《ORACLE 关系数据库系统使用说明》之七

# PRO*C

# 用户指南

国家统计局计算中心

《ORACLE 关系数据库系统使用说明》之七

# PRO*C

# 用 户 指 南

国家统计局计算中心

# 前　言

ORACLE RDBMS是美国Oracle公司的关系型数据库管理系统产品，它采用SQL(美国国家标准学会——ANSI于1986年10月16日正式颁布SQL语言为美国国家标准）作为数据库查询语言，拥有60多种大中型机、小型机、微机上的运行版本，包含有报表生成、全展幕表格、高级语言接口等多种实用工具软件产品，是一种较好的高性能关系型数据库管理系统，其1986年8月宣布排出的版本5.1产品又具有了初步的分布式关系数据库管理系统的先进功能。

本套《ORACLE关系数据库系统使用说明》资料系我们对绝大多数ORACLE RDBMS产品说明书进行编译后形成，它包括了ORACLE产品的以下内容：

1. ORACLE RDBMS Kernel
   - Basic Utilities
   - SQL*Report
   - SQL*Loader
2. ORACLE SQL*Plus
3. ORACLE SQL*Forms
4. ORACLE SQL*Calc
5. ORACLE Pro*C
6. ORACLE pro*Fortran
7. UNIX系统的ORACLE安装与用户指南
8. MS-DOS的ORACLE安装与用户指南
9. ORACLE Report*Writer
10. ORACLE SQL*Net

本套资料由国家统计局计算中心组织编译，参加工作的有：吕春莲、王延华、孔祥清、胡杉、王珙、谢坤等同志。

本套资料供国家统计局系统内部使用。

本册为PRO*C用户指南。

# 目　录

# 第一部分 PRO*C 预编译界面

以下各章将全面论述PRO*C预编译界面。第一部分讲述 PRO*C程序的总体结构和PRO*C语句的语法，还介绍了有关错误处理、事务控制、指针器、预编译器的用途和命令选择项以及编程规则。另外，第一部分给出许多例题来帮助理解概念。

## 第一章 PRO*C介绍

本章对Pro*C作一个大概介绍，讨论为什么要引入Pro*C，给出一般概念和定义，研究部分Pro*C程序，列出了Pro*C程序中各种语句类型。

SQL数据语言是非过程性语言；也就是说，大部分语句的执行独立于它前面的语句和后面的语句。相比之下，常见的程序设计语言如C、FORTRAN、COBOL等，它们都是过程性语言，它们基于象循环（loops）、分支（branches，是件 if/then）这样的程序结构。虽然SQL语言是一种功能很强的语言。但是它的非过程性确实给SQL带来了一些限制。

SQL是一种非过程性语言，有一定的局限性，但它可以嵌入过程型程序设计语言中，我们把这种程序设计语言叫做宿主语言。SQL 和宿主语言相结合编写出的应用程序比单独使用SQL或宿主语言具有更强的功能和灵活性。

ORACLE关系数据库管理系统提供了一些方便的工具，利用这些工具，程序员可以用宿主语言编写程序来访问 ORACLE 数据库中的数据。目前可以在C、FORTRAN、COBOL、PASCAL和 PL/1中使用这些工具。

### 1.1 什么是PRO*C

PRO*C是一个预编译器，它的功能是把包含SQL语句的C程序转化为能够访问和操纵ORACLE数据库中数据的C程序。作为一个预编译器，Pro*C把它输入文件中的 EXEC SQL语句转化为一个在输出文件里的合适的ORACLE调用。和处理普通的C程序一样，输出文件进而被编译、链接和运行。

把Pro*C（或Pro*Fortran，Pro*PL/I）与ORACLE调用界面（以前称为Pro*SQL）相比较，ORACLE调用界面（OCI）是ORACLE 数据库之上的调用界面，是同ORACLE数据库之间的接口。ORACLE调用界面允许用户把 ORACLE 调用直接嵌入到象C、FORTRAN COBOL这样的高级语言中。每一事务都是通过多个调用和使用指针来实现。

## 1.2 PRO*C的特点

PRO*C预编译器有如下几个特点：

- Pro*C调用比Pro*SQL（OCI）调用更容易理解。
- 运行时，一个Pro*C（OCI）调用被自动地翻译成与之等价的几个库调用，因而降低了运行时间。
- 一个程序可使用几个数据库中的数据。
- 可以对多个程序分别进行预编译，然后连结在一起运行。

