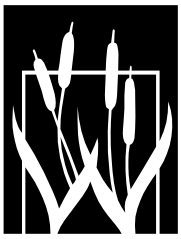

Whitemarsh
Information Systems Corporation

Whitemarsh Metabase Reverse and Forward Engineering

Users Guide

February 2006

Whitemarsh Information Systems Corporation
2008 Althea Lane
Bowie, Maryland 20716
Tele: 301-249-1142
Email: mmgorman@wiscorp.com
Web: www.wiscorp.com

Table of Contents

List of Figures

Tables

1.0 Introduction

The purpose of this Reverse and Forward Engineering Users Guide is to illustrate a specific use of the Whitemarsh metabase. The Whitemarsh metabase is a combination metadata repository and computer-aided-systems-engineering (CASE) environment. The overall domain of the metabase is presented in Figure 1. A more detailed description of the metabase is provided in the document, *Metabase*, which includes its rationale and relationship to the Knowledge Worker Framework. This document can be retrieved from the Free Documents section of the Whitemarsh website, www.wiscorp.com.

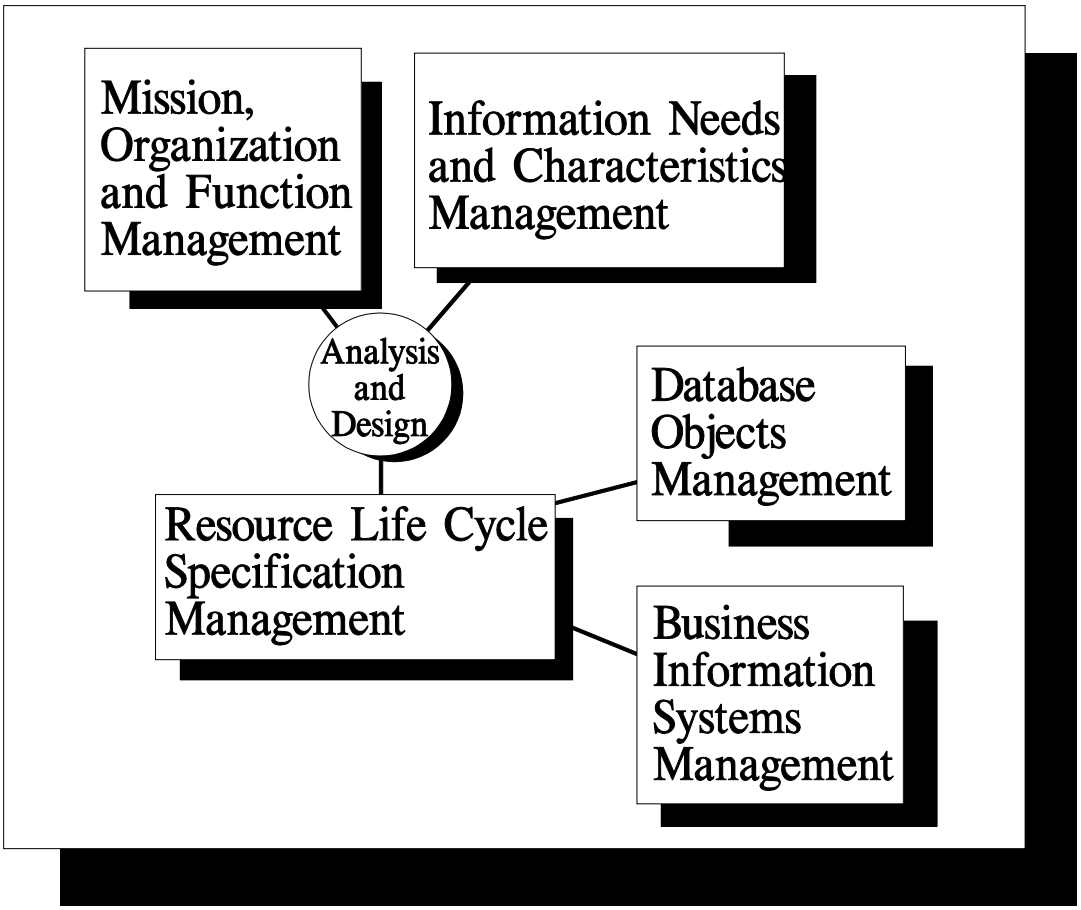

Figure 1. Metabase domain.

This purpose of this guide is to describe how to take a database design from an existing database application of the data architecture class, original data capture, and reverse engineer it into the metabase to create re-usable metadata. The created metadata is then used to "manufacture" another database of a different data architecture class, data warehouse. Because of the architecture of the metabase, there is then a single set of metadata that supports both database designs. In short, an accomplishment of the database concept: define once and use many times, achieves enterprise-wide data semantics.

1.1 Reverse Engineering

This reverse engineering process, depicted in Figure 2, is extremely valuable when trying to create either "intersection" or "union" data models across a set of data models from legacy systems. Under the intersection scenario, each of the legacy data models is imported into the metabase. Then, one by one, their common data structures within tables are "promoted" to the Implemented Data Model layer. Once complete, the Implemented Data Model layer represents

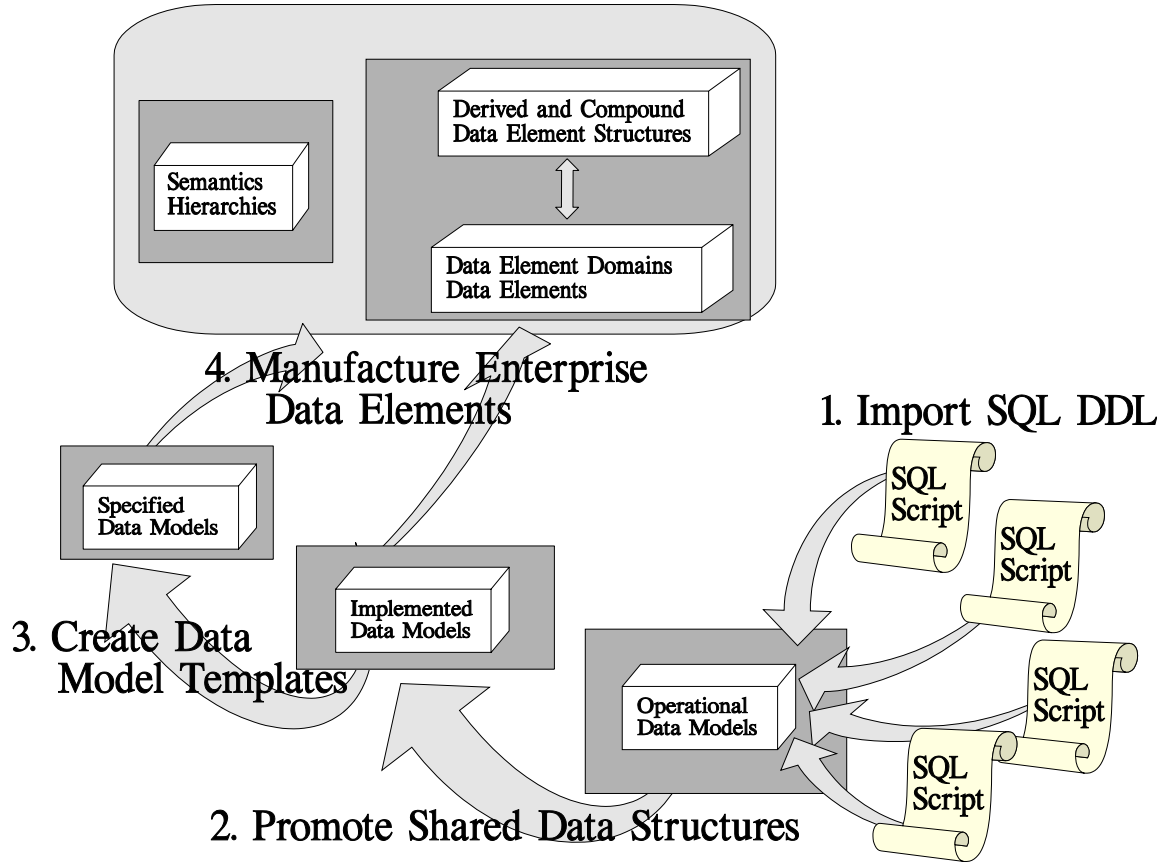

Figure 2. Reverse engineering process flow..

the intersection data across all those legacy models. That intersection data model can then imported back down into the Operational Data Model layer to make a new Operational Data Model. Then, that model can be exported to SQL schema DDL to make the database design that represents the database application that is the "intersection" of the legacy databases. The legacy systems are then able to put and get data to the "intersection database" with the full confidence that the mapping semantics have been pre-engineered.

Under the union scenario, the resulting Implemented Data Model becomes the design of a "Subject Area" database. It is broader than all the feeding databases. Similar to the "intersection" data model scenario, its design is imported back into a different Operational Data Model and then it's SQL schema DDL is used to make the database design of a "union" or subject area database application. The legacy systems are then able to put and get data to the "union database" with the full confidence that the mapping semantics have been pre-engineered.

There are thus three uses for this technique:

- Creating data semantics for enterprise-wide data management
- Creating "intersection data models" across a community of legacy systems that desire to exchange data
- Creating "union data models" that represent a subject area database across a community of systems that are contributing to a more expansive database

The overall process is started by importing the database design into the Operational Data Model (ODM) layer by reading the database's SQL DDL. This ODM design is promoted to an Implemented Data Model layer, and then into the Specified Data Model layer. Within the SDM layer, the set of entities are re-engineered into different and newly created subjects. Then the ISO 11179 Data Element layer including all the upper levels of metadata are created. The reverse engineering ends here.

1.2 Forward Engineering

The Forward engineering process depicted in Figure 3 starts with a requirement to build a data warehouse. The Implemented Data Model layer for this data warehouse is manufactured from the entities within the subjects of the Specified Data Model. Once the Implemented Data Model is built, it is tuned as necessary. It is then employed to create the Operational Data Model. At that layer the SQL DDL is then produced.

The first data model is an original data capture data model. That is, an operational database that is in third normal form. The second data model is a data warehouse with a "star schema" architecture.

The time to accomplish this reverse and forward engineering effort was about 15 staff hours with the reverse engineering part taking the first 12 hours.

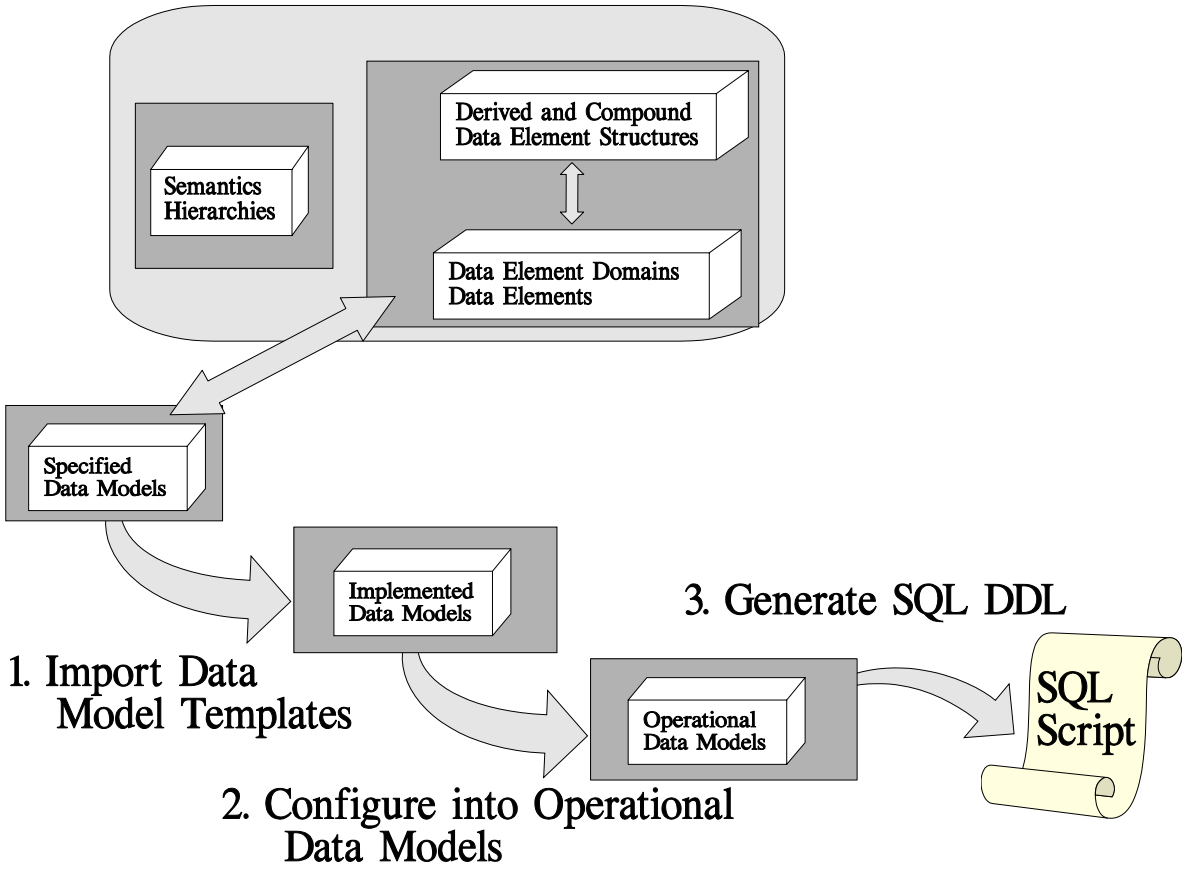

Figure 3. Forward engineering process flow.

1.3 Remainder of the Guide

The process described in the rest of this user guide consists of individual steps. Each includes the step's name, goal, specific steps or actions, and then supplemental description. Overall, there are the following major steps:

- Establishing the metabase work environment
- Reverse Engineering
- Forward Engineering

The case study for this example is an original data capture database for a movies rental store. While the example is clearly trite, and while the "quality" of its design might provoke "violent" arguments among data modelers, it is, what it is, a legacy schema of a production application. And more importantly, it is sufficient to illustrate all the required techniques. The data model diagram for this database is shown in Figure 4. The data definition language file for this example is provided as Attachment 1.

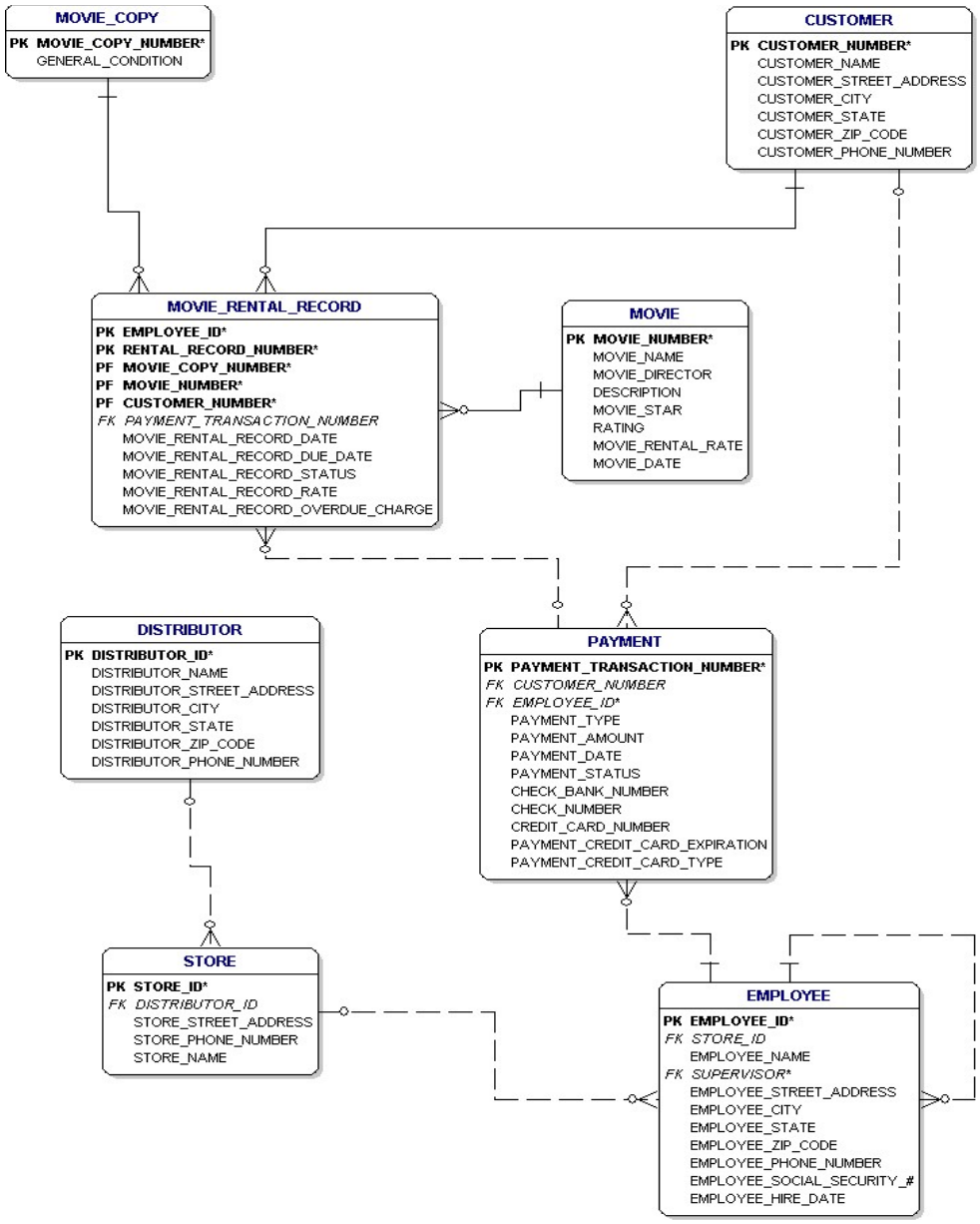

Figure 4. Movie Rental Store Original Data Capture Data Model.

The "story" of the data model is this. There are movies, customers, movie copies, distributors, employees, and stores. There are also movie rental records and payments. One or more stores are served by a distributor. A store may have multiple copies of a movie. Customers may have more than one movie rental record. Customers may make more than one payment, and a payment may relate to more than one movie rental record. Stores may have multiple employees, and an employee may supervise another employee.

Section 2 of this guide are the steps necessary to establish a metabase data instance once the metabase system has been installed. The metabase system getting started guide shows how to install the metabase system.

Section 3 presents the steps whereby a data model represented in SQL DDL is imported into the metabase and is then promoted up two levels so that the first major step of reverse engineering can occur.

Section 4 presents the steps involved in re-engineering the Specified Data Model from a single subject, multi-entity data model into a multi-subject, multi-entity set of data model templates that, in turn, can be used in the construction process of new data models.

Section 5 presents the steps involved in creating ISO 11179 Data Elements that can be employed as fact semantic templates for use in creating entity attributes or table columns.

Section 6 presents the step necessary to create a new database design from the reverse engineering steps accomplished in Section 3, 4, and 5.

An often received and sometimes deserved criticism of “data” folks is that it takes too long to do too little that is ultimately of marginal value. The process in this guide acknowledges this criticism and supports the notion that creating a database design should not be a “start from scratch” effort. Rather, through this guide, and especially the steps in Section 6, the process of creating an entire database design can take from hours to days rather from weeks to months. And, once a well engineered data model is constructed, sophisticated tools can be brought to bear to generate software. The goal is to then turn the criticism into its inverse. That is, that “data” folks can produce a lot in a sort time that is of significant value.

An often asked question is how many database projects are required to be done before there’s a “positive” return on the investment from the approaches described in this guide? The answer is simple: The first. Almost every database design has a number of columns across set of all tables that are essentially the same. Having templates to design them will always make the work go faster. Having the ability to create a running prototype from a database design will always make you more productive. Having the ability to automatically generate attribute, column, or DBMS column names will always make work go faster. Having the ability to standardize value domains will greatly add quality. Experience over the past 30 years with these techniques has always shown a positive ROI on every project in which the metabase techniques have been employed.

1.4 Presumed Knowledge

This guide presumes that you have:

- Downloaded the DBMS Mimer and have it successfully installed
- Downloaded and successfully installed the Whitemarsh metabase system
- Stored the Movies SQL DDL file in a directory for ready access
- Downloaded and read the Data Modeler Architecture and Concept of Operations Guide from the Whitemarsh website
- Down loaded the metabase user guides have acquainted yourself with the Data Element, Specified, Implemented, and Operational Data Modeler's functionality

This guide is not intended to teach the metabase system nor any of the metabase system modules. Rather, this guide is intended to teach the use of the metabase system to accomplish reverse and forward engineering.

This process, reverse and forward engineering is an essential step in building understanding-based interoperable data-based environments. Such an environment, coupled with quality code generators can greatly accelerate the creation and maintenance of information systems.

If you are using a demo version of the metabase system, and if you are using the SQL DBMS, Mimer, then most of the steps in Section 2 are automatically performed during the Metabase's installation process. Actually the only steps that have to be performed are those that connect the automatically installed metabase database instance to Mimer through the ODBC administrator. These steps are described in Section 2.1.

If the Demo version of the metabase is used then an overlay screen is present on every screen that states the metabase system's status: Demo, and the quantity of days and date that the metabase system will no longer become operational. At the end of the demo period the system becomes inactive. However the SQL databases remain intact.

2.0 Establishing the Metabase Work Environment

The steps required to establish the Movies metabase instance are:

- Open Mimer administrator
- Create Movies Metabase database instance
- Create the mbsysdb database instance
- Open the metabase system
- Open the Admin Module of the metabase system
- Set up access to the Movies Metabase metabase
- Run the "AutoData" and "Data Types" bulk loader

2.1 Open Mimer Administrator

Note: all the examples in this guide are accomplished through the SQL DBMS, Mimer. It can be obtained from www.mimer.com. If another SQL DBMS is employed as the "engine" for the metabase system, then while all the steps will essentially be the same, they would have to be "tuned" for the particular DBMS.

This step presumes that you have accomplished the Metabase Quick Start manual steps: Steps 2.1 and 2.2 . If you have not done that then please do so and then restart this guide. Assuming that you have accomplished these steps then open the Mimer Administrator. Do this by starting the Mimer Administrator program, that is, execute the file, mimadmin.exe. This file should be easily available through the "Start then Programs then Mimer SQL Engine 9.2 " sequence.

Once the Mimer Administrator starts, a window like Figure 5 comes up. You will note in this window that there is an overall Metabase program window that contains all the metabase module executables and a number of other useful executables. This was made by creating a folder, for example, Metabase, and then copying into that folder shortcuts to all the items. The Mimer Administrator (mimadmin.exe) was just executed.

This screen shows that there are already two metabase database instances known to ODBC and to Mimer. They are: metabase and mbsysdb. The mbsysdb database is the metabase system's administrative database. It keeps track of users that are currently logged into one or more metabase database instances. The other database, metabase, is the generic name for an instance of a metabase database instance. There can be many other metabase database instances as will be accomplished in the steps that follow.

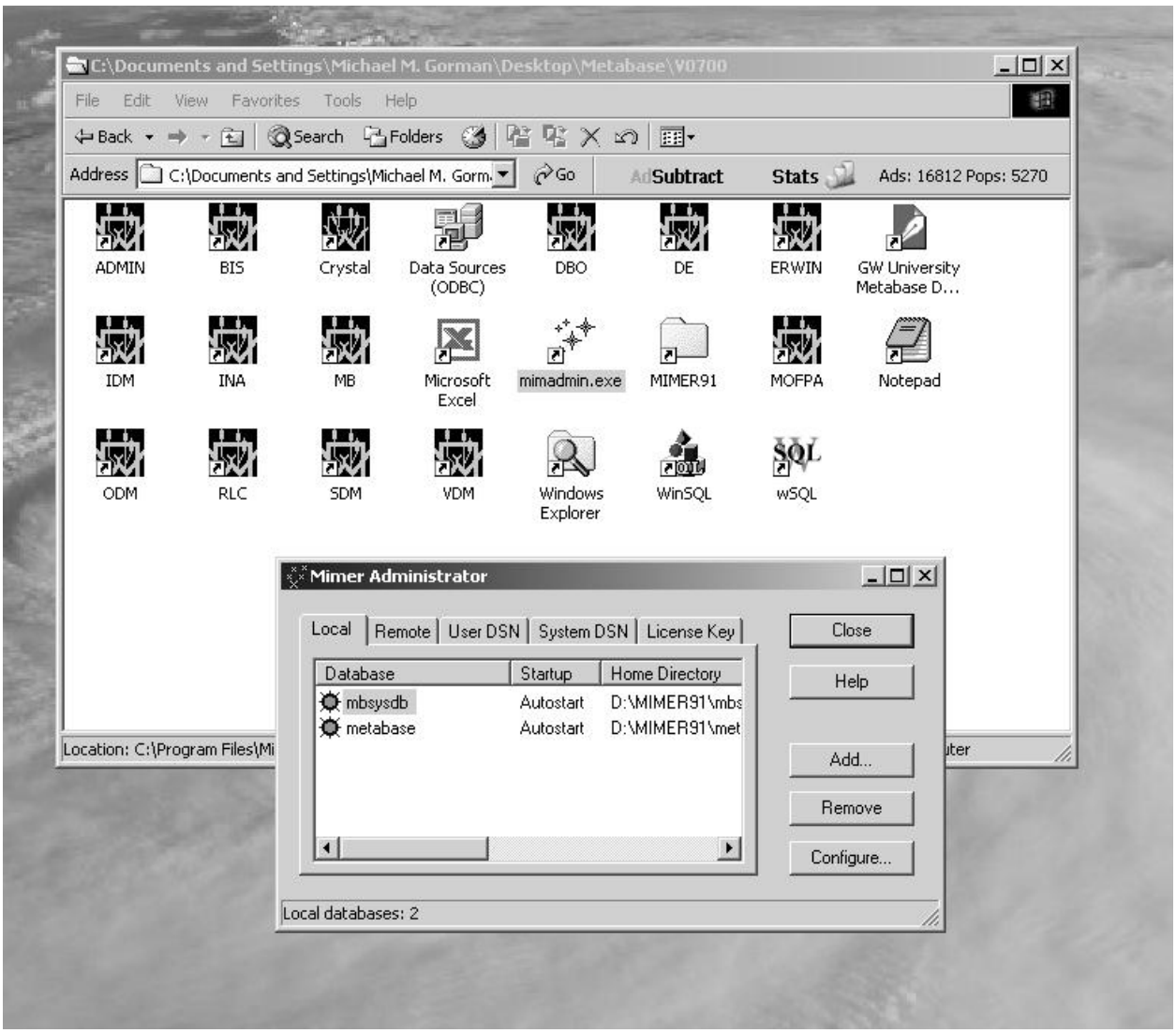

Figure 5. Mimer Administrator Screen.

2.2 Create the Movies Metabase

Starting with the Figure 5 screen, Press the Add button, and an overlay appears, as shown in Figure 6, providing the ability to create a new local database definition. In this screen, add the MoviesDatabase (note that there is NOT a space separating the two words). Then, identify what is desired as the home directory. Choose the option to check all pages versus check just the index pages. Finally, identify to Mimer the maximum quantity of concurrent users. Here, 10 was selected. Finally, press the Apply button. Mimer will then ask if you wish to create "databanks." Click "yes." Note also that the default system administrator user name is SYSADM. This cannot be changed.

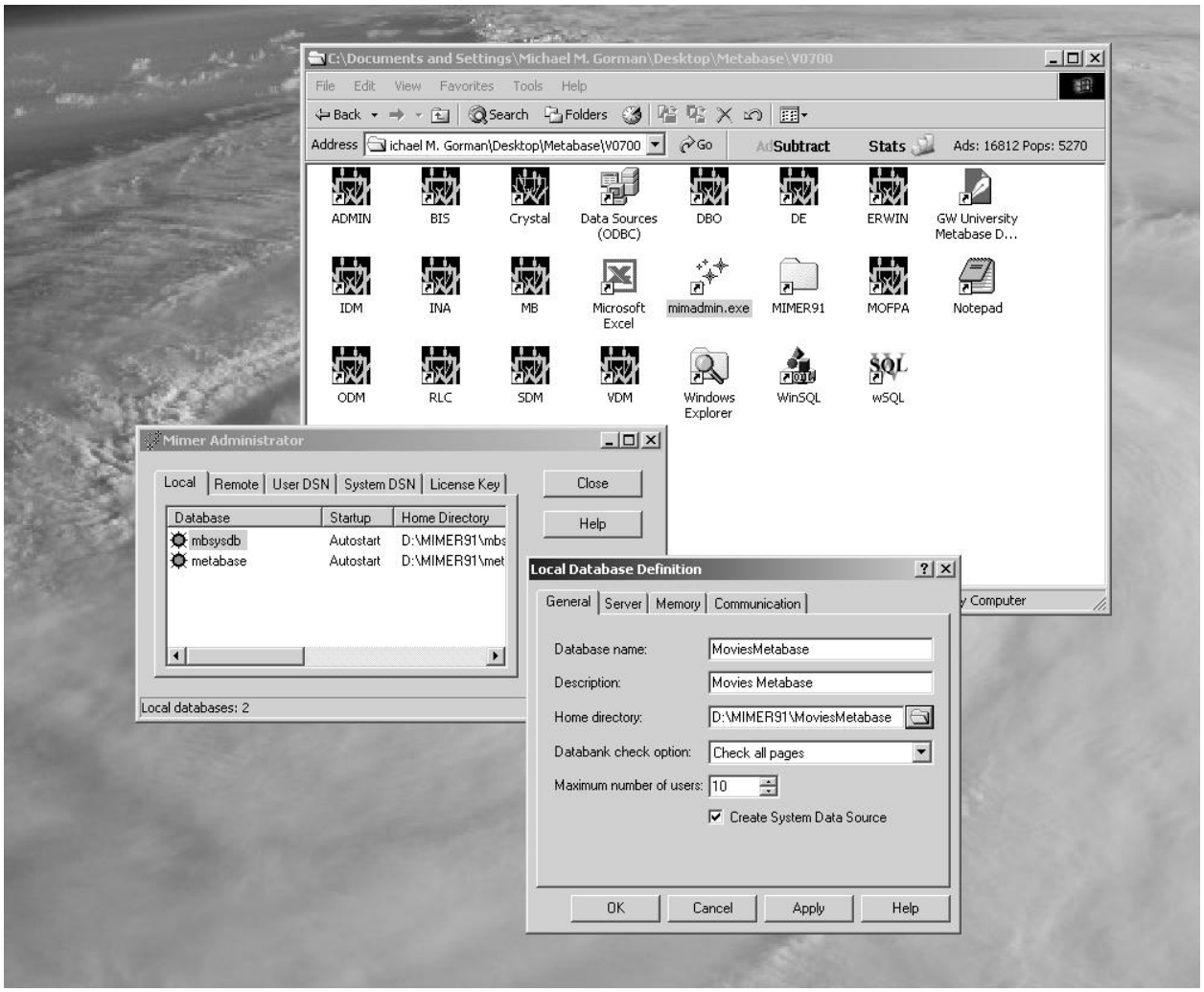

Figure 6. Mimer local database definition screen.

Mimer then provides you the opportunity to put the Mimer database files onto separate disk drives for performance and security reasons. Figure 7 presents the options. In this example, choose what ever is appropriate. Once chosen, also indicate the quantity of pages associated with each file. Each page is a 2K file. If you put the files on other drives, Mimer automatically makes parallel named directories.

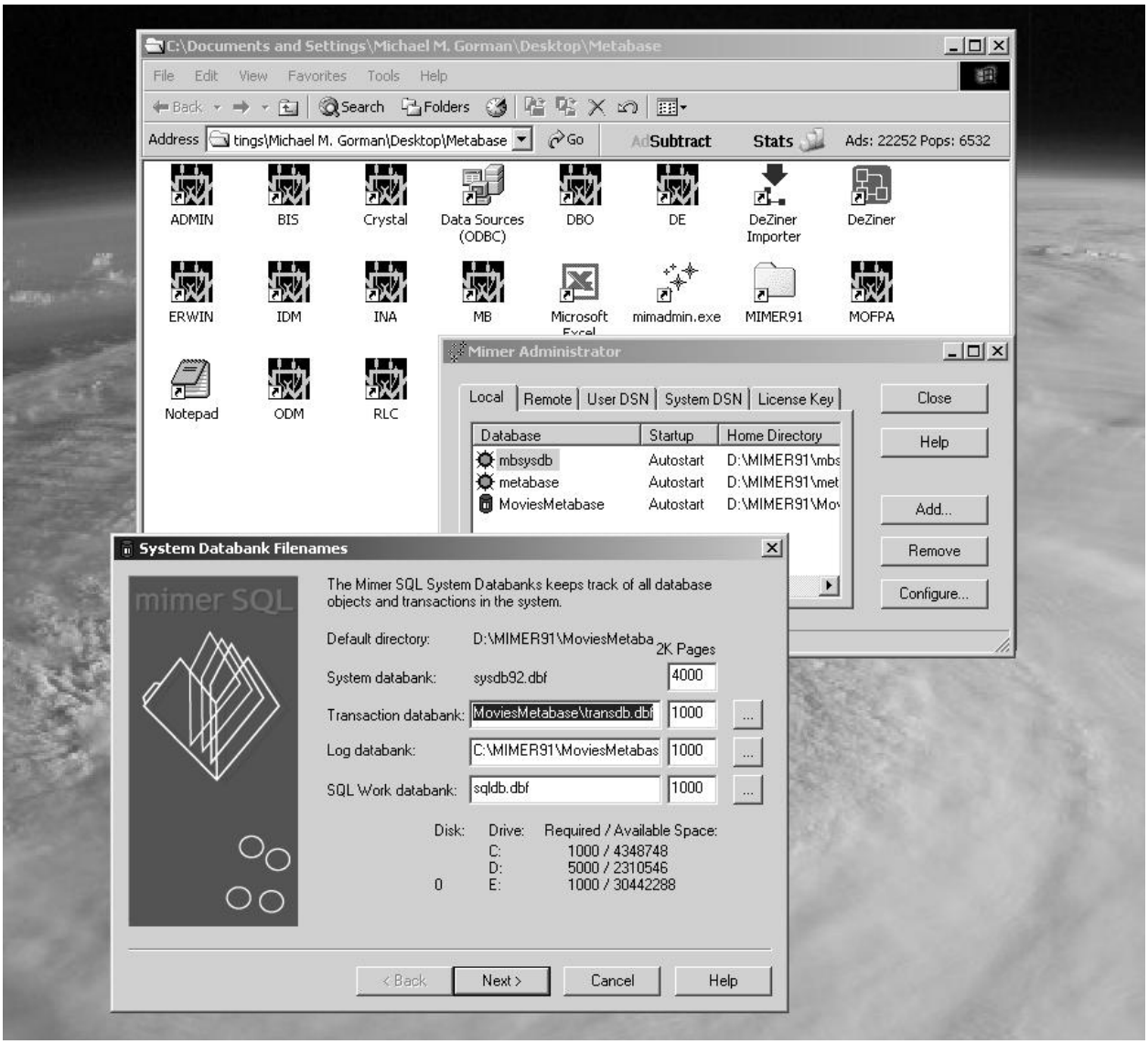

Figure 7. Allocation of drives and page quantities for Mimer database.

Figure 8 is the password screen. Note that you have to enter the password twice. That's because of a requirement for a positive affirmation. If you forget or lose the password then you are completely out of luck. Let that be restated. **If you forget or lose your Mimer Admin databank password then you are completely out of luck.** A question in this regard was posed to the developers of Mimer. The question was: "Is there some super-secret, highly-classified password cracker for a Mimer database?" The answer is:

> *No, the password is stored in the Mimer data dictionary in a one-way enciphered format. The algorithm is based on a paper by H. D. Knoble at Penn State*

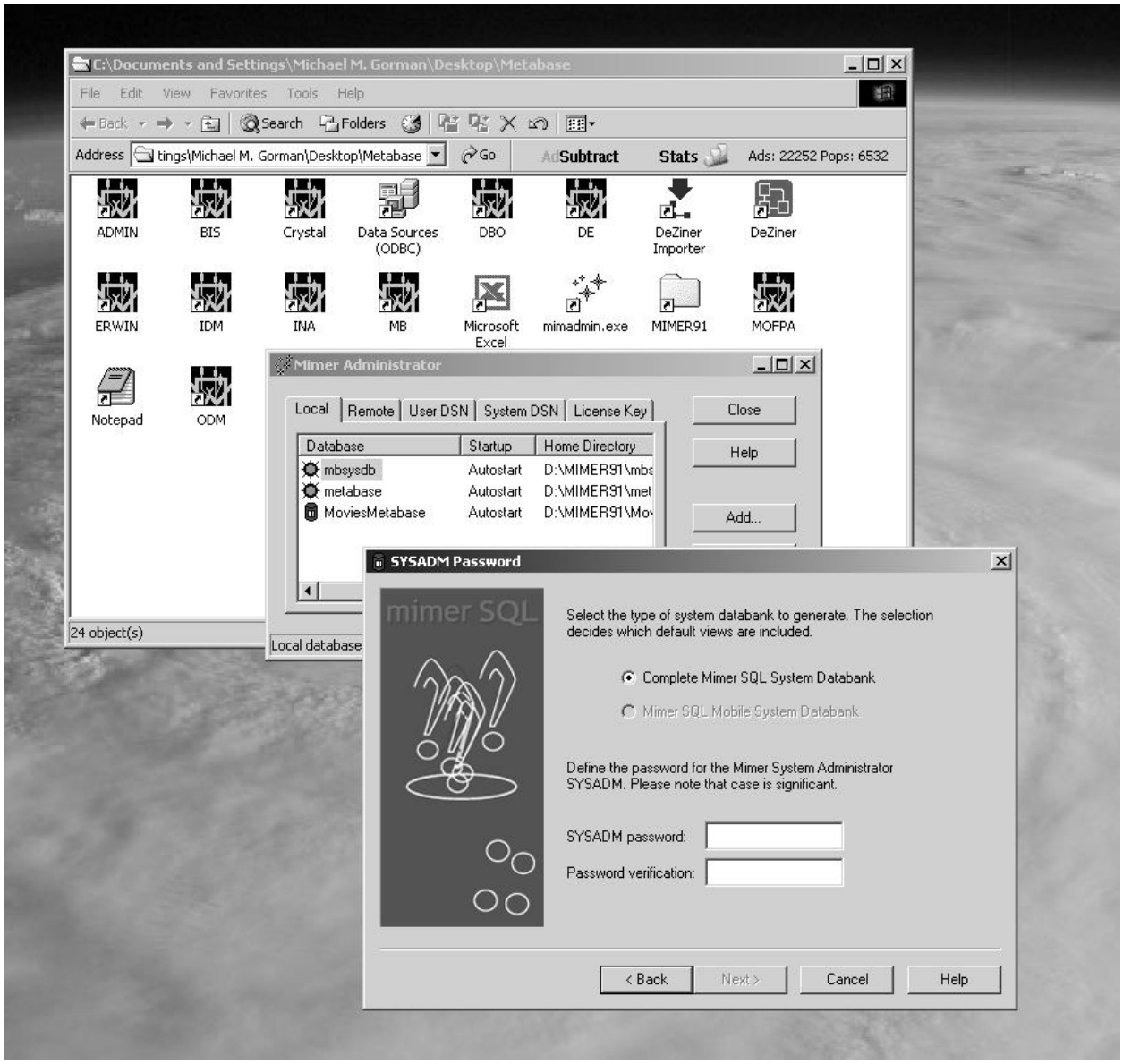

Figure 8. Mimer database password screen.

University, and Mimer's implementation has even been approved by the Swedish equivalent to FBI!

Consequently, I repeat for the last time, **If you forget or lose the password then you are completely out of luck.**

Once the database is created Mimer asks if you want it "started."

Now, execute the WSQL utility that is provided by Mimer from their website. The dialog in Figure 9 merely requires that you enter the user name (SYSADM is the mandatory default name) and password that was previously established. Enter that and proceed on. Then, as shown in Figure 10, enter the Mimer command, "Create Databank MoviesMetabase;" then press the "!" icon to execute the command. This causes Mimer to create a full MoviesMetabase with all the schema information tables. It is "empty," of course. Created but empty.

The next step is to execute the Movies SQL DDL file. Accomplish that by executing three distinct SQL DDL files. This is shown in Figure 11. They are in the C:\Program Files\Wiscorp\Metabase\DDL directory, given that this is where the metabase system was installed. Once the file is located, as presented in Figure 12, press the "!" execute button. Note, also please check the "Continue" radio button so that the command stream just executes through to the end. When finished, there will be a metabase system administrator database.

The first SQL DDL file contains all the Create Table, and Primary and Unique Key statements. The file, MB601_MimerSQL.SQL should be completely generic for all SQL DBMSs. The second file, MB603_MimerSQL.SQL are the auto-incrementing primary key sequence statements. Each SQL DBMS may have a different SQL DDL for this. Mimer's is ISO/ANSI SQL standard compliant. Hence that is why Whitemarsh uses Mimer. The third file, MB604_MimerSQL.SQL, are all the foreign key statements. This file too should be generic across all SQL DBMSs.

