

大型计算机

应用技术培训

DB2 应用开发

目录

第一部分	关系数据库与 SQL 语言	1
第一章	关系数据库的基本概念	2
1-1	数据库对象	3
1-2	存储对象	8
1-3	约束	11
1-4	数据完整性	14
第二章	操作表和视图	17
2-1	数据库使用的数据类型	18
2-2	创建表 CREATE TABLE	21
2-3	CREATE VIEW	29
2-4	删除表或视图	30
2-5	添加数据	31
2-6	更改数据	33
2-7	删除数据	35
2-8	创建索引	36
2-9	提交与回滚	38
第三章	数据查询	39
3-1	示例数据表	40
3-2	基本查询语句	43
3-3	查询语句中使用函数	51
3-4	从一个以上表中查询数据	55
3-5	使用子查询	59
第四章	使用 DB2 编目	61
4-1	什么是 DB2 编目	62
4-2	SYSIBM.SYSCHECKS 表	63
4-3	SYSIBM.SYSCOLUMNS 表	64
4-4	SYSIBM.SYSDATABASE 表	67
4-5	SYSIBM.SYSFOREIGNKEYS 表	68
4-6	SYSIBM.SYSINDEXES 表	69
4-7	SYSIBM.SYSKEYS 表	71
4-8	SYSIBM.SYSTABLES 表	72
第二部分	关系数据库应用编程	74
第五章	基本 SQL 语言编程	75
5-1	嵌入式 SQL	76
5-2	声明表	77
5-3	声明宿主变量	78
5-4	使用宿主变量	80
5-5	使用指示变量	81
5-6	使用宿主结构	82
5-7	检查 SQL 语句执行结果	83
5-8	一个简单的嵌入 SQL 程序	85
第六章	使用 CURSOR 读取多行数据	87

6-1	CURSOR 的功能	88
6-2	使用 CURSOR 的一个例子	89
6-3	CURSOR 的语法	90
第七章	动态 SQL 语言程序设计	92
7-1	什么是动态 SQL	93
7-2	使用 EXECUTE IMMEDIATE 动态执行	94
7-3	使用 PREPARE 和 EXECUTE 动态执行	95
7-4	固定列动态 SELECT	96
7-5	不定列动态 SELECT	98
第八章	SQL 程序的准备	105
8-1	SQL 程序准备过程	106
8-2	COBOL 语言程序准备	107
8-3	C 语言程序准备	110
8-4	BIND 命令详解	111
8-5	DB2 批量程序的执行	119
第九章	DB2I 的使用	120
9-1	DB2I 介绍	121
9-2	DB2I 参数设置	122
9-3	SPUFI	124
9-4	DCLGEN	128

课程介绍

DB2 应用开发

目的：

通过本课程的学习，使学员能够掌握关系数据库 SQL 语言和 DB2 应用编成所需要的各种技术，能够根据应用的业务逻辑编写适当的 SQL 语句进行处理，并能够通过程序控制数据库中的数据。

主要内容：

- ✓ 关系数据库的基本概念
- ✓ 操作表和视图
- ✓ 数据查询
- ✓ 使用 DB2 编目
- ✓ 基本 SQL 语言编程
- ✓ 使用 CURSOR 读取多行数据
- ✓ 动态 SQL 语言程序设计
- ✓ SQL 程序的准备
- ✓ DB2I 的使用

预修课程：

IBM 大型计算机基本操作

COBOL 语言程序设计

长度：

3 天

教程作者：温洪涛 venn@sina.com

第一部分

关系数据库与 SQL 语言

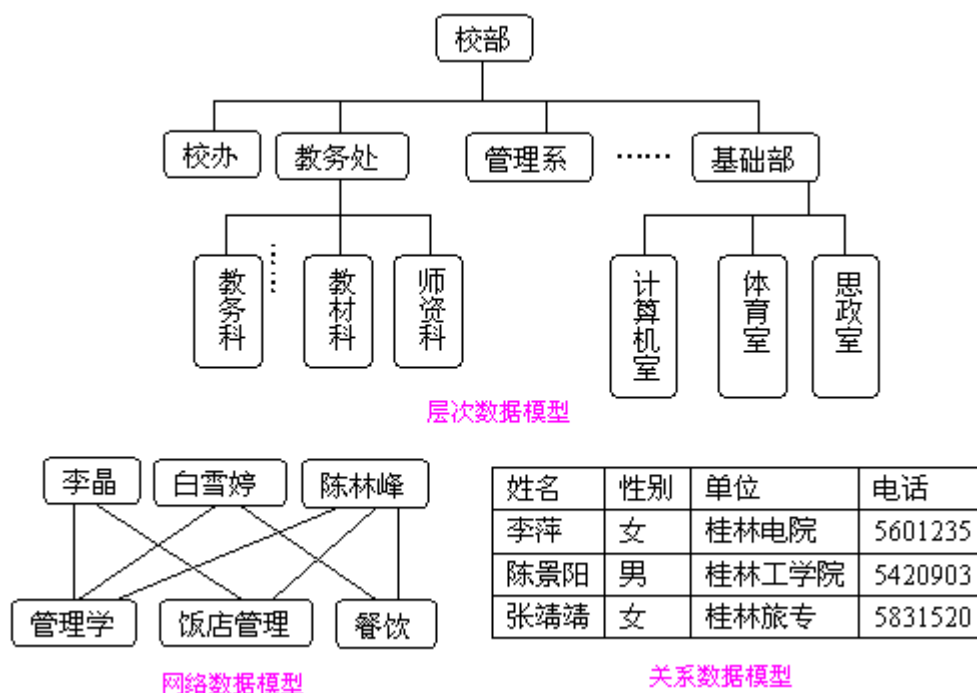
第一章 关系数据库的基本概念

- 数据库对象
- 存储对象
- 约束
- 数据完整性

1-1 数据库对象

- 关系数据库
- 表 (TABLE)
- 视图 (VIEW)
- 索引 (INDEX)

1-1-1 关系数据库



数据库按照其数据模型可分为层次型、网络型和关系型三种

(1) 层次数据模型：用于反映事物间的一对多（1：n）的联系

层次数据模型也称树型，其结构就像一棵倒挂的树。例如描述一个单位的组织机构就可用层次数据模型。

(2) 网络数据模型：用于反映事物间的多对多（n：n）的联系

网络数据模型相对比较复杂。例如一个老师上多门课、一门课可由多个老师上，则老师和课程的关系就是网络型数据模型。

(3) 关系数据模型：用二维表（也称关系）形式来表示事物间的联系

关系数据模型比较常见，其中二维表的行称为记录，列称为字段。关系数据模型有以下三个特性：

一个二维表中所有的记录格式和长度都相同；

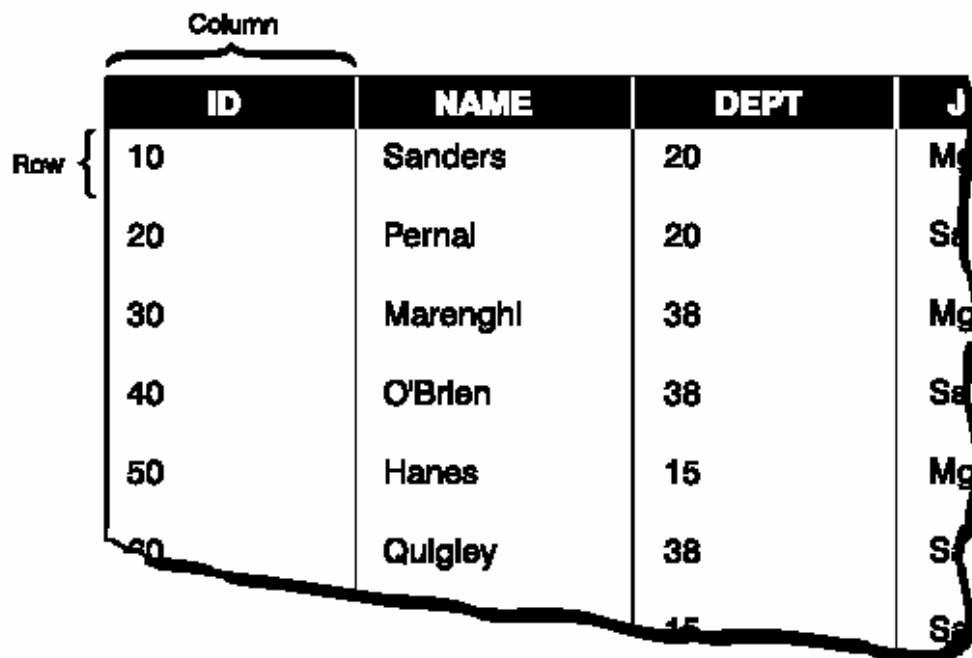
同一字段的类型相同；

行和列的排列顺序随意。

由于关系型数据库逻辑简单,使用和管理比较方便,而且层次型、网络型数据模型都可以用关系型数据模型来表达（通过关联的方法），因此目前流行的数据库都是关系型数据库。

关系数据库数据的查询和维护使用 SQL(Struct Query Language) 语言进行，它由 IBM 在 70 年代开发出来，现在 SQL 语言已经成为一个工业标准，在各种关系型数据库中得到了应用。

1-2-2 表 (TABLE)



The diagram shows a table with four columns and six rows. A bracket above the columns is labeled 'Column' and a bracket to the left of the rows is labeled 'Row'.

ID	NAME	DEPT	J
10	Sanders	20	Me
20	Pernal	20	Se
30	Marengli	38	Mg
40	O'Brien	38	Se
50	Hanes	15	Mg
60	Qulgley	38	Se

表是关系型数据库的基本数据存储逻辑结构, 由定义的列数和可变的行数组成。列是一组相同数据类型的值。存放在表中的行上下间没有逻辑关系。要对结果集进行排序, 必须从表中选择数据的 SQL 语句中显式指定排序。在每个列和行的相交处是一个称为值的特定数据项。

基表是用 CREATE TABLE 语句创建的, 用于保存用户数据。结果表是一组行, 数据库管理程序从一个或多个基表选择或生成这组行以满足查询要求。

1-3-3 视图 (VIEW)

表

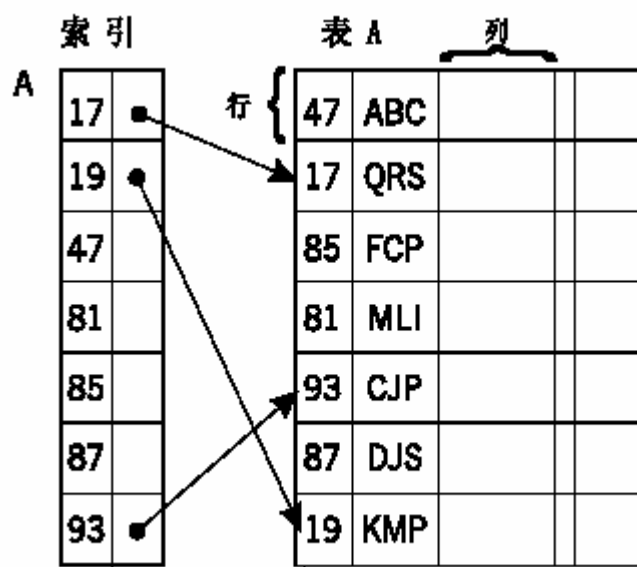
帐号	姓名	余额	密码	状态
00001	张三	100.00	\$i (9C#F	1
00002	李四	2352.10	IL\$23*\$	1
00003	王五	300.00	d*FIH34	3
00004	赵六	10.00	F23=8&(2
00005	韩七	34.12	sd*N2#S,	9

视图

帐号	姓名	余额
00001	张三	100.00
00002	李四	2352.10
00003	王五	300.00
00004	赵六	10.00
00005	韩七	34.12

视图提供了在一个或多个表中查看数据的替代方法,它是表上的一个动态窗口,视图本身并不存储数据,基本数据存储在表里面。每个视图也有一个自己的名字,对用户来说各种操作都与一个表相同。视图允许多个用户查看同一数据的不同表示,可以使用它们来控制对敏感数据的存取,不同的人可以存取数据的不同列或行。

1-4-4 索引 (INDEX)



索引是数据库为加快数据查找速度而建立的一种逻辑实体。索引以在数据表里一定的列作为键值，建立一个可以快速进行查找的数据结构，通过索引查找到键值，就能按照指向数据行的指针快速地定位数据行。

索引有主键索引、唯一索引、非唯一索引等多种，索引可以以数据表的一个或多个列的值作为键值，对一个数据表可以建立一个或多个索引。

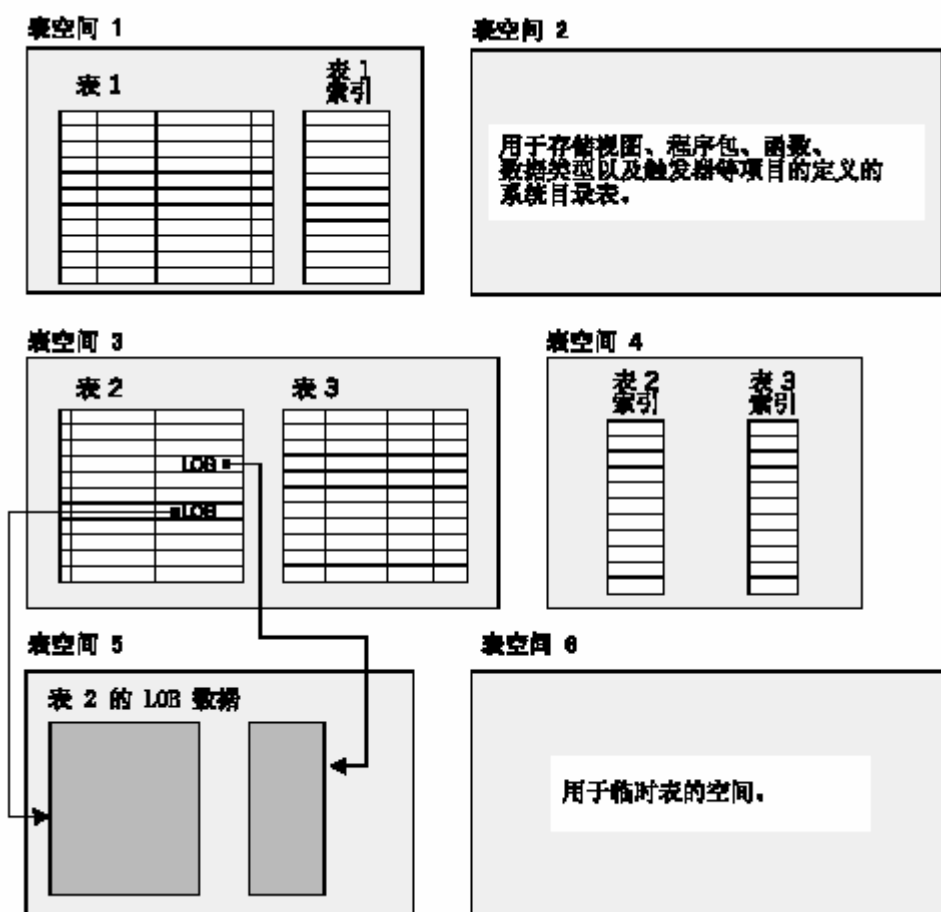
索引建立后，它的使用是透明的，系统会根据查询语言的条件，自动地选取合适的索引使用。

1-2 存储对象

➤ 表空间 (TABLESPACE)

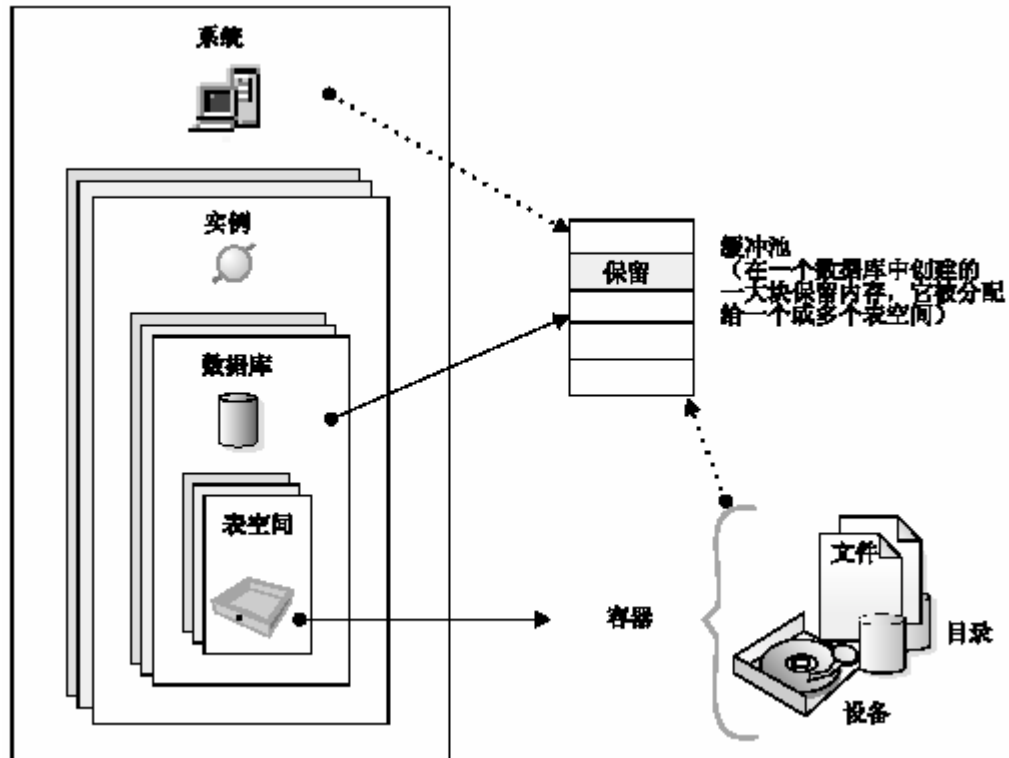
➤ 缓冲池 (BUFFERPOOL)

1-2-1 表空间 (TABLESPACE)



数据库由若干表空间组成，表空间用来存放数据表和索引等实体，可以看作是数据表的容器。建立表空间时制定它在物理存储器上的位置和大小，建数据表时制定存放它的表空间。一个表空间可以存放一个或多个实体，通常在数据库里建立多个表空间，来存储不同类型的数据。

1-2-2 缓冲池 (BUFFERPOOL)



缓冲池指的是从磁盘读取高速缓存表和索引数据页时或修改他们时分配给它们的主存储器。缓冲池的目的是为了改进系统性能，从内存存取数据要比从磁盘存取快得多。有了存储池 DBRMS 磁盘 I/O 次数大大减少，能有效地缩短 I/O 时间。

1-3 约束

- NOT NULL 约束
- UNIQUE 约束
- CHECK 约束
- 主键约束
- 外键约束

1-3-1 NOT NULL 约束，UNIQUE 约束和检查约束（CHECK）

数据库系统中建立数据表时可以根据数据的业务逻辑，对各列数据建立一定的约束条件。D B R M S 在数据插入和更新时自动进行约束检查。凡是违反约束条件的操作将被禁止。

主要约束类型有：

NOT NULL 约束：

空约束只是简单地声明一个字段必须不能是空值，在建表时在指定列上声明。非空约束在功能上等效于创建一个检查约束 CHECK (column_name IS NOT NULL)，但创建一个明确的非空约束效率更高。在大多数数据库设计里，主要的字段都应该标记为非空。

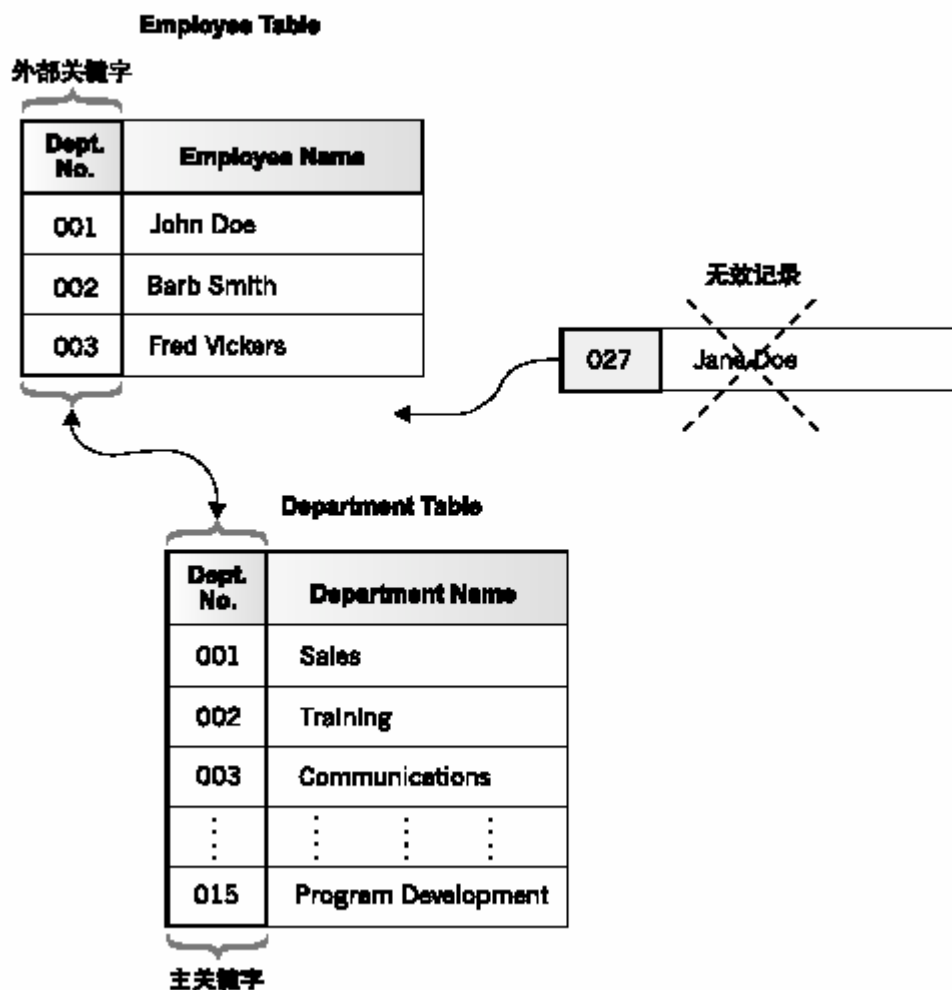
唯一约束：

唯一约束保证在一个字段或者一组字段里地数据与表中其它行的数据相比是唯一的。例如部门表里的部门号是唯一的，且不能为空。

检查约束：

检查约束事最常见的约束类型。它允许你声明在某个字段里的数值必须满足一个任意的表达式。比如，要强制一个正数的产品价格。你还可以给这个约束一个独立的名称，这样就可以令错误信息更清晰，并且在可以用这个名字去修改它。一个检查约束也可以引用若干个字段。

1-3-2 主键/外键 (PRIME KEY/FOREIGN KEY)



主键表示表中一个字段或者是若干个字段的组合可以用于表中的数据行的唯一标识。一个表最多可以有一个主键。依照关系型数据库理论，通常每个表都必须有一个主键，虽然这并不是强制规则，但一般情况下最好还是遵循它。

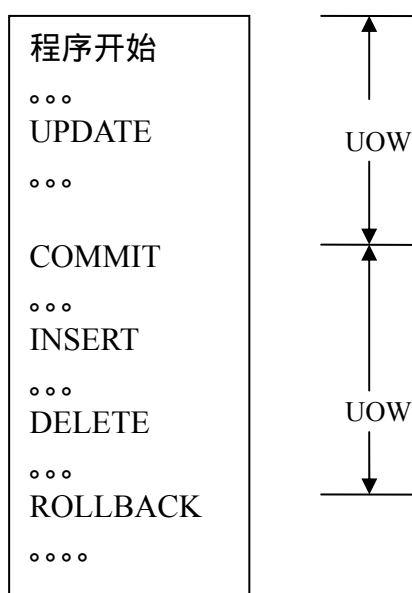
外键约束声明一个字段(或者一组字段)的数值必须匹配另外一个表中某些行出现的数值。 我们把这个行为称做两个相关表之间的参考完整性。

如上图就是比较典型的键字约束，他规定了每个雇员必须隶属于一个存在的部门。

1-4 数据完整性

- 程序控制的数据完整性
- 系统提供的数据完整性

1-4-1 程序控制的数据完整性



数据库操作中把从程序开始或上一次提交（COMMIT）或会滚（ROLLBACK）到下一次提交（COMMIT）或会滚（ROLLBACK）之间的所有操作作为一个工作单元，这个工作单元里所做的一切数据操纵操作被看作一个整体，这个整体可以由程序控制进行提交，把所有操作固定在数据库里；也可以由程序控制进行回滚，使涉及的所有数据恢复到工作单元执行前的状态，就像一切都没有发生过一样。这样程序处理时就可以把需要保持一致的若干项操作组织在一个工作单元中，以保证这些操作如果成功就者完全成功，如果失败就能恢复到像什么都没有发生过一样。

DB2 系统通过 LOG 机制来实现数据完整，每次变动数据的操作，都在专门的 LOG 文件中记录操作前和操作后记录的数据内容，一旦需要会滚，系统依照 LOG 中的记录来恢复数据。

1-4-2 系统控制的数据完整性

若运行 DB2 的系统发生故障或其他意外情况导致突然 DB2 系统突然停止，当 DB2 重启时会根据 LOG 数据对数据表进行恢复以保证中断时正在处理的数据的完整性。对于中断时尚未 COMMIT 的数据，进行自动回滚，对于已经 COMMIT 但尚未更新表的数据，根据 LOG 里的内容把数据改成最新状态。这样就能完整地保证 UOW 的完整性，不论正常运行还是系统失败时，都不会造成 UOW 的不完整。

第二章 操作表和视图

- 数据库使用的数据类型
- 创建表
- 创建视图
- 删除表或视图
- 添加数据
- 更改数据
- 删除数据
- 创建索引

2-1 数据库使用的数据类型

- 定长字符串
- 变长字符串
- 定长图形字符串
- 变长图形字符串
- 二进制字符串
- SMALLINT
- INTEGER
- REAL
- DOUBLE
- DECIMAL(p, s)
- 日期
- 时间
- 时间戳记

数据类型

数据类型是定义常数、列、宿主变量、函数、表达式以及专用寄存器可接受的值。DB2 支持的数据类型主要有：

字符串

字符串为一个字节序列。字符串的长度为序列中的字节数。如果长度为零，则该字符串的值称为空字符串。包括：

定长字符串：

CHAR(x)是定长字符串。长度属性 x 必须在 1 和 254 之间，并包括 1 和 254。

变长字符串：

变长字符串有三种类型：VARCHAR、LONG VARCHAR 以及 CLOB。VARCHAR(x)类型是变长字符串，因此，可以将长度为 9 的字符串插入 VARCHAR(15)中，而该字符串的长度将仍然为 9。

定长图形字符串：

GRAPHIC(x)是定长字符串。长度属性 x 必须在 1 和 127 之间，并包括 1 和 127。

变长图形字符串：

变长图形字符串有三种类型：VARGRAPHIC、LONG VARGRAPHIC 以及 DBCLOB。

二进制字符串

二进制字符串是一个字节序列。它用于保存非传统数据，如图象等。“二进制大对象”(BLOB)是二进制字符串。参见大对象 (LOB)以了解更多信息。

数字：

所有数字都有符号和精度。精度是除符号位以外的位数或数字数。

SMALLINT：

SMALLINT (小型整数) 是精度为 5 位的两字节整数。

INTEGER：

INTEGER (大型整数) 是精度为 10 位的四字节整数。

REAL：

REAL (单精度浮点数) 是实数的 32 位近似值。

DOUBLE：

DOUBLE (双精度浮点数) 是实数的 64 位近似值。DOUBLE 也称 FLOAT。

DECIMAL(p, s)

DECIMAL 是一个十进制数。小数点的位置由数字的精度(p)和小数位(s) 确定。精度是数字的总位数，必须小于 32。小数位是小数部分数字的位数且总是小于或等于精度值。如果未指定精度和小数位，则十进制值的缺省精度为 5，缺省小数位为 0。

