DBMaster

DBMaster DCI 用户手册

SYSCOM

SYSCOM Computer Engineering Co./Corporate Headquarters

B1, 2-7F No. 115 Emei Street, Wanhua District,
Taipei City 108, Taiwan (R.O.C.)

www.dbmaker.com

www.dbmaker.com.tw/service

Document No.645049-237507/DBM54CN-M03312017-DCIU

发行日期: 2017-03-31

对于本手册中没有体现的关于产品最新功能的描述，请在安装完成SYSCOM DBMaster 软件后阅读 README.TXT 文件。

注册商标
SYSCOM，SYSCOM 图标和 DBMaster 是SYSCOM 公司的注册商标。
Microsoft, MS-DOS, Windows 和 Windows NT 是 Microsoft 公司的注册商标。
UNIX 是 The Open Group 的注册商标。
ANSI 是美国国家标准化组织的注册商标。

手册中提到的其他产品名称或许是它们各自持有者的注册商标，仅仅是为提供此信息。
SQL 是行业语言，并不为任何公司或任何组织所有。

注意事项
本手册中有关软件的描述，均以该软件所提供的使用许可为基础。

对于授权许可的详细信息，请与您的经销商联系。关于计算机产品的特殊用途的市场性与适用性，经销商不会给予任何说明和保证。因外界因素如地震、过热、过冷和潮湿而引起产品的任何损坏以及由于使用不正确的电压和不兼容的软硬件而引起的损失和损坏，经销商概不负责。

虽然该手册的内容已经过仔细核对，但错误再所难免。若手册再有改动，不另行通知。还请见谅。

目录

1 简介

本手册针对那些有意将COBOL程序的可靠性和关系数据库管理系统（RDBMS）的灵活性与高效性结合起来使用的软件开发人员，对如何使用DBMaster COBOL界面（DCI）提供了系统的说明。通过DCI，用户可以使用DBMaster数据库引擎更有效地进行管理并与COBOL进行数据整合。

DCI在COBOL程序和DBMaster之间提供了一个信道，并允许COBOL程序有效地访问存储在DBMaster数据库中的信息。COBOL程序通常使用标准的B-TREE文件存储数据，访问存储在B-TREE文件中的信息通常是标准的COBOL I/O语句，如READ、WRITE和REWRITE。

COBOL程序也可以访问存储在DBMaster RDBMS中的数据。传统上，COBOL程序员使用一种称为嵌入式SQL的技术将SQL语句嵌入到COBOL源代码中。在编译源代码之前，一种特殊的预编译程序将SQL语句转换为“calls”到数据库引擎，这些调用会在程序运行时执行以访问DBMaster RDBMS。

使用COBOL程序将信息存储在数据库中，尽管这是一个很好的解决方案，但是它仍然存在一些缺点。首先，它暗示着COBOL程序员必须对SQL语言有很好地了解；其次，用该方法书写的程序不是便携式的——它不能同时满足B-TREE文件和DBMaster RDBMS的需求。此外，SQL语法通常针对不同的数据库有着不同的变化，这就意味着COBOL程序嵌入的SQL语句是有特殊要求的，DBMaster RDBMS不可能与另一种数据库一起使用；最后，嵌入的SQL很难执行现有的程序。实际上，嵌入

SQL的应用需要大量的重新设计，包括大量增加工作存储、数据存储和改写每个I/O声明的逻辑。

有一种可替代的嵌入式SQL，一些供应商开发了从COBOL到数据库的无缝接口，联机时这些接口可将COBOL I/O命令转换成SQL语句。通过这种方法，COBOL程序员不需要熟悉SQL语句，并且COBOL程序依然具有便携性。但是这样以来，性能便成了主要问题。

实际上，SQL有着不同于COBOL I/O语句的设计意图。SQL是一种基于规则的特别查询语言，它可以从一般的规范中查询出任意的数据组合。相比之下，COBOL B-TREE 或其它数据架构的调用是通过明确的traversal key或导航逻辑或者二者并用直接访问数据。因此，去强迫事务繁多且性能敏感的COBOL应用程序完全通过基于SQL的I/O来执行通常是不恰当的。

Syscom的COBOL接口产品DCI，之所以不使用SQL就是这个原因。DCI提供直接的数据存储访问和遍历的方式，在某种意义上类似于COBOL自己访问用户可替代的COBOL文件系统。DCI 在COBOL程序和DBMaster文件系统之间提供了一个无缝接口。面对最终用户，信息在应用程序和数据库之间的转换是无形的。在需要完整的基于SQL文件或数据存储访问的情况下，如桌面决策支持系统（DSS）、数据仓库和4GL应用，DBMaster提供的这些功能都能确保RDBMS的可靠性和稳健性。

Syscom的数据库和DCI产品，在结合基于SQL的数据库存储和报告的灵活性，以及4GL功能和导航数据结构后，提供了惊人的性能。

1.1 其它相关文件

除了本手册之外，我们还为您提供了其它用户手册和参考文献，帮助您更深地了解DBMaster数据库管理系统。

- 有关DBMaster性能和功能的介绍，请参考*DBMaster指南*。
- 有关设计、管理和维护DBMaster数据库的信息，请参考*数据库管理员手册*。
- 有关如何管理DBMaster的信息，请参考*服务器管理工具户手册*。
- 有关DBMaster的配置信息，请参考*配置管理工具用户手册*。
- 有关DBMaster功能的信息，请参考*数据库管理工具用户手册*。
- 有关dmSQL命令行工具的使用方法，请参考*dmSQL使用手册*。
- 有关DBMaster SQL语言的语法和使用的相关信息，请参考*SQL命令与函数参考手册*。
- 有关嵌入式ESQL/C语言的语法和使用，请参考*ESQL/C程序员参考手册*。
- 有关ODBC API和JDBC API的信息，请参考*ODBC程序员参考手册*和*JDBC程序员参考手册*。
- 有关DBMaster的错误信息及警告信息，请参考*错误信息参考手册*。

1.2 技术支持

在软件试用期间，Syscom Computer Engineering Co.("Syscom")将为您提供30天的免费email支持和电话支持。当软件注册后，我们还会再为您提供30天的免费技术支持。如此一来，您就可以获得60天的免费支持。不仅如此，在您购买软件后，Syscom对任何问题都会以email的方式为您提供技术支持。

您除了可以获得免费的技术支持外，还可以以20%的零售价购买其它产品。要想获得更多的详细资料和价格信息，请与sales@dbmaker.com保持联系。

您可以通过任何一种方式（普通信件、电话或email）与Syscom技术支持保持联系，请登录至：www.casemaker.com/support 以获取详细信息。在与Syscom技术支持联系之前，请先查询当前数据库的常见问题解答。

无论您以何种方式与Syscom的技术支持联系时，请务必写上以下有效信息：

- 产品名称和版本号
- 注册号
- 注册的用户名和地址
- 供应商/发行者的地址
- 操作平台和计算机系统配置
- 错误发生前执行的动作
- 如果可以，请提供错误信息和编号
- 其它一些帮助信息

1.3 文档协定

为方便用户的阅读和使用，本手册使用了一种标准的排版约定，注释、程序、示例和命令行都用缩进排版的方式进行了特别的设置。

协定	说明
斜体字	斜体字表示必须输入的信息占位符，例如用户名和表名。此字符可用实际的名称来替换。有时，文档也会使用斜体字来介绍新的关键字，强调着重点。
黑体字	黑体字表示文件名、数据库名、表名称、字段名、用户名和其它数据库对象。它也用于强调程序执行步骤中的菜单命令。
关键字	文字段落中， SQL语言使用的关键字都是以大写字母出现的。
小符号	文档中出现的小写字符表示键盘上的按键，两个键名之间的加号（+）表示在按住第一个键不放的同时，再按第二个键。两个键名之间的逗号（，）表示释放第一个键以后，再按第二个键。
注意	包含一些重要的信息。
➲ **程序**	表示后面跟随的是程序的执行步骤或连续的项目。很多任务都是通过这种方式描述，给用户提供一个逻辑顺序步骤得以效仿。
➲ **示例**	例子用来阐明描述，通常包括屏幕上出现的文本，用户也可以将这些例子输入到计算机中，通过屏幕看到运行结果。当然，示例还包括一些原型和语法。
命令行	包括文本，这些命令都可以输入计算机中，显示在屏幕上。通常用于显示SQL命令的输入输出或dmconfig.ini中的内容。

图1-1 文档协定表

2 DCI基础

本章对如何安装和配置DBMaster DCI环境提供了必要的信息。另外，通过提供使用演示程序的必要信息，向您展示DCI的基本功能。

本章将分别对以下内容做详细的介绍。

- 软件和硬件需求。
- 对UNIX和Windows平台做逐步的安装指导。
- 为DBMaster配置DCI选项。
- DCI演示程序指导。

2.1 DCI概述

尽管传统的COBOL文件系统和数据库都包含数据，但是它们有着显著的不同。数据库比传统的文件系统更稳定可靠，因此，它们作为高效的系统在软硬件发生故障时可进行数据恢复。此外，为了确保数据的完整性，DBMaster RDBMS对参照行为、定义域、字段和表约束方面都提供了支持。

文件系统和数据库

数据库中的数据存储和COBOL索引文件之间有很多相似之处。下表展示了各系统的不同数据结构，以及它们如何对应。

COBOL索引文件系统对象	数据库对象
目录	数据库
文件	表
记录	行
域（Field）	字段（Column）

图 2-1 COBOL和数据库对象结构

在COBOL中，索引文件的操作是执行在记录上，而在数据库中则执行在字段上。从逻辑上讲，COBOL索引文件象征着一个数据库表，COBOL文件的每条记录都代表着数据库中表的一行，而每一个域（field）则代表一个表字段。数据在COBOL中可以拥有多种定义类型，而数据库中的表字段则必须与某个特定的数据类型联系在一起，如integer、character或date。

➲ **示例**

使用以下格式定义COBOL记录：

```
terms-record.
        03      terms-code      PIC 999.
        03      terms-rate      PIC s9v999.
        03      terms-days      PIC 9(2).
        03      terms-descript  PIC x(15).
```

上例列出的COBOL记录在数据库中将如下图所示，请注意每一行如何代表COBOL 01级的一个实例。

terms_code	terms_rate	terms_days	terms_descript
234	1.500	10	net 10
235	1.750	10	net 10
245	2.000	30	net 30
255	1.500	15	net 15
236	2.125	10	net 10
237	2.500	10	net 10
256	2.000	15	net 15

图 2-2 COBOL 记录转换成数据库行

访问数据

ACUCOBOL-GT的普通文件句柄和DCI与Vision文件系统接口，Vision是ACUCOBOL-GT提供的标准索引文件系统。Vision文件的详细信息请参考ACUCOBOL-GT手册。

DCI与数据字典结合使用，在COBOL应用程序接口（API）和DBMaster数据库管理系统之间的数据访问起到了桥梁的作用，用户可以通过API访问数据。此外，复杂查询可以通过DBMaster的任一种SQL接口

（dmSQL和JDBA工具）来实现。数据字典由ACUCOBOL-GT编译器创建，有关数据字典的详细信息将在第3章*数据字典*中讨论。

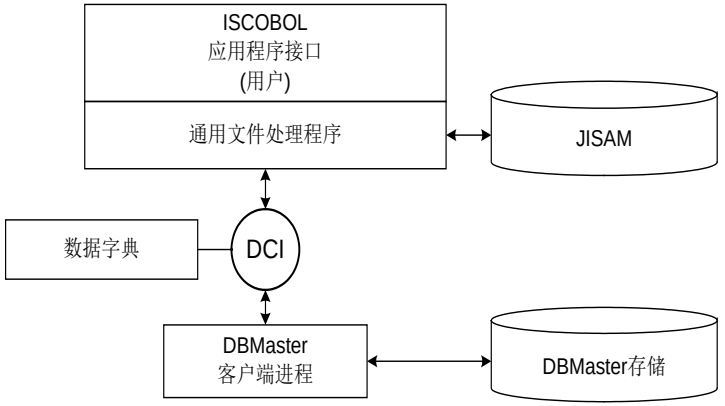


图 2-3 数据流程图

2.2 系统需求

DCI是DBMaster的一个附加模块，必须链接ACUCOBOL-GT运行系统。因此在安装DCI产品时必须安装C编译器，为了实现接口，必须使用ACUCOBOL-GT 4.3或更高的版本。

位于DCI目录的README.TXT文件列出了产品出货的文件。

DCI支持以下平台：

- Windows 32位和x86 64位（Windows2008/7/8/2012/10）
- Linux 32位（glibc2.3）和x86 64位（glibc2.7）
- Windows 32位和x86 64位（Windows2000/XP/2003/Vista）

为了实现DCI功能，您必须安装以下软件：

- DBMaster version 5.2或更高版本
- ACUCOBOL-GT 4.3或更高版本
- 本机必须安装C编译器（例如：Windows平台的Visual C++™ 6.0）

2.3 设置说明

在配置DCI之前，必须安装最新的DBMaster版本并进行配置，请参考DBMaster CD中的快速启动向导来进行DBMaster的安装。

Windows平台设置

在设置DCI之前，必须先将DCI文件从DCI zip文件的源目录（\DCI\OS\）中复制到目的目录。DCI库包括dmdcic.lib、针对ACUCOBOL-GT 5.1 的dmacu51.lib和之前的版本、针对ACUCOBOL-GT 5.2的dmacu52.lib和8.0之前的版本、针对ACUCOBOL-GT 8.0的dmacu80.lib和针对ACUCOBOL-GT 9.0的dmacu90.lib。

➲ **设置DCI**

1. 安装@DM_PRODUCT_NAME@。必须安装最新的@DM_PRODUCT_NAME@版本并配置DCI。
2. 将DCI库中的dmdcic.lib和ACUCOBOL-GT的DCI库从@DM_PRODUCT_NAME@ DCI压缩文件中复制到ACUCOBOL-GT安装目录中。

```
copy DCI\WIN32\dmdcic.lib c:\acucobol\acugt\lib
```

a) 针对ACUCOBOL-GT 5.1或之前的版本：

```
copy DCI\WIN32\dmacu51.lib c:\acucobol\acugt\lib
```

b) 针对ACUCOBOL-GT 5.2或8.0之前的版本：

```
copy DCI\WIN32\dmacu52.lib c:\acucobol\acugt\lib
```

c) 针对ACUCOBOL-GT 8.0：

```
copy DCI\WIN32\dmacu80.lib c:\acucobol\acugt\lib
```

d) 针对ACUCOBOL-GT 9.0：

```
copy DCI\WIN32\dmacu90.lib c:\acucobol\acugt\lib
```

3. 编辑ACUCOBOL-GT运行配置文件**filetbl.c**，该文件和ACUCOBOL-GT库所在同一目录下，例如：*c:\acucbl52\acugt\lib*。

a) 源**filetbl.c**文件包含如下条目：

```
#ifndef USE_VISION
#define USE_VISION  1
#endif
```

添加新的条目如下所示：

```
#ifndef USE_DCI
#define USE_DCI     1
#endif
```

b) 源**filetbl.c**文件包含如下条目：

```
extern DISPATCH_TBL v4_dispatch, ci_dispatch,
bt_dispatch;
```

添加新的条目如下所示：

```
#if USE_DCI
extern DISPATCH_TBL DBM_dispatch;
#endif  /* USE_DCI */
```

c) 源**filetbl.c**文件包含如下条目：

```
TABLE_ENTRY file_table[] = {
#if USE_VISION
  { &v4_dispatch,   "VISIO" },
#endif  /* USE_VISION */
```

添加新的条目如下所示：

```
#if USE_DCI
  { &DBM_dispatch,  "DCI" },
#endif  /* USE_DCI */
```

4. 编辑ACUCOBOL-GT运行配置文件**sub85.c**，该文件和ACUCOBOL-GT库所在同一目录下。

源**direct.c**文件包含如下条目：

```
struct  PROCTABLE WNEAR LIBTABLE[] = {
{ "SYSTEM",     call_system },
```

添加新的条目如下所示：

```
extern int  DCI_GETENV();
extern int  DCI_SETENV();
extern int  DCI_DISCONNECT();
extern int  DCI_GET_TABLE_NAME();
```

```
        extern int  DCI_SET_TABLE_CACHE();
        extern int  DCI_BLOB_ERROR();
        extern int  DCI_BLOB_PUT();
        extern int  DCI_BLOB_GET();
        extern int  DCI_GET_TABLE_SERIAL_VALUE();
        extern int  DCI_FREE_XFD();
        struct  PROCTABLE WNEAR LIBTABLE[] = {
        { "SYSTEM",         call_system },
        { "DCI_SETENV",     DCI_SETENV },
        { "DCI_GETENV",     DCI_GETENV },
        { "DCI_DISCONNECT", DCI_DISCONNECT },
        { "DCI_GET_TABLE_NAME", DCI_GET_TABLE_NAME },
        { "DCI_SET_TABLE_CACHE", DCI_SET_TABLE_CACHE },
        { "DCI_BLOB_ERROR", DCI_BLOB_ERROR },
        { "DCI_BLOB_PUT", DCI_BLOB_PUT },
        { "DCI_BLOB_GET", DCI_BLOB_GET },
        { "DCI_GET_TABLE_SERIAL_VALUE",
DCI_GET_TABLE_SERIAL_VALUE },
        { "DCI_FREE_XFD", DCI_FREE_XFD },
        { NULL,             NULL }
          };
```

5. 编辑ACUCOBOL-GT运行配置文件**direct.c**，该文件与ACUCOBOL-GT库所在同一个目录。

源direct.c文件包含如下条目：

```
        struct  EXTRNTABLE EXTDATA[] = {
        { NULL,         NULL }
        };
```

添加新的条目如下所示：

```
        extern char *dci_where_constraint;
        struct  EXTRNTABLE EXTDATA[] = {
        { "DCI-WHERE-CONSTRAINT",   (char *)
&dci_where_constraint },
        { NULL,         NULL }
        };
```

