# 面向对象数据库的C宿主语言接口实现技术*

周立柱 王健斐

(清华大学计算机科学与技术系,北京 100084)

**摘要** 面向对象的数据库管理系统(OODBMS)通常都具备自身的查询语言,它以交互的方式供用户进行数据定义与数据操纵.除此之外,它还必须为程序员提供可在C、FORTRAN、PASCAL等高级语言里嵌入使用查询语言的宿主语言接口.本文首先介绍这样一种C宿主语言接口C—OSDL,而后重点讨论它的实现技术,尤其是将C—OSDL命令转换成SQL语句的翻译算法以及使用SQL实现支持导航式查找的指针层次管理方法.文中讨论的C—OSDL实现技术以及最初的接口设计是面向对象的数据库技术的深入探讨.

**关键词** 面向对象数据库,查询语言,宿主语言接口,预编译程序.

新兴的应用领域如办公自动化、生产制造自动化、工程设计等,需要新一代数据库管理系统(DBMS)的支持.这些新的DBMS都要具备面向对象的特征.[1,2]面向对象的语义关联数据模型OSAM*及其查询语言OSDL就是为适应这种新兴应用领域的DBMS而研制的.[3]OSAM*数据模型将世界看成是由对象类(object class)及其之间的语义关联(semantic association)组成,对它的概括描述如下:

1. 对象类:划分为实体类(entity class)与域类(domain class).

(a) 实体类:表示现实世界里存在的可单独识别的相同事物的集合.

(b) 域类:表示实体类属性的取值范围.域类可以是整数,字符等这样的原子域,也可以是象由城市、街道、门牌号组成的地址这样的复合域.

2. 语义关联:用以表示两个实体类之间的语义关系,共有G关联、A关联、I关联及C关联四种.

(a) G关联:概括关联(Generalization),说明两个实体类之间的超类与子类关系.

(b) A关联:聚合关联(Aggregation),表示某一对象类是另一对象类的组成部分.

(c) I 关联:交互关联(Interaction),它相当于E—R模型里的relationship.

(d) C关联:组成关联(Composite),主要用于统计.它把某一实体类的全部当成一个对象来处理.

* 本文1991年6月9日收到,1991年9月12日定稿

作者周立柱,47岁,副教授,主要研究领域为数据库,软件工程.王健斐,30岁,工程师,1991年硕士毕业于清华大学,主要研究领域为数据库,软件工程.

本文通讯联系人:周立柱,北京 100084,清华大学计算机科学与技术系

OSAM * 数据库可以用语义图来表示. 例如,图 1 就表示了一个生产制造系统数据库. 图中方框代表实体类,圆圈表示域类. 语义关联由字母及带有箭头的连线表示. 为了方便起见,我们把箭头离开的对象类称之为"关联源",而把箭头到达的对象类称之为"关联目标".

OSAM * 的查询语言 OSDL 提供了查找、删除、插入、更新等语句. 它的查询语句具有以下形式:

```
RETRIEVE        属性 1,…,属性 n
CONTEXT         关联表达式
[VIEWPOINT]     视点
```

其中的任选项 VIEWPOINT 用来控制查询结果是以第一范式还是以嵌套表格的形式在屏幕上显示.