日期时间值：

日期时间值是日期、时间以及时间戳记的表示。日期时间值可以用于某些算术运算和字符串运算并且与某些字符串是相容的，然而它们既非字符串也非数字。包括：

日期(DATE)：

日期值分为三个部分 (年、月以及日)。

时间(TIME)：

时间是用 24 小时制式来指定一天内的时间的值，分为三个部分 (小时、分钟以及秒)。

时间戳记(TIMESTAMP)：

时间戳记为指定日期和时间的值，分为七个部分 (年、月、日、小时、分钟、秒以及微秒)。

空值(NULL)：

空值是一个区别于所有非空值的特殊值。它意味着行中的那一列无任何其他值。所有数据类型都存在空值。

2-1 数据库使用的数据类型（续页 2）

下表显示各种数据类型的特性。所有数字数据类型都定义在某一确定范围内。该数字数据类型范围也包括在此表中。可以使用此表作为正确数据类型用法的快速参考。

数据类型	类型	特性	示例或范围
CHAR(15)	定长字符串	最大长度为 254	'Sunny day '
VARCHAR(15)	变长字符串	最大长度为 4000	'Sunny day'
SMALLINT	数字	长度为 2 字节精度为 5 位	范围为-32768 至 32767
INTEGER	数字	长度为 4 字节精度为 10 位	范 围 为 -2147483648 至 2147483647
REAL	数字	单精度浮点 32 位近似值	范 围 为 -3.402E+38 至 -1.175E-37 或 1.175E-37 至-3.402E+38 或零
DOUBLE	数字	双精度浮点 64 位近似值	范 围 为 -1.79769E+308 至 -2.225E-307 或 2.225E-307 至 1.79769E+308 或零
DECIMAL(5, 2)	数字	精度为 5 小数位为 2	范 围 为 -10**31+1 至 10**31-1
DATE	日期时间	三部分值	1991-10-27
TIME	日期时间	三部分值	13.30.05
TIMESTAMP	日期时间	七部分值	991-10-27-13.30.05.000000

2-2 创建表 CREATE TABLE

- 创建表
- CREATE TABLE 语法
- 列定义子句
- 约束定义子句
- CREATE TABLE 的一些例子

2-2-1 创建表

例：

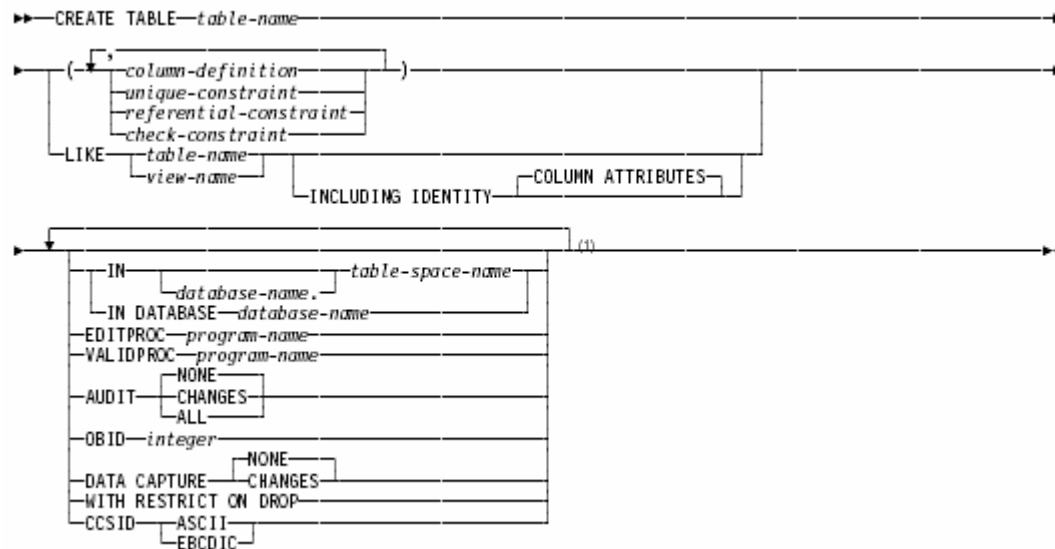
```
CREATE TABLE EMPLOYEE
( ID          SMALLINT      NOT NULL,
  NAME        VARCHAR(9),
  DEPT        SMALLINT      WITH DEFAULT 10,
  JOB         CHAR(5),
  YEARS       SMALLINT,
  SALARY      DECIMAL(7,2),
  BIRTH_DATE  DATE)
```

ID	NAME	DEPT	JOB	YEARS	SALARY	BIRTH_DATE
1	TOM	1	SELLS	5	6000.00	1975-01-12
2	JACK	1	SELLS	4	6000.00	1979-03-24
3	TONY	2	ENGINEER	6	8000.00	1974-12-03
4	TED	2	ENGINEER	8	8500.00	1971-11-23

CREATE TABLE 命令用来创建数据库表，如上例所示，就建立了一个名为 EMPLOYEE 的数据库表，这个表有 7 个字段，可以用来存储如上所示的数据。

CREATE TABLE 命令有丰富的选项，用户可以通过这些选项来灵活地建立所希望的数据表。详细语法如下：

2-2-2 CREATE TABLE 语法



CREATE TABLE 命令常用选项介绍：

table-name：

要建立的表名，不能与系统中其他表、视图、别名相同。

column-definition：

列的定义，详细展开见后。

unique-constraint：

唯一约束定义，详细展开见后。

referential-constraint：

引用约束（主键/外键）定义，详细展开见后。

check-constraint：

检查约束定义，详细展开见后。

LIKE table-name or view-name：

建立一个与另一个表或 VIEW 的字段名称和描述完全相同的一个表

INCLUDING IDENTITY COLUMN ATTRIBUTES：

在建立与另一个表相同的字段时，原表上的 identity column 属性也照样建立。

IN database-name.table-space-name or IN DATABASE database-name：

指明把表建立在那个 TABLESPACE 或那个 DATABASE 上，默认值是建在系统创建的 DSNDB04 数据库里。

2-2-2 CREATE TABLE 语法 (续页)

EDITPROC program-name :

指定编辑例程序名，编辑例程在表上有 LOAD、INSERT、UPDATE 操作时生效，它接到要操作的整行数据，可以进行必要的处理后，再实际存入数据表中。

VALIDPROC program-name :

指明效验程序程序名，效验程序是在数据进入表之前得到控制，它接收到整行数据，可以根据一定的逻辑来判断数据是否合法，是否允许进表。

AUDIT :

指明对表进行的审计操作，NONE: 不进行审计，CHANGE : 仅对对数据表的修改进行审计，ALL : 对表的全部操作进行审计。

DATA CAPTURE :

指明 INSERT、UPDATE 和 DELETE 操作记日志数据(logging)时是否记录附加数据，NONE : 不记录，CHANGES : 记录。

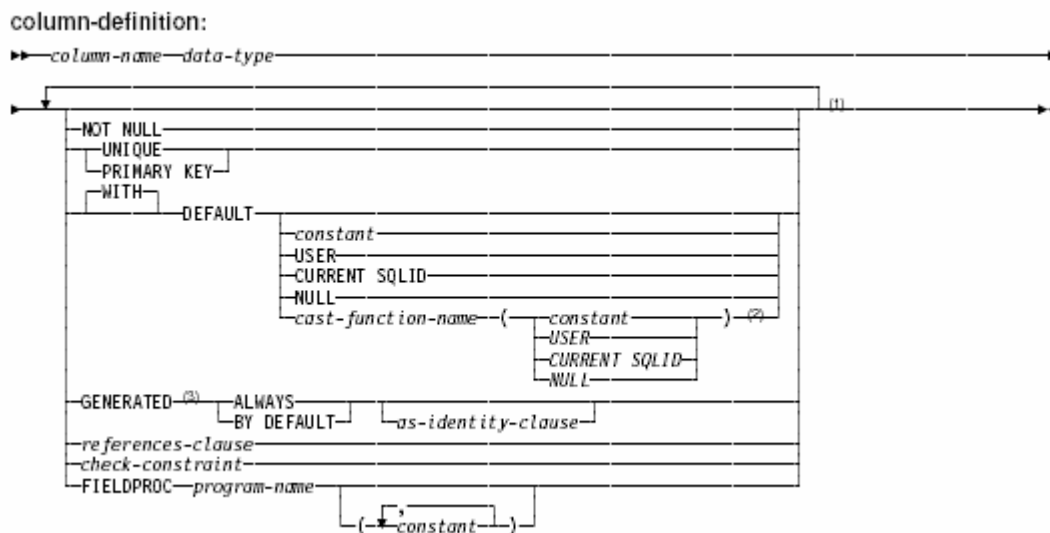
WITH RESTRICT ON DROP :

指明这个表不能被删除，因此，它所在的 TABLESPACE 和 DATABASE 也不能删除。

CCSID encoding-scheme :

指明存放在这个表中的数据使用什么样的数据编码，ASCII : 存储 ASCII 编码的数据，EBCDIC : 存储 EBCDIC 编码的数据。

2-2-3 列定义子句



column-definition 是表中列 (COLUMN) 类型的定义语句 , 这里详细地介绍各种选项的意义 :

column-name :

指定列名 , 同一个表中不能存在相同的列名。

built-in-data-type :

指定列使用哪种系统定义的数据类型 , 可用的选项有 INTEGER、SMALLINT、FLOAT(integer)、REAL、DOUBLE、FLOAT、DECIMAL(integer, integer)、DECIMAL(integer)、CHARACTER(integer)、VARCHAR(integer)、CLOB(integer [K|M|G])、BLOB (integer [K|M|G])、GRAPHIC(integer)、VARGRAPHIC(integer)、DBCLOB(integer [K|M|G])、DATE、TIME、TIMESTAMP 等数据类型。

NOT NULL :

指定这个字段不允许为空。

PRIMARY KEY :

这是定义一个只有一个列的主键的简单写法 , 例如在列 C 后加 PRIMARY KEY , 和 PRIMARY KEY(C) 语句有相同的效果。指定 PRIMARY KEY 的列必须同时指定 NOT NULL 选项。

UNIQUE :

这是定义一个只有一个列的唯一约束的简单写法 , 例如在列 C 后加 UNIQUE 和 UNIQUE(C) 语句有相同的效果。指定 UNIQUE 的列必须同时指定 NOT NULL 选项。

2-2-3 列定义子句 (续页)

DEFAULT :

指定列的默认值，当插入数据时没有指定一个列的值，将自动使用默认值写进表里。

DEFAULT 值可以是：常量：符合字段描述的常量；USER：使用用户名（8 字节）作为默认值，制对字符型列有效；CURRENT SQLID：用当前 SQLID（8 字节）作默认值；NULL：使用空值作默认值。

GENERATED ALWAYS/BY DEFAULT AS IDENTITY :

指定此列是一个系统自动生成序号的“identity column”列。

references-clause :

对于只有一个列的外部键约束的一种简单定义方法。

check-constraint :

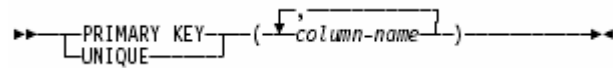
对于只涉及一个列的检查约束的一种简单定义方法。

FIELDPROC program-name :

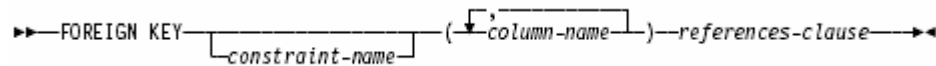
指定列处理函数名。

2-2-4 约束定义子句

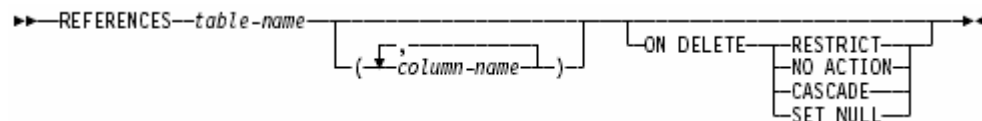
unique-constraint:



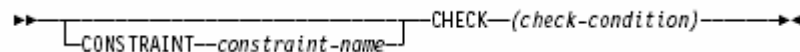
referential-constraint:



references-clause:



check-constraint:



PRIMARY KEY(column-name, ...):

定义一个主键约束，由一个或多个列的联合组成，这些列必须定义 NOT NULL 属性。一个表只能有一个主键，整个主键在表中不能重复。DB2 自动地为 PRIMARY KEY 定义一个主键索引，并通过这个索引来保证唯一性。

UNIQUE:

定义唯一约束，可以由一个或多个列的联合组成，这些列必须定义 NOT NULL 属性。一个表可以有多个 UNIQUE 约束。

FOREIGN KEY constraint-name (column-name, ...)

定义一个 FOREIGN KEY 约束。FOREIGN KEY 约束可以指定一个名字，如果没有指定，系统会自动生成一个。

REFERENCES table-name (column-name, ...)

指明 FOREIGN KEY 的参考对象，table-name 指明 FOREIGN KEY 所参考的表明，(column-name, ...) 指明 FOREIGN KEY 中各个列所参照的列名。

ON DELETE:

指明有参考关系的列删除时的规则。如果表 1 的列对表 2 的列有参照关系，当删除表 2 的某行时，若果表 1 里有参照到表 2 这一行的记录，那么 RESTRICT 或 NO ACTION，对表 2 行的删除失败；CASCADE：同时把表 1 中参考到表 2 这行的行删除；SET NULL：把表 1 里所有参考到表 2 这一行的记录，参考列值设置成 NULL。

CONSTRAINT constraint-name:

给 CHECK 约束起个名字，如果没有指定，系统将自动生成一个。

CHECK (check-condition):

指定一个检查表达式，可以用列名和常数进行运算，只有符合条件时数据操作才会成功

2-2-5 CREATE TABLE 的一些例子

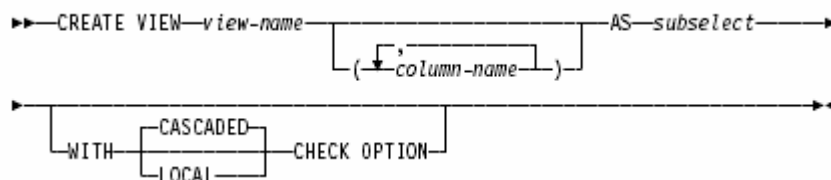
```
CREATE TABLE BOOK (
    ARTICLE_NO          INTEGER          PRIMARY KEY,
    AUTHOR_FIRST_NAMES  CHARACTER(30),
    AUTHOR_LAST_NAMES   CHARACTER(30),
    TITLE               CHARACTER(30),
    ISBN                CHARACTER(13)    UNIQUE,
    WHOLESALE_PRICE      NUMERIC(4, 2),
    RETAIL_PRICE         NUMERIC(4, 2),
    COPIES_AVAILABLE    INTEGER
);

CREATE TABLE CUSTOMER (
    CUSTOMER_NO          INTEGER          PRIMARY KEY,
    FIRST_NAMES          CHARACTER(30),
    LAST_NAMES           CHARACTER(30),
    STREET               CHARACTER(30),
    HOUSE_NO             SMALLINT,
    POSTCODE             CHARACTER(7),
    TOWN                 CHARACTER(30),
    ISO_COUNTRY_CODE     CHARACTER(2)
);

CREATE TABLE BOOKORDER (
    ORDER_NO             INTEGER          PRIMARY KEY,
    CUSTOMER_NO          INTEGER          NOT NULL,
    ORDERED              DATE,
    DELIVERY              DATE,
    STATUS               CHARACTER(8)
    CHECK (STATUS IN ('ACCEPTED',
                     'DELAYED',
                     'SHIPPED',
                     'RETURNED',
                     'PAID')),
    CONSTRAINT CUSTOMER_FK FOREIGN KEY (CUSTOMER_NO)
        REFERENCES KUNDE (KUNDENNAME)
);

CREATE TABLE ORDER_POSITION (
    POSITION_NO           INTEGER          PRIMARY KEY,
    ORDER_NO             INTEGER          NOT NULL,
    ARTICLE_NO           INTEGER          NOT NULL,
    NUMBER               SMALLINT,
    CONSTRAINT ORDER_FK FOREIGN KEY (ORDER_NO)
        REFERENCES BOOKORDER (ORDER_NO),
    CONSTRAINT BOOK_FK FOREIGN KEY (ARTICLE_NO)
        REFERENCES BOOK (ARTICLE_NO)
);
```


2-3 CREATE VIEW



例：

```
CREATE VIEW V_BOOK
      AS SELECT ARTICLE_NO, TITLE, RETAIL_PRICE
      FROM BOOK
      WHERE COPIES_AVAILABLE >= 1
```

建立如下格式的 VIEW

ARTICLE_NO	TITLE	RETAIL_PRICE

CREATE VIEW 用来创建一个 VIEW。VIEW 建立后，数据的增加、修改、删除语句与 TABLE 的完全一样，用户一般不必追究一个名字是表还是视图，只是通过视图进行的添加和修改，新数据必须符合视图的选择条件。

view-name：

给定 VIEW 的名字，不能与现有的表名、VIEW 名、别名相同。

column-name：

给 VIEW 各列指定的名字，如果给出，其个数必须与子查询语句选择出来的列数一样。如果没有指定，则 VIEW 的列名与子查询所查得的列名一致。

AS subselect：

用子查询来定义 VIEW 的列。

WITH ... CHECK OPTION：

使用 VIEW 来添加或修改的记录会直接记录到原表中去，但是记录的数据必须符合 VIEW 的选择条件。由于 VIEW 可以从表来创建，也可以从别的 VIEW 来创建，如果是从别的 VIEW 创建的，就要指明增改数据时，新数据是指要符合本 VIEW 的选择条件（LOCAL），还是要符合下面所有 VIEW 的选择条件（CASCADDED）。例如：

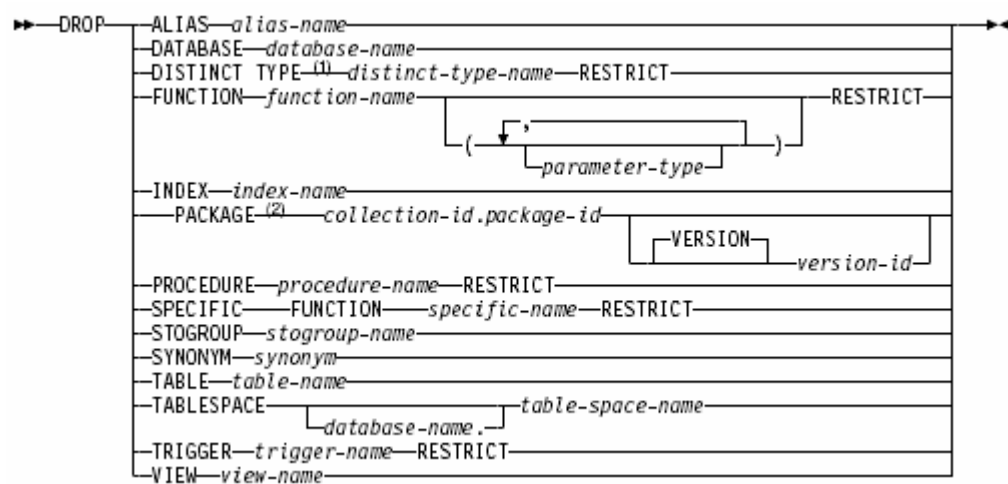
```
CREATE VIEW A AS SELECT * FROM EMPLOYEE WHERE SALARY < 8000;
```

```
CREATE VIEW B1 AS SELECT * FROM A WHERE SALARY > 5000 WITH LOCAL CHECK OPTION;
```

```
CREATE VIEW B2 AS SELECT * FROM A WHERE SALARY > 5000 WITH LOCAL CASCADDED OPTION;
```

向 B1 里插入一条 SALARY 是 9000 的记录会成功，而向 B2 里插入同样的记录将会失败，因为这条记录不符合视图 A 的选择条件。

2-4 删除表或视图



DROP 语句用来删除别名、数据库、功能函数、索引、过程、包、数据表、表空间、触发器、视图等各种对象。

2-5 添加数据

例：

```
CREATE TABLE EMPLOYEE
```

```
( ID          SMALLINT      NOT NULL,
  NAME        VARCHAR(9),
  DEPT        SMALLINT      WITH DEFAULT 10,
  JOB         CHAR(5),
  YEARS       SMALLINT,
  SALARY      DECIMAL(7,2),
  BIRTH_DATE  DATE)
```

```
INSERT INTO EMPLOYEE
```

```
VALUES (12, 'Harris', 20, 'Sales', 5, 18000, '1950-1-1')
```

```
INSERT INTO EMPLOYEE (NAME, JOB, ID)
```

```
VALUES ('Swagerman', 'Prgmr', 500),
       ('Limoges', 'Prgmr', 510),
       ('Li', 'Prgmr', 520)
```

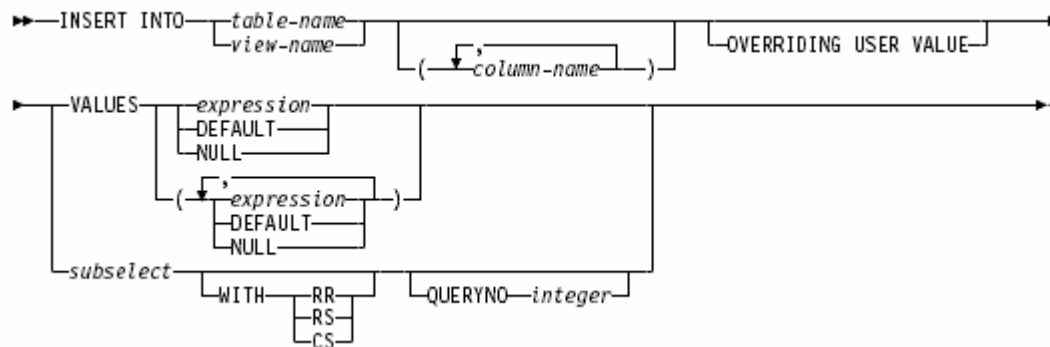
```
SELECT * FROM EMPLOYEE
```

ID	NAME	DEPT	JOB	YEARS	SALARY	BIRTH_DATE
12	Harris	20	Sales	5	18000.00	01/01/1950
500	Swagerman	10	Prgmr	-	-	-
510	Limoges	10	Prgmr	-	-	-
520	Li	10	Prgmr	-	-	-

INSERT 命令用来向数据表或视图里添加数据,向视图中添加数据时,会直接添加到 VIEW 所对应的基表中。INSERT 命令有两种基本方式,一种使用是在命令里给出要添加的数据,另一种方式是使用子查询,把子查询查出的结果赋到表里。INSERT 语句里,在表名后可以给定列名,VALUES 子句中的值顺序要与列名一致,表名后也可以不给出列名,直接接 VALUES 子句,这样将把各个字段值按照建表时列的顺序装入。INSERT 语句中没给定的列名将使用 NULL 或列的 DEFAULT 值填充,如果此列没有 DEFAULT,同时规定了 NOT NULL,插入语句将出错。

INSERT 语句的全部选项如下：

2-5 添加数据（续页）



INSERT INTO table-name or view-name :

插入的表名或列名。

column-name,... :

给定插入记录的列名 ,VALUES 子句中的值将依次赋予各个列。没有给出的列将使用 NULL 或列的 DEFAULT 值填充。

OVERRIDING USER VALUE :

更新自动编号字段。要插入的记录里如果有系统自动生成序号的 “identity column” 型的列，系统将从新生成值，而不是用输入的值。

VALUES (...):

给出要插入的各个字段的值，必须与数据库中列定义的数据类型相匹配。VALUES 里可以给出具体的值，也可以用 “DEFAULT”: 使用列的默认值，“NULL”: 使用 NULL 值填充。

subselect :

子查询，不直接给定要插入的数值，而是使用子查询，从另一个表或视图里查出一条或若干条数据，插入本表。

WITH RR/RS/CS :

指定子查询语句的隔离级别。

QUERYNO integer :

给子查询语句一个记 TRACE 或 EXPLAN 时的编号。

2-6 更改数据

ID	NAME	DEPT	JOB	YEARS	SALARY	BIRTH_DATE
12	Harris	20	Sales	5	8000.00	01/01/1970
500	Swagerman	10	Prgmr	1	5000.00	11/25/1979
410	Tom	10	Prgmr	8	9000.00	01/01/1971
520	Li	10	Prgmr	3	7000.00	01/01/1974

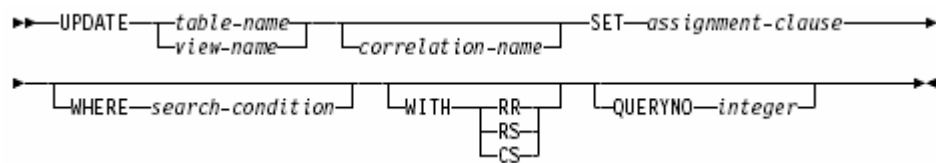
```
UPDATE EMPLOYEE
  SET JOB='PM', SALARY = SALARY + 2000
  WHERE ID = 410
```

ID	NAME	DEPT	JOB	YEARS	SALARY	BIRTH_DATE
12	Harris	20	Sales	5	8000.00	01/01/1970
500	Swagerman	10	Prgmr	1	5000.00	11/25/1979
410	Tom	10	PM	8	11000.00	01/01/1971
520	Li	10	Prgmr	3	7000.00	01/01/1974

UPDATE 命令用来修改表或视图中现有的数据。它把符合 WHERE 条件的所有记录的列更新成命令里指定的值。UPDATE 视图中的记录时，会更新视图所相关的基表。

UPDATE 命令的详细选项如下：

2-6 更改数据（续页）



assignment clause:

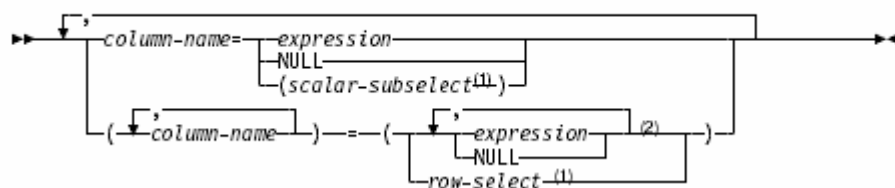


table-name or view-name :

给定要更新的表名或视图名称。

SET :

给出要更新的列和新值。列和值的给出有两种形式，一种是“列名 1=值 1，列名 2=值 2...””，另一种是“(列名 1，列名 2 ...)=(值 1，值 2 ...)”的形式。其中的“值”可以是直接给定的常量，也可以是使用子查询得到的数据。

WHERE search-condition :

指定更新数据的范围，必须符合 search-condition 的数据行才会被更新。

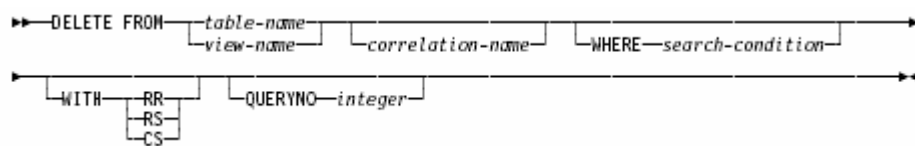
WITH RR/RS/CS :

指定子查询语句的隔离级别。

QUERYNO integer :

给子查询语句一个记 TRACE 或 EXPLAN 时的编号。

2-7 删除数据



例：

ID	NAME	DEPT	JOB	YEARS	SALARY	BIRTH_DATE
12	Harris	20	Sales	5	8000.00	01/01/1970
500	Swagerman	10	Prgmr	1	5000.00	11/25/1979
410	Tom	10	Prgmr	8	9000.00	01/01/1971
520	Li	10	Prgmr	3	7000.00	01/01/1974

DELETE FROM EMPLOYEE WHERE ID = 500

ID	NAME	DEPT	JOB	YEARS	SALARY	BIRTH_DATE
12	Harris	20	Sales	5	8000.00	01/01/1970
410	Tom	10	PM	8	11000.00	01/01/1971
520	Li	10	Prgmr	3	7000.00	01/01/1974