## 1.3 一般概念

编写和运行普通的C程序分为四个步骤：

1．编写程序。
2．编译这个C程序，得到一目标文件。
3．连结这一目标文件得到一可执行文件。
4．运行可执行文件。

而编写和运行Pro*C程序需要在开始附加一些步骤，这五个步骤是：

1．编写Pro*C程序。
2．用Pro*C对这一程序进行预编译，得到输出文件。
3．对预编译的输出文件（程序）进行编译，得到一目标文件。
4．连结这一目标文件，得到一可执行文件。
5．运行程序完成工作。

### 1.3.1 Pro*C命令

运行Pro*C预编译器所需要的语法、可选项将在第七章有关Pro*C调用中详细描述（PCC命令），在此章中还介绍了有关预编译一些文件一起运行以及将Pro*C和OCI一起使用的信息。

### 1.3.2 C命令和SQL语句的混合使用

C程序中任何合法的SQL语句都可以被执行。但是，有些SQL语句是必需的，而且SQL语句必须按照基本的顺序出现；而对于C程序行，只要它合乎C程序的语法规则，它就可以出现在程序中的任何一点。这些内容将在以下章节中详细讨论。

### 1.3.3 前缀EXEC SQL的命令

把SQL语句嵌入到宿主语言程序中后，为了明显地区分SQL语句和宿主语句，在每个SQL语句前加上“EXEC SQL”。于是，预编译程序的目的之一就是把以EXEC SQL开始的语句翻译成用于数据库调用的源语言（宿主语言）代码。

### 1.3.4 以EXEC ORACLE做为前缀的命令

大部分的Pro*C语句前要加EXEC SQL，但也有固定的一小部分语句前需要加EXEC ORACLE。“EXEC ORACLE 选择项”（见2.2.3）中列举出了这些语句并对这个问题进行了讨论。这些语句与SQL不兼容，需要用 ORACLE 预编译器处理。

### 1.3.5 SQL可执行语句和说明语句

Pro*C程序中的SQL语句分为两种类型：可执行语句和说明语句。所有的 SQL 语句，无论是可执行的语句还是说明语句，其前面都要加EXEC SQL。

SQL可执行语句是那些能够对数据库产生实际调用的SQL语句。这些语句包括：数据操纵语句（DML）、数据定义语句（DDL）和数据控制语句(DCL)。但并不局限于这三类语句。一条SQL可执行语句执行后，在SQLCA中留下一组返回码。（这里SQLCA代表SQL通讯数据区）。

一个逻辑工作单元以一条可行的SQL语句开始，CONNECT语句除外；因而CONNECT，COMMIT，ROLLBACK 语句后面的第一条SQL可执行语句是一个新的逻辑工作单元的开始。

说明语句并不产生代码，它对逻辑工作单元不产生影响，SQLCA 也不受这些说明语句的影响。图 3 列出了SQL说明语句。

| #SQL 说明语句 |
| --- |
| BEGIN DECLARE SECTION |
| END DECLARE SECTION |
| WHENEVER…… |
| DECLARE CURSOR…… |
| INCLUDE…… |

图 3、SQL说明语句

### 1.3.6 Pro*C程序的两个部分

一个Pro*C程序包含两个部分，这两个部分都是Pro*C程序所必需的：

- 程序头
- 程序体

程序头定义变量，为Pro*C程序做一般的准备性工作。程序体包含有Pro*C调用，如INSERT语句和UPDATE语句，这些调用完成对ORACLE数据的操纵。

下一章（“PRO*C程序的组成”）将讨论程序头，并给出一些Pro*C程序示例，来说明Pro*C程序的基本规则。

# 第二章　PRO*C程序的组成

本章讨论Pro*C程序的必不可少的组成成分，定义一些Pro*C术语，并给出一些例子。本章末尾给出一些短小的程序例子，通过这些例子你可以了解到编写Pro*C程序至少应该干些什么，Pro*C程序至少应该包括哪些语句。

