TIP Data Base Interface II Reference

IP-634

This edition applies to TIP/ix & TIP/dbi II 2.5 and revision levels of TIP/ix 2.5 until otherwise indicated in a new edition. Publications can be requested from the address given below.

Inglenet Business Solutions Inc reserves the right to modify or revise this document without notice. Except where a Software Usage Agreement has been executed, no contractual obligation between Inglenet Business Solutions Inc. and the recipient is either expressed or implied.

It is agreed and understood that the information contained herein is **Proprietary** and **Confidential** and that the recipient shall take all necessary precautions to ensure the confidentiality thereof.

If you have a license agreement for TIP Studio or TIP/ix with Inglenet Business Solutions Inc., you may make copies of this documentation for internal use. Otherwise, you may not copy or transmit this document, in whole or in part, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of Inglenet Business Solutions Inc.

Inglenet Business Solutions Inc.
Website: http://www.inglenet.com
Help Desk: helpdesk@inglenet.com

TIP Studio, TIP/ix, and TIP/30, and are registered trademarks of Inglenet Business Solutions Inc.:

This documentation occasionally makes reference to the products of other corporations. These product names may be trademarks, registered or otherwise, or service marks of these corporations. Where this is the case, they are hereby acknowledged as such by Inglenet Business Solutions Inc.

This document is valid for TIP/ix version 2018/08/04 2.5 R0 - 0360 or later.

Contents

Introduction

TIP/dbi is a sub-system of TIP/ix that enables access data stored in a relational database. It provides a compatible API to *DMS/2200* and *DMS/80*. TIP/dbi is not identical to DMS, but comes very close in functionality with a few limitations. It also enables TIP/ix to access *regular database tables* as well as *indexed data files that are emulated with relational database tables.*

Product Requirements

The TIP/dbi II product requires:

- TIP/ix 2.5 R0 – 0360 or higher
- A supported relational database, Oracle, MySQL or MS SQL Server
  - An Oracle relational database product. See the Release Notes for a list of the supported database products, or contact the Inglenet sales department for current platform support information. Installation and tuning of the RDBMS is the customer's responsibility.
  - For using MySQL, you should configure MySQL to default to use the Innodb database engine.
  - For accessing MS SQL server, ODBC 3.5 compliant drivers are required. ODBC drivers are available from Easysoft (www.easysoft.com)

Overview

For TIP/dbi to access database tables, it needs to know about the structure of the data. It gets this information from either the DMS schema definition, or a manually produced pseudo-schema derived from the database table definitions or built from COBOL copybooks.

You process this schema with the dbischema utility to generate the required IO-modules, and to produce the required target SQL (that is, a .ddl file for Oracle) to generate all tables and indexes.

You pre-process your DMS COBOL application programs with the dbipre utility (much as they were pre-processed on your mainframe). TIP/dbi accepts the same data manipulation language statements and returns the same status codes as the mainframe database did. The DMCA contains the same data field names, but the format is not exactly the same.

To emulate indexed data, or to get access to regular tables, you create a DMS style schema definition (the pseudo-schema). This pseudo-schema must define the records as both LOCATION MODE INDEXED SEQUENTIAL and WITHIN AREA MIRAM. You can use COPY statements to pull in existing record structure definitions.

SQL Update Consideration

Be very careful if you use SQL to *add records* to a TIP/dbi *DMS* database.

In DMS the system maintains the “set” relationship between record types using an elaborate system of “pointers”. Normally, even a database administrator would not manipulate these “pointers” at all, or if absolutely necessary, with extreme caution.

Once your DMS database has been converted to Oracle, TIP/dbi maintains these “pointers” as columns or tables in the database. Using SQL for read only access against a TIP/dbi controlled database is perfectly safe. However, using SQL to **update** a TIP/dbi controlled database could cause severe data corruption if the pointer structure is violated.

To successfully perform this kind of updating, you must understand the way in which TIP/dbi maintains the “pointers”, to provide the required set relationships required for DMS programs. Failure to properly update the “pointers” at the same time as the data could render the database useless.

To update a DMS database that has been converted to TIP/dbi, Inglenet recommends that you use our TQL query product, or a COBOL program that calls TIP/dbi.

This warning does *not* apply to a TIP/dbi database built from a MIRAM or MSAM file. They have no internal set “pointers” to corrupt and therefore, you can use SQL to read or write against the converted database.

Environment variables

Several programs (including the utilities dbipre, dbischema and I/O modules) test for environment variables to control their operation. These environment variables can be used in the users or developers environment. It is also strongly recommended that these variables are added to the TIP/ix configuration file, $TIPROOT/conf/tipix.conf. For details on how to do this, see the *TIP/ix Installation and Operation* manual.

Variable	Description	Used by	Values
TIPDMS	Directory where the compiled database definition is stored. The defaults is: $TIPROOT/tipfiles/DMS	dbischema, I/O module, TIP/ix	
TIPDMSLOG	Amount of log information to generate Example: TIPDMSLOG=akqs,o=/tmp/dir/dbilog,3M This would re-direct the TIP/dbi log to the file called "dbilog" in the"tmp/dir" directory and limit its size to 3 MB. Once it reaches its maximum size of 3 MB, it starts over-writing at the top in a wrap-around fashion. Note: The default behavior of TIP/dbi (i.e. without 'o' option) is to generate the log files in the directory where the program is run from. The log times in TIP/dbi logs are prefixed with A or B. Having A and B in the same log file means the logfile has wrapped around. When you look at a log, if the first line starts with A and you want to find the end of the log, just search for a B at the beginning of the line. In UNIX vi, that is /^B	I/O module	'c' - commands 's' - statistics only 'd' - log details 'a' - all of above 'k' - retain log file '#M' - specify size of log file. (e.g. 1M=1 Megabyte) 'o' - specify output directory and filename for logfile (Note: This setting re-directs all TIP/dbi logs to a single file) 'q' - log SQL
TIPDMSCOMMIT=n	Specify the interval n of calls to TIP/dbi after which an	I/O module	Integers.

Variable	Description	Used by	Values
	automatic COMMIT will be issued. If this is used, a COMMIT is issued without regard to any application grouping of updates into transactions.		
ORACLE_UID	Oracle user id to be used to connect to Oracle. When the TIP/dbi interface is generated by the program schema, the settings of the Oracle variables are stored in the generated routines (as default values). If schema_PWD, schema_UID and schema_SID are set when TIP/dbi actually connects to Oracle, it uses their values. If not, TIP/dbi then checks if ORACLE_PWD, ORACLE_UID, ORACLE_SID are set, and if set, uses their values. If neither schema nor ORACLE variables are set, TIP/dbi sets them to the values that were present when the schema program was run.	Schema I/O module	
ORACLE_PWD	Oracle user password to be used	schema I/O module	
ORACLE_SID	Oracle database system ID	schema I/O module	
schema_SID	Oracle database system ID. Where ‘schema’ is the ‘schema name’ as defined to TIP/dbi dbischema compiler	I/O module	
schema_TRC	Turn on the Oracle trace option	I/O module	
schema_UID	Oracle user id to be used	I/O module	
schema_PWD	Oracle user password to be used.	I/O module	

Variable	Description	Used by	Values
schema_CON	The complete connect string required for the database. If this is present, it is used instead of the _UID, _PWD, _SID values	I/O module	
schema_DSN	Defines the Data Set Name to be used to connect to an ODBC database connection	I/O module	Should match odbc.ini
schema_ROWS	Defines how many rows of data to read as an 'array fetch' for ISAM type records when using ODBC databases like MySQL and MS SQL Server	I/O module	If you use this keep the number more than 2 and less than 5
schema_TOP	Defines a limit on how many rows the database should place into a result set. This if for ODBC daabases like MySQL and MS SQL server	I/O module	
ORACLE_HOME	Oracle home directory If ORACLE_HOME is not set at execution, TIP/dbi sets it to the value that was present when the schema program was run.	I/O module	
OUTDIR	Default directory to generate output files	dbitosch dbipre	
COBCPY	Define search path for COPY books	dbischema	

Schema compiler

The program **dbischema** reads both DMS/80 and DMS/2200 schema and sub-schema source. The result of the compilation procedure is to produce dictionary file called ***schemaname.*sym**. This dictionary file is created in a private directory.

The base directory is **$TIPROOT/tipfiles.DMS/** or the value of the environment variable **TIPDMS**. The base dictionary directory then becomes **$TIPDMS/*schemaname*.dd3/**.

For example, if TIPDMS is "/u/rjn/dms" and we compile a schema called SPSCHM then a directory "/u/rjn/dms/spschm.dd3" is created as the dictionary for this schema. A file called spschm.sym is created which holds the entire database structure and all of the record definitions.

The schema compiler will then emit a schema definition for the target database.

The command line for **dbischema** may have some options specified and the file name to be compiled. The compiler will figure out whether the input file is DMS/80 or DMS/2200 and if it is a schema or sub-schema. The other possible command line options follow.

Option	Description	Default
-T target	target database (oracle8, oracle9, oracle10, sql2005, mysql, odbc)	For version of Oracle installed on system. If using Microsoft SQL Server 2005 (or later) or any other ODBC database you will also need ODBC drivers and likely unixODBC driver manager. See: www.easysoft.com
-w	print more warning messages	No
-o	Compile schema to be compatible with the older version of TIP/dbi	

If the module being compiled is a schema, then under the dictionary base directory the following files may be created where ***schema*** is the database schema name.

File	Description
schema.ddl	SQL schema definition to create all tables
schema.delete	SQL statements to delete all rows from every table
schema.drop	SQL statements to drop all tables
schema.sym	symbol table and database structure
schema.views	SQL views of folded records. See the -f option.
schema.map	Data-mapping rules file
sqlplans.txt	Database cross reference listing
dictionary.exp	Symbolic dictionary for database access. This is the file which is loaded at run-time by Tip/dbi.

If the module being compiled is a sub-schema, then under the dictionary base directory another directory is created with the same name as the sub-schema. For example, if the sub-schema SPSUBS of SPSCHM is compiled then a directory of **/u/rjn/dms/spschm.dd3/spsubs** is created as the sub-schema base directory. This directory then holds several files which are used by the pre-processor program **dbipre**.

File	Contents
subschemaname.sym	modified symbol table indicating which records and sets are included.
areas	table of areas included
copy.ls	pre-processor COPY book for LINKAGE SECTION
copy.ws	pre-processor COPY book for WORKING-STORAGE SECTION
dbdn	DMS/2200 table of database data names
records	table of record names included
sets	table of sets included
tables	constant tables for WORKING-STORAGE

Additional SCHEMA clauses

A few additional schema definition clauses have been added which the TIP/dbi schema compiler accepts.

At the beginning of the schema definition module the following clauses may be optionally specified following the SCHEMA NAME IS clause. If the keyword **NO** precedes any of these clauses then the option is not selected. Options which may or may not be negated are shown below with [NO].

Clause	Description
[NO] ALLOW ANYCHARDATA	To accept any hex value for PIC X fields. Normally DBI validates the data based on the local language setting. (see LANG env var)
[NO] ALLOW ANYKEYDATA	To accept arbitrary values for keys to select on.
[NO] ALLOW CRLF	Tells TIP/dbi to retain in PIC X fields the ASCII print/carriage control codes TAB, BS, CR, LF, FF. Normally these would be translated to a space and an error message logged.
ALLOW NO FOLD	declares that dbischema should never try to combine fields of an OCCURS into a single SQL column. The default is to combine the fields if they are all display data and the total length is less than 20 bytes.
[NO] CLUSTER ALL	Cluster where possible records and set indexes.
[NO] FOLD ALL	Compact many COBOL data items into SQL CHAR columns. This may improve performance, but the resulting SQL database is not very usable outside of TIP/dbi. Do this for all records. Default NO.
[NO] GENERATE CONSTRAINTS	Generate referential integrity constraints. Default is to have constraints.
[NO] IBMCOMP	COMP-4 & BINARY fields are round up to size of 2, 4 & 8 bytes. Default Yes for OS3/DMS80 schema. Default NO for DMS2200 schema.
[NO] LOW-VALUES ARE NULL	any field with all LOW-VALUES will result in the corresponding column being set to NULL. Default is ARE NULL.
[NO] LOW-VALUES REJECT	To cause even low-values to be treated as bad

Clause	Description
	data. Default is to accept.
[NO] NUMERIC IS CHAR	Normally PIC 9 fields are stored in a manner that easily allows numeric manipulation form SQL. If this option is specified then all PIC 9 DISPLAY fields in every record are treated just like PIC X fields. This reduces the storage requirements within SQL by 1 or 2 bytes per field. Default is NO.
[NO] NUMERIC SPACES IS NULL	If a PIC 9 DISPLAY field contains all SPACES, then the SQL column will be marked as NULL. Likewise when reading a table, if the column was marked NULL, then the field in the COBOL record area will be set to all SPACES. Default NO.
[NO] SPACEOUT CRLF	Tells TIP/dbi to change to a SPACE in PIC X fields the ASCII print/carriage control codes TAB, BS, CR, LF, FF.
[NO] TRIM NAMES	To get column names shorter. This will remove common prefixes and suffixes from the COBOL field names as well as all dashes. Default action is TRIM.
[NO] TRIM NAMES-	To get column names shorter but leave '-' in name as an underscore. This will remove common prefixes and suffixes from the COBOL field names as well as all dashes. Default action is NO TRIM NAMES-. I.e. The '-' is changed to underscore.
[NO] VARCHAR ALL	use varchar instead of char to define the columns
BAD-DATA IS "xxxx"	To compile in the bad data status code.
BINARY TO DISPLAY	Treat all COMP4 & BINARY fields as USAGE DISPLAY.
CACHE SELECT n	'n' is the most number of parsed SQL statements which TIP/dbi run-time should cache in memory. Default is 48
CHECK CHAR FOR NUMERIC	This tells 'dbischema' to check for any field declared as CHAR, if the the field and/or all sub-fields are PIC 9 DISPLAY, then translate any SPACEs in the data field to ZEROs.

Clause	Description
COBOL IS MICRO-FOCUS	Indicate that Micro Focus COBOL is being used. This is the default.
CONCATENATE TRANSACTION END	indicates that for programs using more than one schema such as a DMS and an RDMS schema, if one does Commit/Rollback then the other is also automatically Commit/Rollback processed.
DATABASE BLOCK nnnn RECORDS	Where 'nnnn' is 4096, 2048 or 1024. The smaller the number the larger the data file can be. For 8K block size and 2048 the tablespace file could grow to 16GB, For 8K block size and 1024 the file could grow to 32GB. For 8K block size in the Oracle database and 4096 the largest file is 8GB. The default value is now 2048. The old value used was 4096.
DATABASE-KEY CONTAINS AREA	The bits required to hold the area number occupy the top x bits of the DBKEY value. For example if AREA CONTROL IS 127 then 7 bits are required to hold the area#. The remaining bits of the DBKEY hold the unique sequence value assigned to the record.

Clause	Description
DATABASE-KEY INCLUDES AREA	This is to handle the case of applications which FETCH via the DBKEY of a RECORD which is in multiple areas and the application code does not fill in the Area DBDN. On the 2200 this would work, but on Unix, TIP/dbi needs to know which table to read from. Without this option, there are a possible 2**32 unique values that could be used for a database-key. With this option the number is reduced by a factor equal to the most number of areas that a record is defined in. For example, if a record could be in 12 different areas, then the largest unique value gets reduced to 83 million.
DATABASE-KEY INCLUDES RECORD	This is to handle the case of applications which FETCH via the DBKEY of unknown RECORDs and the application code does not fill in the record name. On the 2200 this would work, but on Unix, TIP/dbi needs to know which table to read from. Without this option, there are a possible 2**32 unique values that could be used for a database-key. With this option the number is reduced by a factor equal to the most number of records defined in the database. Eg. If there are 100 records in the schema then the most records that could ever have existed for any one record type is (2**32 / 100) or 32 million. Note that DBKEY values are never reused.
DATABASE-KEY IS n	'n' declares the number of bytes that will be used to store a DBKEY value. The default is 4 bytes (or 32 bits). The value may not be greater than 8 bytes (or 64 bits).
DATABASE-KEY MULTIPLE SEQUENCE	Declares that for each area that a record belongs to a different SEQUENCE value should be used to assign the next DBKEY value. If USE AREA CODE was declared then all areas of a record are stored in the same table and by default a single SEQUENCE is used to assign values for all area/record combinations. This clause causes a different SEQUENCE to be used for each area/record allowing for less chance of running out of unique values.

Clause	Description
DATABASE-KEY WITH RECORD	This is another option to handle the case of applications which FETCH via the DBKEY of unknown RECORDs and the application code does not fill in the record name. For data fields defined as USAGE DATABASE-KEY there will also be created a second field called DBK-dbkname-R following dbkname. The DBK-dbkname-R field will hold the RECORD name associated with the DBKEY value. A FETCH dbkname will then pick up the record name to be fetched. This allows for any one record type to have had created 2**31 records without the need to reload the database. (Note that DBKEY values are unique per table and never reused.) However this option causes a PIC S9(9) COMP plus a PIC X(30) field to be created each USAGE IS DATABASE-KEY.
DATE IS “format”	Define the default date format. For example “YYMMDD”. See page 26 for details.
DATE ZERO IS NULL	Indicates that a COBOL field defined as a date with a value of ZERO will cause the SQL column to be set to NULL. A NULL date column will also return a value of ZERO to the COBOL field. Default: without this clause a NULL date column would be returned to the COBOL field as LOW-VALUES.
FIND IN VIA AREA	Optimize the table structures as the owner and member of set are always in the same area.
FOLD INDEXES	Fold composite index fields into a single column.
GUESS DATE “format”	Define the default date format. For example “YYMMDD”. This also tells the schema compiler to scan the record structure looking for fields that may be date fields. It looks for the word DATE in the name, etc.
GUESS DATE “format” patterns	Define the default data format and also give more hints about which fields may contain dates. For example: GUESS DATE "YYYYMMDD" "-DT-", "-DATE-".

Clause	Description
INDEXSPACE “name”	the default table space of the relational database for storing indexes is “name”
KEEP DEFAULT	Compile schema to be compatible with the older version of TIP/dbi
MAXIMUM CHAR LENGTH n	The default maximum length of a single CHAR field for Oracle is 2000 and TIP/dbi does not exceed that. However, Oracle 10 & 11 can handle up to 8000. If you want to have TIP/dbi use a larger maximum then define it with this clause.
MAXIMUM DELAY n	n is maximum number of seconds which the thread manager will wait for an idle server process before starting a new process
MAXIMUM IDLE n	n is the maximum number of seconds a server process is allowed to be idle. After this the process is shut down.
MAXIMUM TABLE LENGTH n	Define the maximum amount of data that TIP/dbi should assume Oracle can handle in a single row of a table. The default value is 32000.
MAXIMUM THREADS n	n is the maximum number of database server threads to be executing under TIP
MINIMUM THREADS n	n is the minimum number of database server threads to be executing under TIP
MAXIMUM SEQUENCE n	N is the maximum value that any DBKEY Sequence may be assigned. When the value is reached the Sequence is reset back to 1 and for additional STOREs of new records if the DBKEY is found to be a duplicate then the sequence is incremented and the STORE is attempted again.
NOT DATE patterns	Define a set of partial field names which are not to be used as dates. For example: NOT DATE "-DT-DE-FACT", "-DT-PRELEV-INV", "-DT-LANCE".

Clause	Description
OWNER IS “user”	Declare the database owner user-id. This is user-id is used for security reasons when accessing the database on behalf of the end-user running the transaction program or batch program. The name given will prefix all table names.
PASSWORD “name”	the password for the user id to connect to the database
PIC 1 TO BINARY	Treat all PIC 1 fields as BINARY or appropriate size. This is the default.
PIC 1 TO DISPLAY	Treat all PIC 1 fields as usage DISPLAY.
REMOVE RECORDS (name, name2, …)	declares the named records to be 'obsolete'. They are left in the schema but any reference to them results in a warning message at compile time and an error status at run-time. The $TIPROOT/log/HISTORY will also get the offending program recorded.
REMOVE FILLER	If the last field of a record is FILLER then do not process as part of SQL database definition. This will allow for easy extension of the database later. (This is the default action)
RETAIN FILLER	If the last field of a record is FILLER then process it as a normal data field. Only do this if this field contains valid data for some reason. A better choice would be to use a field name other than FILLER.
RETRY GETUP 0.25	Retry the record lock at 0.25 second intervals (default 0.1)
SEPARATE RECORDS	For records with too many columns and/or too much data, split the record into multiple SQL tables.
TARGETDBMS IS name	Where name could be one of oracle9, oracle10, oracle11, sql2005, mysql or odbc. Used to define which is the target relational database. MSSQL-DATA indicates database migration only and TIP/dbi is not used for runtime.
TABLESPACE “name”	the default table space of the relational database is “name”
TRANSACTION READ	Tells Oracle to establish transaction-level read consistency. If the transaction requires row

Clause	Description
COMMITTED	locks held by another transaction, then it waits until the row locks are released.
TRANSACTION SERIALIZABLE	Tells Oracle to establish transaction-level read consistency. If a serializable transaction attempts to update any resource that may have been updated in a transaction uncommitted at the start of the serializable transaction, then the transaction fails.
TRANSACTION READ WRITE	Tells Oracle to establishes statement-level read consistency.
	If the TRANSACTION clause is defined then it is used for both batch and online. If the TRANSACTION clause is not defined the default for online transaction programs is like TRANSACTION READ WRITE. The default for batch is like TRANSACTION READ COMMITTED.
USER-ID “name”	the Database user id to connect to the database with
WAIT GETUP 1.5	To wait a maximum of 1.5 seconds (default 1.5) (A value of 0 indicates to wait forever using Oracle SELECT FOR UPDATE.)

In each **area** definition the following clauses may be optionally specified:

Clause	Description
INDEXSPACE “name”	the table space of the relational database for storing indexes of all records in this area is “name”
TABLESPACE “name”	the table space of the relational database for all records in this area is “name”

In each **record** definition the following clauses may be optionally specified:

Clause	Description
DATE (names...)	The fields are to be defined as date instead of char. The field must then be in the default date format. See page 26 for

Clause	Description
	details.
DATE “format’ (names...)	The fields are to be defined as date instead of char. The field must then be in the defined date format. See page 26 for details.
DATABASE-KEY (names…)	Each of the field names listed is actually used to hold a DATABASE-KEY of some other record in the database.
ENCRYPT (names…)	The field names listed are encrypted in the database.
FOR UPDATE	Indicates that this record is very often updated when read. If the area is opened for UPDATE then the record is read from the SQL database FOR UPDATE. The default behaviour is that the record is only read for update from the database when some DMS modification verb is used.
FOLD	Compact many COBOL data items into SQL CHAR columns. This may improve performance, but the resulting SQL database is not very usable outside of TIP/dbi.
NO FOLD (***xxxx***, …)	Normally TIP/dbi II will fold an array with PIC X members where the total length is less thatn 20 bytes into just one SQL column. This is being done to improve performance by reducing the number of columns. 'dbischema' will emit a message the following to indicate it is doing the fold: ***Note: Treating xxxx size nn as one column*** If you do not want this to happen add this clause to the record definition.
INDEXSPACE “name”	the table space of the relational database for storing indexes of all records in this area is “name”
NUMERIC IS CHAR	Normally PIC 9 fields are stored in a manner that easily allows numeric manipulation form SQL. If this option is specified then all PIC 9 DISPLAY fields in this record are treated just like PIC X fields. This would reduce the storage

Clause	Description
	requirements within SQL by 1 or 2 bytes per field.
RAW name RAW (name1,name2,…)	"name" is the name of a data field which does not always hold the same type of data. (For example, it may sometimes be characters and sometime binary data). RAW causes the data to be always converted to hexadecimal and stored in the database. This should only be used when absolutely necessary. May not to be supported.
REDEFINES name1 IF condition USE name2 [HIDDEN]	"name1" is the name of a data field which is redefined and each of the redefines is to be stored in separate columns in the same table. For each REDEFINES, there should be an IF/USE definition. IF the condition is true, then the name2 is the one that is valid. If all IF cases are false, then the base field definition is used. HIDDEN indicates that the fields REDEFINEing are not to be inserted by the DML pre-processor.
STORAGE "storage"	string holds target database STORAGE directives.
TABLESPACE "name"	the table space of the relational database for all records in this area is "name"
VARCHAR name VARCHAR (name1,name2,...)	The fields are to be defined as varchar2 instead of char
WITH DATABASE-KEY	For MIRAM/Indexed style records that you want to also have a unique database key column. This would be used for TIPFCS function FCS-GETRN.

Database key

Every record must have a database key. In DMS the database key is sufficient to uniquely identify every record in the database. In the TIP/dbi DMS interface the database key is unique ***only*** for a given record type and the same value may be used for different record types. This means the application must know which record type is wanted. This is may not be a very important restriction as the applications generally always know which record type is wanted when FETCHing by database key.

The next available unique number may be generated in one of a few ways depending on the target database. It is generated using a built-in function of Oracle called a SEQUENCE.

For LOCATION MODE DIRECT records a new database key is returned for each STORE of a new record. The DMS application must not rely on specific database key values being returned exactly as DMS/2200 does. The DIRECT-DBK field is not used to assign new database keys. As long as the application picks up the database key returned in the DMCA after a STORE it should work correctly.

