DBMaster

DCI ユーザーガイド

SYSCOM

CASEMaker Inc./Corporate Headquarters

1680 Civic Center Drive
Santa Clara, CA 95050, U.S.A.

Contact Information:
CASEMaker US Division
E-mail : info@casemaker.com
Europe Division
E-mail : casemaker.europe@casemaker.com
Asia Division
E-mail : casemaker.asia@casemaker.com(Taiwan)
E-mail : info@casemaker.co.jp(Japan)

www.casemaker.com
www.casemaker.com/support

Document No. 645049-236326/DBM54J-M09302015-DCIU
発行日:2015-09-30

本文には記されていない新しい機能についての説明は、CASEMakerのDBMasterをインストールしてからREADME.TXTを読んでください。

登録商標
CASEMaker、CASEMakerのロゴは、CASEMaker社の商標または登録商標です。
DBMasterは、Syscom Computer Engineering社の商標または登録商標です。
Microsoft、MS-DOS、Windows、Windows NTは、Microsoft社の商標または登録商標です。
UNIXは、The Open Groupの商標または登録商標です。
ANSIは、American National Standards Institute, Incの商標または登録商標です。

ここで使用されているその他の製品名は、その所有者の商標または登録商標で、情報として記述しているだけです。SQLは、工業用語であって、いかなる企業、企業集団、組織、組織集団の所有物でもありません。

注意事項
本書で記述されるソフトウェアは、ソフトウェアと共に提供される使用許諾書に基づきます。

保証については、ご利用の販売店にお問い合わせ下さい。販売店は、特定用途への本コンピュータ製品の商品性や適合性について、代表または保証しません。販売店は、突然の衝撃、過度の熱、冷気、湿度等の外的な要因による本コンピュータ製品へ生じたいかなる損害に対しても責任を負いません。不正な電圧や不適合なハードウェアやソフトウェアによってもたらされた損失や損害も同様です。

本書の記載情報は、その内容について十分精査していますが、その誤りについて責任を負うものではありません。本書は、事前の通知無く変更することがあります。

目次

1 はじめに

本書は、信頼性のあるCOBOLプログラムと、柔軟性と効率性のあるリレーショナル・データベース管理システム(RDBMS)の統合を試みているソフトウェアの開発者向けに書かれています。本書は、DCIプログラムは、DBMasterデータベース・エンジンを使って、COBOLのデータを効率的に管理、調整を行うために設計されたDBMaster COBOL Interface (DCI)をどのように使用するかを体系的に解説します。

DCIは、COBOLプログラムとDBMasterの間の通信チャネルを提供します。DBMaster COBOL Interface (DCI)は、COBOLプログラムにDBMasterリレーショナル・データベースに格納される情報に効率的にアクセスすることができるようにします。データを格納するために、COBOLプログラムは、通常標準的なB-TREEファイルを使用します。B-TREE ファイルストアデータに格納される情報には、これまでREAD、WRITE、REWRITEのような標準的なCOBOL I/O文を通じてアクセスしました。

COBOLプログラムは、DBMaster RDBMSに格納されたデータにもアクセスすることができます。従来、COBOLプログラマーは、COBOLのソースコードにSQL文を入れ込む、埋め込みSQLと呼ばれるテクニックを使っていました。ソースコードをコンパイルする前に、特別なプリコンパイラがSQL文をデータベース・エンジンの「コール」に変換します。これらのコールは、ランタイム時に実行され、DBMaster RDBMSにアクセスします。

この技術は、COBOLプログラムを使ってデータベースに情報を格納することにおいて優れた解決策ではありますが、欠点もあります。第一に、この方法は、COBOLプログラマーがSQL言語を熟知していることを前提として

います。次に、この方法で記述されたプログラムは、汎用性がありません—B-TREEファイルとDBMaster RDBMSのどちらでも使用できるわけではありません。更に、SQL構文はデータベース毎に異なります。これは、DBMaster RDBMS向けのSQL文を埋め込んだCOBOLプログラムは、他のデータベースで使用できないことを意味します。最後に、埋め込みSQLを、既存のプログラムで採用することは容易ではありません。実際、埋め込みSQLには、アプリケーションに作業ストレージ、データ・ストレージ、各I/O文のロジックの書き直し等に大規模な追加を含む、リエンジニアリングが必要とされます。

埋め込まれたSQLには修正があります。サプライヤによっては、COBOLからデータベース・インターフェースが開発されています。これらのインターフェースを実行すると、COBOL I/O命令文をSQL文に即時に変換します。このように、COBOLプログラマーはSQLを熟知している必要が無く、COBOLプログラムは、移植性が良くなります。但し、ここでパフォーマンスに問題があります。

実際、SQLにはCOBOL I/O文以外の目的があります。SQLは、一般的な定義からほとんど全ての組み合わせのデータをも見つけることができるSETベースの、アドホック言語です。対照的に、COBOL B-TREE(又は他のデータ構造コールは、縦横無尽に定義されたキーやナビゲーション・ロジックa或いは両方ともを介して、直接データ・アクセスを行います。しかし、トランザクションが多く、パフォーマンスに影響されやすいCOBOLアプリケーションに、SQL-ベースのI/Oを経由して排他的に操作させることは、しばしば不適切な方法となります。

このため、CASEMakerのCOBOLインターフェース製品、DCIはSQLを採用していません。DCIはCOBOL自身が他のユーザーが置き換えることができるCOBOLファイル・システムにアクセスする方法と相似した方法で、直接的で縦横にデータ・ストレージ・アクセスする手段を提供します。DCIは、COBOL プログラムとDBMasterのファイル・システムの間にシームレスなインターフェースを提供します。アプリケーションとデータベース間で変換される情報は、エンドユーザーも目に見えるものです。完全なSQLベースのファイルとデータ・ストレージ・アクセスをも備えている場合、デスクトップ意思決定支援システム(DSS)、データ・ウェアハウスと4GLア

プリケーション用に、DBMasterは、RDBMSの信頼性と強力さと同じく、このような機能を提供します。

CASEMakerのデータベースとDCI製品は、4GLの強力さと、SQL-ベースのデータベースのアクセスとレポートのアドホックな柔軟性を持ったナビゲーション・データ構造を組み合わせています。

1.1 その他のマニュアル

DBMasterには、本マニュアル以外にも多くのユーザーガイドや参照編があります。特定のテーマについて、以下の書籍をご覧下さい。

- DBMasterの能力と機能性についての概要は、*［DBMaster入門編］*を参照して下さい。
- DBMasterの設計、管理、保守についての詳細は、*「データベース管理者参照編」*をご覧下さい。
- DBMasterの管理についての詳細は、*「JServer Managerユーザーガイド」*をご覧下さい。
- DBMasterの環境設定についての詳細は、*「JConfiguration Tool参照編」*をご覧下さい。
- DBMasterの機能についての詳細は、*「JDBA Toolユーザーガイド」*をご覧下さい。
- DBMasterで使用しているdmSQLのインターフェースについての詳細は、*「dmSQLユーザーガイド」*をご覧下さい。
- DBMasterで採用しているSQL言語についての詳細は、*「SQL文と関数参照編」*をご覧下さい。
- ESQLプログラムについての詳細は、*「ESQL/Cプログラマー参照編」*をご覧下さい。
- ODBCとJDBCプログラムについての詳細は、*「ODBCプログラマー参照編」*と*「JDBCプログラマー参照編」*をご覧下さい。

- エラーと警告メッセージについての詳細は、*「エラー・メッセージ参照編」*をご覧下さい。

1.2 字体の規則

本書は、標準の字体規則を使用しているので、簡単かつ明確に読むことができます。

斜体	斜体は、ユーザー名や表名のような特定の情報を表します。斜体の文字そのものを入力せず、実際に使用する名前をそこに置き換えてください。斜体は、新しく登場した用語や文字を強調する場合にも使用します。
太字	太字は、ファイル名、データベース名、表名、カラム名、関数名やその他同様なケースに使用します。操作の手順においてメニューのコマンドを強調する場合にも、使用します。
キーワード	文中で使用するSQL言語のキーワードは、すべて英大文字で表現します。
小さい英大文字	小さい英大文字は、キーボードのキーを示します。２つのキー間のプラス記号 (+) は、最初のキーを押したまま次のキーを押すことを示します。キーの間のコンマ(,)は、最初のキーを放してから次のキーを押すことを示します。
注	重要な情報を意味します。
➲ **プロシージャ**	一連の手順や連続的な事項を表します。ほとんどの作業は、この書式で解説されます。ユーザーが行う論理的な処理の順序です。
➲ **例**	解説をよりわかりやすくするために与えられる例です。一般的に画面に表示されるテキストと共に表示されます。
コマンドライン	画面に表示されるテキストを意味します。この書式は、一般的にdmSQLコマンドやdmconfig.iniファイルの内容の入/出力を表示します。

2 DCIの基礎

この章では、DBMasterのためのDCI環境のセットアップと設定に関する情報とデモ・プログラムの使用についての情報を提供してDCIの基本機能を紹介します。

この章では、下記のトピックスについて解説します。

- ソフトウェアとハードウェアの必要動作環境
- UNIXとWindowsプラットフォームのステップ毎のセットアップ方法
- DBMaster用にDCIを環境設定するオプション
- DCIデモ・プログラムの指導

2.1 DCIについて

従来のCOBOLファイル・システムとデータベースのいずれも、データを格納する点では同じですが、これらの間には大きな違いがあります。一般的に、データベースの方がより強力で、従来のファイル・システムより信頼性があります。更に、ソフトウェアやハードウェアの障害から効率的にデータを回復することができるシステムといえます。加えてDBMasterのRDBMSには、データの整合性を確実にするための、参照アクションとドメイン、カラムや表制約も備わっています。

ファイル・システムとデータベース

データベースとCOBOL索引ファイルによるデータ格納の方法には、いくつかの類似する要素があります。以下の表は、各システムのデータ構造の違いと、どのように対応しているかを表しています。

COBOL 索引ファイル・システム・オブジェクト	データベース・オブジェクト
ディレクトリ	データベース
ファイル	表
レコード	行
フィールド	カラム

図 2-1 COBOLとデータベース・オブジェクトの構造

索引ファイル操作はCOBOLのレコードで実行され、データベースのカラムで実行されます。理論的には、COBOL索引ファイルは、データベースの表を意味します。COBOLファイルの各レコードは、データベースの各行を表し、各フィールドは表のカラムを意味します。データベースの表カラムが、INTEGER、CHAR、DATEのような特定のデータ型に関連付けられるのに対し、COBOLではデータに複数のタイプを指定することがしなければなりません。

➲ **例**

COBOLレコードは、次の形式で定義されています:

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

上記の例のCOBOLレコードは、データベースでは下記のように表されます。各行は、COBOL 01 レベルレコードのterms-recordのインスタンスを代表するかを注意してください。

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

図 2-2 データベースの行に変換されたCOBOLのレコード

データ・アクセス

ACUCOBOL-GTのgenericファイル・ハンドラーは、DCIとVisionファイル・システムのインターフェースです。Visionは、ACUCOBOL-GTと共に提供される標準索引ファイル・システムです。JISAMについての詳細な情報をACUCOBOL-GT参照編をご参考ください。

データ・ディクショナリと組み合わせると、DCIはCOBOLベースのアプリケーション・プログラム・インターフェース（API）のブリッジデータ・アクセスとDBMasterのデータベース管理システムを繋ぐゲートウェイになります。ユーザーは、APIを経由してデータにアクセスすることができます。更に、dmSQLやJDBA ToolのようなDBMasterのSQLインターフェースを利用すると、アドホックなデータ問合せを実行することもできます。データ・ディクショナリは、ACUCOBOL-GTコンパイラで作成します。詳細については、3章の「*データ・ディクショナリ*」で解説します。

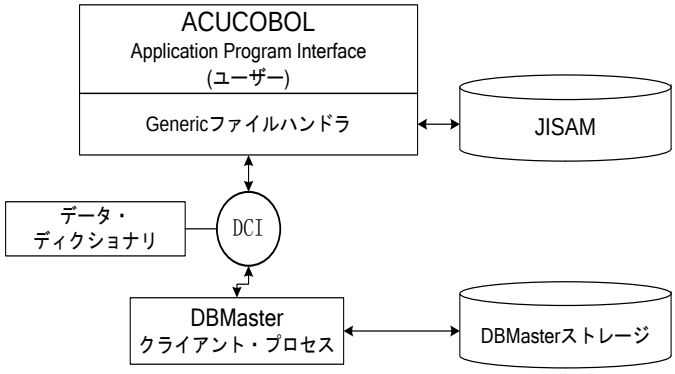


図 2-3 データ・フローチャート

2.2 システム必要環境

DBMasterのDCIは、アドオン・モジュールです。ACUCOBOL-GTランタイム・システムにリンクさせる必要があります。つまり、DCIをインストールするにはCコンパイラが必要です。インターフェースとして、ACUCOBOL-GT Version 4.3以降のランタイムを使用します。

DCIディレクトリにあるREADME.TXTファイルに、製品と共に提供されるファイルの一覧があります。

次のプラットフォームでDCIを使うことができます。

- SCO OpenServer
- Sun Solaris x86
- Windows 9x、ME、NT、2000、XP
- Linux 2.2、2.3
- AIX
- HP/UX
- FreeBSD 4

DCIに、以下のソフトウェアをインストールする必要があります。

- DBMaster version 5.2以上
- ACUCOBOL-GT runtime version 4.3以上
- ローカル機用のCコンパイラ (Windowsプラットフォームの場合、Visual C++™ Version 6.0)

2.3 セットアップの手順

DCIの環境設定をする前に、新規のDBMaster versionをインストールし、環境設定する必要があります。DBMasterのインストールの方法については、DBMasterのCDに含まれるクイック・スタートを参照して下さい。

Windowsでのセットアップ

DCIをセットアップする前、DCIファイルはDCI zip ファイルのソースディレクトリ(\DCI\OS\)から目的ディレクトリまでコピーしなければなりません。DCIライブラリはACUCOBOL-GT 5.1以下バージョンのdmdcic.lib, dmacu51.lib 、ACUCOBOL-GT 5.2～8.0バージョンのdmacu52.lib 、ACUCOBOL-GT 8.0バージョンのdmacu80.lib とACUCOBOL-GT 9.0バージョンのdmacu90.lib を含みます。

➲ **DCIをセットアップする:**

1. @DM_PRODUCT_NAME@をインストールします。最新のDCIを配置する前@DM_PRODUCT_NAME@をインストールしなければなりません。
2. @DM_PRODUCT_NAME@ DCI 圧縮ファイルからDCIライブラリ dmdcic.lib とACUCOBOL-GT のDCI ライブラリをACUCOBOL-GTインストールディレクトリまでコピーします。

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

ACUCOBOL-GT 5.1 以降のバージョン

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

ACUCOBOL-GT 5.2 -8.0バージョン

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

ACUCOBOL-GT 8.0:

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

ACUCOBOL-GT 9.0:

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

3. ACUCOBOL-GT ランタイム配置ファイルfiletbl.cを編集します。これはACUCOBOL-GT ライブラリと同じようなディレクトリにおいています。例：c:\acucobol\acugt\lib。

a) filetbl.cに元々あるエントリ:

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

次のように新たにエントリを追加します:

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

b) filetbl.cに元々あるエントリ:

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

次のように新たにエントリを追加します:

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

c) filetbl.cに元々あるエントリ:

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

d) 次のように新たにエントリを追加します:

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

4. ACUCOBOLランタイム環境設定ファイルsub85.cを編集します。ACUCOBOL-GTライブラリがあるディレクトリにあります。

sub85.cに元々あるエントリ:

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

次のように追加します:

```
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. ACUCOBOLファイルdirect.cを編集します。それは、ACUCOBOL-GTライブラリを含んでいるディレクトリにあります。

direct.cに元々あるエントリ:

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

次のように新たにエントリを追加します：

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

6. AcuGT < 6.0を使用している場合、ファイルwrun32.makを開きます。このファイルはACUCOBOL-GTライブラリと同じなディレクトリに置かなければなりません。例えば、c:\acucobol\acugt\lib。

e) ACUCOBOL-GT 5.1または前のバージョン：以下のファイルを追加する必要があります：dmacu51.lib, dmdcic.lib と dmapi52.lib。プロジェクトを作成して新しいwrun32.exeとwrun32.dllファイルを取得します。

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

f) ACUCOBOL-GT 5.2以上のバージョン：以下のファイルを追加する必要があります：dmacu52.lib, dmdcic.lib と dmapi52.lib。プロジェクトを作成して新しいwrun32.exeとwrun32.dllファイルを取得します。

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

7. ACUCOBOL 6.0または6.1を使用する場合、AcuGTインストールのlib\ディレクトリにある、wrun32.dswという名前のVisual C++プロジェ

クトを開きます。 ファイルdmapi52.lib、dmacu52.lib、dmdcic.libをプロジェクトに追加します。 プロジェクトを作成して、新しいwrun32.dllファイルを取得します。

8. ACUCOBOL 6.2または7.0を使用する場合、AcuGT インストールの lib\ ディレクトリに保存されている wrundll.vcproj という名前のVS2003プロジェクトを開きます。ファイル dmapi52.lib、dmacu52.lib、dmdcic.lib をプロジェクトに追加します。プロパティで、「MFCの共通DLLを使用する」を選択します。プロジェクトを構築して、新しいwrun32.dllファイルを取得します。

9. ACUCOBOL8.0を使用する場合、AcuGT インストールの lib\ ディレクトリに保存されている wrundll.vcproj という名前のVS2005プロジェクトを開きます。

   a) UseOfMFCを0から2に変更します。(例えば、共通DLLのMFCを使います)

   b) dmacu80.lib dmdcic.lib dmapi52.libをAdditionalDependencies,に追加します、例：

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

   c) VS2005を使用してプロジェクトを作成して新しいwrun32.dllファイルを取得します。

   注 *64ビットでACUCOBOL-GT 8.0ランタイムを設定するため、ユーザーはVS2005 x64構成要素をインストールしなければなりません。*

   注 *ランタイムを設定する場合、VS2005エクスプレス版を使用することができません、MFCライブラリがないからです。*

   注 *Fx3 或いは -Fx4コンパイルオプションを使用して以前のACUCOBOL-GTバージョンと同じようなフォーマットを持つXFDファイルを生成することを推薦いたします。ACUCOBOL-GT 8.0コンパイルは"$XFD COMMENT directive"のようなXFD構文をサポートしないからです。しかし、新しいXMLフォーマットを使用するため、DCI配置ファイルにDCI_XML_XFD 1を追加する必要があります。*

10. ACUCOBOL9.0を使用する場合、AcuGT インストールの lib\ ディレクトリに保存されている wrundll.vcproj という名前のVS2008プロジェクトを開きます。

    a) UseOfMFCを0から2に変更します。(例えば、共通DLLのMFCを使います)

    b) dmacu90.lib dmdcic.lib dmapi52.libをAdditionalDependencies,に追加します、例：

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

    c) VS2008を使用してプロジェクトを作成して新しいwrun32.dllファイルを取得します。

    注 *64ビットでACUCOBOL-GT 9.0ランタイムを設定するため、ユーザーはVS2008 x64構成要素をインストールしなければなりません。*

    注 *ランタイムを設定する場合、VS2008エクスプレス版を使用することができません、MFCライブラリがないからです。*

    注 *Fx3 或いは -Fx4コンパイルオプションを使用して以前のACUCOBOL-GTバージョンと同じようなフォーマットを持つXFDファイルを生成することを推薦いたします。ACUCOBOL-GT 8.0コンパイルは"$XFD COMMENT directive"のようなXFD構文をサポートしないからです。しかし、新しいXMLフォーマットを使用するため、DCI配置ファイルにDCI_XML_XFD 1を追加する必要があります。*

11. 新しいwrun32.exeとwrun32.dllファイルを実行パスにいうディレクトリまでコピーします。例：

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

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

12. @DM_PRODUCT_NAME@ installed\binディレクトリにシステム変数PATHを設置します、例えば：

```
set PATH=c:\@DM_PRODUCT_NAME@\5.3\bin:%PATH%
```

ユーザーはc:\@DM_PRODUCT_NAME@\5.3\bin\dmapi53.dllを wrun32.exe と wrun32.dllの位置までコピーできます。 @DM_PRODUCT_NAME@の新しいパッチをインストールと、 dmapi52.dllをそのディレクトリまで更新してください。

13. エントリを通じてリンクを検査する：

```
wrun32 -vv
```

これを通じて、ユーザーのランタイムシステムにリンクする全部の製品のバージョン情報を返します。@DM_PRODUCT_NAME@インタフェースに表示されます。

UNIXセットアップ

DCIのセットアップを行う前に、DCIファイルをDBMaster CDのソース・ディレクトリ (CDROM:\DCI\OS\) から、希望のディレクトリにコピーします。DCIライブラリはACUCOBOL-GT 5.1以下バージョンのdmdcic.lib, dmacu51.lib 、ACUCOBOL-GT 5.2～8.0バージョンのdmacu52.lib 、ACUCOBOL-GT 8.0バージョンのdmacu80.lib とACUCOBOL-GT 9.0バージョンのdmacu90.lib を含みます。

➲ **DCIをセットアップする:**

1. DCIライブラリdmdcic.lib とACUCOBOL-GT のDCI ライブラリをACUCOBOL-GTインストールディレクトリまでコピーします。

例： LinuxにDCIライブラリをセットアップしたければ、ユーザは下記のコマンドをタイプすべきです。

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

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

DCIライブラリをACUCOBOL-GT 5.1以降のバージョンとリンクする：

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

DCIライブラリをACUCOBOL-GT 5.2～8.0のバージョンとリンクする：

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

DCIライブラリをACUCOB-GT 8.0バージョンとリンクする：

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

DCIライブラリをACUCOB-GT 9.0バージョンとリンクする：

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

2. ACUCOBOLランタイム配置ファイル**filetbl.c**.を編集します。
ACUCOBOL-GTライブラリがあるディレクトリにあります。

a) filetbl.c に元々あるエントリ:

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

新たにエントリを追加します:

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

b) filetbl.cに元々あるエントリ:

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

次のように新たにエントリを追加します:

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

c) filetbl.cに元々あるエントリ:

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

次のように新たにエントリを追加します:

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

3. ACUCOBOLランタイム配置ファイル**sub85.c**を編集します。
ACUCOBOL-GTライブラリがあるディレクトリにあります。

sub85.cに元々あるエントリ:

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

次のようにエントリを追加します:

```
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. ACUCOBOL-GTランタイム配置ファイル**direct.c**を編集してください。それは、ACUCOBOL-GTライブラリがあるディレクトリにあります。

direct.cに元々あるエントリ:

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

次のようにエントリを追加します:

```
extern char *dci_where_constraint;
```

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

5. Makefileファイルを開きます。これは、*\usr\acucobol\43\lib*にあります。独自のCルーチンにリンクさせる必要がある場合、独自のCルーチンにあるMakefileファイルの**SUBS=**の行にそれらを追加します。C サブ・ルーチンのリンクについての詳細は、ACUCOBOL-GTコンパイラのマニュアルの付録Cを参照して下さい。

6. **FSI_LIBS=**の行に、/APP_HOME/lib/libdmdcic.aと/APP_HOME/lib/libdmapic.aを追加します。/APP_HOMEは、@DM_PRODUCT_NAME@のインストール・ディレクトリです。@DM_PRODUCT_NAME@がディレクトリー/APP_HOMEにインストールされた場合、**Makefile**は次のストリング(s)を含むでしょう。

   ACUCOBOL 5.1以前のバージョン:
   ```
   FSI_LIBS=libdmacu51.a libdmdcic.a libdmapic.a
   ```

   ACUCOBOL 5.2:
   ```
   FSI_LIBS=libdmacu52.a libdmdcic.a libdmapic.a
   ```

   ACUCOBOL6.0と7.0:
   ```
   FSI_LIBS=libdmacu60.a libdmdcic.a libdmapic.a
   ```

   ACUCOBOL8.0:
   ```
   FSI_LIBS=libdmacu80.a libdmdcic.a libdmapic.a
   ```

   ACUCOBOL9.0:
   ```
   FSI_LIBS=libdmacu90.a libdmdcic.a libdmapic.a
   ```

7. 次に、ACUCOBOL-GTランタイム・システムがあるディレクトリになっていることを確認します。
   ```
   make -f Makefile <enter>
   ```
   これにより、**sub.c**と**filetbl.c**をコンパイルし、ランタイム・システムにリンクします。 失敗になると、out-of-dateシンボテーブルのせいです。以下のように実行してください：
   ```
   ranlib *.a
   ```

そして、このmake を再実行します。失敗になる場合、ACUCORP技術サポートに連絡してください。

8. リンクを確認します。

```
./runcbl -vv
```

これにより、ランタイム・システムにリンクしている製品の全てのバージョン情報が戻ります。@DM_PRODUCT_NAME@のDCIバージョンが表示されていることを確認します。

注 *独自のCルーチンをランタイム・システムにリンクさせることもできます。*

9. 作成したruncblファイルを実行パスにあるディレクトリにコピーします。このファイルには、ランタイム・システムを使用する全ての人への実行許可が必要です。残りのファイルはディストリビューション・ミディアムからインストールしたディレクトリにそのまま残しておくことができます。

10. Fx3 或いは -Fx4コンパイルオプションを使用して以前のACUCOBOL-GTバージョンと同じようなフォーマットを持つXFDファイルを生成することを推薦いたします。ACUCOBOL-GT 8.0コンパイルは"$XFD COMMENT directive"のようなXFD構文をサポートしないからです。しかし、新しいXMLフォーマットを使用するため、DCI配置ファイルにDCI_XML_XFD 1を追加する必要があります。

共有ライブラリ

ACUCOBOL-GTランタイムに再リンクし、実行しようとする際に、以下のようなエラー・メッセージを受け取るかもしれません。

- 「ライブラリをロードできません。そのようなファイルやディレクトリがありません。」
- 「共有ライブラリを開くことができません。」

これは、おそらくオペレーティング・システムが共有ライブラリを位置する可能性がないことを意味しています。共有ライブラリが現在のディレクトリに存在する場合でも、これは起こりえます。例えば、AIX 4.1を動かしているIBM RS/6000では、環境変数LIBPATHは共有ライブラリがあるディレクトリを表示しなければなりません。HP/UXでは、環境変数は

SHLIB_PATHです。UNIX SVR4では、環境変数はLD_LIBRARY_PATHです。UNIXのマニュアルを参照することをお勧めします。

このようなエラーを避けるために、静的リンクで共有ライブラリをランタイムにリンクさせることも可能です。C開発システムのマニュアルを参照して、使用している環境用の正しいフラグを見つけて下さい。

2.4 基本の環境設定

DCIを作動させるためのパラメータがある環境設定ファイルが2つあります。1つ目は**cblconfig**、ACUCOBOLのランタイム環境設定ファイルです。2つ目のDCI_CONFIGファイルは、環境変数(詳細は*「環境設定ファイルの変数」*を参照のこと)で指定するディレクトリにあります。DCIを使いはじめるには、DCI_CONFIGファイルに設定するパラメータがデータベースでどのようにデータを表示させるかを指定します、データベースにアクセスするために基本DBA関数を決定します。DCIを使用する前、次の環境設定変数をセットします。

- DCI_DATABASE
- DCI_LOGIN
- DCI_PASSWD
- DCI_XFDPATH

➲ **例**

基本のDCI_CONFIGファイルは以下のとおりです。

```
DCI_LOGIN SYSADM

DCI_PASSWD

DCI_DATABASE DBMaster_Test

DCI_XFDPATH /usr/dbmaster/dictionaries
```

DCI_DATABASE

DCIからデータベースへの全トランザクションは、DCI_DATABASEで指定します。DBMasterを使用してこれを作成するときこのデータベースは存在しなければなりません。データベース名は、初期設定では大文字と小文字を識別しません。サイズは、128文字以下の文字列です。

➲ **構文**

環境設定ファイルには次のように入力します。この例で、DCI使用のデータベースはDBMaster_Testです。

```
DCI_DATABASE DBMaster_Test
```

注　*詳細な情報を第7章の"DCI_DATABASE"をご参考ください。*

DCI_LOGIN

ユーザー名を持つCOBOLアプリケーションから、データベースのオブジェクトにアクセスすることができます。環境設定変数DCI_LOGINは、DCIを使用するCOBOLアプリケーション用にユーザー名をセットします。既定ではSYSADMにセットされ、データベースへの完全なアクセス権があります。この値を他のユーザー名に簡単に変更することができます。詳細については、7章の「DCI_LOGIN」を参照して下さい。

➲ **構文**

ユーザー名SYSADMでデータベースに接続する場合は、DCIの環境設定ファイルに次のようなエントリを含まなければなりません。

```
DCI_LOGIN SYSADM
```

DCI_PASSWD

DCI_LOGIN変数を通じてユーザー名を指定すると、データベース・アカウントがそれに関連付けられます。SYSADMは、DBMasterの初期設定ではパスワードがありません。アカウント情報(LOGIN、PASSWD)が正しいな

ら、データベース管理者は知っています。。詳しくは、7章の「DCI_PASSWD」を参照して下さい。

➲ **構文**

データベース・アカウントをSYSADMにセットすると、環境設定ファイルは次のようになります。

```
DCI_PASSWD
```

DCI_XFDPATH

DCI_XFDPATHは、データ・ディクショナリを格納するディレクトリ名を指定します。初期設定値は、現在のディレクトリです。

➲ **構文 1**

データベース・ディクショナリをディレクトリ */usr/dbmaster/dictionaries*に格納させる場合、環境設定ファイルには次のように入力します:

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

➲ **構文 2**

複数のパスを指定する必要がある場合、複数のディレクトリをスペースで区切って指定します:

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

➲ **構文 3**

WIN-32環境でダブルクオートを使用すると、「埋め込みスペース」を指定することができます、例えば :

```
DCI_XFDPATH c:\tmp\xfdlist “c:\my folder with space\xfdlist”
```

2.5 Runsqlユーティリティ

DCIのrunsql.acuプログラムは、いくつかの標準のSQLコマンドにアクセスすることができます。COBOLプログラムから呼び出すこともできますし、

コマンドラインから実行することも可能です。CALL文のSQL文のサイズは、32767文字制限で、変数やシングルクオートで括った文字列も使うことができます。

一般的には、runsql.acuは全てのSQL文を実行するために使用しますが、データ回収は含まれません。例えば、SELECT文のようなデータを戻す文を実行することがサポートしません。この種のSQL文が、runsql.acuプログラムに渡された時に、エラーになります。

コマンドが問題無く完了した場合、グローバル変数の戻り値は0です。コマンドが完了しなかった場合、グローバル変数の戻り値には、エラー・コードが含まれます。

➲ **構文 1**

DBMasterのビューを作成する：

```
runcbl runsql.acu
```

➲ **例 1**

次の文は、プログラムを一時停止させSQL文を受け入れさせるために使います。

```
create table TEST (col1 char(10), col2 char(10))

create view TESTW as select * from TEST
```