6. 如果您使用的ACUCOBOL-GT为6.0之前的版本，请打开wrun32.mak文件并搜索LIBS。wrun32.mak文件和ACUCOBOL-GT库位于同一个目录下，例如：c:\acucobol\acugt\lib。

   a) 如果使用ACUCOBOL-GT 5.1或之前的版本，请添加文件：dmacu51.lib、dmdcic.lib和dmapi52.lib。编译工程以获取最新的wrun32.exe和wrun32.dll文件：

   ```
   nmake.exe -f wrun32.mak wrun32.exe.
   ```

   b) 如果使用ACUCOBOL-GT 5.2或之后的版本，请添加文件：dmacu52.lib、dmdcic.lib和dmapi52.lib。编译工程以获取最新的wrun32.exe和wrun32.dll文件：

   ```
   nmake.exe -f wrun32.mak wrun32.exe.
   ```

7. 如果使用ACUCOBOL-GT 6.0或6.1，可打开ACUCOBOL-GT安装目录位于lib下名为wrun32.dsw的工程VS6.0，并添加文件dmacu52.lib、dmdcic.lib和dmapi52.lib，编译该工程以获得最新的wrun32.dll文件。

8. 如果使用ACUCOBOL-GT 6.2或7.0，可打开ACUCOBOL-GT安装目录位于lib下名为wrundll.vcproj的工程VS2003，并添加文件dmacu52.lib、dmdcic.lib和dmapi52.lib，选择使用MFC共同的DLL。编译该工程以获得新的wrun32.dll文件。

9. 如果使用ACUCOBOL-GT 8.0，可编辑ACUCOBOL-GT安装目录位于lib下名为wrundll.vcproj的VS2005工程。

   a) 将UseOfMFC 从“0”更改至“2”（例如：使用共享DLL中的MFC）

   b) 添加dmacu80.lib dmdcic.lib dmapi52.lib 到AdditionalDependencies，例如：

   ```
   AdditionalDependencies="rpcrt4.lib wcvt32.lib wfsi32.lib
   wrunlib.lib dmacu80.lib dmdcic.lib dmapi52.lib"
   ```

   c) 使用VS2005来编译工程并获得最新的new wrun32.dll文件。

   注意 *在编译64位的ACUCOBOL-GT 8.0运行环境时，您必须安装VS2005 x64组件。*

注意 *不能用VS2005 expres版本来编译运行环境，因为它没有MFC库。*

注意 *建议用户使用-Fx3或-Fx4编译选项来产生XFD文件，使其和之前的ACUCOBOL-GT版本采用相同的格式，因为ACUCOBOL-GT 8.0编译器不支持一些XFD语法，如XML格式的“$XFD COMMENT directive”，但是可以通过新的XML格式将DCI_XML_XFD 1添加至DCI配置文件中。*

10. 如果使用ACUCOBOL-GT 9.0，可编辑ACUCOBOL-GT安装目录位于lib下名为wrundll.vcproj的工程VS2008。

a) 将UseOfMFC 从“0”更改至“2”。（例如：使用共享DLL中的MFC）

b) 添加dmacu90.lib dmdcic.lib dmapi52.lib到AdditionalDependencies，例如：

```
AdditionalDependencies="mpr.lib rpcrt4.lib wcvt32.lib
wfsi32.lib wrunlib.lib dmacu90.lib dmdcic.lib dmapi52.lib"
```

c) 使用VS2008编译工程以获得新的wrun32.dll文件。

注意 *在编译64位ACUCOBOL-GT 9.0的运行环境时，必须安装VS2008 x64组件。*

注意 *不能用VS2008 expres版本来编译运行环境，因为它没有MFC库。*

注意 *建议用户使用-Fx3或-Fx4编译选项来产生XFD文件，使其和之前的ACUCOBOL-GT 版本采用相同的格式，因为ACUCOBOL-GT 9.0编译器不支持一些XFD语法，如XML格式的“$XFD COMMENT directive”，但是可以通过新的XML格式将DCI_XML_XFD 1添加至DCI配置文件中。*

11. 复制新的wrun32.exe和wrun32.dll文件到你的执行路径指定的目录中（如果使用ACUCOBOL-GT 6.2或以上版本，仅复制wrun32.dll文件）。

```
copy wrun32.exe c:\acucobol\acugt\bin
```

```
copy wrun32.dll c:\acucobol\acugt\bin
```

12. 为@DM_PRODUCT_NAME@ installed\bin目录设置PATH系统变量，例如：

```
set PATH=c:\@DM_PRODUCT_NAME@\5.2\bin:%PATH%
```

您也可以只复制c:\@DM_PRODUCT_NAME@\5.2\bin\dmapi52.dll到wrun32.exe和wrun32.dll文件所在的位置，如果您已经安装了最新的@DM_PRODUCT_NAME@补丁，请至那个目录更新dmapi52.dll。

13. 验证输入的链接：

```
wrun32 -vv
```

这将返回与您的运行系统链接的所有产品的版本信息，请确保它显示的是@DM_PRODUCT_NAME@接口版本。

UNIX平台设置

在设置DCI之前，必须先将DCI文件从DCI zip文件的源目录（\DCI\OS\）中复制到目的目录。DCI目录中包括libdmdcic.a，针对ACUCOBOL-GT 5.1的libdmacu51.a和之前的版本，针对ACUCOBOL-GT 5.2的libdmacu52.a和8.0之前的版本，针对ACUCOBOL-GT 8.0的dmacu80.lib和针对ACUCOBOL-GT 9.0的dmacu90.lib。

1. 将DCI库中的libdmdcic.a和ACUCOBOL-GT的DCI库复制到ACUCOBOL-GT安装目录。

例如，获得Linux平台的DCI库：

```
cp dci/Linux2.x86/libdmdcic.a /usr/acucobol/lib
cp dci/Linux2.x86/libdmapic.a /usr/acucobol/lib
```

通过ACUCOBOL-GT 5.1或之前的版本链接DCI库：

```
cp dci/Linux2.x86/libdmacu51.a /usr/acucobol/lib
```

通过ACUCOBOL-GT 5.2或8.0之前的版本链接DCI库：

```
cp dci/Linux2.x86/libdmacu52.a /usr/acucobol/lib
```

通过ACUCOBOL-GT 8.0链接DCI库：

```
cp dci/Linux2.x86/libdmacu80.a /usr/acucobol/lib
```

通过ACUCOBOL-GT 9.0链接DCI库：

```
cp dci/Linux2.x86/libdmacu90.a /usr/acucobol/lib
```

2. 编辑ACUCOBOL运行配置文件**filetbl.c**，该文件与ACUCOBOL-GT库所在同一个目录。

   a) 源**filetbl.c**文件包含如下条目：

```
#ifndef USE_VISION
#define USE_VISION  1
#endif
```

   添加新的条目如下所示：

```
#ifndef USE_DCI
#define USE_DCI     1
#endif
```

   b) 源**filetbl.c**文件包含如下条目：

```
extern DISPATCH_TBL v4_dispatch, ci_dispatch, bt_dispatch;
```

   添加新的条目如下所示：

```
#if USE_DCI
extern DISPATCH_TBL DBM_dispatch;
#endif  /* USE_DCI */
```

   c) 源**filetbl.c**文件包含如下条目：

```
TABLE_ENTRY file_table[] = {
#if USE_VISION
{ &v4_dispatch,   "VISIO" },
#endif  /* USE_VISION */
```

   添加新的条目如下所示：

```
#if USE_DCI
{ &DBM_dispatch,  "DCI" },
#endif  /* USE_DCI */
```

3. 编辑ACUCOBOL-GT运行配置文件**sub85.c**，该文件与ACUCOBOL-GT库所在同一个目录。

   源**sub85.c**文件包含如下条目：

```
struct  PROCTABLE WNEAR LIBTABLE[] = {
{ "SYSTEM",     call_system },
```

   添加新的条目如下所示：

```
extern int  DCI_GETENV();
extern int  DCI_SETENV();
extern int  DCI_DISCONNECT();
```

```
extern int  DCI_GET_TABLE_NAME();
extern int  DCI_SET_TABLE_CACHE();
extern int  DCI_BLOB_ERROR();
extern int  DCI_BLOB_PUT();
extern int  DCI_BLOB_GET();
extern int  DCI_GET_TABLE_SERIAL_VALUE();
extern int  DCI_FREE_XFD();
struct  PROCTABLE WNEAR LIBTABLE[] = {
{ "SYSTEM",         call_system },
{ "DCI_SETENV",     DCI_SETENV },
{ "DCI_GETENV",     DCI_GETENV },
{ "DCI_DISCONNECT", DCI_DISCONNECT },
{ "DCI_GET_TABLE_NAME", DCI_GET_TABLE_NAME },
{ "DCI_SET_TABLE_CACHE", DCI_SET_TABLE_CACHE },
{ "DCI_BLOB_ERROR", DCI_BLOB_ERROR },
{ "DCI_BLOB_PUT", DCI_BLOB_PUT },
{ "DCI_BLOB_GET", DCI_BLOB_GET },
{ "DCI_GET_TABLE_SERIAL_VALUE", DCI_GET_TABLE_SERIAL_VALUE },
{ "DCI_FREE_XFD", DCI_FREE_XFD },
{ NULL,             NULL }
};
```

4. 编辑ACUCOBOL-GT的运行配置文件**direct.c**，该文件与ACUCOBOL-GT库所在同一个目录。

源**direct.c**文件包含如下条目：

```
struct  EXTRNTABLE EXTDATA[] = {
{ NULL,         NULL }
};
```

添加新的条目如下所示：

```
extern char *dci_where_constraint;
struct  EXTRNTABLE EXTDATA[] = {
{ "DCI-WHERE-CONSTRAINT",   (char *) &dci_where_constraint },
{ NULL,         NULL }
};
```

5. 打开位于*\usr\acucobol\lib*目录下的Makefile文件，如果您需要链接到您的C例程中，可将它们添加到C程序Makefile文件的**SUBS=**

line中。查阅ACUCOBOL-GT编译文献的附录C以获取更多有关链接C子程序的详细信息。

6. 添加 **/APP_HOME/lib/libdmdcic.a**和**/APP_HOME/lib/libdmapic.a**到**FSI_LIBS=**行，其中的**/APP_HOME**为包含**@DM_PRODUCT_NAME@**的安装目录。如果**@DM_PRODUCT_NAME@** 已经安装在**/APP_HOME**下，那么Makefile将包含以下字符串：

   针对ACUCOBOL-GT 5.1和之前的版本：

   ```
   FSI_LIBS=libdmacu51.a libdmdcic.a libdmapic.a
   ```

   针对ACUCOBOL-GT 5.2：

   ```
   FSI_LIBS=libdmacu52.a libdmdcic.a libdmapic.a
   ```

   针对ACUCOBOL-GT 6.0和7.0：

   ```
   FSI_LIBS=libdmacu60.a libdmdcic.a libdmapic.a
   ```

   针对ACUCOBOL-GT 8.0：

   ```
   FSI_LIBS=libdmacu80.a libdmdcic.a libdmapic.a
   ```

   针对ACUCOBOL-GT 9.0：

   ```
   FSI_LIBS=libdmacu90.a libdmdcic.a libdmapic.a
   ```

7. 接下来，确保所在目录包含ACUCOBOL-GT运行系统，在提示符下键入如下命令：

   ```
   make -f Makefile
   ```

   该命令将编译sub.c和filetbl.c，然后链接运行系统。如果因为过期的符号表而make失败，那么可执行如下命令：

   ```
   ranlib *.a
   ```

   然后再次执行make，如果因为其它原因而make失败，请电话联系ACUCORP技术支持。

8. 通过如下命令对链接进行验证：

   ```
   ./runcbl -vv
   ```

   这将返回所有链接到运行系统的产品版本信息，确保它将回报@DM_PRODUCT_NAME@的DCI版本。

   注意 *您也可以链接运行系统的C例程。*

9. 将新的**runbl**文件复制到您的执行路径下，该文件需要每个使用系统的人都拥有执行权限，其余文件可以留在发行介质安装的目录中。

10. 建议用户使用-Fx3或-Fx4编译选项来产生XFD文件，使其和之前的ACUCOBOL-GT版本采用相同的格式，因为ACUCOBOL-GT 8.0和9.0编译器不支持一些XFD语法，如XML格式的“$XFD COMMENT directive”，但是可以通过新的XML格式将DCI_XML_XFD 1添加至DCI配置文件中。

共享库

当重新链接并执行ACUCOBOL-GT运行系统时，您也许会收到一条错误信息：

“无法载入库，没有这样的文件或目录”

“无法打开共享库”

这可能意味着操作系统无法找到一些共享库，当共享库存在于当前目录中时，这种情况也会发生。

一些版本需要设置环境变量，以指出您系统上的共享库。例如：在IBM RS/6000上运行AIX4.1，环境变量LIBPATH必须指出共享库所在的目录。在HP/UX上，环境变量为SHLIB_PATH；在UNIX SVR4上，环境变量为LD_LIBRARY_PATH。请阅读UNIX操作系统的参考文献以找到适当的方法来定位共享库。

或者，您可以使用静态链接将共享库链接到运行系统中以解决此类问题，请参考C开发系统的参考文献以找到您所使用环境的正确标记。

2.4 基础配置

DCI需要对两个配置文件的参数进行设置，第一个为cblconfig，ACUCOBOL运行配置文件，第二个为位于环境变量（请查看*配置文件变量*章节以获取更详细的信息）决定的目录中的DCI_CONFIG文件。为了能够立刻执行DCI工作，在DCI_CONFIG文件中设置DCI参数，这些参数将决定数据在数据库中的显示方式，以及定义一些DBA功能以对数据库进行访问。使用DCI之前需要设置的配置变量如下所示：

- DCI_DATABASE
- DCI_LOGIN
- DCI_PASSWD
- DCI_XFDPATH

➲ **示例**

该例显示了基本的DCI_CONFIG文件。

```
DCI_LOGIN SYSADM
DCI_PASSWD
DCI_DATABASE DBMaster_Test
```

DCI_XFDPATH /usr/dbmaster/dictionaries

DCI_DATABASE

DCI_DATABASE指定与DCI进行交互的数据库，该数据库必须存在，并可以使用DBMaster设置建立。请注意数据库名称在默认情况下是区分大小写的，并且不能多于128个字符。

➲ **语法**

配置文件中必须包含以下条目。该例中，DCI使用的数据库是DBMaster_Test。

```
DCI-DATABASE DBMaster_Test
```

请参考第6章的*DCI_DATABASE*节以获取更多详细信息。

DCI_LOGIN

为了确保您的COBOL应用程序能够访问数据库中的对象，必须设置一个用户名。配置变量DCI_LOGIN用来为所有使用DCI的COBOL应用设置用户名。最初该变量设置为SYSADM以确保对数据库的完全访问，该参数也可以设置成另一个用户名。请参考第6章的*DCI_LOGIN* 节以获取更详细的信息。

➲ **语法**

为了通过用户名SYSADM连接数据库，必须在DCI配置文件中进行如下设置：

```
DCI_LOGIN SYSADM
```

DCI_PASSWD

一旦用户名通过DCI_LOGIN变量设置，数据库账户就会通过它进行连接。请注意，默认SYSADM账户是没有密码的，如果设置的账户信息（用户名和密码）正确，那么数据库管理员将会记住这些信息，请参考第6章的*DCI_PASSWD*节以获取更详细的信息。

➲ **语法**

如果数据库账户设置为SYSADM，那么配置文件将出现以下信息。

```
DCI_PASSWD
```

DCI_XFDPATH

DCI_XFDPATH用于指定数据库字典存储的目录，默认值为当前目录。

➲ **语法1**

如果要将数据字典存储在/usr/dbmaster/dictionaries目录下，应在配置文件中设置以下条目：

```
DCI_XFDPATH /usrdbmaster/dictionaries
```

➲ **语法2**

如果要设置多个路径，那么不同的路径之间必须用空格分隔，例如：

```
DCI_XFDPATH /usr/dbmaster/dictionaries /usr/dbmaster/dictionaries1
```

➲ **语法3**

在WIN-32环境下使用双引号（""）设置嵌入的空格，例如：

```
DCI_XFDPATH c:\tmp\xfdlist "c:\my folder with space\xfdlist"
```

2.5 Runsql功能

DCI提供了一个名为runsql.acu的实用程序，该程序可以访问一些标准的SQL命令，它可以通过COBOL程序调用或使用命令行工具来执行。SQL命令最高可达32767个字符，并且在CALL语句中可能是一个变量或引用的命令字符串。

作为一般规则，runsql.acu可用于发布除数据检索之外的所有SQL命令，并且不能执行像SELECT这样的语句返回的数据，因为这种语句在传递runsql.acu程序时将返回错误。

当命令执行成功，全局变量的返回代码将为0；如果命令不成功，全局变量的返回代码将包含一个错误代码。

➲ **语法1**

下列语法用于创建一个DBMaster视图。

```
runcbl runsql.acu
```

➲ **示例1**

下例用于暂停程序以接受一条SQL命令。

```
create table TEST (col1 char(10), col2 char(10))
create view TESTW as select * from TEST
```

➲ **语法2**

下列语法用于从COBOL程序内部调用sql.acu。

```
call "runsql.acu" using sql-command
```

➲ **示例2**

```
Call "runsql" using "create view TESTW as select * from TEST".
```

2.6 无效数据

某些数据在COBOL程序中有效而在DBMaster RDBMS数据库中无效。本节列出了无法被RDBMS接受的数据类型以及解决该问题的解决方案。

COBOL值	在什么地方使用是非法的
LOW-VALUES	在USAGE DISPLAY NUMBERS和文本字段
HIGH-VALUES	在USAGE DISPLAY NUMBERS、COMP-2 numbers和COMP-3 numbers
SPACES	在USAGE DISPLAY NUMBERS和COMP-2 numbers
Zero	在DATE字段