本章中的C程序指的是要经过预编译的模块，这一模块可以定义多种功能，也可以是应用程序的多个模块之一。

## 2.1　程序头

程序头是Pro*C程序的开始部分，它分为三个部分：

1．说明部分。其功能是定义宿主变量。

2．INCLUDE SQLCA语句。它定义SQL通讯数据区SQLCA，SQLCA可用于错误处理。（这一语句相当于C语句中的＃include语句）

3．CONNECT语句。这一语句用于与ORACLE RDBMS连结。

只有C语言语句可以放在这些语句之前。

### 2.1.1　程序头：DECLARE部分

说明部分用来说明将在C程序中用到的宿主变量。DECLARE部分的第一条语句是：

```
EXEC SQL BEGIN DECLARE SECTION;
```

最后一条语句是：

```
EXEC SQL END DECLARE SECTION;
```

在这两条语句之间只允许一种类型的语句出现，这种语句用于说明宿主变量或指示变量。

每一个预编译单元只允许出现一个BEGIN/END DECLARE 部分，但一个程序可以包含多个无关的预编译单元。说明部分可以是全程的，也可以是局部的。

当一个宿主变量或者指示变量被C程序中的EXEC SQL语句所引用，但这一变量并没有在说明部分中说明时，那么经过预编译时会出现以下错误信息：

```
Undeclared host variable  a  at  line b  in file c
```

这里a代表变量名，b是行号，c是文件名。

**1．宿主变量**

如果一个变量既在SQL语句中引用，也在宿主语言语句中引用，那么这个变量必须定义成宿主变量。宿主变量必须在DECLARE部分用宿主语言说明。它们的数据类型不必与表中的ORACLE数据类型相匹配，ORACLE将完成宿主语言数据类型和ORACLE数据类型之间的转换。

```
EXEC SQL BEGIN DECLARE SECTION
int pempno;
Char  pname〔11〕
int   pdeptno;
EXEC SQL END DECLARE SECTION
```

图4．DECLARE部分的一个示例

在图4的例子中说明了三个宿主变量：PEMPNO、PNAME和PDEPTNO，这三个变量可以出现在程序体的SQL语句中。例如：

```
EXEC SQL SELECT DEPTNO, ENAME
         INTO :PDEPTNO, :PNAME
         FROM EMP
         WHERE  EMPNO=:PEMPNO;
```

**2．宿主变量使用准则**

一个宿主变量的引用必须遵守如下规则：

- 必须在DECLARE部分被显式地说明；
- 引用宿主变量的语句和说明这一变量的语句对宿主变量必须采用相同的大、小写格式；
- SQL语句引用宿主变量时，宿主变量名前加一冒号（：）；
- C语言语句中宿主变量前不加冒号；
- 宿主变量名不能是SQL保留字（见附录C）；
- 只能在能够使用常量的位置上使用宿主变量；
- 可以有与之相关联的指示变量。

（指示变量在下一小节讨论。）注意，在版本5中Pro*C并不是把终结串置空。

**3．指示变量**

指示变量是可选择变量，每一个指示变量都对应着唯一的宿主变量。指示变量在处理NULL值时特别有用，可用来存贮单个域的返回码以指示“空值返回”或“字符域截取”。

指示变量也可用于插入NULL值。（见“使用指示变量中的返回值”）。

**4．指示变量使用准则**

- 指示变量必须在DECLARE部分说明；
- 引用指示变量的语句和说明这一变量的语句对这一变量必须采用相同的大小写格式；
- 必须把指示变量说明为具有两个字节长的整数型（是“int”还是“Short int”因C编译而异）；

• SQL语句引用指示变量时，变量前加冒号（：）；

• C语句中的指示变量前不加冒号；

• 指示变量名不能是SQL的保留字（见附录C）；

• 指示变量不能单独使用，与之对应的宿主变量必须放在其前才能使用。

例如，现在我们在上一检索中增加两个指示变量DEPTNOI和NAMEI；

```
EXEC SQL SELECT DEPTNO, ENAME
         INTO :PDEPTNO:DEPTNOI, :PNAME:NAMEI
         FROM EMP
         WHERE EMPNO = :PEMPNO;
```

在“使用指示变量中的返回值”中将详细讨论指示变量。

**5．指针作为宿主变量**

在SQL中可以使用指针变量（简单指针变量）。和其他宿主变量一样，指针变量也要在DECLARE部分中说明，说明时，指针变量名前带一星号*。例如：

```
EXEC SQL BEGIN DECLARE SECTION;
int  i, j, *intptr;
char *CP;
EXEC SQL END DECLARE SECTION;
```

在SQL语句中使用指针变量时，变量名前带一冒号而不是星号，因为星号是隐含的：

```
SELECT INTFIELD INTO :intpr FROM……;
```

除字符指针以外，所引用对象的长度由对象类型长度决定，而类型是在说明部分给定，所以，所引用对象的长度、大小在说明语句中指定。对于字符指针，引用对象被认为是空终结串，它的长度在运行时由串长度函数决定。

