北京科海培训中心

• Informix 数据库培训教材之三

ESQL/C 编程指南

方娟 编著

科学出版社

北京科海培训中心

• Informix 数据库培训系列教材之三

ESQL/C 编程指南

方 娟 编著

科学出版社
1999

内 容 简 介

本书是针对在 Informix 数据库系统下应用 C 程序设计语言进行数据库应用系统开发的人员而编写的教材。

全书共分 7 章，首先对 Informix-ESQL/C 的特点、环境及使用进行简单介绍，然后详细讲述如何在 C 语言中嵌入 SQL 语句。书中通过大量实例来介绍 ESQL/C 的数据类型，编写 ESQL/C 程序时应着重考虑的数据类型及其转换，并为程序设计人员在不同情况下灵活应用 ESQL/C 程序如何使用游标和动态 SQL 语句来提高编程的效率。附录为读者提供书中一些内容 C 说明以及几个实用程序。

本书可作为 Informix 数据库培训教材，也可作为数据库应用、开发人员的自学教材。

图书在版编目(CIP)数据

ESQL/C 编程指南/方娟编著。—北京：科学出版社，1999.6

Informix 数据库培训系列教材

ISBN 7-03-007675-3

Ⅰ.E… Ⅱ.方… Ⅲ.C 语言-程序设计 FIV.TP31

中国版本图书馆 CIP 数据核字(1999)第 23681 号

科学出版社出版

北京东黄城根北街 16 号

邮政编码：100717

北京市朝阳区科普印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

*

1999 年 6 月第 一 版 开本：16

1999 年 9 月第 2 次印刷 印张：16 3/8

印数：5 001—10000 字数：398 000

定价：24.00 元

(如有印装质量问题，我社负责调换(环伟))

前　言

当今社会是信息高速膨胀的时代，对于企业的管理者来说，拥有一个高效、稳定、可靠的数据库系统是获得竞争优势的重要手段。Informix 是最先推出对象关系型数据库系统的厂商，Informix 的产品采用了先进的动态可伸缩体系结构，并提供了有效的并行数据处理能力，对管理数据量越来越大、数据复杂程度日易增加的数据库的同时，还可以有效改善系统的性能。此外，该产品还具有极好的稳定性，在一些关键性业务领域如银行、邮电等系统中可以保证系统稳定可靠地不间断运行。

Informix 数据库建立在关系数据库的体系结构之上，使用关系数据库成熟的技术，可以保证了在性能、可靠性、稳定性方面的优势，其沿用的 SQL 查询语言也使得用户可以很快学会使用。INFORMIX-ESQL/C 是嵌入 SQL(EMBEDDED SQL)的简称，它为 C 程序员提供了一种快速而又具有弹性的符合国际标准的应用开发工具。将非过程化的 SQL 数据库语言嵌套于 C 语言中，这样产生出的 ESQL/C 既灵活又易于使用，用它来编写数据库应用软件不但可以大大节省程序编写的时间，提高软件开发的效率，更重要的是提高了开发应用程序的可靠性。

本书介绍了 INFORMIX-ESQL/C 的程序结构，深入浅出地讲解了如何在 C 语言程序中嵌入 SQL 语句，从而编写出包含有 INFORMIX-ESQL/C 语句的 C 程序。

全书从讨论 INFORMIX-ESQL/C 的使用环境开始，详细介绍了各种数据类型和动态 SQL 语句，书中的附录还列出调试过程中的错误信息，可以协助读者快速有效地开发和调试自己的应用程序。

Informix 数据库培训教材目前推出三本：《数据库系统管理指南》、《SQL 查询语言及应用》和本书。这套教材是我们根据近年来长期举办 Informix 数据库培训班的授课经验和实际应用的体验编撰而成。

本书在编写过程中得到了北京工业大学计算机学院宗大华教授的热心指导和帮助，北京大学计算机系史森老师帮助审阅了全部书稿，并提出了宝贵意见，在此一并表示感谢。

由于时间仓促，水平有限，书中难免有不足和错误之外，恳请广大读者批评指正！

作　者

1999 年 5 月北京

目　录

第1章 绪 论

1.1 INFORMIX-ESQL/C 简介

INFORMIX-ESQL/C 是嵌入式 SQL(EMBEDDED SQL)的简称,它为第三代程序语言 C,COBOL,ADA 等的应用开发人员提供了一种快速而又具有弹性的 SQL 界面,并且符合国际标准。用具有这种界面的语言来编写数据库应用软件不但可以大大节省时间,提高软件开发的效率,更重要的是提高了开发应用系统的可靠性。

