是在游标上完成的唯一操作。

当使用滚动游标且到达数据结束时，结果表仍处理更多的数据。可用位置选项组合把游标定位在结果表的任何地方。当到达数据结束时，不必关闭游标。

4.2.4 第四步：用游标检索行

为了把选择行的内容移入到程序的主变量里，使用 FETCH 语句。在 DECLARE CURSOR 语句中的 SELECT 语句给出程序需要的列值的行，只有用 FETCH 语句，SQL 才取出送给应用程序的数据。

当程序处理 FETCH 语句时，SQL 把当前游标位置作为结果表中定位行的起点。这个操作把这行变为当前行。如果指定 INTO 子句，SQL 把当前行的内容送到程序的主变量里，每次

执行 FETCH 语句时，重复这个过程。

SQL 保持当前行的位置，直到执行游标的下一个 FETCH 语句。UPDATE 语句不改变结果表里当前行的位置，而 DELETE 语句可以。

连续游标的 FETCH 语句格式为：

```
EXEC SQL
 FETCH 游标名
   INTO :主变量 – 1 [, :主变量 - 2] ...
END-EXEC.
```

滚动游标 FETCH 语句格式为：

```
EXEC SQL
 FETCH RELATIVE 整数
   FROM 游标名
   INTO :主变量 – 1 [, :主变量 - 2] ...
END-EXEC.
```

4.2.5 第五 a 步：更新当前行

当程序把游标定位在一行时，可以使用有 WHERE CURRENT OF 子句的 UPDATE 语句更新数据。WHERE CURRENT OF 子句规定一个指向需要修改行的游标。UPDATE…WHERE CURRENT OF 语句格式如下：

```
EXEC SQL
 UPDATE 表名
   SET 列 - 1 = 值 [, 列 - 2 = 值] ...
   WHERE CURRENT OF 游标名
END-EXEC.
```

当与游标一起使用时，UPDATE 语句能：

仅能更新一行——当前行。

标识指向更新行的游标。

如果也规定了 ORDER BY 子句，要求用 DECLARE CURSOR 语句中的 UPDATE OF 子句来命名从前更新的列。

当修改一列之后，游标位置保留在该行（也就是说，游标当前行不改变），直到对下一行执行 FETCH 语句为止。

4.2.6 第五 b 步：删除当前行

当程序检索当前行时，可用 DELETE 语句删除该行。为此，执行用游标的 DELETE 语句，WHERE CURRENT OF 子句指定一个指向删除行的游标。DELETE…WHERE CURRENT OF 语句格式为：

```
EXEC SQL
 DELETE FROM 表名
   WHERE CURRENT OF 游标名
END-EXEC.
```

当与游标一起使用时，DELETE 语句能：

仅删除一行——当前行。

使用 WHERE CURRENT OF 子句标识指向删除行的游标。

删除一行后，不能用这个光标更新或删除另一行，除非执行 FETCH 语句来定位游标。

3.1.3 中的“DELETE 语句”给出怎样用 DELETET 语句删除满足条件的所有行。要获取行的复本，检查它，然后删除它时，可以使用 FETCH 和 DELEETE…WHERE CURRENT OF 语句。

4.2.7 第六步：关闭游标

如果使用连续游标处理结果表的各行，且想要再次使用游标，先用 CLOSE 语句关闭游标，然后再重新打开使用。

```
EXEC SQL
     CLOSE 游标名
END—EXEC
```

如果处理好结果表的各行，且不想再使用游标，那么可以让系统关闭游标。当有下面情况下，系统自动关闭游标：

执行没有 HOLD 语句的 COMMIT，且没用 WITH HOLD 子句说明游标，发出

不包括 HOLD 语句的 ROLLBACK。

作业结束。

活动组结束，且在预编译时指定 CLOSQLCSR(*ENDACTGRP)。

调用堆栈的第一个 SQL 程序结束，且当程序预编译时，没有指定

CLOSQLCSR(*ENDJOB)或 CLOSQLCSR(*ENDACTGRP)。

用 DISCONNECT 语句终止与应用服务程序的连接。

释放与应用服务程序的连接，成功的执行了 COMMIT。

发生*RUN CONNECT。

由于打开游标一直保持锁住引用表或视图，那么当不需要游标的时候，必须立即明显的关闭打开的游标。

4.3 使用多行 FETCH 语句

使用多行 FETCH 语句能从带有单一 FETCH 的表或视图中检索多行。通过 FETCH 语句请求的行数，程序控制行组成的块。(OVRDBF 无效)。一个 FETCH 调用请求的最大行数是 32767。一旦检索数据，游标定位在检索的最后一行。

有两种方式定义存放取出行的存储位置：主结构数组，或有关描述行存储区，两种方式都可以放在 SQL 预编译支持的主语言中，(除 REXX 内的主结构数组之外)，详细内容请看第十章和第十五章。两种形式的多行 FETCH 语句都允许应用程序给指示器数组编码，对每个有空属性的主变量，指示器数组包含一个指示器。

多行 FETCH 语句可用连续和滚动游标。对多行 FETCH，定义、打开和关闭游标的操作保持相同，不同的仅是要规定检索的行数和行存放的存储位置。

在每个多行FETCH语句执行之后，信息通过SQLCA返回给程序。除了SQLCODE和SQLSTATE字段外，SQLERRD 还提供以下信息：

SQLERRD3 包括由多行 FETCH 语句检索的行数。如果 SQLERRD3 小于需要的行数，则出现错误或数据结束条件。

SQLERRD4 包括每个检索行的长度。

SQLERRD5 包括从表里取出的最后一行的标志。当游标不是立即反映更新时，它也能用来检查取出表里数据的结束条件。有立即反映更新属性的游标会继续取数，直到接收到 SQLCODE+100，检测到数据结束条件为止。

4.3.1 使用主结构数组的多行 FETCH

为了使用主结构数组的多行 FETCH，应用程序必须定义 SQL 能使用的主结构数组。定义主结构数组，每种语言都有自已的约定和规则，可用变量说明或用编译命令指令检索外部文件说明来定义主结构数组。（例如，COBOL COPY 指令）。

主结构数组由结构体数组组成，每个结构对应结果表的一行，数组里的第一个结构对应第一行，第二个结构对应第二行，等等。SQL 根据主结构数组说明来决定主结构数组里的元素的属性。为了最好发挥其性能，构成主结构数组的项目属性要与检索列的属性相匹配。

考虑下面的例子：

```
        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.
```

在这个例子里，为 CORPDATA.EMPLOYEE 表定义游标来选择 WORKDEPT 列等于‘D11’的所有行，结果表包括 8 行，当使用多行 FETCH 语句时，DECLARE CURSOR 和 OPEN 语句没有任何特定的语法。返回一行的另一 FETCH 语句的同一游标，可在程序的其它地方编码。多行 FETCH 语句用来检索结果表里的所有行，接着的 FETCH，把游标定位在检索的最后一行。

主结构数组 DEPT 和相关的指示器数组 IND—ARRAY 在应用程序里定义，两个数组都有十个元素，指示器数组对结果表的每列都有一项。

主结构数组 DEPT 的元素类型属性和长度要与检索列相匹配。

在多行 FETCH 语句成功完成时，主结构数组包括所有 8 行数据。由于无空值返回，指示器数组 IND—ARRAY 对每行每列里包含零。

返回给应用程序的 SQLCA 包括以下信息：

SQLCODE 包含 0

SQLSTATE 包含‘00000’

SQLERRD3 包含 8，这是取出的行数

SQLERRD4 包含 34，这是每行长度

SQLERRD5 包含+100，它指出结果表的最后行在块中

关于 SQLCA 的说明，请看 AS/400 DB2 SQL 参考一书的附录 B。

4.3.2 使用行存储区的多行 FETCH

应用程序要使用行存储区的多行 FETCH，首先必须定义一个行存储区和与之相关的描述区。行存储区是在应用程序里定义的一个主变量。行存储区包括多行 FETCH 结果，行存储区可以是有足够多字节容纳多行 FETCH 需要的所有行的字符变量。

通过使用多行 FETCH 的行存储区格式的相关描述符，定义 SQLDA，对于每一返回列，SQLDA 包括 SQLTYPE 和 SQLLEN。描述符中提供的信息决定从数据库到行存储区的数据映象。为了得到最好性能，描述符中的属性信息与检索列的属性要匹配。

关于 SQLDA 的说明，请看 AS/400 DB2 SQL 参考一书的附录 C。
考虑下面的 PL/1 例子：

```
      *....+....1....+....2....+....3....+....4....+....5....+....6....+....7...*
     EXEC SQL INCLUDE SQLCA;
     EXEC SQL INCLUDE SQLDA;

...

    DCL DEPTPTR PTR;
    DCL 1 DEPT(10) 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';

...

    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 = 10;         /*REQUESTS 10 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:
```

图 4-1　使用行存储区的多行 FETCH 的例子

在这个例子里，为 CORPDATA.EMPLOYEE 表定义游标，来选择 WORKDEPT 列等于‘D11’的所有行。样本表 EMPLOYEE 在原文的附录 A 中，其中结果表包括 8 行。当使用多行 FETCH 语

句的 DECLARE CURSOR 和 OPEN 语句时，它们设有特别的语法。返回一行的另一 FETCH 语句的游标，可以在程序的别处编码。多行 FETCH 语句用来检索结果表的所有行。接下来的 FETCH，把游标位置保留在块中的第 8 个记录上。

行区 ROWAREA 定义为字符数组，从结果表来的数据放到这个主变量里。在这个例子里，分配给指针变量的是 ROWAREA 的地址。要检查返回行的每一项且与基本结构 DEPT 一起使用。

描述符的项目属性（类型和长度）与检索列的属性要相匹配。这时，不提供指示器区。

当 FETCH 语句完成后，ROWAREA 包括 8 行，返回给应用程序的 SQLCA 包括以下信息：

SQLCODE 包含 0

SQLSTATE 包含‘00000’

SQLERRD3 包含 8，这是返回的行数

SQLERRD4 包含 34，这是取出的行的长度

SQLERRD5 包含+100，指出取出结果表的最后一行

这个例子里，应用程序利用了 SQLERRD5 包括达到文件末的指示，结果是，应用程序不需要再调用 SQL 来检索更多行。如果游标具有立即插入属性，将调用 SQL 添加任何记录，有立即插入属性的光标，是由落实控制级不是*RR 来实现的。

4.4 工作单元和打开游标

当程序完成一个工作单元，它落实或返回所做的修改，除非在 COMMIT 和 ROLLBACK 语句中指定了 HOLD,所有打开的游标都由 SQL 自动关闭。用 WITH HOLD 子句说明的游标在 COMMIT 时不自动关闭，它们在 ROLLBACK 时自动关闭。(在 DECLARE CURSQR 语句中指定的 WITH HOLD 子句被忽略。)

如果想在落实或返回之后从当前游标位置继续处理，必须指定 COMMIT HOLD 或 ROLLBACK HOLD。当规定 HOLD 时，任何打开的游标保持打开且保持它们的游标位置，因此处理可以重新执行。在一个 COMMIT 语句中，游标位置不变。在一个 ROLLBACK 语句里，游位置重存在前一个工作单元的最后一行之后，所有记录锁仍是释放状态。

发出一个没有指定 HOLED 的 COMMIT 或 ROLLBACK 语句,所有锁被释放,所有游标被关闭。可以重新打开游标，但从结果表的第一行开始执行。

注：在 CRTSQLxxx 命令中指定 ALWBLK(*ALLREAD)参数可以将游标位置的重存入变为只读游标。详细内容请看第八章。

第五章　高级编码技术

这一章介绍更先进的 SQL 编码技术。这一章包括：

插入多行
禁止重复行
查找
连接
使用 UNION
使用子查询
使用视图
使用索引
使用系统目录

5.1　高级插入技术

下两节讲述向表中插入行的两个技术，第一节介绍用一个选择语句一次向表中插入多行，第二节介绍用一个块插入语句往主结构数组中插入多行，第二种方法可以用在除 REXX 外的所有语言中。

5.1.1　用选择语句向表中插入行

可以用 INSERT 语句中的选择语句从规定的表或视图中选择零行、一行或多行插入到另一张表中。从中选择行的表和插入的表可以是同一张表，如果它们是同一张表，SQL 先生成一个暂时的结果表来放选择的行，然后从临时表插入到目标表中。

用这种插入语句可把数据移到生成的总计数据中。例如：假设要在一个表中显示每个雇员对一个项目的提交时间，可以生成一个叫 EMP TIME 的表，这个表有 EMPNUMBER、PROJNUMBER、STARTDATE、ENDDATE 和 TTIME 列，然后用下面的插入语句来填充这个表：

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

在插入语句中嵌套的选择语句和使用的检索数据的选择语句没有什么不同，除了 FOR READ ONLY，FOR UPDATE OF 或 OPTIMIZE 子句外，可以用所有键字、列函数或用在检索数据上的技术。SQL 把所有满足条件的行插入到指定的表中。从一个表插入到另一张表中的行并不影响源表和目标表中的原有的行。

5.1.1.1　多行插入的说明

向表中插入多行时，应该考虑下面几点：

在插入语句中隐式或显示给出的列数必须与在选择语句中给出的列数相符。

所选择的列的数据必须与在选择语句中用 INSERT 插入的列的数据兼容。

嵌套在 INSERT 中的选择语句没有行返回时，返回一个 SQLCODE 100 提醒你没有行被插入，如果成功地插入了行，SQLCA 的 SQLERRD(3)有显示实际插入行数的整数。

如果 SQL 在运行一个插入语句时发现了一个错误，则停止操作，如果指定了 COMMIT(*CHG),COMMIT(*CS),COMMIT(*ALL)或者 COMMIT(*RR)，就没有什么插入到表中，并返回一个负的 SQLCODE，如果指定 COMMIT(*NONE)，在出错前插入到表中的任何行都留在表中。

在插入语句中可以用选择语句连接两个或更多的表，用这种方式装入，表可以用 UPDATA、DELETE、INSERT 语句进行操作，因为行是以物理方式存在表中的。

5.1.2 使用块插入语句

一个块 INSERT 可以用一个语句向表中插入多行,除 REXX 外所有的语言都支持块插入语句。插入到表中的数据必须是在主结构数组中，如果在块插入中用到指示器变量，它们也必须在主结构数组中。各种语言的主结构数组的信息，请看说明这种语言的章节。

例如，要往 CORPDATA.EMPLOYEE 表中增加 10 个雇员：

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

DSTRUCT 是在程序中说明的有五个元素的主结构数组，这五个元素分别为 EMPNO、FIRSTNME、MIDINIT、LASTNAME 和 WORKDEPT、DSTRUCT，它们至少有 10 维以满足插入 10 行这个操作。ISTRUCT 是在程序中说明的主结构数组，ISTRUCT 是为指示器所用，至少要有 10 维小整数字段。

非分布式 SQL 应用支持块插入语句,对于分布式应用,应用服务和应用请求都是 AS/400 系统。

5.2 避免重复行

当 SQL 处理一个选择语句时，根据满足选择语句的检索条件，在结果表中有些行要受到限制，在结果表中的一些行可能是重复的，可以通过用 DSITINCT 键字后跟列名来规定不需要重复：

```
SELECT  DISTINCT  JOB，SEX
…
```

DISTINCT 表示只选择一行，如果选择的行与结果表中的另一行重复，这个重复的行被忽略，(即它不放进结果表中)。例如，假设想要一个雇员工作码的清单，但不需知道哪个雇员有哪个工作码，因为在一个部门的几个雇员有相同的工作码，可以用 DISTINCT 来确保在结果表中只有单一的值。

下面的例子可完成这项工作：

```
  DECLARE XMP2 CURSOR FOR
    SELECT DISTINCT JOB
      FROM CORPDATA.EMPLOYEE
      WHERE WORKDEPT = :JOB-DEPT
...
  FETCH XMP2
    INTO :JOB
```

结果是两行（在这个例子中，JOB—DEPT 设置成 D11）。

PIC13

如果在选择子句中不包含 DISTINCT，在结果中可能有重复的行，因为 SQL 对每一个满足查找条件的行都取 JOB 列的值，对于 DISTINCT 来说，空值被看成是重复行。

如果在选择子句中包含了 DISTINCT，也有一个共享权分类顺序，返回的值就更少，这个分类顺序表示相同字符有相同权，如果‘MGR’，‘Mgr’，‘mgr’都在同一表中，这些值中只有一个被返回。

5.3 执行复杂的检索条件

下面介绍可以用检索条件的高级技术。

5.3.1 检索条件中的键字

一个检索条件可以包含键字 BETWEEN、IN、IS NULL 和 LIKE。

注：在下面的例子中，为了简单而使用常量，实际上可以很容易用主变量来代替，记住每一个主变量前要用冒号。

对于字符和 UCS-2 图形列谓词，分类顺序是在计算 BETWEEN、IN、EXISTS 和 LIKE 子句的谓词后，被应用在操作数中。详细内容请看 3.7。

BETWEEN…AND…被用来指定一个检索条件，这个条件是满足低于或在两个值之间的任何值。例如：检索在 1987 年雇用的所有雇员，可以用：

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

键字 BETWEEN 是包含首尾的，更复杂但更明显的检索条件产生同样的结果：

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

IN 是指对那些包括列出值的表达式范围中的行感兴趣。例如，检索在部门 A00、C01 和 E21 的所有雇员名，可以指定：

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

LIKE 指的是对那些列值与提供值相似的行感兴趣。当用 LIKE 时，SQL 检索与指定的字符相似的字符串，相似的程度是由包括在检索条件串中的两个字符来决定的：

_ 一个下划线字符代表任何一个单个字符

% 一个百分号代表一个未知的包含零个或多个字符的串。如果检索以百分号开头的串，那么 SQL 允许零个或多个字符放在列中的匹配值前。否则，检索串必须从列的第一个位置开始。

注：如果是在 MIXED 数据上操作，下面是一些区别：一个 SBCS 下划线字符指的是一个 SBCS 字符，对于百分号没有这样的限制，就是说，一个百分号指的是 SBCS 或 DBCS 字符的任何数量。详细内容请看 AS/400 SQL 参考一书。

在不知道或不关心列值的所有字符时用下划线或百分号。例如，检索住在 Minneapolis 的雇员，可以指定：

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

这时，你不知道 Minneapolis 是街道地址的一部分还是城市名的一部分，SQL 返回在地址列中有字符串 Minneapolis 的行。

另一个例子，显示所有以‘SAN’开头的城市，可以指定：

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

如果要有下划线或百分号的字符串，用 ESCAPE 子句指定一个换码字符。例如，找所有名字里面有百分号的公司，可以指定：

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

第一个和最后一个百分号可以像通常一样解释，而‘@%’这个组合作为实际的百分号。

5.3.1.1 LIKE 的特殊考虑

在检索模式中当主变量被用在串常量的位置上时，应该考虑使用可变长的主变量，这允许你：

—预先把用过的串常量分配给主变量不做任何修改。

—用同样的选择准则和结果，就像使用串常量一样。

在检索模式中当定长主变量用在串常量的位置时，应该确保主变量规定的值与先前串常量用的模式相符。在没有指定值的主变量中的所有字符都初始化为一个空格。

例如，如果检索一个模式为‘ABC%’的串，那么，可以返回下面这样的值：

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

例如，如果用模式为‘AB%’检索在定长为 10 的主变量中的值，可以返回下面这些值：

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

注：所有这些返回值都以 ABC 开头并以至少 6 个空格结束，这是因为在主变量中的后 6 个字符没分配值，所以用空格。

如果要检索一个固定长的主变量，它的后七个字符可以是任何字符，可以用‘ABC%%%%%%%’检索，可以返回下面这些值：

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

5.3.1.2 WHERE 子句中的多检索条件

已知道怎样用一个检索条件限定一个要求，可以用包括多个谓词的检索条件来进一步限定要求。规定的检索条件可以包含任何比较运算符或键字 BETWEEN、IN、LIKE、IS NULL 和 IS NOT NULL。

可以用 AND 或 OR 连接两个谓词，另外，可以用键字 NOT 来指定检索条件是规定的检索条件的相反值。一个 WHERE 子句可以有很多谓词。

AND 是指对于一个指定的行，这个行必须满足检索条件中的两个谓词。例如，检索在部门 D21 并在 1987 年 12 月 31 日以后雇佣的雇员，可以规定：

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

OR 指的是，对于一个指定的行，可以满足检索条件中的某一个或两个。例如，找出在部门 C01 或部门 D11 的雇员，可以规定(1)：

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

NOT 指的是对于指定的行，这行必须不满足查找条件设置的准则或在 NOT 后的谓词，例如，查找在部门 E11 的工作代码不为 analyst 的所有雇员，可以规定：

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

当 SQL 计算一个包含这些连接词的检索条件时，它以一种规范的顺序来完成。 SQL 首先计算 NOT 子语句，然后是 AND 子语句，最后是 OR 子句。

可以用圆括号来改变计算顺序，先计算包含在括号中的检索条件。例如，选择所有在部门 E11 及 E21 中教育程度高于 12 的雇员，可以规定：

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

圆括号也决定查找条件，在此例子中，想要所有满足下面条件的行：
WORKDEPT 的值为 E11 或 E21，以及
EDLEVEL 的值大于 12。
如果没用圆括号：

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

结果是不同的，所选的行满足下面的条件：

```
   WORKDEPT = E11 and EDLEVEL > 12, or

  WORKDEPT = E21
```

(1)也可以用 IN 来指定这个要求：WHERE WORKDEPT IN ('C01', 'D11')。

5.4 从多个表中连接数据

有时你想要看的信息不在一个表中，为了形成结果表中的一行，可以从一个表中检索一些列又从另一张表中检索一些列。可以用选择语句来检索并把从多个表中选择的列值连接成一行。

AS/400 DB2 支持四种不同类型的连接：内部连接、左外部连接、异常连接和交叉连接。

内部连接只返回每个表中符合连接列值的行，表中不匹配的行在结果表中不出现。

左外部连接返回在第一个表（左边的表）中的所有行及第二个表中匹配的行，任何在第二个表中没有匹配的行，则从第二个表的所有列返回空值。

异常连接只返回左边的表中与右边的表不匹配的行，来自右边表中的结果表中列有空值。

交叉连接是被连接（一个笛卡儿乘积）表中行的每一种组合都在结果表中返回一行。

5.4.1 内部连接

用内部连接，一个表中一行的列值与另一表（或同一表）中另一行的列值组合形成一个数据行。SQL 检查连接规定的每个表，来检索满足查找条件的所有行。有两种方法指定一个内部连接：用 JOIN 句法和用 WHERE 子句。

假设想要检索与某个项目有关的所有雇员的雇员号、雇员名和项目号，即要从 CORPDATA.EMPLOYEE 表中查找 EMPNO 和 LASTNAME 列，从 CORPDATA.PROJECT 表中找 PROJNO 列。仅考虑雇员名以 S 打头的人员，为了找到这个信息，需要连接两个表。

5.4.1.1 用 JOIN 句法进行内部连接

用内部连接句法，想要连接的两个表都列在 FROM 子句中，加上连接所需的条件，连接条件在 ON 键字后指定，用来决定两个表怎样互相比较并产生连接结果。条件可以是比较操作符，它不需要相等的操作，可以在用 AND 键字分开的 ON 子句中指定多个连接条件，对实际连接没有影响的其它条件都在 WHERE 子句中规定。

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

在这个例子中，用 EMPNO 和 RESPEMP 列实现两个表的连接。既然只要返回至少以 S 打头的雇员名，这个附加的条件放在 WHERE 子句中。

这个查询返回下列结果：

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

5.4.1.2 用 WHERE 子句的内部连接

用 WHERE 子句实现同样的连接，是用连接条件和在 WHERE 子句中附加的选择条件写出的，被连接的表在 FROM 子句中列出，之间用逗号分开。

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

这个查询与前面的例子返回同样的结果。

5.4.1.3 说明

尽管用 JOIN 句法和 WHERE 子句进行的内部连接返回的结果是相同的，但这两种方法的性能是不同的，用 JOIN 语法将影响查询优化，当做连接时各个表以列出的顺序排列。当用 WHERE 子句时，查询优化将评估要连接的表，如果用另一种顺序执行得更好，则重新排列。

5.4.2 左外部连接

一个左外部连接将返回所有内部连接返回的行，加上第一张表中与第二张表中不匹配的行。

假设要查找所有雇员及当前他们负责的项目，也想知道这些雇员是不是在负责项目，下面的查询将返回所有雇员名大于 S 的列表，以及分配给他们的项目号：

```
SELECT EMPNO, LASTNAME, PROJNO
```

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

这个查询结果包含一些没有项目号的雇员，他们在查询中列出，但他们的项目号返回的是空值。

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	-

5.4.2.1 说明

如果多级连接条件用在左外部连接或者异常连接 ON 条件中使用多个连接条件，ON 条件的所有比较运算必须是相等的运算符；如果用的是不等的比较，则只可规定一个条件。如果把它们移到 WHERE 子句中，则可规定附加的条件。因此，它们将被用作附加的选择条件，而不能用作完成连接。

用 RRN 标量函数可以在左外部连接中对于在右边表的一列返回一个相关的记录号或在异常连接中对于不匹配的行返回一个零值。

5.4.3 异常连接

异常连接仅返回第一个表与第二个表不匹配的记录，用以前一样的表，返回的是不负责项目的雇员：

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

这个连接返回的输出为：

EMPNO	LASTNAME	PROJNO
000170	YOSHIMURA	-
000180	SCOUTTEN	-
000190	WALKER	-

000280	SCHEIDER	-
000300	SMITH	-
000310	SETRIGHT	-

一个异常连接也可以用 NOT EXISTS 谓词作为子查询，前面的查询可以用下面的方法重新写：

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

用这个查询的唯一区别是不从 PROJECT 表中返回值。

5.4.4 交叉连接

交叉连接（或叫笛卡儿乘积连接）将返回一个结果表，这个表是第一个表的每一行与第二个表的每一行相连而得。结果表中的行数是每个表中行的乘积。如果涉及到的表很大，那么连接将花费很长时间。

一个交叉连接可以用两种方法来规定，用 JOIN 句法或在 FROM 子句中列出各表，表间用逗号分离，而不用 WHERE 子句去提供连接准则。

假设有下面的表：

表 5-1　表 A

T1COL1	T1COL2
A1	A A1
A2	A A2
A3	A A3

表 5-2　表 B

T2COL1	T2COL2
B1	BB1
B2	BB2

下面两个选择语句产生相同的结果：

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

这两个选择语句的结果表如下：

T1COL1	T1COL2	T2COL1	T2COL2
A1	AA1	B1	BB1
A1	AA1	B2	BB2
A2	AA2	B1	BB1
A2	AA2	B2	BB2

A3	AA3	B1	BB1
A3	AA3	B2	BB2

5.4.5 在一个语句中用多级连接类型

为了产生想要的结果有时不只需要连接两个表。如果需要返回所有的雇员、他们的部门名和他们负责的项目，就要连接 EMPLOYEE 表、DEPARTMENT 表以及 PROJECT 表，下面的例子给出查询语句和结果。

```
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'
```

EMPNO	LASTNAME	DEPTNAME	PROJNO
000020	THOMPSON	PLANNING	PL21000
000060	STERN	MANUFACTURING SYSTEMS	MA2110
000100	SPENSER	SOFTWARE SUPPORT	OP2010
000170	YOSHIMURA	MANUFACRING SYSTEMS	-
000180	SCOUTTEN	MANUFACTURING SYSTEMS	-
000190	SALKER	MANUFACTURING SYSTEMS	-
000250	SMITH	ADNINISTRATION SYSTEMS	AD3112
000280	SCHNEIDER	OPERATIONS	-
000300	SMITH	OPERATIONS	-
000310	SETRIGHT	OPERATIONS	-

5.4.6 连接的注意事项

在连接两个或多个表时要注意：

如果有通用的列名，必须用表名（或一个相关名）限定每个通用列名，如果列名是唯一的，则不需限定。

如果没有列出所需的列，而是用 SELECT *，SQL 返回的行包含第一个表中的所有列，紧跟着是第二个表中的所有列，等等。

对 FROM 子句中指定的每个表和视图中选择的行，必须要有权限。

分类顺序适用于所有被连接的字符和 UCS-2 图形列。

5.5 用 UNION 键字合并子选择

用 UNION 键字,可以把两个或更多子选择组成一个选择语句。当 SQL 遇到 UNION 键字时，它处理每个子选择形成一个中间的结果表，然后它合并每个子选择的中间结果表，去掉重复的行形成一个合并的结果表，用 UNION 合并来自两个或多个表中的值。当编写包括 ORDER BY 的 SELECT 语句时，可以使用任何一种学过的语句或技术。

当合并来自不同的表的列表值时，可以用 UNION 来消去重复。例如，可以获得一个雇员

号的组合表，它包括：

在部门 D11 的人

分配项目 MA2112、MA2113 和 AD3111 的人。

这个组合列表来自两个表，而且没有重复，要做到这点，规定：

```
MOVE 'D11' TO WORK-DEPT.
...
EXEC SQL
 DECLARE XMP6 CURSOR FOR
 SELECT EMPNO
   FROM CORPDATA.EMPLOYEE
   WHERE WORKDEPT = :WORK-DEPT
 UNION
 SELECT EMPNO
   FROM CORPDATA.EMP_ACT
   WHERE PROJNO = 'MA2112' OR
      PROJNO = 'MA2113' OR
      PROJNO = 'AD3111'
   ORDER BY EMPNO
END-EXEC.
...
EXEC SQL
 FETCH XMP6
   INTO :EMP-NUMBER
END-EXEC.
```

为了更好的理解执行这些 SQL 语句的结果，SQL 通过下面的过程：

PIC14

在用 UNION 时请注意：

ORDER BY 子句必须出现在组成的最后一个子选择之后。在这个例子中，结果用第一个选择的列 EMPNO 排序，ORDER BY 子句规定了合并结果表用分配顺序。

如果命名结果列，那么在 ORDER BY 子句中可以指定名字，如果在每一个合并的选择语句中相应的列有同样的名，则命名结果列。在选择列表中，可用 AS 子句来分配一个列名。

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

如果结果列没有被命名，用数字对结果进行排序，这个数字是在子选择表达式列表中表达式的位置。

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

当生成视图时，不能用 UNION。

为了区分每一行来自哪个子选择，可以在合并每个子选择的选择列表的结尾使用常量，当 SQL 返回结果时，最后一列包括行来源的子选择常量。例如，可以规定：

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

当这行送给程序时，它包含一个值（A1 或 B2）表明这个表是行值的来源。如果在合并中列名不同，当交互 SQL 显示或打印结果时，SQL 用在第一个子选择中规定的一组列名，或用从处理 SQL DESCRIBE 语句中得到的 SQLDA 结果。

有关在 UNION 中列的长度和数据类型兼容情况，请看 AS/400 DB2 SQL 参考的第四章。

注：分类顺序是在 UNION 的各部分合并后对字段进行排序的。在 UNION 运行期间隐式发生的不同处理都使用分类顺序。

5.5.1 规定 UNION ALL

如果要在 UNION 结果中保留重复，指定 UNION ALL 而不仅是 UNION。

PIC15

UNION ALL 操作是可联合的，例如：

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

与下面的语句得到同样结果：

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

当同样的 SQL 语句中用 UNION ALL 作为一个 UNION 操作符，操作的结果依赖计算的顺序。没有圆括号时，是由左到右计算。在有圆括号，先计算括号内的子选择，接着的其它部分从左到右计算。

5.6 使用子查询

到目前为止，在 WHERE 和 HAVING 子句中，用一个文字值，列名，表达式或寄存器来指定查找条件。在这些查找条件中，知道正查找一个规定的值，但有时不能提供这些值，直到从一个表中检索到数据。例如，假设想要一个在某个项目工作的雇员的号码、姓名和工作码，

这个项目号是 MA2100，这个语句的第一部分很容易写：

```
DECLARE XMP CURSOR FOR
SELECT EMPNO, LASTNAME, JOB
  FROM CORPDATA.EMPLOYEE
  WHERE EMPNO ...
```

但是不能再往下写了，因为表 CORPDATA.EMPLOYEE 不包括项目号数据，如果不对 CORPDATA.EMP_ACT 表用另一个 SELECT 语句，就不可能知道哪一雇员在 MA2100 项目中工作。

用 SQL，解决这个问题可以把一个 SELECT 语句嵌在另一个中，内层的 SELECT 语句叫子查询，子查询外部的 SELECT 语句叫外层 SELECT。用子查询，可以只用一个 SQL 语句来检索工作在项目 MA2100 上的雇员，工作代码、雇员号和姓名。

```
DECLARE XMP CURSOR FOR
SELECT EMPNO, LASTNAME, JOB
  FROM CORPDATA.EMPLOYEE
  WHERE EMPNO IN
     (SELECT EMPNO
        FROM CORPDATA.EMP_ACT
  WHERE PROJNO = 'MA2100')
```

为了更好的理解这个 SQL 语句的结果，SQL 通过下列过程：

PIC16

5.6.1 相关性

子查询的目的是指提供限定一行（WHERE 子句）或一组行（HAVING 子句）所需的信息，这些是通过子查询产生的结果表来完成的。从概念上讲，在计算子查询时，要限定一个新行或一组行。事实上，如果子查询与每一行或一组相同，它只计算一次。例如，前面的子查询与表 CORPDATA.EMPLOYEE 的每一行都有同一内容。这样的子查询叫作非相关的。一些子查询行到行或组到组的内容是变化的，允许这种机制叫作相关的。子查询说成是有相关性的，在 5.6.6 中介绍。

5.6.2 子查询和检索条件

子查询经常是检索条件的一部分，检索条件是操作数操作符（子查询）的格式。在例子中，操作数是 EMPNO，操作符是 IN，检索条件可以是 WHERE 或 HAVINOT 子句的一部分。子句可以包括包含子查询的多个检索条件。一个检索条件包含一个子查询，像其它的检索条件一样，可以括在一个圆括号中，前面可以有 NOT，也可用 AND 或 OR 与其它检索条件链接。例如，一些查询的 WHERE 子句可以像这样：

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

子查询也可以出现在其它子查询的检索条件中，这样的子查询叫做嵌套。例如，在一个外层 SELECT 中的子查询中的子查询是嵌套在两层嵌套内。SQL 允许向下嵌套 32 层，最小的嵌套需要嵌套层大于 1。

5.6.3 使用子查询

在 WHERE 或 HAVING 子句中包含一个子查询有四种方法，下面逐一介绍。

5.6.3.1 基本比较

可以在任何比较操作符后立刻用子查询，如果这样做，子查询可以返回至多一个值，这个值可能是列函数或一个算术表达式的结果。然后，SQL 对子查询的结果值和比较操作符左边的值进行比较。例如，假设想要找在公司中教育水平高于平均教育水平的雇员的雇员号、姓名和薪水：

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

SQL 首先计算子查询，然后替换在 SELECT 语句中的 WHERE 子句的结果。在这个例子中，结果是公司内的平均教育水平。除了返回一个单值外，子查询可能根本不返回值。如果是这样，比较的结果是未知的。例如，考虑在这一节中给出的第一个查询，假设当前没有雇员工作在项目 MA2100 上，那么子查询将不返回值，对于每一行，检索条件都是未知的，在这种情况下，查询产生的结果将是空表。

5.6.3.2 限定比较（ALL、ANY 和 SOME）

可用在比较操作符后跟着键字 ALL、ANY 和 SOME 的子查询。用这种方法时，子查询可以返回零个、一个或多个值，包括空值。可以用下面的方法使用 ALL、ANY 和 SOME：

用 ALL 表示你所提供的值必须用指出的方式与子查询返回的所有的值比较。假设用带 ALL 的大于比较操作符：

…WHERE 表达式 > ALL（子查询）

为满足这个 WHERE 子句，表达式的值必须大于子查询返回的所有值（就是说，要比最大值还要大）。如果子查询返回一个空集，（也就是，没有选择值），认为条件是满足的。

用 ANY 或 SOME 表示所提供的值必须用指出的方式与子查询返回的至少一个值进行比较。假设用带 ANY 的大于比较操作符：

…WHERE 表达式 > ANY（子查询）

为满足这个 WHERE 子句，表达式的值必须至少大于子查询返回的一个值。（就是说，大于最低的值）。如果子查询返回空值，则认为条件不满足。

注：当子查询返回一个或多个空值时，可能会让你感到意外，这是因为你对形式逻辑不熟悉，对于这些规则，请看 AS/400 DB2 SQL 参考。

5.6.3.3 使用 IN 键字

可用 IN 去说明表达式中的值必须在子查询返回的值中，这一章的第一个例子说明了这种类型的使用。用 IN 与用 =ANY 或 =SOME 是等价的。ANY 和 SOME 的用法在前面介绍过。也可用带 NOT 的 IN 键字，目的是选择那些值不在子查询返回值中的行，例如，可以用：

…WHERE WORKDEPT NOT IN（SELECT……）

5.6.3.4 使用 EXISTS 键字

到目前为止，在子查询中，SQL 计算子查询并用这个结果作为外层 SELECT 的 WHERE 子句的一部分。相反，当用键字 EXISTS 时，SQL 只简单地检查子查询返回的是一行或多行。如果是，条件就满足，如果不是（如果没返回行），条件就不满足，例如：

```
DECLARE XMP CURSOR FOR
SELECT EMPNO,LASTNAME
  FROM CORPDATA.EMPLOYEE
  WHERE EXISTS
     (SELECT *
        FROM CORPDATA.PROJECT
        WHERE PRSTDATE > '1982-01-01');
```

在这个例子中，如果在 CORPDATA.PROJECT 表中的任何项目有一个的起动日期晚于 1982 年 1 月 1 日，就保留这个检索条件。请注意：这个例子并没有显示出 EXISTS 的全部能力，因为检查外层 SELECT 的结果对于每一行经常是相同的。总之，或者是每一行都在结果中，或者是一行也没有。在更好的例子中，子查询本身是相关的，而且从一行修改为另一行，详细内容请看 5.6.6。

像例子中给出的，不需在 EXISTS 子句的子查询中指定列名，相反，可以用 SELECT *。

也可以用带 NOT 键字的 EXISTS 键字，这样在当指定的数据或条件不存在时用来选择行。可以用：

…WHERE NOT EXISTS（SELECT…）

对于所有子查询的使用类型，除了一个（用 EXISTS 键字的子查询）外，子查询都产生一个一列的结果表。这意味着在子查询中的 SELECT 子句必须命名一个列，或包含一个简单表达式。例如，下面两个 SELECT 子句对于四个使用类型都合适：

SELECT AVG(SALARY)
SELECT EMPNO
子查询产生的结果表可以有零个或多个行，对于某些用法，允许不多于一行。

5.6.4 使用有 UPDATE 和 DELETE 的子查询

到目前为止给出的例子中，已经看到在 SELECT 语句中的子查询，也可以使用在 UPDATE 和 DELETE 语句的 WHERE 子句中的子查询。对于大部分来讲，这和用外层 SELECT 子查询没有什么不同。在使用 UPDATE 或 DELETE 语句的子查询时，子查询不能与外部的 UPDATE 或 DELETE 语句基于同一张表。

5.6.5 使用子查询的几点说明

1、当嵌套 SELECT 语句时，可以嵌套所需满足要求的层数（1~31 层子查询）。尽管对于每一个增加的子查询，性能都会更慢一些。在 SQL 语句中最多可以指定 32 个表。
2、当外部语句是 SELECT 语句时（在嵌套的任一层）：
子查询可以和外部语句基于同一张表或视图，或基于不同的表或视图。
可以在外层 SELECT 的 WHERE 子句中用一个子查询，即使外层 SELECT 是 DECLARE CURSOR、CREATE VIEW 或 INSERT 语句的一部分的时候也可以。
可以在 SELECT 语句的 HAVING 子句中用一个子查询，这样做时，SQL 计算子查询并用它来限制每一组。
3、当语句是 UPDATE 或 DELETE 语句时：
子查询不能和 UPDATE 或 DELETE 语句基于同一张表。
可以在 UPDATE 或 DELETE 语句中的 WHERE 子句中用子查询。
4、子查询的结果必须包含一个单独的列，除非子查询是与 EXISTS 键字一起使用。表中的行数可以是零行到多行，但是对于不包括键字 ALL、ANY 或 SOME 的比较，行数必须是零或 1。
5、子查询不能包括 ORDER BY、UNION、UNION ALL、FOR READ ONLY、UPDATE 或 OPTIMIZE 子句。
6、在子查询中，就像检索条件，比较值必须是兼容的。
7、在子查询中用一个列函数或列名的算术表达式不能使它不兼容。当 SQL 应用列函数或算术操作符后，列的数据类型不能改变。
8、不能在同一查询中用子查询和分布式表。对于分布式表的详细内容，请看 AS/400 DB2 多系统一书。

5.6.6 相关子查询

在前面讨论的子查询中，SQL 计算子查询，替换在查询条件右侧子查询的结果，根据检索条件的值计算外层 SELECT，也可以写一个子查询，当 SQL 检查外层 SELECT 的每一个新行(WHERE 子句)或一组行(HAVING 子句)时，SQL 要重新计算，这叫作相关子查询。

5.6.6.1 WHERE 子句中相关子查询的例子

假设想要一个教育水平高于在他们各自部门中平均教育水平的所有雇员的列表。要得到这些信息，SQL 必须检索 CORPDATA.EMPLOYEE 表，对表中的每个雇员，SQL 要把他的教育水平与部门中的平均教育水平比较。在子查询中，要告诉 SQL 计算在当前行的部门号的平均教育水平。例如：

```
DECLARE XMP CURSOR FOR
SELECT EMPNO, LASTNAME, WORKDEPT, EDLEVEL
  FROM CORPDATA.EMPLOYEE X
  WHERE EDLEVEL >
     (SELECT AVG(EDLEVEL)
        FROM CORPDATA.EMPLOYEE
        WHERE WORKDEPT = X.WORKDEPT)
```

一个相关子查询除了一个或多个相关引用外,看起来就像一个非相关子查询。在例子中，单一的相关引用是子选择 FROM 子句中 X.WORKDEPT 的出现。在这里，量 X 是在外部 SELECT 语句中 FROM 子句里定义的相关名。在这个子句中，X 是作为表 CORPDATA.EPMLOYEE 的相关名介绍的。

现在，考虑对 CORPDATA.EMPLOYEE 的一行执行子查询时发生的情况。在它执行前，X.WORKDEPT 的出现将被这行的 WORKDEPT 列的值所代替。例如，假设这一行是 CHRISTINE I HAAS，它的工作部门是 A00，A00 是这一行 WORKDEPT 的值。对于这行，子查询这样执行：

```
(SELECT AVG(EDLEVEL)
   FROM CORPDATA.EMPLOYEE
   WHERE WORKDEPT = 'A00')
```

因此，对于考虑的行，子查询对 Christine 的部门产生平均教育水平，然后与外部语句中 Christine 自己的教育水平比较，对于 WORKDEPT 有不同值的其它行，这些值出现在 A00 位置的子查询中。例如，对于 MICHAEL L THOMPSON 的行，这个值是 B01，这一行的子查询将传递部门 B01 的平均教育水平。

查询产生的结果表将有下面的值：

PIC17

5.6.6.2 在 HAVING 子句中相关子查询的例子

假设想要所有平均薪水高于所在区平均薪水的部门（WORKDEPT 以同一字母开始的所有部门属于同一区）。为了得到这个信息，SQL 必须查找 CORPDATA.EMPLOYEE 表。对于表中的每个部门，SQL 比较部门的平均薪水和该区的平均薪水，在子查询中，SQL 计算在当前组部门区的平均薪水，例如：

```
DECLARE XMP CURSOR FOR
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))
```

考虑对于一个给出的 CORPDATA.EMPLOYEE 中的部门，当执行子查询时发生什么情况。在它执行前，X.WORKDEPT 的出现被这一组的 WORKDEPT 列的值所代替，假设选择的第一组 WORKDEPT 的值为 A00，这一组的子查询执行是：

```
(SELECT AVG(SALARY)
   FROM CORPDATA.EMPLOYEE
   WHERE SUBSTR('A00',1,1) = SUBSTR(WORKDEPT,1,1))
```

因此，对于考虑的组，子查询产生这一区的平均薪水，然后与部门‘A00’的平均薪水的外层语句比较，对于 WORKDEPT 是 B01 的其它组，子查询将产生部门 B01 所在区的平均薪水。

查询产生的结果表将有下面的值：

PIC18

5.6.6.3 相关名和引用

一个相关引用仅能出现在子查询的检索条件中，引用经常是 X.C 的格式，其中 X 是相关名，而 C 是 X 体现的表中的列名。例如，在前面的例子中，X 体现的是表 CORPDATA.EMPLOYEE，C 代表的是在这个表中的列 WORKDEPT。

相关名是在查询的 FROM 子句中定义的，这个查询可以是外层 SELECT 或是一些包含在引用中的一个子查询。例如，一个包含子查询 A、B 和 C 的查询，A 包含 B，B 包含 C，那么用在 C 中的相关名可能是在 B、A 中定义的或是外层 SELECT。

可以为每一个出现在 FROM 子句中表名定义一个相关名，在表名后简单地包括一个相关名。在表名和它的相关名之间有一个或多个空格，如果相关名后跟另一个表名，则在相关名后放一逗号。例如，下面的 FROM 子句，对表 TABLEA 和 TABLEB 定义了相关名 TA 和 TB，而表 TABLEC 没定义相关名。

```
FROM TABLEA TA, TABLEC, TABLEB TB
```

在子查询中可以出现任何数量的相关名，没有限制。例如，一个在引用中的相关名可在外层 SELECT 中定义，而另一个可在包含的子查询中定义。

在子查询执行前，来自引用列的值经常被相关的引用来代替，这个值象下面那样确定：

注：用 D 来指出已定义相关名的查询，然后子查询或者在 D 的 WHERE 子句中，或者在 HAVNG 子句中。

如果子查询是在 WHERE 子句中，它的结果由 D 来限定一行。然后将从这一行取出替换值，在外部定义查询而子查询出现在外部查询的 WHERE 子句中就是这种情况的例子。

如果子查询是在 HAVING 子句中，它的结果由 D 来限定一组行，然后将从这一

组中取出替换值。在这种情况下，指定的列必须在 D 中的 GROUP BY 子句中被标识，如果不这样，指定的列对这一组有多个值。

5.6.7 在 UPDATE 语句中用相关子查询

在 UPDATE 语句中使用相关子查询时，相关名指的是你要更新的行。例如，当一个项目的所有活动必须在 1983 年 9 月前完成，这个项目为优先项目。应该用下面的 SQL 语句来计算 CORPDATA.PROJECT 表中的项目，对每一个优先项目在 PRIORITY 列（加在 CORPDATA.PROJECT 中的一列）上写一个 1（表明优先的标志）。

```
UPDATE CORPDATA.PROJECT X
  SET PRIORITY = 1
  WHERE '1983-09-01' >
     (SELECT MAX(EMENDATE)
        FROM CORPDATA.EMP_ACT
        WHERE PROJNO = X.PROJNO)
```

当 SQL 检查在 CORPDATA.EMP_ACT 表中的每一行时，它检查所有项目（从 CORPDATA.PROJECT 表）的最大活动期限（EMENDATE）。如果项目最后期限都在 1983 年 9 月之前，则在表 CORPDATA.PROJECT 中的当前行被指出并更新。

5.6.7.1 用 UPDATE 的子查询的限制

一个 UPDATE 语句的子查询必须引用与被更新行的表不同的表，例如，下面的语句看起来是给那些工资少于部门平均薪水的雇员涨 10%薪水，但事实上它并没完成。

```
UPDATE CORPDATA.EMPLOYEE X
  SET SALARY=SALARY*1.10
  WHERE SALARY <
       (SELECT AVG(SALARY)
          FROM CORPDATA.EMPLOYEE
          WHERE X.WORKDEPT=WORKDEPT)
```

操作结果不依赖访问表中行的次序，如果执行这个语句，部门的平均薪水不能与整个更新保持一致而结果变成依靠表中存取行的次序，因此，操作是不允许的。

5.6.8 在 DELETE 语句中使用相关子查询

在 DELETE 语句中使用相关子查询时，相关名表示的是要删除的行。SQL 对在 DELETE 语句中命名表中的每一行计算一次相关子查询，以决定是否删除这一行。

假设在 CORPDATA.PROJECT 表的一行被删除，在 CORPDATA.EMP_ACT 表中与删除的项目相关的行也必须被删除，为了做这些，可以用下面语句：

```
DELETE FROM CORPDATA.EMP_ACT X
  WHERE NOT EXISTS
     (SELECT *
        FROM CORPDATA.PROJECT
        WHERE PROJNO = X.PROJNO)
```

对于表 CORPDATA.EMP_ACT 的每一行，SQL 确定在 CORPDATA.PROJECT 表中是否有一个带同样项目号的行存在。如果不存在，CORPDATA.EMP_ACT 的行被删除。

5.6.8.1 带 DELETE 子查询的限制

一个有 DELETE 语句的子查询必须不能引用与删除行的表相同的表，考虑下面的语句，看起来像是删除每一个不能管理其它部门的部门，但实际上不能完成。

```
DELETE FROM CORPDATA.DEPARTMENT X
  WHERE NOT EXISTS
     (SELECT * FROM CORPDATA.DEPARTMENT
      WHERE ADMRDEPT = X.DEPTNO)
```

操作结果必须不能依靠表中的行的访问次序，如果执行这个语句，它的结果将依赖以前访问的部门的行，或依赖删除它管理的部门行之后，由于这个原因，操作被禁止。

5.6.9 使用相关子查询的几点说明

相关名和与它相关的表名要用一个空格分开，要指定另一表名，用一个逗号放在这个表名前。例如：

```
    FROM CORPDATA.EMPLOYEE X, CORPDATA.PROJECT ....
```

在一个 SELECT 语句中，相关子查询和外层语句可以指定同一个表或不同的表。

在一个 UPDATE 或 DELETE 语句中，相关子查询和外层语句必须引用不同的表。

在一个 INSERT 语句中，在 INSERT 语句中的相关子查询或外层 SELECT 都不能与要插入的表是同一张表。

定义相关名的外层 SELECT 可以连接两个或多个表。

可以在 HAVING 子句中用相关子查询，这样做时，SQL 计算这个外层 SELECT 的子查询每组一次。在 HAVING 子句中引用的列必须指定每一组的特征。（例如，WORKDEPT）可以是由行分组的列，也可以是带有一个列函数的其它列。

可以嵌套相关子查询。

5.7 改变表定义

ALTER TABLE 语句可以用来修改一个表的定义，可以加入新列，改变已有列的定义（长度、缺省值等等），删除一个列以及增加或取消限制。所有这些都可以用一个 ALTER TABLE 语句实现，其限制是在 ADD COLUMN、ALTER COLUMN 和 DROP COLUMN 子句中只能引用一行一次，就是说，不允许在同一 ALTER TABLE 语句中增加一列，然后再改变这一列。

当在表中加一个新列时，列用它的缺省值初始化。如果规定了 NOT NULL，必须指定缺省值。一个替换表最多可以包含 8000 列，列的字节总数不能超过 32766，如果指定 VARCHAR 或 VARGRAPHIC 列，则为 32740。

要改变存在列的数据类型，旧属性和新属性一定要兼容。下面的表给出允许的转换：

表 5-3 允许的转换

FROM 数据类型	TO 数据类型
十进制	数值
十进制	整数，小整数
十进制	浮点
数值	十进制
数值	整数，小整数
数值	浮点
整数，小整数	十进制
整数，小整数	数值
整数，小整数	浮点
浮点	数值
浮点	整数，小整数
字符	DBCS-open
字符	UCS-2 图形
DBCS-open	字符
DBCS-open	UCS-2 图形
DBCS-either	字符
DBCS-either	DBCS-open
DBCS-either	UCS-2 图形
DBCS-only	DBCS-open
DBCS-only	DBCS 图形
DBCS-only	UCS-2 图形
DBCS 图形	UCS-2 图形
UCS-2 图形	字符
UCS-2 图形	DBCS-open
UCS-2 图形	DBCS 图形

当将长度变长时，数据由相应的填充字符填充，当长度变短时，数据由于截断而丢失，一个询问信息将提示你确认该请求。

如果有一个不允许空值的列，现在想改变它为允许空值，用 DROP NOT NULL 子句。如果有一个允许空值的列，你要禁止它为空值，则用 SET NOT NULL 子句。如果这个列中的值是空值，而且没执行 ALTER TABLE，将导致一个-190 的 SQLCODE。

当修改一个列时，只有所规定的属性改变，而其它的属性都保持不变，例如，给出下面表的定义：

```
CREATE TABLE EX1 (COL1 CHAR(10) DEFAULT 'COL1',
                  COL2 VARCHAR(20) ALLOCATE(10) CCSID 937,
```

```
COL3 VARGRAPHIC(20) ALLOCATE(10)
NOT NULL WITH DEFAULT)
```

在运行下面的 ALTER TABLE 语句后：

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

COL2 的长度仍为 10 以及 CCSID 为 937，COL3 的长度仍为 10。

丢失（DROP）一个列将在表定义中删除一列，如果指定 CASCADE，任何与其表为基础的视图、索引和限制都要丢失。如果指定 RESTICT，任何与此表有关的视图、索引和限制都给出 SQLCODE 为 -196，这一列将删不掉。

关于增加和删除限制的细节，请看第六章。

一个 ALTER TABLE 语句的执行是做为落实控制的一组操作进行的，而操作分成下面几步进行：

1、删除限制。

2、删除规定 RESTRICT 选项的列。

3、改变列定义（包括增加和删除指定 CASCADE 选项的列）。

4、增加限制。

在其中每个阶段，用户指定语句的顺序就是执行的顺序，只有一个例外，如果正在删除一些列，而在逻辑上这些操作在增加列定义或修改列定义之前做，此时，ALTER TABLE 执行的结果是增加了记录长度。

5.8 建立和使用视图

用一个视图可以访问一个或多个表中的部分数据。可以在 SELECT 子句中定义视图的列，用 FROM 子句定义视图所用的表。为了定义视图中的行，可以指定 WHERE 子句、GROUP BY 子句或 HAVING 子句。

例如，要建立一个只选择所有经理的姓名和部门的视图，规定下面的语句：

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

如果选择列表要包括其它的列元素，象表达式、函数、常量或特殊寄存器，而没用 AS 子句来命名列，那么必须为视图规定列表。在下面的例子中，视图的列是 LASTNAME 和 YEARSOFSERVICE。

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

前面的视图也可以在选择列表中用 AS 子句命名视图中的列，例如：

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

一旦建立了视图，可以用它选择数据或修改基本表中的数据。

在建立视图时，必须考虑下面的限制：

不能在只读视图中改变、插入或删除数据。一个视图如果包括下列之一则它是只读视图：

—第一个 FROM 子句规定多个表（连接）。

—第一个 FROM 子句规定一个只读视图。

—第一个 SELECT 子句包含任何 SQL 列函数。（SUM、MAX、MIN、AVG、COUNT、STDDEV 或者 VAR）

—第一个 SELECT 子句指定 DISTINCT 键字。

—外部子选择包含一个 GROUP BY 或 HAVING 子句。

—一个子查询，最外部的子选择的基本目标和子查询的表是同一个表。

在上面情况下，可以利用 SQL SELECT 语句从视图中得到数据，但不能用 INSERT，UPDATE 或 DELETE 语句。

在下列条件下，不能插入一行：

—视图基于的表有一列没有缺省值，不允许空，且不在视图中。

—视图有一列是表达式的结果、常量、函数或特殊寄存器，且是在 INSERT 列表中规定的。

—生成视图时，规定了 WITH CHECK OPTION 且行不符合选择准则。

不能更新视图的一列，这一列来源于一个表达式、常量、函数或特殊寄存器。

在定义视图时，不能用 UNION、UNION ALL、FOR UPDATE OF、FOR READ ONLY、ORDER BY 或者 OPTIMIZE FOR n ROWS。

视图用在 CREATE VIEW 语句运行时有效的分类顺序，它在 CREATE VIEW 语句子选择中支持所有字符和 UCS-2，详细内容请看 3.7。

视图也可以用 WITH CHECK OPTION 生成，它规定在通过视图插入或更新数据时所做检查的级别。详细内容请看 6.3。

5.9 使用索引

系统用索引能更快地检索数据。下面的例子是对表 CORPDATA.MPLOYEE 中用列 LAST NAME 建立一个索引：

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

可以生成任何数量的索引，但因为索引是系统管理的，索引数越多可以极大地影响性能，详细内容请看 22.6。

如果生成的索引与一个已有的索引有绝对相同的属性，新的索引可共享已存在的索引的二进制树，否则，要生成另一个二进制树，如果一个新索引的属性与另一个索引非常相同，除了新索引有一个或更少的列，那么仍要生成另一个二进制树，这是因为附加的列将禁止游

标使用索引及禁止对这些附加的列使用 UPDATA 语句。

索引使用 CREAT INDEX 语句运行时有效的分类顺序，它适用于索引中所有的 SBCS 字符字段和 UCS-2 图形，详细内容请看 3.7。

5.10 数据库设计中使用目录

当生成一个集合时，自动创建一个目录。在 QSYS2 库中也经常有系统范围目录。在集合中生成 SQL 目标时，信息要加到系统目录表中和集合目录表中。在库中生成 SQL 目标时，只有 QSYS2 中的目录被更新。详细内容请看 AS/400 DB2 SQL 参考一书。

像下面的例子给出的，可以显示目录信息，但不能 INSERT、DELETE 或 UPDATE 目录信息。要运行下面例子，必须有目录视图上的权限。

注：如果集合被保存、重存或给一个不同的名，通常更新 SQL 目录的操作不会再更新目录，SQL 产品不支持保存一个集合，然后再重存到另一个集合中。

5.10.1 获取表的目录信息

SYSTABLES 对在 SQL 集合中的每个表和视图中都有一行信息，它告诉你目标是表或是视图，目标名和目标的所有者，是在哪个 SQL 集合中，等等。

下面的语句显示 CORPDATA.DEPARTMENT 表的信息：

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

5.10.2 获取列的目录信息

SYSCOLUMNS 对集合中的每个表和视图都有一行信息。

下面语句给出在 CORPDATAH.DEPARTMENT 表中的所有列的名字：

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

前面的例子的结果是对表中每列有一行信息，其中一些信息是不可见的，因为该信息的宽度比显示屏还要宽。

关于每列的更多信息，定义如下选择语句：

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

对于每一列，除了列名字以外，选择语句给出下列信息：

包含列的表的名字
列的数据类型
列的长度
是否允许缺省值

结果如下所示：

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

第六章　　数据完整性

数据的完整性是集合中表之间数据的值要保持一致，这是商业活动中的一个重要原则。例如，银行在表 A 中有一个客户清单，在表 B 中有一个客户的存款清单，那么一个客户要先申请加入表 A 中，该客户的存款才允许加入到表 B 中。

这章介绍系统自动保证这几种关系的不同方法。引用完整性、检查约束和触发器是保证数据完整性的有效方法。另外，在 CREATE VIEW 中的 WITH CHECK OPTION 子句限制通过视图插入和更新数据。

关于数据完整性的其它内容，请看 AS/400 DB2 数据库编程。

6.1 AS/400 DB2 检查约束

一个检查约束限制在一列或一组列中允许值的规则。SQL 支持在 CREATE TABLE 和 ALTER TABLE 语句中的检查约束。这些命令的详细说明，请看 AS/400 DB2 SQL 参考一书。

考虑下面的语句：

```
CREATE TABLE T1 (COL1 INT, COL2 INT CHECK (COL2>0), COL3 INT)
```

这个语句生成有三个列的表，第二列有一个约束条件，它限制该列中的值要为正整数。对这个表，如果用下面的语句：

```
INSERT INTO T1 VALUES (-1, -1, -1)
```

将要失败，因为插入到第二列中的值没有满足约束条件，-1 并不大于零。若用下面的语句：

```
INSERT INTO T1 VALUES (1, 1, 1)
```

将是成功的。该行被插入后，下面的语句就是错误的：

```
ALTER TABLE T1 ADD CONSTRAINT C1 CHECK (COL1=1 AND COL1<COL2)
```

这个修改表的语句试图加入一个约束，用来限制第一列为“1”这个值，让第二列的值大于 1，这个约束条件不被接受，因为已经存在的数据并不满足约束条件的第二部分，（第二列中的“1”并不小于第一列的值“1”。）

6.2 AS/400 DB2 引用完整性

在原文附录 A 的 AS/400 DB2 样本表中：

CORPDATA.EMPLOYEE 是雇员的主列表。

CORPDATA.DEPARTMENT 所有有效的部门号的主列表。

CORPDATA.EMP_ACT 是执行项目的活动主列表。

其它的表引用这些表中相同的描述。当一个表包含了主表中的数据，那么数据应确实出现在主列表中，否则引用无效。包含主列表的表是父表，引用它的表是从属表，这一系列表中一个表引用另一个表的条件是有效的，称作引用完整性。

对引用完整性的讨论要求了解以下术语：

唯一键字是表中唯一标识一行的一列或一组列，尽管一个表可能有几个唯一键字，但是在一个表中不可能有两行有相同的唯一键值。

主键字是不允许为空的唯一键字，一个表中只能有一个主键字。

父键字既可是唯一键字也可是主键字，它们是在引用约束中被引用的。

外来键字是一列或几列的值必须与它们的父键字一致的列。如果用来建立外来键字的列值为空，那么这个规则不适用。

一个父表是一个包含父键字的表。

一个从属表是一个包含外来键字的表。

一个子表是一个从属表或是一个从属表的子表。

引用完整性的实施防止了违反规则，即每一个非空的外来键字的状态必须有相匹配的父键字。

SQL 支持在 CREATE TABLE 和 ALTER TABLE 语句引用完整性的概念。这些命令的详细说明，请看 AS/400 DB2 SQL 参考一书。

6.2.1 生成有引用约束的表

在定义引用约束时，要规定：

一个主键或唯一键字

一个外来键字

删除和修改关系规则。

可选的，可以为约束条件指定一个名字，如果没规定名字，那么就自动生成一个。删除和修改规则规定了当删除和修改父行时，要考虑子行的动作。

例如，在样本雇员表中给出的每一个部门号必须出现在从属表中的规则就是一个引用约束。这个约束保证了每个雇员属于一个部门。下面的 SQL 语言用这些约束关系定义创建了 CORPDATA.DEPARTMENT 和 CORPDATA.EMPLOYEE 表。

```
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))
```

