



RE/data

RE/data
Reference Manual
Version 1.4

RE/data – Reference Manual Version 1.4

Contents

Introduction
Overview
The Data Dictionary Creation Procedures
Code Analysis
The Build Phase
Raw Dictionary
Database Validation
Merge Nodes
Node Maintenance
Attribute Maintenance
Aggregation
Entity Creation
Exports
Data Type Mapping

Appendices

Installation procedure

Notices

Copyright 1990-1998 George James Software Limited.

No part of this publication may be reproduced, transmitted, stored in a retrieval system, nor translated into any human or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of the copyright owner, George James Software Limited, 42 High Street, Shepperton, Middlesex, TW17 9AU, England.

Information contained in this publication is subject to change without notice and does not represent a commitment on the part of George James Software Limited.

Any copyrighted software accompanying this publication is licensed to you only for use in strict accordance with the Software License Agreement accompanying the software. Please read the license agreement carefully before commencing use of the software.

RE/data, RE/2000, RE/m, RE/parser and VC/m are trademarks of George James Software Limited. All other brand and product names are or may be trademarks of, and are used to identify products and services of, their respective owners.

Order code 1007.

Product Support

Support is available to all users of George James Software Products who have a current Software Maintenance Agreement. Support and assistance can be obtained from the following sources:

| | |
|-----------|--------------------------|
| Telephone | +44-1932-252568 |
| Fax | +44-1932-254816 |
| E-mail | support@georgejames.com |
| Web page | georgejames.com/support/ |

| Revision | File Reference | Date | Author | Notes |
|----------|-----------------------|----------------|-------------------|-------------------|
| 0 | t48/gj/doc/redref14.0 | 25 August 1995 | George James | |
| 1 | redref14.1.doc | May 1998 | George James | Reformatted by JN |
| 2 | redref14.2.doc | November 2001 | George James | Update logo |
| 3 | redref14.3.doc | 18 June 2002 | Gail Treves-Brown | Latest logo |



CHAPTER 1

Introduction

Introduction

RE/data is a data analysis tool. It analyses both the code and database contents of M applications and produces a representation of the structure and contents of the database used by the application.

This information about the database can be further manipulated and refined within RE/data's integral data dictionary. RE/data is able to represent both a physical definition of an M database (in terms of Global nodes and Attributes) and a logical definition (in terms of Entities).

It can then be exported in a number of different formats to third party dictionary driven products such as CASE tools, 4GLs and report writers (including generic SQL DDL, Greystone's GT.SQL, InterSystems' M/SQL, KB Systems KB-SQL and ISG's CorVision database).



CHAPTER 2
Overview

Overview

A data dictionary is a description of the structure and contents of a database.

A database is described logically in terms of Entities and Attributes and physically in terms of Records and Fields. Each Record and Field has a physical address which describes where it is actually located in the database. A data dictionary contains the information about Entities, Attributes, Records and Fields that describe the meaning of each of these items.

Virtually all modern database products are driven by a data dictionary or database schema of some kind. M databases do not require a data dictionary, of necessity, although many tools that are layered on top of M databases do.

While it is possible to build an M database application without any kind of data dictionary, if you want to access your database from any dictionary driven tool (for example, all SQL based tools) then you will need a data dictionary.

Ideally the data dictionary should be created at the time that the application is designed and built. There are many text books that prescribe a method for data analysis and design, most using a form of Entity/Relationship modeling. If a data dictionary is not created at the time that the application is originally built, and if it is not maintained in line with the application, then it can be a very expensive and difficult task to create the dictionary retrospectively.

RE/data is designed to aid the process of building a data dictionary for those M applications that do not already have one. It can also help to validate a dictionary that does exist but has not been kept up to date with the application code.

RE/data takes information from various sources as input and combines it to form a skeleton which can be filled in with descriptions, length and data type information where necessary. The primary source that RE/data uses is analysis of global references within the application source code. The second source of information is fully populated production databases. Statistical sampling of data within a database can provide additional information about the data type and length of attributes. It also enables foreign key relationships between globals to be validated. A third source of information that RE/data can use is any existing file layouts or database descriptions. This can either be input directly to RE/data using a custom built import process, or entered manually as and when appropriate.

Once the various data sources have been fed into RE/data they can be refined and edited using RE/data's built in data dictionary. Finally, a fully refined dictionary can be exported in any of the supported dictionary formats for use with third party dictionary driven tools.

The Data Dictionary Creation Procedure

Contents

Estimating

Code analysis

Build the Raw Dictionary

Import File Layouts

Database Validation

Merge Nodes and Aggregate Fields

Dictionary Refinement

Re-validate against Database

Define Logical Entities

Export to Target Dictionary

The Data Dictionary Creation Procedure

This section describes the overall procedure for using RE/data to build a data dictionary.

Re-engineering a data dictionary is as much an art as it is a science. RE/data provides a set of tools that assist in the process, but it is worth remembering that it is the end result that matters and that the best method of achieving these results will vary from project to project.

The starting point, and raw materials available at the start of the dictionary creation process will vary from project to project. The final result of each project may also differ depending upon the requirements of the target dictionary and whether a logical or a physical model is to be created. For these reasons, this section should be considered to be a starting point for planning the dictionary creation process. The steps that should be carried out and their sequence may need to be varied from project to project.

Estimating

One of the most difficult tasks is to estimate how long it will take to create the dictionary using RE/data. It is difficult to provide definitive guidelines because the quality of the various information sources can vary greatly from one project to another.

The following factors should be taken into account:

- The quality of the source code. Consider, use of indirection, naked references, standardized variable names, etc.
- Availability of sample production databases. Validating the dictionary against a production or test database can help in much of the data type and field length information. The more data that can be validated the better the quality of the results.
- Availability of file layout documentation.
- Knowledge of the application by the team performing the dictionary build. Access by the team to application experts.

For large projects it is recommended that a small part of the database be documented and the time taken for this part be used to calibrate how long it will take to build the rest of the dictionary.

If it is not feasible to calibrate the process then the following rule of thumb can be used as an initial estimate:

- Identify the number of global nodes in the application. RE/m or RE/data can be used to obtain an accurate figure for the number of nodes.
- Approximately 25 to 50 global nodes can be documented per man day depending upon the above factors.
- For small projects the learning curve in using the RE/data tool may need to be taken into account.

- For large projects or where there is a long elapsed time for the dictionary build the application being analyzed may be a moving target. Changes to the database due to work in progress must be tracked and incorporated into the dictionary at some point.

Code Analysis

The M code that makes up an application is probably the most accurate source of information about the structure and contents of your database. It is however sometimes a vague and incomplete source. Vague because, for example, the code may not access a particular field in the database, and so there will be no information that can be derived from the code about that field. Incomplete because, for example, any global references performed using indirection may not be 'seen' by the RE/data code analyzer.

The code analysis does however provide a structural framework of the database to which additional information can be added as and when it is discovered. It also provides a large number of 'clues' that can help to decide how a field is used and what it might contain.

The code analysis phase builds a reference database which contains a syntactic and structural analysis of the M source code. This database forms the input to the build phase.

Build the Raw Dictionary

The build phase operates on the reference database and creates an entry in the raw dictionary for each global node (and for each identified field in each node). It is advisable to build every node in the reference database (except temporary and work files) even if the intention is to only document some nodes initially. This is because foreign key relationships between nodes can help to identify fields by virtue of their links to fields in other globals.

Import File Layouts

If you have existing file layout or dictionary information then if you have a large quantity of this information or it's quality is quite high then it may be desirable to import it into RE/data at this stage. If the volume is small or the quality low then it may be better to input it manually (and therefore more selectively) at the 3D/m refinement stage.