1.1.1 ESQL/C 的特点

INFORMIX-ESQL/C 是一个应用开发工具,C 程序员可以通过它拥有一个访问 INFORMIX 的数据库接口。我们知道,C 语言具有支持复杂逻辑控制和存储器管理的功能,是一种非常灵活的开发工具,但用它来开发大型应用系统时,由于编写程序工作量很大,给维护带来诸多的不便。SQL 语言是一种非过程化的国际工业标准查询语言,优点是易学易用,对数据的管理和处理能力强,缺点是对复杂的计算适应性差。综合这两种语言的特点,将 C 语言与非过程化的 SQL 数据库语言有机地加以结合,即把 SQL 语句嵌套于 C 语言中,这样产生出的 ESQL/C 程序既灵活又易于使用,可达到节省编程篇幅、提高开发效率的目的。我们常把嵌入到 C 语言应用程序中的数据库查询语言称为 RDSQL 语言。

INFORMIX-ESQL/C 的开发人员使用头文件和预编译器直接在 C 程序中嵌入 SQL 语句,以便随时获得 SQL 的速度和方便性。用户通过调用 INFORMIX-ESQL/C 众多组织成库的例程,可以引入 SQL 的数据类型,解释各种状态信息。使用 INFORMIX 的子进程,可获得诸如高层数据库存取、数据操作命令以及 SQL 的优化查询工具等高效率的数据库服务。

归结起来,ESQL/C 有如下特点:

- 源程序代码少。在建立、修改数据库和表,建立索引和进行检索时,用户所使用的 RDSQL 程序代码很少。
- 命令功能强。SQL 中的许多命令能够使用户方便地增加、删除和修改数据。当用户想在 C 语言程序中处理某些 ESQL/C 的代码行时,只需将 $ 符号或 EXEC SQL 放在程序行开始处,其他工作即可由 ESQL/C 完成。
- 建索引速度快。使用时如果用户没有建立索引文件,ESQL/C 会自动为你建立索引,使查询工作更加简便易行。
- 查询灵活。在 ESQL/C 中,不仅能够事先确定嵌入所需要的各类查询,还可以使用动态 SQL 语句,随时构造和形成复杂条件的 SQL 操作,为数据库应用带来极大的便利。
- 自然语言的程序语句。使用 ESQL/C 编写应用程序类似于英语自然语言,不但使用便利,而且能执行复杂的语言功能。

1.1.2 ESQL/C 的环境变量

用户使用 INFORMIX-ESQL/C 之前，必须设置两个环境变量：

(1) 设置 INFORMIXDIR 环境变量，以便 INFORMIX-ESQL/C 装入其程序。

(2) 设置 PATH 环境变量，以便 shell 找到正确的 bin 目录来执行 INFORMIX-ESQL/C 程序。

用户可以通过两种方法设置环境变量。一种是在系统提示符下设置环境变量。使用这种方法时，用户每次登录进入系统必须重新设置环境变量。另一种方法是利用. profile 文件(B shell)或. login 文件(C shell)设置这些变量。在这种情况下用户每次登录时系统会通过执行文件自动设置环境变量。通常情况下，INFORMIXDIR 的驻留目录为/user/informix。

使用 B shell 时，可用如下语句设置环境变量：

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

使用 C shell 时，设置语句如下：

```
setenv INFORMIXDIR /user/informix
setenv PATH ${INFORMIXDIR}bin: ${PATH}
```

要注意的是，当用户运行 INFORMIX-ONLINE 时，还应当把环境变量 SQLEXEC 设置为 $INFORMIXDIR/lib/sqlturbo，把环境变量 TBCONFIG 设置为合适的值。另外，如果运行 INFORMIX-NET，INFORMIX-STAR 或 INFORMIX-ONLINE 时，还需要设置一些其他的环境变量。