注意，如果一个指向字符变量的指针用作一输入宿主变量（在USING子句中），那么，它的长度也是由一个串长度调用决定。得出的结果也许并不是所期望的，因为得出的长度值是当前值而不是最大值。

**6．伪类型VARCHAR**

为了容纳变长度的字符串，Pro*C允许使用VARCHAR伪类型。VARCHAR只有DECLARE部分引用，可以认为它是一种C语言的扩充类型，也可认为是一种预定义的数据结构。描述如下：

```
EXEC SQL BEGIN DECLARE SECTION;
VARCHAR jobDesc[40];
EXEC SQL END DECLARE SECTION;
```

这一说明部分中的“VARCHAR jobDesc[40]”，等价于下面的说明语句：

```
Struct {
    unsigned /* 2 bytes */ Short int  len;
    unsigned char arr[40]
    } jobDesc;
```

这里，字符数组的大小与VARCHAR说明中所指定的大小相等，长度域len用来指示变长度字符串的当前长度。

根据定义，VARCHAR变量并不是空终结的，VARCHAR 结构中的len指定串长度，所以 VARCHAR变量作为输出变量时，len由 ORACLE 赋值。当作为输入变量时，len必须由程序给定。在数组arr中，任何null值均被忽略或重写。

字符串可以用两种方法定义，它可以看作是一字符数组，也可以看作一字符串指针：

```
char Str〔100〕;
Char *str;
```

字符数组的长度是由字符串长度来决定的。对于输入和输出，这个长度是接收的长度。输出时，ORACLE将把输出缓冲区填空，用户程序将把输入字符串赋空。

在 SQL语句中，使用聚合名来引用 VARCHAR 变量；和其他变量一样，变量名（聚合名）前加一冒号。下面是运用VARCHAR变量：jobDesc的两个检索：

```
EXEC SQL SELECT JOBDESC INTO :jobDesc FROM EMP
  Where empno= :empno;
```

或者

```
getSTring (jobDesc.arr) ;
/*从用户程序中获得job的描述 */
jobDesc.len=strlen (jobDesc.arr) ;
/*获得字符串的长度 */
  EXEC SQL SELECT … INTO … FROM EMP
      WHERE JOBDESC= :jobDesc;
```

当一个 VARCHAR 变量作为输出宿主变量（在 INTO 子句中）时，ORACLE RDBMS 为长度域len置值，当作为输入宿主变量时（在WHERE 子句中），用户程序必须对长度域置值。

你可以用VARCHAR *变量名这一格式来定义指向VARCHAR类型的指针变量，也可以象C标准类型那样使用反复定义法。例如：

```
VARCHAR    ID1〔10〕, *ID2, …, IDN〔4〕
```

等价于：

```
VARCHAR    ID1〔10〕;
VARCHAR    *ID2;
VARCHAR    IDN〔4〕;
……等
```

当涉及到字符指针时，通过执行strlen（ ）函数以求得字符串缓冲区的长度。输入时，用字符指针作为输入宿主变量比较方便，因为用户程序将用不着为 len（VARCHAR变量中的len）置值，也将用不着为缓冲区（字符数组）填充空白符。但对于输出，使用字符串指针不太好，因为ORACLE必须通过调用 Strlen（ ）来求得缓冲区长度，因而用户也不必清楚缓冲区长度。所以，把输出缓冲区定义成字符数组比定义成

字符指针要方便。

**7. 说明部分的注意事项**

不要把类型定义语句所定义的名字当作变量使用，Pro＊C不认识这些名字。出现这种情况时，Pro＊C将提示错误信息：变量未定义。

不要引用以前定义的结构。同样，也不要把一个变量说明为某种结构类型。但并不等于你不能用结构，可以用下列三种方法使用结构：

- 简单变量与结构的某一域之间相互拷贝数据；
- 设结构中某一域的类型为T，为了访问这一域，定义一个类型为＊T 的变量（指针变量），使这一指针指向结构的这一域；
- 使用描述符和动态语句给宿主变量的定位带来灵活性，在这种情况下可使用结构。

### 2.1.2 程序头：说明SQL通讯数指区

每一个Pro＊C程序必须有事件处理的功能，这可以在程序头包含一个SQL 通讯数据区实现。为了得到这一功能，只便要在程序头部分中加上这么一行语句：

```
EXEC SQL INCLUDE SQLCA;
```

