INFORMIX—ESQL/C
程序员使用手册

（2.10 版）

李　达　文　强　董美婷
曾昱满　周晓萱　汪木兰　编写

■关于本手册的简要说明

INFORMIX-ESQL／C（C程序中内嵌的SQL及工具）是为想要建立带有数据库管理的日常应用程序的C语言程序设计员和想要在PERFORM屏幕格式中及ACE报表中使用C函数的INFORMIX-SQL高级用户而设计的。INFORMIX-ESQL／C包括预处理程序、C函数和前导文件，这些软件允许你执行以下过程：

- 在C程序和例行子程序中嵌入SQL语句（INFORMIX-ESQL／C）
- 执行DECIMAL和DATE类型数据的转换和操作
- 使用C语言实用函数
- 在ACE报表中使用C函数
- 在PERFORM屏幕格式中使用C函数

本手册假定读者会使用C语言程序并熟悉关系数据库结构，第六章“ACE和PERFORM中的C函数：“是假设其读者熟悉关系数据库系统（RDS）公司产品系列中另一次产品INFORMIX-SQL中的ACE和PERFORM编程。你可以从《INFORMIX-SQL用户指南》中读到ACE和PERFORM。

通过INFORMIX-ESQL/C或INFORMIX-SQL建立的数据库表的基本文件和索引结构是按C-ISAM文件来构造的。有关按索引顺序存取的方法的更多介绍，参见《C-ISAM程序员手册》。

本手册分成下列目录介绍：序言，6个章节、12个附录，错误信息节和索引。它们概述如下。

序　言　包含用于文法句子中约定记号的说明。还有关于示范数据库的简要描述。

第一章　给出关系数据库和IBM公司开发的ANSI标准结构化查询语言的RDS扩充RDSQL的总体概述。

第二章　叙述如何编写含嵌入SQL语句的C语言程序。本章解释C程序结构和各个RDSQL语句之间的内在关系。

第三章　按字母顺序描述可在C语言程序中使用的每一个RDSQL语句。可用本章作为关于RDSQL语句使用的语法和规则的参考。

第四章　讨论了RDSQL可识别的数据类型，它们之间的相互关系以及C语言的数据类型。本章亦包含允许对数据类型为DATE和DECIMAL的变量操作的库函数的详细描述。

第五章　描述了用于串操作的附加的实用C函数。

第六章　通过描述在表达式和控制块中插入用户自己书写的C函数的方法来扩展用户编制特定的ACE报表和PERFORM屏幕表格的能力。

附录A　列出INFORMIX-ESQL／C，ACE和PERFORM的前导文件。

附录B　描述组成数据库数据字典的系统目录。

附录C　描述INFORMIX-ESQL／C使用的环境变量。

附录D　列出不能用作RDS QL标识符（表、字段和索引名字）的保留字。

附录E　描述本手册中使用示范数据库Stores。

录附F　描述在INFORMIX-ESQL／C中提供的bcheck实用命令，这些命令检查并恢复索引文件的完整性。

附录G　描述sqlconv实用程序，这些实用程序将由INFORMIX建立的数据库转换成RDS QL可用的数据库。

附录H　描述了dbupdate实用程序，这些实用程序将INFORMIX-ESQL／C1.1版的数据库转换成2.0版或2.0以上版本的数据库。

附录I　描述了允许用户将ASCII文件装入到RDS QL数据库的dbload实用程序。

附录J　扩充第一章中关于外连接的描述。

附录K　描述了dbschema实用程序，你可以用其来重新建立数据库的模式数据、和索引定义。

■ESQL／C2.1版与以前版本的区别

2.1版与2.0版之区别。

INFORMLX—ESQL／C　2.1　版包含如下增强：

- 你可以将要插入到表中的记录或从表中选取出的记录送入缓冲区存放。下述RDSQL语句支持缓冲处理：DECLARE，OPEN，FETCH，INSERT，PUT，FLUSH和CLOSE。
- 你可以给从表中取出记录指定起始位置及对其排序
- 对连接字段不加索引的表的SELECT（带连接）操作更有效
- 在UPDATE语句中提供更大的灵活性。此外，你可以使用更简明的语法。
- 你可以使用ALTER INDEX语句来组织表记录的物理顺序使其同索引顺序匹配。
- 你可以控制在SET LOCK MODE语句中处理上锁记录的策略。
- 在2.1版中包含了附加的实用程序。dbschema允许你保存和重建数据库定义和结构。