在这种情况下，DEPARTMENT 表有唯一的部门号列作为一个主键字，是两个约束关系中的父表：

REPORTS_TO_EXISTS 是一个自引用约束，DEPARTMENT 表是同一关系的父表和从属表。每一个 ADMRDEPT 非空值必须与 DEPTNO 的值相匹配。一个部门必须报告出在数据库中存在的部门。DELETE CASCADE 规则表明，如果有 DEPTNO 值 n 的行被删除，那么表中 ADMRDEPT 值为 n 的各行也应被删除。

WORKDEPT－EXISTS 建立了 EMPLOYEE 表做为一个从属表。WORKDEPT 做为一个外来键字，这样，WORKDEPT 的每一个值必须和 DEPTNO 的值匹配，DELETE SET NULL 的规则是指：如果从 DEPARTMENT 中的 DEPTNO 的值为 n 的一行被删除了，那么在 EMPLOYEE 中 WORKDEPT 的值为 n 的行应置空。UPDATE RESTRICT 规则是指：如果在 EMPLOYEE 中的 WORKDEPT 的值和当前的 DEPTNO 值相匹配，那么不能更新 DEPARTMENT 中的 DEPTNO 的值。

在 EMPLOYEE 表中的 UNIQUE_LNAME_IN_DEPT 约束导致一个部门中的名字要唯一，当这个约束是不可能的时，它解释一个由几列组成的约束是怎样在表级上被定义的。

一旦定义了引用约束，系统就开始在通过 SQL 或其它接口（包括 CL 命令、实用程序和 HLL 语句）完成的每个插入、删除和修改操作中实施这个约束。

6.2.2 取消引用约束

ALTER TABLE 语句能用来每次从表中增加或去掉一个约束条件。如果一个被去掉的约束条件是某些引用约束中的父键字，那么也取消父文件和从属文件之间的约束。

DROP TABLE 和 DROP COLLECTION 语句也取消表或集合的约束。

6.2.2.1 取消约束的例子

下面的例子是在表 DEPARTMENT 中取消 DEPTNO 列的主键字，在表 DEPARTMENT 和 EMPLOYEE 上定义的 REPORTS_TO_EXISTS 和 WORKDEPT_EXISTS 约束也将取消，因为取消的主键字是这些约束关系中的父键字。

```
ALTER TABLE CORPDATA.EMPLOYEE DROP PRIMARY KEY
```

能用名字取消一个约束，下面是例子：

```
ALTER TABLE CORPDATA.DEPARTMENT
        DROP CONSTRAINT UNIQUE_LNAME_IN_DEPT
```

6.2.3 往有引用约束的表中插入数据

在往有引用约束的表中插入数据时，有一些重要的事情要记住，如果对有父键字的父表插入数据时，SQL 不允许：

父键字有重复值。

如果父键字是主键字，主键字的任意一列是空值。

如果往有外来键字的从属表中插入数据：

插入到外键字列的非空值必须与父表中的相应父键字的某些值相等。

如果外来键字的列有空值，全部的外来键字就是空值。如果所有外来键字都是空值，则插入成功（这里无唯一索引错误）。

6.2.3.1 用约束插入数据的例子

修改样板应用项目表（PROJECT）定义两个外来键字：

一个外来键字是引用部门表中的部门号（DEPTNO）

一个外来键字是引用雇员表中的雇员号（RESPEMP）

```
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
```

注意到父表列不在 REFERENCES 子句中规定，只要引用表有一个主键字或者有一个合适的唯一键字可以用来做父键字，那么就不要求规定这些列。

每一个插入到 PROJECT 表的行必须有一个 DEPTNO 的值和在部门表中的 DEPTNO 的某些值相等。（空值是不允许的，因为 DEPTNO 在项目表中被定义为 NOT NULL）。行必须有一个 RESPEMP 的值或是和雇员表中的 EMPNO 的值相等或者为空。

下面的 INSERT 语句不能成功执行，因为没有与 DEPARTMENT 表中 DEPTNO 值（‘A01’）相匹配的值。

```
INSERT INTO  CORPDATA.PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
        VALUES  ('AD3120', 'BENEFITS ADMIN', 'A01', '000010')
```

同样，下面的插入语句也不成功，因为在表 EMPLOYEE 中没有‘000011’的 EMPNO 值。

```
INSERT INTO CORPDATA.PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
        VALUES ('AD3130', 'BILLING', 'D21', '000011')
```

下面的插入语名能成功的完成，因为在表 DEPARTMENT 中有一个匹配的 DEPTNO 值‘E01’，在表 EMPLOYEE 中有一个匹配的 EMPNO 值‘000010’。

```
INSERT INTO CORPDATA.PROJECT (PROJNO, PROJNAME, DEPTNO, RESPEMP)
        VALUES ('AD3120', 'BENEFITS ADMIN', 'E01', '000010')
```

6.2.4 更新有引用约束的表

如果更新一个父表，不能修改从属表行存在的主键字。修改键字将违背从属表的引用约束且使一些行无父表行，此外，不能使主键字的任何一部分为空值。

6.2.4.1 更新规则

在父表中执行 UPDATE 时，对从属表采取的动作，依赖于对引用限制规定的更新规则。如果没有规定更新规则，则使用 UPDATE NO ACTION 规则。

UPDATE NO ACTION：

如果没有其它的行依整于父表中的行，那么可以更新它。如果从属行中存在依赖关系，UPDATE 失败。对从属行的检查是在语句的末尾执行的。

UPDATE RESTRICT：

它指出，如果没有其它的行依整于父表中的行，那么可以更新它。如果从属行中存在依赖关系，UPDATE 失败，对从属行的检查是立即执行的。

在了解了触发器与引用约束的相互影响时，很容易看到 RESTRICT 与 NO ACTION 之间的细微不同。触发器可以定义在操作前或操作后触发（此时是 UPDATE 语句）。一个前触发器是在执行 UPDATE 前触发，因此是在检查约束之前。后触发器是在执行 UPDATE 之后触发，RESTRICT 约束规则之后（在此检查立即执行），但在 NO ACTION 约束规则之前（检查是在语句的末尾执行），触发器和其规则按下列顺序发生：

1.一个前触发器是在 UPDATE 之前和 RESTRICT 或 NO ACTION 约束规则之前触发。

2.一个后触发器将在 RESTRICT 约束规则之后，在 NO ACTION 规则之前触发。

如果正修改一个从属表，一些修改的非空外来键值，必须和从属表的主键字相匹配。例如，在雇员者表中的部门号依赖于部门表中的部门号，你可以不给雇员分配部门（空值），但不能给它分配一个不存在的部门。

如果对一个表的更新违反了引用约束条件从而导致 UPDATE 失败，那么在更新操作之中的所有改变都被还原。详细内容请看 19.2.2 和 19.2.3。

6.2.4.2 UPDATE 规则的例子

例如，如果一个部门对那些在项目表用部门行描述的项目仍然有责任的话，就不能更新部门表中的部门号。

下面的更新是失败的，因为项目表中有依赖于值是‘D01’的 DEPARTMENT.DEPTNO 的行（WHERE 语句指定的行），如果允许这个 UPDATE 的话，那么就打破了在项目表和部门表之间的引用约束。

```
UPDATE CORPDATA.DEPARTMENT
        SET DEPTNO = 'D99'
        WHERE DEPTNAME = 'DEVELOPMENT CENTER'
```

下面的语句也是失败的，因为它违反了存在于部门表的主键字 DEPTNO 和项目表的外来键字 DEPTNO 之间的引用约束。

```
UPDATE CORPDATA.PROJECT
  SET DEPTNO = 'D00'
  WHERE DEPTNO = 'D01';
```

语句试图把所有部门号为‘D01’的改为‘D00’，因为‘D00’不是部门表中主键字 DEPTNO 的值，故语句失败。

6.2.5 从有引用约束的表中删除数据

如果一个表有主键字但没有从属表，那么删除操作就象没有引用约束一样去做，同样，如果一个表只有外来键字没有主键字也是如此。如果一个表有一个主键字和从属表，那么 DELETE 就根据定义的删除规则进行删除或修改行。所有影响关系的删除规则必须满足删除操作能成功进行的目的。如果违反引用约束，删除失败。

当在父表中执行 DELETE 时，在从属表中采取的动作依赖于约束定义的删除规则。如果没有定义删除规则，使用 DELETE NO ACTION。

DELETE NO ACTION：

如果无其它行从属于父表中的行，就可以删除父表中的行。如果关系中存在从属性，那么删除失败。从属行的检查是在语句的末尾执行的。

DELETE RESTRICT：

如果无其它行从属于父表中的行，那么就可删除父表中的行。如果关系中存在从属行，那么删除失败。从属行的检查立即执行。

例如，如果一个部门对那些在项目表中的用部门描述的项目仍然有责任的话，就不能从部门表中删除。

DELETE CASCADE：

删除父表中第一个被指出的行，然后，再删除从属行。

例如，能从部门表中的删除部门行，同时也删除了：

－向它报告的所有部门的行

－向这些部门报告的所有部门等

DELETE SET NULL：

在每一个从属行中可为空白外来键字的列被置成缺省值。就是说，如果这列是被删除行引用的外来键字的一部分，那么该列就置为缺省值。仅仅影响立即派生的从属性。

DELETE SET DEFAULT：

指出在每一个从属行中的外来键字的每一列被置为它的缺省值。这就意味着如果这列是被删除行引用的外来键字的一部分，那么该列置为它的缺省值。仅仅影响立即派生的从属性。

例如，如果一个雇员管理一些部门，可以从雇员表中删除该雇员。在这种情况下，向管理者报告的每个雇员的 MGRNO 的值在部门表中被设置为空白。在生成表时如果定义了缺省值，那么就使用该值。

这是由于在部门表中定义了 REPORTS_TO_EXISTS 约束。

如果一个派生表有 RESTRICT 或 NO ACTINON，且找到删除规则，不能删除的派生行，那么整个 DELETE 失败。

当在程序中执行这个语句时，删除的行数送往 SQLCA 的 SQLERDD(3)。这个数只指出在 DELETE 语句中规定的表中删除的行数，不包括根据 CASCADE 规则删除的行数。在 SQLCA SQLERRD(5)中包含那些所有表中引用约束影响的行数。

在了解了触发器与引用约束的相互影响时，很容易看到 RESTRICT 与 NO ACTION 之间的细微不同。触发器可在操作前或操作后触发(此时是 DELETE)。一个前触发器是在执行 DELETE 前触发，因此是在检查约束之前。后触发器是在执行 DELETE 之后触发，在 RESTRICT 约束规则之后（在此检查立即执行），但在 NO ACTION 约束规则之前（检查是在语句的末尾执行），触发器和其规则按下列顺序发生：

1. 一个前触发器是在 DELETE 之前和 RESTRICT 或 NO ACTION 约束规则之前触发。

2. 一个后触发器将在 RESTRICT 约束规则之后在 NO ACTION 规则之前触发。

6.2.5.1 DELETE CASCADE 的例子

从部门表中删除一个部门，它设定分配给此部门的雇员的 WORKDEPT（在雇员表中）为空。考虑下面的删除语句：

```
DELETE FROM CORPDATA.DEPARTMENT
        WHERE DEPTNO = 'E11'
```

本书原文附录 A 中的 AS/400 DB2 样板表给出表和数据。此时，雇员表中的值为‘E11’的 WORKDEPT 的值置为缺省值。在下面的样板数据中？表示空值。结果如下面所示：

表 6－1 部门表 删除语句完成后表的内容

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

注意在部门表中没有级联删除，因为没有部门和 E11 部门相关。

下面是在删除语句完成前后雇员表受影响的部分。

表 6－2　　雇员表的一部分，删除前的部分内容

EMPNO	FIRSTMNE	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 6－3　　雇员表的一部分，删除后的部分内容

EMPNO	FIRSTMNE	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		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

6.2.6　检查未决

引用约束和检查约束作为检查未决的一个状态，这时有潜在的违例约束。对于引用约束，当潜在的不匹配存在于父键和外来键之间时，发生违例。对于检查约束，存在一个被检查约束所限制的列中潜在的值时，发生违例。当系统确定约束可以违背时（例如重存之后），约束被标记为检查未决。发生这种情况时，约束放在包含约束所使用表中。对引用约束，提供下面的限制：

在从属的文件中不允许输入输出

在父文件中只允许读和插入操作

当检查约束是检查未决时，提供下面的限制：

在文件中不允许读操作

允许插入和修改操作，强制执行约束

要从检查未决中得到约束退出，必须做：

1.用 CHGPFCST 命令停止关系。

2.对引用约束更改键数据（外来键或父键，或二个都改）。对检查约束更改列数据。

3.用 CHGPFCST 命令启动约束。

可用 DSPCPCST 命令显示违背约束标识的行。详细信息请看 AS/400 DB2 数据库程序设计。

6.3 视图中的 WITH CHECK OPTION

WITH CHECK OPTION 是 CREATE VIEW 语句中的子句，它规定用视图来插入和修改数据时，所做的检查级别，如果规定了这个选项，通过视图插入或修改的每一行必须与视图的定义一致。

如果视图是只读的，就不能规定 WITH CHECK OPTION。视图的定义不能包括一个子查询。

如果生成视图时没规定 WITH CHECK OPTION 子句,用视图执行的插入和修改操作不被检查是否与视图定义的一致性。如果视图直接或间接地依赖于包含 WITH CHECK OPTION 的其它视图，那么要做一些检查。因为没有使用视图的定义，可以通过与视图定义不一致的视图插入或修改行，这就意味着不能再次用视图选择行。

检查可以是 CASCADED 或 LOCAL，详细内容请看 AS/400 DB2 SQL 参考一书。

6.3.1 WITH CASCADED OPTION

WITH CASCADED OPTION 规定了通过视图插入或修改的每一行必须与视图定义相一致。另外，当插入或修改一行时，要检查所有从属视图的检索条件。如果一行不和视图定义一致，就不能使用视图恢复该行。

考虑下面的可修改视图：

```
CREATE VIEW V1 AS SELECT COL1
       FROM T1 WHERE COL1 > 10
```

因为没有规定 WITH CASCADED OPTION，尽管插入的值不符合视图的检索条件，下面的插入语句也是成功的：

```
INSERT INTO V1 VALUES (5)
```

在 V1 基础上生成另一个视图，规定 WITH CASCADED OPTION：

```
CREATE VIEW V2 AS SELECT COL1
          FROM V1 WITH CASCADED CHECK OPTION
```

下面的插入语句是失败的，因为产生了和 V2 定义不一致的行：

```
INSERT INTO V2 VALUES (5)
```

考虑另一个在 V2 上生成的视图：

```
CREATE VIEW V3 AS SELECT COL1
        FROM V2 WHERE COL1 < 100
```

下面的插入语句失败，是因为 V3 是依赖于 V2 的，而 V2 有一个 WITH CASCADED OPTION。

```
INSERT INTO V3 VALUES (5)
```

但下面的语句是成功的，因为它与 V2 的定义一致。因为 V3 没有 WITH CASCADED OPTION，所以语句和 V3 的定义不一致就不是问题了。

```
INSERT INTO V3 VALUES (200)
```

6.3.2 WITH LOCAL CHECK OPTION

WITH CASCADED OPTION 和 WITH LOCAL CHECK OPTION 是完全相同的，除了能更新一行以致不能再用视图对行进行恢复，这只能发生在视图直接或间接地依赖于定义了没有 WITH CHECK OPTION 子句的视图时。

例如，考虑前一个例子中使用的同一个可更新视图：

```
CREATE VIEW V1 AS SELECT COL1
        FROM T1 WHERE COL1 > 10
```

在 V1 上生成第二个视图，这次规定 WITH LOCAL CHECK OPTION。

```
CREATE VIEW V2 AS SELECT COL1
        FROM V1 WITH LOCAL CHECK OPTION
```

同一个 INSERT，在前面的 WITH CASCADED OPTION 例子中失败，而此时规定相同的插入成功了，这是因为 V2 没有任何条件，V1 的检索条件不需要检查，因为 V1 没规定检查选项。

```
INSERT INTO V2 VALUES (5)
```

考虑根据 V2 生成的另一个视图：

```
CREATE VIEW V3 AS SELECT COL1
        FROM V2 WHERE COL1 < 100
```

下面的插入又是成功的，因为 V1 的检索条件由于 V2 的 WITH LOCAL CHECK OPTION 而不做检查。

```
INSERT INTO V3 VALUES (5)
```

LOCAL 和 CASCADED CHECK OPTION 之间的不同在于当一行被插入或修改时，有多个从属视图的检索条件被检查。

WITH LOCAL CHECK OPTION 规定在插入或更新一行时，仅检索有 WITH CASCADED OPTION 或 WITH LOCAL CHECK OPTION 的从属视图的检索条件，WITH CASCADED OPTION 规定插入或修改一行时所有从属视图的检索条件都被检查。

6.3.2.1 例子

使用下面的表或视图：

```
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%'
```

根据 INSERT 或 UPDATE 对哪个视图进行操作，确定检查不同的检索条件。

如果操作 V1，无条件被检查，因为 V1 没有规定 WITH CHECK OPTION。

如果操作 V2：

－COL1 必须以 Z 结尾，但不能以 A 开头。这是因为检查是 LOCAL。而视图 V1 没有规定检查选项。

如果操作 V3：

—COL1 必须以 Z 结尾，但不必以 A 开头，V3 没有规定检查选项。这样，它自己的检索条件不必满足，但 V2 的检索条件必须检查，因为 V3 是用 V2 定义的，而 V2 有检查选项。

如果操作 V4：

－COL1 必须以‘AB’开始，以‘YZ’结束。因为 V4 规定了 WITH CASCADED OPTION，从属 V4 的每个视图的每个检索条件必须被检查。

如果操作 V5：

－COL1 必须开始于‘AB’但不必是‘ABC’。这是因为 V5 没规定检查选项。

因此它自己的检索条件不需要被检查。然而，因为 V5 根据 V4 定义，V4 有一个串行检查选项，对 V4，V3，V2，V1 的检索条件必须被检查，这就是 COL1 必须以‘AB’开始，以‘YZ’结尾。

如果用 WITH CASCADED OPTION 生成 V5，对 V5 的操作就意味着 COL1 以‘ABC’开始，以‘YZ’结尾，WITH CASCADED OPTION 增加了一个额外的要求即第三个字母必须为‘C’。

6.4 AS/400 DB2 触发器支持

触发器是一系列动作，在对物理数据库文件执行修改操作时可以自动运行。在下面的论述中，一个表是一个物理文件。修改操作可以是插入，修改或删除的高级语言语句或者是 SQL 的 INSERT，UPDATE 或 DELETE 语句。触发器对于使用商业规则，输入数据的有效性检查和整理审计跟踪等任务是很有用的。

AS/400 DB2，包含一组触发器动作的程序能用它支持的高级语言定义。触发器程序可以有 SQL 语言嵌入其中。要使用触发器支持，必须生成一个触发程序，用 ADDPFTRG CL 命令把它加到物理文件中去，要把触发器加到文件中，必须做：

标识物理文件

标识操作类型

标识完成设计动作的程序

没有 SQL 语句与触发程序中的物理文件相关，在触发程序中的只能是嵌入的 SQL 语句，SQL 的插入、修改或删除能启动触发器。

一旦触发程序和物理文件相关，当对物理文件或表或在其上建立的逻辑文件或视图做修改操作时，系统触发支持调用触发程序。

每一个修改操作能在修改之前或之后调用触发器，这样一个物理文件最多能和六个触发器相关：

删除前触发器	删除后触发器
插入前触发器	插入后触发器
更新前触发器	更新后触发器

6.4.1 触发器样本

触发器程序的例子如下，它由 ILE C 编写，嵌入 SQL 语句。

更多的触发器的例子请看 AS/400 DB2 数据库编程。

```
#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/msghandl"     /* User defined message handler    */
/*****************************************************************/
/* 这是一个触发程序，在对雇员表进行修改时调用它。如果雇员的    */
/* 佣金比最大佣金还多，那么触发程序将增加雇员的工资的1.04%，*/
/* 然后插入RAISE表中。雇员的记录信息从输入参数传给触发程序。*/
/*****************************************************************/
```

```
Qdb_Trigger_Buffer_t *hstruct;
char *datapt;

   /*****************************************************/
   /* 用来存放雇员旧的或传给触发程序的新记录的      */
   /* EMPLOYEE 记录结构。                               */
   /* 注：必须保证所有的数字字段要调准在                */
   /*      WITH  CASCADED  OPTION 中的 4 字节边界，     */
   /*      使用压缩的结构或填充符来做到字节边界调       */
   /*      整。                                         */
   /*****************************************************/

_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;

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;

   /*******************************************************/
   /* 开始监控例外。                                      */
   /*******************************************************/

  _FEEDBACK fc;
  _HDLR_ENTRY hdlr = main_handler;
                            /***************************************/
                            /* 使例外处理活动.                     */
                            /***************************************/
  CEEHDLR(&hdlr, NULL, &fc);
                            /***************************************/
                            /* 保证例外处理程序正确                */
                            /***************************************/
  if (fc.MsgNo != CEE0000)
    {
      printf("Failed to register exception handler.\n");
      exit(99);
    };

   /*******************************************************/
   /* 把数据从触发器缓冲区移到引用的本地结构。            */
   /*******************************************************/

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);

EXEC SQL WHENEVER SQLERROR  GO TO ERR_EXIT;
```

```
   /*******************************************************/
   /* 根据触发器缓冲区中的输入参数设置交易分隔级与 */
   /* 应用程序相同。                                      */
   /*******************************************************/

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;
  }
}

 /*******************************************************/
 /* 如果雇员的佣金比最大佣金还多，那么把雇员的工 */
 /* 资增加1.04%，插入RAISE表中。                        */
 /*******************************************************/

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  */

/*****************************************************************/
/*   INCLUDE NAME : MSGHAND1                                     */
/*                                                               */
/*   说明：信息处理通知程序在缓冲程序中有错。                    */
/*   注：信息处理是用户定义的例程。                              */
/*****************************************************************/
#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           */
/*****************************************************************/
/********  例外处理功能的开始。                                  */
/*****************************************************************/
  void main_handler(_FEEDBACK *cond, _POINTER *token, _INT4 *rc,
                    _FEEDBACK *new)
   {
                          /*************************************/
                          /* 为调用QMHSNDPM初始化变量，用户必*/
                          /* 须生成一个信息文件及定义信息ID与*/
                          /* 下面的数据匹配。                  */
                          /*************************************/
  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];

                          /****************************************/
                          /* 为QMHSNDPM说明错误码结构。           */
                          /****************************************/
struct error_code {
  int bytes_provided;
  int bytes_available;
  char message_id[7];
} error_code;
error_code.bytes_provided = 15;
                          /****************************************/
                          /* 设置错误处理来重新开始，且把标  */
                          /* 志最后一个逃逸的信息为处理过的  */
                          /* 信息。                               */
                          /****************************************/
*rc = CEE_HDLR_RESUME;
                          /****************************************/
                          /* 发送自己的逃逸信息。                 */
                          /****************************************/
QMHSNDPM(message_id,
        &message_file,
        &message_data,
         message_len,
         message_type,
         message_q,
         pgm_stack_cnt,
        &message_key,
        &error_code );
                          /****************************************/
                          /* 检查调用QMHSNDPM是否正确完成。   */
                          /****************************************/
if (error_code.bytes_available != 0)
    {
      printf("Error in QMHOVPM : %s\n", error_code.message_id);
    }
}
```

图 6—1 样板触发器程序

第七章　　存储过程

AS/400　DB2　SQL 存储过程支持提供了定义一个 SQL 应用程序和通过 SQL 语句启动程序的一个方法。存储过程能够在分布和非分布的 AS/400　DB2　SQL 语言应用程序中使用。使用存储过程最大的一个优点是，对于分布式应用程序，在申请或者客户的应用程序中执行一个 CALL 语句就能在应用程序服务器上完成大量的工作。

可以定义一个 SQL 例程或者外部过程。一个外部过程是被高级语言程序支持的（除了系统 36*程序和过程）或者是被 REXX 过程支持。过程不是必须包含 SQL 语句，但它可以包含 SQL 语句。一个 SQL 的过程可以完全用 SQL 语言定义，也能是包含包括 SQL 控制语句的 SQL 语句。

编码存储过程要求用户理解下面内容：

用 CREATE　PROCEDURE 语句定义存储过程

用 CALL 语句调用存储过程

参数传递规则

给激活例程的程序返回一个完成状态的方法。

可以用 CREATE　PROCEDURE 语句定义一个存储过程，它把存储过程和参数定义加到目录表 SYSPROCS 和 SYSPARMS 中。这些定义用系统中 CALL 语句访问。

下面章节介绍用来定义和激活存储过程的 SQL 语句，传给存储过程参数的信息，以及存储过程使用的例子。

7.1　定义外部过程

CREATE　PROCEDURE 语句用于定义外部过程：

为过程命名

定义参数和它们的属性

当系统调用过程时，给出有关存储过程的其它信息。

考虑下面的例子：

```
EXEC SQL CREATE PROCEDURE P1
                 (INOUT PARM1 CHAR(10))
                EXTERNAL NAME MYLIB.PROC1
                LANGUAGE C
                GENERAL WITH NULLS;
```

这个 CREATE　PROCEDURE 语句做的事情如下：

过程名为 P1

定义一个参数，它既被用作输入，又被用作输出。这个参数为 10 位长的字符字段，，能定义的参数类型为 IN，QUT 或 INOUT，参数类型用来确定什么时候参数的值从过程取得和送出。

定义响应过程的程序名，例中是 MYLIB 中的 PROC1。当过程被 CALL 语句激活时，

MYLIB PROC1 就是被调用的程序。

指出过程 P1（程序 MYLIB PROC1）是用 C 语言写的。指明语言是很重要的，因为它影响传递参数的类型，它也影响参数怎样传到过程中。（例如，对于 ILE C 过程，NUL_终端被传为字符型、图型、时期、时间和时间标记参数）。

定义 CALL 类型为 GENERAL WITH NULLS。这就表明过程的参数可能包含空值，因此很类似于一个额外的变元传递给 CALL 语句的过程，这个额外的变元是小整型的 N 维数组，N 是在 CREATE PROCEDURE 语句中说明的参数的数量。

在这个例子中，数组只包含一个元素，因为只有一个参数。

调用一个过程不必非得定义它。然而，在程序中前面的 CREATE PROCEDURE 或者是 DECLARE PROCEDURE 中没找到过程定义，那么用 CALL 语句调用过程时，就会有一定的约束和假设。例如，不能传递空的指针变元，详细内容请看 7.3.2。

7.2 定义 SQL 过程

定义 SQL 过程用 CREATE PROCEDURE 语句：

为过程命名

定义参数和它们的属性

当调用过程时，提供关于使用过程的其它信息

定义过程体，过程体是过程的可执行部分，是单独一个 SQL 语句

考虑下面的例子，输入一个雇员号和一个比率，然后修改雇员的工资：

```
EXEC SQL 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;
```

这个 CREATE PROCEDURE 语句：

命名过程为 UPDATE_SALARY_1

定义参数 EMPLOYEE_NUMBER 是输入参数，是一个长度为 6 的字符型数据，参数 RATE 为输入参数，是十进制类型。

表明该过程是一个修改 SQL 数据的 SQL 过程。

定义这个过程主体是一个 UPDATE 语句。当调用这个过程时，用传过来的 RATE 和 EMPLOYEE_NUMBER 的值执行 UPDATE 语句。

代替一个 UPDATE 语句，能用 SQL 控制语句把逻辑加到 SQL 过程中。SQL 控制语句由以下组成：

一个赋值语句

一个 CALL 语句

一个 CASE 语句

一个复合语句

一个 FOR 语句

一个 IF 语句

一个 LOOP 语句
一个 REPEAT 语句
一个 WHILE 语句

下面的例子从最后一次计算中接收到的雇员号和比率做为输入，这个过程用 CASE 语句来决定修改的相应比率和奖金：

```
EXEC SQL CREATE PROCEDURE UPDATE_SALARY_2
        (IN EMPLOYEE_NUMBER CHAR(6),
         IN RATING INT)
        LANGUAGE SQL MODIFIES SQL DATA
        CASE RATING
          WHEN 1
            UPDATE CORPDATA.EMPLOYEE
              SET SALARY = SALARY * 1.10,
              BONUS = 1000
              WHERE EMPNO = EMPLOYEE_NUMBER;
          WHEN 2
            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;
```

这个 CREATE PROCEDURE 语句做以下事情：

过程名为 UPDATE_SALARY_2

定义参数 EMPLOYEE_NUMBER 是一个输入参数，长为 6 的字符型数据，参数 RATING 也是输入参数，是整型数据。

表明过程是修改 SQL 数据的一个 SQL 过程。

定义过程主体，当调用过程时，检查输入的参数 RATING 然后执行相应的修改语句。

用复合语句可以往过程体中加入多个语句,在复合语句中,可以规定任意多个 SQL 语句。另外，也能说明 SQL 的变量，游标和处理器。

下面的例子是取输入的部门号，返回部门中全部雇员的工资总和，及部门中得到奖金的人数：

```
EXEC SQL
CREATE PROCEDURE RETURN_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);
 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;
```

这个 CREATE PROCEDURE 语句做以下事情：

该过程名为 RETURN_DEPT_SALARY

定义参数 DEPT_NUMBER 为输入参数，长度为 3 的字符型，参数 DEPT_SALARY 是输出参数，十进制类型，参数 DEPT_BONUS_CNT 是输出参数，整型数据。

表明该过程是一个读 SQL 数据的 SQL 过程。

定义过程主体：

— 说明 SQL 变量 EMPLOYEE_SALARY 和 TOTAL_SALARY 是十进制字段。

— 说明 SQL 变量 BONUS_CNT 和 END_TABLE 是整型数据，初值为零。

— 说明游标 C1 是从雇员表中选择列。

— 说明当 NOT FOUND 时的继续处理程序，它把变量 END_TABLE 置为 1。当 FETCH 没有返回行时使用它，当使用这个处理程序时,SQLCODE 和 SQLSTATE 重新初始化为零。

— 说明 SQLEXCEPTION 的一个出口程序。如果调用它，DEPT_SALARY 置空，复合语句的处理被中断。如果发生任何错误，调用这个处理程序，这时，SQLSTATE 不是‘00’，‘01’或‘02’。指示器总是传给 SQL 过程，当过程返回时，DEPT_SALAY

的指示器值为－1。如果调用处理程序，SQLCODE 和 SQLSTATE 重新初始化为零。

如果对 SQLEXCEPTION 没定义处理程序，且错误发生时没有其它的处理程序处理它，那么复合语句的执行被中断，错误返回到 SQLCA 中。象指示器一样，SQLCA 总是从 SQL 过程中返回。

— 游标 C1 包括一个 OPEN，FETCH 和 CLOSE。如果没规定游标 CLOSE，指示器就在复合语句的末尾被关闭，这是因为在 CREATE PROCEDURE 中没规定 SET RESULT SETS。

— 包括一个 WHILE 语句，它循环直到取回最后一个记录。对于取出的每一行，TOTAL_SALARY 增加，如果雇员的奖金大于零，那么 BONUS_CNT 也增加。

— 返回 DEPT_SALARY 和 DEPT_BONUS_CNT 作为输出参数。

复合语句也能分成原子形式，这样如果一个未料到的错误发生了，那么在原子语句以内的语句将被返回。当调用一个包含原子复合语句的过程时，交易必须在一个落实边界上。如果复合语句成功，整个交易被落实。

下面的例子是把部门号做输入，它保证 EMPLOYEE_BONUS 表的存在，插入该部门中得到奖金的所有雇员的姓名，过程返回得奖金的雇员的总数。

```
EXEC SQL
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 42710
             SELECT COUNT(*) INTO CNT
             FROM DATALIB.EMPLOYEE_BONUS;
          DECLARE CONTINUE HANDLER FOR 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;
              IF BONUS > 0 THEN
                SET NAME = FIRSTNME || ' ' || MIDINIT || ' '||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;
```

这个 CREATE PROCEDURE 语句做以下事情；

过程名为 CREATE_BONUS_TABLE

定义参数 DEPT_NUMBER 是输入参数，长为 3 的字符类型，参数 CNT 是输入和输出参数，是一个整型数据。

指出该过程是一个修改 SQL 数据的 SQL 过程。

定义过程的主体：

— 说明 SQL 变量 NAME 作为可变字符

— 说明对 SQLSTATE 42710（表已经存在）的处理，如果 EMPLOYEE_BONUS 表已经存在，调用处理程序，取得表中的记录数。SQLCODE 和 SQLSTATE 重新置为零，通过 FOR 语句处理继续。

— 说明对 SQLSTATE 23505（重复键字）的继续处理，如果过程试图插入表中已经存在的名，调用处理程序并减少 CNT，继续处理 INSERT 语句后的 SET 语句。

— 说明对 SQLEXCEPTION 的 UNDO 处理程序。如果调用，前面的语句被返回，CNT 被置为零，处理继续复合语句后的语句。这时，因为复合语句后没有语句，所以过程返回。

— 使用 FOR 语句说明游标 C1，从雇员表中读记录。在 FOR 语句中，选择列表中的列名做为包含从行中取回的数据的 SQL 变量名。对于每一行，把 FIRSTNME，MIDINIT 和 LASTNAME 的数据之间用一个空格连接在一起，放在 NAME 变量中，SQL 变量 NAME 和 BONUS 插入到 EMPLOYEE_BONUS 表中。因为在生成过程时，选择列表中项的数据类型是已知的，在生成过程时 FOR 语句中规定的表必须存在。

一个 SQL 变量名能被在 FOR 语句中的表名或定义它的复合语句来限定。在这个例子中，FOR_1.BONUS 是包含每一个被选择的行的 BONUS 列值的 SQL 变量。CS1.NAME 是在有起始的号 CS1 指出的复合语句中定义的变量 NAME。参数名也能用过程名来限定，

CREATE_BONUS_TABLE.DEPT_NUMBER 是过程的 CREATE_BONUS_TABLE 的 DEPT_NUBMER 参数，如果限定的 SQL 变量名用在 SQL 语句中，列名也是允许的，变量名和列名相同，那么用名字来引用列。

7.3 调用一个存储过程

用 CALL 语句调用一个存储过程。在 CALL 语句中，要规定存储过程名和一些变元。变元可以是常量、特殊寄存器或主变量。在 CALL 语句中指定的外部存储过程不需要有相应的 CREATE PROCEDURE 语句。用 SQL 过程生成的程序只能由在 CREATE PROCEDURE 语句中规定的过程名来调用。有三种不同的 CALL 语句需要加以区别，因为 AS/400 DB2 SQL 语言对每种都有不同的规则。它们是：

嵌入或动态 CALL 语句，有过程定义存在

嵌入 CALL 语句，无过程定义存在

动态 CALL 语句，无 CREATE PROCEDURE 存在。

注：这里动态是指：

动态的准备和执行的 CALL 语句

在交互环境中发出的 CALL 语句（例如通过 STRSQL 或查询管理）

在 EXECUTE IMMEDIATE 语句中执行的 CALL 语句

下面介绍各种类型。

7.3.1 使用有过程定义的 CALL 语句

这种类型的 CALL 语句从 CREATE PROCEDURE 目录定义中得到关于过程和变元属性的所有信息。下面的 PL/1 例子给出与 CREATE PROCEDURE 语句对应的 CALL 语句。

```
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);
   :
```

当调用这个 CALL 语句时，就调用 MYLIB/PROC1 程序，并传送两个变元。因为程序语言是 ILE C，第一个变元是一个 C NUL-terminated 字符串，包含主变量 HV1 的内容，长度为 11。一个 ILE C 过程的调用，假如参数说明为字符，图型，日期，时间或者是时间标记类型的变量，AS/400 DB2 SQL 语言要加一个字符给参数说明。另一个变元是指示器数组，在这种情况下，数组是一个短整型。因为在 CREATE PROCEDURE 语句中只有一个参数，这个变元包含了在过程入口处指示器变量 IND1 的内容。

因为第一个参数被声明为 INOUT，在返给用户程序之前，SQL 语言用从 MYLIB PROC1 返回的值修改主变量 HV1 和指示器变量 IND1。

注：在 CREATE PROCEDURE 和 CALL 语句中规定的过程名必须严格匹配，以便在程序的 SQL 预编译期间中建立两者之间的联接。

注：对在 CREATE PROCEDURE 和 DECLARE PROCEDURE 语句中都有的嵌入 CALL 语句，使用 DECLARE PROCEDURE 语句。

7.3.2 使用无过程定义时的 CALL 语句

无相应的 CREATE PROCEDURE 语句的静态 CALL 语句按下面的规则处理：

所有的主变量变元做为 INOUT 类型参数处理。

CALL 类型是 GENERAL。（没有指示器变元传递。）

调用的程序根据在 CALL 中规定的过程名决定，如果必要的话，做命名约定。

调用程序的语言根据从系统中取得的信息来决定。

7.3.2.1 无过程定义存在的 CALL 语句例子

下面是 PL/1 的一个例子：

```
DCL HV2 CHAR(10);
   :
EXEC SQL CALL P2 (:HV2);
   :
```

当调用 CALL 语句时，AS/400 DB2 SQL 语言根据标准的 SQL 命名约定找到程序。对于上面的例子，假定为 *SYS 命名约定，（系统命名法），且在 CRTSQLPLI 命令中没规定 DFTRDBCOL 参数。在这种情况下，用库列表查找名为 P2 的程序。因为调用类型是 GENERAL，没有额外的变元传给程序做指示器变量。

注：如果在 CALL 语句中规定了指示器变量，当 CALL 语句执行时它的值小于零，因为没有办法把指示器传给过程，因此产生错误。

假定在库列表中找到 P2 程序，主变量 HV2 的内容传给 CALL 调用的程序，P2 执行完后，从 P2 返回的变元映射回主变量中。

7.3.3 使用带有 SQLDA 的嵌入 CALL 语句

在嵌入的 CALL 语句的任何一个类型里（过程定义存在或不存在），可以传送一个 SQLDA 而不是一个参数列表，如下面的 C 语言例子所示。假定存储过程期望有二个参数，第一个为短整数类型，第二个为长度为 4 的字符类型。

```
#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 is a macro defined in the sqlda include */
 col1 = 431;
 strcpy(col2,"abc");
 strncpy(dap->sqldaid,"SQLDA   ",8);
 dap->sqldabc = bc;          /* bc set in the malloc statement above */
 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;
 ...
}
```

被调用的过程名可以存放在主变量中，主变量用在 CALL 语句中，代替硬编码的过程名。例如：

```
...
main()
{
 char proc_name[15];
 ...
 strcpy (proc_name, "MYLIB.P3");
 ...
 EXEC SQL CALL :proc_name ...;
 ...
}
```

在上面的例子中，如果 MYLIB.P3 是想要的参数，那么可以象前面的例子那样通过 USING DESCRIPTOR 子句传送参数列表或 SQLDA。

在 CALL 语句中使用包含过程名的主变量，且 CREATE PROCEDURE 目录定义存在，它将被使用。过程名不能规定做参数标识。

调用存储过程的更多的例子在本章的后面也有介绍。

7.3.4 无 CREATE PROCEDUR 时使用动态 CALL 语句

下面的规则用来处理这种动态 CALL：

所有变元做为 IN 类型的参数

CALL 的类型是 GENERAL（无指示器变元传递）

调用的程序根据在 CALL 中规定的过程名和命名约定来确定。
调用程序的语言根据从系统中取得的程序信息来确定。

7.3.4.1 无 CREATE PROCEDURE 的动态 CALL 语句的例子

下面是一个动态 CALL 的 C 语言的例子：

```
char hv3[10],string[100];
    :
strcpy(string,"CALL MYLIB.P3 ('P3 TEST')");
EXEC SQL EXECUTE IMMEDIATE :string;
    :
```

这个例子给出一个通过 EXECUTE IMMEDIATE 语句执行的动态 CALL 语句。它传递一个包含‘P3 TEST’的字符变量的参数来调用程序 MYLIB.P3。

象前面的例子一样当执行 CALL 语句传递常量时，要小心处理程序中所期望的变元的长度。如果程序 MYLIB.P3 需要一个只有 5 个字符的变元，那么在例子中规定的常量的最后两个字符将丢失。

注：由于这个原因，在 CALL 语句中使用主变量总是很安全的，这样过程的属性能准确地匹配，字符也不丢失。对于动态 SQL，如果用 PREPARE 和 EXECUTE 语句处理主变量，那么主变量能做 CALL 语句的变元。

对于在 CALL 语句中传递的数值常量，用下面的规则：

所有的整型常量作全字二进制整数传递，

所有的十进制常量做压缩十进制值传递，精度和标度是根据常量的值来确定。例如，值 123.45 用压缩十进制(5,2)传送，同样地，值 001.01 的用精度和标度分别为 5 和 2 传送。

所有浮点常量作为双精度浮点数传送。

在 CALL 动态语句中指定的特殊寄存器象下面一样传送：

CURRENT DATE：
作为 10 字节字符串以 ISO 格式传送

CURRENT TIME：
作为 8 字节字符串以 ISO 格式传送

CURRENT TIMESTAMP：
作为 26 字节字符串以 IBM SQL 格式传送

CURRENT TIMEZONE：
作为压缩十进制数以精度为 6 标度 0 传送

CURRENT SERVER：
以 18 字节变长字符串来传送

USER：
以 18 字节的变长字符串传送

7.4 存储过程参数传递的规则

CALL 语句能把变量传送到用所有支持语言编写的程序和 REXX 过程中。每种语言支持能适合它的不同类型的数据。SQL 数据类型包含在下表中的最左列，该行的其它列指出这个数据类型作为参数类型被哪种语言支持。如果列中包含了“－”，就表示那个语言不支持，一个主变量的说明指出 AS/400 DB2 SQL 语言支持这种作为参数的数据类型。说明指出怎样说明主变量能正确地接受和设置。当调用 SQL 过程时，支持所有的 SQL 数据类型，不在表中给出。

表 7－1 参数的数据类型

SQL 数据类型	C	CL	COBOL/400 和 ILE COBOL/400
短整型	短	－－	PIC S9(4) 二进制
整型	长	－－	PIC S9(9) 二进制
DECIMAL(P,S)	decimal(p,s)	TYPE(*DEC) LEN(P S)	PIC S(9)(P-S)V9(S) PACKED-DECIMAL 注：精度必须不大于 18①
NUMERIC(P,S)	－－	－－	PIC S9(P-S)V9(S) DISPLAY SIGN LEADING SEPARATE 注：精度必须不大于 18②
ERAL or FLOAT(P)	float	－－	COMP-1 注：只支持 ILE COBOL/400③
DOUBLE PRECISION or FLOAT or FLOAT(P)	double	－－	COMP－2 注：只支持 ILE COBOL/400④
CHARACTER(n)	char...[n+1]	TYPE(*CHAR) LEN(n)	PIC X(n)
VARCHAR(n)	char...[n+1]	－－	变长字符串 注：只支持 ILE COBOL/400⑤
VARCHAR(n) FOR BITDATA	VARCHAR Structured Form(see c chapter)	－－	变长字符串 注：只支持 ILE COBOL/400⑥
GRAPHIC(n)	Wchar_t... [n+1]	－－	PIC G(n) DISPLAY-1 or PICN(n) 注：只支持 ILE COBOL/400⑦
VARGARPHIC(n)	VARGRAPHIC Structured form (see c chapter)	－－	变长图形串 注：只支持 ILE COBOL/400⑧
DATE	Char...[11]	TYPE(*CHAR) LEN(10)	PIC X(10)
TIME	Char...[9]	TYPE(*CHAR) LEN(8)	PIC X(8)
TIMESTAMP	Char...[27]	TYPE(*CHAR)	PIC X(26)

		LEN(26)	
Indicator Variable	Short	－－	PIC S9(4) BINARY

表 7－2　　参数的数据类型

SQL 数据类型	FORTRAN	PL/1	REXX
SMALLINT	INTEGER*2	FIXED BIN(15)	－－
INTEGER	INTEGER*4	FIXED BIN(31)	无小数的数字串（和一个可选的前置符号）⑨
DECIMAL(P,S)	－－	FIXED DEC(P,S)	带小数的数字串（和一个可选的前置符号）
NUMERIC(P,S)	－－	－－	－－
REAL or FLOAT(P)	REAL*4	FLOAT BIN(P)	数字串，然后是一个 E（然后是可选的符号）然后是数字
DOUBLE PRECISION or FLOAT or FLOAT(P)	REAL*8	FLOAT BIN(P)	数字串，然后是一个 E（然后是可选的符号）然后是数字
CHARACTER(n)	CHARACTER*n	CHAR(n)	两个单引号内的 n 个字符的串
VARCHAR(n)	－－	CHAR(n)VAR	两个单引号内的 n 个字符的串
VARCHAR(n) FOR BIT DATA	－－	CHAR(n)VAR	两个单引号内的 n 个字符的串
GRAPHIC(n)	－－	－－	以 G’开头的串，然后 n 个双字节字符，然后是’
VARGRAPHIC(n)	－－	－－	以 G’开头的串，然后 n 个双字节字符，然后是’
DATE	CHARACTER*10	CHAR(10)	2 个单引号内的 10 个字符串
TIME	CHARACTER*8	CHAR(8)	2 个单引号内的 8 个字符串
TIMESTAMP	CHARACTER*26	CHAR(26)	2 个单引号内的 26 个字符串
Indicator Variable	INTEGER*2	FIXED BIN(15)	无小数的数值串（和一个可选的前置符号）

表 7－3　　参数的数据类型

SQL 数据类型	RPG	ILE RPG
SMALLINT	包含一个子字段的数据结构。子字段说明中的位置 43 为 B，长度必须为 2，位置 52 为 0。	数据规范表。子字段说明中的位置 40 为 B，长度必须≤4，位置 41－42 为 0
INTEGER	包含一个子字段的数据结构。子字段说明中位置 43 为 B，长度必须为 2，位置 52 为 0。	数据规范表。子字段说明中的位置 40 为 B，长度必须≤4，位置 41－42 为 0
DECIMAL(P,S)	包含一个子字段的数据结构。在子字段说明中位置 43 为 P，位置 52 为从 0 到 9，或者是一个数字型输入字段或计算结果字段	数据规范表。子字段说明中的位置 40 为 B，长度必须≤4，位置 41－42 为 0
NUMERIC(P,S)	包含一个子字段的数据结构，子字段说明中位置 43 为空格，位置 52 为 0—9	数据规范表，子字段规范表中的位置 40 为 S 或空格，位置 41—42 为 00—31。
REAL or FLOAT(P)	－－	数据规范表，位置 40 为 F，长度必须为 4
DOUBLE PRECISION or FLOAT or FLOAT(P)	－－	数据规范表，位置 40 为 F，长度必须为 8。
CHARACTER(n)	没有子字段或包含一个子字段的数据结构。子字段说明中位置 43 和 52 为空白或一个字符输入字段或一个计算结果字段	数据规范表，子字段说明的位置 40 为 A 或空白，位置 41－42 上为空白

VARCHAR(n)	－－	数据规范表，在子字段说明中位置 40 为 A 或空白，位置 41－42 为空白，在位置 44－80 为 VARYING 键字。
VARCHAR(n)FOR BITDATA	－－	数据规范表，在子字段说明中位置 40 为 A 或空白，位置 41－42 为空白，在位置 44－80 为 VARYING 键字。
GRAPHIC(n)	－－	数据规范表，子字段说明中位置 40 为 G
VARGRAPHIC(n)	－－	数据规范表，在子字段说明中位置 40 为 G，44—80 为 VARYING 键字
DATE	没有子字段的数据结构或包含一个子字段的数据结构。在子字段说明中位置 43 和 52 为空白，长度为 10，或者一个字符输入字段或一个计算结果字段	数据规范表，子字段说明中位置 40 为 D，位置 44－80 为 DATFMT(*ISO)。
TIME	没有子字段的数据结构或包含一个子字段的数据结构。在子字段说明中位置 43 和 52 为空白，长度为 10，或者一个字符输入字段或一个计算结果字段	数据规范表，子字段说明中位置 40 为 T，位置 44－80 为 TIMFMT(*ISO)。
TIMSTAMP	没有子字段的数据结构或包含一个子字段的数据结构。在子字段说明中位置 43 和 52 位置为空白，长度为 10，或者一个字符输入字段或一个计算结果字段	数据规范表，子字段说明中位置 40 为 Z。
指示器变量	包含一个子字段的数据结构，在子字段说明中位置 43 为 B，长度为 2，位置 52 为 0。	数据规范表，在子字段说明中位置 40 为 B，长度必须≤4，位置 41－42 为 00。

7.5 指示器变量和存储过程

指示器变量能用在 CALL 语句中，提供主变量给参数使用，从过程传入或传出附加的信息。指示器变量是 SQL 语言中用来指出相关主变量解释为空值的标准方法，这是它们的基本用处。

要指出一个相关的主变量包含一个空值，两字节整型的指示器变量要设为负值，带有指示器变量的 CALL 语句如下处理：

如果这个指示器变量是负的，表明为空值。用缺省值传给 CALL 的相关主变量中，指示器变量传送时不修改。

如果这个指示器变量不为负，这表明主变量包含一个非空值。在这种情况下，主变量和指示器变量传送时都不修改。

注：这些处理规则与从过程中输入或传出参数都是相同的。当指示器变量用在存储过程时，编写处理它们的正确方法是在使用相关主变量之前首先检查指示器变量的值。

下面的例子解释在 CALL 语句中怎样处理指示器变量。注意在使用相关变量之前要对指示器变量做逻辑检查，也要注意指示器变量传给过程 PROC1 的方法（作为由两个字节数组组成的第三个变元）。

假设如下定义一个过程：

```
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
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
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
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

图 7.1　在 CALL 语句中处理指示器变量

7.6　返回调用程序的完成状态

返回发出 CALL 语句的 SQL 程序的一个方法是编码一个额外的 INOUT 参数且把它设置在从过程返回之前，当调用的过程是个已有的程序时，这种做法就不总是可行的。另一方法是给调用过程的程序（操作系统程序 QSQCALL）发送一个逃逸信息，调用的程序是 QSQCALL，每种语言都有标出条件和发送出信息的方法。参考各自语言的手册决定正确的方法。发送这

个信息后，QSQCALL 返回错误放在 SQLCODE/SQLSTATE 中，内容为-443/38501。

7.7 例子

这些例子给出不同语言的 CALL 语句中的变元是怎样传给过程，也给出怎样接收变元放在过程的本地变量中。

第一个例子给出用 CREATE PROCEDURE 定义调用 ILE C 程序来调用 P1 和 P2 过程。过程 P1 是用 C 编写的有 10 个参数，过程 P2 是用 PL/1 编写的有 10 个参数。

假设如下定义两个过程：

```
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
```

7.7.1 例 1.从 ILE C 应用程序中调用过程 ILE C 和 PL/1 过程

```
/*************************************************************/
/*********** 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 */


/*****************************************************/
/* 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 ********************************/
/*****************************************************************/
```

图 7－2　CREATE PROCEDURE 和 CALL 的样板程序

```
/******** 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];

  /* ********************************************************/
  /* 接收参数放到本地变量中－－字符、日期、时间和时  */
  /* 间标记作为NUL终端串被传送—对每一个变量把变元   */
  /* 向量转换成正确的数据类型，注意变元向量能直接使  */
  /* 用，代替往本地变量中拷贝参数，这里的拷贝正是解  */
  /* 释了这种方法。                                  */
  /* ********************************************************/

  /* 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];

/*********************************************************/
/* 拷贝NUL终端串到本地变元中。注意在CREATE       */
/* PROCEDURE中的参数说明为变长字符。对于C语言，    */
```

```
/* 除非在CREATE  PROCEDURE中规定了FOR  BIT  DATA，*/
/* 变长类型都作为NUL终端串传送。                    */
/*********************************************************/
strcpy(parm7,argv[7]);

/*********************************************************/
/* 拷贝数据到本地变量中，注意日期和时间变量总是     */
/* 以ISO格式传送，这样字符串的长度是已知的。        */
/* Strcpy正好适合做这个。                            */
/*********************************************************/
strncpy(parm8,argv[8],10);

/* Copy time into local variable                        */
strncpy(parm9,argv[9],8);

/*********************************************************/
/* 拷贝时间标记到本地变量中，总是传送IBM  SQL时间 */
/* 标记格式，因此字符串的长度是已知的。              */
/*********************************************************/
strncpy(parm10,argv[10],26);

/*********************************************************/
/* 指示器数组作为短整型数组传送。对于每个传送的参 */
/* 数有一个项（这个例子中为10）。下面是对各个变量   */
/* 设置指示器的方法。                                */
/*********************************************************/
   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 ****************************/
```

图 7－3　样板过程 P1

```
/******** 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;
```

图 7－4　样板过程 P2

下面给出从 ILE　C 程序中调用 REXX 过程的例子。
假定如下定义一个过程：

```
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
```

7.7.1.1 例 2.从 C 应用程序中调用 REXX 过程

```
/************************************************************/
/********** 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;
        /* ************************************************************/
        /* 对 CALL 语句说明主变量。                                    */
        /* ************************************************************/
        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()
        {
```

```
        /* ************************************************************/
        /* 调用过程－－从CALL语句中返回的SQLCODE应为零。          */
        /* 如果SQLCODE非零，即过程检测出一个错误。                */
        /* ************************************************************/
        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   */
           :
           }
                :
       }

/****** 结束 SQL C 应用程序         ***********************************/
/*********************************************************************/

/*********************************************************************/
/****** 开始 REXX 成员 TEST/CALLSRC CALLREXX       ******************/
/*********************************************************************/
       /* REXX 源成员 TEST/CALLSRC CALLREXX                        */
       /* 注意传递外部参数给指示器数组                             */
       /* 接收下面的输入变量设置成规定的值：                       */
       /*                                                          */
       /* 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        */

/*************************************************************/
/* 分解自变量为各自的参数。                                  */
/*************************************************************/
parse arg ar1 ar2 ar3 ar4 ar5 ar6 ar7 ar8 ar9 ar10 ar11

/*************************************************************/
/* 验证接收的值                                              */
/*************************************************************/
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

/*************************************************************/
/* 完成其他必须的处理                                        */
/*************************************************************/
   :
/*************************************************************/
/* 由返回码零指出调用成功                                    */
/*************************************************************/
exit(0)

ar1tag:
say "ar1 did not match" ar1
exit(1)
```

```
        ar2tag:
        say "ar2 did not match" ar2
        exit(1)
        :
        :

/************ END OF REXX MEMBER   *******************************/
```

图 7－5　从 C 应用程序中调用 REXX 过程

第八章　动态 SQL 应用

动态 SQL 允许在程序运行过程中定义和执行 SQL 语句。提供动态 SQL 的应用程序接收字符串格式的输入（或建立）SQL 语句，它不必知道将要运行的 SQL 语句的类型。

这些应用是：

建立或接受作为输入的 SQL 语句

为运行准备 SQL 语句

运行该 SQL 语句

处理 SQL 的返回码

交互的 SQL，是动态 SQL 程序的例子。SQL 语句被交互 SQL 动态的处理和运行。

注：

1.在运行时，用动态 SQL 处理语句要优于静态 SQL，其它的处理过程很类似，象预编译，联编，然后再运行。因此，只有需要灵活的动态 SQL 的应用程序才需要使用它，其它的应用程序则应使用正常（静态）的 SQL 语句从数据库中访问数据。

2.包含 EXECUTE 或者 EXECUTE　IMMDIATE 语句和使用 FOR　READ　ONLY 子句用游标只读的程序性能会更好，这是由于用游标可成块检索行。

CRTSQLxxx 的选项 ALWBLK(*ALLREAD)隐含 FOR　READ　ONLY 说明，因此对所有游标及引用游标的删除或更新定位，不用明显编写 FOR　UPDATE　OF，隐含有 FOR　READ　ONLY 的游标将从这个列表的第二项中得到帮助。

有些动态 SQL 语句要求使用地址变量，RPG/400 程序需要 PL/1、COBOL、C 或者 ILE RPG/400 程序来帮助管理地址变量。

这章中的例子都是 PL/1 的例子。下表显示了 AS/400 的 DB2 所支持的语句，并且指出是否可以用在动态应用程序中。

表　8-1. 在动态应用中允许的 SQL 语句

SQL 语句	静态 SQL	动态 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 COLLECTION	Y	Y
CREATE INDEX	Y	Y
CREATE PROCEDURE	Y	Y
CREATE SCHEMA	N	见注 8.
CREATE TABLE	Y	Y
CREATE VIEW	Y	Y
DECLARE CURSOR	Y	见注 4.
DECLARE PROCEDURE	Y	N
DECLARE STATEMENT	Y	N
DECLARE VARIABLE	Y	N
DELETE	Y	Y
DESCRIBE	Y	见注 7.
DESCRIBE TABLE	Y	N
DISCONNECT	Y	N

DROP	Y	Y
END DECLARE SECTION	Y	N
EXECUTE	Y	见注 1.
EXECUTE IMMEDIATE	Y	见注 2.
FETCH	Y	N
GRANT	Y	Y
INCLUDE	Y	N
INSERT	Y	Y
LABEL ON	Y	Y
LOCK TABLE	Y	Y
OPEN	Y	N
PREPARE	Y	见注 3.
RELEASE	Y	N
RENAME	Y	Y
REVOKE	Y	Y
RLOOBACK	Y	Y
SELECT INTO	Y	见注 5.
SELECT statment	Y	见注 6.
SET CONNECTION	Y	N
SET OPTION	Y	见注 9.
SET RESULT SETS	Y	N
SET TRANSACTION	Y	Y
UPDATE	Y	Y
WHENEVER	Y	N

注：

1. 不可准备，但可运行准备好的 SQL 语句。必须在使用 EXECUTE 语句之前用 PREPARE 语句准备。SQL 语句例子请看 8.2.2。
2. 不可准备，但可用没有任何?(问号)参数标志的动态语句字符串 EXECUTE IMMEDIATE 语句可在程序运行期间动态的准备和运行语句字符。例子请看 8.2。
3. 不可准备，但可在运行前用来分析语法、优化、设置动态的 SELECT 语句。例子请看 8.2。
4. 不可准备，但可在运行前定义联系动态 SELECT 语句的游标。
5. SELECT INTO 语句不能准备或用在 EXECUTE IMMEDIATE 语句中。
6. 不能与 EXECUTE 或者 EXECUTE IMMEDIATE 一起使用，但可用 OPEN 语句准备和使用。
7. 不可准备，但可用来返回准备语句的说明。
8. 只可用 RUNSQLSTM 命令来运行。
9. 只能用在运行 REXX 过程中。

8.1 设计和运行动态 SQL 应用程序

为了运行一个动态 SQL 语句，必须使用 EXECUTE 或者 EXECUTE IMMEDIATE 语句，因为动态的 SQL 语句不必在预编译时准备，所以. 必须在运行时准备。EXECUTE IMMEDIATE 语句使得 SQL 语句在程序运行时准备和动态运行。

有两种基本类型的动态 SQL 语句：SELECT 语句和 Non-SELECT 语句。Non-SELECT 语句包括 DELETE，INSERT，UPDATE 这样的语句。

典型的使用 ODBC 接口的客户服务应用程序使用动态的 SQL 来访问数据库。详细信息，请看 Client Access for Windows 3.1 ODBC 用户指南。

8.2 处理 Non-SELECT 语句

要创建一个动态的 SQL Non-SELECT 语句有以下二步：

1.确定要创建的 SQL 语句可以在动态下运行。

2.创建 SQL 语句（使用交互式 SQL 是一种简单的创建确认运行方法，详细内容请看第十七章。）

要运行动态 SQL Non-SELECT 语句：

1.用 EXECUTE IMMEDIATE 语句运行，或用 PREPARE 语句，然后用 EXECUTE 执行准备好的语句。

2.处理 SQL 产生的返回码。

下面是一个运行动态 SQL 应用程序的例子：

```
EXEC SQL
 EXECUTE IMMEDIATE  :字符串
```

8.2.1 动态 SQL 语句的 CCSID

SQL 语句是一个正规的主变量。主变量的 CCSID 用作语句正文的 CCSID。在 PL/1 中，它也可以是一个串表达式。在这种情况下，作业的 CCSID 作为语句正文的 CCSID 来使用。

动态 SQL 语句用语句正文的 CCSID 来处理。这对不同字符的影响最大。例如，(用 CCSID 500 分配为'BA'X，即是说，如果语句正文的 CCSID 是 500，SQL 认为 (分配为'BA'X，如果语句正文的 CCSID 为 65535,SQL 处理可变字符就象它们的 CCSID 为 37 一样,即把(解释为'5F'X

8.2.2 使用 PREPARE 和 EXECUTE 语句

如果 Non-SELECT 语句中包含无参数标记,那么可以用 EXECUTE IMMEDIATE 语句来动态的运行。但如果 Non-SELECT 语句中有参数标记，那么必须用 PREPARE 和 EXECUTE 来运行。

PREPARE 语句准备 Non-SELECT 语句(例如 DELETE 语句)并且给它一个你所选择的名字，如果在 CRTSQLxxx 命令中规定了 DLYPRP (*YES)，那么准备工作将延迟到第一次在 EXECUTE 或 DESCRIBE 语句中使用它时，除非在 PREPARE 语句中具体规定了 USING 子句。在这个例子中，我们称它为 S1。在语句准备好后，可以对参数标记使用不同的值，在同一个程序内运行多次。下面是 PREPARE 语句使用多次的例子：