1.2 在 ESQL/C 程序中诊断错误的方法

正常的数据库管理要求所有修改数据库的语句能连续地完成其功能。在 INFORMIX-ESQL/C 程序中执行一条 SQL 语句时，数据库服务器会返回几种信息，这些信息反映了 SQL 语句的执行情况，这些信息可以这样分类：

- 最近执行的 SQL 语句的完成状态。
- 有关性能的信息。
- 有关可能发生或已经发生的事情的警告。

这些返回信息传送到一个称为 sqlca 的结构中。sqlca 是一个数据结构，它是用户数据段的一部分，当使用 esql 来编译用户的源文件时，它会被自动包含进来。该结构在 sqlca. h 中定义：

```
struct sqlca_s{
    long sqlcode;
    char sqlerrm[72];
    char sqlerrp[8];
```

```
    long sqlerrd[6];
    struct sqlcaw_s{
        char sqlwarn0;
        char sqlwarn1;
        char sqlwarn2;
        char sqlwarn3;
        char sqlwarn4;
        char sqlwarn5;
        char sqlwarn6;
        char sqlwarn7;
    }sqlwarn;
}sqlca;
```

上述 sqlca 结构中的信息反映了 INFORMIX-ESQL/C 语句执行后的情况：成功或异常。其中异常情况包括三种：

(1) 执行成功，但没有找到记录；

(2) 执行成功，但出现警告信息；

(3) 执行结果失败，即发生了错误。

sqlca 结构中的 sqlcode 在最近的 SQL 语句执行完毕后，应当检测它的值，如果不是预定的值，则采取相应的措施。其返回值有三个，即=0，<0 和>0，含义分别如下：

- sqlcode=0 时，说明语句执行成功。
- sqlcode<0 时，说明语句执行后有错误发生，此时可查看有关错误代码所对应的错误信息，然后加以修改。
- sqlcode=1～99，该值依赖于动态 SQL 语句中 SQL 有关执行的设置。
- sqlcode=100，说明在执行 SELECT 或 FETCH 查找操作后返回零记录，在 sqlca.h 中将 SQLNOTFOUND 定义为 100。

sqlerrm[72]是错误消息参数，特定于错误信息。sqlerrp[8]现不用。sqlerrd 是具有 6 个元素的长整数的数组，该数组存放所有操作数据的属性和期望性能的描述。下面介绍每个长整数所代表的含义：

- sqlerrd[0]估计返回记录的数目，在动态 SQL 语句中使用。
- sqlerrd[1]代表 INSERT 语句执行后的 SERIAL 型值或产生的 C-ISAM 错误代码；ISAM 错误代码是更详细的解释。
- sqlerrd[2] RDSQL 语句处理的记录数目。
- sqlerrd[3] 估计的代价，在动态 SQL 语句中使用。
- sqlerrd[4] RDSQL 语句中的错误补偿。
- sqlerrd[5] INSERT 语句执行后最后一条记录的 ROWID 值。

sqlwarn 指明是否有警告信息，包含在 RDSQL 语句执行后标志的各种数据截断或其他警告信息的 8 个字符结构。如果没有发现问题，则这些字符为空。如果发现问题，则相应的 8 个字符被置为 W。如果 sqlwarn1～sqlwarn7 中有一个或多个已被置为 W，那么 sqlwarn0 也被置为 W。如果 sqlwarn0 为空，则说明 sqlwarn1～sqlwarn7 全都为空，就不必去检查这些警

告字符了。

sqlwarn1 如果一个或多个数据项被截断以适应字符型的宿主变量，则它被置为 W。通过检查对应的标识符变量，可以发现究竟是哪一项被截断。

sqlwarn2 在使用聚合函数(如 SUM，AVG，MAX，MIN 等)时，如果遇到空值 NULL，则其值被置为 W。

sqlwarn3 若 SELECT 子句选择列表中的项目数不同于 INTO 子句中的宿主变量数，则 sqlwarn3 被置成 W。

sqlwarn4 若 UPDATE 或 DELETE 语句中没有使用 WHERE 子句，则其值可使用 DESCRIBE 语句置成 W。UPDATE 或 DELETE 语句可以不带 WHERE 子句而作用于整个表，通过检查这个变量，可以避免对表的无意的全局修改。

sqlwarn5 程序执行的操作是 INFORMIX 扩展的，超出了标准的 ANSI SQL，故必须定义 DBANSIWARN 环境变量。

sqlwarn6 和 sqlwarn7 现不用。

当 sqlca.sqlcode 为负时，通过采取正确的操作，你能从失败的数据库操作中复原。通过检查 sqlca.sqlcode=SQLNOTFOUND，你可以在记录返回时书写处理检索结果的代码。