➲ **構文 2**

次は、COBOLプログラムの中からでsql.acuを呼び出すために使用します。

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

➲ **例 2**

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

2.6 無効なデータ

COBOLアプリケーションでは有効で、DBMasterのRDBMSデータベースでは無効なデータがあります。この節では、RDBMSで受け付けることができないデータ型と、この問題を解決するためにDCIが採用している方法をリストします。

COBOLの値	不正とみなされる場所
LOW-VALUES	USAGE DISPLAY NUMBERSとテキスト・フィールド
HIGH-VALUES	USAGE DISPLAY NUMBERS、COMP-2番号、COMP-3番号
SPACES	USAGE DISPLAY NUMBERS、COMP-2 番号
Zero	DATEフィールド

図 2-4 不正なCOBOLデータ

図2-4にいずれかに適用するかどうかを判断するために、他の数値型の内部ストレージ形式を参照します。BINARY数は、BINARYテキスト・フィールドにある全ての値と同様、常に正常です。

データ型によっては、DBMasterに格納する前に変換する必要があります。DCIはこれらの値を次のように変換します。

- 不正なLOW-VALUESは、最小値(0、又は - 99999)、又はDCI_MIN_DATEの初期設定値で格納します。
- 不正なHIGH-VALUESは、最高値(99999)、又はDCI_MAX_DATEの初期設定値で格納します。
- 不正なスペースは、ゼロとして格納します（データ・フィールドの場合、DCI_MIN_DATE）。
- 不正なDATE値は、DCI_INV_DATEの初期設定値で格納します。
- 不正なTIME値は、DCI_INV_DATEの初期設定値で格納します。

データベースからDCIに送信したNullフィールドは、次の方法でCOBOLに変換されます。

- 数値（日付を含む）はゼロに変換されます。
- テキスト（バイナリ・テキストを含む）は、スペースに変換されます。

キー・フィールド以外で上記の変換ルールを修正する場合は、不正なCOBOLデータをNULLに変換するDCI_NULL_ON_ILLEGAL_DATAを使うことができます。

2.7 サンプル・アプリケーション

DCIファイルに含まれるサンプル・アプリケーション・プログラムは、DCIがアプリケーションのデータをどのようにDBMasterデータベースにマップするかを示しています。この節は、以下のを学習します：

- アプリケーション・プログラムをセットアップする方法。
- アプリケーション・オブジェクト・コードを通じてソースコードのコンパイル方法。
- アプリケーションにデータを入力する方法。
- dmSQLとJDBA Toolを使ったデータへのアクセス方法。
- ソースコードを生成する表のスキーマに合わせる方法。

アプリケーションのセットアップ

アプリケーションは、DCIディレクトリにあります。ファイルI INVD.CBL, INVD.FD, INVD.SL, TOTEM.DEF, CBLCONFIG, INVD.XFD, DCI.CFG.から構成されています。アプリケーションは、以下のように直接にINVD.ACUオブジェクトコードファイルから実行できます。アプリケーションはソースコードINVD.CBLからコンパイルされます。

注 *サンプル・アプリケーションをコンパイルするために、予めACUCOBOL 4.3 以上をインストールしておく必要があります。*

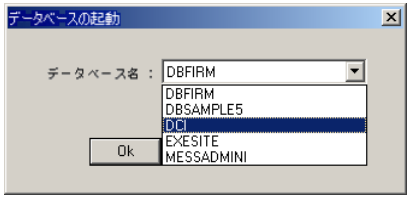
➲ **アプリケーションを実行する:**

1. DCIからのデータを受け入れるために、DBMasterでデータベースを立ち上げます。このプロシージャの例のように、JServer Managerを使って、データベースDCIを作成します。データベースには、全て初期設定を使用します。ログイン名として、初期設定のSYSADMを使用します。初期設定ではパスワードがありません。データベース作成とセットアップについての詳細は、「データベース管理者参照編」、または「JServer Manager ユーザーガイド」を参照して下さい。
2. \DCIディレクトリで、DCI.CFGファイルを開きます。環境設定変数を適切な値にセットします。環境設定のファイルの値についての詳細は、2.4節の「基本の環境設定」を参照して下さい。

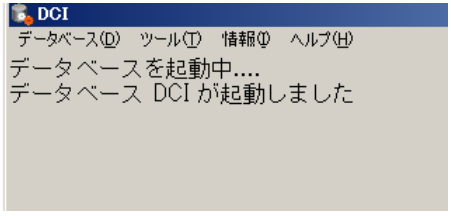
➲ **例**

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

3. DBMasterのサーバー・プログラム(dmserver.exe)を実行します。DCI.CFGファイルで指定したデータベース名を選びます。



4. データベースが正常に起動すると、次のウィンドウが表示されます。問題が発生した場合、或いはエラーが表示された場合は、「エラー・メッセージ参照編」、または「データベース管理者参照編」をご覧下さい。



5. コマンド・プロンプトを開き、..\DCIディレクトリへ移動。
6. コマンド・プロンプトに次を入力し、DCI_CONFIGを定義します。

➲ **構文 6a**

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

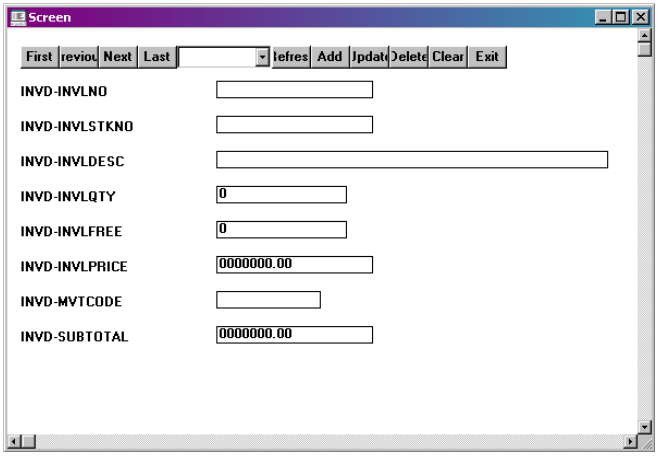
7. WRUN32を使って、COBOLプログラムINVD.ACUを実行します。

➲ **構文 7a**

同じディレクトリで、コマンド・プロンプトに次を入力します。

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

8. CBLCONFIGファイルには、コマンドラインDEFAULT-HOST DCIがあり、初期設定ファイル・システムをセットするために使用されます。詳細については、5章の「コンパイラとランタイム・オプション」を参照して下さい。
9. COBOLのアプリケーションINVD.ACUのウィンドウがデータエントリによって下記のように表示されます。。



注 *レコード追加の方法については、下記「レコードを追加する」を参照して下さい。*

➲ **ソースコードからサンプル・プログラムをコンパイルする:**

1. 上述の「アプリケーションを実行する」のステップ1～6を行います。
2. ..\Acucorp\Acucbl500\AcuGT\sample\def ディレクトリから、DCI ディレクトリに定義ファイルをコピーします。定義ファイルは、acucobol.def、acugui.def、crtvars.def、fonts.def、showmsg.defです。

   注 *ACUCOBOL 4.3ユーザは、 \Acucbl43\AcuGT\sample\ ディレクトリから上記の定義ファイルをコピーするべきです。*
3. コマンド・プロンプトで、DCIディレクトリへ移動。

➲ **構文 3a**

次のエントリを入力します:

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

4. ファイルはコンパイルされ、新しいオブジェクト・コードファイルINVD.ACUとデータ・ディクショナリ・ファイルINVD.XFDが生成されます。前述「アプリケーションを実行する」のステップ6、7に従って、オブジェクト・ファイルを実行します。

レコードを追加する

一旦アプリケーションが起動すると（前節「アプリケーションをセットアップする」を参照のこと）、アプリケーション、つまりデータベースにレコードを追加するだけになります。フィールドINVD-INVLNO は、キー・フィールドですので、一意の値のみが有効なレコードとなります。その他のフィールドは、全て空白にしておくことができます。フィールドへの値の入力が終了したら、［Add］ボタンをクリックします。フィールドに入力した値は、DCI.CFGの変数DCI_DATABASEで指定した、DBMasterのデータベースに保存されます。

➲ 例

アプリケーションのファイル・ディスクリプタは、次のようになります。

```
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.
```

レコードを追加すると、［Screen］ウィンドウの［First］、［Previous］、［Next］、［Last］を使って、入力データを閲覧することができます。各レコードは、キー・フィールドの全ての値が表示されている［Screen］ウィンドウの上部にあるドロップダウン・メニューから選択することができ

ます。「*データにアクセスする*」の節で、DBMasterのSQLベースのツールを使ったブラウズ方法について解説します。

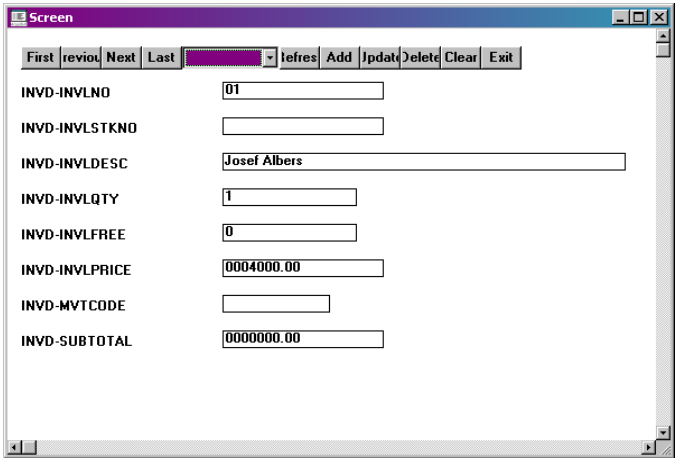


図 2-5 INVD-INVLNOキー・フィールドの入力例

データにアクセスする

サンプル・アプリケーションで作成したレコードは、簡単に閲覧、操作することができます。まず、dmSQL、JDBA ToolのようなDBMasterのツールに慣れることが理想的です。これらのツールについての詳細は、*「データベース管理者参照編」*や*「JDBA Tool ユーザーガイド」*をご覧下さい。次の例は、JDBA Toolを使ってデータにアクセスする方法を表しています。

INVDアプリケーションは、データベース内に作成された表にロックをかけるので、まず終了する必要があります。JDBAでデータベースに接続します。下記のようにデータベース・ツリーの表ノードから、表を見ることができます。

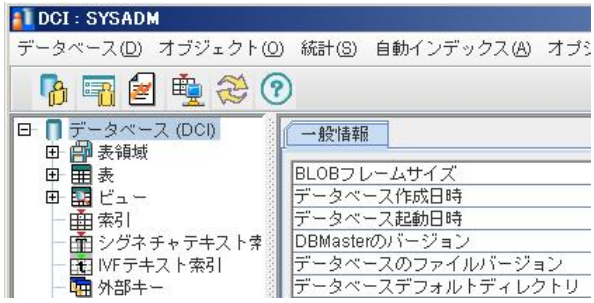


図 2-6 INVDアプリケーションの表ツリー・ノード

表の**SYSADM.invd**をダブルクリックすると、表のスキーマを見ることができます。ここで全カラムとそのプロパティを閲覧することもできます。

図2-7 SYSADM.invdの表スキーマ

［**データの編集**］タブをクリックして、各フィールドの値を確認します。

図 2-8 SYSADM.invdの［データの編集］タブのフィールドの値

3 データ・ディクショナリ

データ・ディクショナリは、拡張ファイル・ディスクリプタ(.XFD)ファイルがどのように生成され、アクセスされるかを記しています。DCIは、ACUCOBOL-GTの特殊な機能を使うことで、COBOLコードに埋め込まれたSQL関数コールを使用しないようにすることができます。「–Fx」オプションを使って、COBOLアプリケーションをコンパイルすると、データ・ディクショナリが生成されます。これらは、「拡張ファイル・ディスクリプタ」(XFDファイル)として知られ、COBOLファイルのディスクリプタを基にしています。DCIは、COBOLアプリケーションのフィールドと、DBMasterの表のカラム間でデータのマップを行うために、データ・ディクショナリを使います。DCIで使用するDBMasterの表にはいずれも、少なくとも1つの対応するデータ・ディクショナリがあります

注 *詳細及び、XFD生成のルールについては、ACUCOBOL-GT ユーザーガイド(5.3章) を参照して下さい。*

3.1 表名を割り当てる

データベースの表は、COBOLアプリケーションのFILE CONTROLセクションにあるファイル・ディスクリプタに対応しています。データベースの表には、128バイト以下(128 ASCII文字)の一意の名前が必須です。

DBMasterの表には、COBOLプログラムの対応ファイル・ディスクリプタ以上のカラムがある可能性があります。

COBOLプログラムの対応ファイル・ディスクリプタとは異なる並びのカラムがある可能性もあります。データベースの表にあるカラムの数は、表にアクセスするCOBOLプログラムでフィールドの数と必ずしも一致する必要はありません。DBMaster表は、COBOL プログラム参照以上のカラムを格納することができます。但し、COBOLプログラムで、DBMaster表以上のフィールドを格納することはできません。新しい行が表に追加された際に、これらの「余分な」カラムが正しくセットされたことを確認します。

ACUCOBOLは、初期設定としてFILE CONTROLセクションから、XFDファイル名を生成します。ファイルのためのSELECT文 に変数ASSIGN *名前* (ASSIGN TO *ファイル名*)がある場合、FILEディレクトリを使ってXFDファイル用の起動名を定義します(4章の「*$XFD* FILEディレクティブ」を参照のこと)。ファイルのためのSELECT文に定数ASSIGN *名* (ASSIGN TO “EMPLOYEE”のような)がある場合、その定数はXFDファイル名を生成するために使用されます。ASSIGN句がデバイスを参照し、総称的な場合 (ASSIGN TO “DISK”のような)、コンパイラは、XFDファイル名を生成するためにSELECT名を使用します。

ファイル名とユーザー名は、大文字と小文字を識別しません。大文字を含むファイル・ディスクリプタは、全て小文字に変換されます。大文字と小文字を識別するオペレーティング・システムを使用している場合、ユーザーはこの点に注意して下さい。

➲ **例 1**

もしFILE CONTROL以下の構文を含むと：

```
SELECT FILENAME ASSIGN TO “Customer”
```

➲ **例2**

FILE CONTROLセクションに、次のテキスト行がある場合、DCIは「customer.xfd」で読み込まれるディクショナリ情報を元に、「*username*.customer」という名前のDBMasterの表を作成します。ACUCOBOL-GT コンパイラは、常に小文字でファイルを生成します。「ユーザー名」の初期設定は、DCI_CONFIGファイルのDCI_LOGIN値で

指定します。或いは、DCI_USER_PATH環境設定変数で変更することができます。

```
SELECT FILENAME ASSIGN TO “CUSTOMER”
```

➲ **例 3**

ファイルに拡張子がある場合、DCIでは“.”文字から“_”に置き換えられます。DCIは、「username.customer_dat」という名前で、DBMasterの表を開きます。

```
SELECT FILENAME ASSIGN TO “customer.dat”
```

➲ **例 4**

DCI_MAPPINGは、ディクショナリcustomer.xmlを使用可能な状態にするために使用されます。DCIは、EFDディクショナリを見つけるために元の名前を使用するので、この例では「customer_dat. xml」という名前のXMLファイルを探します。次の設定では、「customer. xml」という名前のXMLファイルを元にしています。

```
DCI_MAPPING customer*=customer
```

COBOLアプリケーションは、異なるディレクトリにある同じファイル名をもつ別々のファイルを使用するかもしれません。COBOLアプリケーションは、「/usr/file/customer」と「/usr1/file/customer」のような異なるディレクトリにある「customer」という名前のファイルを開きます。ファイル名を一意にするためには、ファイル名にディレクトリ・パスを加えます。これを行うには、DCI_CONFIG変数のDCI_USEDIR_LEVELを「2」に変更します。そうすると、DCIでは次のように表を開きます。