增强的这几点不影响用RDSQL旧版本书写的语句的使用。

2.0版与1.1版之区别。下面几点的增强都已实现。

事务和数据库恢复

视图

- 支持NULL值
- SELECT语句之间的联合
- 使用GRANT语句的附加存取控制
- 支持同义名
- 存取记录的ROWID
- SELECT中的外部连接

■语法约定

下面的注释说明如何解释本手册中出现的语句语法表：

ABC　在语法中以大写字母形式出现的每一项为关键字。无论什么情况，都按语法表所示准确地键入它。例如：

CREATE INDEX indname

意指你必须不增不减空格或字母键入CREATE INDEX或create index。

abc　以小写斜体字母形或出现的每一次均用值来替换。在上面这个例子中，你必须用具体值来替代indname。在每一语句的叙述中，标以“解释”的小节描述了替换斜体字的值的值域。

( )　如左所示打入括号。它们是语句语法的一部分，而非特殊符号。

[ ]　不要将方括号作为INFORMIX—ESQL／C语句的一部分而打入；它们括住语句的任选部分。例如：

CREATE [UNIQUE]

表示你可以打入CREATE INDEX或CREATE UNIQUE INDEX。

|　竖杠表示在几个任选项中选择。例如：

[ONE | TWO [THREE] | FOUR] INDEX

意指你可以打入ONE或TWO或FOUR，并且如果你打入TWO，你还可以打入THREE。

{ }　若你必须在几个任选项中择一，那么任选项用花括号括起并用竖杠分开。例如：

{ONE | TWO | THREE}

意指你必须打入ONE或TWO或THREE而且你打入的不能多于一项。

<u>ABC</u>　若几个任选项中的一个是缺省任选，那么它出现时需加下划线。例如：

[CHOCOLATE | VANILLA | STRAWBERRY]

意指你可以从三个任选项中任选一项，但是如果你未打入其中之一，VANILLA为缺省项。

…　打入其后带省略号（如果你选用的话）的附加条项。省略号表示紧邻其前的条项可被无限重复。例如：

statement
…

意指一个语句序列可跟在列出的语句之后。

■准备使用INFORMIX—ESQL／C

这一节阐明运行INFORMIX—ESQL／C（C程序中内嵌的SQL及工具）之前必须遵循的步骤，而且指导你如何建立示范数据库。本节假定INFORMIX—ESQL／C已按照随该数据库软件包一起提供的信函中的安装指令安装到用户计算机中。

在你遵循下述指令过程中，你将注意到几种不同的字体。你所键入的字符以印刷字体表示；而你必须为其设定值的字则以斜体字体表示。

在UNIX系统中设置环境变量

在你使用INFORMIX—ESQL／C之前，你必须设置以下环境变量：

INFORMIXDIR　INFORMIX—ESQL／C驻留的目录

PATH　　你要执行程序的查找路径。

你可以在系统提示下设置这些环境变量，也可以在你的.profile（Bourne Shell中）或.login（C Shell中）文件中设置这些环境变量。如果你是在系统提示下设置这些变量，那么在下一次注册入系统时你必须重新设置它们。如果你是在.profile或.login文件中设置了这些变量，那么在你每一次注册入系统时UNIX系统自动设置它们。

1.注册。

2.设置环境变量INFORMIXDIR。以下的指令假定INFORMIX—ESQL／C驻留在／usr／informix目录下。如果由于某种缘故INFORMIX—ESQL／C驻留在另外的目录下，则用新的目录名替换／usr／informix。