If it is considered worthwhile to import existing file layout information at this stage then a custom import program will need to be written. Contact George James Software, or your distributor, for details of what is involved in having an import program custom built.

Database Validation

The next step should be to validate the raw dictionary against a production database. The validation process will fill in field lengths and data types. The validation summary report will also highlight nodes in the database that are not represented in the raw dictionary.

It may be advantageous to perform some merging and aggregation before validating the dictionary.

Merge Nodes and Aggregate Fields

The next step is to refine the raw dictionary definitions. The objective of this step is consolidate the raw information from the code analysis into more logical information and to remove *noise* and other spurious information from the dictionary.

- Node merging. Where several instances of global nodes map onto the same table these can be merged together at this point to create a single node definition.
- Field Aggregation. Where several instances of the same attribute occur in different parts of the database, and there are foreign key relationships that link these fields, then these instances can be aggregated to form a single attribute definition.

After merging and aggregation has been completed it is often worthwhile to re-validate the dictionary. This may highlight human errors introduced during this stage.

Dictionary Refinement

The nodes and attributes in the dictionary can now be reviewed and refined with any missing information and descriptions added as necessary.

It is recommended that refining is performed on a global by global basis. Each global should be merged, aggregated and re-validated in turn and then refined before the next global is processed.

Before commencing on this task some consideration should be given to a number of factors concerning the overall design of your dictionary. In particular you should consider the following:

- The characteristics of the target data dictionaries
- Naming conventions of node and attributes
- Partitioning of large data dictionaries into functional areas
- Data types. What data formats are currently used in the application and how do they correspond to the target data dictionary's data types. Consult section 14 for details of how to map user defined data types.

Note that refining is the most time consuming part of the whole dictionary creation process. It can include naming and describing each attribute and can entail a significant amount of detective work to identify the meaning of some fields.

Re-validate against Database

It may be appropriate at this stage to re-run the database validator against a production database. While, in theory, there should be no exceptions because the validation has already been performed, errors may have been introduced inadvertently during the manual refinement phase. Re-running the validator at this point will provide additional assurance of the quality of the dictionary.

Periodic running of the validator can help to assure that the dictionary is accurate and up-to-date. It may also highlight errors in the applications database.

Define Logical Entities

Nodes that have common keys can be consolidated to create logical entities (or views). This can help to combine attributes that are on different physical nodes (on the same or different globals) that are all identified by the same keys, thus reducing the total number of tables required to describe the database.

This step is optional and is not necessary for many dictionaries.

Export to Target Dictionary

The refined dictionary can finally be exported to the target dictionary either on a node by node basis or all in one go. The exact export procedure depends on the particular target dictionary.

The following exports are currently supported:

- Generic SQL DDL
- Greystone GT.SQL DDL
- InterSystems M/SQL Data Dictionary
- KB Systems KB-SQL Data Dictionary
- ISG CorVision Data Dictionary

Other exports are under development and custom exports can be provided to meet specific requirements and to interface with other third party products. Contact George James Software, or your distributor, for details of custom exports.



CHAPTER 4

Code Analysis

Contents

Analyze

Database Size

Special Cases

Suppress Global Environment Information

Code Analysis

The RE/data code analyzer is the primary method for generating a raw dictionary. It uses the standard RE/m code analyzer to parse a set of routines and build a reference database that describes these routines. Once the reference database has been created a raw dictionary can be built on a global node by node basis from the reference database.

Analyze

Consult the RE/m code analysis chapter for full details of how to invoke and operate the RE/data code analyzer.

The RE/data code analysis phase creates a repository of data that is used by the Build phase to create a raw dictionary. Ideally all routines comprising a complete system should be analyzed by the RE/data code analyzer before any raw dictionary is created. This ensures that the Build phase has the most complete, and therefore best, set of information to draw on.

RE/data is driven by two parser tables. One is used by the code analyzer for the initial static analysis, and the other by the Build phase for evaluating data types from expressions. They are named as follows:

○ REDn.n (Main Static Analysis)

This parser table contains a complete definition of M syntax. It contains a large number of calls to sub-routines that create the RE/data repository, so care should be taken if you modify or extend this parser table.

In addition to parsing ANSI 1990 standard M syntax it also allows through a number of common extensions to the M language. They are as follows:

Z commands. Any command starting with a Z is treated as valid. Each argument of such a command is parsed as a valid M expression.

\$Z functions. Any function starting with a Z is treated as valid. Each argument to the function is parsed as a valid M expression.

VIEW and \$VIEW. Any number of arguments to the View and \$View statements are permitted provided they are valid M expressions. Arguments to the View command may be separated either by a comma or by a colon.

Global ^space. References to a global with a name comprising of a single space (DSM and MSM routine name index) are treated as valid and stored in the repository as references to the proposed ANSI standard `ssvn ^$ROUTINE`.

Extended Global Environment Syntax. The de-facto syntax for referencing globals in other UCI's is supported, ie [UCI,VOL].

Null argument \$Text. \$text with a null argument (ie \$T()) is interpreted by RE/data as valid syntax and is processed in exactly the same way by RE/data as \$T(+0).

○ REDXn.n (Data Type Evaluation)

This parser table is invoked from within the build phase of RE/data and is used to analyze expressions and derive data type information (for example if the expression a+b is analyzed it will be deduced that both a and b have numeric meaning and can therefore be considered to be numeric data types).

If any changes are made to the parsing of expression syntax within the standard RE/data parser table then an equivalent change must be made to this parser table, otherwise the RE/data build phase may fail.

Unlike the main parser table for RE/data which can have any name provided it is declared in the resetup Configure screen, the REDXn.n table cannot be re-named.

Database Size

The RE/data reference database is relatively large. Its size will vary depending upon the size and complexity of code being analyzed. As a rough guide approximately 30-50 Mbytes should be allowed for each 1000 routines analyzed.

Once a dictionary has been built it is not essential to retain the reference database. If disk space is tight then it can be deleted by using the Delete option in the Routine maintenance function to delete all routines. This will not delete the contents of any dictionaries.

Note, however, that if you wish to rebuild any part of the dictionary or create a new raw dictionary to compare with an old one, then it would be necessary to reanalyze all of your code once again.

Special Cases

○ Nodes containing commas or close bracket characters

Nodes that contain hard coded keys comprising of string literals that included embedded commas and close bracket characters will be stored in the repository with these characters replaced by a # character. The following example illustrate this:

| Global Node | RE/data Representation |
|---------------------------------|----------------------------|
| <code>^X("ABC,DEF",KEY2)</code> | <code>^X(ABC#DEF,-)</code> |
| <code>^X(KEY1,"",KEY3)</code> | <code>^X(-,#,-)</code> |

This is necessary to avoid the possible confusion over the number of keys contained in a global reference when represented in the standard RE/m regularized form.

○ DSM Routine Global

The pseudo global (comprising of a single space character) used by DSM to represent the list of routines stored in a routine directory is represented by RE/data as `^$ROUTINE`. This is consistent with the proposed ANSI standard ssvn for providing access to a list of routines within a system.

Suppress Global Environment Information

When routines are submitted for analysis you have the option to suppress the environment prefix from any global references that have them. The environment prefix is the information in square brackets or between vertical bars which translates a global reference to another UCI, directory or namespace.

If you answer Yes to this question then references to, say, `^CUST(A,B,C)` and `^|”MGR”|CUST(A,B,C)` will be considered to be references to the same global node. If you answer No then each such reference will be stored as a discrete node definition in the raw dictionary.

If you believe that your system contains references to globals *with the same name* but with a different structure or contents then answer No to this question, otherwise answer Yes.