You can explore that just built MoviesMetabase with WSQL or any other suitable ODBC SQL utility. It is devoid of data but it exists.

Figure 9. Activating the Mimer SQL utility, WSQL.

Figure 10. Entering the Create Databank Mimer command.

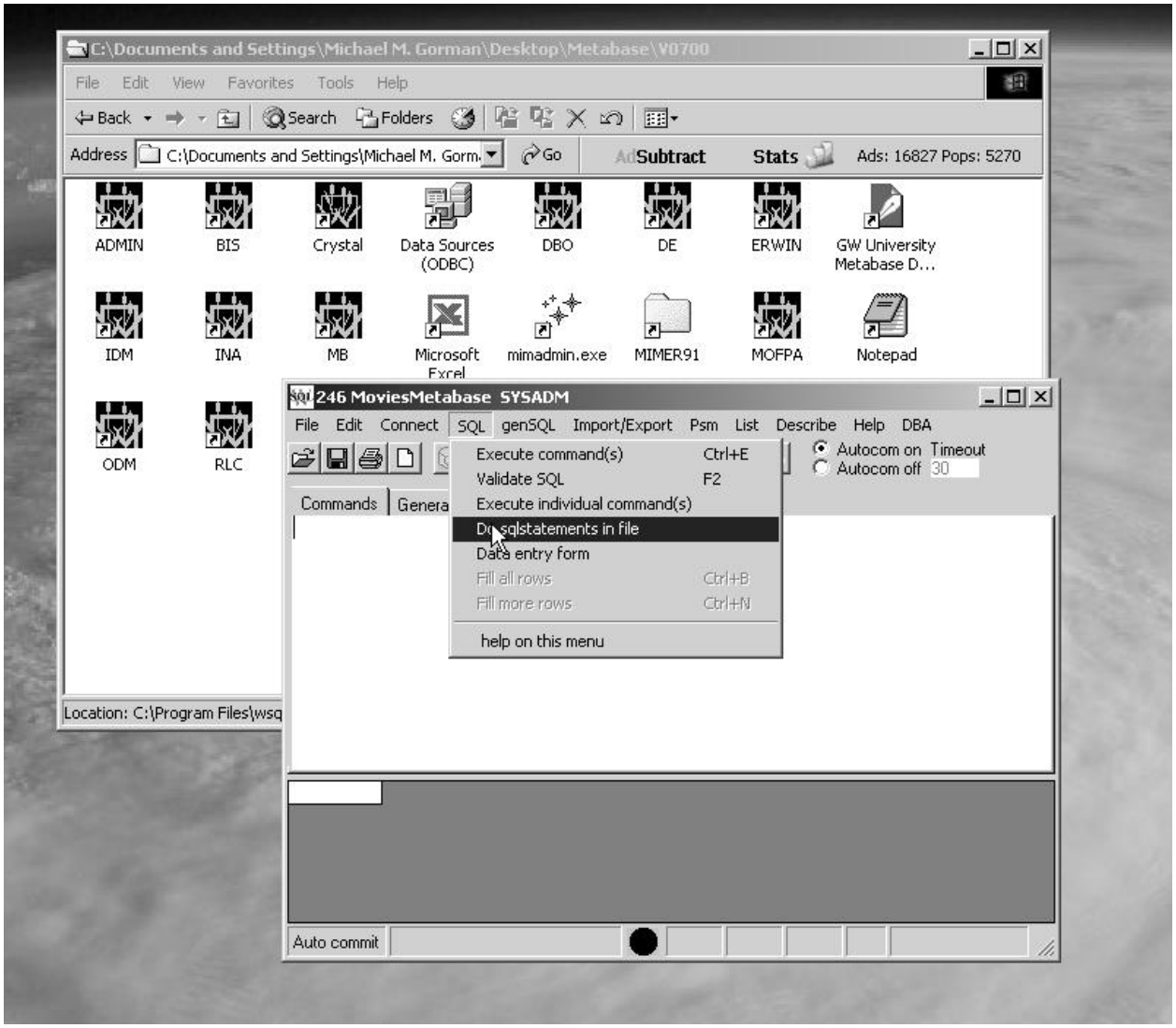

Figure 11. Executing a SQL statements file.

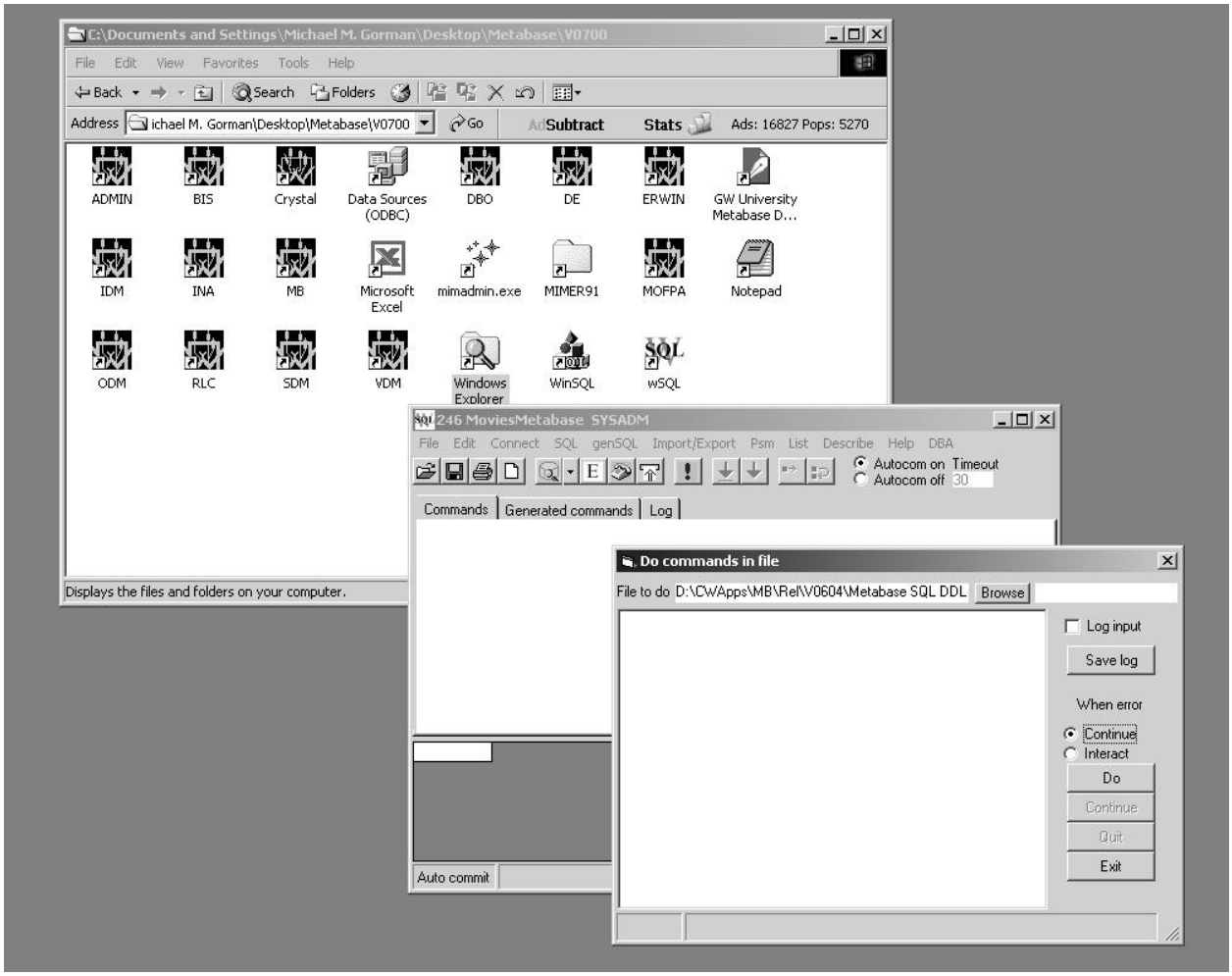

Figure 12. Execute the SQL DDL file.

2.3 Create the mbsysdb Database

The metabase system administrator's database is created the same way as the Movies Metabase. It however only has one SQL DDL file, not three. See the Whitemarsh Metabase Getting Started Guide for more information.

2.4 Final Steps

Once the Movies Metabase instance is created, there are two more steps before "real" work can begin. These steps are:

- Establishing a metabase connection to the Movies Metabase
- Loading the "Unknown" values and then the Data Types values.

2.4.1 Establishing a Metabase System Connection

Figure 13 presents the screen for establishing a metabase system connection. To get this screen, execute the metabase system module, mbAdmin.exe. This module will immediately present you with a user name and password screen. You need to log into this module Accomplish this by entering the administrator user name and password. By default, they are both "sysadm" and "sysadm." The administrator can change both. Additionally, by default, the metabase is provide with two users. Michael M. Gorman, and Matthew G. Gorman. You can of course change those names to the names of the metabase administrator. The Administrator module itself should be secured from general use. The administrator names and passwords enable administrator functionality.

To then know the list of metabase instances, press the DataSet button. Because you have logged in as the administrator, a window of available metabase database instances that it "knows about" is presented. If the Movies Metabase is not there, then just close the window. Then, press the Dataset button. You will be presented with a similar screen as the first select window. If you are using the Demo version of the metabase the Insert, Change, and Delete buttons are not visible. In a production version the Insert, Change, Delete buttons are visible. Press the Insert button. You are then presented with a screen like the one in Figure 14. In the data entry areas enter the information about the Movies Metabase instance. That is, its name, description, the DSN Host Name, User Name (SYSADM), and then the Password that you dare not have forgotten. At that point, click OK and close the window. At that point, the new connection for the metabase is available to be selected. Select it, and then close the window.

The user name and password that has just been entered is the one required by the SQL DBMS as it creates and accesses a metabase database instance. All metabase modules access the SQL DBMS through ODBC. Thus, all metabase modules must pass a user name and password to the SQL DBMS. This screen is the one where that SQL DBMS user name and password is made known to the metabase modules. This SQL DBMS user name and password is not the same user names and passwords that metabase users employ to access the metabase system. The metabase administrator user guide provides full documentation on this topic.

WARNING: Just because you have created and selected a metabase database instance does not mean that it is valid. To determine the validity of the information that you have created, and which, by the way, is now in the mbsysdb database, you must attempt to access that metabase

instance. A quick check is accomplished by pressing the Autodata button. If you receive an error message to the effect that the data source name was not found then there is something wrong with the DSN Host Name, the User Name, and/or the Password. When you get this type of error, the mbAdmin program terminates. Re-start it so that you can fix what you have previously entered by pressing the Dataset button and then the Change button and make your fixes.

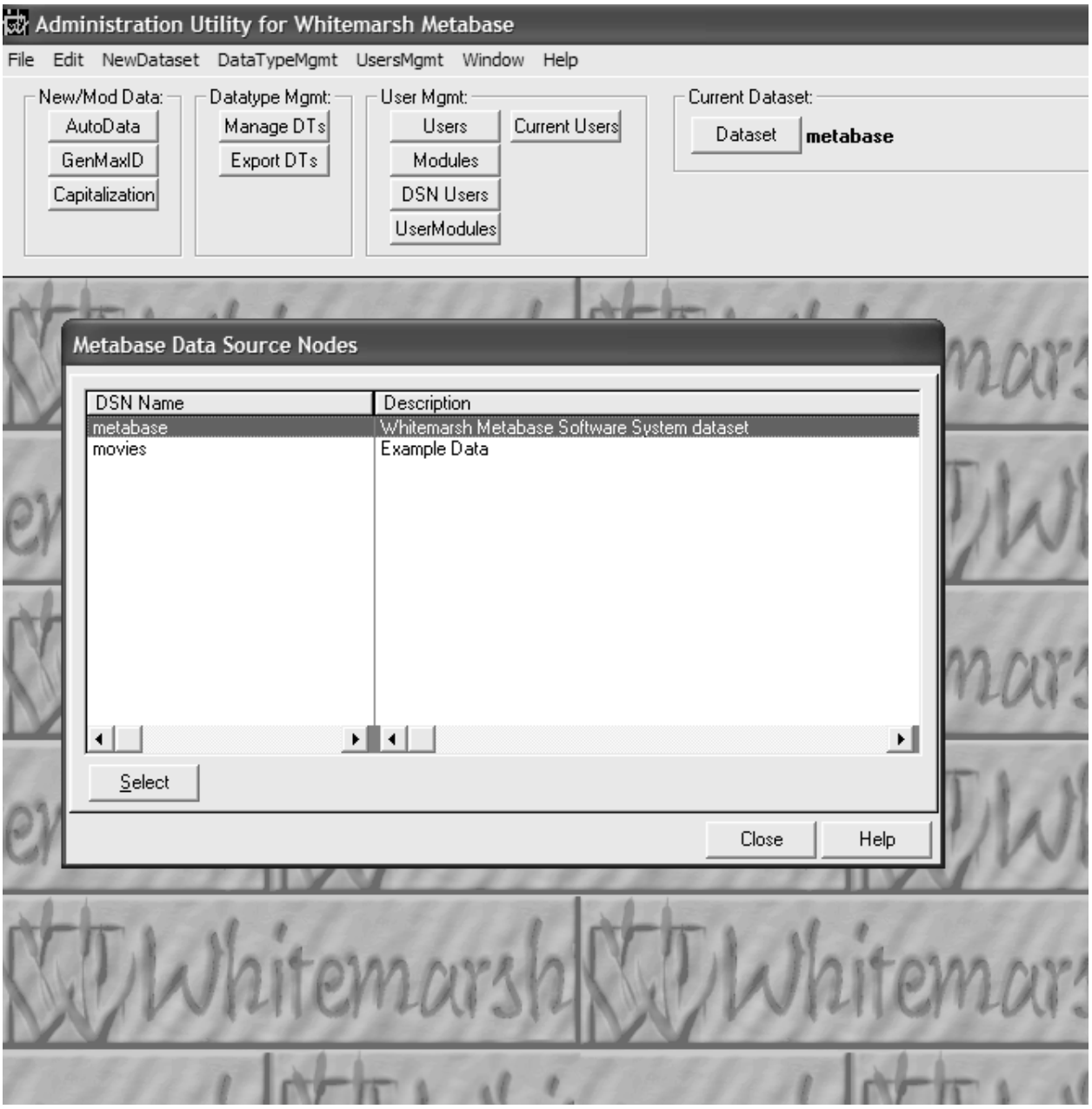

Figure 13. DSN Select screen with Insert, Change, and Delete buttons.

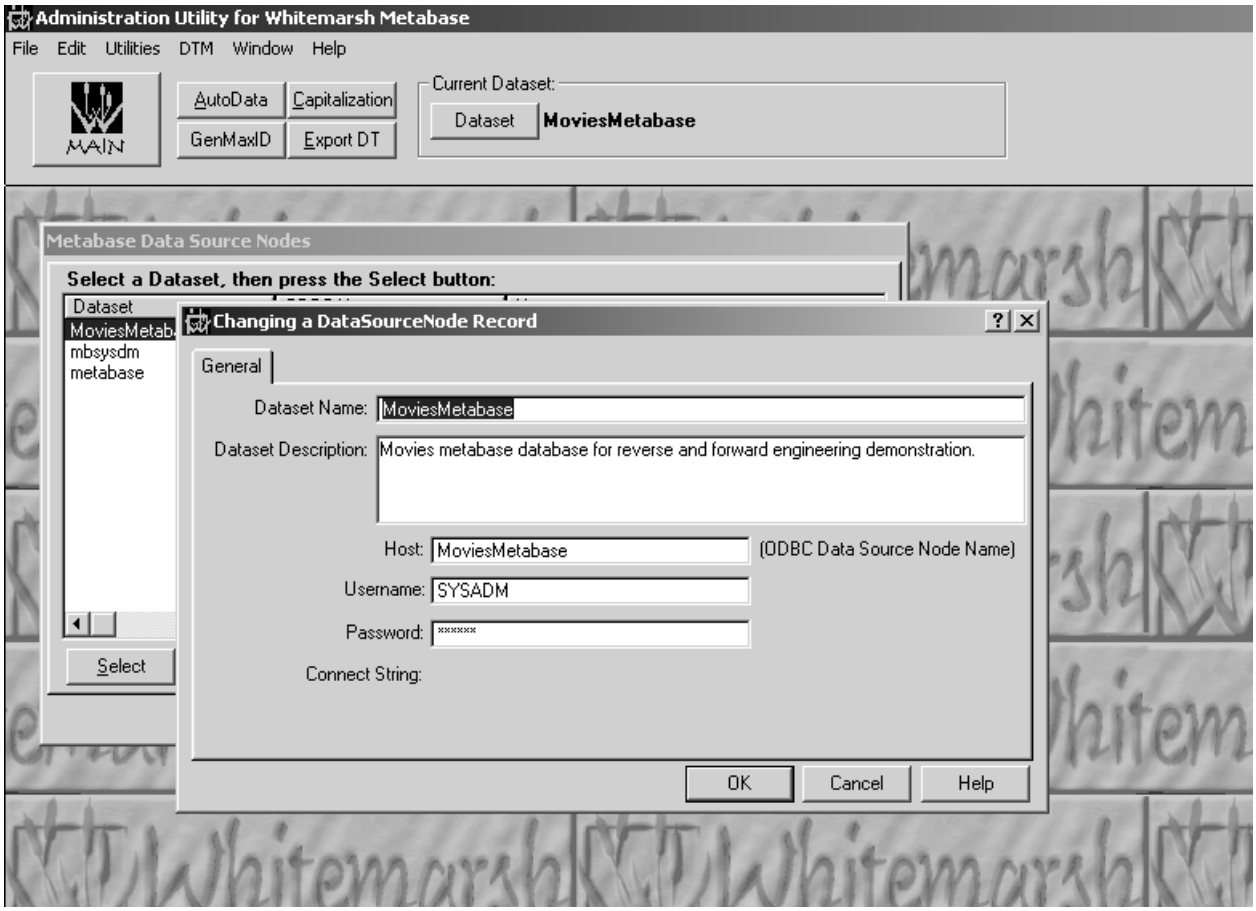

Figure 14. Metabase connection to database update screen.

2.4.2 Loading the "Unknown" Values and then the Data Types values

The last step before starting "real" work is the creation of Unknown and Data Type values. The metabase supports very strong referential integrity. Simply, that means no orphans. But if you are doing reverse engineering and do not have all the upper layers to the make connections, what do you do? The database answer is "Null." While that's a simple answer, every SQL language based DBMS treats Null values just a bit differently. Consequently, the metabase system has engineered a solution. It is that the null value is "Unknown." That value is then pre-loaded into almost all the metabase database tables as the very first (and also sometimes second) row. The database key value for Unknown in the metabase is either "1" or "2." If the metabase system had chosen either DBMS generated value for High Values or Low Values, then if any DBMS had a really different value for that, then there wouldn't be interoperability among metabase database instances. So, the solution chosen was simple, reliable, and predictable.

To generate these Unknown values, the metabase administrator module is executed. The DSN for the specific "new" metabase database instance is chosen, and then the AutoData button is pressed. Figure 15 presents the screen used to execute the Unknown value generator. Press the Create Default/Unknown Data Records "build" button. After some time, a finished message will appear.

The next step is to create the default SQL and DBMS data types. In the metabase there are three levels to the data types: ISO 11179 Data Element level, Implemented Data Model level, and the Operational Data Model level. Figure 16 illustrates the data types hierarchy. This hierarchy is "seen" from the Oracle DBMS and its data types which are on the left column. These are operative at the Operational Data Model level. Each of these data types is derived from an SQL data type that exists at the Implemented data model level (the middle column). These are in turn derived from the Value Domain Data Types from the ISO 11179 data element layer (the right column). In Figure 16, the data types are sorted alphabetically so that the derivation is easily seen from right to left.

There are a number of data type process buttons on this screen. These are all explained in the Operational Data Module user guide.

Generating the data types is a two step process. First the data types loader file must be found. Do that by pressing the Select Data Types file button. Use the Windows browse feature to find the data types file. Most commonly, it will be in the C:\Program Files\Wiscorp\Metabase\REFDATA directory. It is an ASCII file called DataTypesFile.txt. It specifies the data type data for each level of the metabase (Data Element, Implemented and Operational) and also whether the data type has precision and/or scale. The "type" of data types is the second field, that is, VDT for value domain data type, SDT for SQL data type, and DDT for DBMS data type. This file also contains the foreign key values (represented as the parent data type name) back from the SQL Data Type to the Value Domain Data type, or from the DBMS data type to the SQL Data type.

Once the file is found, press the build button. After a bit a finished message will occur. At that point, your Movies Metabase database instance is fully created and is ready for "real work."

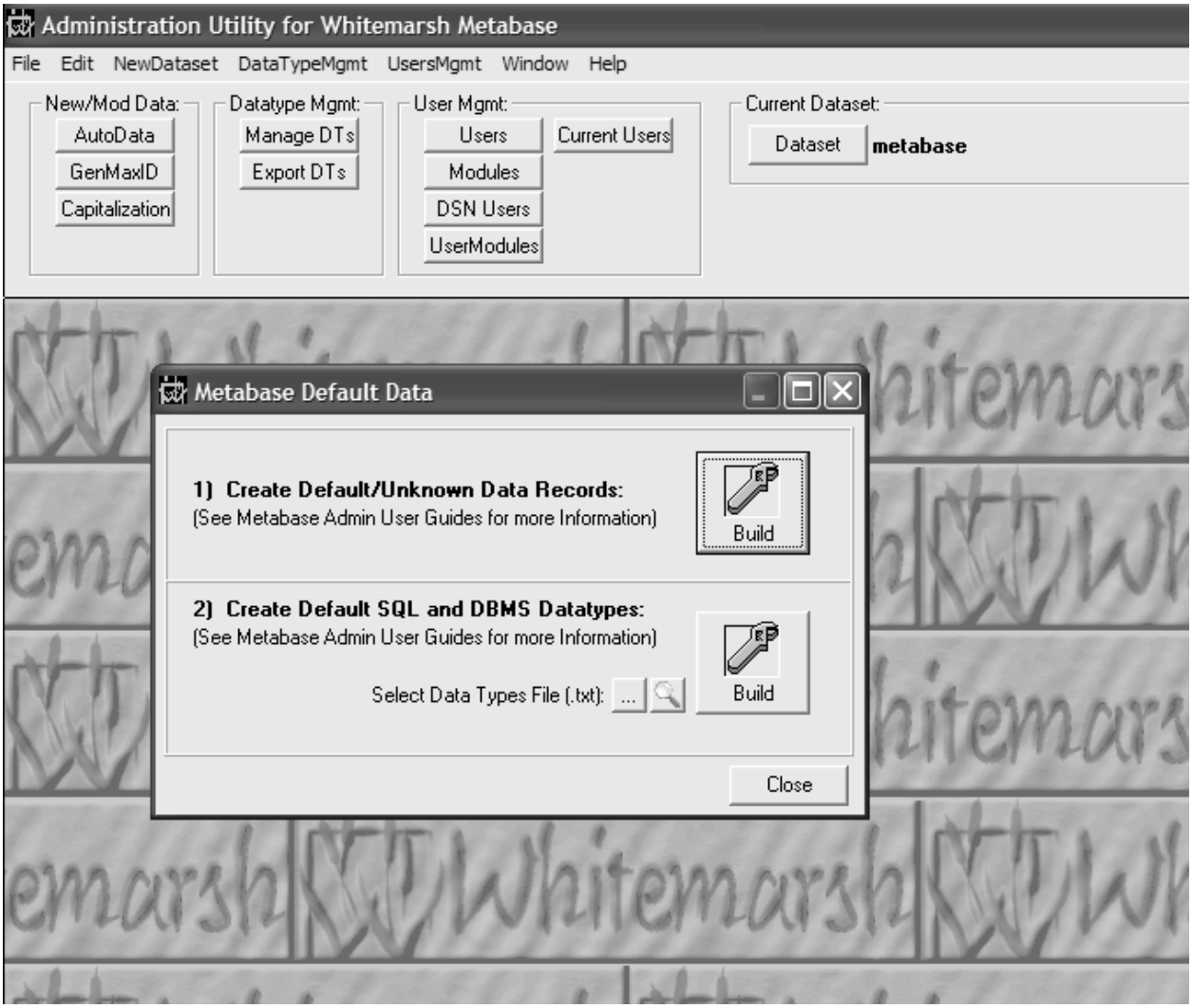

Figure 15. Admin module Unknown and Data Types data generator.

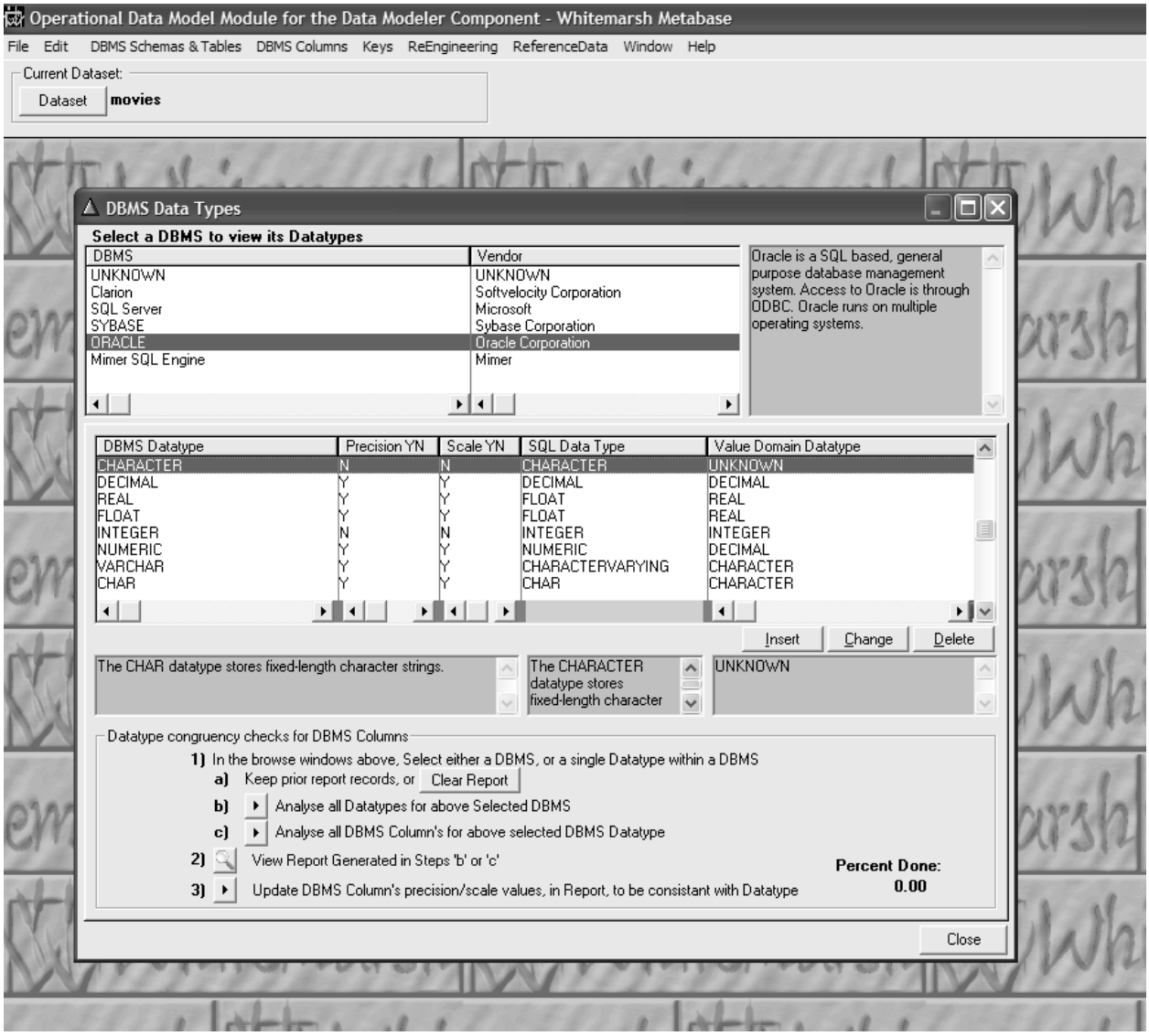

Figure 16. Data types hierarchy.

3.0 Reverse Engineering

The reverse engineering process through which metadata is created, from the Operational to the Implemented to the Specified and finally to the Data Element models, consists of the following major steps:

- Import a SQL database's DDL into the Operational Data Model
- Promote the Operational data model schema to the Implemented Data model
- Promote the Implemented data model schema to the Specified Data Model
- Re-engineer the Specified Data Model
- Create Data Element Model metadata

3.1 Import a SQL database's DDL into the Operational Data Model

The process of importing a SQL database schema into the metabase consists of:

- Creating the SQL DDL text file
- Creating the DBMS data types
- Creating data architecture classes
- Importing the SQL DDL

When the SQL DDL is imported it will be scanned and, if acceptable, will create data records for the following Operational Data Model tables:

- DBMS Table
- DBMS Column
- DBMS Table Primary Key
- DBMS Table Primary Key Column
- DBMS Table Unique Key
- DBMS Table Unique Key Column
- DBMS Table Foreign Key
- DBMS Table Foreign Key Column
- DBMS Table Secondary Key
- DBMS Table Secondary Key Column

The data model diagram for the Operational Data Model is presented in Figure 17. Each "arrow" represents a one-to-many relationship. The "story" of this data model is that for every database there may be one or more DBMS schemas. A DBMS Schema is governed by a DBMS. A database belongs to a database architecture class. For every DBMS schema there may be one or more DBMS Tables. Every DBMS table has at most one primary key. Every DBMS table may have one or more DBMS columns. Every DBMS column has a DBMS data type. Every DBMS table may have one or more foreign keys that represent the relationship between the DBMS table to which the foreign key belongs, and another DBMS table that is associated with the

foreign key's related primary key. A DBMS table may have additional unique keys that are called Candidate Keys. Finally a DBMS table may have one or more non-unique keys that are called, Secondary Keys. Every key, that is, primary, foreign, candidate, or secondary has one or more DBMS columns that constitute the values for the key. Additionally, those DBMS columns exist in a specific sequence within the key.

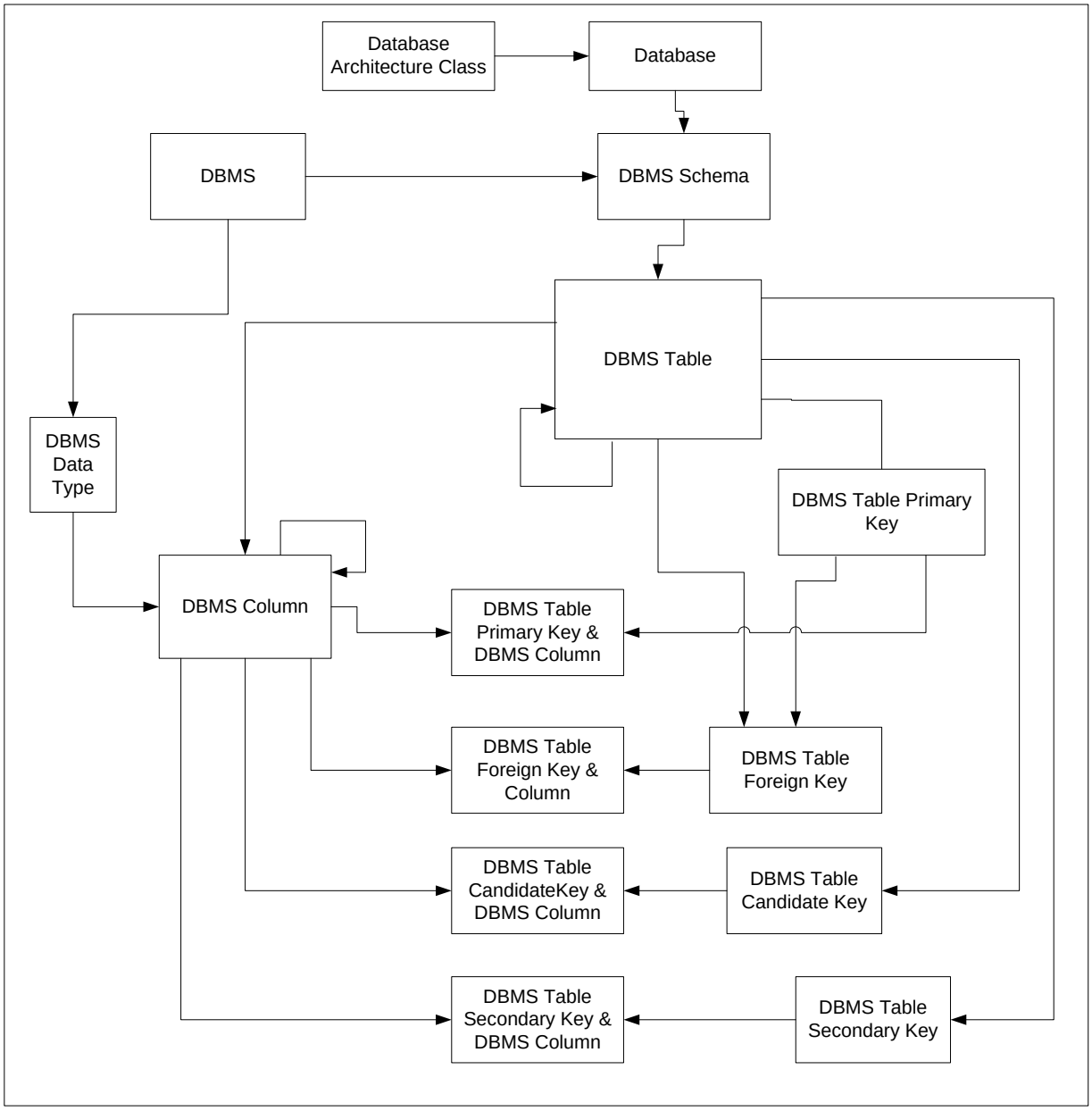

Figure 17. Operational data model diagram.

3.1.1 Creating the SQL DDL Text File

It must be stated at the outset that the metabase will not import every possible variation of SQL DDL. There are too many variations on the theme. Focus has been on ISO/ANSI standard SQL DDL. Thus, what the metabase will import are the fundamental SQL constructs dealing with

- Table
- Column
- Primary Keys in both contained table and Alter Table formats
- Unique Keys in both contained table and Alter Table formats
- Foreign Keys in Alter Table format.

Work is underway to import various default and check constraints.

3.1.2 Creating DBMS Data Types

The most common problem in importing SQL DDL is data types. It is a very good idea to carefully review all the data types within a SQL DDL file and compare them to the DBMS Data Types. If any data types in the SQL DDL file are missing from the DBMS Data Types, the SQL DDL will not completely load. Error messages will be provided, and a log of all the actions is able to be generated. What must then be done is the deletion of the partially loaded Operational Data Model, fix the offending SQL DDL statements, and then re-process the SQL DDL file. This process is too tedious to just iterate through, so it is highly suggested that an examination of the SQL DDL file be performed to ensure that has all its column data types pre-exist in the DBMS data types. Figure 16 presents the DBMS data types screen.

If a DBMS data type is missing, then press the Insert button. Figure 18 is presented. In this particular screen, what is originally presented is the “Insert” screen. The first step is to pick the SQL data type from which this DBMS data type is to be derived. That Select SQL Data Type screen is shown. Pick the appropriate one and press Select. Then close that window. The next steps are to provide the data type a name, enter a picture class (required for Clarion for Windows and described elsewhere), and to indicate whether the data type can have a precision and a scale. Finally, add a description for the data type. If the process of adding DBMS data types is accomplished before loading the SQL DDL, then the schema almost always loads in the first pass.

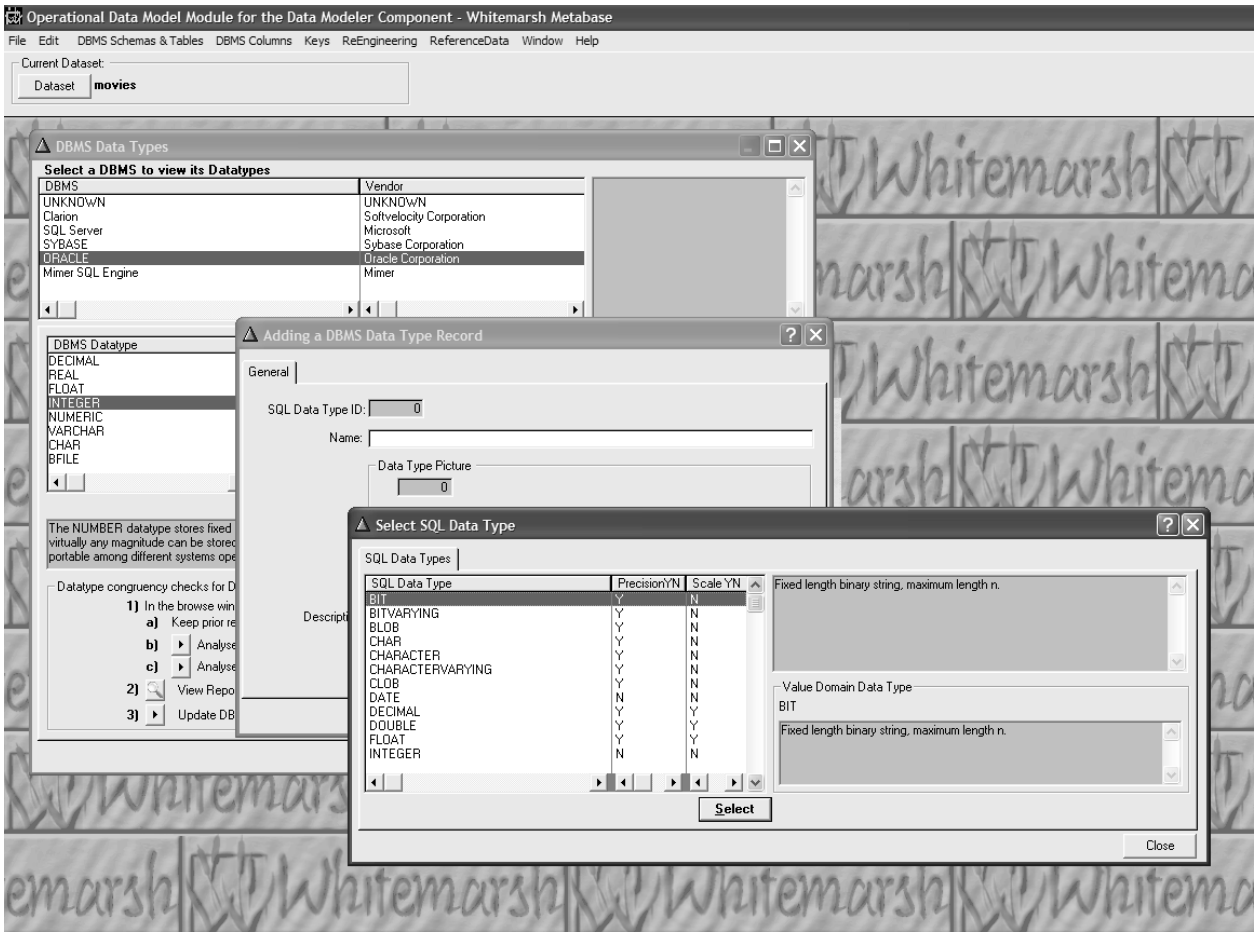

Figure 18. Data types insert screen.

3.1.3 Creating Data Architecture Classes

Databases can be seen generally belong to distinct data architecture classes, which are:

- Reference data
- Original data capture
- Transaction data staging area
- Subject area databases, and
- Data warehouses

The **reference data** architecture class represents data that form the critical characterization and discrimination characteristics of data from within the entire set of business facts. Included, for example are genders, city names, state names, all codes, and like. Ideally, all reference data would be exactly the same across all the other four data architecture classes. Realistically, however, different agencies and providers have different reference data value versions for the

same reference data and different data value versions across time. All reference data must be managed centrally and then distributed in so far as it is possible to all databases of the other data architecture classes.

The **original data capture** data architecture class represent the actual databases that reside within organizations within an enterprise and may provide data to other database classes. These database types are often called OLTP databases because they support on-line transaction processing.

The **transaction data staging area** data architecture class represents the data extracts from the original data capture databases that are then, if necessary, modified to the required semantics for any of the other database architecture classes. Any interface between any database architecture class may proceed through a transaction data staging area.

The **subject area** data architecture class represents the subject-based integrated databases of data that, in turn, support some measure of analyses and reports, analysis results retention, and supports the generation of other classes of databases, that is, data warehouses.