DELETE 命令用来删除表或视图中符合指定条件的记录，对视图的操作将导致数据在原表中被删除。

DELETE 语句的选项：

FROM table-name or view-name：

给定要删除记录的表名或列名。

WHERE search-condition：

给定要删除记录的范围，必须符合 search-condition 的数据行才会被删除。

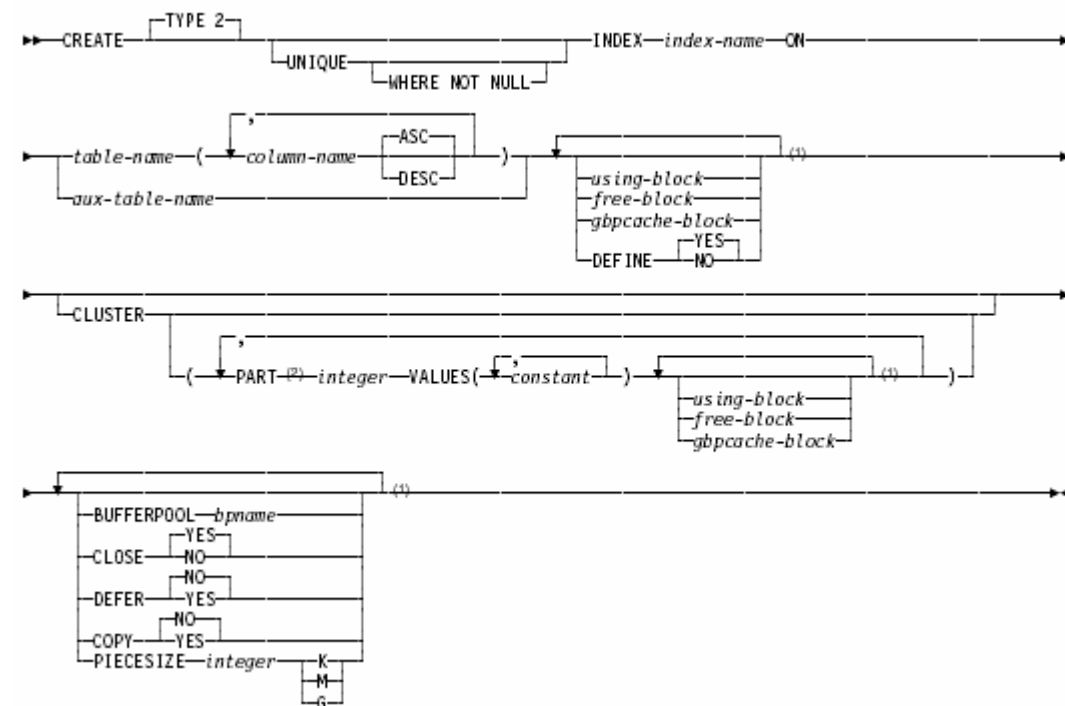
WITH RR/RS/CS：

指定子查询语句的隔离级别。

QUERYNO integer：

给子查询语句一个记 TRACE 或 EXPLAN 时的编号。

2-8 创建索引



例：

```
CREATE INDEX UNIQUE EMP_IDX ON EMPLOYEE(ID);
```

```
CREATE INDEX XEMP2 ON EMP(EMPNO ASC)
  USING STOGROUP DSN8G610 PRIQTY 36 ERASE NO
  CLUSTER
  (PART 1 VALUES('H99'),
   PART 2 VALUES('P99'),
   PART 3 VALUES('Z99'),
   PART 4 VALUES('999'))
  BUFFERPOOL BP1 CLOSE YES COPY YES;
```

`CREATE INDEX` 命令用来创建索引。在一个表上可以建立多个索引，索引的使用是透明的，系统自动选用最优的索引来进行查询。通过创建索引，能够显著提高数据查询的速度，但索引过多也会影响数据插入和更新的速度，并占用更多的空间。

`CREATE INDEX` 选项：

UNIQUE：

指明创建的是唯一索引，索引所涉及的项在全表范围内必须唯一。

2-8 创建索引（续页）

INDEX index-name :

为 INDEX 指定一个名字。

column-name, ... :

指定索引所使用的列。

ASC :

指定索引按升序排列。

DESC :

指定索引按降序排列。

using-block :

用来给定索引数据的存储设置

free-block :

给定索引建立时预留数据空间的情况。

CLUSTER :

指明建立的是 cluster index。对于一些数据量十分巨大的数据库表，可以建立成 PARTITION DATABASE，把表中的数据分配在若干磁盘上，根据记录中某字段一定的值决定记录放在哪个具体部分，这样在数据处理时就可以根据特征值的不同直接到不同磁盘上读写，这极大地提高了数据存取速度。PARTITION DATABASE 的实现是通过定义一个 CLUSTER INDEX 来实现的。CLUSTER 的句法是 PART integer VALUES(constant, ...)，PART 给出分区的编号，VALUES 依次给出各个分区中特征值的上限。

BUFFERPOOL bpname :

指定 INDEX 使用的 BUFFERPOOL。

CLOSE YES/NO :

指明 INDEX 不使用时或 DB2 打开的文件达到上限后能否关闭 INDEX 所在数据集。

DEFER :

指明 INDEX 是否立即创建，NO：立即创建，YES：不立即创建，可以等数据加入后使用 REBUILD INDEX 来真正创建。

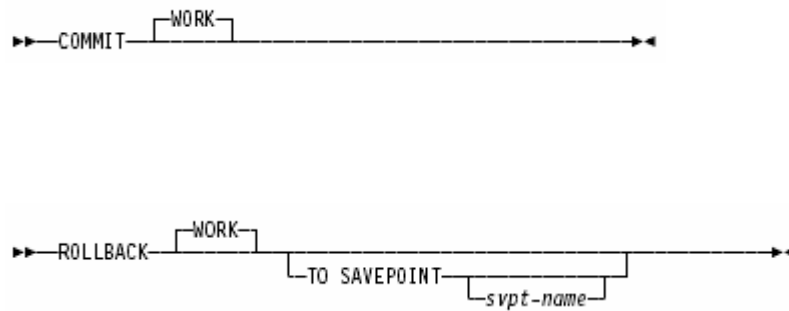
PIECESIZE integer K/M/G :

指明存放索引的单个数据集的大小。K/M/G 分别指单位是 Kilo/Mega/Giga 字节。

COPY YES/NO :

指明索引能否被 COPY 工具备份。

2-9 提交与回滚



COMMIT 与 ROLLBACK 命令用来决定一个工作单元 (Unit Of Work, UOW) 中对数据库的改变是真正落实还是全部恢复到改动以前的状态。所谓工作单元,就是从程序开始或上一个 COMMIT/ROLLBACK 到下一个 COMMIT/ROLLBACK 之间的所有数据库操作,这些操作必须作为一个整体提交或者回滚。

COMMIT 命令把数据真正落实到数据库中; ROLLBACK 命令把数据恢复到改变前的状态。

第三章 数据查询

- 示例数据表
- 基本查询语句
- 查询语句中使用函数
- 从一个以上表中查询数据
- 使用子查询

3-1 示例数据表

```
CREATE TABLE DSN8710.DEPT(  
    DEPTNO    CHAR(3)      NOT NULL,  
    DEPTNAME  VARCHAR(36) NOT NULL,  
    MGRNO     CHAR(6) ,  
    ADMRDEPT  CHAR(3)      NOT NULL,  
    LOCATION  CHAR(16) ,  
    PRIMARY KEY (DEPTNO)  
);  
  
CREATE TABLE DSN8710.EMP(  
    EMPNO     CHAR(6)      NOT NULL,  
    FIRSTNME  VARCHAR(12) NOT NULL,  
    MIDINIT   CHAR(1)      NOT NULL,  
    LASTNAME  VARCHAR(15) NOT NULL,  
    WORKDEPT  CHAR(3) ,  
    PHONENO   CHAR(4) CONSTRAINT NUMBER CHECK  
        (PHONENO >= '0000' AND  
         PHONENO <= '9999') ,  
    HIREDATE  DATE ,  
    JOB       CHAR(8) ,  
    EDLEVEL   SMALLINT ,  
    SEX       CHAR(1) ,  
    BIRTHDATE DATE ,  
    SALARY    DECIMAL(9,2) ,  
    BONUS     DECIMAL(9,2) ,  
    COMM      DECIMAL(9,2) ,  
    PRIMARY KEY (EMPNO) ,  
    FOREIGN KEY RED(WORKDEPT) REFERENCES DEPT  
        ON DELETE SET NULL  
);  
  
ALTER TABLE DSN8710.DEPT  
    FOREIGN KEY RDD(ADMRDEPT) REFERENCES DSN8710.DEPT  
    ON DELETE CASCADE;  
  
ALTER TABLE DSN8710.DEPT  
    FOREIGN KEY RDE(MGRNO) REFERENCES DSN8710.EMP  
    ON DELETE SET NULL;
```

以上是后面查询时用到的表的定义，DEPT 表中定义的 RDD 和 RDE 外键因为存在自指和交叉参考，不能在 CREATE TABLES 时建立，必须使用 ALTER TABLE 命令来加入。

3-1 示例数据表（续页 1）

EMPNO	FIRSTNAME	MIDINIT	LASTNAME	WORKDEPT	PHONENO	HIREDATE	JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
000010	CHRISTINE	I	HAAS	A00	3978	1965-01-01	PRES	18	F	1933-08-14	52750.00	1000.00	4220.00
000020	MICHAEL	L	THOMPSON	B01	3476	1973-10-10	MANAGER	18	M	1948-02-02	41250.00	800.00	3300.00
000030	SALLY	A	KWAN	C01	4738	1975-04-05	MANAGER	20	F	1941-05-11	38250.00	800.00	3060.00
000050	JOHN	B	GEYER	E01	6789	1949-08-17	MANAGER	16	M	1925-09-15	40175.00	800.00	3214.00
000060	IRVING	F	STERN	D11	6423	1973-09-14	MANAGER	16	M	1945-07-07	32250.00	600.00	2580.00
000070	EVA	D	PULASKI	D21	7831	1980-09-30	MANAGER	16	F	1953-05-26	36170.00	700.00	2893.00
000090	EILEEN	W	HENDERSON	E11	5498	1970-08-15	MANAGER	16	F	1941-05-15	29750.00	600.00	2380.00
000100	THEODORE	Q	SPENSER	E21	0972	1980-06-19	MANAGER	14	M	1956-12-18	26150.00	500.00	2092.00
000110	VINCENZO	G	LUCCHESE	A00	3490	1958-05-16	SALESREP	19	M	1929-11-05	46500.00	900.00	3720.00
000120	SEAN		O'CONNELL	A00	2167	1963-12-05	CLERK	14	M	1942-10-18	29250.00	600.00	2340.00
000130	DOLORES	M	QUINTANA	C01	4578	1971-07-28	ANALYST	16	F	1925-09-15	23800.00	500.00	1904.00
000140	HEATHER	A	NICHOLLS	C01	1793	1976-12-15	ANALYST	18	F	1946-01-19	28420.00	600.00	2274.00
000150	BRUCE		ADAMSON	D11	4510	1972-02-12	DESIGNER	16	M	1947-05-17	25280.00	500.00	2022.00
000160	ELIZABETH	R	PIANKA	D11	3782	1977-10-11	DESIGNER	17	F	1955-04-12	22250.00	400.00	1780.00
000170	MASATOSHI	J	YOSHIMURA	D11	2890	1978-09-15	DESIGNER	16	M	1951-01-05	24680.00	500.00	1974.00
000180	MARILYN	S	SCOUTTEN	D11	1682	1973-07-07	DESIGNER	17	F	1949-02-21	21340.00	500.00	1707.00
000190	JAMES	H	WALKER	D11	2986	1974-07-26	DESIGNER	16	M	1952-06-25	20450.00	400.00	1636.00
000200	DAVID		BROWN	D11	4501	1966-03-03	DESIGNER	16	M	1941-05-29	27740.00	600.00	2217.00
000210	WILLIAM	T	JONES	D11	0942	1979-04-11	DESIGNER	17	M	1953-02-23	18270.00	400.00	1462.00
000220	JENNIFER	K	LUTZ	D11	0672	1968-08-29	DESIGNER	18	F	1948-03-19	29840.00	600.00	2387.00
000230	JAMES	J	JEFFERSON	D21	4265	1966-11-21	CLERK	14	M	1935-05-30	22180.00	400.00	1774.00
000240	SALVATORE	M	MARINO	D21	3780	1979-12-05	CLERK	17	M	1954-03-31	28760.00	600.00	2301.00
000250	DANIEL	S	SMITH	D21	0961	1969-10-30	CLERK	15	M	1939-11-12	19180.00	400.00	1534.00
000260	SYBIL	V	JOHNSON	D21	8953	1975-09-11	CLERK	16	F	1936-10-05	17250.00	300.00	1380.00
000270	MARIA	L	PEREZ	D21	9001	1980-09-30	CLERK	15	F	1953-05-26	27380.00	500.00	2190.00
000280	ETHEL	R	SCHNEIDER	E11	8997	1967-03-24	OPERATOR	17	F	1936-03-28	26250.00	500.00	2100.00
000290	JOHN	R	PARKER	E11	4502	1980-05-30	OPERATOR	12	M	1946-07-09	15340.00	300.00	1227.00
000300	PHILIP	X	SMITH	E11	2095	1972-06-19	OPERATOR	14	M	1936-10-27	17750.00	400.00	1420.00

3-1 示例数据表（续页 2）

000310	MAUDE	F	SETRIGHT	E11	3332	1964-09-12	OPERATOR	12	F	1931-04-21	15900.00	300.00	1272.00
000320	RAMLAL	V	MEHTA	E21	9990	1965-07-07	FIELDREP	16	M	1932-08-11	19950.00	400.00	1596.00
000330	WING		LEE	E21	2103	1976-02-23	FIELDREP	14	M	1941-07-18	25370.00	500.00	2030.00
000340	JASON	R	GOUNOT	E21	5698	1947-05-05	FIELDREP	16	M	1926-05-17	23840.00	500.00	1907.00
200010	DIAN	J	HEMMINGER	A00	3978	1965-01-01	SALESREP	18	F	1933-08-14	46500.00	1000.00	4220.00
200120	GREG		ORLANDO	A00	2167	1972-05-05	CLERK	14	M	1942-10-18	29250.00	600.00	2340.00
200140	KIM	N	NATZ	C01	1793	1976-12-15	ANALYST	18	F	1946-01-19	28420.00	600.00	2274.00
200170	KIYOSHI		YAMAMOTO	D11	2890	1978-09-15	DESIGNER	16	M	1951-01-05	24680.00	500.00	1974.00
200220	REBA	K	JOHN	D11	0672	1968-08-29	DESIGNER	18	F	1948-03-19	29840.00	600.00	2387.00
200240	ROBERT	M	MONTEVERDE	D21	3780	1979-12-05	CLERK	17	M	1954-03-31	28760.00	600.00	2301.00
200280	EILEEN	R	SCHWARTZ	E11	8997	1967-03-24	OPERATOR	17	F	1936-03-28	26250.00	500.00	2100.00
200310	MICHELLE	F	SPRINGER	E11	3332	1964-09-12	OPERATOR	12	F	1931-04-21	15900.00	300.00	1272.00
200330	HELENA		WONG	E21	2103	1976-02-23	FIELDREP	14	F	1941-07-18	25370.00	500.00	2030.00
200340	ROY	R	ALONZO	E21	5698	1947-05-05	FIELDREP	16	M	1926-05-17	23840.00	500.00	1907.00

DEPTNO	DEPTNAME	MGRNO	ADMNDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	
B01	PLANNING	000020	A00	
C01	INFORMATION CENTER	000030	A00	
D01	DEVELOPMENT CENTER	-----	A00	
D11	MANUFACTURING SYSTEMS	000060	D01	
D21	ADMINISTRATION SYSTEMS	000070	D01	
E01	SUPPORT SERVICES	000050	A00	
E11	OPERATIONS	000090	E01	
E21	SOFTWARE SUPPORT	000100	E01	

以上是两表里的数据内容。

3-2 基本查询语句

选择所有列 SELECT * :

使用 SELECT * 命令来查询表的所有字段,这时你可以不知道列的名字,显示结果各列按照定义表时的顺序显示:

```
SELECT * FROM DSN8710.DEPT;
```

DEPTNO	DEPTNAME	MGRNO	ADMRDEPT	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	
B01	PLANNING	000020	A00	
C01	INFORMATION CENTER	000030	A00	
D01	DEVELOPMENT CENTER	-----	A00	
D11	MANUFACTURING SYSTEMS	000060	D01	
D21	ADMINISTRATION SYSTEMS	000070	D01	
E01	SUPPORT SERVICES	000050	A00	
E11	OPERATIONS	000090	E01	
E21	SOFTWARE SUPPORT	000100	E01	

选择特定列 SELECT column-name:

SELECT 时指定列名,显示时各列按指定的次序显示,如:

```
SELECT MGRNO,DEPTNAME FROM DSN8710.DEPT;
```

MGRNO	DEPTNAME
000010	SPIFFY COMPUTER SERVICE DIV.
000020	PLANNING
000030	INFORMATION CENTER
-----	DEVELOPMENT CENTER
000060	MANUFACTURING SYSTEMS
000070	ADMINISTRATION SYSTEMS
000050	SUPPORT SERVICES
000090	OPERATIONS
000100	SOFTWARE SUPPORT

选择数据表以外数据：

SELECT 后面可以跟数据表的列名，也可以跟以常量或 HOST VARIABLE 等数据，如：

```
SELECT 'MANAGER NUM: ', MGRNO, 'DEPT NUM: ', DEPTNO FROM DSN8710.DEPT;
```

```
-----+-----+-----+-----+
                MGRNO                DEPTNO
-----+-----+-----+-----+
MANAGER NUM:  000010  DEPT NUM:  A00
MANAGER NUM:  000020  DEPT NUM:  B01
MANAGER NUM:  000030  DEPT NUM:  C01
MANAGER NUM:  -----  DEPT NUM:  D01
MANAGER NUM:  000060  DEPT NUM:  D11
MANAGER NUM:  000070  DEPT NUM:  D21
MANAGER NUM:  000050  DEPT NUM:  E01
MANAGER NUM:  000090  DEPT NUM:  E11
MANAGER NUM:  000100  DEPT NUM:  E21
```

选择表达式：

你可以选择对表的列以及常数进行运算得到结果：

```
SELECT EMPNO, (SALARY + BONUS + COMM) FROM DSN8710.EMP;
```

```
-----+-----+-----+-----+-----+-----+-----+
EMPNO
-----+-----+-----+-----+-----+-----+-----+
000010      57970.00
000020      45350.00
000030      42110.00
000050      44189.00
...

```

SELECT DISTINCT：

SELECT DISTINCT 可以把 DISTINCT 后所跟的列的重复值滤掉，取出所有行在这个列上的所有可能值。

```
SELECT DISTINCT ADMRDEPT FROM DSN8710.DEPT;
```

```
-----+-----+-----+-----+-----+-----+
ADMRDEPT
-----+-----+-----+-----+-----+-----+
A00
D01
E01
```


3-2 基本查询语句 (续页 2)

为选择出的列命名 SELECT AS :

使用 AS 选项, 可以为选出的列起个别名显示

```
SELECT EMPNO AS EMPLOYEE_ID,  
       SALARY+BONUS+COMM TOTAL_SAL FROM DSN8710.EMP;
```

```
-----+-----+-----+-----+-----+-----  
EMPLOYEE_ID      TOTAL_SAL  
-----+-----+-----+-----+-----+-----  
000010           57970.00  
000020           45350.00  
000030           42110.00  
000050           44189.00  
...  
...  
...
```

使用系统函数选取数据 :

可以使用 DB2 提供的系统函数对选出的列进行处理, 系统函数的详细情况见后面详述。

```
SELECT MAX(SALARY) MAX_SAL FROM DSN8710.EMP;
```

```
-----+-----+-----+-----+-----+-----  
MAX_SAL  
-----+-----+-----+-----+-----+-----  
52750.00
```

查询符合指定条件的记录 :

SELECT 可以使用 “ WHERE 条件表达式 ” 的方法来选择查询一定条件的记录, 对每条记录, DB2 把字段值代入条件表达式进行计算, 对于能使表达式的值为真的记录, 才查询出来。

```
SELECT EMPNO, FIRSTNME, SALARY FROM DSN8710.EMP  
       WHERE WORKDEPT='A00';
```

```
-----+-----+-----+-----+-----+-----  
EMPNO  FIRSTNME      SALARY  
-----+-----+-----+-----+-----+-----  
000010  CHRISTINE     52750.00  
000110  VINCENZO       46500.00  
000120  SEAN            29250.00  
200010  DIAN            46500.00  
200120  GREG            29250.00
```

3-2 基本查询语句 (续页 3)

WHERE 中的关系运算符：

关系含义	关系符	示例
空值	IS NULL	PHONENO IS NULL
相等	=	DEPTNO = 'X01'
不等	<>	DEPTNO <> 'X01'
小于	<	AVG(SALARY) < 30000
小于等于	<=	AGE <= 25
大于等于	>=	AGE >= 21
大于	>	SALARY > 2000
相像	LIKE	NAME LIKE '%SMITH%' or STATUS LIKE 'N_'
或者	OR	HIREDATE < '1965-01-01' OR SALARY < 16000
并且	AND	HIREDATE < '1965-01-01' AND SALARY < 16000
否	NOT	NOT AGE=28
在之间	BETWEEN	SALARY BETWEEN 20000 AND 40000
在一定值域	IN (X, Y, Z)	DEPTNO IN ('B01', 'C01', 'D01')

选择有空值的行：

```
SELECT * FROM DSN8710.DEPT WHERE MGRNO IS NULL;
```

```
-----+-----+-----+-----+-----+-----+-----+-----+
DEPTNO  DEPTNAME                                MGRNO  ADMRDEPT  LOCATION
-----+-----+-----+-----+-----+-----+-----+
D01      DEVELOPMENT CENTER                        -----  A00
```

使用等式和不等式：

```
SELECT WORKDEPT, EMPNO FROM DSN8710.EMP
  WHERE NOT (SALARY + BONUS + COMM) > 30000 AND JOB = 'MANAGER';
SELECT HIREDATE, FIRSTNME, LASTNAME FROM DSN8710.EMP
  WHERE HIREDATE < '1960-01-01';
```

选择与某值相似的记录：

使用 LIKE 条件句，可以选择某列的值于给定值相近的记录，LIKE 只用于字符型字段。
LIKE 选项中可以使用的“%”代表一个或多个任意字符，“_”代表一个任意字符，如：

```
SELECT FIRSTNME, LASTNAME, WORKDEPT FROM DSN8710.EMP
  WHERE FIRSTNME LIKE 'EILEE_' AND LASTNAME LIKE 'H%';
-----+-----+-----+-----+-----+-----+-----+
FIRSTNME      LASTNAME      WORKDEPT
-----+-----+-----+-----+-----+-----+
EILEEN        HENDERSON      E11
```

3-2 基本查询语句 (续页 4)

查询字段在某个范围之内的记录：

SELECT 语句可以用 BETWEEN 选项指定选择某字段在一定范围内的记录。

```
SELECT DEPTNO, MGRNO FROM DSN8710.DEPT
WHERE DEPTNO BETWEEN 'C00' AND 'D31';
```

```
-----+-----+-----+-----
DEPTNO  MGRNO
-----+-----+-----+-----
C01      000030
D01      -----
D11      000060
D21      000070
```

查询某字段在指定列表中的记录：

```
SELECT DEPTNO, MGRNO FROM DSN8710.DEPT
WHERE DEPTNO IN ('B01', 'C01', 'D01');
```

```
-----+-----+-----+-----+--
DEPTNO  MGRNO
-----+-----+-----+-----+--
B01      000020
C01      000030
D01      -----
```

使用 CASE 语句：

使用 CASE 语句可以根据查询到的字段值的不同情况作不同的查询输出，如：

```
SELECT EMPNO, FIRSTNAME, MIDINIT, LASTNAME,
CASE
  WHEN EDLEVEL<=12 THEN 'HIGH SCHOOL OR LESS'
  WHEN EDLEVEL>12 AND EDLEVEL<=14 THEN 'JUNIOR COLLEGE'
  WHEN EDLEVEL>14 AND EDLEVEL<=17 THEN 'FOUR-YEAR COLLEGE'
  WHEN EDLEVEL>17 THEN 'GRADUATE SCHOOL'
  ELSE 'UNKNOWN'
END
AS EDUCATION FROM DSN8710.EMP WHERE JOB='CLERK';
```

```
-----+-----+-----+-----+-----+-----
EMPNO  FIRSTNAME  MIDINIT  LASTNAME  EDUCATION
-----+-----+-----+-----+-----+-----
000120 SEAN          O'CONNELL  JUNIOR COLLEGE
000230 JAMES        J         JEFFERSON  JUNIOR COLLEGE
000240 SALVATORE    M         MARINO     FOUR-YEAR COLLEGE
000250 DANIEL       S         SMITH      FOUR-YEAR COLLEGE
000260 SYBIL        V         JOHNSON    FOUR-YEAR COLLEGE
000270 MARIA        L         PEREZ      FOUR-YEAR COLLEGE
200120 GREG         ORLANDO   JUNIOR COLLEGE
200240 ROBERT       M         MONTEVERDE FOUR-YEAR COLLEGE
```

按照一定次序排列查询结果 ORDER BY :

SELECT 查询出来的结果,如果不加指定,是以数据存放的次序显示的,由于数据增添和删除的随机性,造成数据是随机存放的。为了查看的方便,可以使用 ORDER BY 子句指定查询结果的排列方式。ORDER BY 可以指定一个或多个字段,每个字段又可以指定按升序(ASC)排列和降序(DEC)排列,显示时先按靠前的排序字段排序,前一个字段相同时,再按后一个字段排序。

```
SELECT EMPNO, LASTNAME, HIREDATE FROM DSN8710.EMP
      WHERE WORKDEPT = 'A00' ORDER BY HIREDATE ASC;
```

EMPNO	LASTNAME	HIREDATE
000110	LUCCHESI	1958-05-16
000120	O'CONNELL	1963-12-05
200010	HEMMINGER	1965-01-01
000010	HAAS	1965-01-01
200120	ORLANDO	1972-05-05

排序字段不仅可以直接是数据库列,而且可以是某些计算表达式,如:

```
SELECT EMPNO, (SALARY + BONUS + COMM) AS TOTAL_SAL
      FROM DSN8710.EMP ORDER BY TOTAL_SAL;
```

按照某列加总 GROUP BY :

SELECT 可以使用 GROUP BY 语句对符合 WHERE 条件的数据按照某一个列相同的条件进行加总。

```
SELECT WORKDEPT, MIN(EDLEVEL), MAX(EDLEVEL)
      FROM DSN8710.EMP GROUP BY WORKDEPT;
```

WORKDEPT	MIN(EDLEVEL)	MAX(EDLEVEL)
A00	14	19
B01	18	18
C01	16	20
D11	16	18
D21	14	17
E01	16	16
E11	12	17
E21	14	16

GROUP BY 还可以指定一个以上的列进行加总,如:

```
SELECT WORKDEPT, SEX, AVG(SALARY) AS AVG_SALARY
      FROM DSN8710.EMP WHERE WORKDEPT IN ('A00', 'C01')
      GROUP BY WORKDEPT, SEX;
```

WORKDEPT	SEX	AVG_SALARY
A00	F	49625.00000000
A00	M	35000.00000000
C01	F	29722.50000000

3-2 基本查询语句 (续页 6)

给分组限定条件 HAVING :

使用 GROUP BY 子句时可以用 HAVING 子句来限定只显示符合条件的加总记录, HAVING 子句与 WHERE 相似, 但 WHERE 是用来挑选基础记录, HAVING 用来检查 GROUP BY 后的加总记录。如:

```
SELECT WORKDEPT,AVG(SALARY) AS AVG_SALARY
      FROM DSN8710.EMP GROUP BY WORKDEPT
      HAVING AVG(SALARY)>30000 ORDER BY WORKDEPT;
```

WORKDEPT	AVG_SALARY
A00	40850.00000000
B01	41250.00000000
E01	40175.00000000

对比去掉 HAVING 语句后的查询结果:

```
SELECT WORKDEPT,AVG(SALARY) AS AVG_SALARY
      FROM DSN8710.EMP GROUP BY WORKDEPT ORDER BY WORKDEPT;
```

WORKDEPT	AVG_SALARY
A00	40850.00000000
B01	41250.00000000
C01	29722.50000000
D11	25147.27272727
D21	25668.57142857
E01	40175.00000000
E11	21020.00000000
E21	24086.66666666

3-2 基本查询语句 (续页 7)

合并显示结果 UNION :

UNION 子句可以把两个查询的结果纵向连在一起显示。如下面的 SELECT 语句查询所有总部职员和部门领导。

```
SELECT MGRNO, 'LEADER' AS ROLL FROM DSN8710.DEPT
```

```
UNION
```

```
SELECT EMPNO, 'D11' AS ROLL FROM DSN8710.EMP WHERE WORKDEPT = 'D11';
```

```
-----+-----+-----+-----+-----+-----+-----
MGRNO   ROLL
-----+-----+-----+-----+-----+-----+-----
000010  LEADER
000020  LEADER
000030  LEADER
000050  LEADER
000060  D11
000060  LEADER
000070  LEADER
000090  LEADER
000100  LEADER
000150  D11
000160  D11
000170  D11
000180  D11
000190  D11
000200  D11
000210  D11
000220  D11
200170  D11
200220  D11
-----  LEADER
```

3-3 查询语句中使用函数

在 SELECT 语句中可以通过使用函数来实现丰富的功能，常用的函数有：

列函数：

SUM	返回查得各行指定字段的和
MIN	返回查得各行指定字段的最小值
AVG	返回查得各行指定字段的平均值
MAX	返回查得各行指定字段的最小值
COUNT	返回查得的行数
STDDEV	返回查得各行指定字段的统计标准方差
VARIANCE	返回查得各行指定字段的统计方差

例：

```
SELECT SUM(SALARY) AS SUMSAL, MIN(SALARY) AS MINSAL,
       AVG(SALARY) AS AVGSAL, STDDEV(SALARY) AS SDSAL,
       MAX(SALARY) AS MAXSAL, COUNT(*) AS CNTSAL
FROM DSN8710.EMP WHERE WORKDEPT = 'D11';
```

```

+-----+-----+-----+-----+-----+-----+
SUMSAL  MINSAL      AVGSAL                      SDSAL  MAXSAL  CNTSAL
+-----+-----+-----+-----+-----+-----+
276620.00 18270.00 25147.27272727 +0.4198694799164255E+04 32250.00    11