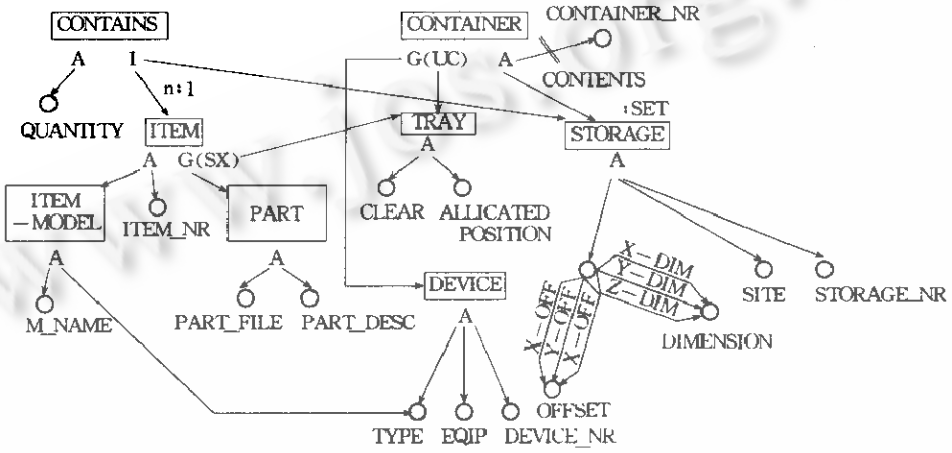


图1 语义图表示的数据库模式

以 OSAM * 及 OSDL 作为参考,我们正在国家 CIMS 实验工程上实现数据库管理系统 CIMBASE.[4,5]

以 OSAM * 模型为基础实现 DBMS 时,我们不仅为终端用户提供了交互式 OSDL 查询语言,而且还专门设计了可供程序员在 C 里使用 OSDL 命令,从而完成对 OSAM * 数据库操纵的用户编程接口——C—OSDL.[5]通过 C—OSDL,程序员可以在 C 语言里嵌入查询、插入、删除、更新等 OSDL 语句,开发应用系统.

C—OSDL 实质上是一个 C 语言的预编译程序. 它对嵌有 OSDL 语句的 C 源程序扫描,进行语法检查,而后再把它们翻译成实现 OSDL 功能的基础 DBMS,例如 ORACLE、INGRES 或其他 DBMS 的数据操纵命令. 这样预处理后的程序再经过 C 编译程序编译后运行,就可以完成对 OSAM * 数据库的各种操作.

实现这样的 C 预编译程序,要解决一系列技术问题,其中最主要的有:

1. 如何将 OSDL 翻译成基础 DBMS 的数据操作命令;

2. 如何实现为解决 OSDL 与 C 之间的失配问题而引入的指针层次机制;

3. 如何解决 OSAM * 模型的向量、矩阵、集合等复杂数据类型在 C 里的使用.

在这篇论文里我们将着重介绍为解决前两个问题而使用的技术. 第三个问题我们将另有文章讨论. 在本文以下各节里,我们将按下列顺序叙述:第一节给出一个嵌入 OSDL 语句

的C程序实例;第二节介绍C—OSDL的结构及工作原理;第三节介绍OSDL语言命令向关系数据库SQL命令的翻译算法;第四节介绍指针层次管理及使用指针的数据存取命令的实现方法.

## 1　C—OSDL程序实例

假定图1所示的数据库里,盛物器CONTAINER与仓位STORAGE之间的A关联表示的是盛物器与它所运载的物件所在仓位之间的关联关系,并且,我们想知道CONTAINER的子类DEVICE中,每一辆小车(type="cart"的DEVICE)的设备号DEVICE—NR及它所运载的所有物件的仓位的信息.这一要求可以用下面的C—OSDL程序来实现.

```
1  /* An example of C—OSDL program. First      */
2  /* indicate the Database to be used.        */
3    OSDL DEFINEDB  'gp1/cambase';
4  /* Define VARs to be used in OSDL statements.      */
5    OSDL define section begin
6          char dev_type[10];
7          int dev_nr,str_nr;
8    OSDL define section end;
9  /* Define communication area with the database,  */
10 /* sqlca is a C structure.                        */
11   osd1 include sqlca;
12
13 main()
14   {
15     strcpy(dev_type,"cart");
16     /* Open database first.       */
17     OSDL connectdb;
18     /* Define a cursor hierarchy and its query.       */
19     OSDL DECLARE RESULT c0 /* c0 is the root. */
20           FROM RETRIEVE device_nr, storage_nr
21                 CONTEXT DEVICE[type= :dev_type]* STORAGE
22                 VIEWPOINT DEVICE;
23     OSDL DECLARE CURSOR c1 /* c1 is a child of c0.  */
24           FOR STORAGE
25           WITHIN c0;
26     /* Open the root cursor to get ready for operation.  */
27     OSDL OPEN c0;
28
29     for (;sqlca.sqlcode != 4;)
30        /* Repeatetaly get a DEVICE instance until no */
31        /* more is available.                         */
32        {
33         /* Use c0 to get DEVICE instance.  */
34         OSDL FETCH c0
35               ATTRIBUTE device_nr
36               INTO :dev_nr;
37         if (sqlca.sqlcode<0) /* an error occurs  */
38           goto errexit;
39
40         for (;sqlca.sqlcode != 4;)
41             /* Repeatetaly get a STORAGE associated  */
```

```
42              /* with DEVICE instance until no more is */
43              {                          /* available. */
44                OSDL FETCH c1
45                      ATTRIBUTE STORAGE_NR;
46                      INTO :str_nr;
47                if (sqlca.sqlcode<0) goto errexit;
48
49                /* Display the required information. */
50                printf("%d %d\n",dev_nr,str_no);
51              }
52          }
53      OSDL CLOSE c0;
54      return(0);
55      errexit:
56              OSDL CLOSE c0;
57              printf("An OSDL error occurs!");
58              return(-1);
59  }
```