CHAPTER 5

The Build Phase

Contents

Build Overview
Preparation
Backward Search Extent
Forward Search Extent
Raw Dictionary Name
Routine Selection
Variable Selection
Build Progress

The Build Phase

Build Overview

Once the reference database has been created for the set of routines under examination, a raw dictionary can be created using the Build option on the main menu.

For each global node analyzed, a raw dictionary will contain information about the way that each piece of data in the node is accessed and used. In particular, where nodes are made up of many bits of different data concatenated together, either as fixed length fields or as variable length fields with a delimiter of some kind, each piece of data is analyzed in turn.

For each identified data item the raw dictionary will store, when applicable, the names of one or more variables that are used to hold the data item, the inferred data type of the item and its relationship to data in other globals.

Conceptually the build process operates in a similar manner to the way that a person would attempt to understand the structure and content of a global. At every place where a given global node is accessed within an application, the build process 'reads' the code forwards and backwards, and follows the data as it is moved from one variable to another. When a variable containing the data is used in an expression then the data type expression parser is invoked to make inferences about the data type of the item. Because the build process is able to look at every occurrence of a global node within an application and can add fragments of information found in unrelated routines, it is possible to build a fairly comprehensive picture of what data is stored in any global node.

The build function creates entries in a raw dictionary on a node by node basis. The dictionary for a single node can be created or a set of nodes can be selected (for example all nodes in a global, or all nodes in a complete application).

When the raw dictionary for a node is created all references to that node within the selected set of routines will be analyzed. Each variable used to refer to keys, fields and sub-fields and any relevant format information is stored against that node in the raw dictionary. If any node has been built previously and has information store in the raw dictionary for it, this will be discarded when the node is re-built. If this information is of value it should be exported to the data dictionary before the node is rebuilt.

Once built, the contents of a raw dictionary can be supplemented by comparing it against a live or production database. See section 7 for further details of the Database Validation process.

Preparation

The following points should be considered before starting to build a raw dictionary:

- 1 The code analysis of the application should have been completed.
- 2 If a standard field delimiter is used and is stored in a variable this should be declared in the appropriate set-up function.

- 3 If the application uses the "@" character as a delimiter anywhere in the system then the raw dictionary internal delimiter should be changed from "@" to a character that is not used by your application.
- 4 Any temporary work files that are used by the application should be identified and not included in the build phase. In this context a temporary work file is any global, that is referred to by the same name, but has a different global structure for each program that uses it. Typically such globals are transient, deleted at either the start or end of the program, and often have a first subscript of \$j or some other unique process identifier.

Backward Search Extent

The backward search extent determines how far backwards the build process will search from the place where a global reference occurs. Typically the backward search extent should be a figure between 10 and 500, although the best setting is dependent upon the coding style used in the application being analyzed.

Note that a higher setting will result in slower performance of the build phase and is also subject to the law of diminishing returns.

Forward Search Extent

The forward search extent determines how far forward the build process will search from the place where a global reference occurs. If fields are normally unpacked and used near the place where they are accessed in the database then the forward search extent can be lower than if the data is unpacked a long way from the point of the database access. Typically the forward search extent should be a figure between 100 and 1000.

As for the backward search extent, setting the forward search extent to a very high value will reduce the performance of the build process, but may not elicit much additional information.

Raw Dictionary Name

Enter the name of a raw dictionary. Multiple raw dictionaries can be created. This enables parts of an application to be analyzed separately and independently if so required, however in most cases where an application operates on one coherent database only one raw dictionary should be required.

Note that if an existing raw dictionary is selected and the nodes selected already exist in that dictionary they will be completely replaced by the results of the analysis of the new nodes. Conversely, if the nodes selected do not exist in the raw dictionary they will be added but will not affect any other nodes that already exist in the dictionary. Consult section 6 for details of how to copy, merge and delete complete raw dictionaries.

Routine Selection

The routine selector provides a way of using only part of the reference database to build the raw dictionary for a node. The default is to analyze all references to each selected node in the reference database.

In some cases it may be desirable to select only a subset of the repository for analysis. For example, a sub-system may use a particular global in a way that is different to the way that the same global is used by other sub-systems. Another example is where a global node is referenced in many hundreds of different places throughout a system. In this case it is likely that all of the useful information that can be gathered will be obtained well before all occurrences of the node have been analyzed. Selection of a sub-set of the routines would result in the same quality of information in the raw dictionary but in a much shorter time.

Note that the selection only controls the selection of the routines to use as start points for the analysis. If a selected routine calls a routine that has not been selected then build process will still analyze the called routine, if it needs to, despite the fact that it is not in the selection set.

Variable Selection

Any global node referenced in the set of parsed routines may be selected in the build function. If the raw dictionary for a node has already been built before in the selected raw dictionary, then the results of the previous analysis will be discarded before the node is reanalyzed.

Build Progress

While the build process is running progress messages will be displayed on line two of the screen. Each selected node is processed in turn. For each node all occurrences of the node in the selected set of routines is analyzed. The progress message will therefore display the current node and the current routine being analyzed. In addition an internal address within the current routine is also displayed.

CHAPTER 6

Raw Dictionary

Contents

Show/Print Raw Dictionary

Copy/Merge Dictionary

Purge

Delete Whole

Raw Dictionary

A raw dictionary comprises of a very physical definition of each global node as accessed by an application. This can optionally be supplemented by data type and length information derived from analysis of a database containing real data.

Multiple raw dictionaries can be created within RE/data. They are all totally independent of each other.

Show/print Raw Dictionary

This inquiry enables the contents of the raw dictionary to be viewed or printed.

One or more global nodes may be selected using the standard variable selection mechanism. Each node is displayed formatted as a record showing all the information that has been derived for each key, field and sub-field related to the node.

The following information is displayed for each node:

- Node

The regularized name of the node. Each variable subscript is represented by a hyphen. Each constant subscript is represented by the value of the constant.

- Field Reference

Each item within a node is identified by a field reference. This can be a key field (subscript), a data field or the whole global node itself.

- Keys

Key fields, or subscripts, are represented by the identifier
Kn where n is the subscript level within the global node.

- Global Node

The field that represents the complete global node is labeled either 'Node' if the node contains data or 'Pointer' if the node does not contain any data fields.

If the code analysis indicates that the node does contain data but the database analysis finds no data in that node then the field reference will be 'Node/Pointer'. If the opposite situation is the case then the field reference will be 'Pointer/Node'.

Each variable that is synonymous with the global node is listed under this heading. A variable is synonymous if at some time it contains the same data as the global node. In a program if a variable is set to a global node, or vice versa, then it is considered to be a synonym.

For example:

| | |
|---------------------------------|--|
| S ABC01=^ABC(CUST) S Z=ABC01 | ABC01 and Z are both synonyms of this node |
| S ^DEF(1,ACCOUNT)=X | X is a synonym of this node |

○ Data Fields

Each pieced field within the global node is represented by it's piece number. For example, if a field has a field reference of 3 then it is the third piece of the global node. The actual delimiter that is used is shown in the delimiter column.

If a field is made up of a range of pieces then it's field reference will be shown as a range. For example, 4-7 means pieces four through seven of the node.

If a field has no delimiter in the delimiter column then this means that it is a fixed length field and in this case the field reference refers to the character positions of the field within the node. For example, a field reference of 1-5 means characters one through five of the node.

Sub-fields are represented as a combination of two field references separated by a period. For example, a field reference of 3.2 means piece two of piece three of the global node. The primary delimiter is shown in the delimiter column for field three. The secondary delimiter is shown against the field 3.2.

