第一篇　PRO＊C 程序设计

第一章　PRO＊C 程序概述

§1.1　什么叫 PRO＊C 程序

在 ORACLE 数据库管理系统中，有三种访问数据库的方法：

(1)用 SQL＊PLUS，它用 SQL 命令以交互的方式访问数据库。

(2)用第四代语言应用开发工具开发的应用程序访问数据库，这些工具有 SQL＊Forms、SQL＊ReportWriter、SQL＊Menu 和 ORACLE＊FORMS 等。

(3)利用在第三代高级语言内嵌入的 SQL 语句，或 ORACLE 库函数调用来访问数据库。

ORACLE 支持在六种高级语言中嵌入 SQL 语句，它们是 C，FORTRAN，PASCAL，COBOL，PL/I 和 Ada。我们把这些语言统称为宿主语言，用它们开发的应用程序叫 PRO 程序。如果宿主语言是 C，则相应的程序叫 PRO＊C，如果宿主语言是 FORTRAN，则相应的程序叫 PRO＊FORTRAN 程序，……。本篇重点介绍 PRO＊C 程序，但它的基本思想和方法也适于 PRO＊FORTRAN、PRO＊COBOL、PRO＊PASCAL、和 PRO＊PL/I 等。而 PRO＊Ada 有些特殊，应参考 PRO＊Ada 的有关手册。

在 PRO＊C 程序中可以嵌入 SQL 语句，利用这些 SQL 语句可以完成动态地建立、修改和删除数据库中的表，也可以查询、插入、修改和删除数据库表中的行，还可以实现事务的提交和回滚。

在 PRO＊C 程序中还可以嵌入 PL/SQL 块，以改进应用程序的性能，特别是在网络环境下，可以减少网络传输和处理的总开销。

利用第三代高级语言内嵌入 SQL 语句来开发应用程序有以下三点好处：

(1) 它把过程化语言和非过程化语言相结合，形成一种更强有力的开发工具。利用它可以开发出满足各种复杂要求的应用程序，还可以引用窗口技术和鼠标技术等。

(2) 可以使开发的应用程序具有管理系统资源使用(如内存分配)、SQL 语句执行和指示器等能力。

(3) 提高了应用程序的执行速度，因为它把 SQL 语句翻译成相应的 ORACLE 库函数调用。

§1.2　ORACLE 预编译程序

ORACLE 预编译程序是一种强有力的应用程序开发工具，它输入 PRO 源程序，经处理

输出相应高级语言的源程序。例如,ORACLE 的 PRO * C 预编辑程序,它输入 PRO * C 源程序,输出 C 的源程序。

ORACLE 提供了六种预编译程序(统称 PRO 系列),它们分别支持以下六种高级语言:

C
Pascal
FORTRAN
COBOL
PL/1
Ada

源程序中嵌入的 SQL 语句经预编译程序处理后,被翻译成相应 ORACLE 库函数的调用。

ORACLE 预编译程序的功能和特点如下:

1. 能用六种通用的高级程序设计语言中的任何一种来编写应用程序。
2. 嵌入的 SQL 语句完全遵循 ANSI 标准。
3. 可采用动态 SQL 技术。
4. 能开发出满足各种需求的应用程序。
5. 自动实现内部和外部数据类型转换。
6. 可嵌入 PL/SQL 块。
7. 能在命令行和程序行上指定预编译可选项。
8. 能用数据类型等价来控制 ORACLE 解释输入数据和格式化输出数据的方式。
9. 能分别预编译。
10. 能检查嵌入的 SQL 语句或 PL/SQL 块的语法和语义。
11. 可利用 SQL * Net 并行存取多个结点上的 ORACLE 数据。
12. 可使用数组 SQL 变量。
13. 能进行条件预编译。
14. 可用高级语言编写 SQL * Forms 的用户出口。
15. 能用 SQLCA 和 ORACA 进行错误诊断。