```

数值函数：

函数	返回	示例
ABS or ABSVAL	返回变量的变量绝对值	ABS(DIFFERENCE)
ACOS	返回变量的反余弦弧度值	ACOS(COSOFANGLE)
ASIN	返回变量的反正弦弧度值	ASIN(SINOFANGLE)
ATAN	返回变量的反正切弧度值	ATAN(TANOFANGLE)
ATANH	返回变量的反双曲正切弧度值	ATANH(TANHOFANGLE)
ATAN2	返回以 x 和 y 坐标表示的反正切弧度值	ATAN2(XCOORD, YCOORD)
BLOB	把一个字符串类型数据转成 BLOB 型数据	BLOB('This is a BLOB')
CEIL or CEILING	把含小数的数值向上取整	CEIL(SALARY)
CHAR	把日期、时间、整型、浮点、数值型变量转为字符变量	CHAR(HIREDATE)
CLOB	把字符变量转为 CLOB 变量	CLOB('This is a CLOB')
CONCAT	把两个字符串连接起来做一个串返回	CONCAT(FIRSTNAME, LASTNAME)
COS	返回变量的余弦值	COS(ANGLE)
COSH	返回变量的双曲余弦值	COSH(ANGLE)
DATE	把字符转换成日期型变量	DATE('1989-03-02')
DAY	取得日期型变量的日号	DAY(DATE1 - DATE2)

3-3 查询语句中使用函数（续页 1）

函数	返回	示例
DAYOFMONTH	同 DAY 但不接受 TIMESpan 和 DATESpan 型变量	DAYOFMONTH (DATE1)
DAYOFWEEK	把日期型变量转为此周的第几日	DAYOFWEEK (HI REDATE)
DAYOFYEAR	把日期型变量转为本年的第多少天数	DAYOFYEAR (HI REDATE)
DAYS	把日期型变量转换成从公元 1 年 1 月 1 日起的日数	DAYS (' 1990-01-08') - DAYS (HI REDATE) +1
DEGREES	把弧度转换成角度	DEGREES (ANGLE)
DBCLOB	把变量转换成 DBCLOB 类型	DBCLOB (GRAPHCOL)
DECIMAL	把数值转换成指定格式的实型变量	DECIMAL (AVG (SALARY), 8, 2)
DIGITS	把数值转成由其数字组成的字符串, 正负号将被舍去	DIGITS (COLUMNX)
DOUBLE	把变量转为 DOUBLE 类型	DOUBLE (SALARY)
EXP	返回参数的自然指数函数	EXP (DOUBLEVAL)
FLOAT	变量转换为浮点类型	FLOAT (SALARY) / COMM
FLOOR	数值变量向下取整	FLOOR (SALARY)
GRAPHIC	把变量转为 GRAPHIC 类型	GRAPHIC (DBCLOB COL)
HEX	返回变量值的十六进制表示	HEX (BCHARCOL)
HOURL	返回时间变量中的小时数	HOURL (TIMECOL) > 12
IFNULL	返回两个参数中第一个非空的值	IFNULL (SMALLINT1, 100) + SMALLINT2 > 1000
INSERT	接收 4 个参数, 它把第四个参数插入到第一个参数的由第二个参数指定的位置, 再从此位起删除第三个参数指定的字节数.	INSERT (' ABCDEF' , 2, 3, ' ****') 返 回 ' A***EF'
INTEGER	把数字或表示数字的字符串转成整数, 小数部分被舍去	INTEGER (AVG (SALARY) +. 5)
JULIAN_DAY	把日期转换为从公元前 4712 年 1 月 1 日起的天数	JULIAN_DAY (HI REDATE)
LEFT	返回字符串左数长度的子串	LEFT (LASTNAME, 1)
LENGTH	返回参数长度	LENGTH (ADDRESS)
LOG	返回自然对数	LOG (NUMVAL)
LOG10	返回以 10 为底的对数	LOG10 (NUMVAL)
LTRIM	返回 去掉最左侧空格后的字符串	LTRIM (LASTNAME)
MICROSECOND	取得时间的毫秒	MICROSECOND (TSTMP COL) <> 0

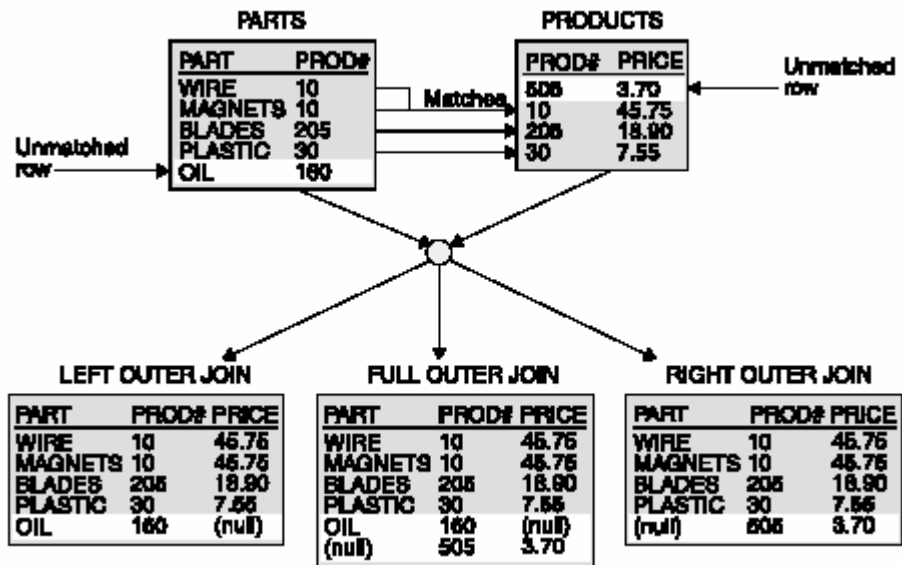
3-3 查询语句中使用函数（续页 2）

函数	返回	示例
MIDNIGHT_SECONDS	取得从 00:00:00 开始的秒数	MIDNIGHT_SECONDS(TIMECOL)
MINUTE	返回时间的分钟数	MINUTE(TIMECOL) = 0
MOD	取两个整数相除的余数	MOD(NUMCOL1, NUMCOL2)
MONTH	取得日期型变量的月份	MONTH(BIRTHDATE) = 5
NULLIF	两个变量相等时返回 NULL 否则返回第一个变量	NULLIF(SALARY, 0)
POSSTR	返回第二个字符串在第一个中第一次出现的位置	POSSTR(NOTE_TEXT, 'Quintana')
POWER	返回变量的若干次方	POWER(DOUBLECOL, 2)
QUARTER	把日期变量转换为第几季度	QUARTER(HIREDATE)
RADIANS	返回整型角度的弧度值	RADIANS(ANGLE)
RAND	返回 0-1 间的随机数, 输入的是种子数字	RAND(SEEDVAL)
REAL	转换成实型数字	REAL(SALARY)
REPEAT	返回指定字符串重复若干遍后的值	REPEAT(' * ', 72)
REPLACE	把第一个变量里包含的第二个变量用第三个替换	REPLACE(DEPTNAME, '_ ', '-')
RIGHT	取字符串的右起若干字符	RIGHT(LASTNAME, 5)
ROUND	把第一个变量在第二个变量指定的小数位数上四舍五入	ROUND(SALARY, 0)
ROWID	把变量转为 ROWID 类型	ROWID(: rowidvar)
RTRIM	去掉字符串右侧空格	RTRIM(LASTNAME)
SECOND	取得时间里的秒数	SECOND(RECEIVED)
SIGN	根据值是负, 0, 正, 返回 -1, 0, 或 1	SIGN(NUMCOL)
SIN	返回弧度的正弦值	SIN(ANGLE)
SINH	返回弧度的双曲正弦值	SINH(ANGLE)
SMALLINT	转换成 SMALLINT 类型	SMALLINT(SALARY)
SPACE	返回指定数目的空格组成的字符串	SPACE(3)
SQRT	返回平方根	SQRT(NUMCOL)
STRIP	清除字符里的所有空格	STRIP(LASTNAME, TRAILING)
SUBSTR	取子字符串	SUBSTR(FIRSTNAME, 2, 3)
TIME	转成 TIME 类型	TIME(TSTMPCOL) < '13:00:00'
TIMESTAMP 转成 TIMESTAMP 类型	TIMESTAMP(DECIMAL, TIMECOL)	
TRANSLATE	把字符串里的第二个变量出现的地方替以第三个变量	TRANSLATE(DEPTNAME, '_ ', '-')

3-3 查询语句中使用函数（续页 3）

函数	返回	示例
TRUNCATE	把变量的小数部分数目截短到第二个变量指定的长度	TRUNCATE(SALARY, 0)
UPPER or UCASE	把变量转换成大写	UPPER(DEPTNAME)
VALUE	对于 NULL 的列, 返回一个值来代替它	VALUE(SMALLINT1, 100) + SMALLINT2 > 1000
VARCHAR	转成 VARCHAR 类型	VARCHAR(JOB)
VARGRAPHIC	转成 VARGRAPHIC 类型	VARGRAPHIC('single-byte')
WEEK	返回日期是本年第多少星期	WEEK(BIRTHDATE)
YEAR	返回年数	YEAR(BIRTHDATE) = 1956

3-4 从一个以上表中查询数据



PARTS 表

PART	PROD#	SUPPLIER
=====		
WIRE	10	ACWF
OIL	160	WESTERN_CHEM
MAGNETS	10	BATEMAN
PLASTIC	30	PLASTIK_CORP
BLADES	205	ACE_STEEL

PRODUCTS 表

PROD#	PRODUCT	PRICE
=====		
505	SCREWDRIVER	3.70
30	RELAY	7.55
205	SAW	18.90
10	GENERATOR	45.75

实际的数据查询中，往往只从一个表中查询数据是不够用的，需要一种方法把一个以上的表的数据根据一定的逻辑联合起来查询。从一个以上的表进行查询使用 JOIN 子句，基本的方式有 INNER JOIN 和 OUTER JOIN 两种：

INNER JOIN：

INNER JOIN 可以指定一定的条件，把符合条件的数据从两个表里提取出来。INNER JOIN 只选出两表按指定条件严格相等的记录。以下两种写法，直接写 WHERE 条件和 INNER JOIN ON 是等价的

```

SELECT PART, SUPPLIER, PARTS. PROD#, PRODUCT
  FROM PARTS, PRODUCTS WHERE PARTS. PROD#=PRODUCTS. PROD#;
SELECT PART, SUPPLIER, PARTS. PROD#, PRODUCT
  FROM PARTS INNER JOIN PRODUCTS
    ON PARTS. PROD# = PRODUCTS. PROD#;
```

3-4 从一个以上表中查询数据（续页 1）

结果如下：

PART	SUPPLIER	PROD#	PRODUCT
=====	=====	=====	=====
WIRE	ACWF	10	GENERATOR
MAGNETS	BATEMAN	10	GENERATOR
PLASTIC	PLASTIK_CORP	30	RELAY
BLADES	ACE_STEEL	205	SAW

INNER JOIN 只从两个表里提取符合 ON 条件的记录，如上面 PARTS 表的“OIL 160 WESTERN_CHEM”的记录因为 PRODUCTS 表里没有相等的记录，所以就不会出现在查询结果里。对于同一个 ON 的值在表里有多多个的情况，结果会以笛卡尔积的方式出现，也就是第一个表里的一行与第二个表中符合条件的每一行都进行组合生成一个结果行。在 WHERE 子句中，除了给出 JOIN 的条件外，还可以添加任何需要的条件，如：

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
FROM PARTS INNER JOIN PRODUCTS
ON PARTS.PROD# = PRODUCTS.PROD# AND SUPPLIER NOT LIKE 'A%';
```

PART	SUPPLIER	PROD#	PRODUCT
=====	=====	=====	=====
MAGNETS	BATEMAN	1	GENERATOR
PLASTIC	PLASTIK_CORP	3	RELAY

同一个表自身的 INNER JOIN：

INNER JOIN 的表也可以是同一个，如下面的语句，用来查找一个表里某个字段重复的行的清单：

```
SELECT DISTINCT(A.PART), A.PROD# FROM PARTS A JOIN PARTS B
ON A.PROD#=B.PROD# AND A.PART<>B.PART;
```

PART	PROD#
-----+-----+	
MAGNETS	10
WIRE	10

上面的 SELECT 语句中由于 JOIN 前后是一个表，两个表的列名都相同，所以用到列时必须指定是哪个表的，为了简便，上例中为每个表起了一个别名。

Full outer join :

除 INNER JOIN 外，还有 OUTER JOIN，下面的例子就是 FULL OUTER JOIN：

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT
FROM PARTS FULL OUTER JOIN PRODUCTS
ON PARTS.PROD# = PRODUCTS.PROD#;
```

PART	SUPPLIER	PROD#	PRODUCT
WIRE	ACWF	10	GENERATOR
MAGNETS	BATEMAN	10	GENERATOR
PLASTIC	PLASTIK_CORP	30	RELAY
OIL	WESTERN_CHEM	160	-----
BLADES	ACE_STEEL	205	SAW
-----	-----	-----	SCREWDRI VER

FULL OUTER JOIN 把两个表的所有符合条件的记录都列出来，如果在 ON 的条件上，另一个表里没有与之相等的记录，则显示 NULL 值。

Left outer join :

LEFT OUTER JOIN 把左面那个表的所有记录都列出来，如果在 ON 的条件上，另一个表里没有与之相等的记录，则显示 NULL 值。

```
SELECT PART, SUPPLIER, PARTS.PROD#, PRODUCT, PRICE
FROM PARTS LEFT OUTER JOIN PRODUCTS
ON PARTS.PROD#=PRODUCTS.PROD# AND PRODUCTS.PRICE>10.00;
```

PART	SUPPLIER	PROD#	PRODUCT	PRICE
WIRE	ACWF	10	GENERATOR	45.75
MAGNETS	BATEMAN	10	GENERATOR	45.75
PLASTIC	PLASTIK_CORP	30	-----	-----
OIL	WESTERN_CHEM	160	-----	-----
BLADES	ACE_STEEL	205	SAW	18.90

Right outer join :

RIGHT OUTER JOIN 把右面那个表的所有的记录都列出来，如果在 ON 的条件上，另一个表里没有与之相等的记录，则显示 NULL 值。

```
SELECT PART, SUPPLIER, PRODUCTS.PROD#, PRODUCT, PRICE
FROM PARTS RIGHT OUTER JOIN PRODUCTS
ON PARTS.PROD# = PRODUCTS.PROD# AND PRODUCTS.PRICE>10.00;
```

PART	SUPPLIER	PROD#	PRODUCT	PRICE
WIRE	ACWF	10	GENERATOR	45.75
MAGNETS	BATEMAN	10	GENERATOR	45.75
-----	-----	30	RELAY	7.55
BLADES	ACE_STEEL	205	SAW	18.90
-----	-----	505	SCREWDRI VER	3.70

3-4 从一个以上表中查询数据（续页 3）

条件放在 ON 子句和 WHERE 子句中的区别：

一个条件可以放在 ON 子句中，也可以放在 WHERE 子句中，放在 ON 子句中的条件要受 INNER 和 OUTER JOIN 的限制，在 WHERE 中则不受限制，如上页的 RIGHT OUTER JOIN 如果改写成这样：

```
SELECT PART, SUPPLIER, PRODUCTS.PROD#, PRODUCT, PRICE
FROM PARTS RIGHT OUTER JOIN PRODUCTS
ON PARTS.PROD# = PRODUCTS.PROD# WHERE PRODUCTS.PRICE>10.00;
```

结果就变为：

PART	SUPPLIER	PROD#	PRODUCT	PRICE
WIRE	ACWF	10	GENERATOR	45.75
MAGNETS	BATEMAN	10	GENERATOR	45.75
BLADES	ACE_STEEL	205	SAW	18.90

3-5 使用子查询

简单子查询：

如果你想从前面使用过的那个 EMP 表中找到所有除在 A00 部门外的单人部门领导的雇员信息查询出来时，你会发现谁是部门领导并没有记录在 EMP 表里，只在 DEPT 表中记载有本部门的领导人雇员号，这时就可以用到子查询：

```
SELECT EMPNO, LASTNAME, WORKDEPT FROM DSN8710.EMP
WHERE EMPNO IN
      (SELECT MGRNO FROM DSN8710.DEPT
       WHERE DEPTNO <> 'A00');
```

EMPNO	LASTNAME	WORKDEPT
000020	THOMPSON	B01
000030	KWAN	C01
000050	GEYER	E01
000060	STERN	D11
000070	PULASKI	D21
000090	HENDERSON	E11
000100	SPENSER	E21

DB2 会先进行里面的一个查询，并返回如下结果：

```
SELECT MGRNO FROM DSN8710.DEPT WHERE DEPTNO <> 'A00';
```

MGRNO
000020
000030

000060
000070
000050
000090
000100

然后把上面的查询变成与

```
SELECT EMPNO, LASTNAME, WORKDEPT FROM DSN8710.EMP
WHERE EMPNO IN('000020', '000030', '000060',
               '000070', '000050', '000090', '000100');
```

一样进行处理

3-5 使用子查询（续页 1）

子查询的动词：

子查询在句子里都是跟在一定的动词后面起作用，这些动词有基本动词“=”、“>”、“<”、等和 ALL, ANY, IN, EXISTS。

基本动词用法如：

```
SELECT EMPNO, LASTNAME, SALARY FROM DSN8710.EMP
WHERE EDLEVEL >(SELECT AVG(EDLEVEL) FROM DSN8710.EMP);
```

ALL 用法如：

```
WHERE expression > ALL (subquery)
```

ANY 用法如：

```
WHERE expression > ANY (subquery)
```

IN 的用法见上页。

EXISTS 用法如：

```
SELECT DEPTNO,DEPTNAME FROM DSN8710.DEPT WHERE EXISTS(
SELECT * FROM DSN8710.EMP WHERE WORKDEPT=DEPTNO AND SALARY>40000);
```

```
DEPTNO  DEPTNAME
```

```
-----+-----+-----+-----+
```

```
A00      SPIFFY COMPUTER SERVICE DIV.
```

```
B01      PLANNING
```

```
E01      SUPPORT SERVICES
```


第四章 使用 DB2 编目

- SYSIBM.SYSCHECKS 表
- SYSIBM.SYSCOLUMNS 表
- SYSIBM.SYSDATABASE 表
- SYSIBM.SYSFOREIGNKEYS 表
- SYSIBM.SYSINDEXES 表
- SYSIBM.SYSKEYS 表
- SYSIBM.SYSTABLES 表

4-1 什么是 DB2 编目

DB2 编目 (CATALOG) 是 DB2 系统用来记录数据库里所有对象信息的数据结构，它由 12 个表空间和 63 个数据表组成，这些表是 DB2 的数据字典，记载有 DB2 的对象 (STOGROUPS、DATABASE、TABLESPACES、PARTITIONS、TABLES、VIEWS、INDEXES、KEYS 等等)、安全性、实用程序、分布计算、环境等信息。DB2 进行如建表、建视图、建索引等操作的时候，都会把信息记入 DB2 编目里，因此 DB2 编目是我们了解 DB2 系统情况的重要手段。

下面是一些常用 DB2 CATALOG 的情况：

DB2 编目	数据内容
SYSIBM.SYSCHECKS	存放系统中的 CHECK 约束的信息，系统为每个表的每一个 CHECK 约束建立一条记录。
SYSIBM.SYSCOLUMNS	存放系统中所有表的数据列的描述信息，系统为 DB2 里定义的每个表的每一行建立一条记录。
SYSIBM.SYSDATABASE	系统为每个 DATABASE 保存一条记录
SYSIBM.SYSFOREIGNKEYS	每个 FOREIGN KEY 约束建立一条记录
SYSIBM.SYSINDEXES	每个索引建立一条记录。
SYSIBM.SYSKEYS	系统为每个 INDEX 的每个键值创建一条记录
SYSIBM.SYSTABLES	系统为每一个表、视图和别名在 SYSIBM.SYSTABLES 表中创建一行记录。

4-2 SYSIBM.SYSCHECKS 表

SYSIBM.SYSCHECKS 表用来存放系统中的 CHECK 约束的信息，系统为每个表的每一个 CHECK 约束建立一条记录。

列名	数据类型	说明
TBOWNER	CHAR(8) NOT NULL	CHECK 所在 TABLE 的创建用户
CREATOR	CHAR(8) NOT NULL	CHECK 创建者 ID
DBID	SMALLINT NOT NULL	CHECK 所存放的数据库 ID
OBID	SMALLINT NOT NULL	内部使用
TIMESTAMP	TIMESTAMP NOT NULL	CHECK 创建时间
RBA	CHAR(6) FOR BIT DATA NOT NULL	CHECK 创建时的 LOG RBA
IBMREQD	CHAR(1) NOT NULL	IBM 内部使用
TBNAME	VARCHAR(18) NOT NULL	CHECK 所在的表名
CHECKNAME	VARCHAR(128) NOT NULL	CHECK 名
CHECKCONDITION	VARCHAR(3800) NOT NULL	CHECK 的内容

如查询 EMP 表上的索引：

```
SELECT TBNAME, CHECKNAME, CHECKCONDITION FROM SYSIBM.SYSCHECKS
      WHERE TBNAME='EMP' ;
```

```
-----+-----+-----+-----+-----+-----+
TBNAME          CHECKNAME          CHECKCONDITION
-----+-----+-----+-----+-----+
EMP              NUMBER              PHONENO >= '0000' AND PHONENO <= '9999'
EMP              PERSON              SEX = 'M' OR SEX = 'F'
```

4-3 SYSIBM.SYSCOLUMNS 表

SYSIBM.SYSCOLUMNS 表中存放有系统中所有表的数据列的描述信息，系统为 DB2 里定义的每个表的每一行建立一条记录。

列名	数据类型	说明
NAME	VARCHAR(18) NOT NULL	列名
TBNAME	VARCHAR(18) NOT NULL	表名
TBCREATOR	CHAR(8) NOT NULL	表的创建者用户名
COLNO	SMALLINT NOT NULL	列序号
COLTYPE	CHAR(8) NOT NULL	数据类型，具体值和含义如下： INTEGER : Large integer SMALLINT : Small integer FLOAT : Floating-point CHAR : Fixed-length character string VARCHAR : Varying-length character string LONGVAR : Varying-length character string DECIMAL : Decimal GRAPHIC : Fixed-length graphic string VARG : Varying-length graphic string LONGVARG : Varying-length graphic string DATE : Date TIME : Time TIMESTMP : Timestamp BLOB : Binary large object CLOB : Character large object DBCLOB : Double-byte character large object ROWID : Row ID data type DISTINCT : Distinct type
LENGTH	SMALLINT NOT NULL	列的长度，具体值如下： INTEGER : 4 SMALLINT : 2 FLOAT : 4 or 8 CHAR : Length of string VARCHAR : Maximum length of string LONGVAR : Maximum length of string DECIMAL : Precision of number GRAPHIC : Number of DBCS characters VARG : Maximum number of DBCS characters LONGVARG : Maximum number of DBCS characters DATE : 4 TIME : 3 TIMESTMP : 10
SCALE	SMALLINT NOT NULL	十进制字长

NULLS	CHAR(1) NOT NULL	是否允许为 NULL N: No, Y: Yes
HIGH2KEY	CHAR(8) NOT NULL FOR BIT DATA	RUNSTAT 生成的统计信息
LOW2KEY	CHAR(8) NOT NULL FOR BIT DATA	RUNSTAT 生成的统计信息
UPDATES	CHAR(1) NOT NULL	此列是否允许更新, N: No, Y: Yes
IBMREQD	CHAR(1) NOT NULL	IBM 使用的标志
REMARKS	VARCHAR(254) NOT NULL	描述该列的, 由 COMMENT ON 语句为列指定的信息
DEFAULT	CHAR(1) NOT NULL	列的省缺值类型 N: 无省缺值 A: 是 'ROWID' 类型 (COLTYPE='ROWID') 并且使用 GENERATED ALWAYS 属性 B: 有根据数据类型而异的省缺值 (数字型是 0, 定长字符是空, 变长字符长度为 0, 时间类型是当前时间) D: 是 'ROWID' 类型 (COLTYPE='ROWID') 并使用 GENERATED BY DEFAULT 属性 I: 定义为序号 (AS IDENTITY) 类型, 并使用 GENERATED ALWAYS 属性. J: 定义为序号 (AS IDENTITY) 类型, 并使用 GENERATED BY DEFAULT 属性 S: SQL 权限 ID (用户名) 作为默认值 U: USER 专用寄存器的值作默认值 Y: 若列允许 NULL, 使用 NULL 做默认值, 否则根据数据类型不同, 数字型是 0, 定长字符是空, 变长字符长度为 0, 时间类型是当前时间 1: 使用 DEFAULTVALUE 列里的字符串作默认值 2: 把 DEFAULTVALUE 列里的内容转成浮点数做默认值 3: 把 DEFAULTVALUE 列里的内容转成十进制数做默认值 4: 把 DEFAULTVALUE 列里的内容转成整数作默认值 5: 把 DEFAULTVALUE 列里的十六进制数作默认值
KEYSEQ	SMALLINT NOT NULL	列在表主键中的位置
FOREIGNKEY	CHAR(1) NOT NULL	标志字符列的特征
FLDPROC	CHAR(1) NOT NULL	标志此列是否有域处理程序
LABEL	VARCHAR(30) NOT NULL	由 LABEL ON 语句为此列指定的标号
STATSTIME	TIMESTAMP NOT NULL WITH DEFAULT	RUNSTATS 对此列进行统计的时间
DEFAULTVALUE	VARCHAR(512) NOT NULL WITH DEFAULT	列默认值
COLCARDF	FLOAT NOT NULL WITH DEFAULT	RUNSTAT 生成的统计信息
LENGTH2	INTEGER NOT NULL WITH DEFAULT	从该列取得数据的最大长度

DATATYPEID	INTEGER NOT NULL WITH DEFAULT	该数据类型的内部标志符
SOURCETYPEID	INTEGER NOT NULL WITH DEFAULT	源数据的内部标志符
TYPESHEMA	CHAR(8) NOT NULL WITH DEFAULT 'SYSIBM'	
TYPENAME	VARCHAR(18) NOT NULL WITH DEFAULT	
CREATEDTS	TIMESTAMP NOT NULL WITH DEFAULT	列创建时间

一般通过查询此表来了解数据表的定义情况，如：

```
SELECT NAME, COLNO, COLTYPE, LENGTH, NULLS, DEFAULT, DEFAULTVALUE
FROM SYSIBM.SYSCOLUMNS WHERE TBNAME='DEPT' ORDER BY COLNO;
```

```
-----+-----+-----+-----+-----+-----+-----+
NAME          COLNO  COLTYPE  LENGTH  NULLS  DEFAULT  DEFAULTVALUE
-----+-----+-----+-----+-----+-----+-----+
DEPTNO              1   CHAR        3    N      N
DEPTNAME            2  VARCHAR       36    N      N
MGRNO               3   CHAR        6    Y      Y
ADMRDEPT            4   CHAR        3    N      N
LOCATION             5   CHAR       16    Y      Y
```

4-4 SYSIBM.SYSDATABASE 表

系统为每个 DATABASE 保存一条记录

列名	数据类型	说明
NAME	CHAR(8) NOT NULL	数据库名
CREATOR	CHAR(8) NOT NULL	数据库拥有者
STGROUP	CHAR(8) NOT NULL	storage group 名
BPOOL	CHAR(8) NOT NULL	数据库默认 buffer pool 名
DBID	SMALLINT NOT NULL	数据库内部标号
IBMQD	CHAR(1) NOT NULL	IBM 使用标志
CREATEDBY	CHAR(8) NOT NULL WITH DEFAULT	数据库创建者
TIMESTAMP	TIMESTAMP NOT NULL WITH DEFAULT	从 DB2 V6 起不再使用
TYPE	CHAR(1) NOT NULL WITH DEFAULT	数据库类型, W: 工作文件数据库, 空白: 非工作文件数据库
GROUP_MEMBER	CHAR(8) NOT NULL WITH DEFAULT	DB2 共享组名
CREATEDTS	TIMESTAMP NOT NULL WITH DEFAULT	数据库创建时间
ALTEREDTS	TIMESTAMP NOT NULL WITH DEFAULT	最近一次 ALTER 时间
ENCODING_SCHEME	CHAR(1) NOT NULL WITH DEFAULT 'E'	数据库数据编码, E: EBCDIC, A: ASCII
SBCS_CCSID	INTEGER NOT NULL WITH DEFAULT	默认 SBCS_CCSID
DBCS_CCSID	INTEGER NOT NULL WITH DEFAULT	默认 DBCS_CCSID
MIXED_CCSID	INTEGER NOT NULL WITH DEFAULT	默认 MIXED_CCSID