If a node contained the following data:

"100001|A|12/1/1993|200.32|0|1"

then the following description would apply for the date field:

| Ref | Del | Name |
|-----|-----|-------|
| 3 | " " | DATE |
| 3.1 | "/" | MONTH |
| 3.2 | "/" | DAY |
| 3.3 | "/" | YEAR |

If the node contained the date field in a fixed length form, for example:

"100001|A|19931201|200.32|0|1"

then the following description would apply for the date field:

| Ref | Del | Name |
|-------|-----|-------|
| 3 | " " | DATE |
| 3.1-4 | | YEAR |
| 3.5-6 | | MONTH |
| 3.7-8 | | DAY |

Note that the field names for month, day and year are indented by one character position to indicate that they are sub-fields.

○ Delimiter

The delimiter column shows the delimiter that is used for the field. If the RE/data analyzer has been able to resolve the delimiter as a constant then it will be shown as a quoted string. If the delimiter was unresolved then a question mark will be shown.

If a field does not have a delimiter then it is a fixed length field. See the section above about field reference notation for more details about the representation of fixed length fields.

○ Field Name

The field name column lists, for each field, each of the variables that are used to hold the data contained in that field. Each variable that is used to represent the field is called a synonym of that field. If a variable representing a field is set to a constant value at some place in the code then a constant (enclosed in quotes) may be shown as a synonym. This often means that this value has some special significance to the application (for example, a hard coded piece of logic).

○ Frequency

This column lists the frequency of reference of each synonym. The frequency is counted as the number of times that the variable is referenced within the scanned code.

If there is no established standard name for a data item the most frequently used variable can be considered to be a good de-facto standard name.

○ Data Type

The Data Type field comprises of two columns. The left hand column contains the data type inferred from analysis of the code (if any) and the right hand column contains the data type of data that is actually found in the database by the database verification process.

On export to the 3D/m dictionary the data type derived from the database will take priority over the other data type. The database data type is generally considered to be more precise because most M code is independent of the actual data type of the data being handled.

The following data types are supported:

| Data Type | Description |
|-----------|--|
| blank | RE/data was unable to derive any type information |
| M | Mixed alphanumeric and numeric |
| C | Character string (alpha or alphanumeric, but <i>not</i> numeric) |
| S | Signed numeric field (positive or negative) |
| N | Numeric field (positive, real or integer) |
| B | Boolean (0 and 1) |
| 1 | The field only ever contains the value 1 |
| 0 | The field only ever contains the value 0 |

In the left hand column if the code analysis reveals that a field is referred to both as a string and as a numeric field then it will have a format code of CN.

In the right hand column if the database validator identifies that a field has both numeric and alphanumeric values then it will be assigned a data type of M. While numeric data can conveniently be considered as a sub-set of the alphanumeric data type, M-technology collates numeric values differently when they are subscripts of an array and so for some applications it may be important to know when you have got a pure alphanumeric field and when you have got a mixed numeric and alphanumeric field.

Data types 0 and 1 are really transitional data types. The database validator may identify that a field contains some instances of the value 1, this may be a numeric field or it may be a Boolean field. As the validator samples more values if it encounters an instance of a 0 then it will change the data type from 1 to B. If it encounters any numeric values other than 0 or 1 it will change the data type to N.

○ Minimum, Maximum and Average Length

These fields show the minimum length, maximum length and arithmetic mean length of the field. This information is obtained only by database analysis.

Only the maximum length is exported to the 3D/m dictionary. The minimum and average lengths are shown for information only.

○ Comments

The comment area is used to list a number of other attributes of each field as and when they occur:

○ Foreign Keys

Where a field is a pointer to another global node, either individually or in other keys, then the node reference of each node that is pointed to will be displayed in this column. The node will be displayed in regularized format, but the subscript of the referenced node that corresponds to the current field will contain an asterisk instead of a hyphen.

If the pointer is to a data field within a node then the node will be displayed followed by the field reference of the field concerned.

The following example shows a node in a transaction file containing a currency field that points to two other global nodes, the first is a node containing a table of currencies with data stored against each currency code. The second is a node that is an index into the transaction file by currency code and transaction reference.

| Global: ^TRANS (-) | | | | |
|---------------------------|------------|-------------|-----------|-------------------|
| Ref | Del | Name | .. | Comments |
| K1 | | transref | | Key: ^TRANSX(-,*) |
| Node | | rec | | |
| 1 | “*” | currency | | Key: ^CCY(*) 97% |
| 2 | “*” | amount | | |
| 3 | “*” | date | | |

Each foreign key is show at both ends of the relationship. For the above example, foreign keys that point to ^TRANSX will also be show as pointers from ^TRANSX to ^TRANS:

| Global: ^TRANSX (-, -) | | | | |
|-------------------------------|------------|-------------|-----------|---------------------|
| Ref | Del | Name | .. | Comments |
| K1 | | currency | | Key: ^TRANS(-) 1 |
| K2 | | transref | | Key: ^TRANS(*) 100% |
| Node | | “” | | |

○ Correlation Coefficients

If the dictionary has been validated against a production database then following the foreign key may be a correlation coefficient. This is expressed as a percentage and indicates the degree of correlation between the current node and the node pointed to by the foreign key. A figure of 100% will indicate a mandatory relationship. A figure of between 1% and 99% indicates an occasional relationship. A figure of 0% indicates no relationship at all. If there is no correlation coefficient

figure against the foreign key then this means that the validator was unable to test the correlation because the subscripts of the designated foreign key are not all available in the current record.

The correlation coefficients are used by the export process to aggregate fields in different globals together as common attributes.

○ Seeds

Where a global node is processed sequentially, if the code analysis identifies a specific seed value for \$next or \$order operations then this is displayed as a seed value. In some cases this can provide evidence of structural subtleties that are not otherwise apparent.

○ Optional Nodes

The database analysis may identify that some nodes are not always present. ie sometimes there is data in the node but sometimes the node is not present or it is a pointer (with a \$data value of 10). When the database analysis identifies this, the node will be labeled 'Optional' in the comment area.

Copy/Merge Dictionary

The Copy option on the Refine menu enables a complete dictionary to be copied or merged with an existing dictionary.

If two dictionaries are merged and both dictionaries contain a definition for the same node then the result will be an aggregate of the two node definitions. If both nodes contain a definition of the same field then these definitions will also be merged. The synonym frequencies will be summed and all data type and length information combined.

Purge

If the reference database is re-built or if changes are made to the application and the code re-analyzed then node definitions in your raw dictionary can and should be re-built. If the application has ceased to use a node that was previously defined in the raw dictionary then the re-building of the dictionary will not automatically remove the definition of that node.

The Dictionary Purge option will remove redundant node definitions from a raw dictionary by checking whether each node in the dictionary still has a corresponding node in the reference database.

It can be used at any time to ensure that all raw dictionary items correspond to the nodes in the reference database. It should not be run if the reference database has been removed in order to save disk space.

Delete Whole

The delete whole option allows a complete dictionary to be deleted. Care should be taken when deleting whole dictionaries as this operation is not un-doable.

CHAPTER 7

Database Validation

Contents

Dictionary

Global Nodes

Director

Number of Nodes to Sample

Sample Frequency

Number of Nodes to Search

Update Dictionary

Validate Foreign Key Relationships

Re-initialize Dictionary

Data Type Mapping

Global Aliases

Database Validation

The Validation option on the main menu enables a raw or refined dictionary to be validated against a database containing real data. This will validate a number of aspects of the dictionary definitions:

- It will confirm whether each global node is actually used by identifying actual physical occurrences of that node in the database.
- It will identify global nodes in the physical database that do not have corresponding node definitions in the raw dictionary. This happens at the node level within a global, it will not identify whole globals that exist in the database but do not have any definitions in the raw dictionary.
- It will determine whether the existence of each node is optional or mandatory.
- It will identify the actual data type and lengths of data items within each global node. It will also identify the existence of fields containing data that are not accessed by the application.