If the DMS application programs use **FETCH *dbkname*** with no record name this will not work as coded. One option is to change the application code so that it supplies the record name (eg. FETCH myrec RECORD dbkname). Other options would be to use the schema clause DATABASE-KEY INCLUDES RECORD which will work if the tables do not contain a lot of records But if you have record types (i.e. Tables that hold 10s of millions of record you could quickly run out of unique database-key values.)

Another option is to use the schema clause DATABASE-KEY WITH RECORD which will create and extra field for each USAGE IS DATABASE-KEY to hold the record name. Then on each MOVE dbkfld1 TO dbkfld1 the database key as well as the associated record name will be copied. This requires the all FETCH dbkname statements MUST have the dbkname field defined as USAGE IS DATABASE-KEY. If dbkname is PIC S9(9) COMP you will get a pre-processing compile error.

Area definition

An AREA can be represented by a TABLESPACE of the relational database. A TABLESPACE allows tables to be grouped together, plus a TABLESPACE may be assigned to its own storage file. The **dbischema** program will optionally emit the SQL schema with the TABLESPACE information. The table-space information is defined by adding a **TABLESPACE “tabname”** clause to an AREA definition. In addition the indexes within an AREA may be declared to be in a separate table space with the **INDEXSPACE “indname”** clause, where **indname** is the table-space to hold the indexes.

DMS applications will often do AREA scans. This will be implemented by reading each record type defined in the area consecutively. All of record type A would be returned and then record type B and so on.

Record definition

The translation of the records is fairly straightforward. Each record becomes a table in the SQL database. All record and field names are translated to lower case and dashes become underscores. If TRIM NAMES is set and 80% of the field names have the same common prefix or suffix then it is removed. If the name is longer than the maximum allowed, it will be truncated and a sequence number is appended to make the name unique within the record.

Every record will have an extra field added which is called **row_*recordname*** that is an INTEGER PRIMARY KEY. Each record will then be assigned a unique number when it is added to the database. This number is then used to locate the record when needed. This unique value is used as the database key.

The 2200 special data types of PIC 1 and Field-data are translated as follows:

All Field data (DISPLAY-1) gets translated into just PIC X and ASCII. When unloading the database, then COBOL/DMS programs generated by dbiunload will convert Field-Data into ASCII during the unload operation.

PIC 1 fields can vary from 1 bit to 36 bits long on the 2200. PIC 1(36) takes up the a word, as does PIC X(4). The following table shows how many PIC 1 bits get mapped to how many PIC 9

From	Up to	# of 9s
1	3	PIC 9
4	6	PIC 9(2)
7	9	PIC 9(3)
10	12	PIC 9(4)
13	16	PIC 9(5)
17	19	PIC 9(6)
20	23	PIC 9(7)
24	26	PIC 9(8)
27	29	PIC 9(9)

From	Up to	# of 9s
30	33	PIC 9(10)
34	36	PIC 9(11)

The COBOL data types are mapped to SQL data types as follows:

COBOL Data Type	SQL Data Type
PIC X(n)	CHAR (n) or VARCHAR(n)
PIC 9(n)	1<= n < 5 then SHORTINT 5 <= n < 10 then INTEGER 10 <= n <= 18 then DECIMAL (n)
PIC 9(n)V9(m)	DECIMAL (n+m,m)
COMP-1	FLOAT
COMP-2	FLOAT (double)

Since SQL does not support **REDEFINES**, these fields are skipped over. Only first definition of a field in the record is emitted as an SQL column unless otherwise directed by the 'REDEFINES name IF condition USE name' clause. All of the record definition including REDEFINES is retained for subschema processing and DML pre-processing. If a field does **OCCUR**, then it is emitted as field01, field02, field03, etc.

If REDEFINES is only be used for convenience, like splitting a PIC X(n) field into two pieces and the redefined data is completely compatible, then the schema compiler just ignores the REDEFINES.

Use of RAW

If REDEFINES is used to redefine different types of data such as COMP-3 redefining PIC X there may be a problem. The SQL database must have only one data type for a given position (column) within the record (row). Even PIC 9 or PIC S9 DISPLAY should not REDEFINE PIC X or any COMP type of fields. If this situation is not avoidable then you should declare a PIC X for the region of the record that is redefined several ways and then declare this field as RAW. (For SQL Server and MySQL that field would be defined as type BINARY.)

```
RECORD MYREC
LOCATION MODE xxxxxx
```

```
RAW PART1
WITHIN MYAREA.
05   THE-RECORD.
     10   FIELD1                PIC X(5).
     10   PART1                 PIC X(10).
     10   FILLER REDEFINES PART1.
          15   FIELD2       PIC S9(7)V99 COMP-3.
          15   FIELD3       PIC S9(7)V99 COMP-3.
```

The above record would result in the follow SQL structure.

```
CREATE TABLE myrec (
         Field1            CHAR(5),
         Part1             RAW(10),
         Row_myrec         INTEGER PRIMARY KEY );
```

REDEFINES Clause

An alternative to declaring a field as RAW is to define how it is used and split it out as separate columns. For example:

```
RECORD MYREC
LOCATION MODE xxxxxx
REDEFINES PART1
IF FIELD1 = "NUM" USE FILLER-1
WITHIN MYAREA.
05   THE-RECORD.
     10   FIELD1                PIC X(5).
     10   PART1                 PIC X(10).
     10   FILLER-1 REDEFINES PART1.
          15   FIELD2       PIC S9(7)V99 COMP-3.
          15   FIELD3       PIC S9(7)V99 COMP-3.
```

The above record would result in the follow SQL structure.

```
CREATE TABLE myrec (
         Field1                CHAR(5),
         Part1                 CHAR(10),
         Field2                DECIMAL (9,2),
         Field3                DECIMAL (9,2),
         Row_myrec         INTEGER PRIMARY KEY);
```

For a given row, the columns that are valid would be set to hold the data and the columns that are not valid would be set NULL.

If a field is declared as the CALC key or INDEX field (DUPLICATES NOT ALLOWED), then it is emitted as a UNIQUE. Secondary INDEX fields are emitted as UNIQUE if DUPLICATES ARE NOT ALLOWED or as a CREATE INDEX when DUPLICATES ARE ALLOWED (FIRST or LAST) or the order is DESCENDING.

When DUPLICATES ARE FIRST or LAST then a column called **pos_*recordname*** becomes part of the index to sequence the duplicates. This column (**pos_*recordname***) holds the same value as **row_*recordname*** when DUPLICATES LAST and it holds negative **row_*recordname*** when DUPLICATES FIRST.

For every set, which a record is a member of, it will have an extra field added which is called **own_*setname***. This **own_*setname*** will hold the same value as **row_*recordname*** of the owner record of that set and is also the first part of the index created to implement the set. If the set is not SORTED then there will also be a **pos_*setname*** added to the record, which will represent the position of the record within that set.

Naming indexes for a Record

TIP/dbi schema accepts a specific index name to be used for the Oracle index. This is defined as a string enclosed in quotes as part of the INDEX definition. For example:

```
LOCATION MODE INDEX SEQUENTIAL
    USING CM-NUMBER
        AS KEY 1 "K1_TXPFILE"
            DUPLICATES NOT ALLOWED
    USING    CM-TELEPHONE
             CM-COMPANY
        AS KEY 2 "TXPFILEIDX3"
            DUPLICATES ALLOWED
```

Date format

You may define how your application stores DATE information. Inside the database, TIP/dbi will always expect dates to be in a full YYYYMMDD format and date/time to be in YYYYMMDDHHMISS format. Date fields could be defined like the following:

```
DATE (CM-DATE, CM-PK-DATE)
DATE "MMDDYYYY" (CM-DATE, CM-PK-DATE)
DATE "YYYYMMDD" CM-SHORT-DATE
DATE "YY%60MMDD" CM-PIVOT-DATE
DATE "YYY+1800MMDD" CM-BASE-DATE
```

The Y is a place holder for a YEAR, MM for month, DD for day, HH hour, MI minutes, SS seconds.

If the Ys are followed by ‘%’ then the digits after the ‘%’ defines a pivot year used to map the YY value into a 4 digit year. In the above example if the YY value is below 60 then it is 19YY else 20YY

If the Ys are followed by ‘+’ then the digits after the ‘+’ are added to the Y value. In above example, the year is 1800 + YYY value.

There is a limit of 30 different DATE formats per schema.

If the day is defined like DDD (3 Ds) then it is taken to be the day of the year. For example:

```
DATE "YYYYDDD" CM-DAY-OF-YEAR
```

If you define data fields and omit the date format then the fields default to the date format defined in the schema header section. If there is no default date format defined then it defaults to **YYYYMMDDHHMISS**.

If the COBOL date field is all ZERO then the SQL date column will be set to the minimum allowable date and if the date column has the minimum allowable date the COBOL field will be given back a value of ZERO. However, if DATE ZERO IS NULL was defined, then a ZERO date field causes the column to be set to NULL.

For MicroSoft SQL Server the minimum date is 1753-01-01.

For MySQL the minimum date is 1000-01-01.

For Oracle the minimum date used is 0001-01-01.

If the COBOL date field is all 9s then the SQL date column will be set to the maximum allowable date value which is normally 9999-12-31.

UCSTDATE format

The OS2200 system has a function UCSTDATE$ which returns the date/time as a 36 bit value as follows:

```
05  TD.
  10 MM PIC 1(6).
  10 DD PIC 1(6).
  10 YY PIC 1(6).
  10 SEC PIC 1(18).

DATE "UCSTDATE" (CM-DATE, CM-PK-DATE)
```

MM is month, DD is day of month, YY is years since 1964 and SEC is seconds past midnight.

With TIP/ix and TIP/dbi the time will be returned as 6 bytes in the following format:

```
05  TD.
  10 MM PIC 99 COMP.
  10 DD PIC 99 COMP.
  10 YY PIC 99 COMP.
  10 SEC PIC 9(5) COMP.
```

Defining DATE “ERTDATE” tells TIP/dbi that the field(s) are a date/time in this format.

Clustering records

TIP/dbi schema compiler accepts the directive CLUSTER in a record definition that owns a SET. When declared, the set owner and member(s) that are LOCATION MODE VIA SET and are in the same area will be defined to be stored in an SQL CLUSTER.

Oracle has a CLUSTER concept, which can be used to group related information together.

Clusters may be used to group tables together on the same database pages much like LOCATION MODE VIA.

Clusters may also be used to store rows into pages based on hashing of selected columns. This is much like LOCATION MODE CALC.

Multiple Area records

DMS/2200 allows records to be located in more than one AREA of the database. The AREA to be used is controlled when the application program places the correct area name into a **'database data name'** as declared in the schema definition. These database data names are copied into the program during the pre-processing procedure.

The **schema** program will generate a target schema such that new tables are created for each area that the original multi-area record was declared to exist in. The table name becomes **recordname$*nnn*** where ***nnn*** is the AREA ID.

Set definition

DMS sets that only have a single member record are implemented by adding an INDEX to the member record table.

- For SORTED sets the index consists of own_setname and the field(s) on which the set is sorted.
- For non-sorted sets the index consists of own_setname and pos_setname.

The **pos_*setname*** field is a number, which is manipulated to maintain the member record sequence within the set. This is a double floating point number.

ORDER FIRST	**Pos_setname decreases for each new member**
ORDER LAST	Pos_setname increases for each new member
ORDER NEXT	Pos_setname is a computed value between adjacent members
ORDER PRIOR	Pos_setname is a computed value between adjacent members
ORDER SORTED	The sort field is used as part of the index

The owner record of a set can be located by taking the **own_*setname*** value to locate the owner record in its table. This **own_*setname*** can also be used to locate entries in the set index of the member table. The list if all members of a set can then be scanned by reading through those entries in the table that match **own_*setname*** ordered by **pos_*setname*** (or sort field for sorted sets).

If the DMS set has multiple types of member records then a table is created in the SQL database to implement this type of set. The table will look like the following:

```
CREATE TABLE setname (
     own_setname                INTEGER,
     pos_setname                FLOAT,
     member_id                  INTEGER,
     member_row                 INTEGER,
     PRIMARY KEY (own_setname, pos_setname))
ALTER TABLE setname ADD (
     CONSTRAINT
     FOREIGN KEY (own_setname)
     REFERENCES   ownerrec (row_ownerrec));
```

After the key portion of the set record will follow **member_id** that is a number indicating which record type is the member and **member_row** corresponds to the **row_*recordname*** of the member record. If it were a sorted set then the sort field would be used in the table in place of **pos_*setname***.

Subschema compiler

A separate utility called **dbisubschema** is used to process DMS subschema definitions. The sub-schema is primarily used by the DMS pre-processor to limit the view of the database. In the case of DMS/2200 all field renames and redefinitions are done in the subschema.

The usage of this utility is as follows:

```
TIP/ix ver 2007/12/21 2.5 R0 - 0124  © 1991-2007 Inglenet Business
Solutions
TIP/dbi Sub-Schema compiler;    Version 1.43 2007/10/15
        © 1991-2007 Inglenet Business Solutions

 dbisubschema [-wOR] subschemafile

Where the options are:
  -w       Report more warnings
  -O       Check for DMS/1100 specifics
  -R       Skip exhaustive check for mixed data types
```

DML pre-processor

The program **dbipre** has been developed to provide many COBOL pre-processing functions. **dbipre** recognizes and processes the DMS **Data Manipulation Language** statements. You should not retain the intermediate output file created by **dbipre** after the COBOL compiler use, since it is created based on the current contents of the matching dictionary files for the DMS database produced by **schema**.

The base dictionary directory is **$TIPROOT/DMS/** or the value of the environment variable **TIPDMS**. The base dictionary directory then becomes **$TIPDMS/*schemaname*.dd3/**.

The command line for **dbipre** may have some options specified and the file name to be processed. The other possible command line options follow.

Option	Description	Default
-d	do create an output file	not created
-v	print more warning messages	No
-fSQL	Pre-process all EXEC SQL for use with TIP/dbi	No

```
TIP/dbi Cobol/DML pre-processor; Version 1.305 2014/08/17
        © 1991-2014 Inglenet Business Solutions
  dbipre [-options] -i filename
 Where the options are:
   -v      Verbose mode
   -i      Name the input file name
   -o      Name the output file name
   -D      Pre-process as a DPS/2200 transaction program
   -O      Pre-process as a DMS/2200 batch program
   -T      Pre-process as a TIP/ix transaction program
   -I      Pre-process as an IMS transaction program
   -d      Do DMS pre-processing
   -P lvl  Parse 'program' & display SCRATCH area to level 'lvl'
   -B lvl  Parse 'filename' as COPY Book & print symbols to level 'lvl'
   -Q lvl  Parse 'filename' as COPY Book & print symbols to level 'lvl'
                 Assume NO IBM COMP format data with -Q
   -s      Module is a subroutine; Default is mainline program
   -e      Process DMS OPEN ALL USAGE verbs for each area in schema
   -E book Copy 'book' for DMS-STATUS SECTION. (default: TC-DMSST)
   -p      Force preprocessing to complete;  Used for COPY book processing
   -fSQL         Process EXEC SQL statements for execution with TIP/dbi II
   -fos2200      Parse Data Division assuming OS/2200 COMP format
   -fibmcomp     Parse Data Division assuming IBM COMP format
   -fnoibmcomp   Parse Data Division assuming NO IBM COMP format
```

The **dbipre** program may read a module called **myprog.dml** and create a module called **myprog.bat** (for batch programs) or **myprog.cbl** (for transaction programs). A makefile can be used to look for the .dml extension and then pre-process the code, compile the resulting COBOL code and then remove the COBOL source.

The DML statements are translated into CALL statements using parameters very similar to those used by DMS/2200. Refer to the appendices of the DMS/2200 programmer's reference manual for details. The parameter list is always ended with a parameter, which points to a word of all HIGH-VALUES.

The Data Management Control Area (DMCA) is intended to be the union of fields used in the DMS/80 DMCA and the DMS/2200 DMCA. Some of the fields are larger (Area PIC X(18), Record PIC X(30), Set PIC X(30)). The DMCA used by the TIP/dbi DMS emulation interface is similar but not identical and application code should not rely on it being identical. In most cases this will be no problem.

The DMCA has a new field (**DML-SEQUENCE**) that holds the source line number of most recent DML statement executed. This is very helpful in debugging applications.

The DMCA has a extra field (**DBI-SQL-STATUS PIC S9(8)**) which holds the exact Oracle status value after any Oracle error has occurred. This is very helpful in debugging applications.

Table of DMS Verbs and matching numeric value

Verb	Description	Verb	Description
2	CLOSE ALL AREAS	3	DELETE record ONLY
4	DELETE record ALL	6	FIND record USING dbkey
7	FIND CURRENT record	8	FIND CURRENT OF name SET
9	FIND CURRENT OF name AREA	10	FIND NEXT record OF name SET
11	FIND NEXT record OF name AREA	12	FIND PRIOR record OF name SET
13	FIND PRIOR record OF name AREA	14	FIND NEXT RECORD OF name SET
15	FIND NEXT RECORD OF name AREA	16	FIND PRIOR RECORD OF name SET
17	FIND PRIOR RECORD OF name AREA	18	FIND FIRST record OF name SET
19	FIND FIRST record OF name AREA	20	FIND FIRST RECORD OF name SET
21	FIND FIRST RECORD OF name AREA	22	FIND LAST record OF name SET

Verb	Description	Verb	Description
23	FIND LAST record OF name AREA	24	FIND LAST RECORD OF name SET
25	FIND LAST RECORD OF name AREA	26	FIND record BOOLEAN VIA INDEX
27	FIND position record VIA INDEX	28	FIND record VIA set USING field(s)
29	Pass data field for FIND-28 search	30	FIND CURRENT OF RUN-UNIT
31	FIND OWNER OF SET	32	FIND record
34	GET record	35	MODIFY record
36	OPEN UPDATE	37	OPEN RETRIEVAL
38	OPEN ALL PROTECTED UPDATE	39	OPEN ALL PROTECTED RETRIEVAL
40	OPEN ALL EXCLUSIVE RETRIEVAL	41	OPEN ALL EXCLUSIVE UPDATE
42	STORE record	44	INSERT record INTO set
46	REMOVE record FROM set	48	BIND record
50	FIND NEXT DUPLICATE record	51	FIND record VIA CURRENT set USING ident
52	DELETE record	53	DELETE record SELECTIVE
54	MOVE CURRENCY STATUS OF RUN-UNIT	55	MOVE CURRENCY STATUS OF name RECORD
56	MOVE CURRENCY STATUS OF name AREA	57	MOVE type CURRENCY STATUS OF name SET
60	IF MEMBER	61	IF OWNER
62	IF NOT MEMBER	63	IF NOT OWNER
64	IF SET EMPTY	65	IF SET NOT EMPTY
70		71	OPEN area RETRIEVAL
72	OPEN area UPDATE	73	OPEN area EXCLUSIVE RETRIEVAL
74	OPEN area EXCLUSIVE UPDATE	75	OPEN area PROTECTED RETRIEVAL

Verb	Description	Verb	Description
76	OPEN area PROTECTED UPDATE	78	OPEN area INITIAL LOAD
81	BIND SubSchema	82	UNBIND SubSchema
90	DEPART	91	KEEP [EXCLUSIVE]
92	FREE WITH CHECKPOINT	93	DEPART WITH ROLLBACK
95	FREE CURRENT OF RUN-UNIT	96	FREE ALL
97	ROLLBACK	86	FIND dbk
99	End of OPEN AREAS		

Application interface modules

The **dbipre** program will process the DML statements creating CALLs to an interface module. Various interface modules may be used depending on the type of program and the type of the target database.

Batch interface for DMS

Batch programs which use DMS/80 or DMS/2200 must also be able to use this software. Batch programs must be linked with the database emulation code and database I/O modules.

The link edit directives which can be used are

```
-L$TIPROOT/lib -ldbixio -ldbirun -ldbi2
```

Batch interface for Indexed Files

Batch programs which use indexed files must also be able to use this software. Batch programs must be linked with the database emulation code and database I/O modules.

When ‘make’ is used from within the schemaname.dd directory, an archive called libschemaname.a is created. This archive must be included as well as $TIPROOT/lib/libbat.a and $TIPROOT/lib/libdbi.a during the compilation of the batch COBOL program.

The link edit directives which can be used are

```
-L$TIPROOT/lib ldbixio -ldbirun -ldbi2
```

TIPIXDMS - transaction program interface

The module **tipixdms** would be called by transaction programs. The same module can be used for all target databases since the TIP database interface process should always provide the same interface and communication method and it must take care of the differences between the target databases.

On IMPART, a request for the specific schema I/O module is sent to the TIP thread manager. Record BIND requests are tabled up for the application process. The I/O module will BIND all of its own record work areas.

Binding establishes the communication needed between the run unit and the RDBMS. For example, it provides the linkage between the RDBMS and the subschema record descriptions in your program.

All other DML verbs are sent as messages to the selected server module. The **status X** command can be used to display active database interface server processes.

On DEPART, the request would be sent to the I/O server. The server would defer doing an actual DEPART until it received a commit/rollback indication from the TIP Commit Manager.

Converted TIP/1100 programs need to be linked with the archive for 2200 programs, $TIPROOT/lib/lib2200.a

The link edit directives which can be used are

```
-L$TIPROOT/lib -l2200
```

TIPFCS interface

The database I/O module must also be able to process TIPFCS style of requests. Each record in the database may be accessible as a simulated indexed data file (ISAM) through the normal TIP CALL TIPFCS and IMS CALL GET interfaces.

When multiple pseudo files (database tables) are being accessed by the same transaction and they are also in the same database, then care is taken to use the same server database process.

Defining a MIRAM Schema

To define a database of relational tables which are to be processed like **MIRAM** (or ISAM) files through the TIPFCS interface you must create a DMS Schema definition (the pseudo-schema).

In the pseudo-schema, define the record as LOCATION MODE INDEX SEQUENTIAL and define all of the indexes. You must also define an AREA NAME MIRAM and declare all records as WITHIN MIRAM. The area name **MIRAM** is treated as a special area.

For a sample schema, see *Migrating an INDEXED File Application* later in this book.

Defining a Table to TIPFCS

To define a file which is really a table of a relational database, you must first construct the MIRAM Schema and build it. Then, using SMFILE:

- define File type as RDBMS
- define Label/Path with the record name defined in the MIRAM Schema

define **FCS Server** with the name of the database I/O module ({MIRAM Schema Name}io)

Generate a MIRAM Schema from Oracle

The **dbitosch.pc** utility is supplied in source code format. You must pre-process it with Oracle **proc** and compile it. A makefile called **Make.pc** is provided for this purpose.

```
make -f Make.pc dbitosch
```

Once compiled, **dbitosch** can be used to extract relational table structures from the Oracle data dictionary to create a MIRAM Schema. You may have to edit the result to add information about TABLE SPACE, user id, passwords etc. You would then process the MIRAM Schema with the TIP schema program and build the I/O module. (Do not use the schema.ddl module since Oracle already has the tables defined.

Syntax:

```
dbitosch options table1 [.. tablen]
```

Where:

-c Generate COBOL copybooks for each table

-C directory
Generate the COBOL copybooks into this directory

-d Use the date format defined by Oracle's NLS_DATE_FORMAT parameter (instead of the default value of "YYYYMMDDHH24MISS") when generating the DATE IS format directive.

-S name
Specify the Schema name

-U user id
Oracle User Id to connect to Oracle with

-P pass
Oracle password to connect to Oracle with

Following the options is a list of table names to be built into the MIRAM Schema definition. The following example constructs a schema named FCSRDMS and generates COBOL copybooks for the Oracle tables ‘tspfile inven orders parts’:

```
dbitosch -S FCSRDMS -U system -P manager\
-C /source/books tspfile inven orders parts
```

Note: The slash at the end of line one is the UNIX line continuation character.

TIPFCS Relative Record Number and Oracle

The TIPFCS interface to relational database will return a relative record number for all I/O requests. This relative record number is computed from the ROWID. With Oracle the ROWID is an 8 byte field while the TIPFCS relative record number is only 4 bytes. This leads to some compromises and restrictions.

An entire Oracle table must be stored in a single UNIX/Window file. Each block of the oracle database must not hold more than 4095 rows (or records) in a single block and there can be no more than 1,048,575 blocks in the file. With an Oracle block size of 4096 the maximum amount of data in a table would be 4 gigabytes, block size of 8192 yields a maximum of 8 gigabytes and 2048 yields 2 gigabytes of data.

If the above limits are exceeded then you cannot use the TIPFCS relative record number to access retrieved records. You can use all other TIPFCS functions. Refer to more detailed discussion on this later in the section titled TIPFCS relative record number & Oracle.