图 2-4 非法的 COBOL 数据

请查看其它数字类型的内部存储格式以确定上述类型哪些适用。BINARY类型始终是合法的，BINARY数字以及BINARY文本字段中的值通常是合法的。

某些数据类型在DBMaster接受之前必须进行转换，DCI可通过以下方法转换这些值：

- 非法的LOW-VALUES：存储的最低可能值 （0 或 - 99999）或 DCI_MIN_DATE 默认值。
- 非法的 HIGH-VALUES：存储的最高可能值（99999）或 DCI_MAX_DATE 默认值。
- 非法的SPACES：存储为0 （或在date字段为DCI_MIN_DATE）。
- 非法的DATE值：存储为DCI_INV_DATE默认值。
- 非法的TIME：存储为DCI_INV_DATE默认值。

从数据库传向DCI的NULL域以下列方式转换为COBOL：

- Numbers （包括日期型）转换为0。
- Text （包括二进制文本）转化为空格。

如果您想更改以上除主键域外的转换规则，您可以使用DCI_NULL_ON_ILLEGAL_DATE转换成NULL非法COBOL数据。

2.7 范例应用程序

DCI文件中的示例应用程序演示了DC如何将应用程序数据映射到DBMaster数据库中，本节学习：

- 设置应用程序。
- 通过编译源代码创建应用程序对象代码。
- 应用程序数据输入。
- 通过dmSQL命令行工具和JDBA工具访问数据。
- 源代码如何符合生成表的结构。

设置应用程序

该应用程序位于\DCI目录下，并且由INVD.CBL、INVD.FD、INVD.SL、TOTEM.DEF、CBLCONFIG、INVD.XFD、DCI.CFG和对象文件INVD.ACU组成。应用程序可以从对象代码（INVD.ACU）直接运行（请参考下面的 “运行应用程序”），或者从源代码INVD.CBL进行编译，详情如下所示（请参考下面的“编译源代码”）。

注意 *要编译示例应用程序，用户应该已安装ACUCOBOL4.3或更高版本。*

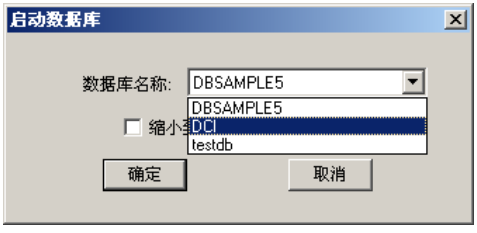
➲ **运行应用程序：**

1. 安装DBMaster数据库以接受来自DCI的数据。作为程序示例，我们可以通过JServer Manager来创建一个数据库DCI，数据库使用所有默认设置，默认登录名为SYSADM并且没有密码。要想获取更多有关创建和设置数据库的信息，请参考*数据库管理员手册*或*JServer Manager用户向导*。
2. 在\DCI目录下，通过任何一种文本编辑器打开DCI.CFG文件，将配置变量设置成合适的值。详情请参考2.4节*基础设置*。

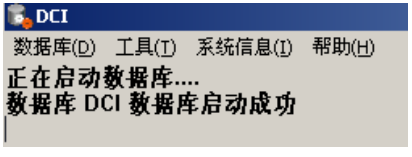
➲ 示例

```
DCI_DATABASE  DCI
DCI_LOGIN     SYSTEM
DCI_PASSWD
//DCI_LOGFILE
DCI_STORAGE_CONVENTION Dca
//DCI_XFDPATH C:\DCI
```

3. 运行DBMaster服务器程序（dmserver.exe），出现如下提示，选择已经在DCI.CFG文件中进行配置的数据库。



4. 数据库将正常启动并出现以下窗口，如果出现问题或错误信息，请参考错误信息参考手册或数据库管理员手册。



5. 在命令提示符下转到.. \DCI目录。
6. 通过在命令提示符中下键入以下指令定义DCI_CONFIG：

➲ 语法6a

```
..\DCI\>SET DCI_CONFIG=C:\..\DCI\DCI.CFG
```

7. 使用WRUN32运行COBOL程序INVD.ACU。

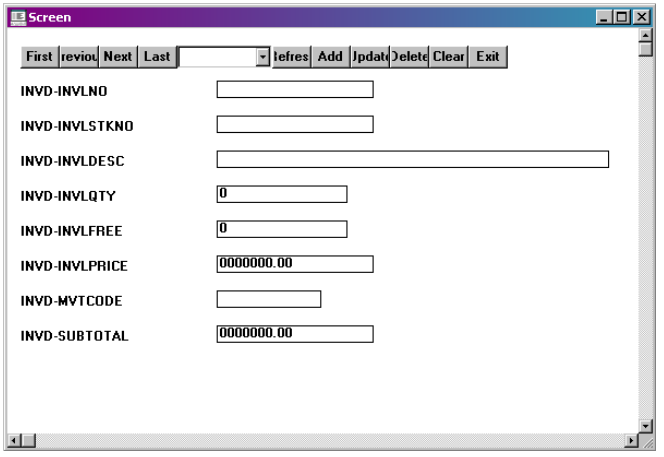
➲ 语法7a

在命令提示符的相同路径下输入如下指令：

```
..\DCI\>WRUN32 -C CBLCONFIG INVD.ACU
```

8. 文件CBLCONFIG包含命令行DEFAULT-HOST DCI并用于设置默认的文件系统，请参考第5章*编译和运行选项*以获取更多信息。

9. INVD.ACU COBOL程序窗口开启后如下所示，允许用户在字段中输入值。



注意 *有关增加记录的用法说明，请参考下节添加记录。*

➲ **从源代码处编译示例程序：**

1. 接着上例“运行应用程序”的步骤1到步骤6。

2. 将acucobol.def、acugui.def、crtvars.def、fonts.def、showmsg.def定义文件从..\Acucorp\Acucbl500\AcuGT\sample\def目录复制到..\DCI目录。

   注意 *ACUCOBOL4.3用户应该从\Acucbl43\AcuGT\sample\处复制以上定义文件。*

3. 在命令提示符下转到..\DCI目录。

   ➲ **语法3a**

   输入以下命令：

```
..\DCI\>ccbl32 -Fx INVD.CBL
```

4. 文件将被编译并且创建一个新的对象代码文件INVD.ACU和数据字典文件INVD.XFD。要运行对象文件，请遵循上例*运行应用程序*的步骤6和步骤7。

添加记录

一旦启动了应用程序 （请查看上节的*运行应用程序*)，给应用程序添加记录就变得很容易，随之而来，为数据库添加记录也就变得很简单。字段INVD-INVLNO是一个主键字段，所以记录的唯一值就变成一个有效的入口，其它字段可以为空。在将值输入到字段后，可点击添加按钮，输入到字段中的值将被立刻保存到DCI.CFG变量指定的DBMaster数据库中，DCI-DATABASE。

➲ **示例**

应用程序的文件描述符如下所示：

```
FD INVD
        LABEL RECORDS ARE STANDARD
        01      INVD-R
                05      INVD-INVLNO             PIC X(10).
                05      INVD-INVLSSTKNO         PIC X(10).
                05      INVD-INVLDESC           PIC X(30).
                05      INVD-INVLQTY            PIC 9(8).
                05      INVD-INVLFREE           PIC 9(8).
                05      INVD-INVLPRICE          PIC 9(7)V99.
                05      INVD-MVTCODE            PIC X(6).
                05      INVD-SUBTOTAL           PIC 9(7)V99.
```

一旦记录添加成功，您就可以点击First、 Previous、Next或Last按钮通过条目来进行浏览。个别记录可以从屏幕上方显示所有主键字段值的下拉菜单中进行选择。下节的*访问数据*将描述如何使用DBMaster SQL工具来浏览数据。

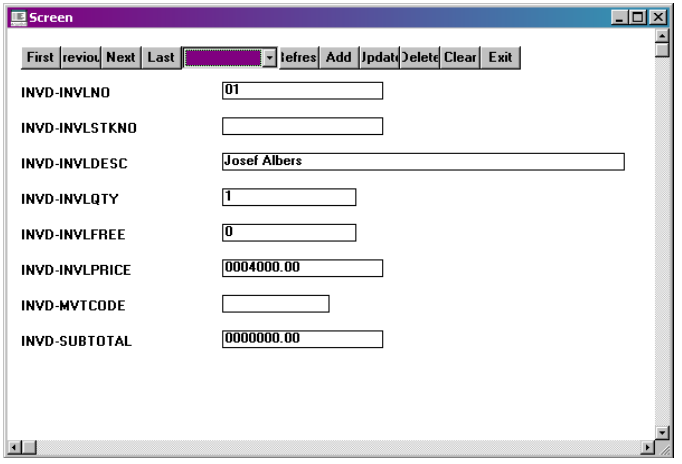


图 2-5 INVD-INVLNO关键字段设置入口

访问数据

为了便于浏览和维护示例应用程序中创建的记录。首先，我们建议您最好熟悉以下任一种DBMaster工具：dmSQL、DBA Tool或JDBA Tool。有关如何使用这些工具的信息，请参考*数据库管理员手册*或*数据库管理工具用户手册*。下例展示了如何使用数据库管理工具来访问数据。

首先必须关闭INVD应用程序，因为它给数据库中创建的表加了锁。通过JDBA工具连接数据库，扩展下图数据库树型结构中的表节点以查看表。

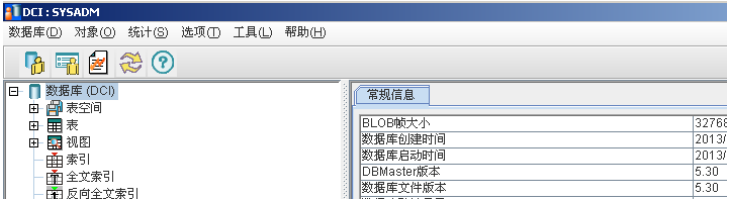


图 2-6 INVD应用程序表的树型节点

双击**SYSADM.invd**表以查看表结构，所有字段和属性都可以在此进行查看。

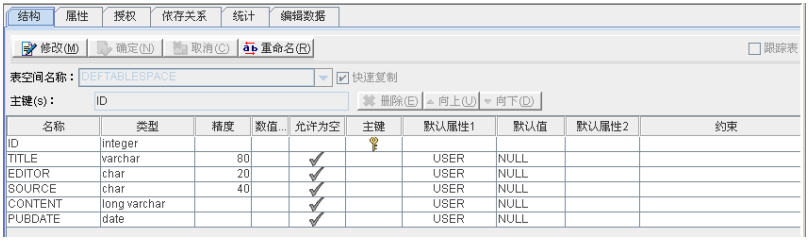


图 2-7 SYSADM.invd表结构

选择**编辑数据**标签，可查看每个字段的值。

图 2-8 SYSADM.invd编辑数据标签字段值

3 数据字典

数据字典用来描述扩展文件描述符文件（.XFD）是如何被创建和访问的。DCI在COBOL代码中避免使用SQL函数调用，而是使用ACUCOBOL-GT的特有功能。当使用“–Fx”选项编译一个COBOL程序时，就会产生数据字典。这些被称作“扩展文件描述符”（XFD文件）的文件是基于COBOL文件描述符的。DCI使用数据字典在COBOL程序与DBMaster表字段之间映射数据。DCI使用的每一个DBMaster表都至少有一个相应的数据字典文件与之关联。

注意 *关于创建XFD的更多信息和规则请参考ACUCOBOL-GT用户手册（第5.3章节）。*

3.1 指定表名称

数据库表对应于COBOL程序的FILE CONTROL节中的文件描述符。数据库的表名称必须是唯一的，且长度要小于128字节（128 ASCII字符）。

DBMaster表中的字段数可以大于对应的COBOL程序文件描述符，字段顺序也可以不同于文件描述符中的顺序。

数据库表中的字段数不必与访问该表的COBOL程序中的字段数完全匹配。DBMaster表中的字段数可以大于COBOL程序涉及的字段数，而COBOL程序中的字段数却不能多于DBMaster表中的字段数。当向表中插入新行时，请确认这些后添加的字段都已设置正确。

ACUCOBOL会默认从FILE CONTROL节产生XFD文件名称。如果SELECT语句中含有ASSIGN变量名（为filename赋值），可使用FILE指示为XFD文件指定一个起始名称（参考第四章的*$XFD FILE 指示*）；如果SELECT语句中包含一个ASSIGN常量名称（如赋值为“EMPLOYEE”），那么该常量将用来产生XFD文件名称；如果是一个普通的ASSIGN短语并指向一个设备（如赋值为“DISK”），那么编译器会使用SELECT名称来产生XFD文件名称。

文件名称和用户名称都是大小写不敏感的。文件描述符中包含的大写字母都将被转换成小写字母。如果使用的操作系统大小写敏感，那么用户就必须注意这一点了。

➲ **示例1**

如果FILE CONTROL包含下行语句：

```
SELECT FILENAME ASSIGN TO "Customer"
```

➲ **示例2**

基于读取“customer.xfd”的字典信息，DCI会产生一个名为“username.customer”的DBMaster表。Acucobol-GT编译器通常会以小写字母创建文件名。“username”的默认值是由DCI_CONFIG文件中的DCI_LOGIN值决定的，可以通过配置变量DCI_USER_PATH来更改。

```
SELECT FILENAME ASSIGN TO "CUSTOMER"
```

➲ **示例3**

如果文件有扩展名，DCI会以字符“_”来代替“.”，并打开名为“username.customer_dat”的DBMaster表。

```
SELECT FILENAME ASSIGN TO "customer.dat"
```

➲ **示例4**

DCI_MAPPING可以使字典customer.xfd可用，因为DCI使用基本名称来查找XFD字典，所以它会寻找名为“customer_dat.xfd”的XFD文件。下面的设置是基于名为“customer.xfd”的XFD文件。

```
DCI_MAPPING customer*=customer
```

COBOL程序会在不同的路径中使用相同的基本文件名，例如，COBOL程序分别在“/usr/file/customer” 和“/usr1/file/customer”路径下打开名为“customer”的文件。为了使文件名唯一，我们应该在文件名中包含路径。一种方法是更改DCI_CONFIG的变量DCI_USEDIR_LEVEL为“2”，然后DCI会以如下列方式打开表：

COBOL	RDBMS	XFD文件名称
/usr/file/customer	usrfilecustomer	usrfilecustomer.xfd
/usr1/file/customer	usr1filecustomer	usr1filecustomer.xfd

图 3-1 示范 DCI_USEDIR_LEVEL为“2”

注意 *请记住DBMaster表名称的长度有最大限制，并且必须使用DCI_MAPPING映射.XFD文件字典定义。*

COBOL 代码	最终文件名称	最终表名称
ASSIGN TO "usr/hr/employees.dat"	employees_dat.xfd	employees_dat
SELECT DATAFILE, ASSIGN TO DISK	datafile.xfd	datafile
ASSIGN TO "-D SYS$LIB:EMP"	emp.xfd	emp
ASSIGN TO FILENAME	（user specified）	（user specified）

图 3-2 不同的COBOL语句产生不同形式的表名称

➲ **示例**

表名称依次由XFD文件名称产生，另一种指定表名称的方法是使用$XFD FILE指示。

```
*(( XFD FILE = PURCHASE-FILE2 ))
FD PURCHASE-FILE.
01 PURCHASE-RECORD.
 05 DATE-PURCHASED.
    10 YYYY            PIC 9(04).
    10 MM              PIC 9(02).
    10 DD              PIC 9(02).
 05 PAY-METHOD         PIC X(05).
```

最终文件名有以下几种形式：

- 编译器用下划线“_”代替“.”，以起始名开头并包含转换的扩展名。
- 从文件名和由DCI_CONFIG变量DCI_USEDIR_LEVEL指定的路径信息创建一个通用的基础名，然后再将基础名减少到32个字符，并依照DCI_CASE值转换为小写字母。

3.2 映射字段和记录

创建的表是基于COBOL文件中的最大记录，它包含记录中的所有字段和主键字段。主键字段是由KEY IS短语在FILE CONTROL节中指定的，它对应于数据库表中的主键，下一节将对其进行详细介绍。注意DCI创建的字段名称与表名称不同，它是大小写敏感的。

➲ **示例1**

下例说明数据是如何被传输的。

```
ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
      SELECT HR-FILE
             ORGANIZATION   IS INDEXED
             RECORD KEY     IS EMP-ID
             ACCESS MODE    IS DYNAMIC.
 DATA DIVISION.
 FILE SECTION.
 FD  HR-FILE
     LABEL RECORDS ARE STANDARD.
 01  EMPLOYEE-RECORD.
     05 EMP-ID              PIC 9(06).
     05 EMP-NAME            PIC X(17).
     05 EMP-PHONE           PIC X(10).
 WORKING-STORAGE SECTION.
 01  HR-NUMBER-FIELD        PIC 9(05).
 PROCEDURE DIVISION.
 PROGRAM-BEGIN.
     OPEN I-O HR-FILE.
     PERFORM GET-NEW-EMPLOYEE-ID.
     PERFORM ADD-RECORDS UNTIL EMP-ID = ZEROS.
     CLOSE HR-FLE.
PROGRAM-DONE.
     STOP RUN.
GET-NEW-EMPLOYEE-ID.
     PERFORM INIT-EMPLOYEE-RECORD.
```

```
    PERFORM ENTER-EMPLOYEE-ID.
INIT-EMPLOYEE-ID.
     MOVE SPACES TO EMPLOYEE-RECORD.
     MOVE ZEROS  TO EMP-ID.
ENTER-EMPLOYEE-ID.
     DISPLAY "ENTER EMPLOYEE ID NUMBER (1-99999),"
     DISPLAY "ENTER 0 TO STOP ENTRY".
     ACCEPT   HR-NUMBER-FIELD.
     MOVE     HR-NUMBER-FIELD TO EMP-ID.
ADD-RECORDS.
     ACCEPT  EMP-NAME.
     ACCEPT  EMP-PHONE.
     WRITE   EMPLOYEE-RECORD.
     PERFORM GET-NEW-EMPLOYE-NUMBER.
```