The outputs from the validation process are threefold:

- 1 A summary report is produced listing the number of nodes sampled and highlighting node level discrepancies between the raw dictionary and the database.
- 2 An exception report is produced that will highlight any data type and length differences detected between one run of the validator and the next.
- 3 The dictionary being validated can, optionally, be updated with length and data type information.

The following parameters are applicable to the validation process:

Dictionary

The validation process requires a raw or refined dictionary containing data definitions to drive its interpretation of the database. Without any database definitions it would be unable to differentiate between fixed and variable subscripts and would also be unable to determine what fields ought to be present and what the delimiters are for each node.

The raw dictionary is also used as the location to store data type and length information that is produced as output from the validation process.

Global Nodes

The nodes to be validated can be selected from the set of nodes that have been defined in the dictionary. This enables validation to be performed on a file-at-a-time basis or for nodes to be selectively included or excluded.

Directory

The location containing the database, against which the validation is to be run, can be specified. It defaults to the current directory but this will not necessarily contain any real data, it is therefore possible to specify an alternative directory/UCI to run against.

The format (and meaning) of this field is dependent upon the M implementation being used.

| M Implementation | Format |
|------------------|-------------------------------------|
| DTM | Namespace |
| DSM | UCI,VOL |
| MSM | UCI,VOL |
| M/SQL | Directory |
| GT.M | Routine directory, Global Directory |

If the database that you wish to validate against is not accessible from the system where RE/data (and the reference database) is installed then it is possible to perform the database validation without copying the complete reference database to the target machine.

This can be achieved simply by installing a fresh copy of RE/data on the system containing the production database and copying the global ^redd and ^redd2 (which contains the dictionaries). Once the validation process has been run against the target database then ^redd and ^redd2 (which will now contain the data type and length information) can be copied back to the original system. The global ^reval contains the summary and exception report details and you may also wish to transfer this back.

WARNING: If you follow this procedure be sure not to use the Build option on the original copy of RE/data as this will update the raw dictionary and a number of indexes contained in other files!

Number of Nodes to Sample

This parameter determines the number of global nodes of each type to sample. It is recommended that a reasonably large number of nodes are sampled in order to gain a statistically reliable result for the data types and lengths.

Sample Frequency

This parameter determines the frequency with which nodes are sampled. In a large database it would be unrepresentative to sample, say, just the first 1000 nodes, so by setting the sample frequency to 7, then every seventh matching node will be sampled. Note that it is recommended that a prime number be used as the sample frequency to avoid any natural regularity in the database.

If a file contained only 10 records then sampling every 7 records would only result in one record being sampled. To avoid this the sampling algorithm will always sample the first n records of the file, where n is the sample frequency, and then every nth record until either the sample maximum or the search maximum is reached.

Number of Nodes to Search

This field determines the maximum number of nodes to search within a database. This operates as a backstop mechanism for the case where a large file actually contains no records of the type being searched for. When the number of nodes that have been searched reaches this limit then processing of the current node will end regardless of the number of matching nodes actually found.

Update Dictionary

It is possible to run the validator without updating the data type and field length information in the dictionary. The primary purpose of this is to revalidate a dictionary against a database and to obtain an exception report of any data in the database that does not conform to the dictionary definitions.

Validate Foreign Key Relationships

The foreign key links in the database can be validated at the same time as the data types and field lengths however this can slow down the validation process and so it is optional.

Re-initialize Dictionary

The field length and data type information in your dictionary can be reset at the start of the validation run. You should do this only if you want to discard the length and format information that you currently have in the dictionary.

Data Type Mapping

The validator is only aware of a limited number of data types. It is able to validate the database against this base set of data types but cannot validate against user defined data types. Where a dictionary contains user defined data types it is possible to define a translation table that maps each user defined data type to one of the base types that the validator understands. See section 14 for details of how to create a data type translation table.

The validator understands the following base set of data types:

| Code | Data Type | Meaning |
|------|------------------------------------|---|
| 0 | 0 | A field containing only 0 or null |
| 1 | 1 | A field containing only 1 or null |
| B | Boolean | A field containing 0, 1 or null |
| N | Numeric | A field containing positive canonical numbers or null |
| S | Signed | A field containing positive and or negative canonical numbers or null |
| C | Character string | A field containing character strings none of which are canonical numbers, or null |
| M | Mixed numeric and character string | A field containing both canonical numbers and character strings |

Global Aliases

In some cases the name of a global in the dictionary may not be the same as the name of the global against which it should be validated. In such cases an alias table can be set up to map the global names in the dictionary to one or more actual global names. This is done by setting up translation table nodes in the following global:

```
>s ^realias(source,target)=""
```

where *source* is the name of the global in the dictionary and *target* is the name of the global against which it should be validated. The following examples show how this can be used:

```
>s ^realias("^abc","^abc1")=""
>s ^realias("^abc","^abc2")=""

>s ^realias("^x@","^x")=""

>s ^realias("^y","^|""MGR""|y")=""

>s ^realias("^[-,-]z","^z")=""
```


CHAPTER 8

Merge Nodes

Contents

Overview of Merge Process

Source Dictionary

Nodes

Target Dictionary

Items to Merge

Fields to Export

Target Node

Merge Nodes

The merge nodes function enables two or more nodes to be consolidated into one definition. It also enables a coarse edit of the fields within node to be performed.

It is often necessary to merge node definitions because the raw dictionary created by the code analysis process can contain multiple instances of nodes that actually refer to the same logical node. The objective of the merge function is therefore to consolidate the physical node references discovered by the code analyzer into logical node definitions.

The merge function has been designed to operate on batches of nodes for maximum productivity, however it can be used to merge any two or more nodes at any time.

It is recommended that nodes are merged on a global by global basis and are aggregated and refined immediately after being merged.

As each global is processed the selected nodes can be copied to a fresh dictionary. This ensures that the original set of nodes is available for reference and also results in a dictionary that does not contain any pointer nodes or nodes that should be discarded.

Overview of Merge Process

The merge process is in three parts. The first part comprises of a screen that enables you to select a set of nodes to be merged. Normally this will be all the nodes in a single global. This screen also allows you to designate the name of the target dictionary.

Each node that has evidence of data (ie it excludes pointer nodes) and will then be presented along with a node selection screen where other candidate nodes can be selected for merging. Any node in the same global with the same number of subscripts and any pointer with the same or fewer subscripts will be candidates for merging. If there are no candidates for merging then this step is skipped.

Once a set of nodes have been selected for merging the fields from each node are consolidated and presented for review and coarse editing. If a node is not merged with any others it will still be presented for review and possible editing.