EXEC SQL INCLUDE 用来把其它文件包括到你的Pro＊C程序中，它与C语言中的＃include相似

你可以用EXEC SQL INCLUDE语句把含有EXEC SQL INCLUDE 语句的文件包括到你的 Pro＊C程序中。这种嵌套的最大层数由你所在的操作系统对打开文件数目的限制决定。

INCLUDE语句引用文件时应当数到大、小写匹配（·h 文件一般采用大写）。进行预编译时，Pro＊C必须能够对SQLCA文件定位，为此，你可以有三种选择：

- 使用"INCLUDE＝…"命令行可选项。
- 把文件存入操作系统公用区中，这样，Pro＊C就能在O/S公用区（例如VMS中的SYS $ORAACLE：SQLCA）中找到文件。
- 把SQLCA文件拷贝到某一目录或磁盘中，从这一目录或磁盘来执行PCC。

如果遇到什么问题，找你的DBA。

EXEC SQL INCLUDE SQLCA语句使得Pro＊C把SQL通讯数据区包括到的你程序中。SQLCA在你的程序中可以是全程的，也可以是局部的包含，这由 INCLUDE 语句在你程序中的逻辑位置决定。SQLCA 含有许多变量的定义，这些变量用于和 ORACLE 的通讯。当程序进行预编译时，ORACLE 用许多宿主语言的变量定义来代替 INCLUDE语句；在程序运行期间，这些变量说明能够使得ORACLE 与你的程序进行交互，相互传递各种操作的有关状态信息。下面是部分SQLCA所包含的信息：

- 警告标志和事针信息
- 错误码
- 操作诊断

例如，通过SQLCA你可以查看一个SQL语句是否执行成功，加果成功，插入或修

改了多少行。如果一个语句因出错而被终止，你可以查出出错原因。

每执行一个操作，ORACLE都要把反馈信息记录在 SQLCA中，在程序的任何一点，用户都可以根据反馈信息采取不同的行动，方法是利用 SQLCA所提供的选择项。

在选择项缺省的情况下，Pro*C程序对错误将置之不理，如果可能，将继续执行程序。利用 SQLCA 中的变量，程序员能够控制一定情况下所执行的动作。例如，使用 WHENEVER 说明语句，你可以指定当检测到错误、警告或特别事件时所执行的动作。这些动作可以包括停止程序执行、转移执行和继续执行。（WHENEVER 语句和其选择项在“WHENEVER 语句”中讨论。）

SQLCA中有许多变量，在此只介绍两个。

sqlca.sqlcode变量：在每一条SQL可执行语句执行后，这一变量存放结果码，这一结果码是一整数，它指出语句的执行情况。零值表示执行成功；负值代表一个ORACLE 错误（可以在“ORACLE 错误信息和代码”中查到），正值表示执行成功，它也代表一个状态码（例如文件结束）。目前，只有一个正值码：1403，它表示“行没有找到”，或代表最后一行已经查询到。

sqlca.sqlwarn 变量：它事实上是一个包含八个警告标志的值组。对于它的第一个元素 sqlca. sqlwarn[0]，如果其它标志被置信，它就被置为“W”，所以它是判断是否出现问题的快速指示器。

如果想了解有关错误处理选择项的更详细的内容，请看“信误检测和恢复”一章。

### 2.1.3 ORACA：对SQLCA的一个扩充

预编译器用户要求能够从预编译器的运行环境中得到比从SQLCA 中得到的更多的信息，而SQLCA是为“标准SQL”准备的数据区，所以1.0版本引入一个ORACLE 通讯数据区，叫做ORACA。

为了在程序中使用ORACA，必须用EXEC SQL INCLUDE语句把ORACA的定义包括到你的程序中，然后通过EXEC ORACLE OPTION 或作为一个命令行选择项选择项ORACA = YES。

### 2.1.4 ORACA中的信息

**ORACA包含下列信息：**

当前SQL语句正文（orastxt）：当遇到信误时orastxt特别有用，你可以检查 ORACLE RDBMS所分析的语句的正文（这里所说的语句是指与指示器相联的语句）。由预编译分析的语句（如CONNECT，FETCH，COMMIT）不在ORACA中出现。ORACA中最多只能出现SQL语句的前70个字符，语句的格式与 SQLCA 中错误信息的格式相同。

出错的文件名（orasfnm）：如果一个应用程序包括几个文件，在编译出错时，ORACA将识别错误是在哪个文件中发生的。

出错的行号（oraslnr）：在文件存放出现错误的行号。

ORACA也包括下列标志，可以使用这些标志把运行选择项传递给 ORACLE 程序