➲ **示例2**

正常情况下，前述程序会将各个字段顺序地写入文件，输出的内容如下所示：

```
ENTER EMPLOYEE ID NUMBER (1-99999), ENTER 0 TO STOP ENTRY
51100
LAVERNE HENDERSON
2221212999
ENTER EMPLOYEE ID NUMBER (1-99999), ENTER 0 TO STOP ENTRY
52231
MATTHEW LEWIS
2225551212
ENTER EMPLOYEE ID NUMBER (1-99999), ENTER 0 TO STOP ENTRY
```

传统的COBOL文件系统会按顺序存储记录。每执行一次写命令，数据便会被送到文件中。如果使用了DCI，数据字典将会为存储到数据库中的数据创建映射。本例中的记录（EMPLOYEE-RECORD）是文件的唯一记录。

➲ **示例3**

数据库会为文件描述符中的每个域创建不同的字段，表名称HR-FILE将与FILE-CONTROL节中的SELECT语句保持一致。例子中的记录也因此将有如下结构：

EMP_ID (INT(6))	EMP_NAME (CHAR(17))	EMP_PHONE (DEC(10))
51100	LAVERNE HENDERSON	2221212999
52231	MATTHEW LEWIS	2225551212

图 3-3 表 EMPLOYEE-RECORD

该表中的字段EMP-ID是由input-output节的KEY IS语句定义的主键。数据字典创建的“mapping”可以用来找回记录并将其放置到正确的字段中。以这种方式存储信息的COBOL程序既可以享用数据库的备份和恢复功能，又能够充分利用SQL的强大功能。

相同的域名称

在COBOL中，把具有相同名称的域标识为一个组。DBMaster不允许在一个表中存在重复的字段名称，如果有相同的域名，DCI不会为那些域创建字段。解决该问题的一种方式是在冲突的域名称前加上NAME指示。更多细节请参考第四章*$XFD NAME指示*。

➲ **示例**

下例程序参考了PERSONNEL和PAYROLL：

```
FD HR-FILE
        LABEL RECORDS ARE STANDARD.
01      EMPLOYEE-RECORD.
        03      PERSONNEL.
                05      EMP-ID          PIC 9(6).
                05      EMP-NAME        PIC X(17).
                05      EMP PHONE       PIC 9(10).
        03      PAYROLL.
                05      EMP-ID          PIC 9(6).
                05      EMP-NAME        PIC X(17).
                05      EMP PHONE       PIC 9(10).
```

长字段名称

DBMaster支持的最大表名称为128字符，对于超过这个长度的域名称DCI会进行截取操作。下面将要描述的OCCURS子句中，截取只是针对原名称，不会涉及到附加的索引号。然而，包括了索引号的最终名称的字符限制仍然是128字符。例如：如果一个字段的名称为Employee-statistics-0，截取后的表名称是Employee_statis_01，所以一定要确保字段名称的前18个字符是唯一的、有意义的。

您可以使用NAME对长名称的域进行重命名。不过需要注意在COBOL程序中，您必须继续使用原名称，NAME指示仅对数据库中对应的字段名称有效。

3.3 使用多记录格式

上一节的例子说明了如何使用域来创建数据库表，但例子中的程序仅演示了只有一条记录的情况。

多个记录格式的存储与单个记录格式的存储完全不同。带有多个记录的COBOL程序会映射文件中的“master”（最大的）记录和所有主键字段。小一点的记录会通过XFD文件映射到数据库表，但不会以分散的定义字段出现在表中，而是会占用数据库中存在的新记录。

➲ **示例1**

还是以前面的例子为例，只是修改文件描述符以包含一些记录。

```
DATA DIVISION
FILE SECTION
FD HR-FILE
  LABEL RECORDS ARE STANDARD.
01 EMPLOYEE-RECORD.
   05 EMP-ID          PIC 9(6).
   05 EMP-NAME        PIC X(17).
   05 EMP PHONE       PIC 9(10).
01 PAYROLL-RECORD.
   05 EMP-SALARY      PIC 9(10).
   05 DD              PIC 9(2).
   05 MM              PIC 9(2).
   05 YY              PIC 9(2).
```

本例中，数据字典由最大的文件创建而来。记录EMPLOYEE-RECORD包含33个字符，而PAYROLL-RECORD包含16个字符，本例中记录被顺序地写入数据库。EMPLOYEE-RECORD用来为表的字段大小和数据类型创建模式。

```
 EMP_ID (INT(6))    EMP_NAME (CHAR(17))    EMP_PHONE (DEC(10))
```

图 3-4 前例中的表

下面记录中的域会根据域的字符位置重新写入到字段中，结果是较小的记录中没有分散的字段。数据可以通过COBOL程序从数据库中返回，因为XFD文件包含到域的映射，但是表中没有字段代表那些域。

前面的例子中，当第一条记录添加到数据库中时，在字段和COBOL域中便建立起了相互关系，而在添加第二条记录时将不再有这样的关系。数据会根据域来占有其相应的字符位置，所以EMP_SALARY的前5个字符会占据EMP_ID字段，EMP_SALARY的后5个字符会占据EMP_NAME字段，DD、MM和YY也会占据EMP_NAME字段。

➲ **示例2**

下例演示将给定的内容输入到COBOL程序：

```
ENTER EMPLOYEE ID NUMBER (1-99999), ENTER 0 TO STOP ENTRY
51100
LAVERNE HENDERSON
2221212999
5000000000
01
04
00
```

会根据相对于表模式中域字符的位置整合和拆分域。此外，字段EMP_NAME的数据类型是CHAR。因为DCI访问了数据字典，所有域都会以正确的位置映射回COBOL程序。

默认情况下，最大记录的域会用来创建表模式，因此在创建文件描述符时必须认真考虑表模式。为了享用SQL的灵活性带来的优势，在即将占有同样字符位置的不同域之间，数据类型要保持一致。如果PIC X被写到DECIMAL类型的数据库字段，数据库将会向程序返回错误。

➲ **示例3**

一个SQL语句查询了EMP_NAME所有字段的第一条记录，显示结果如下所示：

```
51100, LAVERNE HENDERSON, 2221212999
```

➲ **示例4**

一个SQL语句查询了EMP_NAME所有字段的第二条记录，显示结果如下所示：

```
500000, 0000010400
```

3.4 使用XFD文件默认值

在大多数情况下，可以通过指示来覆盖DCI的默认行为。有关更多信息，请参考*XFD指示*。

编译器使用特殊的方法处理下面的COBOL元素：

- REDEFINES子句
- KEY IS短语
- FILLER数据项
- OCCURS子句

REDEFINES子句

REDEFINES子句可以为同一个域创建多个定义。DBMaster支持每个字段有一个数据定义，因此重新定义的域在表中会与原来的域占用同样的位置。默认情况下，数据字典使用次级域的定义来定义字段数据类型。

多个记录定义从本质上来说是重新定义了整个记录，有关多记录定义请参考前面的章节。

合成表模式的数据库字典定义中不包括组，而是使用组中个别的域来产生模式。可以使用USE GROUP指示来组合成组的域。

KEY IS短语

KEY IS短语位于COBOL程序的输入-输出节中，用来定义一个域或一组域以当作所有记录的唯一索引。数据字典会将KEY IS短语中的域映射为数据库中的主键。如果KEY IS短语中的域名称是一个组，组中的次级域会成为表的主键字段。可以使用USE GROUP指示将所有次级域集合到一个域中。更多信息请参考第4章的*$XFD USE GROUP指示*。

FILLER数据项

FILLER数据项是COBOL文件描述符中的占位符，它们没有唯一名称，也不能唯一被参照。如果在字符位置中存在FILLERS，数据字典会映射所有其它命名的域，但不会为FILLER数据项创建不同的域。

如果表模式中一定要包含FILLER，可以通过USE GROUP指示（参考第4章的*$XFD USE GROUP* 指示)或NAME指示（参考第4章的*$XFD NAME* ）将其与其它域结合。

OCCURS子句

使用OCCURS子句，用户就可以根据需要对一个域进行多次定义。DCI必须对每一个数据库字段分配唯一的名称，但是一个OCCURS子句定义的多个域仍具有相同的名称。为了避免这个问题，可以给由OCCURS子句指定的域附加一个连续的索引号。

➲ **示例1**

下面给出部分文件描述符：

```
   03 EMPLOYEE-RECORD OCCURS 20 TIMES.
      05 CUST-ID                          PIC 9(5).
```

➲ **示例2**

为数据库生成下面的字段名称：

```
EMP_ID_1
EMP_ID_2
.
.
.
EMP_ID_5
EMP_ID_6
.
.
.
EMP_ID_19
EMP_ID_20
```

3.5 映射多个文件

在运行状态可以为不同名称的多个文件使用单个XFD文件。如果文件的记录定义都是相同的，那么不必为每一个文件创建单独的XFD。

动态配置变量决定被映射到一个XFD的文件。

假设COBOL程序有一个带有名称变量的SELECT语句，如EMPLOYEE-RECORD。在程序执行中该变量可以有不同的值（如EMP0001和EMP0002）。为了给XFD提供一个基本名称，可使用FILE指示（参见((XFD DATE、USE GROUP) ）。

➲ **示例**

如果“EMP”是基础，那么编译器会产生一个名为“Emp.xfd”的XFD。例子中的星号（“*”）是一个通配符，用来在文件名中代替任意多个字符。映射中不包含文件扩展名“.xfd”。该语句会使所有名称以“EMP”开头的文件都使用“emp.xfd”。在动态配置文件中加入如下代码，以保证所有employee文件（这些文件具有唯一且相关的文件名）使用相同的XFD。

```
DCI_MAPPING EMP* = EMP
```

将在打开文件阶段，读取变量DCI_MAPPING。通配符“*” 和 “?”可以以下列方式使用：

*　　匹配任意个数的字符

?　　匹配任何字符的一个占位符

EMP?????　匹配EMP00001和EMPLOYEE，但是不匹配EMP001或EMP0001

EMP*　以上都匹配

EMP*1　匹配 EMP001、EMP0001和EMP00001，但是不匹配EMPLOYEE

*OYEE　匹配 EMPLOYEE ，不匹配EMP0001或EMP00001

➲ **语法**

*<pattern>*由任何有效的文件名字符组成，可以包含“*”或“?”，DCI_MAPPING变量的语法如下：

```
DCI_MAPPING [<pattern> = base-xfd-name],
```

3.6 映射到多个数据库

可以使用DCI_DB_MAP来参照不同数据库中的表，通过指定不同的文件或COBOL文件的前缀来链接DBMS。可通过下面的例子进行说明。

➲ **示例**

要参照数据库DBNAME（默认），DBCED和DBMULTI中的表idx-1，可在DCI_CONFIG配置文件中添加以下设置：

```
DCI_DB_MAP      /usr/CED=DBCED
DCI_DB_MAP      /usr/MULTI=DBMULTI
```

在这些数据库中通过指定不同的文件来创建表idx-1：

```
...
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.

           SELECT IDX-1-FILE
           ASSIGN TO DISK "/usr/CED/IDX1"
           ORGANIZATION IS INDEXED
           ACCESS IS DYNAMIC
           RECORD KEY IS IDX-1-KEY.

           SELECT IDX-2-FILE
           ASSIGN TO DISK "/usr/MULTI/IDX1"
           ORGANIZATION IS INDEXED
           ACCESS IS DYNAMIC
           RECORD KEY IS IDX-2-KEY.

           SELECT IDX-3-FILE
           ASSIGN TO DISK "IDX1"
           ORGANIZATION IS INDEXED
           ACCESS IS DYNAMIC
           RECORD KEY IS IDX-3-KEY.

      DATA DIVISION.
       FILE SECTION.
```

```
FD  IDX-1-FILE.
01  IDX-1-RECORD.
    03  IDX-1-KEY                      PIC X(10).
    03  IDX-1-ALT-KEY.
        05  IDX-1-ALT-KEY-A            PIC X(30).
        05  IDX-1-ALT-KEY-B            PIC X(10).
    03  IDX-1-BODY                     PIC X(50).

FD  IDX-2-FILE.
01  IDX-2-RECORD.
    03  IDX-2-KEY                      PIC X(10).
    03  IDX-2-ALT-KEY.
        05  IDX-2-ALT-KEY-A            PIC X(30).
        05  IDX-2-ALT-KEY-B            PIC X(10).
    03  IDX-2-BODY                     PIC X(50).

FD  IDX-3-FILE.
01  IDX-3-RECORD.
    03  IDX-3-KEY                      PIC X(10).
    03  IDX-3-ALT-KEY.
        05  IDX-3-ALT-KEY-A            PIC X(30).
        05  IDX-3-ALT-KEY-B            PIC X(10).
    03  IDX-3-BODY                     PIC X(50).
WORKING-STORAGE SECTION.


PROCEDURE DIVISION.
LEVEL-1 SECTION.
MAIN-LOGIC.
    set environment "default-host" to "dci"

*   make IDX1 table on DBCED

    OPEN OUTPUT IDX-1-FILE
    MOVE "IDX IN DBCED" TO IDX-1-BODY
    MOVE "A" TO IDX-1-KEY
    WRITE IDX-1-RECORD
    MOVE "B" TO IDX-1-KEY
    WRITE IDX-1-RECORD
```

```
           MOVE "C" TO IDX-1-KEY
           WRITE IDX-1-RECORD
           CLOSE IDX-1-FILE

      *    make IDX1 table on DBMULTI
           OPEN INPUT IDX-1-FILE
           OPEN OUTPUT IDX-2-FILE
           PERFORM UNTIL 1 = 2
             READ IDX-1-FILE NEXT AT END EXIT PERFORM END-READ
             MOVE IDX-1-RECORD TO IDX-2-RECORD
             MOVE "IDX IN DBMULTI" TO IDX-2-BODY
             WRITE IDX-2-RECORD
           END-PERFORM
           CLOSE IDX-1-FILE IDX-2-FILE

      *    make IDX1 table on DBNAME
           OPEN INPUT IDX-1-FILE
           OPEN OUTPUT IDX-3-FILE
           PERFORM UNTIL 1 = 2
             READ IDX-1-FILE NEXT AT END EXIT PERFORM END-READ
             MOVE IDX-1-RECORD TO IDX-3-RECORD
             MOVE "IDX IN DBNAME" TO IDX-3-BODY
             WRITE IDX-3-RECORD
           END-PERFORM

           CLOSE IDX-1-FILE IDX-3-FILE
```

通过文件的前缀来读取这些数据库中的表idx-1：

```
...
      INPUT-OUTPUT SECTION.
       FILE-CONTROL.

           SELECT IDX-1-FILE
           ASSIGN TO DISK "IDX1"
           ORGANIZATION IS INDEXED
           ACCESS IS DYNAMIC
           RECORD KEY IS IDX-1-KEY.
```

```
DATA DIVISION.
FILE SECTION.
FD  IDX-1-FILE.
01  IDX-1-RECORD.
    03  IDX-1-KEY                      PIC X(10).
    03  IDX-1-ALT-KEY.
        05  IDX-1-ALT-KEY-A            PIC X(30).
        05  IDX-1-ALT-KEY-B            PIC X(10).
    03  IDX-1-BODY                     PIC X(50).

WORKING-STORAGE SECTION.

PROCEDURE DIVISION.
LEVEL-1 SECTION.
MAIN-LOGIC.
    set environment "default-host" to "dci"

    set environment "file-prefix" to "/usr/MULTI:/usr/CED".
    OPEN INPUT IDX-1-FILE
    READ IDX-1-FILE NEXT
    DISPLAY IDX-1-BODY
    ACCEPT OMITTED
    CLOSE IDX-1-FILE

    set environment "file-prefix" to "/usr/CED:/usr/MULTI".
    OPEN INPUT IDX-1-FILE
    READ IDX-1-FILE NEXT
    DISPLAY IDX-1-BODY
    ACCEPT OMITTED
    CLOSE IDX-1-FILE

    set environment "file-prefix" to ".:/usr/CED:/usr/MULTI".
    OPEN INPUT IDX-1-FILE
    READ IDX-1-FILE NEXT
    DISPLAY IDX-1-BODY
    ACCEPT OMITTED
    CLOSE IDX-1-FILE
```

3.7 使用触发器

COBOL触发器是DCI的实用而强大的功能。COBOL触发器可自动执行预先定义的COBOL程序以响应特殊的I/O事件，而无论这些事件是哪个用户或应用程序产生的。

COBOL触发器可用于以下几个方面：

- 实现商业规则
- 为COBOL行为创建一个审核追踪
- 从现有数据获取附加值
- 跨越多个文件复制数据
- 执行安全许可程序
- 控制数据完整性
- 定义非传统的完整性约束

使用下面XFD指示来定义COBOL触发器，以指定I/O事件发生时调用的COBOL程序名。

➲ **语法**

```
$XFD DCI COMMENT COBTRIGGER "cobolpgmname"
```

➲ **示例1**

该“cobolpgmname”是大小写敏感的并且位于CODE-PREFIX目录或当前运行目录下。I/O事件可以被READ （任何一种）、WRITE、REWRITE、DELETE和OPEN。除了OPEN执行BEFORE I/O事件外，COBOL触发器可以执行所有BEFORE和AFTER I/O事件。

```
$xfd dci comment cobtrigger "cobtrig"
```

➲ **示例2**

该“cobolpgmname”必须遵循以下LINKAGE SECTION规则：

```
LINKAGE SECTION.
```

```
01    op-code  PIC x.
88    read-after     value "R".
88    read-before    value "r".
88    write-after    value "W".
88    write-before   value "w".
88    rewrite-after  value "U".
88    rewrite-before value "u".
88    delete-after   value "D".
88    delete-before  value "d".
88    open-before    value "O".

01    record-image   PIC x(32767).

01    rc-error       PIC 99.
```