The following examples shows two different ways of merging nodes within a global and represent two different cases where merging may be required. Assume that a raw dictionary contains the following nodes for a global:

```

^PT( - )
^PT( - , - )
^PT( - , 1 )
^PT( - , 2 )

```

where $\text{^PT}(-, 10)$ and $\text{^PT}(-, 20)$ are actually specific instances of the more general case $\text{^PT}(-, -)$. In such situations it is best to review each node definition and either discard the more general node definition in favor of the specific nodes or use the merge function to merge the specific nodes into the more general node definition. The two choices are shown diagrammatically below:

| Source dictionary | | Target dictionary |
|---------------------|---------|---------------------|
| $\text{^PT}(-)$ | --> | $\text{^PT}(-)$ |
| $\text{^PT}(-, -)$ | Discard | |
| $\text{^PT}(-, 10)$ | --> | $\text{^PT}(-, 10)$ |
| $\text{^PT}(-, 20)$ | --> | $\text{^PT}(-, 20)$ |

Case 1 - Discard general node

| Source dictionary | | Target dictionary |
|---------------------|-------------|--------------------|
| $\text{^PT}(-)$ | --> | $\text{^PT}(-)$ |
| $\text{^PT}(-, -)$ | } | |
| $\text{^PT}(-, 10)$ | } Merge --> | $\text{^PT}(-, -)$ |
| $\text{^PT}(-, 20)$ | } | |

Case 2 - Merge specific nodes

In the second case, it is beneficial to merge nodes $\text{^PT}(-, 10)$ and $\text{^PT}(-, 20)$ with $\text{^PT}(-, -)$ rather than discard them because they may contain valuable information that will be of use later in the refinement process.

Source dictionary

Enter the name of the dictionary from which you wish to select nodes for merging.

Nodes

Select a set of nodes which you wish to consider for merging. It is recommended that merging is performed one global at a time and that all nodes within a global are selected for consideration.

Target dictionary

Enter the name of the dictionary in which the resultant nodes are placed. Normally this should be different from the source dictionary.

If the target dictionary is the same as the source dictionary then pointer nodes which are merged will not be deleted automatically because the information they contain may be useful for merging into more than one descendant node. If necessary pointer nodes can be removed manually.

Items to merge

If there are any candidates for merging then this field will be prompted. If there are no candidate nodes then this field will be skipped.

Press F1 to select from a list of candidate nodes and to review nodes that have been selected for merging automatically.

All pointer nodes which are direct ancestors of the current node will be selected automatically. This ensures that any information about the subscripts of these pointer nodes is taken into account.

Fields to export

The fields from the merged nodes are consolidated together and presented here for review and coarse editing.

Press F1 to review the fields defined for this node. Individual fields can be deleted from the selection list if they are not required in the final node. Within each field it's synonyms are listed with the most frequent synonym first.

Individual synonyms can be deselected if they are inappropriate. Any synonym name may also be changed by selecting it with the cursor (in the left or right hand panel) and then typing E followed by the new name for the synonym.

Target node

Enter the node reference of the target node if it is different from the source node.

CHAPTER 9

Node Maintenance

Contents

Dictionary

Global Node

Global Node

Field

Aggr/Edit/Virt?

Attribute Name

Data Type

Length

Sample Size

Node Maintenance

The node maintenance function enables global node definitions to be refined and new definitions to be created manually.

This function also enables fields linked by foreign key relationships to be aggregated to form re-useable attribute definitions.

Dictionary

Enter the name of the dictionary that you wish to maintain.

Global node

Enter the node reference of the node that you wish to maintain. If it does not exist a new node can be created.

The node reference is expressed in the form:

$$^{\text{gvn}}(-, -, -)$$

where *gvn* is the global name and each hyphen represents a subscript level within the global.

Often a subscript may be a constant rather than a variable. In this case the value of the constant may be entered in place of the hyphen. For example:

$$^{\text{gvn}}(1, -, -, A)$$

It is also possible to use a descriptive label in place of the hyphen, if required, to make the node reference more meaningful. To distinguish the label from a constant value it must be prefixed with a colon. For example:

$$^{\text{gvn}}(:\text{patient}, :\text{admission_date})$$

Global node

At the second global node prompt you can edit the node reference if you wish to change the node reference for this node.

Following this prompt is a repeating set of prompts that enable the fields within the global node to be defined. Each field is described by an attribute, brief details of which can be edited directly on the screen.

Field

This prompt contains the address of the field within the node.

- For variable length delimited fields enter the field reference in the form:

fieldnumber "delimiter"

For example:

10 "*"

This means piece 10 of the node delimited by an asterisk.

- For fixed length fields in a non-delimited record enter the field reference in the following form:

start-end

For example:

15-24

This means a 10 character field in character positions 15 through 24 of the record.

- For key fields (subscripts) enter a K followed by the subscript level within the node. For example:

K3

This means subscript 3 within the node.

- You can also refer to the whole node using the keyword Node or by entering an asterisk as the field reference.

To define sub-pieces of a field enter E at the Aggr/Edit/Virt? prompt. This will display a second screen where you can edit all the details of the attribute. You then can define the sub-pieces of the attribute in exactly the same way as you define the fields within the node.

Aggr/Edit/Virt?

Enter either A, E, V or nothing at this field to:

- A Aggregate together field definitions that are linked to this field by foreign key relationships. If the default value of this field is A then this means that the field has foreign key relationships and may benefit from aggregation.

- E Enter E to edit the full details of the attribute. You can modify the attribute name, data type and length directly on the screen. If you want to add a description or enter other details for the attribute then this option will invoke a second screen where you can edit any of the details of the attribute.
- V Enter V to enter a virtual expression that defines how this field is derived from the node. When a virtual expression is defined for a node this overrides the meaning of the field reference. For virtual fields the field reference serves just as a label to identify the field.

Attribute Name

Enter the name of the attribute that describes this field. Attribute names must be unique within a dictionary. If you enter the name of an existing attribute you will be asked to confirm that you wish to use that attribute to define the field.

If you change the attribute name then remember that it may be referenced in many other places and the name change will apply across the whole dictionary.

If you wish to associate the field with a new attribute and do not want to change the attribute that currently defines it then you should delete the field and then add the field again. Provided that you enter an attribute name that has not already been used then a new attribute will be created automatically.

For keys with a fixed value you should enter the value of the key enclosed in quotes. For example:

| Field | Aggr/Edit/Virt? | Attribute name | Data type | Length | Sample size |
|----------|-----------------|-----------------|-----------|--------|-------------|
| [K1 |][] |]"1" |][|][|][|
| [K2 |][] |]patient |][C |][7 |][44 |
| [K3 |][] |]admission_date |][DY |][|][44 |
| [K4 |][] |] "A" |][|][|][|
| [Pointer |][] |][|][|][|][|

If you enter F1, PF1 or ? at this prompt then you can browse and select from a list of all attributes defined in the dictionary.

Data Type

Enter the data type of the field.

Enter F1, PF1 or ? to browse and select from a list of valid data types.

Length

Enter the maximum length of the field.

Sample Size

This is a display only field that gives an indication of the number of samples that the validation process has made of this field. Where the data type and field length have been obtained by the validation process this information will indicate the degree of confidence that you should place in these values.

CHAPTER 10

Attribute Maintenance

Contents

Dictionary
Attribute
Name
Description
Data Type
Minimum
Maximum
Average
Scale
Sample Size
Sub Fields
Virtual Fields

Attribute Maintenance

The attribute maintenance function enables attributes to be created and maintained. It can be invoked directly or from within the node maintenance function.

Dictionary

Enter the name of the dictionary that you wish to maintain.

Attribute

Enter the name of the attribute that you wish to create or edit.

If the name that you enter is in use then the details of that attribute will be retrieved and displayed for editing. If the name is not in use then a new attribute may be created.

Press F1, PF1 or ? to browse and select from a list of attributes in the dictionary.

Name

If you wish to change the name of an existing attribute then enter the new name here.

Description

You may, optionally, enter a description of the attribute here.

Data Type

Enter the data type of the field.

Enter F1, PF1 or ? to browse and select from a list of valid data types.

Minimum

Enter the minimum length of the attribute. This is optional.

Maximum

Enter the maximum length of the attribute.

Average

Enter the average length of the attribute. This is optional.

Scale

For attributes with numeric data types you can enter the scale. In most cases this identifies the number of decimal places that the attribute has. This is optional.