```
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;
```

上面的例子与例行程序相似，必须知道参数标记的个数以及它们的类型，这是提供输入数据的主变量在写程序时说明的。

注：当应用程序一旦终结，所有与应用程序相联系的准备语句将被撤销。它们之间的连接由 CONNECT（类型 1）语句，DISCONNECT 语句或在成功的落实之后的一个 RELEASE 来结束。

8.3 处理 SELECT 语句和使用 SQLDA

有两个基本的 SELECT 语句类型：固定列表，可变列表。

处理一个固定列表的 SELECT 语句，不需要 SQLDA。

处理一个可变列表的 SELECT 语句，首先要说明 SQLDA 结构。SQLDA 是一个控制模块，用来从应用程序传送主变量输入值到 SQL 中，及从 SQL 接收输出值。另外，有关 SELETE 列表表达式信息可以用 PREPARE 或 DESCRIBE 语句返回。

8.3.1 固定列表的 SELECT 语句

在动态 SQL 中，固定列表 SELECT 语句是为了检索那些预先知道数值和类型的数据的语句。使用这些语句时，可以预料并定义主变量来容纳取出来的数据，因此 SQLDA 是不必要的。每次成功的 FETCH 都返回与上一次相同数量的值，并且这些数据有与上次 FETCH 返回值相同的数据格式。可以规定主变量，适用于任何 SQL 应用程序。

可以在任何 SQL 支持的应用程序中使用固定列表的 SELECT 语句。

为了动态运行固定列表 SELECT 语句，应用程序必须：

1. 把输入的 SQL 语句放入主变量中。
2. 发出一个 PREPARE 语句来验证动态 SQL 语句并把它放到一个可运行的格式中。如果在 CRTSQLxxx 命令中规定了 DLYPRP(*YES)，那么准备将推迟到该语句第一次用在 EXECUTE 或 DESCRIBE 语句中，除非在 PREPARE 语句中规定了 USING 子句。
3. 为语句名说明一个游标。
4. 打开游标。
5. 取出一行放在变量的固定列表中（而不是放入描述符区）。
6. 在到达数据末时，关闭游标。
7. 处理产生的返回代码。

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.
```

注：记住，在这种情况下，由于 SELECT 语句总是返回与先前运行固定列表 SELECT 相同的数值与类型，不必一定使用 SQLDA。

8.3.2 可变列表的 SELECT 语句

在动态 SQL 中，可变列表的 SELECT 是一些返回结果列的数量和格式都是不可确定的。这就是说，不知道需要多少变量或数据的类型。因此，不能提前定义容纳返回列的主变量。

注：在 REXX 中是用步骤 5b，6，7。

如果应用程序接受可变列表 SELECT 语句，那么程序必须：

1. 把输入 SQL 语句放入主变量中。
2. 用一个 PREPARE 语句来检查动态 SQL 语句并把它放到一个可运行的格式中。如果在 CRTSQLxxx 命令中规定了 DLYPRP(*YES)，那么准备将推迟到该语句第一次用在 EXECUTE 或 DESCRIBE 时，除非在 PREPARE 中规定了 USING 子句。
3. 为语句名说明一个游标。
4. 打开游标（在 3 中说明的），它包括动态 SELECT 语句的名字。
5. 用一个 DESCRIBE 语句，申请从 SQL 中获得关于结果表中每列的类型和大小的信息。

注：

   a. 也可以使用带 INTO 子句的 PREPARE 语句，用一个语句来完成 PREPARE 和 DESCRIBE 的功能。
   b. 如果 SQLDA 不够大来容纳每一个取回到的列描述信息，那么程序必须决定需要多大空间、取得存储空间、创建新的 SQLDA，并重新发出 DESCRIBE 语句。

6. 赋值包含一行取回数据所需的存储空间。
7. 把存储地址放入 SQLDA，告诉 SQL 把每个检索来的数据放在哪儿。

8.FETCH 一行。

9.当发现数据结束时，关闭游标。

10.处理产生的 SQL 返回码。

8.3.3 SQL 描述区

可以使用 SQLDA 在 SQL 和应用程序之间传递 SQL 语句的信息。

SQLDA 是一些变量的收集地，在运行 DESCRIBE 和 DESCRIBE TABLE 语句时需要它。在 PREPARE、OPEN、FETCH、CALL 和 EXECUTE 语句中也用到它。SQLDA 与动态 SQL 一起使用，它可以在 DESCRIBE 语句中使用，用主变量的地址修改，然后在 FETCH 语句中重用。

在 SQLDA 中信息的意义取决于它的使用。在 PREPARE 和 DESCRIBE 语句中，SQLDA 提供给应用程序关于 PREPARE 语句的信息。在 DESCRIBE TABLE 中，SQLDA 提供给应用程序在表或视图中列的信息。在 OPEN EXECUTE，CALL 和 FETCH 中，SQLDA 提供有关主变量的信息。

SQLDA 对 DESCRIBE，DESCRIBE TABLE 来说是必须的。

SQLDA 对 EXECUTE，FETCH，OPEN，PREPARE，CALL 来说是可选的。

如果应用程序允许同一时间打开几个游标，你可以编写几个 SQLDA，每一个动态 SELECT 语句有一个 SQLDA。详细内容请看 AS/400 DB2 SQL 参考。

SQLDA 也能用于 C，COBOL，PL/1，REXX 和 RPG。因为 RPG/400 不提供设置指针的方法，所以必须通过 PL/1，C，COBOL 或者 ILE RPG/400 程序在外部设置指针。由于这个区域必须由 PL/1，C，COBOL 或者 RPG/400 程序说明，所以程序必须调用 RPG/400。

8.3.4 SQLDA 格式

SQLDA 由四个变量，接着叫做 SQLVAR 的六个可变顺序的任意数字组成。

注：REXX 中的 SQLDA 有所不同。详细信息，请看本书原文的第十五章。

当一个 SQLDA 用在 OPEN，FETCH，CALL 和 EXECUTE 中时，每一个 SQLVAR 描述一个主变量。

SQLDA 的变量如下（变量名在 C 中要用小写）：

SQLDAID 它用来存储转储，是 8 位字符串，值为‘SQLDA’，用在 PREPARE 或 DESCRIBE 语句中的 SQLDA 后，它不能用于 FETCH，OPEN，CALL 或 EXECUTE。

SQLDABC 它指出 SQLDA 的长度。它是 4 字节整数，值为 SQLN*LENGTH(SQLVAR)＋16，用于 PREPARE 或 DESCRIBE 语句。在被 FETCH，OPEN，CALL，EXECUTE 使用之前，SQLDABC 的值必须等于或大于 SQLN*LENGTH(SQLVAR)+16。它不能用在 REXX 中。

SQLN SQLN 是一个 2 字节的整数，规定 SQLVAR 出现的总数。必须在被任何 SQL 使用之前设置为大于或等于零的值。SQLN 不能在 REXX 中使用。

SQLD SQLD 是一个 2 字节的整数，它规定 SQLVAR 发生的个数；也就是，被 SQLDA 描述的主变量的个数，这个字段是 SQL 在 DESCRIBE 和 PREPARE 语句中设置的。在其它语句中，在使用前必须把这个字段赋于一个大于等于零且小于等于 SQLN 的值。

SQLVAR SQLVAR 的变量是 SQLTYPE，SQLLEN，SQLRES，SQLDATA，SQLIND 和 SQLNAME。这些变量是 SQL 用 DESCIRBE 或 PREPARE 语句设置的。在其它语句中，必须在使用之前设置，这些变量的定义如下：

SQLTYPE SQLTYPE 是一个 2 字节整数，规定在下表给出的主变量的数据类型。
奇数值的 SQLTYPE 表示主变量有由 SQLIND 分配的相关的指示器变量地址。

SQLLEN SQLLEN 是一个 2 字节的整数变量，它描述在图 10-2 中给出的主变量的长度属性。

表 8-2 PREPARE、DESCRIBE、FETCH、OPEN、CALL 和 EXECUTE 使用的 SQLTYPET 和 SQLLEN 的值

SQLTYPE	For PREPARE and DESCRIBE		For FETCH, OPEN, CALL, and EXECUTE	
	COLUMN DATA TYPE	SQLLEN	HOST VARIABLE DATA TYPE	SQLLEN
384/385	日期	10	定长字符串，日期表达式	主变量的长度属性
388389	时间	8	定长字符串，时间表达式	主变量的长度属性
392/393	时间标记	26	定长字符串，时间标记表达式	主变量的长度属性
400/401	N/A	N/A	NUL-终端图形串	主变量的长度属性
448/449	变长字符串	列的长度属性	变长字符串	主变量的长度属性
452/453	定长字符串	列的长度属性	定长字符串	主变量的长度属性
456/457	长变长字符串	列的长度属性	长变长字符串	主变量的长度属性
460/461	N/A	N/A	NUL-终端字符串	主变量的长度属性
464/465	变长图形串	列的长度属性	变长图形串	主变量的长度属性
468/469	定长图形串	列的长度属性	定长图形串	主变量的长度属性
472/473	长变长图形串	列的长度属性	长图形串	主变量的长度属性
476/477	N/A	N/A	PASCAL L-串	主变量的长度属性
480/481	浮点	单精度为 4，双精度为 8	浮点	单精度为 4，双精度为 8
484/485	压缩十进制	字节 1 为精度，字节 2 为范围	压缩十进制	字节 1 为精度，字节 2 为范围
488/489	区位十进制	字节 1 为精度，字节 2 为范围	区位十进制	字节 1 为精度，字节 2 为范围
496/497	大整数	4 (1)	大整数	4
500/501	小整数	2 (1)	小整数	2
504/505	N/A	N/A	DISPLAY SIGN LEADING SEPARATE	字节 1 为精度，字节 2 为范围

SQLRES SQLRES 是 12 字节的保留区域，用来做边界调整。注意，在 OS/400 中，指针在四字边界上。SQLRES 不能用在 REXX 中。

SQLDATA SQLDATA 是一个 16 字节的指针变量，在 SQLDA 用在 OPEN，FETCH，CALL 和 EXCDTE 时，用来规定主变量的地址。

当 SQLDA 用于 PREPARE 和 DESCRIBE 语句时，这个区域用以下信息代替：存储在 SQLDATA 的第 3、第 4 字节的字符、日期、时间、时间标记和图形字段的 CCSID。对于 BIT 数据，CCSID 是 65535，在 XREXX 中，返回的 CCSID 放在 SQLCCSID 中。

SQLIND SQLIND 是一个 16 字节的指针，SQLDA 用在 OPEN，FETCH，CALL 和 EXECUTE 时，用来规定一个短整数的主变量，用作一个空值或非空值的说明。负值说明空，非负值说明为非空。这个指针只有在 SQLTYPE 中包括奇数值才用。

当 SQLDA 在 PREPARE 和 DESCRIBE 中使用时，保留这个区以后使用。

SQLNAME SQLNAME 是一个最大长度为 30 的变长字符变量，它包含在 PREPARE 或

DESCRIBE 之后选择的列、表或系统列的名字。在 OPEN，FETCH，EXECUTE 或 CALL 中，也用它传送字符串的 CCSID。CCSIDS 可以用字符，图形，日期，时间和时间标记主变量传送。

在一个输入 SQLDA 的 SQLVAR 数组入口的 SQLNAME 字段可以用来设置 CCSID。

数据类型	子类型	SQLNAME 长度	SQLNAME 字节 1 & 2	SQLNAME 字节 3 & 4
字符	SBCS	8	X'0000'	CCSID
字符	MIXED	8	X'0000'	CCSID
字符	BIT	8	X'0000'	X'FFFF'
GRAPHIC	不用	8	X'0000'	CCSID
其他数据类型	不用	不用	不用	不用

注：应记住 SQLNAME 只是用来替换 CCSID，使用缺省值的应用程序不必传递 CCSID 信息，如果没有传送 CCSID，那么使用作业的缺省 CCSID。

图形主变量的缺省值是作业 CCSID 的相关双字节 CCSID，如果不存在相关的双字节 CCSID，那么使用 65535。

(1)在 SQLDA 中二进制数长度可为 2 或 4，或者第一字节为精度而第二字节为范围，如果第一字节大于 X'00'，那么它指出精度和范围。

8.3.5 分配 SQLDA 存储的 SELECT 语句例子

SELECT 语句可以从显示工作站或主变量中读数据，也可写在应用程序中。下面的例子是用 SELECT 语句从显示工作站读的例子。

显示 154

注：SELECT 语句没有 INTO 子句，动态 SELECT 语句仅返回一行，一定没有 INTO 子句。

当读语句时，分配一个主变量给它，然后用 PREPARE 语句处理主变量（例如，名为 DSTRING）：

```
EXEC SQL
PREPARE S1 FROM :DSTRING;
```

8.3.5.1 分配存储

可以为 SQLDA 分配存储。请求存储的技术依赖于语言。SQLDA 必须定位在 16 字节的边界上。SQLDA 有一个定长的 16 字节长的标题，在标题之后是变长的数组部分（SQLVAR），它的每个元素为 80 字节。所要分配的存储量的多少依赖于要在 SQLVAR 数组有多少个元素。你所选择的每一列必须对应于 SQLVAR 数组的一个元素。因此，在 SELECT 语句中给出的列数决定了分配多少个 SQLVAR 数组元素。由于 SELECT 语句是在运行时规定的，因此，要访问多少列是很重要的，所以，必须估计列的数量。在这个例子中，一个 SELECT 语句要访问不超过 20 列。在选择列表中的每一项都必须在 SQLVAR 有相应的对应元素，所以 SQLVAR 数组有 20 维（SQLDA 的大小是 20×80，或 1600，加上 16 总数为 1616 字节。）

在 SQLDA 的 SQLN 字段中，为 SQLDA 分配了估计的足够空间之后，设置一个与 SQLVAR 数组元素数相同的初值。在下面的例子中，把 SQLN 设为 20：

在四字的边界上给 SQLDA 赋值 1616 字节的空间，SQLN＝20。

在分配了存储之后，就可以用 DESCRIBE 语句：

```
EXEC SQL
DESCRIBE S1 INTO :SQLDA;
```

在运行 DESCRIBE 语句时，SQL 把值放入提供选择列表信息的 SQLDA 中，下面的图 8－1 给出在 DESCRIBE 运行之后 SQLDA 的内容。

PIC19

图表 8－1 DESCRIBE 运行之后 SQLDA 的内容

SQLDAID 是一个运行 DESCRIBE 时由 SQL 初始的标识字段，SQLDABC 是 SQLDA 的字节总数或大小。现在可以忽略这些。

下面是对 S1 运行 SELECT 语句的例子：

```
  SELECT WORKDEPT, PHONENO
    FROM CORPDATA.EMPLOYEE
    WHERE LASTNAME = 'PARKER'
```

如果 SQLDA 不足够大来容纳描述的 SQLVAR 元素，可以修改 SQLN 的值。例如，要让 SELECT 语句包含 27 个选择列表表达式，而不是 20 或者比估计的少，由于 SQLDA 仅分配了 SQLVAR 20 个元素的空间，SQL 不能描述选择列表，原因是 SQLVAR 有足够多的元素。SQL 把 SQLD 设置为 SELECT 语句规定的列数，忽略余下的空间结构。因此，在 DESCRIBE 之后，要比较 SQLN 和 SQLD。如果 SQLD 的值大于 SQLN，那么根据 SQLD 中的值，分配一个较大的 SQLDA，如下所示：

```
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;
```

如果在 non-SELECT 语句中使用 DESCRIBE，SQL 把 SQLD 置为零。因此，如果程序即可以处理 SELECT 又可以处理 non-SELECT，要描述每个语句（在准备之后）看是否是 SELECT 语句，这个简单的例程是按仅处理 SELECT 语句编写的，不检查 SQLD。

现在程序要分析 SQLVAR 的元素，记住每个元素描述一个选择列表表达式。考虑处理的 SELECT 语句：

```
   SELECT WORKDEPT, PHONENO
```

```
FROM CORPDATA.EMPLOYEE
WHERE LASTNAME = 'PARKER'
```

选择列表的第一项是WORKDEPT。在这节开始时，我们指出每个SQLVAR元素包括SQLTYPE，SQLLEN，SQLRES，SQLDATA，SQLIND，SQLNAME字段。在SQLTYPE字段，SQL返回了一个代码，它描述了表达式的数据类型以及是否可以接受空值。

例如，SQL把SQLVAR元素1中的SQLTYPE设为453。（见图8－1），它规定了WORKDEPT是一个定长的字符串列并且允许在列中使用空值。

SQL把列的长度赋给SQLLEN，由于WORKDEPT的数据类型是字符，所以SQL把SQLLEN设为与字符串长度相等。对于WORKDEPT，它的长度为3，因此，当SELECT语句后来运行时，一个存储空间必需足够大来容纳CHAR(3)的字符串。

由于WORKDEPT的数据类型是CHAR FOR SBCS DATA，所以SQLDATA的前4个字节被设置成字符列的CCSID(见图8-1)。在SQLVAR元素的最后字段是一个变长的字符串，叫做SQLNAME。SQLNAME的头二个字节包含字符数据的长度。字符数据本身通常是在SELECT语句中使用的列名（在上例中的WORKDEPT），而在选择列没有命名时是一个例外，例如函数（例如，SUM(SALARY)），表达式（例如，A＋B-C）和常量。在这种情况下，SQLNAME是一个空串。SQLNAME也可以包括一个标号，而不是名字。PREPARE和DESCRIBE语句的一个参数是USING子句。可以用这种方法规定：

```
EXEC SQL
DESCRIBE S1 INTO:SQLDA
USING LABELS;
```

如果规定NAMES（或省略整个USING参数），那么只有列名放在SQLNAME字段。如果规定SYSTEM NAME，那么只有系统列名放在SQLNAME字段。如果规定LABELS，只有与SQL语句中列出与列相关的标号放在这里。如果规定ANY，有标号的列会把标号放入SQLNAME中；否则，放列名。如果规定BOTH，那么名字和标号及相应长度都放入此字段。在规定BOTH时，要记住用双倍的SQLVAR数组长度，这是因为包括两倍的元素个数。如果规定ALL，那么列名，标号，系统名及其相应的长度被放在该字段，此时，要三倍的SQLVAR数组大小，并且：

名字和标号以其相应的长度放在此字段。

SQLVAR数组的大小要3倍。

在例子中，SQLVAR第二元素包含选择语句的第二列PHONENO的信息。SQLTYPE为453，表示PHONEND是一个CHAR列。由于长度是4的字符数据，SQL把SQLLEN设为4。

在分析完DESCRIBE的结果之后，要为放SELECT语句的产生结果的变量分配存储空间。对于WORKDEPT，必须分配长度为3的字符空间；对PHONENO，必须分配长度为4的字符空间。

分配好存贮空间后，必须使SQLDATA和SQLIND指向适当的区域。对每个SQLVAR元素，SQLDATA指向放结果的地方，SQLIND指向放空指示器的地方。下面的图给出现在的结构：

PIC20

这就是迄今为止所做的工作：

```
This is what was done so far:
```

```
EXEC SQL
INCLUDE SQLDA;
/*Read a statement into the DSTRING varying-length
character string host variable.*/
EXEC SQL
PREPARE S1 FROM :DSTRING;
/*Allocate an SQLDA of 1616 bytes.*/
SQLN =20;
EXEC SQL
DESCRIBE S1 INTO :SQLDA;
/*Analyze the results of the DESCRIBE.*/
/*Allocate storage to hold one row of
the result table.*/
/*Set SQLDATA and SQLIND for each column
of the result table.*/
```

8.3.6 使用游标

现在已准备检索 SELECT 语句的结果了。动态的定义 SELECT 语句必须没有 INTO 子句，因此，所有动态定义 SELECT 语句必须使用游标。特别格式的 DECLARE，OPEN，FETCH 也可用于动态定义的 SELECT 语句。

DECLARE 语句的例子是：

```
EXEC SQL DECLARE C1 CURSOR FOR S1;
```

正如你所看到的，不同的只是用准备的 SELECT 语句的名字，替代了 SELECT 语句本身。对结果行的实际检索是按下面给出的语句进行的：

```
EXEC SQL
OPEN C1;
EXEC SQL
FETCH C1 USING DESCRIPTOR :SQLDA;
DO WHILE (SQLCODE = 0);
/*Display ... the results pointed to by SQLDATA*/
END;
/*Display ('END OF LIST')*/
EXEC SQL
CLOSE C1;
```

打开游标，核定结果表。注意这里 SELECT 并不需要输入主变量，选择的结果行用 FETCH 返回。在 FETCH 语句中，没有列出输出主变量，而用 FETCH 语句通知 SQL 把结果返回到叫做 SQLDA 的描述符所描述的区域中。DESCRIBE 语句设置同样的 SQLDA 作为 SELECT 语句的输出用。实际上，结果返回到被 SQLVAR 元素 SQLDATA 和 SQLIND 所指的存储区域。下图给出在执行 FETCH 语句后的结构：

PIC21

SQLIND 指向的 SMALLINT 的意义与其它指示器变量是一样的：

0	表示返回值非空
<0	表示返回值为空
>0	表示返回值由于存储空间不够大被截断，指示器变量包含截断之前的长度。

注：除非规定了 HOLD，否则动态游标将在 COMMIT 和 ROLLBACK 期间关闭。

8.3.7 使用参数标记

在我们使用的例子中，动态运行的 SELECT 语句的 WHERE 子句中有可预见的参数（输入主变量）。在此例中，它是：

WHERE LASTNAME = 'PARKER'

如果想几次运行同样的 SELECT 语句,并且使用不同的 LASTNAME 的值,可以使用 PREPARE 或 EXECUTE（在 8.2.2 中的使用 PREPARE 和 EXECUTE 语句），象下面这样：

```
SELECT WORKDEPT, PHONENO FROM CORPDATA.EMPLOYEE WHERE LASTNAME = ?
```

当参数是不可预见的，应用程序直到运行的时候才能知道参数的类型和数量，在应用程序运行时才可以安排接受这些信息，用在 OPEN 语句中的 USING DESCRIPTOR 语句，要替换包含在 SELECT 语句中的 WHERE 子句中参数标记主变量中的值。

为了编写这样程序，需要使用带 USING DESCRIPTOR 子句的 OPEN 语句。这个 SQL 语句，不仅用来打开游标，而且用相应的主变量的值替换每一个参数标记。在这个语句中规定的描述符号名必须标识一个包含这些主变量的有效描述的 SQLDA。这个 SQLDA 不象先前介绍的那样，不用来把信息返回到选择列表的一部分的数据项。这就是说，它不用作 DESCRIBE 语句的输出，而是作为 OPEN 语句输入。它提供信息给主变量用来代替在 SELECT 语句的 WHERE 子句中的参数标记。它从应用程序中得到这些信息，这些应用程序设计成把适当的值放在 SQLDA 需要的字段中，然后读 SQLDA 作为 SQL 信息源，用主变量数据替换参数标记。

当使用 SQLDA 来为有 UNIGN DESCRIPTOR 子句的 OPEN 语句做输入时，并不是所有的字段都被填满。SQLDAID，SQLRES 和 SQLNAME 可以为空白（如果需要特殊的 CCSID 可以设置 SQLNAME，在 REXX 为 SQLCCSID)。因此，当这个方法使用主变量来替换参数标记时，应该决定：

有多少？参数标记

这些参数标记的属性和数据类型是什么（SQLTYPE，SQLLEN 和 SQLNAME）

是否需要指示器变量

另外，如果例程是处理 SELECT 语句和 nonSELECT 语句，也要决定语句的种类。(或者，可以编写程序来寻找 SELECT 键字)。

如果应用程序使用参数标记，那么程序必须：

1.把语句读入变长的字符串主变量 DSTRING 中。

2.决定？参数标记的个数。

3.分配 SQLDA 空间。

4.把 SQLN 和 SQLD 设置为？参数标记的个数。

5.把 SQLDABC 设为等于 SQLN*LEGTH(SQLVAR)＋16

6.对于每一个？参数标记：

a.确定数据类型、长度和指示器。

b.设置 SQLTYPE 和 SQLLEN。

c.分配存储空间来放输入值（？值）。

d.设置这些值。

e.为每个？参数标记设置 SQLDATA 和 SQLIND。

f.如果用字符变量，并且它们用的 CCSID 不是作业缺省的 CCSID，那么设置相应的 SQLNAME（在 REXX 中为 SQLCCSID）。

g.如果使用图形变量，并且它们的 CCSID 不是作业 CCSID 相关的 DBCSCCSID，那么把 SQLNAME 设为 CCSID（在 REXX 中为 SQLCCSID）。

h.发出一个带 UNIGN DESCRIPTOR 的 OPEN 语句，来打开游标和为每个参数标记替换主变量的值。

然后，便可以正常的处理语句了。

第九章　在主语言使用 SQL 的一般概念和规则

这章讲述在主语主中使用 SQL 语句的一些基本概念和规则，其中包括：

在 SQL 语句中使用主变量

处理 SQL 错误和返回码

使用 WHENEVER 语句处理异常条件

9.1　在 SQL 语句中使用主变量

当程序检索数据时，这些值放到程序所定义的数据项中和由 SELECT　INTO 和 FETCH 语句的 INTO 子句规定的数据项中，这些数据项被称为主变量。

主变量是在程序中的用 SQL 语句规定的一个字段，通常作为列值的来源或者目的。主变量与列的数据类型必须匹配，主变量可以不做 SQL 目标，例如表或视图，但在 DESCRIBE TABLE 语句中例外。

一个主结构是一组主变量，用作一组选择值的来源或目的。一个主结构数组是用在多行的 FETCH 和成块的 INSERT 语句中的一个主结构数组。

注：在 SQL 语句中，用主变量来替代文字值，可以使应用程序更灵活的处理表或视图中不同的行。

例如，可以把主变量设为感兴趣的部门号，代替在 WHERE 子句中编写实际的部门号。

主变量通常以下面的方式用在 SQL 语句中：

1.用 WHERE 子句：可以使用一个主变量来规定检索条件中谓词的值，或者代替表达式中一个文字值。例如，已定义了一个有雇员号的 EMPID 的字段，就可以用下面的语句来检索一个雇员号为 000110 的雇员名：

```
MOVE '000110' TO EMPID.
EXEC SQL
 SELECT LASTNAME
   INTO :PGM-LASTNAME
   FROM CORPDATA.EMPLOYEE
   WHERE EMPNO = :EMPID
END-EXEC.
```

2.作为列值的接受区域（在 INTO 子句中命名）：可以用主变量来规定一个程序数据区，用它来放检索行的列值。INTO 子名命名一个或多个主变量，来容纳 SQL 返回的列值。例如，假设检索 CORPDATD.EMPLOYEE 表的 EMPNO，LASTNAME 和 WORKDEPT 列的值，可以在程序中定义主变量来接收每一列，然后用 INTO 子句为每一个主变量命名。例如：

```
EXEC SQL
 SELECT EMPNO, LASTNAME, WORKDEPT
```

```
    INTO :CBLEMPNO, :CBLNAME, :CBLDEPT
    FROM CORPDATA.EMPLOYEE
    WHERE EMPNO = :EMPID
END-EXEC.
```

在这个例子中，主变量 CBLEMPNO 接收来自 EMPNO 的值，CBLNAME 接收来自 LASTNAME 的值，而 CBLDEPT 接受来自 WORKDEPT 的值。

3.作为 SELECT 子句中的一个值：当在 SELECT 子句中规定项目列表时，不受表和视图列名的限制，程序可以返回主变量值与文字常量混合的列值。例如：

```
MOVE '000220' TO PERSON.
EXEC SQL
 SELECT "A", LASTNAME, SALARY, :RAISE,
      SALARY + :RAISE
    INTO :PROCESS, :PERSON-NAME, :EMP-SAL,
      :EMP-RAISE, :EMP-TTL
    FROM CORPDATA.EMPLOYEE
    WHERE EMPNO = :PERSON
END-EXEC.
```

它的结果是：

PROCESS	PERSON-NAME	EMP-SAL	EMP-RAISE	EMP-TTL
A	LUTZ	29840	4476	34316

4.作为 SQL 语句其它子句中的一个值：

UPDATE 语句中的 SET 子句

` INSERT 语句中的 VALUES 子句

CALL 语句

详细信息，请看 AS/400 DB2 SQL 参考。

9.1.1 赋值规则

在运行 FETCH 和 SELECT INTO 语句时，SQL 的列值被置（或赋值）给主变量。在运行 INSERT，UPDATE 和 CALL 语句时，SQL 列从主变量中得到值。所有的赋值操作遵循以下规则：

数值和字符串不匹配：

数值不能赋值给字符串列或字符串主变量

字符串不能赋值给数值列或数值型主变量

如果支持 CCSID 间转换，那么所有的字符和 DBCS 图形串要与 USC-2 图形列匹配。如果 CCSID 匹配，那么所有的图形串相匹配，所有的数值都相匹配。在必要的时候，由 SQL 执行转换。如果支持 CCSID 间转换，在赋值操作时，对所有字符和 DBCS 图形串，都要与 UCS-2 图形列相匹配。对于 CALL 语句，如果支持转换，字符和 DBCS 图形参数与 USC-2 参数将匹配。

空值不能赋值给那些没有相关的指示器变量的主变量。

不同类型的日期 / 时间是不匹配的，日期只能与日期或表示日期的字符串相匹配，时间

只能与时间或表示时间的字符串相匹配，时间标记只能与时间标记或表示时间标记的字符串相匹配。

日期只能赋值给日期列、字符列、DBCS-open 或 DBCS-either 列或变量，或字符变量(1)。插入或修改的日期列的值必须是日期或表示日期的串。

时间只能被赋值给时间列、字符列、DBCS-open 或 DBCS-either 列或变量，或字符变量。插入或修改的时间列的值必须是时间或表示时间的字符串。

时间标记只能被赋值给时间列、字符列、DBCS-open 或 DBCS-either 列或变量，或字符变量。插入和修改的时间标记列的值必须是时间标记或代表时间标记的字符串。

(1) DBCS-open 或 DBCS-either 列或变量是在主语言中说明的包括外部描述文件定义的变量。如果作业的 CCSID 指出了 MIXED 数据，或用了 DECLARE VARIABLE 语句并规定了 MIXED CCSID 或 FOR MIXED DATA 子句，则也说明了 DBCS-open 变量。

9.1.1.1 字符串赋值规则

字符串赋值规则是：

当一个字符串赋值给一列时，字符串的长度不能大于列的长度。（结尾空格通常也包括在字符串长度之内。然而，对于字符串赋值来说结尾空格不在串长度之内）。

当 MIXED 字符结果列赋值给 MIXED 列时，MIXED 字符结果列的值必须是一个有效的 MIXED 字符串。

当结果列的值赋值给主变量和结果列的串值大于主变量的长度属性时，字符串从右边按必要的字符数截断。如果发生这种情况，SQLWARN0 和 SQLWARN1（在 SQLCA 中）将被置为 W。

当结果列的值赋值给定长的主变量或当主变量的值赋值给定长的字符结果列且串中长度小于目的的长度属性，那么在串右边将加必要的空格。

由于赋值给主变量的长度小于串的长度，MIXED 字符结果串列被截断时，在串结尾的转入字符要保留。因此，结果仍是一个有效的 MIXED 字符串。

9.1.1.2 CCSIDs 规则

在字符或图形值之间赋值时，必须考虑 CCSIDs，这也包括主变量的赋值，数据库管理使用一般的系统服务来转换 SBCS 数据、DBCS 数据、MIXED 数据和图形数据。

CCSIDs 的规则如下：

如果源 CCSID 与目的 CCSID 相匹配，不需转换可赋值。

如果源或目的子类型是 BIT，不需转换可赋值。

当值是空或是空串时，不需转换可赋值。

如果在 CCSID 之间没有定义转换，不能赋值且将发出错误信息。

当定义了转换并且是必需的，那么在执行赋值前，源值将转换为目的的 CCSID。

有关 CCSID 信息，请看 International Application Development 一书。

9.1.1.3 数值赋值的规则

数值赋值的规则是：

数值的整数部分在把它转换为浮点数时可能会改变，单精度浮点字段只能有七位十进数字，多于 7 个数字的整数部分将进行舍入。双精度浮点字段只能有 16 位十进数字，多于 16 个数字的整数部分将进行舍入。

数值的整数部分永远不能截断，如果必要截断小数部分。如果数值在转换时不能适合目标主变量或列，将返回一个负的 SQLCODE。

在把小数，整数，二进制数赋值给小数，整数或二进制列或主变量时，数值被转换，如果必要将转换为目标的精度与标度。要增加或删除必要的前置零，在小数部分，加必要的结尾零，或者缩减必要的结尾数字。

在把二进制或浮点数赋值给十进制数或数值列或主变量时，这些数将首先转换为临时的十进制或数值数，然后再转换，如果必要将转换为目标的精度和标度。

—当一个零标度的半字二进制整数（SMALLINT）被转换为十进制数或数值时，临时结果的精度为 5 标度为 0。

—当一个全字二进制（INTEGER）被转换为十进制或数值时，临时结束的精度为 11，标度为 0。

—当一个浮点数被转换为十进数或数值时，临时结果的精度为 31，标度是在不丢失表示数值的整数部分所代表的最大标度。

9.1.1.4 日期、时间、时间标记的赋值原则

当把日期赋值给主变量，日期转换为 CRTSQLxxx 命令中的 DATFMT 和 DATSEP 参数规定的相应字符串，前置零不会被从日期的任何部分省略，主变量必须是定长或者变长字符串，对于*USA、*EUR、*JIS 或*ISO 日期格式，长度至少为 10 字节，对于*MDY、*DMY 或*YMD 日期格式长度至少为 8 字节，对于*JUL 日期格式至少为 6 字节。如果长度大于 10，那么字符串在右边填空格。在 ILE RPG 中，主变量也可以是日期变量。

当把时间赋值给主变量，时间转换为 CRTSQLxxx 命令中的 TIMFMT 和 TIMSEP 参数规定的相应字符串，不能省略前置零。主变量必须是定长或变长的字符串变量，如果主变量的长度大于表示时间的串长度，串在右边填空格。

如果使用*USA 格式，主变量的长度必须不少于 8。

如果使用*HMS，*ISO，*EUR 或*JIS 格式，如果包括秒，主变量的长度必须不能小于 8 字节，如只包括小时和分，则不能少于 5 字节。在这种情况下，SQLWARN0 和 SQLWARN1 设为 n。如果规定了指示器变量，那么它设置为秒截断的实际数。在 ILE RPG 中，主变量也可以是时间变量。

当时间标记赋值给主变量时，时间标记将转换为相应的字符串，不能去掉任何部分的前置零。主变量必须是定长或者长度至少是 19 字节的变长字符串变量。如果长度小于 26，主变量不包括微秒的所有数字。如果长度大于 26，主变量将在右边填空格。在 ILE RPG 中主变量也可以是时间标记变量。

9.1.2 指示器变量

指示器变量是一个半字节整数变量，用来说明与其相关的主变量是否分配了空值：

如果结果列的值为空，SQL 把指示变量置为-1。

如果没有使用指示变量且结果列是空值，那么返回负的 SQLCODE。

如果结果列的值产生了一个数据映射错误，SQL 把指示器变量置为-2。

还可以使用指示器变量来验证检索到的串值有无截断。如果发生截断，指示器变量为串的原长度。

当数据库管理从结果列中返回一个值时，可以检测指示器变量，如果指示器变量的值小于零，便可以知道结果列的值为空。当数据库管理返回一个空值，主变量设为结果列的缺省值。

可以在主变量之后立即规定指示器变量或在 INDICATOR 键字之后立即规定它，例如：

```
EXEC SQL
   SELECT COUNT(*), AVG(SALARY)
   INTO :PLICNT, :PLISAL:INDNULL
   FROM CORPDATA.EMPLOYEE
   WHERE EDLEVEL < 18
END-EXEC.
```

然后，可以检测 INDNULL 来看是否有一个负值。如果有，便可以知道 SQL 返回了一个空值。

一般总是使用 IS NULL 谓词来检测列中的空值。例如：

```
        WHERE  表达式  IS  NULL
```

不能用下面方法检测 NULL：

```
MOVE -1 TO HUIND.
EXEC SQL...WHERE 列名 = :HUI :HUIND
```

EQUAL 谓词在与空值比较时，总是被核定为假，这个例子的结果是没有行被选择。

9.1.2.1 使用主结构

可以规定一个指示器结构来支持主结构（定义一个半字节整数变量数组）。如果返回到主结构的结果列的值能为空，便可以加一个指示器结构名给主结构名，这就允许 SQL 通知程序，返回到主结构中的主变量的每个空值。

例如，用 COBOL：

```
01  SAL-REC.
    10 MIN-SAL            PIC S9(6)V99 USAGE COMP-3.
    10 AVG-SAL            PIC S9(6)V99 USAGE COMP-3.
    10 MAX-SAL            PIC S9(6)V99 USAGE COMP-3.
01  SALTABLE.
02  SALIND                PIC S9999 USAGE COMP-4 OCCURS 3 TIMES.
01  EDUC-LEVEL            PIC S9999 COMP-4.
   ...
    MOVE 20 TO EDUC-LEVEL.
   ...
    EXEC SQL
```

```
SELECT MIN(SALARY), AVG(SALARY), MAX(SALARY)
  INTO :SAL-REC:SALIND
  FROM CORPDATA.EMPLOYEE
  WHERE EDLEVEL>:EDUC-LEVEL
END-EXEC.
```

在这个例子中,SALIND 是一人有 3 个值的数组,每一个检测是否为负值。如果 SALIND(1)包括一个负值，那么主结构中（在这里是 MIN-SAL）相应的主变量对于选择的行不会改变。

在上面的例子中，SQL 把选择行的列值放入主结构，因此，必须使用相应指示器变量的结构来决定哪些选择列值是空的。

9.1.2.2 用来设置空值

可以用一个指示器变量在一个列中设置空值。当处理 UPDATE 或 INSERT 语句时，SQL 检测指示器变量（如果存在），如果它是负值，那么列值设为空。如果它是大于-1 的值，那么相关主变量包括列的值。

例如，可以规定放到列中一个值（使用 INSERT 或 UPDATE 语句），但不能保证这个值做为输入数据规定了。为了提供把列值置为空的能力，可以写下面的语句：

```
EXEC SQL
 UPDATE CORPDATA.EMPLOYEE
   SET PHONENO = :NEWPHONE:PHONEIND
   WHERE EMPNO = :EMPID
END-EXEC.
```

当 NEWPHONE 包含非空值时，用下面的语句把 PHONEIND 置为零：

```
MOVE  0   to  PHONEIND
```

否则，通知 SQL NEWPHONE 包含了一个空值，把 PHONEIND 设为负值，如下：

```
MOVE  -1   to  PHONEIND
```

9.2 处理 SQL 错误返回码

当程序处理 SQL 语句时，SQL 把返回码放在 SQLCODE 和 SQLSTATE 字段中，返回码指出语句运行成功或失败。如果 SQL 在处理语句时遇到了错误，那么 SQLCODE 是一个负数，并且 SUBSTR(SQLSTATE，1，2)不是‘00’，‘01’或‘02’。如果在处理语句时，SQL 遇到了一个异常但是有效的条件,那么 SQLCODE 是一个正数并且 SUBSTR(SQLTATE,1,2)是‘01’或‘02’。如果 SQL 语句在处理时没有遇到错误或者警告条件,那么 SQLCODE 是 0,SQLTATE 是“00000”。

注：有这种情况，值为 0 的 SQLCODE 返给程序，而结果可能不令人满意。例如，如果程序运行的结果是一个值被截断了，返给程序的 SQLCODE 为零。但 SQL 的一个警告标志(SQLWARN1)指出这个截断。在这种情况下，SQLTATE 不是“00000”。

注意，如果不检测负的 SQLCODE 值或者规定 WHENEVER SQLERROR 语句，程序将继续下一条语句，在错误之后继续运行将产生不可预测的结果。

SQLSTATE 的主要目的，是在不同的 IBM 关系的数据库系统之间，对公共返回条件提供

公共返回码。在处理分布数据操作问题时，SQLSTATE 特别有用。详细内容请看 AS/400 DB2 SQL 参考。

由于 SQLCA 是一个有价值的错误诊断工具，在应用程序中编写必要的指令来显示在 SQLCA 中的一些信息是一个非常好的想法。下面的 SQLCA 字段特别重要：

SQLCODE　　返回码
SQLSTATE　　返回码
SQLERRE(3)　被 SQL 修改、插入或删除的行数量
SQLWARNO　　如果被置为 W，那么至少设置一个 SQL 的警告标志
（SQLWARN1 到 SQLWARNA）

SQLCA 的详细信息，请看 AS/400 DB2 SQL 参考中的附录 B。SQLCODEs 和 SQLSTATEs 的列表，见本书原文附录 B。

9.3 用 WHENEVER 处理异常条件

WHENEVER 语句使 SQL 检测 SQLSTATE 和 SQLCODE 并且继续执行程序，或在运行 SQL 语句时有错误、异常或警告出现，转移到程序中的其它区域。一个异常条件处理子程序（程序的一部分）将检测 SQLCODE 或 SQLSTATE 字段，对错误或异常条件采取动作。

注：WHENEVER 不允许在 REXX 过程中使用。

WHENEVER 语句允许规定一个条件为真时做什么，对于相同情况可以规定多个 WHENEVER 语句。这样做时，第一个 WHENEVER 将适用于在源程序中所有后续的 SQL 语句，直到规定另一个 WHENEVER。

WHENEVER 语句格式如下：

EXECSQL
WHENEVER条件　动作
END-EXEC

可以规定三种条件：

SQLWARNING　　规定 SQLWARING 指出，当 SQLWARNO＝W 或 SQLCODE 中包括了不是 100 的正值时（SUBSTR(SQLSTATE，1，2) ='01'）时所采取的动作。
注：几种不同的原因都可设置 SQLWARNO。例如，如果列值在它被移入主变量时被截断，程序可能不把这当成错误。

SQLERROR　　规定 SQLERROR 指出，在返回 SQL 语句结果的错误码时所采取的动作。
(SQLCODE < 0) (SUBSTB(SQLSTATE，1，2) > '02')

NOT FOUND　　说明 NOT FOUND 来指出由于下面的情况，返回 SQLCODE=+100 且 SQLSTATE=‘02000’时采取的动作：
由于在单行的 SELECT 发出之后或在游标发出的第一个 FETCH 之后，程序规定的数据并不存在。
在 FETCH 之后，没有满足游标选择语句的行用来检索。
在一个 UPDATE、DELETE 或 INSERT 之后，没有行满足检索条件。

可以规定下面那些要采取的动作：

CONTINUE　　程序继续执行下一条语句。
GO TO 标号　程序转移到程序中的其它区域。区域的标号前可能有一个冒号。
WHENEVER...GO TO 语句：

在 COBOL 中，必须是一个节的名字，或一个没有限定的段名。

在 PL/1 和 C 中，必须是一个标号。

在 RPG 中，是 TAG 的标号。

例如，如果用游标检索行，希望的是，当发出 FETCH 语句时 SQL 不用去找另外的行。为了准备这个情况，规定 WHENEVER NOT FOND GO TO 语句使 SQL 能转到程序中的发出 CLOSE 语句另一个地方，来适当地关闭游标。

注：WHENEVER 语句将影响所有后续的源 SQL 语句，直到另一个 WHENEVER 语句出现。

换句话说，在两个 WHENEVER 语句（或者如果只有一个话，那么在这个之后）之间的所有 SQL 语句都由第一个 WHENEVER 语句控制而不管程序所用的路径。

正因为如此，WHENEVER 语句必须放在它影响的第一个 SQL 语句之前，如果 WHENEVER 在 SQL 语句之后，那么转移的分支将不会依据 SQL 语句设置的 SQLCODE 和 SQLSTATE 的值。然而，如果程序直接检测 SQLCODE 或 SQLSTATE，那么检测必须在 SQL 语句运行之后执行。

WHENEVER 语句不提供 CALL 子例程选项。由于这个原因，可以在每个 SQL 语句运行之后检查 SQLCODE 和 SQLSTATE 值并且调用子例程，而不使用 WHENEVER 语句。

第十章　C 程序中使用 SQL 语句（略）

第十一章　COBOL 程序中使用 SQL 语句（略）

第十二章　PL/I 程序中使用 SQL 语句（略）

第十三章　RPG/400 程序中使用 SQL 语句（略）

第十四章　　ILE　RPG/400 程序中使用 SQL 语句

这章介绍，在 ILE　RPG/400 程序中嵌入 SQL 语句的编码的必要内容，主变量的必要条件已经定义了。

14.1　定义 SQL 通讯区

SQL 的预编译程序在第一个计算规范表前自动的把 SQLCA 放入 ILE　RPG/400 程序的定义规范表中。在源程序中不必编写 INCLUDE　SQLCA 语句，如果源程序中有 INCLUDE　SQLCA，那么将接受它，但它是多余的。在 ILE　RPG/400 中定义的 SQLCA 为：

```
D*      SQL Communications area
D SQLCA           DS
D  SQLAID                 1      8A
D  SQLABC                 9     12B 0
D  SQLCOD                13     16B 0
D  SQLERL                17     18B 0
D  SQLERM                19     88A
D  SQLERP                89     96A
D  SQLERRD               97    120B 0 DIM(6)
D  SQLERR                97    120A
D   SQLER1               97    100B 0
D   SQLER2              101    104B 0
D   SQLER3              105    108B 0
D   SQLER4              109    112B 0
D   SQLER5              113    116B 0
D   SQLER6              117    120B 0
D  SQLWRN                121    131A
D   SQLWN0               121    121A
D   SQLWN1               122    122A
D   SQLWN2               123    123A
D   SQLWN3               124    124A
D   SQLWN4               125    125A
D   SQLWN5               126    126A
D   SQLWN6               127    127A
D   SQLWN7               128    128A
D   SQLWN8               129    129A
D   SQLWN9               130    130A
D   SQLWNA               131    131A
```

```
D  SQLSTT                132     136A
D*  End of SQLCA
```

注：在 RPG/400 中，变量的名被限制为 6 个字符，标准的 SQLCA 名被转变为长度为 6 的 RPG/400 格式。为了保持与那些转换为 ILE RPG/400 的 RPG/400 程序的兼容性，SQLCA 的名字将保留不变。ILE RPG/400 定义的 SQLCA 增加了一个 SQLERRD 字段，它定义为一个 6 个整数的数组，SQLERRD 定义优于 SQLERR 定义。

14.2 定义 SQL 描述区

下面的语句需要 SQLDA：

```
EXECUTE...USING DESCRIPTOR 描述名

FETCH...USING DESCRIPTOR 描述名

OPEN...USING DESCRIPTOR 描述名

CALL...USING DESCRIPTOR 描述名

DESCRIBE 语句名 INTO 描述名

DESCRIBE TABLE 主变量 INTO 描述名

PREPARE 语句名 INTO 描述名
```

与 SQLCA 不一样，在程序中可以有多个 SQLDA，并且一个 SQLDA 可以有任何有效的名称。

动态 SQL 是一个先进的程序设计技术。用动态 SQL，程序可以在其运行时编写和运行 SQL 语句。动态运行的带有选择列表变量（就是说，作为查询一部分返回的数据列表）的 SELETE 语句需要一个 SQL 描述区（SQLDA），这是因为不能预先知道要接收 SELECT 的结果，要分配多少个及什么类型的变量。

在 ILE RPG/400 程序中能规定 INCLUDE SQLDA 语句，语句的格式如下：

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8.
C/EXEC SQL INCLUDE SQLDA
C/END-EXEC
```

INCLUDE SQLDA 产生下面的数据结构：

```
D*      SQL Descriptor area
D SQLDA           DS
D  SQLDAID                1      8A
D  SQLDABC                9     12B 0
D  SQLN                  13     14B 0
```

```
D  SQLD                    15     16B 0
D  SQL_VAR                        80A   DIM(SQL_NUM)
D                          17     18B 0
D                          19     20B 0
D                          21     32A
D                          33     48*
D                          49     64*
D                          65     66B 0
D                          67     96A
D*
D SQLVAR           DS
D  SQLTYPE                  1      2B 0
D  SQLLEN                   3      4B 0
D  SQLRES                   5     16A
D  SQLDATA                 17     32*
D  SQLIND                  33     48*
D  SQLNAMELEN              49     50B 0
D  SQLNAME                 51     80A
D*  End of SQLDA
```

用户对 SQL_NUM 的定义负责。SQL-NUM 必须定义为满足 SQL-VAR 空间大小的数值常量。

由于 ILE RPG/400 不支持数组中的数据结构，INCLUDE SQLDA 产生了两个数据结构。第二个数据结构是用来建立/引用包括字段描述的 SQLDA 部分。

要建立 SQLDA 的字段描述，程序在 SQLVAR 的子字段建立字段描述，然后做一个 MOVEA，把 SQLVAR 传送到 SQL_VAR，n，这里 n 是 SQLDA 的字段数，所有字段描述都要这样做。

当引用 SQLDA 的字段描述时，用户要做一个 MOVEA，把 SQL_VAR，n 传送给 SQLVAR，这里的 n 是要处理的字段描述数。

14.3 嵌入 SQL 语句

在 ILE RPG/400 中编写的 SQL 语句必须放在计算部分，即在第六列有一个 C。SQL 语句可以放在详细计算，总计计算或者一个 RPG 的子例程中，SQL 语句以 RPG 语句的逻辑执行。

键字 EXEC SQL 指出 SQL 语句的开始。EXEC SQL 必须出现在源语句的 8 到 16 列，在第 7 列为（ / ）。SQL 语句从 17 列开始到 80 列。

键字 END-EXEC 结束 SQL 语句。END-EXEC 必须出现在源语句中 8 到 16 列,第 7 列为(/)。17 到 80 列必须为空。

在 SQL 语句中，大小写都可以。

14.3.1 例子

在 ILE RPG/400 程序中，UPDATE 语句可以向下面这样编写：

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8.
```

```
C/EXEC SQL UPDATE DEPARTMENT
C+           SET MANAGER = :MGRNUM
C+           WHERE DEPTNO = :INTDEP
C/END-EXEC
```

14.3.2 注释

除了 SQL 注释（－－）外，ILE RPG/400 的注释可以包括在任何允许空格的 SQL 语句中。为了在 SQL 语句内嵌入一个 ILE RPG/400 注释，在第 7 列写一个星号(*)。

14.3.3 SQL 语句的续行

SQL 语句需要续行时，可以使用 9 到 80 列。第 7 列必须是一个“＋”号，第 8 列必须是空白，要续行的第 80 列与续行的第 9 列相接。

包括 DBCS 数据的常量可以把转入字符放在要续的 81 列并且把转出字符放在续行的第 8 列来连续跨越多行。

在例子中，SQL 语句有一个有效的图形常量 G'<AABBCCDDEEFFGGHHIIJJKK>。

```
G'<AABBCCDDEEFFGGHHIIJJKK>'.

*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8.
C/EXEC SQL   SELECT * FROM GRAPHTAB WHERE GRAPHCOL =  G'<AABBCCDDEE>
C+<FFGGHHIIJJKK>'
C/END-EXEC
```

14.3.4 包含代码

使用下面的 SQL 语句，可以包括 SQL 语句和 RPG 计算规范：

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
C/EXEC SQL INCLUDE member-name
C/END-EXEC
```

RPG /COPY 语句可以用来包括 SQL 语句或 RPG 规范。

14.3.5 顺序号

SQL 预编译程序所产生的源语句的顺序号，是根据 CRTSQLRPGI 命令中的 OPTION 参数的键字*NOSEQSRC/*SEQSRC 规定的，当规定*NOSEQSRC 时，用输入的源成员的顺序号，对*SEQSRC，顺序号从以 000001 开始每次增加 1。

14.3.6 名字

任何有效的 ILE RPG/400 变量名都可以用做主变量，但有以下限制：

不能用以 SQ、SQL、RDI 或 DSN 开头的主变量名和外部入口名，这些名字是为数据库管理保留的。主变量名字的长度限制在 64。

14.3.7 语句标号

TAG 语句可以放在任何 SQL 语句之前，在 EXEC SQL 前写 TAG。

14.3.8 WHENEVER 语句

GOTO 子句的目标必须是 TAG 语句标号。必须遵循 GOTO/TAG 的范围规则。

14.4 使用主变量

在 SQL 语句中使用的所有主变量必须被明确的说明。

嵌入在 ILE RPG/400 中的 SQL 不使用 SQL BEGIN DECLARE SECTION 和 EDN DECLARE SECTION 语句来标识主变量，不要把这些语句放入源程序。所有 SQL 语句中的主变量必须在前面加一个冒号(：)。

在程序中主变量的名字必须是唯一的，即使这个主变量在不同的过程中也必须这样。

使用主变量的语句必须在变量说明的范围内。

14.4.1 说明主变量

SQL ILE RPG/400 的预编译程序仅认识有效的 ILE RPG/400 主变量说明的子集。

所有在 ILE RPG/400 中定义的变量都可以在 SQL 语句中使用，但除了以下的情况：

指针
表
UDATE
UDAY
UMONTH
UYEAR
向前字段
命名常量
多维数组
定义需要的*SIZE 和*ELEM 的分辨率
定义需要的常量分辨率，当常量用在 OCCURS 和 DIM 中例外。

使用 ILE RPG/400 的 CALL/PARM 功能，把用做主变量的字段传给 SQL。如果一个字段不能用做 PARM 的结果字段，它就不能作为主变量使用。

日期和时间主变量总是被赋值给由 SQL 预编译程序所产生的结构中相应的日期和时间

子字段。使用在 CRTSQLRPGI 命令的 DATFMT，DATSEP，TIMFMT 和 TIMSEP 参数规定的格式与分隔符说明生成的日期和时间的子字段。从用户说明的主变量的格式到预编译程序所描述的格式之间的转换，发生在往 SQL 产生的结构中分配或从 SQL 产生的结构赋值的过程中。如果 DATFMT 参数值是系统格式(*MDY，*YMD，*DMY，*JUL)，那么所有的输入、输出主变量的值必须在 1940-2039 范围内。如果任何日期值超出了这个范围，那么在预编译程序的 DATFMT 必须规定为 IBM SQL 格式*ISO，*USA，*EUR，*JIS 中的一个。

14.5 使用主结构

如果在数据结构中存在子字段，ILE RPG/400 数据结构的名字可以作为主结构的名字，在 SQL 语句中的数据结构名字隐含了组成数据结构的子字段名字的列表。

当数据结构的子字段并不存在时，数据结构的名字是字符型的主变量，这时允许字符变量长度大于 256。当这种支持不提供额外的功能时，由于一个字段可以定义为最大长度 32766，这就需要与 RPG/400 程序的兼容性。

在下面的例子中，BIGCHR 是一个没有子字段的 ILE 数据结构，SQL 把任何引用 BIGCHR 都作一个长度为 642 的字符串来对待。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
DBIGCHR           DS            642
```

在下面的例子中，PEMPL 是一个由 EMPNO，FIRSTN，MIDINT，LASTNAME 和 DEPTNO 子字段组成的数据结构，对 PEMPL 的引用使用子字段。例如，
CORPDATA.EMPLOYEE 的第一列放到 EMPNO 中，第二列放到 FIRSTSN 中等等。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
DPEMPL            DS
D EMPNO                   01      06A
D FIRSTN                  07      18A
D MIDINT                  19      19A
D LASTNA                  20      34A
D DEPTNO                  35      37A

...
C                    MOVE       '000220'       EMPNO

...
C/EXEC SQL
C+ SELECT *  INTO :PEMPL
C+ FROM  CORPDATA.EMPLOYEE
C+ WHERE  EMPNO = :EMPNO
C/END-EXEC
```

当编写一个 SQL 语句时，引用子字段可以是限定的，用数据结构的名字，跟着一个句点，后接子字段名。例如，PEMPL.MIDINT 与仅规定 MIDINT 是一样的。

14.6 使用主结构数组

主结构数组是作为一个发生的数据结构被定义的。当处理多行时，一个发生的数据结构可用在 SQL FETCH 和 INSERT 语句中。在支持使用多行块的数据结构时，必须考虑下面的内容：

所有的子字段必须是有效的主变量。

所有的子字段必须是相邻的，第一个 FROM 的位置必须是 1，TO 和 FROM 位置不能有重叠。

在主结构中的日期和时间子字段的格式与分隔符与 CRTSQLRPGI 命令中的 DATFMT，DATSEP，TIMFMT 和 TIMSEP 参数不一致，那么主结构数组是不可用的。

除了成块的 FETCH 和成块的 INSERT 之外，对于所有的语句，如果使用发生的数据结构，那么使用当前的发生值。对于成块的 FETCH 和成块的 INSERT，发生值置为 1。

在下面的例子中，使用叫做 DEPT 的主结构数组，成块的 FETCH 语句从 DEPARTMENT 表中检索 10 行数据。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
DDEPARTMENT       DS                  OCCURS(10)
D DEPTNO                  01     03A
D DEPTNM                  04     32A
D MGRNO                   33     38A
D ADMRD                   39     41A

DIND_ARRAY        DS                  OCCURS(10)
D INDS                             4B 0 DIM(4)
...
C/EXEC SQL
C+ DECLARE C1 FOR
C+    SELECT *
C+       FROM  CORPDATA.DEPARTMENT
C/END-EXEC
...

C/EXEC SQL
C+   FETCH C1 FOR 10 ROWS
C+     INTO :DEPARTMENT:IND_ARRAY
C/END-EXEC
```

14.7 使用外部文件描述

SQL 的预编译用与 ILE RPG/400 编译非常相似的方式来处理 ILE RPG/400 源程序。这就意味着预编译处理/COPY 语句来定义主变量。如果规定了不同的名字，那么要得到外部描述文件的字段定义且重新命名，数据结构的外部定义格式可以用来获得作为主变量使用的列

名的拷贝。

SQL 预编译怎样来检索和处理日期和时间字段定义，是根据在 CRTSQLRPGI 命令的 OPTION 参数是否规定了*NOCVTDT 或*CVTDT。如果规定了*NOCVTDT，那么日期和时间字段定义检索时包含格式和分隔符。如果规定了*CVTDT，那么在日期和时间字段定义检索时，忽略格式和分隔符，并且预编译程序将假设变量的说明是字符格式的日期 / 时间主变量。*CVTDT 是对于 RPG/400 预编译兼容的选项。

在下面的例子中，样板表的 DEPARTMENT 作为 ILE RPG/400 程序中的一个文件来使用，SQL 预编译检索 DEPARTMENT 的字段（列）定义做为主变量使用。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
FDEPARTMENTIP   E             DISK     RENAME(ORIGREC:DEPTREC)
```

注：只有在用 ILE RPG/400 语句来对文件执 I/O 操作时，才需要在 ILE RPG/400 程序中为文件编写一个 F 表。如果仅用 SQL 语句执行 I/O 操作，可使用外部数据结构来包含一个文件（表）的外部定义。

在下面的例子中，样板表作为一个外部的数据结构，SQL 的预编译检索字段（列）定义作为数据结构的子字段。子字段的名字可以用作主变量名，数据结构名 TDEPT 可以用作主结构的名字。下例给出，如果程序需要，文件名可以被重新命名。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
DTDEPT          E DS                  EXTNAME(DEPARTMENT)
D DEPTN         E                     EXTFLD(DEPTNAME)
D ADMRD         E                     EXTFLD(ADMRDEPT)
```

如果 GRAPHIC 或 VARGRAPHIC 列有一个 UCS-2 CCSID，生成的主变量也有一个 USC-2 CCSID 分配给它。

14.7.1 主结构数组的外部文件描述考虑

对于设备文件，如果没规定 INDARA 并且文件包含指示器，那么说明不能作主结构数组使用。指示器区字段被包含在生成的结构中，并且可能使存储空间被分开。

如果规定了 OPTION (*NOCVTDT)并且文件中的日期和时间字段定义格式和分隔符与在 CRTSQLRPGI 命名中的 DATFMT，DATSEP，TIMFMT，TIMSEP 参数不一致，那么主结构数组不能使用。

在下面的例子中，DEPARTMENT 表包括在 ILE RPG/400 程序中并且说明为一个主结构数组，用一个成块的 FETCH 语句来把检索的 10 行数据放入主结构数组中：

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
DDEPARTMENT     E DS                  OCCURS(10)

...

C/EXEC SQL
C+   DECLARE C1 CURSOR FOR
```

```
C+       SELECT *
C+         FROM CORPDATA.DEPARTMENT
C/END-EXEC

...