The **data warehouse** (wholesale and retail) data architecture represents the transformed and likely redundant sets of data that serve special reports and analyses. The key set of differences between wholesale data warehouses and retail data warehouse is one of volume, duration and specialization. Data mart data warehouse designs are commonly created along the lines of "star schemas" or "snow-flake schemas," and when compared to wholesale data warehouse have smaller volumes, shorter durations, and are more specialized.

Each of these data architecture classes have distinct data modeling, data normalization, and data update characteristics. These data architecture classes and their characteristics are described in other Whitemarsh documents.

The process of creating a data architecture class involves selecting the Data Architecture class item from the Reference Data menu item in the Operational Data Model metabase module. A list of data architecture classes appears. To add one, press Insert. At that point, a screen like Figure 19 appears. Add the appropriate definition, press OK, close the menu, and move onto the next step. In this case we are entering the data architecture class, Original Data Capture because that is the class of the database that will be imported.

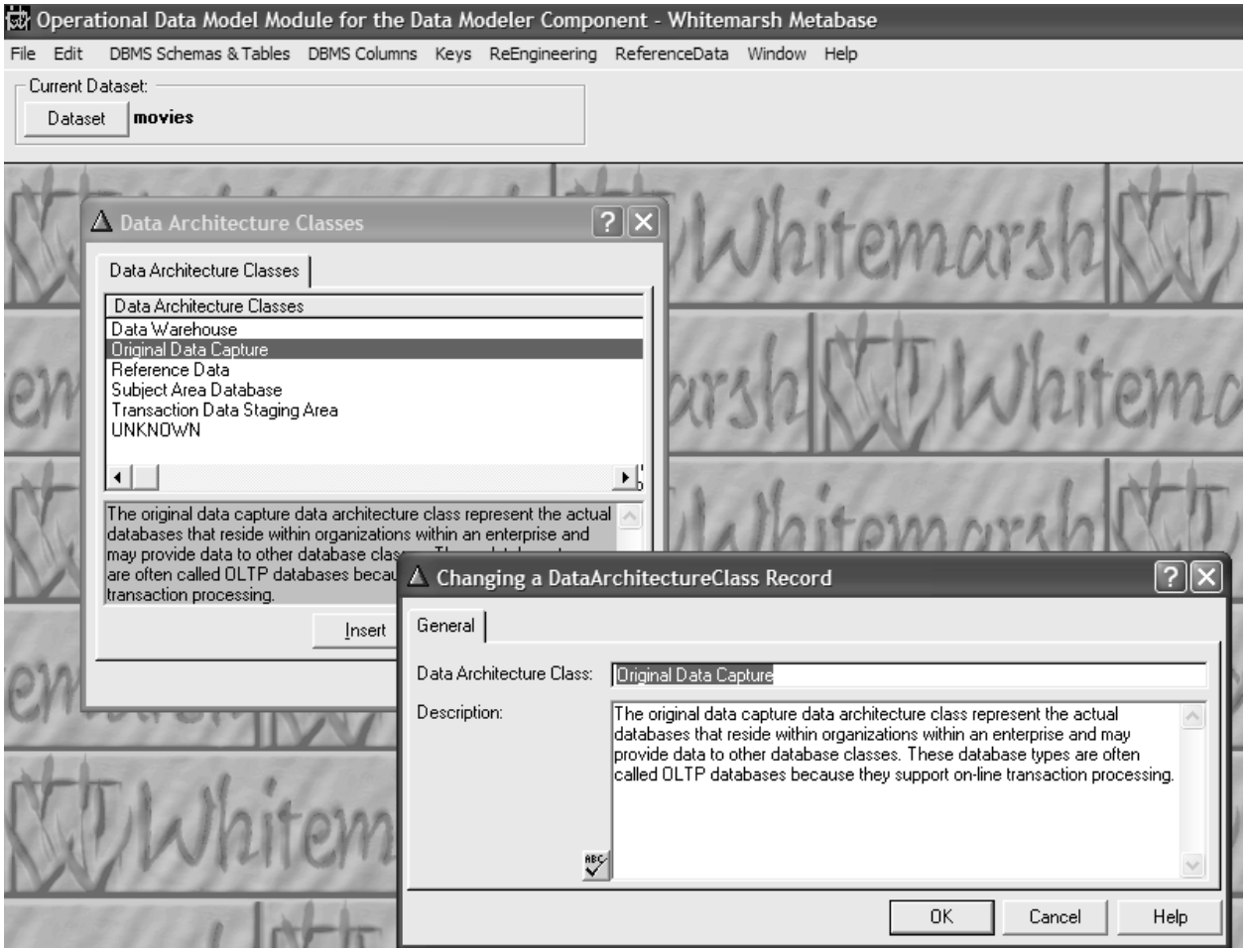

Figure 19. Data Architecture Classes.

3.1.4 Importing the SQL DDL

The process of importing the SQL DDL consists of creating the Operational Data Model Database, and Schema. Then starting the import process once the SQL DDL file is selected and the write to log option is selected.

The first step is to create the database. Figure 20 presents that screen that contains the current set of databases. Within this screen there are three databases, Unknown, Movies Data Warehouse, and Movies Original Data Capture.

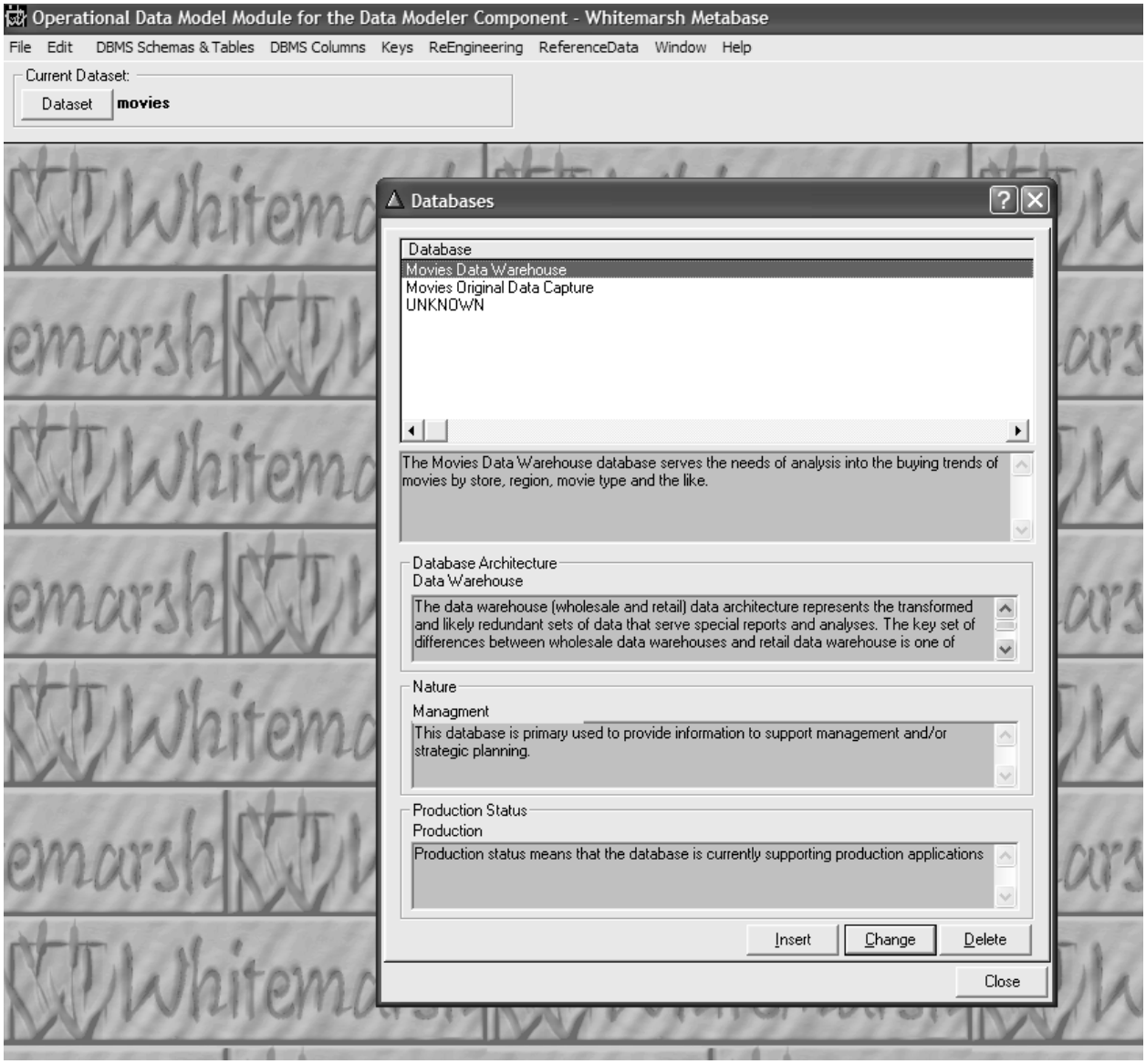

Figure 20. List of databases.

For the purposes of this example, presume that the Movies' Original Data Capture is not there. Press insert, and Figure 21 comes up. Enter the name, "Movies Original Data Capture." At this point the data architecture class is "Unknown." To select a known data architecture class, enter a "zero" in the Data Architecture Class Id field and press enter. At that point, a Data Architecture selection screen appears as displayed in Figure 22. Select, for the purposes of this exercise, Original Data Capture, press Select, and close the window. Then, press OK to close this window. Choose the production status and nature of the database in the same way you chose the data architecture class. That is, enter a zero, tab through, and pick from the selection screen.

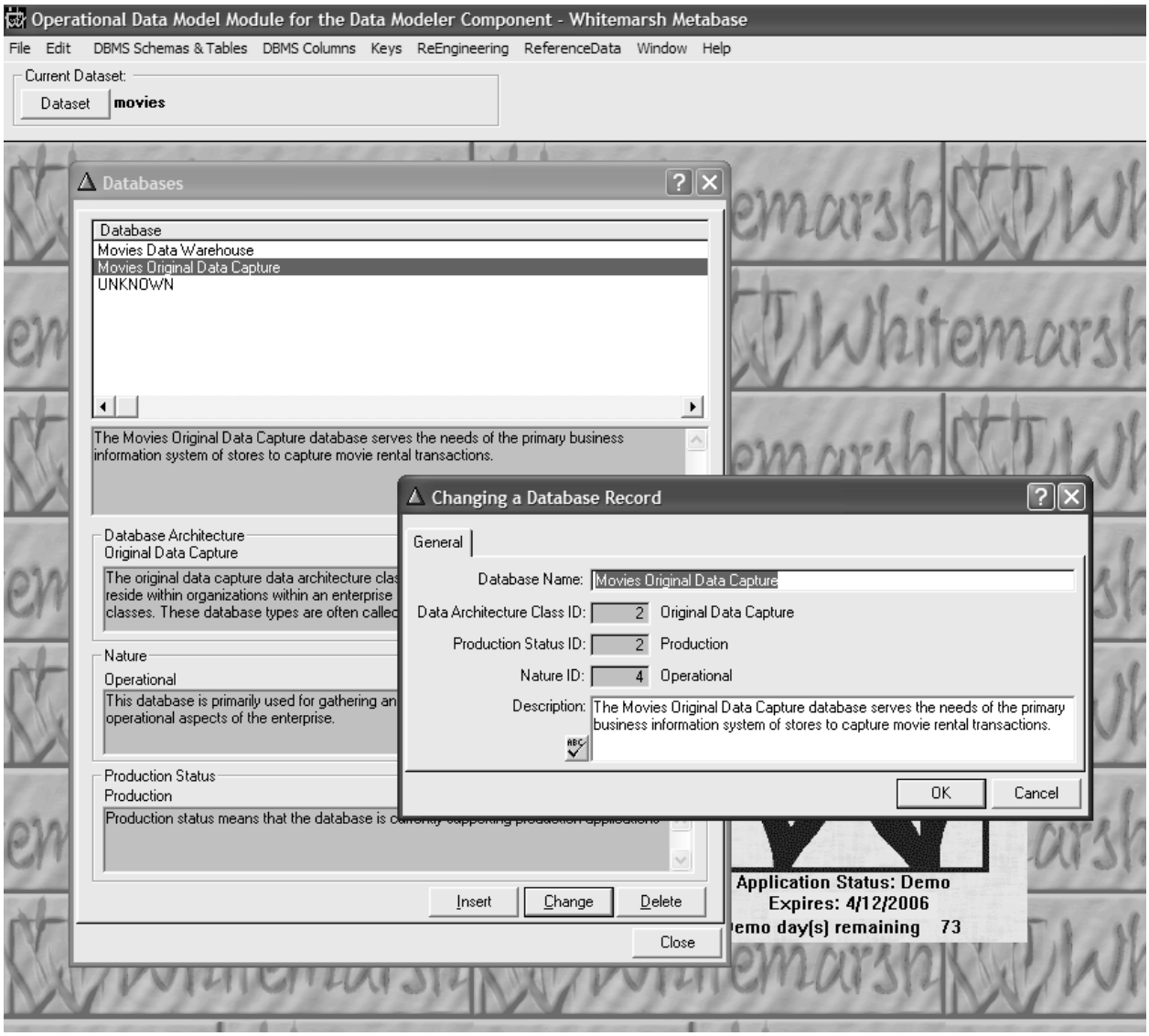

Figure 21. Database update screen.

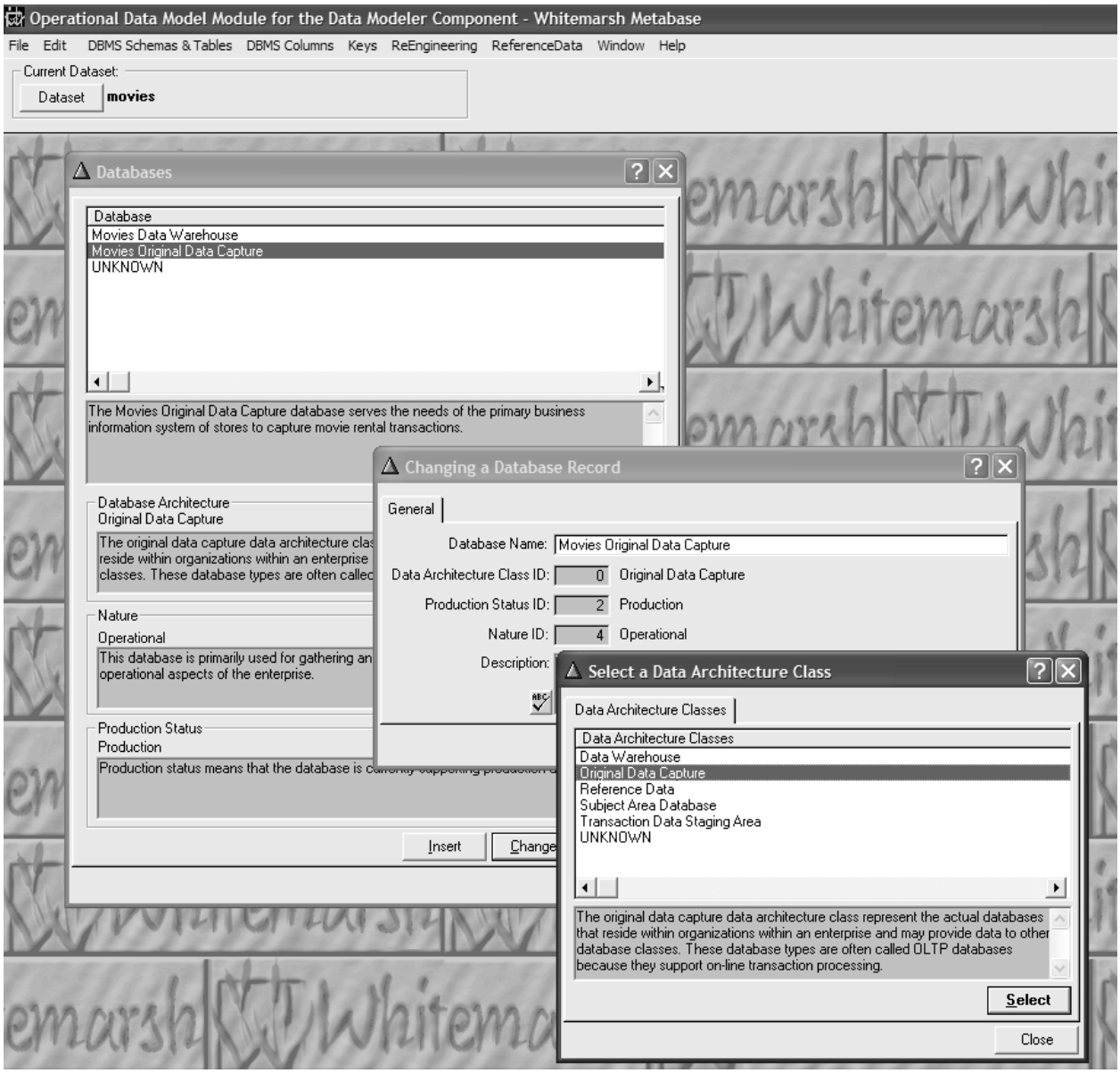

Figure 22. Data architecture class selection.

The next step is to create a schema for that database. A database may have more than one schema across time. To do that, choose DBMS schema under the DBMS Schemas and Tables menu item. Figure 23 then is shown. Select the database, Movies Original Data Capture, then assuming there isn't already a Movies Original Data Capture schema, press Insert. The update screen overlay also shown in Figure 23 then is presented. Create the name and then press enter. If perchance you have chosen the wrong database, that is, not Movies Original Data Capture, then just go to the Id entry, type a zero and press tab. At that point a select screen will appear. Select the "right" database and then press Select. You need then to select the specific DBMS that is to govern this schema. This is employed for data type checking. Select the appropriate SQL DBMS and press OK. Once complete with this screen, click OK. Your choices will appear on the DBMS schemas screen. Close that screen.

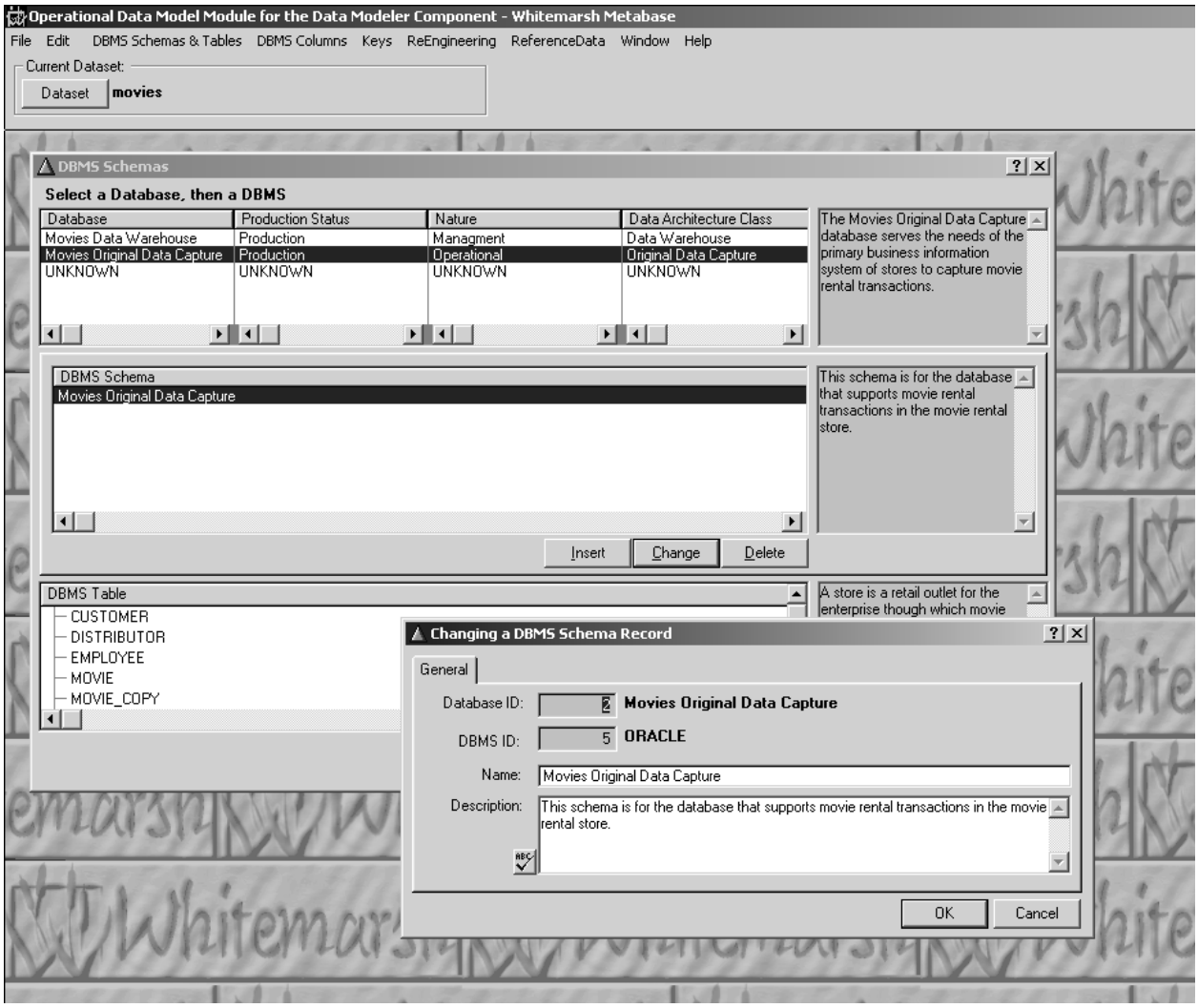

Figure 23. Creating a DBMS Schema.

Now, to actually do the SQL DDL importing, activate the SQL importing by following the menu item selection shown in Figure 24. That brings up the screen in Figure 25. This screen shows that the schema, Movies Original Data Capture exists. If it did not exist then you could created it through the Insert button. Now, select the SQL DDL file with the button, SQL DDL on the right. Browse till you find the file using the file selection dialog at the bottom of this figure. Once found, highlight the file and press the Open button. The window automatically closes and the name of the selected file appears across the middle of the window. Now, you can view the file with the View button. Once you are satisfied, there's one last step. Do you want to create a log of all the actions? If so, then press the Log Import Y/N radio button yes. If you want the file "cleared" before you start importing, press the clear button. Then press the Import button. The importing process first constructs complete SQL statements eliminating all unnecessary short lines, and the like. It then processes the file, one SQL statement at a time. As statements are successfully processed, the metadata is built in the metabase. Once the process is finished, a message appears.

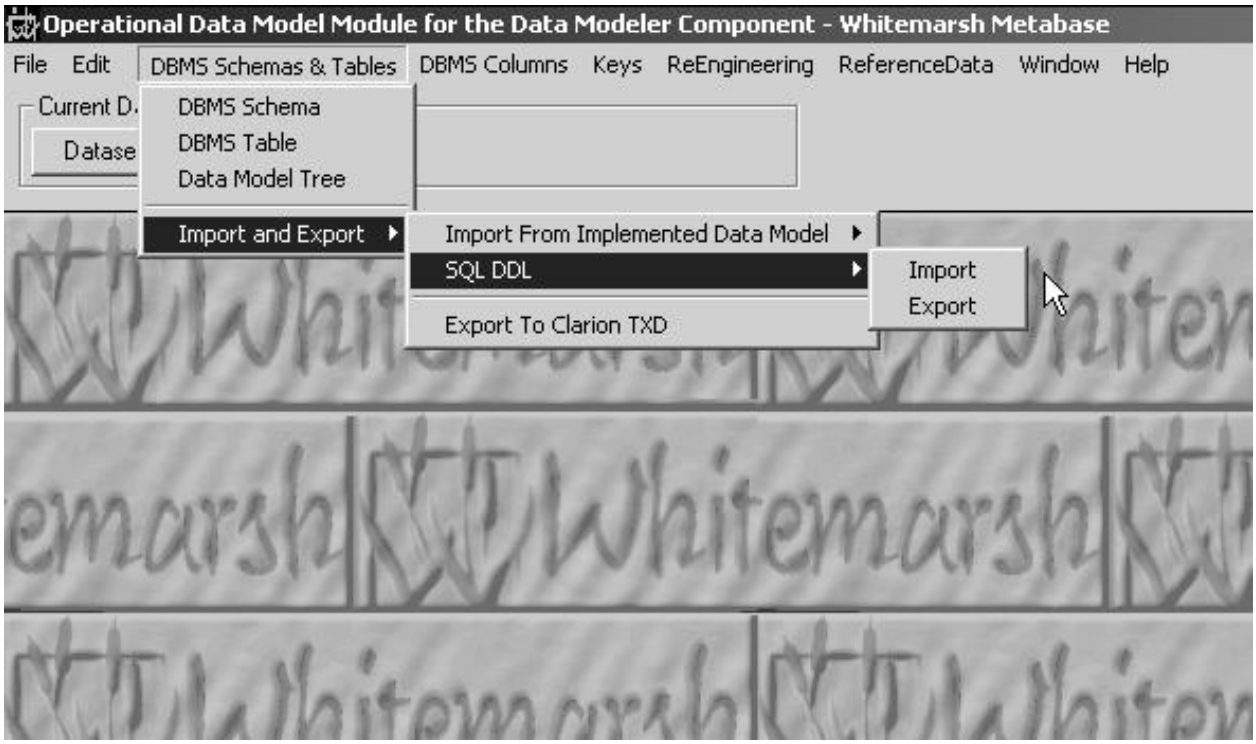

Figure 24. Activating the SQL DDL import process.

What will then have been accomplished is the creation of:

- DBMS Tables
- DBMS Columns
- DBMS Primary keys
- DBMS Unique Keys
- DBMS Foreign keys

If there is an error, you can view the log file to see what was successfully and what the statement was that was being processed at the time of the failure. For example, if a foreign key creation statement references a non-existing table and/or column then the load will fail. Figure 26 shows the first several lines of the SQL DDL import of the Movies schema. The reason "DBMS" is prefixed to all of the items above is because the items are associated with a specific DBMS and thus, they are the tables associated with Oracle, or Sybase, etc.

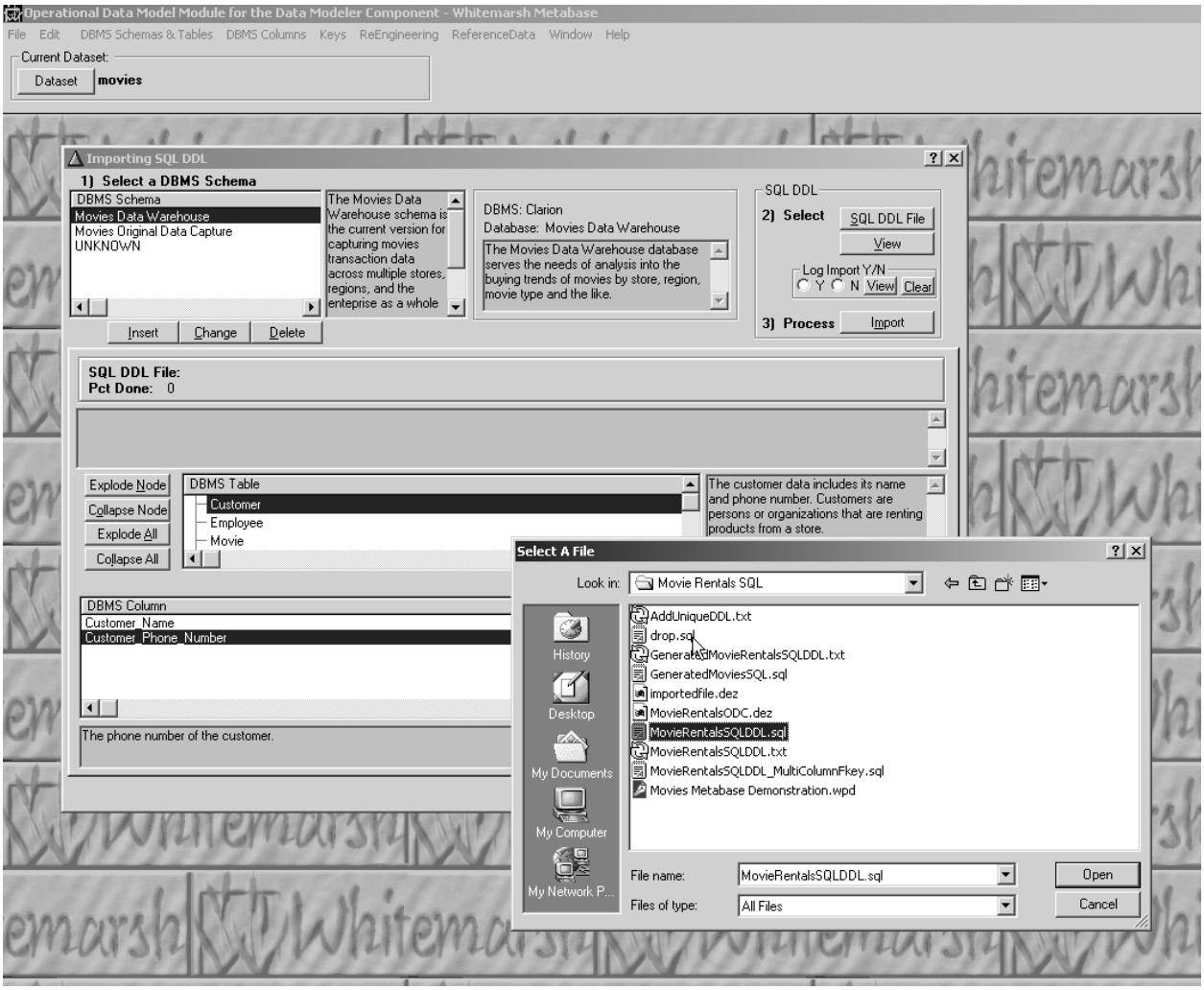

Figure 25. Importing SQL DDL screen.

You should view the imported model by checking the item within the Operational Data Model module. There you can see what the tables and columns are, what the primary and foreign keys are, and the various columns that belong to the primary and foreign keys. Once you are satisfied that the data model has been successfully loaded, proceed to the next step.

Note, that during your review, you will see that the Implemented Data Model to which these DBMS tables et al have been related is "Unknown." That is because the Implemented Data Model does not yet exist. This is an example of the strong referential integrity contained in the metabase. Actually, all the DBMS columns are related to just one Implemented Data Model column, Unknown, that is contained within the Unknown Table of the Unknown Schema.

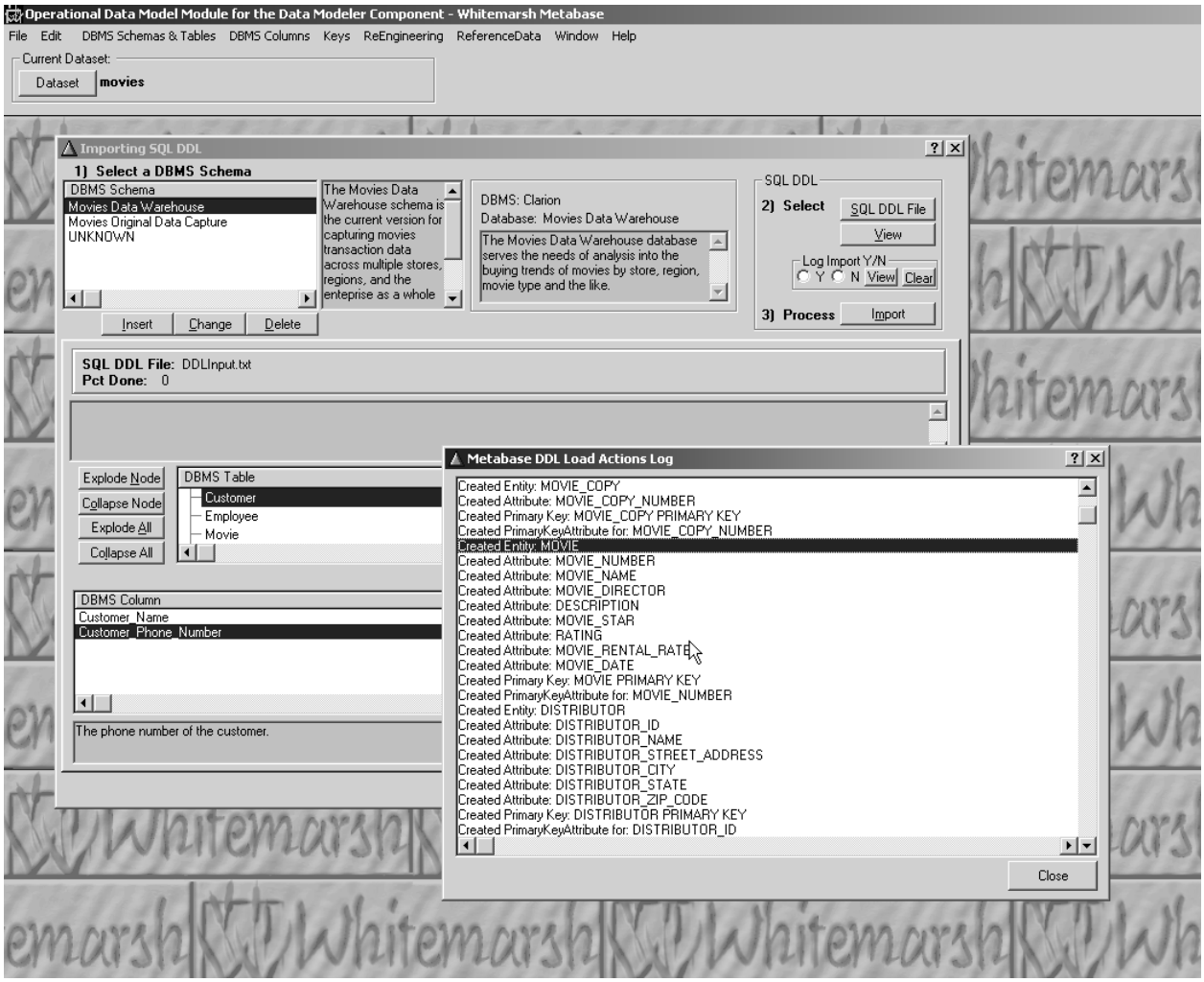

Figure 26. SQL DDL log file view.

3.2 Promote the Operational Data Model to Implemented Data model

Promoting the Operational Data Model to the Implemented Data Model simply makes a "copy" of the Operational Data Model, that is, schema, tables, columns, and all keys (primary, foreign, and unique, but not secondary) and creates a complete clone at the Implemented Data Model layer.

The data model diagram of the Implemented Data Model is presented in Figure 27. Each "arrow" represents a one-to-many relationship. The "story" of this data model is that for every schema there may be one or more tables. Every table has at most one primary key. Every table may have one or more columns. Every column is governed by a SQL data type. Every table may have one or more foreign keys that represent the relationship between the table to which the foreign key belongs, and another table associated with the foreign key's related primary key. A table may have additional unique keys that are represented as Candidate Keys. Every key, that is, primary, foreign, or candidate has one or more columns that constitute the value basis for the relationship, that is, the existence of rows of data in the table, Table Primary Key & Column, or Table Foreign Key & Column, or finally, Table Candidate Key & Column. Additionally, those columns exist in a specific sequence within that key-based relationship.

You have probably noticed that the text is almost exactly the same as in Section 3.1. Yes, that is correct. So, a good question is what is the relationship between Implemented Data Model and Operational Data Model. The answer is simple. For every Implemented Data Model there can be one or more Operational Data Models, each with a different design but all mapped back to the same Implemented Data Model. The basis of the relationship is not, however, Implemented Data Model Schema to Operational Data Model Schema, nor Implemented Data Model Table to Operational Data Model Table. That would mean that the Implemented Data Model is merely a transformation to the Operational Data Model. Rather, the relationship is between Implemented Data Model table column to Operational Data Model DBMS table column.

Changed, however, are the data types. The DBMS data types are changed to their corresponding ANSI SQL data types. If a DBMS data type is an "unknown" SQL data type then, "unknown" will also be shown at the Implemented Data Model layer. That is ultimately not good, so ensure that there is a mapping among the data types from across the Operational, Implemented, and Value Domain data type layers. Facilities within each of the operational, implemented, and data element modules exist to ensure complete data type mapping.

The process of making an Implemented Data Model is straight forward: select it and then press the promote button. A "percent done" display provides a general idea of the percent of tables that have been promoted from the Operational Data Model to the Implemented Data Model.

Figure 28 presents the screen. It is found under the Re-Engineering menu item. Select the DBMS then the Schema underneath the DBMS that is to be promoted. Then just press the Promote button. The percent done will be displayed as it is building the Implemented Data Model's

- Tables
- Columns
- Primary keys
- Unique Keys
- Foreign keys

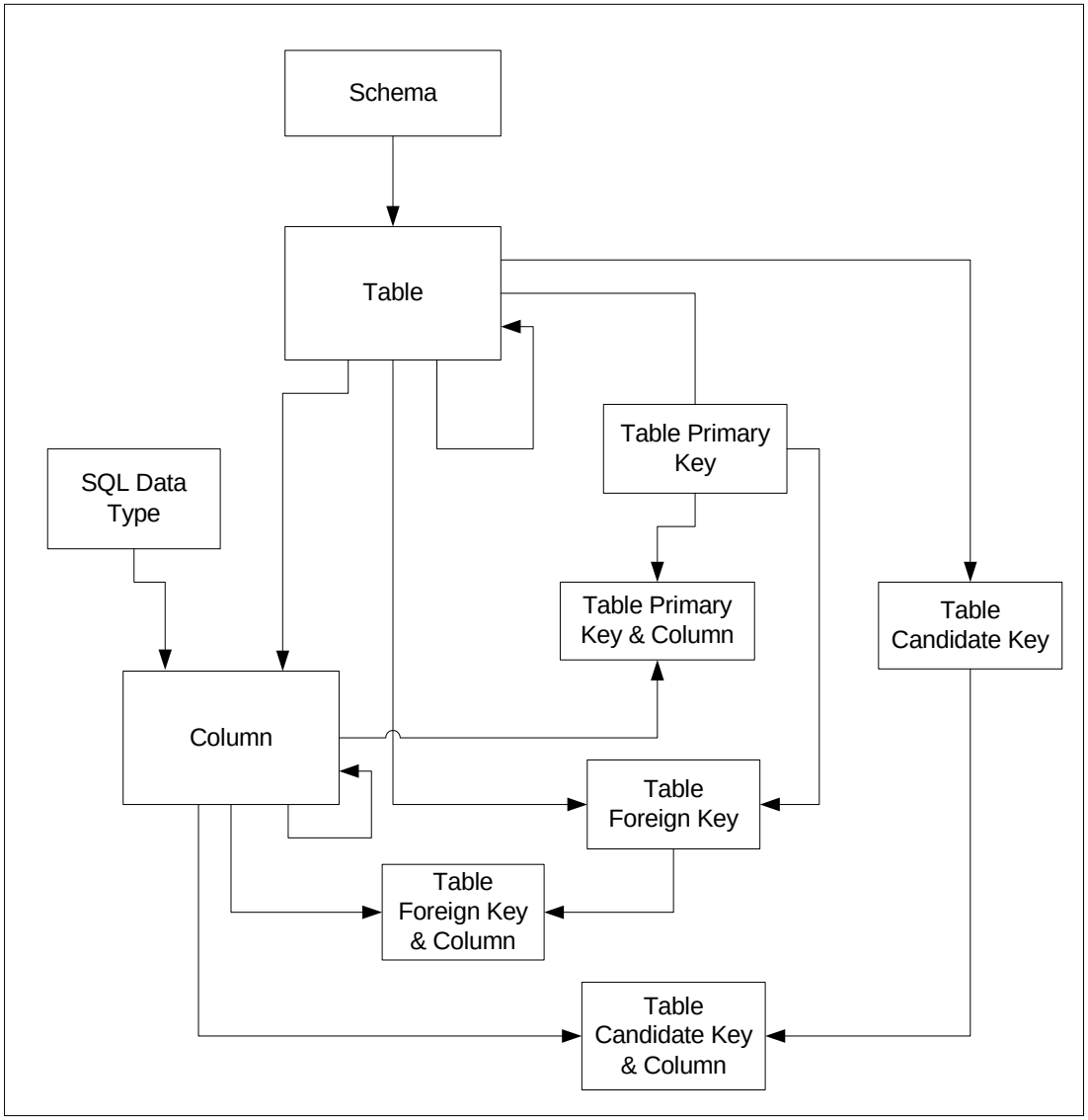

Figure 27. Implemented Data Model diagram.