➲ **示例3**

*Op-code*为基于I/O事件的DCI值；*record-image*包含了before/after I/O事件的COBOL记录值；通过以下值，*rc-error*可用于强迫COBOL I/O事件出错：

```
88  F-IN-ERROR                  VALUES 1 THRU 99.
88  E-SYS-ERR                   VALUE 1.
88  E-PARAM-ERR                 VALUE 2.
88  E-TOO-MANY-FILES            VALUE 3.
88  E-MODE-CLASH                VALUE 4.
88  E-REC-LOCKED                VALUE 5.
88  E-BROKEN                    VALUE 6.
88  E-DUPLICATE                 VALUE 7.
88  E-NOT-FOUND                 VALUE 8.
88  E-UNDEF-RECORD              VALUE 9.
88  E-DISK-FULL                 VALUE 10.
88  E-FILE-LOCKED               VALUE 11.
88  E-REC-CHANGED               VALUE 12.
88  E-MISMATCH                  VALUE 13.
88  E-NO-MEMORY                 VALUE 14.
88  E-MISSING-FILE              VALUE 15.
88  E-PERMISSION                VALUE 16.
```

```
88  E-NO-SUPPORT                VALUE 17.
88  E-NO-LOCKS                  VALUE 18.
88  E-INTERFACE                 VALUE 19.
```

3.8 使用视图

DCI允许使用DBMaster视图来代替表。既然这样，DCI用户就必须手动创建视图并且清楚以下限制：

当视图为单个表视图并且原始表的映射字段没有表达式、集合或UDF时，可以打开视图并且执行所有DML操作。

针对其它类型的视图，可以打开视图作为OPEN INPUT并且只能执行READ操作。

➲ **示例**

下例展示了如何创建并打开一个视图，假设创建了如下两张表t2和t3：

```
create table t2 ( c1 char(30), c2 int);
create table t3(c1 int);
```

并且在表中填充了一些数据。

```
identification division.

       file-control.
         select miofile assign to ws-nomefile
              organization indexed
              access mode dynamic
              record key rec
              .
       data division.
       file section.
      $XFD FILE=miofile
       fd miofile.
       01 rec.
          03 c1 pic x(30).
          03 c2 pic 9(9).

       working-storage section.
       01 ws-nomefile pic x(30).
       01 sql-command pic x(1000).
       procedure division.
```

```
 main.
      set environment "default_host" to "dci"
      display "Enter the name of the view to create:" no
      accept ws-nomefile

      inspect ws-nomefile replacing trailing spaces
              by low-value

      string "create view " delimited by size
             ws-nomefile delimited by low-value
           " as (select c1, c2 from t2 where c2 in (select max(c1)
-            " from t3));" delimited by size
             x"00" delimited by size
         into sql-command

      display sql-command
      accept omitted

      call "i$io" using 15, "dci", sql-command
      if return-code not = 0
         display "Errore : " return-code
         accept omitted
         stop run
      end-if

      string "commit;" delimited by size
             x"00" delimited by size
         into sql-command
      call "i$io" using 15, "dci", sql-command
      if return-code not = 0
         display "Errore : " return-code
         accept omitted
         stop run
      end-if
      open input  miofile
      perform until 1=2
         read miofile next
           at end exit perform
         end-read
```

```
            display rec
        end-perform
        close miofile

        exit program
```

3.9 使用同义字

DCI允许使用DBMaster同义字来代替表或视图。用户可以在表、视图或远程数据库的表和视图上创建同义字。如果视图的同义字不是单个表视图，那么用户只能通过同义字执行OPEN INPUT。

3.10 打开远程数据库中的表

用户可以在COBOL SELECT语句中添加特殊的符号“@”来访问远程数据库中的表或视图。

```
SELECT tb1 ASSIGN TO RANDOM, "lnk1@tb1"
```

若想使用不同的用户名和密码，需在**dmconfig.ini**配置文件中设置**DD_DDBMD**=1并创建远程数据库链接。

➲ **示例**

连接数据库dci_db1并且访问数据库dci_db2中的表。

1. 在**dmconfig.ini**配置文件中设置**DD_DDBMD**=1。
2. 然后在**dci_db2**数据库中创建表。

   注意 *使用dmSQL命令行工具在dci_db2数据库中创建表*
3. 使用COBOL程序连接dci_db1，然后打开在dci_db2数据库中的表。

```
dmconfig.ini
[DCI_DB1]
DB_SVADR = 127.0.0.1
DB_PTNUM = 22999
DD_DDBMD = 1

[DCI_DB2]
DB_SVADR = 127.0.0.1
DB_PTNUM = 23000
DD_DDBMD = 1

Use dmSQL tool to create the table
connect to DCI_DB2 SYSADM;
create table tb1 (c1 int not null, c2 int, c3 char(10), primary key c1);
commit;
disconnect;

COBOL program
```

```
identification division.
program-id.RemoteTable.
date-written.
remarks.
environment division.
input-output section.
file-control.
      SELECT tb1 ASSIGN TO RANDOM, "dci_db2@tb1"
             ORGANIZATION IS INDEXED
             ACCESS IS DYNAMIC
             FILE STATUS IS I-O-STATUS
             RECORD KEY  IS C1.

data division.
file section.
FD  tb1.
01  tb1-record.
        03 C1             PIC 9(8) COMP-5.
        03 C2             PIC 9(8) COMP-5.
        03 C3             PIC X(10).
working-storage section.
77   I-O-STATUS pic xx.
procedure division.
main.
      set environment "default_host" to "dci"
      call "DCI_SETENV" using "DCI_DATABASE" "DCI_DB1"
      call "DCI_SETENV" using "DCI_LOGIN" "SYSADM"

      open i-o tb1
      move 100 TO C1.
      move 200 TO C2.
      move "AAAAAAAAAA" TO C3.
      write tb1-record.
      initialize tb1-record.
      read tb1 next.
          display C1, C2, " ", C3.
      close tb1.
      accept omitted.
      stop run.
```

3.11 使用DCI_WHERE_CONSTRAINT

DCI_WHERE_CONSTRAINT用于为随后的START操作指定附加的WHERE条件。为了符合Acu4gl，DCI同样也支持4gl_where_constraint。

➲ **示例**

如果您想从A开始查询城市名称，可添加以下代码：

```
WORKING-STORAGE SECTION.
01  dci_where_constraint pic x(4095) is external.
...

PROCEDURE DIVISION.
...

* to pecify dci_where_constraint
move low-values to dci_where_constraint
  open i-o idx-1-file
  move "city_name = 'a%'" to dci_where_constraint
  inspect dci_where_constraint replacing trailing spaces by low-values.

  move spaces to idx-1-key
  start idx-1-file key is not less idx-1-key
  ....
```

4 XFD指示

指示是位于COBOL文件描述符中的注释，用于改变已经构建的数据库表。指示被用于改变数据库中的数据并且为数据库的域分配名称。指示同样可以为.XFD文件命名，为二进制大型对象（BLOB）域分配数据以及添加注释。

4.1 使用指示语法

在相关的COBOL代码行之前，每一个指示都单独排成一行。所有的指示都有$XFD前缀，第七列符号$紧跟在XFD后。

- **语法1**

  下列命令为一个未定义的COBOL变量提供了唯一的数据库名。指令在它影响的指定行之前直接被找到，本例中第二个例子是COBOL定义变量*qty*。

```
. . .
    03 QTY        PIC 9(03).
 01 CAP.
$XFD  NAME=CAPQTY
    03 QTY        PIC 9(03).
```

- **语法2**

  指令可以使用下列ANSI适用语法来指定：

```
*(( XFD NAME=CAPQTY ))
```

- **语法3**

  可以组合多条指令。指令前附加签注$XFD，并且由空格和逗号隔开时，指令可以在同一行。

```
$XFD NAME=CAPQTY, ALPHA
```

- **语法4**

  或者以下列方式使用：

```
*(( XFD NAME=CAPQTY, ALPHA))
```

4.2 使用XFD指示

当映射一个COBOL文件描述符到数据库域时需要使用指示。$XFD前缀向编译器表明在数据字典产生期间使用了处理命令。

$XFD ALPHA指示

为了在数字类型的主键中存储非数字数据，如LOW-VALUES值或特定代码，该指示允许在COBOL程序中被定义为数字类型的数据项，在数据库中转化为文字数字文本（CHAR（n）n 1-列最长度）。

➲ **语法1**

```
$XFD ALPHA
```

➲ **语法2**

```
*(( XFD ALPHA ))
```

将一个非数字类型的值如“A234”更改为主键，若不使用$XFD ALPHA指示，该操作会被数据库拒绝。

➲ **示例1**

假设确立主键IS代码被指明并且已经有下列记录定义。CODE-NUM是数字类型值并且是主键，因此数据项目组在数据库中被忽视。

```
01 EMPLOYEE-RECORD.
    05 EMP-KEY.
       10 EMP-NUM          PIC 9(5).
```

➲ **示例2**

使用$XFD文字指示会改变非数字值如“A234”，这样记录就不会被数据库拒绝，因为“A234”是一个文字数值并且代码数是数字值。

```
01 EMPLOYEE-RECORD.
    05 EMP-KEY.
$XFD  ALPHA
       10 EMP-NUM          PIC 9(5)
```

➲ **示例3**

现在，下列操作将被使用而不必担心有任何错误弹出。

```
MOVE "C0531" TO CODE-KEY.
WRITE CODE-RECORD.
```

$XFD BINARY指示

为了允许字段中的数据可以成为任何类型的文字数据，您可以使用BINARY指示。LOW-VALUES情况下，在数据类型的域中COBOL允许LOW VALUES和HIGH-VALUES，然而DBMaster不允许。BINARY指示将COBOL域转化为DBMaster BINARY数据类型。

➲ **语法1**

```
$XFD BINARY
```

➲ **语法2**

```
*(( XFD BINARY ))
```

➲ **示例**

下例允许下限值被移到CODE-NUM中。

```
01 EMPLOYEE-RECORD.
    05 EMP-KEY.
       10 EMP-TYPE      PIC X.
$(( XFD BINARY ))
       10 EMP-NUM       PIC 9(05).
       10 EMP-SUFFIX    PIC X(03).
```

$XFD COMMENT DCI SERIAL n指示

这个指示被用于定义连续的数据域以及可选择的开始数“n”。为使得DBMaster生成一个连续的数字，插入一条记录并且为连续的域赋予0值。若插入一条记录，但为连续的域赋予非0的整数数值时，DBMaster不会生成一个连续的数字。若提供的整数值比上次生成的整数值大，DBMaster会重新设置上次生成的结果并以提供的整数值作为开始。

➲ **语法1**

```
$XFD COMMENT DCI SERIAL 1000
```

➲ **语法2**

```
*(( XFD COMMENT DCI SERIAL 1000 ))
```

➲ **示例**

```
01 EMPLOYEE-RECORD.
    05 EMP-KEY.
       10 EMP-TYPE      PIC X.
$(( XFD COMMENT DCI SERIAL 250 ))
       10 EMP-COUNT     PIC 9(05).
```

$XFD COMMENT DCI COBTRIGGER指示

定义一个COBOL程序时，该指示可作为读取、写入、重写或删除的I/O事件的触发器。它使得定义的COBOL程序在每一个I/O事件后都会自动调用。

➲ **语法1**

```
$XFD COMMENT DCI COBTRIGGER "cblprogramname"
```

➲ **语法2**

```
*(( XFD COMMENT DCI COBTRIGGER "cblprogramname"))
```

$XFD COMMENT指示

该指示用来识别一个XFD文件的注释。这样，信息就能嵌入到XFD文件中使得其它应用程序可以访问数据字典。以注释形式嵌入的信息使用指示并不能干扰DCI界面的处理。每一条在XFD文件中的注释都会因为在第一列中的“#”符号而被识别。

➲ **语法1**

```
$XFD COMMENT text
```

➲ **语法2**

```
*(( XFD COMMENT text ))
```

$XFD DATE指示

DATE类型数据是DBMaster支持的一个特殊的数据格式而COBOL并不支持。将它转换成为数据类型的数据，可发挥这种数据类型的优势。DATE指示的目的是存储数据库中的一个域。这个指示将date区别于其他数字，因此数据库管理系统中可以使用date支持的工具。

- **语法1**

```
$(( XFD DATE=date-format-string ))
```

- **语法2**

```
*((XFD DATE= ))
```

如果date格式字符串被指明，那么六个阿拉伯数字（或者六个字符）域将以YYMMDD格式在数据库中被返回。八个阿拉伯数字域将以YYYYMMDD格式被返回。

date格式字符串是一个确定的日期格式描述，由字符组成。

字符	说明
M	月 （01-12）
Y	年 （2 or 4 digit）
D	月中的天数 （01-31）
J	Julian 日期 （00000000-99999999）
E	年中的天数 （001-366）
H	时 （00-23）
N	分 （00-59）
S	秒 （00-59）
T	百分之一秒

图 4-1 日期格式串字符

date格式的字符串的每一个字符都可以看作是一个存储在特定区域代表信息类型的占位符。字符同样决定有多少个阿拉伯数字被每一种类型的数据使用。

例如，尽管一般使用两个阿拉伯数字来表示月份，如果您指定MMM为您的日期格式，结果中的日期将会使用三个数字来表示月份并用0填写左侧的数字。如果月份格式为M，结果中的日期将使用单个数字并且会截去左侧的数字。

JULIAN日期

Julian日期的定义多种多样，因此DATE指示可以灵活表示。许多定义Julian日期为每年开始，一月一号为001，一月二号为002等等。为使用这种定义方法，需在数据格式中使用FEE（年的天数）。

其他方法将Julian定义为基于给定日期的第多少天。在DATA指令中，这种定义是通过字符J来实现的，例如，一个六位数字日期可以通过指令$XFD DATE=JJJJJJ来实现。默认的Julian日期基本格式为*01/01/0001AD*。

您可以通过设置配置文件变量DCI_JULIAN_BASE_DATE来定义您的Julian日期计算的基本日期。DCI认为下列范围内的日期是有效的：01/01/0001至12/31/9999。

如果COBOL程序试图写入一条包含DCI知道的日期是无效的，DCI插入一个数据值依赖于由DCI_INV_DATE, DCI_MIN_DATE以及DCI_MAX_DATE指定的设置，配置变量在数据域内设置并写入记录。

如果COBOL程序试图从一个有NULL数据字段的表中插入一条记录，零值将会插入到COBOL记录的字段中。

如果数据域有两位数的年份，那么从0到19将被认为是2000至2019年被插入，并且20到99将被认为是1920和1999年。您可以通过改变变量DCI_DATE_CUTOFF的值来更改操作。同样的，有关配置变量DCI_MAX_DATE 和 DCI_MIN_DATE，当数据作为主键时将被认为是无效日期。

注意 *如果一个域作为主键的一部分，那么它不能为空值。*

USING GROUP ITEMS

当您使用USE GROUP指示时，您可以在数据项目组前设置DATE指示。

➲ **示例1**

```
$XFD DATE
    05 DATE-PURCHASED      PIC 9(08).
    05 PAY-METHOD          PIC X(05).
```

字段date-purchased会有八位数，并且会在数据库中以YYYYMMDD的格式输入数据。

➲ **示例2**

```
$(( XFD DATE, USE GROUP ))
    05 DATE-PURCHASED.
       10 YYYY             PIC 9(04).
       10 MM               PIC 9(02).
       10 DD               PIC 9(02).
    05 PAY-METHOD          PIC X(05).
```

$XFD FILE指示

FILE指示将数据字典的文件扩展命名为.XFD。这个指示在指定的SELECT COBOL语句中创建不同的.XFD文件名时是必需的。另一种情况是没有指明COBOL文件名时需要这种指示。

➲ **语法1**

```
$XFD FILE=filename
```

➲ **语法2**

```
*(( XFD FILE=filename ))
```

➲ **示例**

本例中，ACUCOBOL-GT编译器使得XFD文件名调用CUSTOMER.xfd。

```
ENVIRONMENT DIVISION.
 FILE-CONTROL.
 SELECT FILENAME ASSIGN TO VARIABLE-OF-WORKING.
 . . .
 DATA DIVISION.
```

```
 FILE SECTION.
$XFD FILE=CUSTOMER
 FD FILENAME
 . . .
```

$XFD NAME指示

NAME指示为下一条命令限定的域分配一个DBMaster RDBMS字段名。DBMaster所有的字段名都是唯一的并且必须小于128个字符。这个指示可以避免在创建字段名时出现矛盾的或多余的名称。

➲ **语法1**

```
$XFD NAME=columnname
```

➲ **语法2**

```
*(( XFD NAME=columnname ))
```

➲ **示例**

在DBMaster RDBMS中COBOL域cus-cod会形成一个名为customercode的RDBMS域。

```
$XFD NAME=customercode
    05 cus-cod        PIC 9(05).
```

$XFD NUMERIC指示

如果声明作为文字字符时，NUMERIC指示会将后面的域作为无正负的整数。

➲ **语法1**

```
$XFD NUMERIC
```

➲ **语法2**

```
*(( XFD NUMERIC ))
```

➲ **示例**

customer-code域在DBMaster表中将以INTEGER类型数据被存储。

```
$xfd numeric
```

```
        03      customer-code           PIC x(7).
```

$XFD USE GROUP指示

USE GROUP指示在DBMaster表中为单独的字段分配一组项目。数据库字段的结果集默认的数据类型是文字数字类型（CHAR (n), 当 n = 1 - 字段最大长度）。如果数据作为不同类型（BINARY、DATE、NUMERIC）被存储，那么指示要与其他指示结合。组中合并的域将会提高访问数据库的速度，因此应认真考虑哪些域需要被结合。