在上面的程序中,凡是以 OSDL 开头的语句都是由 C—OSDL 提供的对 OSAM * 数据库的操作或必要的变量及指针说明.程序中所加注释说明了它们的作用.这里,强调说明以下几点:

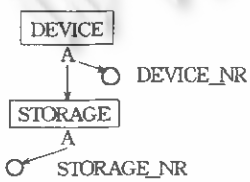

图2

首先,第 20 行到第 22 行的 RETRIEVE 语句定义了我们所要的查找对象及其结果.这一查询结果可以用图 2 所示结果模式来表示,结果模式里 DEVICE 与 STORAGE 之间的 A 关联是 DEVICE 由它的超类 CONTAINER 那里继承而来.

第二,19 行把 20 行到 22 行查询结果的视点 DEVICE 指定为根指针,第 23 行到第 25 行,把 STORAGE 指定为子指针,从而形成了由 C0,C1 构成的指针层次,为对查询结果进行数据导航作好准备.

第三,29 行与 40 行形成了两层循环,分别控制对 DEVICE 实例,以及与之关联的 STORAGE 实例的逐个访问.在这两重循环里的 FETCH 语句将指针 C0 与 C1 指向的实例的属性分别取入宿主变量 dev_nr 与 str_nr 之中.C—OSDL 语句中的宿主变量用":"加以标识.

## 2 C—OSDL 预编译器的结构及工作原理

图 3 给出了 C—OSDL 的主要结构及工作流程.图中的"词法分析器"对源程序扫描并记录下嵌入的所有 OSDL 语句的位置."语法分析器"根据字典对识别出的 OSDL 语句进行分析并产生出一系列信息表格.C—OSDL 的中心部分是"翻译器",它把嵌入的 OSDL 语句翻译成实现 OSDL 功能的关系数据库上的操作,这些操作以 C 的函数的形式存放起来,最后由"结果安装器"对原来的 OSDL 语句进行替换,得到最终的 C 代码文件.

在 C—OSDL 所处理的数据操纵语句中,插入、删除、更新相对而言较为容易.较难处理的是以 RETRIEVE 语句为中心而定义的指针以及利用指针进行数据访问的 FETCH 语句.C—OSDL 解决这一难点的基本思想要点如下:

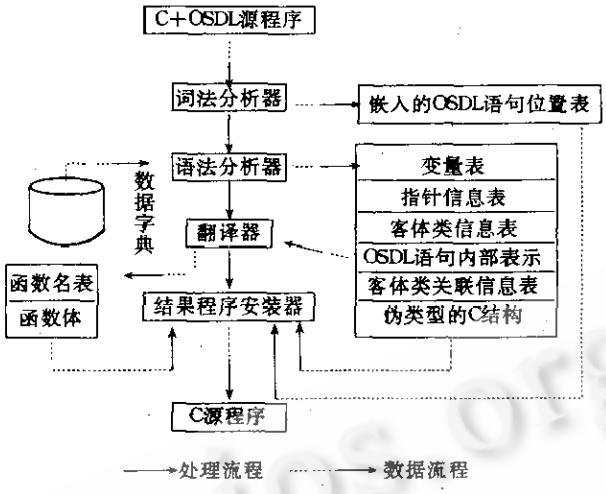


图3 C—OSDL预编译程序结构及工作流程

1. 根据 RETRIEVE 语句里 CONTEXT 子句给出的关联表达式,生成查询结果中所有对象(用对象标识 OID 表示)及其之间关联的关系数据库表格——结果关联表.