Database Password Encryption – dbipwd

The utility `dbipwd` may be used to store the user/password information into a small text file in an encrypted format. The data is only decidable by the TIP/dbi runtime code. The command usage is:

```
TIP/dbi Password Encoder;  Version 1.13 2010/04/27
       © 1991-2010 Inglenet Business Solutions

Utility for encoding Database password information for TIP/dbi
Usage: dbipwd -s schema -d dsn -u user -p password -c connect-string

Where:
        -s schema     defines the TIP/dbi schema name
        -u user       defines the Userid for connecting to the database
        -p password   defines the password for connecting to the database

        -S sid        defines the ORACLE_SID value to use
                              (Optionally used for Oracle)
        -d dsn        defines the ODBC DataSetName defined in /etc/odbc.ini
                              (Used for MySQL, DB2 & MS SQL)
        -c string     Could be used as an alternative to user/password to
                      define a full connection string for the database
        -w writepwd   Once defined this password must be for updates
                      This is a 'write password' for the password file
        -g group      The password file will have its Unix group set to that
                      Given. This user running dbipwd must also be a member
                      of this Unix group
```

A text file called *schema*`.pwd` is created in $TIPROOT/conf.

Former RDMS 2200 program may be converted to using procob and a CALL 'ORARDMPWD' will be inserted to look for the schema.pwd file of the DEFAULT SCHEMA name and then set the user & password from the information in the password file.

The most common example of use of this might be as follows:

```
dbipwd –s payschema –u payusr –p paypwd
```

Using the Database Interface

There are a number of steps to go through when using the database interface. The steps vary slightly, depending on whether a true DMS schema is used, or a pseudo-schema for C-ISAM type access.

In the following sample the steps described are for an Oracle database, and the compiler is Micro Focus. (If you use a different RDBMS or compiler, a few changes may be necessary.)

Description	Syntax Example
Process schema and sub-schemas	dbischema -M myschema.sch dbischema -M mysubschema.sch
Make sure your RDB is running	su - oracle dbstart
Create the Oracle tables	cd $TIPDMS/myschema.dd sqlplus system/manager <*.ddl
Pre-process and make your data loading program	dbipre -d myload.dml make myload
Load the data	myload | tee myload.log
Pre-process and make your on-line and batch programs	

Performance Considerations

This section only provides some general guidelines for performance based upon some of our experiences. Proper configuration and maintenance of the Relational Database Management System used is ultimately a customer responsibility. For more information refer to your *Oracle Server Administrator's Guide* and the *Oracle Server Tuning Guide* for information on Oracle configuration and tuning.

A number of parameters and data base configuration options can be used to considerably improve performance when using TIP/dbi with a relational database. These are discussed individually below.

It is assumed that sufficient processing power, ample system memory and proper placement of database files across disks and controllers are already in place.

Choosing the best type of disk configuration will play an important factor in overall performance, and will be driven by the type of applications being run (write versus read ratio, transaction sizes, record contention, etc.). Often self managed, conventional disk configurations, provide better performance than RAID 5. And mirrored disk configurations may be better in high query oriented systems since the data requests can be serviced from two sources.

Database configuration

Block Size

It is recommended that the database be created with a larger block size than the default of 2048. Considerable improvements can be achieved by using a block size of 8192. The data base block size needs to be determined at product installation time, and cannot be changed without recreation of the system.

Please check the relevant documentation for your version of UNIX/LINUX to determine what the maximum block size allowed is. On several versions 8192 is the UNIX/LINUX OS maximum, whereas the Oracle block size can be larger.

There are also some drawbacks when using very large block sizes (larger than 8192), mostly having to do with database segment concurrency. Please check your Oracle documentation for more information.

The recommendation is to set db_block_size = 8192 in your initxxx.ora file. This must be done before creating any Oracle database.

DB_INIT Parameters

Default database installation parameters tend to provide a configuration for a small system setup. The Oracle parameters (init*sid.*ora) can be modified to increase parameters like DB_BLOCK_BUFFERS and SHARED_POOL_SIZE. These parameters are read at database instance startup time. The size of these parameters are dependent on available physical memory. Allocating more than available physical memory would performance degradation.

Unless your system is quite small (less than 500 Mb of data, less than 20 concurrent users) rather than the default “small” the “medium” or “large” options should be evaluated.

Default TIP/dbi user id/password

It is not recommended to use the DBA user id/password for TIP/dbi access to Oracle. The TIP/dbi user id/password can be coded into the schema source, or set as an environment variable. The environment variable will override what is coded in the schema source.

As a rule application specific user id/passwords should be used. These should not have access to the SYSTEM table space.

Folding Tables

Traditionally DMS and C-ISAM files tend to use many fields per record, which translates into many columns per table in the RDBMS. This causes overhead in a RDBMS, the more columns names there are, the more elements need to be processed. For retrieval of single records, or small groups of records, as happens in on-line programs, the impact is insignificant.

However, when complete files are read from top to bottom, as is often the case with traditional mainframe batch programs, processing tends to be significantly slower when comparing it to ISAM file access on a similar UNIX platform. When migrating applications from older mainframe hardware platforms, this effect is often negated by the increase in performance of the new hardware.

The number of elements that need to be processed by the RDBMS can be drastically reduced by folding the tables, which reduces the number of columns used in the RDBMS. This is done by specifying "FOLD ALL" at the schema level, or "FOLD RECORD" at the record level.

Individual column names can still be accessed with view tables, that are generated by dbipre as part of the schema processing.

DML_LOCKS

How they are handled:

DBI never substitutes FOR UPDATE on SELECTs with a WHERE clause as this would effectively lock most of the records in the table. So DBI does a read and then locks the record by re-reading it via its unique ROW_recordname value with FOR UPDATE. This SELECT will never be logged due to its unimportance.

What DBI does:

- If it was an OPEN PROTECTED UPDATE, then dbi issues a re-read FOR UPDATE on every record read.
- If it was an OPEN EXCLUSIVE, then dbi issues a LOCK TABLE EXCLUSIVE for each table in the area.
- If it was an OPEN FOR UPDATE, then dbi issues a LOCK TABLE ROW SHARE for each table in the area.
- If the area was OPEN FOR UPDATE and the FETCH had KEEP or EXCLUSIVE or you had FOR UPDATE in the schema then each record read in that area would be locked as it is read.
- If the application is OPENing all areas in the schema then some sort of table lock will get issued before anything is read.

What to do:

To elevate this problem you may be required to increase the size of the dml_locks in the init.ora file. Oracle error –55 indicates the value for dml_locks is too low. Add more DML_LOCKS until the problem goes away. Adding more DML_LOCKS does not affect performance as long as you make appropriate adjustment to your SHARED_POOL_SIZE parameter.

Minimum DML _LOCKS requirement for TIP/dbi is calculated as:

```
No. of tables * number of dbi threads.
```

TIP/dbi SCHEMA source parameters

Placement of Index and Data

TABLESPACE and INDEXSPACE can be placed in several places in the schema source, for the whole database, the area level and the record level.

This allows for proper load balancing between various table spaces, and ultimately disk drives and disk controllers.

Application data should not be placed in the SYSTEM table space.

CLUSTER

Allows for the placement of SET OWNER record and SET MEMBER record on the same database page in Oracle. This functionality is similar to DMS behavior.

TIP/dbi Environment Variables

Use of TIPDMSCOMMIT

The TIPDMSCOMMIT environment variable specifies the number of calls to TIP/dbi before an automatic commit will be issued. This is required for programs that were not written based a transaction concept, typically batch programs. This may create problems when reading large files from top to bottom.

Performing frequent commits uses Oracle resources, but not performing commits will use of Oracle work space (logs and temp). Running out of space in the Oracle temporary tablespace

Through testing a trade-off point has to be found where this parameter is significantly high enough, so not too many commits are performed, but is low enough that no file system problems arise.

TIPDMSLOG

TIPDMSLOG is both a schema clause for on-line, and an environment variable for batch programs. Turning on all logging options will significantly slow down the application.

For all production environments logging should be turned off.

TIPFCS Relational database access

Inglenet wants to provide access to relational databases through the TIP/ix TIPFCS call interface and IMS/2200 CALL interface as well. In order to accomplish this we require a database interface process similar to the current **TIPFCS**. Relational 'tables' are part of some 'database'. So, both the table name and database name must be defined for TIP to access the information. It makes sense to group all tables of a given database into a configured database interface process.

The approach taken here is to make up a DMS schema definition which copies in all record structures and defines the key fields as if they were DMS indexed records. This schema definition is then run through the schema processor and creates the SQL database definition plus generates the correct file definition records in TIP so that each record is defined as a logical file inside TIP. Since there would be no sets defined the SQL schema would be fairly straightforward.

As an option, in the case where the legacy view of the data and the relational definition are identical, then we should only need to somehow associate the table name with the TIPFCS or IMS/2200 file name. This should work for both transaction programs as well as batch programs through the DBIXFH interface to Micro Focus Cobol.

Generating Database Interface

Once the DMS schema and sub-schemas have been compiled, the SQL database structure is known and the mapping rules defined, the next step is to generate the TIP/dbi interface code to support the database. The utility for doing this is called dbigen. (This utility is normally run automatically by dbischema.) If it finds any inconsistencies the appropriate error messages will be reported.

Syntax:

```
dbigen –d dms-schema –s sql-schema –m mapping-rules –t template-files
```

Where:

dms-schema
is the name of the DMS schema to be generated

dql-schema
is the name of the SQL schema definition. Default is the one generated by schema.

mapping-rules
is the name of the data mapping rules files. Default is the one generated by schema.

template-files
is the location to find the template files used for code generation. Default is system.

Database unload/reload

TIP/dbi is a sub-system of TIP used for the migration of mainframe DMS applications to Oracle on Unix/Linux. The purpose of the `dbiunload` utility is to generate the COBOL DML programs to unload the database from the mainframe and then `dbireload` is used to reload the data into Oracle.

A utility program called **dbiunload** has been developed to help address the problem of actually moving the data from mainframe to UNIX/LINUX. This utility runs on UNIX/LINUX and is part of the TIP/dbi system. Once the TIP/dbi schema compiler has compiled the DMS schema and sub-schema source, then the `dbiunload` utility may be used.

The utility does an analysis of the database structure to determine the correct order in which the records should be processed. For example, all records that own a set must be loaded before any of the set members. When records are members of several sets, then all owner records must be loaded before loading the member records.

Disclaimer

You should always view the programs generated by `dbiunload` as good working examples. A programmer, who is familiar with the database, should review the generated programs and make any required changes to ensure their correct operation.

This is an automated process and there could very well be some subtle issues that are not handled correctly.

DBIBLDUL options

The command summary usage is as follows:

```
TIP/dbi Schema Unload program generator, Version 1.17 2015/06/30
Proper Command format follows

 dbibldul [options] -S schema -U subSchema -P prefix

Where:
  -S schema     Name to be used
  -U subSchema  Name to be used
Where [options] are:
  -P prefix     of program names to generate
  -g            Generate 'unloadplan.txt' in current directory
  -b n          where n is maximum block size of files
                Default is 8192
  -c cmdfile    read commands from named command file
  -i incfile    read commands to include/exclude certain AREAs
  -d dispname   is the environment name to DISPLAY UPON
                SPECIAL-NAMES. dispname IS DBILOG.
  -f n          where n is maximum files per program; Default 29
  -k            Unload via Index scan if possible
  -m            if the Unload is to run on mainframe
```

```
                (The default is to run on mainframe)
  -l            if the Unload is to run on this system
  -O            unload same record of different areas to one file
  -o            unload each record to a different file
  -B            if using MS SQL, generate 'bcp' format files

  The unload progams are generated in the current directory
If -g is not specified then 'unloadplan.txt' is read
  1st create unloadplan.txt, and verify it, then run dbibldul with no options
to do the actual generation of the COBOL unload programs

 The unload progams are compiled and run on the 2200 mainframe
and 'dbireload' is used to load that data into relational database
on Linux

Note: It's a good idea to double check what this utility produces.
```

The program takes the following command line options:

```
dbibldul -S schema -U subSchema -P progName
[-b n -f n -i n -m] areaNames
```

Where:

-S schema
Name to be used. Required parameter.

-U subschema
Name of the sub-schema to be used. Required parameter.

Ideally this sub-schema only references the records which are to be moved.

-P progName
Name of programs to generate. Required parameter.

-m If the Unload is to run on mainframe

-b n n is maximum block size of files. Default value is 8000.

-f n n is maximum files per program. Default value is 31.

-i n n is records added before a checkpoint. Default value is 500.

areas Names of the individual areas of the sub-schema to be processed.

ALL Indicates that all areas of the sub-schema are to be processed.

This is the default choice if no area names are given.

The generated COBOL DML unload programs will be called ***progName*ul.dml**.

If more than one program is required then a sequence number will be appended to **ul** as required. The number of programs generated is determined by the number of records and areas divided by maximum number of files per program. The host operating system will have some limit on how many files can be defined in a single program.

DBIBLDUL operation

Each record type in the database is copied out to a single sequential file. The records are stored in the sequential file in complete DISPLAY format. Using all DISPLAY format data allows these intermediate files to be easily moved from the mainframe. When coming from OS/2200, the sequential file can be created in ASCII and copied over the network.

The database unload programs may be run in any order provided that the source database is not changed until the entire unload sequence has been completed.

Once all of the data is on the target platform, the `dbireload` utility is used for additional processing and to invoke the Oracle `sqlldr` to bulk load the data.

If you know that some areas of the database do not have interdependent records, it would be possible to generate a few sets of unload/reload programs. Then you could run each set concurrently and therefore achieve some overlap in the database re-construction process.

An example of the format of the sequential intermediate file is as follows:

```
01   RD0705.
     02   FILLER.
          10   RD0705-FUNC       PIC X.
          10   RD0705-CODE       PIC 9(4).
          10   RD0705-AREA       PIC 9(4).
          10   RD0705-DBKEY      PIC 9(9).
     02   FILLER.
*         Owner of Set BATCH-DETAIL
          10   RD0705-OR0013     PIC 9(9).
     02   RD0705-DATA.
          05   PD-CUST-NO-705         PIC X(11).
          05   PD-CUST-PO-NO-705 PIC X(18).
          05   PD-PAYMENT-AMT-705    PIC S9(7)V99
                                  SIGN TRAILING
SEPARATE.
```

The record name is **RD<*code*>,** where **<*code*>** is the DMS record code.

The header portion consists of four fields.

Filed	Description
FUNC	Intermediate file function code.

Filed	Description
	"A" means add this record "U" means update this record "D" means DELETE ONLY this record "X" means DELETE ALL this record
CODE	DMS record-id code to identify the record type. (The above example would hold "0705".)
AREA	DMS area-id code to identify the area from which the record came. This is used for DMS/2200 when the record resides in multiple areas of the database.
DBKEY	DMS database key from the source database.
DATA	Group item for the record data. All fields are in complete display format.

For each set that the record was a member of there will be a DBKEY (e.g. RD0705-OR0013) which identifies the owner record of the set.

Multi-record sets

Sets that have multiple record types as members are copied out onto a single intermediate file. In this case the intermediate file is using REDEFINES and the RDxxx-CODE field is used to identify the specific record.

Sorted sets

Sorted sets are the easiest to reload, since the sort field is used to correctly position the member record within the set. All that is required during the reload is that the correct owner record be current of the set.

Ordered sets

Ordered sets are more complicated. The issue is to ensure that the member record is placed into the new database in the correct sequence.

Records that are members of a set are unloaded, in groups, based on the owner of the set. The procedure is to scan the area and for each owner record copy all member records out to the intermediate file.

When loading, the owner record is made current and the new member record is then stored. This requires that the records be unloaded in the correct sequence such that doing a STORE with the owner record current results in the correct placement into the set.

ORDER	STORE with owner current places new record	Unload member via FETCH
FIRST	FIRST	PRIOR
LAST	LAST	NEXT
NEXT	FIRST	PRIOR
PRIOR	LAST	NEXT

This technique works when the record is only a member of one set, but if the record is a member of several sets of different ordering methods, it is not reasonable to only fetch the owner of the set. In this case an extra field is created in the intermediate file which is the database key of the next or prior member of the set such then if this member is made current and then the new record is stored to the database, it will be correctly placed into the set ordering.

Data reformatting

Since the use of **dbiunload**, and the generated programs, results in a database unload/reload, there is an opportunity to change the format of the new database that is created.

This could be done by using the existing schema/sub-schema to generate the database unload programs. But before actually compiling and running the load programs, they could be manually altered to use a new schema/sub-schema. As long as both sets of programs have the same view of the intermediate data this procedure would work. This technique could be used to re-arrange the placement of data fields, alter the size or format of data fields and to add new data fields to a record.

As long as you view the programs generated by `dbibldul` as good working examples, you could achieve many possible re-organization and/or reformatting of the database by manually changing the programs generated by `dbibldul`.

DBIRELOAD Utility

The command summary usage is as follows:

```
TIP/dbi data loader (1.137 2015/07/11) - © 1991-2015 Inglenet Business Solutions
Usage is:
  dbireload [-opts] schemaname

Options:
  -i dir     Input directory path
               Defaults to current directory
  -o dir     Output directory path
               Defaults to current directory; Used as work area
  -s schema  Schema name, already compiled by dbischema
  -c         Invoke sqlplus to drop/create all tables
  -C         Skip enable of CONSTRAINTS a end of load
  -k         Do detailed checking of numeric fields
  -b         In PIC 9 fields, if bad data set to ZERO
  -B         In PIC 9 fields, if bad data do NOT set to ZERO
                   -B is the default action
  -e         In PIC 9(9) or 9(10) COMP fields, if all spaces set to ZERO
  -E         In PIC 9(9) or 9(10) COMP fields, if spaces do NOT set to ZERO
                  -e is the default action
  -z         In PIC 9 fields, if all spaces set to ZERO
  -Z         In PIC 9 fields, if spaces do NOT set to ZERO
                   -z is the default action
  -A         In PIC X fields, if bad data set to SPACE
  -t         Translate character fields for Austria
  -F         Force reload ignoring all errors (database may be suspect)
  -g connect Supply complete Oracle Connection string
  -l rec     Just load/unload the one named record type
  -L file    'file' is in /etc/default/tipdbi.map format & indicates files to load
             -L is used with schemas which are ISAM emulation
             All files defined in this control file are loaded
  -U file    'file' is in /etc/default/tipdbi.map format & indicates files to unload
             -U is used with schemas which are ISAM emulation
             All files defined in this control file are unloaded
             LOCATION DIRECT records get unloaded as DAM files
```

If all records in the schema are MIRAM/ISAM, then dbireload will read the file defined with the –L option to get the location of the ISAM files and then the data from the defined files are loaded into Oracle. The format of this file is the same as that used for `/etc/default/tipdbi.map`. (See the section on TIP/dbi batch interface to Micro Focus Cobol) This will result in any previous data in the tables being lost.

DBKEY work file

An ISAM file is used to map the old database key to the new database key. The intermediate sequential files all hold database keys from the old database. Whenever new record is stored into the new database a record is added to the work file holding the old and new database keys.

Since, UNIX/LINUX may have a limit of 2GB per file and each record in this work file uses about 36 bytes, there may be a limit of 54 million records.

Unload/Reload Example

The following steps would be followed. Get the DMS schema properly defined with all of the options required and field redefinitions, etc. And compile the schema using dbischema and subschema using dbisubschema. For example:

```
dbischema dg-schema.sch

dbisubschema dg-ucs.sub
```

Then run dbiunload to generate the unloadplan.txt file. For example:

```
dbiunload -S dg-schema -U dg-ucs –g
```

This creates a file which is later used by dbiunload to generate the COBOL/DMS unload programs. You have an opportunity to modify the unload plan before code generation. A sample unload plan looks like the following:

```
# Schema: dg-schema
# Define order to process records for unload
# Command line options follow
-S DG-SCHEMA
-U DG-UCS
-f 29
-b 8192
-p dg
-xi unix-ip-address
-xd /2200-sun/cnv
-xu user-id-goes-here
-xp password-goes-here
-m
#
# Sequence of following determines order of unload/reload
```

```
# G marks start of group
# A areaname indicates Area name in group
# R unload record
# T prodarea testarea (For Area name changes)
#

G 1 New unload program starts here C
A A-BEDATA
R BENUM
A A-DGVI
R DGVI
A A-BBDATA
R BB-STAMM
A A-BENAMDATA
R BENAM
A A-DGAKT
R DGAKTHD
A A-DGHI
R DGHI
A A-DGHV
R DGHVH1
A A-DGKTO
R DGUPHD
R DGKTO
A A-DGMVCD
R DGMVCD
A A-DGNI
R DGNI
# Previous unload will use 28 files

G 2 New unload program starts here
A A-DGSALD
R DGSALDKB
R DGSALDHD
A A-DGST
R DGST
A A-DGSTOP
R DGSTOP
A A-DGVS
R DGVSKTO
A A-DGZG
R DGZGHD
A A-DIVDATA
R DGDIV
A CKPT-AREA
R CKPT-REC
# Previous unload will use 22 files
. . . . . .
```

Then run dbibldul to generate the COBOL/DMS unload programs

```
>dbibldul
Generate for unload running on 2200
Generating for SubSchema DG-UCS of Schema DG-SCHEMA
Loading /home/rjn/dms/dg-schema.dd3/dictionary.exp
Loaded /home/rjn/dms/dg-schema.dd3/dictionary.exp in 0 seconds
Unload programs generated for use on OS/2200 mainframe
Zip file created dgul.zip with unload programs
```

The programs along with some starter ECL is generated and placed into a ZIP file for easy file transfer.

Now transfer the unload programs to the mainframe, compile and execute to create a collection of test files holding the data. Then transfer these text files back to the UNIX/LINUX system to a unique directory and run `dbireload` from inside that directory to load the data into Oracle.

Indexed File interface

The database I/O module must also be able to process TIPFCS style of requests. Each record in the database may be accessible as a simulated MASM/ISAM file through the normal CALL TIPFCS and IMS CALL GET interfaces. This will require changes to the TIP startup procedures to recognize file defined as database tables.

Defining an MSAM/ISAM Schema

To define a database of relational tables which are to be processing like **MSAM** or (MIRAM/ISAM) files through the TIPFCS interface you must create a DMS Schema definition defining the record as **LOCATION MODE INDEX SEQUENTIAL** and defining all of the indexes. You must also define an **AREA NAME MIRAM** and declare all records as **WITHIN MIRAM**. The area name **MIRAM** is treated as a special area. An example follows:

```
000001 IDENTIFICATION DIVISION.
000002 SCHEMA NAME IS FCSRDMS.
000003* Sample data base definition for use with TSTFCS, TSP
000004 DATA DIVISION.
000006     VARCHAR ALL
           WAIT GETUP 1.5  RETRY GETUP 0.05
           TRIM NAMES
000007     LOW-VALUES ARE NULL
000008 AREA SECTION.
000009
000010     AREA NAME MIRAM.
000011     AREA CODE IS 10.
000012
000013 RECORD SECTION.
000014
000015
000016 RECORD NAME INVEN.
000017 RECORD CODE 100.
000018 LOCATION MODE INDEX SEQUENTIAL
000019     USING IN-PART-NUM                    AS KEY  1
000020                              DUPLICATES NOT ALLOWED
000021 WITHIN MIRAM AREA.
000022 COPY GC-INREC .
000023
       RECORD NAME DAMINV.
       RECORD CODE 200.
       LOCATION MODE DIRECT
       WITHIN MIRAM AREA.
       COPY GC-INREC .

000024 RECORD NAME TSPFILE.
000025 RECORD CODE 101.
000026 VARCHAR CM-COMPANY
000027 VARCHAR (CM-NUMBER CM-ADDRESS-1)
000031 LOCATION MODE INDEX SEQUENTIAL
000033     USING CM-NUMBER                      AS KEY  1
000034                              DUPLICATES NOT ALLOWED
000035     USING CM-COMPANY                     AS KEY  2
000036                              DUPLICATES ALLOWED
000037     USING CM-TELEPHONE                   AS KEY  3
000038     CHANGES NOT ALLOWED    DUPLICATES ALLOWED
000032     USING CM-NUMBER, CM-TELEPHONE, CM-COMPANY CM-STATUS AS KEY 4
000034                              DUPLICATES NOT ALLOWED
000039 WITHIN MIRAM AREA.
000040 COPY TC-TSP .
```

```
000041
000052 RECORD NAME VBINVEN.
000053 RECORD CODE 104.
000054 LOCATION MODE INDEX SEQUENTIAL
000055     USING VB-PRID                        AS KEY  1
000056                            DUPLICATES NOT ALLOWED
000057 WITHIN MIRAM AREA.
000058 COPY TC-INVEN .
000059
```

For INDEXED records only the data fields get mapped to columns in the relational database. However, if you wanted to be able to use the TIPFCS function FCS-GETRN (read by record number) then each row needs a unique number to be assigned. TIP/dbi will do this if you add the clause WITH DATABASE-KEY to the record definition. For example:

```
RECORD NAME TSPFILE.
 RECORD CODE 101.
 LOCATION MODE INDEX SEQUENTIAL
     USING CM-NUMBER                      AS KEY  1
                            DUPLICATES NOT ALLOWED
     USING CM-COMPANY                     AS KEY  2
                            DUPLICATES ALLOWED
     USING CM-TELEPHONE                   AS KEY  3
     CHANGES NOT ALLOWED    DUPLICATES ALLOWED
     USING CM-NUMBER, CM-TELEPHONE, CM-COMPANY CM-STATUS AS KEY 4
                            DUPLICATES NOT ALLOWED
     WITH DATABASE-KEY
 WITHIN MIRAM AREA.
```

Loading ISAM/Sequential data file to database

A utility called `dbiexpimp` can be used to read the ISAM file and load the data into the database table. The command usage is as follows:

```
TIP/dbi table load/unload, (c) Copyright 1994-2008 Inglenet Business Solutions
Usage:
    dbiexpimp -<options>
Where possible options are:
   -s        schema name  to be used for operation (Required)

   -a        Flat file has <CR> as record separator
   -b n      Bulk load 'n' records at a time (Max 50)
   -c        Log load details to schema.log
   -C n      COMMIT every 'n' records
   -i name   input file name (Sequential)
   -o name   output file name (Sequential)
   -I name   input file name (ISAM)
   -O name   output file name (ISAM)
   -r name   record name
   -t        empty data base table before loading
   -L        Load ISAM file named in /etc/default/tipdbi.map matching -r name
             -L implies truncate the table (-t)
   -U        Unload ISAM file named in /etc/default/tipdbi.map matching -r name
   -B        Unload ISAM file named in /etc/default/tipdbi.map matching -r name in
                                                                    BCP format
   -l        Load data from file named in /etc/default/tipdbi.map matching -r name
   -u        Unload data to file named in /etc/default/tipdbi.map matching -r name
   -m n      Stop loading after 'n' records
   -X        Open input ISAM file with Exclusive lock
```

If the ISAM file has been defined in /etc/default/tipdbi.map (or your local $HOME/.tipdbi.map) then you can reload the ISAM datafile into the data by simply running:

```
dbiexpimp –L myrec
```

With the above example, dbiexpimp will scan the tipdbi.map file and find the record (myrec) and then know which ‘schema’ to open.