C/EXEC SQL
C+         FETCH C1 FOR 10 ROWS
C+            INTO :DEPARTMENT
C/END-EXEC
```

14.8 确定相同的 SQL 与 RPG 数据类型

预编译程序将根据下表来决定主变量的基本 SQLTYPE 和 SQLLEN，如果主变量带有一个指示器变量，那么 SQLTYPE 是基本 SQLTYPE 加 1。

表 14-1. ILE RPG/400 说明与典型的 SQL 数据类型的对照

RPG 数据类型	D 表第 40 列	D 表第 41,42 列	其他 RPG 编码	主变量的 SQLTYPE	主变量的 SQLLEN	SQL 数据类型
数据结构(没有此字段)	空白	空白	长度 = n n=32766	452	n	CHAR(n)
计算结果字段（6970 列=空白）	n/a	n/a	长度 =n n= 32766 （59-63 列）	452	n	CHAR(n)
定义规范表	A	空白	长度 =n n= 1-254 44-80=VARYING	448	n	VARCHAR (n)
定义规范表	A	空白	长度>n n>254. 44-80=VARYING	456	n	VARCHAR (n)
定义规范表	B	0	长度 =n n = 4	500	2	SMALLINT
定义规范表	I	0	长度=n n=5	500	2	SMALLINT
定义规范表	B	0	长度 n n= 9 和 5	496	4	INTEGER
定义规范表	I	0	长度 =n n = 10	496	4	INTEGER
定义规范表	B	1-4	长度 = n n = 2	500	2	DECIMAL(4,s) s=41，42 列
定义规范表	B	1-9	长度 = n n = 4	496	4	DECIMAL(9,s) s=41, 42 列
定义规范表	P	0 到 30	长度 = n n = 1-16	484	字节 1 为 p,字节 2 为 s	DECIMAL(p,s) p = n*2-1 且 s = 41, 42 列
定义规范表	F	空白	长度 = n n = 4	480	4	FLOAT （单精度）
定义规范表	F	空白	长度 = n n = 8	480	8	FLOAT （双精度）
定义规范表,不是子字段	空白	0 到 30	长度 = n n = 1-16	484	字节 1 为 p，字节 2 为 s	DECIMAL(p,s) p =n*2-1 且 s =41, 42 列
输入字段（36 列 = p）	n/a	n/a	长度 = n n = 1-16 （37-46 列）	484	字节 1 为 p，字节 2 为 s	DECIMAL(p,s) p =n*2-1 且 s =47, 48 列
输入字段（36 列= 空白或 S)	n/a	n/a	长度 = n n = 1-30 （37-46 列）	484	字节 1 为 p，字节 2 为 s	DECIMAL(p,s) p = n 且 s = 47, 48 列
输入字段	n/a	n/a	长度 = n	484	字节 1 为 p，字	DECIMAL(p,s)

(36 列 = B)			n = 2 或 4 （37-46 列）		节 2 为 s	如果 n=2 和 9，p=4 ，如果 n=4 s = 第 47, 48 列
计算结果字段 (69,70 列 &ne. 空白)	n/a	n/a	长度 = n n = 1-30 （59-63 列）	484	字节 1 为 p，字节 2 为 s	DECIMAL(p,s) p = n 且 s = 第 64, 65 列
数据结构子字段	空白	0 到 30	长度 = n （n=1-30）	488	字节 1 为 p，字节 2 为 s	NUMERIC(p,s) p = n 且 s = 41, 42 列
定义规范表	S	0 到 30	长度 = n (n=1-30）	488	字节 1 为 p，字节 2 为 s	NUMERIC(p,s) p = n 且 s =41, 42 列
输入字段（36 列 = G	n/a	n/a	长度 = n (n=1-32766)	468	m	GRAPHIC(m) m = n/2 m =(TO-FROM-1)/2
定义规范表	G	空白	长度 = n (n=1-127) 44-80 列 =VARYING	464	n	VARGRAPHIC(n)
定义规范表	G	空白	长度=n n>127. 44-80 列为 VARYING .	472	n	VARGRAPHIC(n)
定义规范表	D	空白	长度 = n n = 6, 8 或 10	384	n	DATE (DATFMT, DATSEP 在 44-80 列规定)
输入字段（36 列=D）	n/a	n/a	长度 = n n= 6, 8, 或 10(37-46 列)	384	n	DATE (在 31-34 列规定格式)
定义规范表	T	空白	长度 = n n = 8	388	n	TIME (TIMFMT, TIMSEP 在 44-80 列规定)
输入字段 (36 列 = T)	n/a	n/a	长度 = n n =8 (37-46 列)	388	n	TIME (在 31-34 列规定格式)
定义规范表	Z	空白	长度 = n n = 26	392	n	TIMESTAMP
输入字段 (36 列 = Z)	n/a	n/a	长度 = n n =26 (37-46 列)	392	n	TIMESTAMP

注：

1.在第一列中的术语“定义规范表”包括数据结构体子字段，除非明显的指出其它内容。

2.在定义规范表中，二进制字段的长度取决定于下面两点：

FROM(26—32 列)不空，那么长度＝TO－FROM＋1

FROM(26—32 列)为空，如果 33－39 列小于 5，则长度为 2，如果 33－39 列大于 4 则长度为 4。

3.SQL 将使用 CRTSQLRPGI 命令中的 DATE/TIME 格式生成日期 / 时间子字段。

下表可用来决定与给出的 SQL 数据类型相等的 RPG 数据类型：

表 14－2 SQL 数据类型与典型 RPG 说明之间的对应表

SQL 数据类型	RPG 数据类型	说 明
SMALLINT	定义规范表 40 列为 I，长度必须为 5 并且 42 列为 0 或者 定义规范表 40 列为 B，长度为 4 并且 42 列为 0	
整型	定义规范表	

	40 列为 I，长度必须为 10 并且 42 列为 0 或者 定义规范表 40 列为 B，长度必须为 9 和 5 并且 42 列为 0	
小数	定义规范表 40 列为 P 或非子字段 40 列为空白， 41,42 列为 0－30 或在非定义规范表中定义为一个数值	最大长度为 16(精确度为 30) 最大范围为 30
NOMERIC	定义规范表。40 列为 S 或子字段的 40 列为空白，41,42 列为 0－30	最大长度为 30(精确度为 30) 最大范围为 30
单精度浮点	定义规范表。40 列为 F，长度必须是 4。	
双精度浮点	定义规范表。40 列为 F，长度必须是 8。	
字符(n)	定义规范表。40 列为 A 或空白，41,42 列为空白 或者定义无小数的输入字段 或者定义无小数的计算结果字段	
CHAP(n)	在数据结构体中没有子字段的数据结构体	1<n<32766
VARCHAR(n)	定义规范表。40 列为 A 或空白并且 44－80 列为 VARYING	1<n<32740
GRAPHIC(n)	定义规范表。40 列为 G，或者 36 列为 G 的输入字段	1<n<16383
VARGRAPHI(n)	定义规范表。40 列为 G 并且 44－80 列为 VARYING	1<n<16370
DATE 日期	字符字段 或者定义规范表，40 列为 D 或者 36 列为 D 为输入字段定义	如果格式是*USA,*JIS, *EUR 或*ISO，那么长度必须至少为 10。 如果格式是*YMD,*DMY,*MDY ,那么长度至少为 8。 如果格式是*JUL，那么长度至少必须为 6。
	字符字段 或者定义规范表，40 列为 T 或者 36 列为 T 的输入字段定义	长度必须是至少 6，要包括秒，长度必须至少 8。
	字符字段 或者定义规范表，40 列为 Z 或者 36 列为 Z 的输入字段定义	长度必须至少是 19，要包括微秒，长度必须至少为 26。 如果长度小于 26，微秒部分将被截断。

14.8.1 ILE RPG/400 变量说明及用法

ILE RPG/400 所有的数值类型都有精度和范围。ILE RPG/400 定义数值操作，假设数据是压缩格式，这就意味着，在操作执行之前，有二进制变量参与的操作要有隐含的转换为压缩格式的操作（如果必要，要转换回二进制格式）。当执行 SQL 操作时，数据要对准隐含的小数位。

14.9 使用指示器变量

一个指示器变量是一个长度小于 5（2 字节）的二进制字段。

通过说明变量元素长度为 4，0 和在定义规范中规定 DIM 来定义指示器数组。

在检索时，指示变量用来指示与它相联系的主变量是否被分配一个空值，在赋值给列时，一个负的指示器变量用来指示分配一个空值。

详细信息，请看 AS/400 DB2 SQL 参考。

指示器变量与主变量的说明方法一样，并可以混在一起说明。

14.9.1 例子

给出下面的语句：

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
C/EXEC SQL FETCH CLS_CURSOR INTO :CLSCD,
C+                               :DAY :DAYIND,
C+                               :BGN :BGNIND,
C+                               :END :ENDIND
C/END-EXEC
```

变量说明如下：

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
D CLSCD           S               7
D DAY             S               2B 0
D DAYIND          S               2B 0
D BGN             S               8A
D BGNIND          S               2B 0
D END             S               8
D ENDIND          S               2B 0
```

14.10 取多行区的 SQLDA 的例子

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....8.
C/EXEC SQL INCLUDE SQLDA
C/END-EXEC
DDEPARTMENT       DS                  OCCURS(10)
D DEPTNO                  01     03A
D DEPTNM                  04     32A
D MGRNO                   33     38A
D ADMRD                   39     41A
...
```

```
DIND_ARRAY        DS                  OCCURS(10)
D INDS                             4B 0 DIM(4)
...
C* setup number of sqlda entries and length of the sqlda
C                   eval      sqld = 4
C                   eval      sqln = 4
C                   eval      sqldabc = 336
C*
C* setup the first entry in the sqlda
C*
C                   eval      sqltype = 453
C                   eval      sqllen  = 3
C                   eval      sql_var(1) = sqlvar
C*
C* setup the second entry in the sqlda
C*
C                   eval      sqltype = 453
C                   eval      sqllen  = 29
C                   eval      sql_var(2) = sqlvar
...
C*
C* setup the forth entry in the sqlda
C*
C                   eval      sqltype = 453
C                   eval      sqllen  = 3
C                   eval      sql_var(4) = sqlvar

...
C/EXEC SQL
C+ DECLARE C1 FOR
C+    SELECT *
C+       FROM  CORPDATA.DEPARTMENT
C/END-EXEC
...

C/EXEC SQL
C+   FETCH C1 FOR 10 ROWS
C+     USING DESCRIPTOR :SQLDA
C+     INTO :DEPARTMENT:IND_ARRAY
C/END-EXEC
```

第十六章　　准备和运行有 SQL 语句的程序

这章讲述关于如何准备和运行应用程序的一些任务。这些任务分为：预编译、编译、联编、运行。

16.1　SQL 预编译的基本处理

在运行一个带有 SQL 语句的程序之前，必须对其预编译和编译。(1)程序的预编译由 SQL 预编译器来做，SQL 预编译器检查源程序的每个语句，并且完成以下任务：

查找 SQL 语句和主变量名的定义。变量名和定义可用来检查 SQL 语句，在 SQL 预编译完成处理后，可以通过清单看是否有错误发生。

检查每条 SQL 语句的语法错误及有效性，有效性检查可以在输出清单上显示出错信息，帮助你改正错误。

检查使用数据库描述的 SQL 语句的有效性。在预编译时，要检查 SQL 语句的有效表、视图、列名。如果规定的表、视图不存在，或者在预编译和编译时，没有被授权用此表、视图，则在运行时检查有效性。如果运行时表、视图仍不存在，则发生错误。

注：

1.当获得外部定义时，发生替换。详细信息，请看 AS/400　DB2 数据库编程和数据管理。

2.为使 SQL 语句有效，必须对引用的表和视图有某些（至少有*OBJOPR 权）权限。处理 SQL 语句的实际权限检查在运行时完成。详细信息，请 AS/400　DB2　SQL 参考。

3.如果在 CRTSQLxxx 命令中规定了 RDB 参数，预编译会访问规定的关系数据库来得到表和视图的描述。

准备在主语言中编译的 SQL 语句。对于绝大多数 SQL 语句，SQL 预编译在 SQL 接口模块中插入一个注释或 CALL 语句：

－QSQROUTE

－QSQLOPEN

－QSQLCLSE

－QSQLCMIT

对于一些 SQL 语句（例：DECLARE 语句），SQL 编译器不产生主语言语句（除注释外）。

产生关于每条被预编译的 SQL 语句的信息。这方面信息被内部地存储在一个临时源文件成员中，在联编过程中可用到它。

为在预编译时得到完整的诊断信息，要规定以下事项之一：

CRTSQLxxx 的 OPTION(*SOURCE　*XREF)，此时 xxx 为 CBL、PLI 或 RPG。

CRTSQLxxx 的 OPTION(*XREF) OUTPUT(*PRINT)，此时 XXX 为 CI，CBI 或 RPGI。

(1)在 REXX 过程中的 SQL 语句不用预编译和编译。

16.1.1 预编译的输入

在预编译中，应用编程语句和嵌入的 SQL 语句是它的主要输入。在 PL/1 和 C 程序中，SQL 语句必须用 CRTSQLPLI 和 CRTSQLCI 命令的 MARGINS 参数。

SQL 预编译假定主语言语句的语法是正确的。如果主语言的语法不正确，预编译不能正确地识别 SQL 语句和主变量说明。在预编译能通过的源语句格式是有限制的。应用语言编译器不能接受的文字和注释会进一步扰乱预编译源码检测过程并且引起错误。

SQL 的 INCLUDE 声明用来从 CRTSQLxxx(2)的 INCFILE 参数规定的文件中得到辅助输入。INCLUDE 语句可以从规定的成员中读输入直到成员结束。被包含的成员不能得到别的预编译 INCLUDE 语句，但可能得到应用程序和 SQL 语句。

其它的预处理器可在 SQL 预编译前处理源语句。然而任何在 SQL 预编译之前运行的预处理器必须要能通过 SQL 语句。

如果在应用程序源文中规定混合的 DBCS 常数，那么，源文件必须是混合的 CCSID。

(2)命令中的 xxx 指主语言的标识：CBL COBOL/400 语言为 CBL，ILE COBOL/400 为 CBLI，AS/400 PL/I 为 PLI，ILE C/400 为 CI，RPG/400 为 RPG，ILE RPG/400 为 PRGI。

16.1.2 源文件 CCSIDs

SQL 预编译器用源文件的 CCSID 读源记录，在处理 SQL INCLUDE 语句时，包括的源文件在必要情况下会被转换成初始源文件的 CCSID。如果 INCLUDE 源文件不能被转换成初始文件的 CCSID，那么就会产生错误。

SQL 预编译会用源 CCSID 处理 SQL 语句，这会影响可变字符。例如符号（ 在 CCSID 500 中分配为'BA'X，在版本 2.11 前，SQL 在 CCSID 37 的'5F'X 寻找符号（，这就是说，如果源文件的 CCSID 是 500，SQL 会到'BA'X 去找符号（。

如果源文件的 CCSID 是 65535，SQL 处理可变字符如同它的 CCSID 是 37，即 SQL 在'5F'X 处查找符号（。

16.1.3 预编译的输出

下面介绍由预编译产生的不同输出。

16.1.3.1 清单

输出清单会被送到 CRTSQLxxx 命令的 PRTFILE 指定的打印文件上，以下内容会输出列打印文件中：

预编译选项

在 CRTSQLxxx 命令中规定的选项。

预编译源文件

如果清单选项有效的话，输出有预编译分配的记录号的源文件语句。

预编译交叉引用

如果 OPTION 参数中规定了*XREF, 那么输出将提供交叉引用清单。这个清单给出

含有引用的主名和列名的 SQL 语句的记录号。

预编译诊断

提供诊断信息，显示错误的语句记录号。

打印文件的输出使用CCSID的值为65535,当此数据输出到打印文件时不进行转换。

16.1.3.2 临时源文件成员

由预编译处理的源语句写到QTEMP库中的QSQLTEMP中，（ILE RPG/400为QSQLTEMP1)。在预编译修改的源代码中，SQL语句已经被转换成注释和对SQL运行时的调用。临时源文件成员名与CRTSQLxxx命令的PGM或OBJ参数规定的名称一样,在输入给编译程序时不可修改此成员。在SQL生成QSQLTEMP或QSQLTEMP1文件时，它用源文件的CCISD值做为QSQLTEMP或QSQLTEMP1的CCSID值。

如果想稍后编译，在预编译后QSQLTEMP或QSQLTEMP1可以移到永久库中，不可以改变源成员的记录，否则编译失败。

SQL预编译用CRTSRCPF命令产生QSQLTEMP和QSQLTEMP1文件,如果命令的缺省值修改，那么结果不可预测。

16.1.3.3 预编译输出

预编译的输出能提供关于程序源文件的信息。要产生清单：

对于非 ILE 的预编译，在 CRTSQLxxx 命令的 OPTION 参数中规定*SOURCE(*SRC)和*XREF选项。

对于 ILE 预编译，在CRTSQLxxx命令中规定OPTION(*XREF)和OUTPUT(*PRINT)。预编译的输出格式如下：

```
5769ST1 V4R2M0 980228         Create SQL COBOL Program          CBLTEST1
       01/01/98 11:14:21   Page   1
Source type..............COBOL
Program name.............CORPDATA/CBLTEST1
Source file..............CORPDATA/SRC
Member...................CBLTEST1
 1 Options..................*SRC      *XREF     *SQL
Target release...........V4R2M0
INCLUDE file.............*LIBL/*SRCFILE
Commit...................*CHG
Allow copy of data.......*YES
Close SQL cursor.........*ENDPGM
Allow blocking...........*READ
Delay PREPARE............*NO
Generation level.........10
Printer file.............*LIBL/QSYSPRT
```

```
Date format..............*JOB
Date separator...........*JOB
Time format..............*HMS
Time separator ..........*JOB
Replace..................*YES
Relational database.....*LOCAL
User ....................*CURRENT
RDB connect method.......*DUW
Default Collection.......*NONE
Package name.............*PGMLIB/*PGM
Dynamic User Profile.....*USER
User Profile.............*NAMING
Sort Sequence............*JOB
Language ID..............*JOB
IBM SQL flagging.........*NOFLAG
ANS flagging.............*NONE
Text.....................*SRCMBRTXT
Source file CCSID........65535
Job CCSID................65535
 2  Source member changed on 01/01/98  10:16:44
```

1　调用 SQL 预编译时规定的选项清单。
2　源成员最后更改的日期。

```
5769ST1 V4R2M0 980228      Create SQL COBOL Program      CBLTEST1          01/01/98 11:14:21   Page   2
1 Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8    2  SEQNBR   3 Last Change

    1          IDENTIFICATION DIVISION.                                                  100
    2          PROGRAM-ID.  CBLTEST1.                                                    200
    3          ENVIRONMENT DIVISION.                                                     300
    4          CONFIGURATION SECTION.                                                    400
    5          SOURCE-COMPUTER. IBM-AS400.                                               500
    6          OBJECT-COMPUTER. IBM-AS400.                                               600
    7          INPUT-OUTPUT SECTION.                                                     700
    8          FILE-CONTROL.                                                             800
    9              SELECT OUTFILE, ASSIGN TO PRINTER-QPRINT,                             900
   10                 FILE STATUS IS FSTAT.                                           1000
   11          DATA DIVISION.                                                         1100
   12          FILE SECTION.                                                          1200
   13          FD  OUTFILE                                                            1300
   14              DATA RECORD IS REC-1,                                              1400
   15              LABEL RECORDS ARE OMITTED.                                         1500
```

```
01  REC-1.                                                          1600
    05  CC                      PIC X.                              1700
    05  DEPT-NO                 PIC X(3).                           1800
    05  FILLER                  PIC X(5).                           1900
    05  AVERAGE-EDUCATION-LEVEL PIC ZZZ.                            2000
    05  FILLER                   PIC X(5).                          2100
    05  AVERAGE-SALARY          PIC ZZZZ9.99.                       2200
01  ERROR-RECORD.                                                   2300
    05  CC                      PIC X.                              2400
    05  ERROR-CODE              PIC S9(5).                          2500
    05  ERROR-MESSAGE           PIC X(70).                          2600
WORKING-STORAGE SECTION.                                           2700
    EXEC SQL                                                       2800
      INCLUDE SQLCA                                                2900
    END-EXEC.                                                      3000
77  FSTAT                       PIC XX.                           3100
01  AVG-RECORD.                                                   3200
    05  WORKDEPT                PIC X(3).                         3300
    05  AVG-EDUC                PIC S9(4) USAGE COMP-4.           3400
    05  AVG-SALARY              PIC S9(6)V99 COMP-3.              3500
PROCEDURE DIVISION.                                                3600
*****************************************************************3700
*  这个程序要得到部门的平均教育程度和平均工资。                 *  3800
*                                                               *  3900
*****************************************************************4000
A000-MAIN-PROCEDURE.                                               4100
    OPEN OUTPUT OUTFILE.                                           4200
*****************************************************************4300
* 设置 WHENEVER 语句来处理 SQL 错误.                           *  4400
*****************************************************************4500
    EXEC SQL                                                       4600
      WHENEVER SQLERROR GO TO B000-SQL-ERROR                       4700
    END-EXEC.                                                      4800
*****************************************************************4900
* 说明游标                                                      *  5000
*****************************************************************5100
    EXEC SQL                                                       5200
      DECLARE CURS CURSOR FOR                                      5300
        SELECT WORKDEPT, AVG(EDLEVEL), AVG(SALARY)                 5400
          FROM CORPDATA.EMPLOYEE                                   5500
          GROUP BY WORKDEPT                                        5600
    END-EXEC.                                                      5700
*****************************************************************5800
*  打开游标                                                     *  5900
```

```
60          ************************************************************      6000
61               EXEC SQL                                                     6100
62                 OPEN CURS                                                  6200
63               END-EXEC.                                                    6300
```

1 记录号是预编译读源文件时分配的记录号，可用来标识出错信息和 SQL 运行时处理的源记录。

2 序号来自源记录，序号是用 SEU 编辑源成员可以看到序号。

3 源记录最后更改的日期如果为空，则指出自记录产生起从来没修改过。

```
 5769ST1 V4R2M0 980228          Create SQL COBOL Program        CBLTEST1       01/01/98 11:14:21   Page 3
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
64          ************************************************************      6400
65          *  给出所有结果行                                          *      6500
66          ************************************************************      6600
67               PERFORM A010-FETCH-PROCEDURE THROUGH A010-FETCH-EXIT         6700
68                 UNTIL SQLCODE IS = 100.                                    6800
69          ************************************************************      6900
70          *  关闭游标                                                *      7000
71          ************************************************************      7100
72               EXEC SQL                                                     7200
73                 CLOSE CURS                                                 7300
74               END-EXEC.                                                    7400
75               CLOSE OUTFILE.                                               7500
76               STOP RUN.                                                    7600
77          ************************************************************      7700
78          *  输入一行且把信息传给输出记录.                          *      7800
79          ************************************************************      7900
80           A010-FETCH-PROCEDURE.                                            8000
81               MOVE SPACES TO REC-1.                                        8100
82               EXEC SQL                                                     8200
83                 FETCH CURS INTO :AVG-RECORD                                8300
84               END-EXEC.                                                    8400
85               IF SQLCODE IS = 0                                            8500
86                 MOVE WORKDEPT TO DEPT-NO                                   8600
87                 MOVE AVG-SALARY TO AVERAGE-SALARY                          8700
88                 MOVE AVG-EDUC TO AVERAGE-EDUCATION-LEVEL                   8800
89                 WRITE REC-1 AFTER ADVANCING 1 LINE.                        8900
90           A010-FETCH-EXIT.                                                 9000
91               EXIT.                                                        9100
92          ************************************************************      9200
93          *  有一个 SQL 错误，把错误号传给错误记录，且停止运行。    *      9300
```

```
   94         *                                                              *                9400
   95         ****************************************************************                9500
   96          B000-SQL-ERROR.                                                                9600
   97              MOVE SPACES TO ERROR-RECORD.                                               9700
   98              MOVE SQLCODE TO ERROR-CODE.                                                9800
   99              MOVE "AN SQL ERROR HAS OCCURRED" TO ERROR-MESSAGE.                         9900
  100              WRITE ERROR-RECORD AFTER ADVANCING 1 LINE.                                10000
  101              CLOSE OUTFILE.                                                            10100
  102              STOP RUN.                                                                 10200
* * * * *  E N D  O F  S O U R C E  * * * * *

5769ST1 V4R2M0 980228     Create SQL COBOL Program     CBLTEST1           01/01/98 11:14:21   Page 4
CROSS REFERENCE
 1                              2         3
Data Names                    Define    Reference
AVERAGE-EDUCATION-LEVEL         20       IN REC-1
AVERAGE-SALARY                  22       IN REC-1
AVG-EDUC                        34       SMALL INTEGER PRECISION(4,0) IN AVG-RECORD
AVG-RECORD                      32       STRUCTURE
                                         83
AVG-SALARY                      35       DECIMAL(8,2) IN AVG-RECORD
BIRTHDATE                       55       DATE(10) COLUMN IN CORPDATA.EMPLOYEE
BONUS                           55       DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
B000-SQL-ERROR                  ****     LABEL
                                         47
CC                              17       CHARACTER(1) IN REC-1
CC                              24       CHARACTER(1) IN ERROR-RECORD
COMM                            55       DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
CORPDATA                        ****    4  COLLECTION
                                        5  55
CURS                            53       CURSOR
                                         62 73 83
DEPT-NO                         18       CHARACTER(3) IN REC-1
EDLEVEL                         ****     COLUMN
                                         54
                                           6
EDLEVEL                         55       SMALL INTEGER PRECISION(4,0) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
EMPLOYEE                        ****     TABLE IN CORPDATA                                                7
                                         55
EMPNO                           55       CHARACTER(6) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
ERROR-CODE                      25       NUMERIC(5,0) IN ERROR-RECORD
ERROR-MESSAGE                   26       CHARACTER(70) IN ERROR-RECORD
ERROR-RECORD                    23       STRUCTURE
FIRSTNME                        55       VARCHAR(12) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
```

```
FSTAT                          31        CHARACTER(2)
HIREDATE                       55        DATE(10) COLUMN IN CORPDATA.EMPLOYEE
JOB                            55        CHARACTER(8) COLUMN IN CORPDATA.EMPLOYEE
LASTNAME                       55        VARCHAR(15) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
MIDINIT                        55        CHARACTER(1) COLUMN (NOT NULL) IN CORPDATA.EMPLOYEE
PHONENO                        55        CHARACTER(4) COLUMN IN CORPDATA.EMPLOYEE
REC-1                          16
SALARY                         ****      COLUMN
                                         54
SALARY                         55        DECIMAL(9,2) COLUMN IN CORPDATA.EMPLOYEE
```

1　数据名是用在源语句中的符号名。

2　定义列规定定义名字的行号，行号是由 SQL 预编译生成的。***表示目标没定义或预编译不认识这个说明。

3　引用列包括二类信息：

　　什么样的符号名定义为 4

　　符号名出现的行号 5

如果符号名引用一个有效的主变量，也要标识数据类型 6 或数据结构 7。

```
5769ST1 V4R2M0 980228     Create SQL COBOL Program    CBLTEST1   01/01/98 11:14:21  Page 5
  CROSS REFERENCE
  SEX                             55        CHARACTER(1) COLUMN IN CORPDATA.EMPLOYEE
  WORKDEPT                        33        CHARACTER(3) IN AVG-RECORD
  WORKDEPT                        ****      COLUMN
  54 56
  WORKDEPT                        55        CHARACTER(3) COLUMN IN CORPDATA.EMPLOYEE
  No errors found in source
  102 Source records processed
  * * * * *  E N D  O F  L I S T I N G  * * * * *
```

图 16-1 样板 COBOL 预编译输出格式

16.2　非 ILE 预编译命令

DB2 查询管理和 SQL 开发工具包括下列主语言的非 ILE 预编译命令：

CRTSQLCBL，CRTSQLPLI，CRTSQLRPG，某些选项只适合于一定的语言。例如，选项*APOST 和*QUOTE 仅用于 COBOL，它们不包括在其它语言的命令中。

16.2.1　编译非 ILE 应用程序

预编译成功地完成后，除非规定了*NOGEN，SQL 预编译会自动调用主语言编译器。运行

规定了程序名、源文件名、预编译生成的源成员名、说明和 USRPRF 的 CRTxxxPGM 命令。

在这些语言中，传送下列参数：

对 COBOL，*QUOTE 或*APOST 传给 CRTCBLPGM

对 RPG 和 COBOL，SAAFLAG（*FLAG）传给 CRTxxxPGM 命令

对 RPG 和 COBOL，在 CRTxxxPGM 命令中规定 CRTSQLxxx 命令中的 STRSEQ 和 LANGID 参数。

对 RPG 和 COBOL，在 CRTxxxPGM 命令中要规定 CVTOPT(*DATETIME *VARCHAR)。

对 RPG 和 COBOL，在 CRTxxxPGM 中规定 CRTSQLxxx 命令中的 TGTRLS 的参数值。

在 CRTPLIPGM 命令不规定 TGTRLS。程序可以保存或重存由 CRTSQLPLI 命令中 TGTRLS 参数规定的级别。

对 PL/1，MARGING 放在临时源文件中。

对所有语言，在 CRTxxxPGM 命令中规定 CRTSQLxxx 命令中的 REPLACE 参数。

如果软件包做为预编译处理的一部分生成，则 CRTSQLPGM 命令规定 CRTSQLxxx 中的 REPLACE 做参数值。

对于所有语言，如果规定 USRPRF(*USER)或 USRPRF(*NAMING)且用系统命名方式(*SYS)，那么在 CRTxxxPGM 命令中规定 USRPRF(*USER)，如果规定用 USRPRF(*OWNER)或 USRPRT(*NAMING)且用 SQL 命名方式(*SQL)，则在 CRTxxxPGM 命令规定 USRPRF(*OWNER)。

CRTxxxPGM 命令所有其他参数都用缺省值。

可以在预编译命令中规定 OPTION 为*NOGEN 来中断对主语言编译器的调用。*NOGEN 规定不调用主语言编译器。使用 CRTSQLxxx 命令中的目标名作为成员名，预编译器在 QTEMP/QSQLTEMP 文件中生成源成员。现在，可明确调用主语言编译器，规定源成员在 QTEMP/QSQLTEMP 文件中，且修改缺省值。如果预编译和编译分开执行，那么可用 CRTSQLPKG 为分布式程序建立 SQL 软件包。

注：在执行 CRTxxxPGM 命令前不要改变 QTEMP/QSQLTEMP 的源成员，否则编译会失败。

16.3 ILE 预编译命令

在 DB2 查询管理和 SQL 开发工具中，有下列 ILE 预编译命令：CRTSQLCI、CRTSQLCBLI 和 CRTSQLRPGI。这是分别对应于下列主语言用的预编译命令：ILE C／400、ILE COBOL／400、ILE RPG／400。可以规定要求的参数及其余参数取缺省值，缺省值只用于你正在用的语言，例如：选项*APOST 和*NOTE 是 COBOL 特有的，其它语言没有。

16.3.1 编译 ILE 应用程序

除非规定*NOGEN，完成预编译后 SQL 预编译器自动调用主语言编译器。如果规定*MOUDLE 选项，SQL 预编译器执行 CRTxxxMOD 生成模块。如果规定*PGM 选项，SQL 预编译器执行 CRTBNDxxx 生成程序。如果规定*SRVPGM，SQL 执行 CRTxxxMOD 命令生成模块，接着用 CRTSRVPGM 生成服务程序。

在这些语言中，传送下列参数：

如果在 CRTSQLxxx 中规定 DBGVIEW(*SOURCE)，那么在 CRTxxxMOD 和 CRTBNDxxx 中要规定 DBGVIEW(*ALL)。

如果在 CRTSQLxxx 中规定 OUTPUT(*PRINT)，那么它传送给 CRTxxxMOD，CRTBNDxxx。

如果在 CRTSQLxxx 中规定 OUTPUT(*NONE)，那么在 CRTxxxMOD 或 CRTBNDxxx 中不能规定。

在 CRTxxxMOD、CRTBNDxxx、CRTSRVPGM 中可规定 CRTSQLxxx 的 TGTRLS 参数值。

在 CRTxxxMOD、CRTBNDxxx、CRTSRVPGM 中可规定 CRTSQLxxx 的 REPLACE 参数值。

如果软件包作为预编译生成的一部分，那么 CRTSQLPKG 中规定 CRTSQLxxx 的 REPLACE 参数值。

如果 OBJTYPE 是*PGM 或*SRVPGM，也规定了 USRPRF(*USER)或 USRPRF(*NAMING)且用系统命名方式(*SYS)，那么，CRTBNDxxx 或 CRTSRVPGM 要规定 USRPRF(*USER)。

如果 OBJTYPE 是*PGM 或*SRVPGM,且规定了 USRPRF(*OWNER)或 USRPRF(*NAMING)且用 SQL 命名方式(*SQL)，则在 CRTBNDxxx 或 CRTSRVPGM 命令中要规定 USRPRF(*OWNER)。

在 C 中，在临时源文件中要设置 MARGINS。

在 COBOL 中，传送 CRTBNDCBL 或 CRTCBLMOD 上的*QUOTE 或*APOST。

在 RPG 和 COBOL 中，在 CRTxxxMOD 和 CRTBNDxxx 中规定 CRTSQLxxx 的 SRTSEQ 和 LANGID 参数。

在 COBOL 中，CRTCBLMOD 和 CRTBNDCBL 中总要规定 CVTOPT(*VARCHAR *DATETIME *PICGGRAPHIC *FLOAT)。

在 RPG 中，如果规定了 OPTION(*CVTDT)，那么，在 CRTRPGMOD 和 CRTBNDRPG 中要规定 CVTOPT(*DATETIME)。

在预编译命令的 OPTION 中规定*NOGEN 就可中断调用主语言编译器。*NOGEN 规定不调用主语言编译器。预编译用 CRTSQLxxx 规定的程序名作为成员名，在 QTEMP/ QSQLTEMP 文件中生成源成员。然后，可以明确调用主语言编译，规定 QTEMP/QSQLTEMP 文件的源成员且修改缺省值。如果分开做预编译与编译，那么可用 CRTSQLPKG 为分布式程序建立 SQL 软件包。

如果晚一些生成程序或服务程序，那么在 CRTBNDxxx、CRTPGM 或 CRTSRVPGM 命令中就不能正确地设置 USRPRF 参数。只有 USRPRF 参数改正后，SQL 程序运行才有可预见性。如果用系统命名方式，那么 USRPRF 参数必须设置为*USER，如果用 SQL 命名方式，则 USRPRF 参数应设置为*OWNER。

16.4 解释应用程序编译错误

请注意：如果把预编译和编译分开，且源程序引用外部描述文件，那么被引用文件在预编译和编译之间不允许更改，如若不然，由于在临时修改的字段定义没有在临时源成员中体现修改，将导致不可预测的结果发生。

外部描述文件的实例是：

在 COBOL 中 COPY DDS

PL/1 中的％INCLUDE

C 中的＃pragma mapinc 和 #include

RPG 中的数据结构。

当 SQL 预编译没有识别主变量时,试图编译源文件,在编译没识别出 EXEC SQL 语句时，忽视这些错误。编译器是按 SQL 预编译器定义来解释主变量说明的。

16.4.1 编译时错误及警告信息

下面介绍的条件可能会在编译过程中产生错误或警告信息。

16.4.1.1　在 PL/1 或 C 的编译中（略）

16.4.1.2　在 COBOL 的编译中（略）

16.4.1.3　在 RPG 的编译中

在第 8-16 列没有写 EXEC　SQL，且在第 7 列前没有/号，SQL 预编译就不认识 SQL 语句，因此，它会按原样传给编译器。

16.5　联编应用程序

运行应用程序前，必须建立程序和表及视图之间的联系，这个过程叫联编。联编结果是一个可访问方案，访问方案是描述满足每个 SQL 查询必需动作的一个控制结构。访问方案包含程序信息和程序要用到数据的相关信息。

非分布式 SQL 程序，访问方案存储在程序中。分布式 SQL 程序在 CRTSQLxxx 命令中规定了 RDB 参数的访问方案存储在规定的相关数据库的 SQL 软件包中。

在生成程序目标时，SQL 会自动联编且生成访问方案。对非 ILE 编译，它作为成功的 CRTxxxPGM 结果。对 ILE 编译，它表示成功的 CRTBNDxxx，CRTPGM 或 CRTSRVPGM 的结果。如果 AS/400　DB2 在运行中检测到访问方案无效（例如引用的表在不同库中）或对数据库的修改改变了性能（例如附加的索引），那么自动产生新的访问方案。联编做以下三件事：

1. 用数据库中的描述使 SQL 语句有效。在联编中检查 SQL 语句的有效表、视图和列名。如果在预编译或编译时某一表或视图不存在，则在运行时检查有效性。如果运行时表或视图不存在，会返回负的 SQLCODE。
2. 选择索引来访问程序所用的数据。选索引时，要考虑表大小和其它的因素，它会考虑所有能访问数据的索引且决定用哪一个来做访问路径。
3. 它尝试建立访问方案。若所有 SQL 语句都有效，会建立联编过程且在程序中存储访问方案。

如果程序访问的表或视图的特性已改变，访问方案不再有效。若试图运行有无效访问方案的程序时，系统会试图自动重建访问方案。若无法重建访问方案，则返回负的 SQLCODE。此时，必须修改 SQL 语句，重新执行 CRTSQLxxx。

例如：如果程序包括引用 TABLEA 的 COLUMNA 的 SQL 语句，且用户删除、重建 TABLEA 而使 COLUMNA 不存在。当调用程序时，因为 COLUMNA 不存在，自动重联编不会成功。此时，必须修改程序源码，重新执行 CRTSQLxxx 命令。

16.5.1　程序引用

在生成程序时，SQL 语句引用的所有集合、表、视图、SQL 软件包和索引都被放在库中目标信息区（OIR）中。

可用 DSPPGMREF 命令显示程序中引用的所有目标。如果用 SQL 命名转换，存在 OIR 中的库名有下列三种之一的方式：

1. 若 SQL 名被完全限定，集合名做名字限定存储。
2. 若 SQL 名没完全限定且没有规定 DFTRDBCOL 参数，语句的授权 ID 做名字限定存储。
3. 若 SQL 名没有完全限定且规定了 DFTRDBCOL，在 DFTRDBCOL 参数中规定的集合名做

名字限定存储。

若用系统命名转换，文件库名可用三种方法存储在 OIR 中，存储的库名可用下列三种之一的方式：

1.若目标名完全限定，库名做限定名存储。

2.若目标没完全限定，且没规定 DFTRDBCOL 参数，用*LIBL。

3.若 SQL 名没完全限定，而规定了 DFTRDBCOL 参数，在 DFTRDBCOL 规定的集合名做限定名存储。

16.6 显示预编译选项

成功编译 SQL 应用程序后，可用 DSPMOD，DSPPGM，DSPSRVPGM 命令显示 SQL 预编译时规定的一些选项。这些信息在修改程序源码时可能会用到。当要再次编译程序时，这些 SQL 预编译选项可在 CRTSQLxxx 中规定。

可用 PRTSQLINF 命令确定 SQL 预编译时规定的一些选项。

16.7 运行嵌入 SQL 的程序

在预编译、编译成功的完成后，运行嵌入 SQL 语句的主语言程序同运行主语言程序一样。在系统命令行写：

CALL 程序名

详细信息，请看 CL 程序设计。

16.7.1 OS/400 DDM 考虑

SQL 不支持通过 DDM 访问远程文件，SQL 支持通过 DRDA 的远程访问。

16.7.2 替换考虑

可用替换（OVRDBF 规定）直接引用不同的表或视图，或修改程序或 SQL 软件包的某些操作属性，若规定了替换，要处理以下参数：

TO FILE

MBR

SEQONLY

INHWRT

WAITRCD

所有其它替换参数都忽略，SQL 软件包语句的替换由以下两步完成：

1.在 OVRDBF 中规定 OVRSCOPE(*JOB)参数。

2.用 SBMRMTCMD 命令送给应用服务器。

详细信息，请看 AS/400 DB2 数据库编程、数据管理。

16.7.3 SQL 返回码

在本书原文附录 B 提供一系列 SQL 返回码、SQLCODEs 和 SQLSTATEs。

第十七章 使用交互 SQL

这一章介绍怎样用交互 SQL 运行 SQL 语句和如何使用提示功能，也介绍浏览信息和使用

交互 SQL 技术。SQL 的使用，见第二章。使用远程交互 SQL 的一些特别考虑，请看 17.1.9。

17.1 交互 SQL 的基本功能

用交互 SQL，程序员或数据库管理员可以迅速简便地定义、更新、删除、查找数据，用来检测、做问题分析、数据库维护。用交互 SQL，程序员可以在应用程序运行 SQL 语句前对其进行检查并且可在表中插几行。数据库管理员用交互 SQL 可以分配及取消授权，生成和丢弃集合、表、视图或从系统目录表中选取信息。

运行交互 SQL 语句后，会显示完成信息或出错信息。另外，在长时间运行语句期间也会显示状态信息。

把光标放在信息行上按 F1，可得到此信息的帮助。

交互 SQL 的基本功能为：

— 写交互 SQL 语句且运行它
— 重试和编辑说明
— SQL 语句的提示
— 翻页通过先前语句和信息
— 调用对话服务
— 激活目录选取功能
— 退出交互 SQL

提示功能允许键入完整的 SQL 语句和部分 SQL 语句，按 F4 键，会得到语句的句法提示，按 F4 键，也可得到所有 SQL 语句的菜单，从这个菜单可以选择语句及提示语句的语法。

列表选择功能允许从授权的关系数据库、集合、表、视图、列、约束或软件包列表中进行选择。

可把从列表中选择的选项插入到光标指定处的 SQL 语句中。

对话服务功能可以做：

— 修改对话属性
— 打印当前对话
— 从当前对话中取消所有项
— 把对话保存到源文件中

17.1.1 启动交互 SQL

在 AS/400 命令行写 SRTSQL 即启动交互 SQL，关于命令及其参数的完整描述，见原文附录 D。

出现输入 SQL 语句显示，这是交互 SQL 主要的显示，从这个显示，可以进入 SQL 语句及使用：

F4＝提示
F13＝对话服务
F16＝选择集合
F17＝选择表
F18＝选择列

---

*　　*
*　　Enter SQL Statements　　*
*　　*

```
* Type SQL statement, press Enter.                                    *
*     Current connection is to relational database rdjacque.          *
* ===>_____________________________________________________________   *
*     _____________________________________________________________   *
*     _____________________________________________________________   *
*     _____________________________________________________________   *
*     _____________________________________________________________   *
*     _____________________________________________________________   *
*     _____________________________________________________________   *
*     _____________________________________________________________   *
*     _____________________________________________________________   *
*     _____________________________________________________________   *
*     _____________________________________________________________   *
*     _____________________________________________________________   *
*     _____________________________________________________________   *
*     _____________________________________________________________   *
*     _____________________________________________________________   *
*                                                                     *
*                                                            Bottom   *
* F3=Exit   F4=Prompt   F6=Insert line   F9=Retrieve   F10=Copy line   *
* F12=Cancel            F13=Services     F24=More keys                *
*                                                                     *
*                                                                     *
```

```
  Press F24=More keys to view the remaining function keys.
  *                                                                  *
* *     _____________________________________________________        *
* *     _____________________________________________________        *
* *     _____________________________________________________        *
* *                                                  Bottom          *
  *                                                                  *
* * F14=Delete line   F15=Split line     F16=Select collections (libraries)  *
* * F17=Select tables                    F18=Select columns   F24=More keys  *
* *           (files)                              (fields)          *
  *                                                                  *
```

注：若用系统命名转换，括号中的名会取代它上面出现的名。

一个交互式对话由下面内容组成：

规定的 STRSQL 参数值。

对话中输入的 SQL 语句，跟在每条 SQL 语句相应的信息。

用对话服务功能改变的参数值。

已经选择的列表。

交互 SQL 提供唯一的对话 ID（由用户 ID 和当前工作站 ID 构成），它允许有相同 USERID 的多个用户同时从多个工作站使用交互 SQL，也可以，一个 USERD 同时在一个工作站运行多个交互 SQL 对话。

若 SQL 对话已存在且要再进入，忽略 STRSQL 规定的任何参数，使用已存在的 SQL 对话参数。

17.1.2 使用语句入口功能

选择交互 SQL 最先出现的就是语句入口功能，处理完每条交互 SQL 语句后都会返回到语句入口功能。

用语句入口功能，可以写或提示完整的 SQL 语句，然后按 Enter 键将其提交处理。

17.1.2.1 键入语句

写在命令行的语句可能不止占一行。在交互 SQL 中不能写 SQL 语句的注释。在处理语句时，语句和结果信息会在显示中上移，然后可写另一语句。

若 SQL 认识一条语句，但有语法错误，显示中的语句及结果文本信息也会上移。在输入区，语句的拷贝与放在有句法错处的光标一起显示。可将光标放至信息处按 F1 键得到关于此错误的更多信息。

可以用翻页键看先前的语句、命令和信息，把光标放在以前用过的语句上且按 F9 键，可把此语句拷贝到输入区，若用更多空间写 SQL 语句，可在显示中向下翻页。

17.1.3 提示

提示功能对要使用的语句提供有关句法必要的信息，在三种语句处理方式(*RUN、*VLD、*SYN)下都可使用提示功能。

在使用提示时有两个选项：

按 F4 键前写语句的动词。

分析语句后，完整的子句会填到提示显示中。

若键入 SELECT 按 F4 键，会出现下面的显示：

```
*                                                                    *
*                          Specify SELECT Statement                  *
*                                                                    *
* Type SELECT statement information.  Press F4 for a list.           *
*                                                                    *
*   FROM tables . . . . . _____________________________________      *
*   SELECT columns  . . . . . _____________________________________  *
*   WHERE conditions  . . . . . _____________________________________*
*   GROUP BY columns  . . . . . _____________________________________*
*   HAVING conditions . . . . . _____________________________________*
*   ORDER BY columns  . . . . . _____________________________________*
*   FOR UPDATE OF columns . . . _____________________________________*
*                                                                    *
*                                                          Bottom    *
```

```
* Type choices, press Enter.                                                  *
*                                                                            *
*   DISTINCT rows in result table  . . . . . .  N   Y=Yes, N=No              *
*   UNION with another SELECT  . . . . . . . .  N   Y=Yes, N=No              *
*   Specify additional options . . . . . . . .  N   Y=Yes, N=No              *
*                                                                            *
*                                                                            *
*                                                                            *
* F3=Exit         F4=Prompt   F5=Refresh   F6=Insert line   F9=Specify subquery   *
* F10=Copy line   F12=Cancel  F14=Delete line  F15=Split line  F24=More keys     *
*                                                                            *
*                                                                            *
```

在‘输入 SQL 语句’显示上键入任何内容前按 F4 键，会显示一语句列表，它随当前交互 SQL 语句处理方式而有所不同。在非*NONE 的语言句法检查方式下，列表包含所有的语句。对运行和有效性方式，只显示可在交互 SQL 中运行的语句。可选择要使用的语句号，系统会给出相应的提示。

若没有键入任何内容按 F4 键，出现下面的显示：

```
*                                                                            *
*                              Select SQL Statement                          *
*                                                                            *
* Select one of the following:                                               *
*                                                                            *
*      1. ALTER TABLE                                                        *
*      2. CALL                                                               *
*      3. COMMENT ON                                                         *
*      4. COMMIT                                                             *
*      5. CONNECT                                                            *
*      6. CREATE COLLECTION                                                  *
*      7. CREATE INDEX                                                       *
*      8. CREATE TABLE                                                       *
*      9. CREATE VIEW                                                        *
*     10. DELETE                                                             *
*     11. DISCONNECT                                                         *
*     12. DROP COLLECTION                                                    *
*     13. DROP INDEX                                                         *
*     14. DROP PACKAGE                                                       *
*                                                              More...       *
* Selection                                                                  *
*      __                                                                    *
*                                                                            *
```

```
* F3=Exit    F12=Cancel                                                  *
*                                                                       *
*                                                                       *
```

如果在提示显示上按 F21，提示会显示格式化的语句。

在提示中按 Enter 键，由提示屏建立的语句会被插入到对话区。当语句处理方式为*RUN 时，则运行此语句。如出错，提示会保留控制。

17.1.3.1　语法检查

当输入 SQL 语句进入提示时，会做检查。提示不接收有语法错误的语句。必须纠正语法错误或删除错误部分，否则无法提示。

17.1.3.2　语句处理方式

可在‘修改对话属性’显示中选择语句处理方式，用*RUN 或*VLD 方式下，仅提示在交互 SQL 中运行的语句。用*SYN，允许全部 SQL 语句。在*SYN 和*VLD 方式下，不实际运行语句，只检查句法和目标是否存在。

17.1.3.3　子查询

凡在有 WHERE 和 HAVING 子句的显示中都可进行子查询。要看子查询的显示，在光标处于 WHERE 或 HAVING 输入行时，按 F9 键，会出现一个显示，允许你输入子选择信息。在按 F9 键时，如果光标在子查询的括号内，子查询的信息就会添入下一显示。若光标在括号外，则下一显示为空，更多信息请看 5.6。

17.1.3.4　CREATE　TABLE 提示

当提示 CREATE　TABLE 时，可分别输入列定义。把光标放在列定义段，按 F4 键，会显示出输入列定义信息的空间。

若输入多于 18 个字符的列名，按 F20 键会显示全屏，出现可容 30 个字符的窗口。

编辑键，F6＝插入行，F14＝删除行，可在列定义列表中加、删列定义入口。

17.1.3.5　输入 DBCS 数据

处理多行 DBCS 数据与‘输入 SQL 语句’显示和 SQL 提示的方法一样。每行都有相同数目的转入，转出字符。若处理多行的 DBCS 数据输入，要去掉多余的转入转出字符，如果一行的最后一列有一个转入字符，下行第一列有一个转出字符，在合并两行时，由提示删除转入转出字符。若一行的最后两列有一个转入字符后跟单节空白，且下行第一列有一转出字符，在两行合并会删除转入、空白和转出字符。这个移动允许 DBCS 信息作为连续的字符串来读。

假定输入下列的 WHERE 条件，会在两行的每个字串区域的开始和末尾显示转换字符。

```
*                                                                       *
*                          Specify SELECT Statement                     *
*                                                                       *
* Type SELECT statement information.  Press F4 for a list.              *
*                                                                       *
```

```
*   FROM tables . . . . . .  TABLE1_____________________________________    *
*   SELECT columns  . . . . .  *_____________________________________    *
*   WHERE conditions  . . . . .  COL1 = '<AABBCCDDEEFFGGHHIIJJKKLLMMNNOOPPQQ>    *
*                              <RRSS>'_____________________________________    *
*   GROUP BY columns  . . . . .  _____________________________________    *
*   HAVING conditions . . . . .  _____________________________________    *
*   ORDER BY columns  . . . . .  _____________________________________    *
*   FOR UPDATE OF columns . . .  _____________________________________    *
*                                                                          *
```

在按 Enter 键时，字符串放在一起，删除额外的转换字符，语句在显示中有如下内容：

```
SELECT * FROM TABLE1 WHERE COL1 = '<AABBCCDDEEFFGGHHIIJJKKLLMMNNOOPPQQRRSS>'
```

17.1.4 使用列表选择功能

在某个提示显示按 F4 键，或在‘输入 SQL 语句’显示中按 F16、F17、F18 键，可用选择功能。按下功能键后，会出现一可选列表，其中包括授权的数据库、集合、表、视图、列、约束或软件包。若你要表的列表，又没先选择集合，则告诉你要先选集合。

在列表上可选一项或多项，用数字规定它们出现的顺序。退出列表功能时，所做的选择会插入到显示中光标的位置上。

可选最感兴趣的列表。例如，若要一列的列表而不能保证正确地选择列，则可，按 F18 键来选列，然后，从列中，按 F17 键选择表，若首先选择表列表，表名会插入到语句中。

任何时候，都可以在‘输入 SQL 语句’显示中，输入一个 SQL 语句来请求一个列表，从列表所做的选择插入到‘输入 SQL 语句’显示中，它们用在列表显示中你规定的数字顺序插入到光标所指的位置中。虽然选择的列表信息加上了，但也必须输入语句的键字。

列表功能尽可能提供选择的列、表和 SQL 软件包需要的限定，然而，有时列表功能不能断定 SQL 语句的意图。你需要重看 SQL 语句，验证列、表、SQL 软件包是否正确的限定。

17.1.4.1 使用列表选择功能的例子

下例说明使用列表功能来建立 SELECT，假定你有：

刚在 AS/400 命令行上输入 STRSQL 启动交互 SQL。

没做列表选择和输入。

选定*SQL 命名转换。

开始用 SQL 语句：

1.在第一个 SQL 输入行输入 SELECT

2.在第二个行写 FROM

3.将光标放在 FROM 后

```
*                                                                          *
*                                                                          *
* Type SQL statement, press Enter.                                         *
* ===> SELECT                                                             *
```

```
*      FROM _                                                          *
*                                                                      *
*                                                                      *
*                                                                      *
*                                                                      *
*                                                                      *
*                                                                      *
```

4.按 F17＝选表，出现表列表，因为想在 FROM 后写表名。
也可出现集合列表而不是表，因为你刚进入 SQL 对话，而未选择集合来处理。
5.在 YOURCOLL2 集合的 Seq 列中写 1

```
*                                                                      *
*                                                                      *
*                                                                      *
* Type sequence numbers (1-999) to select collections, press Enter.    *
*                                                                      *
* Seq   Collection      Type   Text                                     *
*       YOURCOLL1       SYS    Company benefits                        *
* 1     YOURCOLL2       SYS    Employee personal data                  *
*       YOURCOLL3       SYS    Job classifications/requirements        *
*       YOURCOLL4       SYS    Company insurances                      *
*                                                                      *
*                                                                      *
*                                                                      *
```

6.按 Enter 键.
出现‘选择和排序表’显示，给出 YOURCALL2 集合中的表。
7.接着在 PEOPLE 表的 Seq 列写 1。

```
*                                                                      *
*                                                                      *
* Type sequence numbers (1-999) to select tables, press Enter.         *
*                                                                      *
* Seq  Table                  Collection    Type  Text                 *
*      EMPLCO                 YOURCOLL2     TAB   Employee company data *
* 1    PEOPLE                 YOURCOLL2     TAB   Employee personal data *
*      EMPLEXP                YOURCOLL2     TAB   Employee experience  *
*      EMPLEVL                YOURCOLL2     TAB   Employee evaluation reports *
```

```
*      EMPLBEN                YOURCOLL2     TAB   Employee benefits record      *
*      EMPLMED                YOURCOLL2     TAB   Employee medical record       *
*      EMPLINVST              YOURCOLL2     TAB   Employee investments record   *
*                                                                               *
*                                                                               *
```

8.按 enter 键，再次出现‘输入 SQL 语句’的显示，上面有表名 YOURCOLL2.PEOPLE，插在 FROM 后，表名用集合名以*SQL 命名方式限定。

```
*                                                                               *
*                                                                               *
* Type SQL statement, press Enter.                                              *
* ===> SELECT                                                                   *
*      FROM YOURCOLL2.PEOPLE _                                                  *
*                                                                               *
*                                                                               *
*                                                                               *
*                                                                               *
*                                                                               *
*                                                                               *
```

9.把光标定位在 SELECT 后。
10.按 F18＝选列，出现列的列表，选择和排序列显示，给出在 PEOPLE 表中的所有列。
11.在 NAME 的 Seq 列中写 2。
12.在 SOCSEC 的 Seq 列中写 1。

```
*                                                                               *
*                                                                               *
* Type sequence numbers (1-999) to select columns, press Enter.                 *
*                                                                               *
* Seq  Column               Table                Type          Digits  Length   *
* 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 键；

再出现‘输入 SQL 语句’显示，在 SELECT 后出现 SOCSEC、NAME。

```
*                                                                    *
*                                                                    *
* Type SQL statement, press Enter.                                   *
* ===> SELECT SOCSEC, NAME                                          *
*      FROM YOURCOLL2.PEOPLE                                         *
*                                                                    *
*                                                                    *
*                                                                    *
*                                                                    *
*                                                                    *
*                                                                    *
*                                                                    *
```

14.按 Enter 键。

你生成的语句现在运行。

一旦使用列表功能，选择的值会一直有效，直到修改它们或在‘修改会话属性’显示上修改集合列表。

17.1.5 会话服务描述

在‘输入 SQL 语句’显示中按 F13 键可看到交互 SQL‘会话服务’显示。

从这个显示可改变会话属性，打印、清除或把会话存到源文件中。

选项 1（改会话属性）显示‘修改会话属性’显示，由此可选择影响交互 SQL 会话的当前值。

可修改以下会话属性：

落实控制属性
语句处理控制
SELECT 输出设置
集合列表
列出类型（所有的系统和 SQL 目标或只有 SQL 目标）
显示数据时，数据刷新选项
允许拷贝数据选项
命名选项
编程语言
日期格式
时间格式
日期分隔符
时间分隔符
小数点表示
SQL 串界线

分类顺序

语言标识

选项 2（打印当前会话），给出‘修改打印’的显示。让你立即打印当前会话而后继续工作，会有打印机信息的提示。所有输入的 SQL 语句及所有显示的信息会象出现在‘输入 SQL 语句’显示中那样打印出来。

选项 3（从当前会话删除所有项），可以从‘输入 SQL 语句’显示中和会话历史记录中删除所有 SQL 语句，有提示保证能实际删除不要的信息。

选项 4（把会话保存到源文件）可调用‘修改源文件’显示，让你把会话保存在源文件中，会提示你源文件名，此功能用 SEU 在主语言程序嵌入源文件。

注：选项 4 可用在 SQL 的 HLL 中嵌入源 SQL 语句，由选项 4 建立的源文件可编缉或用作 RUNSQLSTM 命令的输入源文件。

17.1.6 结束交互 SQL

在‘输入 SQL 语句’显示上按 F3＝退出，可结束交互 SQL 环境且做下列之一：

1.保存、退出会话，结束交互 SQL，保存当前会话，并用来启动下次交互 SQL。

2.不保存、退出会话，结束交互 SQL 且不保存会话。

3.恢复会话，不结束交互 SQL，返回到‘输入 SQL 语句’显示，当前会话参数仍有效。

4.在源文件中保存会话，出现‘修改源文件’显示，允许选择保存会话的文件，在交互 SQL 不能再恢复及处理这个会话。

注：

1.用选项 4 可在 HLL 中嵌入源 SQL 语句，用 SEU 把语句拷入程序中，源文件也可编辑，用做 RUNSQLSTM 命令的输入源文件。

2.若行已修改，且锁定了当前工作单元，而又试图退出交互 SQL，会显示警告信息。

17.1.7 使用已有的 SQL 会话

若在‘结束交互 SQL’显示中用选项 1，保存了一个交互 SQL 会话，可在任何工作站上恢复会话，若用选项 1 在不同工作站上保存多个会话，交互 SQL 会首先尝试恢复你的工作站上的会话。若没有合适的会话，交互 SQL 就会把检索范围扩大到所有属于你用户 ID 的会话。如果没有可用的用户 ID 会话，系统会为用户 ID 和当前工作站建立一个新的会话。

例如：在工作站 1 保存会话，在工作站 2 保存另一个会话，而你在工作站 1 上工作，交互 SQL 会首先在工作站 1 上恢复保存的会话，若正在使用会话，交互 SQL 会在工作站 2 上恢复保存的会话，若这个会话也在用，那么系统就会为工作站 1 建立第二个会话。

可是，假定你在工作站 3 上工作，想用与工作站 2 相联的会话，要先用‘结束交互 SQL’显示中的选项 2 从工作站删除会话。

17.1.8 SQL 会话的恢复

若先前的 SQL 会话非正常结束，交互 SQL 会在启动下一次会话时给出‘恢复 SQL 会话’显示，由这个显示，可以做下列之一：

用选项 1 恢复旧的会话（恢复已有的 SQL 会话）

用选项 2 删除旧的会话，启动新的会话（删除已有的 SQL 会话，启动新会话）

若你选删除旧的会话继续新会话，使用输入 STRSQL 时规定的参数。若选择恢复旧会话或输入一个先前保存的会话，会忽视 STRSQL 时规定的参数，而用旧会话的参数，有信息指出哪个参数值改过了。

17.1.9　用交互 SQL 访问远程数据库

用交互 SQL，可用 CONNECT 语句与远程关系数据库通讯。交互 SQL 使用 CONNECT 语句的类型 2 语义（分布式工作单元）。在启动 SQL 会话时，交互 SQL 会与本地 RDB 建立隐含连接。当完成 CONNECT 语句时，会显示已建立与数据库联系。如果启动一个新会话且未规定 COMMIT(*NONE)，或重存一个保存的会话且保存会话的落实级不是*NONE，则连接会以落实控制登记。隐含联系和可能的落实控制登记可能会影响下面的与远程数据库的连接。详细信息请看 24.6.1。在连接远程系统前建议做下面的事情：

当连到不支持分布式工作单元的应用服务器上时，用 REALEASE　ALL　COMMIT 结束以前的连接，包括对本地的隐含连接。

当连到非 AS/400　DB2 应用服务器上时，用 REALEASE　ALL　COMMIT 结束以前的连接，包括对本地隐含连接，把落实控制级至少改到*CHG。

连到 AS/400 非 DB2 应用服务器时，一些会话属性会改变到应用服务器支持的属性。下表显示改变的属性：

表 17－1　值表

会话属性	原值	新值
日期格式	*YMD *DMY *MDY *JUL	*ISO *EUR *USA *USA
时间格式	有一个：分隔符的*HMS 有其它分隔符的*HMS	*JIS *EUR
落实控制	*CHG *NONE *ALL	*CS 可重复读
命名转换	*SYS	*SQL
允许复制数据	*NO，　*YES	*OPTIMIZE
数据刷新	*ALWAYS	*FORWARD
小数点	*SYSVAL	*PERIOD
分类顺序	非*HEX 的任何值	*HEX

注：

1. 若连到运行 V2.3 之前版本的 AS/400 系统上，分类顺序值为*HEX。
2. 连到 DB2/2 或 DB2/6000 应用服务器，规定的日期、时间格式必须是相同的格式。

完成连接后，会有信息显示会话属性已修改。可用‘会话服务’显示来显示修改过的会话属性。在运行交互 SQL 时，不能对缺省活动组建立别的联系。

连到有交互 SQL 的远程系统时，仅句法的语句处理方式会用本地系统支持的的句法来检查语法错误而不是用远程系统的。同样，SQL 提示和列表支持也用本地系统支持的命名转换和语句句法，而在远程运行语句。由于两个系统 SQL 支持级别不同，在远程系统运行时可能会发现语句句法错误。

在连到本地数据库可使用集合、表的列表，只有连到支持 DESCRIBE　TABLE 语句的关系数据库管理才可用列的列表。

在结束有未决修改的连接或有用保护对话的连接时，联系仍保留，若不在连接上完成别的工作，在下个 COMMIT 或 ROLLBACK 操作时会结束联系。也可在退出交互 SQL 执行 RELEASE ALL 和 COMMIT 来结束联系。

使用交互 SQL 对 AS/400 非 DB2 的应用服务器的远程访问，来做某些启动工作。详细信息请看分布式数据库程序设计。

注：用通讯跟踪的输出，会有引用‘CREATE TABLE XXX’语句，它用来决定软件包的存在，它是正常处理的一部分，可忽略。

第十八章　使用 SQL 语句处理器

这章介绍 SQL 语句处理器。

在运行 RUNSQLSTM 命令可用 SQL 语句处理器。

SQL 语句处理器允许从源文件中执行 SQL 语句，允许不编译源文件，就可重复运行源成员中的语句或修改它们。它可使数据库环境的启动更容易。可用 SQL 语句处理器的语句如下：

- ALTER TABLE
- CALL
- COMMENT ON
- COMMIT
- CREATE COLLECTION
- CREATE INDEX
- CREATE PROCEDURE
- CREATE SCHEMA
- CREATE TABLE
- CREATE VIEW
- DELETE
- DROP
- GRANT (Package Privileges)
- GRANT (Procedure Privileges)
- GRANT (Table Privileges)
- INSERT
- LABEL ON
- LOCK TABLE
- RENAME
- REVOKE (Package Privileges)
- REVOKE (Procedure Privileges)
- REVOKE (Table Privileges)
- ROLLBACK
- SET TRANSACTION
- UPDATE

用源成员，语句以分号结束且不用 EXEC SQL 开头。若源成员记录长度大于 80，只读前 80 个字符，源成员的注释可以是行注释也可是块注释。行注释用（— —）开始，于行尾

结束。块注释以/*开始，可越过多行直至下一个*/ 出现，源文件中只可有 SQL 语句和注释，SQL 语句的输出清单、结果信息送到打印文件。缺省打印文件为 QSYSPRT。

18.1 出错后语句的执行

当语句返回的错误级别大于 RUNSQLSTM 命令中 ERRLVL 规定的值，语句失败，会检查源成员的其他语句的句法，但不会执行它们。大多数 SQL 错误级别为 30，若在语句失败后继续处理，把 RUNSQLSTM 命令的 ERRLVL 参数设为 30 或更高。

18.2 SQL 语句处理器的落实控制

在 RUNSQLSTM 命令中规定落实控制级。若规定了不是*NONE 的落实控制级，SQL 语句在落实控制下运行。若成功执行了全部的语句，在 SQL 语句处理器完成时会执行 COMMIT，否则做 ROLLBACK。若返回码级别小于或等于 RUNSQLSTM 的 ERRLVL 参数规定的值，则认为语句执行成功。

源成员中可用 SET TRANSACTION 语句覆盖 RUNSQLSTM 规定的落实控制级。

注：作业必须在工作单元边界使用落实控制的 SQL 语句处理器。

18.3 SQL 语句处理器的模式

SQL 语句处理器支持 CREATE SCHEMA 语句，这是一个复杂的语句，可看作有两个部分，第一部分规定模式的集合，第二部分包含定义集合中目标的 DDL 语句。可用下列两种方法之一写第一部分：

CREATE SCHEMA 集合名

用规定的集合名生成集合

CREATE SCHEMA AUTHORIZATION 授权名

以授权名作为集合名生成。运行模式时，用户必须有权限处理名为授权名的用户配置文件。

语句的授权名的权限应包括：

— 运行 CREATE COLLECTION 的权限

— 运行在 CREATE SCHEMA 语句中每个 SQL 语句的权限

CREATE SCHEMA 语句的第二部分包括从零到任何数的下列语句：

- CREATE TABLE
- CREATE VIEW
- CREATE INDEX
- GRANT (Table Privileges)
- COMMENT ON
- LABEL ON

这些语句直接跟在第一部分后，不用分号分开。如果其它的 SQL 语句跟在模式定义后，模式的最后一个语句必须以分号结束。

模式语句的第二部分建立或引用的目标都必须在模式建立的集合中，所有非限定的引用都隐含地被集合限定，所有限定的引用都必须被生成的集合限定。

18.4 SQL 语句处理器的源成员清单

```
5769ST1 V4R2M0 980228     Run SQL Statements   SCHEMA   01/01/98 15:35:18   Page   1
Source file..............CORPDATA/SRC
Member..................SCHEMA
Commit..................*NONE
Naming..................*SYS
Generation level.........10
Date format..............*JOB
Date separator...........*JOB
Time format..............*HMS
Time separator ..........*JOB
Default Collection.......*NONE
IBM SQL flagging.........*NOFLAG
ANS flagging.............*NONE
Decimal point............*JOB
Sort Sequence............*JOB
Language ID..............*JOB
Printer file.............*LIBL/QSYSPRT
Source file CCSID........65535
Job CCSID................0
Statement processing.....*RUN
Allow copy of data.......*OPTIMIZE
Allow blocking...........*READ
Source member changed on 01/01/98  11:54:10

 5769ST1 V4R2M0 980228    Run SQL Statements   SCHEMA   01/01/98 15:35:18   Page   2
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
    1
    2     DROP COLLECTION DEPT;
    3     DROP COLLECTION MANAGER;
    4
    5     CREATE SCHEMA DEPT
    6          CREATE TABLE EMP (EMPNAME CHAR(50), EMPNBR INT)
    7                               -- EMP will be created in collection DEPT
    8          CREATE INDEX EMPIND ON EMP(EMPNBR)
    9                               -- EMPIND will be created in DEPT
   10          GRANT SELECT ON EMP TO PUBLIC;  -- grant authority
   11
   12     INSERT INTO DEPT/EMP VALUES('JOHN SMITH', 1234);
   13                               /* table must be qualified since no
   14                                  longer in the schema */
   15
   16     CREATE SCHEMA AUTHORIZATION MANAGER
```

```
  17                              -- this schema will use MANAGER's
  18                              -- user profile
  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                  -- Two statements can be on the same line
* * * * *  E N D  O F  S O U R C E  * * * * *

5769ST1 V4R2M0 980228   Run SQL Statements   SCHEMA   01/01/98 15:35:18   Page      3
Record  *...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8   SEQNBR  Last change
MSG ID  SEV  RECORD  TEXT
SQL7953   0       1  Position 1 Drop of DEPT in QSYS complete.
SQL7953   0       3  Position 3 Drop of MANAGER in QSYS complete.
SQL7952   0       5  Position 3 Collection DEPT created.
SQL7950   0       6  Position 8 Table EMP created in collection DEPT.
SQL7954   0       8  Position 8 Index EMPIND created on table EMP in DEPT.
SQL7966   0      10  Position 8 GRANT of authority to EMP in DEPT completed.
SQL7956   0      10  Position 40 1 rows inserted in EMP in DEPT.
SQL7952   0      13  Position 28 Collection
                     MANAGER created.
SQL7950   0      19  Position 9 Table EMP_SALARY created in collection
                     MANAGER.
SQL7951   0      21  Position 9 View LEVEL created in collection MANAGER.
SQL7954   0      23  Position 9 Index SALARYIND created on table EMP_SALARY
                     in MANAGER.
SQL7966   0      25  Position 9 GRANT of authority to LEVEL in MANAGER
                     completed.
SQL7966   0      25  Position 37 GRANT of authority to EMP_SALARY in MANAGER
                     completed.