如果你使用的是Bourne Shell，打入

```
INFORMIXDIR = /usr/informix
export  INFORMIXDIR
```

如果你使用的是C　Shell，则打入

```
setenv  INFORMIXDIR  /usr/informix
```

3.含INFORMIX—ESQL／C的目录包括在PATH环境变量中。

如果你使用的是Bourne Shell，打入

```
PATH= $PATH：$INFORMIXDIR/bin  export  PATH
```

如果你使用的是C　Shell，打入

```
Setenv  PATH $（PATH）：$（INFORMIXDIR）/bin
```

除了上列的环境变量外，你可以遵循附录C中的指令设置其它环境变量。

在DOS系统中设置环境变量

在你能够使用INFORMIX—ESQL／C之前，你必须遵循以下这些步骤：

- 如果INFORMIX—ESQL／C驻留的目录非／informix，则设置环境变量INFORMIXDIR。
- 使用PATH命令使得PC—DOS去为执行程序查找其所在的目录／informix／bin。
- 在用户的CONFIG.SYS文件中设置变量FILES值为20（或大于20），设置变量BUFFERS值为8（或大于8）。

你可以在操作系统提示下设置INFORMIXDIR环境变量及运行PATH命令也可以把它们放在用户的AUTOEXEC.BAT文件中。如果你是在系统提示下执行设置，那么在下一次引导系统时你必须重新执行设置。如果你把这些命令放在AUTOEXEC.BAT文件中，则PC—DOS在你每一次启动系统后自动设置它们。

1.启动系统（如果系统还未启动的话）

2.如果INFORMIX—ESQL／C驻留的目录非／informix，则通过打入

```
set  INFORMIXDIR = dirname
```

其中dirname为新目录名字，来设置环境变量INFORMIXDIR。

3.使用PATH命令使得PC—DOS去为可执行程序查找其所在的目录／informix

/bin;

PATH/informix/bin[; dirname…]

如果需要的话，你可以在全路径名前指定设备名。方括号表示该命令的任选部分。如果你想在查找路径中放置其它目录名，不必打入方括号。同样，必须保证你在两个不同的路径名之间打入的是分号。

4.使用系统编辑程序给CONFIG.SYS事件中的FILES和BUFFERS设置以下值：

FILES=20

BUFFERS=8

在你保存了这个修改后，重新引导机器。如果你不在AUTOEXEC.BAT文件中放置变量INFORMIXDIR和命令PATH，那么在你重新启动系统时就必须重新设置它们。

此外，你可以通过遵循附录C中的指令设置其它环境变量。

■在DOS系统下启动INFORMIX—ESQL/C

为了在DOS系统下使用INFORMIX—ESQL/C，务必使得计算机就绪且正在运行，亦即操作系统的提示符出现在屏幕上。

为了编译或执行使用了INFORMIX—ESQL/C的程序，你必须执行下两个步骤：

1.装入数据库存取控制程序。数据库存取控制程序是INFORMIX—ESQL/C的一部分。这一部分执行完成数据库管理操作所必需的数据存取和文件操作。

2.编译并运行INFORMIX—ESQL/C程序。

装入数据库存取控制程序

为装入数据库存取控制程序，打入

```
startsql
```

在一次人机对话中你只应执行一次该命令。在你从命令行中执行命令

```
exit
```

以前数据库存取控制程序一直保留在内存中。在你执行这条命令后，它把数据库存取控制程序从内存中删去。因为数据库存取控制程序要占内存空间，所以在你完成全部所要求的INFORMIX—ESQL/C操作后必须删去它。这样便可释放出供运行系统中其它程序的内存空间。