Defining a Table to TIPFCS

To define a file which is really a table of a relational database. You must first construct the MIRAM Schema and build it. Then, using SMFILE:

- Define File type as ORACLE (this may be changed to RDBMS later)
- Define Label/Path with the record name defined in the MIRAM Schema
- Define FCS Server with the name of the database I/O module ({MIRAM Schema Name}io)

TIP/ix fopen command

The TIP/ix fopen command (when run from the Unix shell prompt) has some options to change the mode of the file inside the TIP system.

From Unix shell:

`fopen -d filename schema`
Sets file to be RDBMS using the named schema

`fopen -d filename schema`
Sets file to be RDBMS using the named schema

`fopen -x filename`
Sets file to be ISAM

From TIP shell:

`fopen,dbi filename schema`
Sets file to be RDBMS

`fopen,isam filename`
Sets file to be ISAM

For example:

```
fopen -d bmkacct bmark
fopen -d bmkcust bmark
fopen -d bmkcard bmark

fopen -x bmkacct "-A"
fopen -x bmkcust "-B"
fopen -x bmkcard
```

Generate an ISAM Schema from Oracle/SQL

There is a utility called **dbitosch.pc** that will be supplied in source code format. This program must be pre-processed with Oracle proc and compiled. Once compiled it can be used to extract relational table structures from the Oracle data dictionary to create a MIRAM Schema. You may have to edit the result to add information about TABLE SPACE, User-id, password etc. You would then process the MIRAM Schema with the TIP/dbi schema program and build the I/O module. (Do not use the schema.ddl module since Oracle already has the tables defined. Dbitosch takes the following command line options:

Option	Description
-c	generate COBOL COPY books for each table
-C directory	generate the COBOL COPY books into this directory
-S name	specify the Schema name
-U userid	Oracle User Id to connect to Oracle with
-P pass	Oracle password to connect to Oracle with

Following the options is a list of table names to be built into the MIRAM Schema definition. The following example constructs a schema named FCSRDMS and generates COBOL COPY books for the Oracle tables 'tspfile inven orders parts':

```
dbitosch -S FCSRDMS -U system -P manager \
 -C /source/books tspfile inven orders parts
```

A similar tool will be developed for SQL Server 2000.

TIPFCS relative record number & Oracle

The TIPFCS interface to relational database will return a relative record number for all I/O requests. This relative record number is computed from the ROWID. With Oracle the ROWID is an 18-byte field while the TIPFCS relative record number is only 4 bytes. These difference leads to some compromises and restrictions.

Extended ROWIDs use a base 64 encoding of the physical address for each row selected. For example, the following query

```
SELECT ROWID, ename FROM emp
WHERE deptno = 20;
```

might return the following row information:

```
ROWID              ENAME
------------------ ----------
AAAAaoAATAAABrXAAA BORTINS
AAAAaoAATAAABrXAAE RUGGLES
AAAAaoAATAAABrXAAG CHEN
AAAAaoAATAAABrXAAN BLUMBERG
```

An extended ROWID has a four-piece format, OOOOOOFFFBBBBBBRRR:

Where:

OOOOOO
The data object number identifies the database segment (AAAAao in the example). Schema objects in the same segment, such as a cluster of tables, have the same data object number.

FFF
The data file that contains the row (file AAT in the example). File numbers are unique within a database.

BBBBBB
The data block that contains the row (block AAABrX in the example). Block numbers are relative to their data file, not table-space. Therefore, two rows with identical block numbers could reside in two different data files of the same table-space.

RRR The row in the block.

We are hoping that the "data object number" and "datafile" do not change and we'll just pickup the "data block" and "row".

```
                                                  11111111
                                        012345678901234567
                                        ------------------
      ROWID has a four-piece format, OOOOOOFFFBBBBBBRRR:
```

Since data block is 6*6 = 36 bits and row is 3*6 = 18 bits and we must squeeze this into 32 bit we have a problem.

You may want to look at defining DATABASE BLOCK nnn RECORDS in the schema. The default is now 2048 records maximum per database block.

DATABASE BLOCK 2048 RECORDS

The default is to take just the bottom 11 bits of RRR will be taken and just the bottom 21 bits of BBBBBB will be taken. This means that no block should hold more than 2048 rows and no tablespace file should have more than 2097151 blocks or this scheme will not work. If using an 8K block size the max file size is then 16GB.

DATABASE BLOCK 4096 RECORDS

The default is to take just the bottom 12 bits of RRR will be taken and just the bottom 22 bits of BBBBBB will be taken. This means that no block should hold more than 4096 rows and no tablespace file should have more than 1048576 blocks or this scheme will not work. If using an 8K block size the max file size is then 8GB.

DATABASE BLOCK 1024 RECORDS

The default is to take just the bottom 10 bits of RRR will be taken and just the bottom 23 bits of BBBBBB will be taken. This means that no block should hold more than 1024 rows and no tablespace file should have more than 4194303 blocks or this scheme will not work. If using an 8K block size the max file size is then 32GB.

TIP/dbi batch interface to Micro Focus Cobol

Micro Focus COBOL provides for a user written file handler. TIP/dbi provides such a file handler for interfacing **batch** programs through TIP/dbi to Oracle. To compile a batch COBOL program to use this file handler you would use the option CALLFH"DBIXFH". A sample `Makefile` follows.

```
MYTIP=../..

CFLAGS= $(CFLGS)
FLAGS = -gx -C "NOWARNING VSC2 IBMCOMP CALLFH\"DBIXFH\""
MFBAT   = -P -Ox -C "NOWARNING VSC2 IBMCOMP CALLFH\"DBIXFH\" FCDREG"
BAT     = -L$(TIPROOT)/lib -ldbixio -ldbirun -ldbi2 -lbat
BINDIR=$(MYTIP)/bin

.bat:
    cob $(MFBAT) -c -k $<
    cob $(MFBAT) $*.o -o $* -L$(LIBHOME) $(BAT)
    $(RM) $(@F).o $(@F).idy $(@F).int
    $(MV) $(@F) $(BINDIR)
```

The above Makefile is just a sample and you would update your own procedure as needed.

The DBIXFH module will read through a control file called **`/etc/default/tipdbi.map`** on the Unix system for each OPEN of the file. If the OPEN is for a TIP/dbi (Oracle) table, then the DBIXFH module handles the I/O requests. If the file is not found in the control file (**`/etc/default/tipdbi.map`**) then DBIXFH just passes the I/O request onto the normal COBOL file handler EXTFH.

A sample **`/etc/default/tipdbi.map`** file follows:

```
#TIP/dbi batch file mapping
#Path to Isam File, Table Name
[fcsrdms]
/u/tipsrc/tipfiles/tspfix TSPFILE
# Define files for MCC TIP/dbi test
[mccsch]
/home1/mcc/files/cldetcum CDETCUM
/home1/mcc/files/clhdrcum CHDRCUM
/home1/mcc/files/corsdesc CORDESC
/home1/mcc/files/courreq.dates REQDATE
```

The name in square brackets is the schema name, which your application will be using. If the line begins with #, then it is a comment line.

The software searched for the tipdbi.map file to be used. It first checks for a **`$HOME/.tipdbi.map`**, then **`/etc/default/tipdbi.map`** and then **`/etc/tipdbi.map`**.

Batch WHERE & ORDER BY clauses

In addition there is a new function code so that your application program may supply the WHERE and ORDER BY clauses of a SELECT and place the file in START mode.

```
WORKING-STORAGE SECTION.
01       TSPFH-WHERE-PACKET.
         05   FILLER            PICTURE 9(4) COMP SYNC VALUE 24.
         05   FILLER            PICTURE XX.
         05   FILLER            PICTURE X(20)
                                 VALUE "Disk = '8417' ".
01       TSPFH-ORDER-PACKET.
         05   FILLER            PICTURE 9(4) COMP SYNC VALUE 24.
         05   FILLER            PICTURE XX.
         05   FILLER            PICTURE X(20)     VALUE "Telephone ".
   …
     COPY TC-DBISL.
   …
LINKAGE SECTION.
01       DBI-FH.
     COPY TC-DBIFH.
   …
PROCEDURE DIVISION.
   …
         SET ADDRESS OF DBI-FH      TO ADDRESS OF FH--FCD OF TSPSEQ.
         SET FCD-WHERE-CLAUSE       TO ADDRESS OF TSPFH-WHERE-PACKET.
         SET FCD-ORDER-CLAUSE       TO ADDRESS OF TSPFH-ORDER-PACKET.
         CALL "DBIXFH"   USING      DBIFH-SELECT,
                                        DBI-FH.
         READ TSPSEQ.
```

The compile time option FCDREG is what allows you to access “ADDRESS OF FH--FCD OF”.

Batch SELECT column list

You may also limit the columns to be returned with the DBIFH-COLUMNS function. If the table has a lot of columns but your only need a few and the application is going to read a few columns from many records this option may help improve performance by reducing the data exchanged with the database.

```
WORKING-STORAGE SECTION.
01  TSPFH-COLUMNS.
    05  FILLER               PIC 9(4) COMP-4 VALUE 50.
    05  FILLER               PIC X(48) VALUE
           "number1,company,noterminals".
  …
     COPY TC-DBISL.
   …
LINKAGE SECTION.
01       DBI-FH.
     COPY TC-DBIFH.
   …
PROCEDURE DIVISION.
   …
         SET ADDRESS OF DBI-FH      TO ADDRESS OF FH--FCD OF TSPSEQ.
         SET FCD-SELECT-LIST        TO ADDRESS OF TSPFH-COLUMNS.
         CALL "DBIXFH"   USING      DBIFH-COLUMNS,
                                         DBI-FH.
         READ TSPSEQ.
```

Online transaction WHERE & ORDER BY clauses

Online transactions use TIPFCS to access all data and TIP/ix TIPFCS interface supports TIP/dbi by defining the file to be of type RDBMS and giving the 'schema' name. Most of the TIPFCS functions are supported. In addition the FCS-SELECT function has been added to allow application program to supply special ORDER BY and WHERE clauses. For example:

```
05  INVEN-SEL.
    10  SEL-LEN                     PICTURE 9(4) BINARY.
    10  FILLER                      PICTURE XX.
    10  SEL-TEXT                    PICTURE X(64).

05  INVEN-ORD.
    10  ORD-LEN                     PICTURE 9(4) BINARY.
    10  FILLER                      PICTURE XX.
    10  ORD-TEXT                    PICTURE X(64).
    ...
    MOVE "QTY > 20"              TO  SEL-TEXT
    MOVE 13                      TO  SEL-LEN
    MOVE "PART_NUM,QTY"          TO  ORD-TEXT
    MOVE 16                      TO  ORD-LEN
    CALL "TIPFCS"              USING FCS-SELECT
                                     INVEN-PACKET
                                     INVEN-SEL
                                     INVEN-ORD
    CALL "TIPFCS"              USING FCS-GET
                                     INVEN-PACKET
                                     INVEN-RECORD
```

The file access is treated like an FCS-SETL, except that the positioning is lost on any transaction end point, so you cannot keep it active across terminal I/O. You may optionally omit the 4th parameter of FCS-SELECT. The length of the text areas may from 4 to 1024.

Online transaction SELECT column list

You may also limit the columns to be returned with the DBIFH-COLUMNS function. If the table has a lot of columns but your only need a few and the application is going to read a few columns from many records this option may help improve performance by reducing the data exchanged with the database.

```
01  LIST-COLS.
     05  FILLER          PICTURE 9(4) COMP-4 VALUE 50.
     05  FILLER          PICTURE X(48) VALUE
         "number1,company,telephone,dpmgr,date*".

 01  ALL-COLS.
     05  FILLER          PICTURE 9(4) COMP-4 VALUE 6.
     05  FILLER          PICTURE X(4) VALUE "*".
…
PROCEDURE DIVISION …
CALL "TIPFCS" USING FCS-COLUMNS TSPFL-PACKET LIST-COLS.

CALL "TIPFCS" USING FCS-COLUMNS TSPFL-PACKET ALL-COLS.
```

Only the names columns plus any index columns will be read. This can be used in combination with FCS-SELECT. On the next transaction end point, the list of columns reverts back to all columns in the table. You can manually revert back to all columns by setting the column name list to an asterisk "*".

The first field of the packet is a COMP-4 length and the value includes the 2 bytes for the length field. The length of the text areas may from 3 to 1024.

The FCS-GETUP, FCS-PUT, FCS-ADD functions will use all columns in the table regardless of FCS-COLUMNS.

TIP/dbi ODBC interface

TIP/dbi II interfaces with Oracle using Oracle Call Interface (OCI). The OCI module can be optionally replaced with an ODBC module and then TIP/dbi can use any ODBC 3.5 compliant database. On Unix/Linux an ODBC driver manager is normally used and a popular one is unixODBC which is available for free download at www.unixodbc.org.

To access Microsoft SQL Server you may also need a driver from Easysoft (www.easysoft.com).

You can tell TIP/dbi that it should compile the schema for ODBC by specifying:

```
dbischema -T sql2005 myschema.sch
```

Or by adding TARGETDBMS IS ODBC to the start of the schema definition text file as follows:

```
IDENTIFICATION DIVISION.
 SCHEMA NAME IS DMSSCH.
      INDEXSPACE "dbidx".
      TABLESPACE "dbdata".
      MAXIMUM IDLE 5000
      TARGETDBMS IS SQL2005
```

‘ODBC” indicates a generic ODBC database. You may also specify SQL2005 if using Microsoft SQL server 2005 or later and then TIP/dbi will use some special features of MS SQL Server.

For TIP/dbi on Unix/Linux to access an ODBC database like Microsoft SQL Server (2005 or later) you will also need an ODBC driver. This is available for a fee from Easysoft (www.easysoft.com). Once installed you will need to setup the /etc/odbc.ini file and an example follows:

```
[rjnxps]
Driver      = Easysoft ODBC-SQL Server
Description     = SQL Server DSN created during installation
Server      = 192.168.1.131
Port        = 1433
User        = mydoman\myuid
Password        = mypassword
Language        = us_english
Database        = master
Logging     = No
LogFile     = /tmp/odbc.log
QuotedId        = Yes
AnsiNPW     = Yes
Mars_Connection     = Yes
```

In the above example you must define Mars_Connection = Yes as this normally defaults to ‘No’. TIP/dbi requires this option to be enabled. You also need a /etc/odbcinst.ini which is used by the ODBC driver manager.

```
[Easysoft ODBC-SQL Server]
Driver          = /usr/local/easysoft/sqlserver/lib/libessqlsrv.so
Setup           = /usr/local/easysoft/sqlserver/lib/libessqlsrvS.so
Threading               = 0
FileUsage               = 1
DontDLClose             = 1
UsageCount              = 2
```

ODBC Interactive utility

On Linux, TIP/dbi II also comes with a utility called dbisql which is an enhanced version of the isql utility that comes with unixODBC. The enhanced version will collect several input lines and executed them when it sees a line terminated with a semi-colon (;) or a line that starts with the work 'go'. Each command is committed as executed unless you enter a command of 'commit'. One 'commit' is executed then it batches the commands into a transaction until the next 'commit' command. Once you have run dbischema to create the SQL table definitions you could move to the schema.dd3 directory and run the following command to define the tables to the database.

```
dbisql rjnxps <schema.ddl
```

Where 'rjnxps' is the name of your local database definition in the odbc.ini file and schema is the name of the TIP/dbi II schema you defined.

MySQL Support

On Redhat Linux, MySQL comes with the operating system. For TIP/dbi to work correctly, you need to configure MySQL to use the Innodb database engine.

You must install unixODBC and unixODBC-devel (use yum install).

A sample entry for MySQL in /etc/odbc.ini might look like the following:

```
[mytest]
Driver = MySQL
Description = ODBC Driver for local mySQL
Server = localhost
User        = yourname
Password    = yourpwd
Database    = mytest
```

The corresponding entry in /etc/odbcinst.ini might look like:

```
[MySQL]
Description     = ODBC for MySQL
Driver      = /usr/lib/libmyodbc3.so
Setup       = /usr/lib/libodbcmyS.so
FileUsage       = 1
```

If you want to turn on ODBC tracing add the following to /etc/odbcinst.ini

```
[ODBC]
Trace = Yes
TraceFile = /tmp/sql.log
```

To disable tracing, change it to read Trace = No.

For **MariaDB** the entry in /etc/odbcinst.ini might look like:

```
[MySQL]
Description     = ODBC for MySQL
Driver64        = /usr/lib64/libmaodbc.so
Setup64         = /usr/lib64/libodbcmyS.so
FileUsage       = 1
```