4-5 SYSIBM.SYSFOREIGNKEYS 表

SYSIBM.SYSFOREIGNKEYS 表为每个 FOREIGN KEY 约束建立一条记录

列名	数据类型	说明
CREATOR	CHAR(8) NOT NULL	FOREIGN KEY 所在表的属主
TBNAME	VARCHAR(18) NOT NULL	表名
RELNAME	CHAR(8) NOT NULL	约束名称
COLNAME	VARCHAR(18) NOT NULL	约束的列名
COLNO	SMALLINT NOT NULL	列在表中的序号
COLSEQ	SMALLINT NOT NULL	列在外键中的序号
IBMREQD	CHAR(1) NOT NULL	IBM 使用的标志

例：

```
SELECT TBNAME, RELNAME, COLNAME, COLNO, COLSEQ
FROM SYSIBM.SYSFOREIGNKEYS WHERE TBNAME='EMP' ;
```

```
-----+-----+-----+-----+-----+-----
TBNAME          RELNAME  COLNAME          COLNO  COLSEQ
-----+-----+-----+-----+-----+-----
EMP              RED      WORKDEPT              5      1
```


4-6 SYSIBM.SYSINDEXES 表

SYSIBM.SYSINDEXES 表为每个索引建立一条记录。

列名	数据类型	说明
NAME	VARCHAR(18) NOT NULL	INDEX 名
CREATOR	CHAR(8) NOT NULL	创建 INDEX 的用户
TBNAME	VARCHAR(18) NOT NULL	INDEX 所在表名
TBCREATOR	CHAR(8) NOT NULL	表的创建者
UNIQUERULE	CHAR(1) NOT NULL	是否唯一索引，取值含义如下： D No (duplicates are allowed) U Yes P Yes, and it is a primary index C Yes, and it is an index used to enforce UNIQUE constraint N Yes, and it is defined with UNIQUE WHERE NOT NULL R Yes, and it is an index used to enforce the uniqueness of a non-primary parent key G Yes, and it is an index used to enforce the uniqueness of values in a column defined as ROWID
COLCOUNT	SMALLINT NOT NULL	KEY 包含的列数
CLUSTERING	CHAR(1) NOT NULL	是否 CLUSTER 索引, N: No, Y: Yes
CLUSTERED	CHAR(1) NOT NULL	表是否真的按照 CLUSTER 索引组织了数据
DBID	SMALLINT NOT NULL	数据库 ID
OBID	SMALLINT NOT NULL	索引的内部对象 ID
ISOBJID	SMALLINT NOT NULL	索引空间的内部对象 ID
DBNAME	CHAR(8) NOT NULL	数据库名
INDEXSPACE	CHAR(8) NOT NULL	索引空间名
NLEAF	INTEGER NOT NULL	索引中的活动叶子页面数
NLEVELS	SMALLINT NOT NULL	索引中的叶子页面数
BPOOL	CHAR(8) NOT NULL	INDEX 使用的 buffer pool 名
PGSIZE	SMALLINT NOT NULL	索引页的字节数
ERASERULE	CHAR(1) NOT NULL	索引删除时它所用的数据文件是否删除
CLOSERULE	CHAR(1) NOT NULL	DB2 达到最大打开文件数时, 可否关闭此索引的数据集
SPACE	INTEGER NOT NULL	磁盘上为索引分配了多少 K 的空间
IBMREQD	CHAR(1) NOT NULL	IBM 使用的标志
CLUSTERRATIO	SMALLINT NOT NULL WITH DEFAULT	按 CLUSTER INDEX 次序存放的记录的比例
CREATEDBY	CHAR(8) NOT NULL WITH DEFAULT	INDEX 的创建者

STATTIME	TIMESTAMP NOT NULL WITH DEFAULT	最近一次 RUNSTATS 的时间
INDEXTYPE	CHAR(1) NOT NULL WITH DEFAULT	INDEX 类型
FIRSTKEYCARDF	FLOAT NOT NULL WITH DEFAULT -1	INDEX 的第一列有多少种不同的值
FULLKEYCARDF	FLOAT NOT NULL WITH DEFAULT -1	整个 INDEX 有多少种不同的值
CREATEDTS	TIMESTAMP NOT NULL WITH DEFAULT	建立 INDEX 的时间
ALTEREDTS	TIMESTAMP NOT NULL WITH DEFAULT	最近一次修改 INDEX 的时间
PIECESIZE	INTEGER NOT NULL WITH DEFAULT	存储 INDEX 的单个数据集的最大大小(KB)
COPY	CHAR(1) NOT NULL WITH DEFAULT 'N'	索引是否指定了 COPY YES
COPYLRN	CHAR(6) NOT NULL WITH DEFAULT X'000000000000'	COPY YES 时放索引建立时的 RBA, 否则全 0
CLUSTERRATIOF	FLOAT NOT NULL WITH DEFAULT	按 CLUSTER 顺序排列的记录百分比

例：

```
SELECT NAME, TBNAME, UNIQUERULE, COLCOUNT, CLUSTERING
FROM SYSIBM.SYSINDEXES WHERE TBNAME='EMP'
```

```
-----+-----+-----+-----+-----+-----+-----+
NAME          TBNAME          UNIQUERULE  COLCOUNT  CLUSTERING
-----+-----+-----+-----+-----+-----+
XEMP1          EMP              P              1      Y
XEMP2          EMP              D              1      N
```

INDEX 具体建立在那些列上，还要去看 SYSIBM.SYSKEYS 表

4-7 SYSIBM.SYSKEYS 表

系统为每个 INDEX 的每个键值创建一条记录

列名	数据类型	说明
IXNAME	VARCHAR(18) NOT NULL	INDEX 名
IXCREATOR	CHAR(8) NOT NULL	INDEX 的创建者
COLNAME	VARCHAR(18) NOT NULL	列名
COLNO	SMALLINT NOT NULL	列在表里的序号
COLSEQ	SMALLINT NOT NULL	列在 INDEX 里的序号
ORDERING	CHAR(1) NOT NULL	索引的升序降序, A: 升序, D: 降序
IBMREQD	CHAR(1) NOT NULL	IBM 用的标志

例：

```
SELECT IXNAME, COLNAME, COLNO, COLSEQ, ORDERING
FROM SYSIBM.SYSKEYS WHERE IXNAME='XEMP1' ORDER BY COLSEQ;
```

```
-----+-----+-----+-----+-----+-----+-----
IXNAME          COLNAME          COLNO  COLSEQ  ORDERING
-----+-----+-----+-----+-----+-----+-----
XEMP1           EMPNO                1      1  A
```

4-8 SYSIBM.SYSTABLES 表

系统为每一个表,视图和别名在 SYSIBM.SYSTABLES 表中创建一行记录.

列名	数据类型	说明
NAME	VARCHAR(18) NOT NULL	表,视图或别名的名字
CREATOR	CHAR(8) NOT NULL	表的拥有者
TYPE	CHAR(1) NOT NULL	类型,取值如下: A: 别名, G: 临时全局表, T: 表, V: 视图, X: 辅助表
DBNAME	CHAR(8) NOT NULL	所在 DATABASE 名
TSNAME	CHAR(8) NOT NULL	所在 TABLESPACES 名
DBID	SMALLINT NOT NULL	DATABASE 标号
OBID	SMALLINT NOT NULL	表的对象标号
COLCOUNT	SMALLINT NOT NULL	列数
EDPROC	CHAR(8) NOT NULL	edit procedure 名
VALPROC	CHAR(8) NOT NULL	validation procedure 名
CLUSTERTYPE	CHAR(1) NOT NULL	是否使用了 RESTRICT ON DROP 选项
NPAGES	INTEGER NOT NULL	表的总页数, RUNSTATS 生成
PCTPAGES	SMALLINT NOT NULL	活动页的比例, RUNSTATS 生成
IBMREQD	CHAR(1) NOT NULL	IBM 使用的标志
REMARKS	VARCHAR(254) NOT NULL	存放建表时 COMMENT ON 子句的信息
PARENTS	SMALLINT NOT NULL	参照其他表的约束的个数
CHILDREN	SMALLINT NOT NULL	被其他表参照的个数
KEYCOLUMNS	SMALLINT NOT NULL	主键列数
RECLENGTH	SMALLINT NOT NULL	所有行最大长度
STATUS	CHAR(1) NOT NULL	表定义的状态, I: 表定义的不完整 TABLESTATUS 列里是原因, X: 定义完整有主键, 空格: 定义完整无主键
KEYOBID	SMALLINT NOT NULL	为保证主键唯一而创建的唯一索引 的对象 ID
LABEL	VARCHAR(30) NOT NULL	建表时用 LABEL ON 指定的信息
CHECKFLAG	CHAR(1) NOT NULL WITH DEFAULT	CHECK 状态, C: 表处于 check pending 状态, 空格: 正常
AUDITING	CHAR(1) NOT NULL WITH DEFAULT	审计状态, A: AUDIT ALL, C: AUDIT CHANGE,

		blank : AUDIT NONE
CREATEDBY	CHAR(8) NOT NULL WITH DEFAULT	表的创建者
LOCATION	CHAR(16) NOT NULL WITH DEFAULT	别名对象的实际名字的位置
TBCREATOR	CHAR(8) NOT NULL WITH DEFAULT	别名所指实际对象的创建者
TBNAME	VARCHAR(18) NOT NULL WITH DEFAULT	别名所引用的表的实际名字
CREATEDTS	TIMESTAMP NOT NULL WITH DEFAULT	创建时间
ALTEREDTS	TIMESTAMP NOT NULL WITH DEFAULT	最后更改时间
DATA CAPTURE	CHAR(1) NOT NULL WITH DEFAULT	记录 DATA CAPTURE 选项的值
RBA1	CHAR(6) NOT NULL WITH DEFAULT FOR BIT DATA	表创建时 LOG 的 RBA 值
RBA2	CHAR(6) NOT NULL WITH DEFAULT FOR BIT DATA	表修改时的 LOG RBA 值
PCTROWCOMP	SMALLINT NOT NULL WITH DEFAULT	活动行的被压缩比例, RUNSTATS 生成
STATSTIME	TIMESTAMP NOT NULL WITH DEFAULT	最近 RUNSTATS 的时间
CHECKS	SMALLINT NOT NULL WITH DEFAULT	表上定义的 CHECK 数目
CARDF	FLOAT NOT NULL WITH DEFAULT -1	表中总行数
CHECKRID5B	CHAR(5) NOT NULL WITH DEFAULT	第一个违反约束的 行的 ID
ENCODING_SCHEME	CHAR(1) NOT NULL WITH DEFAULT 'E'	表的数据编码, E: EBCDIC, A: ASCII
TABLESTATUS	VARCHAR(10) NOT NULL WITH DEFAULT	表定义不完整的原因, L: 没有为 LOB 定义辅助表, P: 没为主键定义索引, R: 没有在 ROWID 上定义索引

例 :

```
SELECT NAME, COLCOUNT, KEYCOLUMNS, CREATEDTS
      FROM SYSIBM.SYSTABLES WHERE CREATOR='DSN8710' AND TYPE='T' ;
```

```
-----+-----+-----+-----+-----+-----+-----+
NAME                COLCOUNT  KEYCOLUMNS  CREATEDTS
-----+-----+-----+-----+-----+-----+-----+
TDSPTXT                3           0  2002-05-23-11.57.32.571318
DEPT                   5           1  2002-05-23-11.57.28.835709
EMP                   14           1  2002-05-23-11.57.29.436761
TCONA                  5           0  2002-05-23-11.57.32.128325
TOPTVAL               11           0  2002-05-23-11.57.32.374508
MAP_TBL                4           0  2002-05-23-11.57.33.090576
EDEPT                  7           0  2002-05-23-11.57.32.770638
EEMP                  16           0  2002-05-23-11.57.32.817518
PROJ                   8           1  2002-05-23-11.57.30.490636
PARTS                  4           0  2002-05-23-11.57.31.887246
ACT                    3           1  2002-05-23-11.57.30.919431
PROJACT                5           3  2002-05-23-11.57.31.301635
```

第二部分

关系数据库应用编程

第五章 基本 SQL 语言编程

- 嵌入式 SQL
- 声明表
- 声明宿主变量
- 使用宿主变量
- 使用指示变量
- 使用宿主结构
- 检查 SQL 语句执行结果

5-1 嵌入式 SQL

DB2 支持多种语言的编程，如 C、C++、ASSEMBLER、COBOL、FORTRAN、PL/I 等等，这些语言称为宿主语言，SQL 语句以嵌入的方式与宿主语言结合在一起，SQL 语句可以与宿主语言变量间传递数据，宿主语言以这种方法操纵数据库。SQL 语言嵌入宿主语言时，SQL 语句以“EXEC SQL”开始，结束标志根据不同语言有所差别：

语言	SQL 结束符	例子
Assembler	行结束	EXEC SQL SELECT * FROM DEPT
C	分号	EXEC SQL SELECT * FROM DEPT ;
COBOL	END-EXEC	EXEC SQL SELECT * FROM DEPT END-EXEC
FORTRAN	行结束	EXEC SQL SELECT * FROM DEPT
PL/I	分号	EXEC SQL SELECT * FROM DEPT ;

使用嵌入式 SQL 的程序先要通过 DB2 的预编译器进行预编译，把嵌入的 SQL 语句翻译成宿主语言的代码和函数，然后经过宿主语言编译器的编译，最后把 DB2 的库函数 LINK 进来，就能生成最终的可执行程序。

DSNCLI MODULE
库函数

在程序使用表或视图前，首先要声明表的结构它各个列的数据类型。虽然表的声明不是必须的，但是这样是比较好的编程风格，它的一个好处是文档作用，你能通过表的声明了解表中的各个列以及其数据类型；另一个好处是 DB2 的预编译器能根据表的声明来核对程序中为各列指定的宿主变量类型是否合适。COBOL 语言的表声明要放在 WORKING-STORAGE SECTION 或 LINKAGE SECTION 段中，C 语言中放在变量定义处。

对于前面的 DSN8710.DEPT 表，要在程序中作如下格式的表声明：

```
EXEC SQL
  DECLARE DSN8710.DEPT TABLE
    (DEPTNO CHAR(3) NOT NULL,
     DEPTNAME VARCHAR(36) NOT NULL,
     MGRNO CHAR(6) ,
     ADMRDEPT CHAR(3) NOT NULL,
     LOCATION CHAR(16) )
END-EXEC.
```

使用前声明表或是视图
用DECLARE子句。

5-3 声明宿主变量

应用程序为了和 SQL 语句进行数据交换,要使用宿主变量。宿主变量要在程序里声明。对于相同的 SQL 数据类型,在不同语言中有不同的对应数据类型,举 C 语言和 COBOL 语言中的情况如下:

C 语言中宿主变量必须在 EXEC SQL BEGIN DECLARE SECTION 和 EXEC SQL END DECLARE SECTION 之间声明:

SQL Data Type	C Data Type	Notes
SMALLINT	short int	
INTEGER	long int	
DECIMAL(p, s) or NUMERIC(p, s)	double	如果你的编译器提供 decimal 类型, 请使用 decimal
REAL or FLOAT(n)	float	1<=n<=21
DOUBLE PRECISION or FLOAT(n)	double	22<=n<=53
CHAR(1)	char	
CHAR(n)	char[n+1]	增加一个字节存储 EOS 字符
VARCHAR(n)	struct { short; char [n]; }	等价于一个结构, 结构第一个变量是 short int 型, 存储数据长度, 第二个是字符数组, 存储实际数据

例如:

```
EXEC SQL BEGIN DECLARE SECTION;
short edlevel;          /* EDLEVEL  SMALLINT*/
struct { short len;
          char  x1[15];
        } lname;         /* LASTNAME  VARCHAR(15)*/
char empno[7];          /* EMPNO    CHAR(6)*/
long salary;            /* SALARY   DECIMAL(9,2)*/
EXEC SQL END  DECLARE SECTION;
```

COBOL 语言中:

SQL Data Type	COBOL Data Type	Notes
<u>SMALLINT</u>	<u>S9(4) COMP-4 or BINARY</u>	
<u>INTEGER</u>	<u>S9(9) COMP-4 or BINARY</u>	
<u>DECIMAL(p, s) or NUMERIC(p, s)</u>	<u>S9(p-s)V9(s) COMP-3 or</u> S9(p-s)V9(s) PACKED-DECIMAL DISPLAY SIGN LEADING SEPARATE	0<=s<=p<=31
REAL or FLOAT (n)	COMP-1	1<=n<=21
DOUBLE PRECISION, DOUBLE or	COMP-2	22<=n<=53

DECIMAL ((P,S) 表示P位数 S位小数 用COBOL表示就是9 (P-S)V(S)

varchar(n)在COBOL中对应的数据类型包含两种变量,一个存数据长度,用PIC S9(4) COMP表示,另一个存实际数据用PIC X(n)表示

Float (n)		
CHAR(n)	X(n)	1<=n<=255
<u>VARCHAR(n)</u>	<u>PIC S9(4) USAGE COMP.</u> <u>PIC X(n).</u>	<u>等价于包含两个变量的结构</u> <u>第一个存数据长度,第二个存</u> <u>实际数据</u>

例：

```

01 H-EMPTBL.
   05 H-EMPNO   PIC X(6).
   05 H-NAME.
       49 H-NAME-LN   PIC S9(4) COMP-4.
       49 H-NAME-DA   PIC X(32).
   05 H-ADDRESS.
       49 H-ADDRESS-LN   PIC S9(4) COMP-4.
       49 H-ADDRESS-DA   PIC X(36).
   05 H-EMPLOC   PIC X(4).
   05 H-SSNO     PIC X(11).
   05 H-BORN     PIC X(10).
   05 H-SEX      PIC X(1).
   05 H-HIRED    PIC X(10).
   05 H-DEPTNO   PIC X(3).
   05 H-JOBCODE  PIC S9(3)V COMP-3.
   05 H-SRATE    PIC S9(5) COMP.
   05 H-EDUC     PIC S9(5) COMP.
   05 H-SAL      PIC S9(6)V9(2) COMP-3.
   05 H-VALIDCHK PIC S9(6)V COMP-3.

```

DATE是字符型数据

各字段依次对应如下表的相应列：

SYSADM.EMP TABLE

```

(EMPNO CHAR(6) NOT NULL,
 NAME  VARCHAR(32),
 ADDRESS VARCHAR(36) ,
 EMPLOC CHAR(4) NOT NULL,
 SSNO   CHAR(11),
 BORN   DATE,
 SEX    CHAR(1),
 HIRED  CHAR(10),
 DEPTNO CHAR(3) NOT NULL,
 JOBCODE DECIMAL(3),
 SRATE   SMALLINT,
 EDUC    SMALLINT,
 SAL     DECIMAL(8,2) NOT NULL,
 VALCHK  DECIMAL(6))

```

05 H-NAME.
49 H-NAME-LN PIC S9(4) COMP-4.
49 H-NAME-DA PIC X(32).

使用宿主变量取得数据：

宿主变量使用在 SQL 语句中时前面要加“：”作识别标志。从 SQL 语句中返回单行数据，使用 INTO 子句，见如下说明；返回多行数据时，要使用 CURSOR 操作，见后面的介绍。

宿主变量先在工作单元节中定义 在 SQL 语言中使用时前面要加 : 作为标示符表示这是主机变量 注意只有在 SQL 语言中才用主机变量也就是：在其他地方就是变量

例如，如果你想从 DSN8710.EMP 表中取得某一雇员的 EMPNO, LASTNAME, WORKDEPT 数据，你可先定义宿主变量，然后用如下的 INTO 子句把数据读入变量中去。

变量先定义后使用。

```
EXEC SQL
  SELECT EMPNO, LASTNAME, WORKDEPT
    INTO :CBLEMPNO, :CBLNAME, :CBLDEPT
  FROM DSN8410.EMP
  WHERE EMPNO = :EMPID
END-EXEC.
```

SELECT 子句选择的列表里，不但可以有表的列名，还可以有宿主变量，如：

```
MOVE '000220' TO PERSON.
EXEC SQL
  SELECT EMPNO, LASTNAME, SALARY, :RAISE, SALARY + :RAISE
    INTO :EMP-NUM, :PERSON-NAME, :EMP-SAL, :EMP-RAISE, :EMP-TTL
  FROM DSN8410.EMP
  WHERE EMPNO = :PERSON
END-EXEC.
```

使用宿主变量更新和插入数据：

宿主变量可以用来更新和插入数据，更新数据时把宿主变量用于 SET 子句中，插入数据时把宿主变量用于 VALUES 子句中。

例：

```
EXEC SQL
  UPDATE DSN8410.EMP
    SET PHONENO = :NEWPHONE
  WHERE EMPNO = :EMPID
END-EXEC.
```

INDICATOR

指示变量是一个 SMALL INTEGER 型的变量，它有以下功能：

- 查询时指示所查得的变量是否 NULL，输入时指示是否输入一个 NULL 值。
- 如果查询出的变量的长度比给出的宿主变量长，数据会被截断，指示变量保存字段的原长。
- 指示数据传递给宿主变量时不允许转换格式
- 时间变量传递给宿主变量时如果被截断，保存其中的秒数

指示变量的作用：
指示变量中存的值为负值或是正值为负值，表示取得的字段为空，或是在更新或是插入操作置字段为空。为正值表示表示取得的字段的原长，可以用来判断是否产生了截断，将取得的字段传给主机变量或是将主机变量中的值传给字段

查询数据时，如果查得的列是 NULL，DB2 会给与这个列的宿主变量相关的指示变量里放一个负值，假如这时你没给宿主变量绑定一个指示变量，就会引发一个错误。当一次查询完成后，如果指示变量是负值，说明这个列查得的是 NULL，相关的宿主变量的值并不发生改变；如果指示变量的值是正数，里面存放的是从表中查出数据的长度，用这个值可以检查是否发生了截断。指示变量在 C 语言中定义的数据类型是“short int”，COBOL 语言中的类型是“PIC S9(4) COMP”。指示变量的格式有两种，一种是直接在宿主变量后加“:”，后面跟指示变量名，另一种是在宿主变量和指示变量间加 INDICATOR 关键字。以下两种格式是等价的。

```
EXEC SQL
  SELECT PHONENO
    INTO :CBLPHONE:INDNULL
  FROM DSN8710.EMP
  WHERE EMPNO = :EMPID
END-EXEC.
```

```
EXEC SQL
  SELECT PHONENO
    INTO :CBLPHONE INDICATOR :INDNULL
  FROM DSN8710.EMP
  WHERE EMPNO = :EMPID
END-EXEC.
```

插入和更新操作时，使用指示变量还可以把某列设成 NULL 值。当 DB2 处理 INSERT 和 UPDATA 语句时，它先检查指示变量，如果指示变量小于 0，会直接把这列置为 NULL；如果指示变量大于等于 0，才会使用宿主变量的值。

例如以下语句

```
EXEC SQL
  UPDATE DSN8710.EMP
  SET PHONENO = :NEWPHONE:PHONEIND
  WHERE EMPNO = :EMPID
END-EXEC.
```

PHONEIND

当 PHONEIND 小于 0 时，PHONEIND 会置 NULL，否则会置成 NEWPHONE 里的值。

在一次使用多个列时，可以向下面这样把各个宿主变量罗列出来：

```
EXEC SQL
    SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT
    INTO :EMPNO, :FIRSTNME, :MIDINIT, :LASTNAME, :WORKDEPT
    FROM DSN8710.VEMP
    WHERE EMPNO = :EMPID
END-EXEC.
```

也可以把这些宿主变量先按次序放在一个结构变量里，SQL 语句直接使用这个结构名字，这时如果需要使用指示变量，也要按照变量个数指定一个指示变量数组，如下例：

<p>COBOL 语言中使用宿主结构：</p> <pre>01 PEMP-ROW. 10 EMPNO PIC X(6). 10 FIRSTNME. 49 FIRSTNME-LEN PIC S9(4) USAGE COMP. 49 FIRSTNME-TEXT PIC X(12). 10 MIDINIT PIC X(1). 10 LASTNAME. 49 LASTNAME-LEN PIC S9(4) USAGE COMP. 49 LASTNAME-TEXT PIC X(15). 10 WORKDEPT PIC X(3). 10 EMP-BIRTHDATE PIC X(10). 01 INDICATOR-TABLE. 02 EMP-IND PIC S9(4) COMP OCCURS 6 TIMES.</pre>	<p>C 语言中使用宿主结构：</p> <pre>EXEC SQL BEGIN DECLARE SECTION; struct{ char empno[7]; struct { short len; char val[15]; }first_name; char midinit; struct { short len; char val[15]; }last_name; char work_dept[4]; char emp_birthday[11]; }pemp_row; short ind[6]; EXEC SQL END DECLARE SECTION;</pre>
<p>EXEC SQL</p> <pre> SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, BIRTHDATE INTO :PEMP-ROW:EMP-IND FROM DSN8710.EMP WHERE EMPNO = :EMPNO END-EXEC.</pre>	<p>EXEC SQL</p> <pre> SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, BIRTHDATE INTO :pemp_row:ind FROM DSN8710.EMP WHERE EMPNO = :EMPNO;</pre>

EMP-IND是
一维表

5-7 检查 SQL 语句执行结果

使用 SQL 语句的程序通过一个叫 SQLCA (SQL communication area) 的数据区来与 DB2 通信当 DB2 处理完一条 SQL 语句后，把返回码放到 SQLCODE 和 SQLSTATE 变量里，程序通过判断它们的值来了解执行是否成功以及错误信息。

SQL 执行情况主要通过 SQLCODE 体现，当 SQL 语句执行后，如果 SQLCODE 等于 0，说明执行成功；如果大于 0，说明有警告信息，返回的是警告信息代码；如果小于 0，说明执行失败，返回的是失败代码。可以根据 SQLCODE 的值去查《DB2 代码信息》，找到具体的错误说明，作为修改程序的参考。当 SQL 的查询语句没有附和条件的记录时，SQLCODE 返回+100。

SQLCA 是一个结构在 DB2 中有定义，程序中使用 “EXEC SQL INCLUDE SQLCA;” 语句来包括进来。

下面是 C 语言中 SQLCA 的定义

```
SQL_STRUCTURE sqlca
{
    _SQLOLDCHAR    sqlcaid[8];           /* 标志串： = 'SQLCA' */
    long           sqlcabc;               /* SQLCA 字节长 = 136 */
    long           sqlcode;              /* SQL 返回码 */
    short          sqlerrml;              /* SQLERRMC 的长度 */
    _SQLOLDCHAR    sqlerrmc[70];         /* 字符的错误信息 */
    _SQLOLDCHAR    sqlerrp[8];           /* 诊断信息 */
    long           sqlerrd[6];           /* 诊断信息 */
    _SQLOLDCHAR    sqlwarn[11];          /* 警告标志 */
    _SQLOLDCHAR    sqlstate[5];          /* SQL 语句执行后的状态码 */
};
```

下面是 COBOL 语言中的 SQLCA 定义

```
01 SQLCA.
    05 SQLCAID      PIC X(8).
    05 SQLCABC      PIC S9(9) COMP-4.
    05 SQLCODE     PIC S9(9) COMP-4.
    05 SQLERRM.
        49 SQLERRML PIC S9(4) COMP-4.
        49 SQLERRMC PIC X(70).
    05 SQLERRP      PIC X(8).
    05 SQLERRD      OCCURS 6 TIMES
                    PIC S9(9) COMP-4.
```

5-7 检查 SQL 语句执行结果 (续页)

```
05 SQLWARN.  
    10 SQLWARN0 PIC X.  
    10 SQLWARN1 PIC X.  
    10 SQLWARN2 PIC X.  
    10 SQLWARN3 PIC X.  
    10 SQLWARN4 PIC X.  
    10 SQLWARN5 PIC X.  
    10 SQLWARN6 PIC X.  
    10 SQLWARN7 PIC X.  
05 SQLEXT.  
    10 SQLWARN8 PIC X.  
    10 SQLWARN9 PIC X.  
    10 SQLWARNA PIC X.  
    10 SQLSTATE PIC X(5).
```

SQLCODE 的用法例子：

```
EXEC SQL SELECT EMPNO, FIRSTNME, EDLEVEL, BIRTHDATE, SALARY  
        INTO :szEmpNO, :FirstName, :nEdutLevel, :szBirthDay, :lSalary  
        FROM DSN8710.EMP WHERE EMPNO='000050';  
if(SQLCODE!=0)  
{  
    printf("SQL Error, SQLCODE=%d, SQLERRMC: %s\n", SQLCODE,  
          sqlca.sqlerrmc);  
    exit(0);  
}
```