§1.3 PRO * C 程序的组成及举例

1.3.1 PRO * C 程序举例

为了对 PRO * C 程序有一个感性认识,我们首先来浏览一个 PRO * C 程序实例:

例 1.1 简单查询

```
/*******************************************
 *   功能:
 *   该程序先与 ORACLE 相连接,然后提示用户输入一个职员号,
 *   最后根据该职员号查询数据库,查询出该职员的名字、工资、附加工资,
 *   并显示结果。
```

```
*   当输入职员号为 0 时，程序结束。
*********************************************************/
/********* 外部说明部 *********/
    #include <stdio.h>
    /*          说明段，说明 SQL 变量          */
    EXEC SQL BEGIN DECLARE SECTION;
        VARCHAR username[20];
        VARCHAR password[20];
    EXEC SQL END DECLARE SECTION;
    /*          SQL 通讯区说明          */
    EXEC SQL INCLUDE sqlca;
    /*          C 函数说明，错误处理          */
    void sqlerror();
/********** 程序体 **********/
    main()
    {
     /****** 内部说明部 *********/
     /* 内部说明段，说明局部 SQL 变量 */
     EXEC SQL BEGIN DECLARE SECTION;
       int    emp_number;
       VARCHAR emp_name[15];
       float salary;
       float commission;
     EXEC SQL END DECLARE SECTION;

     /*          C 变量的说明          */
     int total_number;

     /* 登录到 ORACLE */
     strcpy(username.arr, "SCOTT");          /* 用户名 */
     username.len=strlen(username.arr);
     strcpy(password.arr, "TIGER");          /* 口令 */
     password.len=strlen(password.arr);
     EXEC SQL WHENEVER SQLERROR DO sqlerror();
                    /*错误处理说明语句*/
     EXEC SQL CONNECT :username IDENTIFIED BY :password;
                    /* 连接到 ORACLE 数据库管理系统 */
     printf("\nConnected to ORACLE as user: %s\n",username.arr);
```

```
        /* 循环查询职员信息 */
        total_number = 0;
        while (1)
    {   emp_number = 0;
        printf("\nEnter employee number (0 to quit): ");
        scanf("%d", &emp_number);
        if(emp_number == 0) break;
        /* 查询语句 */
        EXEC SQL WHENEVER NOT FOUND GOTO notfound;
        EXEC SQL SELECT ENAME, SAL, COMM
                INTO :emp_name, :salary, :commission
                FROM EMP
                WHERE EMPNO = :emp_number;
        /* 输出查询结果 */
        printf("\n\nEmployee\tSalary\tCommission\n");
        printf("--------\t------\t----------\n");
        emp_name.arr[emp_name.len] = '\0';
        printf("%-8s\t%6.2f\t%6.2f\n",
                emp_name.arr, salary, commission);
        total_number = total_number + 1;
        continue;
notfound:
        printf("\nNot a valid employee number - try again.\n");
    }
    printf("\n\nTotal number queried was: %d\n", total_number);
    printf("\nHave a good day.\n");
    /* 结束处理,退出 ORACLE 系统 */
    EXEC SQL COMMIT WORK RELEASE;
    exit(0);
    }
    /* 错误处理 */
    void sqlerror()
    {
        EXEC SQL WHENEVER SQLERROR CONTINUE;
        printf("\nORALCE error detected:\n");
        /* 打印错误文本 */
        printf("\n% .70s \n", sqlca.sqlerrm.sqlerrmc);
        /* 回滚事务 */
        EXEC SQL ROLLBACK RELEASE;
```

```
    exit(1);
}
```