系统中由于没有自动异常检测机制，所以你必须检测 sqlcode 和 sqlwarn 的值。原则上，在每个 SQL 请求之后都应当检测 sqlcode 的值，这里我们可以使用 esql 指令 WHENEVER 语句来简化异常检测。使用 WHENEVER 的方法如下：

```
WHENEVER exception action
  exception NOT FOUND,SQLWARNING,or SQLERROR
    action {GO TO|GOTO} label,CALL func,STOP or CONTINUE
```

上述语法中，esql 预处理器按如下方式解释三种情况的异常(exception)：

```
NOT FOUND SQLCODE==100
SQLWARNING SQLCA. SQLWARN. SQLWARN0= ='W'
SQLERROR SQLCODE<0
```

WHENEVER 指令的作用域是 esql 处理的源文件，它对于该源文件来说是全局性的，但它的作用被限制在该源文件范围之内。在一个源文件的开头，有一个隐含的 WHENEVER 指令，用来指明如何处理三种情况中的每种异常，隐含的动作都是 CONTINUE。

下面通过一段程序说明 WHENEVER 语句的应用范围：

```
Insert Stock(){
    $ WHENEVER sqlerror GOTO SqlError;
    $ insert into stock…;
    exit(0);
    SqlError:
    …
}
UpdateStock{
    $ update stock…;
    exit(0);
    SqlError:
```

```
...
}
DeleteStock{      $WHENEVER sqlerror continue;
    $delete from stock...;
    ...
}
```

在本程序中 ESQL 语句前使用 $ 符号标识。

从上述函数中我们发现:Insert Stock()和 UpdateStock 函数都需要一条带标号 SqlError:的语句。这是因为,esql 施加的动作是当错误发生时转移到此标号位置。ZnsPrt Stock()函数中的 WHENEVER 语句一直作用到 DeleteStock()函数中的 WHENEVER 语句,即到 DeleteStock()函数的 WHENEVER 语句之前一直有效。由于 DeleteStock 函数中 WHENEVER 语句的动作是 continue,所以此函数中不需要有带标号的语句。

从本例可以看出,WHENEVER 语句所说明动作的有效性是从 esql 扫描到这点开始,直到源文件的结尾处,范围包括其间遇到的所有函数;或者直到后面的头一条处理同一个异常的 WHENEVER 语句。上例中第二个 WHENEVER 语句仍然处理 sqlerror 异常。

上述 WHENEVER 语句中的异常有三种状态:NOTFOUND,SQLWARNING 或 SQLERROR。上例中的异常即为 SQLERROR。下面分别介绍在何时返回以上三种情况。

NOTFOUND

自动异常检测 NOTFOUND 意味着没有找到满足 SQL 语句中条件的记录。前面提到过,这是由 sqlcode 的值为 SQLNOTFOUND 或 100 来指明。

数据库服务器返回 SQLNOTFOUND 有两种情况:

- 使用 SELECT 语句查询记录时,没有找到相应的数据。
- 执行 FETCH 语句时超出了多条记录集合的范围。在 ANSI 模式的数据库中使用 WHERE 子句时,不存在满足条件的记录。