5-8 一个简单的嵌入 SQL 程序

下面是一个完整的嵌入式 SQL 程序的例子，由两个文件组成，一个是 COPY BOOK，定义表和宿主变量，另一个是 COBOL 程序。

COPY BOOK：

```
EXEC SQL DECLARE DSN8710.EMP TABLE
( EMPNO                CHAR(6) NOT NULL,
  FIRSTNME              VARCHAR(12) NOT NULL,
  MIDINIT               CHAR(1) NOT NULL,
  LASTNAME               VARCHAR(15) NOT NULL,
  WORKDEPT               CHAR(3),
  PHONENO                CHAR(4),
  HIREDATE               DATE,
  JOB                    CHAR(8),
  EDLEVEL                SMALLINT,
  SEX                    CHAR(1),
  BIRTHDATE              DATE,
  SALARY                 DECIMAL(9, 2),
  BONUS                  DECIMAL(9, 2),
  COMM                   DECIMAL(9, 2)
) END-EXEC.

01 DCLEMP.
   10 EMPNO                PIC X(6).
   10 FIRSTNME              PIC X(12).
       49 FIRSTNME-LEN      PIC S9(4) USAGE COMP.
       49 FIRSTNME-TEXT     PIC X(12).
   10 MIDINIT               PIC X(1).
   10 LASTNAME              PIC X(15).
       49 LASTNAME-LEN      PIC S9(4) USAGE COMP.
       49 LASTNAME-TEXT     PIC X(15).
   10 WORKDEPT              PIC X(3).
   10 PHONENO               PIC X(4).
   10 HIREDATE              PIC X(10).
   10 JOB                   PIC X(8).
   10 EDLEVEL               PIC S9(4) USAGE COMP.
   10 SEX                   PIC X(1).
   10 BIRTHDATE             PIC X(10).
   10 SALARY                 PIC S9(7)V9(2) USAGE COMP-3.
   10 BONUS                  PIC S9(7)V9(2) USAGE COMP-3.
   10 COMM                   PIC S9(7)V9(2) USAGE COMP-3.
```

COBOL 程序：

```
IDENTIFICATION DIVISION.
PROGRAM-ID.      SAMPLE1.
AUTHOR.          ERIC.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 EDU-LEVEL PIC 9(4).
01 MY-SQL-CODE PIC S9(9).
EXEC SQL INCLUDE SQLCA END-EXEC.
EXEC SQL INCLUDE EMP END-EXEC.
```

因为EDLEVEL不是显示型字符, COMP表示内部2进制或浮点表示
所以要定义一个变量转化成显示型字符

SQLCODE PIC S9(9) COMP-4.
为了要把SQLCODE显示出来也要定义一个变量转化

5-8 一个简单的嵌入 SQL 程序 (续页)

```
PROCEDURE DIVISION.  
    MOVE '000050' TO EMPNO.  
    EXEC SQL SELECT FIRSTNME, EDLEVEL, BIRTHDATE  
        INTO :FIRSTNME, :EDLEVEL, :BIRTHDATE  
        FROM DSN8710.EMP  
        WHERE EMPNO=:EMPNO  
    END-EXEC.  
    IF SQLCODE < 0  
    THEN  
        MOVE SQLCODE TO MY-SQL-CODE  
        DISPLAY 'SQL CODE:' MY-SQL-CODE ', ERROR:' SQLERRMC  
        GOBACK  
    END-IF.  
    MOVE EDLEVEL TO EDU-LEVEL.  
    DISPLAY 'EMPNO:      ' EMPNO.  
    DISPLAY 'FIRSTNME:   ' FIRSTNME-TEXT.  
    DISPLAY 'EDLEVEL:    ' EDU-LEVEL.  
    DISPLAY 'BIRTHDATE:  ' BIRTHDATE.  
    GOBACK.
```

下面是运行结果：

```
EMPNO:      000050  
FIRSTNME:   JOHN  
EDLEVEL:    0016  
BIRTHDATE:  1925-09-15
```

第六章 使用 CURSOR 读取多行数据

- CURSOR 的功能
- 使用 CURSOR 的一个例子
- CURSOR 的语法

6-1 CURSOR 的功能

前面的例子中,我们能够一次使用 SQL 语句从表中读取一条记录,但是如果符合 WHERE 条件的记录数大于 1,就会返回错误。这时就需要使用 CURSOR 来一次一条地处理多行记录。可以认为 DB2 先建立了一个结果表,用来存放符合条件的记录, DB2 使用一个游标在结果集中依次把记录传递给程序,记录全部读取后,程序将达到 SQLCOD=100。每个 CURSOR 有一个自己的名字,程序中可以同时存在几个 CURSOR,使用不同的名字分别引用。

一个 CURSOR 使用以下操作来处理数据:

DECLARE CURSOR: 定义 CURSOR 所查找数据的 SQL 语句

OPEN CURSOR: 打开 CURSOR, 生成结果集)

FETCH CURSOR: 从结果集中读取下一条记录数据

CLOSE CURSOR: 释放结果集

因为满足WHERE的记录有时候不止一条,这时候就有了结果集,可以用CURSOR一次一条的读取结果集中的记录,当程序达到SQLCODE=100,即找不到符合的记录,表示读取完毕.

6-2 使用 CURSOR 的一个例子

EXEC SQL

```
DECLARE THISEMP CURSOR FOR  
  SELECT EMPNO, LASTNAME,  
         WORKDEPT, JOB  
  FROM DSN8610.EMP  
  WHERE WORKDEPT = 'D11'  
FOR UPDATE OF JOB
```

END-EXEC.

...

EXEC SQL

```
  OPEN THISEMP
```

END-EXEC.

...

循环

{

EXEC SQL

```
  FETCH THISEMP
```

```
  INTO :EMP-NUM, :NAME2,
```

```
  :DEPT, :JOB-NAME
```

END-EXEC.

如果 sql code=0 跳出循环

...

对于某些雇员

EXEC SQL

```
  UPDATE DSN8610.EMP
```

```
    SET JOB = :NEW-JOB
```

```
    WHERE CURRENT OF THISEMP
```

END-EXEC.

...

对于另一些雇员

EXEC SQL

```
  DELETE FROM DSN8610.EMP
```

```
    WHERE CURRENT OF THISEMP
```

END-EXEC.

...

}

...

EXEC SQL CLOSE THISEMP END-EXEC.

<-声明 CURSOR

DECLARE CURSOR名 CURSOR
FOR SELECT语句
为哪个SELECT语句产生的结果集
定义CURSOR
一个CURSOR对应一个结果集

注意：
是在工作单元节中定义
CURSOR 把CURSOR当作是变
量用

<-打开 CURSOR

<-读取下行记录 CURSOR

通过循环依次读取记录
FETCH CURSOR名 INTO 主机变量
读取一条记录并把记录传给主机变量

<-更新 CURSOR 当前位置的数据

CURSOR用于WHERE子句,
表示条件,当前CURSOR所指的
记录数据

<-删除 CURSOR 当前位置的记录

<-关闭 CURSOR

定义 CURSOR :

DECLARE CURSOR 为游标指定一个名字和 SELECT 语句，语法如下：

```
EXEC SQL
    DECLARE cursor-name CURSOR FOR
    SELECT column-name-list
    FROM table-name
    WHERE search-condition
    FOR UPDATE OF column-name
END-EXEC.
```

上面给出的只是十分简单的 SELECT 语句，实际你可以使用各种复杂的 SELECT 语句定义游标。如果你想读取的同时更新某些记录，就要使用 FOR UPDATE OF 子句，把需要更新的列名列出来，这样在打开游标时，DB2 会在符合条件的记录上加更新锁，防止其他程序同时对这些记录进行更新造成数据错乱。

打开 CURSOR :

CURSOR 在使用之前必须打开，DB2 在这时会根据 SELECT 语句生成结果集。OPEN 语法如下：

```
EXEC SQL
    OPEN cursor-name
END-EXEC.
```

cursor-name 必须是已经声明过的游标名字，程序通过检查 SQLCODE 来确定 CURSOR 打开成功。

使用 CURSOR 读取数据 :

使用 FETCH 命令把结果集里的数据读取到宿主变量里来。OPEN 命令只生成结果集，必须通过一条条 FETCH 语句来真正取得这些结果。FETCH 执行时，DB2 先把记录指针移向下一条记录，并把它作为当前记录，然后把当前记录的值传递到 INTO 指定的宿主变量中去。这个过程在 FETCH 的一次又一次执行中不断重复，直至达到所有记录结束，这时 SQLCODE 会返回+100。FETCH 语法如下：

```
EXEC SQL
    FETCH cursor-name
    INTO :host-variable1, :host-variable2
END-EXEC.
```

FETCH 的游标名必须已经打开，INTO 指定的宿主变量个数与类型应与 DECLARE CURSOR 时指定的一致，可以使用宿主结构变量。

更新 CURSOR 当前位置的数据：

使用 FETCH 读取数据后，程序可以使用 UPDATE...WHERE CURRENT OF 语句来更新当前位置的记录，语法如下：

```
EXEC SQL
    UPDATE table-name
    SET column1 = value, column2 = value
    WHERE CURRENT OF cursor-name
END-EXEC.
```

游标 UPDATE 数据要符合以下要求：

- 一次只能更新一行数据——当前行
- WHERE CURRENT OF 指定要更新那个 CURSOR 的当前行
- 这个游标在 DECLARE CURSOR 时指定 FOR UPDATE OF 的列都要在 SET 子句中给出值。
- UPDATE 命令并不移动当前游标位置
- UPDATE 的数据违反表上的任意 CHECK、UNIQUE INDEX 或外键约束时 将会出错。

删除 CURSOR 当前行：

使用 DELETE...WHERE CURRENT OF 语句可以删除当前行，语法如下：

```
EXEC SQL
    DELETE FROM table-name
    WHERE CURRENT OF cursor-name
END-EXEC.
```

使用游标的 DELETE 语句要符合以下要求：

- 一次只能删除一行——当前行。
- 由 WHERE 子句指定删除哪个游标的当前行。

关闭游标：

当处理结束后，要关闭游标释放资源，语法如下：

```
EXEC SQL
    CLOSE cursor-name
END-EXEC.
```

如果程序结束时没有关闭的游标，系统会自动关闭它。

定义 WITH HOLD 的游标：

当程序完成一个工作单元 (UOW)，也就是发出 COMMIT 或 ROLLBACK 命令时，DB2 会关闭所有打开着的游标，如果你不想这样做，就要在定义游标时使用 WITH HOLD 选项，这样当工作单元结束时，游标不受影响，写法如下例：

```
EXEC SQL
    DECLARE EMPLUPDT CURSOR WITH HOLD FOR
    SELECT EMPNO, LASTNAME, PHONENO, JOB, SALARY, WORKDEPT
    FROM DSN8710.EMP
    WHERE WORKDEPT < 'D11'
ORDER BY EMPNO
END-EXEC.
```

第七章 动态 SQL 语言程序设计

- 什么是动态 SQL
- 使用 EXECUTE IMMEDIATE 动态执行
- 使用 PREPARE 和 EXECUTE 动态执行
- 固定列动态 SELECT
- 不定列动态 SELECT

7-1 什么是动态 SQL

把 SQL 语句预先写好在程序里，是一种简单高效的编程手段，但是往往缺乏一定的灵活性，一些情况下，我们希望 SELECT 语句的条件并不预先写定，而是根据输入的数据决定使用什么样的 WHERE 语句；另一些情况下，我们可能并不能预先确定需要选取列的数据类型，希望能动态地决定；甚至我们希望查询哪个表都可以由用户输入；这时我们就需要用到动态 SQL。动态 SQL 不是把 SQL 语句预先写到程序里，而是在程序运行时根据需要写到宿主变量里，然后动态地执行。

一般情况下，静态 SQL 编程能够满足 90% 以上的需求，由于动态 SQL 要在运行时动态绑定和 PREPARE，执行效率比静态 SQL 差很多，因此当业务逻辑允许时，尽量通过灵活地使用宿主变量来把语句变为静态 SQL。

7-2 使用 EXECUTE IMMEDIATE 动态执行

EXECUTE IMMEDIATE 命令可以动态地执行一条 SQL 语句，其语法为：

```
EXEC SQL  
    EXECUTE IMMEDIATE :sql_statements;
```

sql_statements 是一个宿主变量，它里面存放需要运行的 SQL 语句，它的类型是 VARCHAR 型变量，由于 EXECUTE IMMEDIATE 没有数据返回，所以它只能用于执行不需要返回的操作。

例如如下语句执行一条删除语句：

```
EXEC SQL BEGIN DECLARE SECTION;  
struct { short len;  
        char x1[56];  
        } stmtbf;  
EXEC SQL END   DECLARE SECTION;  
  
sprintf (stmtbf.x1,  
        "DELETE FROM DSN8710.EMP WHERE EMPNO = '000190'");  
stmtbf.len = strlen(stmtbf.x1);  
EXEC SQL EXECUTE IMMEDIATE :stmtbf
```

7-3 使用 PREPARE 和 EXECUTE 动态执行

使用 EXECUTE IMMEDIATE 命令时，每次执行都要重新动态地 PREPARE SQL 语句，这将消耗很大的系统资源。对于一种常见的情况：具体的 SQL 语句需要在运行时才能确定，类似的 SQL 语句要运行多次，每次运行只有部分参数不同，这时如果每个 SQL 语句都用 EXECUTE IMMEDIATE 语句执行，将造成很大的浪费，对于这类应用，使用 PREPARE 和 EXECUTE 方式是更好的选择。PREPARE 一次准备 SQL 语句，后面每次运行，只要给相关的宿主变量赋值即可，不必再次 PREPARE。具体用法参照下面的例子：

```
struct { short len;
        char x1[56];
    } stmtbf;
struct { char deptno[4];
        struct { short len;
                char x[36];
            } deptname;
        char mgrno[7];
        char admrdept[4];
    } hv2;
short ind[4];

...
sprintf (stmtbf.x1,
        "INSERT INTO DEPT VALUES (?, ?, ?, ?)");
stmtbf1.len = strlen(stmtbf.x1);
printf("***      begin prepare          ***$n");
EXEC SQL PREPARE s1 FROM :stmtbf;
...
printf("***      begin execute          ***$n");
EXEC SQL EXECUTE s1 USING :hv2:ind;
...
printf("***      insert an other record ***$n");
EXEC SQL EXECUTE s1 USING :hv2:ind;
```

7-4 固定列动态 SELECT

固定列的动态 SELECT 是指在编写程序时,SQL 语句查询出来的数据列的个数和类型都已经知道,但记录需要满足的条件尚不确定,需要在运行时决定的情况。

固定列的动态 SELECT 做法:

1. 运行时根据数据情况决定 SQL 语句:

程序运行时,根据输入的数据决定 SELECT 语句,把语句组织成字符串,存放到一个宿主变量里,这个宿主变量必须以 VARCHAR 数据类型定义。SELECT 语句所选择出来的列必须是已知的,它的 WHERE 字句中可以使用宿主变量,在 OPEN 时指定值。

2. 准备语句:

为语句起一个名字,然后从上面的宿主变量中准备语句,如:

```
EXEC SQL PREPARE STMT FROM :DSTRING;
```

准备的语句名字叫“STMT”,“DSTRING”是存有 SELECT 语句的宿主变量。

3. 声明游标:

使用 DECLEAR CURSOR 命令从准备好的语句中声明游标,如:

```
EXEC SQL DECLARE C1 CURSOR FOR STMT;
```

声明了叫做“C1”的游标,使用的是前面准备好了的“STMT”

4. 打开游标:

使用 OPEN 命令打开游标,如:

```
EXEC SQL OPEN C1;
```

如果 SELECT 语句中用到了宿主变量作参数,在 OPEN 时要给定宿主变量的值,例如如下语句:

```
EXEC SQL OPEN C1 USING :PARM1, :PARM2, :PARM3, :PARM4;
```

PARM1 到 PARM4 会依次取代 SELECT 语句中使用的 4 个“?”,形成实际使用的 SQL 语句。

5. 从游标中取得数据:

使用 FETCH 语句从游标中取得数据,例如如下语句:

```
EXEC SQL FETCH C1 INTO :NAME, :PHONE;
```

INTO 指定的变量的个数、长度和类型必须与 PREPARE 时给的 SELECT 语句一致。可以在循环中多次 FETCH 得到多行符合条件的数据,当符合条件的数据结束时,返回 SQLCODE=100。

6. 关闭游标

程序结束前要关闭游标,格式如下:

```
EXEC SQL CLOSE C1;
```

如果程序退出时还有没关闭的游标,系统会自动为它关闭。

7-4 固定列动态 SELECT (续页)

一个例子：

```
struct { short len;
        char x1[56];
    }inpstr;

...
sprintf (inpstr.x1,
        "SELECT EDLEVEL, LASTNAME FROM EMP WHERE EMPNO = '000010'");
inpstr.len = strlen(inpstr.x1);
printf("$n***      begin prepare          ***");
EXEC SQL PREPARE STAT1 FROM :inpstr;
printf("$n***      begin declare          ***");
EXEC SQL DECLARE C2 CURSOR FOR STAT1;
printf("$n***      begin open              ***");
EXEC SQL OPEN C2;

...
printf("$n***      begin fetch              ***");
EXEC SQL FETCH C2 INTO :edlevel, :lname;
printf("$n***      returned values          ***");
printf("$nedlevel = %d", edlevel);
printf("$nlname = %s\n", lname.x1);

...
printf("$n***      begin close              ***");
EXEC SQL CLOSE C2;
```

7-5 不定列动态 SELECT

不定列动态 SELECT 是灵活程度最高的一种 SELECT 语句，在编写程序时，并不知道程序要去查那些表，从表中查出多少列数据，这些数据都是什么类型，所有这一切，都要在运行时得到。

这时需要使用一种称作 SQLDA 的数据结构来与数据库打交道，SQLDA 里包括一部分头信息，然后是若干个成为 SQLVAR 的数据结构，每个 SQLVAR 对应 SELECT 语句中的一个列，SQLVAR 里包括列的数据类型、数据长度、列名和一个指向实际存储 SELECT 语句读取到的数据区的指针，一个指向这个列的指示变量的指针。应用程序先建立一个这样的数据结构，动态地决定 SQL 语句，并把 SQL 语句组到一个字符串里去，然后运行“PREPARE statement 名 INTO :SQLDA FROM :SQL 语句字符串”或“DESCRIBE statement 名 INTO :SQLDA”命令，DB2 就会把这个 SQL 语句需要有多少个 SQLVAR 来接收数据，每个 SQLVAR 对应列的数据类型、长度和列名填记到 SQLDA 结构中去，这时程序要去为每个 SQLVAR 分配存储数据的空间，分配指示变量空间，然后把 SQLDA 中每个 SQLVAR 的相应指针指到对应的地址，这样做了以后，程序就可以用这个做好的 SQLDA 结构去 FETCH 数据了，使用“EXEC SQL FETCH CSR1 USING DESCRIPTOR :SQLDA”命令，就可以把一条记录的数据填到上面为各个列分配的存储空间里去，应用程序从这些存储空间按照列的类型去使用这些变量，就可以进行各种业务逻辑的处理了。

下面是一个有 3 个 SQLVAR 的 SQLDA 的具体结构：

长度	类型	名称	含义		
8 字节	字符串	SQLDAID	存放“SQLDA ”作标志符		
4 字节	INTEGER	SQLDABC	SQLDA 结构总长度		
2 字节	SMALLINT	SQLLN	SQLDA 里分配的 SQLVAR 个数		
2 字节	SMALLINT	SQLLD	SQL 语句需要的 SQLVAR 个数		
2 字节	SMALLINT	SQLTYPE	SQLVAR 数据类型 具体数值与类型见的定义， 参见 SQL REFERENCE		SQLVAR 1
2 字节	SMALLINT	SQLEN	SQLVAR 数据长度		
4 字节	指针	SQLDATA	指针，指向实际存储 SELECT 语句读取到的数据区		
4 字节	指针	SQLIND	指向一个 SMALLINT 型的变量， 这个变量里面存放列的指示变量		
2 字节	SMALLINT	SQLNAME LENGTH	SQLVAR 对应列的列名长度	SQLVAR 列名	
30 字节	字符串	SQLNAME DATA	SQLVAR 对应列的列名		
2 字节	SMALLINT	SQLTYPE	SQLVAR 数据类型		SQLVAR 2
2 字节	SMALLINT	SQLEN	SQLVAR 数据长度		
4 字节	指针	SQLDATA	指针，指向实际存储 SELECT 语句读取到的数据区		

4 字节	指针	SQLIND	指向一个 SMALLINT 型的变量， 这个变量里面存放列的指示变量		L V A R 2
2 字节	SMALLINT	SQLNAME LENGTH	SQLVAR 对应列的列名长度	SQLVAR 列名	
30 字节	字符串	SQLNAME DATA	SQLVAR 对应列的列名		
2 字节	SMALLINT	SQLTYPE	SQLVAR 数据类型		S Q L V A R 3
2 字节	SMALLINT	SQLLEN	SQLVAR 数据长度		
4 字节	指针	SQLDATA	指针，指向实际存储 SELECT 语句读取到的数据区		
4 字节	指针	SQLIND	指向一个 SMALLINT 型的变量， 这个变量里面存放列的指示变量		
2 字节	SMALLINT	SQLNAME LENGTH	SQLVAR 对应列的列名长度	SQLVAR 列名	
30 字节	字符串	SQLNAME DATA	SQLVAR 对应列的列名		

下面是 COBOL 语言中定义 SQLDA 的一个具体例子：

```
*****
*  SQL DESCRIPTOR AREA                                *
*****

01  SQLDA.
    02  SQLDAID      PIC X(8)  VALUE 'SQLDA  '.
    02  SQLDABC      PIC S9(8) COMPUTATIONAL VALUE 33016.
    02  SQLN         PIC S9(4) COMPUTATIONAL VALUE 750.
    02  SQLD         PIC S9(4) COMPUTATIONAL VALUE 0.
    02  SQLVAR       OCCURS 1 TO 750 TIMES
                        DEPENDING ON SQLN.
    03  SQLTYPE      PIC S9(4) COMPUTATIONAL.
    03  SQLLEN       PIC S9(4) COMPUTATIONAL.
    03  SQLDATA      POINTER.
    03  SQLIND       POINTER.
    03  SQLNAME.
        49  SQLNAMEL  PIC S9(4) COMPUTATIONAL.
        49  SQLNAMEC  PIC X(30).
```

7-5 不定列动态 SELECT (续页 2)

下面是一个 C 语言里定义 SQLDA 的例子

```
EXEC SQL BEGIN DECLARE SECTION;
    struct {
        char    sql daid[8]; /*标志串*/
        long    sql dabc;    /*SQLDA 长度*/
        short   sql n;       /*SQLDA 结构中定义的 SQLVAR 的个数*/
        short   sql d;       /*SQL 语句需要的 SQLVAR 的个数*/
        struct {
            short sql type; /*SQLVAR 数据类型*/
            short sql len;  /*SQLVAR 数据长度*/
            char *sql data; /*指向最终存放 SQLVAR 数据的指针*/
            short *sql ind; /*指向指示变量的指针*/
            struct {          /*SQLVAR 的列名, VARCHAR*/
                short length; /*列名长*/
                char data[30]; /*列名*/
            } sql name;
        } sql var[750];
    } SQLDA;
EXEC SQL END DECLARE SECTION;
```

上面可以看到, SQLVAR 定义的都是 750 个, 这是因为一条 SELECT 语句最大可以选择的列数是 750 列。

请参照下面的具体例子来学习不定列的动态 SQL 编程方法, 这个例子从 SYSIN 读入一条 SELECT 语句, 动态地执行它, 把此语句选择出来的列列名、各记录的数据值显示出来。为了程序简洁, 它只处理部分 SQL 的数据类型。

```
#include <stdlib.h>
#define SQL_TYP_DATE          384
#define SQL_TYP_DATE2        385
#define SQL_TYP_TIME          388
#define SQL_TYP_TIME2        389
#define SQL_TYP_CHAR          452
#define SQL_TYP_CHAR2        453
#define SQL_TYP_VARCHAR       448
#define SQL_TYP_VARCHAR2     449
#define SQL_TYP_LONG          456
#define SQL_TYP_LONG2        457
#define SQL_TYP_FLOAT         480
#define SQL_TYP_FLOAT2       481
#define SQL_TYP_DECIMAL       484
#define SQL_TYP_DECIMAL2     485
#define SQL_TYP_INTEGER       496
#define SQL_TYP_INTEGER2     497
#define SQL_TYP_SMALL         500
#define SQL_TYP_SMALL2       501
```


7-5 不定列动态 SELECT (续页 3)

```

EXEC SQL INCLUDE SQLCA;
EXEC SQL INCLUDE SQLDA;

/*void ShowSQLDA(char*); */
void CheckSQLCODE(char*p);
void main()
{
    int      nIdx;
    char      szTitle[256];
    int      nTitleFieldLen;
    char      szFormatString[256];
    int      nMallocSize;
    char      szTemp[256];
    char      szTemp2[256];
    short int nVCHARLen;
    EXEC SQL BEGIN DECLARE SECTION;
        struct {
            char  sqlDaid[8]; /*标志串*/
            long  sqlDabc;    /*SQLDA 长度*/
            short sqln;       /*SQLDA 结构中定义的 SQLVAR 的个数*/
            short sqld;       /*SQL 语句需要的 SQLVAR 的个数*/
            struct {
                short sqlType; /*SQLVAR 数据类型*/
                short sqlLen;  /*SQLVAR 数据长度*/
                char *sqlData; /*指向最终存放 SQLVAR 数据的指针*/
                short *sqlInd; /*指向指示变量的指针*/
                struct {
                    short length; /*列名长*/
                    char data[30]; /*列名*/
                } sqlName;
            } sqlVar[750];
        }SQLDA;
        struct{
            short len;
            char szSQL[256];
        }Stmt;
        /*存放 SQL 语句*/
        short int indval[750];
    EXEC SQL END DECLARE SECTION;

    /*初始化 SQLDA: */
    memcpy(SQLDA.sqlDaid, "SQLDA ", 8);
    SQLDA.sqlDabc=sizeof(SQLDA);
    SQLDA.sqln=750;

    /*读入 SQL 语句: */
    gets(Stmt.szSQL);
    Stmt.len=strlen(Stmt.szSQL);
    if(Stmt.len<1){
        printf("Please input SQL in SYSIN\n");
        exit(0);
    }
    printf("%s\n", Stmt.szSQL);

```

7-5 不定列动态 SELECT (续页 4)

```

EXEC SQL DECLARE VLSQL STATEMENT;
CheckSQLCODE("DECLARE STATEMENT");

EXEC SQL PREPARE VLSQL INTO :SQLDA FROM :Stmt;
CheckSQLCODE("PREPARE");

/*为 SQLVAR 分配数据空间，并指定指示变量：*/
szFormatString[0]='\0';
szTitle[0]='\0';
nTitleFieldLen=0;
for(nIdx=0;nIdx<SQLDA.sql d;nIdx++){
    SQLDA.sql var[nIdx].sql ind=&(indval[nIdx]);
    switch(SQLDA.sql var[nIdx].sql type){
        case SQL_TYP_DATE:
        case SQL_TYP_DATE2:
        case SQL_TYP_TIME:
        case SQL_TYP_TIME2:
        case SQL_TYP_CHAR:
        case SQL_TYP_CHAR2:
            /*留一个字节 EOS:*/
            nMallocSize=SQLDA.sql var[nIdx].sql len+1;
            nTitleFieldLen=SQLDA.sql var[nIdx].sql len;
            break;
        case SQL_TYP_VARCHAR:
        case SQL_TYP_VARCHAR2:
            /*一个 short 变量，字符串留一个字节 EOS*/
            nMallocSize=SQLDA.sql var[nIdx].sql len+1+
                sizeof(short);
            nTitleFieldLen=SQLDA.sql var[nIdx].sql len;
            break;
        case SQL_TYP_LONG:
        case SQL_TYP_LONG2:
        case SQL_TYP_FLOAT:
        case SQL_TYP_FLOAT2:
        case SQL_TYP_DECIMAL:
        case SQL_TYP_DECIMAL2:
        case SQL_TYP_INTEGER:
        case SQL_TYP_INTEGER2:
        case SQL_TYP_SMALL:
        case SQL_TYP_SMALL2:
            nMallocSize=SQLDA.sql var[nIdx].sql len;
            nTitleFieldLen=SQLDA.sql var[nIdx].sql len=18;
            break;
        default:
            printf("\nNot support SQLVAR type:%d.Program stop.\n",
                SQLDA.sql var[nIdx].sql type);
            exit(0);
            break;
    }
    SQLDA.sql var[nIdx].sql data=malloc(nMallocSize);
    if(SQLDA.sql var[nIdx].sql data==0){
        printf("malloc error, size=%d\n", nMallocSize);
        exit(0);
    }
}