Message Summary
Total    Info    Warning    Error    Severe    Terminal
13     13         0         0         0          0
00 level severity errors found in source
* * * * *  E N D  O F  L I S T I N G  * * * * *
```

图 18-1　SQL 语句处理器的 OSYSPRT 清单

第十九章　AS/400　DB2 的数据保护

这章介绍授权用户保护 SQL 数据的安全措施及保证数据完整的方法。

19.1 安全

系统安全功能可管理 AS/400 中的所有目标，包括 SQL 语句。用户可用 SQL　GRANT，REVOKE 语句或 CL 命令 EDTOBJAUT、GRTOBJAUT 和 RVKOBJAUT 对 SQL 目标授权。请看安全参考一书。

SQL　GRANT，REVOKE 可在 SQL 软件包、SQL 过程、表、视图、及列上操作。SQL　GRANT，REVOKE 语句分配私人和公共权限，某些情况中要用 EDTOBJAUT，GRTOBJAUT，RVKOBJAUT 命令做授权处理。

SQL 语句权限检查取决于语句是静态的，动态的，还是交互的。

对于静态 SQL 语句：如果 USRPRF 值为*USER，本地运行的 SQL 语句的权限是用运行程序用户的配置文件检查的，远程运行 SQL 语句的权限是应用服务器的用户配置文件检查的，*USER 是系统命名的缺省值。

若 USRPRF 值为*OWNER，本地运行 SQL 语句的权限是用运行程序的用户配置文件和程序的所有者的配置文件检查的，远程运行 SQL 语句的权限是用应用服务器作业和 SQL 软件包所有者的用户配置文件检查的，使用其中较高的权限，*OWNER 是 SQL 的命名的缺省值。

对于动态的 SQL 语句：

如果 USRPRF 值为*USER，本地运行 SQL 语句的权限是用运行程序本人的用户配置文件检查的，远程运行 SQL 语句的权限是用应用服务器作业的用户配置文件检查的。

如果 USRPRF 值为*OWNER 且 DYNUSRPRF 为*USER，本地运行的 SQL 语句的权限是用运行程序的本人用户配置文件检查的，远程运行的 SQL 语句的权限是用应用服务器作业的用户配置文件检查的。

如果 USRPRF 值为*OWNER 且 DYNUSRPRF 为*OWNER，本地运行 SQL 语句的权限是用程序所有者和运行程序的用户配置文件检查的，远程运行 SQL 语句的权限是用 SQL 软件包所有者和应用服务器作业的用户配置文件检查的，使用其中较高的权限。考虑到安全性，应慎用 DYNUSRPRF 为*OWNER 的参数值，运行程序者可由此选项获得程序和软件包主人的访问权限。

对于交互式 SQLSTM，会检查是否有处理声明的权限，已采用的权限不用于交互式 SQL 声明。

19.1.1　授权 ID

授权 ID 标识 AS/400 系统中唯一用户，是一个用户配置文件目标，它用系统的 CRTUSRPPF 生成授权 ID。

19.1.2　视图

视图防止未授权用户访问敏感数据。应用程序可访问表中需要的数据而不能访问授限数据。用视图在 SELECT 列表中不规定某些列，就可以限制访问这些列，（例如，职员工资）。规定 WHERE 子句，可用视图限制访问表中某些行，（例如，只允许访问与特定部门号码有联系的行。）

19.1.3　审查

AS/400 DB2 设计符合 U. S.政府 C2 安全标准，此标准的关键特点是可审查系统上的活动。AS/400 DB2 采用系统安全功能管理的审查实用程序，在目标、用户、系统级上都可执行审查。系统值 QAUDCTL 控制审查是在目标级还在用户级上，CHGUSRAUD 和 CHGOBJAUD 规定审查哪些用户和目标，系统值 QAUDLVL 控制审查活动的类型（例如：授权失败，生成，删除，分配等）详细信息请看安全参考一书。

AS/400 DB2 也可用 AS/400 DB2 日志支持审查行修改。

19.2 数据完整性

数据完整性保护数据不被未授权的用户毁坏或改变，防止系统操作、硬件失败（例如磁盘的物理破坏）程序错误，作业完成之前中断，或干预同时运行的应用程序。数据完整性由以下功能保证：

并发性
日志
落实控制
原子操作
约束
保存 / 重存
破坏容差
索引恢复

19.2.1 并发性

并发性指多个用户同时访问或修改同一表或视图数据，而又没有失去数据完整性的风险。AS/400 DB2 数据管理自动提供此功能，隐含的要用锁，使并发用户保护表和行使其不能同时修改同一数据。

AS/400 DB2 会要求行加锁确保完整性，然而有些情况会要求 AS/400 DB2 请求更排它的表级锁而非行级锁。详细情况请看 19.2.3。

有时，程序要求防止同一程序中其它语句运行的锁。例如，游标保持的当前行的更新锁，（排它），也能由同一程序别的游标得到（或用没分配游标的 DELETE 或 UPDATE），这将妨碍引用。第一个游标定位的 UPDATE 或 DELETE 语句，直到执行另一个 FETCH，一个游标定位的当前行上的读锁不会防碍同一程序中另一个游标得到同一行的锁。

支持缺省和用户规定的锁等待超时值，DB2 用缺省记录等待时间 60 秒和缺省文件等待时间（*IMMED）生成表，视图和索引。这个锁等待时间用在 DML 语句中。用 CL 命令 CHGPF，CHGLF，OVRDEF 可改变这些值。

用于所有 DDL 语句和 LOCK TABLE 锁等待时间是作业缺省等待时间（DFTWAIT），可用 CL 命令 CHGJOB，CHGCLS 更改此值。

在规定了大的记录等待时间的环境中，会提供死锁检查。例如，假定作业对行 1 有一排它锁，另一作业对行 2 有排它锁，若第一个作业要锁住行 2，但由于第二个作业占此锁，它必须等待。若第二个作业要锁住行 1，AS/400 DB2 会检测两个作业死锁，错误会返给第二个作业。

可用 SQL LOCK TABLE 语句明显的防止同时要用此表的其他用户，用 COMMIT(*RR)也可防止别的用户在同一工作单元用同一表。

为改善性能，AS/400 DB2 会频繁打开 ODP，此性能也在 ODP 引用的表上留下一个锁，但不在行中留锁。留在表中的锁会防止别的作业对此表操作。大多数情况下，AS/400 DB2 会检测出有锁的作业，并对其它作业标识此事件，此事件会导致 AS/400 DB2 关闭与此表有

关的任意 ODP，且仅由性能原因打开。注：锁等待时间必须大到能标识事件及其它作业关闭 ODPs，否则会出错。

除了用 LOCK TABLE 语句取得表锁，也用 COMMIT(*ALL)或 COMMIT(*RR)，由一个作业读的数据可立即被另一作业修改。通常读数据同时执行 SQL 语句，这样是并发的，（例如在 FETCH 期间）。但在下列情况下，在 SQL 语句执行前读数据，因此数据不是并发的，（例如，在 OPEN 期间）：

规定了 ALWCPYDTA(*OPTIMIZE)且优化断定作拷贝比不作会执行的更好。

有些查询要求数据库管理建立临进结果表，临时结果表的数据不会影响打开游标后发生的修改。在下列情况下需要临时结果表：

—ORDER BY 规定的列的存储字节总长度超 2000 字节

—ORDER BY 和 GROUP BY 子句规定不同的列或不同顺序的列

—规定 UNION 或 DISTINCT

—ORDER BY 或 GROUP BY 规定的列不全来自同一个表

—由 JOINDFT 键字定义连接一个逻辑文件和另一文件

—连接或规定 GROUP BY 依赖多个数据库成员的逻辑文件

—查询包括至少一个文件是有 GROUP BY 子句的视图的连接

—查询包括有 GROUP BY 子句，它引用一个有 GROUP BY 的子句的视图。

在打开查询时，会计算基本子查询的值。

19.2.2 日志

AS/400 DB2 日志提供审核、跟踪和向前向后恢复。向前恢复用老的表加上记录在日志中的修改，向后恢复从表中取消记录在日志中的修改。

在生成集合时， 在集合中生成日志和日志接收器。如果在 CREATE COLLECTION 或 CREATE SCHEMA 中规定了 ASP 子句，则日志和日志接收器仅生成在用户 ASP 中，但由于把日志接收器放在它们自己的 ASP 能改变性能，则把后来生成的日志接收器放在不同的 ASP 中。

在集合中生成表时，会自动记到 AS/400 DB2 生成在集合的日志中（QSQJRN）。若库中有名为 QSQJRN 的日志，在非集合中创立的表也有日志启动。此后，就可用日志功能管理日志、日志接收器，被日志的表。把表移入集合中，不会有发生自动修改日志状态。若重存一个表，使用正常的日志规则，就是说，如果在保存时日志表，重存时会日志到同一日志中，如果在保存时没日志，则在重存时也不做日志。

在 SQL 集合中生成的日志正常地记录所有对 SQL 表的修改，也可用系统日志功能将 SQL 表记到不同的日志中。若一个集合中的表是另一集合中表的父表，这么做是必要的。这是因为 AS/400 DB2 要求在引用约束中的父代和子代文件在对父表进行更新和删除操作时要记在同一日志中。

用户可用日志功能停止对表做日志，但那样做会妨碍应用程序在落实控制下运行。若对有引用约束的父表规定 NO ACTION，CASCADE，SET NULL，SET DEFAULT 删除原则停止日志，会阻止所有更改、删除操作。若规定 COMMIT(*NONE)，应用程序仍能发挥功能，但这样做就不能对日志和落实控制提供相同级别的完整性。

19.2.3 落实控制

AS/400 DB2 落实控制提供一个方法处理一组数据库的修改（更新、插入、DDL 操作或删除）作为一个工作单元交易处理。落实操作保证这一组操作完成，返回操作保证这组操作返

回去。可在不同接口执行落实操作，它们是：

SQL COMMIT 语句

CL COMMIT 命令

语言落实语句（例如 RPG 的 COMIT 语句）

返回操作也可在不同接口进行：

SQL ROLLBACK 语句

CL ROLLBACK 命令

语言返回语句（例如 RPG 的 ROLBK 语句）

不能落实或返回的 SQL 语句是：

DROP COLLECTION

GRANT 或 REVOKE（当目标有权限保护时）

如果 SQL 语句用不是 COMMIT(*NONE)的级别执行，或执行 RELEASE 语句时没有启动落实控制,那么 AS/400 DB2 就会隐含调用 CL 命令 STRCMTCTL 设置落实控制环境。它在 STRCMTCTL 中规定 LCKLVL NFYOBJ(*NONE)和 CMTSCOPE(*ACTGRP)参数，LCKLVL 是 CRTSQLxxx，STRSQL,RUNSQLSTM 命令中 COMMIT 参数的锁级别。在 REXX 中,规定的 LCKLVL 是 SET OPTION 语句的锁级别。（1）可用 STRCMTCTL 规定不同的 CMTSCOPE，NFYOBJ，LCKLVL。若规定 CMTSCOPE(*JOB)启动作业级落实定义，AS/400 DB2 在此活动组用作业级落实定义。

注：使用落实控制时，用数据操作语句的应用程序引用的表必须做日志。

对于使用列功能，GROUP BY，HAVING 且在落实控制下的游标，ROLLBACK HOLD 不影响游标的位置，另外，落实控制会发生以下情况：

若这些游标规定了 COMMIT(*CHG)和(ALWBLK(*NO)或 ALWBLK(*READ))之一，会有信息显示（CPI430B）说明不允许 COMMIT(*CHG)。

若这些游标规定了 COMMIT(*ALL)，COMMIT(*RR)或 COMMIT(*CS)且有 KEEP LOCKS 子句，AS/400 DB2 会在共享模式下（*SHRNUP）锁住所有引用的表，锁可防止并发应用处理对表执行除只读以外的其它操作，会有信息（SQL7902 或 CPI430A）显示，指出 COMMIT(*ALL)，COMMIT(*RR)或 COMMIT(*CS)与 KEEP LOCKS 子句对游标一起规定了，这是不允许的，也可能发送信息 SQL0595。

对于规定了 COMMIT(*ALL)，COMMIT(*RR)或 COMMIT（*CS）及 KEEP LOCKS 的游标，或需用目录文件或要求临时结果表的游标，AS/400 DB2 会用共享模式锁住所有引用的表。这将阻止并发处理执行除只读外的任何表操作。信息（SQL7902 或 CPI430A）显示 COMMIT（*AL）要求但未允许，也可发送信息 SQL0595。

若规定了 ALWBLK(*ALLREAD)和 COMMIT(*CHG)，在预编译程序时，所有只读游标允许处理成块的行，ROLLBACK HOLD 不会往回卷游标位置。

若要求 COMMIT(*RR)，会锁住表直至查询结束。若游标是只读，将锁住表（*SHRNUP），若游标为更新模式，将锁住表(*EXCLRD)，由于别的用户都锁在表外，运行重复读就可防止对表格的并发访问。

若规定了非 COMMIT(*NONE)的隔离级，应用程序执行 ROLLBACK 或活动组正常结束（且落实定义不是*JOB），所有在此工作单元的更新，插入，删除和 DDL 操作都被返回。若应用程序执行 COMMIT 或者活动组正常结束，则所有这些操作都落实。

在一工作单元完成之前，AS/400 DB2 用行锁阻止别的作业访问来修改数据，若规定了 COMMIT(*ALL)，取用的行读锁也可阻止其它作业修改在工作完成前读数据，但不阻止其它作业读未修改的记录。这就保证了，若同工作单元重读一个记录，它会得到同一结果。读锁不阻止其它作业取回相同行。

落实控制可处理同一工作单元中四百万行的修改。若规定 COMMIT(*ALL)或

COMMIT(*RR)，此限制包含所有读的行，（如果在一个工作单元中，一行被修改或读多次，它仅做一次计算），用大量锁会影响系统性能，且不允许并发用户访问锁住的行，直到工作单元结束。最好保持工作单元中处理的行数少一些。

落实控制允许在一个工作单元中每个日志在落实控制下打开 512 个文件，或用未修改关闭。

COMMIT HOLD 和 ROLLBACK HOLD 允许保持游标打开，在启动另一工作单元时不用 OPEN，若连到不是 AS/400 的远程数据库时不能用 HOLD 值，当连到不是 AS/400 的远程数据库时也支持此类游标。这种游标在 ROLLBACK 被关闭。

表 19－1　记录锁持续时间

SQL 语句	COMMIT 参数(见注 6)	记录锁期间	类型
SELECT INTO	*NONE	无锁	
	*CHG	无锁	
	*CS （见注 8）	当读和释放时有行锁	READ
	*ALL （见注 2）	读直到 ROLLBACK 或 COMMIT	READ
FETCH （只读游标）	*NONE	无锁	
	*CHG	无锁	
	*CS （见注 8）	读直到下一个 FETCH	READ
	*ALL （见注 2）	读直到 ROLLBACK 或 COMMIT	READ
FETCH （更新或删除属性的游标）（见注 1）	*NONE	记录不更新或删除读直到下一个 FETCH 记录更新或删除读直到 UPDATE 或 DELETE	UPDATE
	**CHG	记录不更新或删除读直到下一个 FETCH 记录更新或删除读直到 COMMIT 或 ROLLBACK	UPDATE
	*CS	记录不更新或删除读直到下一个 FETCH 记录更新或删除读直到 COMMIT 或 ROLLBACK	UPDATE
	*ALL	读直到 ROLLBACK or COMMIT	UPDATE(3)
INSERT （目的表）	*NONE	无锁	
	*CHG	插入直到 ROLLBACK 或 COMMIT	UPDATE
	*CS	插入直到 ROLLBACK 或 COMMIT	UPDATE
	*ALL	插入直到 ROLLBACK 或 COMMIT	UPDATE(4)
INSERT （子选择中的表）	*NONE	无锁	
	*CHG	无锁	
	*CS	在读时每个记录加锁	READ
	*ALL	读直到 ROLLBACK 或 COMMIT	READ
UPDATE （无游标）	*NONE	在更新时每个记录加锁	UPDATE
	*CHG	读直到 ROLLBACK 或 COMMIT	UPDATE
	*CS	读直到 ROLLBACK 或 COMMIT	UPDATE
	*ALL	读直到 ROLLBACK 或 COMMIT	UPDATE
DELETE （无游标）	*NONE	在删除时每个记录加锁	UPDATE
	*CHG	读直到 ROLLBACK 或 COMMIT	UPDATE
	*CS	读直到 ROLLBACK 或 COMMIT	UPDATE
	*ALL	读直到 ROLLBACK 或 COMMIT	UPDATE
UPDATE （有游标）	*NONE	当记录更新时释放锁	UPDATE
	*CHG	读直到 ROLLBACK 或 COMMIT	UPDATE
	*CS	读直到 ROLLBACK 或 COMMIT	UPDATE
	*ALL	读直到 ROLLBACK 或 COMMIT	UPDATE
DELETE （有游标）	*NONE	当记录删除时释放锁	UPDATE
	*CHG	读直到 ROLLBACK 或 COMMIT	UPDATE
	*CS	读直到 ROLLBACK 或 COMMIT	UPDATE
	*ALL	读直到 ROLLBACK 或 COMMIT	UPDATE
子查询(更新或删除属性的游标或 UPDATE 或无游标的 DELETE)	*NONE	读直到下一个 FETCH	READ
	*CHG	读直到下一个 FETCH	READ
	*CS	读直到下一个 FETCH	READ
	*ALL （见注 2）	读直到 ROLLBACK 或 COMMIT	READ
子查询（只读游标或 SELECT INTO）	*NONE	无锁	
	*CHG	无锁	
	*CS	当读时每个记录加锁	READ
	*ALL	读直到 ROLLBACK 或 COMMIT	READ

注：

1.若结果表不是只读，或下列之一为真，则游标用 UPDATE 或 DELETE 打开：
游标以 FOR UPDATE 定义
游标不以 FOR UPDATE，FOR READ ONLY 或 ORDER BY 子句定义且程序至少包含下列之一：
—游标 UPDATE 引用相同游标名
—游标 DELETE 引用相同游标名
—在 CRTSQLxxx 中规定 EXECUTE 或 EXECUTE IMMEDIATE 语句和 ALWBLK(*READ)或 ALWBLK(*NONE)。
2.为满足 COMMIT(*ALL)可排它的锁住一个表或视图，如果子选择包括 UNION，或查询过程要用临时结果，排它锁会看不到没落实的修改。
3.若行没有更新或删除，锁降到*READ。
4.目标表中行用 UPDATE 锁，子选择表中行用 READ 锁。
5.为满足重复读要使表或视图用排它锁，重复读下仍可锁住行，所要求的锁和它们的持续时间标识为*ALL。
6.重复读记录锁（*RR）与*ALL 的锁一样。
7.关于隔离级和锁的详细解释，见 AS/400 DB2 SQL 参考的第一章。
8.若以*CS 规定 KEEP LOCK 子句，任何读锁都会保持到关闭游标，或保持到完成 COMMIT 或 ROLLBACK。若每个子句不用游标，锁保持到 SQL 语句完成。

（1）规定的 LCKLVL 只是缺省锁级别，启动落实控制后，SET TRANSACTION SQL 语句和在 CRTSQLxxx，STRSQL 或 RUNSQLSTM 命令中 COMMIT 规定的锁级别会代替缺省锁级别。

19.2.4 原子操作

在 COMMIT(*CHG)，COMMIT(*CS)或 COMMIT(*ALL)下运行，要保证所有操作是原子操作。就是说，不非得启动，它们将完成或将出现，当然不考虑功能什么时候或怎样结束及中断。

如果规定了 COMMIT(*NONE)，某些数据库数据定义功能不是原子的，以下 SQL 数据定义语句可保证是原子的：

ALTER TABLE (见注 1)
COMMENT ON (见注 2)
CREATE PROCEDURE
LABEL ON (见注 2)
GRANT (见注 3)
REVOKE (见 3)
DROP TABLE (见注 4)
DROP VIEW (见注 4)
DROP INDEX
DROP PACKAGE
DROP PROCEDURE
RENAME

注：

1. 若需要加上或删去约束，以及更改列定义，那么操作是一次处理一个，所以整个 SQL 语句不是原子的，操作顺序是：
删除约束

放弃规定 RESTRICT 选项的列
所有其它列定义修改（DROP COLUMN CASCADE，ALTER COLUMN，ADD COLUMN）

2. 若为增加约束语句 COMMENT ON 和 LABLE ON 规定多列，每次处理一列，所以整个 SQL 语句不是原子的，但每个列或目标的 COMMENT ON 或 LABLE ON 是原子的。
3. 若为 GRANT，REVOKE 语句规定多个表、SQL 软件包或用户，每次处理一个表，所以整个 SQL 语句不是原子的，但每个表的 GRANT，REVOKE 是原子的。
4. 如在 DROP TABLE 或 DROP VIEW 期间需要放弃从属视图，每次处理一个从属视图，所以整个 SQL 语句不是原子的。

以下数据定义声明不是原子型的，因为它们包含了不止一个 AS/400/DB2 数据库操作。

```
CREATE COLLECTION
CREATE TABLE
CREATE VIEW
CREATE INDEX
CREATE SCHEMA
DROP COLLECTION
DROP SCHEMA
```

例如，生成 AS/400 DB2 物理文件后，在增加成员前，可中断 CREATE TABLE，在处理生成语句时，如操作非正常结束，要放弃目标再重新生成。在处理 DROP COLLECTION 语句时，可能必须再放弃集合或用 CL 命令 DLTLIB 来删除集合中的剩余部分。

19.2.5 约束

AS/400 DB2 支持唯一、引用和检查约束。唯一约束保证键字值的唯一性。引用约束是从属表所有外来键字的非空值在父表中有相应的父代键字。检查约束是限制在一列和一组列中的允许值。

AS/400 DB2 在任何 DML 语句执行期间强制检查约束的有效性，某些操作（例如重存从属表）会引起检查约束成为未知的。这时，可避免使用 DML 语句直到 AS/400 DB2 能检查约束的有效性为止。

用索引完成唯一约束，如果实行唯一约束的索引无效，可用 EDTRBDAP 命令显示任何当前需要重建的索引。

若 AS/400 DB2 当前不知道引用约束或检查约束是否有效，会认为约束处于检查未决状态，可用 EDTCPCST 命令显示任何当前要重建的索引。

19.2.6 保存/重存

AS/400 保存功能用来把表、视图、索引、日志、日志接收器、SQL 软件包、SQL 产品和集合保存到硬盘上或一些外部介质上。保存的版本可在后来重存到 AS/400 系统上，保存/重存功能可保存整个集合，某些目标或在给定日期和时间前修改过的一些目标，所有恢复到以前状态用的信息都保存，此功能用以前版本的表或整个集合的数据来恢复损坏的表。

当重存为 SQL 过程生成的程序时，它会自动加到 SYSPROCS 和 SYSPARMS 目录中，已不存在的过程用同名来重存，生成在 QSYS 中的 SQL 程序在重存时不会做为 SQL 过程生成。

分布的 SQL 程序或其相关软件包保可保存或重存到多个 AS/400 系统中，这允许不同系

统的多个 SQL 程序拷贝访问同一应用服务器上相同 SQL 软件包，也允许单个的分布式 SQL 程序连到任何数量的有重存 SQL 软件包的应用服务器上,SQL 软件包不能重存到不同的库中。

注：将一个集合重存到已有的库中，或重存到不同名的集合中，这一过程不能重存日志、日志接收器或 IDDU 字典。若用不同名重存集合，集合中的目录视图仅影响旧集合的目标，在 QSYS2 中的目录视图将适当地反映所有目标。

19.2.7 破坏容差

AS/400 系统提供几个方法来减少或消除硬盘错误引起的损失。例：镜象保护，检查和及 RAID 磁盘可减少硬盘出现问题可能性，AS/400 DB2 也有一定数量的容差给硬盘或系统错误引起的损失。

DROP 操作总能成功而不管有无损坏，这就保证如果有损坏，至少表、视图、SQL 软件包或索引能删除、重存或再生成。

当硬盘错误损害表中行的一小部分时，AS/400 DB2 数据库管理允许尽可能的读出行。

19.2.8 索引恢复

AS/400 DB2 提供处理索引恢复的几种功能：

系统管理的索引保护：

EDTRCYAP 命令允许用户在系统或电源出故障时，通知 AS/400 DB2 系统保证恢复系统所有索引需要的时间限于规定时间之内，系统在系统日志中自动记下足够的信息来把恢复时间限制在一定范围内。

索引日志：

AS/400 DB2 有一索引日志功能，使其不必在电源和系统故障后全部重建整个索引。若已对索引日志，数据库自动保证索引与表中数据同步而不必从头重建。SQL 索引不是自动日志的，要用 CL 命令 STRJRNAP 来对 AS/400 DB2 生成的索引做日志。

索引重建：

系统中所有的索引都有一维护选项，它规定何时维护索引。SQL 索引是用* IMMED 维护属性生成的。

在电源故障或系统失败时，如索引没有上述技术的保护，修改处理中的索引可能要数据库管理重建以确保与实际数据一致。系统中所有索引都有一恢复选项，它规定若有必要何时重建索引，具有 UNIQUE 属性的所有 SQL 索引都是*IPL 恢复属性（即这些索引在 OS/400 启动前要重建），其它 SQL 索引都是*AFTIPL 恢复选项（即在 OS 启动之后，索引异步重建），在 IPL 期间，操作员可看一个显示，它给出需要重建的索引和它们的恢复选项，操作员可替换这些恢复选项。

索引的保存和重存：

用 SAVOBJ 或 SAVLIB CL 命令的 ACCPTH(*YES)保存表时，可用保存 / 重存功能保存索引，在重存已保存的索引时，不需要重建索引，先前没有保存和重存的索引会由数据库管理自动和异步重建。

19.2.9 目录完整性

目录中有表、视图、SQL 软件包、索引、过程和集合中参数的信息，数据库管理确保所有时候目录中的信息都是正确的，在对表、视图、SQL 软件包、索引及产品及目录参数修改时，要避免最终用户明显的修改目录信息及隐含地维护目录信息。

目录完整性的维护不管集合中的目标是由 SQL 语句，OS/400 CL 命令，系统/38 环境 CL 命令，系统 / 36 环境功能还是任何其它 AS/400 系统的产品来做修改，都是一样。例：运

行 SQL　DROP 语句,执行 OS/400　DLTF 命令,用系统/38　DLTF　CL 命令或在 WRKF 或 WRKODJ 显示中用选项 4 都可用来删除表，而不考虑删除表的界面，执行删除时，数据库管理都会从目录中取消表的描述。下面是一功能列表，它对目录有影响。

表 19－2　目录中各种功能的作用

功能	目录中的作用
往表中加约束	信息加到目录中
删除表约束	在目录中删除相关信息
在集合中生成目标	信息加到目录中
删除集合中的目标	在目录中删除相关信息
往集合中重存目标	信息加到目录中
修改目标的长注释	在目录中重新注释
修改目标标号(说明)	目录中标号更新
修改目标所有者	目录中所有者更新
从集合中移出目标	取消目录中相关信息
目标移入集合中	信息加到目录中
目标改名	目录中目标名字更改

19.2.10　用户辅助存储池（ASP）

在 CREATE　COLLECTION 和 CREATE　CHEMA 语句中用 ASP 子句可以在用户 AS 中生成 SQL 集合，也可用 CRTLIB 在用户 ASP 中生成库，然后可用这个库接收 SQL 表，视图，索引。详细内容请看备份和恢复。

第二十章　测试

这章介绍如何为应用程序中的 SQL 语句建立一个测试环境，及怎样调试程序。

20.1　建立测试环境

测试应用程序必须做以下事情：

授权。你需要有建立表和视图，访问 SQL 数据及运行程序的权限。

测试数据结构。如果要在表和视图中更改、插入或者删除数据，应该测试数据去鉴定程序的运行。如果程序仅仅从表和视图中获得数据，那么在测试程序时应考虑用产品级数据，建议使用 CL 命令 STRDBG 及 UPDPROD(*NO)可以保证产品级数据不会改变。详细内容请看 CL 程序设计手册。

测试输入数据。测试程序用的输入数据的有效性及所想要的输入可能的条件，如果输入数据无效就不能够保证输出数据有效。

如果程序验证输入数据有效，包括验证有效数据和无效数据，那么处理有效数据，将无效数据查出来。

可在后续测试中刷新数据。

为了彻底地测试程序，应尽可能的使用更多的途径来测试程序。例如：

用输入数据迫使程序运行到其每个分支。

检查结果，例如程序要更新一行，要选择这一行来看更新是否正确。

一定要测试程序的错误例程，再用输入数据使程序尽可能多的接触到可能存在的错误。

测试程序用的编辑和有效的例程。向程序提供尽可能多的不同组合的输入数据以便尽可能地验证，是否正确编辑数据的有效性。

20.1.1　设计测试数据结构

要测试访问 SQL 数据的应用程序，可以生成测试的表和视图：

测试现存表的视图。如果应用程序不是修改数据且数据在一个或多个产品级表中存在，那么应该考虑使用现存表的视图。建议你使用 SRTDBG 命令及 UPDPROD(*NO)参数来保证产品级数据不会改变。

测试表。当应用程序生成、修改或者删除数据时，可用包含测试数据的表来测试应用程序。

可用 CL 命令 CRTDUPOBJ 来生成一个测试表或索引的复本。

20.1.1.1　权限

在建立一个表之前，你必须有建立表和使用表所在集合的权限，另外也必须有权建立和执行你想要测试的程序。

如果想使用现存的表和视图，必须有权访问这些表及视图。

如果想建一个视图，必须有权建立视图和有权使用该视图所依赖的每个表，详细信息请看 AS/400　DB2　SQL 参考。

20.2　测试 SQL 应用程序

测试 SQL 应用程序有两段：程序调试段和性能验证段。

20.2.1 程序调试段

这段保证 SQL 查询是正确的，并且程序产生正确的结果。

调试有 SQL 语句的程序和没有 SQL 语句的程序，大部分是一样的。但是，当 SQL 语句在调试方式下运行时，数据库管理器把运行 SQL 语句的信息放到作业日志中，这个信息指出 SQL 语句的 SQLCODE。如果语句运行成功，那么 SQLCODE 的值为零，并发送完成信息，负的 SQLCODE 导致诊断信息，一个正的 SQLCODE 表示消息信息。

信息是用 SQL 前缀的四位代码或者是用 SQ 前缀的 5 位代码，例如，-204 的 SQLCDOE 的信息为 SQL0204，30000 的 SQLCODE 信息为 SQ30000。

和 SQLCODE 相关的是 SQLSTATE，SQLSTATE 是在 SQLCA 提供的返回码，它指出不同于 IBM 关系数据库产品的一般的错误条件。不同的数据库产品的相同的错误条件可以产生相同的 SQLSTATE，相同的错误条件却不能产生相同的 SQLCODE。在确定从 AS/400 系统非 DB2 关系数据库操作中返回的错误时，这个返回码特别有用。

对于非 ILE 的调试程序在调方式引用的高级语言语句号必须从编译清单中获得，而对于 ILE 调试程序，应规定 DBGVIEW(*SOURCE)做预编译，然后用源码级调试。

SQL 总是把负的 SQLCODE 和非+100 的正返回码信息放到作业日志中，而不考虑它是否在调试方式下。

20.2.2 性能验证段

这段测试鉴定索引的可用性及用数据库管理允许的查询方式，在预期的响应时间能否解决查询。一个 SQL 应用的性能依赖于访问表的属性，如果使用小表，响应查询的时间不会受索引的影响，但当你用大表对数据库运行相同查询，而没有相应的索引，查询的响应时间就会很长。

测试环境应该尽量地与产品环境相似。测试集合中的表应用产品集合中有相同名称和结构。在这两个集合中，表上应有相同的索引，表中的行数和值都应该类似。

20.2.3 SQL 性能验证使用的 CL 命令

SQL 性能验证可用下列命令：

CHGQRYA：

修改查询属性命令可以防止用户过长的查询，可用来设置 QRYTIMLMT 参数来避免启动一个查询后时间大于指定时间的查询，它停止建立临时索引和其他资源集中的查询操作。

由查询时间限制在作业日志中生成的性能信息，即使未在调试状态，也会取消查询。详细内容请看本书第二十一章。

DSPJOB：

有 OPTION(*OPNF)参数的 DSPJOB 命令，用来显示在一个应用程序运行时使用的索引和表。

有 OPTION(*JOBCLK)的 DSPJOB 命令，用来分析目标和行锁采用，它显示目标和锁定的行和占有此锁作业的名称。

有 OPTION(*CMTCTL)DSPJOB 命令，显示程序运行的隔离级，在一个交易中锁定的行数和待处理的 DDL 功能。显示的隔离级是缺省的隔离级，实际的隔离级，任何 SQL 程序都可用，是用 CRTSQLxxx 命令中的 COMMIT 参数规定的。

PRTSQLINF

此命令允许打印一个嵌入程序中的有关 SQL 语句的信息、SQL 软件包或服务程序中的 SQL 语句信息。信息包括：SQL 语句，运行语句所使用的

访问计划和用在预编译命令中的参数清单。详细内容请看 AS/400　DB2　CL 命令一书的附录 D。

TRCJOB

用来获得运行 SQL 语句程序的跟踪信息。跟踪作业输出列出为每个 SQL 语句打开处理初始数据库时所用的索引和文件，及运行每个 SQL 语句的处理单元和使用的页资源。

20.2.4　性能信息

可以分析由数据库管理提供的放在工作日志中的 SQL 语句的功能和结构。这个信息是在调试方式下由 SQL 程序或者是交互 SQL 发出的，适当时数据库管理可以发送任何下列信息，变量（&1,&x）是替代变量，包含目标名或者其它替换值。这些信息是：

- CPI4321 Access path built for file &1.
- CPI4322 Access path built from keyed file &1.
- CPI4323 The OS/400 query access plan has been rebuilt.
- CPI4324 Temporary file built for file &1.
- CPI4325 Temporary result file built for query.
- CPI4326 File &1 processed in join position &X.
- CPI4327 File &1 processed in join position 1.
- CPI4328 Access path &4 was used by query.
- CPI4329 Arrival sequence access was used for file &1.
- CPI432A Query optimizer timed out.
- CPI432B Subselects processed as join query.
- CPI432C All access paths were considered for file &1.
- CPI432D Additional access path reason codes were used.
- CPI432E Selection fields mapped to different attributes.
- CPI432F Access path suggestion for file &1.
- CPI4330 &6 tasks used for parallel &10 scan of file &1.
- CPI4331 &6 tasks used for parallel index created over file &1.
- CPI4332 &1 host variables used in query.
- CPI4333 Hashing algorithm used to process join.
- CPI4334 Query implemented as reusable ODP.
- CPI4335 Optimizer debug messages for hash join step &1 follow:
- CPI4336 Group processing generated.
- CPI4337 Temporary hash table built for hash join step &1.
- CPI4338 &1 Access path(s) used for bitmap processing of file &2.
- CPI4341 Performing distributed query.
- CPI4342 Performing distributed join for query.
- CPI4345 Temporary distributed result file &4 built for query.
- SQL7910 SQL cursors closed.
- SQL7911 ODP reused.
- SQL7912 ODP created.

- SQL7913 ODP deleted.
- SQL7914 ODP not deleted.
- SQL7915 Access plan for SQL statement has been built.
- SQL7916 Blocking used for query.
- SQL7917 Access plan not updated.
- SQL7918 Reusable ODP deleted.
- SQL7919 Data conversion required on FETCH or embedded SELECT.
- SQL7939 Data conversion required on INSERT or UPDATE.

这些信息提供了查询如何运行的反馈，也指出可以令查询速度加快的一些方法。

这些信息包括分析产生信息原因的内容，引用的目标名和可能的用户回答。

信息发送的时间不是必需的，它指出完成相应功能的时间，一起发送的还有查询运行的开始时间。

对下列信息产生的原因和用户回答是简短的解释，实际的信息帮助是很完整的，可用来确定其含义及响应每个信息。

下面是对于每个信息可以采取的用户动作：

CPI4321－－为文件&1 建立访问路径。

这条信息指明建立了一个临时访问路径来处理查询，新的访问路径由读取规定文件的所有记录来生成。

在每一个查询运行建立的访问路径所需要的时间是非常明显的。考虑建立一个文件（CRTLF）或一个 SQL 索引（CREATE INDEX SQL 语句）：

覆盖信息帮助中命名的文件。

用在帮助信息中命名的键字段。

用在信息帮助中的规定的升序或降序。

用在信息帮助中的规定的分类排序表。

考虑生成一个有选择或省略语句的逻辑文件，它匹配或部分匹配包括常量的谓词。数据库管理器考虑使用选择或省略的逻辑文件，尽管它们在查询中未明确规定。

对于某些查询，优化可以考虑建立一个访问路径，当一个查询把顺序字段作为一个访问路径的键字字段时，且在规定的记录选择使用不同的字段时，有可能发生此情况。如果选择的记录有大概 20%记录或更多返回，那么优化可以建立一个新的访问路径以便快速的访问数据，新的访问路径将需要读的数据数量降至最低程度。

CPI4322－－在键字文件&1 上建立访问路径

这条信息指出建立一个从键字文件的访问路径中建立的一个临时访问路径。

一般来说，这种操作不会使用大量的时间或资源，因为仅要读文件中一段数据。有时，通过建立一个逻辑来改善性能或用 SQL 索引来满足信息帮助中访问路径的需要。

详细内容看前一个信息 CPI4321。

CPI4323－－AS/400 查询访问计划已经重建

这条信息可由不同的原因发送，在信息帮助中提供这个原因。

大多数情况是当被查询的文件环境改变，使当前的访问路径作废时会发出此信息。文件环境变更的例子是当查询需要的访问路径不在系统时。

一个访问路径包括查询如何运行及列出运行查询用的访问路径，如果必须的访问路径不再可用，那么查询要再一次优化，建立一个新的访问路径，替代旧的那一个。

这种在运行时再次优化查询及建立新的访问计划是 AS/400 DB2 的功能，它允许查询使用数据库的最新数据而不用用户干预来尽可能有效的运行。

这种信息不常出现，不用采取什么动作。例如，当一个 SQL 软件包在恢复之后第一次使用或优化检查出发生了变化时，（例如建立一个新的索引），就会发出这条信息，它批准隐含的重建。但是，应该避免过度的重建，因为这会产生额外的查询处理，过度的重建可能指出一个应用设计问题或者实行一个无效的数据库管理。

CPI4324－－为文件&1 建立临时文件

在查询处理开始之前，指定文件中的数据要拷贝到一个临时的物理文件中以简化查询的运行，这个信息帮助包含了发出信息的原因。

如果规定的文件选择的行数很少，通常少于 1000 行，那么查询实现中的行选择部分就不花费太多的资源和时间，但如果查询花费比允许多的资源和时间，就要考虑修改查询，这样就不需要临时文件。

这样做的一个办法是把查询分成几步，考虑在子选择中用一个 INSERT 语句，仅把需要的记录选进物理文件中，然后再对剩余的查询使用文件记录。

CPI4325－－为查询建立临时结果文件

建立临时结果文件来放查询的中间结果，信息帮助中有需要临时结果文件的原因。

在某些情况下，建立一个临时结果文件提供了更快的运行查询方法，对于其它有许多记录要拷贝到临时结果文件中去的会花费大量的时间，但如果一个查询花费的时间和资源多于允许程度，就考虑修改查询，使其不再需要临时结果文件。

CPI4326－－在连接位置&11 处理文件&1

这条信息提供了用访问路径访问文件数据时规定的连接位置，连接位置属于文件连接的顺序。

文件连接的顺序对查询的效能有明显示的影响。把有较小数量的选择记录文件与有较大数量的选择记录文件连接时，系统处理两个带有不同数量的选择记录文件的连接会更有效。例如，连接两个文件，有较少选择记录的文件在连接位置 1，而有较多选择记录的文件在连接位置 2。

如果规定了 GROUP　BY 或者 ORDER　BY 子句，此时查询中的文件引用子句中全部列时，文件在最后连接顺序当中将成为第一个文件。如果被引用文件是一个大文件，查询就会变得很慢，为了提高性能，考虑下面方法之一：

从一个不同的文件加一个额外的列到子句中，用一个临时的结果表允许系统用最有效的连接顺序来排序。

在 ORDER　BY 子句中规定 ALWCPYDTA(*OPTIMIZE)参数，系统会用最有效的连接顺序来排序文件。

当由于上述建议而改变时，查询一个临时结果表可以用来改变连接顺序。在大多数情况下，连接顺序改进的效能会弥补由临时结果导致的性能损失。

如果查询使用 JOIN 子句或者引用在文件规范表内的一个连接逻辑文件，那么文件规定的顺序会帮助决定优化使用的连接顺序，如果查询包括连接逻辑文件，或者用 JOIN 子句规定左外部连接或异常连接，优化就不能改变文件连接。

CPI4327－－在连接位置 1 处理文件&1。

这条信息提供在用到达顺序文件中选择记录时，连接的主文件或第一个文件的名字。

看先前的信息 CPI4326，它是有关连接位置和连接性能的信息。

CPI4328－－文件&4 的访问路径由查询所用

这条信息指出查询使用已存在的访问路径

使用访问路径的原因参见信息帮助。

CPI4329－－文件&1 使用到达顺序访问路径

没有访问路径用来访问规定文件中的数据，记录按到达顺序依次检索。

如果规定记录选择，用访问路径可以提高查询性能。

如果不存在访问路径，可以生成一个与记录选择中某一字段匹配的键字段，如果记录选择（WHERE 子句）选择文件中 20%或更少的记录，可以仅建立一个访问路径。

为了强制使用已存在的访问路径，修改查询中的 ORDER BY 子句来规定访问路径的第一个键字段。

CPI432A－－文件&1 的查询优化超时

当花费在优化查询所用的时间超出这行查询估计的有关时间，且超出被查询文件中的记录数时，优化会停止考虑访问路径。一般的说，文件中记录越多，要考虑的访问路径数量越大。

当估计这行查询的时间超出后，优化使用当前最好的方式进行查询，或者找到一个访问路径来获得好的性能，或者必要的话建立访问路径。超过估计的查询运行的时间，意味着优化不考虑使用最佳访问路径运行查询。

信息帮助中有优化超过估计时间前需考虑的访问路径列表。为确保一个访问路径是优化考虑的，可规定与此访问路径有关的逻辑文件做为查询文件。优化会首先考虑在查询中或 SQL 语句中规定的文件访问路径，记住，SQL 索引是不能查询的。

可以删除不再需要的访问路径。

CPI432B－－子选择做连接查询处理

两个或多个 SQL 子选择由查询优化合并在一起且做一个连接查询处理，一般来讲，这种处理方式是一个不错的性能选项。

CPI432C－－考虑文件&1 所有的访问路径

优化考虑所有建立在某个文件上的访问路径，既然优化检查文件所有的访问路径，就可以确定文件当前最佳访问路径。

信息帮助包括访问路径列表。每个访问路径有一个原因码，原因码解释为何不用访问路径。

CPI432D－－使用附加的访问路径原因码

这一信息前发出了 CPI432A 和 CPI432C 信息，由于信息长度的限制，由 CPI432A 和 CPI432C 使用的一些原因码在 CPI432D 的信息帮助中加以解释。使用这个信息的信息帮助去解释 CPI432A 或 CPI432C 返回的信息。

CPI432E－－选择字段被映射为不同属性

这一信息指出查询优化不能考虑用索引处理一个或多个查询选择规定，如果有索引可用，可用来限制查询仅处理少数几行，那么查询性能可以起作用。

比较值和比较列的属性必须匹配，否则会发生转换以使之相符，一般的这种转换发生在有最小属性的值映射给其他值的属性，当比较列的属性已经映射或与比较值可进行比较时，优化就不再使用索引来完成这个选择。

CPI4338－－使用访问路径&1 处理文件&2 的位映象

优化选择使用多个访问路径，用连接查询选择（WHERE 子句）来建立位映象，结果位映象指示出实际选择的记录。

理论上说，每个记录包括一位位映象，被选择的记录对应位为‘1’，其他位为“0”。

位映象一旦建立，即被用于防止在未被查询选择的记录做映象，位映象的使用取决于位映象是与到达顺序或主访问路径一起处理。当位映象与到达顺序一起处理时，在些信息前会发送 CPI4327 或 CPI4329，这时，位映象用来帮助从表中选择查询要选择的记录映象。

当位映象与主访问路径一起使用，在此信息前会发送信息 CPI4326 或 CPI4328。在从表中映象记录前，要再检查由主访问路径选择的记录映象。

20.2.5 性能信息及打开数据路径

以下 SQL 运行时间信息引用打开数据路径。

打开的数据路径（ODP）定义在打开游标或运行其他 SQL 语句时产生的、一个内部目标，它直接联接数据，这样输入/输出的操作才得以进行，在 OPEN、INSERT、UPDATE、DELETE 和 SELECT INTO 语句中用 ODPs 完成对数据的操作。

即使用 SQL 关闭游标，SQL 命令已经运行，数据库管理在多数的情况下会保留 SQL 操作的相关 ODP 以便在下一次运行时再使用。这样，一个 SQL CLOSE 命令可以关闭游标但留下有用的 ODP，在下一次打开光标时再次使用，这能明显减少运行 SQL 命令的处理和响应时间。

当反复运行 SQL 命令时，重复使用 ODPs 是争取快速操作重点考虑的问题。

以下信息在 SQL 运行时产生：

SQL7910－－所有 SQL 游标关闭

当作业调用堆栈中不再有运行 SQL 命令程序时，发送此信息。

除非规定了 CLOSQLCSR(*ENDJOB)或是 CLOSQLCSR(*ENDACTGRP)，重用 ODPs 的 SQL 环境垮越程序调用，直到运行 SQL 语句的实际程序完成。除了与*ENDJOB *ENDACTGRP 游标相关的 ODP 外，所有 ODPs 当在调用堆栈所有 SQL 程序完成及 SQL 环境结束时被删除。

这种完成处理包括关闭光标，删除 ODPs，去掉准备的命令，及解锁。

放在应用程序中第一个运行的 SQL 语句可使 SQL 环境在应用执行期间保持活动，这就允许反复调用程序时，可在其他 SQL 程序中重用 ODPs，也能规定 CLOSQLCSR(*ENDJOB)或 CLOSQLCSR(*ENDACTGRP)。

SQL7911－－重用 ODP

这一信息指出语句最后一次运行或是为游标运行 CLOSE 语句时，ODP 没被删除，现在即可再次使用。这样做通过消除不必要的打开、关闭操作而有效利用资源。

SQL7912－－ODP 生成

找不到能够再次使用的 ODP，第一次运行这条语句或打开游标时，总是必须生成一个 ODP，但如果在每一次运行这条语句或打开游标时，都出现这条信息，则建议用 22.19.1 的内容去修改应用程序。

SQL7913－－ODP 删除

对于每个作业仅运行一次的程序，这一信息是正常的。但，如果这一信息在每一次运行语句或打开游标都出现的话，建议用 22.19.1 的内容去修改应用程序。

SQL7914－－ODP 未删除

如果语句重新运行或是游标重新打开的话，ODP 能再次使用。

SQL7915－－SQL 语句的访问方案已经建立

即使所需表丢失了，AS/400 预编译仍允许生成程序目标，在这种情况下，此程序第一次运行时建立访问方案，此信息指出访问方案已建立且成功的存在程序目标中。

SQL7916－－用于查询使用块操作

在运行这条语句时，SQL 会向数据库管理请求多个记录，而不是一次请求一条记录。

SQL7917－－访问方案未更新

数据库管理为这一语句重建访问方案，但程序不用新的访问方案更新，当前运行程序的另一个作业有程序访问方案的共享锁。程序不能用新的访问方案更新直到作业获得程序访问方案的排它锁，共享锁释放后才能获得排它锁。

语句仍会运行，且使用新的访问方案。然而只要语句运行，访问方案就还会重建一直持

续至程序更新。

SQL7918－－删除重用的 ODP

在这一语句中存在可重用的 ODP，但作业库列表和替换规定已经修改了。

现在语句引用不同的文件或使用与已有 ODP 中不同的替换规定，已有的 ODP 不能重用，必须生成新的 ODP。为可重用 ODP，要避免修改库列表和替换规定。

SQL7919－－在 FETCH 或 SELECT 中需要数据转换。

当往主变量中映象数据时，需要数据转换。当将来运行这些语句时，要求做数据转换的就要慢于没有要求转换的。语句能成功运行，消除这些数据转换则能提高性能，例如，在把一定长度的字符串映象给不同长度字符串的主变量时，会由于发生数据转换而发出此信息。在把数值值映象给不同类型的主变量时（十进制整数），也会产生这种错误。为避免多数的转换，使用与列匹配的主变量类型和长度。

SQL7939－－在 INSERT 或 UPDATE 中需要数据转换。

插入和更新值的属性和接受值的列的属性是不同的，既使值必须转换，它们就不能直接移入列中，如果插入或更新值的属性与接收值的列的属性匹配，可以提高性能。

第二十一章　使用 AS/400　DB2 预测查询管理

如果查询估计或预计的运行时间（弹性执行时间）过长，AS/400　DB2 预测查询管理会停止查询的开始，管理在查询运行前开始而不是在查询运行的同时动作，管理器可以在 AS/400 的交互或批作业中使用,它可以用在 AS /400　DB2 查询的所有界面且不限制使用 SQL 查询。

管理器在查询开始之前对其进行预测和停止的能力是很重要的,在长时间运行查询又非正常结束查询且得不到任何结果，这会浪费系统资源。

AS/400　DB2 的管理器以估计查询时间为基准，如果查询估计的运行时间超过用户限定时间，从开始就会被停止查询。

时间限制由用户定义,在 CHGQRYA　CL 命令中的 QRYTIMLMT 参数中规定以秒为单位的时间值，没有 SQL 语句设置限制。

管理器是与查询优化一起使用的,当用户要求 AS/400　DB2 运行查询时,发生下列情况：

1.由优化评估查询访问方案

作为评估的一部分，优化预测或估计查询的运行时间，这有助于决定访问的最好途径来从查询中获取数据。

2.估计的运行时间与当前作业或用户对话中有效的用户定义查询时间限制做比较。

3.如果预计查询的时间少于或等于查询时间限制，查询管理器让查询不间断地运行，并且不向用户发送信息。

4.如果超出时间限制，查询信息 CPA4259 发送给用户，这个信息指出估计的查询处理时间 XX 秒超过 YY 秒的时间限制。

注：对此信息建立一个缺省的回答，这样用户就不用回答该信息，从而结束这一查询要求。

5.如果没用缺省回答，用户可以选择以下做法之一：

在实际开始运行前结束查询要求。

即使运行时间超出管理器时间限制仍继续运行查询。

21.1 取消查询

当一个查询希望运行比预先时间限制长一些时，管理器发出询问信息 CPA4259，用户进入 C 取消该查询或者进入 I 忽略时间限制，使查询运行直到完成。如果用户进入 C，发出逃逸信息 CPF427F 给 SQL 运行时码，SQL 返回 SQLCODE －666。

21.2 一般实施考虑

当使用管理器时要记住优化估计的查询运行时间仅仅是估计值，实际的查询时间可能比估计的时间长或者短，但是两个值大致相同。

21.3 用户应用程序实施的考虑

在 CHGQRYA 命令中对管理器规定的时间限制是为作业或交互用户对话建立的。CHGQRYA 命令也可以导致管理器影响系统中不是当前作业的其它作业。这是由 JOB 参数实现的。在源作业运行 CHGQRYA 命令以后，在目的作业上的管理器的影响不依赖于源作业。查询时间限制在作业或用户对话期间都保持有效，直至用 CHGQRA 改变了时间限制。，在程序控制下，查询时间根据生成应用的功能、日工作时间和可用操作系统资源的总量，用户能得到不同的查询时间限制。这在平衡系统资源的临时查询需求时提供很大的灵活性。

21.4 对查信息的缺省回答控制

系统管理员可以使用下面的 CHGJOB 命令来控制交互作业用户忽略数据库查询询问信息：

如果在 CHGJOB 命令的 INQMSGRPY 参数规定*DFT，那么交互用户看不到询问信息，查询立即被取消。

如果在 CHGJOB 命令的 INQMSGRPY 参数规定*RQD，那么交互用户可以看到询问信息，必须回答询问。

如果在 CHGJOB 命令的 INQMSGRPY 参数规定*SYSRPYL，系统回答列表用来决定交互的用户能否看到询问和是否有回答的必要。系统回答列表项基于用户配置文件名、用户标识或处理名来制定不同的缺省回答。

对于查询信息 CPA4259 的信息数据，可用全限定的作业名。这允许用键字 CMPDTA 来选择处理应用或用户配置文件的系统回答列表项，用户配置文件名有 10 个字符，从 51 列开始，处理名字为 10 个字符，从 27 列开始。下面的例子加一个回答列表项，规定用 C 做缺省回答，它取消用户配置文件名是“QPGMR”的作业请求：

```
ADDRPYLE  SEQNBR(56) MSGID(CPA4259) CMPDTA(QPGMR       51) RPY(C)
```

下面的例子应答一个列表项，它规定用 C 做缺省回答，取消处理过程名为“QPADEV0011”作业的请求。

```
ADDRPYLE  SEQNBR(57) MSGID(CPA4259) CMPDTA(QPADEV0011 27) RPY(C)
```

21.5 便用性能测试管理

查询管理使你不必运行几个反复的查询来优化性能，如果查询时间限制用 CHGQRYA

QRYTIMLMT（0）设置为零，查询信息总是送往用户，指出时间超过了查询时间限制给程序员提示查询信息，并且从 PRTSQLINF 命令也能看到同样的信息。

此外，如果查询被取消，查询优化要评估访问方案且把优化信息送到作业日志中，即使作业不在调试方式也发生这种情况。用户或程序员可以浏览在作业日志中的优化信息，看是否要用附加调整来得到优化的查询性能，实际的数据查询不再活动，所以能节省系统资源，如果被查询的文件有大量记录，这种方法能明显的节省系统资源。

21.6　例子

要设置或修改当前作业或用户会话的查询时间限制，用 CHGQRYA 命令，用下面的命令可以把查询时间限制设为 45 秒：

```
CHGQRYA    JOB(*) QRYTIMLMT(45)
```

如果用户运行查询估算时间不到或等于 45 秒，则查询不中断运行，这个时间限制在作业或用户会话期间保持有效，直到用 CHGQRYA 修改。

如果查询运行时间估算是 135 秒，则会有信息发给用户，指出运行时间超过 45 秒。

要修改查询时间限制，用 CHGQRYA 命令的 JOB 参数。要对作业 123456 /USERNAME/JOBNAME 设置查询时间限制为 45 秒，用下面命令：