如果你试图装入数据库存取控制程序，而它又已经装入到内存，则INFORMIX—ESQL/C发出STARTSQL已经运行的信息。

进入INFORMIX—ESQL/C

在你装入了数据库存取控制程序之后，打入

```
esql
```

如果你试图运行esql而又未装入数据库存取控制程序，那么INFORMIX—ESQL/C发出请运行STARTSQL的信息。

如果你想查看数据库存取控制程序是否已经装入，打入

```
set
```

这将给你提供一张环境变量表。如果数据库存取控制程序已经装入，则表中列出变量SQLCADDR。下面是SQLCADDR出现在环境变量表中的例子。其中的数字表示数据库存取控制程序装入到内存的地址。

SQLCADDR = 333744a8

■示范数据库

INFORMIX—ESQL／C软件包带有示范数据库。本手册中援引的大多数例子都是基于示范数据库stores。这个在附录E中详细描述并列出的数据库是关于体育用具批发商、他们的顾客及顾客已发出的订货单。你可以在当前目录下通过键入esqldemo来建立数据库stores。这个shell命令删除所有标有stores的数据库并装入示范数据库及本手册中使用的程序例子。

数据库stores包含五个表，

customer　含作为批发商顾客的零售商店的信息。

order　含已发出订单的信息。

items　含每一份订单中每条项目的信息。

stock　含有一张批发商供应的库存商品项目表。

manufact　含关于库存项目生产产家的数据

建立示范数据库

下面的指令告诉你如何将示范数据库复制到你选择的目录下。

1.通过打入

mkdir dirname

其中dirname为新目录的名字，给示范数据库建立新目录。

2.通过打入

cd dirname

改当前目录为新目录。

3.通过打入

esqldemo

建立示范数据库。

这个命令在当前目录下建立示范数据库及其示范程序、屏幕表格和报表。如果这个命令不能有效执行，那么弄清楚你是否已经正确设置了所有环境变量。

在作了某种修改后，如果你想恢复原始的示范数据库也可以运行这条命令。

现在你的计算机已经为使用INFORMIX—ESQL／C作好了一切准备。

■在UNIX系统下对数据库存取的限制和开放

为了使得INFORMIX—ESQL／C能够存取数据库，当前用户必须具有数据库路径上每一个目录的读和执行权限。（RDSQL中的GRANT语句和REVOKE语句仅提供给数据库中的表，而非数据库本身。）

1.通过键入下述命令：

chmod 700 dirname

其中dirname是包含数据库的目录你可以限制别人对数据库的存取。

2.通过键入下述命令：

chmod 755 dirname

并假定dirname之上的一切目录至少具有相同的权限，你可以使得数据库为所有用户存取。

目　录

第一章　使用RDSQL

1.1　本章概述

关系数据库系统公司已开发出RDSQL作为IBM公司开发的ANSI标准结构查询语言（SQL）的一种扩充。RDS公司对SQL语言所作的扩充允许用户修改数据库，修改表名和字段名，及增加ANSI标准的SQL语句的功能。

在RDS公司数据库产品系列中，RDSQL扮演几种角色。在INFORMIX—SQL中，RDSQL既是交互式的查询语言，又是INFORMIX—SQL报表书写程序ACE中用户用以选择数据的语言。

你可以从《INFORMIX—SQL用户手册》中读到这些用法。在INFORMIX—E-SQL／C中，RDSQL是你嵌入到C语言应用程序中的数据库查询语言。

本章描述RDSQL并给出其语句的概述。在第三章，你可以读到INFORMIX—E-SQL／C中用到的RDSQL语句的所有语法和规则。这些语句按字母顺序组织。语法的表达使用序言中定义的约定。第三章中描述的语句不同于《INFORMIX—SQL参考手册》第二章中描述的语句。这个差别反应了INFOMIX—ESQL／C和INFORMIX—SQL——亦即编程使用和交互式使用在应用上的改变。

1.2　关系数据库

RDSQL语句建立关系数据库并对其操作。关系数据库由一个或多个表组成，这些表依次由记录和字段构成。每个记录包含字段值的一个特殊集。数据库是作为当前目录的子目录而建立的。子目录名是数据库名加扩展名.dbs。数据库子目录包含定义在数据库字典中的九个系统目录表。它也包含组成数据库的表。每一个表由扩展名分别为.dat和.idx的数据文件和索引文件表示。系统目录在附录B中描述。