➲ **语法1**

```
$XFD USE GROUP
```

➲ **语法2**

```
*(( XFD USE GROUP ))
```

➲ **示例1**

通过增加USE GROUP指示，字段code-key的数据作为单独的数据域被存储。

```
01 CODE-RECORD.
$XFD USE GROUP
    05 CODE-KEY.
       10 AREA-CODE-NUM    PIC 9(03).
       10 CODE-NUM         PIC 9(07).
```

➲ **示例2**

USE GROUP指示可以兼容其他指示。在数据库中这个域会形成一个单独的DATE类型数据。

```
$(( XFD DATE, USE GROUP ))
    05 DATE-PURCHASED.
       10 YYYY             PIC 9(04).
       10 MM               PIC 9(02).
       10 DD               PIC 9(02).
```

$XFD VAR-LENGTH指示

VAR-LENGTH指示强制DBMaster使用BLOB域来保存COBOL域。如果COBOL域超过或接近常规数据类型允许的字段大小时，使用这个指示是非常有用的，请参考第8章*COBOL转换*。

由于BLOB域不能被任何主键使用并且比其他正常数据类型如CHAR的返回速度要慢，因此我们建议您在必要时使用。

➲ **语法1**

```
$XFD USE VAR-LENGTH
```

➲ **语法2**

```
*(( XFD USE VAR-LENGTH ))
```

➲ **示例**

```
$XFD USE VAR-LENGTH
    05 LARGE-FIELD    PIC X(10000).
```

$XFD 文件名的WHEN指示

WHEN指示被用于构建确定的DBMaster列，并且不会按照默认方式构建。通过在代码中指定WHEN指示，紧跟在指示后的域（以及组项目中下属的域）在数据库表中会立即出现一个字段或许多字段。

不论是否清楚，数据库都会存储并返回所有域。此外，还包括主键域与数据库表中自动生成的字段最大记录的域。当您想包括多重记录定义或重新定义数据库表时，WHEN指示仅用于保证另外的域会成为清楚的字段。

字段如何被使用是WHEN指示的作用之一。其他您想在数据库表中清楚说明的域不会被填充或者作为主键域而占用同样的域。

➲ **语法1**

等于

```
$XFD WHEN field=value
```

➲ **语法2**

小于或等于)

```
$XFD WHEN field<=value
```

➲ **语法3**

小于

```
$XFD WHEN field<value
```

➲ **语法4**

大于或等于

```
$XFD WHEN field>=value
```

➲ **语法5**

大于

```
$XFD WHEN field>value
```

➲ **语法6**

不等于

```
$XFD WHEN field!=value
```

➲ **语法7**

OTHER与“=”来一起使用。本例中，如果相同级别列出的WHEN条件没有实现时，则必须使用这些域或OTHER后面的域。OTHER在一条记录定义前使用并且一次只能定义一条记录。域中的数据不符合其他WHEN指示限定的条件时，WHEN指示必须配合OTHER使用。否则，结果将不确定。

```
$XFD WHEN field=OTHER
```

➲ **语法8**

Value是引号内使用的准确数值，而且该域就是之前定义的COBOL域。

```
*(( XFD WHEN field(operator)value ))
```

➲ **示例**

允许使用引号（""）内的准确数值。

```
     05 AR-CODE-TYPE        PIC X.
$XFD WHEN AR-CODE-TYPE="S"
     05 SHIP-CODE-RECORD    PIC X(04).
$XFD WHEN AR-CODE-TYPE="B"
     05 BACKORDER-CODE-RECORD REDEFINES SHIP-CODE-RECORD.
$XFD WHEN AR-CODE-TYPE=OTHER
     05 OBSOLETE-CODE-RECORD  REFEFINES SHIP-CODE-RECORD.
```

TABLENAME选项

WHEN指示里的TABLENAME选项是根据WHEN指示运行时的值改变表的名称。

在WHEN语句中使用TABLENAME选项时，请注意配置变量DCI_DEFAULT_RULES与 filename_RULES DCI。

➲ **示例1**

下例是一个在两个表名中使用“When”指示的COBOL FD结构：

```
FILE SECTION.
$XFD FILE=INV
 FD  INVOICE.
$XFD WHEN INV-TYPE = "A" TABLENAME=INV-TOP
 01  INV-RECORD-TOP.
     03 INV-KEY.
        05 INV-TYPE         PIC X.
        05 INV-NUMBER       PIC 9(5).
        05 INV-ID           PIC 999.
     03 INV-CUSTOMER        PIC X(30).
$XFD WHEN INV-TYPE = "B" TABLENAME=INV-DETAILS
 01  INV-RECORD-DETAILS.
     03 INV-KEY-D.
        05 INV-TYPE-D       PIC X.
        05 INV-NUMBER-D     PIC 9(5).
        05 INV-ID-B         PIC 999.
     03 INV-ARTICLES        PIC X(30).
     03 INV-QTA             PIC 9(5).
     03 INV-PRICE           PIC 9(17).
```

➲ **示例2**

DCI界面根据例1中inv-type域的值来给两个表命名为“**inv-top**”和“**inv-details**”。DCI通过检查inv-type域的值来确定在何处填写记录。

```
*MAKE TOP ROW
   MOVE "A" TO INV-TYPE
   MOVE 1 TO INV-NUMBER
   MOVE 0 TO INV-ID
   MOVE "acme company" TO INV-CUSTOMER
   WRITE INV-RECORD-TOP
*MAKE DETAIL ROWS
   MOVE "B" TO INV TYPE
   MOVE 1 TO INV-NUMBER
   MOVE 0 TO INV-ID
   MOVE "floppy disk" TO INV-ARTICLES
   MOVE 10 TO INV-QTA
   MOVE 123 TO INV-PRICE
   WRITE INV-RECORD-DETAILS
```

运行上述代码，DCI会在“**INV-TOP**”中填写“**TOP-ROW**”记录，在“**INV-DETAILS**”表中填写“**DETAIL-ROW**”。当DCI读取了以上数据，就可以使用连续读取或者使用主键访问来填写记录。如果您打算通过记录类型来连续读取，您必须设置DCI_DEFAULT_RULES = POST或= COBOL。或者如果您打算连续读取内部记录类型，您必须设置DCI_DEFAULT_RULES=BEFORE 或= DBMS。

使用这个规定利弊参半。若有100% COBOL ANSI读取动作，您应该使用“**POST**”或“**COBOL**”方法，但是这个方法会降低性能（要读取更多记录且包括同时打开的表）。

如果您使用“**BEFORE**”或“**DBMS**”方法，当$WHEN条件读取记录级别匹配时要开启包含的所有表。

➲ **示例3**

换言之，如果您使用之前的所有记录，请参考下述代码：

```
      OPEN INPUT INVOICE.
*  to see the customer invoice
      READ INVOICE NEXT.
```

```
      DISPLAY "Customer: " INV-CUSTOMER
      DISPLAY "Invoice number: " INV-NUMBER
*  to see the invoice details
      READ INVOICE NEXT.
      DISPLAY INV-ARTICLES.
```

如果方法是**POST**或**COBOL**，**open input**会开启所有表，并且**read next**会读取所有不同的表。

➲ **示例4**

匹配的表会在**start**声明级别时被开启。如果方法是**BEFORE**或 **DBMS**，请参见下列代码：

```
        open input invoice.
*     to see the customer invoice
        move "A" to  inv-type
        move 1   to  inv-number
        move 0   to  inv-id
        start invoice key is = inv-key.
        read invoice next
        display "Customer  " inv-customer
display "Invoice number "inv-number
*     to see the  invoice details
        move "B" to  inv-type
        move 1   to  inv-number
        move 0   to  inv-id
start invoice key is = inv-key.
        read invoice next
        display inv-articles
```

$XFD COMMENT DCI SPLIT

当DCI界面生成一个新表时，DCI SPLIT指示用于限定一个或多个表拆分点的开始。

➲ **示例1**

COBOL FD结构使用DCI SPLIT指示。

本例中三个DBMaster表分别为：INVOICE, INVOICE_A与INVOICE_B，它们在拆分点之间的域被创建。

```
FILE SECTION.
FD  INVOICE.
01  INV-RECORD-TOP.
     03 INV-KEY.
        05 INV-TYPE         PIC X.
        05 INV-NUMBER       PIC 9(5).
        05 INV-ID           PIC 999.
     03 INV-CUSTOMER        PIC X(30).
$XFD DCI SPLIT
     03 INV-KEY-D.
        05 INV-TYPE-D       PIC X.
        05 INV-NUMBER-D     PIC 9(5).
        05 INV-ID-B         PIC 999.
$XFD DCI SPLIT
     03 INV-ARTICLES        PIC X(30).
     03 INV-QTA             PIC 9(5).
     03 INV-PRICE           PIC 9(17).
```

5 编译和运行选项

本章将描写ACUCOBOL-GT使用的配置设定，以指定用于什么文件系统。

5.1 使用ACUCOBOL-GT默认文件系统

正如ACUCOBOL-GT配置文件中定义的一样，COBOL程序打开的现有文件是与它们各自的文件系统相关联的。当一个COBOL程序创建了新文件，您需要指定使用什么样的文件系统。所以需要对ACUCOBOL-GT配置文件进行设置以保证新文件能够使用所选的文件系统。

如果没有为新文件指定其它的文件系统，那么DEFAULT-HOST设置将会告诉ACUCOBOL应该使用哪个文件系统。默认情况下，ACUCOBOL会使用Vision文件系统。可以通过filename-HOST为指定的文件设置文件系统。在设置中，filename将用具体的文件名称来代替。

下面ACUCOBOL-GT配置文件中的变量会保证所选的文件系统能被使用。

➲ **语法1**

```
DEFAULT-HOST (*)
```

➲ **语法2**

```
filename-HOST (*)
```

5.2 使用DCI默认文件系统

为了充分发挥DBMaster的优势，充分享有DBMaster带来的高可靠性以及复制、备份和完整性约束等特性，我们建议您使用DEFAULT-HOST DCI而避免使用ACUCOBOL-GT Vision文件系统。如果没有指定文件系统，默认情况将使用Vision文件系统。

➲ **语法1**

在本例中，所有的新文件都是DBMaster文件，除非新文件已经被指定到不同的文件系统中。

```
DEFAULT-HOST DCI
```

➲ **语法2**

通过下面的命令，除非特别指定了其它文件，否则创建的新文件都将是Vision文件。

```
DEFAULT-HOST VISION
```

5.3 使用多个文件系统

Filename-HOST用来关联新文件到指定的文件系统，与DEFAULT-HOST变量的不同之处在于它是将单独数据文件与文件系统相关联。通过这种方式，文件就可以使用不同的文件系统，而不是默认文件系统了。

要完成关联，可用文件名称代替配置文件“DEAFAULT”，不用使用路径名称或文件后缀。DEFAULT-HOST可以与filename-HOST同时使用。

➲ **示例**

在本例中，file1和file2将会使用DBMaster，而其它文件则会使用vision文件系统。

```
DEFAULT-HOST VISION
file1-HOST DCI
file2-HOST DCI
```

5.4 使用环境变量

为了能够在程序执行期间设置文件系统，可在COBOL代码中指定下列代码。（*)只能用于ACUCOBOL运行期间。同时，请记住指定文件系统通常也是在运行配置文件时完成的，并且在COBOL程序中不能更改。

关于如何使用ACUCOBOL-GT编译器和运行环境的详细说明，请参考*ACUCOBOL-GT用户手册*。

➲ **语法1**

```
SET ENVIRONMENT "filename-HOST" TO filesystem (*)
```

➲ **语法2**

```
SET ENVIRONMENT "DEFAULT-HOST" TO filesystem (*)
```

6 配置文件变量

本章列举了DCI可接受数据的范围以及COBOL数据类型与DBMaster数据类型之间的映射表。配置文件变量可用于修改DCI的标准行为并在文件中以DCI_CONFIG名称保存。

6.1 设置DCI_CONFIG变量

通过为环境变量DCI_CONFIG设置值，就可以为配置文件设置不同的地址。这个指定的环境变量值可以是一个完整的路径名称或者是配置文件路径，这样DCI就会寻找保存在环境变量指定路径下的文件DCI_CONFIG。如果配置变量指定的文件不存在，DCI将不显示错误信息并假设没有指定配置变量。该变量可在COBOL运行配置文件里设定。

➲ **语法1**

在UNIX环境下，DCI将寻找文件DCI_CONFIG。这个环境变量用来设定DCI配置文件的路径和名称。若使用Bourne shell，将使用以下命令。

```
DCI_CONFIG=/usr/marc/config;export DCI_CONFIG
```

➲ **语法2**

在DOS环境下，DCI读取路径c:\etc\test下名为DCI_CONFIG的配置文件。

```
set DCI_CONFIG=c:\etc\test
```

➲ **语法3**

在Unix环境下，DCI 在/home/test路径下使用文件“DCI”。

```
DCI_CONFIG=/home/test/dci; export DCI_CONFIG
```

DCI_CASE

COBOL里的文件名是不区分大小写的，而表名称是区分大小写的。DCI_CASE配置变量决定了文件名称将如何转化为表名，设置为*lower*表示文件名将全部使用小写字母转化成表名；设置为*upper*表示文件名将全部使用大写字母转化成表名；设置为*ignore*表示不会全部以小写或者大写字母转化。DCI_CASE的默认设置是*lower*，如果您的文件名称是DBCS，需设置DCI_CASE为*ignore*。

➲ **示例**

```
DCI_CASE IGNORE
```

DCI_COMMIT_COUNT

DCI_COMMIT_COUNT配置变量表明在什么条件下执行COMMIT WORK操作。有两个值：0和<n>。

DCI_COMMIT_COUNT = 0

无自动提交（默认值）.ss

DCI_COMMIT_COUNT = <N>

在此条件下，DCI会等待直到执行的WRITE、REWRITE、AND DELETE操作数量等于<n>值时，才执行COMMIT WORK语句。此规则仅应用于文件以“输出”或者“唯一”模式被打开的情况下。

DCI_DATABASE

DCI_DATABASE用于指定DBMaster安装期间的数据库名称。

➲ **示例1**

若使用的数据库名称为DBMaster_Test，以下代码必须包含在配置文件中。

```
DCI_DATABASE DBMaster_Test
```

➲ **示例2**

有时，数据库名称无法提前得知，因此必须在运行时间内进行动态设置。例如，在COBOL程序里写入类似于下面的特殊代码，以下代码必须在首次执行OPEN语法之前执行。

```
CALL "DCI_SETENV" USING "DCI_DATABASE" , "DBMaster_Test"
```

➲ **示例3**

若想在不同的数据库下访问表，您可以使用DCI_DATABASE连接一个以上的数据库并且在不同的数据库之间动态转换。

```
* connect to DBNAME to access idx-1-file
CALL "DCI_SETENV" USING "DCI_DATABASE" "DBNAME"
....
```

```
open output idx-1-file
....
* connect to DCIDB to access idx-2-fileCALL "DCI_SETENV" USING "DCI_DATABASE"
"DCIDB"
....
open output idx-2-file

* to switch dynamically to DBNAME connection
CALL "DCI_SETENV" USING "DCI_DATABASE" "DBNAME"
close idx-1-file
...
```

DCI_DATE_CUTOFF

此变量使用两位数字的值设置两位数字的年份。两位数字的年份在程序中被理解为20世纪和21世纪的年份。

DCI_DATE_CUTOFF的默认值是20。这样，2000年将被添加到小于“20”的两位数年份中（或者您给这个变量赋予的任何值），因此会认为是21世纪的一部分。1900将被添加到大于“20”的两位数年份中（或者您给这个变量赋予的任何值），它就是20世纪的一部分。一个COBOL日期例如99/10/10将被转化为1999/10/10，00/02/12将被转化为2000/02/12。

DCI_DEFAULT_RULES

此变量是位于多重定义文件中的WHEN指示的默认管理方式。BEFORE语法表明当$WHEN条件匹配时会打开一个表。POST语法表明当用COBOL程序打开多重定义文件时，所有关系表将被打开。

可能的取值为：

POST 或者COBOL

BEFORE 或者 DBMS

DCI_DEFAULT_TABLESPACE

此变量用来设置新表即将保存到的默认表空间，指定的表空间必须已经存在于数据库中。如果此变量没有指定表空间，那么新表将在默认的用户表空间中创建。

DCI_DUPLICATE_CONNECTION

当使用不同的数据库连接，但相同的COBOL过程打开同一个表，并在相同的COBOL程序中打开相同的表，对同一条记录锁定两次时，可使用DCI_DUPLICATE_CONNECTION来获得一个锁。

默认值是关闭（0）。

➲ **示例**

允许一个COBOL程序在使用不同的数据库连接时获取一个表锁，请参照以下代码：

```
DCI_DUPLICATION_CONNECTION 1
```

DCI_GET_EDGE_DATES

若用户在DATE域输入一个low/high值时，DCI_SET_EDGE_DATE会指定显示的值。当用户在COBOL程序的DATE域输入low/high值时，例如输入00010101/99991231，日期将使用COBOL的low/high值00000000/99999999显示。当此变量被使用时，DATA域的low/high值将显示为数据库的low/hih值00010101/99991231。当DATE域是主键的一部分时此规则也适用，默认值为关闭。

➲ **语法**

下行必须在dci.cfg文件中添加。

```
DCI_GET_EDGE_DATES 1
```

DCI_INV_DATE

当一个不正确的日期格式已经写入数据库时，为了避免出现问题，此变量用来设定一个无效的日期（如2000/02/31）。默认变量是99991230（9999，十二月三十日）。

DCI_LOGFILE

此变量用来指定通过界面执行I/O操作的DCI日志文件的路径名。*dci_trace.log*日志文件保存在/tmp路径下以便于调试。日志文件的使用会降低DCI的性能。因此，除非确实需要，否则我们不建议您在配置文件中设置此变量。