```
CHGQRYA    JOB(123456/USERNAME/JOBNAME) QRYTIMLMT(45)
```

如果运行查询时间不大于 45 秒则正常运行查询，否则会将信息送到用户，指出查询时间超时，这个时间限制在此作业运行期间一直有效，直到用命令 CHGQRYA 来改它。

第二十二章　AS/400 DB2 数据管理和查询优化

这一章为设计一个 SQL 更有效地利用系统资源的程序给出一些指导。做为一般原则，大部分的指导可以被忽略，不会影响结果的正确性，但是如果在设计程序运用这些方法，会使程序运行更加有效。

注：本章信息很复杂，阅读本章，会得到有关 AS/400 系统的许多有帮助的信息。

如果能够知道 AS/400 DB2 如何处理查询，那么就很容易理解本章讨论的有关性能的影响。有两个 AS/400 DB2 的重要部分：

1.数据管理方法

这些方法是用来从磁盘中取数据的规则，这个方法包括使用索引和行选择技术。另外，也可用并行访问方法与 DB2 对称多处理操作系统特性。

2.查询优化

查询优化可用来实施查询和选择最有效的技术。

22.1　数据管理方法

这节介绍 AS/400 DB2 和 AS/400 特许内码访问数据的基本技术。

22.1.1 访问路径

访问路径用来规定查询中定位数据的方法，一条访问路径可以是索引的，顺序的或两者结合的。

22.1.1.1 顺序访问路径

一条顺序访问路径用来安排存在表中的行定位在查询规定的数据顺序，用顺序访问路径处理表类似于在传统系统中处理顺序的或直接的文件。

22.1.1.2 键字顺序访问路径

键字顺序访问路径提供按键字字段（索引）的内容来访问表，键字顺序是检索行的顺序。这种访问路径，在表中增加或删除行时或者修改索引列内容时是自动维护的，键字访问路径的最好例子是用 GREATE INDEX 语句来生成索引。

用来生成键字顺序访问路径所用的列最好是：

在 WHERE 子句中选择谓词频繁引用的列，

在 GROUP BY 和 ORDER BY 子句中频繁引用的列，

用来连接表所用的列。

如果想进一步了解访问路径，请参考数据管理。

22.1.2 访问方法

特许内码和 AS/400 DB2 共享访问方法，特许内码做包括选择、连接、散列和访问路径建立的低级处理。

查询优化处理对每个查询选择最有效的访问方法，并且在访问方案中保留这些信息，类型依赖行的数目，页故障的数目（1）及其它限制。

优化用于检索数据的可能方法。包括：

数据空间扫描方法(22.1.2.4)

并行预取方法(22.1.2.5)

键字选择方法(22.1.2.7)

键字定位方法(22.1.2.9)

并行表或者索引预装入(22.1.2.12)

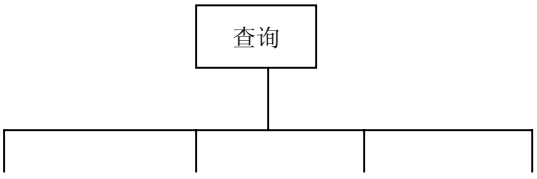
索引之索引方法(22.1.2.13)

仅索引访问方法(22.1.2.11)

散列方法(22.1.2.14)

位图处理方法(22.2)

DB2 对称的多处理特性提供优化及包括并行处理的检索数据方法。对称多处理(SMP)是对单系统并行形式的改进，这里用多处理共享内存和硬盘资源，这样就改进了单处理的最终结果。这种并行处理意味着数据库管理能同时有多个系统处理器同时处理一个查询，这样，CPU 边界查询性能明显改善，这是由于在多处理器系统上用分布处理器装入跨越多个处理器。



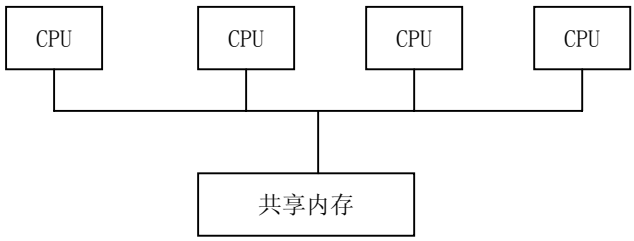


图 22-1　数据库对称多处理

在你的系统如果有对称多处理特性可用下列方法优化：

并行数据空间扫描方法(22.1.2.5)
并行键字选择方法(22.1.2.7)
并行键字定位方法(22.1.2.9)
并行仅索引访问方法（并行或非并行）(22.1.2.10)
并行散列方法(并行或非并行)(22.1.2.13)
并行位图处理方法(22.2)

(1)当程序引用不在主存的 4 字节页时，发生中断。

22.1.2.1　排序

为保证结果的特殊顺序，必须规定 ORDER BY 子句，在用并行访问方法前数据库管理用一种顺序方式处理表中的行（键字顺序），即使在原查询要求中没有包括排序，在结果中也能排序。因为并行方法会同时处理表行的一块和索引值,检索的排序变得更随机且不可预测，ORDER BY 子句是保证结果有规定顺序列的唯一方法，但是，只有当绝对必要时，才可指定排序列请求，因为排序结果会增加 CPU 的利用和响应时间。

22.1.2.2　实现并行处理

应用程序和用户必须实现对查询的并行处理，优化不会自动使用并行作为访问方式，可用系统值 QQRYDEGREE 和 CHGQRYA 命令的 DEGREE 参数来控制查询优化所用的并行级别。详细内容请看 22.27。

系统启动时，会生成一数据系统任务给数据管理使用，DB 管理用这些任务处理从硬盘设备获得的数据，既然这些任务可同时在多处理器上运行，就可减少查询时间，这些任务甚至可以做很多并行查询的 I/O 和 CPU 处理，I/O 和 CPU 资源计数会传送给作业，应用程序这类的 I/O 和 CPU 资源总计，可由 WRKACTJOB 命令准确地显示。

22.1.2.3　自动数据展开

AS/400　DB2 会跨过硬盘设备在可用 ASP 上自动展开定位数据，这就保证了数据的展开无用户干予。数据展开让数据管理很容易在不同硬盘设备上并行处理成块记录。

即使 AS/400　DB2 会跨越硬盘设备在一个 ASP 中展开数据，有时数据宽度的定位（数据的连续集）可能无法平缓地展开。当设备空间分配不均，或新设备添加到 ASP 时,会发生这种情况。可用保存，删除，然后重存表来再次展开数据空间的分配。

22.1.2.4　数据空间扫描访问方法

在无法保证序列表中的行用无序处理时，若想得到特定序列的结果必须规定 QRDER　BY

子句。读表中所有行，每行都要符合选择限制，给应用程序只返回适合限制的行。

由于下列原因数据空间扫描是有效的：

因为处理给定页的所有行，并且一旦页在主存，就不用再检索，所以页的 I/O 操作的数减小。

数据库管理很容易预测数据检索的页顺序。由于这个原因，数据库管理能调度从辅存到主存的页的 I/O，这通常叫作予取。这样做当数据库管理需要访问数据时在主存中有可用的页。

当要选择的行占很大百分比时用这个方法非常好，百分比通常是 20%或更多。

当从表中选择的行包括删除的行时，数据空间扫描处理可以造成不利影响。这是因为删除操作仅对删去行做标记，对数据空间扫描处理，数据库管理读所有删除的行，因为删除的行没有一个真的没有了。可以使用 RGZPFM 命令去消去删除的行，对物理文件规定 REUSEDLT(*YES)也可重用删除记录的空间。在用 CREATE TABLE 语句生成表后，可在 CRTLF 命令中指定 REUSEDLT(*YES)。

数据空间扫描处理在表中选择，因为要检查表中所有行，这导致使用不必要的 I/O 和处理单元资源。

由 QPRTSQLINF 命令建立的信息描述使用数据空间选择方法的 SQL 程序的查询，会出现下面内容：

```
SQL4010  Arrival sequence access for file 1.
```

特许内码当数据空间扫描处理时，会对选择使用两种算法之一，中间缓冲选择和数据空间元素选择。

下列伪码解释中间缓冲选择算法：

```
DO  UNTIL  END  OF  FILE

1、定位下一行（或第一个）记录；
2、把所有列值映射给中间缓冲，完成所有派生的操作；
3、使用复制到中间缓冲的列值对 TRUE 和 FALSE 值做选择评估；
4、IF  选择为 TURE，
          THEN
          把值从中间缓冲复制到用户的回答缓冲中，
          ELSE
          无操作
END
```

数据空间项算法如下：

```
DO  UNTIL  END   OF   FILE

1、计算检索限制，限制通常为已在活动内存中的记录数，或已做 I/O 请求装入到
   内存中；
2、DO  UNTIL（到达检索限制或记录选择限制是 TURE）
```

a.定位下一个（或第一个）记录；

b.评估选择限制，它不需要派生值而直接处理数据空间记录。

END

3、IF 选择为真

THEN

a.把所有列值映射到中间缓冲，完成所有的派生操作；

b.把值从中间缓冲复制到用户回答缓冲。

ELSE

END 无操作

由于下列两个原因，数据空间项算法提供比中间缓冲选择更好的性能：

数据移动和计算仅在被选择的记录上完成。

数据空间项选择算法第二步的循环生成一个可执行码，当实际选择很小百分比记录时，AS/400 DB2 运行这个很小的程序直到找到记录。

这类查询实际不需使用数据空间扫描方法，任何查询接口可以利用这种改进，然而，下列内容决定一个选择能否作为数据空间选择完成：

谓词中没有一个操作数是任何类型的派生值、函数、子串、连接或数值表达式。

当选择谓词的两个操作数是数值列，两列都有相同类型、范围和精度，否则操作数要映射为派生值，例 DECIMAL(3,1)仅能与另一个 DECIMAL(3,1)列比较。

当选择谓词的一个操作数是数值列而另一个是字母或主变量，类型必须相同且字母/主变量的精度和范围必须少于或等于列的精度和范围。

如果表是由 SQL CREATE TABLE 生成的，那么只能包括压缩十进制或数值类型列的选择谓词。

不能在选择谓词中引用变长字符列。

当选择谓词的一个操作数是字符列，另一个是字母或主变量，主变量的长度不能大于列的长度。

字符列数据的比较不需要 CCSID 或键盘转换。

避免中间缓冲选择是很重要的，因为在数据项选择高于 70-80%时，CPU 使用和响应时间会降低。大多数从数据空间选择受益的查询是那些实际选择少于文件的 60%的，选择记录的百分比越低，越要注意受益。

21.1.5 并行预取访问方法

AS/400 DB2 使用并行预取处理来缩短长时间运行的 I/O 边界数据空间扫描查询所需的时间。

这个方法与数据空间扫描方法有相同特性，除了 I/O 处理用并行完成，这是由对表的预取数据启动多输入流完成。当下列为真时这个方法最有效：

数据展开跨越多个磁盘设备。

查询不是 CPU 处理集中的。

对每个输入流保持数据收集有相当数量的主存。

以前讲过 AS/400 DB2 能自动展开数据跨越磁盘设备而不用用户干预，允许数据库管理并行预取表数据。数据库管理使用任务从不同磁盘设备检索数据。通常请求是针对整个范围（数据的连续集），这样由于磁盘设备能用平滑的顺序的访问数据，故性能提高了。由于这种优化，并行预取能比 SETOBJACC 命令更快的把数据预装入到活动主存。虽然 AS/400 DB2

在一个 ASP 中可跨越磁盘展开数据，有时数据空间范围的分配不能平缓的展开，当在设备中有不平缓的空间分配时或新的装置加到 ASP 中时会发生这种情况。数据空间的分配可用保存和重存表来重展开。

优化查询选择能利用这类实施优点来候选查询，估计查询处理所需的 CPU 时间，并把估计时间与输入处理所用时间的总量相比较，当估计的进程时间超过 CPU 时间时，优化查询选那些用并行 I/O 实施的查询。

并行预取要求并行 I/O 处理必须由系统值 QQRYDEGREE 或 CHGORYA 命令中的 DEGREE 参数激活，详细内容请看 22.27。由于用并行预取处理的查询会过份地使用主存和磁盘 I/O 资源，所以要限制和控制使用并行预取的查询。并行预取使用多个磁盘臂，但每个查询用的 CPU 很少，并行预取 I/O 将剧烈地使用 I/O 资源。在系统中允许并行预取查询与超落实 I/O 子系统一起可加重超落实问题。

可以在共享存储池中用*CALL 分页选项运行作业，这样，能更有效地利用活动主存。

AS/400　DB2 使用自动系统调整去决定在这个处理要用多少内存。运行时，如果内存统计指出它没有超落实主存资源，则特许内码允许使用并行预取，详细内容请看数据管理。

并行预取要求有足够的主存去储存通过多次输入流取用的数据，对于大的文件，典型的扩展尺寸是 1 兆字，即必须有 2 兆内存可用才能同时使用二个输入流。在池中增加可用内存可允许使用更多的输入流。如果有大量的可用内存，表的全部数据空间在打开查询时可以装入到活动内存中去。

由 PRTSQLINF 命令生成的信息说明查询使用并行预取访问方式的 SQL 程序查询，出现的信息为：

```
SQL4023   Parallel dataspace pre-fetch used
```

22.1.2.6　并行数据空间扫描方法（仅当安装了 DB2 对称多处理性能时才可用）

AS/400　DB2 用此并行访问方法缩短长时间运行的数据空间扫描查询所需的处理时间，并行数据空间扫描方法象并行预取访问方法一样会减少 I/O 处理时间。另外，在系统上有多个处理器运行时，把数据空间扫描处理分割成可同时在多个处理器上运行的任务，这样减少了查询时间，所有选择和列处理都在任务中完成，应用程序作业调度这些任务的工作请求，把结果合并到结果缓冲区并返给应用程序。

当下列为真时，此方法最有效：

数据展开跨越多个硬盘设备

系统有多个处理器可用

有大容量主存可用容纳数据及结果缓冲区。

象前面提到的，AS/400　DB2 不要用户干预能自动展开数据跨越硬盘设备，且允许数据库管理用并行方式预取表数据。

查询优化选择能利用此类操作优点的候选查询，优化通过评估处理查询所用的 CPU 时间和比较输入处理所用的时间总数，优化根据它计算所用的任务数来减少数据空间扫描的时间，它计算系统中基于处理数的任务数，作业池中可用的内存，DEGREE 查询属性的当前值，如果并行数据空间扫描是最快的访问方法，就选择它。

并行数据扫描需要用系统值 QQRYDEGREE 或 CHGQRYA 中的 DEGREE 参数来激活 SMP 并行处理。详细内容请看 22.27。

并行数据空间扫描不能用于要求以下之一的查询：

规定*ALL 的落实控制级。

嵌套的循环连接方法。

向后翻卷。例如，对用 OPNQRYF 命令定义的查询一般不能使用并行数据空间扫描。（这类应用可以定位最后一个记录）。SQL 定义的查询若定义做可翻卷的，可以使用这种方法。

在产生临时结果期间可使用平行数据空间扫描法，例如排序或散列操作，而不管用什么界面来定义查询。

游标位置的恢复。例如做为 SQL ROLLBACK HOLD 语句或 ROLLBACK 命令的结果，要恢复游标的位置。提交控制级不是*NONE 的的 SQL 应用程序要规定*ALLREAD，做为预编译参数 ALWBLK 的值来允许使用这种方法。

更新或删除能力。

用页选项*CALC 在共享缓冲池运行这个作业，这样就会更有效的使用活动内存。

并行数据空间扫描需要活动的内存来缓冲检索来的数据，并为每个任务分开结果缓冲区。每个任务需要的内存大约是两兆字节，要有四个数据空间扫描同时运行必须有八兆内存可用。在存储池中增加可用内存数量，允许使用更多的输入流，访问有大量变长列的表的查询或生成比表实际记录长的结果值的查询，需要为每个任务分配更多的内存。

如果多数记录锁冲突或者有发生数据映射错误并行的数据空间扫描性能会受限制。

22.1.2.7 键字选择访问方法

这种访问方法需要键字顺序访问路径。读全部索引，所有对索引的键字列引用的选择标准都要应用在索引中。这种方法的优点是数据空间仅访问那些要读检索行附合索引选择标准的那些数据，不能由键字选择方法完成的选择是在数据空间级完成的。

如果检索条件适合于大量的行，键字选择存取方法是非常昂贵的，这是因为：

要处理整个索引

从索引中选择的每个键字，发生一个数据空间随机的输入输出。

正常情况下当检索条件适合于相当多的行时，这种优化选择用数据空间扫描处理。如果被选择的键字小于 20%或者强迫使用索引操作，优化会选择键字选择方法。强制使用索引的选项包括：

排序

分组

连接

在这些情况下，优化可以选择生成一个临时索引而不使用已有的索引。当优化生成一个临时索引时，它使用 32K 页尺寸，用 CREATE INDEX 语句生成的索引通常只用 4K 页尺寸。当建立临时索引时，优化能处理尽可能多的选择。几乎所有由优化建立的临时索引都是选择/省略或稀少索引。最后，优化在生成索引时也能使用多并行任务。依据从很少几页翻动的性能改进，页尺寸是不同的，使用并行任务来生成索引的能力足够抵消生成一个索引的花费，数据空间选择用来建立临时键字访问路径。

如果查询规定排序需要索引要用键字选择方法，就可以用下列预编译参数的组合在查询分类中使用排序，以提高查询性能：

_ ALWCPYDTA(*OPTIMIZE), ALWBLK(*ALLREAD), and COMMIT(*CHG or *CS)

_ ALWCPYDTA(*OPTIMIZE) and COMMIT(*NONE)

22.1.2.8 并行键字选择访问方法（仅在安装 DB2 对称多处理特性时才可用）

对于并行键字选择访问方法，可能的键字值被逻辑的分开，每一部分由分开的任务处理，

就像使用索引选择访问方法一样。查询优化决定当前处理的分部数，因为不按顺序处理键字，如果用索引做排序，优化就不能使用这个方法。包含来自现存索引键字大部分的键字分区，将来会分开作为别的部分来处理。

以下例子给出，优化选择键字选择方法的查询：

```
CREATE INDEX X1
            ON EMPLOYEE(LASTNAME,WORKDEPT)

DECLARE BROWSE2 CURSOR FOR
  SELECT * FROM EMPLOYEE
  WHERE WORKDEPT = 'E01'
  OPTIMIZE FOR 99999 ROWS
```

如果优化选择用四级并行运行这个查询，以下是可同时处理的逻辑键字部分：

LASTNAME 的值引导字符分区开始	LASTNAME 的值引导字符分区结束
'A'	'F'
'G'	'L'
'M'	'S'
'T'	'Z'

若第一、二分区有很少几个键字，这些键值的处理会完成得比第三、四分区快些，在完成前两个分区后，剩下的后两个分区键值会进一步分割。以下给出在第一、二分区完成处理后可以处理四个分区，会有分割发生：

LASTNAME 的值引导字符分区开始	LASTNAME 的值引导字符分区结束
'O'	'P'
'Q'	'S'
'V'	'W'
'X'	'Z'

并行键字选择不可用于要求以下之一的查询：

规定*ALL 的落实控制级。

嵌套的循环连接方法。

向后翻卷。例如，对用 OPNQRYF 命令定义的查询一般不能使用并行数据空间扫描。（这类应用可以定位最后一个记录）。SQL 定义的查询若定义做可翻卷的，可以使用这种方法。在产生临时结果期间可使用平行数据空间扫描法，例如排序或散列操作，而不管这用什么界面来定义查询。

游标位置的恢复。例如做为 SQL ROLLBACK HOLD 语句或 ROLLBACK 命令的结果，要恢复游标的位置。提交控制级不是*NONE 的 SQL 应用程序要规定*ALLREAD，做为预编译参数 ALWBLK 的值来允许使用这种方法。

更新或删除能力。

用页选项*CALL 在共享缓冲区运行这个作业，这样会更有效地使用活动内存。

并行键字选择需要 SMP 并行处理，这是由系统值 QORYDEGREE 或由 CHGORYA 中的中

的 DEGREE 参数激活的。请看 22.7。

22.1.2.9　键字定位访问方法

此访问方法与键字选择访问方法十分相似，它们都需要键字顺序访问路径，用键字选择访问方法，从索引开始启动处理直至末尾。用键字定位访问方法，对键字符合某些或全部选择标准范围内的索引直接做选择，从这个范围内读所有键字，再执行余下的键字选择。这与键字选择方法执行的选择很相似。任何通过键字定位或键字选择完成的选择在数据空间级上执行，因为键字定位只处理索引中的键字的子集，所以键字定位的性能比键字选择方法性能更好。

当选择的行比例少于 20%时，键字定位方法最有效。如果超过 20%的行时，优化一般会选择：

使用数据空间扫描处理（如不要求索引）
使用键字选择（如果要求索引）
使用查询分类例程（如给出条件）

对于不要求索引的查询（无排序、分组或连接操作），优化会试图寻找已有的索引做键字定位，如果找不到已有的索引，优化会停止使用键字访问数据，因为用数据空间扫描处理比建立一个索引然后运行键字定位更快。

下例解释优化选择键字定位方法的查询：

```
CREATE INDEX X1 ON EMPLOYEE(WORKDEPT)

DECLARE BROWSE2 CURSOR FOR
  SELECT * FROM EMPLOYEE
  WHERE WORKDEPT = 'E01'
  OPTIMIZE FOR 99999 ROWS
```

此例中，数据库支持用 X1 定位到值为 WORKDEPT 的第一个索引输入。对于每个值为‘E01’的键字，它随机访问数据空间(2)且选择行，当索引选择移到索引值 E01 外，查询结束。

注：此例中所有处理的键字项和获取的行都符合选择标准。如果加上不能用键字定位完成的其它选择（例如在多列上选择的列不符合索引的第一个键字列），优化会用键字选择完成尽可能多的其它选择，剩下的选择会在数据空间级上运行。

由 PRTSQLINF 命令生成的信息描述了在 SQL 程序中的查询，会显示如下：

```
SQL4008  Access path X1 used for file 1.
SQL4011  Key row positioning used on file 1.
```

键字定位访问方法有附加的处理性能，它完成的选择范围越过几个值，例如：

```
CREATE INDEX X1 EMPLOYEE(WORKDEPT)

DECLARE BROWSE2 CURSOR FOR
  SELECT * FROM EMPLOYEE
  WHERE WORKDEPT BETWEEN 'E01' AND 'E11'
  OPTIMIZE FOR 99999 ROWS
```

以上例子中，数据库支持定位第一个索引项的值为'E01'，并且处理行，直到处理最后一个值为‘E11’的索引项。

由 PRTSQLIHF 命令生成的信息描述了在 SQL 程序中的查询，会显示如下：

```
SQL4008  Access path X1 used for file 1.
SQL4011  Key row positioning used on file 1.
```

做为这种访问方法的进一步扩充，可以使用叫做多范围键字定位，它允许在多列上索引的第一个键字列上选择行的多个值范围。

```
CREATE INDEX X1 ON EMPLOYEE(WORKDEPT)

DECLARE BROWSE2 CURSOR FOR
  SELECT * FROM EMPLOYEE
  WHERE WORKDEPT BETWEEN 'E01' AND 'E11'
    OR WORKDEPT BETWEEN 'A00' AND 'B01'
  OPTIMIZE FOR 99999 ROWS
```

在上例中，因为二次定位和处理技术，对每个值范围使用一次。

由 PRTSQLINF 命令生成的信息描述了在 SQL 程序中的查询，它可以显示为：

```
SQL4008  Access path X1 used for file 1.
SQL4011  Key row positioning used on file 1.
```

所有键字定位例子都只用一个键，是索引中的最左键，键字定位也处理多个键。（虽然这些键必须是连在最左键的）。

```
CREATE INDEX X2
   ON EMPLOYEE(WORKDEPT,LASTNAME,FIRSTNME)

DECLARE BROWSE2 CURSOR FOR
  SELECT * FROM EMPLOYEE
  WHERE WORKDEPT = 'D11'
    AND  FIRSTNME = 'DAVID'
  OPTIMIZE FOR 99999 ROWS
```

因为两个选择键字（WORKDEPT 和 FIRSTNME）不是连着的，此例不支持多键字定位。因此，只有选择 WORKDEPT = ‘D11’ 部分可提供给索引（单键字定位），当可以接受时，它意味着行处理从第一个键字 ‘D11’ 开始，然后用键字选择对 WORKDEPT 键字值为“011”的 9 个键字，处理 FIRSTNME = ‘DAVID’。

由生成下列的索引 X3，上述查询例子会用多键字定位运行。

```
CREATE INDEX X3
```

```
ON EMPLOYEE(WORKDEPT, FIRSTNME, LASTNAME)
```

多键字定位支持能为选择的各部分做键字定位，这样改善了性能，起始值是由把两个选择值合并为‘D11DAVID’而建立的，选择定位在最左两个键有这个值的索引项上。

由 PRTSQLINF 命令产生的信息描述了 SQL 程序中的查询，显示如下：

```
SQL4008  Access path X3 used for file 1.
SQL4011  Key row positioning used on file 1.
```

下面这个例子给出如何使用键字定位：

```
CREATE INDEX X3 ON EMPLOYEE(WORKDEPT,FIRSTNME)

DECLARE BROWSE2 CURSOR FOR
  SELECT * FROM EMPLOYEE
  WHERE  WORKDEPT = 'D11'
    AND  FIRSTNME IN  ('DAVID','BRUCE','WILLIAM')"
  OPTIMIZE FOR 99999 ROWS
```

查询优化分析 WHERE 子句，然后用相等的格式重写这个子句：

```
DECLARE BROWSE2 CURSOR FOR
  SELECT * FROM EMPLOYEE
  WHERE   (WORKDEPT = 'D11' AND FIRSTNME = 'DAVID')
        OR (WORKDEPT = 'D11' AND FIRSTNME = 'BRUCE')
        OR (WORKDEPT = 'D11' AND FIRSTNME = 'WILLIAM')
  OPTIMIZE FOR 99999 ROWS
```

在重写的查询格式中,对WORKDEPT和FIRSTNME连结的值实际上有三个分开的键值范围。

```
Index X3 Start value       Index X3 Stop value
    'D11DAVID'                 'D11DAVID'
    'D11BRUCE'                 'D11BRUCE'
    'D11WILLIAM'               'D11WILLIAM'
```

键字定位每个范围上完成，明显地减少了选择的键的数目，刚好为三，所有的选择能通过键字定位完成。

在下面例子中，可进一步使用复杂的范围分析。

```
DECLARE BROWSE2 CURSOR FOR
  SELECT * FROM EMPLOYEE
  WHERE (WORKDEPT = 'D11'
              AND FIRSTNME IN ('DAVID','BRUCE','WILLIAM'))
        OR (WORKDEPT = 'E11'
```

```
            AND FIRSTNME IN ('PHILIP','MAUDE'))
      OR (FIRSTNME BETWEEN 'CHRISTINE' AND 'DELORES'
            AND WORKDEPT IN ('A00','C01'))
```

查询优化分析 WHERE 子句并把子句重写相等的格式：

```
DECLARE BROWSE2 CURSOR FOR
  SELECT * FROM EMPLOYEE
  WHERE   (WORKDEPT = 'D11' AND FIRSTNME = 'DAVID')
       OR (WORKDEPT = 'D11' AND FIRSTNME = 'BRUCE')
       OR (WORKDEPT = 'D11' AND FIRSTNME = 'WILLIAM')
       OR (WORKDEPT = 'E11' AND FIRSTNME = 'PHILIP')
       OR (WORKDEPT = 'E11' AND FIRSTNME = 'MAUDE')
       OR (WORKDEPT = 'A00' AND FIRSTNME BETWEEN
                                    'CHRISTINE' AND 'DELORES')
       OR (WORKDEPT = 'C01' AND FIRSTNME BETWEEN
                                    'CHRISTINE' AND 'DELORES')
       OPTIMIZE FOR 99999 ROWS
```

在这个查询中，对 WORKDEPT 和 FIRSTNME 的连接值实际上有七个分开的键值范围：

```
Index X3 Start value       Index X3 Stop value
   'D11DAVID'                 'D11DAVID'
   'D11BRUCE'                 'D11BRUCE'
   'D11WILLIAM'               'D11WILLIAM'
   'E11MAUDE'                 'E11MAUDE'
   'E11PHILIP'                'E11PHILIP'
   'A00CHRISTINE'             'A00DELORES'
   'C01CHRISTINE'             'C01DELORES'
```

键字定位在每个范围上执行，只有键值在范围中之一的那些行能返回，所有选择可用键字定位完成，它明显地改进了查询的性能。

(2)当键可能与数据空间行的顺序不同，会产生随机访问。

22.1.2.10　并行键字定位访问方法（只在安装了 DB2 对称多处理特性时才可用）

使用并行键定位访问方法，已存在的键字范围，被共存在各自的数据库任务中的独立任务同时处理，共存任务的数目由优化控制。查询将启动查询键字范围的处理，直到达到使用并行的级别。当这些范围处理结束，启动列表中下一个，当一个范围处理结束且列表中不再有范围要处理，仍有键处理的范围将被分割，就象用并行键字选择方法一样。数据库管理试图保持所有经常使用的任务，每个任务处理一个键字范围。无论是使用单值，值范围还是多范围键字定位，范围将进一步分配并同时处理。因为不是按顺序处理键字，如果用索引来排序，优化就不能使用这种方法。

如果 SQL 语句用并行四级运行，考虑下面的例子：

```
DECLARE BROWSE2 CURSOR FOR
  SELECT * FROM EMPLOYEE
  WHERE   (WORKDEPT = 'D11' AND FIRSTNME = 'DAVID')
       OR (WORKDEPT = 'D11' AND FIRSTNME = 'BRUCE')
       OR (WORKDEPT = 'D11' AND FIRSTNME = 'WILLIAM')
       OR (WORKDEPT = 'E11' AND FIRSTNME = 'PHILIP')
       OR (WORKDEPT = 'E11' AND FIRSTNME = 'MAUDE')
       OR (WORKDEPT = 'A00' AND FIRSTNME BETWEEN
                                    'CHRISTINE' AND 'DELORES')
       OR (WORKDEPT = 'C01' AND FIRSTNME BETWEEN
                                    'CHRISTINE' AND 'DELORES')
  OPTIMIZE FOR 99999 ROWS
```

数据库管理用下列键字范围启动：

	Index X3 Start value	Index X3 Stop value
Range 1	'D11DAVID'	'D11DAVID'
Range 2	'D11BRUCE'	'D11BRUCE'
Range 3	'D11WILLIAM'	'D11WILLIAM'
Range 4	'E11MAUDE'	'E11MAUDE'
Range 5	'E11PHILIP'	'E11PHILIP'
Range 6	'A00CHRISTINE'	'A00DELORES'
Range 7	'C01CHRISTINE'	'C01DELORES'

范围 1 到 4 在独立的任务同时处理，这四个之中一个范围处理结束，马上开始第五个范围。当另一个范围完成后，启动第六个范围等等。当四个范围中某个处理结束并且列表中没有新的范围启动，留在其它键字范围中的其余工作将被分割，每一部分被独立处理。

并行键字定位不能用于那些有如下需要的查询：

规定*ALL 的提交控制级

嵌套的循环连接方法。

向后翻卷，例如，对 OPNQRYF 命令定义的查询，一般不能使用并行键字定位，因为应用程序试图去定位最后记录且检索先前的记录，规定为可翻卷的 SQL 定义的查询，可用此方法。

在建立临时结果期间可用并行键字定位，例如分类或散列操作，不论用何种界面定义查询。

游标位置的恢复。例如做为 SQL ROLLBACK HOLD 语句或 ROLLBACK 命令的结果，查询需要恢复游标的位置，提交控制级不是*NONE 的 SQL 应用要规定*ALLREAD 做为预编译参数 ALWBLK 的值来允许使用这种方法。

更新或删除能力。

用页选项*CALC 共享存储池运行作业，可以更有效利用活动内存。

并行键字定位需要 SMP 并行处理，可用系统值 QQRYDEGREE 或 CHGQRYA。

22.1.2.11 仅索引访问方法

仅索引访问方法用来与其他键字选择或键字定位方法一起使用，也包括这些方法的并行选项，这些选择的处理与以前讲过的方法没有不同。

所有的数据都是从索引中提取而不是对数据空间的随机 I/O 操作，索引项然后用于任何导出的输入或在查询中规定的结果映象。在下列情况下，优化选择这种方法：

查询中引用的所有列都能在永久索引中或在优化决定生成的临时索引的字段中找到。

数据值必须可以从索引中提取，并以可读的格式返回给用户，换句话说，没有一个符合查询列的键字段有：

— 规定绝对值
— 规定交替分配顺序和分类顺序
— 规定区位或数字分区

对非 SQL 用户，没有变长或空字段能要求键反馈。

下面的例子说明，优化选择将执行仅索引访问的查询。

```
CREATE INDEX X2
   ON EMPLOYEE(WORKDEPT,LASTNAME,FIRSTNME)

DECLARE BROWSE2 CURSOR FOR
  SELECT FIRSTNME FROM EMPLOYEE
  WHERE WORKDEPT = 'D11'
  OPTIMIZE FOR 99999 ROWS
```

在这个例子中，数据库管理用 X2 对 WORKDEPT=‘D11’定位索引项，并从这些项中提取列 FIRSTNME 的值。

注：在执行仅索引访问时，索引键字段不与索引的最左边的键相邻，索引中的任何键字段都可用来为仅索引查询提供数据类似，用索引做数据源，所以数据库管理在选择完成后可结束查询。

由 PRTSQLINF 命令生成的信息描述了在 SQL 程序中的这个查询，会如下显示：

```
SQL4008  Access path X2 used for file 1.
SQL4011  Key row positioning used on file 1.
SQL4022  Index only access used on file 1.
```

注：仅索引访问是在特殊文件上完成的，因此在一些或所有文件的连接查询中可以使用仅索引方法。

22.1.2.12 并行表或基于预装入索引访问方法

一些用键字选择的查询需要大量的随机 I/O 来访问一个索引或一个表，所以索引或表中相当大比例的数据将被引用。AS/400 DB2 试图避免这种随机 I/O，是用在查询处理开始时，初始化基于表或基于索引的预装入实现的。数据并行地装入到活动内存，就如同并行预取一样做法。在表或索引装入内存后，实现了对数据的随机访问而没有额外的 I/O。如果 I/O 并行处理已被激活，基于查询优化的开销会认可从表索引预装入得到好处的查询和目标。详细内容请看 22.27。

并行预装入方法可以与其他任何数据访问方法一起使用，当查询打开时预装入开始，在

预装入完成前，控制返回给应用程序，应用程序继续用其他不必知道预装入的方法来继续取行。

22.1.2.13　从索引产生索引的访问方法

数据库管理可以从一个已存在的索引建立一个临时索引，而不必从数据库中读所有的行。通常讲，这种选择是最有效的方法之一。建立的临时索引只包括符合选择谓词的那些行，这类似于由选择/省略逻辑文件建立的键字访问路径或稀少索引，在下面这些情况时，优化会选择这种方法：

由于使用分组，排序或连接处理查询需要一个索引。

存在永久索引，它选择的列是最左边的键字且最左边的键选择性强。

选择的列不同于 ORDER　BY 或连接的 TO 列。

为使用从索引到索引访问方法，数据库管理要做：

1.在永久索引上与查询选择限制一起使用键字定位。

2.用选择的行项，在新的临时索引上建立索引项。

结果是包括用需要的键字顺序对符合选择标准的行的索引。

一般从索引到索引的访问方法例子如下：

```
CREATE INDEX X1 ON EMPLOYEE(WORKDEPT)

DECLARE BROWSE2 CURSOR FOR
  SELECT * FROM EMPLOYEE
  WHERE WORKDEPT = 'D11'
  ORDER BY LASTNAME
  OPTIMIZE FOR 99999 ROWS
```

在这个例子中，用主键字段 LASTNAME 生成一个临时的选择/省略索引，它仅包含 WORKDEPT ＝“D11”的行的索引项。如果 WORKDEPT ＝“D11”，大约小于 20%的行被选择，由 PRTSQLINF 命令产生的信息描述了 SQL 程序中的查询，显示如下：

```
SQL4012  Access path created from keyed file X1 for file 1.
SQL4011  Key row positioning used on file 1.
```

除了使用从索引到索引的访问方法，也可以指定下面的预编译选项之一来使用查询分类例程。

```
_   ALWCPYDTA(*OPTIMIZE), ALWBLK(*ALLREAD), and COMMIT(*CHG or *CS)

_   ALWCPYDTA(*OPTIMIZE) and COMMIT(*NONE)
```

22.1.2.14　散列访问方法

散列访问方法必用成组或相关方法为处理数据的查询提供一个替换方法，键字顺序访问路径（索引）用来分类和组合数据，在分组或连接查询操作中是很有效的。但假如优化已对查询生成了临时索引，在要求的查询能运行前生成这个索引时，要使用额外的处理器时间和资源。

散列访问方法能完成键字顺序访问路径或做为一个替代方法，对每个选择的行，在行中规定的分组或连接值通过散列功能运行，计算好的散列值用来检索散列表的特定部分。散列表类似于一个临时工作表，但结构不同，它是基于指定查询的逻辑分区。如果在表中找不到行的源值，这就表明这个源值在数据库表中第一次遇到。用这个值来初始化新的散列表项，且根据查询操作完成其它处理，如果在表中找到了行的源值，取回这个值的散列表项；基于请求的操作执行附加的查询（如分组或连接），散列法只能关联（或分组）同一个值，不能用来排列散列表行升序或降序。

除非需要一个临时结果，散列法只有在规定了 ALWCPYDTA(*OPTIMIZE)选项时才能使用，因为由数据库管理建立的散列表是所选行的临时拷贝。

散列算法允许数据库管理建立平衡好的散列表，来随机和分布的分配源数据，散列表本身是基于需要的查询操作和要处理的源值数目来分配的，散列算法保证新的散列表项平滑的跨过散列表分开。为保证在散列表的不同分区进行相同数目项的扫描，这种平衡分布 是必须的。若一个散列表分区包括散列表项的大多数，分区的扫描要检查散列表的大多数项，这不是非常有效的。

既然散列方法典型地顺序处理表中的行，数据库管理很容易预见查询处理所需要的数据库存储页的顺序，这与数据空间扫描访问方法的优点很相似。这种可预见性允许数据库管理能把表页异步的调往主存（也叫做预取），预取对散列方式可做有效的 I/O 操作，以致改善查询性能。

比较而言，用键字顺序访问方法的查询处理导致对数据库表的随机 I/O 操作。检验每个键值，因为数据在索引中的键字顺序与数据库表中行的物理顺序不符，所以 I/O 操作是随机的，随机的 I/O 可降低查询性能，因为它导致不需要的 I/O 和处理器单元资源。

键字顺序访问路径也用于散列方法，键字访问路径可以明显地减少用散列方法处理的表行数目。这可能抵消用键字顺序访问路径的随机 I/O 开销。

在查询打开以前，生成和分布散列表，一旦散列表完成分布给指定的数据库记录，数据管理使用散列表开始返回查询结果，根据需要的查询操作，可能会有附加的处理结果散列表行。

因为表行的块自动扩展，散列访问方法也能用并行完成，以致记录中的一些组同时被散列，它能缩短散列在数据库表中的所有行所用的时间。

22.2 位图处理方法

这种方法在访问数据空间时产生位图，位图处理方法可用于：

在使用键字定位和或键字选择方式组合的键字顺序访问路径时，减少随机的 I/O 发生。

允许用多键字顺序访问路径来访问特殊表。

用这个方法，优化选择一个多个键字顺序访问路径，帮助在数据空间中选择记录。对每个索引分配临时位图，（且初始化），每个位图对每个在数据空间的记录包含一位（bit），对每个索引，键字定位和键字选择方法在选择标准中使用。

对每个选择的索引，与记录有关的位被置为‘1’（即开），不访问数据空间，当索引处理完成后，位图包含数据空间选择记录的信息。对每个索引重复这个过程，如果使用两个以上的索引，临时位图会逻辑的 AND 或 OR 在一起得到一个结果位图。一旦结果位图建立，它避免从数据空间映射记录，除非用查询选择。

产生位图的索引不是实际用来访问选择记录的，这点很重要。基于这个原因，它们被称为第三级索引，而用来访问最终记录的索引称为主索引，主索引用表排序，成组连接和在不使用位图时做选择。

位图处理方法，用来连接主访问方法数据空间扫描，键字选择或键字定位。位图处理，象并行预取和并行表／索引预装入一样不是实际从数据空间选择记录，它是主要方法的辅助功能。

如果位图与数据空间扫描方法一起使用，位图初始做一个跳_顺序处理。数据空间扫描（和并行数据空间扫描）使用位图来‘跳过’非选择的记录，它有以下几个优点：

没有 CPU 处理非选择的记录，

减省 I/O，不把整个数据空间的内容放进内存。

下例说明查询优化选择位图处理方法与数据空间扫描一起使用的查询：

```
CREATE INDEX IX1 ON EMPLOYEE (WORKDEPT)
CREATE INDEX IX2 ON EMPLOYEE (SALARY)

DECLARE C1 CURSOR FOR
  SELECT * FROM EMPLOYEE
  WHERE WORKDEPT = 'E01' OR SALARY>50000
  OPTIMIZE FOR 99999 ROWS
```

在这个例子中，使用索引 IX1 和 IX2，数据库管理首先从选择对索引 IX1，选择 WORKDEP＝E01 的结果生成位图（用键字定位）。然后从对索引 IX2，选择 SALARY>5000 的结果产生一个位图（再用键字定位）。

接下来，数据库管理使用 OR 逻辑连接这两个位图，最后初始化数据空间扫描，数据空间扫描使用位图跳过数据空间记录，仅检索由位图选择的那些记录。

这个例子也说明位图操作的附加功能（可能用索引的 AND 选择，而现在位图处理允许用多个索引），在使用位图处理时，多个索引可用在选择中，这里 OR 是主要的布尔操作。

由 PRTSQLINF 命令生成的信息说明了这个查询。信息如下：

```
SQL4010  Arrival access for file 1.
SQL4032  Access path IX1 used for bitmap processing of file 1.
SQL4032  Access path IX2 used for bitmap processing of file 1.
```

如果用位图连接键字选择或键字定位选择，隐含地用位图做为主索引访问手段，下例解释用位图处理连接键字定位做主索引。

```
CREATE INDEX PIX ON EMPLOYEE (LASTNAME)
CREATE INDEX TIX1 ON EMPLOYEE (WORKDEPT)
CREATE INDEX TIX2 ON EMPLOYEE (SALARY)

DECLARE C1 CURSOR FOR
  SELECT * FROM EMPLOYEE
  WHERE WORKDEPT = 'E01' OR SALARY>50000
  ORDER BY LASTNAME
```

在这个例子中，索引 TIX1 和 TIX2 用位图处理。数据库管理能从适合选择 WORKDETP=‘E01’的索引 TIX1 的结果中产生位图，（用键字定位），然后对从适合选择 SALARY>50000

的索引 TIX2 的结果产生位图（再次使用键字定位）。

数据库管理使用 OR 组合这两个位图，使用索引 PIX 初始键选择方法，对索引 PIX 的每项，检验位图，若此项是位图选择的，那么选取及处理数据库记录。由 PRTSQLINF 命令生成的信息描述了这个查询，信息如下：

```
SQL4008  Access path PIX used for file 1.
SQL4032  Access path TIX1 used for bitmap processing of file 1.
```

22.3 数据访问方法总结

下列表对数据管理方法做一总结。

表 22-1 数据管理方法总结

访问方法	选择处理	什么时候用最好	什么时候用不好	什么时候选择	优点
数据空间扫描	读所有行，选择标准用于数据空间的数据	选择大于 20%的行	选择小于 20%	非排序、分组或连接且选择大于 20%的行	减少用予取处理的 I/O 页
并行予取	用并行流从辅存中检索数据，读所有行，选择标准用于数据空间的数据	选择 20%的行 1、有足够活动内存可用 2、查询不在 I/O 边界上 3、数据展开跨越多个磁盘单元	选择小于 20%的行，查询在 CPU 边界	非排序、分组或连接且选择大于 20%的行	减少用并行予取页 I/O 的等待时间
并行数据空间扫描	用并行任务选择和读数据	选择大于 10%的行及大表 1、有足够活动内存可用 2、数据展开跨越多个磁盘单元 3、安装了 DB2 对称多处理 4、多处理器系统	选择小于 10%的行，查询是在单处理器系统的 CPU 边界	1、安装了对称多处理 2、I/O 边界或在多处理器系统运行	在多处理系统中明显提高性能
键字选择	选择标准用于索引	排序、分组和连接	选择大量的行	要用索引，且不能用键字定位方法	仅访问匹配键字选择标准的行的数据空间
并行键字选择	选择标准用于并行任务的索引	索引的尺寸比数据空间少很多，安装了对称多处理系统	选择大量的行	不需要对结果排序	由于并行任务执行 I/O 能充分利用多处理系统，因此有更好的 I/O 重叠
键字定位	选择标准用于一定范围的索引项，一般使用选项	选择小于 20%的行	选择大于 20%的行	选择列适合最左键且选择小于 20%的行	仅选择匹配选择标准的索引和数据空间
并行键字定位	选择标准用于并行业务中一定范围内的索引项	选择小于 20%的行，必须安装 DB2 对称多处理	选择大量的行	1、不需要对结果排序 2、选择行匹配最左键且选择小于 20%的行	1、仅访问匹配选择标准行的索引和数据空间 2、用于并行任务完成 I/O，所以有更好的 I/O 重叠 3、能最有效的利用多处理系统
从索引产生索引	在永久索引上定位行，在选择的索引上建立临时索引	排序，分组和连接	选择大于 20%的行	不存在满足排序的索引，但存在满足选择的索引且选择小于 20%的行	仅访问满足选择标准行的索引和数据空间
分类例程	用数据空间扫描处理或键字定位来排序数据读取	选择大于 20%的行或多行的结果	选择小于 20%的行或少量行结果	规定排序，没有满足排序的索引存在或期望得到大量结果	看这个表中的‘数据空间扫描和键定位’
仅索引	用与任何其它索引访问方法联用	用在查询中的所有行都有键字段，必须安装 DB2 对称多系统	选择小于 20%的行，或少量行结果	用在查询中的所有行都有键字段，必须安装 DB2 对称多系统	减少对数据空间的 I/O

并行表/索引予取	用并行装入表或索引，避免随机访问	发生过量的目标随机活动，有活动的内存用来装整个目标	活动内存已经超_落实	从处理查询引起过量随机活动且有活动内存用来装整个目标	避免由I/O边界查询引起的随机页I/O
并行或非并行散列方式	有共同值的行组合在一起	长时间运行的分组和/或连接查询	短时间运行的查询	规定连接和分组	与索引方式比较，减少了随机I/O，安装DB2对称多处理可利用SMP并行特性
位图处理	用键字定位/键字选择来建立位图，用位图可避免接触表中的所有行	选择适用于索引且或者选择大于 5%或者小于 25%的行，或者选择中有 OR 操作来产生仅用一个索引	选择大于 25%的行	索引适合选择标准	减少对数据空间的页 I/O，允许每个表用多个索引

22.4 优化

优化是AS/400 DB2很重要的部分。这是因为：

用键来决定影响数据库的性能。

鉴定用做查询的技术

选择最有效技术

象SELECT语句这样的数据操作语句，只是指定用户想得到的数据，而不指明取得数据的方法。访问数据的路径是由优化选择并存在访问方案中。这节介绍查询优化为完成这些任务而使用的技术，包括：

成本估算

访问方案验证

连接优化

分组优化

22.4.1 成本估算

在运行时，优化基于数据库的当前状态，计算完成费用来为查询选择一个最佳的访问方法，优化模拟下列操作的访问成本：

从表中直接读取行（数据空间扫描处理）

通过访问路径读取行（使用键字选择或键字定位）

直接从数据空间生成访问路径

从已有的访问路径建立访问路径（从索引产生索引）

使用查询分类例程或散列方法（如果条件许可）

典型的计算成本的方法是以下内容的总和：

启动成本

成本与给定的优化模式有关。预编译选项ALWCPYDTA和OPTIMIZE FOR n ROWs子句指出，查询优化所要达到的优化目标，优化使SQL查询达到下面二个之中的一个目标：

1.减少从表中行的第一个缓冲区获得记录所需的时间,这个目标偏向于优化不建立索引。

数据扫描或已存在索引都是比较适合的。这个模式可以用两种方式规定：

a.OPTIMIZE FOR n ROWs允许用户指定他们想从查询中得到的行的数目。

优化用这个值来决定返回行的比率和可到的优化。一个小值指出优化将减少获得第一个n行需要的时间。

b.指定ALWCPYDTA(*NONE)或ALWCPYDTA(*YES)预编译选项，允许优化减少获

得结果行的第一个 3%所用的时间。

只在没规定 OPTIMIZE FOR n ROWs 时，这个选项才有效。

2.减少把所有选择行都返回给应用程序的整个查询处理时间，不要把优化偏向任何特殊的访问方法，这个方式可以用下两种方法规定：

a.OPTIMIZE FOR n ROWs 允许用户指定他们想从查询中得到的行的数目。优化用这个值来决定返回行的比率和可达到的优化，大于或等于预期结果行数的值指出优化减少运行整个查询所需的时间。

b.规定 ALWCPYDTA(*OPTIMIZE)预编译选项。

只在没规定 OPTIMIZE FOR n ROWs 时，这个选择项才有效。

任何访问路径产生的成本。

预计读行的页故障数量和处理预计行数的成本。

页故障和行处理数目可以通过统计优化来预测从数据库中得到的目标，包括：

—表尺寸

—行尺寸

—索引尺寸

—键字尺寸

如果只是完成索引的访问，页故障也要受很大的影响，因此剔除对数据空间的随机 I/O。

预定处理的行数的范围要基于行选择谓词中的关系操作符，缺省的过滤因子，类似于取得：

— 等于 10%

— 小于、大于、小于等于或大于等于 33%

— 不等于 90%

— BETWEEN 范围的 25%

— 每个 IN 列表值的 10%

键字范围估算是优化使用的一种方法，它用来从一个或多个选择谓词选择的预计行数更准确的估算。优化通过提供选择谓词而不用已存索引的最左键进行估算。通过基于键字范围的估算，能进一步优化缺省的过滤因子。若一个索引存在，它的最左键符合用在行选择谓词中的列，这个索引就可以用来估算符合选择标准的键的数目。估算键的数目是基于页的数目和机器索引的键密度且不用实际访问这些键就可以做。用在选择谓词中的全索引覆盖列都能明显的帮助优化。

页故障和处理的行数是依据优化选择的访问类型，详细内容请看 22.1。

22.4.2 访问方案验证

一个访问方案是一个控制结构，它描述满足各个查询必要的动作。一个访问方案包括关于数据和如何取出它的信息，对每个查询，无论何时发生优化，都要有一个如何访问所需数据的优化方案。信息被保存在一个小方案里，小方案与查询定义样板（QDT）用在优化接口中以产生一个访问方案。对动态 SQL，建立一个访问方案，但不存储它，每次运行 PREPARE 语句时，生成一个新的访问方案。对 AS/400 DB2 程序，访问方案保存在与程序有关的空间中或有嵌入 SQL 语句的程序包中。

22.4.3 优化决策规则

为完成优化功能，优化使用一组规则来选择访问数据的方法。优化要做以下事情：

在选择子句中对每个谓词决定缺省过滤因子。

从内部存储的信息中取出表的属性。

在选择谓词符合一个索引的最左键时，执行一个估算键范围来决定谓词的真实过滤因子。如果需要索引，决定建立一个表的索引的成本。

若提供选择条件且需要索引，决定使用分类例程或散列方法的成本。

若不需要索引，决定数据空间扫描处理的成本。

对每个可用索引，从最近生成的到最早生成的排序，优化做下列事情直至超出时间限制：

— 从内部存储状态中取出索引属性
— 确定索引是否适合选择标准
— 确定使用估算页缺省的索引成本及用过滤因子来帮助确定成本。
— 挑出最省的一个，用这个索引成本与以前成本（当前最好的一个）做比较。
— 继续检索最佳索引直至超时或再没有索引

时间限制因素控制选择一个实施办法所用的时间，它是基于到目前为止花费了多少时间和当前找到的最佳成本，动态 SQL 查询优化时间有限制，而静态 SQL 查询优化时间没限制。

对小的表，在查询优化中花费很少时间，对大的表，查询优化要考虑更多的索引。一般来讲，在超出运行时间以前，优化要考虑需 5 至 6 个索引（对每个连接的表）。

22.4.4 连接优化

连接操作是一个复杂的功能，它需要特别注意改善性能。这节介绍 AS/400 DB2 实施连接查询及查询优化怎样选择优化方式，也介绍帮助避免和解决性能问题所用的设计要点和技术。

22.4.4.1 嵌套循环连接

AS/400 DB2 提供了一个嵌套的循环连接方式。对于这种方式，表在连接时的处理是有序的，这种顺序称为连接次序。在最终连接次序中的第一个表称为主表，其它的表则称为辅表，每一个连接表定位称为拨号。连接期间，AS/400 DB2 做以下事情：

1.访问第一个主表中由谓词定位主表而选择的行。

2.从主表的连接列建立一个键值。

3.使用键定位来确定第一个行的位置，此行要使用满足连接条件的键字访问路径来适应第一个辅表的连接条件，或定位辅表的行选择列。

4.如果可能，使用位图选择。

5.确定行是否是通过对第一个辅表拨号用剩余选择分配而被选择的。

如果没选择辅助拨号的行，那么寻找满足连接条件的下一行，重复第一步至第五步，直至从辅表中选出满足连接条件和剩余选择的行。

6.返回结果连接行。

7.再次处理最后一个辅表，以找出满足该拨号中连接条件的下一个行。

在处理期间，当不再有满足连接条件的行时，则返回到前一个逻辑拨号，同时试图读出满足连接条件的下一行。

8.主表中全部选择的行都被处理时，结束过程。

要注意嵌套循环的以下特点：

- 如果对一个表规定 ORDER BY 或 GROUP BY 来处理连接结果，则这个表成为主表，并且用键字访问路径来处理它。
- 如果不是在主表拨号或在来自多个拨号的列上指定结果行的排序或分组，AS/400 DB2 将查询处理分为两部分：
    1. 处理省略排序或分组的连接查询且把结果行写往临时工作表，这就让优化把连接查询中的任何表都可做为主表的候选者来考虑。

2.然后再用临时工作表的数据处理排序或分组。

在规定了 ALWCPYDTA(*OPTIMIZE)预编译参数时，查询优化也可决定将查询过程分为这两部分以提高性能。

所有从每个辅助拨号中得出的满足连接条件的行都用键字访问路径定位，从辅表中每一次随机顺序取行，这个随机的磁盘 I/O 时间占查询处理时间的很大百分比。既然对从主表和满足连接条件的先前辅助拨号都要检索一个给定的辅助拨号，后部的拨号要做大量的检索。对后部拨号处理的低效率明显表现在较长的查询处理时间，这就是为什么对于连接查询来说，注意性能方面的考虑能够减少连接查询运行时间的原因。

另一方面，所有从辅助拨号中选择的行都用键字访问路径存取。如果找不到高效的键字访问路径，则创建一个临时键字访问路径。一些连接查询在辅助拨号上建立临时访问路径，甚至在对所有连接键字都存在访问路径时也这样做。因为对于长时间运行查询的辅助拨号来说，效率是很重要的，故查询优化可选择建立临时访问路径，它仅包括传送给拨号的本地行选择键字。这个行选择的预处理，允许数据库管理用一个传送来处理行选择来代替每一次行都与拨号进行匹配处理。

22.4.4.2 散列连接

散列连接方式与嵌套循环连接是相似的，它不用键字访问路径去分配辅助表中的匹配行，而是生成散列临时结果表来包括所有由局部选择从表中选出的行。散列表的结构是：有相同连接值的行装入到同一个散列表分区，对任何给定连接值的行的定位都能由对连接值提供的一个散列函数来找到。

散列连接优于嵌套循环连接有以下几点：

散列临时结果表的结构要比索引简单，所以建立和处理散列表用很少的 CPU。

在散列结果表中的行包括查询需要的所有数据，所以在处理散列表时不需要用随机 I/O 来访问表中的数据空间。

既然连接值是群集的，所以对所给连接值相匹配的全部行通常都用单一 I/O 访问。

散列临进结果表能用 SMP 并行来建立。

不象索引，散列表中的项不能被更新来反映基本表中列值的改变，散列表的存在不影响系统中其它更新作业的处理成本。

散列连接不能用于下列查询：

执行子查询

执行 UNION 或 UNION ALL

执行左外部连接或例外连接

用 DDS 生成的连接逻辑文件

对预编译参数 ALWCPYDTA 规定*NO 或*YES 值的数据访问要求活动访问。散列连接只用于用 ALWCPYDTA(*OPTIMIZE)的查询运行。这个参数能够在预编译命令、STRSQL 命令或 OPNQRYF 命令中规定。Client Acless/400 ODBC 的驱动程序和查询管理的驱动程序总是使用这种方式。

如果用临进结果运行查询，则散列连接能用 OPTIMIZE(*YES)。

做为 SQL ROLLBACK HOLD 语句或 ROLLBACK 命令的结果要求重存游标位置。对于使用了非*NONE 的落实控制级的 SQL 应用程序，要求对 ALWBLK 预编译参数规定*ALLREAD 的值。

查询属性 DEGREE，可用 CHGQRYA 修改，它能使优化选择可以或不可以使用散列连接，

但是，如果查询属性 DEGREE 设为*OPTIMIZE 或*NBRTASK 或*MAX，散列连接查询能够使用 SMP 并行操作。

散列连接用在许多建立了临时索引的情况，用散列连接实现的连接查询有如下几点之一时会是相当类似：

连接的不同表的所有行包含在产生的结果行中。

为连接表规定明显的非连接选择，它减少了包含在连接结果表中的行数。

下面是连接查询的例子，它处理查询表中的所有行：

```
SELECT *
  FROM EMPLOYEE, EMP_ACT
 WHERE EMPLOYEE.EMPNO = EMP_ACT.EMPNO
OPTIMIZE FOR 99999999 ROWS
```

这个查询用下面几步完成：

1.临时散列表有在表 EMP_ACT 上用 EMPNO 键，在查询打开时发生。

2.对从 EMPLOYEE 表中取回的每一行，该临时散列表将检索任何匹配的连接值。

3.对每个找到的匹配行，返回结果行。

由 PRTSQLINF 命令生成的信息，描述了 SQL 程序中散列连接查询，显示以下内容：

```
SQL402A  Hashing algorithm used to process join.
SQL402B  File EMPLOYEE used in hash join step 1.
SQL402B  File EMP_ACT used in hash join step 2.
```

下面是连接查询的例子，该连接查询能由局部选择明显减少连接查询的表：

```
SELECT EMPNO, LASTNAME, DEPTNAME
  FROM EMPLOYEE, DEPARTMENT
 WHERE EMPLOYEE.WORKDEPT = DEPARTMENT.DEPTNO
   AND EMPLOYEE.HIREDATE BETWEEN 1996-01-30 AND 1995-01-30
   AND DEPARTMENT.DEPTNO IN ('A00', 'D01', 'D11', 'D21', 'E11')
OPTIMIZE FOR 99999999 ROWS
```

这个查询用下面几步完成：

1.临时散列表建在表 DEPARTMENTE 上，它使用包括与选择谓词匹配行的 DEPTNO 键值，DEPTNO IN(‘A00’，‘D01’，‘D11’，‘D21’，‘E11’) 查询打开时发生。

2.对每一个从表 EMPLOYEE 中取回与选择谓词匹配的行，临时散列表检索匹配的连接值。

3.对每个找到的匹配行，返回结果行。

由 PRTSQLINF 命令生成的信息描述了 SQL 程序中的散列连接查询，信息显示如下：

```
SQL402A  Hashing algorithm used to process join.
SQL402B  File EMPLOYEE used in hash join step 1.
SQL402B  File DEPARTMENT used in hash join step 2.
```

当排序，分组，不等选择用从不同表中的导出的操作数规定或结果列是从不同表的列中

导出时，将做散列连接处理，并且将连接的结果行写到一个临时表中，然后，象第二步那样，查询用临时表来完成。

下面是一个连接查询的例子，用从不同表的列中导出的操作数规定选择：

```
SELECT EMPNO, LASTNAME, DEPTNAME
  FROM EMPLOYEE, DEPARTMENT
 WHERE EMPLOYEE.WORKDEPT = DEPARTMENT.DEPTNO
       AND EMPLOYEE.EMPNO > DEPARTMENT.MGRNO
OPTIMIZE FOR 99999999 ROWS
```

这个查询的实现步骤是：

1.临时散列表用 DEPTNO 键字在表 DEPARTMENT 上建立，这发生在查询打开时。

2.对于每一个从 EMPLOYEE 表中取回的行，临时散列表将检索相匹配的连接值。

3.对每个找到匹配行，结果行写入临时表中。

4.在所有的连接结果行写入临时表后，从临时文件中读出由 EMPNO>MGRNO 选择的行，并且返给应用程序。

由 PRTSQLINF 命令生成的信息描述了 SQL 程序中散列连接查询，信息显示如下：

```
SQL402A  Hashing algorithm used to process join.
SQL402B  File EMPLOYEE used in hash join step 1.
SQL402B  File DEPARTMENT used in hash join step 2.
SQL402C  Temporary result table created for hash join query.
```

22.4.4.3 连接优化算法

查询优化程序必须为连接查询确定连接列、连接操作符、局部行选择、键字访问路径使用和拨号次序。

连接列和连接操作符依靠于：

查询的连接列规定

连接次序

连接列与其它行选择的相互作用

使用键字访问路径

未实现拨号的连接规定可延迟至它们能在下一个拨号中作处理或者，如果这个拨号正完成内部连接，则做为行选择处理。

对于给定的拨号，只有作为拨号连接列可用的连接规定，才被连接在先前的拨号上。例如，对于第二个拨号，只有能被用来满足连接条件的连接规则约定是主拨号中引用列的连接规定。同样，第三个拨号也只能使用主拨号和第二个拨号中引用列的连接规定等等。引用最后拨号的连接规定被延迟至处理被引用的拨号。

对于任何给定的拨号，只有一种连接操作符类型正常实施，例如，一个内部连接的连接规定有一个“＝”的连接操作符，且另外一个有连接操作符“>”，优化试图实现与“＝”操作符的连接。“>”连接规定要在找到与“＝”匹配的行之后，象行选择那样处理。另外，使用相同操作符的多个连接规定是一起实现的。

注：左外部连接或例外连接只允许一种连接操作符。

当查找已有的键字访问路径去访问辅助拨号时，查询优化查找访问路径的最左键列。对于给定的拨号和键字访问路径，可以使用最左键列的连接规定。例如：

```
DECLARE BROWSE2 CURSOR FOR
 SELECT * FROM EMPLOYEE, EMP_ACT
  WHERE EMPLOYEE.EMPNO = EMP_ACT.EMPNO
    AND EMPLOYEE.HIREDATE = EMP_ACT.EMSTDATE
 OPTIMIZE FOR 99999 ROWS
```

对于有 EMPNO，PROJNO 和 EMSTDATE 键列的 EMP_ACT 的键字访问路径，只在 EMPNO 列上实现连接操作。连接结束后，用列 EMSTDATE 做行选择。

当对辅助拨号选择最好的键字访问路径时，查询优化也使用局部行选择。上一个例子用局部谓词表示则为：