2. 根据定义的指针层次,利用关系表格上的指针操作,[6]在结果关联表上实现数据导航,获得所需查找对象的 OID 值.

3. 根据 OID 值以及存放实体类的关系表格,得到相应对象的属性值.

下面我们讨论 RETRIEVE 语句里关联表达式的翻译及指针管理.

## 3 翻译算法

RETRIEVE 语句里关联表达式的翻译算法与如何用关系表格来存储 OSAM * 数据库密切相关. OSAM * 数据库在关系数据库里以下述方法存储:

1. 若 C 是 OSAM * 数据库里的实体类,$A_1$,$A_2$,…,$A_n$ 是与 C 有 A 关联的域类(即它们是 C 的属性),则为 C 生成一个对应的关系表格 C(OID,$A_1$,$A_2$,…,$A_n$). 这里的 OID 属性用来存放 C 的实例的 OID 值.

2. 若 C 是某一 A 关联的源,而 D 是同一 A 关联的关联目标,则在用上述方法为 D 生成的关系表格里增加一个与 C 同名的属性,该属性的值域是 C 的实例的 OID 值. 若 c 是关系表格 C 的一个元组,d 是关系表格 D 的一个元组,则元组 d 所代表的对象是元组 c 所代表的对象的 A 关联目标的充要条件为 c. OID=d. C.

3. 若实体类 D 是实体类 C 的一个子类,c 与 d 分别是为它们所生成的同名关系表格 C,D 里的元组,则元组 c 所代表的对象与元组 d 所代表的对象同一(identical)的充要条件是 c. OID=d. OID.

4. 若实体类 E 的 I 关联同时连接到实体类 C 与 D 上(例如,图 1 中的 CONTAINS 的 I 关联同时连接了 ITEM 与 STORAGE),则在为 E 生成的同名关系表格中增加一个以 C 命名的属性与一个以 D 命名的属性. 属性 C 的值域为实体类 C 的实例的 OID 值,D 的值域为

实体类D的实例的OID值.若c、d、e分别为关系表格C、D、E的元组,则e所代表的对象既与c有I关联,又与d有I关联的充要条件是:

$$e.C=c.OID \text{ 并且 } e.D=d.OID$$

5.C关联的实现较为简单,不是翻译算法的重点,略去.

根据以上方法,为图1的OSAM*数据库生成的部分关系表格为:

```
ITEM_MODEL(OID,M_NAME,TYPE,ITEM)
ITEM(OID,ITEM_NR)
PART(OID,PART_FILE,PART_DESC)
CONTAINS(OID,ITEM,STORAGE,QUANTITY).
```

RETRIEVE语句中的CONTEXT子句以关联表达式的形式表示了查询结果中的对象及其之间语义联系.关联表达式由关联运算"*",非关联运算"!",以及更为复杂的格运算组成.关联"*"以及非关联"!"是二目运算,它们都要使用运算对象间存在的A关联、G关联、I关联或C关联关系,这在具体运算中由用户决定.

关联运算"*"与非关联运算"!"的实现是通过关系中的联结(Join)操作完成的,联结运算中用到的谓词公式与"*","!"具体使用哪一个关联有关,我们使用下面的函数根据两个实体类之间的关联给出必需的谓词公式:

```
Function  FORMP(C,D,ASSO)
输入:实体类C、D及它们之间的关联ASSO
输出:联结C、D所需的谓词公式
FORMP(C,D,ASSO)
  {
  IF  ASSO是G关联  THEN  RETURN "C.OID=D.OID";
  IF  (ASSO是A关联,并且C是关联源,D是关联目标) OR (ASSO是I关联,并且C是关联目标,D是关联源)
      THEN RETURN("C.OID=D.C");
  IF  (ASSO是A关联,并且C是关联目标,D是关联源) OR (ASSO是I关联,并且C是关联源,D是关联目标)
      THEN RETURN("C.D=D.OID");
  }
```

下面,我们给出关联表达式中基本运算的翻译算法,算法中的关系表格TMP是存放结果的结果关联表.