COBOL	**RDBMS**	**XFD**ファイル名
/usr/file/customer	usrfilecustomer	usrfilecustomer. xfd
/usr1/file/customer	usr1filecustomer	usr1filecustomer. xfd

図 3-1 DCI_USEDIR_LEVELを「2」にした例

注 *DBMasterの表名にある最大長の制限に注意して下さい。DCI_MAPPINGは、XFDファイルをディクショナリ定義にマップするために使用します。*

COBOLコード	結果ファイル名	結果表名
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)

図 3-2別々のCOBOL文から形成された表名の例

➲ **例5**

表名は、次々にXFDファイル名から生成されます。表名を指定するためのもう1つの方法は、$XFD FILEディレクティブを使用することです。

```
   *(( 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).
```

要約すると、ファイル名は次のように形成されます。

- コンパイラは、拡張子を変換し、（.）をアンダースコア（_）に置き換え、起動名にインクルードします。

- ファイル名とディレクトリ情報から、DCI_CONFIGの変数DCI_USEDIR_LEVELで定義したように、ユニバーサル基礎名を構築します。元の名前は32文字に縮められ、DCI_CASEの値によっては、小文字に変換されます。

3.2 カラムとレコードのマッピング

作成された表は、COBOLファイルの最大レコードに基づいています。レコードの全てのフィールドとキー・フィールドが含まれています。キー・フィールドは、KEY IS句を使って、FILE CONTROLセクションに定義します。キー・フィールドは、データベースの表の主キーに相当します。詳細については、後ほど説明します。DCIでは、表名と異なり、大文字と小文字を識別して、データベースのカラム名を生成します。

➲ **例 1**

以下はデータが転送される方法を表しています。

```
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.
```

➲ 例 2

処理プログラムは、通常全てのフィールドをファイル順に書き込みます。出力は次のようになります。

```
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
```

従来のCOBOLファイル・システムでは、レコードは順番に格納されます。書き込み命令文を実行するたびに、データはファイルに送信されます。DCIを使用すると、データ・ディクショナリは、データベースにデータを格納するためのマップを作成します。この場合、レコード(EMPLOYEE-RECORD)は、ファイルにあるレコードのみです。

➲ 例 3

データベースは、ファイル・ディスクリプタに各フィールドに対し、別個のカラムを生成します。表名は、FILE-CONTROLセクションのSELECT文に指定されているHR-FILEになります。つまり、例にあるデータベースのレコードは、次のような構造になります。

EMP_ID (INT(6))	EMP_NAME (CHAR(17))	EMP_PHONE (DEC(10))
51100	LAVERNE HENDERSON	2221212999

EMP_ID (INT(6))	EMP_NAME (CHAR(17))	EMP_PHONE (DEC(10))
52231	MATTHEW LEWIS	2225551212

図 3-3 表EMPLOYEE-RECORD

この表では、INPUT-OUTPUTセクションのKEY IS句で定義したように、カラムEMP-IDが主キーになります。データ・ディクショナリは、レコードを回収するために「マッピング」を作成し、正しい位置にそれらを配置します。COBOLアプリケーションでこのように情報を格納すると、SQLの能力を活用することと同様に、データベースのバックアップとリカバリの機能を活用することができます。

同一のフィールド名

COBOLでは、同一の名前を持つフィールドは、グループ・アイテムを与えることでそれらを識別します。DBMasterでは、表に同一名のカラムが存在することはできません。同じ名前のフィールドがある場合、DCIでは、これらのフィールドに対応するカラムは生成されません。

このような場合の解決策として、NAMEディレクティブを追加します。これは、競合するフィールドのうち一方或いは双方に替わりの名前を付けます。（4章の「*$XFD NAME*」を参照のこと）

➲ **例**

次の例では、プログラムでPERSONNELとPAYROLLが参照されます。

```
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).
```

長いフィールド名

DBMasterサポートできる表名のサイズは128文字以下です。フィールド名のサイズがこれ以上の場合、DCIでフィールド名が切り縮められます。後で説明するOCCURS句の場合、元の名前を切り捨て、付属の索引番号は切り捨てません。但し、索引番号を含む最終的な名前は、32文字に限られます。例えば、フィールド名がTaipei-brunch-Employee-statistics-01の場合、表名ではTaipei-brunch-Employee-statis-01のように短縮されます。つまり、フィールド名の最初の18文字を、一意で意味のあるものに指定することが重要です。

NAMEディレクティブを使うと、長い名前のフィールド名を付け替えることができます。但し、COBOLアプリケーション内では、元の名前が使い続けられることに注意して下さい。NAMEディレクティブは、データベース内の対応するカラム名にのみ影響します。

3.3 複数のレコード形式を使う

これまでの節では、フィールドがどのように使用されてデータベースの表が生成されるのかについて説明しました。但し、これらの例の場合、アプリケーションには1つのレコードしかありません。

複数のレコード形式は、単一レコード形式とは違う方法で格納されます。複数のレコードのあるCOBOLプログラムは、ファイルの「マスター」(最大)レコードとファイルのキー・フィールドから全レコードをマップします。小さいレコードは、XFDファイルによってデータベースの表にマップされますが、表では別個に定義されたカラムという形にはなりません。替わりに、データベースの既存カラムに新しいレコードとなります。

➲ **例 1**

前節の例のファイル・ディスクリプタを修正し、複数のレコードを含めます。

```
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).
```

この例では、データ・ディクショナリが最も大きいファイルから作成されます。レコードEMPLOYEE-RECORDには、33文字あります。一方レコードPAYROLL-RECORDには16文字しかありません。この場合、レコードはデータベースに順序とおりに入力されます。レコードEMPLOYEE-RECORDを使って、表のカラムのサイズとデータ型のスキーマが作成されます。

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

図 3-4 前述の例の表

あとに続くレコードのフィールドは、フィールドの文字位置に応じてカラムに書き込まれます。結果、小さいレコード用に別個に定義されたカラムは存在しません。XFDファイルにはフィールドのためのマップがあるの

で、COBOLアプリケーションで、データをデータベースから回収することができますが、これらのフィールドを示す表のカラムはありません。

前述の例では、最初のレコードがデータベースに入力された際、カラムとCOBOLフィールド間には相関性があります。ところが、2つ目のレコードが追加された時には、そのような相関性はありません。データは、フィールドに応じて対応する文字を入れます。つまり、EMP_SALARYの最初の5文字が、EMP_IDカラムに入ります。EMP_SALARYの後半の5文字は、EMP_NAMEカラムに入ります。フィールドDD、MM、YYも、EMP_NAMEカラムに格納されます。

➲ **例 2**

COBOLアプリケーションに次の入力をします:

```
ENTER EMPLOYEE ID NUMBER (1-99999), ENTER 0 TO STOP ENTRY

51100

LAVERNE HENDERSON

2221212999

5000000000

01

04

00
```

フィールドは、表のスキーマに関連するフィールドの文字位置に応じて、統合され、分割されます。更に、カラムEMP_NAMEのデータ型はCHARです。DCIはデータ・ディクショナリにアクセスしたので、全フィールドは正しい位置で再度COBOLアプリケーションにマップされます。

これは非常に重要な事です。初期設定では、最も大きいレコードのフィールドが、表のスキーマを生成するために使用されます。そのため、ファイル・ディスクリプタ作成時には、慎重に表スキーマを考慮する必要があります。SQLの柔軟性を活用するためには、同じ文字の位置を占有する異なるレコードの間でデータ型を整合させます。PIC Xフィールドが、

DECIMALデータ型のデータベース・カラムに書き込まれた場合、データベースはアプリケーションにエラーを戻します。

➲ **例 3**

EMP_RECORD表にある全カラムの最初のレコードを選択するSQLのSELECT文を実行すると、次のように表示されます:

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

➲ **例 4**

EMP_RECORD表にある全カラムの2番目のレコードを選択するSQLのSELECT文を実行すると、次のように表示されます:

```
500000, 0000010400
```

3.4 XFDファイルの初期設定を使用する

DCIでは、初期設定のふるまいを変更するために、様々なディレクティブを使うことができます。詳細については、4章の「*XFDディレクティブ*」を参照して下さい。

コンパイラは、次のCOBOL要素を取り扱うために特殊な方法を使用します。

- REDEFINES句
- KEY IS句
- FILLERデータ・アイテム
- OCCURS句

REDEFINES句

REDEFINES句は、同じフィールドに複数の定義を設けます。DBMasterでは、1つのカラムのシングルデータ定義をサポートします。そのため、再定義されたフィールドは、表では元のフィールドと同じ位置に格納されま

す。初期設定では、データ・ディクショナリは、カラムのデータ型を定義するために、下位のフィールドのフィールド定義を使用します。

複数のレコードの定義は、本質的にはレコード域全体を再定義します。詳細と複数のレコード定義については、前節を参照して下さい。

グループ・アイテムは、結果として生じた表のスキーマのデータ・ディクショナリ定義に含まれません。替わりに、グループ・アイテムにある個々のフィールドは、スキーマを生成するために使用されます。グループ化されるフィールドは、USE GROUPディレクティブを使って統合されます。

KEY IS句

COBOLプログラムのINPUT-OUTPUTセクションのKEY IS句は、全レコードの一意索引のようなフィールドや、フィールドのグループを定義します。データ・ディクショナリは、KEY IS句にあるフィールドを、データベースにある主キーにマップします。KEY IS句にあるフィールド名がグループ・アイテムの場合、グループ・アイテムの下位のフィールドが、表の主キー・カラムになります。前下位フィールドを１つのフィールドに集める場合は、USE GROUPディレクティブを使用します。(7章の「*$XFD USE GROUPディレクティブ*」を参照のこと)。

FILLERデータ・アイテム

FILLERデータ・アイテムは、COBOLファイル・ディスクリプタのプレースホルダーです。これらには一意の名前がありません。一意に参照することはできません。データ・ディクショナリは、フィルタが文字列の位置に存在しているかのように、他の名前のフィールドをマップしますが、FILLERデータ・アイテム用に異なるフィールドを生成しません。

FILLERを表スキーマに含む必要がある場合、USE GROUPディレクティブ(4章の「$EFD USE GROUPディレクティブ」を参照のこと)、或いはNAMEディレクティブ(4章の「$EFD NAMEディレクトリ」を参照のこと)を使って、他のフィールドと統合されます。

OCCURS句

OCCURS句は、必要な数だけ、フィールドを定義することができるようにします。DCIでは、データベースの各カラム用に一意の名前を割り当てる必要があります。但し、OCCURS句で定義された複数のフィールドは全て同じ名前になります。この問題を解決するために、OCCURS句で指定したフィールドには、連番の索引番号が付加されます。

➲ **例 1**

ファイル・ディスクリプタの一部:

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

➲ **例 2**

次のカラム名が、データベースに生成されます。

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

3.5 複数のファイルをマップする

ランタイムに異なる名前を持つ複数のファイルに、単一のXFDファイルを使うことができます。ファイルのレコード定義が同じ場合、各ファイル用に別々のXFDファイルを作成する必要はありません。

ランタイム環境設定変数DCI_MAPPINGは、どのファイルをXFDのファイルにマップするかを定義します。

COBOLアプリケーションにEMPLOYEE-RECORDのような変数ASSIGN名があるSELECT文があると想定します。プログラム実行の際、この変数が、EMP0001とEMP0002のような別の値だと仮定します。XFDの元にする名前を与えるために、FILEディレクティブを使います。((XFD DATE, USE GROUP)を参考してください)

➲ **例**

「EMP」を元にする場合、コンパイラは「Emp.xfd」という名前のXFDを生成します。例にあるアスタリスク(“*”)は、ファイル名の数字を意味するワイルドカードの文字です。ファイルの拡張子「.xfd」は、マップに含むことはできません。この文は、XFD「emp.xfd」を「EMP」で始まる名前の全ファイルに使用させるようにします。それぞれ一意で相関名を持っているemployeeファイルに全て同じXFDを必ず使用させるために、ランタイム環境設定ファイルに次のエントリを追加します。

```
DCI_MAPPING EMP* = EMP
```

DCI_MAPPING変数は、ファイルを開く際に読み込まれます。「*」と「?」のワイルドカードをパターンの中で使用することができます。

*　　複数文字に相当します。

?　　１文字に相当します。

EMP?????　　EMP00001とEMPLOYEEに相当しますが、EMP001やEMP0001には当てはまりません。

EMP*　　上記の全てに当てはまります。

EMP*1	EMP001とEMP0001とEMP00001に当てはまりますが、EMPLOYEEには当てはまりません。
*OYEE	EMPLOYEEに当てはまりますが、EMP0001やEMP00001には当てはまりません。

➲ **構文**

DCI_MAPPING変数の構文。*<pattern>*の部分には有効なファイル名を表す文字列や、「*」や「?」を指定します:

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

3.6 複数のデータベースにマップする

DB_DCI_MAPで複数のデータベースの表を参照することは、複数のファイルあるいはDBMSへのCOBOLファイル接頭辞リンクを指定することにより可能です。

➲ **例**

データベースDBNAME (デフォルト)、DBCEDおよびDBMULTIの中のテーブル**idx1**を参照するには、DCI_CONFIG構成ファイル中に次のセッティングを加えてください。

```
DCI_DB_MAP      /usr1/CED       DBCED
DCI_DB_MAP      /usr1/MULTI     DBMULTI
```

複数のファイルを指定して、これらのデータベースに表**idx1**を作成します：

```
...
       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
```

ファイル接頭辞によって、複数のデータベースの表**idx-1**を読み取ります：

```
....
     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 トリガーを使う

COBOLトリガーは、非常に役立つDCIの強力な機能です。COBOLのトリガーは、どのユーザー、或いはプログラムがそれらを生成したかに関わらず、特定のI/Oイベントに呼応して予め定義したCOBOLプログラムを自動的に実行させます。
COBOLトリガーは、次のような場合に使用することができます。

- ビジネス・ルールを取り入れる
- COBOLの動きの監査記録を作成する
- 既存データから追加の値を引き出す
- 複数のファイルにまたがるデータをレプリケートする
- セキュリティ権限のプロシージャを実行する
- データ整合性を制御する
- 規則的でない整合性制約を定義する

I/Oイベントが発生した際に、呼び出されるCOBOLプログラム名を指定するために、次のXFDディレクティブを使用して、COBOLトリガーを定義します。

➲ **構文**

```
XFD DCI COMMENT COBTRIGGER “cobolpgmname”
```

➲ **例 1**

「cobolpgmname」は、大文字と小文字を識別します。CODE-PREFIXディレクトリ、又は現在のディレクトリにあります。I/OイベントはREAD (any)、WRITE、REWRITE、DELETE、OPENのいずれかです。COBOLトリガはBEFORE I/Oイベントを実行するOPENを除き、BEFOREおよびAFTER I/Oイベントを実行します。

```
$xfd dci comment cobtrigger “cobtrig”
```

➲ 例 2

「cobolpgmname」は、LINKAGE SECTIONルールに従います:

```
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.
```

➲ 例3

*Op-code*は、I/Oイベントに基づいてDCIから見積もられます。*record-image*は、I/Oイベントの前後のCOBOLのレコード値を含みます。rc-errorは、次の値を使ってCOBOL I/Oイベント・エラーを強制的に行うために使用します。

```
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 ビューを使う

DCIにより、表の代わりにDBMasterビューを使用できます。 この場合、DCIユーザーはビューを手動で作成し、次の制限に注意する必要があります。

- 単一の表ビューであり元の表のプロジェクションカラムに式、集合、UDFがない場合、ビューを開きすべてのDML操作を行います。
- 表の他の種類に関して、OPEN INPUTとして開き、READ操作のみを実行します。

➲ **例**

以下の例は、ビューを作成しそれを開く方法を示しています。 以下のように作成された**t2**および **t3**という名前の2つの表があると仮定します:

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

また、この表にはいくつかのデータが入っています。

```
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 シノニムを使用する

DCIにより、表またはビューの代わりにDBMasterシノニムを使用できます。 ユーザーは表、ビューまたはリモート・データベースの表またはビュ

ーにシノニムを作成できます。 ビューの同意語が単一表のビューでない場合、ユーザーはそのシノニムを持つ入力のみ開くことができます

3.10 リモート・データベースの表を開く

ユーザーはCOBOL SELECT文に特殊なトークン“@”を追加することによって、リモート・データベースの表またはビューにアクセスすることができます。例:

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

異なるユーザー名とパスワードを使用する場合、dmconfig.iniでDD_DDBMD=1を設定し、リモート・データベースのリンクを作成します。

➲ **例**

データベースdci_db1に接続し、データベースdci_db2の表にアクセスするとします。

1. dmconfig.iniでDD_DDBMD=1を設定します。
2. dci_db2に表を作成します。

   注 *dmSQLツールを使用して、dci_db2データベースに表を作成します。*

3. COBOLプログラムを使用してdci_db1に接続し、dci_db2に表を開きます。

```
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).
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 "AAAAAAAAA" TO C3.
      write tb1-record.
      initialize tb1-record.
      read tb1 next.
          display C1, C2, " ", C3.
      close tb1.
      accept omitted.
      stop run。
```

3.11 DCI_WHERE_CONSTRAINTを使用する

DCI_WHERE_CONSTRAINTは次のSTARTオペレーションのための付加的なWHERE条件を指定します。Acu4glと互換性をもつために、DCIはさらに**4gl_where_constraint**を支援します。

➲ **例：**

以下の例は**A**で始まる都市名を問い合わます：

```
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
  ....