```
DECLARE BROWSE2 CURSOR FOR
 SELECT * FROM EMPLOYEE, EMP_ACT
  WHERE EMPLOYEE.EMPNO = EMP_ACT.EMPNO
    AND EMPLOYEE.HIREDATE = EMP_ACT.EMSTDATE
    AND EMP_ACT.PROJNO = '123456'
 OPTIMIZE FOR 99999 ROWS
```

用键列 EMPNO，PROJNO 和 EMSTDATE 的键字访问路径由把连接和选择组合成一个操作来完成，而不是对所有三列操作，这样能充分利用这个访问路径。

在生成临时键字访问路径时，最左键列是在拨号定位中可用的连接列。对于这个拨号的所有局部行选择是在选择键包含到临时键字访问路径时被处理的。临时键字访问路径类似于建立在选择/省略键字的逻辑文件上的访问路径。上个例子的临时索引有 EMPNO 和 EMSTDATE 键字段。

在上述例子中，在实现时，或者使用已有索引或者创建临时索引。使用已有的索引可提供快速的性能，这是因为合并连接和选择不象建一个临时索引花费那么多。

在已有可用索引时把使用相等选择做为最左键的列和连接列合并一起做键列是个好主意。

22.4.4.4 连接次序优化

如果引用任何逻辑文件或用左外部连接或例外连接来执行连接的任一拨号时，连接次序是固定的。否则，用下面的连接次序规则来确定表的顺序：

1.对每一个独立的表确定一个访问方式做为主拨号的候选者。

2. 基于局部行选择估算每个表返回的行数。

如果用行次序和分组来处理连接查询用一步完成，那么有次序或分组列的表做为主表。

3.对每个作为主表和第一个辅表的候选表的连接组合，确定访问方式、成本和期望返回的行数。

四个表连接次序组合估算将是：

1-2 2-1 1-3 3-1 1-4 4-1 2-3 3-2 2-4 4-2 3-4 4-3

4.选择最低连接成本的组合，如果成本大致相同，选择所选行数最少的组合。

5.对每个剩余表与前一个辅表的连接，确定成本、访问方式和期望的行数。

6.对每个表选择最低成本的访问方式。

7.选择有最低连接成本的辅表。如果成本大致相同，选择所选行数最少的组合。

8.重复第四步到第七步，直至确定最低成本的连接次序。

当引用连接逻辑文件或用左外连接或例外部连接来执行连接拨号时，查询优化按照规定的次序，将循环通过全部拨号，并且确定最低成本的访问方式。

注：如用 JOIN 语法来执行一个内部连接，则在 FROM 子句中规定的表示表次序的连接组合成本总数将减少，这给了用户一个用来影响由优化选择的最终连接次序的方法。

22.4.4.5 连接辅助拨号的成本及访问路径

在 22.4.4.4 的第 3 步和 5 步中，查询优化为已给定拨号组合估算了成本和选定了一个访问方式，该选择与那些行选择是相似的，除了必须使用键字访问路径外。

在查询优化比较各种可能的访问选择时，它必须对每个候选者分配一个成本数，并用这个值去确定耗费运行时间最少的方法，这个成本值是 CPU 和 I/O 时间的组合，它是基于下面的假设条件：

必须从辅存中取回表页和键字访问路径页。例如，查询优化没有意识到整个表做为 SETOBJACC 命令的结果装入到了活动内存，用这个命令可以有效地提高查询性能，而查询优化不用更改查询方法就能获得表的内存驻留状态的好处。

查询只是系统的一个运行过程。不允许给系统 CPU 利用或因其它进程使用同资源而发生 I/O 等待，CPU 相关成本依据系统查询的相对运行速度可伸缩。

列的值跨过表均匀分布。例如，如果表中 10%的行有相同值，那么则可假定表中每十行都包含这个值。

一个列的值与行中其它列的值是独立的。例如，表中有一名为 A 的列，50%的行的这列值为 1，同时有一名为 B 的列，50%行的这列值为 2，那么期望查询选择 A＝1，B＝2 的行占表中行数的 25%。

对于辅助拨号连接计算的主要因素是前一拨号中选择的行数和从前一拨号中选择的每行的平均匹配行数。这两个因素都能由估算与所给的拨号相匹配的行数来得到。

当连接操作符不是等号时，匹配行的期望数根据下面的缺省过滤因子确定：

33%为小于，大于，小于等于或大于等于。

90%为不等于

25%为 BETWEEN 范围

10%为每个 IN 列表值

例如，当连接操作符为小于时，相匹配行的期望数为 0.33×（拨号中的行数），如果对当前拨号没有规定连接规则，则假定笛卡尔积为操作符。对于笛卡尔积，除非局部行选择用到键字访问路径上，则匹配的行数目为拨号中每一个行。

当连接操作符是等号时，行的期望数是所给定值双倍行的平均数。

AS/400 执行索引维护（插入或删除索引中的键值）且管理索引中给定键列的唯一值的运行计数，这些统计数与索引目标放在一起并长期保留。当执行查询优化时，它则使用这些统计信息，保留这些统计数据不增加，索引维护总开销，这些统计信息只对下列索引有用：

包括非变长字符键。

注：如果你用变长字符列做连接列，可用 CRTLF 命令生成一个索引，用它把变长字符列转换为定长字符键，有定长字符键定义的索引覆盖变长数据，能提供重复值统计的平均数。

是在安装 V2.3 或以后版本的 AS/400 系统上生成或重建的。

注：如果连接键值有一个高或低的重复平均数时，查询优化能使用在更早版本的 OS/400 系统中生成的索引。如果仅用一个连接键字定义索引，则根据索引大小估算。在多数情况下，索引中的附加键字通过索引估算匹配行是无效的。某些连接查询性能可由于重建这些访问路径而改善。

重复值统计的平均数只保留索引最左的 4 个键，对于规定多于 4 个连接列的查询，建立多个附加索引是有益的，这样能在四个最左键列中找到重复值统计平均数的索引，如果某些连接列中有些许唯一值，这是很重要的。

---

3 列键

键 1	键 2	键 3

键 1 的唯一键数

键 1 和键 2 联合的唯一键数

键 1 键 2 键 3 联合的唯一键数

图 22-2　三个键索引重复值的平均数

这些统计值作为重建或生成的索引的一部分被保留。

对相等连接或其它连接符号的缺省过滤值，使用重复数据的平均数，我们现在就有匹配行的数目。下面的公式用来计算从前一拨号中得到的连接行的数目：

NPREV = Rp * M2 * FF2 * ..... *Mn * FFn .....

说明：　NPREV：所有从前拨号得到的连接行数。

RP：从主拨号中选择的行数。

M2：与第 2 个拨号匹配的行数。

FF2：对拨号 2 的谓词局部过滤减少因子，拨号 2 已不能适用上述 M2。

Mn：与拨号 n 匹配的行数。

FFn：对拨号 n 的谓词局部过滤减少因子，拨号 n 已不能适用上述 Mn。

注：对当前拨号前的每个辅助拨号，把一对匹配行（Mn）和过滤减少因子相

乘。

现在已从前一拨号计算出了连接的行数，优化已准备好生成访问方式的成本花费。

22.4.4.6 临时键字访问路径或散列临时结果表

由查询优化分析的第一个访问路径选择是建立在一个临时键字访问路径或者从表中得到的散列临时结果表上的。通过建立在表或散列表上的临时键字访问路径得到的连接辅助拨号的基本计算公式如下：

```
JSCOST  =  CRTDSI +
           NPREV *((MATCH * FF * KeyAccess)
           + (MATCH * FF * FCost)) *
           FirstIO
```

说明： JSCOST：连接辅助成本

CRTDSI：建立临时键字访问路径或散列临时结果表的成本

NPREV：所有前拨号的连接行数

MATCH：匹配的行数（通常是平均重复值）

Key Access：访问在键字访问路径或散列表中的一个键字的成本

FF：这个拨号的局部谓词的过滤因子

Fcost：从表中访问一行的成本

First IO：由于优化策略来优化第一个缓冲检索，用来降低非启动成本的降低

比率。详细内容请看 22.4.1。

如果没找到可用的键字访问路径或者临时键字访问路径或散列表比现存的键字访问路径性能好,则使用这种辅助拨号的访问方式。这种方式比使用任何已有的键字访问路径都好，这是因为在满足下列条件时，在生成键字访问路径或散列表时，已经完成了行选择：

匹配数（MATCH）高；

所有前拨号的连接行数（NPREV）高；

有某些过滤减少（FF<100%）

22.4.4.7 临时键字访问路径或键字访问路径的散列表

这个访问方式选择的基本成本公式同使用临时键字访问路径或由表建的散列表的公式一样，只有一个例外，建立临时键字访问路径的成本 CRTDSI 计算要包括用已有的键字访问路径的行的选择，这个访问方式用来连接辅助拨号存取上，然而，从键字访问路径来生成可能少些花费。

22.4.4.8 使用已有的键字访问路径

最后的访问方式是用一个已有的键字访问路径。用已有的键字访问路径来访问一个连接辅助拨号成本的公式如下：

```
JSCOST  =  NPREV *((MATCH * KeyAccess)
          + (MATCH * FCost)) *
           FirstIO
```

说明：　JCOST：连接辅助成本

NPREV：所有前拨号的连接行数

MATCH：用这个键字访问路径找到的匹配键的数目

KeyAccess：用键字访问路径访问一个键的成本

Fcost：从表中访问一行的成本

FirstIO：由于优化策略来优化第一个缓冲检索，用来降低非启动成本的降低比率。详细内容请看 22.4.1。

如果首先使用 I/O 优化，它就非常类似访问方法，因此整个成本减少。如果从全部前拨号的连接的行数（NPREV）和匹配键数（MATCH）都很低，这可能是最有效的方式。

查询优化在以下情况时考虑使用一个索引，该索引只有连接列的子集做为最左引导键：

它能从重复值统计平均数中确定有重复值行的平均数是相当低的

从前拨号选择的行数是很小的

22.4.4.9　用过渡关闭生成的谓词

对连接查询，查询优化可能做一些特殊处理以产生附加选择，当逻辑上属于查询的一组谓词推断出额外谓词时，查询优化则产生附加谓词，其目的是为了在连接优化期间提供更多的信息。

22.4.4.10　由过渡关闭附加的谓词例子

```
SELECT * FROM EMPLOYEE, EMP_ACT
  WHERE EMPLOYEE.EMPNO = EMP_ACT.EMPNO
    AND EMPLOYEE.EMPNO = '000010'
```

优化把查询修改为：

```
SELECT * FROM EMPLOYEE, EMP_ACT
  WHERE EMPLOYEE.EMPNO = EMP_ACT.EMPNO
    AND EMPLOYEE.EMPNO = '000010'
    AND EMP_ACT.EMPNO  = '000010'
```

下列规则确定哪一个谓词加到其它连接拨号上：

拨号偏差必须有等号连接操作符

谓词是独立的，这意味着这个谓词的假条件将省去行

谓词的一个操作是等号连接列，其它的是文字或主变量

谓词操作符不能是 LIKE 或 IN

谓词间不能用 OR 连接

拨号的连接类型是内部连接

查询优化产生一个新的谓词，不管这个谓词是否在 WHERE 子句中存在。

有些谓词是冗余的。当查询中其它谓词的前面评价已经确定了谓词提供的结果时，出现这种冗余。冗余谓词能由你规定或在谓词操作期间由查询优化产生。有＝，>，>=。<，<=或 BETWEEN 谓词操作符的冗余谓词被合并成一个谓词以反映最可选择的范围。

22.4.4.11　一个查询的多个连接类型

尽管多种连接类型（内部，左外部连接和例外连接）能在用 JOIN 语句的查询中，AS/400 特许内码对于整个查询只能支持一种连接类型，这就要求优化来确定查询的整个连接类型是什么。

优化程序将估算连接准则和可能规定的记录选择，来确定每个拨号及整个查询的连接类型。一旦知道这些信息，优化将用表的相关记录号来生成附加选择，模拟在查询中可能出现的不同连接类型。

既然用左外部连接或例外连接不匹配的行要返回一个空值，任何对拨号规定的独立选择，（除非是 IS NULL 谓词选择），包括任何在 WHERE 子句中指定的附加连接都将引起消除所有不匹配记录。在规定 IS NULL 谓词时，将引起拨号的连接类型改成内部连接。

在下面的例子中，在 EMPLOYEE 和 DEPARTMENT 表间规定一个左外部连接，在 WHERE 子句中，有两个选择谓词，也可用在 DEPARTMENT 表上：

```
SELECT EMPNO, LASTNAME, DEPTNAME, PROJNO
 FROM CORPDATA.EMPLOYEE XXX LEFT OUTER JOIN CORPDATA.DEPARTMENT YYY
         ON XXX.WORKDEPT = YYY.DEPTNO
       LEFT OUTER JOIN CORPDATA.PROJECT ZZZ
         ON XXX.EMPNO = ZZZ.RESPEMP
 WHERE XXX.EMPNO = YYY.MGRNO AND
         YYY.DEPTNO IN ('A00', 'D01', 'D11', 'D21', 'E11')
```

第一个选择谓词，XXX.EMPNO ＝ YYY.MGRNO 是一个附加连接条件，它要加到连接准则上并作为‘内部连接’的连接条件计算。第二个是一个独立的选择谓词，它将排除任何不匹配的记录，这两个选择谓词中的任一个将使 DEPARTMENT 表的连接类型由左外部连接变成内部连接。

尽管表 EMPLOYEE 和 DEPARTMENT 间的连接改为了内部连接，整个查询仍然要保持左外部连接以满足 PROJECT 表的连接条件。

注：多连接类型是由不匹配行查询的附加选择支持，故规定它时要很小心。这意味着满足连接准则的结果行的数在选择之前能变得相当大，这种选择或省略不匹配的行是根据各自的拨号连接类型确定的。

详细内容请看 5.4 或 AS/400 DB2 SQL 参考。

22.4.4.12　连接查询性能问题的原因

上面介绍的优化算法使大多数的连接查询受益，但也有小部分查询的性能可能降低。这发生在以下情况时：

为潜在连接列提供了重复值统计平均数的访问路径不可用。

注：请看 22.4.4.5 的表，它提供了如何避免有关索引统计的约束或在索引不存在时，在潜在连接列上创建附加索引的一些建议。

因为选择列上不存在索引，当对表用局部选择时，查询优化用缺省过滤因子来估算选择的行数。

在选择列上生成的索引允许查询优化用键范围估算得出比较准确的过滤估计。

为连接列选择的特殊值产生出的相匹配的行数比表中连接列全部值的重复值的平均数大得多（也即数据不是均匀分布的）。

使用 DDS 建立一个有选择/省略规定的键字访问路径的逻辑文件匹配局部行选择，这将提供给查询优化一个更准确的估算与被选择键相匹配的行数。

注：优化能从数据没有均匀分布的选择/省略访问路径中更进一步确定。

查询优化错误的估计了从回答中取回的行数。

对于 SQL 程序，规定了预编译选项 ALWCPDTA(*YES)使它更像使用已有索引的查询程序，同样，规定了 ALWCPYDTA(*OPTIMIZE)使它更象创建临时索引的查询程序。SQL 子句 OPTIMIZE FOR n ROWS 也能影响查询优化。

22.4.5 分组优化

这部分介绍 AS/400 DB2 实施分组的技术及如何选择查询优化。

22.4.5.1 分组散列方法

这项技术使用基础散列访问方法对选择表行分组和汇总。对每个选择的行，规定的分组值通过散列功能运行。计算的散列值和分组值用来快速查找分组值相应的散列表项，如果当前分组值已经有一行在散列表中，则取出散列表项，且根据请求的分组列操作（例如 SUM 或 COUNT）与当前表行汇总（更新），如果没找到当前分组值的散列表项，新的项插入到散列表中并且用当前分组值初始化。

接收第一组结果所用的时间要比接收其它组的执行时间长，这是因为访问必须先建立并普及散列表，一旦散列表完全普及，数据库管理就使用这个表开始返回分组结果。在返回结果之前，数据库管理必须提供分组选择标准或在散列表中汇总项的次序。

在合并系数高时，分组散列方式是最有效率的。合并系数是选择表行与计算分组结果的比率。如果每一个数据库表行都有自由唯一的分组值，则散列表会太大，这将降低散列访问方式的速度。

由第一次确定的规定分组列唯一值数，优化可估算合并系数，然后，优化会检查表的总行数和指定的选择标准，同时使用检查结果去估算合并系数。

在分组列上建立索引有助于优化更准确的估计系数，索引能提高准确率是因为它的统计值中包括键列的重复值的平均数。详细内容请看 22.4.4.5。

优化也用组估算的期望数来计算散列表中的分区数。当散列表非常平衡时，散列表访问方式更加有效，散列表分区数直接影响散列表项在表中如何分布及其分布的均匀程度。

当分组值由非数值类型的列与除整型（二进制）外的数据类型组成时，散列功能执行地比较好。另外规定不与变长和空列属性有关的分组值列允许散列函数更高效地执行。

22.4.5.2 键字顺序分组方法

这个方法用键选择或键字访问方式去实行分组，需要一个包含所有分组列的索引连接最左键字段。数据库管理用键字访问路径访问每个组并且完成需要的汇总功能。

按定义，既然索引已把所有的键值组合在一起，第一组结果能在比散列方式短的时间内返回，这是因为散列方式要求临时结果表。如果应用程序不必取回所有的组结果或者如果索引已经存在相匹配的组列，则该方法是有益的。

当分组通过索引实现，同时没有永久索引满足分组列时，生成临时索引，在查询中规定的分组列作为这个索引的键字段。

22.4.5.3 排除分组列

要计算所有的分组列来确定它们是否能从分组列表中取消，仅考虑那些有独立选择谓词的等号操作符的分组列，这保证了列只能匹配单一值，且不能帮助确定唯一组。这个处理允

许优化考虑更多的索引去实现查询，并且它减少作为临时索引或散列表的键字段而增加的列数。

下面的例子说明一个查询程序，优化能排除一个分组列：

```
DECLARE  DEPTEMP  CURSOR
  SELECT EMPNO,LASTNAME,WORKDEPT
    FROM CORPDATA.EMPLOYEE
    WHERE EMPNO = '000190 '
    GROUP BY EMPNO, LASTNAME, WORKDEPT
```

在这个例子中，优化从分组字段表中取消 EMPNO，这是因为有 EMPNO ＝ ‘000190’选择谓词，只有规定 LASTNAME 和 WORKDEPT 做键字段的索引才被考虑去执行查询，并且，如果要求临时索引或散列，则不用 EMPNO。

注：尽管 EMPNO 也能从分组列表中取消，但如果永久索引存在全部 3 个分组列，则优化仍可选择使用该索引。

22.4.5.4 增加附加的分组列

排除分组列的同一逻辑也能被用来增加查询的附加分组列，当我们尽力去确定是否能用索引来实现分组时就使用该逻辑。

下面的例子说明优化增加附加分组列查询：

```
CREATE INDEX X1 ON EMPLOYEE
       (LASTNAME, EMPNO, WORKDEPT)

DECLARE DEPTEMP CURSOR FOR
  SELECT LASTNAME, WORKDEPT
    FROM CORPDATA.EMPLOYEE
    WHERE EMPNO = '000190'
    GROUP BY LASTNAME, WORKDEPT
```

对于这个查询请求，当对查询考虑 X1 时，优化可增加 EMPNO 作为附加分组列。

22.5 改善连接查询的性能

观注一个性能不好的连接查询或者要生成使用连接查询的新应用程序时，下面的检查表可能会很有用处。

表 22－2　　生成使用连接查询的应用程序的检查表

要做什么	这样做的帮助
检查数据库设计，确保对所有连接列与/或行选择列有可用的索引。如用 CRTLF，确保索引是非共享的。	这给查询优化－个较好时机去选择高效的访问方式，这是因为它算出重复值的平均数。许多查询可以使用现有索引来实现查询，同时避免生成临时索引的成本。
检查查询了解一些复杂谓词是否加到其它拨号上去，来允许优化得到更好的选择每个拨号的方法。	既然查询优化不能增加谓词给那些由 OR 或连接非独立谓词连接的谓词，或者 LIKE 或 IN 的谓词操作符，那么增加谓词可能有助于修改查询。
创建键字访问路径，它要包括与使	这有助于看统计特性对整个表是否一致。例如，如果有

用 CRTLF 命令的查询相匹配的选择/省略规则。	一个高重复因子的值并且剩余的列值是唯一的，则选择/省略键字访问路径允许优化去改变键值分布并对选择值实行最优化。
指定 ALWCPYDTA(*OPTIMIZE) 或 ALWCPYDTA(*YES)	如果查询正创建临时键字访问路径，并且，感到如果优化程序使用现存的路径处理时间较好的话，则指定 ALWCPYDTA(*YES)。 如果查询未创建临时键字访问路径，同时，感到如果创建临时键字访问路径处理时间较好的话，则指定 ALWCPYDTA(*OPTIMIZE)。也可规定 OPTIMIZE FOR n ROWS，来通知应用程序优化去读每一个结果行，这要给 n 设一大的数，在查询结束前也可给 n 设为小的数。
使用连接逻辑文件或 JOIN 语句	如果查询优化未选择最有效的连接次序，这会提高性能，采取这个动作的责任是，查询不可能依靠转换连接次序来进一步改善性能。
规定 ALWCPDTA（*OPTIMIZE）来允许查询优化使用分类例程。	规定了次序和全部键列来自一个拨号，这就允许查询优化去考虑全部可能的连接顺序。

22.6 有效使用 SQL 索引

AS/400 DB2 为访问查询表提供两个基本方式：表扫描（顺序的）和索引检索（直接的）。索引检索通常比表扫描更有效。然而，当检索很大百分比的页时，表扫描要比索引检索有效。

如果 AS/400 DB2 不能用索引去访问表中数据，它不得不读表中全部数据。很大的表就出现特殊性能问题：取表中所有数据的高成本。下面的建议帮助你设计编码，使用面向 AS/400 DB2 实现可用索引的优点。

1.避免数值转换

当一个列值和主变量比较时，要尽量指定相同的数据类型和属性。如果主变量或文字值有比列更好的精度，则 AS/400 DB2 不能使用命名列的索引。如果相比较的两项是不同的数据类型，AS/400 DB2 将不得不转换其中一个的值，这就可能导致不准确（由于有限的机器精度）。例如，EDUCLVL 是一半字整型值（SMALLINT），要规定：

```
... WHERE EDUCLVL < 11 AND
                    EDUCLVL >= 2
```

2.避免字符串填充

在把定长字符串列值与主变量或文字值相比较时，尽量使用相同长度。如果文字值或主变量的长度多于列的长度，则 AS/400 DB2 不能使用索引。例如：EMPNO 是 CHAR(6)，DEPTNO 是 CHAR（3）。要规定：

```
... WHERE EMPNO > '000300' AND
                    DEPTNO < 'E20'
```

3.避免使用以%或_开头的 LIKE 结构

百分号(%)或下划线(_)在用到 LIKE 谓词的结构中，规定一个字符串类似于要选择的行的列值，在字符串中间或结尾处使用指示字符，就如：

```
... WHERE LASTNAME LIKE 'J%SON%'
```

它们能取索引的好处，但如果用在字符串开头，如：

```
... WHERE LASTNAME LIKE '%SON'
```

它们能让 AS/400 DB2 避免使用任何在 LASTNAME 列中定义的索引，来限制行扫描的数量，这样应该避免在字符串的开头使用这两个符号，尤其在特别大表时更要注意。

4.小心 AS/400 DB2 不能使用下例中的索引：

对希望更新的列，例如，程序可以包括：

```
EXEC SQL
 DECLARE DEPTEMP CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT
      FROM CORPDATA.EMPLOYEE
      WHERE (WORKDEPT = 'D11' OR
                WORKDEPT = 'D21') AND
               EMPNO = '000190'
      FOR UPDATE OF EMPNO, WORKDEPT
END-EXEC.
```

尽管不想更新雇佣的部门，AS/400 DB2 不能使用 WORKDEPT 做键字的索引。

如果索引中使用的所有可更新的列也用在查询中,做为一个有等号操作符的独立选择谓词，AS/400 DB2 能使用索引，在前例中，将使用 EMPNO 键字索引。

如果 FOR UPDATE OF 列表只命名你想更新的列 WORKDEPT,AS/400 DB2 操作将更有效率，除非你想更新列，就不要用 FOR UPDATE OF 列表规定一个列。

由于动态 SQL 或 FOR UPDATE 子句没有规定可用的游标,同时程序包括一个 UPDATE 语句，则全部列都能被更新。

同一行中一个列与另一个列相比较，例如：

```
EXEC SQL
 DECLARE DEPTDATA CURSOR FOR
    SELECT WORKDEPT, DEPTNAME
      FROM CORPDATA.EMPLOYEE
      WHERE WORKDEPT = ADMRDEPT
END-EXEC.
```

尽管对 WORKDEPT 有一个索引和对 ADMRDEPT 有另一个索引，AS/400 DB2 不能使用其中任何一个索引，因为要查看表中每一行，故索引不能增加受益。

22.7 使用有分类排序的索引

下面介绍索引如何与分类排序表一起处理的有用信息，详细内容请看 AS/400 DB2 SQL 参考。

22.7.1 使用有选择、连接或分组的索引和分类排序

使用已有的索引前，AS/400 DB2 要确保列的属性（选择、连接或分组列）要匹配于已

有索引中键字段的属性，分类排序表是一个附加的必须被比较的属性。

与查询相关的分类排序表必须匹配于建立已有索引的分类排序表，AS/400 DB2 要比较分类排序表，如果它们不匹配，不能用已有的索引。

对此有一例外，如果与查询相关的分类排序表是唯一权顺序表(包括*HEX)，AS/400 DB2 操作就象是没有规定分类排序表选择、连接或分组列，而要使用下面的操作符和谓词：

等号（＝）操作符

不等号（[＝或<>）操作符

LIKE 谓词

IN 谓词

当这些条件为真时，AS/400 DB2 则可自由的使键字段与列匹配的已有索引，且此索引要满足：

索引不包括分类排序表。

或索引包括唯一权分类排序表。

注：表并非必须匹配与查询相联的唯一权分类排序队列表。

一个表使用多个索引，位图处理有特殊考虑。如果两个或多个索引在有查询选择引用的索引间有公共字段，则这些索引必须使用同一分类排序表或使用非分类排序表。

22.7.2 排序

除非优化选择分类去满足排序要求，与索引相关的分类排序表必须匹配于与查询相关的分类排序表。

在使用分类时，在分类过程中会发生转换。既然分类处理是分类排序的要求，这就允许 AS/400 DB2 使用任何符合选择标准的已有索引。

22.7.3 索引例子

根据例子要求，假设生成三个索引：

用*HEX 做为分类顺序生成索引 HEXIX

```
CREATE INDEX HEXIX ON STAFF (JOB)
```

用唯一权分类排序生成索引 UNQIX

```
CREATE INDEX UNQIX ON STAFF (JOB)
```

用共享权分类排序生成索引 SHRIX

```
CREATE INDEX SHRIX ON STAFF (JOB)
```

22.7.3.1 例 1

用无分类排序表的相等选择：

```
SELECT * FROM STAFF
  WHERE JOB = 'MGR'
```

AS/400 DB2 能使用索引 HEXIX 或索引 UNQIX。

22.7.3.2 例 2

用唯一权分类排序表的相等选择：

```
(SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

   SELECT * FROM STAFF
    WHERE JOB = 'MGR'
```

AS/400 DB2 能使用索引 HEXIX 或索引 UNQIX。

22.7.3.3 例 3

用共享权分类排序表的相等选择：

```
(SRTSEQ(*LANGIDSHR) LANGID(ENU)).

   SELECT * FROM STAFF
    WHERE JOB = 'MGR'
```

AS/400 DB2 仅使用索引 SHRIX。

22.7.3.4 例 4

用唯一权分类排序表的大于选择：

```
(SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

   SELECT * FROM STAFF
    WHERE JOB > 'MGR'
```

AS/400 DB2 仅能使用索引 UNQIX。

22.7.3.5 例 5

用唯一权分类排序表的连接选择：

```
(SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

   SELECT * FROM STAFF S1, STAFF S2
    WHERE S1.JOB = S2.JOB
```

或使用了 JOIN 语句的同一查询：

```
   SELECT *
    FROM STAFF S1 INNER JOIN STAFF S2
```

```
        ON S1.JOB = S2.JOB
```

AS/400 DB2 对查询能使用索引 HEXIX 或索引 UNQIX。

22.7.3.6 例 6

用共享权分类排序表的连接选择：

```
(SRTSEQ(*LANGIDSHR) LANGID(ENU)).

  SELECT * FROM STAFF S1, STAFF S2
   WHERE S1.JOB = S2.JOB
```

或使用 JOIN 语句的同一查询：

```
  SELECT *
   FROM STAFF S1 INNER JOIN STAFF S2
        ON S1.JOB = S2.JOB
```

AS/400 DB2 对任一查询仅使用索引 SHRIX。

22.7.3.7 例 7

用非分类排序表的排序（STRSEQ（*HEX））。

```
    (SRTSEQ(*HEX)).

  SELECT * FROM STAFF
   WHERE JOB = 'MGR'
   ORDER BY JOB
```

AS/400 DB2 只能使用索引 HEXIX。

22.7.3.8 例 8

用唯一权分类排序表的排序：

```
      (SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

  SELECT * FROM STAFF
   WHERE JOB = 'MGR'
   ORDER BY JOB
```

AS/400 DB2 只能使用索引 UNQIX。

22.7.3.9 例 9

用共享分类排序表的排序：

```
    (SRTSEQ(*LANGIDSHR)  LANGID(ENU)).

  SELECT * FROM STAFF
   WHERE JOB = 'MGR'
   ORDER BY JOB
```

AS/400 DB2 只能使用索引 SHRIX。

22.7.3.10 例 10

用 ALWCPYDTA(*OPTIMIZE)和唯一权分类排序表排序：

```
    ALWCPYDTA(*OPTIMIZE)
        (SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

  SELECT * FROM STAFF
   WHERE JOB = 'MGR'
   ORDER BY JOB
```

AS/400 DB2 对选择能使用索引 HEXIX 或索引 UNQIX，在使用*LANGIDUNQ 分类排序表分类期间做排序。

22.7.3.11 例 11

用非分类排序表的分组：

```
    (SRTSEQ(*HEX)).

  SELECT * FROM STAFF
   GROUP BY JOB
```

AS/400 DB2 使用索引 HEXIX 或索引 UNQIX。

22.7.3.12 例 12

用唯一权分类排序表的分组：

```
    (SRTSEQ(*LANGIDUNQ)  LANGID(ENU)).

  SELECT * FROM STAFF
   GROUP BY JOB
```

AS/400 DB2 能使用索引 HEXIX 或索引 UNQIX。

22.7.3.13 例 13

用共享权分类排序表的分组：

```
    (SRTSEQ(*LANGIDSHR)  LANGID(ENU)).

  SELECT * FROM STAFF
   GROUP BY JOB
```

AS/400 DB2 只能使用索引 SHRIX。

下面这些例子假设建在列 JOB 或 SALARY 上有多于 3 个的索引，在例子前有 CREATE INDEX 语句：

假定用*HEX 分类顺序生成索引 HEXIX2：

```
   CREATE INDEX HEXIX2 ON STAFF (JOB, SALARY)
```

假定生成了索引 UNQIX2 且分类顺序是唯一权：

```
    CREATE INDEX UNQIX2 ON STAFF (JOB, SALARY)
```

假定生成了索引 SHRIX2，使用共享权分类顺序：

```
   CREATE INDEX SHRIX2 ON STAFF (JOB, SALARY)
```

22.7.3.14 例 14

在同一列上用唯一权分类排序表排序和分组：

```
    (SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

  SELECT * FROM STAFF
   GROUP BY JOB, SALARY
   ORDER BY JOB, SALARY
```

AS/400 DB2 能使用 UNQIX2 去满足分组和排序的需要，如果索引 UNQIX2 不存在，AS/400 DB2 用*LANGIDUNQ 分类顺序表生成一个索引。

22.7.3.15 例 15

在同一列上用 ALWCPYDTA(*OPTIMIZE)和唯一权分类排序表排序和分组：

```
      ALWCPYDTA(*OPTIMIZE)
        (SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

  SELECT * FROM STAFF
   GROUP BY JOB, SALARY
   ORDER BY JOB, SALARY
```

AS/400 DB2 使用 UNQIX2 去满足分组和排序的要求。如果 UNQIX2 不存在，AS/400 DB2

会用*LANGIDUNQ 分类顺序表生成索引，或使用索引 HEXIX2 去满足分组，且完成分类来满足排序。

22.7.3.16 例 16

在同一列上用共享权分类排序表的排序和分组：

```
    (SRTSEQ(*LANGIDSHR) LANGID(ENU)).

  SELECT * FROM STAFF
   GROUP BY JOB, SALARY
   ORDER BY JOB, SALARY
```

AS/400 DB2 使用 SHRIX2 去满足分组和排序的要求。如果 SHRIX2 不存在，AS/400 DB2 用*LANGIDSHR 分类顺序生成一个索引。

22.7.3.17 例 17

在同一列上用 ALWCPYDTA(*OPTIMIEZ)和共享权分类排序表的分组和排序：

```
    ALWCPYDTA(*OPTIMIZE)
       (SRTSEQ(*LANGIDSHR) LANGID(ENU).

  SELECT * FROM STAFF
   GROUP BY JOB, SALARY
   ORDER BY JOB, SALARY
```

AS/400 DB2 使用 SHRIX2 去满足分组和排序的要求。如果 SHRIX2 不存在，AS/400 DB2 用*LANGIDSHR 分类顺序生成一个索引。

22.7.3.18 例 18

在不同列上用唯一权分类排序表的排序和分组：

```
    (SRTSEQ(LANGIDUNQ) LANGID(ENU)).

  SELECT * FROM STAFF
   GROUP BY JOB, SALARY
   ORDER BY SALARY, JOB
```

AS/400 DB2 使用 HEXIX2 或 UNQIX2 去满足分组的要求，会生成一个临时结果表来放分组结果，在临时结果表上会用*LANGIDUNQ 分类排序表建立一个临时索引，以满足排序的要求。

22.7.3.19 例 19

在不同列上用 ALWCPYDTA(*OPTIMIZE)和唯一权分类排序表的分组和排序：

```
ALWCPYDTA(*OPTIMIZE)
   (SRTSEQ(*LANGIDUNQ) LANGID(ENU)).

SELECT * FROM STAFF
 GROUP BY JOB, SALARY
 ORDER BY SALARY, JOB
```

AS/400　DB2 使用 HEXIX2 或 UNQIX2 去满足分组的要求，要做分类以满足排序的要求。

22.7.3.20　例 20

在不同列上，用 ALWCPYDTA(*OPTIMIZE)和共享权分类排序表的分组和排序：

```
ALWCPYDTA(*OPTIMIZE)
     (SRTSEQ(*LANGIDSHR) LANGID(ENU)).

SELECT * FROM STAFF
 GROUP BY JOB, SALARY
 ORDER BY SALARY, JOB
```

AS/400　BD2 使用 SHRIX2 去满足分组要求，要做分类以满足排序要求。

22.8　使用 VARCHAR 和 VARGRAPHIC 数据类型的技巧

变长列（VARCHAR 或 VARGRAPHIC）支持允许把表中任何数目的列定义为变长，如果你使用 VACHAR 或 VARGRAPHIC 支持，表的大小通常要减少。

变长列中的数据在内部存储在两个区域内，一是定长区或 ALLOCATE 区，另一个是溢出区。如果规定缺省值，分配的长度至少要和值一样长。下面的观点有助于确定使用存储区的最好方式：

当定义一个变长数据表时，必须决定 ALLOCATE 区的宽度，如果主要目的是：

空间存储：使用 ALLOCATE(0)

性能：ALLOCATE 区应足够宽以能够合并至少 90%到 95%的列值。

平衡空间存储和性能是可行的。在下面的电子电话本例子中，使用下述数据：

8600 个名字，分为：最终名，初始名，中间名

最终，初始和中间列是变长的

最短的最终名是 2 个字符，最长的为 22 个字符。

这个例子给出使用变长列怎样存储空间，定长列的表使用最多的空间。此表仔细计算了分配尺寸表使用较少的磁盘空间，没有分配尺寸定义的表（与所有存在溢出区的数据一起）使用最少的磁盘空间。

支持种类	最终名 最大/分配	初始名 最大/分配	中间名 最大/分配	物理文件尺寸 总数	溢出区的记录数
定长	22	22	22	576K	0
变长	40/10	40/10	40/7	408k	73
变长缺省值	40/0	40/0	40/0	373K	8600

在许多应用程序中，必须考虑性能问题。如果使用缺省值 ALLOCATE(0)，它将加倍磁盘

装置的通信量。

ALLOCATE(0)要求两个读：一是读行的定长部分，一是读溢出空间。变长操作要仔细选择 ALLOCATE，减少溢出空间、增强性能，物理文件的大小比定长需少 28%，因为 1%的行在溢出区域，故访问最少要求两个读，变长的实现和定长大致相同。

要生成用 ALLOCATE 键字的表：

```
CREATE TABLE PHONEDIR
         (LAST    VARCHAR(40) ALLOCATE(10),
          FIRST   VARCHAR(40) ALLOCATE(10),
          MIDDLE  VARCHAR(40) ALLOCATE(7))
```

如果用主变量去插入或更新变长列，主变量也应是变长的。因为从定长主变量不能截断空格，使用定长主变量将引起更多的行流入到溢出空间，这将增加表的尺寸。

在这个例子中，定长主变量被用来在表中插入行：

```
01  LAST-NAME PIC X(40).
   ...
    MOVE "SMITH" TO LAST-NAME.
    EXEC SQL
      INSERT INTO PHONEDIR
       VALUES(:LAST-NAME, :FIRST-NAME, :MIDDLE-NAME, :PHONE)
    END-EXEC.
```

主变量 LAST_NAME 不是变长的，后有 35 个空格的字符串“SMITH”插入到 VARCHAR 列 LAST 中，值要比分配的尺寸 10 长，35 个结尾空格的 30 个是在溢出区中。

在这个例子中，变长主变量被用来在表中插入行：

```
01  VLAST-NAME.
    49 LAST-NAME-LEN PIC S9(4) BINARY.
    49 LAST-NAME-DATA PIC X(40).
   ...
    MOVE "SMITH" TO LAST-NAME-DATA.
    MOVE 5 TO LAST-NAME-LEN.
    EXEC SQL
      INSERT INTO PHONEDIR
     VALUES(:VLAST-NAME, :VFIRST-NAME, :VMIDDLE-NAME, :PHONE)
    END-EXEC.
```

主变量 VLAST_NAME 是变长的，数据的实际长度为 5，这个值比分配的长度要短，它能被放在列的固定部分。

对包括变长列的表运行 RGZPFM 命令能提高性能，溢出区中不能使用的碎片被 RGZPFM 命令压缩，这减少了从溢出区中读行时间，增加了引用的位置，同时，为串行批处理产生最佳顺序。

为变长列选择适当的最大长度，选择的长度太长会增加处理访问组(PAG)，大的 PAG 会

降低性能，很大的最大值使 SEQONLY(*YES)效率很低，大于 2000 字节长的变长列，不能选做键字列。

22.9 从多个表中选择数据时的性能改善

如果正考虑用选择语句访问两个或两个以上的表，在 22.6 中给出的所有建议都是适用的，下面的建议是对访问几个表的选择语句的特殊指导。对于多个表的连接，要提供关于连接列的冗余信息。在请求一个连接时，如果给优化额外信息处理，则它能确定一个最好的连接方式。这个附加信息可能是冗余的，但它有助于优化序。例如对下面的编码：

```
EXEC SQL
  DECLARE EMPACTDATA CURSOR FOR
  SELECT LASTNAME, DEPTNAME, PROJNO, ACTNO
       FROM CORPDATA.DEPARTMENT, CORPDATA.EMPLOYEE,
              CORPDATA.EMP_ACT
      WHERE DEPARTMENT.MGRNO = EMPLOYEE.EMPNO
             AND EMPLOYEE.EMPNO = EMP_ACT.EMPNO
END-EXEC.
```

在 WHERE 子句中提供给优化稍多一点的数据，用下面编码来代替上面的编码：

```
EXEC SQL
  DECLARE EMPACTDATA CURSOR FOR
  SELECT LASTNAME, DEPTNAME, PROJNO, ACTNO
       FROM CORPDATA.DEPARTMENT, CORPDATA.EMPLOYEE,
              CORPDATA.EMP_ACT
       WHERE DEPARTMENT.MGRNO = EMPLOYEE.EMPNO
              AND EMPLOYEE.EMPNO = EMP_ACT.EMPNO
              AND DEPARTMENT.MGRNO = EMP_ACT.EMPNO
END-EXEC.
```

22.10 减少打开数据库操作的数目来改善性能

SQL 数据管理语言语句必须做数据库打开操作来建立数据的打开数据路径（ODP），打开数据路径是指通过该路径，完成文件的输入/输出操作，它连接 SQL 应用程序和文件。程序中打开操作的数目能大大地影响性能，数据库打开操作发生在下面的语句中：

OPEN 语句
SELECT INTO 语句
有 VALUES 子句的 INSERT 语句
有 WHERE 条件的 UPDATE 语句
有 WHERE CURRENT OF 游标和引用操作数和函数的 SET 子句的 UPDATE 语句
有 WHERE 条件的 DELETE 语句

有选择语句的 INSERT 语句需要两个打开操作，子查询的某些格式也要每个子选择用一个打开。

为减少打开次数，AS/400 DB2 在重运行语句时，可保留 ODP 且可重用 ODP，但发生下列情况时不可以：

GROUP BY 包括来自有多个表的列

ODP 使用主变量建立子集临时索引，OS/400 的数据库支持可以选择建立临时索引，该索引的项，仅是那些与 SQL 语句中规定的行选择匹配的行。如果主变量用在行选择中，临时索引将没有对主变量表中不同值要求的项。

在主变量值上规定排序

主变量用来规定 LIKE 谓词的结构，主变量值有下划线(_)或多个查找结构：如“%ABC%DEF”包含两个结构，ABC 和 DEF

自 ODP 打开，发生 OVRDBF 或 DLTOVR 命令，这会影响 SQL 语句的执行。

注：只有影响了被引用表名的覆盖才会引起 ODP 在给定的程序启用中关闭。

在最后打开已执行修改了库列表，这将修改用系统命令方式做非限定引用时选择的文件名，被改文件是指在系统命令方式中被无限定安排对象所选择的文件。

被查询的文件是连接逻辑文件并且它的连接类型（JDFTVAL）不匹配于查询中指定的连接类型。

逻辑文件规定的格式引用多个物理文件。

文件是复合 SQL 视图，它要求临时文件包括 SQL 视图的结果。

AS/400 DB2 仅能重用被同一语句打开的 ODP，在程序中后来编码的恒等语句不重用其它任何语句打开的 ODP。如果这个恒等语句在程序中必须运行多次，那么在子例程中编码一次，同时调用子例程运行这个语句。

由 AS/400 DB2 打开的 ODP 在发生下面情况之一时被关闭：

完成 COLSE，INSERT，UPDATE，DELETE 或 SELECT 语句，并且该 ODP 需要临时结果或子集临时索引

发出 RCLRSC 命令，在下列情况下发出 RCLRSC 命令：

— 调用堆栈上的第一个 COBOL 程序结束

— COBOL 程序发布 STOP RUN 语句

RCLRSC 不关闭由 CLOSQLCSR(*ENDJOB)预编译程序生成的 ODP。

当调用堆栈中上有包括 SQL 语句的最后程序时，排除使用 CLOSQLCSR(*ENDJOB)预编译程序生成的 ODP，或用 CLOSQLCSR(*ENDACTGRP)预编译模块生成的 ODP。

用当一个 CONNECT（类型 1）语句修改活动组的应用服务时，所有为这个活动组生成的 ODP 都关闭。

当 DISCONNENT 语句结束了与应用服务的连接时，对应用服务的全部 ODP 关闭。

当释放连接结果成功的 COMMIT 结束时，应用服务的全部 ODP 被关闭。

你可以用下列方法控制 AS/400 DB2 是否保持打开的 ODP：

使用 CLOSQLCSR(*ENDJOB)或 CLOSQLCSR（*ENDACTGRP），

设计应用程序，使发布 SQL 语句的程序一直在访问堆栈上。

在 SET 子句中任何表达式包括操作符和函数时，AS/400 DB2 对每个 UPDATE WHERE CURRENT OF 的第一执行作一个打开操作，这个打开可以避免在主语言中编码函数或操作符。

例如：下面的 UPDATE 引起 AS/400 DB2 去做一打开操作：

```
EXEC SQL
  FETCH EMPT INTO :SALARY
END-EXEC.
```

```
EXEC SQL
 UPDATE CORPDATA.EMPLOYEE
   SET SALARY = :SALARY + 1000
   WHERE CURRENT OF EMPT
END-EXEC.
```

相反，使用下面的编码技术来避免打开：

```
EXEC SQL
 FETCH EMPT  INTO  :SALARY
END EXEC.

ADD 1000 TO SALARY.

EXEC SQL
 UPDATE CORPDATA.EMPLOYEE
   SET SALARY = :SALARY
   WHERE CURRENT OF EMPT
END-EXEC.
```

用 CL 命令 TRCJOB 和 DSPJRN 来确定由 SQL 语句所完成的打开数目。

22.11 由数据库管理分块考虑带来的性能改善

为了提高性能，SQL 运行时企图在任何可能的时候从数据库管理回收并插入成块的记录。可使用CL命令的SEQONLY参数来控制分块,OVRDBF优先调用包括SQL语句或用CRTSQLxxx命令中 ALWBLK 参数的应用程序，数据库管理不允许在下述情况下分块：

游标是更新或删除属性

行长度加上反馈信息长度大于 32767，反馈信息的最小值是 11 个字节，反馈大小，是由游标所用索引中键字段的字节数和键字段数增加的，如有任何一个，则为空属性。

规定了 COMMIT(*CS)，且没规定 ALWBLK(*ALLREAD)

规定了 COMMIT(*ALL)并且下列为真：

- 没用 SELECT INTO 语句或成块 FETCH 语句
- 查询未使用列函数或规定由列分组
- 不非得生成临时结果表

规定了 COMMIT(*CHG)且没规定 ALWBLK(*ALLREAD)。

游标至少包括一个子查询,并且最外面的子选择提供了相关的子查询引用或者最外面的子选择处理子查询，该子查询有 IN，＝ANY 或<>ALL 子查询的谓词操作符，这将都作一个相关引用。

在下面的情况下，SQL 运行时通过数据库管理自动地分块记录：

INSERT

如果 INSERT 语句包含一个选择语句，被插入的记录成块，并在块不满时不实际地插入到目的表中，SQL 运行时自动地为块插入做分块。

注：如果规定有 VALUES 子句的 INSERT，到程序结束为止，SQL 运行时不实际关闭被使用的内部临时表去实现插入。如果再运行同一 INSERT 语句不需要全打开，并且应用程序运行快得多。

OPEN

当记录被取回时，在下列条件为真时在 OPEN 语句下分块：

—游标只用在 FETCH 语句上

—在程序中没有 EXECUTE 或 EXECUTE IMMEDIATE 语句，或者规定了 ALWBLK(*ALLREAD)，或用 FOR FETCH ONLY 子句说明了游标。

—规定了 COMMIT(*CHG) 和 ALWBLK(*ALLREAD)，规定了 COMMIT(*CS) 和 ALWBLK(*ALLREAD)，或规定了 COMMIT(*NONE)。

22.12 使用 FETCH FOR n ROWS 的性能改善

成功地完成多个 FETCH 语句的应用程序可用 FETCH FOR n ROWS 来提高性能。用这个子句，能从一个表中的取回 n 行数据，并把它们放到有一个 FETCH 的主结构数组中或行存储区中。

详细内容请看 AS/400 DB2 SQL 参考。

使用了 FETCH 语句的 SQL 应用程序，在没用 FOR n ROWS 子句时，能用多行 FETCH 语句取回多行来提高性能。在主结构数组或行存储区被 FETCH 填满后，应用程序能循环通过数组或存储区的数据来处理每一个记录，因为只调用一次 SQL 运行并且全部的数据都同时返给应用程序，所以语句运行的比较快。

可修改应用程序来允许数据管理对被 SQL 运行时从表中取回的记录进行分块。详细内容请看 22.11。

在下表中，程序试图取出 100 行放到应用程序中。当实现分块时，注意调用 SQL 运行时和数据库管理的次数：

表 22－3　　使用 FETCH 语句调用的次数

	不使用分块的数据库管理	使用分块的数据库管理
单行 FETCH 语句	100 个 SQL 调用 100 个数据库调用	100 个 SQL 调用 1 个数据库调用
多行 FETCH 语句	1 个 SQL 运行时调用 100 个数据库调用	1 个 SQL 运行时调用 1 个数据库调用

22.12.1SQL 分块的性能改善

当使用 FETCH FOR n ROWS 时，要考虑下面几点的特殊的性能问题，注意下面内容有助于改善 SQL 分块性能：

主结构数组中的属性信息或者与行存储区有关的描述要匹配于取回行的属性，应用程序使用一个多行 FETCH 调用取回尽可能多的行，多行 FETCH 请求的分块因子不是由系统页尺寸或 OVRDBF 命令的 SEQONLY 参数控制，它们是由多行 FETCH 请求中要求的行数所控制。

在程序中，不要对同一游标用混合的单行 FETCH 和多行 FETCH 请求，如果游标的一个 FETCH 做为多行 FETCH 对待，则游标的一个取回都被认为是多行 FETCH。这样每一个单行的 FETCH 请求被认为是一行的多个 FETCH。

PRIOR，CURRENT，RELATNE 滚动选项不能用于多行 FETCH 语句，为了让应用程序随机移动游标，数据库管理必须与应用程序一起维护相同的游标的位置，这样，SQL 运行时把指定了这些选项的可滚动的全部 FECTH 请求都认为是多行 FETCH 请求。

22.13 使用 INSERT n ROWS 的性能改善

成功地完成多个 INSERT 语句的应用程序可以用 INSERT n ROWS 来提高性能。用这个子句，能把来自主结构数组的一行或多行数据插入到目标表中。这个数组必须是一个结构数组，它的结构元素与目标表中的列要一致。

循环 INSERT…VALVES 语句(不用 n ROWS 子句)的 SQL 应用程序可以用 INSERT n ROWS 把多行插入表中来提高其性能。在应用程序已经用记录循环充满了主数组后，用一个 INSERT n ROWS 语句可把整个数组插入到表中。因为仅调用一次 SQL 运行并且所有数据都被同时插入到目标表中，所以语句运行起来快得多。

在下面的表中，程序试图插入 100 行到表中，在分块操作时，注意调用 SQL 运行时和调用数据库管理的次数间的不同。

表 22－4 使用 INSERT 语句的调用次数

	未使用分块的数据库管理	使用分块的数据库管理
单行 INSERT 语句	100 个 SQL 运行时调用 100 个数据库调用	100 个 SQL 运行时调用 1 个数据库调用
多行 INSERT 语句	1 个 SQL 运行时调用 100 个数据库调用	1 个 SQL 运行时调用 1 个数据库调用

22.14 分页交互显示数据时性能的改善

在用大表时，分页性能常常被降低，这是因为在 STRSQL 命令中 REFRESH(*ALWAYS)参数直接从表中动态地取回表中最后一个数据，而规定 REFRESH(*FORWARD)能改善分页性能。

在用 REFRESH(*FORWARD)交互显示数据时，选择语句的结果复制到临时文件中，同时，可向前推页通过显示器。其它共享此表的用户能修改正显示的选择语句结果行。如果向前或向后推已显示过的行，则显示的行是临时文件中的那些行，而不是在更新的表中的那些行。

可在对话服务显示中修改刷新选项。

22.15 用有效的 SELECT 语句的性能改善

在 SELECT 语句的选择列表中规定的列数会使数据库管理从基础表中取回数据并把它们映射到应用程序的主变量中。减少规定的列数，能节省处理单元资源使用。编码 SELECT * 很方便，它对于应用程序比明确写上实际需要的列要快，如果要用索引访问或所有的列都参与分类操作，这是特别重要的。

22.16 使用活动数据的性能改善

在不使用数据复制取回数据时，术语‘活动数据’引用数据库管理使用的访问类型。使用这种类型的访问，返给程序的数据总是反映数据库中数据的当前值。程序员能够控制数据库管理是使用数据复制取回数据还是直接取回数据，这些是用在预编译程序命令中或 STRSQL 命令中规定 ALWCPYDTA 参数来实现的。

规定 ALWCPYDTA(*NO)指示数据库管理使用活动数据。因为不用重新关闭或打开游标来重定位要取的数据，所以活动数据访问可改善性能。可用在显示中生成一个列表来体现这个优点，如果显示屏一次只能显示表的 20 个元素，那么在显示开始的 20 个元素后，应用程序能请求显示下面 20 行。设计用在非 OS/400 操作系统上的典型 SQL 应用程序，它的结构如下：

```
EXEC SQL
    DECLARE C1 CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT
```

```
       FROM CORPDATA.EMPLOYEE
      ORDER BY EMPNO
END-EXEC.

EXEC SQL
    OPEN C1
END-EXEC.

*    PERFORM FETCH-C1-PARA  20 TIMES.

     MOVE EMPNO to LAST-EMPNO.

EXEC SQL
    CLOSE C1
END-EXEC.

*    Show the display and wait for the user to indicate that
*    the next 20 rows should be displayed.

EXEC SQL
    DECLARE C2 CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT
      FROM CORPDATA.EMPLOYEE
    WHERE EMPNO > :LAST-EMPNO
    ORDER BY EMPNO
END-EXEC.

EXEC SQL
    OPEN C2
END-EXEC.

*    PERFORM FETCH-C21-PARA  20 TIMES.
*    Show the display with these 20 rows of data.

EXEC SQL
    CLOSE C2
END-EXEC.
```

在上面的例子中，要注意不得不打开一个附加的游标以继续这个列表和得到当前数据，这能导致创建附加的 ODP，它会增加 AS/400 系统的处理时间。代替上面的例子，程序员可用下面的 SQL 语句来设计应用程序，其中规定 ALWCPYDTA(*NO)：

```
EXEC SQL
    DECLARE C1 CURSOR FOR
```

```
    SELECT EMPNO, LASTNAME, WORKDEPT
      FROM CORPDATA.EMPLOYEE
     ORDER BY EMPNO
END-EXEC.

EXEC SQL
    OPEN C1
END-EXEC.

*    Display the screen with these 20 rows of data.

*    PERFORM FETCH-C1-PARA  20 TIMES.

*    Show the display and wait for the user to indicate that
*    the next 20 rows should be displayed.

*    PERFORM FETCH-C1-PARA  20 TIMES.

EXEC SQL
    CLOSE C1
END-EXEC.
```

在上面的例子中，如果把 FOR 20 ROWS 子句用在多行 FETCH 语句中，查询能完成的更好，之后，可用一个操作取回 20 行。

22.17 使用 ALWCPYDTA 参数的性能改善

用分类或散列方式估算查询代替使用或创建索引方式，能更好地完成一些复杂的查询。靠使用分类或散列，数据库管理能把选择从分类或分组处理中分离出来，这种分离允许对选择使用效率高的索引。例如：考虑下面的 SQL 语句：

```
EXEC SQL
    DECLARE C1 CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT
      FROM CORPDATA.EMPLOYEE
      WHERE WORKDEPT = 'A00'
     ORDER BY LASTNAME
END-EXEC.
```

在上面的例子中，规定 ALWCPYDTA(*NO)或 ALWCPYDTA(*YES)时，数据库管理可以试着从有 LASTNAME 列的第一个索引中创建索引，如果有这样的索引，用这个索引扫描表中的行以便选择与 WHERE 条件匹配的行。

如果规定 ALWCPDTA(*OPTIMIZE)，数据管理使用有第一个索引列 WORKDEPT 的索引，然后复制与 WHERE 条件匹配的所有行，最后，用 LASTNAME 中的值来分类复制的行。这个行选择处理是相当有效的，这是因为是立即使用这个索引来定位选择的行。

ALWCPYDTA(*OPTIMIZE)会优化处理查询需要的总时间，但因为数据复制要在返回结果表第一行之前做，所以接收第一行所需的时间可能会增加。详细内容请看 22.18。

包含连接操作的查询也可从 ALWCPYDTA(*OPTIMIZE)中获益，这是因为连接顺序可以不顾 ORDER BY 规定而被优化。

注：散列方式不能用在包括嵌套的连接查询的分组操作中，也不用生成临时结果表。

22.18 使用优化子句的性能改善

若一个应用程序不是用一个游标取回整个结果表，使用 OPTIMIZE 子句可以改进性能。查询优化使用 OPTIMIZE 子句中规定的值修改成本估算来取行中的子集。

假设如下查询返回 1000 行：

```
EXEC SQL
    DECLARE C1 CURSOR FOR
    SELECT EMPNO, LASTNAME, WORKDEPT
      FROM CORPDATA.EMPLOYEE
      WHERE WORKDEPT = 'A00'
    ORDER BY LASTNAME
    OPTIMIZE FOR 100 ROWS
END EXEC.
```

优化计算如下成本：

优化比率=对 n 行最优值 / 用回答列表估算的行数

使用一个临时建立的索引的成本：

取得回答列表行成本 + 建立索引成本 + 用临时索引取得行的成本 × 优化比率

使用 SORT 的成本：

取得回答列表行成本 + SORT 输入成本 + SORT 输出处理成本 × 优化比率

使用已有索引的成本：

使用索引取得回答列表行的成本 × 优化比率

在上面的例子中，用分类和建立索引估算的成本值不能由优化比率调整，它使优化来平衡最优和处理需要。如果优化数比结果表中的行数大，成本估算没有调整的必要。若在查询中没规定 OPTIMIZE 子句，那么根据 ALWCPYDTA 的值或输出设备来用缺省值。

语句类型	ALWCPYDTA（*OPTIMIZE）	ACWCPYDTA（*YES 或*NO）
DECLARE CURSOR	在结果表中的数或行	3%或结果表中的行数
嵌入的选择	2	2
对显示的 INTERACTIVE 选择输出	3%或结果表中的行数	3%或结果表中的行数
对打印或数据库文件的 INTERACTIVE 选择输出	在结果表中的行数	在结果表中的行数

OPTIMIZE 子句影响查询优化：

使用一个现有索引（规定一个小的数），

建立一个索引或运行一个分类或在回答列表中规定一个大量而且可用行的散列表。

22.19 保留游标位置来改善性能

可通过保留游标位置改进性能。在下面两节提供对非-ILE 和 ILE 程序调用保存游标的信息。

22.19.1 对非-ILE 程序调用保留游标位置的性能改善

对非-ILE 程序调用，CLOSQLCSR 参数允许规定如下范围：

游标
准备语句
锁

当正确使用 CLOSQLCSR 参数时，它可以减少 SQL 需要的 OPEN，PREPARE，和 LOCK 语句个数，它还可以允许保留游标位置跨过程序调用来简化应用程序。

*ENDPGM　这是所有非-ILE 预编译的缺省值，用这个选项，仅在程序在调用堆栈中打开时，游标保持打开和可访问。当程序结束，SQL 游标不再使用，准备语句也丢失，但锁仍然保持直到调用堆栈中最后一个 SQL 程序完成。

*ENDSQL　用这个选项由程序建立 SQL 游标和准备语句保持打开，直到完成在调用堆栈中的最后一个 SQL 程序，它们不能被其它程序使用，仅可对同一程序不同的调用使用，锁保持直至完成在调用堆栈中的最后一个。

*ENDJOB　这个选项允许保持 SQL 游标、准备语句和锁在整个工作过程中是活动的。当在堆栈中最后一个 SQL 程序完成，任何通过*ENDJOB 程序建立的 SQL 资源仍为活动的，锁保持有效。没用 CLOSE，COMMIT 或 ROLLBACK 明显打开的游标保持打开，准备语句对同一程序的后读调用仍然可用。

22.20 保留跨越 ILE 程序调用的游标位置的性能改善

对 ILE 程序调用，CLOSQLCSR 参数允许规定如下范围：

游标
准备语句
锁

当恰当使用 CLOSQLCSR 参数时可以减少需要的 SQLOPEN，PREPARE，和 LOCK 语句。它还可以在调用中允许保留游标位置简化程序，。

*ENDACTGRP 对 ILE 预编译它是缺省值。用这个选项，SQL 游标和准备语句保持开直至活动组中程序运行结束，它们不能被其它程序使用，仅可对同一个程序进行不同调用。锁保持直至动态组结束。

*ENDMOD　当打开的模块是活动时游标保持打开和可访问，当模块结束，游标不再使用，SQL 准备语句将丢失，锁仍保持直到在调用栈最后一个 SQL 程序完成。

22.21 对所有程序调用保留游标位置的一般规则

当使用 CLOSQLCSR（*ENDPGM）或 CLOSQLCSR（*ENDMOD）编译程序时，每次调用时程序和模块都必须打开游标，才能访问数据。若 SQL 程序或模块被调用几次，想使用 ODP 的优点，在程序和模块结束之前必须明确地关闭游标。

使用 CLOSQLCSR 参数且规定*ENDJOB，*ENDSQL 或*ENDACTGRP，不需在每次调用中都使用 OPEN 和 CLOSE 语句，为了能少运行几条语句，可以在程序和模块调用之间保持游标位置。

下面例子帮助说明使用 CLOSQLCSR 参数的优点：

```
EXEC SL
 DECLARE DEPTDATA CURSOR FOR
```

```
    SELECT EMPNO, LASTNAME
      FROM CORPDATA.EMPLOYEE
      WHERE WORKDEPT = :DEPTNUM
END-EXEC.

EXEC SQL
 OPEN DEPTDATA
END-EXEC.

EXEC SQL
 FETCH DEPTDATA INTO :EMPNUM, :LNAME
END-EXEC.

EXEC SQL
 CLOSE DEPTDATA
END-EXEC.
```

若此程序被另一个 SQL 程序调用几次，它将能使用一个可再利用的 ODP，只要 SQL 在调用程序之间保持活动，不需用 OPEN 语句请求数据库的打开操作，然而在每一个 OPEN 语句后，游标定位仍在第一个结果行上，FETCH 语句总是返回第一行。

在下面例子中，已取消 CLOSE 语句：

```
EXEC SQL
 DECLARE DEPTDATA CURSOR FOR
   SELECT EMPNO, LASTNAME
     FROM CORPDATA.EMPLOYEE
     WHERE WORKDEPT = :DEPTNUM
END-EXEC.