从上例我们可以看到：该实例程序是由外部说明部和程序体两部分所组成。

外部说明部主要说明程序中所引用的外部变量及函数等，它由说明段、通讯区说明和C的有关说明等所组成。

说明段是由SQL语句

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

开始，由

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

结束。其间包含若干类型说明语句，用以说明SQL变量的类型。

语句

```
EXEC SQL INCLUDE Sqlca;
```

说明SQL通讯区，用以记录执行每一个嵌入SQL语句的状态信息。

程序体是由若干函数(简单的可以仅由一个主函数)所组成，其中有一个主函数。一般来说，每一个函数又包含如下成分：

- 内部说明部：说明函数内所用的局部变量，它的组成类似外部说明部，也是由说明段、局部通讯区说明及C的局部变量说明。
- C语言的可执行语句。
- 嵌入式SQL语句或PL/SQL块：用以实现对数据库中数据的操作。

在主函数内应包含一个登录语句，如

```
EXEC SQL CONNECT : username IDENTIFIED BY : Password;
```

以实现与数据库系统的连接。

在函数内除包含一般的SQL语句(SELECT，INSERT……)外，还应包含错误处理的说明语句，如

```
EXEC SQL WHENEVER……;
```

以便指出在执行SQL语句时，如果发生错误，应如何处理。

上例展示了具有一个源文件的程序之组成，对于具有多个源文件的程序，其每个源文件的组成与上述类似。

1.3.2 PRO*C程序的一般组成

从上述的例子我们可以得出这样一个结论：PRO*C程序实际是内嵌有SQL语句或PL/SQL块的C程序，因此它的组很类似C程序。但因为它内嵌有SQL语句或PL/SQL块，所以它还含有与之不同的成分。二者构成的差别可用图1-1和图1-2表示。

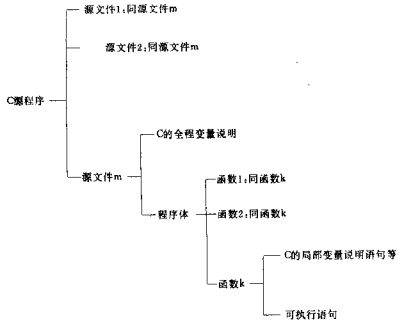


图1-1　C源程序的组成

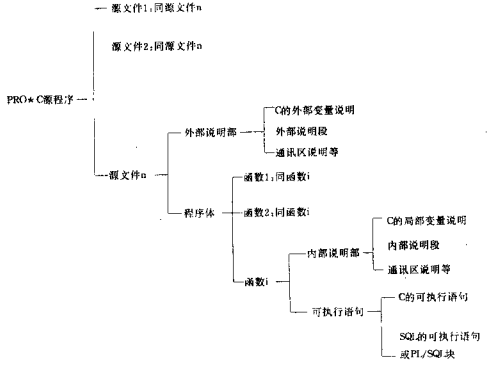


图1-2　PRO*C源程序的组成

注：C或PRO*C的一个源文件也可以只包含说明部分，而不包含程序体。

§1.4 开发和运行一个 PRO＊C 应用程序的基本步骤

PRO＊C 应用程序的开发和运行的基本步骤类似于 C 程序，唯一不同之处是在进行 C 编译之前需先进行预编译。图 1-3 说明应用程序的开发步骤。图 1－4 表示 PRO＊C 程序的处理过程。

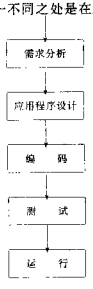


图1-3　应用程序开发步骤

§1.5 PRO＊C 程序书写格式的几点说明

为了使书写的程序可读性强，在编码时应遵循软件工程中所规定的编码规则。前面 1.3 节的实例程序也同时展示出一个 PRO＊C 程序的典范书写格式，供读者参考。读者应特别注意以下几点：

1. 要采用锯齿形的书写格式，

2. 要正确使用注释，

在 SQL 语句中，凡能放置空格的地方(除 EXEC SQL 关键字之间外)都可放置 C 语言风格的注释(/＊…＊/)。除此之外，还可在 SQL 语句行的末尾放置 ANSI 风格的注释，如下例所示：