   * to remove dci_where_constraint
```

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

4 EFDディレクティブ

ディレクティブは、COBOLファイル・ディスクリプタに記述されるコメントです。COBOLのファイル・ディスクリプタは、データベースの表をどのように構築するかを定義します。ディレクティブは、データをデータベースに定義される方法を調整し、データベースのフィールドに名前を割り当てします。ディレクティブは、.XFDファイルに名前を割り当てたり、バイナリ・ラージオブジェクト（BLOB）にデータを格納したり、コメントを追加したりするためにも使用します。

4.1 ディレクティブ構文を使う

各ディレクティブは、COBOLコードに関連する直前の1行に配置します。いずれのディレクティブにも、接頭語$XFDを付けます。7番目のカラムの$記号の直後には、XFDが付きます。

➲ **構文1**

次のコマンドは、未定義のCOBOL変数に一意のデータベース名を与えます。ディレクティブは、目的の行に直接に見つかります。この場合、COBOLの2番目のインスタンスは、変数*qty*で定義されます。

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

```
    03 QTY        PIC 9(03).
```

➲ **構文 2**

次のANSI-互換構文を使ってディレクティブを定義することも可能です。

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

➲ **構文 3**

複数のディレクティブを同時に指定することもできます。接頭語$XFDを付け、同じ行にディレクティブを記述し、スペースかカンマで仕切ります。

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

➲ **構文 4**

次の方法を使用することもできます。

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

4.2 XFDディレクティブを使う

COBOLのファイル・ディスクリプタが、データベースのフィールドにマップされる際に、ディレクティブが使用されます。接頭語$XFDは、データ・ディクショナリ生成の間に処理コマンドが使用されることを、コンパイラに示しています。

$XFD ALPHAディレクティブ

このディレクティブは、数値キーに非数値データLOW-VALUESや特殊なコード等を格納するために、COBOLプログラムで数値として定義されたデータ・アイテムを、データベースで英数字のテキスト（CHAR (n) n 1-最大カラム長）として扱うことができるようにします。

➲ **構文 1**

```
$XFD ALPHA
```

➲ 構文 2

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

$XFD ALPHAディレクティブを使わずに、「A234」のような非数値をキーに移動させます。

➲ 例 1

定義されたKEY ISコード・キーを作成してみます。以下にレコード定義があります。CODE-NUMは数値で、グループ・アイテムはデータベースで無視されるので、ここではキー・フィールドになります。

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

➲ 例 2

$XFD ALPHAディレクティブを使うと、「A234」のような非数値が変換され、「A234」は英数字値でCODE-NUMは数値なので、データベースでレコードは、拒否されません。

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

➲ 例 3

ここで、拒否される心配無く、次の演算を行うことができます。

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

$XFD BINARYディレクティブ

BINARYディレクティブは、フィールドのデータであらゆるタイプの英数字データ（LOW-VALUES）を使用できるようにします。LOW-VALUESの場合、例えばCOBOLでは、数値フィールドにLOWとHIGH-VALUESのいずれも使用します。でも、DBMasterでは不可能です。

BINARYディレクティブは、COBOLフィールドをDBMasterのBINARYデータ型に変換します。

➲ **構文1**

```
$XFD BINARY
```

➲ **構文2**

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

➲ **例**

以下は、LOW-VALUESをCODE-NUMに移動させます。

```
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).
```

$EFD COMMENT DCI SERIAL n ディレクティブ

このディレクティブは、シリアルのデータ・フィールドとオプションの開始番号「n」を定義するために使用します。DBMasterにシリアル番号を生成し、レコードを挿入し、シリアル・フィールドに0の値を与えます。新しい行を挿入し、0の値の代わりに整数値を与える場合、DBMasterはシリアル番号を生成しません。供給した整数値が最後に生成したシリアル番号

より大きい場合、DBMasterは与えた整数値から始まるシリアル番号を生成して連番をリセットします。

➲ 構文 1

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

➲ 構文 2

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

➲ 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ディレクティブ

このディレクティブは、READ WRITE REWRITEやDELETE等I/Oイベントのトリガーのようなプログラムを定義できるようにします。COBOLプログラムで定義したトリガーは、各I/Oイベントの前後に自動的に呼び出されます。

➲ 構文 1

```
$XFD COMMENT DCI COBTRIGGER “cblprogramname”
```

➲ 構文 2

```
*(( XFD COMMENT DCI COBTRIGGER “cblprogramname”))
```

$XFD COMMENTディレクティブ

このディレティブは、XFDファイルにコメントを識別します。この方法で、XFDファイルに情報を埋め込むと、他のアプリケーションからデー

タ・ディクショナリにアクセスすることができます。このディレクティブを使って、コメント形式に埋め込まれた情報は、DCIインターフェースの処理を妨げません。各コメントは、column 1に「#」記号を入れることで、XFDファイルで認識されます。

➲ **構文 1**

```
$XFD COMMENT text
```

➲ **構文 2**

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

$XFD DATEディレクティブ

DATEデータ型は、DBMasterがサポートしている特殊なデータ形式で、COBOLではサポートされていません。このデータ型のフィールドのプロパティを活用するためには、数値型のデータから変換してこのデータ型のフィールドのプロパティを活用できます。DATEディレクティブの目的は、データベースのフィールドに格納することです。このディレクティブは、他の数値とDATEを区別するので、プロパティをRDBMSのDATEに関連付けることができます。

➲ **構文 1**

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

➲ **構文 2**

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

*date-format-string*が定義されていない場合、6桁(或いは6文字)のフィールドがデータベースではYYMMDDのように回収されます。8桁のフィールドは、YYYYMMDDのように回収されます。

*date-format-string*は、希望するデータ形式を文字で構成した文字列です。

文字	解説
M	月 (01-12)

Y	年 (2、又は4桁)
D	月の内の日付 (01-31)
J	ユリウス暦(00000000-99999999)
E	1年のうちの日付 (001-366)
H	時間 (00-23)
N	分 (00-59)
S	秒 (00-59)
T	100分の1秒

図 4-1 date-format-string文字

date-format-stringにある各文字は、その場所にある情報のタイプを表すプレースホルダーとしてみなされます。文字は、各データ型が何桁であるかも決定します。

例えば、月は一般的には2桁で表わされますが、DATEの形式にMMMと定義する場合、結果のDATEは3桁になり、値の左に0が付け足されます。月がMと定義する場合、結果のDATEは1桁になり、左側が切り捨てられます。

ユリウス暦

ユリウス暦の定義は変化するので、DATEディレクティブでユリウス暦を柔軟に表わすことができます。多くのソースでは、ユリウス暦を1年の内の日、1月1日を001、1月2日を002というように定義しています。ユこの定義方法を使用する場合は、DATE形式にFEE（1年の内の日）を使うだけです。

その他は、ユリウス暦を特定の基礎日以降の日数で定義します。この定義は、DATEディレクティブでは文字Jで表されます。例えば、6桁のDATEフィールドはディレクティブ$XFD DATE=JJJJJJが実現できます。ユリウス暦でこの形式に使用する初期設定の基礎日は、*01/01/0001AD*です。

環境設定の変数DCI_JULIAN_BASE_DATEを設定することによって、ユリウス暦用に独自の基礎日を定義することができます。

DCIの有効範囲は次のとおりです。

01/01/0001から12/31/9999

DCIで無効であるDATE値を含むレコードが、COBOLプログラムによって書き込まれた場合、DCIはDCI_INV_DATE、DCI_MIN_DATE、DCI_MAX_DATE環境設定変数で指定した設定に基づいて、DATE値をDATEフィールドに挿入し、レコードに書き込みます。

COBOLプログラムで、NULLのDATEフィールドのある表からレコードに挿入しようと、COBOLレコードのそのフィールドにはゼロが挿入されます。

DATEフィールドに2桁の年がある場合、0年から19年までは、2000年から2019年として挿入され、20年から99年は1920年から1999年として挿入されます。変数DCI_DATE_CUTOFFの値を変更することで、このルールを変更することができます。また、DATEがキーにある時の無効なDATE値についての詳細は、環境設定変数DCI_MAX_DATEとDCI_MIN_DATEを参照して下さい。

注　*フィールドがキーの一部として使用されている場合、フィールドはNull値を取ることはできません。*

グループ・アイテムを使う

USE GROUPディレクティブを使用している場合、グループ・アイテムの前に、DATEディレクティブを置くことができます。

➲ **例 1**

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

date-purchasedカラムは8桁になり、データベースではYYYYMMDD形式のDATEデータ型になります。

➲ 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ディレクティブ

FILEディレクティブは、ファイルの拡張子がXFDのデータ・ディクショナリに名前を付けます。このディレクティブは、SELECT COBOL文で定義したものとは異なるXFD名を作成する時に必要になります。COBOLファイル名が一定でない時、このディレクティブが必要になります。

➲ 構文1

```
$XFD FILE=filename
```

➲ 構文2

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

➲ 例

この場合、ACUCOBOL-GTコンパイラは、CUSTOMER.xfdというXFDファイル名を生成します。

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

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

$XFD NAMEディレクトリ

NAMEディレクティブは、次の行で定義したフィールドにRDBMSのカラム名を割り当てます。DBMasterでは、全カラム名は一意で、そのサイズは128文字以下でなければなりません。このディレクティブは、互換性が無い、或いは重複する名前が生成されないようにするために使用します。

➲ **構文 1**

```
$XFD NAME=columnname
```

➲ **構文 2**

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

➲ **例**

COBOLフィールドの*cus-cod*は、DBMasterのRDBMSで、*customercode*という名前のカラムにマップされます。

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

$XFD NUMERICディレクティブ

NUMERICディレクティブは、その後のフィールドが英数字の場合、そのフィールドの値を符号無しの整数として扱うようにします。

➲ **構文 1**

```
$XFD NUMERIC
```

➲ **構文 2**

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

➲ 例

フィールド*customer-code*は、DBMasterの表ではINTEGERデータ型のデータとして格納されます。

```
$xfd numeric
        03      customer-code            PIC x(7).
```

$XFD USE GROUPディレクティブ

USE GROUPディレクティブは、DBMasterの表の単一カラムにアイテムの集まりを割り当てます。データベースのカラムに代入されるデータ郡の初期設定のデータ型は、英数字（CHAR (n)、n=1～最大カラム長）です。データが他のデータ型（BINARY、DATE、NUMERIC）に格納されている場合、このディレクティブを他のディレクティブと統合することができます。グループにフィールドを統合すると、データベースの処理速度は向上します。そのため、どのフィールドを統合するかの検討が必要です。

➲ 構文 1

```
$XFD USE GROUP
```

➲ 構文 2

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

➲ 例1

USE GROUPディレクティブを追加することで、*CODE-KEY*という名前の単一の数値カラムに、データが格納されます。

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

➲ 例 2

USE GROUPディレクティブは、他のディレクティブと合わせることができます。この例では、このフィールドは、データベースにあるDATEデータ型の単一のカラムにマップされます。

```
$(( 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ディレクティブ

VAR-LENGTHディレクティブは、COBOLフィールドを保存するために、BLOBフィールドをDBMasterに強制的に使用させます。これは、COBOLフィールドが標準的なデータ型（*Chapter 8のCOBOL Conversions*を参照のこと）のカラムの最大許容サイズに近いかそれ以上の場合、役に立ちます。

BLOBフィールドはキー・フィールドで使用することができないので、CHARのようなデータ型の通常のフィールドより検索が遅くなります。必要な場合にのみ、このディレクティブを使用することを推奨します。

➲ 構文1

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

➲ 構文2

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

➲ 例

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

ファイル名のための$XFD WHENディレクティブ

WHENディレクティブは、初期設定では通常構築されないDBMasterの特定カラムを生成するために使用します。コードにWHENディレクティブを指定することで、このディレクティブの直後に続くフィールド（グループ・アイテムの場合、下位のフィールドも含む）が、データベースの表で、明示的なカラムとして現れます。

データベースは、それらが明示的かどうかに関わらず全てのフィールドを格納し、検索します。キー・フィールドと最大レコードのフィールドも、自動的にデータベースの表では明示的なカラムになります。WHENディレクティブは、データベースの表に複数のレコード定義やREDEFINESを含もうとする時に、追加フィールドを確実に明示的なカラムにする場合にだけ使用します。

WHENディレクティブに、カラムをどのように使用するかという1つの条件を定義します。データベースの表で明示的なカラムにしたい追加フィールドをFILLERにしたり、キー・フィールドと同じ領域を占有させたりすることはできません。

- **構文1**

  イコール

```
$EFD WHEN field=value
```

- **構文2**

  小なりイコール

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

- **構文3**

  小なり

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

- **構文4**

  大なりイコール

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

➲ **構文5**

大なり

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

➲ **構文6**

等しくない

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

➲ **構文7**

OTHERは、記号「=」とのみ使用することができます。この場合、OTHERの後のフィールドは、WHEN条件や同じレベルにある条件リストに合致しない場合にのみ使用することができます。OTHERは、各レベルに一度だけ、1つのレコード定義の前と、各レコード定義の中に使用することができます。フィールドのデータが他のWHENディレクティブで定義した明示的な条件に合致しないような場合、OTHERと共にWHENディレクティブを使用する必要があります。さもないと、結果が定義されません。

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

➲ **構文8**

valueの部分は、括弧で表される明示的なデータ値です。フィールドは、既にCOBOLフィールドで定義されています。

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

➲ **例**

クオート(“”)内にある明示的なデータ値が有効です。

```
     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オプション

WHENディレクティブには、ランタイム時にWHENディレクティブの値に応じて表名を変更するためのTABLENAMEオプションがあります。

WHEN句でTABLENAMEオプションを使用する場合、DCI_DEFAULT_RULESとfilename_RULES DCIの環境設定変数を確認して下さい。

➲ **例 1**

「When」ディレクティブで2つの表名を使っているCOBOL FDの構造