1.3　RDSQL　标识符

RDSQL标识符是一个实体名，可以包含字母，数字和下划线（_）。首字符必须是字母。如果没有与此相反的规定，标识符可以有1到18个字符。

数据库　其名可以有1到10（UNIX系统）或1到8（DOS系统）个字符。

表　　其名在数据库中必须唯一。

字段　　其名在一个表中必须唯一。在一个数据库中也可以有重复的字段名。当在不同表中的字段名不唯一时，使用记号table.column指明所指的字段。

因为你必须于出现在RDSQL语句中的C语言变量（即宿主变量，参见第2章）之前冠以美元符号（$）或冒号（：），所以在C语言变量与RDSQL标识符之间不会有二义性。例如，如果你将lname定义为宿主变量并且你还要引用同名的数据库字段，则使用 $lname作为宿主变量名：

```
SELECT lname INTO  $lname FROM customer
```

1.4 数据库数据类型

你必须给数据库中的一切字段指定数据类型（参见第3章中的 CREATE TABLE 语句）。RDSQL的有效数据类型如下：

CHAR（n）　是长度为n的字符串（其中1＜n＜32，767）。

SMALLINT　是范围从－32，767到＋32，767的整数。

INTEGER　是范围从－2，147，483，647到＋2，147，483，647的整数。

DECIMAL〔（m〔，n〕）〕

是有效数字（精度）为m（＜＝32），小数点右边数字（标度)为n（＜＝m）的十进制浮点数。如果你给定了m和n值，则十进制变量可进行定点运算。所有绝对值小于0.5×10^{-n}的数具有零值。无错存贮的该类型变量的最大绝对值为$10^{m-n}-10^{-n}$。

在该类型的语法表示中，第二个参量是任选的，如果略去的话，该变量作浮点十进制数处理。这意味着DECIMAL（m）变量具有精度m且其绝对值范围为10^{-128}到10^{126}。如果不指定参数，DECIMAL被当作DECIMAL（16）浮点数处理。

SMALLFLOAT　是与C浮点数据类型相一致的二进制浮点数。SMALLFLOAT数据类型的值域同于宿主机上C浮点数据类型的值域。在小机器上，RDSQL将SMALLFLOAT扩充成DECIMAL（8）。

FLOAT　是与C双精度数据类型相一致的二进制浮点数。FLOAT数据类型的值域同于宿主机C双精度数据类型的值域。在小机器上，RDSQL将FLOAT扩充为DECIMAL（16）。

MONEY〔（m〔，n〕）〕

象DECIMAL数据类型一样，可取两个参数。对类型为MONEY（m，n）字段值的限制同于对类型DECIMAL（m，n）字段值的限制。类型MONEY（m）定义为DECIMAL（m，2），而且若不给定参数，则取MONEY为DECIMAL（16，2）。无论参数值为何，数据类型MONEY总是当作十进制定点数处理。

SERIAL（n）　是由RDSQL自动指定的唯一的序列整数。你可以指定初值n。缺省的起始值为1。

DATE　是以第4章描述的某一种格式的字符串打入的日期，且其存放自1899年12月31日以来的天数。

1.5 RDSQL语句概要

用于INFORMIX—ESQL／C的语句有五种不同的类型

- 数据的定义和管理
- 数据的操作
- 指针管理
- 数据的完整性
- 动态管理

1.5.1 RDSQL数据的定义和管理

数据的定义和管理语句包括对数据库及其表、视图、同义名和索引的建立、取消和存取控制的语句，还包括对表和字段修改和重新命名语句。在所列出的语句中，只有DATABASE语句在对已有的数据库数据操作前是必须的。