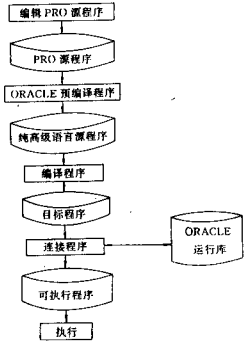


图 1-4　PRO＊C 程序的处理过程

```
EXEC SQL SELECT ENAME, SAL
    INTO :emp_name, :salary     -- output host variable
    FROM EMP
    WHERE DEPTNO=:dept_number;
```

但注释不能嵌套。

3. 嵌入式 SQL 语句或 PL/SQL 块中的关键词应大写。如 SELECT，……。

§1.6 参考资料

本篇参考了如下一些资料：

1. Programmer's Guide to the ORACLE precompilers, V1.5, part NO. 5315－15－1292

2. Pro＊C Supplement to the ORACLE precompilers Guide, V1.5, part NO. 5452－15－1292

3. ORACLE Server SQL Language Reference Manual, Part NO. 778--70

4. <<ORACLE 应用系统开发工具>> 孙宏昌、刘金亭、何毅华著

第二章　PRO＊C 程序设计的基础知识

§2.1　说明段

2.1.1　SQL 变量

在 PRO＊C 程序中含有 C 和 SQL 两种语言的语句，为了解决它们之间的信息传递问题，特引入了一种变量，称为 SQL 变量。

SQL 变量可在 SQL 语句中引用，也可在 C 的语句中引用，而 C 的变量只能在 C 的语句中引用，同 C 语言中的变量一样，SQL 变量也必需是先说明后引用，否则就会出现如下的错误信息：

Undeclared host variable "X" at line N in file "F"

其中 X 是 SQL 变量名，N 是 X 所在源程序行号，F 是该编译单位的源文件名。

2.1.2　说明段

SQL 变量必须在说明段说明。说明段是由语句

EXEC SQL BEGIN DECLARE SECTION;

开始，由语句

EXEC SQL END DECLARE SECTION;

结束。在这两个语句之间只允许包含下列语句：

(1) SQL 变量的类型说明语句

(2) EXEC SQL INCLUDE 语句

(3) EXEC SQL VAR 语句

(4) EXEC SQL TYPE 语句

但不允许包含说明为结构类型的 SQL 变量。

在 PRO＊C 程序中允许定义局部的和全程的说明段，在每个预编译单位内允许有多个说明段，但必须至少有一个全程说明段。说明段的任务是说明变量的数据类型。

例如：

```
EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR name[15];
    VARCHAR password[10];
    int emp_no;
    float salary;
    char dept_name[50];
```

```
    short ind_sal;
EXEC SQL END DECLARE SECTION;
```

§2.2 数据类型和转换

ORACLE 支持内部和外部两类数据类型，内部数据类型描述 ORACLE 按什麼格式把数据存储在数据库的表列中，也用它来描述数据库的伪列。外部数据类型描述如何把数据存储在 SQL 变址中。

在预编译时，说明段中的每一个 SQL 变量都与一个外部类型码相关联。在运行时，把每一个 SQL 变址的外部类型码传递给 ORACLE，以实现内部和外部数据类型之间的自动转换，这种转换称之为缺省数据类型转换。也可使用动态方法或数据类型等价来替代这种缺省数据类型转换。

2.2.1 内部数据类型

表 2-1 中列出了 ORACLE 的所有内部数据类型。数据库中的表列或伪列使用这些数据类型。

表 2-1 ORACLE 内部数据类型

名 字	代 码	描 述
VARCHAR2	1	变长串，(≤2000 字节。
NUMBER	2	定点或浮点数
LONG	8	变长串，2147483647 字节
ROWID	11	16 进制数串
DATE	12	定长日期/时间值，7 字节
RAW	23	定长二进制数据，255 字节
LONGRAW	24	变长二进制数据，2147483647 字节
CHAR	96	定长串，255 字节
MLSLABEL	105	定长二进制标号，5 字节