1.C*D:求实体类C与D互为关联的全部对象.

(a) CREATE TABLE TMP(C,D);

(b) 根据"*"所使用的关联ASSO调用函数FORMP(C,D,ASSO)得到谓词公式P;

(c)
```
INSERT  INTO  TMP
        SELECT  C.OID,D.OID
        FROM  C,D
        WHERE  P;
```

2.C! D:计算C与D里互不关联的全部对象.

(a) CREAT TABLE TMP(C,D);

(b) 根据"!"所使用的关联ASSO,调用函数FORMP(C,D,ASSO)得到谓词公式P;

(c)
```
INSERT  INTO  TMP(C)
      SELECT  OID
      FROM  C
      WHERE  NOT  EXISTS
            (SELECT *
             FROM  D
```

```
                WHERE  P);
```

(d)
```
INSERT  INTO  TMP(D)
    SELECT  OID
    FROM  D
    WHERE  NOT  EXISTS
        (SELECT *
        FROM  C
        WHERE  P);
```

3. C * and(D,E)

计算C、D、E里符合下述条件的对象c∈C, d∈D, e∈E:c与d有关联,并且c与e有关联.

(a) CREAT  TABLE  TMP(C,D,E);

(b) 确定C * D之间的"*"所使用的关联ASSO1,确定C * E之间的"*"所使用的关联ASSO2,生成C * D所使用的谓词公式Pcd=FORMP(C,D,ASSO1);生成C * E所使用的谓词公式Pce=FORMP(C,E,ASSO2);

(c)
```
INSERT  INTO  TMP
    SELECT  C.OID,D.OID,E.OID
    FROM  C,D,E
    WHERE  Pcd  AND  Pce;
```

4. C * or(D,E)

计算C、D、E里符合下述条件的全部对象c∈C,d∈D,e∈E:c与d有关联,或c与e有关联.

(a) CREAT  TABLE  TMP(C,D,E);

(b) 同翻译C * and(D,E)的步骤b);

(c)
```
INSERT  INTO  TMP(C,D)
    SELECT  C.OID,D.OID
    FROM  C,D
    WHERE  Pcd;
```

(d)
```
INSERT  INTO  TMP(C,E)
    SELECT  C.OID,E.OID
    FROM  C,E
    WHERE  Pce;
```

关联表达式所涉及的其他一些基本运算的翻译以及更详细的讨论,见参考资料[7].

## 4 指针管理及其操作

C-OSDL对于对象的访问是通过指针层次的定义及利用指针的FETCH语句来实现的.在指针层次中,双亲指针的移动带动子女指针的移动,并且由双亲指针所指向的对象与子女指针所指向的对象之间必然存在着语义关联.在指针层次中只有双亲指针对子女指针存在着上述制约关系,反之不成立.

上节介绍的翻译算法为RETRIEVE语句产生了结果关联表.结果关联表的元组由符合查询条件的对象OID组成.

C-OSDL所提供的利用指针访问数据库的操作具有下述形式:

```
FETCH 指针C ATTRIBUTE 属性1,…,属性n
            INTO 变量1,…,变量n
```

这里的指针C是为RETRIEVE语句结果模式中的某一实体类E而定义的,属性1,…,属性n是E的属性.

C－OSDL的指针操作是转换成对关系表格上的指针操作来完成的.关系数据库中的关系指针采用下述语句定义:[8]

```
DECLARE 关系指针名 CURSOR FOR
        SELECT [DISTINCT]…FROM…WHERE…,
```

相应的使用指针获取元组数据的语句是

```
FETCH 关系指针名 INTO 变量1,…,变量n;
```

为了利用关系指针实现OSDL指针层次及其操作,C－OSDL在翻译阶段首先完成下述准备工作:

1. 在上节的算法所产生的结果关联表TMP上为每一个OSDL指针层次中的指针P定义一个同名的关系指针与一个系统变量SYS_CP.具体方法是:若CR是根指针,E是CR所指向的实体类,则对应的关系指针为:

```
DECLARE CR FOR
  SELECT DISTINCT E FROM TMP;
```