例如:

```
UPDATE from student
WHERE student_num>100 and student_num<0
```

此时将返回 SQLNOTFOUND。

SQLWARNING

SQLWARNING 指的是"警告"异常,它说明此时程序发生了特定情况,这由 sqlca 结构 sqlwarn 域中 sqlwarn0 的值设为'W'来指明。

前面初步介绍了 sqlwarn 域中各个域的取值所代表的含义,表 1.1 说明了对于不同语句 sqlwarn 域的含义。

表 1.1 不同语句 sqlwarn 域的含义

语 句	sqlwarn 域	含 义
DATABASE	sqlwarn1	使用事务
	sqlwarn2	ANSI 模式兼容
	sqlwarn3	由 Online 服务器创建
INSERT	sqlwarn4	不兼容的浮点格式
DELETE	sqlwarn4	不兼容的浮点格式
UPDATE	sqlwarn4	不兼容的浮点格式
SELECT	sqlwarn1	数据被截断
	sqlwarn2	聚集操作产生 SQL 空值
	sqlwarn3	选择字段数目不等于宿主变量的数目
	sqlwarn4	不兼容的浮点格式
OPEN	sqlwarn4	不兼容的浮点格式
EXECUTE	sqlwarn4	不兼容的浮点格式
PREPARE	sqlwarn4	准备无 WHERE 子句的 UPDATE 或 DELETE 语句
ALL	sqlwarn5	设置了 DBANSIWARN 环境变量，并且执行了 INFORMIX 对 ANSI SQL 的扩展操作

假设我们有这样一条语句：

```
$ WHENEVER sqlwarning CALL sqlwarning;
```

则说明无论何时执行一条 SQL 语句发生警告异常，控制将传递给函数 sqlwarning()。在该函数中将使用变量监视最近执行的 SQL 语句，使其可以显示合适的警告信息。

SQLERROR

自动异常检测 SQLERROR 代表“错误”异常，这由 sqlcode 返回一个负值来指明。在以下几种情况下可能出现“错误”异常，即 SQL 请求失败：

硬件层次——控制器出错或磁盘坏扇区等。

内核部分——信号量不够(INFORMIX-ONLINE)或文件表溢出等。

存取方法——INFORMIX-SE 的 ISAM 或 INFORMIX-ONLINE 的 RSAM，如索引键重复，插入空值到非空字段等。

语法分析器——错误语法，未知对象或无效语句等等。

应用过程——用户或锁表(Lock Table)溢出(INFORMIX-ONLINE)等。

例如：

```
$ WHENEVER sqlerror goto sqlError;
  ...
sqlError:
   (void)printError("main");
   (void)Exitpgm(-1);
```

这是一个自动检测 SQLERROR 异常的例子。无论何时执行 SQL 语句发生错误，控制将传递给标号为 sqlError 的语句。当相应的错误消息显示之后，调用函数 Exitpgm()。Exitpgm 是结束事务，关闭数据库并退出程序。另外，如果错误发生在 Exitpgm()自身，则控制

传递给本程序块中标号为“sqlError:”的语句，显示错误消息，程序以状态-1退出。

本章主要讲述了INFORMIX-ESQL/C的一些预备知识，以了解有关ESQL/C的特点及其环境，并且介绍了异常检测的有关知识，这些都是编程的基础，对以后的学习有一定的帮助。

1.3 应用示例

通过学习上节的内容，我们已经掌握了几种异常检测的情况，下面通过一个实例来说明异常检测的应用。

我们给出module 1和module 2两个程序块：

```
module 1
    Insert_Book( ){
    EXEC SQL WHENEVERsqlerror goto sqlerror;
    EXEC SQL insert into book…;
    exit(0);
    sqlError:
    …
  }
  Delete_Book( ){
  …
  Book_next( );
  EXEC SQL delete from Book…;
  …
  sqlError:
  …
  }
module 2
  Query_Book( ){
    EXEC SQL whenever NOTFOUND goto Noit;
    EXEC SQL whenever sqlerror call Doit;
    EXEC SQL fetch first BookRW…;
    return;
    Noit:
    …
    }
Book_next{
    EXEC SQL fetch next BookRW…;
    return;
    Noit:
    …
    }
```

读懂上述模块并回答以下两个问题：

(1) 删除操作发生错误时,Delete_Book( )中会发生什么情况?

(2) fetch 操作发生错误时,Delete_Book( )调用 Book_next()之后会发生什么情况?

现在我们分析以上两个模块程序。在 module1 中,编译时 esql 通过顺序扫描代码,能够在每个 SQL 语句执行后立即自动插入错误陷阱代码。实际上有这样的条件存在,即 if (sqlca. sqlcode<0) goto sqlError; 因此,在 delete 操作发生错误时,由于存在错误陷阱代码,控制将转移到标号 sqlError 语句并向下执行。