2.2.2 SQL 伪列和函数

伪列不是一个表中的实际列，但其处理类似于实际列。例如，它们的值也必须从一个表中选择。在有些应用中，使用从伪表中选择伪列的值是方便的。

SQL 识别表 2-2 所列出的伪列：

表 2-2 SQL 所识别的伪列

伪 列	内部数据类型	代 码	描 述
NEXTVAL	NUMBE	2	它为指定列返回一个序号，用它来产生事务处理的唯一序号。
CURRVAL	NUMBER	2	用它返回指定序列的当前序号。引用前应先引用 NEXTVAL，以产生一个序号。
ROWNUM	NUMBER	2	用它返回指定序列的序号，用于表中选择行。
LEVEL	NUMBER	2	用该伪列返回树结构中一节点的层次号。
ROWID	ROWID	11	返回 16 进制的行地址。

SQL 识别表 2-3 所列出的函数：

表 2-3　SQL 所识别的函数

函　数	对应的内部数据类型	代　码	描　述
USER	VARCHAR2	1	返回 ORACLE 当前用户名
UID	NUMBER	2	返回赋给 ORACLE 用户的唯一 ID 号
SYSDATE	DATE	12	返回当前日期和时间

可以在 SELECT、INSERT、DELETE 和 UPDATE 中引用伪列和函数。例如下面的语句是使用 SYSDATE 计算某职员被雇用的月数：

```
EXEC SQL SELECT MONTHS_BETWEEN(SYSDATE, HIREDATE)
    INTO :months_of_service
    FROM EMP
    WHERE EMPNO=:emp_number;
```

2.2.3　外部数据类型

外部数据类型包括全部的内部数据类型和普通宿主语言中所提供的几个数据类型。表 2—4 列出了全部的外部数据类型。

表 2-4　外部数据类型

名　字	代　码	描　述
VARCHAR2	1	用以存储变长字符串，最大长度 2000 字节。在输入时，ORACLE 删去尾部空格后再存入表列中。
NUMBER	2	二进制数，用来存储定点数或浮点数。可指定精度(1～38)和定标(－84～127)。
INTGER	3	有符号整数，用 2 或 4 字节的二进制数表示。
FLOAT	4	浮点数，通常要求 4 或 8 字节存储。
STRING	5	是以 Null 结尾的字符串，其它类似 VARCHAR2。
VARNUM	6	变长二进制数，它类似于 NUMBER，唯一区别是第一字节存储该值长度。
LONG	8	变长字符串，最大长度 2G 字节，其他类似 VARCHAR2。
VARCHAR	9	变长字符串，它含 2 字节的长度字段和小于 65533 字节的串字段，对于 VARCHAR 数组元素，串字段最大长度为 65530。
ROWID	11	二进制值，标识表中的行。占用 13 字节。
DATE	12	定长日期/时间值，占用 7 字节，自左至右存放世纪、年、月、日、小时、分和秒。注
VARRAW	15	变长二进制数据，存储二进制数据或字符串。它由 2 字节长度字段和 65533 字节的串字段组成。
RAW	23	定长二进制数据，存储二进制串，最大长度 255 字节。
LONGRAW	24	变长二进制数据，最大长度 2G 字节，其它同 RAW。
UNSIGNED	68	无符号整数，是二或四字节的二进制数。必须指定长度。
LONGVARCHAR	94	变长字符串，也是由长度和串字段所组成，长度占 4 字，串字段(2G－5)字节。
LONGVARRAW	95	变长二进制数据，其它类似于 LONGVARCHAR。
CHAR	96	定长字符串，最长 255 字节。
CHARZ	97	C 中定长以 null 结尾的字符串，最长 255 字节。
MLSLABEL	106	变长二进制数据。

注：日期的表示格式如表 2-5。

表 2-5 DATE 格式

字 节	1	2	3	4	5	6	7
含 义	世纪	年	月	日	小时	分	秒
例 子							
06-DEC-1990	119	190	12	6	1	1	1

2.2.4 SQL 变量的数据类型

在说明段能为 SQL 变量指定的数据类型如表 2-6 所示。

表 2-6 SQL 变量的数据类型

C 的数据类型	描 述
char	单字符
char[n]	n 个字符数组
int	整数
short	短整数
long	长整数
float	单精度浮点数
double	双精度浮点数
VARCHAR[n]	变长字符串