若CR不是根指针,E是CR指向的实体类,CP是CR的双亲指针,F是CP所指向的实体类,则对应的关系指针为:

```
DECLARE CR FOR
    SELECT DISTINCT E
  FROM TMP
  WHERE F=SYS_CP
```

与指针有关的系统变量SYS_CP将用于保存对指针CP的FETCH语句而得到的对象OID.由于C－OSDL指针操作中规定了首先执行对双亲指针的FETCH操作,因此WHERE子句的变量SYS_CP保存了双亲指针所指对象的OID,于是F=SYS_CP实现了双亲指针CP对子女指针CR的制约.

2. 为每一个C－OSDL指针P及它的FETCH操作所涉及的属性1,…,属性n,在指针所指实体类对应的关系表格T上定义关系指针CT:

```
DECLARE CT FOR
      SELECT 属性1,…,属性n
      FROM T
      WHERE OID=SYS_P;
```

注意,变量SYS_P里保存了在结果关联表TMP上得到的P所指向的对象OID.

以上面的准备工作为基础,C－OSDL的指针操作

```
FETCH 指针C ATTRIBUTE 属性1,…,属性n
            INTO 变量1,…,变量n
```

翻译如下:

1. 在结果关联表TMP上执行关系指针操作　　FETCH E INTO SYS_C;

这里的E是C所指的实体类名,它是TMP的一个属性,SYS_C为系统变量.

2. 找到实体类E所对应的关系表格T及其关系指针TP,执行在T上的下列指针操作:

```
FETCH TP INTO 变量1,…,变量n;
```

至此,变量1,…,变量n里存放了实体类E的属性1,…,属性n的属性值.

**结论**:本文讨论了面向对象数据库的C预编译程序的技术实现,特别是它的查询语句翻译,指针层次管理以及利用指针进行导航的操作翻译算法.C—OSDL已在SUN工作站、HP 9000/840以及VAX 3400上运行.初步测试表明,C—OSDL预编译程序从功能上达到了设计要求.目前,它的进一步的完善与改进正在进行之中.

## 参考文献


1 Atkinson M. The object—oriented system manifesto. ALTAIR Technical Report No. 30—89: GIP ALTAIR, LeChesnay, France: Sept. 1989.

2 Stonebraker M. Third—generation database system manifesto. Proceedings of the IFIP DS—4 OO Database. July 1990.

3 Su S. An object—oriented semantic association model (OSAM*). In A. I. in S. Kurama *etal*. eds Industrial Engineering and Manufacturing: Theoretical Issues and Applications, American Institute of Industrial Engineering, 1988.

4 张霞,郑怀远. CIMSERC—DDBMS全局数据模型OSAM*.第一届中国计算机集成制造系统(CIMS)学术会议,北京:1990.11.

5 周立柱,王小京,衣丰超.面向对象的语义关联数据模型查询语言在C语言中的嵌入.软件学报,1992;3(1):60—64.

6 Date C. An introduction to database system, Vol. 1, 4th Ed. Addison—Wesley Publish Company: 1986.

7 王健斐.C—OSDL预编译程序的设计和实现.清华大学计算机系硕士学位论文,1990.12.

8 PRO*C User's Guide. Oracle Corp., 1987.


# THE IMPLEMENTATION OF A C HOST LANGUAGE INTERFACE FOR AN OBJECT—ORIENTED DATABASE SYSTEM


Zhou Lizhu and Wang Jianfei

*(Department of Computer Science and Technology, Tsinghua University, Beijing 100084)*



**Abstract**　An Object Oriented Database Management System (OODBMS) provides a query language that allows end users to define and manipulate database interactively. Besides, it must have host language interfaces for embedding query statements in C, FORTRAN, PASCAL or other languages. This paper presents the implementation of such a host language interface — C—OSDL. In particular, the authors focus on the algorithms to translate C—OSDL commands into SQL statements and a method to manage cursor hierarchy for navigational retrieval of the database. The implementation and initial design of C—OSDL interface are an in—depth study on object oriented database technology.



**Key words**　Object—oriented database, query language, host language interface, pre—compiler.