```
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).
```

➲ **例 2**

DCIインターフェースは、例1の*INV-TYPE*フィールドの値に基づいて、「INV-TOP」と「INV-DETAILS」という名前の2つの表を生成します。DCIは、レコードをどこに入れるかを知るために、*INV-TYPE*フィールドの値をチェックします。

```
*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
```

前述のコードを実行すると、DCIは「INV-TOP」表に「TOP-ROW」レコードを、「INV-DETAILS」表に「DETAIL-ROW」を入れます。DCIが上記のレコードを読み込む際、順に従って読み込むか、或いはキーを使用して入力されたレコードにアクセスします。レコードのタイプを通じて、順番

とおりに読み込もうとする場合、DCI_DEFAULT_RULESをPOST、又はCOBOLにセットします。替わりに、レコードのタイプ内で順序どおりに読み込もうとする場合、DCI_DEFAULT_RULESをBEFORE、又はDBMSにセットします。

この規則を使うには、利点と欠点があります。COBOL ANSIの読み込みルールを100%採用するためには、「POST」か「COBOL」の方法を使用しなければなりません。但し、この方法はパフォーマンスを下げます（読み込むレコードが増加し、関連する全ての表が同時にオープンされるため）。

「BEFORE」か「DBMS」の方法を用いた場合は、$WHEN条件が読み込むレコードのレベルに合致した時、関連する表がオープンします。

➲ **例3**

言い換えると、前のレコードを使う場合、次の文をコードします。

```
      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.
```

「POST」や「COBOL」を使用する方法の場合、「open input」は表と「read next」の両方をオープンし、別々の表を通じて読み込みます。

➲ **例 4**

合致した表が、「start」文のレベルでオープンします。

「BEFORE」や「DBMS」を使用する方法の場合、コードを次のように修正します。

```
        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コメントDCI SPLIT

DCI SPLITディレクティブは、インタフェースが新しいDBMSテーブルを作成する1つまたは複数のテーブルの分割起点を定義するのに使用されます。

➲ **例1**

DCI SPLITディレクティブを使用するCOBOL FD構造

この例では、分割点の間のフィールドにINVOICE、INVOICE_A、INVOICE_Bという名前の3つのDBMasterテーブルが作成されます。

```
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 コンパイラとランタイム・オプション

この章では、どのファイル・システムを使用するかを指定するために、ACUCOBOL-GTにセットする環境設定について説明します。

5.1 ACUCOBOL-GTの初期設定ファイルシステムを使う

COBOLアプリケーションで開く既存のファイルは、ISCOBOL環境設定ファイルで定義したように、それぞれ各ファイル・システムに関連付けられています。COBOLアプリケーションで新しいファイルを作成した時、どのファイル・システムを使用するかを指定する必要があります。ACUCOBOL-GT環境設定ファイルをセットし、新しいファイルが選択したファイル・システムを使用できるようにします。

DEFAULT-HOSTの設定は、新しいファイルに他のシステムが指定されていない場合、どのファイル・システムを使用するかをACUCOBOLに伝えます。この変数をセットしない場合、ACUCOBOLでは初期設定としてVisionファイル・システムが使用されます。filename-HOST設定は、特定のファイルにファイル・システムをセットできるようにします。*filename*にファイル名を置き換えます。

ACUCOBOL-GT環境設定ファイルにある次の変数は、選択したファイル・システムを使用させるようにします。

➲ **構文1**

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

➲ **構文2**

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

5.2 DCIの初期設定ファイル・システムを使う

DBMasterの信頼性と、レプリケーション、バックアップ、整合性制約のような機能を活用するためには、ACUCOBOL-GTのVisionファイル・システムを使用せず、DEFAULT-HOST DCIを使うことを推奨します。ファイル・システムが指定されていない場合、Visionファイル・システムが初期設定として使用されます。

➲ **構文1**

この場合、新しいファイルが別のファイル・システムに割り当てられていない限り、全てDBMasterのファイルになります。

```
DEFAULT-HOST DCI
```

➲ **構文2**

ファイル・システムが定義されていない場合に、全ての新規ファイルがVisionファイルになるようにするためには、次のように指定します。

```
DEFAULT-HOST VISION
```

5.3 複数のファイル・システムを使う

`Filename-HOST`は、特定のファイル・システムに新しいファイルを関連付けるために使用します。これは、`1`つのデータファイルを`1`つのファイル・システムに関連付けるので、`DEFAULT-HOST`変数とは異なります。こ

の方法では、初期設定のファイル・システムと異なるファイル・システムを使用するファイルを使うことができます。

これを実現するために、環境設定ファイル「DEFAULT」値の代わりに、ディレクトリ名やファイル拡張子の無いファイル名を用います。DEFAULT-HOSTとFilename-HOSTは、一緒に使用することができます。

➲ **例**

この場合、file1とfile2はDBMasterを使用します。それ以外のファイルは、Visionファイル・システムを使用します。

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

5.4 環境変数を使う

プログラムの実行時にファイル・システムの設定を行えるようにするために、COBOLコードに次のように定義します。(*)は、ACUCOBOLのランタイム時にのみ使用されます。また、ファイル・システムの定義は通常ランタイム環境設定ファイルで行われ、COBOLプログラムでは変更しないことに留意して下さい。

注 *ACUCOBOL-Gコンパイラとランタイムの使い方についての詳細は、ACUCOBOL-GTの「ユーザー・マニュアル（2.1節と2.2節）」を参照して下さい。*

➲ **構文1**

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

➲ **構文2**

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

6 環境設定ファイルの変数

環境設定ファイルの変数は、DCIの標準的なふるまいを修正するために使用し、DCI_CONFIGと呼ばれるファイルに格納されます。

6.1 DCI_CONFIG変数を設定する

DCI_CONFIGという名前の環境変数に値を設定することで、環境設定ファイルに異なるアドレスを与えることができます。この環境変数に、環境設定ファイルを割り当てます。環境設定変数に指定したファイルが存在しない場合、DCIはエラーを表示せず、環境変数が割り当てられていないものとみなします。この変数は、COBOLのランタイム環境設定ファイルでセットします。

- **構文 1**

  UNIXでは、DCIはDCI_CONFIGファイルを探します。この環境変数は、DCI環境設定ファイルのパスと名前を作成するために使用します。Bourneシェルを使用している場合、次のコマンドを使うことができます。

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

- **構文2**

  DOSで、DCIのDCI_CONFIGに環境設定ファイルを割り当てます。<ファイル名>の欄には、絶対パスか相対パスと共に実際の環境設定ファイル名を指定します。

```
set DCI_CONFIG=<ファイル名>
```

➲ 構文3

UNIXでは、DCIはディレクトリ/home/testの「DCI」という名前のファイルを活用します。

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

DCI_CASE

COBOLのファイル名は、大文字と小文字を識別しません。一方、表名は大文字と小文字を識別します。このDCI_CASE環境設定の変数は、ファイル名をどのように表の名前に変換させるかを定義します。これを*lower*に設定すると、表名に変換されるファイル名が全て小文字になることを意味します。*upper*に設定すると、表名に変換されるファイル名が全て大文字になることを意味します。*ignore*にすることは、ファイル名が全て小文字あるいは全て大文字の表の名前に変換されないことを意味します。DCI_CASEのデフォルト設定は*lower*です。あなたのファイル名がDBCSワードである場合は、DCI_CASEを*ignore*に設定してください。

➲ 例

```
DCI_CASE IGNORE
```

DCI_COMMIT_COUNT

DCI_COMMIT_COUNT環境設定変数は、COMMIT WORK演算が行われる条件を指定します。有効な値は0と<*n*>の2つです。

DCI_COMMIT_COUNT=0

自動コミットは行われません。(初期設定値)

DCI_COMMIT_COUNT=<N>

この場合、WRITE、REWRITE、DELETE演算の数が、値<*n*>になった時に、COMMIT WORK文が実行されます。このルールは、ファイルが「output」や「exclusive」モードの場合にのみ適用されます。

DCI_DATABASE

DCI_DATABASEは、DBMasterのセットアップの際に、作成されるデータベース名を指定するために使用します。

➲ 例1

データベースがDBMaster_Testという名前の場合、環境設定ファイルには次のように入力します。

```
DCI_DATABASE DBMaster_Test
```

➲ 例2

データベース名が事前にわからない場合、ランタイム時に動的にセットします。このような場合、COBOLプログラムに下記のリストのような特殊なコードを記述します。以下のコードは、最初のOPEN文が実行される前に、実行されなければなりません。

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

➲ 例 3

異なるデータベース上の表にアクセスするため、DCI_DATABASEを使用することによって、複数のデータベースに接続し、動的にデータベースを切り替えることができます。

```
* 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-file
CALL "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

この変数は2桁の数値で、プログラムで2桁の年号から20世紀、或いは21世紀に翻訳するように指定します。

DCI_DATE_CUTOFFの初期設定は20です。この場合、2桁の年号が「20」（若しくは、この変数に与えた値）より小さい場合、元の2桁の数値に2000が付加され、21世紀になります。2桁の年号が「20」（若しくはこの変数に与えた値）以上の場合、元の2桁の数値に1900が付加され、20世紀になります。99/10/10のようなCOBOLの日付は、1999/10/10に変換されます。00/02/12のようなCOBOLの日付は、2000/02/12に変換されます。

DCI_DEFAULT_RULES

WHENディレクティブのための初期設定の管理方法は、複数の定義ファイルにあります。BEFORE文は、$WHEN条件が合致した際に、表がオープンすることを意味します。POST文は、COBOLアプリケーションが複数の定義ファイルを開いた時に、関連する全ての表がオープンすることを意味します。

有効な値は以下のとおりです。

POSTやCOBOL

BEFOREやDBMS

DCI_DEFAULT_TABLESPACE

この変数は、新しい表を格納する初期設定の表領域をセットするために使用します。データベースに既に存在する表領域のみを、指定することができます。この変数で表領域を指定すると、新しい表は初期設定のユーザー表領域に作成されます。

DCI_DISCONNECT

DCI_DISCONNECTはデータベース接続を切断するために使用されます。

➲ **例 1**

cobolプログラムに1つの接続だけがある場合は、次のコードを使用してデータベースから切断してください。

```
CALL "DCI_DISCONNECT".
```

➲ **例 2**

cobolプログラムに複数の接続がある場合、次のコードを使用して特定のデータベースから切断してください。

```
CALL "DISCONNECT" USING "DBSAMPLE5"
```

DCI_DUPLICATE_CONNECTION

DCI_DUPLICATE_CONNECTIONは、同一のCOBOLアプリケーションが同一の表を開き、同一のレコードを2回ロックする必要が場合に、同じ表を同じCOBOLプロセスで、しかし異なるデータベース接続を使用して開くことによって、ロックを得るために使用されます。

デフォルト値はオフです(0)。

➲ **例**

COBOLアプリケーションが異なるデータベース接続を使用して表のロックを得ることを可能にします：

```
DCI_DUPLICATION_CONNECTION 1
```

DCI_GET_EDGE_DATES

DCI_SET_EDGE_DATEはユーザがDATEフィールドに最低値/最高値を入力した場合、表示される値を指定するために使用されます。例えば、ユーザがCOBOLプログラムにDATEフィールドに対する最低値/最高値を入力する時、00010101/99991231の入力によって、日付はCOBOLの最低値/最高値00000000/99999999を使用して表示されるでしょう。この変数が使用される時、DATEフィールドの最低値/最高値はデータベースの最低値/最高値00010101/99991231を使用して表示されるでしょう。DATEフィールドがキーの一部である場合も、この規則は適用されます。デフォルト値はオフです。

➲ **シンタックス：**

次の行は、**dci.cfg**ファイルに追加します:

```
DCI_GET_EDGE_DATES 1
```

DCI_INV_DATE

この変数は、不正な日付フォーマットがデータベースに書き込まれた際に、起こりえる問題を回避するために、無効な日付(2000/02/31のような)を設けます。この変数の初期設定は、99991230(9999年12月30日)です。

DCI_LOGFILE

この変数は、DCIログファイルのパス名を指定します。DCIログファイルには、インターフェースで実行される全てのI/O演算が書き込まれます。*/tmp*ディレクトリに格納される*dci_trace.log*ログファイルは、デバックのために使用します。ログファイルを使うと、DCIのパフォーマンスが下がりますので、必ず必要な場合にのみ使用することが理想的です。

➲ **例**

Config.iniファイルのログファイルのエントリ例は以下のように示します:

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

DCI_LOGIN

DCI_LOGINは、データベース・システムに接続するためのユーザー名を指定する変数です。初期設定値はありません。そのため、ユーザー名が定義されていない場合、ログインは使用されません。

DCI_LOGIN変数で指定したユーザー名には、データベースに対してRESOURCE以上の権限が必要です。加えて、ユーザーには既存のデータ表への許可が必要です。新しいユーザーは、JDBA Tool、或いはdmSQLを使って作成します。

注 *ユーザー作成についての詳細は、「JDBA Toolユーザーガイド」、又は「データベース管理者参照編」をご覧下さい。*

➲ **例**

Config.cfgファイルのユーザー名、JOHNDOEの入力例は以下のように示します:

```
DCI_LOGIN JOHNDOE
```

DCI_JULIAN_BASE_DATE

DATEディレクティブと共に使用するこの変数は、ユリウス暦の計算の基礎日をセットするために使用します。フォーマットはYYYYMMDDです。この変数の初期設定値は、西暦1年1月1日です。

この変数の1つの用法は、1850以降の日付を使用しているCOBOLプログラムです。これらの日付は、DATEディレクティブを$XFD DATE=JJJJJJ (dateフィールドには同じ数の文字数がなければなりません)にセットし、DCI環境設定変数DCI_JULIAN_BASE_DATEを18500101にセットすることでデータベースに格納することができます。

DCI_LOGTRACE

この変数は、トレースログに様々なレベルを設けます。

- 0: トレースしない。

- 1: 接続トレース
- 2: レコードI/Oトレース
- 3: 完全トレース
- 4: 内部デバッグ・トレース

DCI_MAPPING

この変数は、DCIシステムにある特定のXFDファイルと特定のファイル名を関連付けるために使用します。これにより、1つのXFDファイルを複数のファイルと関連させることもできます。「*pattern*」の部分には、有効なファイル名の文字を指定します。また、複数の文字を意味するワイルドカード「*」記号や、1文字に相当する疑問符「?」も含むことができます。複数回使うことができます。

➲ **構文**

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

➲ **例 1**

パターンに「CUST*1」と指定すると、「CUST01」、「CUST001」、「CUST0001」、「CUST00001」のようなファイル名は、XFDファイル「customer.XFD」に関連づけられます。

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

➲ **例2**

パターンに「CUST????」と指定すると、「CUSTOMER」や「CUST0001」のようなファイル名は、XFDファイル「cust.XFD」に関連付けられます。

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

DCI_MAX_ATTRS_PER_TABLE

DBMasterの表は、2000カラムまでしか格納することができません。2000を超えるフィールドがあるCOBOLファイルは、全てのフィールドを表のカラ

ムにマップすることができません。DCI_MAX_ATTRS_PER_TABLE変数は、表を2つ以上の別々の表に分割するフィールドの数を指定します。結果的に生成された表には一意の名前が必要なので、DCIは表名にアンダースコア(_)とその後に連続する文字(A、B、C…)を付加します。