Sample Size

This is a display only field that gives an indication of the number of samples that the validation process has made of this attribute. Where the data type and length fields have been obtained by the validation process this information will indicate the degree of confidence that you should place in these values.

Sub-fields

Any attribute may be defined as a compound of a number of other fields. Virtual fields may also be defined that are derived from this attribute. For example an eight digit Product Code may be made up of three other attributes:

- Product Category being the first character of the Product Code
- Product ID being the second through seventh characters of the Product Code.
- Check Digit being the last character.

This could be defined as follows:

| Sub-field | Edit/Virt? | Attribute name | Data type | Length | Sample size |
|-----------|------------|------------------|-----------|--------|-------------|
| [1 |][| Product_Category |][C |][1 |][|
| [2-7 |][| Product_ID |][N |][6 |][|
| [8 |][| Check_Digit |][N |][1 |][|
| [|][| |][|][|][|
| [|][| |][|][|][|
| [|][| |][|][|][|
| [|][| |][|][|][|

Each sub-field is an attribute in its own right and may be used to define other fields within the dictionary.

An attribute may also be defined as an instance of another attribute. For example, an audit trail file may contain information about the change of name of a patient. The attribute Patient_Name is already defined in the dictionary. The audit trail record however needs to contain two instances of this attribute, the Old_Patient_Name and the New_Patient_Name. Two new attributes can be created to describe the old and new patient name. By defining each of these attributes as an instance of the Patient_Name attribute they can inherit the properties of Patient_Name and avoid the need to duplicate the length, format and other description information.

The following example shows how the New_Patient_Name attribute can be defined as an instance of the Patient_Name attribute:

```

Name                [New_Patient_Name                ]
[The new name of a patient after a name change transaction has been ]
[recorded.                                               ]
[                                                         ]
[                                                         ]
[                                                         ]
Data type           Minimum   Maximum   Average   Scale     Sample size
[                 ] [         ] [         ] [         ] [         ] [         ]
Sub-field   Edit/Virt?   Attribute name   Data type   Length   Sample size
[*          ][ ][Patient_Name] ][Char]    ][40]    ][      ]
[          ][ ][           ] ][      ]    ][      ]
[          ][ ][           ] ][      ]    ][      ]
[          ][ ][           ] ][      ]    ][      ]
[          ][ ][           ] ][      ]    ][      ]
[          ][ ][           ] ][      ]    ][      ]
[          ][ ][           ] ][      ]    ][      ]
[          ][ ][           ] ][      ]    ][      ]
[          ][ ][           ] ][      ]    ][      ]
[          ][ ][           ] ][      ]    ][      ]
OK Cancel

```

Note that the Data Type and length fields have been left blank. They are inherited from the Patient_Name attribute. If the Maximum Length field had been valued then this would be used instead of the maximum length of the Patient_Name attribute.

Virtual Fields

It is possible to define virtual fields within a dictionary. These are fields that belong to a node (and will become columns within a table in an SQL schema) but do not physically exist in the database. They are normally fields that have their values derived from other fields in the database. A field can be defined as a virtual field simply by entering V at the Edit/Virt/Aggr? prompt and then entering the algorithm that derives the virtual field from other fields. The syntax of the algorithm will vary depending upon the target system. For example, InterSystems M/SQL expects M code with attribute names enclosed within curly brackets, whereas KB/SQL expects an SQL expression. It is possible to enter different versions of the algorithm if you plan to export your dictionaries to more than one target system.

CHAPTER 11

Aggregation

Contents

Aggregation Overview

Correlation Coefficient

Global Node(s)

Name

Description, Data Type, Lengths and Sub-fields

Processing

Aggregation

Any field that has foreign key relationship defined as a result of source code analysis can be aggregated with some or all of the fields linked by the foreign key relationship. The benefit of aggregating such fields is threefold. Firstly the fragments of knowledge about each individual field can be consolidated to quickly create a more complete definition of the attribute. Secondly representing several fields by a single attribute gives economy of definition which eases maintenance and encourages re-use. Thirdly, the use of common attributes enables joins between tables to be automatically generated in many relational DBMSs.

Aggregation Overview

When a field is selected for aggregation, its foreign key relationships are followed through the database and each linked field is added to a list. The selection of fields can be fine tuned by adjusting the correlation coefficient threshold to disregard foreign key relationships that do not have a high correlation in the database. Once the list of linked fields has been created it can be reviewed and manually edited.

Once the list of fields has been created they are aggregated to form a single attribute definition containing a super-set of the individual fields from which it was formed.

The attribute details are then presented for review and manual editing. Finally, when the attribute is filed, each field in the aggregation selection list is replaced with the attribute definition that has just been created.

```

Current field      [^G(1,-,-,A) field K2      ]
Correlation coefficient [85 ]
Global node(s)    [                          ]      8 items
-----
Name              [Patient_Name              ]
[                ]
[                ]
[                ]
[                ]
[                ]
Data type         Minimum   Maximum   Average   Scale     Sample size
[Char             ] [1       ] [45      ] [12      ] [        ] [2082    ]
Sub-field   Aggr/Edit/Virt?   Attribute name   Data type   Length   Sample size
[           ][ ][           ] [           ] [           ] [           ] [           ]
[           ][ ][           ] [           ] [           ] [           ] [           ]
[           ][ ][           ] [           ] [           ] [           ] [           ]
[           ][ ][           ] [           ] [           ] [           ] [           ]
[           ][ ][           ] [           ] [           ] [           ] [           ]
[           ][ ][           ] [           ] [           ] [           ] [           ]
[           ][ ][           ] [           ] [           ] [           ] [           ]
[           ][ ][           ] [           ] [           ] [           ] [           ]
OK Cancel
    
```

Correlation Coefficient

This field enables you to specify a threshold for the aggregation of attributes using the foreign key correlation coefficients. If this figure is set to 100% then when fields are aggregated only those foreign key relationships that have a correlation coefficient of 100% will be used in the aggregation process.

In practice, you should experiment a little with your own data to determine the optimum setting for your system. If you find that unlike fields are being aggregated together (eg CUSTOMER NAME aggregated with DATE OF BIRTH) then you have the correlation coefficient threshold set too low.

Typically a figure of between 75% and 95% will give the best results, but this can vary greatly depending upon the quality of your information sources and the coding styles used in the applications under analysis.

Global Node(s)

This field contains a list of nodes and fields that are linked by foreign key relationships. The contents of the list will vary depending upon the correlation coefficient threshold selected at the previous prompt.

Enter F1, PF1 or ? to browse the list of fields selected. Items can be manually de-selected from the list if required.

Name

The attribute name defaults to the most frequently occurring local variable name used to represent the attribute. Enter ?? to browse and select from the list of local variables used by this attribute.

If an attribute already exists that you want to use then enter the name of the attribute here or enter F1, PF1 or ? to browse and select from a list of attributes.

Description, Data Type, Lengths and Sub-fields

Refer to the Attribute Maintenance section for details of how to value these fields.

Processing

Once the aggregated attribute has been created all fields in the node selection list from which was created will reference the new attribute.

CHAPTER 12

Entity Creation

Contents

Create

Add/Modify

Delete

Show/Print

Exports

Dictionary Name

Global Node(s)

Entities

Add/remove Prefix

Data Type Mapping

Entity Creation

The physical nodes in a database can be grouped together into logical entities. Each entity comprises of a set of fields from one or more global nodes. In an SQL schema these logical entities would generally be represented as views of physical base tables.

Entities can either be created manually, or semi-automatically using the create and re-create options in the Entity sub-menu.