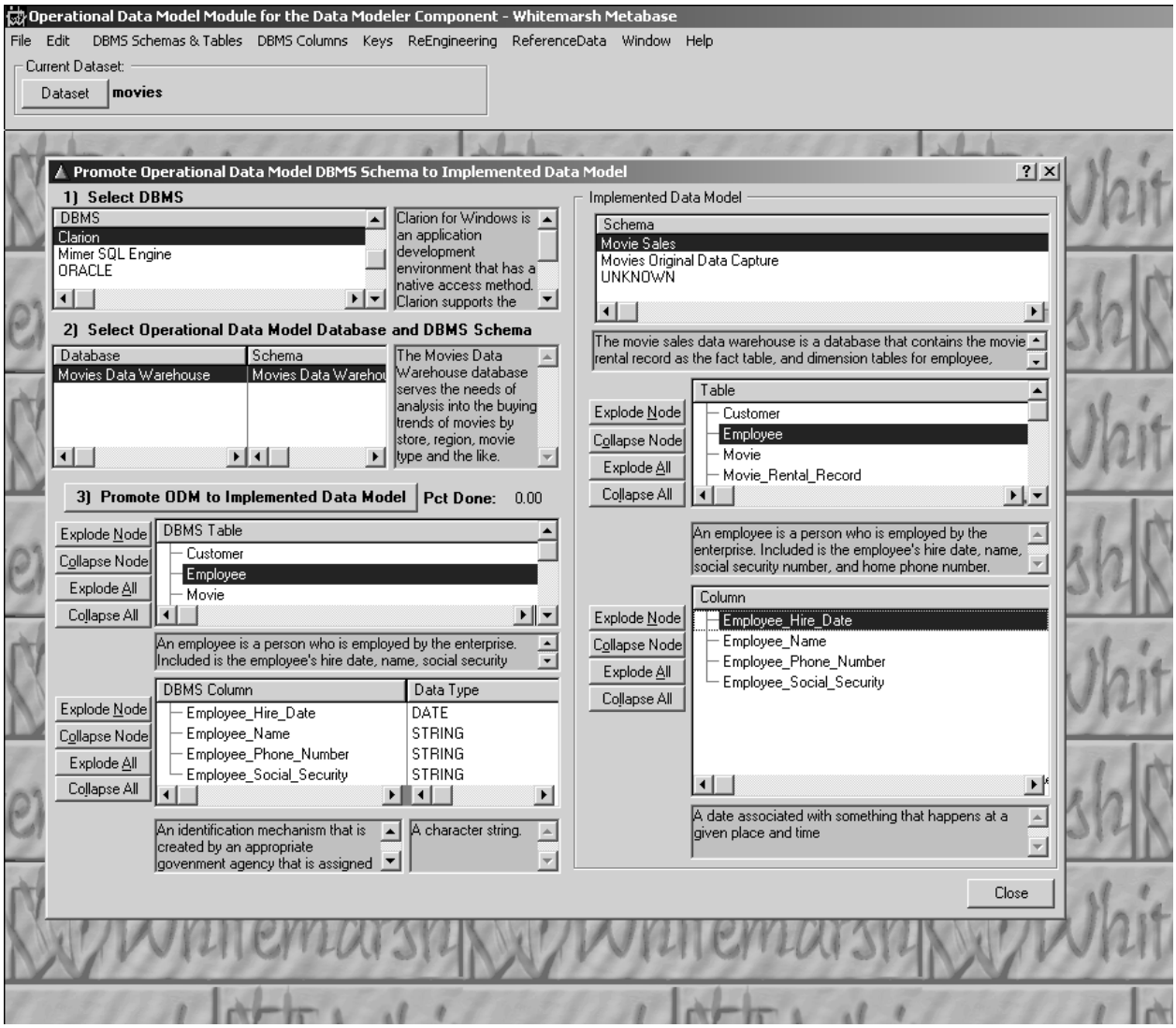

Figure 28. Promotion of an Operational Data Model to an Implemented Data Model.

The right side of this screen shows the completed result. This result will not be shown until the promotion process is complete. Selecting the newly created Schema will then cause the tables and columns for the highlighted table to be displayed.

To verify what was built, execute the Implemented Data Model and review the promoted data model. When you review it you will see that it too has “upstream” unknown references to attributes for the columns and for data elements for the columns.

If for some reason you do not wish the Implemented Data Model to remain in the metabase, removing it is **not** just the press of a button. That is because of referential integrity. Once built, the Implemented Data Model’s columns are now the “upstream” references to the Operational

Data Model's DBMS columns. So, to delete an Implemented Data Model, you must make all the upstream references "unknown." Then the referential integrity "problem" will go away and you can delete the Implemented Data Model.

To make the references "unknown," go back to the Operational Data Model, and under the menu item, Re-Engineering, select the menu item, Remove DBMS Table DBMS Column to Column References. Figure 29 presents that window. Highlight the table, and press the button to remove the references. On a table by table basis the references will be changed to "unknown." Thereafter, you can return to the Implemented Data Model module, select the Schema menu item underneath the Schema and Tables menu item, select the schema and press the delete button. It will give you a confirmation question to affirm. Once you say yes, the Implemented Data Model schema and all associated tables, columns, and keys will be deleted from the database. **WARNING. This is NOT undoable**. So, if done by mistake, just let it continue to the end, and then repeat this promotion step.

3.3 Promote Implemented Data Model to Specified Data Model

This step is largely the same as the previous step. So, Figure 30 shows the promotion screen. Go forth and promote. Once the promotion is complete you will be able to execute the Specified Data Model and then see that there is a single new subject, Movies Original Data Capture and a whole collection of entities and attributes corresponding to the Implemented Data Model's tables and columns. Figure 30 shows additional subject areas that are created during the next step.

There will not, however, be any data types. That is because the Specified Data Model is too abstract to be bound by data types. Data types exist only at the Data Element Model, Implemented Data Model and Operational Data Model levels. That's OK, so don't be concerned. Proceed to the next step.

The data model diagram for the Specified Data Model is presented in Figure 31.The data model diagram for the Operational Data Model is presented in Figure 17. Each "arrow" represents a one-to-many relationship. The "story" of this data model is that for every subject there may be one or more entities. Every entity has at most one primary key. Every entity may have one or more attributes. Every entity may have one or more foreign keys that represent the relationship between the entity to which the foreign key belongs, and another entity that is associated with the foreign key's related primary key. An entity may have additional unique keys that are represented as Candidate Keys. Every key, that is, primary, foreign, or candidate has one or more attributes that constitute the value basis for the relationship. Additionally, those attributes exist in a specific sequence within that key-based relationship.

The relationship between the Specified Data Model and the Implemented Data Model is again one-to-many. There is also this key difference: A Specified Data Model relationship may cross subjects whereas in the Implemented or Operational Data Models, a primary-foreign key based relationship must always be contained within a single schema.

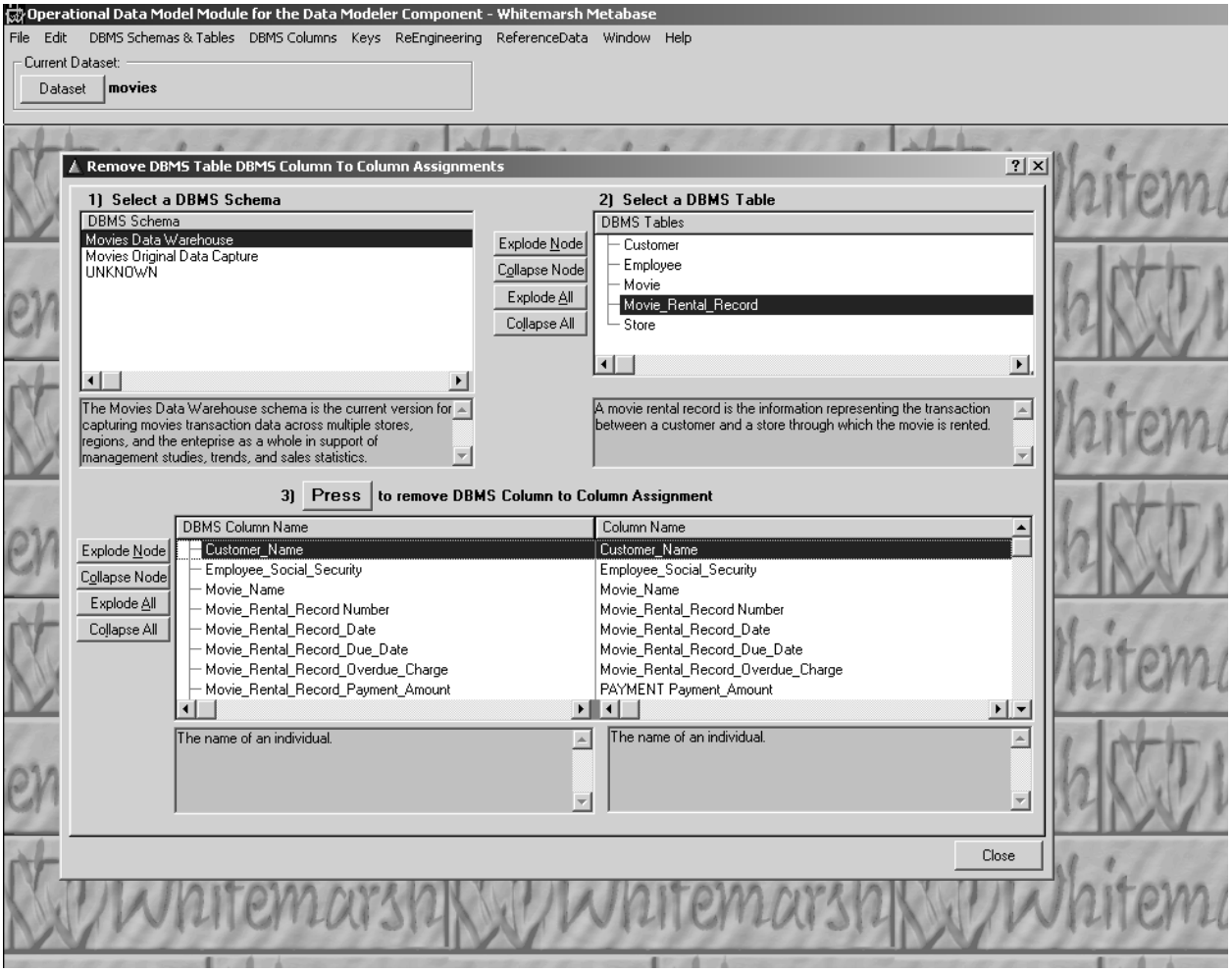

Figure 29. Reassigning “unknown” to DBMS Table Columns.

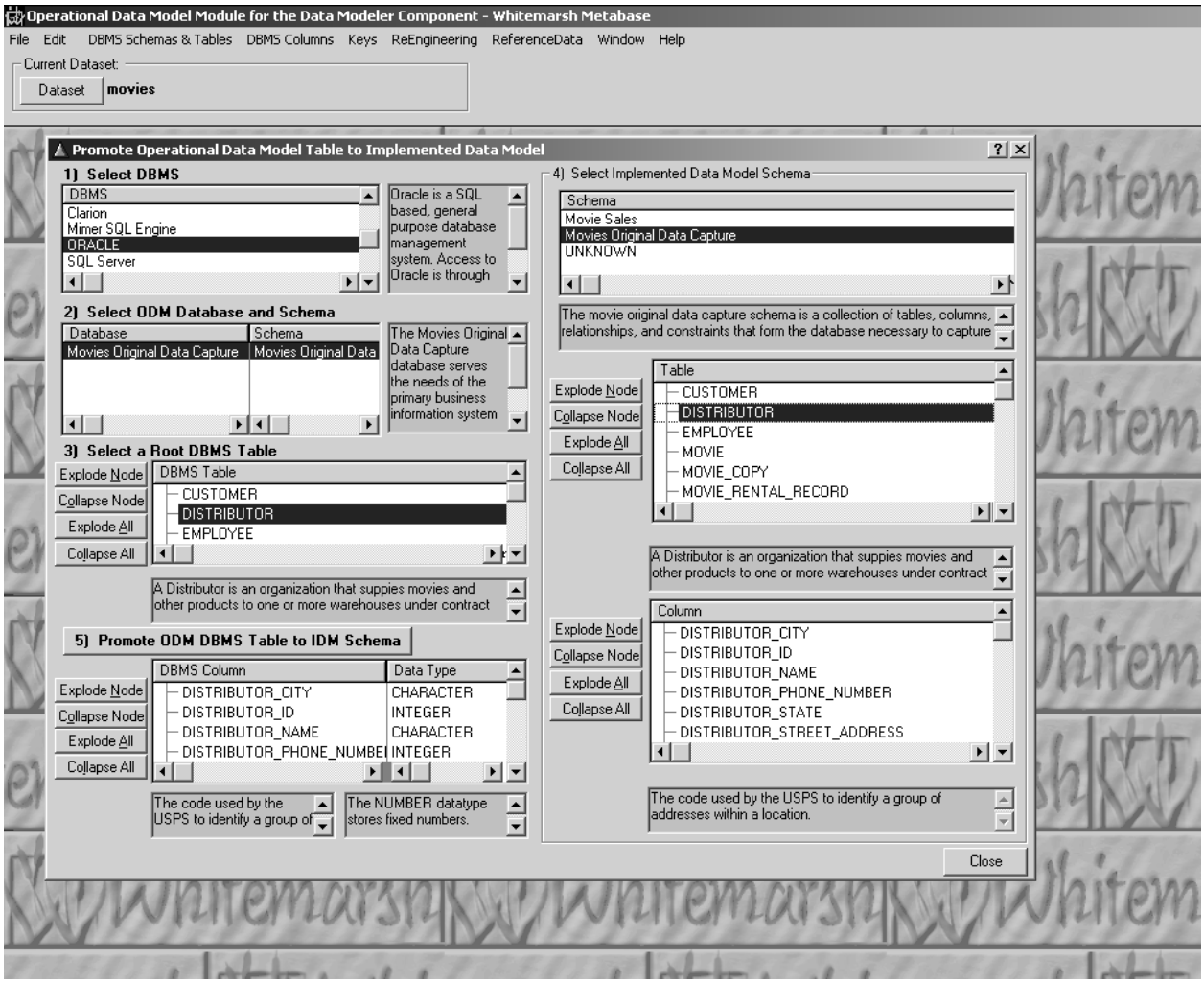

Figure 30. Promotion of Implemented Data Model to Specified Data Model.

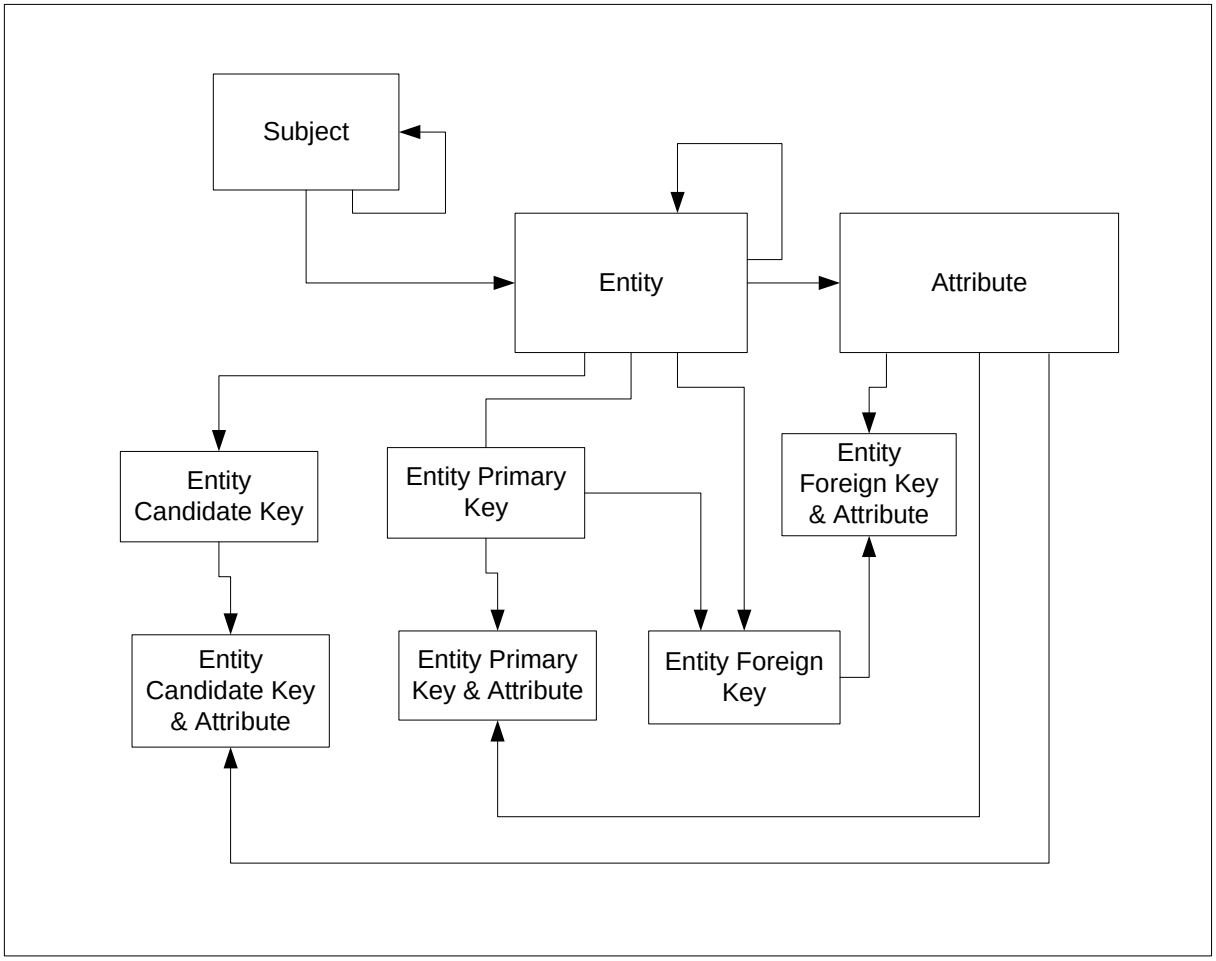

Figure 31. Specified Data Model diagram.

4.0 Specified Data Model Re-Engineering

This next step is very significant. The Specified Data Model is transformed from a single subject with a collection of entities and attributes to multiple subjects each with a set of entities and attributes. Unique to the Specified Data Model there can be relationships (primary key to foreign key) across subject areas. In short, it is transformed from a single data model of entities to a collection of data model templates, each of which is composed of entities that can then, in turn, be employed to construct data models within the Implemented Data Model.

The entities that were promoted to the Specified Data Model from the Implemented Data Model were:

- Customer
- Distributor
- Employee
- Movie
- Movie Copy
- Movie Rental Record
- Payment
- Store

The first task is to figure out what each of these entities really represents. That is, what is the business subject within which these entities naturally occur. The table that follows presents the result of that analysis. In this example, the quantity of entities are few and so the perceived value of having seven subjects for eleven entities doesn't seem to be too high. As the examples become more sizable and as the quantity of Implemented Data Models mapped to the evolved Specified Data Model grows, the value of the templates similarly grow. Each use of a template represents data standardization. The greater the use the greater the standardization.

Subject	Entity
Business Transaction	Movie Rental Record
	Payment
Information Technology	Information Technology Attributes
Location	Location Address
Locator	Telephone Number
Organization	Distributor
	Store
Person	Customer

Subject	Entity
	Employee
Product	Movie
	Movie Copy

Table 1. Subjects and Entities.

Three of the entities are brand new. That is, Location Address, Telephone Number, and Information Technology Attributes. Location Address arose from an analysis number of the existing entities. There was a set of common attributes which was factored out. The telephone number attribute was similarly factored out of existing entities and put into the Locator entity. Finally, because almost every entity had a surrogate key identifier it too was factored out and placed into a new subject, Information Technology, to hold Information Technology Attributes.

Remember the task here is not to build a data model that is to exist within the context of a database's schema, but to build collections of data model templates that are in turn are used to for build data models. Each template is bounded within a subject and consists of one or related entities and their attributes. Further, every entity must be precise, that is, have a sufficient quantity of attributes to have a "natural" primary key. In this example, surrogate keys have been used as the basis for primary keys. Those are not sufficient for business entity modeling.

Look, for example, at the column Telephone Number in every one of the tables. Other than the name, it's the same. Now suppose the Telephone Number was a bit more complicated, say, Country Code, Area Code, Exchange, Number, and Extension. To effect a standardization across all the tables, a whole lot would have to either be remembered and/or copied into all the tables across all the schemas of the entire enterprise in a very disciplined way. The Specified Data Model, in contrast, is the "copy" process. Once designed, the data model template can be used and re-used to accomplish data standardization.

Given that you've been reasonably convinced, then the steps that must be done are:

- Create the subjects
- Move entities from one subject to another
- Create new entities
- Factor out attributes from one entity to another
- Re-engineering primary and foreign keys
- Maintain attributes

4.1 Create Subjects

The process of creating a subject is relatively easy. Select the menu item, Subjects, under the main menu item, Subjects and Entities. Figure 32 presents the Subject browse and then update screen. Because subjects can be nested, either select a subject within which you wish to nest a new subject, or select the string, “Hierarchy” and then press enter to create a new top level subject. In this example, the new subject was Business Transaction. Create the name, its abbreviations, and a descriptive definition of what a Business Transaction is. A subject may only have one entity or it may have more than one. That all depends on the nature and complexity of the subject. Create all the subjects that are the table above, that is, Business Transaction, Information Technology, Location, Locator, Organization, Person, and Product.

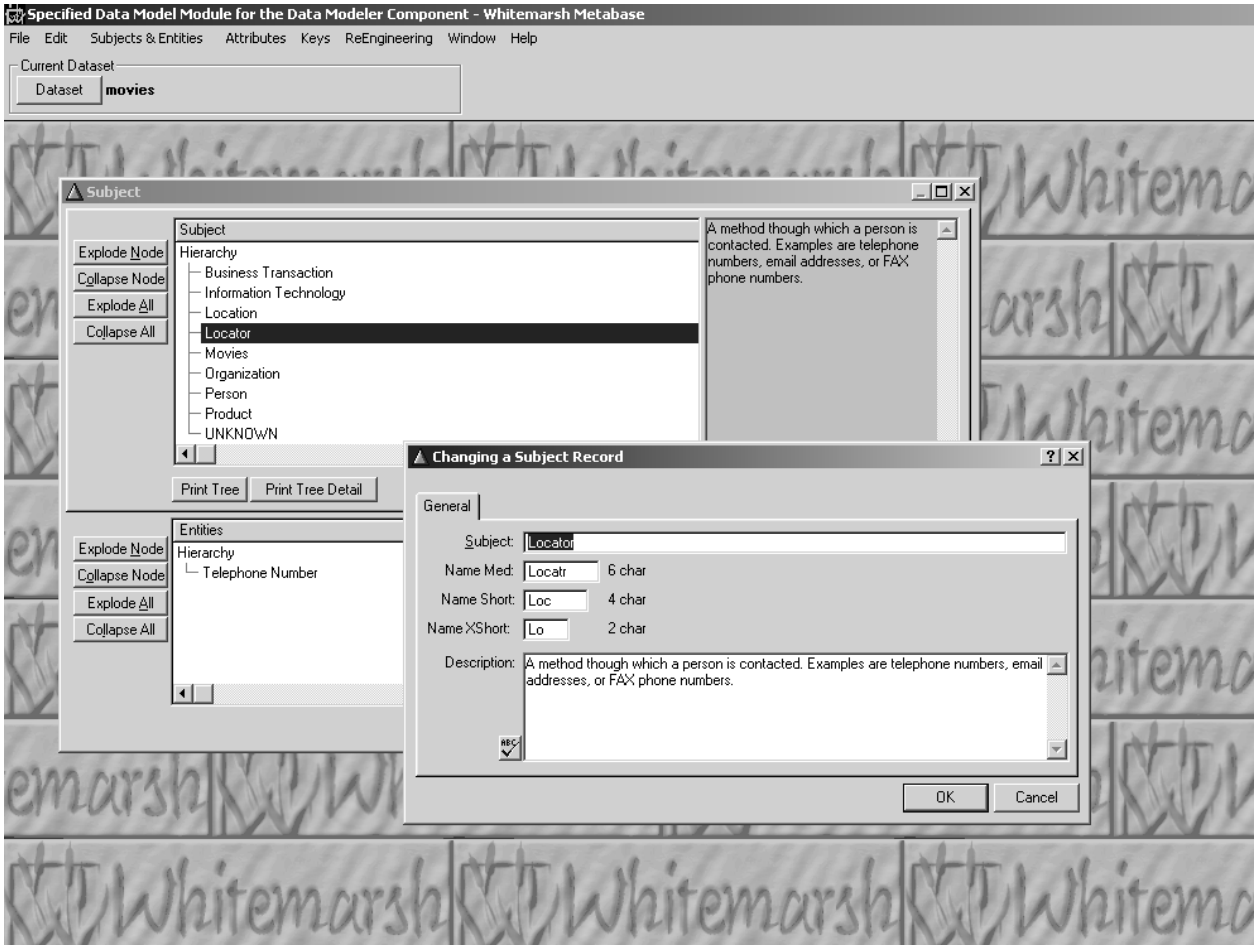

Figure 32. Creating a new subject.

4.2 Move Entities

The process of moving entities is a re-engineering step. So, under Re-Engineering, activate Reassign Entities to Subjects. Figure 33 it then displayed. In this process, select the Subject in the left Screen. Then "check" one or more entities. Then on the right screen, check the move-to subject. Finally, press the button to move the entity from one subject to the next. Table 1 above provides a list of where each of the entities are to be moved from the original subject, Movies Original Data Capture.

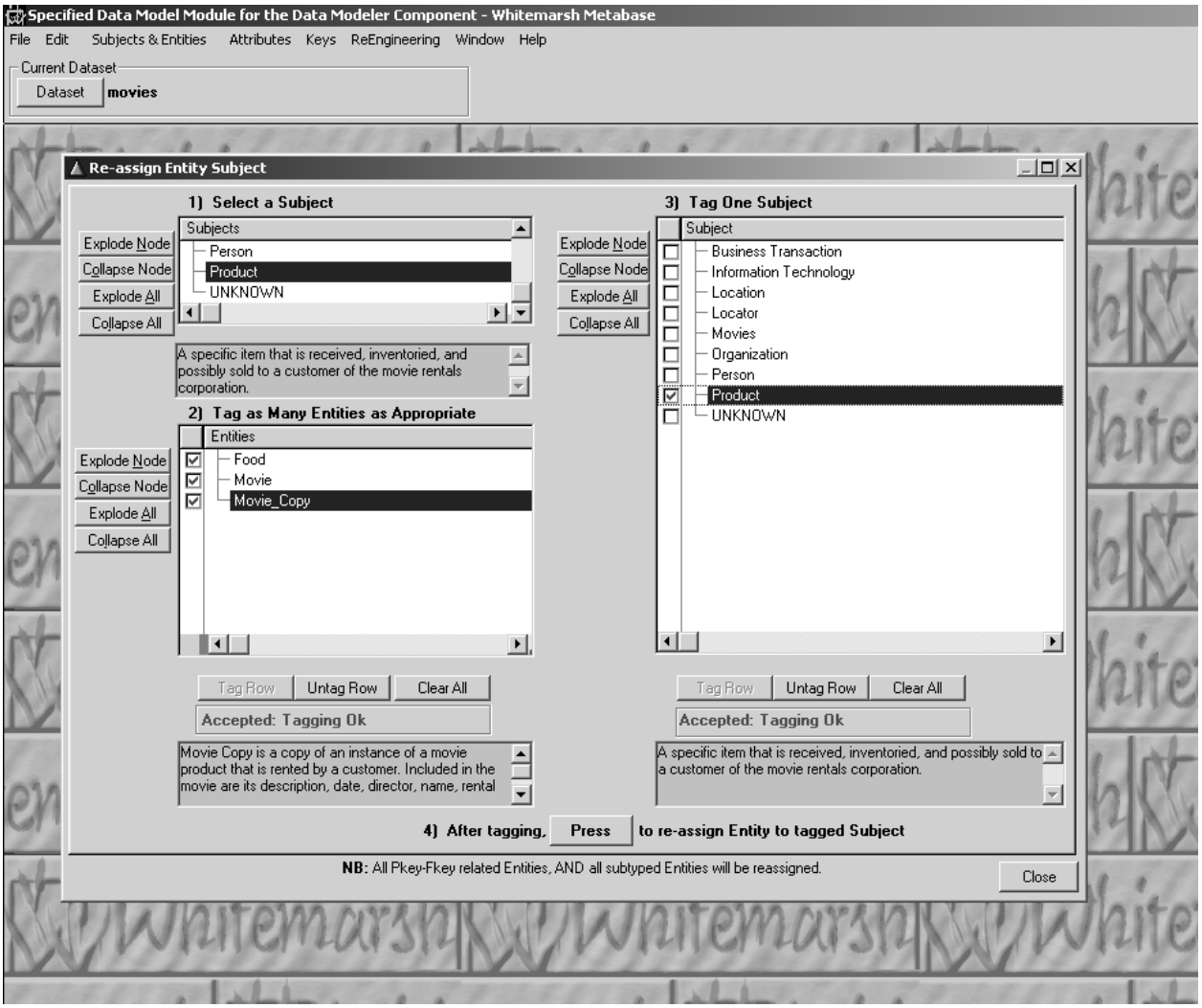

Figure 33. Reassigning entities to different subjects.

As each entity is moved so too are all the attributes, primary keys and candidate keys. Foreign keys are different. In the Specified Data Model, foreign keys are allowed to span subjects. That is because a subject is considered a "soft" container. Certainly, an entity in one subject may be the parent of an entity in another subject. For example, an Organization may have an Address. In contrast, tables in different schemas of the Implemented Data Model and the Operational Data Model cannot have foreign keys that span either schemas or DBMSs schemas as these are considered "hard" containers.

The ultimate goal of moving these entities is to empty the Movies Original Data Capture subject area of all its entities. Once that is done, use the same screen that inserts subjects to then delete the Movies Original Data Capture subject.

4.3 Create New Entities

Three of the subjects, Information Technology, Location, and Locator need entities. Table 1 shows what the entities are to be. To add a new entity, click the Entities menu item within Subjects and Entities. Select the Subject, for example, Location, then if the Entity is not be nested then select Hierarchy and press Insert. The screen is shown in Figure 34. Name the entity, for example, Location Address, create the abbreviations, and then provide a definition. If the entity is nested, then select the parent entity and press Insert. The new entity will then appear as a child.

Similarly, create the new entities from Table 1 until all the entities are properly located within all the subjects.

4.4 Factor Out Attributes

The three new entities do not have any attributes. Are there new attributes? No. They are attributes factored out of existing entities. That does not mean that these are look-up table attributes. Rather it means that these attributes properly belong in a different entity of a different subject. Remember, this effort is not about building a data model, it's about building data model templates. You are specifying data model templates that are to be used in building the "real" data models that exist in the Implemented and Operational Data Models.

From Table 2, you can see that the attributes for Location Address, and Telephone number come from other entities. The address attributes within Location Address are the street address, city, state, and zip code attributes. The attribute within Telephone Number are the various telephone numbers from the other entities. Finally, the attribute, Information Technology Identifier comes from all the "Id" attributes from the other entities. These "Id" attributes are however not just straight "factoring" operations and will be dealt with in the Re-engineering Keys step that follows.

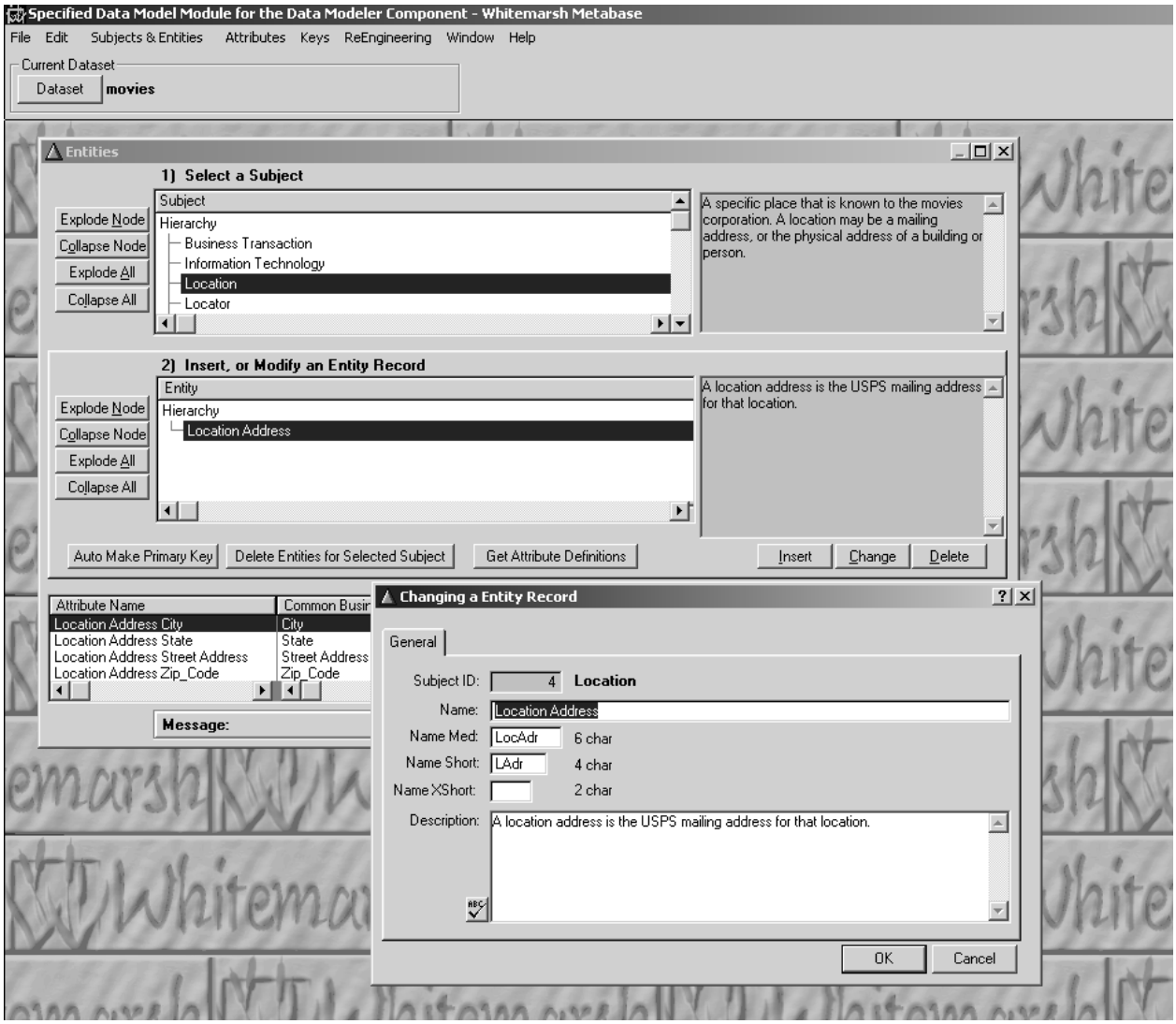

Figure 34. Creating a new entity.

The attribute refactoring process is similar to the entity move process. Figure 35 presents the screen. The process of moving an attribute consists of selecting from the left side of the screen, the subject, then the entity, then check one or more attributes that are to be moved. On the right side of the screen, select the target subject and then check the target entity.

WARNING: You cannot successfully move an attribute that is part of either a primary or a foreign key. If you could, then the relationships between entities would also be changed. The primary or foreign key attribute move requires actions from the Re-engineer Keys step.

In this example, choose an entity with an address, for example, Distributor, and then Tag Distributor Street Address, Distributor State, Distributor City, and Distributor Zip Code. Then select the subject, Location, and tag the Location Address. Finally press “Press” to re-assign

attributes. This will then "move" the address specific attributes from Distributor to Location Address.

Now, if we then went to all the other entities with address and moved those to Location Address then we would have multiple sets of address attributes in one entity. So, do not do that. Instead, do these steps:

- Rename the address attributes to be more generic.
- Execute the Implemented Data Model module and "re-assign" all the address columns in all the tables to this one set of address attributes in the Location Address entity.
- Delete the address attributes from the remaining Specified Data Model Entities.

These three steps, detailed in Section 4.4.2, will then have established Location Address as the data model template that was employed as the model for all the uses of Address in the Implemented Data Model tables. Further, it will have removed these address attributes from the various Specified Data Model entities, thus making them more "single purpose."

Subject	Entity	Attribute
Business Transaction	Movie_Rental_Record	Customer_Number
		Employee_Id
		Movie_Copy_Number
		Movie_Number
		Movie_Rental_Record_Date
		Movie_Rental_Record_Due_Date
		Movie_Rental_Record_Overdue_Charge
		Movie_Rental_Record_Rate
		Movie_Rental_Record_Status
		Payment_Transaction_Number
		Rental_Record_Number
	Payment	Check_Bank_Number
		Check_Number
		Credit_Card_Number
		Customer_Number

Subject	Entity	Attribute
		Employee_Id
		Payment_Amount
		Payment_Credit_Card_Expiration
		Payment_Credit_Card_Type
		Payment_Date
		Payment_Status
		Payment_Transaction_Number
		Payment_Type
Information Technology	Information Technology Attributes	Information Technology Attributes Identifier
Location	Location Address	Location Address City
		Location Address State
		Location Address Street_Address
		Location Address Zip_Code
Locator	Telephone Number	Telephone Number Phone_Number
Organization	Distributor	Distributor_Id
		Distributor_Name
	Store	Distributor_Id
		Store_Id
		Store_Name

Subject	Entity	Attribute
Person	Customer	Customer_Name
		Customer_Number
	Employee	Employee_Hire_Date
		Employee_Id
		Employee_Name
		Employee_Social_Security_#
		Store_Id
		Supervisor
Product	Movie	Description
		Movie_Date
		Movie_Director
		Movie_Name
		Movie_Number
		Movie_Rental_Rate
		Movie_Star
		Rating
	Movie_Copy	General_Condition
		Movie_Copy_Number

Table 2. Subjects, Entities, and Attributes.

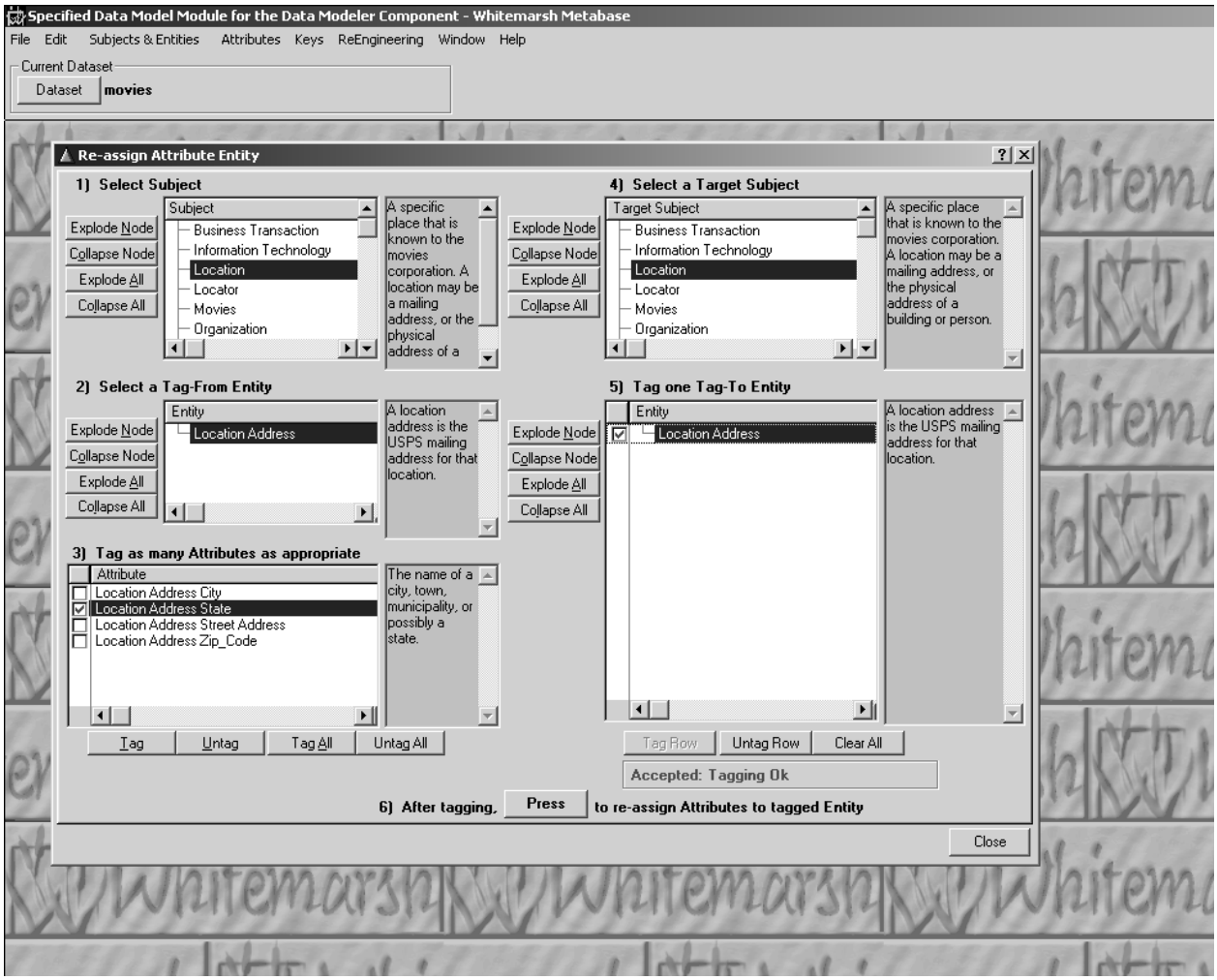

Figure 35. Moving attributes from one entity to another.