CREATE DATABASE　创建一个数据库目录，设立系统目录，并且使新的数据库成为当前数据库。在任何时候都不应有多于一个的当前数据库。

DATABASE　选择数据库并使它成为当前数据库。在任何时候都不应有多于一个的当前数据库。DATABASE语句具有EXCLUSIUE任选，该任选以互斥方式打开数据库并且只允许数据库管理员(DBA)存取打开的数据库。该任选在检查数据库并将其归档时特别有用。

CLOSE DATABASE　关闭当前数据库文件且使所有数据库均非当前数据库。在无当前数据库时只允许的RDSQL语句如下表所示：

- CREATE DATABASE
- DATABASE
- DROP DATABASE
- START DATABASE

DROP DATABASE　删除所有表、索引、系统目录，和数据库的子目录。如果该数据库目录下不存在别的文件，那么该子目录也被删除。

CREATE TABLE　建立表并定义其中的字段及数据类型。

ALTER TABLE　增加或删除表中字段或是修改已有字段的数据类型。

RENAEM TABLE　修改表名。

DROP TABLE　删除表的所有数据及索引并且删去其在系统目录中的登记项。

CREATE VIEW　定义一个其字段和记录均从已有的表和视图中择取出的表。若基本表变化了，则建立在其上的视图也作相应的改变。关于如何使用视图的说明参见本章后一小节“视图”。

DROP VIEW　从系统目录中删除视图的定义，依照被删除视图而定义的所有视图也一同被删除。而该视图的基本表不受影响。

CREATE SYNONTM　给表或视图定义别名。同义名仅对创建它的用户有效，也只能被其创建者取消。这意味着如果几个人都想使用同一个对象，则它们必须各自创建该对象的同义名。就INFORMIX—ESQ-L／C程序而言，创建者就是运行创建同义名程序的用户。

DROP SYNONYMO　从系统目录中删除同义名。

RENAME COLUMN　修改字段名。

CREATE INDEX　在表的一个或多个字段上建立索引

DROP INDEX　删除原先定义的索引。

ALTER INDEX　允许你修改表以使得表中记录的排列顺序同于索引顺序。此索引称作聚类索引。

有关索引的进一步介绍参见本章后面的一小节“索引策略”。

仅当DBA或表主人特地授予存取权限时，用户才能对数据库及表中的指定字段进行存取。RDSQL提供以下影响数据存取的语句：

GRANT	对特定用户或普通用户授予数据库存取权限。
REVOKE	取消特定用户或普通用户的数据库存取权限。 更详细的内容参见本章后面的“用户的地位和权限”一节。
UPDATE STATISTICS	通过确定并加入所指表的记录数来更新系统目录。RDSQL使用此信息优化检索，但在每一次插入和删除后RDSQL不会自动地更新系统目录。

1.5.2 数据操作

最常用的语句是如下数据操作语句：

DELETE	从表中删除一个或多个记录。
INSERT	把一个或多个记录增加到表中。
SELECT	从一个或多个表中取出数据。
UPDATE	修改一个或多个表记录中的数据。

SELECT 语句是RDSQL中最重要的也是最复杂的语句。尽管其语法是在第3章中中详细定义，我们还是以下面例子说明它的使用。

```
select lname, company
  into $p_lname, $p_company
  from customer
  where customer_num = 101
```

该语句检索表customer并返回顾客编号为101的单个记录。从这一记录中，它选取与联系人姓及公司名称相对应的字段的值并把其放进宿主变量p_lname和p_company中。

```
select quantity, total_price
  into $quantity, $total_price
  from items
  where order_num = 1001
```

这一语句指示了另一个返回单记录SELECT语句。在本例中，宿主变量quantity和total—qrice具有与表items中对应字段相同的标识符。这里无二义性存在，因为前缀 $把字段名与程序变量名区分开了。