Create

The create option enables selected nodes to be matched together into sets of nodes with common keys. Each set of nodes are then presented as a candidate entity. Individual nodes can be removed from the set at this point if they are not required. Once the set of nodes has been decided then all the fields within this set of nodes are presented and individual fields can be selectively included or excluded from the entity.

In create mode only those nodes that do not already belong to an entity are considered for matching. In re-create mode all nodes are considered for matching.

In the following example all of the nodes listed would be candidates for matching because they all have the same set of variable keys.

| Node |
|---|
| ^PROD (PRODUCT , SUPPLIER) |
| ^SUPP (SUPPLIER , 10 , PRODUCT) |
| ^SUPP (SUPPLIER , 10 , PRODUCT , "A") |
| ^SUPX ("A" , PRODUCT , SUPPLIER) |

Note that entities cannot be created from an unrefined dictionary as the matching algorithm relies on the existence of common attributes to enable nodes with matching keys to identified.

Add/Modify

The Add/Modify option enables individual entity definitions to be viewed and edited manually. The entity name may also be edited within this screen.

Delete

The delete option enables one or more entity definitions to be deleted from a dictionary.

Show/Print

The show and print options enable entity definitions to be displayed along with details of each attribute that belongs to the entity. Entity definitions can be printed with or without attribute comments.

Exports

Contents

Dictionary Name

Global Node(s)

Entities

Add/remove Prefix

Exports

The export functions enable the contents of the RE/data refined dictionary to be exported in a variety of different formats to third party tools and data dictionaries.

This section describes the functionality of the generic SQL export. Each of the other export functions have similar functionality but may vary from the generic SQL export model to accommodate the particular characteristics of the target system.

The export will operate on either physical global nodes or on logical entities. The output is in the form of ANSI standard SQL Data Definition Language (DDL) scripts. The output can either be displayed on the screen or directed to a pre-defined output device which may be either a printer or a sequential file. In most cases if the DDL is to be imported to a third party product then the output should be directed to a sequential file.

If global nodes are selected then each node will become one SQL table. Each field within the node will become a column within the table. Each variable subscript will be marked as NOT NULL PRIMARY KEY to indicate that it is mandatory and forms part of the primary key to the table. Any fixed subscripts within the node will be ignored.

If entities are selected then each entity will become an SQL table. Each attribute that belongs to the entity will become a column within the table. As with global nodes, any fields that are subscripts will be marked as NOT NULL PRIMARY KEY in the DDL.

The following inputs are required to initiate an SQL export:

Dictionary Name

Enter the name of the refined dictionary from which you wish to export data.

Global Node(s)

Enter the global nodes which you wish to export. Use F1 to invoke the standard node selection dialog.

Entities

In addition to selecting global nodes, or instead of, you may select entities that have been defined for this dictionary.

Add/remove Prefix

It is possible to add to or remove a prefix from the names of the nodes and attributes as they are exported to the SQL DDL. If you wish to add a prefix then just enter the prefix as you would like it to appear.

If you wish to remove a prefix then enter the characters you wish to remove preceded by a minus sign. If any item does not have this prefix then it's name will remain unchanged.

CHAPTER 14

Data Type Mapping

Contents

RE./data Data Types

SQL Data Types

Data Type Mapping

RE/data allows considerable flexibility in permitting users to define their own set of data types and to then define translation tables from their own set of data types to those required by each target data dictionary.

Data types are defined in a global named ^redf.

This is a translation table between RE/data data types and each dictionary's data types. It can be re-defined to correspond to the data types used during dictionary creation.

```
^redf("SQL",RE/d_data_type)=sql_data_type[\implied_length[\scale]]
```

The following table gives an example of how data types might be defined for generation of SQL DDL. By defining data types that are tuned to the particular application's conventions the data types that are used in any particular target data dictionary may be selected to give the best results for that environment.

| RE/data type | Description | Length | SQL data type |
|--------------|-------------------------|----------|---------------|
| M | Mixed | variable | VARCHAR |
| C | Character | variable | VARCHAR |
| N | Numeric | variable | NUMERIC |
| S | Signed Numeric | variable | NUMERIC |
| DH | Date in \$H format | 5 | DATE |
| DT | Time in \$H format | 5 | TIME |
| B | Boolean | 1 | INTEGER\1 |
| Y | Y or N | 1 | CHAR\1 |
| I\8 | Date in YYYYMMDD format | 8 | INTEGER |
| C\8 | Time in HH:MM:SS format | 8 | VARCHAR |

RE/data Data Types

The valid RE/data data types are set up manually in ^redf. For example:

```
>s ^redf("default","B")="B"  
>s ^redf("default","D")="I\8"
```

This table is used for data type validation in the Attribute maintenance function.

SQL Data Types

The SQL data type equivalencies are set up manually in ^redf. For example:

```
>s ^redf("SQL","B")="NUMERIC\1"  
>s ^redf("SQL","I")="INTEGER"
```

Where a data type in one form has an implied length (eg Y has an implied length of 1) and this is translated to a data type of a more generic form that does not have an implied length (eg Y translates to CHAR) then the second piece of ^redf delimited by “\” should contain the implied length.

Data types for other supported export functions are defined in a similar manner to SQL data types.

Installation Procedure

Contents

Restore Globals
Restore Routines
Install RE/m drivers
Configure RE/m
Unique Delimiter
Known Constants

Installation Procedure

RE/data is supplied on one 3 1/2 inch diskettes in MS-DOS format. The following procedure should be used to install RE/data.

Restore Globals

Using a standard global restore utility restore one of the following files:

| M Implementation | File to restore |
|-------------------------|------------------------|
| DTM | RED.GSA |
| MSM | RED.MSM |
| GTM | RED.GTO |
| DSM | RED.GTO |
| M/SQL | RED.GTO |

Restore Routines

Using a standard routine restore utility restore the file RED.RSA.

Install RE/m drivers

Install the RE/data implementation specific drivers using the following command:

```

>d ^reins

R E / m
Reverse Engineering for M Applications
Copyright 1990,1994 George James Professional Services

For support call +44-1932-252568 (Europe)
                or 617-937-9000 (North America)

What type of M system are you running?
  1 Greystone GT.M
  2 Micronetics MSM
  3 DataTree DTM
  4 InterSystems M/SQL
  5 Digital VAX/DSM
M system (1-5)? 3
What type of Operating System are you running?
  1 MS-DOS, OS/2, etc
  2 VMS
  3 UNIX
Operating System? 1
RE/m - installing DTM drivers .....
RE/m - installing on-line reference manual ...
RE/data - installing on-line reference manual ...
Installation complete

```

Configure RE/m

Run ^resetup and select the configure option.

Consult the RE/m reference manual for details of how to fill in this screen. For RE/data the header line selection expression and extraction expression fields are not used, so any valid expression may be entered here. Also for RE/data the active parser table should be REDn.n (where n.n is the current version number) unless you have created a custom parser table.

Set up device types, terminals and printers as required. Consult the RE/m reference manual for full details.

Unique Delimiter

Parts of the RE/data database requires a delimiter that is different from any delimiter used within your application. This delimiter is present as an “@” character, however if your application is known to use this character as a delimiter then you should change this to some value that is not used by your application.

This value can be changed in the Setup option on the RE/data main menu.

Known Constants

Often delimiters within an application are not referenced directly, but via a variable which is pre-set to a constant value. For example, the variable % may be pre-set to “\”. Whenever a backslash delimited node is un-pieced the variable % would be used rather than the literal “\”.

Where such delimiters are known about they should be made known to RE/data using the Known Delimiter setup function. RE/data makes use of this information during the Build phase while it analyses the un-piecing of data from global nodes.

The Known Delimiters function is under the Setup option on the RE/data main menu.

RE/data is now configured and ready to use.