4.4.1 Rename Attributes

The process of renaming attributes is illustrated in Figure 36. It is activated from the main menu item, Attributes, then Maintenance then Maintain Attributes. Select the subject, then entity, then the attribute that is to be changes (i.e., renamed) and press "Change." For the purposes of this example, select the "State" name.

The first thing to do is to clear the check mark on the "Freeze all names." This was placed on all similar update forms to ensure that names were not changed accidently. In this case it's on purpose, so clear the check mark.

Now, the question is what should the name be? Names in the Whitemarsh metabase are the result of an automatic construction process. Generally, an attribute's name consists of the following parts:

< modifiers > <entity name > < common business name > < class words >

Where, modifiers and class words are assigned through the Attribute Meta Category Value assignment menu item. To better understand this process consult the *Specified Data Model* user guide and the *Data Modeler Architecture and Concept of Operations* book.

Now, clear whatever name is in the Common Business Name, the Name, and the User Set Name entry boxes. Now enter just "State" in the Common business name. Press the "Reset" button. At that point, because no modifiers and class words have been assigned, the name will appear as: <entity name> <common business name>. Construct the abbreviations for that attribute name.

If you wish you can always change the name to Location State, or keep it as Location Address State, and then just change the user set name to Location State. Finally, declare whether a null value is allowed and then enter a description.

Now, do this attribute name change for all the remaining attributes within Location Address. Upon completion the attributes should be similar to those in Table 2. Now, repeat this process for Telephone Number.

The next step finishes the reassignment.

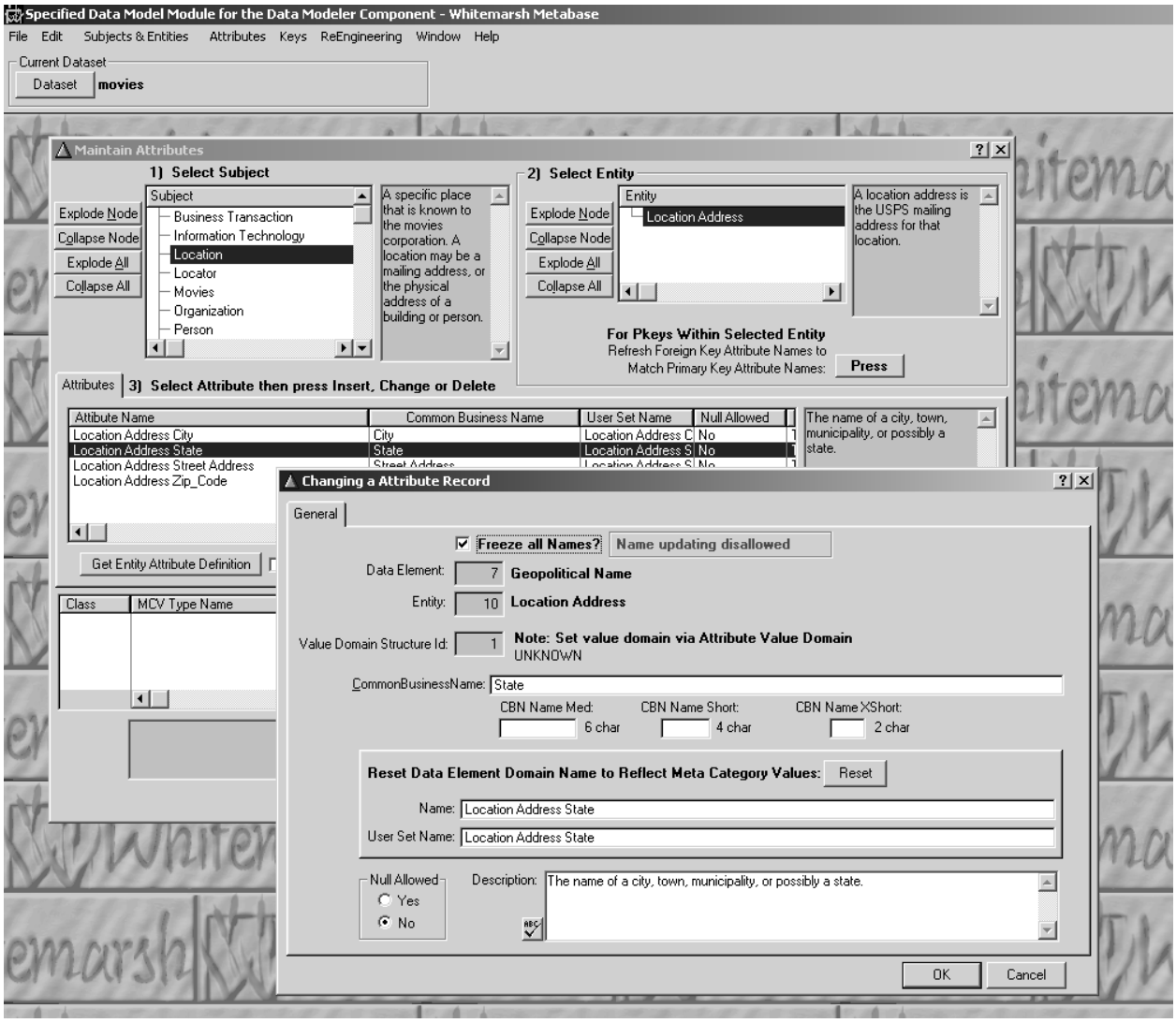

Figure 36. Rename attributes.

4.4.2 Re-assign Column Attributes

When the address attributes are moved to the Location Address entity, the references from, for example, the Distributor table now point back to these moved attributes, even when renamed. That is not the case with all the other address columns within the other tables. That must now be fixed. The process, ReAssign Columns to Attributes from within the Implemented Data Model module accomplishes the reassignment.

Execute the Implemented Data Modeler, select the Movies Original Data Capture data set, and then select the Re-Assign Columns to Attributes underneath the Re-Engineering menu item.

Figure 37 is then presented. On the left side, the Implemented Data Model side, select schema, table, and then tag one or more columns. On the right side, the Specified Data Model side, select the subject, entity, and then tag the attribute. Then press “Press” to reassign the column to the attribute.

In this example, proceed through the various Specified Data Model entities looking for address type attributes. Reassign them to the appropriate Specified Data Model attribute. Once you are complete, do the same for Telephone number. Once all the re-assignments are completed, then the Specified Data Model attributes will be “free” to be deleted. By “free” it is meant that the Specified Data Model attributes are now not referenced by any Implemented Data Model column. Remember, the metabase has strong referential integrity. So, if you try to delete an attribute and if that attribute is related to a column then the delete operation will fail. Hence we had to “move” the relationship of the address columns to the Location Address entity’s attributes.

When you are finished, close the window.

Do not reassign the “Id” columns from the Implemented Data Model to the attribute, Information Technology Attributes Identifier within Specified Data Model’s entity, Information Technology Attributes. The reason for not doing this reassignment is explained in Section 4.5, Re-Engineering Keys, below.

4.4.3 Delete Attributes

The process of deleting an attribute is accomplished in the Maintain Attributes screen, Section 4.4.1. Instead of Change, press Delete. If the attribute is “unconnected” with a column or meta category value then the deletion should occur.

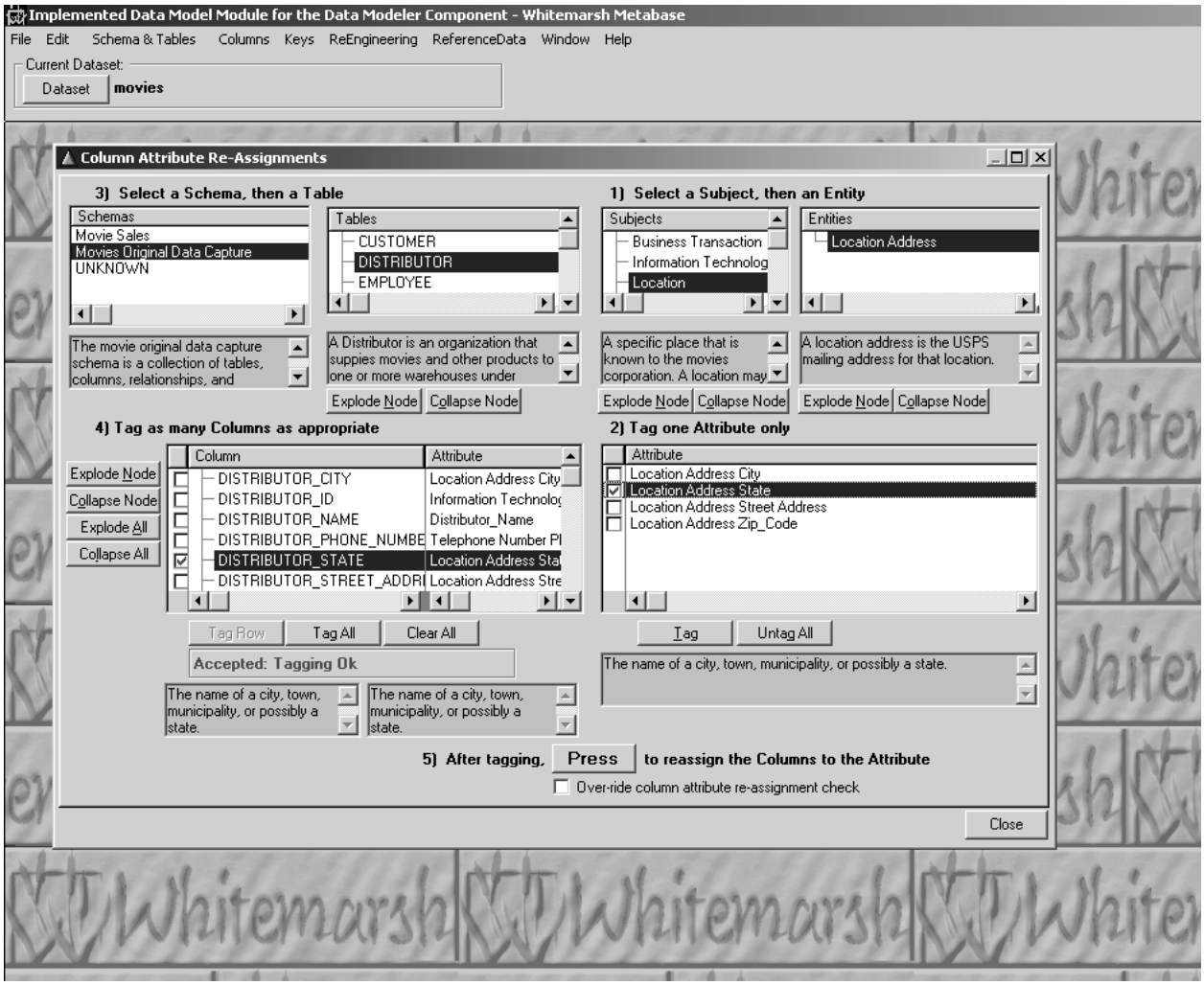

Figure 37. Column attribute reassignments.

4.5 Re-engineering Keys

An examination of the "Id" columns in the tables show that all the "Id" columns have a data type of INTEGER. That suggests that the integer number is just a numeric identifier that has no real meaning, such as a customer's name. In the Specified Data Model the primary keys should always have primary keys based on natural attributes, not "surrogate keys." A surrogate key, according to a definition from Michelle A Poolet of the SQL Server Magazine, is "an artificially produced value, most often a system-managed, incrementing counter whose values can range from 1 to n, where n represents a table's maximum number of rows." To resolve this situation the following actions need to occur:

- Remove the foreign keys that are based on the surrogate primary keys
- Remove the primary keys that are surrogate keys
- Remove unnecessary attributes
- Create new primary keys that are based on natural attributes
- Create new foreign keys

.

4.5.1 Remove the Foreign Keys

The process of removing is relatively simple. Figure 38 shows the Foreign Key screen. Select Entity Foreign Keys within the Keys menu item. Highlight the subject, then entity, and then foreign key. Then, press the Delete. If none of the foreign key's attributes are referenced by columns, AND if none of the foreign key's attributes are part of a primary key, then the foreign key definition, the foreign key references to the entity's attributes that constitute the foreign key, and the attributes themselves will all be deleted.

If however, the foreign key has attributes that are referenced by columns then the delete request will be rejected. Further, if the foreign key has attributes that are part of a primary key then the delete request will be rejected

The practical consequence of the above is that because the "Id" columns (for example, Employee Id or Customer Number) that are part of the Specified Data Model's Primary Keys and by inference part of the Foreign Keys, they must all be re-assigned to the attribute, Information Technology Attributes Identifier within Specified Data Model's entity, Information Technology Attributes.

When this is done, then all the "Id" attributes will be "free" of their references to columns. And because they are "free" the foreign keys will then be able to be deleted. The columns to be reassigned are listed in Table 3. Accomplish the reassignment through the process described in Section 4.4.2 Re-assign Column Attributes, above.

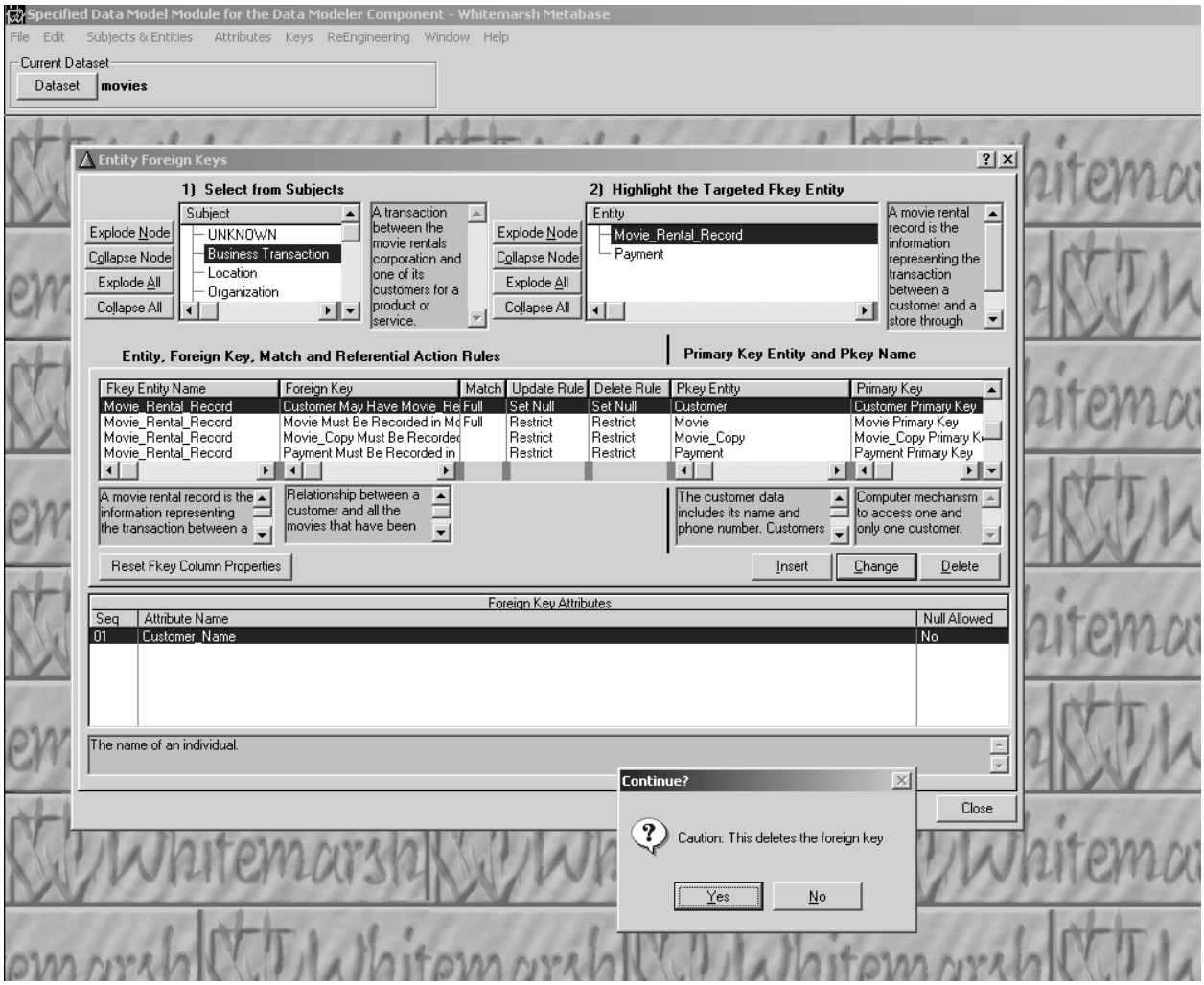

Figure 38. Foreign key delete screen.

NB: The scenario in this guide presumes that you have not moved ahead and build data elements and or used Specified Data Model entities to be referenced by any other column. If you have then the attribute will not be "truly" free. Unless and until the attribute is completely un-referenced by any column can the foreign key be deleted. If you have moved ahead and the foreign keys are not able to be deleted, you must go back and reset all the columns to an "unknown" data element, and you must reset all "Id" columns to the attribute, Information Technology Attributes Identifier.

Once the columns are re-assigned away from the "Id" attributes, delete the foreign keys.

Subject	Entity	Attribute
Business Transaction	Movie_Rental_Record	Customer_Number
		Employee_Id
		Movie_Copy_Number
		Movie_Number
		Payment_Transaction_Number
		Rental_Record_Number
		Customer_Number
		Employee_Id
		Payment_Transaction_Number
Organization	Distributor	Distributor_Id
	Store	Distributor_Id
		Store_Id
Person	Customer	Customer_Number
	Employee	Employee_Id
		Store_Id
		Supervisor
Product	Movie	Movie_Number
	Movie Copy	Movie_Copy_Number

Table 3. Surrogate Key "Id" based attributes.

4.5.2 Remove the Primary Keys

Once the foreign keys are deleted, then the primary keys supporting them can be deleted as well. Figure 39 presents the Primary Key delete screen. Delete all the primary keys.

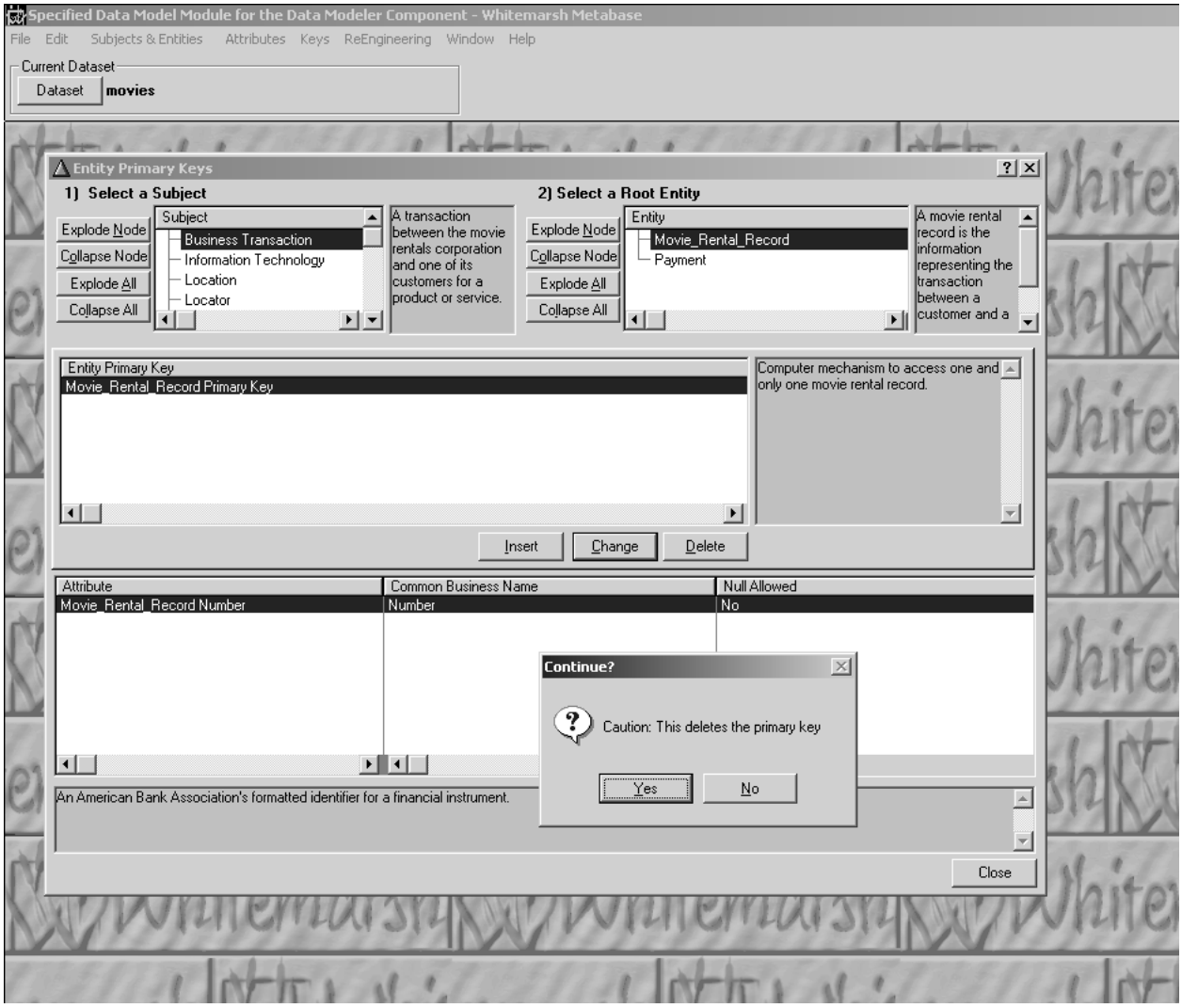

Figure 39. Primary key delete.

4.5.3 Delete Unnecessary Attributes

The process of deleting unnecessary attributes is accomplished in the Maintain Attributes menu items. It is shown in Figure 40. The attributes became unnecessary because they existed solely to support the surrogate keys that are not appropriate for the Specified Data Model. Table 3 is a guide to all those surrogate key attributes. The step after this one is to re-make the primary keys based on "natural" business data.

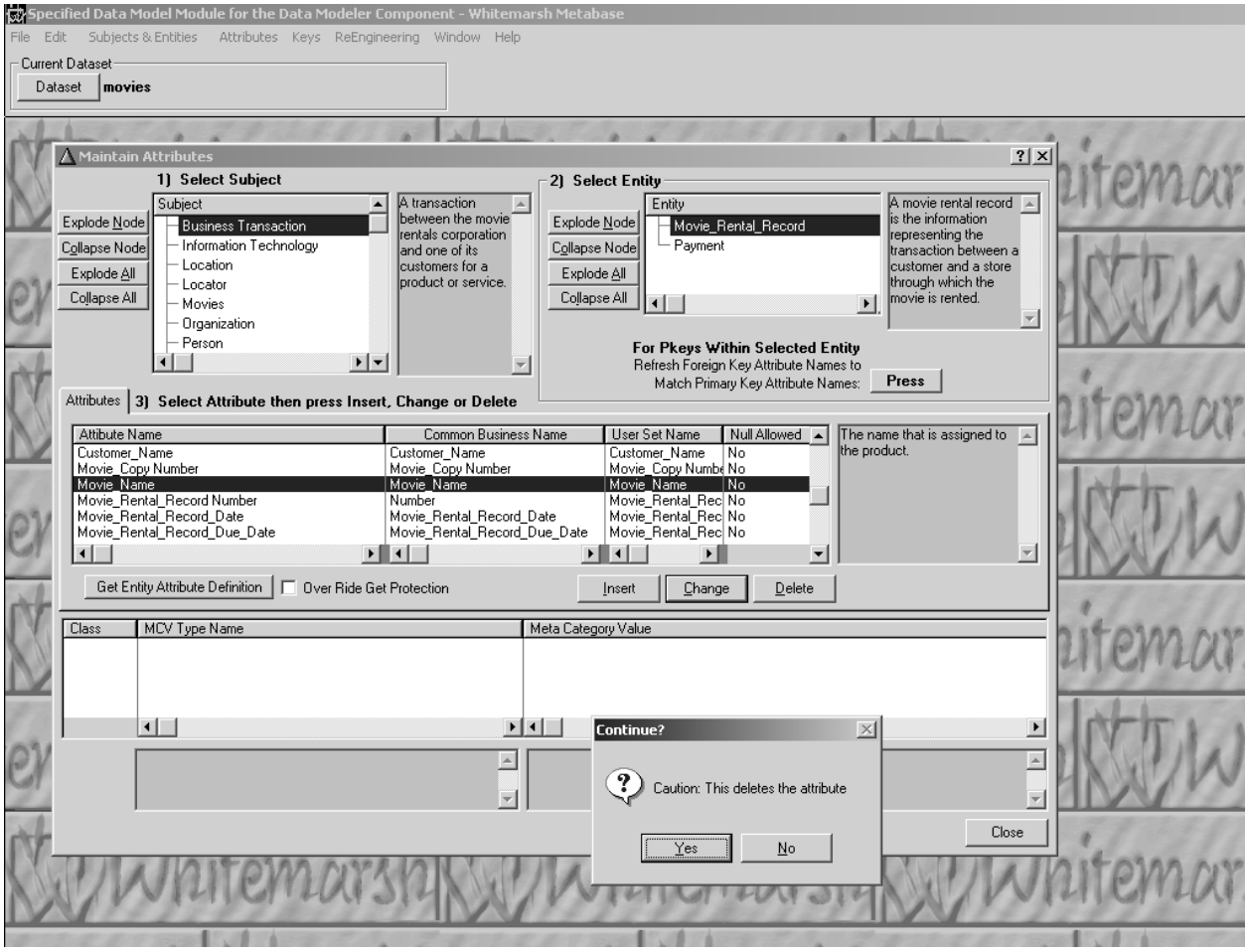

Figure 40. Delete attributes.

At the end of this step, proceed though the entities and make sure that all the attributes are truly business based. Payment, for example, is an interesting entity in this regard. Check bank number, and check number are business-based attributes because the values are from "outside" of IT. The bank number is the bank routing number. The check number is a similar type of number. So too is the credit card number. Hence, while all these attributes are numbers, they are business-based numbers whose values are controlled outside the domain of the movies system. A case could be made that the bank routing number and the credit card number are really formatted strings of all numbers and thus their data types in the Implemented and Operational Data models should therefore be character data types. That is not a "battle" germane to this example, however. And finally, there are no data types in the Specified Data Model as data types are not appropriate for this class of data model.

4.5.4 Create New Primary Keys

This step makes primary keys based on "natural" entity attributes. For example, for employee, while the surrogate key attribute was Employee Id, the "natural" primary key attribute would be Employee Social Security Number. For store, it is Store Name.

Adding a primary key is a two step process. First, add the primary key, and second, add the column(s) belonging to the primary key. The first step is displayed in Figure 41. In this step, both the primary key browse screen and the Adding a primary key overlay is shown. Select Primary Key from the Keys menu item. Select the subject, then the entity, then press Insert. The overlay then appears. If you tab through the Entity Id entry field, the Primary Key's name is automatically generated. You can change it if you wish. Add a description. The bottom browse is for the currently assigned attributes for the primary key. Since there are none, this is empty. Now, click OK and the screen closes. The newly added primary key now appears in the primary key browse list. Now proceed to add a Primary Key to every entity. The next step will be to add the columns to the Primary Keys.

The process of adding a primary key attribute is displayed in Figure 42. Select the subject, the entity, then tag the primary key. Tag then the attributes that are to appear in the primary key. Then press the Build button. If the attribute you are adding is allowed to contain Null, then an error message is presented. You must go back to Maintain Attributes and change the null-allowed indicator to "No."

A review of Figure 4 shows that the independent entities are really: Movie, Customer, Distributor and Store. Entities Movie Copy, Payment, and Movie Rental Record do not have natural attributes to then make primary keys. So, you must make new attributes, such as Movie Copy Number for Movie Copy, Payment Number for Payment, and Movie Rental Record Number for Movie Rental Record. These are not IT numbers, but are numbers assigned by the store that would be printed on the rental record contracts, payment slips, or recorded onto the movie copies to distinguish them.

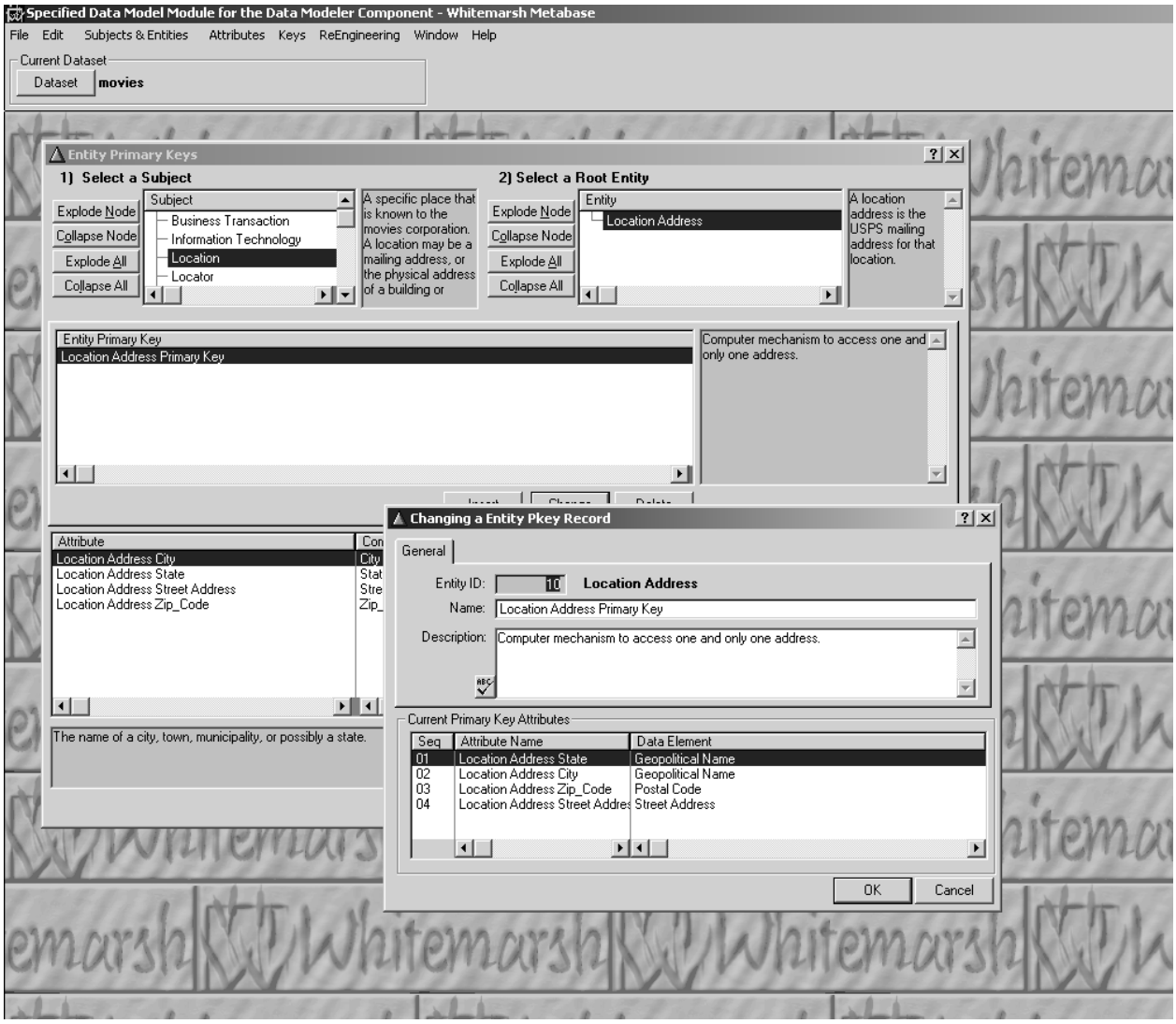

Figure 41. Primary key creation.

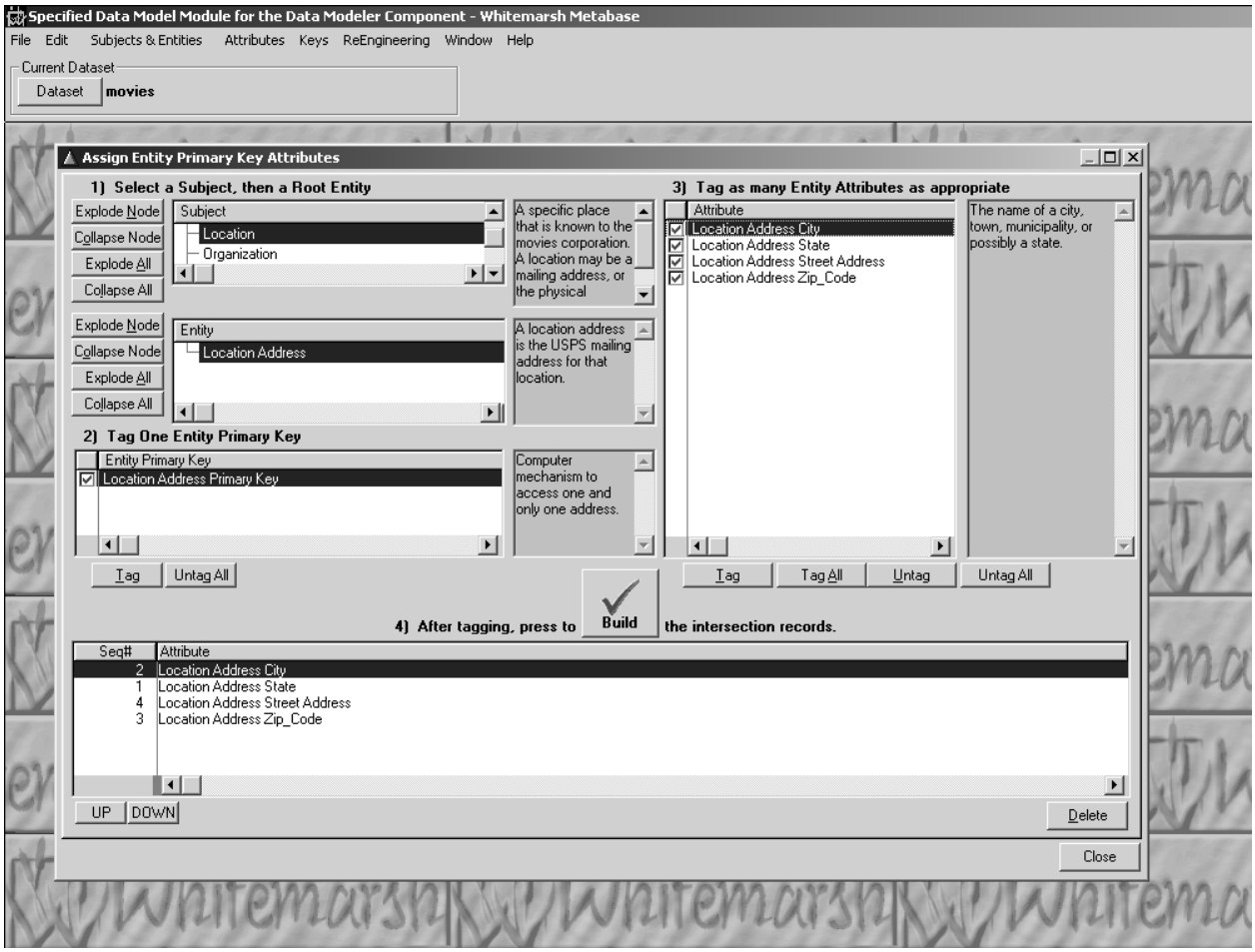

Figure 42. Primary key attribute creation.

Store is not dependent merely because the relationship between Distributor and Store. Rather, the relationship is based on one of "servicing." A distributor services a particular store.

The primary keys of the two new entities should also be created. That is, Location Address, and Telephone Number.

4.5.5 Create New Foreign Keys

The process of making a foreign key is as follows:

- Select the target entity
- Pick the primary key of the source entity
- Create an Action Word
- Select the proper Referential Action.

At that point, the foreign key is constructed. Constructed too are the source entity's primary key's attributes within the target entity that then serve as the foreign key's attributes. Finally constructed are the relationships between the foreign key and its attributes, and between the primary key and this newly created foreign key.

The foreign key add process begins by selecting the Foreign Key menu item within Keys. When the Foreign key browse comes up, as shown in Figure 43, select the subject, then target foreign key, then press the Insert button. If after pressing the insert button you cancel the operation, then a "fake" foreign key will have been made. To get rid of the "fake" foreign key, just reselect a different target entity and then select the original one. The "fake" foreign key will then disappear from the database.

After the Insert button is pressed, Figure 44 is presented. Tab through the Primary Key entry box which shows a "0." At that point, Figure 45, the primary key select screen is presented. It enables you to select the source entity's primary key. Select the subject and then the primary key. Press the Select button, then press Close. Once the primary key is selected you are returned to Figure 44. You can then enter an "action phrase" so that the foreign key's name is automatically created. When you add the action phrase, the software automatically creates a Foreign Key Name. The metabase constructs it as follows:

< source entity > < must|may > < action phrase > < target entity name >

In this particular example, if the action word was "Have" then the foreign key name would be:

Customer Must Have Movie Rental Record.

The three remaining items are Key Match options, Unique indicator, and Referential Actions. Key match options are full or partial. The unique choice causes the relationship to either be one-to-one or one-to-many. Referential actions are listed. If the choice is Set Null then the <must|may> is not must, but may.

Once all these choices are made, press the button, "Press to Create Foreign Key or to Commit Changes" Once pressed, press the Cancel/Close button. It is essential that you commit the changes. Otherwise if you merely Cancel/Close then the "fake" Fkey will be created that gets deleted in the manner described above.

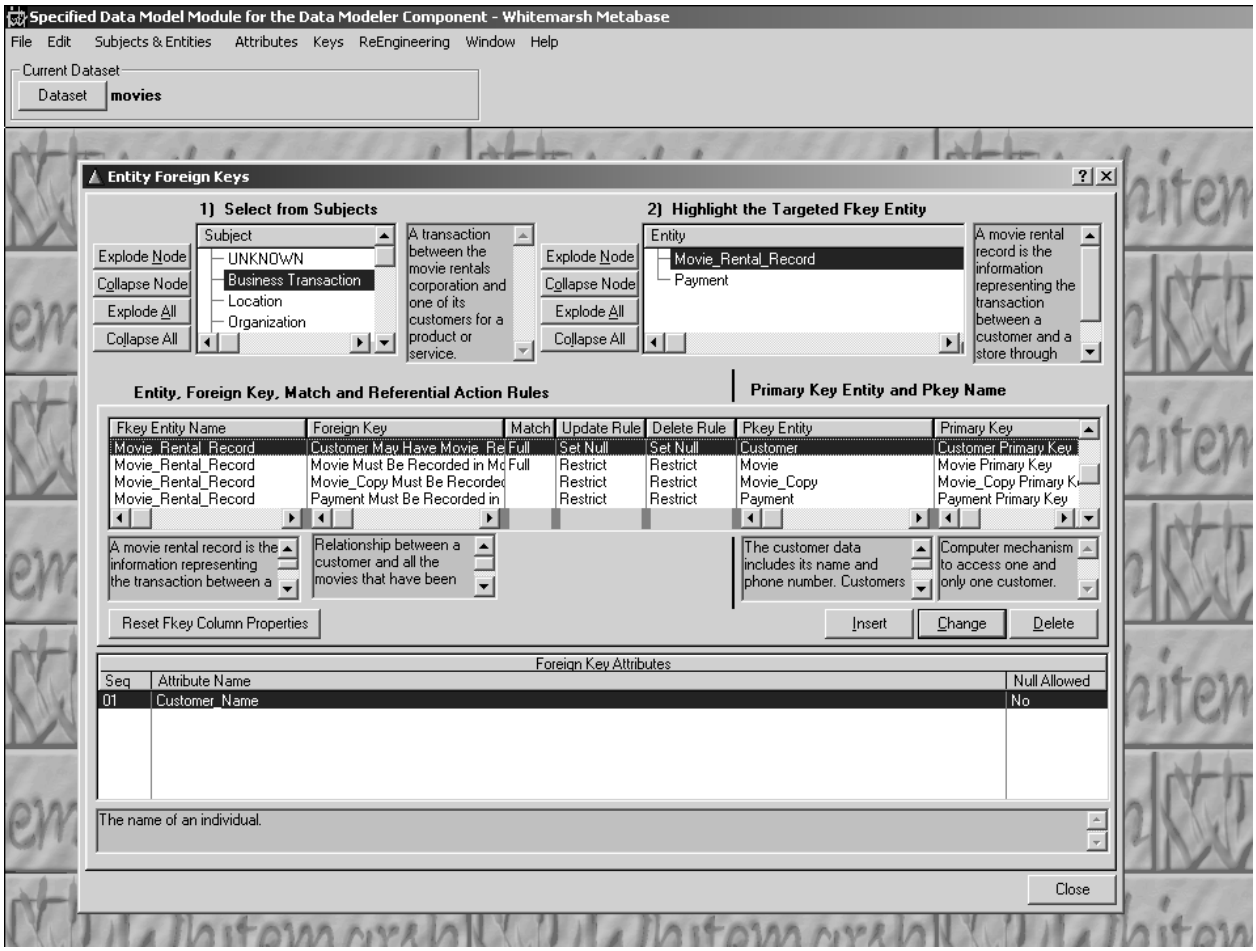

Figure 43. Foreign key add.