返回单记录的SELECT语句叫作单SELECT语句而且可以独立操作。关于如何处理返回多个记录的SELECT语句参见下一节内容“指针管理”。

1.5.3 指针管理

在前一小节例子中，SELECT语句准确返回单个记录，且返回给程序变量的值是无二义性的。若可以返回多个记录，则必须有一种将记录彼此区分开的设备，这种设备称为指针。

由SELECT语句返回的记录集叫做该语句的工作集。在INFORMIX—ESQL／C程序中，你一次只能对工作集中的一个记录进行操作。这个记录称为当前记录且由指针

来指示。

指针可以处于两个状态（打开或关闭）中的一个。若指针处于打开状态，则它指向当前记录，或两个记录之间，或首记录之前，或末记录之后。若指针处于关闭状态，尽管它还保持与SELECT语句的联系，但已不再与工作集相关。

使用以下RDSQL语句去定义和操作指针：

DECLARE　命名指针并将其与特定的SELECT语句相联系。

OPEN　对SELECT语句设置指针为打开状态。OPEN语句使得SELECT语句连同当前程序变量一块执行并使指针指向运行结果工作集的首记录前一位置。如果指针处于打开状态，那么程序变量以后的变化不影响工作集。

FETCH　将指针移至下一个记录并从该记录取值。如果指针已移到最后一个记录之外，则出错变量sqlca.sqlcode取值为SQLNOTFOUND（=100）。（参见第2章的“出错处理和sqlca结构”）。SQLNOTFOUND表示工作集的末尾。

CLOSE　置指针为关闭状态且释放工作集。除OPEN语句外，所有不引用该指针的语句现在可以执行。

你可以用DELETE语句的专门格式来删除指针指示的当前记录。在DELETE语句执行之后，指针被留在前后两个记录之间而且在用FETCH语句重新定位之前不能再被使用。UPDATE语句允许你修改当前记录。在UPDATE语句执行之后，指针位置不改变。

为了具体说明指针管理语句，考虑如下例子：

```
$ declare X cursor for
    select order_num, order_date
      from orders
      where paid_date is null
        and ship_date> $p_date
  for update of paid_date
```

这个语句命名了指针X并将其同关键字FOR之后的SELECT语句相联系。该SELECT语句返回交货日期迟于宿主变量p_date的未付款订货单的订货单编号及订货日期。该语句也使得以后的UPDATE语句能够修改paid_date字段。DECLARE语句不能执行上述检索，它仅能将指针指定给SELECT语句：

```
open x
```

若在后来的程序中你使用了OPEN语句，则它使用程序变量p_date的当时值运行上述SELECT语句，确定满足WHERE子句的记录。指针x现在指向返回的第一个记录之前的位置：

```
fetch x into  $order_num,  $order_date
```

FETCH语句将指针X移到第一个记录并且用第一个记录中的字段order_num和order-date的值填写程序变量order_num和order_date：

```
UPDATE orders
  set paid-date = TODAY
      where current of x
```

UPDATE语句将当前（首）记录的订单交款日期改为当天。而指针仍保持指向首记录：

```
close x
```

现在指针x被置成关闭状态，并且其工作集被完全解除。这时一条立即执行的FETCH语句将会出错，因为关闭状态的指针不指向任何东西。如果你后来执行了语句

```
open  x
```

指针X在OPEN语句执行之后将返回到打开状态，并带有一个取决于程序变量p-date值的新工作集。

1.5.4 移动指针

如果你想取记录并控制提取顺序，你可以说明移动指针。

你可以从两个方向之一提取工作集记录：列表中的后一记录（缺省FETCH），或是前一（上一）记录。通过发出一系列FETCH NEXT和FETCH PREVIOUS语句，你可以在工作集上前后移动指针。

此外，你可以FETCH CURRENT（取出工作集的当前记录），FETCH FIRST（取出工作集的首记录），FETCH LAST（取出工作集的末记录），FETCH RELATIVE n（取出相对于工作集当前指针位置的第几个记录）以及FETCH ABSOLUTE m（取出工作集的第m个记录）。

下面的例子说明了移动指针及提取工作集的末记录。

```
$   declare q_curs scroll cursor for
      select order_date from orders
      where customer_num = 118

...

$   open q_curs


$   fetch last q_curs into $order_date
```