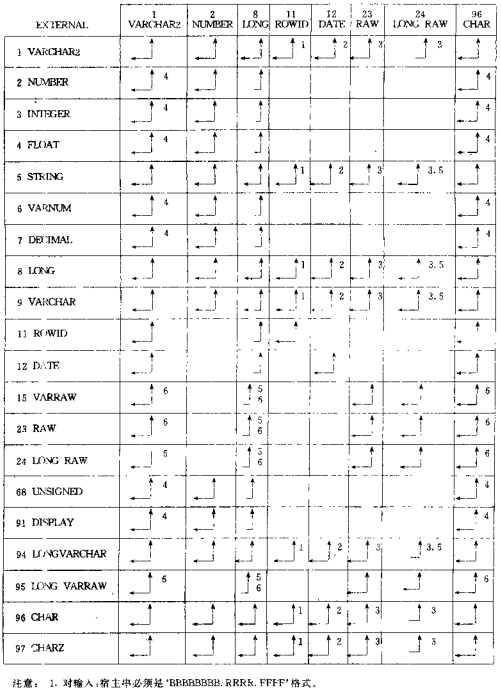
这些数据类型实际上就是 C 语言的数据类型，其中 VARCHAR 可视为 C 数据类型的扩充。

2.2.5 数据类型转换

在预编译时，对于每一个 SQL 变量都分配一个外部数据类型。例如，把 VARCHAR2 外部数据类型分配给字符型 SQL 变量。这样预编译后，每一个变量都与一个外部数据类型码相关联。在运行时，SQL 语句中的每一个 SQL 变量的外部类型码被传递给 ORACLE，ORACLE 使用这些代码实现内外数据类型之间的转换。

在查询时，在把选择的列(或伪列)赋给输出 SQL 变量之前，如果表列和变量的类型不一致时，ORACLE 就把表列的数据类型转换成 SQL 变量的外部数据类型。同样，在把 SQL 变量的值存入数据库表列(或二者进行比较)之前，如果二者数据类型不一致时，ORACLE 就把 SQL 变量的外部数据类型转换成目标列的内部数据类型。

但在进行转换时，内部数据类型与外部数据类型必须相兼容，否则就无法实现。因此，程序的设计者要确保内外数据类型之间是可转换的。图 2-1 表示支持的数据类型转换。



注意：1. 对输入，宿主串必须是'BBBBBBBB.RRRR.FFFF'格式。
对输出，以相同格式返回值。↑表示只输入。

2. 对输入，宿主串必须是缺少 DATE 字符格式。对输出，以相同格式返回列值。 ↑表示只输出。

3. 对输入，宿主串必须是 16 进制格式，输出时以相同格式返回。

4. 对输出列值必须表示一个有效的数，↑表示输入或输出。

5. 长度必须小于或等于 2000。

6. 对输入，列值必须以 16 进制格式存放。对输出，列值必须是 16 进制格式。

图 2-1　支持的数据类型转换

§2.3 数据类型等价

数据类型等价向用户提供一种控制 ORACLE 解释输入数据和格式化输出数据的方法。可把 C 语言的数据类型与 ORACLE 的外部类型等价，也可把用户定义的类型与 ORACLE 外部类型等价。对这两种情况，可指定除 NUMBER(不需要，可用 VARNUM 替代)外的任何外部数据类型。

2.3.1 SQL 变量等价

按照缺省规定，PRO * C 预编译程序把指定的外部数据类型分配给每一个 SQL 变量。如表 2-7 所示：

表 2-7　　外部数据类型与 SQL 变量类型的缺省对应

SQL 变量的数	外部数据类型	数据类型代码据类型
char,char[n],char *	VARCHAR2	1
char,char[n],char *	CHARZ	97(当 MODE=ANSI 时)
int, int *	INTEGER	3
short, short *	INTEGER	3
long, long *	INTEGER	3
float, float *	FLOAT	4
double, double *	FLOAT	4
VARCHAR[n]	VARCHAR	9