```

7-5 不定列动态 SELECT (续页 5)

```

memset(SQLDA.sql var[nl dx].sql data, 0x00, nMal l ocSi ze);

strcat(szTi tle, " ");
memcpy(szTemp, SQLDA.sql var[nl dx].sql name. data,
        SQLDA.sql var[nl dx].sql name. l ength);
szTemp[SQLDA.sql var[nl dx].sql name. l ength]='\0';
memset(szTemp+SQLDA.sql var[nl dx].sql name. l ength, ' ',
        si zeof(szTemp)-SQLDA.sql var[nl dx].sql name. l ength);
strncat(szTi tle, szTemp, nTi tleFi el dLen);
}

EXEC SQL DECLARE CSR1 CURSOR FOR VLSQL;
CheckSQLCODE("DECLARE CURSOR");

EXEC SQL OPEN CSR1 ;
CheckSQLCODE("OPEN CSR1");

printf("%s\n", szTi tle);
printf("-----+-----+-----+-----+-----+");
printf("-----+-----+-----+-----+-----+");
while(1){
    EXEC SQL FETCH CSR1 USING DESCRIPTOR :SQLDA;
    i f(SQLCODE==100)break;
    CheckSQLCODE("FETCH");
    printf("\n");
    for(nl dx=0; nl dx<SQLDA.sql d; nl dx++){
        swi tch(SQLDA.sql var[nl dx].sql type){
            case SQL_TYP_DATE:
            case SQL_TYP_DATE2:
            case SQL_TYP_TIME:
            case SQL_TYP_TIME2:
            case SQL_TYP_CHAR:
            case SQL_TYP_CHAR2:
                spri ntf(szTemp, " %%-%ds", SQLDA.sql var[nl dx].sql l en);
                i f(*(SQLDA.sql var[nl dx].sql i nd)<0){
                    pri ntf(szTemp, "NULL");
                }
                el se{
                    pri ntf(szTemp, SQLDA.sql var[nl dx].sql data);
                }
                break;
            case SQL_TYP_VARCHAR:
            case SQL_TYP_VARCHAR2:
                spri ntf(szTemp, " %%-%ds", SQLDA.sql var[nl dx].sql l en);
                i f(*(SQLDA.sql var[nl dx].sql i nd)<0){
                    pri ntf(szTemp, "NULL");
                }
                el se{
                    memcpy(&nVCHARLen, SQLDA.sql var[nl dx].sql data,
                        si zeof(short));
                    SQLDA.sql var[nl dx].sql data[
                        (si zeof(short)+nVCHARLen)]='\0';
                    pri ntf(szTemp, SQLDA.sql var[nl dx].sql data+
                        si zeof(short));
                }
                break;
        }
    }
}
break;

```

7-5 不定列动态 SELECT (续页 6)

```

        case SQL_TYP_LONG:
        case SQL_TYP_LONG2:
        case SQL_TYP_FLOAT:
        case SQL_TYP_FLOAT2:
        case SQL_TYP_DECIMAL:
        case SQL_TYP_DECIMAL2:
        case SQL_TYP_INTEGER:
        case SQL_TYP_INTEGER2:
        case SQL_TYP_SMALL:
        case SQL_TYP_SMALL2:
            if(*(SQLDA.sql var[nl dx].sql ind)<0){
                printf(szTemp, "NULL");
            }
            else{
                printf("%18d", SQLDA.sql var[nl dx].sql data);
            }
            break;
        }
    }
}
}
}
void CheckSQLCODE(char* p)
{
    if(SQLCODE!=0)
    {
        printf("SQL error at \"%s\" , SQLCODE=%d, ERROR: %s\n",
            p, SQLCODE, sql ca.sql errmc);
        exit(0);
    }
}

```

当输入 “ SELECT * FROM DSN8710.DEPT ” 时输出如下：

```

SELECT * FROM DSN8710.DEPT

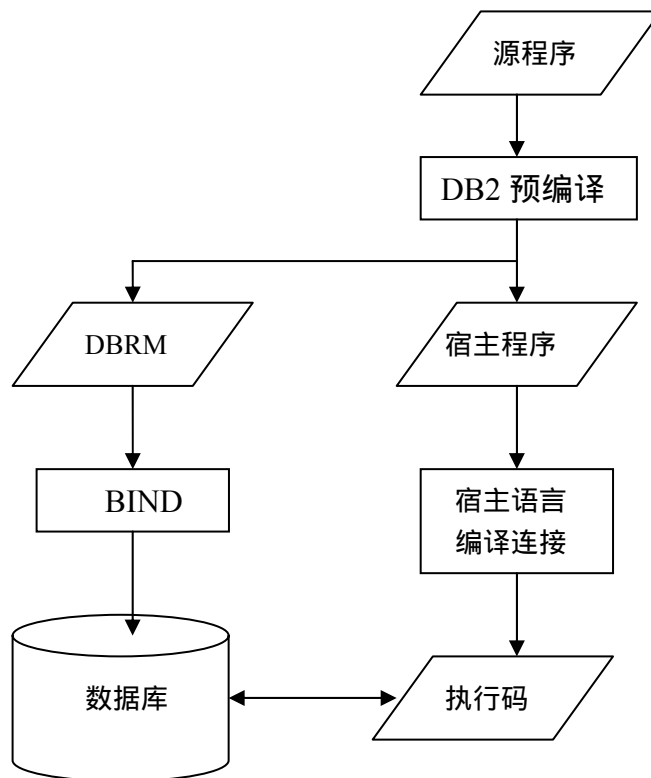
```

DEP	DEPTNAME	MGRNO	ADM	LOCATION
A00	SPIFFY COMPUTER SERVICE DIV.	000010	A00	
B01	PLANNING	000020	A00	
C01	INFORMATION CENTER	000030	A00	
D01	DEVELOPMENT CENTER	NULL	A00	
D11	MANUFACTURING SYSTEMS	000060	D01	
D21	ADMINISTRATION SYSTEMS	000070	D01	
E01	SUPPORT SERVICES	000050	A00	
E11	OPERATIONS	000090	E01	
E21	SOFTWARE SUPPORT	000100	E01	

第八章 SQL 程序的准备

- SQL 程序准备过程
- COBOL 语言程序准备
- C 语言程序准备
- DB2 批量程序的执行

8-1 SQL 程序准备过程



DB2 程序的准备过程分为预编译、编译、连接和 BIND 几个过程。程序通过 DB2 的预编译，把程序中 EXEC SQL 的语句全部翻译成宿主语言的语句和函数调用，同时生成一个叫做 DBRM 的文件。预编译程序程序中具体的 SQL 语句以特定的格式存储在 DBRM 文件中。然后调用宿主语言的编译连接过程，把预编译好的代码生成可执行代码。对于 DBRM 文件，要使用 DB2 的 BIND 命令，把 DBRM BIND 到 DB2 数据库中去，BIND 时 DB2 根据统计信息来确定索引的使用情况，数据的存取路径等。这样，最后的执行部分实际是分为两块，一块是程序的执行码，一块是 BIND 到 DB2 里的 DBRM，这两者必须是同一次编译产生的，否则程序执行时 SQLCODE 会出 -805 错误。

8-2 COBOL 语言程序准备

根据下面的例子 JCL 我们来了解 COBOL 语言程序的准备过程：

编译使用到 JCL 过程 DEVPO1.PROC(DB2BCOMP)如下：

```
*****
//DB2BCOMP PROC WSPC=500,                <- 使用到的中间文件一次分配空间大小
//                                     MEM=,          <- 被预编译的程序名
//                                     LOADLIB=' DEVPO1.LOAD', <- 程序执行码存放的库
//                                     PRFX=' DEVPO1'      <- DBRM/COPYBOOK 库的前缀
//*****
//*          PRECOMPILE THE IBM COBOL PROGRAM          *
//*****
//PC      EXEC PGM=DSNHPC, REGION=4096K,
//          PARM=' HOST(IBMCOB), SOURCE, APOST'
//STEPLIB DD DSN=DSN710.SDSNLOAD, DISP=SHR
//          DD DSN=DSN710.SDSNEXIT, DISP=SHR
//DBRMLIB DD DSN=&PRFX. . DBRM(&MEM), DISP=SHR          <- 生成 DBRM 的名字
//SYSCIN  DD DSN=&&DSNHOUT, DISP=(MOD, PASS), UNIT=SYSDA, <- 预编译完的代码
//          SPACE=(800, (&WSPC, &WSPC))                存入临时文件里
//SYSLIB  DD DSN=&PRFX. . COPY, DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSTEM  DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSUT1  DD SPACE=(800, (&WSPC, &WSPC), , , ROUND), UNIT=SYSDA
//SYSUT2  DD SPACE=(800, (&WSPC, &WSPC), , , ROUND), UNIT=SYSDA
//*****
//*          COMPILE THE IBM COBOL PROGRAM IF THE PRECOMPILE          *
//          RETURN CODE IS 4 OR LESS.                                *
//*****
//COB      EXEC PGM=IGYCRCTL, REGION=2048K, COND=(4, LT, PC),
//          PARM=' DATA(31), DYNAM, SIZE(MAX), APOST, RENT, LIB, XREF, OFFSET, DBCS,
//          SOURCE, SSRANGE, TEST'
//STEPLIB DD DSN=IGY.SIGYCOMP, DISP=SHR
//SYSLIB  DD DSN=&PRFX. . COPY, DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSTEM  DD SYSOUT=*
//SYSLIN  DD DSN=&&LOADSET, DISP=(MOD, PASS), UNIT=SYSDA, <- 结果放入另一个临时
//                                                         文件
//          SPACE=(800, (&WSPC, &WSPC))
//SYSIN   DD DSN=&&DSNHOUT, DISP=(OLD, DELETE)          <- 使用上步的临时文件作输入。
//SYSUT1  DD SPACE=(800, (&WSPC, &WSPC), , , ROUND), UNIT=SYSDA
//SYSUT2  DD SPACE=(800, (&WSPC, &WSPC), , , ROUND), UNIT=SYSDA
//SYSUT3  DD SPACE=(800, (&WSPC, &WSPC), , , ROUND), UNIT=SYSDA
//SYSUT4  DD SPACE=(800, (&WSPC, &WSPC), , , ROUND), UNIT=SYSDA
//SYSUT5  DD SPACE=(800, (&WSPC, &WSPC), , , ROUND), UNIT=SYSDA
```

8-2 COBOL 语言程序准备 (续页 1)

```
//SYSUT6 DD SPACE=(800, (&WSPC, &WSPC), , , ROUND), UNIT=SYSDA
//SYSUT7 DD SPACE=(800, (&WSPC, &WSPC), , , ROUND), UNIT=SYSDA
//*****

//* PRELINK STEP. *
//*****

//PLKED EXEC PGM=EDCPRLK, REGION=2048K, COND=((4, LT, PC), (4, LT, COB))
//STEPLIB DD DSN=CEE. SCEERUN, DISP=SHR
//SYSMSGSGS DD DSN=CEE. SCEEMSGP(EDCPMSGGE), DISP=SHR
//SYSIN DD DSN=&&LOADSET, DISP=(OLD, DELETE) <- 使用上步生成的临时文件
                                                作输入。

//SYSMOD DD DSN=&&PLKSET, UNIT=SYSDA, DISP=(MOD, PASS), <- 结果放入第三个
// SPACE=(32000, (30, 30)), 临时文件
// DCB=(RECFM=FB, LRECL=80, BLKSIZE=3200)
//SYSLIB DD DUMMY
//SYSDEFSD DD DUMMY
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTEM DD SYSOUT=*
//*****

//* LINKEDIT IF THE PRECOMPILE AND COMPILE *
//* RETURN CODES ARE 4 OR LESS. *
//*****

//LKED EXEC PGM=IEWL, PARM='MAP',
// COND=((4, LT, PC), (4, LT, COB), (4, LT, PLKED))
//SYSLIB DD DSN=CEE. SCEELKED, DISP=SHR
// DD DSN=DSN710. SDSNLOAD, DISP=SHR
// DD DSN=ISP. SISPLoad, DISP=SHR
// DD DSN=&LOADLIB, DISP=SHR
//SYSLIN DD DSN=&&PLKSET, DISP=(OLD, DELETE) <- 使用上步生成的
// DD DDNAME=SYSIN 临时文件作输入。
//SYSLMOD DD DSN=&LOADLIB(&MEM), DISP=SHR <- 最后执行代码放入这里
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD SPACE=(1024, (50, 50)), UNIT=SYSDA
//*
//***** DB2 BIND (PACKAGE AND PLAN)
//*

//BND EXEC PGM=IKJEFT01, DYNAMNBR=20, COND=(4, LT)
//STEPLIB DD DSN=DSN710. SDSNLOAD, DISP=SHR
// DD DSN=DSN710. SDSNEXIT, DISP=SHR
```


8-2 COBOL 语言程序准备 (续页 2)

```
//DBRMLIB DD DSN=&PREFIX..DBRM,DISP=SHR      <- 拿第一步生成的 DBRM 来 BIND
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSTSIN DD DUMMY
*****
```

下面是调用以上过程进行处理的 JCL :

```
*****
//COMP JOB NOTIFY=&SYSUID
//* DB2 COBOL BATCH PROCESS
//MYPROC JCLLIB ORDER='DEVPO1.PROC'           <- 指向 JCL 过程所在库
//CICSCBL EXEC DB2BCOMP, MEM=SAMP1            <- 给定程序名
//PC.SYSIN DD DSN=DEVPO1.SRC(SAMP1),DISP=SHR  <- 给定源码的库和成员名
//BND.SYSTSIN DD *                            <- 指定 BIND 参数, 具体选项
DSN SYSTEM(DSND)                             参见 SQL 参考手册。
BIND PACKAGE(DCBPKG) MEMBER(SAMP1) ACT(REP) ISO(CS) VAL(BIN)
*****
```

8-3 C 语言程序准备

根据下面的例子 JCL 我们来了解 COBOL 语言程序的准备过程：

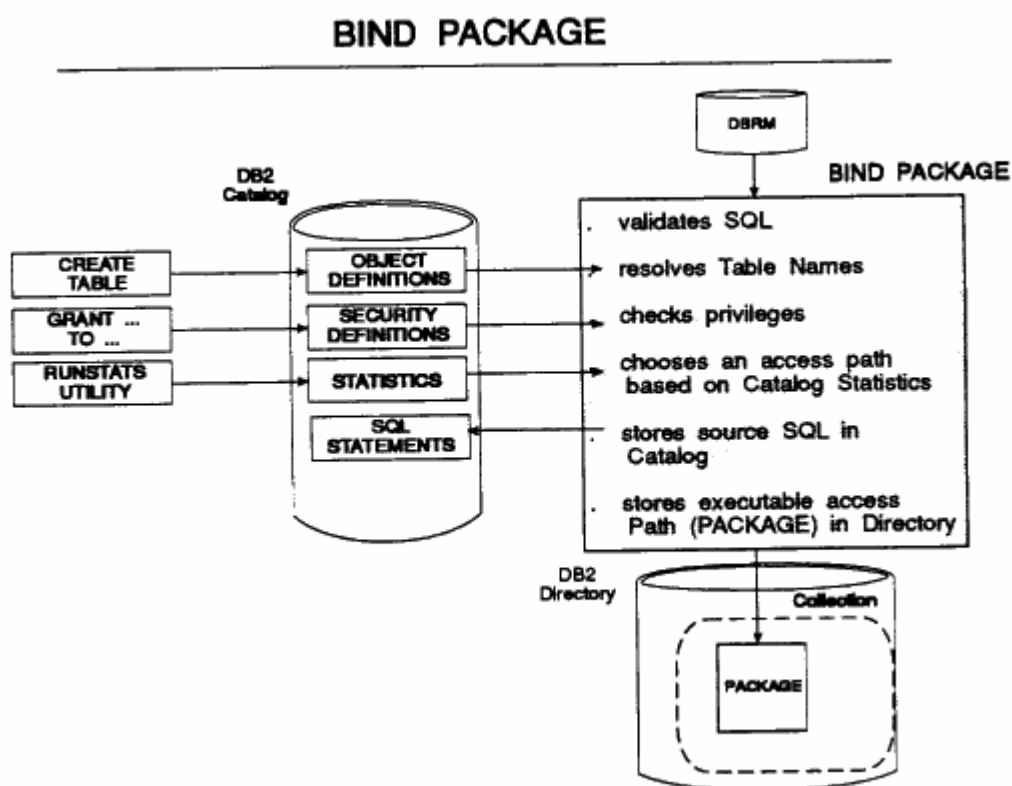
```
*****
//CDB2BAT JOB NOTIFY=$SYSUID,
//          CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),REGION=OM
//* C BATCH DB2 PREPARE COMPALE LINK AND BIND
//JOBLIB DD DSN=CEE.SCEERUN,DISP=SHR      <- C 语言系统执行程序库
//          DD DSN=SYS2.DB2.SDSNLOAD,DISP=SHR  <- DB2 系统执行程序库
//*      STEP 1 : PREPARE C PHONE PROGRAM
//CCB1 EXEC DSNHC,                          <- DB2 提供的 C 语言预编译、编译、连接过程。
//          MEM=DCBTEST2,                    <- 程序名
//          COND=(4,LT),
//          PARM.PC='HOST(C),SOURCE,XREF,MARGINS(1,80),STDSQL(NO)',
//          PARM.C='SOURCE LIST MARGINS(1,80) SEQ(81,90) LONGNAME',
//          PARM.LKED='AMODE=31,RMODE=ANY,MAP,LIST'
//PC.DBRMLIB DD DSN=DEVPO1.DBRM(DCBTEST2),DISP=SHR <- 生成的 DBRM 文件名字
//PC.SYSLIB DD DSN=DEVPO1.H,DISP=SHR              <- C 头文件所在库名
//PC.SYSIN DD DSN=DEVPO1.SRC(DCBTEST2),DISP=SHR   <- 源程序名字
//C.SYSLIB DD DSN=CEE.AEDCSRC7,DISP=SHR
//PLKED.SYSLIB DD DSN=DEVPO1.OBJ,DISP=SHR          <- 生成的目标文件所在库
//PLKED.SYSIN DD DSN=&LOADSET,DISP=(OLD,DELETE)
//LKED.SYSLMOD DD DSN=DEVPO1.LOAD(DCBTEST2),DISP=SHR <- 生成可执行代码的名字
//LKED.SYSIN DD *                                  <- 使用的系统库
        INCLUDE SYSLIB(DSNELI)
//*
//SYSTSPRT DD SYSOUT=*
//*      STEP 2 : BIND PROGRAMS
//BIND EXEC PGM=IKJEFT01,DYNAMNBR=20,COND=(4,LT)
//STEPLIB DD DSN=SYS2.DB2.SDSNLOAD,DISP=SHR      <- DB2 执行程序安装库
//DBRMLIB DD DSN=DEVPO1.DBRM,DISP=SHR            <- 第一步生成 DBRM 所在库
//SYSUDUMP DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
//SYSTSIN DD *                                   <- 指定 BIND 参数，具体选项
        DSN SYSTEM(DSND)                          参见 SQL 参考手册。
        BIND PACKAGE(DCBPKG) MEMBER(DCBTEST2) ACT(REP)ISO(CS) VAL(BIN)
/*
```

8-4 BIND 命令详解

- 关系数据库
- 表 (TABLE)
- 视图 (VIEW)

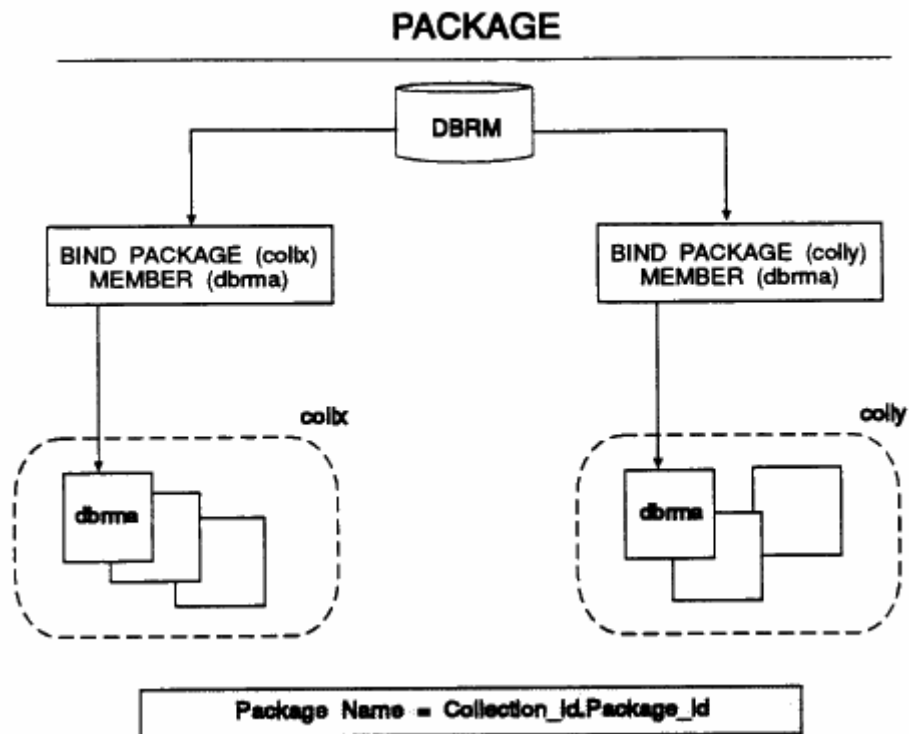
一个 DB2 程序，在预编译时就被分为两个部分，一个部分是翻译后的宿主语言原码，另一个部分就是 DBRM，在 DBRM 中保存经过解码以后的程序用到的所有 SQL 语句，这部分信息将在后面 BIND 进数据库中，保存在 DB2 里。为了保证这两部分能够一一对应，在预编译时，DB2 在宿主程序源码和 DBRM 中都写入一个相同的时间戳数据称做 CONSISTENCY TOKEN，当程序编译连接结束最后运行时，它以 DBRM 名、CONSISTENCY TOKEN 和语句号向 DB2 发出 SQL 执行请求。

DBRM 通过 BIND PACKAGE 放入 DB2 目录 (DB2 Directory) 中保存，程序运行时就是从 DB2 DIRECTORY 中查找到实际的 SQL 语句执行的。所谓 PACKAGE 就是带有访问路径信息的，经过绑定以后的 DBRM。



在 BIND PACKAGE 时 DB2 会进行如下操作：

- 检验：** 检查 SQL 涉及的表或视图是否存在
- 确定对象名称：** 在程序里对表和视图名没有给定属主时，根据默认或 BIND 参数指定，确定成“属主名.表名”的完整格式。
- 授权检查：** 检查作 BIND 的用户对 SQL 语句所涉及的数据库对象有没有操作权限。
- 确定存取路径：** 根据表的结构和所建立的主键、索引等的情况，确定每个 SQL 语句使用哪个索引，如何存取数据。DB2 会根据各个表的统计情况自动选用效率最高的存取路径。表的数据统计要通过 RUNSTATS 来更新。
- 数据存放：** BIND 把表的存取路径信息存放在 DB2 Directory 里，把 SQL 源语句存放在 DB2 Catalog 里，这样便于重新评估，修改存取路径。



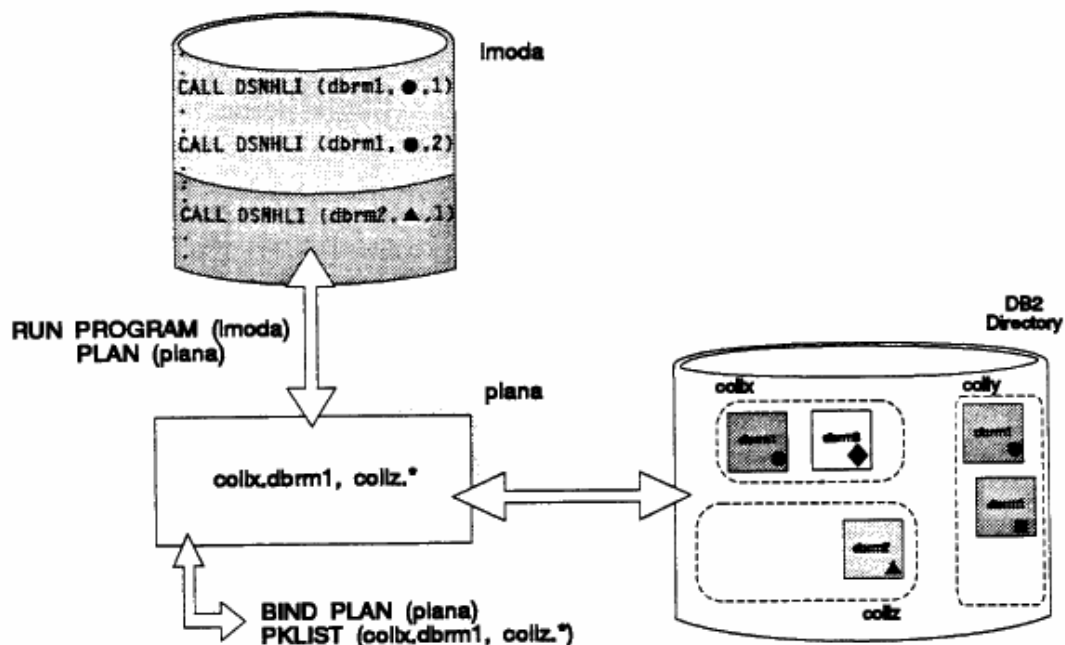
BIND PACKAGE 命令可以在 TSO 命令行也可以通过 JCL 提交，BIND PACKAGE 的简单格式是：
“ BIND PACKAGE (collection-id) MEMBER (package-id) ”

PACKAGE 总是与一个 COLLECTION 相联系的，collection-id 就是这个 COLLECTION 的名字。package-id 总与 DBRM 的名字一致。COLLECTION 只是为了把若干 PACKAGE 组织起来而起的一个组名，这个名字不需要预先定义，而是在第一个使用这个 COLLECTION 名字时自动创建。一个 PACKAGE 的全名就是“ collection-id.package-id ”，一个 PACKAGE 允许 BIND 到一个以上 COLLECTION，而一个 COLLECTION 一般也有多个 PACKAGE。

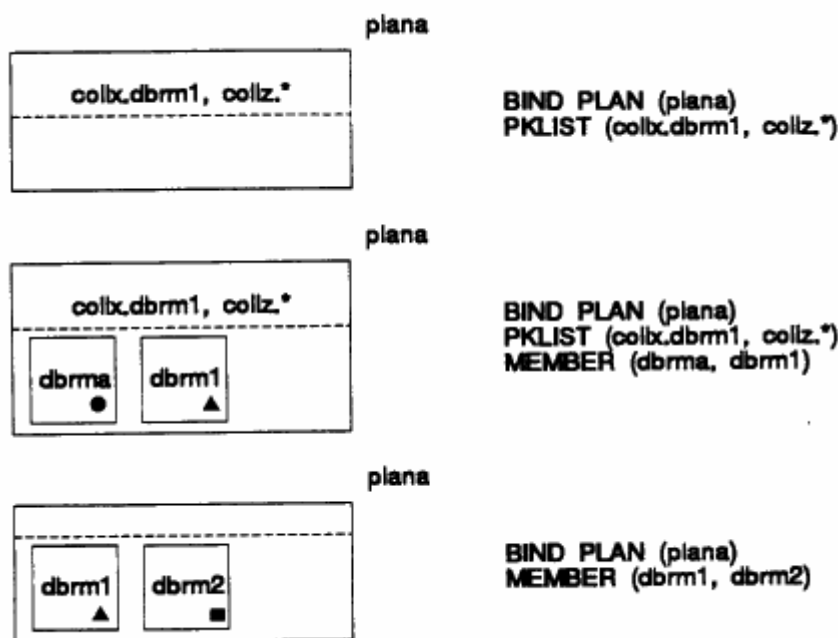
在程序与 PACKAGE 之间还有一个结构，叫做 PLAN，每个 PACKAGE 必须与一个 PLAN 相联系，PLAN 里包含一个 PACKAGE LIST，里面有一系列指向 PACKAGE 的指针，程序执行时根据 DBRM 名和 CONSISTENCY TOKEN 到 PLAN 里去查找符合条件的 PACKAGE 拿来运行。

PACKAGE LIST 是单个或多个 PACKAGE 的名单，他们包括在一个 PLAN 里面，PACKAGE LIST 是通过 DB2 命令 BIND PLAN 来加进 PLAN 里去的

8-4-1 DBRM PACKAGE 和 PLAN （续页 2）



在 DB2 V2.3 之前的版本，DBRM 是直接 BIND 进 PLAN 里的，后来为了便于对 DBRM 的管理，增加了 PACKAGE LIST 的概念，PACKAGE LIST 实际上只是个指针，指向实际的 DBRM，并没有直接把 DBRM 放进 PLAN 中去：



程序运行时要设置好使用的 PLAN 的名字，当发生 DB2 调用时，DB2 根据 PLAN 名，先在直接 BIND 在 PLAN 里的 MEMBER 里找有没有与程序 CONSISTENCY TOKEN 一致的 DBRM，然后再根据 PACKAGE LIST 的先后到里面去找有没有与程序 CONSISTENCY TOKEN 一致的 DBRM，若找到，就按照已准备好的访问路径和 SQL 语句去执行，如果没有找到，会给程序返回-805 错误。