Sample DMS schema

DMS/2200 sample Schema

```
*
*    ********************************************************
*    *   DMS/1100 TEST SCHEMA DESCRIPTION STATEMENTS        *
*    ********************************************************
*
 IDENTIFICATION DIVISION
 SCHEMA NAME IS DMS11 IN TIP FILE 39
 INDEXSPACE "TOOLS".
 TABLESPACE "USERS".

 DATA DIVISION

*
*    ********************************************************
*    *   DATABASE DATA NAMES                                *
*    ********************************************************
*
 DATA NAME SECTION
 01  DBDN-CUST-AREA                 USAGE AREA-NAME    CODE 10
 01  DBDN-PROD-AREA                 USAGE AREA-NAME    CODE 11
 01  DBDN-ORD-AREA                  USAGE AREA-NAME    CODE 63
 01  DBDN-FREE-LIMIT                PIC   9(5)
                                     USAGE COMP         CODE 12
 01  DBDN-FREE-COUNT                PIC   9(5)
                                     USAGE COMP         CODE 13
 01  DBDN-PAY-AREA                  USAGE AREA-NAME    CODE 14
 01  DBDN-SDT-REC                   USAGE RECORD-NAME  CODE 65
 01  DBDN-SDT-SET                   USAGE SET-NAME     CODE 93
****  FOLLOWING DATA NAMES ARE USED FOR UTILITY SOFTWARE.
 01  CM-RECORD-NAME                 USAGE RECORD-NAME  CODE 122
 01  CM-AREA-NAME                   USAGE AREA-NAME    CODE 123
 01  CM-SET-NAME                    USAGE SET-NAME     CODE 124
 01  CM-AREA-KEY                    USAGE AREA-KEY     CODE 125
 01  CM-DATABASE-KEY                USAGE DATABASE-KEY CODE 126
*
*    ********************************************************
*    *              AREA DESCRIPTION STATEMENTS             *
*    ********************************************************
*
 AREA SECTION
      AREA CONTROL 2047 AREAS
      AREA LOOKS INCLUDE QUICK-BEFORE-LOOKS

 AREA NAME IS CUSTOMER-AREA.
     AREA CODE IS 10
     AREA MAPS TO TIP FILE
     ALLOCATE 224990 PAGES
         DYNAMICALLY EXPANDABLE TO 262143 PAGES
     PAGES ARE 1792 WORDS
     LOAD IS 85 PERCENT

 AREA NAME IS ORDER-AREA
     AREA CODE IS 20
     MODE IS DATA
     ALLOCATE 100 PAGES
     EXPANDABLE TO 260000  PAGES
     PAGES ARE 672 WORDS
     TRAINING QUALIFIER IS DMSCIF

 AREA NAME IS ORDER-AREA2
     AREA CODE IS 22
```

```
    MODE IS DATA
    ALLOCATE 100 PAGES
    EXPANDABLE TO 260000  PAGES
    PAGES ARE 672 WORDS
    TRAINING QUALIFIER IS DMSCIF

AREA NAME IS ORDER-INDEX
    AREA CODE IS 25
    MODE IS INDEX AREA
    TABLESPACE "TOOLS"
    AREA MAPS TO TIP FILE
    ALLOCATE 500 PAGES
        EXPANDABLE TO 32767 PAGES
    PAGES ARE 896 WORDS

AREA NAME IS PRODUCT-AREA
    AREA CODE IS 30
    AREA MAPS TO TIP FILE
    ALLOCATE 224990 PAGES
        DYNAMICALLY EXPANDABLE TO 262143 PAGES
    PAGES ARE 1792 WORDS
    LOAD IS 85 PERCENT

AREA NAME IS PRODUCT-AREA2
    AREA CODE IS 32
    AREA MAPS TO TIP FILE
    ALLOCATE 224990 PAGES
        DYNAMICALLY EXPANDABLE TO 262143 PAGES
    PAGES ARE 1792 WORDS
    LOAD IS 85 PERCENT

AREA NAME IS PRODUCT-INDEX
    AREA CODE IS 35
    MODE IS INDEX AREA
    TABLESPACE "USERS"
    AREA MAPS TO TIP FILE
    ALLOCATE 500 PAGES
        EXPANDABLE TO 32767 PAGES
    PAGES ARE 896 WORDS

AREA NAME IS PAYMENTS-AREA.
    AREA CODE IS                    70
    MODE IS DATA
    ALLOCATE 8 PAGES
    1 OVERFLOW AT END
    EXPANDABLE TO 260000  PAGES
    PAGES ARE 448 WORDS
    TRAINING QUALIFIER IS DMSCIF

*
*    ********************************************************
*    *             RECORD DESCRIPTION STATEMENTS            *
*    ********************************************************
*
 RECORD SECTION.

 RECORD   NAME CUSTOMER.
     RECORD  CODE  611.
     LOCATION MODE IS CALC DMSCALC
     IN DBDN-CUST-AREA
     USING CUST-NO-611 DUPLICATES ARE NOT ALLOWED
     INDEX USING ASCENDING CUST-NAME-S-611 AS KEY 1
     WITHIN                          CUSTOMER-AREA
     RECORD MODE IS ASCII

     05  CUST-NO-611                          PIC X(11).
*    COMMENT "CUSTOMER NUMBER.  THERE IS A UNIQUE NUMBER FOR
*            EACH CUSTOMER IN THE FILE".
     05  CUST-NAME-S-611                      PIC X(35).
*    COMMENT "CUSTOMER NAME ".
     05  CUST-ADDR-S-611                      PIC X(30).
```

```
*    COMMENT "CUSTOMER ADDRESS".
     05  CUST-CREDIT                          PIC A(3).
*    COMMENT "CUSTOMER CREDIT RATING".
         88  EXCELLENT          VALUE "AAA".
     05  CUST-BIG-611               PIC S9(14)V999 COMP-4.
     05  TEXT-611
         10  TXT-CHAR-611  PIC X OCCURS 100 TIMES
     05  FILLER                               PIC X(25).

 RECORD   NAME CUST-ORDER.
     RECORD  CODE  620.
     LOCATION MODE IS CALC DMSCALC
         IN DBDN-ORD-AREA
     USING FO-NO-620        DUPLICATES NOT ALLOWED.
     WITHIN        ORDER-AREA
     WITHIN        ORDER-AREA2.

     05  FO-NO-620                            PIC 9(8) COMP-3.
     05  ORDER-TEXT.
*    COMMENT "FACTORY ORDER NUMBER".
         10  CUST-PO-NO-620                   PIC X(18).
*    COMMENT "CUSTOMER PURCHASE ORDER NUMBER".
         10  FILLER                           PIC X(27).
*    COMMENT "RESERVED SPACE FOR FUTURE USE".
         10  DATE-SHIP-620                    PIC X(6).
*    COMMENT "SHIP DATE OF THE ORDER
*             SHOULD BE FOUR DAYS BEFORE THE DATE REQUIRED
*             FORMAT IS MMDDYY".
         10  DATE-REQ-620                     PIC X(6).
         10  DATE-NUM-620 REDEFINES DATE-REQ-620 PIC 9(6).
*    COMMENT "DATE REQUIRED BY THE CUSTOMER
*             FORMAT IS MMDDYY".
         10  DATE-PROM-620                    PIC X(6).
*    COMMENT "DATE ORDER WAS PROMISED BY SALEMAN".

     05  TEST-TBL OCCURS 5 TIMES.
         10  ENTRY-CODE                       PIC XX.
         10  BIN-NUM                          PIC 999V9 BINARY.
         10  PAC-NUM                          PIC S9V99 COMP-3.
     05  TEST-TBL2 OCCURS 5 TIMES.
         10  BIN-NUM2                         PIC 9999 BINARY.
     05  TEST-TBL3 OCCURS 5 TIMES.
         10  CODE-A                           PIC XX.
         10  CODE-B                           PIC 99.
     05  SMALL-BIN1                           PIC 99 COMP.
     05  SMALL-BIN2                           PIC 9 COMP.
     05  SMALL-BIN3                           PIC S9 COMP.
     05  TEST-TBL4 OCCURS 5 TIMES.
         10  CODE-C                           PIC 9999 COMP.
         10  CODE-D OCCURS 5 TIMES            PIC X.
*    COMMENT "Above tables to test Schema conversion"
     05  FILLER                               PIC X(32).
*    COMMENT "RESERVED SPACE FOR FUTURE USE".

 RECORD   NAME ORDER-ITEM.
     RECORD  CODE  621.
     LOCATION MODE VIA ITEM         SET
     WITHIN        ORDER-AREA
     WITHIN        ORDER-AREA2.

     05  PROD-NO-621                          PIC X(12).
     05  FILLER                               PIC XXX.
     05  LOT-NO-621                           PIC X(7).
     05  FILLER                               PIC X(4).
     05  QTY-KEY.
       10  QTY-ORD-621                        PIC S9(7)V9 COMP-4.
       10  QTY-SHIP-621                       PIC S9(6) COMP-3.
     05  QTY-BIG-621                          PIC S9(17) COMP-4.
     05  QTY-SEP-621                          PIC S9(4)V99
                        SIGN LEADING SEPARATE.
     05  QTY-TRL-621                          PIC S9(4)V99
```

```
                        SIGN TRAILING SEPARATE.
     05  FILLER                              PIC X(20).

 RECORD   NAME ORD-REMARK.
     RECORD  CODE  622.
     LOCATION MODE DIRECT  CM-DATABASE-KEY, DBDN-ORD-AREA
     WITHIN        ORDER-AREA       AREA.

     03  ORD-REM-CD-622.
*    COMMENT "GROUP LEVEL REFERRING TO COMPLETE REMARK".
     05  REMARK-CD-622                   PIC X.
*    COMMENT "REMARK CODE".
     05  REMARK-SEQ-622                  PIC X.
*    COMMENT "SEQUENCE OF REMARK".
     05  REMARK-622                      PIC X(75).
*    COMMENT "REMARK TEXT".
     05  FILLER                          PIC XXX.
*    COMMENT "RESERVED SPACE FOR FUTURE USE".

 RECORD   NAME PRODUCT.
     RECORD  CODE  631.
     LOCATION MODE CALC
     IN DBDN-PROD-AREA USING PROD-NO-631  DUPLICATES NOT ALLOWED
     INDEX USING ASCENDING PROD-NO-631 AS KEY 1 DUPLICATES NOT
     WITHIN PRODUCT-AREA
     WITHIN PRODUCT-AREA2 INDEX AREA IS ORDER-INDEX.

     05  PROD-NO-631                     PIC X(12).
*    COMMENT "PRODUCT NUMBER EACH PRODUCT HAS A
*            UNIQUE NUMBER".
     05  FILLER                          PIC XX.
*    COMMENT "RESERVED SPACE FOR FUTURE USE".
     05  PROD-DES-INT-631                PIC X(15).
*    COMMENT "INTERNAL DESCRIPTION OF PRODUCT".
     05  PROD-DES-EXT-631                PIC X(30).
*    COMMENT "EXTERNAL DESCRIPTION OF THE PRODUCT".
     05  FILLER                          PIC X(53).
*    COMMENT "RESERVED SPACE FOR FUTURE USE".


 RECORD NAME VENDOR.
        RECORD CODE 651.
      NUMERIC IS CHAR
      PROTECTED UPDATE
      LOCATION MODE IS INDEX SEQUENTIAL
         IN DBDN-PROD-AREA
      USING ASCENDING KEY VENDOR-NAME-651
         LINKS ARE NEXT
         DUPLICATES ARE NOT ALLOWED
      WITHIN PRODUCT-AREA INDEX AREA IS PRODUCT-INDEX
      WITHIN PRODUCT-AREA2 INDEX AREA IS PRODUCT-INDEX.

      05  VENDOR-NAME-651              PIC X(20).
      05  VENDOR-ID                    PIC 9(10).
      05  VENDOR-ADDR                  PIC X(30).
      05  FILLER                       PIC X(52).
*
 RECORD   NAME PAYMENT-BATCH.
     RECORD   CODE 700.
     LOCATION MODE DIRECT
     WITHIN   PAYMENTS-AREA AREA.

*    THE BATCH NUMBER WILL DETERMINE THE PAGE NUMBER IN THE AREA
*    THE RECORD NUMBER WILL ALWAYS BE SET TO 1

 05  PB-BATCH-NUM-700             PIC S9(5).
 05  PB-CLERK-NUM-700             PIC 9(4) BINARY.
 05  PB-BATCH-TOTAL-700           PIC 9(9)V99 COMP-3.
*
 RECORD   NAME PAYMENT-DETAIL.
     RECORD   CODE 705.
```

```
     LOCATION MODE CALC
             IN DBDN-PAY-AREA
     USING PD-CUST-NO-705 DUPLICATES ARE ALLOWED.
     WITHIN   PAYMENTS-AREA AREA.

 05  PD-CUST-NO-705               PIC X(11).
 05  PD-CUST-PO-NO-705            PIC X(18).
 05  PD-PAYMENT-AMT-705           PIC S9(7)V99 COMP-3.
*
 RECORD   NAME PAYMENT-NOTES.
     RECORD   CODE 710.
     LOCATION MODE VIA DETAIL-NOTES SET.
     WITHIN   PAYMENTS-AREA AREA.

 05  PN-COMMENT-710               PIC X(50).
*
 RECORD   NAME PAYMENT-NUTES.
     RECORD   CODE 715.
     LOCATION MODE VIA DETAIL-NOTES SET
       INTERVAL 12 PAGES
     WITHIN   PAYMENTS-AREA AREA.

 05  PN-COMMENT-715               PIC X(50).

 RECORD   NAME PART-DETAIL.
     RECORD   CODE 805.
     LOCATION MODE CALC
             IN DBDN-PAY-AREA
     USING PD-CUST-NO-805 DUPLICATES ARE LAST.
      REDEFINES PD-PART-TEMP
                IF PD-PART-TYPE = 'A' USE PD-PART-AMT-805
     WITHIN   PAYMENTS-AREA AREA.

 05  PD-CUST-NO-805               PIC X(11).
 05  PD-CUST-PO-NO-805            PIC X(18).
 05  PD-PART-TYPE                 PIC X.
 05  PD-PART-TEMP                 PIC X(5).
 05  FILLER REDEFINES PD-PART-TEMP.
     10  PD-PART-AMT-805          PIC S9(7)V99 COMP-3.
*
 RECORD   NAME PART-NOTES.
     RECORD   CODE 810.
     LOCATION MODE VIA PARTS-NOTES SET.
     WITHIN   PAYMENTS-AREA AREA.

 05  PN-SORT-810                  PIC X(8).
 05  PN-COMMENT-810               PIC X(50).
*
 RECORD   NAME PART-NUTES.
     RECORD   CODE 815.
     LOCATION MODE VIA PARTS-NOTES SET.
     WITHIN   PAYMENTS-AREA AREA.

 05  PN-SORT-815                  PIC X(8).
 05  PN-COMMENT-815               PIC X(50).
*
 RECORD   NAME PART-NUTEZ.
     RECORD   CODE 817.
     LOCATION MODE VIA PARTS-NOTES SET
     WITHIN   PAYMENTS-AREA AREA.

 05  PN-SORT-817                  PIC X(8).
 05  PN-COMMENT-817               PIC X(50).
*
*
*
*    ********************************************************
*    *              SET DESCRIPTION STATEMENTS              *
*    ********************************************************
*
 SET SECTION.
```

```
SET NAME ORDOR.
ORDER              SORTED.
MODE                CHAIN    LINKED PRIOR.
OWNER    CUSTOMER.
MEMBER   CUST-ORDER
         LINKED OWNER
         ASCENDING  KEY FO-NO-620  DUPLICATES NOT ALLOWED
         MANDATORY AUTOMATIC.

SET NAME ITEM.
ORDER     NEXT.
MODE                CHAIN    LINKED PRIOR.
OWNER    CUST-ORDER.
MEMBER   ORDER-ITEM
         MANDATORY AUTOMATIC LINKED OWNER.

SET NAME QTY.
ORDER     SORTED.
MODE                CHAIN    LINKED PRIOR.
OWNER    CUST-ORDER.
MEMBER   ORDER-ITEM  ASCENDING KEY QTY-KEY DUPLICATES ALLOWED
         MANDATORY AUTOMATIC LINKED OWNER.

SET NAME QTY-SHIP.
ORDER     SORTED.
MODE                CHAIN    LINKED PRIOR.
OWNER    CUST-ORDER.
MEMBER   ORDER-ITEM  ASCENDING KEY QTY-SHIP-621
                     DUPLICATES ALLOWED
         MANDATORY AUTOMATIC LINKED OWNER.

SET NAME SPEC-REMARK.
ORDER               LAST.
MODE                CHAIN    LINKED PRIOR.
OWNER    CUST-ORDER.
MEMBER   ORD-REMARK
         OPTIONAL  MANUAL.

SET NAME PROD-ORD.
ORDER              SORTED.
MODE                CHAIN    LINKED PRIOR.
OWNER    PRODUCT.
MEMBER   ORDER-ITEM
         OPTIONAL  AUTOMATIC LINKED OWNER
         ASCENDING  KEY LOT-NO-621       DUPLICATES NOT ALLOWED.

SET NAME VEND-PROD.
ORDER               SORTED.
MODE                CHAIN LINKED PRIOR.
OWNER   VENDOR.
MEMBER  PRODUCT
        MANDATORY AUTOMATIC LINKED OWNER
        ASCENDING KEY PROD-NO-631
        DUPLICATES NOT ALLOWED.

SET NAME BATCH-DETAIL.
ORDER               PRIOR.
MODE                CHAIN LINKED PRIOR.
OWNER   PAYMENT-BATCH.
MEMBER  PAYMENT-DETAIL
    MANDATORY AUTOMATIC LINKED OWNER.

SET NAME DETAIL-NOTES.
ORDER               PRIOR.
MODE                CHAIN LINKED PRIOR.
OWNER   PAYMENT-DETAIL
MEMBER  PAYMENT-NOTES
        OPTIONAL MANUAL
MEMBER  PAYMENT-NUTES
        OPTIONAL MANUAL.
```

```
*
 SET NAME PAYMENT-NOTES.
 ORDER               PRIOR.
 MODE                CHAIN LINKED PRIOR.
 OWNER   PAYMENT-DETAIL
 MEMBER  PAYMENT-NOTES
         OPTIONAL MANUAL
 MEMBER  PAYMENT-NUTES
         OPTIONAL MANUAL.
*
 SET NAME PARTS-NOTES.
 ORDER               SORTED.
 MODE                CHAIN LINKED PRIOR.
 OWNER   PART-DETAIL
 MEMBER  PART-NOTES
         DESCENDING KEY PN-SORT-810 DUPLICATES NOT ALLOWED
         OPTIONAL MANUAL
 MEMBER  PART-NUTES
         DESCENDING KEY PN-SORT-815 DUPLICATES NOT ALLOWED
         OPTIONAL MANUAL.
 MEMBER  PART-NUTEZ
         DESCENDING KEY PN-SORT-817 DUPLICATES NOT ALLOWED
         OPTIONAL MANUAL.
*
```