Now, for the purposes of this example, create the following foreign keys:

- Customer May Have Movie Rental Record
- Customer May Make Payment
- Movie must Be Recorded in Movie Rental Record
- Movie May Have Movie Copy
- Movie Copy must Be Recorded in Movie Rental Record
- Payment must Affect Movie Rental Record
- Store May Have Employee
- Employee May Make Payment
- Employee May Supervise Employee
- Distributor May Service Store
- Distributor Must Have Address

- Store Must Have Address
- Customer Must Have Address
- Employee Must Have Address
- Distributor Must Have Phone Number
- Store Must Have Phone Number S
- Customer Must Have Phone Number
- Employee Must Have Phone Number

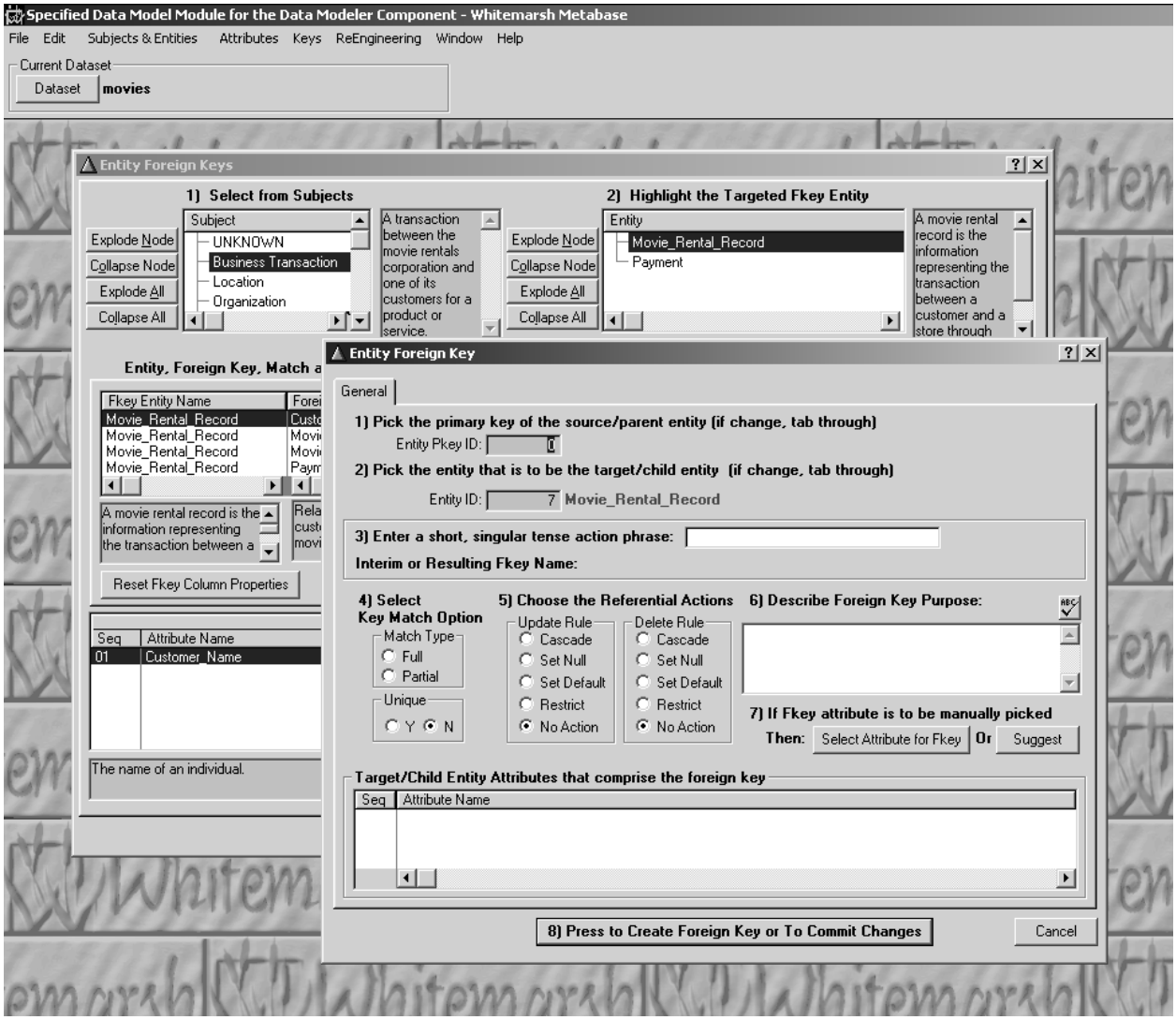

Figure 44. Foreign key add form.

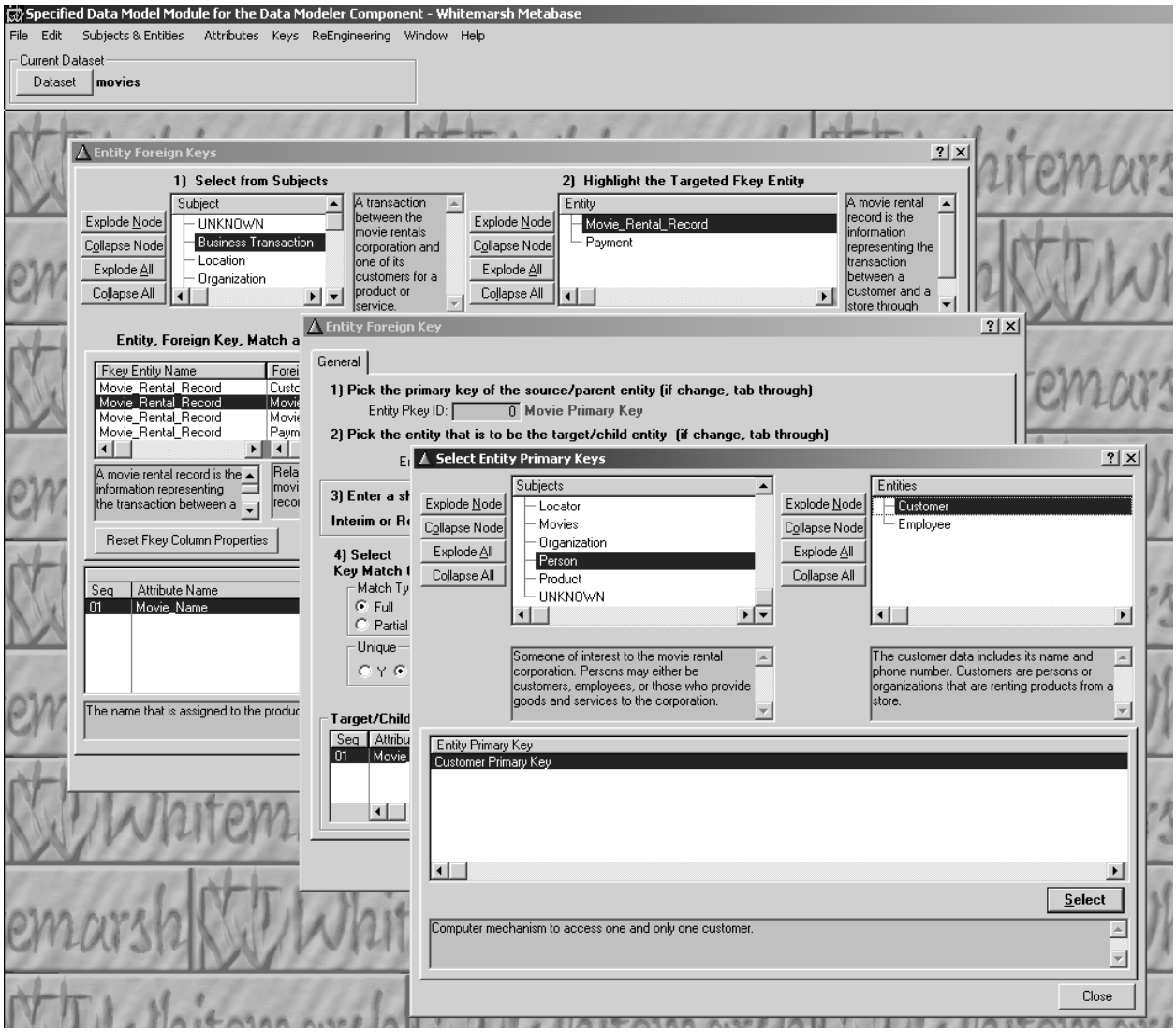

Figure 45. Foreign key add: primary key select.

4.6 Maintain Attributes

Prior to leaving the Specified Data Model, you should give it a final run-through and make sure of the following:

- All entities have a definition and abbreviations.
- All attributes have a definition and abbreviations.
- All attributes have the right business basis for allowing or disallowing null.
- All primary and foreign keys have the proper name and definition.
- All attributes in both primary and foreign keys are in the correct major to minor sequence.

5.0 Data Element Creation

The next step in the overall reverse engineering process is to make data elements. First, what is a data element? Is it a column in a table? Is it an attribute in an entity? Is it a field in a File? Is it a data entry or display object on a screen? No, no, no, and no. A data element is none of those. A data element within the scope of ISO standard 11179 is a stand alone, context independent business fact template. Nothing more and nothing less.

To put it more succinctly, a data element is a semantic rubber stamp. If you need to "brand" a column in a table with the semantics of a data element, just stamp it. Thus, the column is not the stamp. It employs the stamp for the essence of its meaning. Similarly, if you need to "brand" an attribute in an entity with the semantics of a data element, just stamp it, or if you need to "brand" a field in a file with the semantics of a data element, just stamp it. And finally, if you need to "brand" a data entry or display object on a screen with the semantics of a data element, just stamp it. Thus, a data element is a semantic layer "above" all of these semantic uses. Again, the column, for example, is not the stamp; it employs the stamp for the essence of its meaning and thus, the stamp (i.e., the data element) is not "used up."

It is critical to know whether somebody means a real data element or just a column, attribute, or field during a discussion. When they start talking about data elements, ask the simple question: If you have 10 database tables and each has 10 columns, how many data elements will you have? If the person answers, 100, then they believe that a data element is a column. If they say something to the effect that they don't know because it is unclear how many times a given data element has been employed to provide semantics to the column, then you are speaking with someone who understands the power of the data element.

To illustrate the benefit of a data element, Table 4 shows the reuse of the data elements that will be built in this Section of the paper and the attributes that are "branded" by their semantics. To the right of each attribute is its entity. The ration for just this simple database is almost 2:1. Building data elements is not easy. It takes analysis, reflection, and clear thinking. Additionally, building data elements is not exactly the same skill set as building data models. It requires critical thinking about concepts, about the concepts surrounding value domains, the construction of value domains, and the construction of the data element concepts.

The value from building data elements starts with re-use. In the little example in this paper, if the data elements were accomplished first then the definition of about 50% of the attributes would have been less. If it takes about 2 hours to define a data element, and each is employed two hundred times across an enterprise's set of databases then you've just saved four hundred hours of analysis and design. So, for about $200 of investment you may have saved $40,000 effort. Not a bad return on your investment.

Extending that a bit further, an average enterprise has about 100 databases. Each has about 150 tables. Each table has on average about 10 columns. That's 150,000 columns. If a data element is a column and each takes about 2 hours to design and define, that's about $30 million. Now, if

alternatively a data element maps to about 100 columns then the quantity of data elements is about 1500, and the cost is about $300,000. Still sizable, but certainly less than $30 million. It should seem that no more justifications or arguments need be made.

Data Element	Attribute	Entity
Assessment	General_Condition	Movie_Copy
	Rating	Movie
Communications Locator String	Telephone Number Phone_Number	Telephone Number
Event Date	Employee_Hire_Date	Employee
	Movie_Date	Movie
	Movie_Rental_Record_Due_Date	Movie_Rental_Record
	Payment_Credit_Card_Expiration	Payment
	Payment_Date shit head	Payment
Financial Amount	Movie_Rental_Rate	Movie
Financial Institution Identifier	Check_Bank_Number	Payment
Financial Instrument Identifier	Check_Number	Payment
	Movie_Rental_Record Number	Movie_Rental_Record
	Payment Number	Movie_Rental_Record
	Payment Number	Payment
Geopolitical Name	Location Address City	Location Address
	Location Address State	Location Address
Government Institution Identifier	Employee_Social_Security	Employee
	Employee_Social_Security_#	Payment
	Supervise Employee_Social_Security	Employee
Identifier	Information Technology Attributes Identifier	Information Technology Attributes

Data Element	Attribute	Entity
Organization Name	Distributor_Name	Distributor
	Distributor_Name	Store
	Store_Name	Employee
	Store_Name	Store
Payment Mechanism Name	Payment_Credit_Card_Type	Payment
Payment Mechanism Number	Credit_Card_Number	Payment
Payment Mechanism Type	Payment_Type	Payment
Person Name	Customer_Name	Customer
	Customer_Name	Movie_Rental_Record
	Customer_Name	Payment
	Employee_Name	Employee
	Movie_Director	Movie
	Movie_Star	Movie
Postal Code	Location Address Zip_Code	Location Address
Product Description	Description	Movie
Product Name	Movie_Name	Movie
	Movie_Name	Movie_Copy
	Movie_Name	Movie_Rental_Record
Property Identifier	Movie_Copy Number	Movie_Copy
	Movie_Copy Number	Movie_Rental_Record
Street Address	Location Address Street Address	Location Address
Transaction Amount	Payment_Amount	Payment

Data Element	Attribute	Entity
Transaction Date	Movie_Rental_Record_Date	Movie_Rental_Record
	Movie_Rental_Record_Overdue_Charge	Movie_Rental_Record
	Movie_Rental_Record_Rate	Movie_Rental_Record
Transaction Status	Movie_Rental_Record_Status	Movie_Rental_Record
	Payment_Status	Payment

Table 4. Data element reuse within entities of the Movies metabase.

But more importantly, data elements are the corner stone for data interoperability. It is from the corner stone of the data element that attributes, and then columns, and then DBMS columns, and finally SQL view columns proceed. Each layer may be more refined but it's essential meaning proceeds from the data element. Consequently there is traceability across entities, tables, DBMS tables, views and then into the actual information system. Figure 46 makes illustrates the concept of the data element being the cornerstone.

This example, from the ISO 11179 data element metadata framework, starts with the premise that the logistics facts are to be represented across the data model layers, that is, specified, implemented, and operational. In this example, a logistic fact starts with a fundamental concept, Materiel Resource. That is, that the facts are going to be related to the characteristics of materiel resources. The next step is to then identify the concepts that will form the foundation for the various types of value domains. Note in the diagram, that conceptual value domains are independent of the concepts. Thus, conceptual value domains can be associated with many different concepts.

In this example, the concept behind the value domain is a physical measure. There could be many types of physical measures, for example, quantities, size, weight, and the like. When a concept is combined with a conceptual value domain, the ISO 11179 result is a data element concept. Then, when the data element concept is, in turn, combined with a more refined value domain a data element results.

Thus, in this specific example, the data element concept, Physical Item Balance is combined with a type of value domain, quantity, to then produce a Supply Item Resource Quantity data element. There could be many more data elements proceeding from this example such as Physical Item Reorder Quantity, Physical Item Quantity on Hand, Physical Item Minimum Allowed Quantity, and the like. All these data elements proceed from the one concept, from the one conceptual value domain, and from the one data element concept. Again this is an example of define once and use many times. With this strategy you could quicky find in any information system all the uses of material resource data. Or a subset dealing with physical item balances, or those dealing with quantities in some way. These levels of abstraction are very powerful.

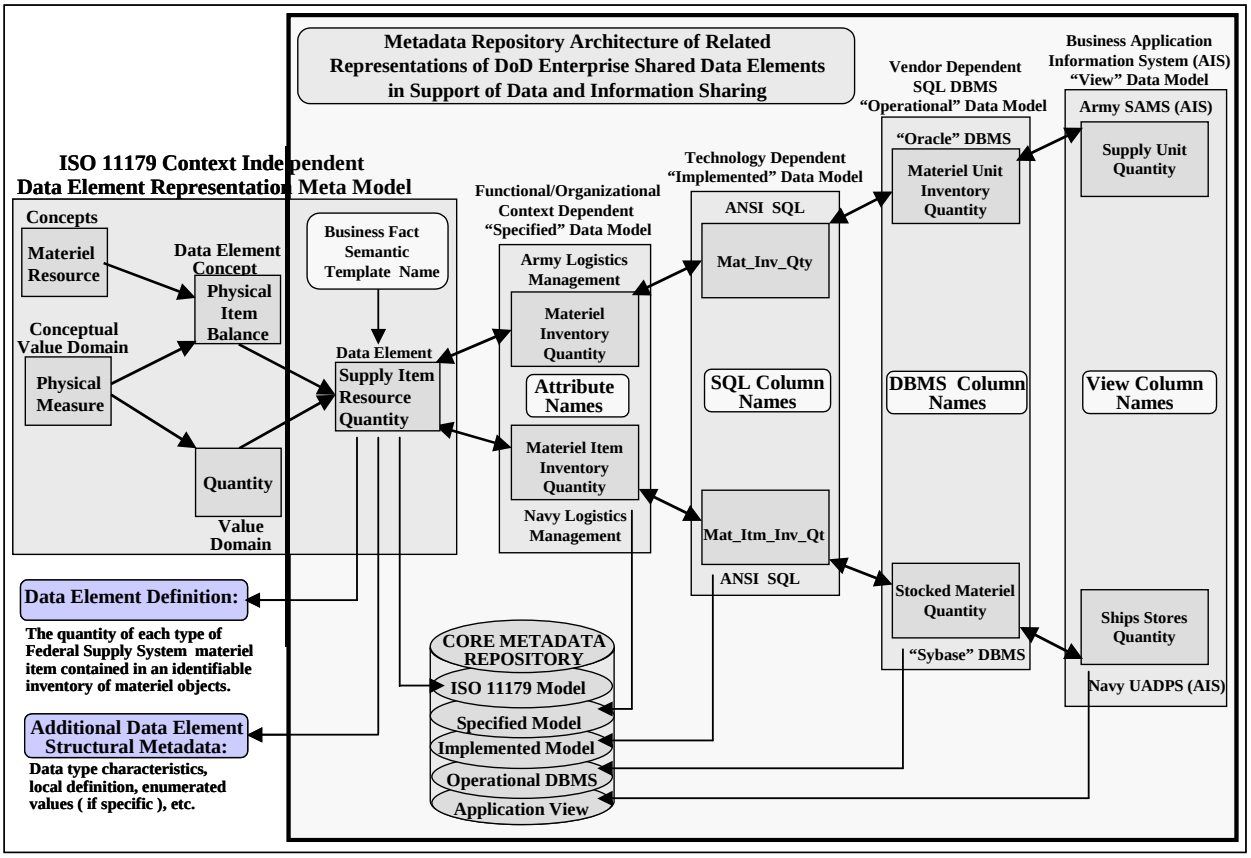

Fig-11

Figure 46 Logistics example about the power of data elements.

While this use of data elements is important, a number of additional critical points need to be conveyed. In this particular example, the ISO 11179 based shared data element, Supply Item Resource Quantity, has been defined and is available as a unitary fact template for use in creating data models. The first layer of data model is the Specified Data Model, and within that data model, there is some entity that has supply item resource quantity as the template for the two attributes, material inventory quantity, and material item inventory quantity. Note that the names are different not only with respect to the ISO 11179 based shared data element, but also from each other. That's perfectly acceptable because what these two attributes really are is Supply Item Resource Quantity.

Further, note that the systems employing this ISO 11179 based shared data element are from the Navy and the Army. The significance of this is that the Army and Navy are free to employ their own particular business area languages/dialect terms in naming their specified, implemented, operational, and view deployments of ISO 11179 data elements. And, since legacy systems have already done that, this metadata strategy will accommodate those existing differences as opposed to requiring those legacy schemas to be re-engineered at most likely great expense and disruption to business operations

The next level of data model, the Implemented Data Model for the Army, has a table that contains the column, Mat_Inv_Qty. For the Navy, the column is Mat_Itm_Inv-Q. Not only are these names different one from the other, they are also different from their "parent" attribute names. Again this is perfectly acceptable because these names are all mapped, thus traceable back to their source. The Operational Data Model level also shows a similar pattern as does the View Model layer that supports the business information system.

This approach works quite acceptably for legacy systems. Their database schemas already exist at the view and Operational Data Model levels. These schemas and views are imported into the metabase. Given that the ISO 11179 and the Specified Data Model layer of shared data segments have been populated, then the task that remains is the construction of the Implemented Data Model layer that maps one to the other.

Once these are built and mapped, then there is semantic mapping across legacy systems, between legacy systems of different Operational Data Models and their common Implemented Data Model, and from Specified Data Model templates of standard data structures across all implemented and Operational Data Models. Finally, there exists mapping from concepts and conceptual value domains to all data element concepts, and from data element concepts to all ISO 11179 based shared data elements, and finally, from all ISO 11179 based shared data elements to all uses of that shared data element across all uses within information systems within the inventory of information systems.

The data element layer, thus, represents the corner stone of data interoperability. From this cornerstone the layers proceed outward. But more importantly, an operational system's DBMS table columns can trace back to the progenitor data elements. And, when two such systems accomplish this, the common hits represent potential areas of interoperability. The precision and granularity of the DBMS tables still need to be examined. But at least it is a start.

The propose of this section then is to set out how to build the data element layers in such a way that the resulting metadata can be used over and over again. Further, as new Operational Data Model systems are brought into this environment, two things will start to become clear. First, their Specified Data Model templates, seen through Implemented Data Models, will likely already be sufficient, and thus, no new ones will have to be created. Similarly, the data element layer will also be sufficient. Thus, to bring an Operational Data Model into a state of interoperability will mainly be just a matter of importing Operational Data Models building and building mappings to already existing implemented, specified, and data element layers.

In the prior sections, the process proceeded from operational to implemented to specified. In this section, however, the material proceeds from the "top" of the ISO 11179 data element model down through the creation of the data elements. The section then proceeds to connect attributes to the data elements. Finally the section concludes by synchronizing the Implemented Data Model columns to the data elements. The steps are thus:

- Build Concepts

- Build Conceptual Value Domains
- Build Data Element Concepts
- Build Value Domains
- Build Data Elements
- Connect Specified Data Model Attributes to Data Elements
- Synchronize Implemented Data Model Columns to Data Elements

The Data Element data model diagram is presented in Figure 47. The "story" of ISO 11179 data element metadata is that concepts can either be simple or complex. Regardless, concepts within their structures are the source for the concepts supporting data elements, that is, data element concepts. Similarly, the concepts supporting value domains, that is conceptual value domains can also be either simple or complex. Regardless, conceptual value domains are similarly the conceptual source for the conceptual value domains supporting data element concepts. Conceptual value domains are also able to be represented more precisely by value domains. Value domains, which too can be simple or complex are able to be related to each other for the purposes of tracking evolution and transformations. Data element concepts can be simple or complex. Together, value domains and data element concepts form the basis for one or more data elements.

In ISO standard 11179, Concepts are called Objects, and Conceptual Value Domains are called Conceptual Domain. A close reading, however, of the standard will lead you to see that the names provided in this document and in the metabase system, that is, Concepts (for objects) and Conceptual Value Domains (for Conceptual Domains) are more reflective of the real intent.

So, other than for a renaming, this model is a faithful third normal form database implementation of the conceptual model that resides in the ISO 11179 Standard, Part 3.

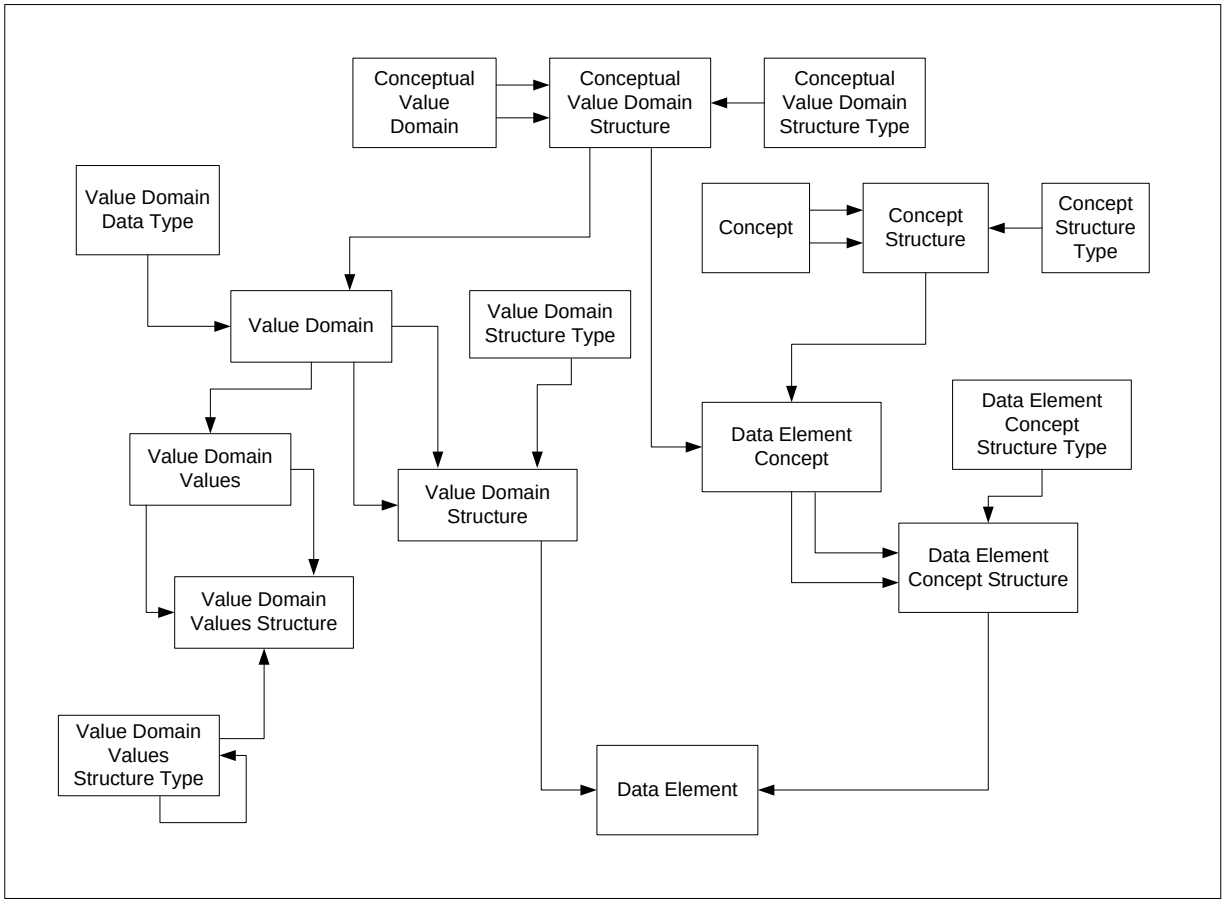

Figure 47. Data Element Data Model diagram.

5.1 Build Concepts

The process of building concepts consists of the following steps:

- Building concepts
- Building concept structure types
- Building conceptual structures

The three part construct for concepts enables concepts to be either simple or complex. By simple, the concepts are either single or exist in hierarchies. By complex, concepts can be engineered to be complete networks of concepts. In the example from this guide, concepts are just two level hierarchies. The purpose of the conceptual structure type is to label the relationship that the concept members have with one another within a collection of concept

structure records. For example, containment. Each relationship has two forms for the names: passive and active. A passive name is, for example, “is contained in,” while the active form is: “contains.” This enables the production of reports that “tell a story.” For example, if there are three concepts, Asset, Real Asset, and Abstract Asset, bound together through the containment relationship, then the relationship sentences able to be produced are:

- Asset contains real asset
- Asset contains abstract asset
- Abstract asset is contained in asset
- Real asset is contained in asset

5.1.1 Build Concepts

Figure 48 presents the Concepts browse screen. To get the screen, activate the Data Element Module, select the data set, and then choose, Concepts under the Concepts main menu item. To insert a new concept, press Insert. Figure 49 will then appear. Provide the appropriate name and definition, and then press OK. The newly added concept appears in the browse list.

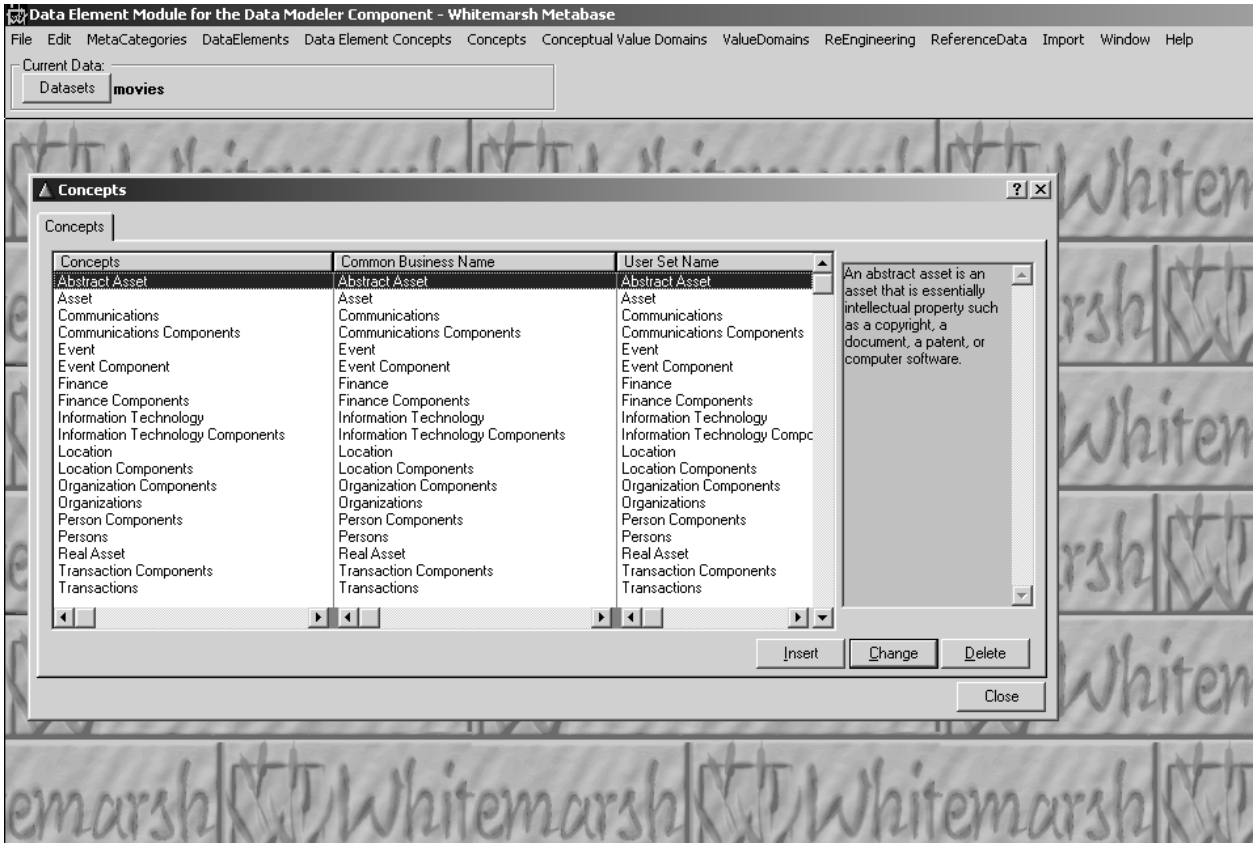

Figure 48. Concept browse.

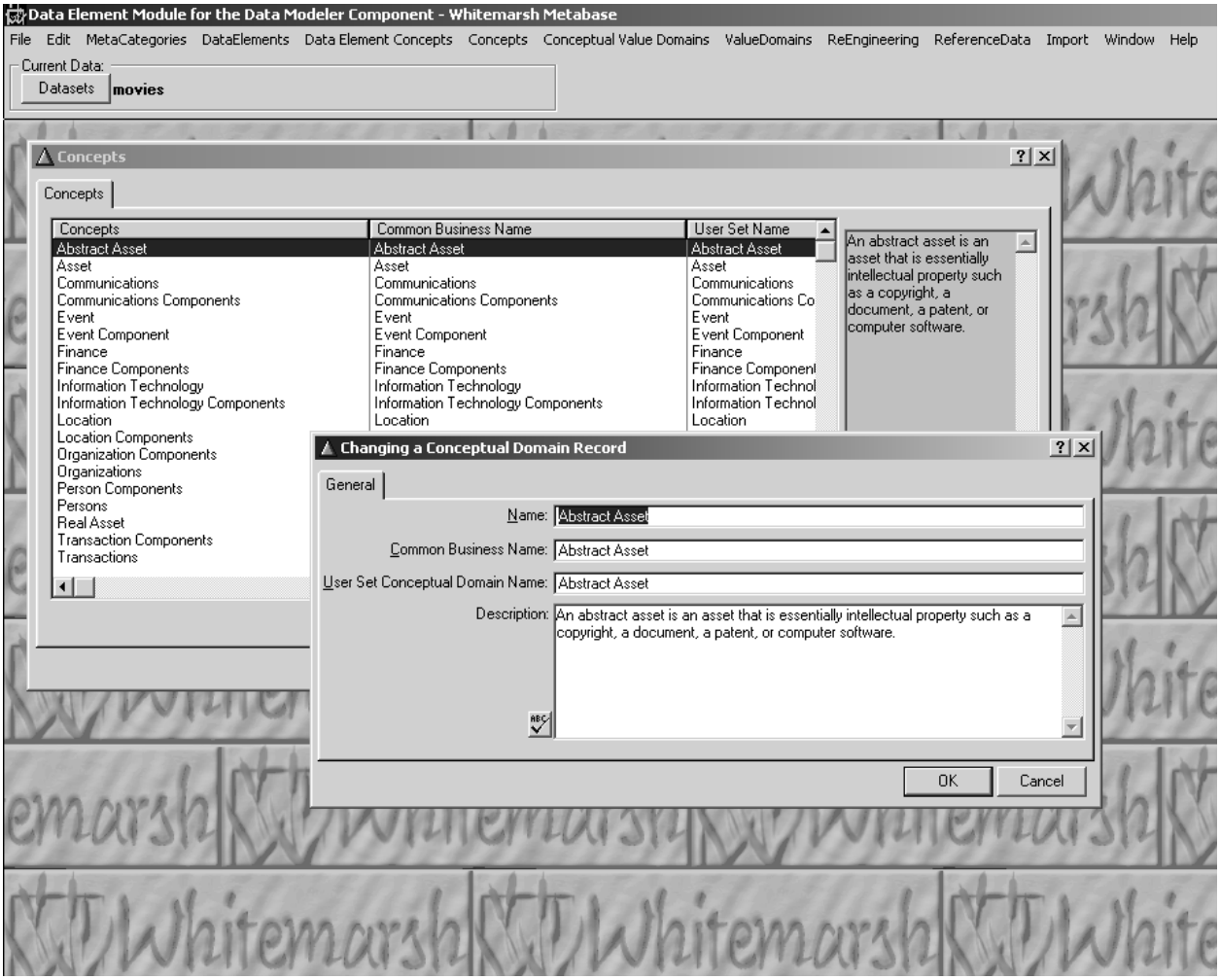

Figure 49. Concept update screen.

That's the mechanical process of creating and entering concepts. But more importantly, "Where do concepts come from?" A Google search produces only about 30 hits, including one from the works of Mao Tse-tung From the 8th Party congress, The First Speech May 8, 1958 (http://www.marxists.org/reference/archive/mao/selected-works/volume-8/mswv8_10.htm). While certainly interesting, a more practical source is from the lecture notes of James Hampton. His web page, http://www.staff.city.ac.uk/hampton, contains links to his works. In these notes he suggests a classical explanation of concepts, and his explanation seems quite on point.

Hampton states that "A concept is a class of things which all have certain attributes in common, Everything which is in the class must possess all these attributes. Everything which possesses all these attributes must be in the class." He further states in these notes that each instance of a concept has the following characteristics:

- Each class at any level has no overlap with any other at the same level
- Every individual object must belong to one and only one class at each level
- There are no residual classes ("things that don't fit elsewhere")
- Everything that is true of Vertebrates is also true of all the classes at every level below Vertebrate which are subordinate to it, and true of all the individuals in those classes
- If an object belongs to a particular class, it also must belong to all the classes that are superordinate to that class

Hampton then indicates in his notes that benefits are derived from this classical definition. That is,

- Taxonomic Structure. Subsets in the tree are mutually exclusive and jointly exhaustive of the next class up. A "clean" way to divide up the world
- Efficient Storage – each concept needs only its link to a superordinate plus its distinctive attributes
- Logical inferences – taxonomies support many simple deductive inferences, including negative conclusions such as that no horses are donkeys, no horses are reptiles etc.
- Solid basis for semantics of natural language, if language is to support rationality

Finally, Hampton suggest that “If we examine the concepts that best fit the classical model, they are often those which we learn through our culture - scientific and mathematical concepts learned at school, legal definition of crimes as felonies or misdemeanors, or kinship concepts learned by each group of language speakers for their own language.”

The only exception to the above is that Hampton seems to define a concept within the context of a “thing.” That seems too “real.” Some of the concepts that have been created for the purpose of this example are definitely not “real” in the sense of a “real thing” such as an automobile or a person. Some of the concepts are purely abstract. However, as the process of creating data elements proceeds from concepts to data element concepts to data elements, the “realness” increases.

Given the guidance from James Hampton, Table 5 contains an enumeration and description of the concepts that seem to related to all the different subjects, entities, and attributes within the Movies domain. The goal here is to have a place for everything and everything in a place. This process is iterative and changes during the construction of these conceptual hierarchies/networks.

It is important that the concepts be comprehensive and inclusive rather than being required to be exactly correct and absolutely precise. Once hypothesized it’s ultimately a matter of adjusting the concepts and their interrelationship with each other until there’s a rational and reasonable place for everything.

Concept	Concept Description
Abstract Asset	An abstract asset is an asset that is essentially intellectual property such as a copyright, a document, a patent, or computer software.
Asset	A component of the enterprise that has value. An asset may be either real or abstract.
Communications	Various modes of communications among parties and organizations.
Communications Components	Different components associated with communications.
Event	An occurrence of an activity about which characteristic data is recorded.
Event Component	Some well bounded part of an event.
Finance	Various aspects of financial activities and transactions among organizations and partites.
Finance Components	Various components associated with finance.
Information Technology	Various aspects of information technology dealing with computers, systems, databases, and the like.
Information Technology Components	Varous components dealing with information technology.
Location	A specific place at which an event occurs or a facility exists or person resides.
Location Components	Some well bounded characterisitc or property of a location.
Organization Components	Various components associated with an organization.
Organizations	Various aspects associated with formal and informal organizaitons such as companies, agencies, or groups.
Person Components	Some well bounded aspect of a person.
Persons	A human engaged in a process or event, or related to a process or event.
Real Asset	A real asset is an asset that is real in nature. That is, property, machinery, furniture and fixtures, and buildings.
Transactions	Well bounded activities that ususally represent some aspect of commerce.
Transaction Components	Some well bounded part of a transaction.

Table 5. Concepts related to the movies domain.

5.1.2 Build Concept Structure Types

Concept Structure Types provide the “glue” that holds siblings together across an identified collection of concepts within another concept that exists at a higher level. Examples of level phrases include:

Phrases	
Active	**Passive**
Includes	Is included in
Contains	Is contained in
Directs	Is directed by
Specifies	Is specified in
Controls	Is controlled by
Supports	Is supported by

Table 6. Levels phrases

Figure 50 presents the browse for Conceptual Structure Types. To insert a new phrase-set, press Insert. Figure 51 is presented. Add its name, active and passive phrases, and then a description of the basis for its use.

5.1.3 Build Concept Structures

The main purpose of a concept structure is to proceed from generalized to specialized. This is illustrated in Table 7. Data Element Concepts then proceed from the leafs of Concepts contained within their structures.