➲ **例 1**

COBOLのファイルには、300フィールドと次の文があります:

```
SELECT FILENAME ASSIGN TO “customer”
```

➲ **構文**

次の行は、**dci.cfg**ファイルに追加します:

```
DCI_MAX_ATTRS_PER_TABLE = 100.
```

➲ **例 2**

3つの表が以下の名前で生成されます:

```
customer_a

customer_b

customer_c
```

DCI_MAX_BUFFER_LENGTH

DCI_MAX_BUFFER_LENGTHは、DCI_MAX_ATTRS_PER_TABLEが実行する機能のように、COBOLデータ・レコードを複数のデータベースの表へ分割する目的で使用されます。しかしながら、表がどこで分割されるか決めるために使用されるしきい値は、バッファの長さによって決定されます。デフォルト値は32640です。

➲ **例 1**

COBOLレコード・サイズは、9000バイトのデータおよび次のSQL文を含んでいます :

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

➲ シンタックス：

次の行は、dci.cfgファイルに追加します:

```
DCI_MAX_BUFFER_LENGTH 3000
```

➲ 例 2

3つの表が以下の名前で生成されます:

```
customer_a

customer_b

customer_c
```

DCI_MAX_DATE

この変数は、無効な日付がデータベースに誤って書き込まれた際の問題を回避するために、高い値の日付を設けるために使用します。この変数の初期設定は、99991231（9999年12月31日）です。

DCI_MIN_DATE

この変数は、無効な日付がデータベースに誤って書き込まれた際の問題を回避するために、低い値や0、空白の日付を設けるために使用します。この変数の初期設定は、00010101（西暦1年1月1日）です。

DCI_NULL_ON_ILLEGAL_DATA

DCI_NULL_ON_ILLEGAL_DATAは、COBOLデータがデータベースによって不正と見なされるかどうか、格納される前に判断します。値を1にすると、不正な全データ（キー・フィールドを除く）は格納される前にNullに変換されます。値を0（初期設定値）にすると、次のような変換が行われます。

- 不正なLOW-VALUESは、最小値(0、又は-99999...)、或いはDCI_MIN_DATEの初期設定値として格納されます。

- 不正なHIGH-VALUESは、最大値(99999...)、或いはDCI_MAX_DATEの初期設定値として格納されます。
- 不正なSPACESは、ゼロとして格納されます（日付フィールドの場合、DCI_MIN_DATE）。
- 不正なDATE値は、DCI_INV_DATEの初期設定値として格納される。
- 不正なTIMEは、DCI_INV_DATEの初期設定値として格納されます。
- キー・フィールドにある不正なデータは、環境設定変数の値に関わらず、常に変換されます。

DCI_PASSWD

ユーザー名が一旦DCI_LOGIN変数を経由して定義されると、データベースのアカウントに関連付けられます。パスワードは、このデータベース・アカウントに割り当てられます。これは、変数DCI_PASSWDを使って行うことができます。

➲ **例1**

データベースのアカウントに、パスワードSUPERVISORを割り当てる場合、環境設定ファイルに次のように指定します。

```
DCI_PASSWD SUPERVISOR
```

➲ **例 2**

プログラム実行の際に、ユーザーからパスワードを受け付けることもできます。この方法は、信頼性がより高くなります。これを行うためには、DCI_PASSWD変数が回答に応じてセットされなければなりません。

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

但し、この場合、環境変数の読み込みと書き込むを行うために、ネイティブAPIの呼び出しを完了しなければなりません。

➲ **構文 1**

この文は、環境変数を書き込み、又は更新するために、COBOLプログラムで使用します。

```
CALL “DCI_SETENV” USING “environment variable”, value.
```

➲ **構文 2**

この文は、環境変数の読み込みを行うために、COBOLプログラムで使用します。

```
CALL “DCI_GETENV” USING “environment variable”, value.
```

DCI_STORAGE_CONVENTION

この変数は、COBOLのストレージ規則をセットします。現在、DBMasterでは、4つのタイプの値をサポートしています。

DCI

IBMのストレージ規則を選択します。これは、RM/COBOL-85を含む様々な他のCOBOLバージョンに対して同様、IBM COBOL互換です。X/Open COBOL標準とも互換しています。

DCM

Micro Focusのストレージ規則を選択します。Micro Focus "ASCII"のsign-storageオプションが使用されている場合（Micro Focusの初期設定）、Micro Focus COBOLに互換します。

DCN

異なる数字フォーマットが使用されるようになります。このフォーマットは、正数のCOMP-3アイテムが、"x0C"の代わりに正数の符号付き値として"x0B"を使用することを除き、"-DCI"オプションが使用された場合に使用されるものと同じです。このオプションは、NCR COBOLに互換しています。

DCA

ISCOBOLのストレージ規則を選択します。これは初期設定の設定です。この規則は、RM/COBOL（RM/COBOL-85では無く）や、ISCOBOLの以前のバージョンによって生成されたデータにも互換性があります。

DCI_USEDIR_LEVEL

この変数が > 0にセットされた場合、表の名前に加えてディレクトリを使用します。

➲ **例 1**

次の文では、/usr/test/01/clients は、01clientsを意味します。

```
DCI_USEDIR_LEVEL 1
```

➲ **例 2**

次の文では、/usr/test/01/clientsは、test01clientsを意味します。

```
DCI_USEDIR_LEVEL 2
```

➲ **例 3**

次の文では、/usr/test/01/clientsは、usrtest01clientsを意味します。

```
DCI_USEDIR_LEVEL 3
```

DCI_USER_PATH

変数DCI_USER_PATHは、DCIでファイルを探す際にに、ユーザー名や名前を指定することができるようにします。ユーザー引数は、ファイルやシステム上のユーザー名に関しては、ピリオド(.)を使用することができます。

➲ **構文**

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

ファイルに与えたOPEN文のタイプが、この設定の結果を決定します。

OPEN文	DCI_USER_PATH	DCI検索順	結果
OPEN INPUT、又はOPEN I/O	Yes	1-USER_PATHにあるユーザー一覧 2-現在のユーザー	最初の有効なファイルがオープンします。
OPEN INPUT、又はOPEN I/O	No	DCI_LOGINに関連付けられているユーザー	有効なユーザー/ファイル名を持つ最初のファイルがオープンします。
OPEN OUTPUT	Yes、又はno	ユーザーを検索しない	DCI_LOGINに関連する名前で、新しい表が作成されます。

図 6-1 OPEN文のタイプ

DCI_XFDPATH

DCI_XFDPATHは、データ・ディクショナリが格納されるディレクトリの名前を指定します。初期設定値は、現在のディレクトリです。

➲ **例 1**

ディレクトリ*/usr/dbmaster/dictionaries*にデータ・ディレクトリを格納する場合は、環境設定ファイルに次のように入力します。

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

➲ **例2**

複数のパスを指定する必要がある場合、各ディレクトリをスペースで区切って指定します。

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

➲ **例3**

WIN-32環境では、「埋め込みスペース」はダブルクオート「"」を使って指定することができます。

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

DCI_XML_XFD

DCI_XML_XFD は 1 を設置すると、DCI ランタイムは ACUCOBOL XML フォーマット XFD ファイルを使用する必要があることを示します。0 を設定すると、ユーザーは以前の ACUCOBOL コンパイルから生成する古い xfd フォーマットを使用すること、または ACUCOBOL ユーザーは–Fx3 で自分の COBOL プログラムをコンパイルしていることを示します。ディフォルト値は 0 です。

ACUCOBOL-GT コンパイラは XML フォーマットで"$XFD COMMENT directive"ような XFD 構文 をサポートしないからです。だから、XML フォーマットを使用しないことを推薦します。

➲ **例 1**

以下の構文はChapter 4.2にサポートしません。

```
$XFD COMMENT  DCI SERIAL  n directive

$XFD COMMENT DCI COBTRIGGER Directive

$XFD COMMENT Directive

$XFD COMMENT DCI SPLIT
```

*<filename>*_RULES

マルチ定義ファイルの初期設定を管理します。*<filename>*は、実際のファイル名を表しています。

➲ 例

次のコマンドが使用された際、CLIENTファイルを除き、全てのファイルはPOSTルールを使用します。

```
DCI_DEFAULT_RULES       POST

CLIENT_RULES            BEFORE
```

DCI TABLE CACHE 変数

クライアント/サーバーのネットワーク・トラフィックを減少するため、初期設定によって、DCIはクライアント・データ・バッファからあらかじめデータを読み取ります。初期設定の前読み取りバッファ最大値は8kb/(レコード・サイズ)あるいは5つのレコードのどちらか小さい方です。

もしユーザのアプリケーションが小さな表を読み込む場合、8 kb/(レコード・サイズ)未満である少数のレコードを読むことはありえるでしょう。例えば、平均レコード・サイズが20バイトでレコード数が合計1000の表については、DBMasterは一度に400のレコード(8kb/20) 読み込みますが、ユーザのアプリケーションは4つあるいは5つのレコードを読んだだけで、すぐSTARTステートメントを再び呼び出すとします。この場合、次の変数でキャッシュサイズを縮小させて、パフォーマンスを改善することができます。これらの変数を使用する場合アプリケーションおよびデータの振る舞いを注意深く考慮してください。さもないと、それはネットワーク・トラフィックを増加させて、パフォーマンスの減少を引き起こすかもしれないからです。

下記はDCI_CONFIGファイルの中でセットするべき3つのDCI_CACHE変数です：

- DCI_DEFAULT_CACHE_START—STARTあるいはREAD用に最初にキャッシュする読み取りレコードをセットします。初期設定値は8kb/(レコード・サイズ)、あるいは5つのレコードの大きい方です。

- DCI_DEFAULT_CACHE_NEXT―それは、STARTあるいはREAD用に最初にキャッシュされたレコードが読込まれたか廃棄された後、次の読み取りレコードをセットします。初期設定値はは8kb/(レコード・サイズ)、あるいは5つのレコードの大きい方です。
- DCI_DEFAULT_CACHE_PREV―それは、STARTあるいはREAD用に最初にキャッシュされたレコードが読み込まれたか廃棄された後、直前のレコードをキャッシュするための読み取りレコードをセットします。

初期設定値はDCI_DEFAULT_CACHE_NEXT/2です。

DCI_CONFIGの中でこれらの変数をセットすることは、ユーザのアプリケーションのすべての表に影響をあたえるでしょう。

➲ **例**

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

DCI_TABLESPACE

この変数では、表を作成する表領域を定義します。 ワイルドカードでも機能します。 まず、表を作成することが大切です。 表が作成されたら、DCIはこの変数の値をモニターしません。

➲ **例1**

表領域tbs1に顧客表を作成する場合:

```
DCI_TABLESPACE customer =tbs1
```

➲ **例2**

表領域tbs1にcustで始まるすべての表を作成する場合。

```
DCI_TABLESPACE cust*=tbs1
```

DCI_AUTOMATIC_SCHEMA_ADJUST

この変数は、XFDが表スキーマと異なるとき、表スキーマ定義を変更するようにDCIに指示します。 この変数は分割表と互換性がありません(2000より大きいカラム数を持ち、そのレコードサイズが32KBより大きい表は、BLOBフィールドを排除します)。

この変数の可能な値は次の通りです:

0　初期設定、何もしない

1　表に新しいフィールドを追加し、XFDにないフィールドを削除します

2　表に新しいフィールドを追加し、XFDにないフィールドを削除しません

DCI_INCLUDE

この変数は、追加DCI_CONFIGファイルを含むことができます。 COBOL COPY文として機能し、さらに複雑な設定の定義を可能にします。

➲ 例

```
DCI_INCLUDE /etc/generic_dci_config
```

DCI_IGNORE_MAX_BUFFER_LENGTH

この変数はDCI_MAX_BUFFER_LENGTH値の設定を無視するために使用されます。 記録長が32K以上の場合、表を分割することはありません。 初期設定はオフです。

DCI_NULL_DATE

DCIがこの値で日付フィールドに書き込むときNULLを書き込み、DCIがNULL値で日付を読み込むとき、COBOLプログラムにDCI_NULL_DATEを返します。

DCI_NULL_ON_MIN_DATE

この変数を1に設定すると、次のアクションが発生します。 COBOLプログラムが0の値をDATEフィールドに書き込むとき、値はNULLとしてデータベースに保存されます。 同様に、NULL値がデータベースから読み込まれるとき、COBOL FDは0になります。

DCI_DB_MAP

この変数は、異なるデータベースの表として異なるディレクトリにファイルをマップするために使用されます。 詳細については、複数のデータベースにマップするを参照してください。

DCI_VARCHAR

この変数を1に設定すると、以下のアクションが発生します: プログラムが (OPEN OUTPUTの動詞により) 新しいテーブルを作成すると、CHARとして作成されたすべてのフィールドがVARCHARになります。

DCI_GRANT_ON_OUTPUT

この新しいオプションを使うと、テーブル作成 (OPEN OUTPUT) 中にGRANTコマンドを指定できます。

➲ 例

```
DCI_GRANT_ON_OUTPUT user1=SELECT

DCI_GRANT_ON_OUTPUT user2=SELECT,INSERT,UPDATE
```

outputがオープンした後、user1が表からデータが選択できます、 user2はデータが選択できて、変更することもできます。

7 DCI関数

このセクションでは、COBOLプログラムで呼び出すことのできるDCI機能を一覧表示します。 これらの機能を有効にするには、これらの関数をsub85.cに追加し、DCIランタイムを再編成しなければなりません。

7.1 DCI関数を呼び出す

COBOLプログラムに追加してDCI関数を呼び出します：

```
CALL “dci_function_name” USING variable [, variable, ...]
```

COBOLプログラムに呼び出すことができます。

DCI_SETENV

この関数は、環境変数を書き込みます。

➲ **構文**

```
CALL “DCI_SETENV” USING “environment variable”, value
```

➲ **例**

```
call "DCI_SETENV" using  "DCI_DATABASE" , "DBSAMPLE5"
```

DCI_GETENV

この関数は、環境変数を書き込んだり更新します。

➲ 構文

```
CALL “DCI_GETENV” USING “environment variable”, variable
```

➲ 例

```
ALL "DCI_GETENV" USING "DCI_DATABASE”, ws_dci_database
```

DCI_DISCONNECT

この機能は、データベース接続を切断します。

➲ 例1

cobolプログラムに接続が1つしかない場合、次のコードを使用してデータベースから切断してください。

```
CALL "DCI_DISCONNECT"
```

➲ 例2

COBOLプログラムに接続が2つ以上ある場合、次のコードを使用して特定のデータベースを切断してください。

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

DCI_GET_TABLE_NAME

この関数はパスしたCOBOL名の表名を取得します、(有効な表名をいつでもすぐ知ることができるとは限りませんが、それはこれらの場合には操作が1つではないためです: XFD WHEN ... TABLENAME.).