SQL schema

Read this over carefully and compare back to the original DMS schema to see how things are structured and related. You should see that the generated DDL is about as good as you could do manually. All record relationships are maintained using typical methods used in any other relational database. This would be given to Oracle sqlplus to define the table structures. This is also read by dbigen along with the data mapping rules to produce the definition of how TIP/dbi is to manage the database.

```
spool ddl
--
--  Data Base description for DMS11
--
--  Target DBMS System: Oracle Database Server
--
-- Created by TIP/dbi version: 2005/12/08 2.5 R0 - 0067 on platform LINUX
--
--          largest record is 212 bytes
--


--
-- Drop referential contraints for Sets
--
ALTER TABLE cust_order$20 DROP CONSTRAINT set_ordor_20;
ALTER TABLE cust_order$22 DROP CONSTRAINT set_ordor_22;
ALTER TABLE payment_detail DROP CONSTRAINT set_batch_detail;

--
-- Table   1, Rcsz= 198, SR620 : CUST-ORDER$20 : of ORDER-AREA :
--             46 SQL columns
--           Estimated Max Row Size  288
--
DROP TABLE  cust_order$20 CASCADE CONSTRAINTS;
CREATE TABLE cust_order$20 (
        fo_no_620                       DECIMAL(8) NOT NULL,
        CONSTRAINT k0_calc_sr0620_20 UNIQUE(fo_no_620)
```

```
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        cust_po_no_620                  CHAR(18),
        filler01                        CHAR(27),
        date_ship_620                   CHAR(6),
        date_req_620                    CHAR(6),
        date_prom_620                   CHAR(6),
        entry_code01                    CHAR(2),
        bin_num01                       DECIMAL(9,1),
        pac_num01                       DECIMAL(3,2),
        entry_code02                    CHAR(2),
        bin_num02                       DECIMAL(9,1),
        pac_num02                       DECIMAL(3,2),
        entry_code03                    CHAR(2),
        bin_num03                       DECIMAL(9,1),
        pac_num03                       DECIMAL(3,2),
        entry_code04                    CHAR(2),
        bin_num04                       DECIMAL(9,1),
        pac_num04                       DECIMAL(3,2),
        entry_code05                    CHAR(2),
        bin_num05                       DECIMAL(9,1),
        pac_num05                       DECIMAL(3,2),
        bin_num201                      SMALLINT,
        bin_num202                      SMALLINT,
        bin_num203                      SMALLINT,
        bin_num204                      SMALLINT,
        bin_num205                      SMALLINT,
        test_tbl301                     CHAR(4),
        test_tbl302                     CHAR(4),
        test_tbl303                     CHAR(4),
        test_tbl304                     CHAR(4),
        test_tbl305                     CHAR(4),
        small_bin1                      DECIMAL(2),
        small_bin2                      DECIMAL(2),
        small_bin3                      DECIMAL(2),
        code_c01                        SMALLINT,
        code_d01                        CHAR(5),
        code_c02                        SMALLINT,
        code_d02                        CHAR(5),
        code_c03                        SMALLINT,
        code_d03                        CHAR(5),
        code_c04                        SMALLINT,
        code_d04                        CHAR(5),
        code_c05                        SMALLINT,
        code_d05                        CHAR(5),
        row_cust_order                  INTEGER NOT NULL,
        CONSTRAINT pk_cust_order$20 PRIMARY KEY(row_cust_order)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        own_ordor                       INTEGER NOT NULL
       )
                    STORAGE (FREELISTS 3 )
       TABLESPACE USERS;
--
-- Table   2, Rcsz= 198, SR620 : CUST-ORDER$22 : of ORDER-AREA2 :
--             46 SQL columns
--           Estimated Max Row Size  288
--
DROP TABLE  cust_order$22 CASCADE CONSTRAINTS;
CREATE TABLE cust_order$22 (
        fo_no_620                       DECIMAL(8) NOT NULL,
        CONSTRAINT k0_calc_sr0620_22 UNIQUE(fo_no_620)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        cust_po_no_620                  CHAR(18),
        filler01                        CHAR(27),
        date_ship_620                   CHAR(6),
        date_req_620                    CHAR(6),
        date_prom_620                   CHAR(6),
        entry_code01                    CHAR(2),
        bin_num01                       DECIMAL(9,1),
        pac_num01                       DECIMAL(3,2),
        entry_code02                    CHAR(2),
        bin_num02                       DECIMAL(9,1),
```

```
        pac_num02                       DECIMAL(3,2),
        entry_code03                    CHAR(2),
        bin_num03                       DECIMAL(9,1),
        pac_num03                       DECIMAL(3,2),
        entry_code04                    CHAR(2),
        bin_num04                       DECIMAL(9,1),
        pac_num04                       DECIMAL(3,2),
        entry_code05                    CHAR(2),
        bin_num05                       DECIMAL(9,1),
        pac_num05                       DECIMAL(3,2),
        bin_num201                      SMALLINT,
        bin_num202                      SMALLINT,
        bin_num203                      SMALLINT,
        bin_num204                      SMALLINT,
        bin_num205                      SMALLINT,
        test_tbl301                     CHAR(4),
        test_tbl302                     CHAR(4),
        test_tbl303                     CHAR(4),
        test_tbl304                     CHAR(4),
        test_tbl305                     CHAR(4),
        small_bin1                      DECIMAL(2),
        small_bin2                      DECIMAL(2),
        small_bin3                      DECIMAL(2),
        code_c01                        SMALLINT,
        code_d01                        CHAR(5),
        code_c02                        SMALLINT,
        code_d02                        CHAR(5),
        code_c03                        SMALLINT,
        code_d03                        CHAR(5),
        code_c04                        SMALLINT,
        code_d04                        CHAR(5),
        code_c05                        SMALLINT,
        code_d05                        CHAR(5),
        row_cust_order                  INTEGER NOT NULL,
        CONSTRAINT pk_cust_order$22 PRIMARY KEY(row_cust_order)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        own_ordor                       INTEGER NOT NULL
       )
                    STORAGE (FREELISTS 3 )
       TABLESPACE USERS;
--
-- Table   3, Rcsz= 212, SR611 : CUSTOMER : 7 SQL columns
--           Estimated Max Row Size  212
--
DROP TABLE  customer CASCADE CONSTRAINTS;
CREATE TABLE customer (
        cust_no_611                     CHAR(11) NOT NULL,
        CONSTRAINT k0_customer UNIQUE(cust_no_611)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        cust_name_s_611                 CHAR(35) NOT NULL,
        cust_addr_s_611                 CHAR(30),
        cust_credit                     CHAR(3),
        cust_big_611                    DECIMAL(17,3),
        text_611                        CHAR(100),
        row_customer                    INTEGER NOT NULL,
        CONSTRAINT pk_customer PRIMARY KEY(row_customer)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS
       )
                    STORAGE (FREELISTS 3 )
       TABLESPACE USERS;
 CREATE INDEX k1_customer ON customer
        (cust_name_s_611, row_customer)
 STORAGE (FREELISTS 3)  TABLESPACE TOOLS;
--
-- Table   4, Rcsz=  80, SR622 : ORD-REMARK : 7 SQL columns
--           Estimated Max Row Size  104
--
DROP TABLE  ord_remark CASCADE CONSTRAINTS;
CREATE TABLE ord_remark (
        cd                              CHAR(1),
        seq                             CHAR(1),
```

```
        remark                          CHAR(75),
        row_ord_remark                  INTEGER NOT NULL,
        CONSTRAINT pk_ord_remark PRIMARY KEY(row_ord_remark)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        aid_spec_remark                 INTEGER,
        own_spec_remark                 INTEGER,
        pos_spec_remark                 FLOAT
       )
                    STORAGE (FREELISTS 3 )
      TABLESPACE USERS;
--
-- Table   5, Rcsz=  76, SR621 : ORDER-ITEM$20 : of ORDER-AREA :
--             19 SQL columns
--           Estimated Max Row Size  124
--
DROP TABLE  order_item$20 CASCADE CONSTRAINTS;
CREATE TABLE order_item$20 (
        prod_no_621                     CHAR(12),
        filler01                        CHAR(3),
        lot_no_621                      CHAR(7) NOT NULL,
        filler02                        CHAR(4),
        qty_ord_621                     DECIMAL(18,1) NOT NULL,
        qty_ship_621                    DECIMAL(6) NOT NULL,
        qty_big_621                     DECIMAL(17),
        qty_sep_621                     DECIMAL(7,2),
        qty_trl_621                     DECIMAL(7,2),
        row_order_item                  INTEGER NOT NULL,
        CONSTRAINT pk_order_item$20 PRIMARY KEY(row_order_item)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        aid_item                        INTEGER NOT NULL,
        own_item                        INTEGER NOT NULL,
        pos_item                        FLOAT NOT NULL,
        aid_qty                         INTEGER NOT NULL,
        own_qty                         INTEGER NOT NULL,
        aid_qty_ship                    INTEGER NOT NULL,
        own_qty_ship                    INTEGER NOT NULL,
        aid_prod_ord                    INTEGER,
        own_prod_ord                    INTEGER
       )
                    STORAGE (FREELISTS 3 )
      TABLESPACE USERS;
--
-- Table   6, Rcsz=  76, SR621 : ORDER-ITEM$22 : of ORDER-AREA2 :
--             19 SQL columns
--           Estimated Max Row Size  124
--
DROP TABLE  order_item$22 CASCADE CONSTRAINTS;
CREATE TABLE order_item$22 (
        prod_no_621                     CHAR(12),
        filler01                        CHAR(3),
        lot_no_621                      CHAR(7) NOT NULL,
        filler02                        CHAR(4),
        qty_ord_621                     DECIMAL(18,1) NOT NULL,
        qty_ship_621                    DECIMAL(6) NOT NULL,
        qty_big_621                     DECIMAL(17),
        qty_sep_621                     DECIMAL(7,2),
        qty_trl_621                     DECIMAL(7,2),
        row_order_item                  INTEGER NOT NULL,
        CONSTRAINT pk_order_item$22 PRIMARY KEY(row_order_item)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        aid_item                        INTEGER NOT NULL,
        own_item                        INTEGER NOT NULL,
        pos_item                        FLOAT NOT NULL,
        aid_qty                         INTEGER NOT NULL,
        own_qty                         INTEGER NOT NULL,
        aid_qty_ship                    INTEGER NOT NULL,
        own_qty_ship                    INTEGER NOT NULL,
        aid_prod_ord                    INTEGER,
        own_prod_ord                    INTEGER
       )
                    STORAGE (FREELISTS 3 )
```

```
       TABLESPACE USERS;
--
-- Table   7, Rcsz=  35, SR805 : PART-DETAIL : 6 SQL columns
--           Estimated Max Row Size   60
--
DROP TABLE  part_detail CASCADE CONSTRAINTS;
CREATE TABLE part_detail (
        cust_no_805                    CHAR(11) NOT NULL,
        cust_po_no_805                 CHAR(18),
        part_type                      CHAR(1),
        part_temp                      CHAR(5),
        part_amt_805                   DECIMAL(9,2),
        row_part_detail                INTEGER NOT NULL,
        CONSTRAINT pk_part_detail PRIMARY KEY(row_part_detail)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS
       )
                   STORAGE (FREELISTS 3 )
       TABLESPACE USERS;
 CREATE INDEX k0_part_detail ON part_detail
        (cust_no_805, row_part_detail)
 STORAGE (FREELISTS 3)  TABLESPACE TOOLS;
--
-- Table   8, Rcsz=  58, SR810 : PART-NOTES : 4 SQL columns
--           Estimated Max Row Size   76
--
DROP TABLE  part_notes CASCADE CONSTRAINTS;
CREATE TABLE part_notes (
        pn_sort_810                    CHAR(8) NOT NULL,
        pn_comment_810                 CHAR(50),
        row_part_notes                 INTEGER NOT NULL,
        CONSTRAINT pk_part_notes PRIMARY KEY(row_part_notes)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        own_parts_notes                INTEGER
       )
                   STORAGE (FREELISTS 3 )
       TABLESPACE USERS;
--
-- Table   9, Rcsz=  58, SR815 : PART-NUTES : 4 SQL columns
--           Estimated Max Row Size   76
--
DROP TABLE  part_nutes CASCADE CONSTRAINTS;
CREATE TABLE part_nutes (
        pn_sort_815                    CHAR(8) NOT NULL,
        pn_comment_815                 CHAR(50),
        row_part_nutes                 INTEGER NOT NULL,
        CONSTRAINT pk_part_nutes PRIMARY KEY(row_part_nutes)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        own_parts_notes                INTEGER
       )
                   STORAGE (FREELISTS 3 )
       TABLESPACE USERS;
--
-- Table  10, Rcsz=  58, SR817 : PART-NUTEZ : 4 SQL columns
--           Estimated Max Row Size   76
--
DROP TABLE  part_nutez CASCADE CONSTRAINTS;
CREATE TABLE part_nutez (
        pn_sort_817                    CHAR(8) NOT NULL,
        pn_comment_817                 CHAR(50),
        row_part_nutez                 INTEGER NOT NULL,
        CONSTRAINT pk_part_nutez PRIMARY KEY(row_part_nutez)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        own_parts_notes                INTEGER
       )
                   STORAGE (FREELISTS 3 )
       TABLESPACE USERS;
--
-- Table  11, Rcsz=  13, SR700 : PAYMENT-BATCH : 4 SQL columns
--           Estimated Max Row Size   36
--
DROP TABLE  payment_batch CASCADE CONSTRAINTS;
```

```
CREATE TABLE payment_batch (
        batch_num                       DECIMAL(5),
        clerk_num                       SMALLINT,
        batch_total                     DECIMAL(11,2),
        row_payment_batch               INTEGER NOT NULL,
        CONSTRAINT pk_payment_batch PRIMARY KEY(row_payment_batch)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS
       )
                    STORAGE (FREELISTS 3 )
       TABLESPACE USERS;
--
-- Table  12, Rcsz=  34, SR705 : PAYMENT-DETAIL : 6 SQL columns
--           Estimated Max Row Size   64
--
DROP TABLE  payment_detail CASCADE CONSTRAINTS;
CREATE TABLE payment_detail (
        cust_no                         CHAR(11) NOT NULL,
        cust_po_no                      CHAR(18),
        payment_amt                     DECIMAL(9,2),
        row_payment_detail              INTEGER NOT NULL,
        CONSTRAINT pk_payment_detail PRIMARY KEY(row_payment_detail)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        own_batch_detail                INTEGER NOT NULL,
        pos_batch_detail                FLOAT NOT NULL
       )
                    STORAGE (FREELISTS 3 )
       TABLESPACE USERS;
 CREATE INDEX k0_payment_detail ON payment_detail
        (cust_no)
 STORAGE (FREELISTS 3)  TABLESPACE TOOLS;
--
-- Table  13, Rcsz=  50, SR710 : PAYMENT-NOTES : 6 SQL columns
--           Estimated Max Row Size   80
--
DROP TABLE  payment_notes CASCADE CONSTRAINTS;
CREATE TABLE payment_notes (
        pn_comment_710                  CHAR(50),
        row_payment_notes               INTEGER NOT NULL,
        CONSTRAINT pk_payment_notes PRIMARY KEY(row_payment_notes)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        own_detail_notes                INTEGER,
        pos_detail_notes                FLOAT,
        own_payment_notes               INTEGER,
        pos_payment_notes               FLOAT
       )
                    STORAGE (FREELISTS 3 )
       TABLESPACE USERS;
--
-- Table  14, Rcsz=  50, SR715 : PAYMENT-NUTES : 6 SQL columns
--           Estimated Max Row Size   80
--
DROP TABLE  payment_nutes CASCADE CONSTRAINTS;
CREATE TABLE payment_nutes (
        pn_comment_715                  CHAR(50),
        row_payment_nutes               INTEGER NOT NULL,
        CONSTRAINT pk_payment_nutes PRIMARY KEY(row_payment_nutes)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        own_detail_notes                INTEGER,
        pos_detail_notes                FLOAT,
        own_payment_notes               INTEGER,
        pos_payment_notes               FLOAT
       )
                    STORAGE (FREELISTS 3 )
       TABLESPACE USERS;
--
-- Table  15, Rcsz= 112, SR631 : PRODUCT$30 : of PRODUCT-AREA :
--             7 SQL columns
--           Estimated Max Row Size   84
--
DROP TABLE  product$30 CASCADE CONSTRAINTS;
CREATE TABLE product$30 (
```

```
        prod_no_631                     CHAR(12) NOT NULL,
        CONSTRAINT k0_calc_sr0631_30 UNIQUE(prod_no_631)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        filler01                        CHAR(2),
        prod_des_int_631                CHAR(15),
        prod_des_ext_631                CHAR(30),
        row_product                     INTEGER NOT NULL,
        CONSTRAINT pk_product$30 PRIMARY KEY(row_product)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        aid_vend_prod                   INTEGER NOT NULL,
        own_vend_prod                   INTEGER NOT NULL
       )
                    STORAGE (FREELISTS 3 )
      TABLESPACE USERS;
--
-- Table  16, Rcsz= 112, SR631 : PRODUCT$32 : of PRODUCT-AREA2 :
--             7 SQL columns
--           Estimated Max Row Size   84
--
DROP TABLE  product$32 CASCADE CONSTRAINTS;
CREATE TABLE product$32 (
        prod_no_631                     CHAR(12) NOT NULL,
        CONSTRAINT k0_calc_sr0631_32 UNIQUE(prod_no_631)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        filler01                        CHAR(2),
        prod_des_int_631                CHAR(15),
        prod_des_ext_631                CHAR(30),
        row_product                     INTEGER NOT NULL,
        CONSTRAINT pk_product$32 PRIMARY KEY(row_product)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        aid_vend_prod                   INTEGER NOT NULL,
        own_vend_prod                   INTEGER NOT NULL
       )
                    STORAGE (FREELISTS 3 )
      TABLESPACE USERS;
--
-- Table  17, Rcsz= 112, SR651 : VENDOR$30 : of PRODUCT-AREA :
--             4 SQL columns
--           Estimated Max Row Size   72
--
DROP TABLE  vendor$30 CASCADE CONSTRAINTS;
CREATE TABLE vendor$30 (
        name_651                        CHAR(20) NOT NULL,
        CONSTRAINT k1_index_01_0651_30 UNIQUE(name_651)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        id                              CHAR(10),
        addr                            CHAR(30),
        row_vendor                      INTEGER NOT NULL,
        CONSTRAINT pk_vendor$30 PRIMARY KEY(row_vendor)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS
       )
                    STORAGE (FREELISTS 3 )
      TABLESPACE USERS;
--
-- Table  18, Rcsz= 112, SR651 : VENDOR$32 : of PRODUCT-AREA2 :
--             4 SQL columns
--           Estimated Max Row Size   72
--
DROP TABLE  vendor$32 CASCADE CONSTRAINTS;
CREATE TABLE vendor$32 (
        name_651                        CHAR(20) NOT NULL,
        CONSTRAINT k1_index_01_0651_32 UNIQUE(name_651)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS,
        id                              CHAR(10),
        addr                            CHAR(30),
        row_vendor                      INTEGER NOT NULL,
        CONSTRAINT pk_vendor$32 PRIMARY KEY(row_vendor)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE TOOLS
       )
                    STORAGE (FREELISTS 3 )
      TABLESPACE USERS;
```

```
--
--  Set definitions
--

--
--  Set:       ORDOR                              ORDER SORTED NDUP
--       OWNER  CUSTOMER
--       MEMBER CUST-ORDER$20            FO-NO-620        (Id 200620) length 9
--       MEMBER CUST-ORDER$22            FO-NO-620        (Id 220620) length 9
--
DROP   TABLE ordor CASCADE CONSTRAINTS;
CREATE TABLE ordor (
        own_ordor                      INTEGER NOT NULL,
-- Member sort field FO-NO-620
        pos_ordor                      DECIMAL(9) NOT NULL,
        member_id                      INTEGER NOT NULL,
        member_row                     INTEGER NOT NULL,
        CONSTRAINT pk_ordor
        PRIMARY KEY (own_ordor, pos_ordor, member_id)
       )
 STORAGE (FREELISTS 3)  TABLESPACE TOOLS;
CREATE UNIQUE INDEX x_ordor ON ordor (member_id, member_row);
--
--  Set:       ITEM                               ORDER NEXT
--       OWNER  CUST-ORDER                         Multi-Area
--       MEMBER ORDER-ITEM$20        VIA   (Id 200621)
--       MEMBER ORDER-ITEM$22        VIA   (Id 220621)
--
DROP   TABLE item CASCADE CONSTRAINTS;
CREATE TABLE item (
        aid_item                       INTEGER NOT NULL,
        own_item                       INTEGER NOT NULL,
        pos_item                       FLOAT NOT NULL,
        member_id                      INTEGER NOT NULL,
        member_row                     INTEGER NOT NULL,
        CONSTRAINT pk_item
        PRIMARY KEY (aid_item,
                     own_item, pos_item)
       )
 STORAGE (FREELISTS 3)  TABLESPACE TOOLS;
CREATE UNIQUE INDEX x_item ON item (member_id, member_row);
--
--  Set:       QTY                                ORDER SORTED DUPS
--       OWNER  CUST-ORDER                         Multi-Area
--       MEMBER ORDER-ITEM$20            QTY-ORD-621      (Id 200621) length 9
--       MEMBER ORDER-ITEM$22            QTY-ORD-621      (Id 220621) length 9
--
DROP   TABLE qty CASCADE CONSTRAINTS;
CREATE TABLE qty (
        aid_qty                        INTEGER NOT NULL,
        own_qty                        INTEGER NOT NULL,
-- Member sort field QTY-ORD-621
-- Member sort field QTY-SHIP-621
        pos_qty                        DECIMAL(18,1) NOT NULL,
        pos_qty1                       DECIMAL(6) NOT NULL,
        member_id                      INTEGER NOT NULL,
        member_row                     INTEGER NOT NULL
       )
 STORAGE (FREELISTS 3)  TABLESPACE TOOLS;
CREATE UNIQUE INDEX x_qty ON qty (member_id, member_row);
CREATE INDEX pk_qty ON qty
       (aid_qty,own_qty, pos_qty ,
                      pos_qty1, member_id, member_row)
 STORAGE (FREELISTS 3)  TABLESPACE TOOLS;
--
--  Set:       QTY-SHIP                           ORDER SORTED DUPS
--       OWNER  CUST-ORDER                         Multi-Area
--       MEMBER ORDER-ITEM$20            QTY-SHIP-621     (Id 200621) length 7
--       MEMBER ORDER-ITEM$22            QTY-SHIP-621     (Id 220621) length 7
--
```

```
DROP   TABLE qty_ship CASCADE CONSTRAINTS;
CREATE TABLE qty_ship (
        aid_qty_ship                    INTEGER NOT NULL,
        own_qty_ship                    INTEGER NOT NULL,
-- Member sort field QTY-SHIP-621
        pos_qty_ship                    DECIMAL(7) NOT NULL,
        member_id                       INTEGER NOT NULL,
        member_row                      INTEGER NOT NULL
       )
 STORAGE (FREELISTS 3)  TABLESPACE TOOLS;
CREATE UNIQUE INDEX x_qty_ship ON qty_ship (member_id, member_row);
CREATE INDEX pk_qty_ship ON qty_ship
       (aid_qty_ship,own_qty_ship, pos_qty_ship, member_id, member_row)
 STORAGE (FREELISTS 3)  TABLESPACE TOOLS;
--
--  Set:       SPEC-REMARK                          ORDER LAST
--       OWNER  CUST-ORDER                         Multi-Area
--       MEMBER ORD-REMARK
--
CREATE INDEX spec_remark ON ord_remark
       (
                    aid_spec_remark,
                    own_spec_remark,
           pos_spec_remark
       )
 STORAGE (FREELISTS 3)  TABLESPACE TOOLS;
--
--  Set:       PROD-ORD                             ORDER SORTED NDUP
--       OWNER  PRODUCT                            Multi-Area
--       MEMBER ORDER-ITEM$20            LOT-NO-621       (Id 200621) length 7
--       MEMBER ORDER-ITEM$22            LOT-NO-621       (Id 220621) length 7
--
DROP   TABLE prod_ord CASCADE CONSTRAINTS;
CREATE TABLE prod_ord (
        aid_prod_ord                    INTEGER NOT NULL,
        own_prod_ord                    INTEGER NOT NULL,
-- Member sort field LOT-NO-621
        pos_prod_ord                    CHAR(7) NOT NULL,
        member_id                       INTEGER NOT NULL,
        member_row                      INTEGER NOT NULL,
        CONSTRAINT pk_prod_ord
        PRIMARY KEY (aid_prod_ord,
                    own_prod_ord, pos_prod_ord, member_id)
       )
 STORAGE (FREELISTS 3)  TABLESPACE TOOLS;
CREATE UNIQUE INDEX x_prod_ord ON prod_ord (member_id, member_row);
--
--  Set:       VEND-PROD                            ORDER SORTED NDUP
--       OWNER  VENDOR                             Multi-Area
--       MEMBER PRODUCT$30               PROD-NO-631      (Id 300631) length 12
--       MEMBER PRODUCT$32               PROD-NO-631      (Id 320631) length 12
--
DROP   TABLE vend_prod CASCADE CONSTRAINTS;
CREATE TABLE vend_prod (
        aid_vend_prod                   INTEGER NOT NULL,
        own_vend_prod                   INTEGER NOT NULL,
-- Member sort field PROD-NO-631
        pos_vend_prod                   CHAR(12) NOT NULL,
        member_id                       INTEGER NOT NULL,
        member_row                      INTEGER NOT NULL,
        CONSTRAINT pk_vend_prod
        PRIMARY KEY (aid_vend_prod,
                    own_vend_prod, pos_vend_prod, member_id)
       )
 STORAGE (FREELISTS 3)  TABLESPACE TOOLS;
CREATE UNIQUE INDEX x_vend_prod ON vend_prod (member_id, member_row);
--
--  Set:       BATCH-DETAIL                         ORDER PRIOR
--       OWNER  PAYMENT-BATCH
--       MEMBER PAYMENT-DETAIL
--
```

```
CREATE UNIQUE INDEX batch_detail ON payment_detail
       (
                  own_batch_detail,
           pos_batch_detail
       )
 STORAGE (FREELISTS 3)  TABLESPACE TOOLS;
--
--  Set:       DETAIL-NOTES                      ORDER PRIOR
--       OWNER  PAYMENT-DETAIL
--       MEMBER PAYMENT-NOTES        VIA   (Id 700710)
--       MEMBER PAYMENT-NUTES        VIA   (Id 700715)
--
DROP   TABLE detail_notes CASCADE CONSTRAINTS;
CREATE TABLE detail_notes (
        own_detail_notes               INTEGER NOT NULL,
        pos_detail_notes               FLOAT NOT NULL,
        member_id                      INTEGER NOT NULL,
        member_row                     INTEGER NOT NULL,
        CONSTRAINT pk_detail_notes
        PRIMARY KEY (own_detail_notes, pos_detail_notes)
       )
 STORAGE (FREELISTS 3)  TABLESPACE TOOLS;
CREATE UNIQUE INDEX x_detail_notes ON detail_notes (member_id, member_row);
--
--  Set:       PAYMENT-NOTES                     ORDER PRIOR
--       OWNER  PAYMENT-DETAIL
--       MEMBER PAYMENT-NOTES              (Id 700710)
--       MEMBER PAYMENT-NUTES              (Id 700715)
--
DROP   TABLE S_payment_notes CASCADE CONSTRAINTS;
CREATE TABLE S_payment_notes (
        own_payment_notes              INTEGER NOT NULL,
        pos_payment_notes              FLOAT NOT NULL,
        member_id                      INTEGER NOT NULL,
        member_row                     INTEGER NOT NULL,
        CONSTRAINT pk_payment_notes
        PRIMARY KEY (own_payment_notes, pos_payment_notes)
       )
 STORAGE (FREELISTS 3)  TABLESPACE TOOLS;
CREATE UNIQUE INDEX x_payment_notes ON S_payment_notes (member_id, member_row);
--
--  Set:       PARTS-NOTES                       ORDER SORTED NDUP
--       OWNER  PART-DETAIL
--       MEMBER PART-NOTES           VIA PN-SORT-810      (Id 700810) length 8
--       MEMBER PART-NUTES           VIA PN-SORT-815      (Id 700815) length 8
--       MEMBER PART-NUTEZ           VIA PN-SORT-817      (Id 700817) length 8
--
DROP   TABLE parts_notes CASCADE CONSTRAINTS;
CREATE TABLE parts_notes (
        own_parts_notes                INTEGER NOT NULL,
-- Member sort field PN-SORT-810
        pos_parts_notes                CHAR(8) NOT NULL,
        member_id                      INTEGER NOT NULL,
        member_row                     INTEGER NOT NULL,
        CONSTRAINT pk_parts_notes
        PRIMARY KEY (own_parts_notes, pos_parts_notes, member_id)
       )
 STORAGE (FREELISTS 3)  TABLESPACE TOOLS;
CREATE UNIQUE INDEX x_parts_notes ON parts_notes (member_id, member_row);
--
-- Most columns in any table was 46
-- Most indexes on any table was 4
--
-- Sequences for Unique DBKEY values for each DMS Record
--
DROP   SEQUENCE dbk_cust_order$20;
CREATE SEQUENCE dbk_cust_order$20            START WITH  1 INCREMENT BY  1 CACHE 5;
DROP   SEQUENCE dbk_cust_order$22;
CREATE SEQUENCE dbk_cust_order$22            START WITH  1 INCREMENT BY  1 CACHE 5;
DROP   SEQUENCE dbk_customer;
CREATE SEQUENCE dbk_customer                 START WITH  1 INCREMENT BY  1 CACHE 5;
```

```
DROP   SEQUENCE dbk_ord_remark;
CREATE SEQUENCE dbk_ord_remark                START WITH  1 INCREMENT BY  1 CACHE 5;
DROP   SEQUENCE dbk_order_item$20;
CREATE SEQUENCE dbk_order_item$20             START WITH  1 INCREMENT BY  1 CACHE 5;
DROP   SEQUENCE dbk_order_item$22;
CREATE SEQUENCE dbk_order_item$22             START WITH  1 INCREMENT BY  1 CACHE 5;
DROP   SEQUENCE dbk_part_detail;
CREATE SEQUENCE dbk_part_detail               START WITH  1 INCREMENT BY  1 CACHE 5;
DROP   SEQUENCE dbk_part_notes;
CREATE SEQUENCE dbk_part_notes                START WITH  1 INCREMENT BY  1 CACHE 5;
DROP   SEQUENCE dbk_part_nutes;
CREATE SEQUENCE dbk_part_nutes                START WITH  1 INCREMENT BY  1 CACHE 5;
DROP   SEQUENCE dbk_part_nutez;
CREATE SEQUENCE dbk_part_nutez                START WITH  1 INCREMENT BY  1 CACHE 5;
DROP   SEQUENCE dbk_payment_batch;
CREATE SEQUENCE dbk_payment_batch             START WITH  1 INCREMENT BY  1 CACHE 5;
DROP   SEQUENCE dbk_payment_detail;
CREATE SEQUENCE dbk_payment_detail            START WITH  1 INCREMENT BY  1 CACHE 5;
DROP   SEQUENCE dbk_payment_notes;
CREATE SEQUENCE dbk_payment_notes             START WITH  1 INCREMENT BY  1 CACHE 5;
DROP   SEQUENCE dbk_payment_nutes;
CREATE SEQUENCE dbk_payment_nutes             START WITH  1 INCREMENT BY  1 CACHE 5;
DROP   SEQUENCE dbk_product$30;
CREATE SEQUENCE dbk_product$30                START WITH  1 INCREMENT BY  1 CACHE 5;
DROP   SEQUENCE dbk_product$32;
CREATE SEQUENCE dbk_product$32                START WITH  1 INCREMENT BY  1 CACHE 5;
DROP   SEQUENCE dbk_vendor$30;
CREATE SEQUENCE dbk_vendor$30                 START WITH  1 INCREMENT BY  1 CACHE 5;
DROP   SEQUENCE dbk_vendor$32;
CREATE SEQUENCE dbk_vendor$32                 START WITH  1 INCREMENT BY  1 CACHE 5;

--
-- Sequences for Positioning values for DMS Sets
--
DROP   SEQUENCE posxspec_remark;
CREATE SEQUENCE posxspec_remark               START WITH  1 INCREMENT BY  1 CACHE 8;
--
-- For Resequencing Position values for NEXT/PRIOR Sets
--
DROP   SEQUENCE posxitem;
CREATE SEQUENCE posxitem                      START WITH  1 INCREMENT BY  1 CACHE
10;
DROP   SEQUENCE posxbatch_detail;
CREATE SEQUENCE posxbatch_detail              START WITH  1 INCREMENT BY  1 CACHE
10;
DROP   SEQUENCE posxdetail_notes;
CREATE SEQUENCE posxdetail_notes              START WITH  1 INCREMENT BY  1 CACHE
10;
DROP   SEQUENCE posxpayment_notes;
CREATE SEQUENCE posxpayment_notes             START WITH  1 INCREMENT BY  1 CACHE
10;

--
-- Define referential constraints for Sets
--
ALTER TABLE cust_order$20 ADD (
      CONSTRAINT   set_ordor_20
      FOREIGN KEY (own_ordor)
      REFERENCES   customer (row_customer)
);

ALTER TABLE cust_order$22 ADD (
      CONSTRAINT   set_ordor_22
      FOREIGN KEY (own_ordor)
      REFERENCES   customer (row_customer)
);

ALTER TABLE payment_detail ADD (
      CONSTRAINT   set_batch_detail
      FOREIGN KEY (own_batch_detail)
      REFERENCES   payment_batch (row_payment_batch)
```

```
);

spool OFF
```

Data Mapping rules

The data-mapping file describes where every field of each DMS record is. For each DMS record field it defines the table name and column name where the data can be found. By parsing the SQL DDL, dbigen will know what indexes and constraints are available for facilitate reaching the data. Dbigen will optimize all paths to the data and then generate the run-time code to support the defined structures.

If you want a different SQL structure, then you are free to make changes or invent your own and then create the data-mapping rules that define how to get the data from the SQL database that the DMS records have defined.

The more general syntax for defining the data-mapping rules follows:

```
DMS SCHEMA IS "dms schema file name"
SQL SCHEMA IS "SQL DDL file name"
RECORD <dms-rec-name> [AREA <area-name>] {
 LOCATION INDEX n IS <index-name>;
 LOCATION CALC IS <index-name>;
 LOCATION DIRECT;
 DBKEY IS <table.column>;
 INDEX n IS <index-name>;
  <fld1> IS <table.column1>;
  <fldx> IS <constant>;
  IF <expr> THEN …fields… ELSE …fields… ;
  IF <expr> THEN …fields… ;
  <fldz> IS <function> ( <table.column>,
…params…);
  <fldn> IS <table.column>
}

…fields… could be one or more of:
  <fld1> IS <table.column1>;
  <fldx> IS <constant>;
  { …fields… }

SET <dms-set-name> {
 Reference <member.column> is <owner.column>
 ORDER BY <member.column1> [DESC|ASC];
}
```

If the COBOL data item is in an OCCURS, then the data mapping rules would indicate which occurrence is being referenced. For example:

```
05  MYTBL PIC X(5) OCCURS 3 TIMES.
```

Would be mapped as:

```
mytbl(1) IS myrec.mytbl01;
mytbl(2) IS myrec.mytbl02;
mytbl(3) IS myrec.mytbl03;
mytbl(4) IS myrec.mytbl04;
mytbl(5) IS myrec.mytbl05;
```

If the COBOL data items had been combined (aka. FOLDed) into one column then the data mapping rule may look like:

```
mytbl(1) IS myrec.mytbl[1:5];
mytbl(2) IS myrec.mytbl[6:5];
mytbl(3) IS myrec.mytbl[11:5];
mytbl(4) IS myrec.mytbl[16:5];
mytbl(5) IS myrec.mytbl[21:5];
```

The COBOL data item may be followed by subscripts inside parenthesis. When a SQL column holds more than one COBOL data item, the 1 relative position and length is defined inside square brackets.

For each DMS record, the data-mapping file describes each field and what table holds the data. It also describes which column holds the numeric DBKEY value for the record and which indexes represent the CALC key and DMS indexes.

For each DMS set, the column used to reference the owner of the set is listed, the column used for positioning members is listed and the ORDER BY to fetch the members is listed.