除了缺省的FETCH NEXT语句之外的所有FETCH语句都要求你首先说明移动指针。缺省的FETCH语句既能对移动指针操作，也能对规则指针操作。

1.5.5 插入指针

通过在DECLARE语句中的说明，你可以把指针同INSERT语句相联系。插入指针通过把内存中的数据放入缓冲区并在在缓冲区端时将其写盘的手段允许数据更高效率的插入列数据库。这不同于其它IBM实现方法，在那些方法中，程序员有权选择是否阻止数据输入（把数据插入缓冲区）。

下面两个语句允许你对缓冲区操作。

PUT　　把一条记录插入缓冲区。

FLUSH　强制写缓冲区。

一旦缓冲区填满或指针处于关闭状态，RDSQL自动地将记录写入表中。如果你想更快地写出记录，可以使用FLUSH语句。

如果你不想把插入数据写进缓冲区，请使用普通的INSERT语句。

CLOSE语句清洗缓冲区并关闭打开的指针。在清洗缓冲区把其内容写回磁盘时，如果RDSQL碰到错误，它停止清洗并且不关闭指针。跟在最后一个成功插入记录的缓冲记录被丢弃。检查sqlca.sqlcode的值以得到问题来源的信息。检查sqlca.sqlerrd〔2〕以获得成功插入到数据序的记录数。（更多的内容参见第2章的“出错处理和sqlca结构”一节）。

如上所述，CLOSE语句不仅释放指针，它还清除所有保留在插入缓冲区中的记录。FLUSH语句是使得RDSQL把插入缓冲区中的记录插入到数据库中而不关闭指针的一种RDS扩充。指针对多个PUT语句保持打开状态。你可以使用FLUSH语句强制插入，但不使用FLUSH语句你就无法延迟插入。RDSQL在缓冲区满时自动清洗它。紧跟在一次清洗后，sqlca.sqlerrd〔2〕被设置成表示物理地插入到数据库的记录的数目的正数值。

对在其值域中仅包含常数的插入指针的处理不尽相同。记录不是一次一条地送入缓冲区，也不是一次一条地写到磁盘上。代之以RDSQL保留有指示插入记录数目的计数器。在插入指针关闭之前，在插入操作期间被插入的记录不放入数据库。一旦指针关闭，则记录数据写盘且sqlca.sqlerrd〔2〕被置成指示物理地插入到数据库中记录数目的正数值。

如果你的数据库带有事务处理，则你必须在BEGIN WORK-COMMIT WORK或BEGIN WORK-ROLLBACK WORK块中包含所有的PUT语句。（参见紧跟在事务处理讨论之后的“数据完整性”一节内容）。

考虑如下所述的使用插入指针的优缺点：

- 如果程序不经常需要插入记录，则建立插入指针的系统开销可能大于对纪录进行缓冲所增加操作带来的开销。
- 如果你想使得别的程序在记录放入缓冲区后能立即取出记录，你就必须显式地刷新缓冲区以便那些记录放入表中。每一次FLUSH都减少了缓冲的优点。如果你不想要插入缓冲，请使用规范的INSERT语句。

小心！不关闭指针而退出程序将使得缓冲区不刷新。因最后的刷新动作丢失，故记录可以被插入到该缓冲区中。你不能靠结束程序去关闭指针和刷新缓冲区。

尽管COMMIT WORK和ROLLBACK WORK语句可以关闭所有打开的指针，但不要把这两个语句中的一个用于此目的。如果一个COMMIT WORK语句试图关闭一个以上的指针并在关闭过程中失败了，那么你便不知道是哪个指针出了问题，也不知道插入到数据库中的记录数量。

在每一次插入后和你做了CLOSE语句后，总要检查一遍变量sqlca.sqlcode的值。

1.5.6 数据完整性

RDSQL以几种方式提供数据的完整性。你可以防止不同的数据库用户同时修改相