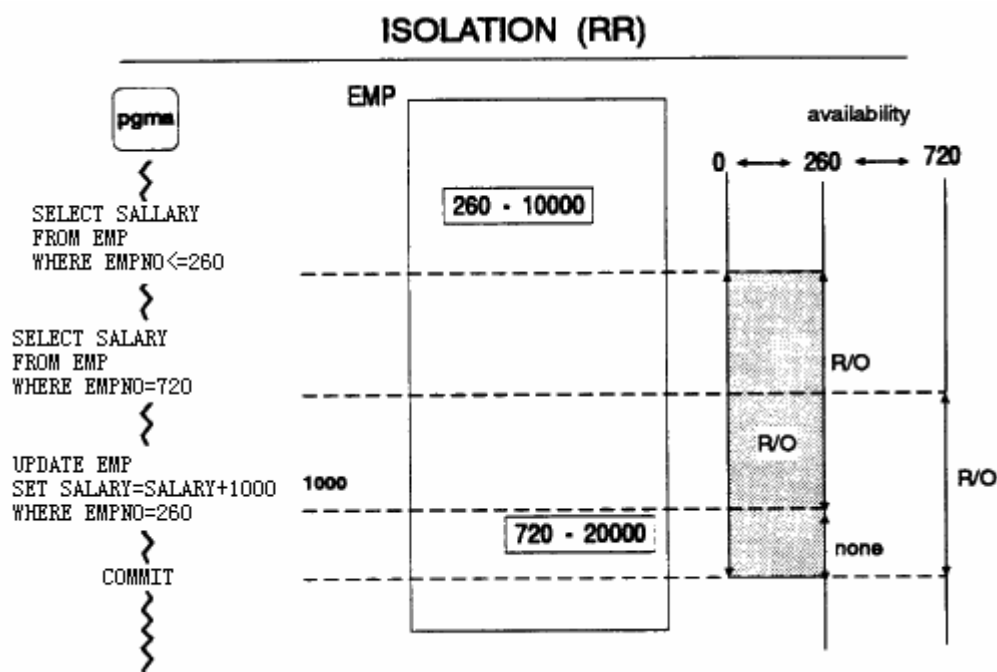
8-4-1 DBRM PACKAGE 和 PLAN （续页 3）

系统中的 PACKAGE，PACKAGELIST 和 PLAN 的信息记录在 DB2 CATALOG 表里面，可以通过查询以下表来了解系统中定义的情况：

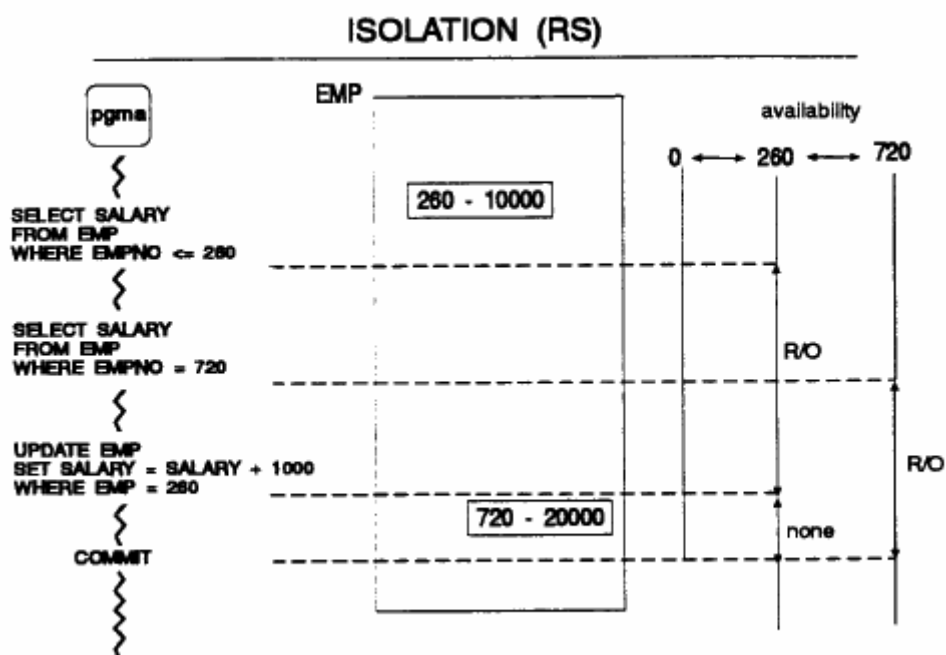
表名	说明
SYSIBM.SYSPACKSTMT	每个 PACKAGE 中的每条 SQL 语句有一条记录
SYSIBM.SYSPACKAGE	每个 PACKAGE 有一条记录
SYSIBM.SYSPACKLIST	每个 PLAN 相关的每个 PACKAGE LIST 有一条或一条以上记录
SYSIBM.SYSPLAN	每个 PLAN 有一条记录

8-4-2 隔离级别

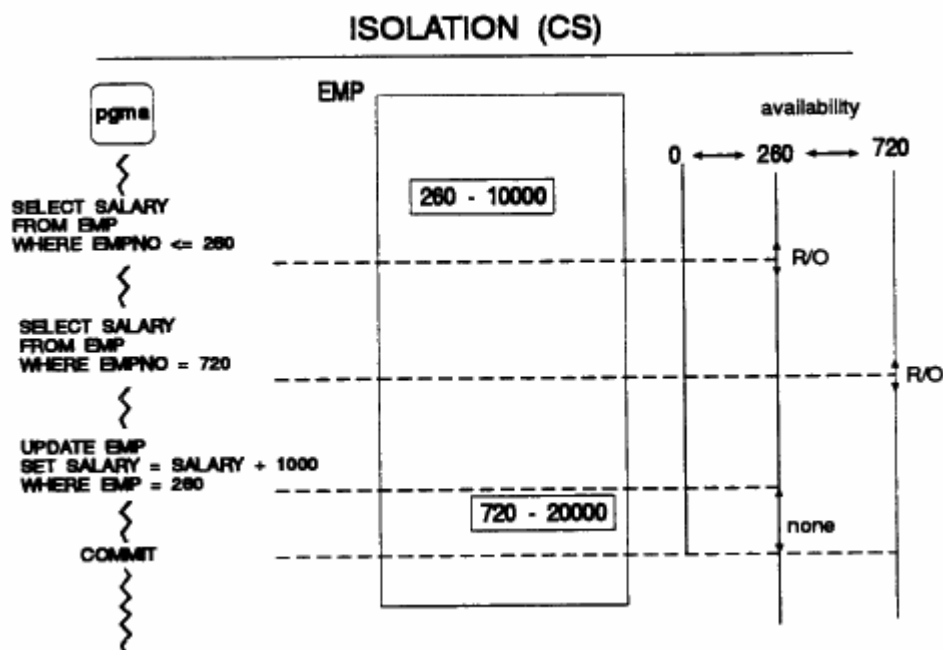
一个 DB2 数据库可能有许多程序在同时进行访问，为了应付两个或两个以上程序对同一条记录进行处理时避免出现混乱的情况，程序要对数据加锁。对数据加锁保证数据正确的同时会造成并发性的降低，DB2 设计了不同的隔离级别，以限定程序中锁的有效时间，使得不同业务逻辑的程序可以选择不同的隔离级别，以有效地提高程序的并发性。DB2 提供的隔离级别一共 4 种：



ISOLATION(RR)：可重复读。程序中的页锁和行锁一直保持，直到一个工作单元结束，这期间其他程序不能修改该表。在同一个工作单元内，如果按相同的条件作两次查询，返回的数据是完全一样的。在如上的例子中，第一条 SELECT 执行后，所有 ENPNO 260 的记录对其他程序就不能改变了而且不允许添加小于 260 的记录。



ISOLATION(RS)：读稳定。程序中的页锁和行锁一直保持，直到一个工作单元结束，这期间对已有数据不能更改，但是可以 INSERT 新的记录。在同一个工作单元里如果按相同的条件作两次查询，第一次得到的数据第二次能原样得到，但是有可能查询到比第一次更多的记录。在上面的例子中，虽然 260 的记录不能被别的程序改动，但其他程序可以 INSERT 小于 260 的记录。



ISOLATION(CS)：游标稳定。使用 CURSOR 处理数据时，凡已被 FETCH 出来的数据允许被其他程序更改，还没有被读过的数据不允许其他程序修改。在同一个工作单元内，如果按相同的条件作两次查询，返回的数据可能是不一样的。一般情况下 CS 级别即能保证数据的有效性，又能提供最大的可并发性。

ISOLATION(UR)：脏读。UR 级别程序读取数据时不加锁，这时它可能读到“脏”数据，比如另一个程序在一个 UOW 里对两个账户转帐，如果正在从 A 账户完成扣款，但尚未在 B 账户存款时，UR 的程序对账户进行查询，可能会发现有一笔钱不翼而飞的错误。

8-4-3 BIND 选项

BIND PACKAGE 命令用来把预编译生成的 DBRM BIND 进 PACKAGE 里去，BIND PLAN 命令用来把一定的 PACKAGE LIST 加入到 PLAN 中取。其常用选项如下：

ACQUIRE：	程序使用一个 PLAN 时何时开始为所用到的对象加锁，可选：ALLOCATE，PLAN 被打开时就加锁；USE，当第一次用到对象时加锁，是默认值。
ACTION：	指明绑定时如果发现已有重名的 PACKAGE 如何处理，可选 REPLACE 和 ADD，REPLACE 是默认值。
DEGREE：	是否使用 DB2 的并发处理特性来优化效率，可选：1，不允许；ANY，允许；1 是默认值。
EXPLAIN：	是否对 SQL 语句的效率进行分析，把结果写入 PLAN_TABLE，可选 YES/NO，NO 是默认值。
FLAG：B	IND 时显示的信息，I：所有信息；W：警告以上信息；E：错误信息；C：结束信息
ISOLATION：	见前面的介绍，RR 是默认值。
LIBRARY：	给定到那个库里去找 DBRM 文件。
MEMBER：	DBRM 名字
PACKAGE：	指明 BIND 到哪个 PACKAGE 上。
QUALIFIER：	使用数据库对象的默认前缀，如使用 QUALIFIER (DSN8701) BIND，则程序可以只写“SELECT 。。。 FROM EMP”，BIND 时会自动改成“SELECT 。。。 FROM DSN8701.EMP”
RELEASE：	指定程序获得的锁何时释放，COMMIT：在程序 COMMIT 或 ROLLBACK 时就释放；DEALLOCATE：程序运行结束时才释放；COMMIT 是默认值。
VALIDATE：	权限校验在何时进行，RUN：BIND 时如果 SQL 语句对所操作的对象没有权限，给出警告信息，正常生成 PACKAGE，待实际运行时再次检验；BIND：BIND 时如果权限通不过就出错，不生成 PACKAGE

8-5 DB2 批量程序的执行

DB2 程序可以是联机程序也可以是批量程序，联机程序在 CICS 系统控制下运行，批量程序要使用 DB2 的 RUN 命令来执行，具体命令参见如下例子：

```
//RUNCOB JOB (ACCOUNT),'NAME',NOTIFY=&SYSUID
//GO      EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
DSN SYSTEM(DSND) RETRY(0) TEST(0)
RUN PROGRAM(SAMP1) PLAN(BATP001) -
  LIB('DEVP01.LOAD') -

END
/*
```

首先使用 DSN 命令给定运行相关的数据库名：DSND，然后使用 RUN 命令指明程序名：SAMP1，执行计划名：BATP001，和程序所在库名：DEVP01.LOAD。

第九章 DB2I 的使用

- 什么是动态 SQL

9-1 DB2I 介绍

DB2I 是 9000 系统上 DB2 系统为用户提供的—个交互式的应用工具，在 DB2I 里，用户可以执行任意 SQL 语句、自动生成数据库表结构定义、准备用户程序、预编译用户程序、BIND 计划或包、执行用户程序、执行 DB2 命令、运行 DB2 工具程序。DB2I 是 DB2 开发人员经常会使用到的工具，这里我们就经常用到的一些功能作简要介绍。

下面是 DB2I 的初始画面：

DB2I PRIMARY OPTION MENU		SSID: DSND
COMMAND ===>		
Select one of the following DB2 functions and press ENTER.		
1	SPUFI	(Process SQL statements)
2	DCLGEN	(Generate SQL and source language declarations)
3	PROGRAM PREPARATION	(Prepare a DB2 application program to run)
4	PRECOMPILE	(Invoke DB2 precompiler)
5	BIND/REBIND/FREE	(BIND, REBIND, or FREE plans or packages)
6	RUN	(RUN an SQL program)
7	DB2 COMMANDS	(Issue DB2 commands)
8	UTILITIES	(Invoke DB2 utilities)
D	DB2I DEFAULTS	(Set global parameters)
X	EXIT	(Leave DB2I)
F1=HELP	F2=SPLIT	F3=END
F7=UP	F8=DOWN	F9=SWAP
F4=RETURN	F10=LEFT	F11=RIGHT
F5=RFIND	F6=RCHANGE	F12=RETRIEVE

各选项的功能是：

- | | | |
|---|---------------------|---|
| 1 | SPUFI | 用来编写 SQL 语句，并运行产生结果 |
| 2 | DCLGEN | 自动生成不同语言的表结构和宿主语言的定义 |
| 3 | PROGRAM PREPARATION | 通过多层画面输入准备程序所需要的各种参数，来完成程序的预编译、编译、预连接、连接和执行各个步骤 |
| 4 | PRECOMPILE | 通过画面输入必要的参数来对源程序进行预编译 |
| 5 | BIND/REBIND/FREE | 绑定、重绑定或释放计划或包 |
| 6 | RUN | 执行 SQL 程序 |
| 7 | DB2 COMMANDS | 执行 DB2 命令 |
| 8 | UTILITIES | 执行其他 DB2 工具程序 |
| D | DB2I DEFAULTS | 设置全局参数 |
| X | EXIT | 退出 |

下面我们对常用的一些功能进行介绍：

9-2 DB2I 参数设置

初始画面选择“D”进入参数设置功能，画面如下：

DB2I DEFAULTS PANEL 1	
COMMAND ==>	
Change defaults as desired:	
1 DB2 NAME	==> DSND (Subsystem identifier)
2 DB2 CONNECTION RETRIES	==> 0 (How many retries for DB2 connection)
3 APPLICATION LANGUAGE	==> COBOL (ASM, C, CPP, COBOL, COB2, IBMCOB, FORTRAN, PLI)
4 LINES/PAGE OF LISTING	==> 60 (A number from 5 to 999)
5 MESSAGE LEVEL	==> I (Information, Warning, Error, Severe)
6 SQL STRING DELIMITER	==> DEFAULT (DEFAULT, ' or ")
7 DECIMAL POINT	==> . (. or ,)
8 STOP IF RETURN CODE >=	==> 8 (Lowest terminating return code)
9 NUMBER OF ROWS	==> 20 (For ISPF Tables)
10 CHANGE HELP BOOK NAMES?	==> NO (YES to change HELP data set names)

选项功能：

- | | |
|------------------------------|---|
| 1 DB2 NAME | 给定 DB2 子系统实例的名字 |
| 2 DB2 CONNECTION RETRIES DB2 | 连接失败时的 RETRY 次数 |
| 3 APPLICATION LANGUAGE | 编成语言的种类，对生成表的宿主变量有影响，可选范围有(ASM, C, CPP, COBOL, COB2, IBMCOB, FORTRAN 和 PLI) |
| 4 LINES/PAGE OF LISTING SQL | 语句输出时多少行打印分页标志 |
| 5 MESSAGE LEVEL | 显示哪级以上的信息，可选有 I：Information, W：Warning, E：Error, S：Severe |
| 6 SQL STRING DELIMITER | SQL 中字符串使用什么符号标志开始和结束 |
| 7 DECIMAL POINT | 小数点使用什么符号 |
| 8 STOP IF RETURN CODE | 返回码大于等于多少时停止下面的执行 |

9-2 DB2I 参数设置 (续页)

第一页参数配置完成后，提交，进入下一页：

DB2I DEFAULTS PANEL 2

COMMAND ===>

Change defaults as desired:

1 DB2I JOB STATEMENT: (Optional if your site has a SUBMIT exit)
===> //DEVP124A JOB (ACCOUNT), 'NAME'
===> /*
===> /*
===> /*

COBOL DEFAULTS:	(For COBOL, COB2, or IBMCOB)
2 COBOL STRING DELIMITER ===> DEFAULT	(DEFAULT, ' or ")
3 DBCS SYMBOL FOR DCLGEN ===> G	(G/N - Character in PIC clause)

选项说明：

- 1 DB2I JOB STATEMENT DB2 自动创建 JCL 时 JOB 语句中的选项
- 2 COBOL STRING DELIMITER COBOL 程序中字符串的界定符
- 3 DBCS SYMBOL FOR DCLGEN DCLGEN 双字节数据的处理

选择完成，提交，参数设置结束。

9-3 SPUFI

SPUFI 是 DB2I 提供的动态执行 SQL 语句的工具,在 SPUFI 中你可以编写各种 SQL 语句,并提交,察看结果。

在 DB2I 初始页面打“1”进入 SPUFI 的第一个配置画面:

SPUFI		SSID: DSND
====>		
Enter the input data set name:		(Can be sequential or partitioned)
1	DATA SET NAME ...	====> 'DEVP124.SRC(SELECT)'
2	VOLUME SERIAL ...	====> (Enter if not cataloged)
3	DATA SET PASSWORD	====> (Enter if password protected)
Enter the output data set name:		(Must be a sequential data set)
4	DATA SET NAME ...	====> 'DEVP124.OUTPUT'
Specify processing options:		
5	CHANGE DEFAULTS	====> YES (Y/N - Display SPUFI defaults panel?)
6	EDIT INPUT	====> YES (Y/N - Enter SQL statements?)
7	EXECUTE	====> YES (Y/N - Execute SQL statements?)
8	AUTO COMMIT	====> YES (Y/N - Commit after successful run?)
9	BROWSE OUTPUT ...	====> YES (Y/N - Browse output data set?)
For remote SQL processing:		
10	CONNECT LOCATION	====>

选项说明:

Enter the input data set name:

- | | | |
|---|-------------------|---|
| 1 | DATA SET NAME | 输入用的数据集名,这个数据集用来存放 SQL 语句,可以是一个顺序数据集或一个分区数据集的成员,使用前必须已经在系统中定义好. |
| 2 | VOLUME SERIAL | 数据集所在卷名,如果这个数据集没有编目,必须指定卷名 |
| 3 | DATA SET PASSWORD | 如果数据集是被口令保护的,输入数据集口令 |

Enter the output data set name:

- | | | |
|---|---------------|-------------------------------|
| 4 | DATA SET NAME | 输出数据集名,必须是顺序数据集,使用前必须在系统里定义好. |
|---|---------------|-------------------------------|

Specify processing options:

- | | | |
|---|-----------------|------------------------------------|
| 5 | CHANGE DEFAULTS | 是否每次进入 SPUFI 参数修改画面,也就是本画面,进行参数修改. |
| 6 | EDIT INPUT | 是否可以编辑输入的 SQL 语句 |
| 7 | EXECUTE | 是否可以运行输入的 SQL 语句 |
| 8 | AUTO COMMIT | 是否在成功执行 SQL 后自动 COMMIT 它 |
| 9 | BROWSE OUTPUT | 是否允许浏览输出结果数据 |

9-3 SPUFI (续页 1)

输入选项提交，进入下一个配置画面：

CURRENT SPUFI DEFAULTS		SSID: DSND
===>		
Enter the following to control your SPUFI session:		
1 SQL TERMINATOR ..	===> ;	(SQL Statement Terminator)
2 ISOLATION LEVEL	===> CS	(RR=Repeatable Read, CS=Cursor Stability)
3 MAX SELECT LINES	===> 100000	(Maximum number of lines to be returned from a SELECT)
Output data set characteristics:		
4 RECORD LENGTH ...	===> 4092	(LRECL=Logical record length)
5 BLOCK SIZE	===> 4096	(Size of one block)
6 RECORD FORMAT ...	===> VB	(RECFM=F, FB, FBA, V, VB, or VBA)
7 DEVICE TYPE	===> SYSDA	(Must be DASD unit name)
Output format characteristics:		
8 MAX NUMERIC FIELD	===> 33	(Maximum width for numeric fields)
9 MAX CHAR FIELD ..	===> 80	(Maximum width for character fields)
10 COLUMN HEADING ..	===> NAMES	(NAMES, LABELS, ANY or BOTH)

选项说明：

Enter the following to control your SPUFI session:

- 1 SQL TERMINATOR SQL 语句结束符,通常使用“;”，一个 SQL 语句可以写多行，遇到结束符才认为一个 SQL 语句结束。
- 2 ISOLATION LEVEL 隔离级别 RR=Repeatable Read, CS=Cursor Stability
- 3 MAX SELECT LINES SELECT 语句允许返回的最大行数

Output data set characteristics:

- 4 RECORD LENGTH 输出数据集记录长
- 5 BLOCK SIZE 输出数据集记录块长
- 6 RECORD FORMAT 记录格式(RECFM=F, FB, FBA, V, VB, or VBA)
- 7 DEVICE TYPE 设备类型

Output format characteristics:

- 8 MAX NUMERIC FIELD 数字型数据显示的最大宽度
- 9 MAX CHAR FIELD 字符型数据显示的最大宽度
- 10 COLUMN HEADING 列头显示信息

第二个配置页提交后，就进入 SQL 编辑状态，对输入文件进行编辑。这时可以输入任何合法的 SQL 语句，每个语句可以有多行，以一个“；”号结束。文件中可以编辑多个 SQL 语句，这些语句会依次执行，凡是前两列是“--”的行，被视作注释，不被运行。如下就是一个这样的编辑画面：

```
File Edit Edit_Settings Menu Utilities Compilers Test Help  
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS  
EDIT          DEVP124.SRC(SELECT) - 01.99                      Columns 00001 00072  
***** Top of Data *****  
==MSG> -Warning- The UNDO command is not available until you change  
==MSG>           your edit profile using the command RECOVERY ON.  
000001 --SELECT * FROM DSN8710.EMP;  
000002 SELECT NAME,COLNO,COLTYPE,LENGTH,NULLS,DEFAULT,DEFAULTVALUE  
000003         FROM SYSIBM.SYSCOLUMNS WHERE TBNAME='DEPT' ORDER BY COLNO;  
000004 SELECT DEPTNO, MGRNO FROM DSN8710.DEPT  
000005        WHERE DEPTNO BETWEEN 'COO' AND 'D31';  
***** Bottom of Data *****  
  
Command ==> ;;;                                         Scroll ==> CSR  
F1=Help      F2=Split    F3=Exit     F5=Rfind    F6=Rchange   F7=Up  
F8=Down      F9=Swap     F10=Left   F11=Right   F12=Cancel  
. . . . .
```

编辑结束后，在“Command ==>”处输入三个分号，按 PF3 提交执行，会出现如下的结果画面：

提交后进入对结果数据集的浏览画面如下：

```
Menu Utilities Compilers Help  
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS  
BROWSE      DEVP124.OUTPUT                      Line 00000000 Col 001 080  
***** Top of Data *****  
-----+-----+-----+-----+-----+-----+-----+  
--SELECT * FROM DSN8710.EMP;                                00000181  
    SELECT NAME,COLNO, COLTYPE, LENGTH, NULLS, DEFAULT, DEFAULTVALUE          00000299  
        FROM SYSIBM.SYSCOLUMNS WHERE TBNAME='DEPT' ORDER BY COLNO;         00000399  
-----+-----+-----+-----+-----+-----+-----+  
NAME           COLNO   COLTYPE     LENGTH  NULLS  DEFAULT  DEFAULTVALUE  
-----+-----+-----+-----+-----+-----+-----+  
DEPTNO             1   CHAR            3    N       N  
DEPTNAME           2  VARCHAR          36    N       N  
MGRNO              3   CHAR            6    Y       Y  
ADMRDEPT           4   CHAR            3    N       N  
LOCATION            5   CHAR           16    Y       Y  
  
DSNE610I NUMBER OF ROWS DISPLAYED IS 5  
DSNE616I STATEMENT EXECUTION WAS SUCCESSFUL, SQLCODE IS 100  
  
-----+-----+-----+-----+-----+-----+-----+  
    SELECT DEPTNO, MGRNO FROM DSN8710.DEPT                        00000499  
Command ==>                                          Scroll ==> CSR  
F1=Help    F2=Split  F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap  
F10=Left   F11=Right  F12=Cancel
```

可以使用 PF7/PF8 PF10/PF11 等功能键以及 FIND 等编辑命令对结果数据进行浏览。浏览结束,可在“Command ==>”输入“;;;”,按 PF3 键从新进入前面的 SQL 命令编辑画面。或者直接按 PF3 键退出 SPUI

9-4 DCLGEN

DCLGEN 是 DB2 为嵌入式 SQL 编程方便而开发的工具,它可以根据表在 DB2 中的定义自动地以指定的宿主语言生成表的声明和宿主变量的定义。

在 DB2I 的初始界面里选择 2 就进入 DCLGEN 画面如下:

DCLGEN		SSID: DSND
===>		
Enter table name for which declarations are required:		
1 SOURCE TABLE NAME	===> DEPT	(Unqualified)
2 TABLE OWNER ===> DSN8710	(Optional)
3 AT LOCATION ===>	(Optional)
Enter destination data set: (Can be sequential or partitioned)		
4 DATA SET NAME	... ===> 'DEVP01.COPY(DEPT)'	
5 DATA SET PASSWORD	===>	(If password protected)
Enter options as desired:		
6 ACTION ===> REPLACE	(ADD new or REPLACE old declaration)
7 COLUMN LABEL ===> NO	(Enter YES for column label)
8 STRUCTURE NAME	.. ===>	(Optional)
9 FIELD NAME PREFIX	===> WK-	(Optional)
10 DELIMIT DBCS ===> NO	(Enter YES to delimit DBCS identifiers)
11 COLUMN SUFFIX	... ===> YES	(Enter YES to append column name)
12 INDICATOR VARS	.. ===> YES	(Enter YES for indicator variables)

选项说明:

- | | |
|---------------------|-----------------------|
| 1 SOURCE TABLE NAME | 要生成定义的表名 |
| 2 TABLE OWNER | 表的属主用户名 |
| 3 AT LOCATION | 分布式数据库的远程地点名 |
| 4 DATA SET NAME | 生成的数据表描述和宿主变量描述存放的数据集 |
| 5 DATA SET PASSWORD | 如果数据集被口令保护,要给出口令 |
| 6 ACTION | 新描述是添加还是覆盖到数据集中去 |
| 7 COLUMN LABEL | 是否预留注释行 |
| 8 STRUCTURE NAME | 宿主变量结构名,可选 |
| 9 FIELD NAME PREFIX | 宿主变量名前缀,可选 |
| 10 DELIMIT DBCS | 是否区分双字节标识 |
| 11 COLUMN SUFFIX | 是否增加列名前缀 |
| 12 INDICATOR VARS | 是否用指示变量 |

9-4 DCLGEN (续页)

如上画面提交后，DEVP01.COPY(DEPT)就变成如下内容：

```
*****
* DCLGEN TABLE(DSN8710.DEPT) *
* LIBRARY(DEVP124.COPY(AAA)) *
* LANGUAGE(COBOL) *
* NAMES(WK-) *
* QUOTE *
* DBCSDELIM(NO) *
* COLSUFFIX(YES) *
* INDVAR(YES) *
* ... IS THE DCLGEN COMMAND THAT MADE THE FOLLOWING STATEMENTS *
*****
EXEC SQL DECLARE DSN8710.DEPT TABLE
( DEPTNO CHAR(3) NOT NULL,
  DEPTNAME VARCHAR(36) NOT NULL,
  MGRNO CHAR(6),
  ADMRDEPT CHAR(3) NOT NULL,
  LOCATION CHAR(16)
) END-EXEC.
*****
* COBOL DECLARATION FOR TABLE DSN8710.DEPT *
*****
01 DCLDEPT.
* DEPTNO
10 WK-DEPTNO PIC X(3).
10 WK-DEPTNAME.
* DEPTNAME LENGTH
49 WK-DEPTNAME-LEN PIC S9(4) USAGE COMP.
* DEPTNAME
49 WK-DEPTNAME-TEXT PIC X(36).
* MGRNO
10 WK-MGRNO PIC X(6).
* ADMRDEPT
10 WK-ADMRDEPT PIC X(3).
* LOCATION
10 WK-LOCATION PIC X(16).
*****
* INDICATOR VARIABLE STRUCTURE *
*****
01 IDEPT.
10 INDSTRUC PIC S9(4) USAGE COMP OCCURS 5 TIMES.
*****
* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 5 *
*****
```

应用程序就可以直接 EXEC SQL INCLUDE 这样的文件了。具体 DCLGEN 会生成哪种宿主语言的定义，要依 DB2I 全局参数定义的语言种类而定。