Data Mapping REDEFINES

For the example of REDEFINES on the previous pages as follows:

```
RECORD MYREC
LOCATION MODE xxxxxx
REDEFINES PART1
IF FIELD1 = "NUM" USE FILLER-1
WITHIN MYAREA.
05  THE-RECORD.
     10  FIELD1  PIC X(5).
     10  PART1   PIC X(10).
     10  FILLER-1 REDEFINES PART1.
         15  FIELD2   PIC S9(7)V99 COMP-3.
         15  FIELD3   PIC S9(7)V99 COMP-3.
```

The data mapping rules would look like the following:

```
Record MYREC {
...
 FIELD1 is myrec.field1;
 IF FIELD1 == "NUM" THEN {
   FIELD2 is myrec.field2;
   FIELD3 is myrec.field3;
 } ELSE {
   PART1 is myrec.part1;
 };
}
```

Sample Data Mapping rules

The following would be automatically generated by dbischema and is used by dbigen to generate the final definition of how TIP/dbi is to manage the database.

```
DMS Schema is "dms11";
SOURCE IS DMS2200;
SQL DDL IS "dms11.ddl";

RECORD CUST-ORDER$20 {
   AREA ORDER-AREA;
   CALC IS k0_calc_sr0620_20;
   DBKEY IS cust_order$20.row_cust_order
         VIA SEQUENCE dbk_cust_order$20;
   FO-NO-620 is cust_order$20.fo_no_620;
   CUST-PO-NO-620 is cust_order$20.cust_po_no_620;
   FILLER01 is cust_order$20.filler01;
   DATE-SHIP-620 is cust_order$20.date_ship_620;
   DATE-REQ-620 is cust_order$20.date_req_620;
   DATE-PROM-620 is cust_order$20.date_prom_620;
   ENTRY-CODE(1) is cust_order$20.entry_code01;
   BIN-NUM(1) is cust_order$20.bin_num01;
   PAC-NUM(1) is cust_order$20.pac_num01;
   ENTRY-CODE(2) is cust_order$20.entry_code02;
   BIN-NUM(2) is cust_order$20.bin_num02;
   PAC-NUM(2) is cust_order$20.pac_num02;
   ENTRY-CODE(3) is cust_order$20.entry_code03;
   BIN-NUM(3) is cust_order$20.bin_num03;
   PAC-NUM(3) is cust_order$20.pac_num03;
   ENTRY-CODE(4) is cust_order$20.entry_code04;
   BIN-NUM(4) is cust_order$20.bin_num04;
   PAC-NUM(4) is cust_order$20.pac_num04;
   ENTRY-CODE(5) is cust_order$20.entry_code05;
   BIN-NUM(5) is cust_order$20.bin_num05;
   PAC-NUM(5) is cust_order$20.pac_num05;
   BIN-NUM2(1) is cust_order$20.bin_num201;
   BIN-NUM2(2) is cust_order$20.bin_num202;
   BIN-NUM2(3) is cust_order$20.bin_num203;
   BIN-NUM2(4) is cust_order$20.bin_num204;
   BIN-NUM2(5) is cust_order$20.bin_num205;
   TEST-TBL3(1) is cust_order$20.test_tbl301;
   TEST-TBL3(2) is cust_order$20.test_tbl302;
   TEST-TBL3(3) is cust_order$20.test_tbl303;
   TEST-TBL3(4) is cust_order$20.test_tbl304;
   TEST-TBL3(5) is cust_order$20.test_tbl305;
   SMALL-BIN1 is cust_order$20.small_bin1;
   SMALL-BIN2 is cust_order$20.small_bin2;
   SMALL-BIN3 is cust_order$20.small_bin3;
   CODE-C(1) is cust_order$20.code_c01;
   CODE-D(1) [1:5] is cust_order$20.code_d01;
   CODE-C(2) is cust_order$20.code_c02;
```

```
   CODE-D(2) [1:5] is cust_order$20.code_d02;
   CODE-C(3) is cust_order$20.code_c03;
   CODE-D(3) [1:5] is cust_order$20.code_d03;
   CODE-C(4) is cust_order$20.code_c04;
   CODE-D(4) [1:5] is cust_order$20.code_d04;
   CODE-C(5) is cust_order$20.code_c05;
   CODE-D(5) [1:5] is cust_order$20.code_d05;
}

RECORD CUST-ORDER$22 {
   AREA ORDER-AREA2;
   CALC IS k0_calc_sr0620_22;
   DBKEY IS cust_order$22.row_cust_order
         VIA SEQUENCE dbk_cust_order$22;
   FO-NO-620 is cust_order$22.fo_no_620;
   CUST-PO-NO-620 is cust_order$22.cust_po_no_620;
   FILLER01 is cust_order$22.filler01;
   DATE-SHIP-620 is cust_order$22.date_ship_620;
   DATE-REQ-620 is cust_order$22.date_req_620;
   DATE-PROM-620 is cust_order$22.date_prom_620;
   ENTRY-CODE(1) is cust_order$22.entry_code01;
   BIN-NUM(1) is cust_order$22.bin_num01;
   PAC-NUM(1) is cust_order$22.pac_num01;
   ENTRY-CODE(2) is cust_order$22.entry_code02;
   BIN-NUM(2) is cust_order$22.bin_num02;
   PAC-NUM(2) is cust_order$22.pac_num02;
   ENTRY-CODE(3) is cust_order$22.entry_code03;
   BIN-NUM(3) is cust_order$22.bin_num03;
   PAC-NUM(3) is cust_order$22.pac_num03;
   ENTRY-CODE(4) is cust_order$22.entry_code04;
   BIN-NUM(4) is cust_order$22.bin_num04;
   PAC-NUM(4) is cust_order$22.pac_num04;
   ENTRY-CODE(5) is cust_order$22.entry_code05;
   BIN-NUM(5) is cust_order$22.bin_num05;
   PAC-NUM(5) is cust_order$22.pac_num05;
   BIN-NUM2(1) is cust_order$22.bin_num201;
   BIN-NUM2(2) is cust_order$22.bin_num202;
   BIN-NUM2(3) is cust_order$22.bin_num203;
   BIN-NUM2(4) is cust_order$22.bin_num204;
   BIN-NUM2(5) is cust_order$22.bin_num205;
   TEST-TBL3(1) is cust_order$22.test_tbl301;
   TEST-TBL3(2) is cust_order$22.test_tbl302;
   TEST-TBL3(3) is cust_order$22.test_tbl303;
   TEST-TBL3(4) is cust_order$22.test_tbl304;
   TEST-TBL3(5) is cust_order$22.test_tbl305;
   SMALL-BIN1 is cust_order$22.small_bin1;
   SMALL-BIN2 is cust_order$22.small_bin2;
   SMALL-BIN3 is cust_order$22.small_bin3;
   CODE-C(1) is cust_order$22.code_c01;
   CODE-D(1) [1:5] is cust_order$22.code_d01;
   CODE-C(2) is cust_order$22.code_c02;
   CODE-D(2) [1:5] is cust_order$22.code_d02;
   CODE-C(3) is cust_order$22.code_c03;
   CODE-D(3) [1:5] is cust_order$22.code_d03;
   CODE-C(4) is cust_order$22.code_c04;
   CODE-D(4) [1:5] is cust_order$22.code_d04;
   CODE-C(5) is cust_order$22.code_c05;
   CODE-D(5) [1:5] is cust_order$22.code_d05;
}

RECORD CUSTOMER {
   AREA CUSTOMER-AREA;
   CALC IS k0_customer;
   INDEX 1 IS k1_customer;
   DBKEY IS customer.row_customer
         VIA SEQUENCE dbk_customer;
   CUST-NO-611 is customer.cust_no_611;
   CUST-NAME-S-611 is customer.cust_name_s_611;
   CUST-ADDR-S-611 is customer.cust_addr_s_611;
   CUST-CREDIT is customer.cust_credit;
   CUST-BIG-611 is customer.cust_big_611;
```

```
   TEXT-611 is customer.text_611;
}

RECORD ORD-REMARK {
   AREA ORDER-AREA;
   DBKEY IS ord_remark.row_ord_remark
         VIA SEQUENCE dbk_ord_remark;
   REMARK-CD-622 is ord_remark.cd;
   REMARK-SEQ-622 is ord_remark.seq;
   REMARK-622 is ord_remark.remark;
}

RECORD ORDER-ITEM$20 {
   AREA ORDER-AREA;
   DBKEY IS order_item$20.row_order_item
         VIA SEQUENCE dbk_order_item$20;
   PROD-NO-621 is order_item$20.prod_no_621;
   FILLER01 is order_item$20.filler01;
   LOT-NO-621 is order_item$20.lot_no_621;
   FILLER02 is order_item$20.filler02;
   QTY-ORD-621 is order_item$20.qty_ord_621;
   QTY-SHIP-621 is order_item$20.qty_ship_621;
   QTY-BIG-621 is order_item$20.qty_big_621;
   QTY-SEP-621 is order_item$20.qty_sep_621;
   QTY-TRL-621 is order_item$20.qty_trl_621;
}

RECORD ORDER-ITEM$22 {
   AREA ORDER-AREA2;
   DBKEY IS order_item$22.row_order_item
         VIA SEQUENCE dbk_order_item$22;
   PROD-NO-621 is order_item$22.prod_no_621;
   FILLER01 is order_item$22.filler01;
   LOT-NO-621 is order_item$22.lot_no_621;
   FILLER02 is order_item$22.filler02;
   QTY-ORD-621 is order_item$22.qty_ord_621;
   QTY-SHIP-621 is order_item$22.qty_ship_621;
   QTY-BIG-621 is order_item$22.qty_big_621;
   QTY-SEP-621 is order_item$22.qty_sep_621;
   QTY-TRL-621 is order_item$22.qty_trl_621;
}

RECORD PART-DETAIL {
   AREA PAYMENTS-AREA;
   CALC IS k0_part_detail;
   DBKEY IS part_detail.row_part_detail
         VIA SEQUENCE dbk_part_detail;
   PD-CUST-NO-805 is part_detail.cust_no_805;
   PD-CUST-PO-NO-805 is part_detail.cust_po_no_805;
   PD-PART-TYPE is part_detail.part_type;
   IF PD-PART-TYPE == "A" THEN {
      PD-PART-AMT-805 is part_detail.part_amt_805;
   } ELSe {
      PD-PART-TEMP is part_detail.part_temp;
   };
}

RECORD PART-NOTES {
   AREA PAYMENTS-AREA;
   DBKEY IS part_notes.row_part_notes
         VIA SEQUENCE dbk_part_notes;
   PN-SORT-810 is part_notes.pn_sort_810;
   PN-COMMENT-810 is part_notes.pn_comment_810;
}

RECORD PART-NUTES {
   AREA PAYMENTS-AREA;
   DBKEY IS part_nutes.row_part_nutes
         VIA SEQUENCE dbk_part_nutes;
   PN-SORT-815 is part_nutes.pn_sort_815;
   PN-COMMENT-815 is part_nutes.pn_comment_815;
```

```
}

RECORD PART-NUTEZ {
   AREA PAYMENTS-AREA;
   DBKEY IS part_nutez.row_part_nutez
         VIA SEQUENCE dbk_part_nutez;
   PN-SORT-817 is part_nutez.pn_sort_817;
   PN-COMMENT-817 is part_nutez.pn_comment_817;
}

RECORD PAYMENT-BATCH {
   AREA PAYMENTS-AREA;
   DBKEY IS payment_batch.row_payment_batch
         VIA SEQUENCE dbk_payment_batch;
   PB-BATCH-NUM-700 is payment_batch.batch_num;
   PB-CLERK-NUM-700 is payment_batch.clerk_num;
   PB-BATCH-TOTAL-700 is payment_batch.batch_total;
}

RECORD PAYMENT-DETAIL {
   AREA PAYMENTS-AREA;
   CALC IS k0_payment_detail;
   DBKEY IS payment_detail.row_payment_detail
         VIA SEQUENCE dbk_payment_detail;
   PD-CUST-NO-705 is payment_detail.cust_no;
   PD-CUST-PO-NO-705 is payment_detail.cust_po_no;
   PD-PAYMENT-AMT-705 is payment_detail.payment_amt;
}

RECORD PAYMENT-NOTES {
   AREA PAYMENTS-AREA;
   DBKEY IS payment_notes.row_payment_notes
         VIA SEQUENCE dbk_payment_notes;
   PN-COMMENT-710 is payment_notes.pn_comment_710;
}

RECORD PAYMENT-NUTES {
   AREA PAYMENTS-AREA;
   DBKEY IS payment_nutes.row_payment_nutes
         VIA SEQUENCE dbk_payment_nutes;
   PN-COMMENT-715 is payment_nutes.pn_comment_715;
}

RECORD PRODUCT$30 {
   AREA PRODUCT-AREA;
   CALC IS k0_calc_sr0631_30;
   INDEX 1 IS k0_calc_sr0631_30;
   DBKEY IS product$30.row_product
         VIA SEQUENCE dbk_product$30;
   PROD-NO-631 is product$30.prod_no_631;
   FILLER01 is product$30.filler01;
   PROD-DES-INT-631 is product$30.prod_des_int_631;
   PROD-DES-EXT-631 is product$30.prod_des_ext_631;
}

RECORD PRODUCT$32 {
   AREA PRODUCT-AREA2;
   CALC IS k0_calc_sr0631_32;
   INDEX 1 IS k0_calc_sr0631_32;
   DBKEY IS product$32.row_product
         VIA SEQUENCE dbk_product$32;
   PROD-NO-631 is product$32.prod_no_631;
   FILLER01 is product$32.filler01;
   PROD-DES-INT-631 is product$32.prod_des_int_631;
   PROD-DES-EXT-631 is product$32.prod_des_ext_631;
}

RECORD VENDOR$30 {
   AREA PRODUCT-AREA;
   INDEX 1 IS k1_index_01_0651_30;
   DBKEY IS vendor$30.row_vendor
```

```
          VIA SEQUENCE dbk_vendor$30;
    VENDOR-NAME-651 is vendor$30.name_651;
    VENDOR-ID is vendor$30.id;
    VENDOR-ADDR is vendor$30.addr;
}

RECORD VENDOR$32 {
    AREA PRODUCT-AREA2;
    INDEX 1 IS k1_index_01_0651_32;
    DBKEY IS vendor$32.row_vendor
          VIA SEQUENCE dbk_vendor$32;
    VENDOR-NAME-651 is vendor$32.name_651;
    VENDOR-ID is vendor$32.id;
    VENDOR-ADDR is vendor$32.addr;
}

# SET ORDOR  AUTOMATIC  SORTED    2 members
SET ORDOR {
  REFERENCE ordor.own_ordor IS customer.row_customer;
  ORDER BY
    ordor.own_ordor,
    ordor.pos_ordor,
    ordor.member_id;

  WHEN   ordor.member_id IS 200620
  MEMBER CUST-ORDER$20 {
      ordor.pos_ordor IS cust_order$20.fo_no_620
      ordor.member_row IS cust_order$20.row_cust_order;
      ordor.own_ordor IS cust_order$20.own_ordor;
  }

  WHEN   ordor.member_id IS 220620
  MEMBER CUST-ORDER$22 {
      ordor.pos_ordor IS cust_order$22.fo_no_620
      ordor.member_row IS cust_order$22.row_cust_order;
      ordor.own_ordor IS cust_order$22.own_ordor;
  }
}

# SET ITEM  AUTOMATIC  ORDER NEXT     2 members
SET ITEM {
  POSITION IS item.pos_item
        VIA SEQUENCE posxitem;

  WHEN  item.aid_item IS 200620
  OWNER CUST-ORDER OF AREA ORDER-AREA
    REFERENCE TO cust_order$20.row_cust_order;
  WHEN  item.aid_item IS 220620
  OWNER CUST-ORDER OF AREA ORDER-AREA2
    REFERENCE TO cust_order$22.row_cust_order;

  ORDER BY
    item.aid_item,
    item.own_item,
    item.pos_item;

  WHEN   item.member_id IS 200621
  MEMBER ORDER-ITEM$20
  POSITION IS order_item$20.pos_item
  REFERENCE FROM order_item$20.own_item {
      item.member_row IS order_item$20.row_order_item;
      item.aid_item IS order_item$20.aid_item;
      item.own_item IS order_item$20.own_item;
  }

  WHEN   item.member_id IS 220621
  MEMBER ORDER-ITEM$22
  POSITION IS order_item$22.pos_item
  REFERENCE FROM order_item$22.own_item {
      item.member_row IS order_item$22.row_order_item;
      item.aid_item IS order_item$22.aid_item;
```

```
      item.own_item IS order_item$22.own_item;
  }
}

# SET QTY  AUTOMATIC  SORTED    2 members
SET QTY {

  WHEN  qty.aid_qty IS 200620
  OWNER CUST-ORDER OF AREA ORDER-AREA
    REFERENCE TO cust_order$20.row_cust_order;
  WHEN  qty.aid_qty IS 220620
  OWNER CUST-ORDER OF AREA ORDER-AREA2
    REFERENCE TO cust_order$22.row_cust_order;

  ORDER BY
    qty.aid_qty,
    qty.own_qty,
    qty.pos_qty,
    qty.pos_qty1,
    qty.member_id,
    qty.member_row;

  WHEN   qty.member_id IS 200621
  MEMBER ORDER-ITEM$20
  REFERENCE FROM order_item$20.own_qty {
      qty.pos_qty IS order_item$20.qty_ord_621
      qty.pos_qty1 IS order_item$20.qty_ship_621
      qty.member_row IS order_item$20.row_order_item;
      qty.aid_qty IS order_item$20.aid_qty;
      qty.own_qty IS order_item$20.own_qty;
  }

  WHEN   qty.member_id IS 220621
  MEMBER ORDER-ITEM$22
  REFERENCE FROM order_item$22.own_qty {
      qty.pos_qty IS order_item$22.qty_ord_621
      qty.pos_qty1 IS order_item$22.qty_ship_621
      qty.member_row IS order_item$22.row_order_item;
      qty.aid_qty IS order_item$22.aid_qty;
      qty.own_qty IS order_item$22.own_qty;
  }
}

# SET QTY-SHIP  AUTOMATIC  SORTED    2 members
SET QTY-SHIP {

  WHEN  qty_ship.aid_qty_ship IS 200620
  OWNER CUST-ORDER OF AREA ORDER-AREA
    REFERENCE TO cust_order$20.row_cust_order;
  WHEN  qty_ship.aid_qty_ship IS 220620
  OWNER CUST-ORDER OF AREA ORDER-AREA2
    REFERENCE TO cust_order$22.row_cust_order;

  ORDER BY
    qty_ship.aid_qty_ship,
    qty_ship.own_qty_ship,
    qty_ship.pos_qty_ship,
    qty_ship.member_id,
    qty_ship.member_row;

  WHEN   qty_ship.member_id IS 200621
  MEMBER ORDER-ITEM$20
  REFERENCE FROM order_item$20.own_qty_ship {
      qty_ship.pos_qty_ship IS order_item$20.qty_ship_621
      qty_ship.member_row IS order_item$20.row_order_item;
      qty_ship.aid_qty_ship IS order_item$20.aid_qty_ship;
      qty_ship.own_qty_ship IS order_item$20.own_qty_ship;
  }

  WHEN   qty_ship.member_id IS 220621
  MEMBER ORDER-ITEM$22
```

```
  REFERENCE FROM order_item$22.own_qty_ship {
      qty_ship.pos_qty_ship IS order_item$22.qty_ship_621
      qty_ship.member_row IS order_item$22.row_order_item;
      qty_ship.aid_qty_ship IS order_item$22.aid_qty_ship;
      qty_ship.own_qty_ship IS order_item$22.own_qty_ship;
  }
}

# SET SPEC-REMARK  MANUAL     ORDER LAST
SET SPEC-REMARK {
  POSITION IS ord_remark.pos_spec_remark
        VIA SEQUENCE posxspec_remark;

  WHEN  ord_remark.aid_spec_remark IS 200620
  OWNER CUST-ORDER OF AREA ORDER-AREA
    REFERENCE ord_remark.own_spec_remark IS cust_order$20.row_cust_order;
  WHEN  ord_remark.aid_spec_remark IS 220620
  OWNER CUST-ORDER OF AREA ORDER-AREA2
    REFERENCE ord_remark.own_spec_remark IS cust_order$22.row_cust_order;

  ORDER BY
           ord_remark.aid_spec_remark,
           ord_remark.own_spec_remark,
           ord_remark.pos_spec_remark;
}

# SET PROD-ORD  AUTOMATIC  SORTED    2 members
SET PROD-ORD {

  WHEN  prod_ord.aid_prod_ord IS 300631
  OWNER PRODUCT OF AREA PRODUCT-AREA
    REFERENCE TO product$30.row_product;
  WHEN  prod_ord.aid_prod_ord IS 320631
  OWNER PRODUCT OF AREA PRODUCT-AREA2
    REFERENCE TO product$32.row_product;

  ORDER BY
    prod_ord.aid_prod_ord,
    prod_ord.own_prod_ord,
    prod_ord.pos_prod_ord,
    prod_ord.member_id;

  WHEN   prod_ord.member_id IS 200621
  MEMBER ORDER-ITEM$20
  REFERENCE FROM order_item$20.own_prod_ord {
      prod_ord.pos_prod_ord IS order_item$20.lot_no_621
      prod_ord.member_row IS order_item$20.row_order_item;
      prod_ord.aid_prod_ord IS order_item$20.aid_prod_ord;
      prod_ord.own_prod_ord IS order_item$20.own_prod_ord;
  }

  WHEN   prod_ord.member_id IS 220621
  MEMBER ORDER-ITEM$22
  REFERENCE FROM order_item$22.own_prod_ord {
      prod_ord.pos_prod_ord IS order_item$22.lot_no_621
      prod_ord.member_row IS order_item$22.row_order_item;
      prod_ord.aid_prod_ord IS order_item$22.aid_prod_ord;
      prod_ord.own_prod_ord IS order_item$22.own_prod_ord;
  }
}

# SET VEND-PROD  AUTOMATIC  SORTED    2 members
SET VEND-PROD {

  WHEN  vend_prod.aid_vend_prod IS 300651
  OWNER VENDOR OF AREA PRODUCT-AREA
    REFERENCE TO vendor$30.row_vendor;
  WHEN  vend_prod.aid_vend_prod IS 320651
  OWNER VENDOR OF AREA PRODUCT-AREA2
    REFERENCE TO vendor$32.row_vendor;
```

```
  ORDER BY
    vend_prod.aid_vend_prod,
    vend_prod.own_vend_prod,
    vend_prod.pos_vend_prod,
    vend_prod.member_id;

  WHEN   vend_prod.member_id IS 300631
  MEMBER PRODUCT$30
  REFERENCE FROM product$30.own_vend_prod {
      vend_prod.pos_vend_prod IS product$30.prod_no_631
      vend_prod.member_row IS product$30.row_product;
      vend_prod.aid_vend_prod IS product$30.aid_vend_prod;
      vend_prod.own_vend_prod IS product$30.own_vend_prod;
  }

  WHEN   vend_prod.member_id IS 320631
  MEMBER PRODUCT$32
  REFERENCE FROM product$32.own_vend_prod {
      vend_prod.pos_vend_prod IS product$32.prod_no_631
      vend_prod.member_row IS product$32.row_product;
      vend_prod.aid_vend_prod IS product$32.aid_vend_prod;
      vend_prod.own_vend_prod IS product$32.own_vend_prod;
  }
}

# SET BATCH-DETAIL  AUTOMATIC  ORDER PRIOR
SET BATCH-DETAIL {
  POSITION IS payment_detail.pos_batch_detail
        VIA SEQUENCE posxbatch_detail;
  REFERENCE payment_detail.own_batch_detail IS payment_batch.row_payment_batch;
  ORDER BY
           payment_detail.own_batch_detail,
           payment_detail.pos_batch_detail;
}

# SET DETAIL-NOTES  MANUAL     ORDER PRIOR    2 members
SET DETAIL-NOTES {
  POSITION IS detail_notes.pos_detail_notes
        VIA SEQUENCE posxdetail_notes;
  REFERENCE detail_notes.own_detail_notes IS payment_detail.row_payment_detail;
  ORDER BY
    detail_notes.own_detail_notes,
    detail_notes.pos_detail_notes;

  WHEN   detail_notes.member_id IS 700710
  MEMBER PAYMENT-NOTES
  POSITION IS payment_notes.pos_detail_notes {
      detail_notes.member_row IS payment_notes.row_payment_notes;
      detail_notes.own_detail_notes IS payment_notes.own_detail_notes;
  }

  WHEN   detail_notes.member_id IS 700715
  MEMBER PAYMENT-NUTES
  POSITION IS payment_nutes.pos_detail_notes {
      detail_notes.member_row IS payment_nutes.row_payment_nutes;
      detail_notes.own_detail_notes IS payment_nutes.own_detail_notes;
  }
}

# SET PAYMENT-NOTES  MANUAL     ORDER PRIOR    2 members
SET PAYMENT-NOTES {
  POSITION IS S_payment_notes.pos_payment_notes
        VIA SEQUENCE posxpayment_notes;
  REFERENCE S_payment_notes.own_payment_notes IS payment_detail.row_payment_detail;
  ORDER BY
    S_payment_notes.own_payment_notes,
    S_payment_notes.pos_payment_notes;

  WHEN   S_payment_notes.member_id IS 700710
  MEMBER PAYMENT-NOTES
  POSITION IS payment_notes.pos_payment_notes {
```

```
        S_payment_notes.member_row IS payment_notes.row_payment_notes;
        S_payment_notes.own_payment_notes IS payment_notes.own_payment_notes;
    }

    WHEN   S_payment_notes.member_id IS 700715
    MEMBER PAYMENT-NUTES
    POSITION IS payment_nutes.pos_payment_notes {
        S_payment_notes.member_row IS payment_nutes.row_payment_nutes;
        S_payment_notes.own_payment_notes IS payment_nutes.own_payment_notes;
    }
}

# SET PARTS-NOTES  MANUAL     SORTED    3 members
SET PARTS-NOTES {
    REFERENCE parts_notes.own_parts_notes IS part_detail.row_part_detail;
    ORDER BY
      parts_notes.own_parts_notes,
      parts_notes.pos_parts_notes,
      parts_notes.member_id;

    WHEN   parts_notes.member_id IS 700810
    MEMBER PART-NOTES {
        parts_notes.pos_parts_notes IS part_notes.pn_sort_810
        parts_notes.member_row IS part_notes.row_part_notes;
        parts_notes.own_parts_notes IS part_notes.own_parts_notes;
    }

    WHEN   parts_notes.member_id IS 700815
    MEMBER PART-NUTES {
        parts_notes.pos_parts_notes IS part_nutes.pn_sort_815
        parts_notes.member_row IS part_nutes.row_part_nutes;
        parts_notes.own_parts_notes IS part_nutes.own_parts_notes;
    }

    WHEN   parts_notes.member_id IS 700817
    MEMBER PART-NUTEZ {
        parts_notes.pos_parts_notes IS part_nutez.pn_sort_817
        parts_notes.member_row IS part_nutez.row_part_nutez;
        parts_notes.own_parts_notes IS part_nutez.own_parts_notes;
    }
}
```

Upgrading from old TIP/dbi

This newer version of TIP/dbi has been in development since 2001 and it has several advantages over the first version of TIP/dbi.