在 module 2 中,根据我们以前学过的知识,如某条语句发生错误,则有 if(sqlca. sqlcode <0) call Doit,即 sqlcode 返回负值时,执行的动作是 call Doit。因此在运行时,若 fetch 操作发生错误,由于错误陷阱代码存在,控制暂时转向 Doit()函数,然后返回 Book_next()。然后,控制返回到 Delete_Book()函数而不是执行标号为 Noit 的函数。这部分代码仅当 fetch 语句没有发现错误时才执行,这是由 module 2 中的第一个 whenever 语句决定的。

以上是对此例的分析,通过本节的学习,我们对异常检测部分有了更进一步的了解。

第 2 章　ESQL/C 程序的编译和链接

2.1　概　　述

本节首先介绍 INFORMIX-ESQL/C 的程序结构，了解如何在 C 语言程序中嵌入 SQL 语句，从而编写出包含有 INFORMIX-ESQL/C 语句的 C 程序。

首先我们应当掌握如何将 SQL 语句嵌入 C 程序中。任何可以交互式输入的 SQL 语句，例如数据定义语句、数据检索语句、数据控制语句等，都可遵循以下规则嵌入 C 程序。

(1) 可以使用两种方法指明 SQL 语句：一种是在 SQL 语句之前用美元符号($)作为前缀，另一种是在语句之前使用 ANSI 标准的 EXEC SQL 关键字来标识。这两种方法可以在 C 程序中混合使用，但从维护方便考虑，通常只使用其中一种方式。

· 使用美元符号($)标识，例如：

```
#include <stdio.h>
 main(argc,argv)
   int argc;
   char  * argc;
   {
     $database book;
     $delete from book where book_num=100;
     ...
   }
```

· 使用 ANSI 标准的 EXEC SQL 关键字标识，例如：

```
#include <stdio.h>
main(argc,argv)
  int argc;
  char  * argc;
  {
    EXEC SQL database book;
    EXEC SQL delete from book where book_num=100;
    ...
  }
```

(2) 与 C 程序相同，在每条 SQL 语句的最后以分号结束。

(3) SQL 语句可以加在 C 程序语句的任何地方。

(4) SQL 语句中可以有变量，变量可以出现在交互式 SQL 语句中常数可以出现的任何位置。可执行 SQL 语句中的这种变量称为宿主变量。

2.2　宿主变量

宿主变量(host variable)，即用于 SQL 语句的 C 语言的普通变量，INFORMIX-ESQL/C

程序通过宿主变量在C语句和SQL语句中传递数据。例如，使用UPDATE语句将数据从数据库中查找出来，放到宿主变量中，C语句就能对该数据进行处理。由于宿主变量可以出现在嵌入的SQL语句中任何常量出现的地方，故在SQL语句中使用宿主变量时，必须把它们同SQL语句中的名称区分开，这样才能使编译时识别出哪些标识符是宿主变量。通常情况下，INFORMIX-ESQL/C规定在宿主变量前必须加美元符号($)。也可以使用ANSI标准中规定的冒号(:)来做前缀。除了变量说明必须以美元符号($)为前缀之外，宿主变量均被说明为原始的C变量。

请看下面的例子。

(1) UPDATE语句中出现宿主变量

```
$update book set book_price= $NewPrice;
```

而ANSI标准表示为：

```
$update book set book_price= :NewPrice;
```

(2) INSERT语句中出现宿主变量

```
$insert into book values(1, "语言学", $Newprice,"10/10/1982");
```

(3) 在UPDATE、DELETE语句中的where子句中出现宿主变量

```
$update book set book_price=50 where book_num= $number;
$delete from book where book_num= $num;
```

实际上，宿主变量就是普通的C变量，因此使用时应把它们说明和定义成某种明确的数据类型，这些数据类型包括：

- 简单类型(如整型、字符型、double、float等)
- 结构
- 数组
- 指针

与其他C变量相同，可在定义宿主变量时对它们进行初始化。为了使ESQL/C编译时知道宿主变量的存在和类型，应为预编译器标明宿主变量的定义。这里有两种方法，一种是把美元符号($)加在定义语句前面，另一种方法是使用ANSI标准的方式，即使用一对语句：

```
EXEC SQL BEGIN DECLARE SECTION
EXEC SQL END DECLARE SECTION
```

其中间部分为数据变量定义语句。下面是使用这两种方法的例子。

(1) 定义简单类型的宿主变量

```
$int number;
$double shju;
```

(2) 定义结构类型的宿主变量

```
$struct student{
    int no;
```