在说明段，可通过使用 VAR 语句来把 SQL 变量与外部数据类型等价，以替代这种缺省分配。VAR 语句的格式如下：

EXEC SQL VAR host_variable IS type_name[(legth)];

其中：

host_variable 是 SQL 变量，type_name 是外部数据类型，legth 是该外部数据类型的长度。

SQL 变量等价在以下几方面是有用的：

(1)在 C 程序中，字符串必须以 Null 终结，利用 SQL 变量等价，可保证从数据库表列 SELECT 或 FETCH 出的串以 Null 终结。例如，想把职员名从 EMP 表列中选择到字符数组中，然后传递给要求以 Null 终结串的函数。这时，只需把该宿主数组等价于外部数据类型 STRING 即可，不需再用 Null 显式地结束该串。如下例所示：

```
EXEC SQL BEGIN DECLARE SECTION;
  ...
  int emp_number;
  char emp_name[11];
  EXEC SQL VAR emp_name IS STRING(11);
EXEC SQL END DECLARE SECTION;
...
```

```
main()
{ ...
    EXEC SQL SELECT ENAME INTO :emp_name
        FROM EMP
        WHERE EMPNO=:emp_number;
    printf("\n %s", emp_name);
    ...
}
```

在此例中，EMP 表中 ENAME 的长度是 10 个字符，因此，应分配 emp_name 11 个字符，以便存放 Null 终结符。当把 ENAME 列的值 SELECT 到 emp_name 中时，程序接口就 Null 终结该值，而不需再编码给 emp_name 加 Null 字符的语句。

(2)当想让 ORACLE 的列只存储不解释的数据时，可用数据类型等价。例如，想在 LONG RAW 数据库表列存储浮点数数组时，可把数组与该外部数据类型等价。例如：

```
EXEC SQL BEGIN DECLARE SECTION;
    ...
    float salary[100];
    EXEC SQL VAR salary IS LONRAW(400);
EXEC SQL END DECLARE SECTION;
```

(3)可用数据类型等价来替代缺省分配。例如，如果把 DATE 列 SELECT 到字符型 SQL 变量中时，ORACLE 返回如下 9 字节长的标准格式串：

DD-MON-YY

但是，如果把 SQL 变量与 DATE 外部数据类型等价，则 ORACLE 返回如表 2－5 所示的 7 字节值。

2.3.2 用户定义类型等价

能把用户定义的数据类型与 ORACLE 外部数据类型等价。其方法是，先定义一个适合需要的新数据类型；然后，在说明段，用 TYPE 语句把新数据类型与外部数据类型等价。

用 TYPE 语句能把一个外部数据类型与用户定义的类型等价。其语句文法如下：

EXEC SQL TYPE user_type IS type_name[(length)][(REFERENCE)]

例如，假定用户需要一个保留图形字符串的变长串数据类型。这时可首先定义一个结构，该结构具有一个类型为 short 的长度字段和一个长度≤65533 字节的数据字段。然后用 typedef 定义一个基于该结构的新数据类型。最后再把新数据类型与 VARRAW 外部数据类型等价。如：

```
struct screen{
    short len;
    char buff[4000];
};
typedef struct screen graphics;
EXEC SQL BEGIN DECLARE SECTION;
```

```
        EXEC SQL TYPE graphics IS VARRAW(4000);
        graphics crt; /* 定义类型为 graphics 的宿主变量 */
        ...
    EXEC SQL END DECLARE SECTION;
```