- ✓ TIP/dbi II runs much faster than the older version due to better database search optimization and internal caching of SELECT statements.
- ✓ TIP/dbi II uses the Oracle Call Interface which performs faster than the the embedded SQL used by the older TIP/dbi.
- ✓ TIP/dbi II supports Oracle 8 up thru Oracle 10g.
- ✓ TIP/dbi II supports data mapping. This gives you significantly more control over how your COBOL records are mapped to relational tables.
- ✓ TIP/dbi II supports conditional REDEFINES. This allows each of the redefined data fields to be mapped to separate columns, making the data visible to tools outside of TIP/dbi. The old TIP/dbi would have overlapped the data items and/or encoded the data as RAW (hexadecimal) which makes the data unusable outside of TIP/dbi.

If you want better performance or you want to be able to better use your data with software tools outside of TIP/dbi, then you should upgrade to TIP/dbi II.

The old TIP/dbi creates a series of files and data dictionary in a directory called **`$TIPDMS/schema.dd`** while TIP/dbi II uses a directory called **`$TIPDMS/schema.dd3`** .

Unload/Reload

If you have been using RAW or FOLDing the data with your current version of TIP/dbi then you would want to unload your current database to sequential files and then reload the data into a new database structure using TIP/dbi II.

The `dbiunload` utility discussed in previous section can be used to generate some COBOL/DMS programs that can be compiled on Unix using the older TIP/dbi system and then executed to unload the data into a series of sequential files. This same method is used to unload data from the mainframe DMS database. Once the data has been unloaded, then you can reload this data into your new Tip/dbi II managed database using the `dbireload` utility.

Be sure to recompile all of you application using the TIP/dbi II utilities (`dbipre`).

Keep existing database

If you have not been using RAW or FOLD and you have no needs to change the structure of the relational database then you could have TIP/dbi II use the exact same tables and data as the old TIP/dbi.

Each DMS record needs to have a unique database key. The older Tip/dbi generated unique numbers using values from a special table called *schema*_next_unq which TIP/dbi would read and write each time it needs another database key. TIP/dbi II uses an Oracle SEQUENCE to generate unique database keys. The SEQUENCE is much faster than updating a separate Oracle table. But to use your existing database tables, the correct values must be transferred to the corresponding SEQUENCE.

To keep your existing table structure run dbischema with the -o option or add the clause KEEP DEFAULT to the schema header section before you run dbischema.

DBIUPGRADE

The utility dbiupgrade can be used to cross-check that your existing table structure matches what TIP/dbi II expects. It does this by reading the old *schema*.ddl from $TIPDMS/*schema*.dd and comparing that to the new structure defined in $TIPDMS/*schema*.dd3/*schema*.ddl.

The dbiupgrade usage is:

```
TIP/dbi II Convert from old TIP/dbi;  Version c,v 1.3 2005/12
  © 1991-2005 Inglenet Business Solutions
  dbiupgrade -s schema -d oldschema.ddl
 Where the options are:
  -s    schema name as compiled by dbischema (TIP/dbi II)
  -d    SQL DDL file generated by old TIP/dbi
        Default: located in parellel directory
```

When run the utility reads the values from *schema*_next_unq and writes out a file $TIPDMS/*schema*.dd3/*schema*.upgrade which can be processed by sqlplus to create the required SEQUENCE with the correct values. For example:

```
DROP SEQUENCE dbk_budget;
CREATE SEQUENCE dbk_budget START WITH 1 INCREMENT BY 1 CACHE 5;
DROP SEQUENCE dbk_cust_order;
CREATE SEQUENCE dbk_cust_order START WITH 281 INCREMENT BY 1 CACHE 5;
DROP SEQUENCE dbk_customer;
CREATE SEQUENCE dbk_customer START WITH 1041 INCREMENT BY 1 CACHE 5;
DROP SEQUENCE dbk_ind_m;
CREATE SEQUENCE dbk_ind_m START WITH 1 INCREMENT BY 1 CACHE 5;
```

You must stop using the old TIP/dbi, then run dbiupgrade, then use sqlplus to create the SEQUENCE, then recompile your applications with TIP/dbi II and then you can start using TIP/dbi II.

TIP/dbi database maintenance

Updating the Schema definition

The supplied program **dbidiff.pc** provides functionality that will allow you to make modifications to DMS database schema record definitions by adding new fields to the end of the record, which are not part of any SET relationship. This also requires using the option REMOVE FILLER, which is the default.

dbidiff can then be used to compute alterations that need to be done to an existing Oracle database structure for it to match the new DMS schema without requiring the existing database to be unloaded and reloaded. This is a significant operational advantage and will work for the majority of changes that you might want to make to the DMS database structure. However some major changes such as splitting a record in two and establishing a SET relationship between the new records might require a manual database unload and reload using special written COBOL/DMS programs.

There is a utility called **dbidiff.pc** which will be supplied in source code format. This program must be pre-processed with Oracle **proc** and compiled. Once compiled it can be used to compute the difference between ***schema*.ddl** and what is actually defined in the Oracle data dictionary. `dbidiff` takes the following command line options:

Option	Description
-U userid	Oracle User Id to connect to Oracle with
-P pass	Oracle password to connect to Oracle with

Following the options is the Oracle ***schema*.ddl** file name to be processed. The ***schema*.ddl** file is parsed and a ***schema*.alter** file is produced which can be used to update the Oracle table definitions to match ***schema*.ddl.**

```
dbidiff -U user -P passwd prodschm.ddl
```

To make the changes inside the Oracle database you would need to run:

```
sqlplus user/passwd <prodschm.alter
```

Where `user` is the Oracle user-id for the Oracle schema holding the database and `passwd` is the corresponding Oracle password for that user.

Example Schema Change Procedures

This example show using some scripts editmod/postmod for managing your code via CVS/RCS. If you are not using source control management or yoru are not using these scripts or something like them, just ignore these steps in the following examples.

By and large the procedures for making changes to the schema or sub-schema follow similar steps to the methodology employed in the DMS-2200 environment, but obviously there will be several operational differences.

If the change to the schema involves a key field or is part of a set relationship or index, then usually an unload/reload sequence would be required as part of the change.

In the TIP/dbi environment, “skeleton” unload/reload programs are built with the **dbiunload** utility.

However, if the change entails something no more complex than the addition of a field to the end of a record or a simple change to the size of the field, then the unload/reload steps should not be required in the TIP/dbi environment.

It would not likely be possible to cover each and every possible change scenario that may be required, but we will attempt to outline the steps required to make the most common types of changes that may be encountered.

To add a new data field to the end of a record, the following steps would be followed:

All changes should be completed in a “test” environment with a backup of the affected database tables taken first, and obviously with TIP/ix down (or at least the application in question disabled) and no other processing of any kind occurring against the subject database during the entire time that changes are being made.

```
tipctl s (or use the site script to disable the application in question)
cd $TIPSITE/schema
editmod schema-name
using “vi” or the editor of choice, make the required change to the schema
dbischema schema-name
dbisubschema sub-schema-name (for each sub-schema as required)
cd $TIPDMS/schemaname.dd3
dbidiff -U user -P passwd schemaname.ddl
sqlplus user/passwd <schemaname.alter
cd $TIPSITE/schema
postmod schema-name
make and test needed application code changes
```

This will create a new *schemaname.ddl* and dictionary.exp file in the $TIPDMS/*schemaname*.dd3 directory.

If you have a separate production system and after all of you changes have been tested you will want to move them to production. You will need to copy over the contents of the schemaname.dd3 directory, run the schemaname.alter SQL script on the production system and copy over all of the changed application programs as well.

From that new "ddl" file, you could use an editor to extract the new field definition and convert it into an Oracle "alter table" command such as:

```
alter table recordname add column fieldname
```

The TIP/dbi utility called **dbidff** may also be used after the schema has been processed by **dbischema** in order for it to display the differences, and even generate the "alter" commands. (where `userid` is the Oracle userid for the schema and `password` is the Oracle password.)

```
cd $TIPDMS/schemaname.dd3
dbidiff –U userid –P password schemaname.ddl
```

You must use `sqlplus` to run the commands to actually make the change to the Oracle database but it is a good idea to review what `dbidiff` created to verify that it all looks correct.

(The example below shows "manual" starting of `sqlplus`, where `userid` is the Oracle userid for the schema and `password` is the Oracle password.)

```
cd $TIPDMS/schemaname.dd3
sqlplus userid/password < schemaname.alter
```

For example, let us add a new field to the EMISSIONS record:

```
 RECORD NAME IS EMISSIONS
     RECORD CODE IS 157
     LOCATION MODE IS DIRECT EMISSIONS-KEY STATION-AREA
     WITHIN MVIP-STATION
     RECORD MODE IS ASCII

            03  ST-EMI-FILLER                PIC X(4)

JDSJDS    03  JDS-NEW-FIELD             PIC X(5)


</home/davids/vsp/schema> dbischema mvip-schema.sch
TIP/dbi II Schema Compiler;   Version  1.123 2014/08/17
        © 1991-2014 Inglenet Business Solutions

 Oracle9 version: 9.2.0.1.0
SUN Default: NO SEQUENTIAL FIELDS BY RECORD
Reading `mvip-schema.sch'
Removing old /export/spare/home2/projects/vsp/dbidd/mvip-schema.dd3/mvip-
schema.sym
Default TABLESPACE is:      MVIP_DATA_TS1
Default INDEXSPACE is:      MVIP_INDEX_TS1
```

```
Reading Data Base Data Names at line 11
Reading AREAs starting at line: 42

If the last field of a record is FILLER, then it is not part of the SQL
schema
Reading RECORDs starting at line: 246
INCLUDE ST-LOCATION
Reading SETs starting at line: 1144
Finished parsing 1321 lines
Generating with Oracle 9 limits: Max CHAR 1995, Max Columns 995
TRIM FIELD NAMES; Keeping '-'
 LOW-VALUES ARE NULL
Note: Treating XREF-CT-STKR-TBL size 15 as one column
Note: Treating IS-TRN-INSRT-TBL size 8 as one column
Note: Treating IS-TRN-STKR-ISS-TBL size 18 as one column
Note: Treating XREF-MC-STKR-TBL size 15 as one column
Note: Treating CN-SRJ-EXP-TBL size 11 as one column
Note: Treating XREF-SR-STKR-TBL size 14 as one column
Note: Treating ST-GEN-ISSUE size 6 as one column
Note: Treating XREF-TM-STKR-TBL size 16 as one column
Note: Treating TRAN-KEY-CD size 6 as one column
Note: Treating ST-INV-QTR-TBL-MC size 17 as one column
Create Database script already exists
Create Tablespace script already exists
Create User script already exists
Writing SQL Schema definition: `/home2/projects/vsp/dbidd/mvip-
schema.dd3/mvip-schema.ddl'
Most columns in any table was 302
Most indexes on any table was 3
53 records defined
Total 1321 lines in DMS2200 schema

TIP/dbi Data Mapping Compiler;  Version 1.173 2014/09/26
  © 1991-2014 Inglenet Business Solutions
    DBI Generator schema: mvip-schema
   Schema dictionary: /home2/projects/vsp/dbidd/mvip-schema.dd3/mvip-schema.sym
   SQL DDL definition: /home2/projects/vsp/dbidd/mvip-schema.dd3/mvip-schema.ddl
      Data mapping rules: /home2/projects/vsp/dbidd/mvip-schema.dd3/mvip-schema.map
    Generating output to: /home2/projects/vsp/dbidd/mvip-schema.dd3
08:13:45.07  >>> Parsing SQL DDL <<<
08:13:45.11  >>> Opened DMS schema dictionary and parsed SQL DDL <<<
08:13:45.35  Finished parsing data mapping rules, now generating TIP/dbi run-
time
08:13:45.35   >>> Maximum sequence value is 4,294,967,295 <<<
08:13:45.52   >>> Now writing dictionary <<<

08:13:45.64  TIP/dbi generation completed

</home/davids/vsp/schema> cd $TIPDMS/mvip-schema.dd3
</home/davids/vsp/ddidd/mvip-schema.dd3> view mvip-schema.ddl

--
-- Table   8, Rcsz=   9, SR157 : EMISSIONS : 3 SQL columns
--           Estimated Max Row Size   20
--
DROP TABLE  emissions CASCADE CONSTRAINTS;
CREATE TABLE emissions (
        st_emi_filler                CHAR(4),
        jds_new_field               CHAR(5),
        row_emissions                INTEGER NOT NULL,
        CONSTRAINT pk_emissions PRIMARY KEY(row_emissions)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE MVIP_INDEX_TS1
       )
                STORAGE (FREELISTS 3 )
```

```
        TABLESPACE MVIP_DATA_TS1;
```

The **dbidiff** utility will compare the newly created "*schemaname*.ddl" structure to the existing database layout in Oracle showing the differences and producing an SQL script to make the changes to the database in Oracle.

```
</home/davids/vsp/ddidd/mvip-schema.dd3> dbidiff
TIP/ix DMS to Oracle Schema difference generator (Ver: 1.10 2010/09/09)
   © 1991-2010 Inglenet Business Solutions
Oracle User/Password must be supplied with -U

  Compute the difference between a generated DDL and
     the currently active DDL in Oracle
  This will work for changing current data fields or
     the addition of non-key fields

Command usage:

dbidiff  [options] schema.ddl
Where:
   -v             'verbose', List table sizes
   -U user        to be used to Connect to Oracle
   -P password    to be used to Connect to Oracle

Environment variables:
 ORACLE_UID       Oracle DBA user id
 ORACLE_PWD       Oracle DBA password


</home/davids/vsp/ddidd/mvip-schema.dd3> dbidiff -U * -P * mvip-schema.ddl
TIP/ix DMS to Oracle Schema difference generator (Ver: 1.10 2010/09/09)
   © 1991-2010 Inglenet Business Solutions
Alter Schema written out to mvip-schema.alter

</home/davids/vsp/ddidd/mvip-schema.dd3> cat mvip-schema.alter
spool alter
ALTER TABLE emissions
     ADD    (jds_new_field                  CHAR(5))
;
spool OFF
exit;

</home/davids/vsp/ddidd/mvip-schema.dd3> sqlplus User/Password  <*mvip-
schema.alter

SQL*Plus: Release 8.1.5.0.0 - Production on Thu Feb 19 16:13:20 2004

(c) Copyright 1999 Oracle Corporation.  All rights reserved.


Connected to:
Oracle8i Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production

SQL> SQL>   2    3
Table altered.
```

```
SQL> SQL> Disconnected from Oracle8i Enterprise Edition Release 8.1.5.0.0 -
Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production
</home/davids/vsp/ddidd/mvip-schema.dd3> sqlplus User/Password


SQL> desc emissions ;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
---
 ST_EMI_FILLER                                      CHAR(4)
 ROW_EMISSIONS                             NOT NULL NUMBER(38)
 JDS_NEW_FIELD                                      CHAR(5)
```

At this point the database is ready, and could be “populated” as required using Oracle commands or a COBOL program.

Re-compile all programs affected using the appropriate “make” script(s).

A second backup of the affected tables may be taken if desired before enabling TIP/ix with the new field in place.

The TIP/ix system may be restarted so that the change can be tested. Instead of having to shut all of TIP/ix down, you could also down just the schema you are working on using the command `tippcstm down` *`myschema`*.

```
tipctl b (and/or use the site script to enable the application in question)
```

To change the size of data field within an existing record, the following steps would be followed:

Essentially if the change being done is to simply modify the size of a data field, then the steps included above for adding a new column would be identical except that the “alter table” command would become:

```
Alter table recordname modify column fieldname …
```

For example, if the newly added field JDS-NEW-FIELD in the EMISSIONS record should have been 6 characters instead of 5, then we would do something like the following:

After checking out the *schemaname*.sch file from CVS using the editmod *schemaname*.sch command, you would simply use the text editor of your choice to make the change to the schema.

```
       RECORD NAME IS EMISSIONS
           RECORD CODE IS 157
           LOCATION MODE IS DIRECT EMISSIONS-KEY STATION-AREA
                WITHIN MVIP-STATION
           RECORD MODE IS ASCII

                 03  ST-EMI-FILLER              PIC X(4)
JDSJDS     03  JDS-NEW-FIELD          PIC X(6)
```

The schema must then be processed by the **dbischema** utility:

```
</home/davids/vsp/schema> dbischema mvip-schema.sch
TIP/dbi Schema Compiler;     Version  1.37 2004/02/10
        ® 1991-2004 Inglenet Business Solutions

 Oracle8 version: 8.1.5.0.0
Reading `mvip-schema.sch'
Removing old /home/davids/vsp/ddidd/mvip-schema.dd3/mvip-schema.sym
Default TABLESPACE is:      MVIP_DATA_TS1
Default INDEXSPACE is:      MVIP_INDEX_TS1
Reading Data Base Data Names at line 11
Reading AREAs starting at line: 40

If the last field of a record is FILLER, then it is not part of the SQL
schema
Reading RECORDs starting at line: 228
INCLUDE ST-LOCATION
Reading SETs starting at line: 1038
Finished parsing 1207 lines, 0 seconds elapsed, 1207 lines per second
Generating with Oracle 8 limits: Max CHAR 1995, Max Columns 995
Create Database script already exists
Create Tablespace script already exists
Writing SQL Schema definition: `/home/davids/vsp/ddidd/mvip-schema.dd3/mvip-
schema.ddl'
Most columns in any table was 500
Most indexes on any table was 3
49 records defined

Total 1 seconds elapsed, 1207 lines per second DMS2200
TIP/dbi Data Mapping Generator;  Version 1.47 2004/02/10
  ® 1991-2004 Inglenet Business Solutions
    DBI Generator schema: mvip-schema
       Schema dictionary: /home/davids/vsp/ddidd/mvip-schema.dd3/mvip-
schema.sym
      SQL DDL definition: /home/davids/vsp/ddidd/mvip-schema.dd3/mvip-
schema.ddl
      Data mapping rules: /home/davids/vsp/ddidd/mvip-schema.dd3/mvip-
schema.map
    Generating output to: /home/davids/vsp/ddidd/mvip-schema.dd3
 >>> Opened DMS schema dictionary and parsed SQL DDL <<<
Finished parsing data mapping rules, now generating TIP/dbi run-time
  >>> Maximum sequence value is 999,999,999 <<<

TIP/dbi generation completed
```

The updated Oracle version of the schema may be viewed and verified:

```
</home/davids/vsp/schema> cd $TIPDMS/mvip-schema.dd3
</home/davids/vsp/ddidd/mvip-schema.dd3> view mvip-schema.ddl
--
-- Table   8, Rcsz=   8, SR157 : EMISSIONS : 3 SQL columns
--           Estimated Max Row Size   20
--
DROP TABLE  emissions CASCADE CONSTRAINTS;
CREATE TABLE emissions (
       st_emi_filler                CHAR(4),
```

```
 jds_new_field                CHAR(6),
 row_emissions                   INTEGER NOT NULL,
 CONSTRAINT pk_emissions PRIMARY KEY(row_emissions)
 USING INDEX STORAGE (FREELISTS 3)  TABLESPACE MVIP_INDEX_TS1
)
            STORAGE (FREELISTS 3 )
TABLESPACE MVIP_DATA_TS1;
```

The schema layout has been changed, and now the actual database must be updated to reflect the altered size of the field that was changed.

The **dbidiff** utility is used to compare the schema layout to the Oracle structure and show the differences and the changes that need to be made to the database.

```
</home/davids/mvip-schema.dd3> dbidiff -U xxx -P yyy mvip-schema.ddl
TIP/ix DMS to Oracle Schema difference generator (Ver: 1.1 2004/02/04 )
   ® 1991-2004 Inglenet Business Solutions
Alter Schema written out to mvip-schema.alter

A visual check of the changes that TIP/dbi will require to be made to the
database may be done prior to actually running the script.
</home/davids/vsp/ddidd/mvip-schema.dd3> cat mvip-schema.alter
spool alter
ALTER TABLE emissions
-- length 6 > hLength 5
     MODIFY (jds_new_field                  CHAR(6))
;
spool OFF
exit;
```

After verification that the changes are what is required, the database is then updated to reflect the new schema structure within the Oracle tables:

```
</home/davids/mvip-schema.dd3> sqlplus user/pwd < mvip-schema.alter

SQL*Plus: Release 8.1.5.0.0 - Production on Thu Feb 19 16:39:32 2004

(c) Copyright 1999 Oracle Corporation.  All rights reserved.


Connected to:
Oracle8i Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production

SQL> SQL>   2    3    4
Table altered.

SQL> SQL> Disconnected from Oracle8i Enterprise Edition Release 8.1.5.0.0 -
Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production


SQL> desc emissions ;
```

```
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
---
 ST_EMI_FILLER                                      CHAR(4)
 ROW_EMISSIONS                             NOT NULL NUMBER(38)
 JDS_NEW_FIELD                                      CHAR(6)
```

To Add a New Record Type Into The Database

The process to insert a totally new record type into an existing schema would follow similar steps to adding a new column or increasing the size of a column, but with the exception being that a DMS COBOL "load" program must be created in order to populate the database in a manner that TIP/dbi will understand.

```
JDSJDS RECORD NAME IS ZMISSIONS
JDSJDS     LOCATION MODE IS DIRECT EMISSIONS-KEY STATION-AREA
JDSJDS     WITHIN MVIP-STATION
JDSJDS     RECORD MODE IS ASCII
JDSJDS
JDSJDS     03  ST-EMI-FILLER-Z             PIC X(4)
JDSJDS

--
-- Table  50, Rcsz=   4, SR9998 : ZMISSIONS : 2 SQL columns
--           Estimated Max Row Size   16
--
DROP TABLE  zmissions CASCADE CONSTRAINTS;
CREATE TABLE zmissions (
        st_emi_filler_z               CHAR(4),
        row_zmissions                 INTEGER NOT NULL,
        CONSTRAINT pk_zmissions PRIMARY KEY(row_zmissions)
        USING INDEX STORAGE (FREELISTS 3)  TABLESPACE MVIP_INDEX_TS1
       )
                  STORAGE (FREELISTS 3 )
       TABLESPACE MVIP_DATA_TS1;
```

To make changes to records which involve sets or indexes

To make a change to the database which involves changes or additions to sets or indexes, then the database records involved should be unloaded using the old schema, and then reloaded using the new schema.

If the site does not have their own unload/reload programs, then the **dbiunload** utility supplied with TIP/dbi can be used to create "skeleton" COBOL programs to accomplish this task.

The programs generated by the **dbiunload** utility must be closely examined and must be tweaked "manually" to ensure that the desired results are achieved during the unload/reload process.

```
cd $TIPSITE/dba_programs
dbiunload –S schemaname –U subschemaname –l areaname(s) or ALL
make clean
make
```

Here is an example where the MVIP-STATION area is going to be unloaded in preparation for a change to that area -→

```
(SUNFIRE1:/home1/davids)1235 $ unload.sunfire1.mvip
Start MVIP Unload Programs at Wednesday February 18 15:40:01 EST 2004
Start mvipsul0 at Wednesday February 18 15:40:01 EST 2004
Start mvipsul at Wednesday February 18 15:40:01 EST 2004
PROCESS EMISSIONS
UNLOADED 000000000001 EMISSIONS RECORDS
PROCESS STATION-GENERIC
UNLOADED 000000000053 STATION-GENERIC RECORDS
PROCESS SET GEN-LOC
UNLOADING ST-LOCATION FROM AREA MVIP-STATION
UNLOADED 000000000055 ST-LOCATION RECORDS
PROCESS SET GEN-INV
UNLOADING YR-INVENTORY FROM AREA MVIP-STATION
UNLOADED 000000000072 YR-INVENTORY RECORDS

UNLOAD COMPLETED
UNLOADED 000000000001 EMISSIONS RECORDS
UNLOADED 000000000053 STATION-GENERIC RECORDS
UNLOADED 000000000055 ST-LOCATION RECORDS
UNLOADED 000000000072 YR-INVENTORY RECORDS
End MVIP Unload Programs at Wednesday February 18 15:40:03 EST 2004
```

The sample runstream for this would look something like this -→

```
(SUNFIRE1:/home1/davids)1245 $ cat unload.sunfire1.mvip
#!/usr/bin/ksh
echo Start MVIP Unload Programs at `date`
export TIPDMSLOG=
cd /home1/davids
# rm -f log.* WORK*
echo Start mvipsul0 at `date`
$TIPSITE/bin/mvipsul0
echo Start mvipsul  at `date`
$TIPSITE/bin/mvipsul

echo  End MVIP Unload Programs at `date`
(SUNFIRE1:/home1/davids)1246 $
```

The files produced by running this job would be -→

```
-rw-rw-rw-   1        1749 Feb 18 15:40 FDXREF
-rw-rw-rw-   1        3072 Feb 18 15:40 CKPTFLE.idx
-rw-rw-rw-   1          81 Feb 18 15:40 CKPTFLE
-rw-rw-rw-   1       27648 Feb 18 15:40 SET0158
-rw-rw-rw-   1       11770 Feb 18 15:40 SET0161
-rw-rw-rw-   1       11077 Feb 18 15:40 FD0158
-rw-rw-rw-   1          24 Feb 18 15:40 FD0157
-rw-rw-rw-   1         373 Feb 18 15:40 log.MVIP-SCHEMA.5144
```