➲ **示例**

日志文件登入Config.ini文件的示例：

```
DCI_LOGFILE /tmp/dci_trace.log
```

DCI_LOGIN

DCI_LOGIN变量允许规范的用户名连接到数据库系统，它没有默认值。因此，如果没有指定用户名，就不能登录。

通过DCI_LOGIN变量指定的用户名应该有RESOURCE权限或者更高的数据库权限。另外，用户需要拥有对现有数据表的访问操作权限。可以通过JDBA工具或者dmSQL来创建新用户。

注意 *关于创建新用户的详细信息，请参考JDBA用户使用手册或数据库管理员手册。*

➲ **示例**

用户登录名为JOHNDOE，并在Config.cfg文件中生成的示例：

```
DCI_LOGIN JOHNDOE
```

DCI_JULIAN_BASE_DATE

此变量与DATE指示一起用来设定Julian日期计算的基础日期。所使用的格式是YYYYMMDD，此变量默认值为January 1st, 1 AD。

此变量的用法之一是COBOL程序可以使用1850年以前的日期。这些日期保存在数据库中，它们设定DATE指示$XFD DATE = JJJJJJ（日期字段必须是相同的字符数）和设定DCI的配置变量DCI_JULIAN_BASE_DATE为18500101。

DCI_LOGTRACE

此变量用来设置不同的日志级别。

- 0：不记录
- 1：记录连接
- 2：记录i/o
- 3：记录满
- 4：记录内部调试

DCI_MAPPING

在DCI系统中，此变量使用时需结合特定的XFD文件名。通过这种方式，一个XFD能够和多个文件结合。“***pattern***”可以编制任意有效的文件名字符，它可能包括通配符“*”（表示任意字符），或者问号“?”（代表一个字符并且可以多次使用）。

➲ **语法**

```
DCI_MAPPING [pattern = base-xfd-name]  ...
```

➲ **示例1**

格式“CUST*1”和base-XFD-name “CUSTOMER”将使文件名如“CUST01”、“CUST001”、“CUST0001”和“CUST00001”与XFD "customer.XFD"结合。

```
DCI_MAPPING CUST*1=CUSTOMER ORD*=ORDER "ord cli*=ordcli"
```

➲ **示例2**

格式“CUST????”和base-XFD-name “CUST”将使文件名如“CUSTOMER”和“CUST0001”与XFD "cust.XFD"结合。

```
DCI_MAPPING  CUST????=CUST
```

DCI_MAX_ATTRS_PER_TABLE

一个DBMaster表最多有2000列，一个COBOL文件则超过2000个字段，所以不可能使所有表中的字段全部相对应。DCI提供了DCI_MAX_ATTRS_PER_TABLE配置变量来指定被分为两个或更多的独立表的域数量。生成表必须有唯一的表名，因此DCI会以下划线（_）字符的形式添加在表名之后，这些字符是连续的（A, B, C…）。

➲ **示例1**

一个COBOL文件有300个字段，如下所示：

```
SELECT FILENAME ASSIGN TO "customer"
```

➲ **语法**

必须在dci.cfg文件里添加下行内容：

```
DCI_MAX_ATTRS_PER_TABLE = 100.
```

➲ **示例2**

将生成以下名称的三个表：

```
customer_a
customer_b
customer_c
```

DCI_MAX_BUFFER_LENGTH

DCI_MAX_BUFFER_LENGTH用于将一个COBOL数据记录分为多个数据库表，类似于DCI_MAX_ATTRS_PER_ TABLE的功能。cutoff值通常通过缓冲区长度决定哪一个表将被分裂，默认值为32640。

➲ **示例1**

一个COBOL记录大小包括9000个数据字节，如下：

```
SELECT FILENAME ASSIGN TO "customer"
```

➲ **语法**

以下行必须在dci.cfg文件中添加：

```
DCI_MAX_BUFFER_LENGTH 3000
```

➲ **示例2**

将生成以下名称的三个表：

```
customer_a
customer_b
customer_c
```

DCI_MAX_DATE

此变量用来设立一个high-value日期来避免无效的日期写入数据库这样的错误，默认值为99991231（December 31st, 9999）。

DCI_MIN_DATE

此变量用来设立一个low-value，0或空日期。避免无效的日期错误地写入数据库。默认值为00010101 （January 1st, 1AD）。

DCI_NULL_ON_ILLEGAL_DATE

DCI_NULL_ON_ILLEGAL_DATE决定数据库中不合规定的COBOL数据在保存之前如何转化。值1使所有不合规定的数据（除了主键字段）在保存之前转化为空。值0（默认值）将引起以下转化：

- 违规的LOW-VALUES：保存为最小的可能值（0或-99999…）或DCI_MIN_DATE默认值。
- 违规的HIGH-VALUES：保存为最大的可能值（99999…）或DCI_MAX_DATE默认值。

- 违规的SPACES：保存为0（或在一个日期域中的DCI_MIN_DATE）。
- 违规的DATE值：保存为DCI_INV_DATE默认值。
- 违规的TIME：保存为DCI_INV_DATE默认值。
- 在主键域中的违规数据也被转化，包括配置变量的值。

DCI_PASSWD

一旦用户名由DCI_LOGIN变量指定，那么数据库账号将与其相结合。如果需要为数据库账号指定密码，可通过变量DCI_PASSWD来完成。

➲ **示例1**

若您想指定数据库账号的密码为SUPERVISOR，必须在配置文件中进行如下设置。

```
DCI_PASSWD SUPERVISOR
```

➲ **示例2**

密码也可以由执行程序的用户接受，这便拥有了更高的可靠性。DCI_PASSWD变量必须根据响应设定。

```
ACCEPT RESPONSE NO-ECHO.
CALL "DCI_SETENV" USING  "DCI_PASSWD" , RESPONSE.
```

在这种情况下为了读和写环境变量，您应该提供一个本地API来调用。

➲ **语法1**

此语法用于在COBOL程序中写或升级环境变量。

```
 CALL "DCI_SETENV" USING "environment variable", value.
```

➲ **语法2**

此语法用于在COBOL程序中读环境变量。

```
 CALL "DCI_GETENV" USING "environment variable", value.
```

DCI_STORAGE_CONVENTION

此变量用于设置COBOL存储协定，DBMaster支持以下四种类型：

DCI

选择IBM存储协定。它将与IBM COBOL兼容，也与其它COBOL版本相容，包括RM/COBOL-85，同时也与X/Open COBOL标准兼容。

DCM

选择MicroFocus存储协定。当Micro Focus "ASCII" sign-storage（此为Micro Focus默认）选项被使用时，它与Micro Focus COBOL兼容。

DCN

将引起一个不同的数字格式被使用。除非正COMP-3项使用"x0B"作为正值来代替"x0C"之外，此格式与选择"-DCI"选项时使用的格式相同。该选项与NCR COBOL是兼容的。

DCA

选择ISCOBOL存储协定，这是默认设置。此协定也与RM/COBOL（非RM/COBOL-85）生成的数据和之前的ISCOBOL版本兼容。

DCI_USEDIR_LEVEL

如果该变量设置为大于0，那么将使用除表名之外的目录。

➲ **示例1**

下行相当于：/usr/test/01/clients 01clients

```
DCI_USEDIR_LEVEL 1
```

➲ **示例2**

下行相当于：/usr/test/01/clients test01clients

```
DCI_USEDIR_LEVEL 2
```

➲ **示例3**

下行相当于：/usr/test/01/clients usrtest01clients

```
DCI_USEDIR_LEVEL 3
```

DCI_USER_PATH

当DCI寻找一个文件或多个文件时，变量DCI_USER_PATH将考虑到用户名或名称的规范性。用户参数可以是与文件相关联的句点（.）或者是系统的用户名。

➲ **语法**

```
DCI_USER_PATH user1 [user2] [user3] .
```

一个文件决定的OPEN语法类型将决定这种设置的结果。

OPEN 语法	DCI_USER_PATH	DCI SEARCH Sequence	结果
OPEN INPUT 或 OPEN I/O	是	1-列出USER_PATH的用户 前用户	第一个有效文件将被打开。
OPEN INPUT 或 OPEN I/O	否	与DCI_LOGIN关联的用户	第一个带有有效用户名/文件名的文件将被打开。
OPEN OUTPUT	是或否	不寻找用户	会产生一个与DCI_LOGIN相关联的新表。

图 6-1 OPEN 语句类型

DCI_XFDPATH

DCI_XFDPATH用于指定数据字典被储存的路径名，默认值是当前路径。

➲ **示例1**

为了在路径/usr/dbmaster/dictionaries下储存数据字典，可在配置文件中设置以下条目：

```
DCI_XFDPATH /usr/dbmaster/dictionaries
```

➲ **示例2**

若需要指定更多的路径，不同路径之间必须以空格分隔。

```
DCI_XFDPATH /usr/dbmaster/dictionaries /usr/dbmaster/dictionaries1
```

➲ **示例3**

在WIN-32环境下，“嵌入的空格”可使用双引号指定。

```
DCI_XFDPATH c:\tmp\xfdlist "c:\my folder with space\xfdlist"
```

DCI_XML_XFD

DCI_XML_XFD设置为1表示DCI运行时需要使用ACUCOBOL XML 格式的XFD文件，默认值为0表示用户将使用旧的XFD格式，该格式由之前的ACUCOBOL编译器产生或者当ACUCOBOL用户通过–Fx3选项来编译他们的COBOL程序

因为ACUCOBOL-GT编译器不支持XFD的一些语法，如XML格式中的"$XFD COMMENT directive"，所以当用户想在COBOL程序中使用相关语法时，不建议用户采用XML格式。

➲ **示例**

第4.2章节中不支持下列语法：

```
$XFD COMMENT  DCI SERIAL  n directive
$XFD COMMENT DCI COBTRIGGER Directive
$XFD COMMENT Directive
$XFD COMMENT DCI SPLIT
```

*<filename>*_RULES

多元定义文件提供的默认管理方法，以实际的文件名取代<filename>。

➲ **示例**

当执行以下命令时，除CLIENT文件以外的所有文件将使用POST规则。

```
DCI_DEFAULT_RULES       POST
CLIENT_RULES            BEFORE
```

DCI TABLE CACHE变量

默认情况下，DCI会将客户数据预读到缓冲区，这样就会减少客户端/服务器的网络通信量。默认的最大预读缓存是小于8kb/（记录大小）或者5条记录。

用户程序可能只读取一个小表并只读取小于8kb/（记录大小）的一些记录。例如，对于一个平均记录大小为20个字节和共1000条记录的表，DBMaster将可能读取大约400条记录（8kb/20）但是用户程序可能只读取4或5条记录然后再次调用START语句。这种情况下，可设置以下变量来减少缓存大小并提高性能。当使用这些变量时要仔细考虑应用程序和数据行为，否则有可能降低网络速度并导致性能降低。

在DCI_CONFIG文件中可设置以下三个DCI_CACHE：

- DCI_DEFAULT_CACHE_START-设置START或READ首次向缓存读取的记录数。默认最大值是8kb/（记录大小）或5条记录。
- DCI_DEFAULT_CACHE_NEXT –在首次向缓存读取记录后，为START或READ设置下一次读取的记录。默认值是8kb/（记录大小）或5条记录。
- DCI_DEFAULT_CACHE_PREV –在首次向缓存读取记录后，为缓存前一条记录而设置读取记录。

默认设置为DCI_DEFAULT_CACHE_NEXT/2。

DCI_CONFIG中设置的变量将影响用户应用程序里的所有表。

➲ **示例**

```
DCI_DEFAULT_CACHE_START        10
DCI_DEFAULT_CACHE_NEXT         10
DCI_DEFAULT_CACHE_PREV         5
```

DCI_TABLESPACE

DCI_TABLESPACE允许您指定在哪一个表空间里创建表，它常与通配符一起应用。首次创建表是非常重要的，一旦有表存在，DCI将不会监视此变量的值。

➲ **示例1**

下例说明您如何在表空间tbs1中创建用户表：

```
DCI_TABLESPACE customer=tbs1
```

➲ **示例2**

下例说明您在表空间tbs1中创建以cust开头的所有表：

```
DCI_TABLESPACE cust*=tbs1
```

DCI_AUTOMATIC_SCHEMA_ADJUST

当XFD与表模式不一致时，此变量会命令DCI改变表模式的定义。此变量与分开的表（列数＞2000，记录大小超过32KB-除去BLOB字段）是不兼容的。

可能的变量值：

- 0　默认，无作用
- 1　为表添加新的字段，删除非XFD内的字段
- 2　为表添加新的字段，但是不删除非XFD内的字段

DCI_INCLUDE

此变量允许包含一个DCI_CONFIG附加的文件。操作如COBOL COPY语句一样，允许定义更多的复杂配置。

➲ 示例

```
DCI_INCLUDE /etc/generic_dci_config
```

DCI_IGNORE_MAX_BUFFER_LENGTH

此变量忽略DCI_MAX_BUFFER_LENGTH值。当记录长度大于32k时它将拆分此表，默认设置为off。

DCI_NULL_DATE

当DCI用该变量写一个日期域时它将写入NULL值，当DCI读取一个为NULL值的日期时，它会将DCI_NULL_DATE返回到COBOL程序中。

DCI_NULL_ON_MIN_DATE

此变量设置为1时将发生以下动作。当一个COBOL程序在DATE域写入0值时，这个值将在数据库中保存为NULL。同样地，当从数据库读取一个NULL值时，COBOL FD将是0。

DCI_DB_MAP

此变量用于在不同路径下映射文件来作为数据库不同的表，详细信息请参考映射多元数据库。

DCI_VARCHAR

此变量设置为1时将发生以下动作：当一个COBOL程序创建一个新表时（通过OPEN OUTPUT动词），所有以CHAR类型创建的域将变为VARCHAR类型。

DCI_GRANT_ON_OUTPUT

这个新的选项允许您在创建表期间（OPEN OUTPUT）指明对表允许的操作。

➲ **示例**

```
DCI_GRANT_ON_OUTPUT user1=SELECT
DCI_GRANT_ON_OUTPUT user2=SELECT,INSERT,UPDATE
```

在打开输出后，user1可以从表中选择数据，user2可以选择和更改数据。

7 DCI函数

本章列出了可以被COBOL程序调用的DCI函数。要想使用这些函数，用户必须将这些函数添加到sub85.c中，然后重新创建DCI运行环境。

7.1 调用DCI函数

您可以在您的COBOL程序中添加如下代码来调用那些DCI函数：

```
CALL "dci_function_name" USING variable [, variable, ...]
```

DCI_SETENV

该函数用来编写或更新环境变量。

➲ **语法**

```
 CALL "DCI_SETENV" USING "environment variable", value
```

➲ **示例**

```
call "DCI_SETENV" using  "DCI_DATABASE" , "DBNAME"
```

DCI_GETENV

该函数用来读取环境变量。

➲ **语法**

```
CALL "DCI_GETENV" USING "environment variable", variable
```

➲ **示例**

```
CALL "DCI_GETENV" USING "DCI_DATABASE", ws_dci_database
```

DCI_DISCONNECT

该函数用来断开数据库连接。

➲ **示例1**

如果COBOL程序中只有一个连接，可使用如下代码断开连接。

```
CALL "DCI_DISCONNECT".
```

➲ **示例2**

如果COBOL程序中存在多个连接，可使用如下代码断开一个指定连接。

```
CALL "DCI_DISCONNECT" USING "DBSAMPLE5"
```

DCI_GET_TABLE_NAME

该函数用来获取传递过来的COBOL名称的表名称。（通常并不是很快就能获取有效的表名称，因为诸如XFD WHEN ... TABLENAME的情况会有很多操作）。

```
CALL "DCI_GET_TABLE_NAME" USING ws-filename, ws-dci-file-name
```

DCI_SET_TABLE_CACHE

对于表在START或READ语句前设置变量，该函数用来动态更改缓存设置。

➲ **示例**

```
…
WORKING-STORAGE SECTION.
       01 CACHE-START PIC 9(5) VALUE 10.
       01 CACHE-NEXT  PIC 9(5) VALUE 20.
       01 CACHE-PREV  PIC 9(5) VALUE 30.
…
PROCEDURE DIVISION.
      OPEN INPUT IDX-1-FILE
           MOVE SPACES TO IDX-1-KEY
           CALL "DCI_SET_TABLE_CACHE" USING CACHE-START
                                            CACHE-NEXT
                                            CACHE-PREV
           START IDX-1-FILE KEY IS NOT LESS IDX-1-KEY.
           PERFORM VARYING IND FROM 1 BY 1 UNTIL IND = 10000
              READ IDX-1-FILE NEXT AT END EXIT PERFORM END-READ
              DISPLAY IND AT 0101
           END-PERFORM
           CLOSE IDX-1-FILE
```

DCI_BLOB_ERROR

在调用了DCI_BLOB_GET或DCI_BLOB_PUT后，可用该函数来获取错误信息。

➲ 示例

```
     working-storage section.
      77   BLOB-ERROR-ERRNO pic S9(4) COMP-5.
      77   BLOB-ERROR-INT-ERRNO pic S9(4) COMP-5.

  PROCEDURE DIVISION.
 CALL "DCI_BLOB_ERROR" USING BLOB-ERROR-ERRNO
                             BLOB-ERROR-INT-ERRNO
       DISPLAY "BLOB-ERROR-ERRNO=" BLOB-ERROR-ERRNO.
       DISPLAY "BLOB-ERROR-INT-ERRNO=" BLOB-ERROR-INT-ERRNO.
```

DCI_BLOB_GET

该数据可以使用户更加有效地在COBOL程序中使用BLOB数据。通过命令DCI_BLOB_GET，您可以快速有效地访问BLOB。在使用DCI_BLOB_GET时，您需要参照以下规则：

- 用户表中必须包含BLOB（long varchar/long varbinary）数据类型
- 在COBOL FD中用户不能设置BLOB类型字段
- 用户只能在READ、READ NEXT或者READ PREVIOUS命令后使用DCI_BLOB_GET命令