预编译程序为长度字段分配 sizeof(short)个字节，为数据字段分配 4000 或更多的字节(取决于系统进行边界调整的需要)。

能显式地或隐式地把一个用户定义的类型说明为一个指针，然后在 EXEC SQL TYPE 中使用新类型。在这种情况下，必须在该语句的尾部使用 REFERENCE 子句。如下例所示：

```
    typedef unsigned char * my_raw;
    typedef char my_char[10];
    EXEC SQL BEGIN DECLARE SECTION;
        EXEC SQL TYPE my_raw IS VARRAW(4000) REFERENCE;
        EXEC SQL TYPE my_char IS VARRAW(10) REFERENCE;
        my_raw graphics_buffer;
        my_char name
    EXEC SQL END DECLARE SECTION;
    ...
    graphics_buffer=(my_raw) malloc(4004);
```

这里为 graphics_buffer 分配了 4004 字节的内存单元，比该类型的长度多 4 字节。这是因为预编译程序要返回 2 字节(即 short 类型)长度和系统所要求的字边界调整

§2.4 SQL 变量的说明和引用

从数据结构角度来分，SQL 变量有简单变量，数组和指针。从功能来分，又有输入、输出和指示器变量。而输入和输出变量又叫宿主变量。

在 SELECT 或 FETCH 语句的 INTO 子句中所引用的变量叫输出 SQL 变量，因为 ORACLE 把从数据库表列中选择的值存放到这类变量中。而在其它 SQL 语句(INSERT,UPDATE.....)中所包含的变量都叫输入 SQL 变量，ORACLE 把输入 SQL 变量中的数据输入给 ORACLE 存入数据库表列中，或与数据库表列中的数据进行比较。输入 SQL 变量也被用在 WHERE、HAVING 和 FOR 子句中。在 SQL 语句中，凡是允许出现值和表达式的地方，都允许出现输入 SQL 变量。

SQL 变量的名字可任意长，但只有开始 31 个字符有效。为了与 ANSI 和 ISO 标准兼容，要求名字以字母开始，且不包含连线和下划线字符，其长度必须≤18 个字符。SQL 变量的名字不允许与 ORACLE 保留字同名。

2.4.1 SQL 变量的说明和引用

(1)SQL 变量的说明

SQL 变量在引用之前必须先说明。所谓说明，就是在说明段，使用 C 的类型说明语句为每一个 SQL 变量指定一个 ORACLE 能支持的 C 数据类型。C 的类型必须与数据库列的数

据类型相兼容。在说明段能为 SQL 变量指定的数据类型如表 2—6 所示。

下面是 SQL 变量说明的例子：

```
EXEC SQL BEGIN DECLARE SECTION;
        int emp_number;
        char name[15];                          (1)
        float salary=2500.00;                   (2)
        int var1,var2,var3;                     (3)
        float comm[50],var4[30];                (4)
        char username[50][15];                  (5)
        char work_date[10]="25—FEB—94";         (6)
EXEC SQL END DECLARE SECTION;
```

对简单 C 数据类型只能说明一维数组(如(1)和(4))，不允许说明二维或更高维数的数组。ORACLE 把 username[50][15]视为一维字符串数组。

对标准 C 数据类型，能使用一个说明语句同时说明几个 SQL 变量，如语句(3)和(4)等价于

```
EXEC SQL BEGIN DECLARE SECTION;
        int var1;
        int var2;
        int var3;
        float comm[50];
        float var4[30];
EXEC SQL END DECLARE SECTION;
```

可在说明段初始化 SQL 变量，如(2)和(6)，但不允许初始化数组，例如下面的说明是无效的：

```
EXEC SQL BEGIN DECLARE SECTION;
        int dept_number[3]={10,20,30};
EXEC SQL END DECLARE SECTION;
```

PRO*C 允许在说明段使用下列存储类型说明符来定义 SQL 变量：

- auto
- extern
- static

但不允许使用 register 存储类说明符。

SQL 变量的存储类型决定了它被访问的场所和怎样被存储。下面的例子说明怎样在说明段编码存储类说明符。

```
main( )
{
    EXEC SQL BEGIN DECLARE SECTION;
        auto int m,n;           /* auto is the default */
        extern int sum;
```