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

DCI_SET_TABLE_CACHE

この関数は、表のキャッシュを動的に変更し、START または READ文の前でこれらの変数を設定します。

➲ 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

この関数は、DCI_BLOB_GET または DCI_BLOB_PUTを呼び出した後でエラーを取得します。

➲ 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

この関数は、COBOLプログラムでBLOBデータのより効率的な使用を可能にします。 DCI_BLOB_GETコマンドを使用することによって、COBOLを使用するBLOBデータに素早く効率的にアクセスできます。
DCI_BLOB_GETコマンドを使用しているとき、以下に一覧表示された規則に従う必要があります。

- ユーザーの表は、BLOB(長いvarchar/長いvarbinary)データタイプを持つ必要があります。
- ユーザーは、COBOL FDのBLOBタイプでフィールドを設定することができません。
- ユーザーはREAD、READ NEXTまたはREAD PREVIOUSコマンドの後でDCI_BLOB_GETコマンドのみを使用することができます。

➲ **例**

ユーザーは以下の構文による表を作成します:

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

次は、COBOLプログラムにおけるDCI_BLOB_GETの使用に関する、実際的なアプリケーションを提供します。

```
       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

この関数は、COBOLプログラムでBLOBデータのより効率的な使用を可能にします。 DCI_BLOB_PUTコマンドを使用することによって、BLOBにデータを挿入できます。 DCI_BLOB_PUTコマンドを使用しているとき、以下に一覧表示された規則に従う必要があります。

- ユーザーの表は、BLOB(長いvarchar/長いvarbinary)データタイプを持つ必要があります。
- ユーザーは、COBOL FDのBLOBタイプでフィールドを設定することができません。
- ユーザーは、WRITE または REWRITEコマンドの前でDCI_BLOB_PUTコマンドのみを呼び出すことができます。

- ユーザーがWRITEコマンドの前でDCI_BLOB_PUTを呼び出さない場合、初期設定値がblobカラムに挿入されます。
- ユーザーがREWRITE文の前でDCI_BLOB_PUTを呼び出さない場合、blobカラムは更新されません。

➲ 例

次は、COBOLプログラムにおけるDCI_BLOB_PUTの使用に関する、実際的なアプリケーションを提供します。

まず、ユーザーは表を作成します:

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

表が作成されると、ユーザーは次を続行します。

```
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

この関数は、WRITE文の後でシリアル値を取得します。

➲ 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

この関数は、DCIがキャッシュに保存しているXFD画像を取り除くために使用されます。 表がこの接続によりすでに開かれた後で変更されたXFDをリロードする上で、役に立ちます。

```
CALL "DCI_FREE_XFD"
```

DCI_UNLOAD_CONFIG

この関数は、既存の配置をアンロードします。 DCI_SETENVを呼んで新規な配置を作成することができます。ThinClient 環境に非常に役立ちます.

```
call "DCI_UNLOAD_CONFIG" .
```

8 COBOLの変換

この章では、DCIのためのデータ許容範囲の一覧と、COBOLのデータ型をどのようにDBMasterのデータ型にマップするかを指定する表を合わせて説明します。

変換の際にDCIでは、トランザクションが実施されます。全てのI/O処理はトランザクションを使って行われます。DCIは、AUTOCOMMITをOFFにセットし、DBMasterのトランザクションを管理し、ユーザーにレコードを変更できるようにします。DCIは、START TRANSACTION、COMMIT/ROLLBACK TRANSACTIONのような、COBOLトランザクション文を完全にサポートしています。

DCIでは、レコードの暗号化、レコード圧縮、交互の照合シーケンスを使用することができません。これらのオプションがコードに含まれている場合は、無視されます。DCIは、XFDデータ定義の「P」PICture編集関数もサポートしていません。ファイル名は全て小文字に変換されます。

DBMasterデータベースの設定	許容範囲
索引キーのサイズ	4000
キーあたりのカラム数	32
CHARフィールドのサイズ	32640
同時RDBMS接続	4800
カラム名の文字数	128

DBMasterデータベースの設定	許容範囲
1つのプロセスで同時にオープンされるデータベースの表数	256

図 8-1 DBMasterのデータベース設定の許容範囲表

8.1 特別なディレクティブを使う

DBMasterでは、Visionのファイル・システムと同じソートや検索シーケンスを使うことができます。但し、符号付き数値データを含む各キー・フィールドの前には、BINARYディレクティブを置く必要があります。highとlow値は、キー・フィールドを複雑にします。

DBMasterのOID、VARCHAR(サイズ)、FILEデータ型をサポートする特別なディレクティブは、現在ありません。

DBMasterのデータ型	ディレクティブ
DATE	XFD DATEを使う
TIME	XFD DATEを使う
TIMESTAMP	XFD DATEを使う
LONGVARCHAR	XFD VAR-LENGHを使う
LONGVARBINARY	XFD VAR-LENGH*を使う
BINARY	XFD BINARYを使う
SERIAL	XFD COMMENT DCI SERIAL を使う

図 8-2DBMasterのデータ型をサポートする特別なディレクティブ

8.2 COBOLのデータ型をマップする

DCIは、DBMasterのデータベースの表にある全カラムを作成する際に、COBOLに最も適合するデータ型を生成します。COBOLにあるデータ型の

データは、データベースのカラムに含むことができます。最初に、指定したXFDディレクティブがチェックされます。

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

図 8-3 COBOLからDBMasterのデータ型への変換表

8.3 DBMasterのデータ型をマップする

DCIは、ネイティブ・データ型からCOBOLデータ型にCOBOL-風のMOVEを行うことで、データベースからデータを読み込みます（ほとんどにCHAR相当があるので、それらをdmSQLで表示させることができます）。

データベースのデータ型をCOBOLのデータ型に完全に合致させる必要はありません。PIC X(nn)は、データベースのCHAR相当のデータ型を持つ各カラムに使用することができます。PIC 9(9)は、データベースのINTEGER型により合致するCOBOLの型です。データベースのデータ型について詳しくなると、マッチするCOBOLのデータ型をより柔軟に見つけることができます。例えば、DBMasterのデータベースにあるカラムに、ゼロから99 (0-99)の間の値しかない場合、対応するCOBOLのデータ型はPIC 99が充分となります。

COMP-typesの選択は、使用するCOBOLのデータにほとんど影響が無いので、プログラマーの考慮に委ねます。BINARYデータ型は、COBOLには無関係なので、通常変更無く書き込まれます。但し、BINARYカラムを綿密に分析すると、異なる解決策が見出されるかもしれません。DECIMAL、NUMERIC、DATE、TIMESTAMPデータ型に完全に合致するCOBOL型はありません。それらはデータベースから文字形式で戻されます。そのため、最適のCOBOLの同等データ型は、USAGE DISPLAYです。

以下の表は、データベースのデータ型とCOBOLデータ型の最適の組み合わせを表しています。

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

DBMaster	COBOL		DBMaster	COBOL
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)

図 8-4 DBMasterからCOBOLデータ型への変換表

8.4 ランタイム・エラーのトラブル・シューティング

ランタイム・エラー の形式は、「9D, xx」です。「9D」は、ファイル・システムのエラーを意味し、（FILE STATUS変数で報告される）、「xx」は二次エラー・コードを表しています。

エラー	定義	解説	解決策
9D,01	There is a read error on the dictionary file.	XFDファイルの読み込みの際にエラーが発生しました。XFDファイルが壊れています。	-XFDで再コンパイルし、ディクショナリ・ファイルを再生成します。
9D,02	There is a corrupt dictionary file. The dictionary file cannot be read.	COBOLファイル用のディクショナリ・ファイルが壊れています。	-XFDで再コンパイルし、ディクショナリ・ファイルを再生成します。
9D,03	A dictionary file (.xfd) has not been found.	COBOLファイル用のディクショナリ・ファイルが見つかりません。	DCI_XFDPATH 環境設定変数に正しいディレクトリを指定します（–Fxを使った再コンパイルが必要かもしれません）。
9D,04	There are too many fields in the key.	1つのキーに16を超えるフィールドがあります。	キー定義をチェックして、不正なキーを再編成し、–Fxで再コンパイルします。

9D,12	There is an unexpected error on a DBMaster library function.	DBMasterのライブラリ関数が予期しないエラーを戻しました。	
9D,13	The size of the “xxx” variable is illegal.	FDにある基本のデータ・アイテムが255バイトを超えています。	
9D,13	The type of data for the“xxx” variable is illegal.	使用しているデータ型に合う、DBMasterのデータ型がありません。	
9D,14	There is more than one table with the same name.	同じ名前を持つ表が複数あります。	

図 8-5 DCIの二次エラー表

8.5 ネイティブSQLエラーのトラブル・シューティング

DBMaster用にDCIを使用している際に、データベースからネイティブSQLエラーが返されることがあります。 エラー番号と用語は、データベースによって異なるかもしれません。

番号	定義	解説	解決策
9D, ???,????	Invalid column name or reserved word.	カラムには、データベース用に予約された用語を使って名前が付けられます。	CREATE TABLEのファイル・トレースを、データベースが予約している用語リストと比較します。NAMEディレクティブを、無効なカラムのFDフィールドに適用し、新しいXFDファイルを作成するために再コンパイルします。
9D, ??	Journal full, command rolled back to internal savepoint		COBOLプログラムの中で "start transaction/commit/rollback"コードを加えてください。あるいは、DCI構成ファイルの中でDCI_COMMIT_COUNTを

			セットしてください。
9D, 5503	無効なキー名	表には索引がありません	正しい索引名とカラムで索引を作成します
9D, 5504	ホスト変数を使用できません		ユーザーはrunsql.acuでホスト変数を使用できません
9D, 5508	INSERT/UPDATE/DELETE権がありません	ユーザーは挿入/削除/更新権がない表のI-Oを開くことができません	そのテーブルでOPEN INPUT
9D, 5512	選択問合せを発行できません		ユーザーはrunsql.acuで選択文を発行できません
9D,5513	dciを接続しているときクライアント-サーバーバージョンが一致しません	ユーザーのDCIランタイムはdmserverより新しい日付です	新しいDCIランタイムを実行する前に、ユーザーはそのdmserverをアップグレードする必要があります
9D, 5514	無効なカラム数	COBOL FDカラム数は表のカラム数より大きい数です	ユーザーはFDカラム数と表のカラム数をチェックする必要があります
9D, 5515	無効なXFDカラム名またはデータタイプと長さが一致しません	COBOL FDカラム名またはカラムタイプが表定義に一致しません	FDと表定義を比較します。COBOL FDを変更または表を修正して、このプログラムを回復してください。
9D, 5518	DCI blobデータがNullです	ユーザーがカラムからblobを取得するとき、データはNULLです	
9D, 5519	DCI blobファイルが存在しない		ユーザーはblobファイルが存在したことを確認すべきです。

図 8-6ネイティブSQLエラー表

8.6 Visionファイルを変換する

DCIには、COBOLファイルをRDBMSの表に変換するためのサンプル・プログラムがあります。DCI_MIGRATEプログラムを使う前に、変換するVisionファイルと、Visionファイル用のXFDデータ・ディクショナリが必要です。又、DCIにリンクするACUCOBOLランタイム・システム4.3以上が

インストールされ、DCI_MIGRATEオブジェクト・プログラムが既に整っていることが前提です。

DCI_Migrateを使う

これは、COBOLのvisionファイルをDBMasterの表に変換する汎用プログラムです。DBMasterを使うために必要な、最小限のDCI環境設定を正しくセットします（DCI_LOGIN、DCI_DATABASE、DCI_PASSWD等）。又XFDファイル名が*dbm_table_name*のXFDファイルに合致させます。或いはDCI_MAPPINGを使って、名前とロケーションを定義します。

プログラムDCI_MIGRATEは、visionファイルを読み込み、DCIを通じてDBMasterのタプルに書き込みます。更に移植後、Visionレコードと DBMasterの行を読み込むことによってそれらを比較し、全てのレコードが正しいかをチェックします。

DCI_MIGRATEプログラムは、次のようにレポートします。

- Total record read successful
- Total record write successful
- Total record read unsuccessful
- Total record write unsuccessful
- Total record compared successful
- Total record compared unsuccessful

DCI_MIGRATEオプション	結果
--help	オンライン・ヘルプを表示します。
--nowait	interactive modeの際、ユーザーの確認を待ちません。
--noverify	確認プロセスを省略します。

--nomigrate	移植プロセスを省略します。
--visdbm	visionファイルをDBMasterの表に変換します(初期設定)。
--dbmvis	DBMasterの表をvisionファイルに変換します。

図 8-7 DCI_MMIGRATEオプションの結果表

➲ **構文 1**

vision_file_name は、変換されるVisionファイルの名前、*dbm_table_name* はDBMasterの表の名前です。

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

➲ **構文 2**

DCI_MIGRATEという名前の環境変数を「yes」に設定すると、レポートをOFFにします。以降レポートは、「dbm_table_name.log」というファイルを付加します。

```
DCI_MIGRATE = yes
```

➲ **構文3**

「dump」をDCI_MIGRATE設定に加えることで、失敗した操作のレコードを破棄することができます。(スペースは仕切りと見なされます。スペースが埋め込まれたログファイル名は認められません。)

```
DCI_MIGRATE = yes dump
```

➲ **構文4**

DCIMIGRATE_COMMIT_COUNTを設定すると、コミットカウントを100から指定した数まで変更できます。

```
DCIMIGRATE_COMMIT_COUNT = 200
```

用語集

API

Application Programming Interface: APIは、アプリケーションとオペレーティング・システム間のインターフェースです。

BLOB

バイナリ・ラージオブジェクト。データベースに格納される大きいサイズのデータ。表に異なるレコードとして格納されることはありません。BLOBは、普通のレコード同様にデータベースを経由してアクセスすることができません。データベースは、BLOBの名前と位置にのみアクセスすることができます。典型的には、他のアプリケーションがこのデータを読み込むこむために使用します。

バッファ

バッファは、内部のメモリ領域です。入出力操作の間、一時的にデータが格納されます。

クライアント

中央のサーバー・コンピュータに格納されているデータにアクセス、操作することができるコンピュータ。

カラム

同じデータ型からなる複数のレコードで定義されるデータベースの表にあるデータの集まり。

データ・ディクショナリ

拡張ファイル・ディスクリプタとも呼ばれています。データベースのスキーマとCOBOLアプリケーションのファイル・ディスクリプタ間のマップ（リンク）の役割を果たします。

ディレクティブ

移行しているフィールドを、初期設定のDCI設定以外のデータ型にセットするCOBOLコードに配置されたオプションのコメント。

フィールド

おおまかにデータベースのカラムに対応しているCOBOLファイル・ディスクリプタの一部。COBOLレコードに含まれる不連続のデータ・アイテム。

ファイル・ディスクリプタ

ファイル・ディスクリプタは、処理によって操作されるファイルを識別するための整数です。ファイルの読み込み、書き込み、クローズ等の操作は、入力パラメータとして、ファイル・ディスクリプタを使用します。

索引ファイル

全レコードを一意に識別するキーの一覧を含んだファイル。

キー

データベースでレコードを識別するために使用する一意の値（詳細については、*主キー*を参照のこと）。

主キー

主キーは、一意（又はキー）の値のカラムです。表にある個々のレコードを識別するために使用します。

問合せ

DBMasterでは、データの問合せを実行するために使用されるSQL文は、テ特定の情報を取得するために、ユーザーが発する要求。

レコード

COBOLでは、Data Divisionで定義される関連するフィールドの集まりです。DBMasterでは、レコードは行とも呼ばれています。表のカラムの関連するデータ・アイテムの集まりを意味します。

リレーショナル・データベース

リレーショナル・データベースは、データベース・システムです。異なるデータベースにある内部データベースの表がキーや一意索引を用いることによって、もう一方に関連付けることができます。

スキーマ

カラムで定義されるデータベースの表の構造。データ型、サイズ、カラム数、キー、制約、これら全てで表のスキーマを定義します。

サーバー

サーバーは中央コンピュータ。ネットワーク環境設定ファイルを格納し、操作します。同時に、データを格納（データベース）するデータベース管理システムを含み、ネットワークを経由してデータをクライアントに分散させることもできます。

SQL

Structured Query Language: DBMasterと他のODBC互換プログラムがデータへのアクセスと操作に用いる言語。

表

レコードを格納するための、カラムと行で構成されるデータベースの論理的なストレージ単位。

EFDファイル

拡張ファイル・ディスクリプタの頭文字、又はデータ・ディクショナリ。データ・ディクショナリのためのファイル拡張子も形成します。

索引