➲ 示例

用户通过下面命令创建表：

```
CREATE TABLE BLOBTB (
         SB_CODCLI  char(8),
         SB_PROG    SERIAL,
         IL_BLOB    LONG VARBINARY,
         PRIMARY KEY ("sb_codcli")) LOCK MODE ROW NOCACHE;
```

下列给出在实际应用中COBOL使用DCI_BLOB_GET的例子。

```
      identification division.
```

```
       program-id. blobtb.
       date-written.
       remarks.
       environment division.
       input-output section.
       file-control.
           SELECT BLOBTB ASSIGN TO RANDOM, "BLOBTB"
                       ORGANIZATION IS INDEXED
                       ACCESS IS DYNAMIC
                       FILE STATUS IS I-O-STATUS
                       RECORD KEY  IS SB-CODCLI.
      *=============================================================*
       data division.
       file section.
       FD  BLOBTB.
       01  SB-RECORD.
           03  SB-CODCLI         PIC X(8).
           03  SB-PROG           PIC S9(9) COMP-5.
      *=============================================================*
       working-storage section.
       77   I-O-STATUS pic xx.
       77   BLOB-ERROR-ERRNO pic S9(4) COMP-5.
77   BLOB-ERROR-INT-ERRNO pic S9(4) COMP-5.

procedure division.
main.
              open i-o blobtb
              initialize sb-record.
              READ blobtb next.
              CALL "DCI_BLOB_GET" USING "il_blob" "laecopy.bmp" 1.
              CALL "DCI_BLOB_ERROR" USIG BLOB-ERROR-ERRNO
                                         BLOB-ERROR-INT-ERRNO
              DISPLAY "BLOB-ERROR-ERRNO=" BLOB-ERROR-ERRNO.
              DISPLAY "BLOB-ERROR-INT-ERRNO=" BLOB-ERROR-INT-ERRNO.
              DISPLAY "SB-CODCLI=" SB-CODCLI.
              DISPLAY "SB-PROG=" SB-PROG.
              close blobtb.
              ACCEPT OMITTED.
              stop run.
```

DCI_BLOB_PUT

该数据可以使用户更加有效地在COBOL程序中使用BLOB数据。使用DCI_BLOB_PUT命令，您可以向BLOB中插入数据。在使用DCI_BLOB_PUT时，您需要参照以下规则：

- 用户表中必须包含BLOB（long varchar/long varbinary）数据类型
- 在COBOL FD中用户不能设置BLOB类型字段
- 用户只能在WRITE或REWRITE命令前调用DCI_BLOB_PUT命令
- 如果用户没有在WRITE语句前调用DCI_BLOB_PUT，那么默认值将插入到blob字段。
- 如果用户没有在REWRITE语句前调用DCI_BLOB_PUT，那么blob字段将不会被更新。

➲ **示例**

下列给出在实际应用中COBOL使用DCI_BLOB_PUT的例子。

首先，用户创建表：

```
CREATE TABLE BLOBTB (
          SB_CODCLI  char(8),
          SB_PROG    SERIAL,
          IL_BLOB    LONG VARBINARY,
          PRIMARY KEY ("sb_codcli")) LOCK MODE ROW NOCACHE;
```

一旦用户创建了表，继续下列操作：

```
identification division.
       program-id. blobtb.
       date-written.
       remarks.
       environment division.
       input-output section.
       file-control.
           SELECT BLOBTB ASSIGN TO RANDOM, "BLOBTB"
                       ORGANIZATION IS INDEXED
                       ACCESS IS DYNAMIC
                       FILE STATUS IS I-O-STATUS
```

```
                    RECORD KEY  IS SB-CODCLI.
     *================================================================*
      data division.
      file section.
      FD  BLOBTB.
      01  SB-RECORD.
          03  SB-CODCLI         PIC X(8).
          03  SB-PROG           PIC S9(9) COMP-5.
     *================================================================*
      working-storage section.
      77   I-O-STATUS pic xx.
      77   BLOB-ERROR-ERRNO pic S9(4) COMP-5.
77   BLOB-ERROR-INT-ERRNO pic S9(4) COMP-5.

procedure division.
main.
             open i-o blobtb
             move "AAAAAAAA" TO SB-CODCLI.
             move 0 TO SB-PROG.
             CALL "DCI_BLOB_PUT" USING "il_blob" "laetitia.bmp".
             WRITE SB-RECORD.
             close blobtb.
             ACCEPT OMITTED.
             stop run.
```

DCI_GET_TABLE_SERIAL_VALUE

该函数用于WRITE语句后获取序列号。

➲ **示例**

```
     FD  SERIALTB.
      01  SB-RECORD.
          03  SB-CODCLI         PIC X(8).
     $XFD COMMENT dci serial
          03  SB-PROG           PIC S9(9) COMP-5.
      working-storage section.
      77   I-O-STATUS pic xx.
      77   SERIAL-NUM pic S9(9) COMP-5.
```

```
procedure division.
main.
       open i-o serialtb
       move "AAAAAAAA" TO SB-CODCLI.
       move 0 TO SB-PROG.
       WRITE SB-RECORD.
       CALL "DCI_GET_TABLE_SERIAL_VALUE" USING SERIAL-NUM.
       DISPLAY "SERIAL-NUM=" SERIAL-NUM.
```

DCI_FREE_XFD

该函数用来清除DCI保存在缓存中的XFD图片，对于表已经被连接打开而需要重新加载XFD非常有帮助。

```
CALL "DCI_FREE_XFD"
```

DCI_UNLOAD_CONFIG

该函数用来卸载当前配置，然后通过调用DCI_SETENV创建一个新的配置，这将在ThinClient环境中非常有帮助。

```
call "DCI_UNLOAD_CONFIG"
```

8 COBOL转换

在转换时DCI会强迫使用事务，所有的I/O操作都是使用事务完成的。DCI设置AUTOCOMMIT off，并管理DBMaster事务记录变化。DCI完全支持诸如START TRANSACTION、COMMIT/ROLLBACK TRANSACTION等COBOL事务语句。

DCI不支持记录加密，记录压缩和可选择的排序序列。如果代码中包含这些选项，它们将被忽略。DCI同样不支持在XFD数据定义中使用“P” PICture编辑函数，并且所有文件名将被转换成小写字母。

DBMaster 数据库设置	范围限制
索引键大小	4000
每个键的字段数	32
CHAR字段长度	32640 字节
同时RDBMS连接数	4800
字段名称字符数	128
单个进程可以打开的数据库表个数	256

图 8-1 DBMaster数据库设置范围限制表

8.1 使用特殊指示

DBMaster可以使用相同类别或返回次序作为Vision文件系统，但需要在每一个主键字段前标注BINARY指示，并且要包含带有正负号的数值数据。在主键字段中High值和low值可以创造出很多复杂因素。

DBMaster OID，VARCHAR（size)和FILE数据类型目前均不支持特殊指示。

DBMaster 数据类型	指示
DATE	使用XFD DATE
TIME	使用XFD DATE
TIMESTAMP	使用XFD DATE
LONGVARCHAR	使用XFD VAR-LENGH
LONGVARBINARY	使用XFD VAR-LENGH*
BINARY	使用XFD BINARY
SERIAL	使用XFD COMMENT DCI SERIAL

图 8-2 数据类型支持的特殊指示

8.2 数据类型映射

DCI会在创建数据库表的字段时，为COBOL数据类型建立它认为最好的匹配。COBOL数据类型中的任何数据都可以在数据库字段中使用，指定的XFD指示将最先被检查。

COBOL	DBMaster		COBOL	DBMaster
9(1-4)	SMALLINT		9(5-9) comp-4	INTEGER
9(5-9)	INTEGER		9(10-18) comp-4	DECIMAL(10-18)
9(10-18)	DECIMAL(10-18)		9(1-4) comp-5	SMALLINT
s9(1-4)	SMALLINT		9(5-10) comp-5	DECIMAL(10)
s9(5-9)	INTEGER		s9(1-4) comp-5	SMALLINT
s9(10-18	DECIMAL(10-18)		s9(5-10) comp-5	DECIMAL(10)
9(n) comp-1 n (1-17)	INTEGER		9(1-4) comp-6	SMALLINT
s9(n) comp-1 n (1-17)	INTEGER		9(5-9) comp-6	INTEGER
9(1-4) comp-2	SMALLINT		9(10-18) comp-6	DECIMAL(10-18)
9(5-9) comp-2	INTEGER		s9(1-4) comp-6	SMALLINT
9(10-18) comp-2	DECIMAL(10-18)		s9(5-9) comp-6	INTEGER
s9(1-4) comp-2	SMALLINT		s9(10-18) comp-6	DECIMAL(10-18)
s9(5-9) comp-2	INTEGER		signed-short	SMALLINT
s9(10-18) comp-2	DECIMAL(10-18)		unsigned-short	SMALLINT
9(1-4) comp-3	SMALLINT		signed-int	CHAR(10)

9(5-9) comp-3	INTEGER		unsigned-int	CHAR(10)
9(10-18) comp-3	DECIMAL(10-18)		signed-long	CHAR(18)
s9(1-4) comp-3	SMALLINT		unsigned-long	CHAR(18)
s9(5-9) comp-3	INTEGER		float	FLOAT
s9(10-18) comp-3	DECIMAL(10-18)		Double	DOUBLE
9(1-4) comp-4	SMALLINT		PIC x(n)	CHAR(n) n 1-max column length

图 8-3 COBOL到DBMaster数据类型转换表

8.3 数据类型映射

DCI从数据库中读取数据是通过执行类似COBOL的MOVE动作实现的，该动作从原数据类型迁移为COBOL数据类型（大部分都有CHAR表示法，所以您可以使用dmSQL命令行工具来显示它们）。

不必担心数据库数据类型向COBOL数据库类型映射的正确性。考虑到数据库数据类型有CHAR表示法，可以为每个字段使用PIC X（nn）。PIC 9(9)是针对带有INTEGER数据类型的数据库较为贴切的COBOL映射。您对数据库类型了解的越多，就越能灵活地查找映射的COBOL类型。例如，DBMaster数据库的一个字段中仅包含0到99（0-99）的数，那么映射PIC99便足够了。

COMP类型在COBOL数据使用中几乎毫无用处，因此程序员基本不会选择COMP类型。BINARY数据类型因为相对于COBOL来说是外来类型，所以通常会被不加变化地重写。不管怎样，通过对BINARY字段贴切的分析，您可以找到不同的解决方案。数据类型DECIMAL, NUMERIC, DATE和TIMESTAMP没有确切的COBOL映射，它们将以字符形式被数据库返回。所以，最好的对等COBOL数据类型将会是USAGE DISPLAY。

下表举例说明了数据库数据类型和COBOL数据类型之间的最好映射：

DBMaster	COBOL		DBMaster	COBOL
SMALLINT	9(1-4)		INTEGER	9(5-9) comp-4
INTEGER	9(5-9)		DECIMAL(10-18)	9(10-18) comp-4
DECIMAL(10-18)	9(10-18)		SMALLINT	9(1-4) comp-5
SMALLINT	s9(1-4)		DECIMAL(10)	9(5-10) comp-5
INTEGER	s9(5-9)		SMALLINT	s9(1-4) comp-5
DECIMAL(10-18)	s9(10-18)		DECIMAL(10)	s9(5-10) comp-5
INTEGER	9(n) comp-1 n (1-17)		SMALLINT	9(1-4) comp-6
INTEGER	s9(n) comp-1 n (1-17)		INTEGER	9(5-9) comp-6
SMALLINT	9(1-4) comp-2		DECIMAL(10-18)	9(10-18) comp-6
INTEGER	9(5-9) comp-2		SMALLINT	s9(1-4) comp-6
DECIMAL(10-18)	9(10-18) comp-2		INTEGER	s9(5-9) comp-6
SMALLINT	s9(1-4) comp-2		DECIMAL(10-18)	s9(10-18) comp-6
INTEGER	s9(5-9) comp-2		SMALLINT	signed-short
DECIMAL(10-18)	s9(10-18) comp-2		SMALLINT	unsigned-short

SMALLINT	9(1-4) comp-3		CHAR(10)	signed-int
INTEGER	9(5-9) comp-3		CHAR(10)	unsigned-int
DECIMAL(10-18)	9(10-18) comp-3		CHAR(18)	signed-long
SMALLINT	s9(1-4) comp-3		CHAR(18)	unsigned-long
INTEGER	s9(5-9) comp-3		FLOAT	float
DECIMAL(10-18)	s9(10-18) comp-3		DOUBLE	Double
SMALLINT	9(1-4) comp-4		CHAR(n) n 1-max column length	PIC x(n)

图 8-4 DBMaster向COBOL数据类型转换表

8.4 处理动态错误

动态错误的格式是“9D,xx”，其中“9D”表示文件系统错误（据变量FILE STATUS报告的），而“xx”表示二级错误代码。

错误	定义	说明	解决方案
9D,01	在字典文件中存在读取错误。	在读取XFD文件时发生错误，XFD文件被破坏。	使用-Fx重新编译，重新创建字典文件。
9D,02	字典文件破损，无法读取字典文件。	COBOL文件的字典文件发生破损。	使用-Fx重新编译，重新创建字典文件。
9D,03	找不到字典文件（.xfd）。	COBOL文件的字典文件发生破损。	在配置文件中为变量DCI_XFDPATH指定正确的路径（必要时使用-Fx重新编译）
9D,04	主键中字段过多。	主键中超过16个字段。	检查主键的定义，对不正确的主键重新构造，使用-Fx重新编译。
9D,12	DBMaster 库函数中发生意外错误。	一个DBMaster库函数返回意外错误。	
9D,13	“xxx”变量大小违例。	FD中的基础数据项大于255字节。	
9D,13	“xxx”变量的数据	没有DBMaster	

	类型违例。	类型匹配使用中的数据。	
9D,14	存在相同的表名称。	列出表名称时，存在相同的表名称。	

图 8-5 DCI 二级错误表

8.5 处理标准SQL错误

在DBMaster中使用DCI时也可能产生一些标准的SQL错误，错误代码和提示会因数据库的不同而不同。

Number	定义	说明	解决方案
9D, 6523,6018	无效的字段名称或保留字。	字段名称使用的单词是数据库中的保留字	在数据库保留字列表中比较CREATE TABLE的文件踪迹， 在无效字段的FD字段前使用NAME指示，并重新编译创建新的XFD文件。
9D, 1310	日志满，命令回滚到内部保存点		在COBO程序中添加代码"start TRANSACTION/commit/rollback"，或在DCI配置文件中设置DCI_COMMIT_COUNT
9D, 5503	无效的主键名	表中没有索引	使用正确的索引名称和字段创建索引
9D, 5504	不能使用主变量		在sql命令中用户不能使用主变量
9D, 5508	没有INSERT/UPDATE/DELETE权限	对于没有INSERT/UPDATE/DELETE权限的表，用户不能进行I-O操作	OPEN INPUT该表
9D, 5512	不能进行		在sql命令中，用

	select查询		户不能输入select语句
9D,5513	DCI连接时，客户端-服务器版本不匹配	用户的DCI运行时间要新于dmserver	在运行新DCI之前，用户需要更新dmserver
9D, 5514	无效的字段数	COBOL FD字段数大于表的字段数	用户需要检查FD字段数和表的字段数
9D, 5515	无效的XFD字段名称或数据类型和长度不匹配	COBOL FC字段名或字段类型与表中的定义不匹配	对比FD和表定义，通过修改表或COBOL FD来修订该问题。
9D, 5518	DCI blob数据为空	用户从字段中获取blob数据，然而数据为空	
9D, 5519	DCI blob文件不存在		用户需要确认blob文件是否存在

图 8-6 标准SQL错误表

8.6 转换Vision文件

DCI提供了范例程序来将COBOL文件转换为关系数据库中的表。在使用DCI_MIGRATE程序之前，首先要转换Vision文件，并需要Vision文件的XFD数据字典。另外，必须安装要链接到DCI的ACUCOBOL运行系统4.3或更高版本，并准备好DCI_MIGRATE对象程序。

使用DCI_Migrate

这是一个一般用途的程序，用来将COBOL vision文件转换成DBMaster表。为保证运行正确，必须定义最基本的DCI配置（DCI_LOGIN、DCI_DATABASE、DCI_PASSWD等），保证.XFD文件名与dbm_table_name的匹配，或使用DCI_MAPPING来指定名称和路径。

DCI_MIGRATE通过DCI读取vision文件然后写入DBMaster记录。另外在移植后，它会再次读取vision记录，与DBMaster记录进行比较以检查所有记录是否正确。

DCI_MIGRATE程序会提供如下报告：

- 成功读取记录数
- 成功写入记录数
- 未成功读取记录数
- 未成功写入记录数
- 成功比对记录数
- 未成功比对记录数

DCI_MIGRATE选项	结果
--help	显示在线帮助。
--nowait	交互式模式下，不等待用户确认。
--noverify	跳过确认步骤。
--nomigrate	跳过迁移步骤。
--visdbm	将vision文件转换成DBMaster表（默认）。
--dbmvis	将DBMaster表转换为vision文件。

图 8-7 DCI_MMIGRATE选项结果表

➲ **语法1**

*vision_file_name*是将要被转换的Vision文件名称，*dbm_table_name*是DBMaster表名称。

```
runcbl DCI_MIGRATE vision_file_name dbm_table_name [options]
```

➲ **语法2**

设置环境变量DCI_MIGRATE为“yes”可以开启报告，然后报告会增补一个名为“dbm_table_name.log”的文件。

```
DCI_MIGRATE = yes
```

➲ **语法3**

在DCI_MIGRATE设置中添加“dump”可以取消未成功的操作（空格会被当做隔离符对待，Log文件名称中不能包含空格）。

```
DCI_MIGRATE = yes dump
```

➲ **语法4**

将名为DCIMIGRATE_COMMIT_COUNT的环境变量提交数从100更改为指定的数值。

```
DCIMIGRATE_COMMIT_COUNT = 200
```