Concept	Contained Concept
Asset	Abstract Asset
	Real Abstract
Communications	Communications Components
Event	Event Component
Finance	Finance Components
Information Technology	Information Technology Components
Location	Location Components

Concept	Contained Concept
Organizations	Organization Components
Persons	Person Components
Transactions	Transaction Components

Table 7. Concepts within Concept Structures

Building the Concept Structures is slightly more complicated than just adding a new set of data. The process involves deciding whether this is a new root-structure concept or one that is contained within another. If it is contained, then the process of picking the right concept has to be done somewhat carefully. For example, you should not try to insert a concept as a child of itself. Nor should you include the same concept more than once in a set of siblings. That would cause redundancy (e.g., A contains B, and A contains B). Nor should you choose a concept that already exists in a higher level. That would cause in infinite recursion (e.g, A contains B contains A). These cases are ruled out via the software which chases up the trees looking for these classes of errors.

Figure 52 presents an explosion browse for concepts. Select the Concept Structure Type, and then the Concept. In this example, note that in this browse that Asset contains both Real Asset and Abstract Asset. In Figure 53, however, which displays an implosion browse, it shows that Abstract Asset is contained in Asset. Hence the names, Explosion and Implosion.

Figure 54 represents an insert attempt. To add a new concept within Asset, highlight Asset and then press the Insert button on an Explosion browse. Figure 54 is then displayed. What is presented is a complete collection of all concepts that are eligible for inclusion in the concept structure. In this particular example, Abstract Asset is highlighted. Note the error, that is, that the “Select disabled. Attempting to insert a twin.” That is because Abstract Asset is already within the structure. If you had highlighted Asset, then the error message would have read, “Select disabled. Attempting to insert the same as selected.” Finally, highlight Abstract Asset and press Insert. Then select Asset. The error message then reads, “Select disable. Insert will result in recursion.”

Deleting a concept from a concept structure is relatively straight forward. Highlight the Concept and press the Delete button. If the concept can be safely deleted then it will. Otherwise an error message will be displayed. Because of strong referential integrity you are only allowed to delete ends of structures.

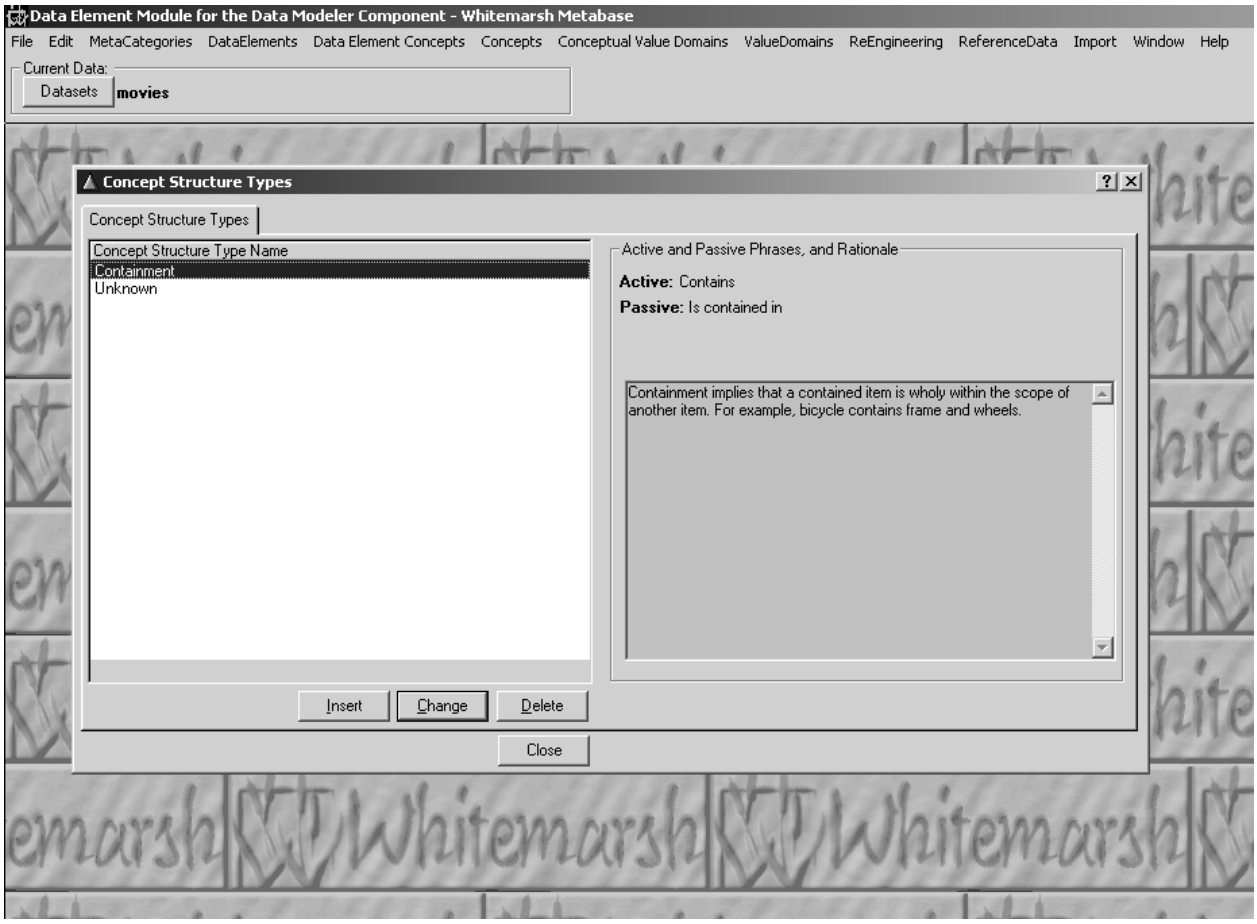

Figure 50. Concept structure type browse.

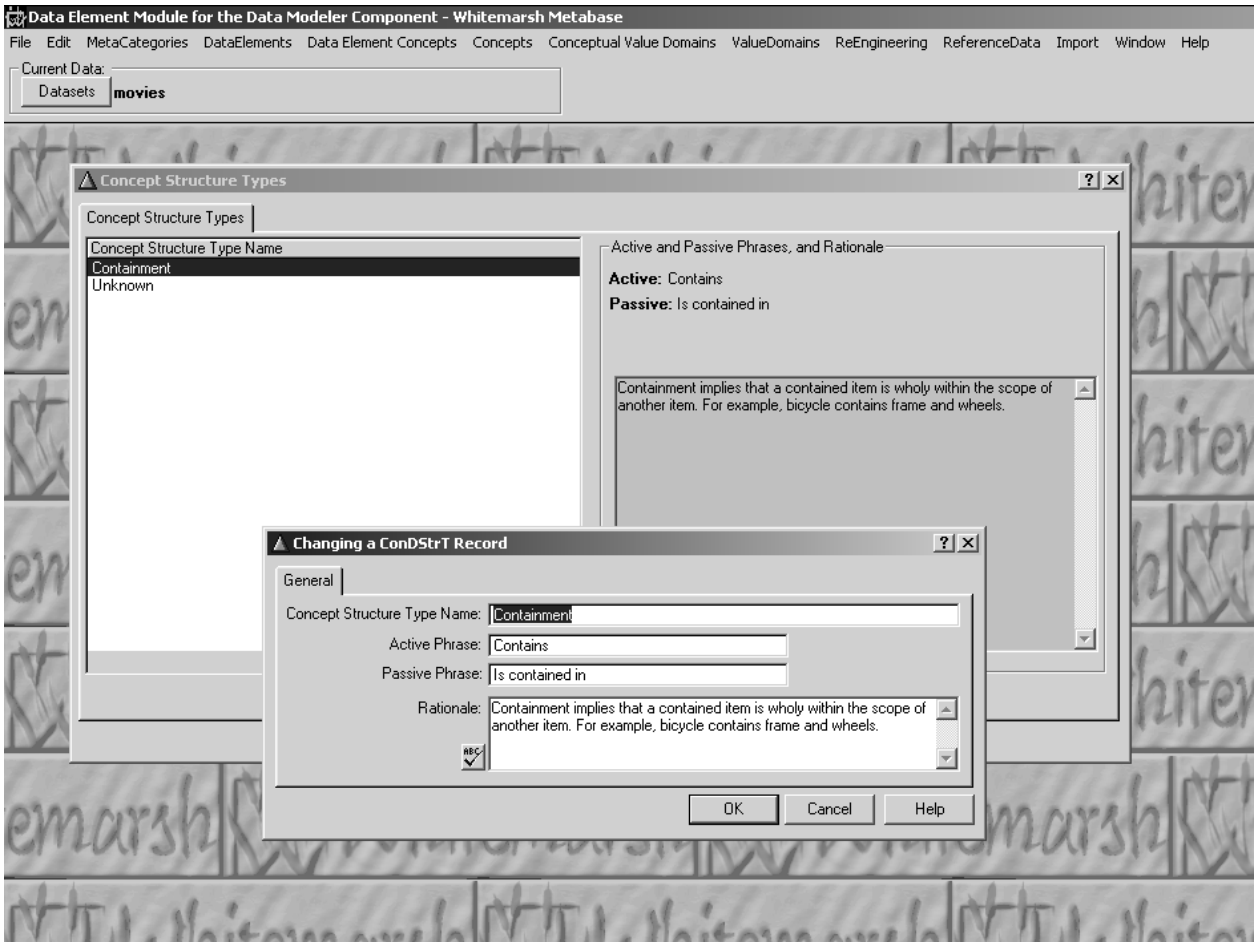

Figure 51. Concept structure type update form.

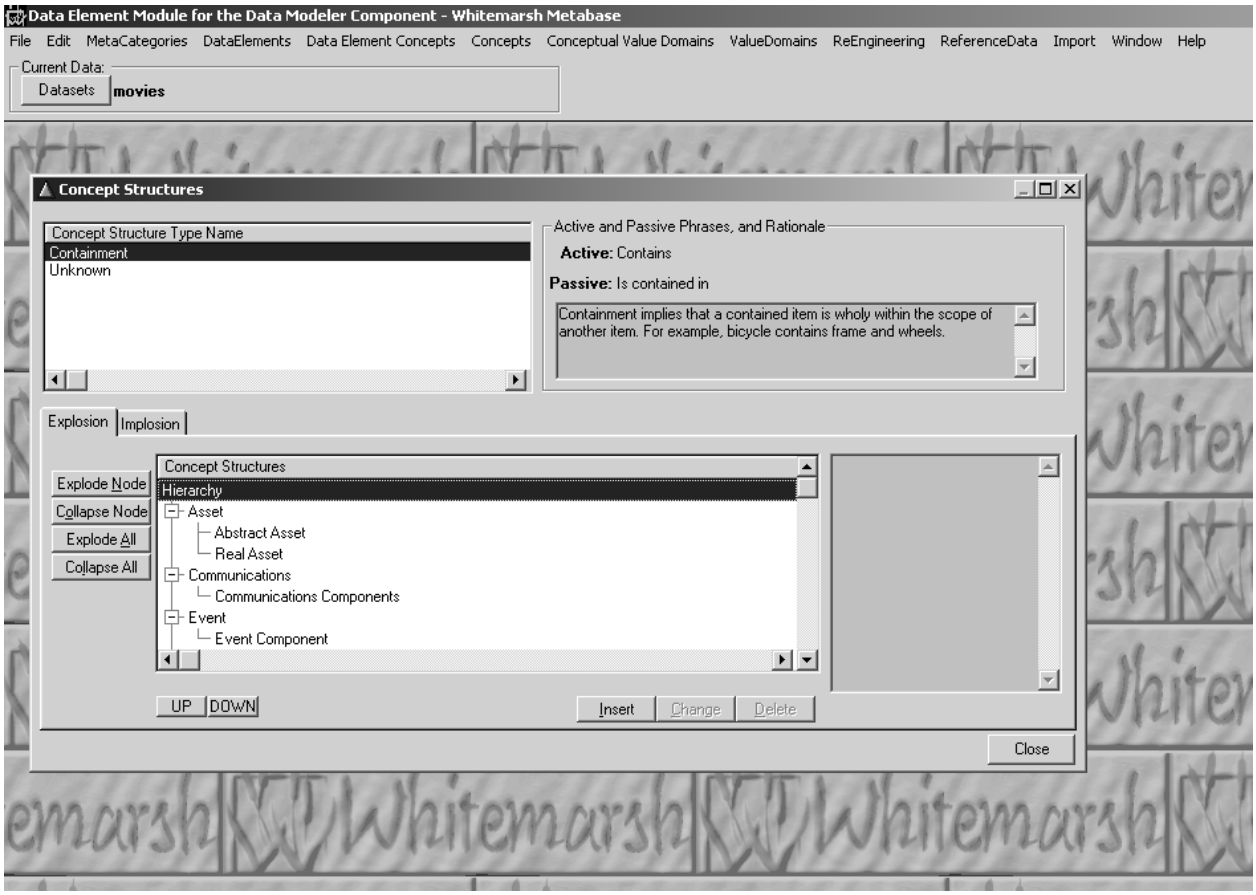

Figure 52. Concept structure explosion.

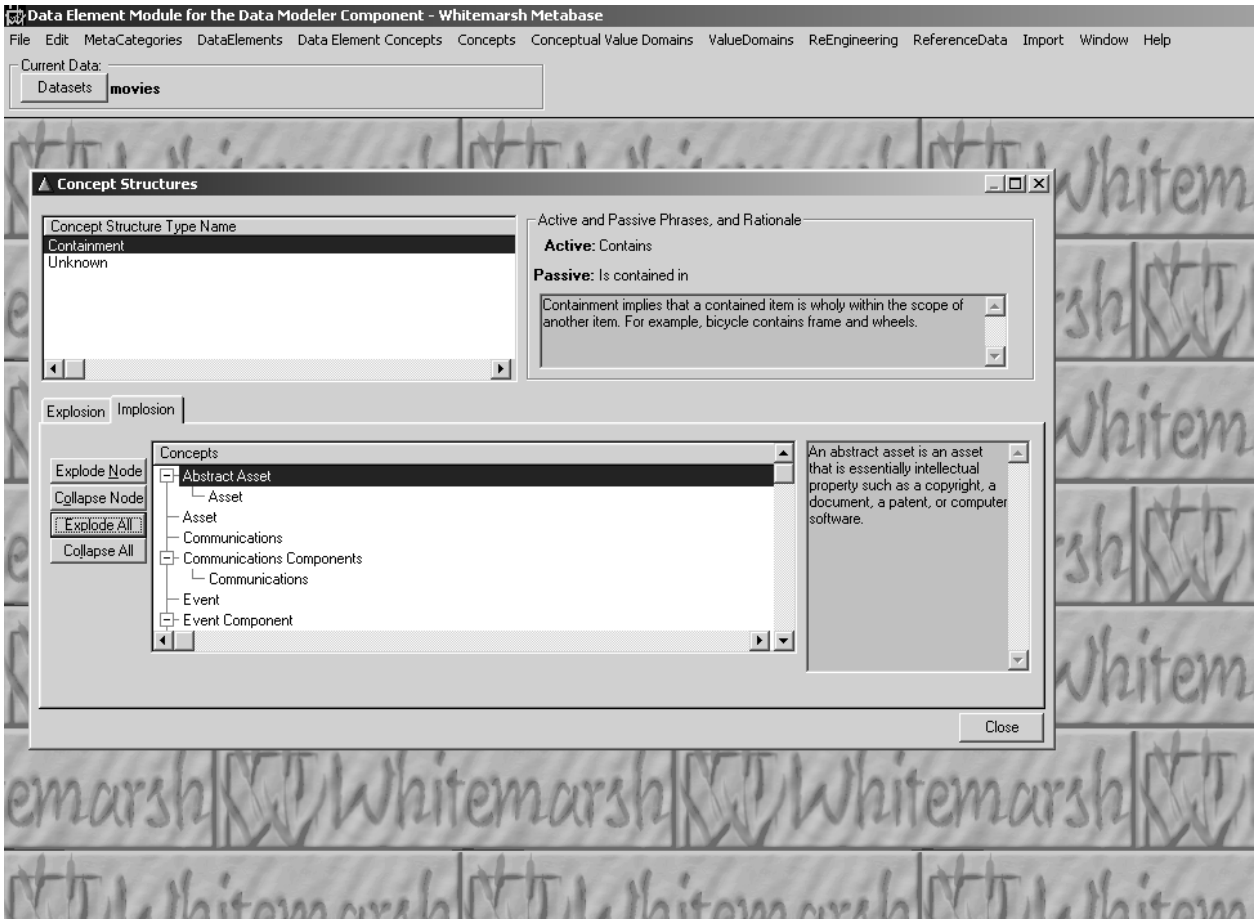

Figure 53. Concept structure implosion.

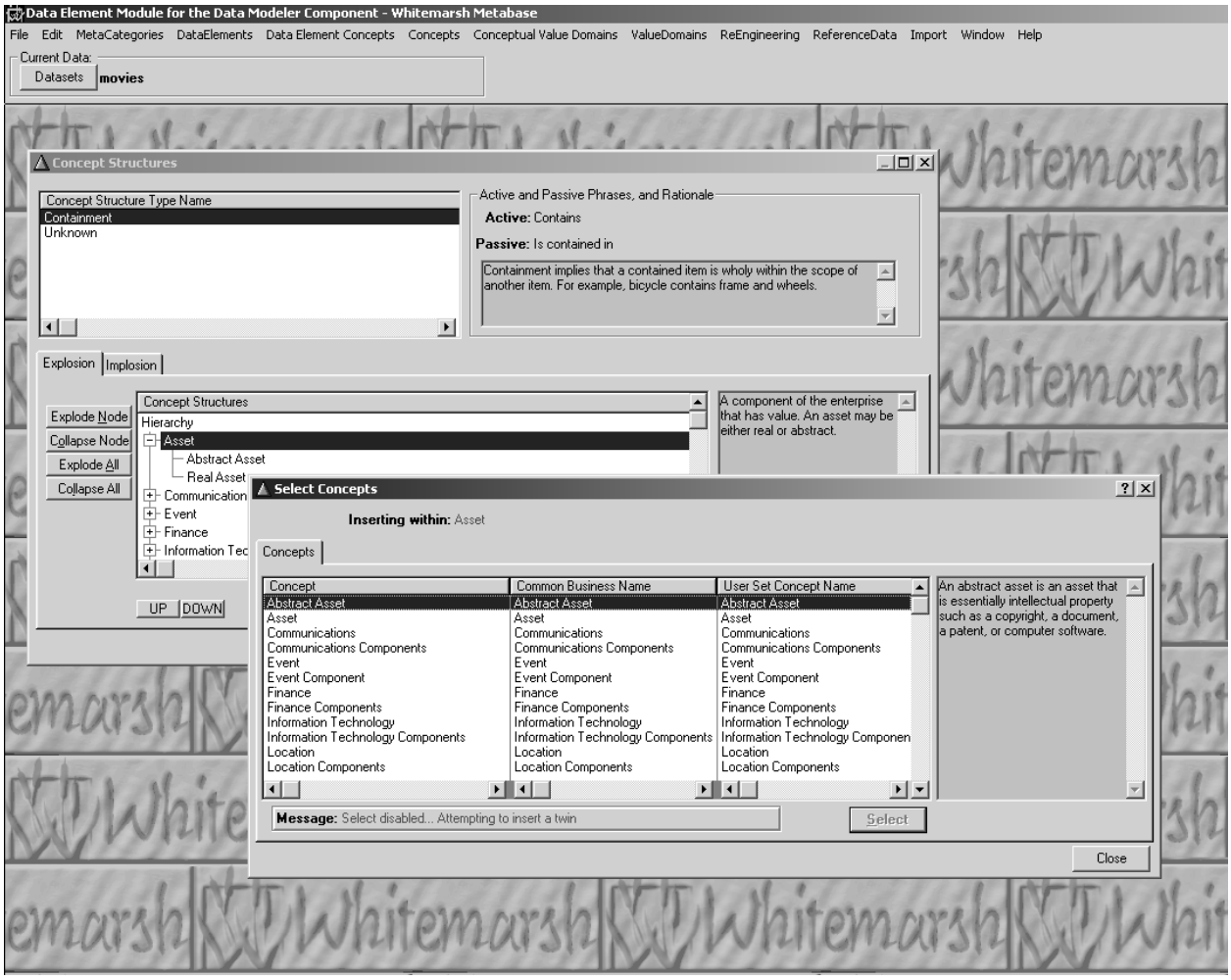

Figure 54. Concept structure insert.

5.2 Build Conceptual Value Domains

Building Conceptual Value Domains is similar to building concepts. They consist of three parts:

- Conceptual Value Domains
- Conceptual Value Domain Structure Types
- Conceptual Value Domain Structures

While the building process is similar, Conceptual Value Domains are quite different than Concepts. Conceptual Value domains are parallel with Concepts. They both contribute to the semantic construction of Data Element Concepts (see Section 5.3). Conceptual Value Domains are all about the concepts supporting value domains. Hence the name. They are different from value domains because there may be multiple value domains for one Conceptual Value Domain. In the ISO 11179 standard, the name for this is Conceptual Domains. It's definition, however,

leads to a better understanding under the name, Conceptual Value Domain. The method of discovery of Conceptual Value Domains parallels that of Concepts. Please refer back to the Hampton quotes above. Table 8 enumerates the conceptual value domains that appear appropriate for the movies domain.

Conceptual Value Domain Structures similarly provide the ability to proceed from generalized to specialized. An example of this is provided in Table 9.

Conceptual Value Domains	Description
Alphabetic Strings	A string of alphabetic characters for use in names, definitions, and descriptions.
Date Interval	A specific duration between two dates that includes those dates.
Dates	Representation of legal dates in all forms.
Decimals	A set of numbers with decimal positions of an arbitrary scale.
Descriptions	Textual descriptions of concepts.
Discrete Dates	Specific Date related to one day.
Formatted String	A string of numbers, letters. and special characters that conform to a specific format for a specific purpose. Examples would include Reservation Numbers, Credit Card Numbers, Check Numbers, and the like.
Integers	A range of numbers that are whole that may be both positive or negative.
Money	Monetary representations expressed as numbers, and if appropriate partial monetary amounts expressed in decimals.
Names	Textual names of concepts.
Numbers	Numeric values that may be integers, decimals, real, complex, or imaginary.
Specific Money Amount	A specific money amount that may represent the cost of a transaction or asset.

Table 8. Conceptual Value Domains

The key to assessing the sufficiency of the Conceptual Value Domains to determine that one or more of the attributes are represented by one Conceptual Value Domain. Conceptual Value Domain, like Concepts may be either simple or complex. Complex Conceptual Value Domains may either be hierarchies or networks.

In this particular example, there is a two level hierarchy that seems sufficient. It is provided in Table 9.

Conceptual Value Domains	Contained Conceptual Value Domain
Alphabetic Strings	Descriptions
	Names
	Formatted String
Dates	Date Interval
	Discrete Date
Money	Specific Money Amount
Numbers	Decimals
	Integers

Table 9. Conceptual Value Domain Structures

Because the mechanics of constructing Conceptual Value Domains is exactly the same as constructing Concepts, refer back to previous sections.

5.3 Build Data Element Concepts

Data Element Concepts are the first step on the road to enumerated data elements. Data Element Concepts are similar to Concepts and Conceptual Value Domains in that they are at the "conceptual" level, not at an enumeration level. Data Element Concepts are a merger of Concepts and Conceptual Value Domain, and thus represent a binding of a Conceptual Value Domain to a Concept. Data element Concepts bring a greater precision to a concept. Because of the construction technique, a given concept may be bound multiple ways by a Conceptual Value Domain.

Table 10 Shows the "marriage" of a Concept with one or more Conceptual Value Domains so as to generate a specific Data Element Concept. The quantity of Data Element Concepts here is 44. The quantity of Concepts is just 11. The listing of Data Element Concepts for a particular Concept is provided in Table 11. As to Conceptual Value Domains, the quantity used is 6 and their relationship to Data element concepts is provided in Table 12.

Concept	Conceptual Value Domain	Data Element Concept	Data Element Concept Description
Abstract Asset	Descriptions	Abstract Asset Description	The description associated with an abstract asset.
	Integers	Abstract Asset Computer Generated Identifier	A Computer Generated Identifier that is employed by the computer to uniquely identify an asset.
		Abstract Asset Integer Identifier	An abstract asset Integer Identifier is an integer number that is employed to uniquely identify an asset.
	Names	Abstract Asset Characteristics	The characteristics of an asset including for example, its identifier, name, description, etc.
		Abstract Asset Name	The name associated with the abstract asset.
Communications Components	Names	Communications Locator Characteristics	Characteristics examples of communications components.
		Communications Locator Names	Components of various communications mechanisms such as email addresses, phone numbers, URL, etc.
		Email Address	An email address of an organization or a person
		Telephone Number	A telephone number of an oranization or party.
Event Component	Descriptions	Event Description	Description of an event.
	Discrete Dates	Event Date	Specific data associated with an event.
	Names	Event Characteristics	Specific named characteristics of an event.
		Event Name	Name of an event.
Finance Components	Formatted String	Financial Instrument Identifier	The identifier that is associated with a financial instrument. Examples could be credit card numers, check card numbers, check numbers, bank routing numbers, and the like.
	Names	Financial Characteristics	Characteristics of financial transactions or events.

Concept	Conceptual Value Domain	Data Element Concept	Data Element Concept Description
		Financial Transaction Form	The form of a financial transaction. This might include cash, wire, credit card, or check.
	Specific Money Amount	Financial Amount	A specific amount of money.
		Financial Institution Identifier	The identifier of a financial institution that uniquely identifies to to some processing organization or activity.
Information Technology Components	Integers	Information Technology Components Integer Identifers	
	Names	Information Technology Components Characteristics	The specific names of characteristics associated with information technology components.
Location Components	Formatted String	Postal Code	A postal code is a formatted string created by a postal service to assist in the location of specific addresses for the purpose of delivering the mail.
	Names	Geopolitical Names	Geopolitical names could be countries, counties, states, provinces, citites, and the like.
		Location Address	The complete set of location address parts.
		Location Characteristics	The specific names of characteristics associated with a location.
Organization Components	Descriptions	Organization Description	A description of the organization.
	Formatted String	Organizational Identifier	The identifier that is associated with a organizational instrument. Examples could be a designation created by the organization that uniquely identifies it.
	Names	Organization Name	The name of the organization.
		Organizational Characteristics	Various characteristics of an organization.
Person Components	Discrete Dates	Person Birth Date	Person Birth Date

Concept	Conceptual Value Domain	Data Element Concept	Data Element Concept Description
		Person Related Event Dates	Specific dates related to a person.
	Names	Person Characteristics	Identifed and named characteristics about persons.
		Person Name Part	A specific part of a person's name.
Real Asset	Descriptions	Real Asset Description	The description associated with a real asset.
	Integers	Real Asset Computer Generated Integers	A Computer Generated Identifier that is employed by the computer to uniquely identify a real asset.
		Real Asset Integer Identifier	An Real Asset Integer Identifier is an integer number that is employed to uniquely identify an asset.
	Names	Real Asset Characteristics	Characteristics of real assets such as names, identifiers, and the like.
		Real Asset Name	The name of the real assset.
Transaction Components	Discrete Dates	Transaction Date	A specific date on which a transaction occurs.
	Integers	Transaction Number	The number that is assigned to a transaction so that it can be uniquely identified.
	Names	Transaction Characteristics	Specific names associated with characterizations of transactions such as dates, amounts, descriptions and names.
		Transaction Name	The specific name that commonly signifies the kind or nature of a transaction.
		Transaction State	The state of a transaction. For example, submitted, in process, completed, etc.
	Specific Money Amount	Transaction Amount	An amount of money that is tendered to partially or completely satisfy a transaction between two parties.

Table 10. Data Element Concepts created from Concepts and Conceptual Value Domains.

Concept	Data Element Concept
Abstract Asset	Abstract Asset Characteristics
	Abstract Asset Computer Generated Identifier
	Abstract Asset Description
	Abstract Asset Integer Identifier
	Abstract Asset Name
Communications Components	Communications Locator Characteristics
	Communications Locator Names
	Email Address
	Telephone Number
Event Component	Event Characteristics
	Event Date
	Event Description
	Event Name
Finance Components	Financial Amount
	Financial Characteristics
	Financial Institution Identifier
	Financial Instrument Identifier
	Financial Transaction Form
Information Technology Components	Information Technology Components Characteristics
	Information Technology Components Integer Identifers
Location Components	Geopolitical Names
	Location Address
	Location Characteristics
	Postal Code
Organization Components	Organization Description
	Organization Name
	Organizational Characteristics
	Organizational Identifier
Person Components	Person Birth Date

Concept	Data Element Concept
Abstract Asset	Abstract Asset Characteristics
	Person Characteristics
	Person Name Part
	Person Related Event Dates
Real Asset	Real Asset Characteristics
	Real Asset Computer Generated Integers
	Real Asset Description
	Real Asset Integer Identifier
	Real Asset Name
Transaction Components	Transaction Amount
	Transaction Characteristics
	Transaction Date
	Transaction Name
	Transaction Number
	Transaction State

Table 11. Concepts and related Data Element Concepts.

Conceptual Value Domain	Data Element Concept
Descriptions	Abstract Asset Description
	Event Description
	Organization Description
	Real Asset Description
Discrete Dates	Event Date
	Person Birth Date
	Person Related Event Dates
	Transaction Date
Formatted String	Financial Instrument Identifier
	Organizational Identifier

Conceptual Value Domain	Data Element Concept
	Postal Code
Integers	Abstract Asset Computer Generated Identifier
	Abstract Asset Integer Identifier
	Information Technology Components Integer Identifers
	Real Asset Computer Generated Integers
	Real Asset Integer Identifier
	Transaction Number
Names	Abstract Asset Characteristics
	Abstract Asset Name
	Communications Locator Characteristics
	Communications Locator Names
	Email Address
	Event Characteristics
	Event Name
	Financial Characteristics
	Financial Transaction Form
	Geopolitical Names
	Information Technology Components Characteristics
	Location Address
	Location Characteristics
	Organization Name
	Organizational Characteristics
	Person Characteristics
	Person Name Part
	Real Asset Characteristics
	Real Asset Name
	Telephone Number
	Transaction Characteristics
	Transaction Name

Conceptual Value Domain	Data Element Concept
	Transaction State
Specific Money Amount	Financial Amount
	Financial Institution Identifier
	Transaction Amount

Table 12. Conceptual Value Domains and related Data Element Concepts.

5.3.1 Data Element Concepts

Data Element Concepts are a subordinate menu item under the menu item, Data Elements. Fundamentally, the process of building a Data Element Concept is to pick the appropriate Concept, the select an appropriate Data Element Concept, and then enter in the rest of the Data Element Concept information. Figure 55 presents the Data Element Concept browse. What you're actually picking is not Concept or Conceptual Value Domain, but an instance of a Concept Structure and a Conceptual Value Domain Structure. The reason for picking within the context of a structure is that a "part" could be in multiple structures. Hence, the need to pick the Concept or Conceptual Value Domain within the context of their structures. The Movies example doesn't require it, but the need for complex structures exists.

Once the Concept and Conceptual Domains are properly selected, and selection is allowed only on leafs, press Insert. Note, selection is only allowed when leafs are selected. Otherwise a message is displayed that indicates that selection is prevented. Figure 56 then presents itself. The form automatically identifies that the data element concept is within the context of a particular conceptual value domain structure and concept structure. Then enter abbreviations and a brief description of the data element concept.

When all the data element concepts are created, a review should support the conclusion that when all the attribute are finally mapped to the data elements that there will be a home for each.

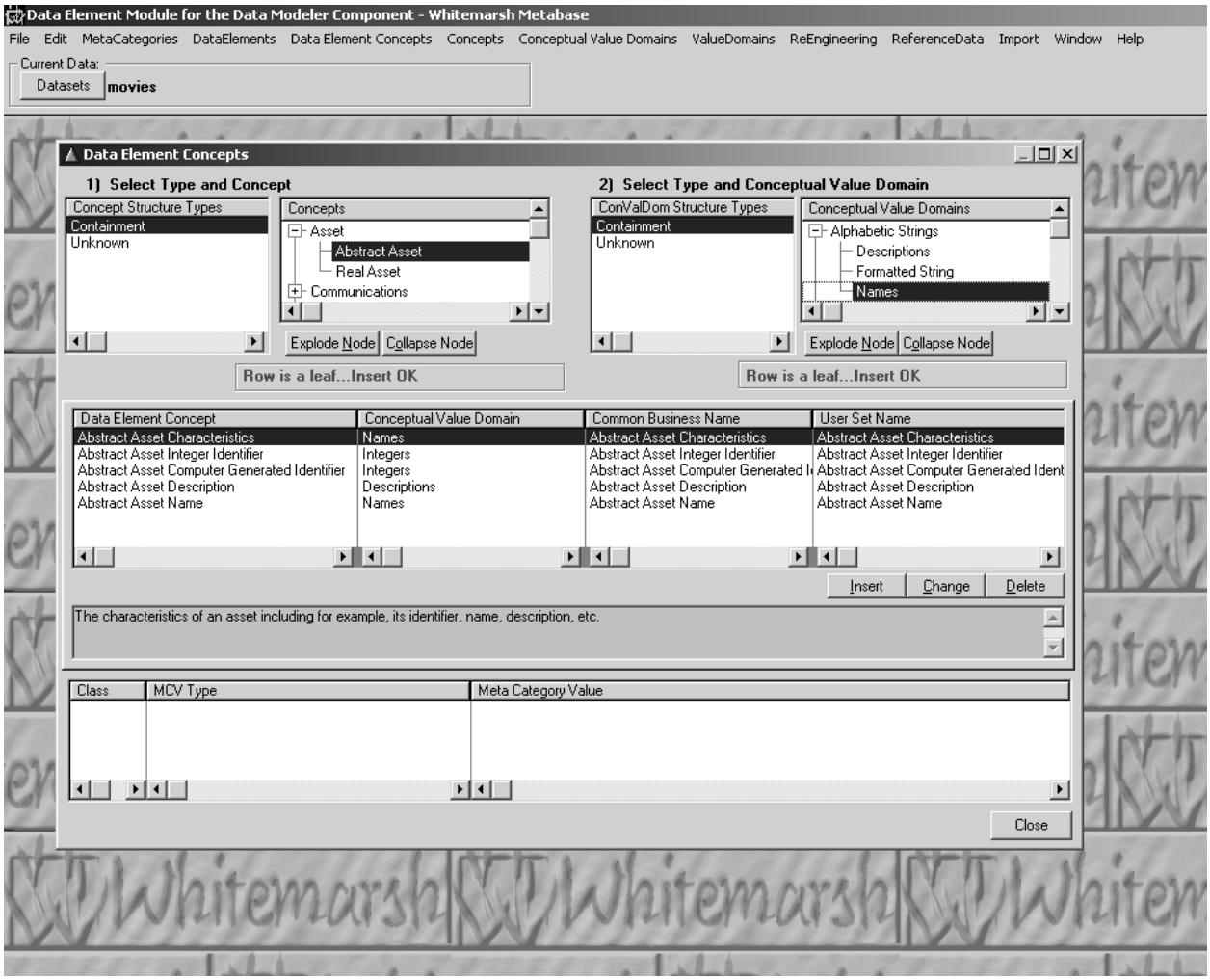

Figure 55. Data element concept browse.

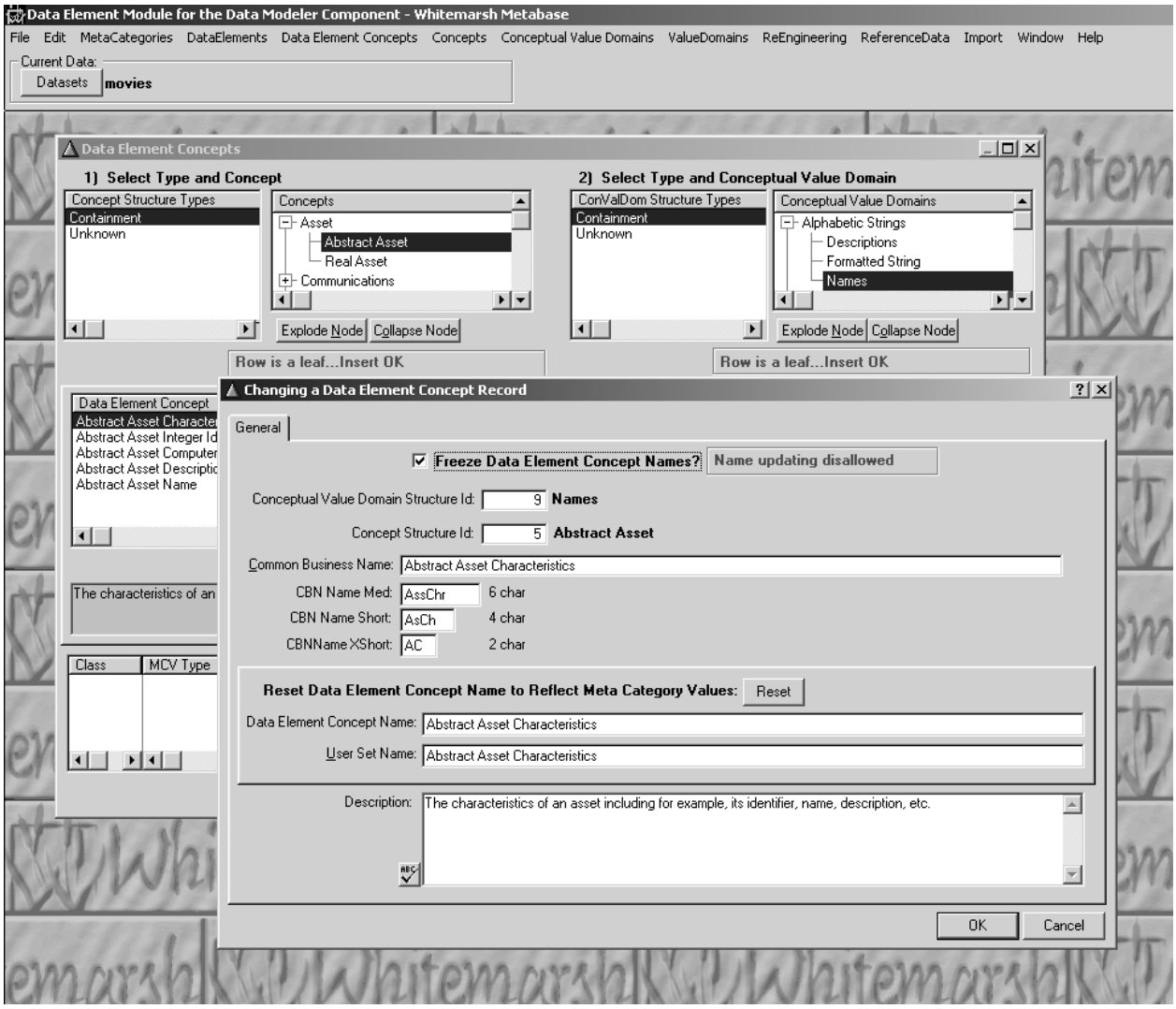

Figure 56. Data element concept update form.

5.3.2 Data Element Concept Structure Type

Data Element Concept Structure Types are essentially the same as for all the others, so refer back to the instruction under Concept for its creation.

5.3.3 Data Element Concept Structures

The Data Element Concept Structures are essentially the same as for all the others, so refer back to the instruction under Concept Structures for their creation. You should create all the Data Element Concept Structures identified in Table 13. Data Elements are defined within the context of the Data Element Concept leafs. Once these are all built then proceed to the next step, building Value Domains.

Data Element Concept	Contained Data Element Concept
Abstract Asset Characteristics	Abstract Asset Computer Generated Identifier
	Abstract Asset Description
	Abstract Asset Integer Identifier
	Abstract Asset Name
Communications Locator Characteristics	Communications Locator Names
	Email Address
	Telephone Number
Event Characteristics	Event Date
	Event Description
	Event Name
Financial Characteristics	Financial Amount
	Financial Institution Identifier
	Financial Instrument Identifier
	Financial Transaction Form
Information Technology Components Characteristics	Information Technology Components Integer Identifiers
Location Characteristics	Geopolitical Names
	Location Address
	Postal Code
Organizational Characteristics	Organization Description

Data Element Concept	Contained Data Element Concept
	Organization Name
	Organizational Identifier
Person Characteristics	Person Birth Date
	Person Name Part
	Person Related Event Dates
Real Asset Characteristics	Real Asset Computer Generated Integers
	Real Asset Description
	Real Asset Integer Identifier
	Real Asset Name
Transaction Characteristics	Transaction Amount
	Transaction Date
	Transaction Name
	Transaction Number
	Transaction State

Table 13. Data Element Concept Structures

5.4 Build Value Domains

Value domains are the specification and enumeration of the value domains associated with Conceptual Value Domains. Table 14 presents value domains for the various Conceptual Value domains. The purpose of value domains is that they will help restrict the specific values that are allowed in Data Elements and downstream in entity attributes, table columns, or DBMS table columns. In this particular example the value domains are just general purpose. They can however be very specific including enumerations of specific values. This is not covered in this guide.