  IF CURSOR-CLOSED IS = TRUE THEN
EXEC SQL
 OPEN DEPTDATA
END-EXEC.
EXEC SQL
 FETCH DEPTDATA INTO :EMPNUM, :LNAME
END-EXEC.
```

若该程序用*ENDJOB 或*ENDACTGRP 选项预编译且活动组保持活动，游标位置不变。下面情况下游标位置亦不变：

程序用与*ENDSQL 选项预编译。

SQL 在程序调用间保持活动。

这个策略结果是，每次调用程序都用游标取回下一个记录。在后续数据请求时，不需要 OPEN 语句，事实上，出现-502 的 SQLCODE，可以忽略这个错误，或跳出 OPEN。可以先用 FETCH 语句，如操作失败可仅运行 OPEN 语句。

这个技术也用在准备语句中。程序先试图做 EXECUTE，若失败，再做 PREPARE。结果是假定选择了当前的 CLOSQLCSR 语句，仅在第一个调用程序时需 PREPARE 语句。当然，若在程序调用之间修改语句，可以在所有情况下执行 PREPARE。

主程序也可以通过仅在第一次调用时发送一个特定参数值完成这种控制，这个特定参数值指出，由于它是第一次调用，以下程序将完成 OPENs，PREPAREs 和 LOCKs。

注：若使用 COBOL 程序，不能使用 STOP　RUN 语句，当调用堆栈中第一个 COBOL 程序结束或运行 STOP　RUN 语句时做一个 RCLRSC 操作，这个操作关闭 SQL 游标，*ENDSQL 选项没按期望来工作。

22.22　SQL　PREPARE 语句的性能改善

SQL　PREPARE 语句运行处理与预编译时处理很相似，要准备的语句是：

句法检查

确认目标使用的有效性

建立一个访问方案

当一个语句再一次执行或打开，数据库管理将再确认访问方案是否仍然有效。打开的大部分对在 PREPARE 处理期间的有效性检查来说是多余的。DLYPRP（*YES）参数规定程序中的 PREPARE 语句是否对动态语句完全有效。确认是在动态语句打开或执行时完成的。这个参数能对使用 PREPARE 语句的程序的性能有明显改变，这是因为它削减了多余检查。规定这个预编译选项的程序在运行 OPEN 或 EXECUTE 语句后要检查 SQLCODE 和 SQLSTATE 来保证语句有效。如果在 PREPARE 语句中用 INTO 子句，或在对语句发出 OPEN 之前，DESCRIBE 语句使用动态语句，则 DLYPRP（*YES）不能提供任何性能改善。

22.23　使用长目标名对性能的影响

用在 SQL 语句中的长目标名被转化为系统目标名，这个转化有一些性能影响。若长目标名用一个库名限定，在预编译时转化成短名。在这种情况下，语句执行时没有性能影响，然而，若长名没限定，转化在执行时完成，就有一个小的性能影响。

22.24　使用预编译选项的性能改善

生成程序时有几种预编译选项对改进性能有利，它们仅仅是一个选项，因为使用它们可能影响程序的功能。基于这个原因，这些参数的缺省值是确保应用程序从以前版本成功移植的值，但用这些选项的其它值可以改进性能。下表给出这些预编译选项和它们的对性能的影响。

预编译选项	理想值	改　善	注意事项	相关题目
ALWCPYDTA	*OPTIMIZE	查询的排序和分组标准影响选择标准	对数据的拷贝可在查询打开时完成	请看 22.17
ALWBLK	*ALLREAD	附加的只读游标使用分块	ROLLBACK　HOLD 可不更改只读游标的位置，定位更新或删除的动态处理可能失败	请看 22.11
CLOSQLCSR	*ENDTOB *ENDSQL　或 *ENDACTGRP	可跨越程序范围保持游标定位	在程序范围结束时，不做明确的关闭 SQL 游标动作	请看 22.19.1

DLYPRP	*YES	使用 SQL PREPARE 语句的程序可以运行的较快	对准备语句的有效性检查要延迟到语句运行或打开时做	请看 22.22
TGTRLS	*CURRENT(缺省值)	预编译能生成在当前版本中可用的性能改善目标码	程序不能在先前版本系统下使用	请看 22.25

某些选项对大多数应用程序都可用，用 CRTDUPOBJ 命令来生成 CRTSQLxxx 和 CHGCMDDFT 的复本，来为预编译选定理想值。可用 DSPGM、DSPSRVPGM、DSPMOD 或 PRTSQLINF 命令来显示程序目标所用的预编译选项。

22.25 由结构参数传递技术产生的性能改善

SQL 预编译使用两种方法把主变量传递给 SQL 运行时变量，第一个方法是把每个主量分别做为一个参数传递：

```
CALL QSQROUTE
     (SQLCA, hostvariable1, hostvariable2, hostvariable3)
```

第二种方法是建立一个数据结构，在语句中引用的每个主变量做为一个元素，然后数据结构作为一个参数传递：

```
CALL QSQROUTE
     (SQLCA, hostvariable structure)
```

第二种方法的性能好一些

注：在 PL/1 和 RPG/400 程序中不能使用结构参数传递技术。

22.25.1 参数传递的后台信息

程序中生成的附加码是为了在调用 SQL 程序之前把输入主变量数据移到结构中，在调用 SQL 程序之后，附加码把数据从结构中移出送到输出主变量中。

预编译生成结构首先是 SQL 标题，接下来是输入主变量，输出主变量，输入及输出主变量的指示器。因为输出主变量是建立在邻接的存储空间中，SQL 运行时支持可以检查 I/O 缓冲区是否匹配（检查每个结果列属性是否匹配主变量属性），若它们都匹配，就用一个结构移动数据。

结构中 SQL 标题包含语句唯一的信息，这使 SQL 在每次调时不用重建信息。一些信息是通过 SQL 运行时支持加上和处理的。SQL 运行时支持建立一个 SQLDA，对原始的参数列表，主变量地址每次对语句调而不同，因此，SQL 运行时支持对每次语句调用重建 SQLDA。在结构参数传递的同时，生成结构作为一个静态变量，且元素的地址不改变。SQL 运行时支持在第一次调用语句时建立 SQLDA 并且存储 SQLDA，以便在以后的语句调用时使用。

用第一种传递参数类型，在一个程序中能引用的主变量数大约是 4000，这是因为一个程序要限制在 4100 个指针之内，每个参数要一个指针。用结构参数传递，对每个 SQL 语句仅要传递两个参数，这样就不用限制 4000 个指针了。

22.25.2 结构参数传递技术的区别

在应用程序嵌入 SQL 语句时，要注意在参数传递技术的下列不同之处：

返回使用结构参数传递技术并且主变量数比选择列表中列数多时返回 SQL CODE +326。

对第一种参数传递技术忽略附加的输出主变量，但对结构参数传递技术，主变量设为 SQL 建立的结构中的值（字符主变量用空格，数值主变量用零）。

注意：当附加输出主变量与其它主变量重叠时会导致不正确的结果。

这种情况的一个 RPG/400 例子是：

```
ISTRUC       DS
I                    1   1 A
I                    2   2 B
I                    1   2 C

    SELECT * INTO :STRUC FROM ATABLE
```

在上面的例子中，若 ATABLE 只有一个或两个列，SQLCODE 将为+326。当从 SQL 结构对 C 赋值时，A 和 B 为空格来代替 A 和 B 相应列的值。

对第一种参数传递技术在处理主变量数据发生转换错误时(由于数值数据无效)SQLCODE 为 -302 或 -304，对结构参数传递技术，SQL 不检查这个错误，转换错误出现在引用主变量的主语言语句中，例如若一个 DECIMAL（5,2）的输入主变量包含无效数据'FFFFFF'X，当数据移到数据结构时，在主语言中会返回一个错误。

由 SQL 创建结构使用以字母 SQL 开头的名字，如果程序使用以 SQL 开头的变量名，就与 SQL 创建的名矛盾。

SQL 生成的数据结构的内容必须不能由应用程序修改。

22.26 监控数据库查询性能

可以对某个查询或系统中每个查询收集性能统计，有两种方法：

使用 STRDBMON 和 ENDDBMON 命令。

用有 STRDBMON 参数的 STRPFRMON 命令。

可以用这些性能统计产生不同的报表。例如：可以生成包含下列内容的查询报表：

使用充分的系统资源

使用极长时间运行

由于查询时限不能运行

在执行中创建一个临时键字访问路径

在执行时使用查询分类

使用由查询优化建议的键字做索引的逻辑文件能加快运行。详细内容请看 AS/400 DB2 程序设计。

22.27 控制并行处理

本节介绍如何打开和关闭并行处理。若安装了 DB2 对称多处理特性，也打开和关闭 SMP。通过系统值 QQRYDEGREE 对系统权控制来实现的，在作业级上，这个控制可用 CHGQRYA 命令中的 DEGREE 参数实现。

虽然并行对一个系统和给定的作业可用，但在作业中这行的单个查询实际上不使用并行

方式，这或许因为功能限制，或优化选择非并行方式，因为它运行快些。详细内容可看前面有关章节。

由于用并行访问方式处理的查询过份地使用主存、CPU 和磁盘资源，要限制和控制使用并行访问方式查询的数量。

22.27.1 控制并行处理系统权

QQRYDEGREE 系统值可以用来控制系统的并行处理，系统的当前值可用下列命令显示和修改：

WRKSYSVAL—处理系统值
CHGSYSVAL—修改系统值
DSPSYSVAL—显示系统值
RTVSYSVAL—取系统值

QQRYDEGREE 的某些值控制系统中所有作业是否允许使用缺省值，可用的值如下：

*NONE　　对数据库查询处理不允许并行处理
*IO　　对查询可以使用 I/O 并行处理
*OPTIMIZE　查询优化可以选择使用任何数量的 I/O 或 SMP 并行处理任务来进行查询。SMP 并行处理仅当安装了 DB2 对称多处理特性时才能使用。查询优化根据作业共享存储池中的内存来选择使用并行处理以减少逃逸时间。
*MAX　　查询优化可以选择使用 I/O 或 SMP 并行处理进行查询，SMP 并行处理仅当安装了 DB2 对称多处理特性时才能使用。查询优化所做的选*OPTIMIZE 相似，除了优化假设存储池中所有的活动内存都可由查询使用。

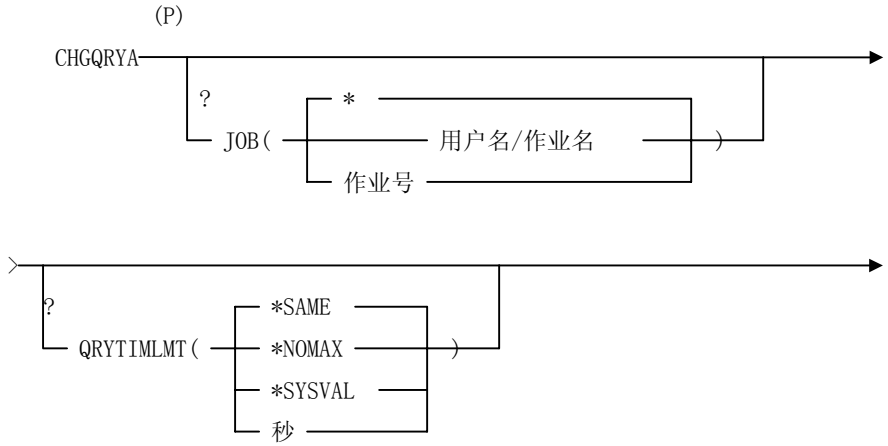
QQRYDEGREE 的缺省值为*NONE，这样若想在系统中运行的作业都使用并行查询处理，就必须修改缺省值。

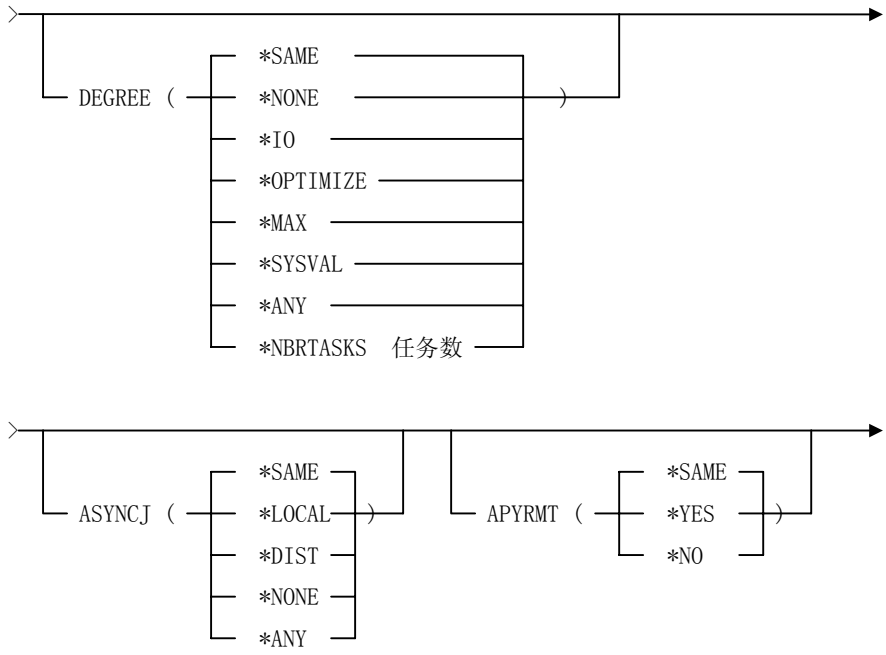
22.27.2 控制一个作业的并行处理

查询并行处理也可以使用 CHGQRYA 命令的 DEGREE 参数来在作业级上控制。并行处理选项允许当运行数据库查询时在作业中规定使用的任务数，可以在交互作业中对 CHGQRYA 命令做提示来显示 DEGREE 查询属性的当前值。

修改 DEGREE 查询属性不影响那些已经开始或使用可重用 ODPs 的查询。

Job: B,I Pgm: B,I REXX: B,I Exec





注：

（1）*ANY 等价于*IO;

（2）在这点以前的所有参数都可用位置格式。

DEGREE 键字的参数值为：

*SAME　　并行级查询属性没有改变。

*NONE　　对数据库查询不允许并行处理。

*IO　　当数据库查询优化选择使用 I/O 并行处理时，可以使用的任务数。不允许 SMP 并行处理。

*OPTIMIZE　　查询优化可以选择使用任意任务数的 I/O 或 SMP 并行处理来做查询。SMP 并行处理仅可在安装了 DB2 对称多处理特性时使用。并行处理及所用的任务数取决于系统中可用处理器的个数。

在作业运行存储池中作业共享的活动内存总数及是否期望由 CPU 处理和 I/O 资源限制查询的逃逸时间，查询优化根据存储池中作业共享的内存来选择最小逃逸时间的实施办法。

*MAX　　此参数与参数值*OPTIMIZE 类似，除了优化假设存储池中所有活动内存可查询中使用。

*NBRTASKS　　任务数查询优化选择使用 SMP 并行处理时规定使用的任务数，也允许 I/O 并行处理。SMP 并行处理仅当安装了 DB2 对称多处理特性时才能使用。使用的任务数比系统中可用的处理器数少时，会限制运行给查询同时使用的处理器数。一个大的任务数保证让查询使用系统中所有可用的处理器来运行查询，太多的任务数会降低性能，因为超出过多活动内存并且过度消耗管理所有任务的资源。

*SYSVAL　　规定使用的处理选项设为 QQRYDEGREE 系统值的当前值。

*ANY　　*ANY 参数与*IO 有相同意义，*ANY 值与前面版本兼容。

作业的 DEGREE 属性初值为*SYSVAL。

有关 CHGQRYA 命令的信息请看 CL 参考。

第二十三章　解决公共数据库问题

这章介绍解决某些公共数据库问题的技术，这些技术有助于你做下列工作：

通过检索数据来分页
用反序检索数据
往表的末尾加数据
更新从表中取出的数据
更新以前取出的数据
修改表定义

23.1　通过检索数据来分页

当程序从数据库取出数据，FETCH 语句允许程序向前通过数据分页。若使用滚动游标，只要在 FETCH 语句中规定滚动选择，程序可以在文件任何处分页，这样就允许程序多次取回数据。数据分页选项在 4.1.2 表中列出。

23.2　反序检索

若对 DEPTNO 每个值仅有一行，下面的语句规定唯一的行序：

```
SELECT * FROM DEPARTMENT
  WHERE LOCATION = 'MINNESOTA'
  ORDER BY DEPTNO
```

要用反序检索同样的行，象下面语句那样规定降序：

```
SELECT * FROM DEPARTMENT
  WHERE LOCATION = 'MINNESOTA'
  ORDER BY DEPTNO DESC
```

在第二个语句中的游标将用与第一个语句游标的相反顺序来检索行，但这要保证第一个语句规定一个唯一顺序。

若在一个程序中需要这两个语句，那么在 DEPTNO 列上建两个索引，一个为升序，另一个为降序。

23.3　表尾位置的建立

对一个滚动游标，表尾由“FETCH AFTER FROM C1”决定，一旦游标定位在表尾，程序使用 PRIOR 或 RELATIVE 滚动选项来定位且从表尾开始取数据。

23.4　往表尾加数据

返回给程序的顺序取决于 SQL 语句中 ORDER　BY 子句，要把数据加到表的末尾，在表定义中要有一个顺序号列，这样在从表中检索数据时，用 ORDER　BY 子句命名这个列。

23.5 从表中检索时更新数据

可以在检索时更新每行数据。在选择语句中，使用 FOR UPDATE OF，接着写要更新的列，然后使用游标控制的 UPDATE 语句。WHERE CURRENT OF 子句命名游标，它指出要更新的列。如果 FOR UPDATE OF，ORDER BY，FOR READ ONLY 或 SCROBL 子句没规定 DYNAMIC 子句，要更新所有的列。

如果规定且运行多行 FETCH 语句，游标定位在块的最后一行上，因此，若在 UPDATE 语句上规定 WHERE CURRENT OF 子句，块的最后一行被更新。若块中的一行必须更新，程序必须首先把游标定位在那行，然后可规定 UPDATE WHERE CURRENT OF。考虑下面的例子：

表 23-1 更新一个表

可滚动游标 SQL 语句	注释
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.	
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.	程序中的 DEPTINFO 和 IND-ARRAY 说明为一个主结构数组和一个指示器数组。
... 确定部门 D11 中的任何一个雇员是否收到小于 $500.00 bonus。如果是，把记录更改为新的$500.00 的最小值。	
EXEC SQL FETCH RELATIVE :NUMBACK FROM THISEMP END-EXEC.	... 定位块中的记录来用与输入相反的顺序来更新。
EXEC SQL UPDATE CORPDATA.EMPLOYEE SET BONUS = 500 WHERE CURRENT OF THISEMP END-EXEC.	... 更新部门 D11 中雇员的 the bonus，它是在新的最小$500.00 之下。
EXEC SQL FETCH RELATIVE :NUMBACK FROM THISEMP FOR 5 ROWS INTO :DEPTINFO :IND-ARRAY END-EXEC.	... 定位在同一块中的起始部分，它已经输入好而且再一次输入块中(NUMBACK -(5 - NUMBACK - 1))
... 返回来确定在块中是否有其他的雇员 bonus 在 $500.00 以下。 ... 返回来输入并处理下一块行。	
CLOSE-THISEMP. EXEC SQL CLOSE THISEMP END-EXEC.	

23.5.1 限制

不能在包含下列任一元素的选择语句中使用 FOR UPDATE OF：

第一个 FROM 语句标识多个表或视图。

第一个 FROM 语句标识只读视图。

第一个 SELECT 语句规定键字 DISTINCT。

外部子选择包含 GROUP BY 子句。

外部子选择包含 HAVING 子句。

第一个 SELECT 语句包含列函数。

选择语句包含一个子查询，它的外部子选择的基本目标与子查询是同一个表。

选择语句包含 UNION 或 UNION ALL 操作。

选择语句包含一个 FOR READ ONLY 语句。

规定 SCROLL 键字但没有 DYNAMIC。

若规定了 FOR UPDATE OF 子句，你不能更新没在 FOR UPDATE OF 子句命名的列，但你可以在 FOR UPDATE OF 子句中命名不在 SELECT 表中的列，例子如下：

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

不能命名比在 FOR UPDATE OF 子句中需要多的列，当访问表时，不能使用这些列的索引。

23.6 更新先前检索的数据

可以做下列之一来翻页和更新先前检索的数据：

使用 UPDATE 语句与 WHERE 子句命名所有行中的列或规定表的唯一键字。可以使用 WHERE 子句中主变量写一个语句，并用变量的不同值运行相同的语句多次来更改不同行。

对一个滚动游标，程序可以使用适当滚动选项检索取回的行，然后用 UPDATE 语句中的 WHERE CURRENT OF 子句，可把行改变为恰当的值。

23.7 修改表的定义

可以用 SQL ALTER TABLE 语句或 CHGPF 命令来增加、去掉或更改表中的列。使用 SQL ALTER TABLE 语句请看 AS/400 DB2 参考。如何使用 CHGPF 命令请看 CL 参考。也可动态地建立一个表的视图，它仅包括你所需要的列，且按要求的顺序出现

第二十四章 分布式关系数据库功能

一个分布式关系数据库由一组 SQL 目标组成，这些目标分散在互连计算机系统中，这些关系数据库可以是同一类型（例如 AS/400 DB2），也可以是不同类型（OS/390 DB2，VM DB2，UDB 或支持 DRDA 的非 IBM 数据库管理系统）。每个关系数据库都有数据库管理系统来管理其环境中的表，数据库管理系统之间用允许的方式互相通讯和操作来运行在另一系统中关系数据库上的 SQL 语句。

应用程序请求支持连接的应用边，应用服务器是本地的或远程数据库用它来连接应用请

求。对 AS/400 DB2 提供分布式关系数据库结构特性（DRDA）支持，它允许应用请求与服务器连接。另外 AS/400 DB2 可以调用已有程序来访问不支持 DRDA 的其它数据库中的其他数据，应用请求驱动程序（ARD）调用这些已存在的程序。

AS/400 DB2 支持两级分布式关系数据库：

远程工作单元(RUW) 远程工作单元是准备和运行 SQL 语句仅出现在一个应用服务器中的工作单元。AS/400 DB2 支持 APPC 或 TCP/IP 上的 RWN。

分布工作单元(DUW) 分布工作单元是准备和运行出现在多个应用服务器中的 SQL 语句的工作单元,但一个 SQL 语句只能引用在一个应用服务器中的目标。AS/400 DB2 仅支持 APPC 上的 DUW。

分布式关系数据库的详细资料，请看分布式数据库程序设计。

24.1 AS/400 DB2 分布式关系数据库支持

DB2 查询管理和 SQL 开发工具特许程序支持用下列 SQL 语句交互地访问分布式数据库：

- CONNECT
- SET CONNECTION
- DISCONNECT
- RELEASE
- DROP PACKAGE
- GRANT PACKAGE
- REVOKE PACKAGE

这些语句的详细描述，请看 AS/400 DB2 SQL 参考。

由开发工具提供的附加支持是通过 SQL 预编译命令的参数实现的：

CRTSQLCI
CRTSQLCBL
CRTSQLCBLI
CRTSQLPLI
CRTSQLRPG
CRTSQLRPGI

关于 SQL 预编译命令的详细内容，请看第十六章。CRTSQLPKG 命令能用分别生成的 SQL 程序来生成 SQL 程序包。CRTSQLPKG 和 CRTSQLxxx 命令语法和参数定义，请看 AS/400 DB2 CL 命令描述。

24.2 AS/400 DB2 分布式关系数据库样板程序

远程工作单元数据库样本程序与 SQL 产品随机器带来，其中在 QSQL 库中有一些文件和成员，可帮助你建立一个可运行分布式 AS/400 DB2 样本程序的环境。

为了使用这些文件和成员，你需要运行在文件 QSQL/QSQSAMP 中的 SETUP 批作业。它允许你做下列事情来定制样本：

在本地和远程生成 QSQSAMP 库。

在本地和远程设置关系数据库目录。

在本地生成应用控制板。

预编译、编译和运行程序来生成样板应用的集合、表、索引和视图。

把数据装入到在本地和远程表中。

预编译和编译程序。

对远程应用程序建立 SQL 程序包。

预编译、编译和运行程序以更新在部门表中的列。

在运行 SETUP 前，要编辑 QSQL/QSQSAMP 文件中的 SETUP 成员，所用指令放在成员的注释中，为运行 SETUP，在 AS/400 命令行上规定下列命令：

```
========> SBMDBJOB QSQL/QSQSAMP SETUP
```

等待批作业完成。

要使用样本程序，在命令行上规定下列命令：

```
========> ADDLIBLE QSQSAMP
```

为调用第一个显示允许定制样本程序，在命令行上规定下列命令：

```
========> CALL QSQ8HC3
```

出现下面的显示，从这个显示，可定制数据库样本程序。

```
*                                                                    *
*                                                                    *
*                                                                    *
* 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............:   ___________________________                     *
*                                                                    *
*                                                                    *
*                                                                    *
*                                                                    *
*                                                  Bottom            *
*                                                                    *
```

```
* F3=Exit                                                              *
*                                    (C) COPYRIGHT IBM CORP. 1982, 1991 *
*                                                                      *
```

24.3 SQL 程序包支持

OS/400 程序支持称为 SQL 程序包的目标。(OS/400 的目标类型为*SQLPKG)。SQL 程序包中包括控制结构和访问方案，它是运行分布程序时处理应用服务器上的 SQL 语句必须的。在下列情况下可以生成 SQL 程序包：

在 CRTSQLxxx 命令中规定 RDB 参数且程序目标成功地生成，SQL 程序包在 RDB 参数中规定的系统上生成。

若预编译没成功或编译仅建立了模块目标，SQL 软件包不能生成。

使用 CRTSQLPKG 命令。如果在预编译时没生成程序包或程序包需要放在不是预编译命令中指定的 RDB 上时，用 CRTSQLPKG 命令。

DLTSQLPKG 命令允许删除本地系统中的 SQL 程序包。

要在远程系统上生成程序包，只有在有一定数据的授权 ID 分配给程序包时，才能生成。要运行程序，在 SQL 程序包中必须包括 EXECUTE 特权的授权。在 AS/400 系统中，EXECUTE 特权包括*OBJOPR 和*EXECUTE 系统权限。

CRTSQLPKG 命令的语法，在 AS/400 DB2 CL 命令描述的附录 D 中。

24.3.1 在 SQL 程序包中有效的 SQL 语句

与另一个 AS/400 连接的程序可以使用在 AS/400 DB2 参考中给出的任一语句，但 SET TRANSACTION 语句除外。使用涉及非 AS/400 DB2 系统的 AS/400 DB2 系统编译的程序，也能使用运程系统支持的可执行 SQL 语句。预编译将继续对它不明白的语句发出诊断信息。在建立 SQL 程序包过程中,这些语句被发送到远程系统。当语句不能在当前应用服务器上运行时，SQL 运行时支持将返回-84 或-525 的 SQLCODE。允许仅在应用请求和应用服务器均是 OS/400 V2.2 或其后时，才能在分布程序中使用多行 FETCH，成块的 INSERT 和滚动游标。详细内容请看 AS/400 DB2 SQL 参考的附录。

24.3.2 生成 SQL 程序包的考虑

当生成 SQL 程序包时，有许多事情要考虑到，下面分别介绍。

24.3.2.1 CRTSQLPKG 权限

当在 AS/400 系统上生成 SQL 程序包时，所用的授权 ID 必须对 CRTSQLPKG 命令有*USE 的权限。

24.3.2.2 生成非 AS/400 DB2 上的程序包

对非 AS/400 DB2 生成程序和 SQL 程序包时，并且想用对关系数据库是唯一的 SQL 语句，CRTSQLxxx 的 GENLVL 参数要设为 30。在信息的严重级别小于 30 时，生成程序，如大于 30，语句对任何关系数据库都无效，例如，没定义和不能用的主变量或常数是无效的且生成比 30 大的信息级别。

当运行比 10 大的 GENLVL 时，将检查预编译清单中的非期望信息。当对 DB2 通用数据库建立一个程序包时，必须设置 GENLVL 参数值小于 20。

若 RDB 参数指定的系统不是 AS/400 DB2 系统，在 CRTSQLxxx 命令中不能使用下列的选

项：

- COMMIT(*NONE)
- OPTION(*SYS)
- DATFMT(*MDY)
- DATFMT(*DMY)
- DATFMT(*JUL)
- DATFMT(*YMD)
- DATFMT(*JOB)
- DYNUSRPRF(*OWNER)
- TIMFMT(*HMS) （ 如果规定了 TIMSEP(*BLANK) 或者 TIMSEP(',') ）
- SRTSEQ(*JOBRUN)
- SRTSEQ(*LANGIDUNQ)
- SRTSEQ(*LANGIDSHR)
- SRTSEQ(库名/表名)

注：与 DB2 通用数据库应用服务器连接时，要考虑下列附加规则：
规定的日期和时间格式必须是同一格式
必须对 TEXT 参数使用*BLANK 的值
不支持缺省集合（DFTRDBCOL）
生成程序包的源程序的 CCSID 必须不是 65535，如果用 65535，则生成空程序包。

24.3.2.3 目的版本（TGTRLS）

当建立程序包时，检验 SQL 语句来确定哪个版本支持这些功能，这个版本作为程序包的重存级。例如，一个程序包包含 CREATE TABLE 语句，并对表加上 FOREIGN KEY 约束，那么程序包重存级将是版本 3.1，这是因为版本 3.1 之前不支持 FOREIGN KEY 约束。当 TGTRLS 参数是*CURRENT 时支持 TGTRLS 信息。

24.3.2.4 SQL 语句尺寸

建立 SQL 程序包不能处理预编译处理的相同尺寸的 SQL 语句，在 SQL 程序预编译期间，SQL 语句被放置在与程序相关的空间中。这时，每个标记被一个空格分开。另外，当规定 RDB 参数时，原语句的主变量用‘H’代替，生成 SQL 程序包功能把这个语句与这些语句的主变量列表一起传达给应用服务器。在标记之间的附加空格和主变量替换可以导致超过 SQL 语句尺寸的最大值（SOL0101 原因码 5）。

24.3.2.5 不需程序包的语句

在一些情况下，你要建立一个 SQL 程序包，但 SQL 程序包没生成，而程序仍运行。这种情况发生在程序仅包含不需要 SQL 程序包运行的 SQL 语句。例如：一个仅包含 DESCRIBE TABLE 的语句的程序，在 SQL 程序包建立中会生成信息 SQL5041。不需 SQL 程序包的 SQL 语句有：

- DESCRIBE TABLE
- COMMIT
- ROLLBACK
- CONNECT

- SET CONNECTION
- DISCONNECT
- RELEASE

24.3.2.6 程序包目标类型

SQL 程序包是非-ILE 目标，并在缺省活动组中运行。

24.3.2.7 ILE 程序和服务程序

ILE 程序和服务程序把几个包括 SQL 语句的模块联编，它必须对每个模块有各自的 SQL 程序包。

24.3.2.8 程序包生成连接

程序包生成的连接类型基于使用 RDBCNNMTH 参数需要的连接类型，若规定 RDBCNNMTH（*DUM），要使用落实控制并且连接是只读连接。若连接是只读的，那么程序包生成失败。

24.3.2.9 工作单元

因为程序包生成隐含一个落实或返回操作，落实的定义在试图生成程序包前必须是在工作单元边界。对一个在工作单元边界的落实定义，下列条件一定全为真：

SQL 是在工作单元边界上。

没有本地或 DDM 打开文件使用落实控制打开，并且没有用未决修改关闭本地或 DDM 文件。

没有 API 资源登录。

没有与 DRDA 或 DDM 无关的 LU6.2 资源登录。

24.3.2.10 生成本地程序包

在 RDB 参数中规定的名字是本地系统名。若它是本地系统名，SQL 程序包将在本地系统上生成。SQL 程序包能保存（SAVOBJ 命令）然后再重存到（RSTOBJ 命令）另一个 AS/400 系统中。当运行与本地体系统连接的程序时，不能使用 SQL 程序包，若规定 RDB 参数为*LOCAL，不会生成*SQLPKG 目标，但程序包信息用*PGM 目标保存。

24.3.2.11 标号

可以使用 LABEL ON 语句对 SQL 程序包建立一个描述。

24.3.2.12 一致性标记

当调用 SQL 程序包时，要检查程序和与它相关的 SQL 程序包是否包含一致性标记。一致性标记必须匹配，否则程序包不能使用，程序和 SQL 程序包可能会出现不协调。假设程序在 AS/400 系统上，而应用服务器在另一个 AS/400 系统中，程序在会话 A 中运行并在会话 B 中重建（在这儿也重生成 SQL 包）。下一次对会话 A 中程序的调用会导致一致性标记错。为避免在每次调用时对 SQL 程序包定位，SQL 维护一个 SQL 包地址表在每个会话中使用。

当会话 B 重建程序包时，旧的 SQL 程序包移至 QRPLOBJ 库中。会话 A 中的 SQL 程序包地址仍然有效（从运行程序的会话生成程序或 SQL 包，或在生成程序前提交一个远程命令来删除老的 SQL 包也能避免发生这种情况）。

要使用新的程序包，应终止与远程系统连接。你能关闭会话然后再启动，或使用 STRSQL

命令对非保护网络连接发出 DISCONNECT，或对保护的连接用 RELEASE 后接 COMMIT，然后用 RCLDDMCNV 终止网络连接，再次调用程序。

24.3.2.13　SQL 和递归

在预编译时从一个注意键激活 SQL 程序，将得到不可预测的结果。

CRTSQLxxx、CRTSQLPKG、STRSQL 命令和 SQL 运行时环境是不递归的，若试图递归会产生
不可预测结果。若命令运行时发生递归（或运行一个嵌入 SQL 语句的程序），作业在命令完成之前被中断，开始另一个 SQL 功能。

24.4　SQL 的 CCSID 考虑

若正运行一个分布应用程序且系统中有一个不是 AS/400 系统，在 AS/400 中作业的 CCSID 的值不能设为 65535。

在请求远程系统生成一个 SQL 程序包之前，应用请求总要把在 RDB 参数中规定的名、SQL 程序包、库名以及 SQL 包的说明从作业的 CCSID 转换成 CCSID 500，这是 DRDA 要求的。当远程关系数据库是 AS/400 系统，名字不用从 CCSID 500 转换或作业 CCSID。

建议不要对表、视图、索引、集合，库或 SQL 程序包名使用定界标识符，在不同 CCSID 系统之间不会发生名字转换。考虑下面的例子：系统 A 用 CCSID37 运行，系统 B 由 CCSID 为 500 运行。

生成一个程序，它在系统 A 上生成名“a　¦c”的表。

在系统 A 上保存程序“a　¦c”，然后把它重存到系统 B 上。

在 CCSID 37 中，空格的码点为 X‘5F’，而在 CCSID 500 它是 X‘AB’。

在系统 B 中名字显示为“a[b]c”，若生成一个程序引用名为“a　¦c”的表，程序就找不到这个表。

在 SQL 目标名中不能使用@，#和$字符，它们的码点依赖所用的 CCSID。若你使用定界名或三种自然扩展，在未来版本中，名字分辨功能很可能失败。

24.5　连接管理和活动组

24.5.1　连接和对话

在由 DRDA 使用 TCP/IP 之前，术语‘连接’是很明确的。它引用一个从 SQL 观点的连接。就是，连接开始于一个 CONNECT TO 某些 RDB，结束于 DESCONNET 完成或 RELEASE ALL 紧跟一个成功的 COMMIT 出现。APPC 对话能否保持，取决于作业的 DDMCNV 属性值和是与 AS/400 还是与其他类型系统对话。

TCP/IP 术语不包含术语‘对话’，但有一个类似的概念，由 DRDA 支持的 TCP/IP，代替使用术语‘对话’，在本书中，多用术语‘连接’，除非特别指出是关于 APPC 对话。于是，现在有两种不同类型的连接读者须注意：上面介绍的 SQL 连接，和代替术语‘对话’的‘网络’连接。

在两类连接之间可能混淆，该词将用‘SQL’或‘网络’来限定，允许读者理解。

SQL 连接在活动组级管理，每个作业中的活动组管理它自己的连接并且这些连接不能跨越活动组共享。对在缺省活动组中运行的程序，连接仍做为 V2.3 以前那样管理。

下面是一个在多活动组中运行的应用程序，它仅用来解释活动组间的相互作用，连接管理和落实控制，它不是一个可取的编码。

24.5.2 PGM1 原码

```
*                                                                *
*        ....                                                    *
*        EXEC SQL                                                *
*          CONNECT TO SYSB                                       *
*        END-EXEC.                                               *
*        EXEC SQL                                                *
*          SELECT ....                                           *
*        END-EXEC.                                               *
*        CALL PGM2.                                              *
*        ....                                                    *
*                                                                *
```

生成 PGM1 程序和 SQL 程序包用的命令：

```
CRTSQLCBL PGM(PGM1) COMMIT(*NONE) RDB(SYSB)
```

24.5.3 PGM2 的源码

```
*                                                                *
*     ...                                                        *
*             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;                                    *
*   ....                                                           *
*                                                                  *
```

生成 PGM2 程序和 SQL 程序包用的命令：

```
CRTSQLCI OBJ(PGM2) COMMIT(*CHG) RDB(SYSC) OBJTYPE(*PGM)
```

24.5.4 PGM3 的源码

```
*                                                                  *
*       ...                                                        *
*        EXEC SQL                                                  *
*          INSERT INTO TAB VALUES(:st1);                           *
*        EXEC SQL COMMIT;                                          *
*       ....                                                       *
*                                                                  *
```

生成 PGM3 程序和 SQL 程序包所用的命令：

```
CRTSQLCI OBJ(PGM3) COMMIT(*CHG) RDB(SYSD) OBJTYPE(*MODULE)
CRTPGM PGM(PGM3) ACTGRP(APPGRP)
CRTSQLPKG PGM(PGM3) RDB(SYSD)
```

PIC22

在这个例子中，PGM1 是一个非-ILE 程序，使用 CRTSQLCBL 命令生成，这个程序在缺省活动组下运行。PGM2 用 CRTSQLCI 命令生成，在系统命名的活动组下运行。PGM3 也用 CRTSQLCI 命令生成，但它在名为 APPGRP 活动组下运行，因为 APPGRP 对 ACTGRP 参数不是缺省值，要分别发出 CRTPGM 命令。

用 CRTPGM 命令后跟 CRTSQLPGK 命令在 SYSD 关系数据库中生成 SQL 程序包目标。在这个例子中，用户没有明显的开始作业级落实定义，SQL 隐含开始落实控制。

1、调用 PGM1，并在缺省活动组中运行。

2、PGM1 与 SYSB 关系数据库连接，并运行 SELECT 语句。

3、PGM1 调用 PGM2，PGM2 在系统命名的活动组中运行。

4、PGM2 不与 SYSC 关系数据库连接，因为 PGM1 和 PGM2 在不同的活动组中，由 PGM2 启动的在系统命名活动组中的连接不能取消由 PGM1 启动的在缺省活动组中的连接，两者连接都是活动的。PGM2 打开游标并且取出、更新一行，PGM2 在落实控制下运行，是在工作单元的中间，不在一个可连接状态。

5、PGM2 调用 PGM3，PGM3 在活动组 APPGRP 中运行。

6、INSERT 语句是在活动组 APPGRP 运行的第一个语句，第一个 SQL 语句导致隐含连接 SYSD 关系数据库。一行插进 SYSD 数据库的 TAB 表中，然后落实插入。在系统命名的活动组中的未决修改不被落实，因为落实控制是由 SQL 用活动组范围的落实启动的。

7、PGM3 结束，并且控制返给 PGM2，PGM2 取和更改另一行。

8、再次调用 PGM3 插入行。在第一次调用 PGM3 时做了隐式连接，它不能在后续调用做，因为活动组不能在调用 PGM2 之间结束。最后，PGM2 处理所有的行并且落实与系统命名活动组有关的工作单元。

24.5.5 对同一个关系数据库的多个连接

若不同活动组与同一数据库连接，每 SQL 连接都有自己的网络连接和自己的应用服务作业。若活动组与落实控制一起运行，在一个活动组落实的修改，不能在其它活动组修改落实，除非使用作业级落实定义。

PIC23

24.5.6 对缺省活动组的隐式连接管理

应用请求程序可与应用服务器隐式连接。隐式 SQL 连接发生在应用请求程序检测到缺省活动组的第一个活动 SQL 程序发出第一个 SQL 语句且下列各项为真时：

发送的 SQL 语句不是有参数的 CONNECT 语句。

SQL 在缺省活动组不是活动的。

对一个分布式程序，隐式 SQL 连接是对在 RDB 参数中规定的关系数据库。对非分布式程序，隐式 SQL 连接是对本地关系数据库。

当 SQL 成为不活动时，SQL 将结束在缺省活动组中的任何活动连接。在下列情况下，SQL 成为不活动：

应用请求检测到处理的第一活动 SQL 程序已经结束且下列情况都为真：

—没有未决 SQL 修改。

—没有用保护连接所做的连接。

—SET TRANSACTION 语句不活动。

—没有用 CLOSQLCSR(*ENDJOB)预编译的程序运行。

假如有未决修改、保护的连接，或者一个活动的 SET TRANSACTION 语句，SQL 放 在结束状态，如果有用 CLOSQLCSR(*ENDJOB)预编译的程序运行，SQL 对缺省活动组保持活动直到作业结束。

SQL 在结束状态且在工作单元结束，这种情况发生在，在 SQL 程序外发出 COMMIT 或 ROLLBACK 命令。

在作业结束。

24.5.7 对非缺省活动组的隐式连接管理

应用请求程序可以与一个应用服务器隐式连接。当应用请求检测到对活动组发出的第一个 SQL 语句不是一个有参数的 CONNECT 语句时，出现隐式 SQL 连接。

对一个分布式程序，隐式 SQL 连接是对在 RDB 参数中规定的关系数据库。对一个非分布式程序，隐式 SQL 连接是对本地关系数据库。

在处理的下列时间，不会出现隐式连接：

当活动组结束时，若落实控制不活动，活动组级落实控制是活动的，或者作业级落实定义是在工作单元边界上。

作业级落实定义是活动的且不在工作单元边界上，那么 SQL 放在退出状态。

SQL 在退出状态，作业级落实定义是被落实或返回。

一个作业结束。

24.6 分布式支持

AS/400 DB2 支持两级分布式关系数据库：

远程工作单元(RUW)

远程工作单元是在准备和运行 SQL 语句仅发生在一个应用服务器中的工作单元。在一个应用请求有一个应用处理的活动组可与一个应用服务连接，在一个或多个工作单元中，运行任何数量的静态或动态 SQL 语句，它引用在应用服务器中的目标。远程工作单元也叫作为 DRDA 级 1。

分布式工作单元(DUW)

分布式工作单元是准备和运行 SQL 语句能发生在多个应用服务器上的工作单元,但一个 SQL 语句仅能引用在一个应用服务器上的目标。分布式工作单元也叫作 DBDA 级 2。

分布式工作单元允许做下列事情：

—更新访问在一个逻辑工作单元的多个应用服务器或者

—对一个应用服务器做更新访问，对多个应用服务器做读访问，它们都使用一个逻辑工作单元。

多应用服务器能否在一个工作单元上更新,取决于在应用请求上和应用服务器上是否有同步点管理以及在应用请求和应用服务器之间有没有两段落实协议支持。

同步点管理是一个系统元件，它在两段落实协议的参与者之间协调落实和返回操作。当运行分布式更新时，在不同系统上操作的同步点管理确保资源达到一个一致状态。由同步点管理使用的协议和流程也叫作两段落实协议。

假如使用两段落实协议，连接是被保护资源，否则连接是非保护资源。

在系统间使用的数据传输协议类型影响网络连接是保护和非保护的，用 OS/400 V4.2，TCP/IP 连接是非保护的，于是他们仅能用一有限方式协调分布式工作单元。

例如，第一个连接是用 TCP/IP 从程序到 AS/400，此时可做更新，但任何后续的连接，即使是用 APPC，也是只读的。

注意，在用交互 SQL 时，第一个 SQL 连接是往本地系统，这样，要想对使用 TCP/IP 的远程系统上做更新，必须先有 RELEASE ALL，接着用 COMMIT 来结束所有的 SQL 连接，然后再用 CONNECT TO 连接远程 TCP 系统。

24.6.1 确定连接类型

当建立一个远程连接时，将使用非保护或保护网络连接。考虑可落实的更新，SQL 连接在建立时，可以是只读的，不可更新的或不知道是否是可更新的。一个可落实的更新是在落实控制下运行的任何插入，删除，更新，或 DDL 语句。若连接是只读的，用 COMMIT(*NONE)的修改仍可运行。在一个 CONNECT 或 SET CONNECTION 后，SQLCA 的 SQLERRD(4)指出了连接的类型，SQLERRD(4)也指出连接使用的是非保护或保护网络连接。规定的值可以是：

1、可落实更新在连接上完成，连接是非保护的。这将发生在下列情况下：

连接是用远程工作单元建立的（RDBCNNMCH(*RUN)），这也包括本地连接和用远程

工作单元的应用请求驱动（ARD），若使用分布工作单元的建立连接，下列均成立：

—连接是非本地的。

—应用服务器不支持分布工作单元。比如，V3.1 之前 OS/400 上的 DB2 应用服务器。

—发出连接的程序的落实控制级不是*NONE。

—与其他应用服务器不存在可完成落实更新的连接，或者所有与应用服务器的连接（它不支持分部工作单元）是只读连接。

—在落实定义的落实控制下没有打开可更新的本地文件。

—对落实定义的落实控制下没有使用不同连接的打开可更新 DDM 文件。

—对落实定义没有 API 落实控制资源。

—对落实定义没有保护连接登录。

若用落实控制运行，对远程连接，SQL 将对远程连接登录一个一段可更新 DRDA 资源或对本地资源和 ARD 连接登录两段可更新的 DRDA 资源。

2、在连接上没有可落实更新完成。连接为只读的，网络连接是非保护的。

对使用远程工作单元连接管理（*RUN）的应用编译，永远不会出现这种情况。

对分布式工作单元应用，仅当在建立连接时下列情况为真才可能发生这种情况：

连接是非本地的。

应用服务器不支持分布工作单元。

下列至少一个为真：

—发出连接的程序落实控制级是*NONE。

—有另外一个连接应用服务器存在，它不支持分布式工作单元且应用服务器可以完成可落实更新。

—另一个对应用服务器的连接存在，它支持分布式工作单元（包括本地）。

—在落实定义的落实控制下有打开可更新的本地文件。

—在落实定义的落实控制下有打开的使用不同连接的可更新 DDM 文件。

—对落实定义没有一段 API 落实控制资源。

—对落实定义有保护连接登录。

若用落实控制运行，SQL 将登录一个一段只读资源。

3、若不知道是否可以完成可落实更新，连接是保护的。

对用远程工作单元连接管理(*RUW)编译的应用程序，永远不会出现这种情况。

对分布式工作单元应用，当建立连接时且下列条件均为真时会出现这种情况：

连接不是本地的。

发出连接的程序落实控制级不是*NONE。

应用服务器支持分布式工作单元和两段落实协议（保护的连接）。

若用落实控制运行，SQL 将登录两段不确定资源。

4、若不知道是否可以完成落实更新，连接是非保护的。

这对远程工作单元连接管理(*RUN)编译的应用程序，永远不会发生这种情况。

对分布工作单元，当建立连接时下列条件均为真才会发生这种情况：

连接不是本地的。

应用服务器支持分布工作单元。

应用服务器不支持两段落实协议或发出连接的程序落实控制级是*NONE。

若用落实控制运行，SQL 将登录一个一段 DRDA 不确定资源。

5、若不知道是否可以完成落实更新且连接是使用分布工作单元的本地连接或是用分布式工作单元的 ARD 连接。

若用落实控制运行，SQL 将登录两段 DRDA 不确定资源。

有关两段和一段资源的详细内容，请看备份和恢复。

下表总结出能引起远程分布式工作单元连接的连接类型，在成功的运行 CONNECT 和 SET CONNECTION 语句后设置 SQLERRD(4)。

表 24-1　连接类型总结				
在落实控制下连接	应用服务器支持两段落实	应用服务器支持分布工作单元	其他可更新的一段资源寄存器	SQLERRD(4)
No	No	No	No	2
No	No	No	Yes	2
No	No	Yes	No	4
No	No	Yes	Yes	4
No	Yes	No	No	2
No	Yes	No	Yes	2
No	Yes	Yes	No	4
No	Yes	Yes	Yes	4
Yes	No	No	No	1
Yes	No	No	Yes	2
Yes	No	Yes	No	4
Yes	No	Yes	Yes	4
Yes	Yes	No	No	N/A *
Yes	Yes	No	Yes	N/A *
Yes	Yes	Yes	No	3
Yes	Yes	Yes	Yes	3
*DRDA does not allow protected connections to be used to application servers which only support remote unit of work (DRDA1). This includes all DB2 for AS/400 TCP/IP connections.				

24.6.2　连接和落实控制限制

当使用落实控制连接时有一些限制，这些限制仅用在试图运行使用落实控制的语句，而不能用在用 COMMIT（*NONE）建立的连接中。

若两段不确定或可更新资源或一段可资源被登录，另一个一段可更新资源不能被登录。

另外，当保护连接不活动和 DDMCNV 作业属性是*KEEP，这些不用的 DDM 连接将导致用 RUW 连接管理编译的 CONNECT 语句失败。

若运行 RUW 连接管理和使用作业的落实定义，有下面一些限制：

若作业做落实定义用在多个活动组中，所有 RUW 连接必须是本地关系数据库。

若连接是远程，仅一个活动组可以使用 RUW 连接的作业级落实定义。

24.6.3　确定连接状态

没有参数的 CONNECT 语句可用来确定当前的连接是否是可更新的或当前工作单元是不时是只读的，值 1 或 2 将返回给 SQLERRD(3)，SQLERRD(3)的值由如下内容确定：

1.　在对工作单元上的连接可完成可更新落实：

当下列之一为真时会发生这种情况：

SQLERRD(4)的值为 1

以下均为真：

—SQLERRD(4)的值为 3 或 5

—不存在不支持能完成可落实更新的分布式工作单元的应用服务器

—下列之一为真：

—第一个落实更新在使用保护连接的连接上完成，在本地数据库或在一个 ARD 程序连接上完成。

—在落实控制下有打开可更新的本地文件。

—有使用保护连接的打开可更新 DDM 文件。

—有两段 API 落实控制资源。

—没发生落实更新。

以下均为真：

—SQLERRD(4)的值为 4

—不存在不支持能完成落实更新的分布式工作单元的应用服务器。

—第一个可落实更新在这个连接上完成或没有落实更新发生。

—没有使用保护连接的打开可更新 DDM 文件。

—没有落实控制下的打开可更新本地文件。

—没有两段 API 落实控制资源。

2. 在这个工作单元的连接没有落实更新可完成。

当下列之一为真时会发生这种情况：

SQLERRD(4)的值为 2。

SQLERRD(4)的值为 3 或 5 且下列之一为真：

—存在仅支持远程工作单元的可更新应用服务器连接。

—第一个落实更新是在使用非保护连接的连接上完成的。

SQLERRD(4)的值为 4 并且下列之一为真：

—存在支持远程工作单元的可更新应用服务器连接。

—第一个落实更新没在这个连接上完成。

—有使用保护连接的打开可更新 DDM 文件。

—有在落实控制下的打开可更新本地文件。

—有两段 API 落实控制资源。

下表总结了如何用 SQLERRD(4)的值来确定 SQLEERD(3)，此时有仅支持远程工作单元应用服务器不可更新连接，并且发生第一个可落实更新。

表 24-2 确定 SQLERRD(3)值的总结

SQLERRD(4)	对应用服务器的远程工作单元存在可更新的连接	发生第一次可落实的更新 *	SQLERRD(3)
1	--	--	1
2	--	--	2
3	Yes	--	2
3	No	没更新	1
3	No	一段	2
3	No	这个连接	1
3	No	两段	1
4	Yes	--	2
4	No	没更新	1
4	No	一段	2
4	No	这个连接	1
4	No	两段	2
5	Yes	--	2
5	No	没更新	1
5	No	一段	2
5	No	这个连接	1
5	No	两段	1

*这列的项如下定义：

没有可更新指出完成了一个非落实更新，没有用保护连接更新的 DDM 文件打开，没有为更新打开的本地文件，没有登录的落实控制 API。

一段指出第一个落实更新用非保护连接完成或为用非保护连接更新打开 DDM 文件。

二段指出在一个两段分布式工作单元应用服务器上完成了一个落实更新，DDM 文件为用保护连接的更新打开，落实控制 API 登录或在落实控制下打开更新用的本地文件。

在 SQLERRD(4)的值是 3、4 或是 5 且 SQLERRD(3)的值是 2 时，如果试图在连接上完成一个落实更新，这个工作单元会是一个返回请求状态。如果是这个状态，仅允许 ROLLBACK 语句，所有其它语句都导致 SQLCODE -918。

24.6.4 分布式工作单元连接考虑

当连接一个分布式工作单元应用时，要注意一些问题，本节加以介绍。

若在多个应用服务器上完成更新并且使用落实控制，更新上所做的所有连接都要使用落实控制。若连接没用落实控制且后来要做落实更新，结果是这个工作单元是只读的连接。

其他非 SQL 落实资源，如本地文件、DDM 文件和落实控制 API 资源，将影响连接的可更新和只读状态。

若对不支持分布式工作单元的应用服务器使用落实控制连接（例如用 TCP/IP 的 V4.2 的 AS/400），那连接是可更新的或只读的，若连接是可更新的，那它是仅可更新的连接。

24.6.5 结束连接

因为远程连接使用资源，不再使用的连接要在可能的情况下立即结束。连接可以显式或隐式的结束。隐式结束请看本书 24.5.6 和 24.5.7 的内容。连接的显式结束可用 DISCONNECT 或 RELEASE 语句后接成功的 COMMIT。DISCONNECT 语句仅可用在使用非保护连接或本地连接上，当运行 DISCONNECT 语句时结束连接。RELEASE 语句可用在保护或非保护连接上，当运行 RELEASE 语句时，连接没有结束，仅把它放到释放状态，在释放状态的连接仍可使用。在 COMMIT 成功运行后连接才结束，ROLLBACK 或不成功的 COMMIT 不能结束在释放状态下的连接。

当建立远程 SQL 连接时，使用（APPC 或 TCP/IP）DDM 网络连接。当 SQL 连接结束时，网络连接被置为不可用状态或丢弃。无论网络连接被丢弃或在不可用状态，都依靠 DDMCNV 作业属性。若作业属性值为 *KEEP 且是与另一个 AS/400 连接，则连接成为不可用，若作业属性值为*DROP 且是与另一个 AS/400 连接则，连接被丢弃，若连接是对非 AS/400，连接总是被丢弃。*DROP 在下列情况下是合适的：

当管理不使用连接的成本很高且不立即使用相关连接。

当运行混合程序，一些用 RUW 连接管理编译而一些用 DUW 连接管理编译，当保护连接存在时，试图到远程位置运行用 RUW 连接管理编译的程序会失败。

当运行使用 DDM 或 DRDA 保护连接时，对非使用保护连接的落实和返回要导致附加的系统开销。

RCLDDMCNV 命令可用来结束所有不再使用的连接。

24.7 分布式工作单元

分布式工作单元允许访问在同一工作单元中的多个应用服务，每个 SQL 语句仅能访问一个申请服务，使用分布式工作单元允许修改在一个工作单元的多个应用服务的落实或返回。

24.7.1 管理分布式工作单元连接

CONNECT、SET CONNECTION、DISCONNECT 和 RELEASE 语句用来管理在 DUW 环境中的连接。当程序使用 RDBCNNMTH(*DUW)预编译时，分布式工作单元 CONNECT 运行，它是缺省值。这种格式的 CONNECT 语句不能取消存在的连接，而是把先前的连接放在暂停状态。在 CONNECT 语句中规定的关系数据库成为当前连接。CONNECT 语句仅能用开始新的连接；若想在已有的

连接之间转换，必须用 SET CONNECTION 语句。因为连接使用系统资源，当他们不再需要时要结束连接。RELEASE 或 DISCONNECT 语句可用于结束连接，为结束连接，RELEASE 语句后要有一个成功的 COMMIT。

下面是使用落实控制的 DUW 环境下运行的 C 程序例子：

```
*                                                                    *
*      ....                                                          *
*       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;                      *
*      /*    连接系统                    */                          *
*      EXEC SQL CONNECT TO LOCALSYS;                                 *
*      EXEC SQL CONNECT TO SYSB;                                     *
*      EXEC SQL CONNECT TO SYSC;                                     *
*      /* 把本地系统作为当前连接                          */          *
*      EXEC SQL SET CONNECTION LOCALSYS;                             *
*      /* 打开游标        */                                        *
*      EXEC SQL OPEN C1;                                             *
*                                                                    *
*      while (SQLCODE==0)                                            *
*         {                                                          *
*          /* 输入第一行          */                                *
*          EXEC SQL FETCH C1 INTO :partnumber,:price;                *
*          /* 更新这行它指出更新已经传播给其他部分         */         *
*          EXEC SQL UPDATE PARTS SET SITES_UPDATED='Y'                *
*                      WHERE CURRENT OF C1;                          *
*          /* 确认部件数据是否在SYSB中          */                  *
*          if ((partnumber > 10) && (partnumber < 100))              *
*             {                                                      *
*              /* 把SYSB作为当前连接且更改价格         */            *
*              EXEC SQL SET CONNECTION SYSB;                         *
*              EXEC SQL UPDATE PARTS                                 *
*                          SET PRICE=:price                          *
*                          WHERE PARTNO=:partnumber;                 *
*              }                                                     *
*                                                                    *
*           /* 确认部件数据是否在 SYSC中         */                 *
```

```
*            if ((partnumber > 50) && (partnumber < 200))                *
*               {                                                        *
*                /* 使 SYSC 作为当前连接并且更改价格          */          *
*                EXEC SQL SET CONNECTION SYSC;                           *
*                EXEC SQL UPDATE PARTS                                   *
*                            SET PRICE=:price                            *
*                            WHERE PARTNO=:partnumber;                   *
*               }                                                        *
*            /* 落实在所有3部分所作的修改              */                *
*            EXEC SQL COMMIT;                                            *
*            /* 设置本地作为当前连接，这样能输入下一行     */            *
*            EXEC SQL SET CONNECTION LOCALSYS;                           *
*           }                                                            *
*        done:                                                           *
*                                                                        *
*        EXEC SQL WHENEVER SQLERROR CONTINUE;                            *
*        /* 释放连接，使它不在可用                                 */     *
*        EXEC SQL RELEASE SYSB;                                          *
*        EXEC SQL RELEASE SYSC;                                          *
*        /* 关闭游标          */                                        *
*        EXEC SQL CLOSE C1;                                              *
*        /* 做另外一个落实，它将结束释放的连接。本地连接由于它没有释放   *
*           它仍然是活动的。 */                                          *
*        EXEC SQL COMMIT;                                                *
*       ...                                                              *
*                                                                        *
```

图 24-4　分布式工作单元程序的例子

在这个程序中,有三个应用服务活动:本地系统LOCALSYS和两个远程系统:SYSB和SYSC。SYSB和SYSC也支持分布式工作单元和两段落实。对每个在此交易中的应用服务,用CONNECT语句初始所有连接使其活动。当使用DUW时，CONNECT语句不能取消连接而是把先前的连接置为暂停状态，在所有应用服务之后做连接。本地连接用SET CONNECTION语句做当前连接，然后打开游标且取第一行数据，接着确定要更新那个应用服务的数据。若SYSB需要更新，SYSB用SET CONNECTION语句做当前连接且运行更新，对SYSC也同样，然后落实修改。因为使用两段落实，它确保在本地系统和两个远程系统能落实修改。因为游标说明为 WITH HOLD,它在落实后仍保持打开。当前的连接改变为对本地系统连接，这样能取出下一行数据，重复这一系列取、更新和落实，直至所有数据都处理完。在取完所有数据之后，释放对两个远程系统的连接，它们不能取消已连接的，因为它们使用保护连接，在释放连接后，发出落实来结束连接，本地系统仍连接并继续处理。

24.7.1.1　检查连接状态

若在只读连接环境下运行，在做落实更新前要检查连接状态，这将避免工作单元进入返

回请求状态。下列 COBOL 例子显示如何检查连接状态。

```
*                                                                *
*          ...                                                   *
*           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'*
*       ...                                                      *
*                                                                *
```

图 24-5　检查连接状态的例子

24.7.2　游标和准备的语句

游标和准备语句的作用域可达到编译单元和连接，编译单元作用域意味着从另外一个编译程序调用的程序不能使用由调用程序打开或准备的游标或准备语句，连接的作用域意味着程序中的每个连接有它自己各自要求的游标或准备语句。

下面的分布式工作单元例子给出如何在两个不同连接中打开同一个游标，导致游标 C1 的两种情况。

```
*                                                                *
*       .....                                                    *
*        EXEC SQL DECLARE C1 CURSOR FOR                          *
*                    SELECT * FROM CORPDATA.EMPLOYEE;            *
*        /* 做本地连接并打开 C1          */                      *
*        EXEC SQL CONNECT TO LOCALSYS;                           *
*        EXEC SQL OPEN C1;                                       *
*        /* 做远程系统连接并打开 C1   */                         *
*        EXEC SQL CONNECT TO SYSA;                               *
*        EXEC SQL OPEN C1;                                       *
```

```
*        /* 做处理      */                                          *
*        while (NOT_DONE) {                                         *
*           /* 从本地系统取得一行数据  */                           *
*           EXEC SQL SET CONNECTION LOCALSYS;                       *
*           EXEC SQL FETCH C1 INTO :local_emp_struct;               *
*           /* 从远程系统取得一行数据   */                          *
*           EXEC SQL SET CONNECTION SYSA;                           *
*           EXEC SQL FETCH C1 INTO :rmt_emp_struct;                 *
*           /* 处理数据  */                                         *
*           .....                                                   *
*        }                                                          *
*        /* 关闭远程系统游标  */                                    *
*        EXEC SQL CLOSE C1;                                         *
*        /* 关闭本地系统游标   */                                   *
*        EXEC SQL SET CONNECTION LOCALSYS;                          *
*        EXEC SQL CLOSE C1;                                         *
*        .....                                                      *
*                                                                   *
```

图 24-6　DUW 程序中游标的例子

24.8　应用请求驱动程序

为补充由 DRDA 实施产品提供的数据库访问，AS/400 DB2 提供一个接口，来在 AS/400 DB2 应用请求上写退出程序来处理 SQL 请求，这样的退出程序称为应用请求驱动。在下列操作中，AS/400 调用 ADR 程序：

在使用 CRTSQLPKG 或 CRTSQLxxx 命令生成程序包期间，在 RDB 参数符合 ARD 程序中的相应 RDB 名。

当前的连接是对 ARD 程序的相应 RDB 名时处理的 SQL 语句。

这些调用允许 ARD 程序把 SQL 语句和有关语句信息传给远程关系数据库，并把结果返回给系统，然后系统把结果返给应用程序或用户。由 ARD 程序所做的对关系数据库的访问很类似于在不同环境中对 DRDA 应用服务的访问。

有关应用请求驱动程序的详细信息，请看系统 API 参考。

24.9　问题处理

捕捉和报告 AS/400 分布式数据库功能错误信息的主要对策叫做第一次失败数据捕捉(FFDC)。FFDC 支持的目的是提供用 OS/400　DDM 检测到的准确的错误信息，由此可生成 APAR(1)，由此功能键结构和 DDM 数据流自动转储到假脱机文件中。错误信息的前 1024 字节也记录在系统错误记录中。FFDC 在 OS/400 应用请求和应用服务功能上均为活动的。但对记录的 FFDC 数据，第一次发生错误的错误信息自动转储意味着故障没再生成且由客户报告。QSFWERRLOG 必须设为*LOG。

注：不是所有负的 SQLCODE 都转储，仅转储那些能用来产生 APAR 的内容。详细内容请看分部式数据库问题确定手册。

当检查到一个 SQL 错误时，SQLCODE 和相应的 SQLSTATE 一起返回给 SQLCA，详细信息请

看本书原文的附录 B。

(1) APAR 为授权问题分析报告。