Conceptual Value Domain	Value Domain	Description	Pre-cision	Scale	Null-Allowed
Date Interval	Quantity of Days	The quantity of days between two fixed dates. This interval is computed elsewhere. It is normally a quantity of days not to exceed a year.	0	0	No
Decimals	Decimal numbers of varying sizes	Decimal numbers of varying precisions and scales.	14	3	No
Descriptions	Character strings of arbitrary length	Generalized strings of various lengths to hold definitions and or descriptions.	255	0	No
Discrete Dates	Specific Legal Dates	A specific legal date on a calendar. It would hold someone's Birthdate, the due date for a payment, and the like.	0	0	No
Formatted String	Engineered strings of specific length and construction	A string of fixed length and construction that would hold for example, a telephone number that contains country code, area code, exchange, and number. Or a Social Security Number that contains three numerics separated by two hyphens.	128	0	No
Integers	Integer Numbers	Integers of various sizes.	0	0	No
	Large Integers	An integer of maximum size allowed by the DBMS. Commonly this is allowed up to 64 bits.	0	0	No
	Medium Integers	An integer of medium size allowed by the DBMS, that is, up to 32 bits.	0	0	No
	Small Integers	An integer of small size, one byte of 8 bits.	0	0	No
Names	Name strings of arbitrary length	A string of arbitrary length that holds the name of the object.	255	0	No
Specific Money Amount	Amounts less than 1,000	A specific money amount of less than $100, USD.	9	2	No
	US Dollars	Monies in United States Dollar amounts.	14	2	No

Conceptual Value Domain	Value Domain	Description	Pre-cision	Scale	Null-Allowed

Table 14. Value domain

The creation of Value Domains contains these steps:

- Create Value Domain
- Create Value Domain Structure Type
- Create Value Domain Structure

5.4.1 Create Value Domain

The process of creating a Value Domain consists of selecting the appropriate Conceptual Value Domain and then entering the appropriate other value domain information. Figure 57 presents the Value Domain Browse. Select the Conceptual Value Domain and press Insert to add a new Value Domain. Figure 58 is then presented. Enter the information about the value domain, click OK. You are then back at the browse screen. Add more value domains, change them or close the screen.

5.4.2 Create Value Domain Structure Type

Value Domain Structure Types are essentially the same as for all the others, so refer back to the instruction under Concept for its creation with this one.

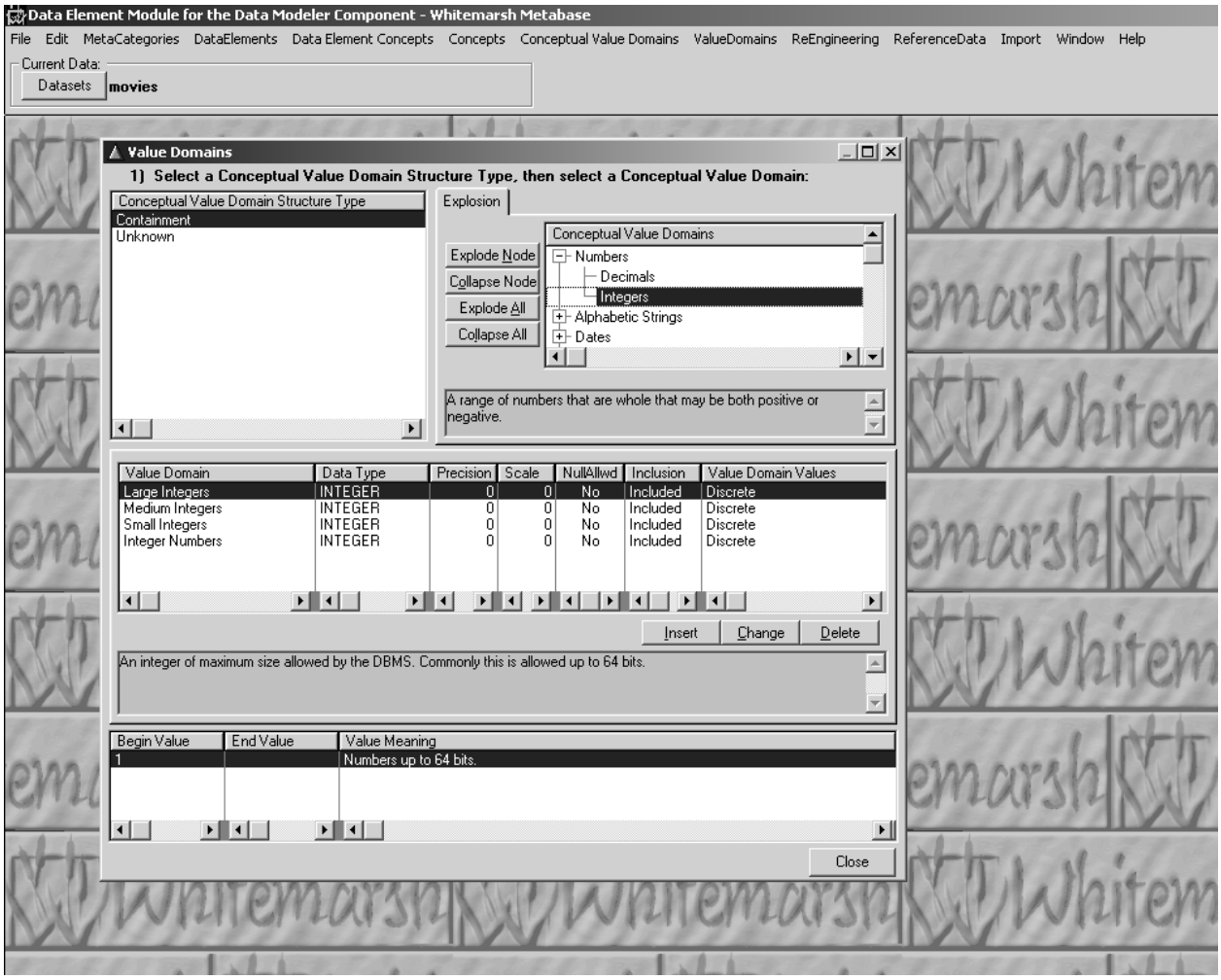

Figure 57. Value domain browse.

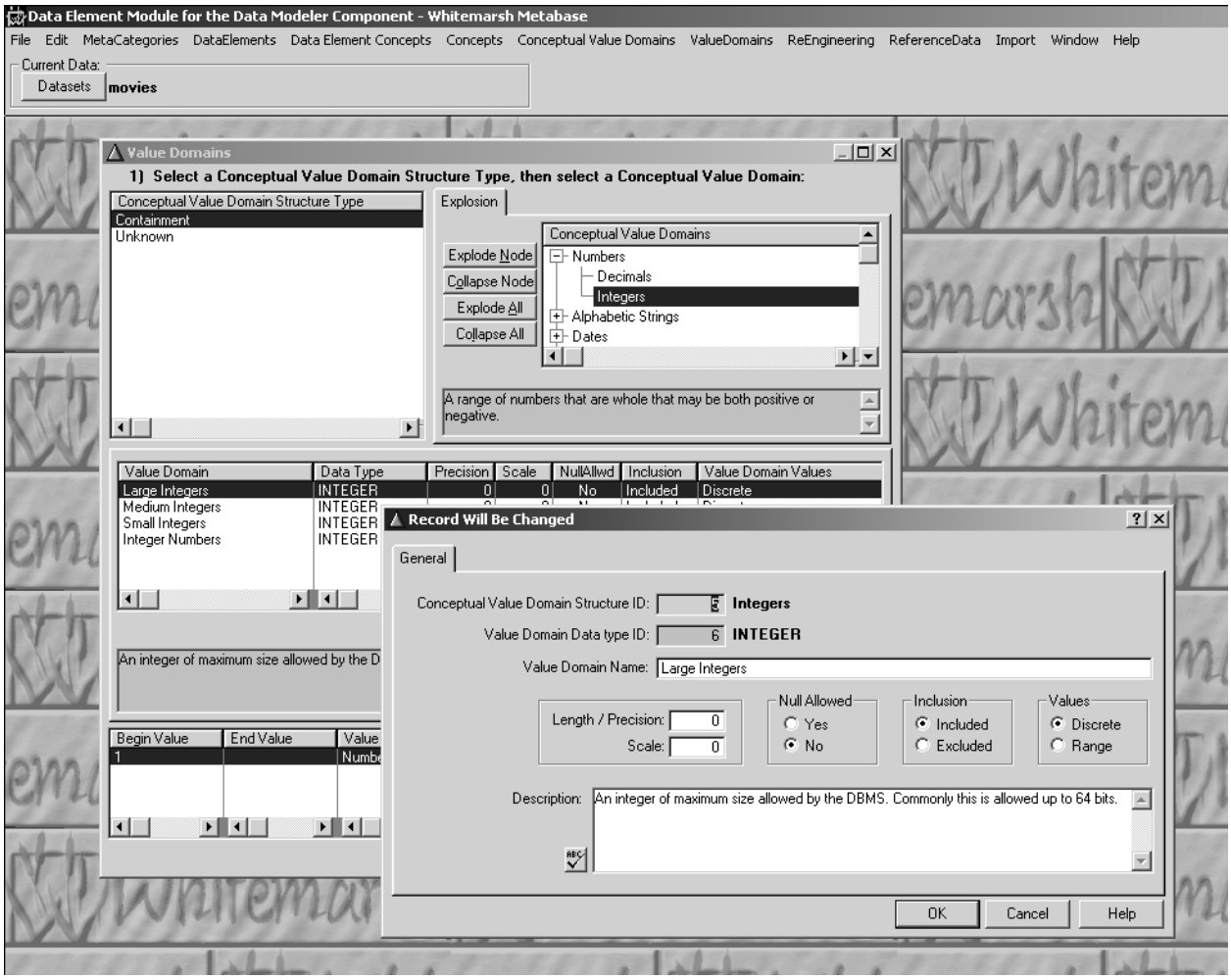

Figure 58. Value domain update form.

5.4.3 Create Value Domain Structure

The Value Domain Structures are essentially the same as for all the others, so refer back to the instruction under Concept Structures for their creation. You should create all the Value Domain structures identified in Table 14. Once these are all built then proceed to the next step, Build Data Elements.

Value Domain	Contained Value Domain
Amounts less than 1,000	
Character strings of arbitrary length	
Decimal numbers of varying sizes	
Engineered strings of specific length and construction	
Integer Numbers	Large Integers
	Medium Integers
	Small Integers
Name strings of arbitrary length	
Quantity of Days	
Specific Legal Dates	
US Dollars	

Table 14. Value Domain Structures

5.5 Build Data Elements

The process of building a data element is similar to that of building Data Element Concepts except that data elements do not have Data Element Structures or Data Element Structure Types. In short, all data elements are atomic and not derived. The data element module has processes for defining both compound (not atomic) and derived data elements. Since these are needed for this example they are not presented.

The process of creating a data element starts with it's browse. It is presented in Figure 59. Select the appropriate business domain, for example, Finance, and then press Insert. Figure 60 then appears. The business domain is filled in automatically. The Data Element Concept Id entry field is initially set to zero. Tab through it. A Data Element Concept select screen will then appear. Select the data element concept for the data element. The Value Domain Id field cannot be valued and so it is set to "1" (that is, unknown). The value domain for the data element is set within the Data Element Value Domains menu item that is within the Data Element menu item.

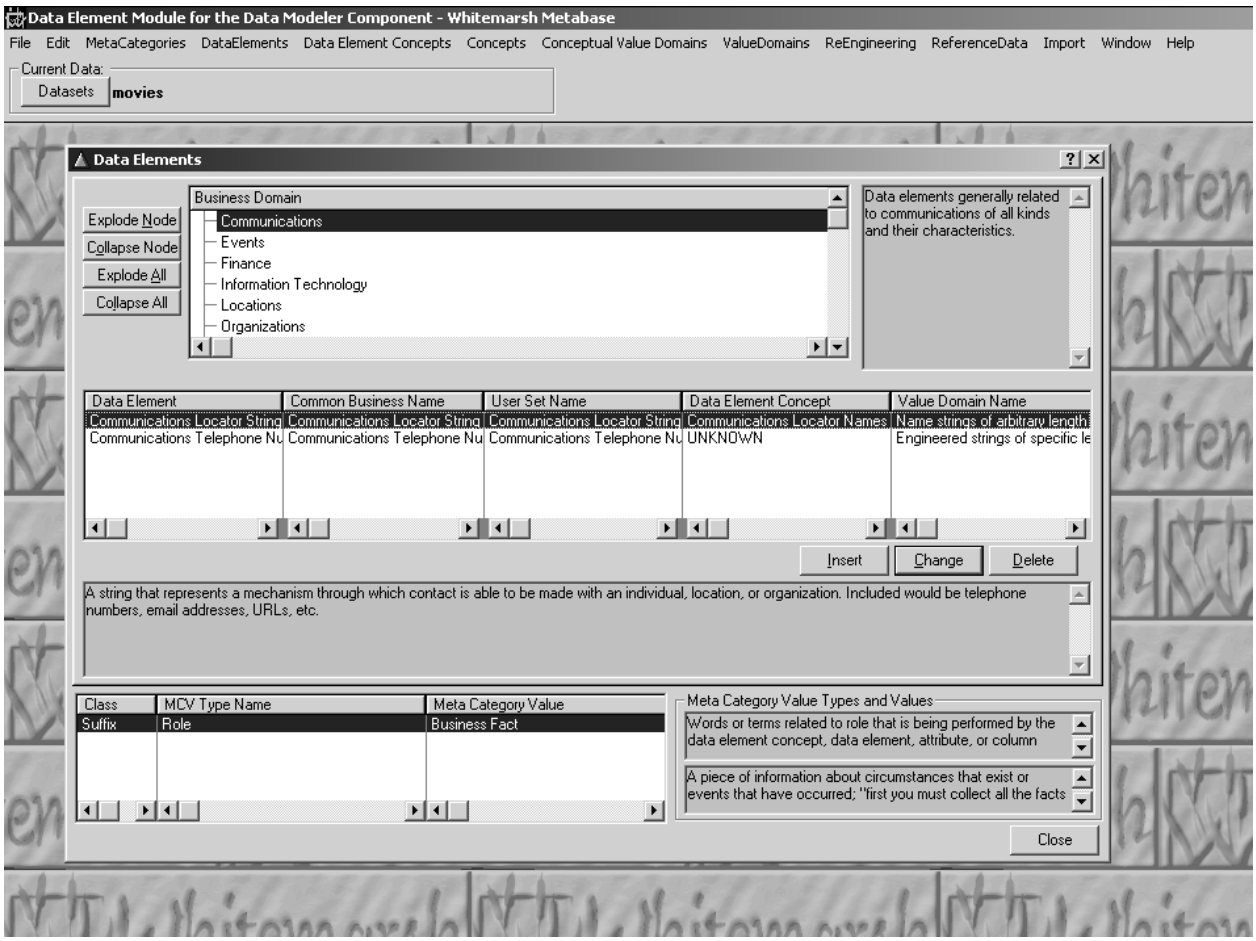

Figure 59. Data element browse.

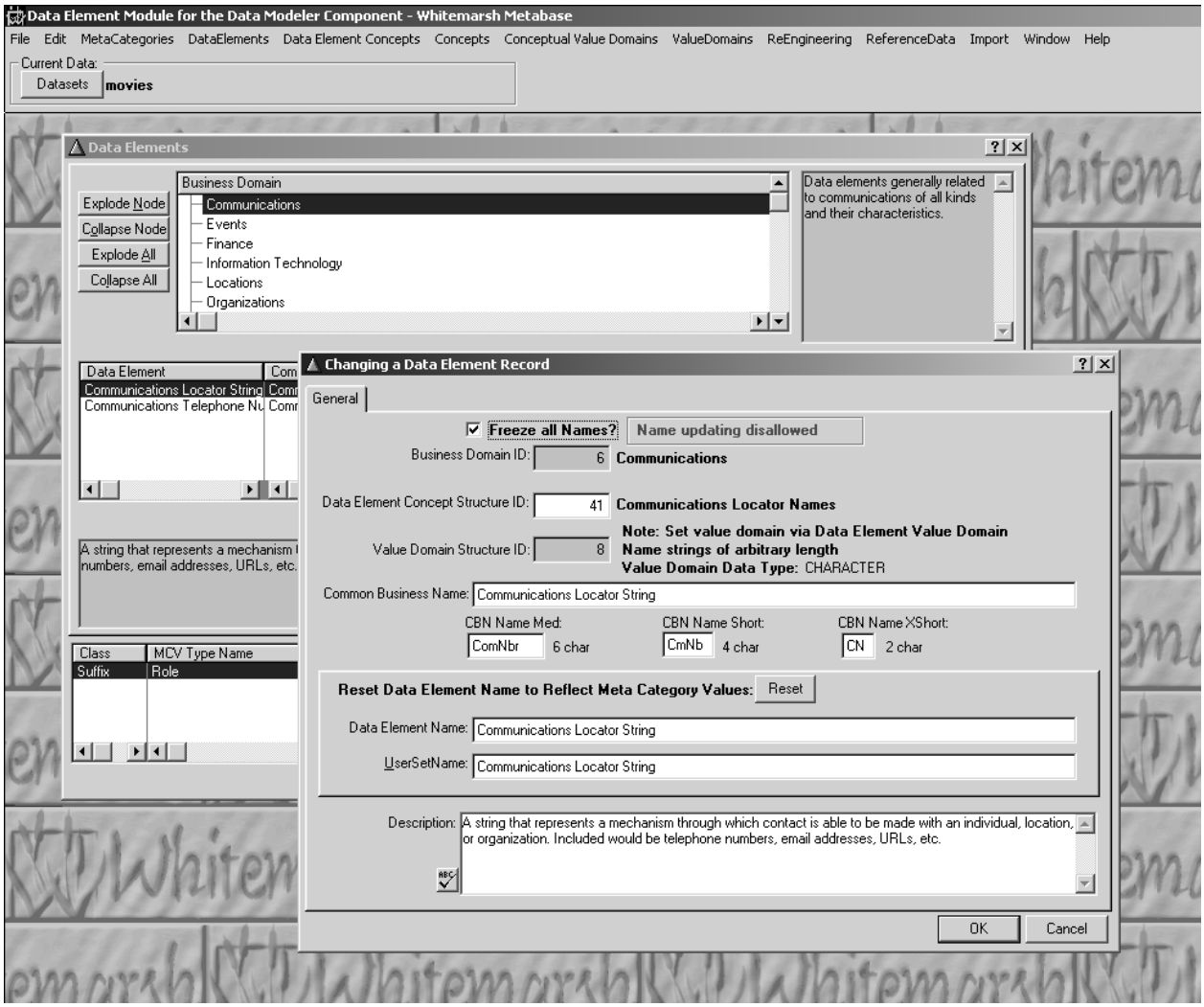

Figure 60. Data element form.

Enter a common business name for the data element. One is automatically created for you and consists of the combination of the Business Domain name and the Data Element Concept name. Change it as you wish. Enter the abbreviations, and then press the “Reset” button. That process ensures that the name is fully constructed and sets the constructed values into Data Element Name and the User Set Name. Change these if you wish. Finally, enter a description. Press OK and you are taken back to the Data Element browse. The newly created data element then appears.

A data element is a slightly generalized version of an entity attribute or a table column. An attribute may have a more refined value domain. A column may have a more refined data type and also a more refined value domain. Table 15 presents the results of this top-down Concept to Data Element Concept to Data Element exercise. The next step is to then relate all the attributes from the entities to the data elements.

Concept	Data Element Concept	Data Element
Abstract Asset	Abstract Asset Characteristics	Product Description
Communications Components	Communications Locator Names	Communications Locator String
Event Component	Event Date	Event Date
	Event Name	Assessment
Finance Components	Financial Instrument Identifier	Financial Instrument Identifier
		Payment Mechanism Name
		Payment Mechanism Number
Information Technology Components	Information Technology Components Integer Identifiers	Identifier
Location Components	Location Address	Geopolitical Name
		Postal Code
		Street Address
Organization Components	Organization Identifier	Government Institution Identifier
		Financial Institution Identifier
	Organization Name	Organization Name
Person Components	Person Name Part	Person Name
Real Asset	Real Asset Name	Product Name
Transaction Components	Transaction Amount	Financial Amount
		Transaction Amount

Concept	Data Element Concept	Data Element
	Transaction Date	Transaction Date
	Transaction Name	Payment Mechanism Type
	Transaction State	Transaction Status

Table 15. Concepts, Data Element Concepts and Data Elements.

5.6 Connect Specified Data Model Attributes to Data Elements

This section started with Table 4. That is, the list of data elements to which the Attributes are to be attached. This list of data elements was created top-down, but always with careful notice of the attributes to which they will ultimately map.

There is another way to create data elements: Bottom up. You can also create Data Element Concepts, and even Concepts bottom up as well. Check the Re-engineering menu items within the Data Element module for the process.

The process of attaching attributes to data elements begins with the Specified Data Model module Re-engineering menu item, Reassign Attributes to Data Elements. The screen is shown in Figure 61. In this example, select the subject then the entity. Then tag one or more attributes. Tag the data element to which the attribute is to be reassigned. It's initial assignment is always to "Unknown." Hence it is always being reassigned. Press the "press" button to effect the reassignment. Once reassigned, proceed to the next attribute and the next entity and the next subject area for reassignment. Once completed, the reassignments should look like Table 4.

5.7 Synchronize Implemented Data Model Columns to Data Elements

Once the attributes are re-assigned, the column data element assignments still have to be accomplished. Since the process was reverse engineering, and since data elements certainly did not exist at the time the Implemented Data Model was created, then the only data element that the columns could be assigned to was the "Unknown" data element.

To accomplish the re-assignments, bring up the Implemented Data Modeler, and then under the Re-Engineering menu item, activate Column Data Element Re-Assignments. This is shown in Figure 62. Select the Movies Original Data Capture schema, then the Movie Rental Record table. Then note that the data element assigned to all the columns is "Unknown." Note, however, that from the prior step, all the attributes are shown. That is, all the columns are assigned to attributes. It is unreasonable for an attribute to be assigned a data element different from the column's data element. Not only is it unreasonable, it is an error. So, the choices are two: reassign the columns, one-by-one, as was done for attributes in the prior step, or assign them en-mass.

The process of en-mass assignment merely requires you to select the schema and the table. Then press the Reset Column Data Elements button that is right above the Attribute browse. Once the process finishes, proceed to the next table and repeat until all the column are re-assigned.

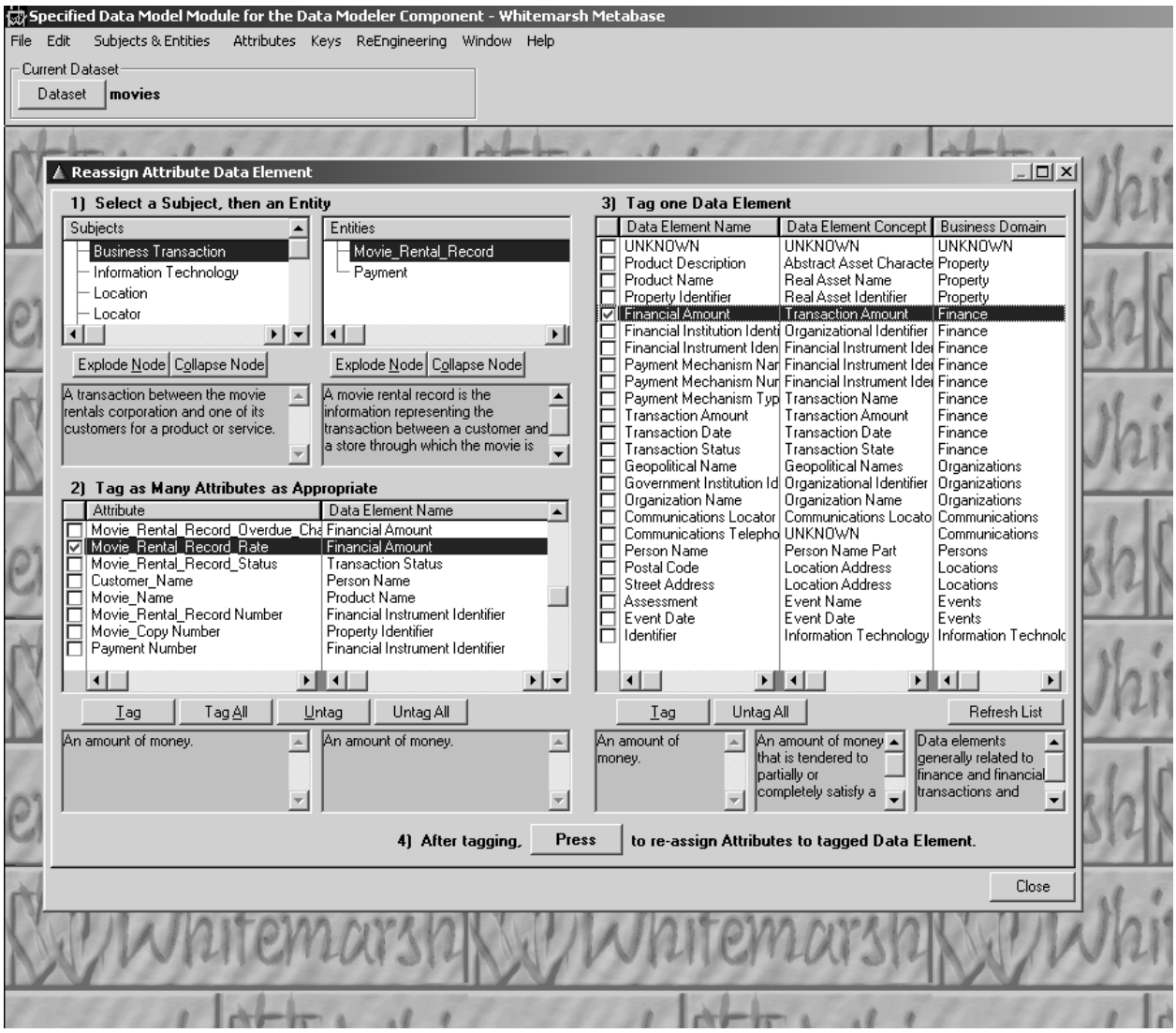

Figure 61. Attribute data element reassign screen.

5.8 Data Element Summary

The business case for accomplishing the building of data elements is simple: time and money. If the strategy set out in this document had been employed by the U.S. Department of Defense in the 1990s (as they were advised to do so), then the U.S. DoD would have had a successful data standardization process; they would have saved hundreds of millions of dollars; and they would have largely succeeded in achieving data interoperability.

So, the choice is simple. Spend tens-of-thousands, or waste hundreds-of-millions.

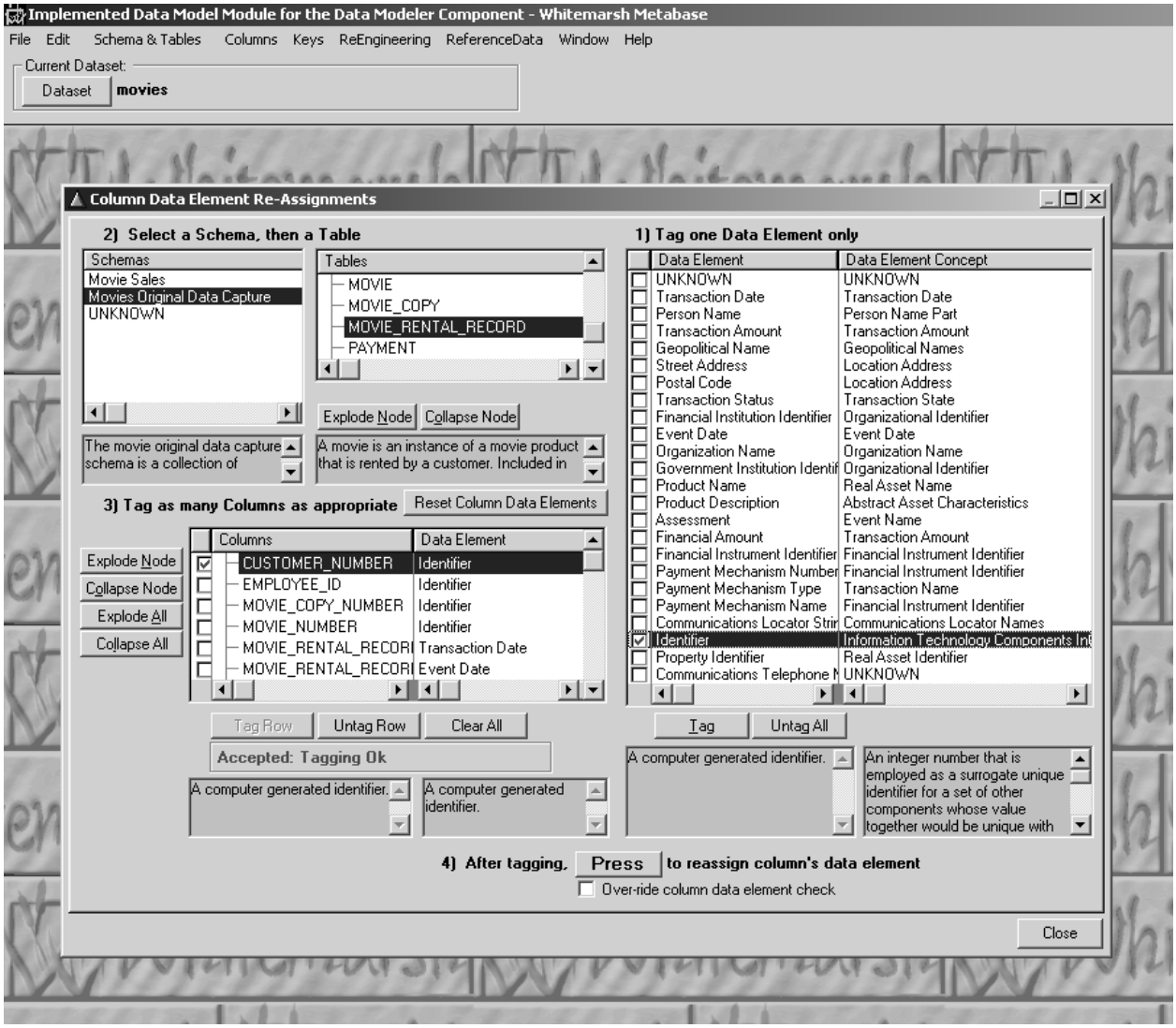

Figure 62. Column data element reassignment screen.

6.0 Forward Engineering

The forward engineering process causes metadata to be imported into the Implemented Data Model from the Specified Data Model, and from the Implemented Data Model into the Operational Data Model. Once in the Operational Data Model, SQL DDL can be generated and then delivered over to the DBMS for creation of the database itself.

One or more Specified Data Model templates may contribute to the creation of the Implemented Data Model. The following alternatives are available:

- One entity becomes one table.
- More than one entity can be combined into one table.
- One entity can be employed in more than one table.
- Part of an entity can be employed in one or more tables.

Similarly, one or more Operational Data Models may be created from the Implemented Data Model. The data modeling alternative listed above apply as well. Thus, the Specified, Implemented, and Operational data models each may be different with different quantities of tables, columns, with different primary and foreign keys, and finally, different names. Despite all these differences they may be related and their relationships are all contained within the metabase. The relationships are based on the Entity Attributes from the Specified Data Model being related to th Table Columns within the Implemented Data Model, which are in turn related to the DBMS Table Columns within the Operational Data Model.

This not only maximizes flexibility on the part of the data modeler but it also enables the database concept: Define once and use many times, to be accomplished again. The steps associated with forward engineering are:

- Employ Specified Data Model entities to create an Implemented Data Model
- Employ Implemented Data Model entities to create an Operational Data Model
- Generate SQL DDL

6.1 Build an Implemented Data Model

The process of building an Implemented Data Model consists of the following steps:

- Create a schema
- Import Specified Data Model entities
- Add or delete columns as appropriate
- Adjust column data types as appropriate
- Create primary and foreign keys

6.1.1 Create a schema

Start this the Implemented Data Model building process by creating a schema. Start the Implemented Data Model module and then select the Schema process under the Schema and Tables main menu item. Figure 63 will then be shown.

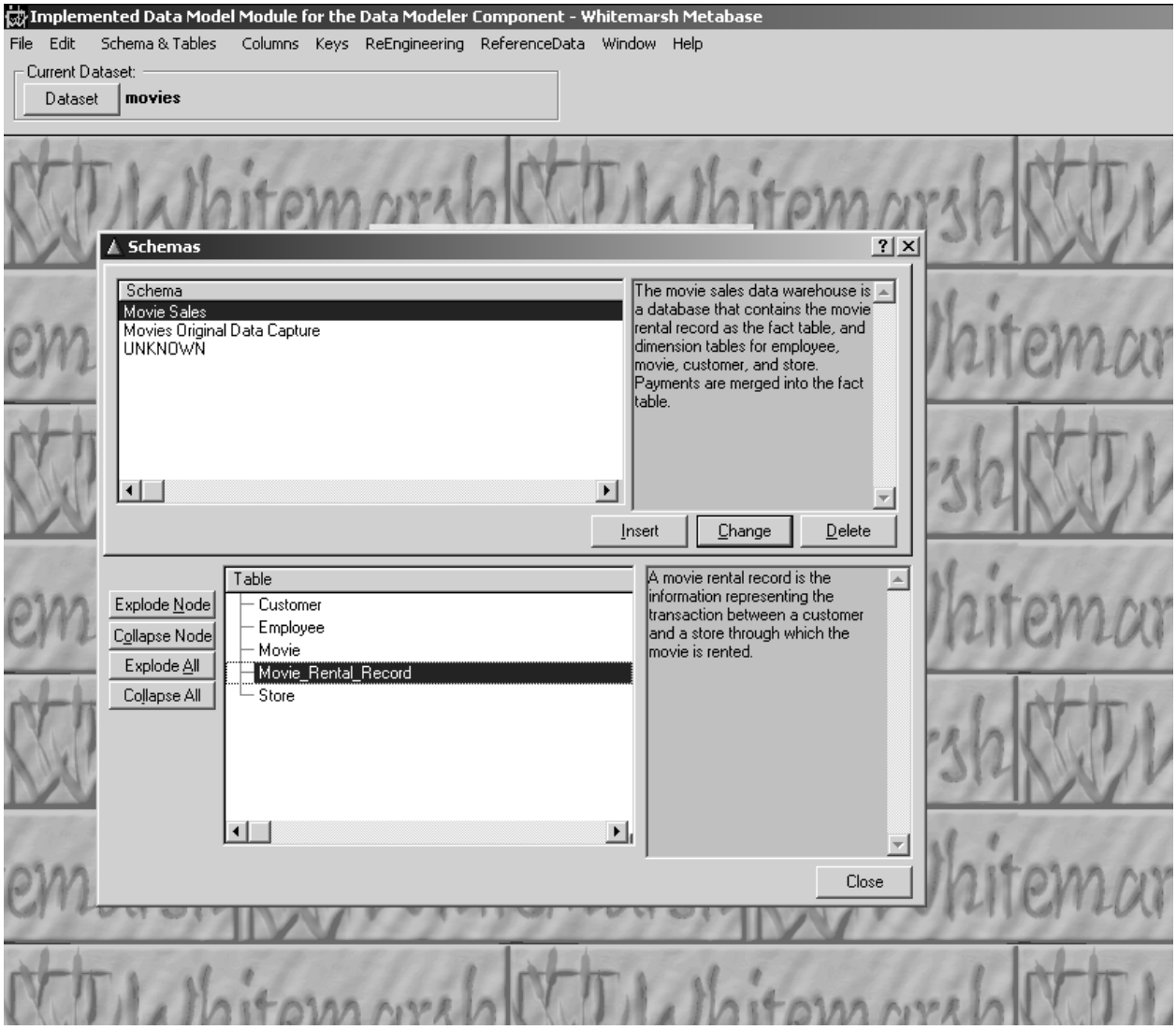

Figure 63. Schema browse.

This schema is for a data warehouse. It is going to contain five tables. The fact table is going to be based on the movie rental record. There are going to be four dimensions: Customer, Employee, Movie, and Store. To start the process, press Insert. Figure 64 will be presented. Enter the name of the database, Movie Sales. Create the abbreviations, and then create a description. Press Ok, and then press Close to end this step.

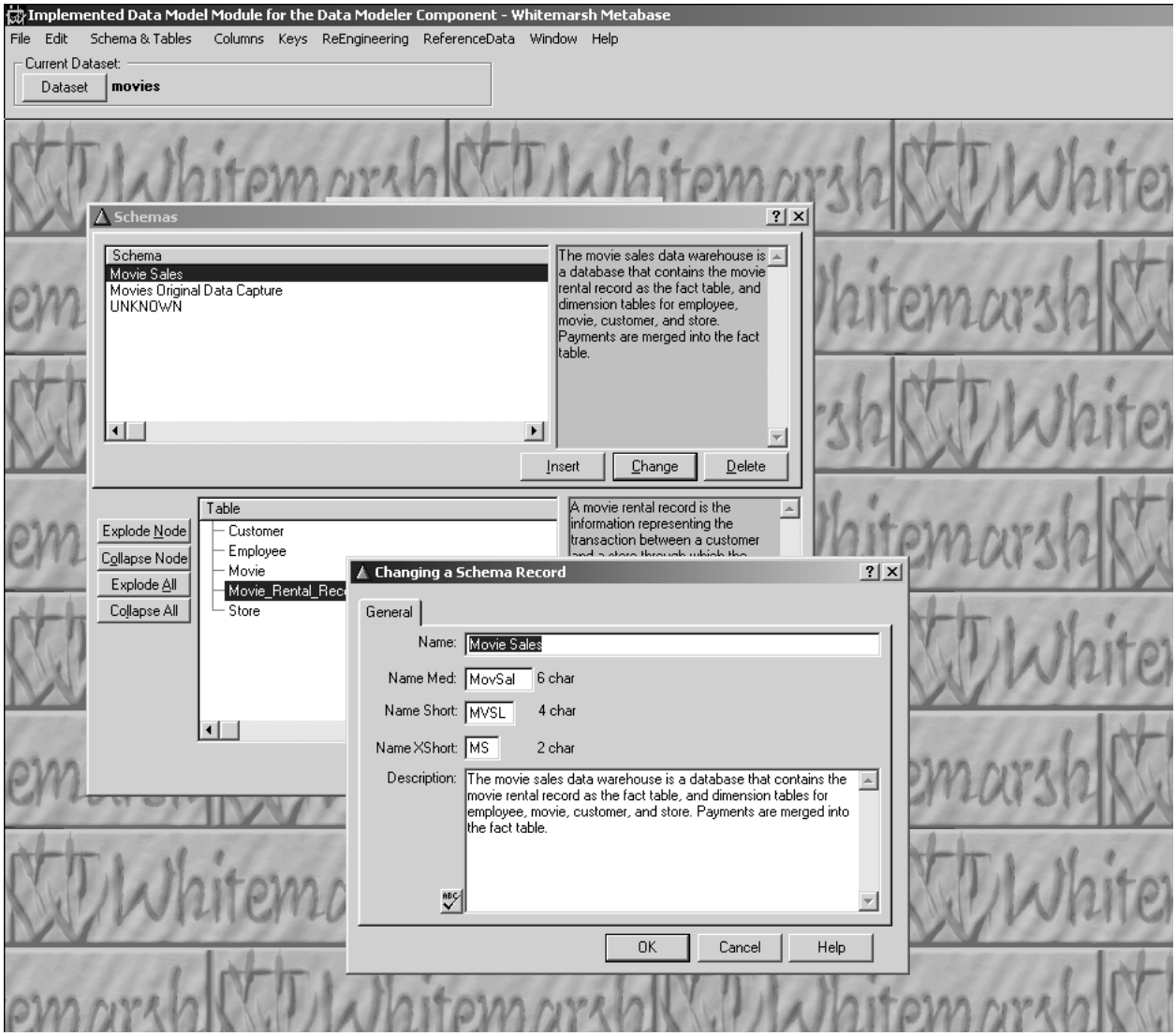

Figure 64. Schema update form.

6.1.2 Import Specified Data Model Entities

The process of creating Implemented Data Model entities consists identifying a subject, and an entity. Identify the receiving schema, and then press the Import entity button.

Figure 65 presents the screen to import a single entity. You can also import all the entities from a subject, or all the entities that are related through primary and foreign keys. In this case we are going to import the following entities: Movie Rental Record, Employee, Movie, Customer, Store, and Payment. This will form the basis of the data warehouse. The fact table is the Movie Rental Record. We'll add to the fact table some columns from Payment, so the fact table becomes Movie Rental Record Payment. Then we'll have dimensions of Employee, Movie, Customer, and Store.

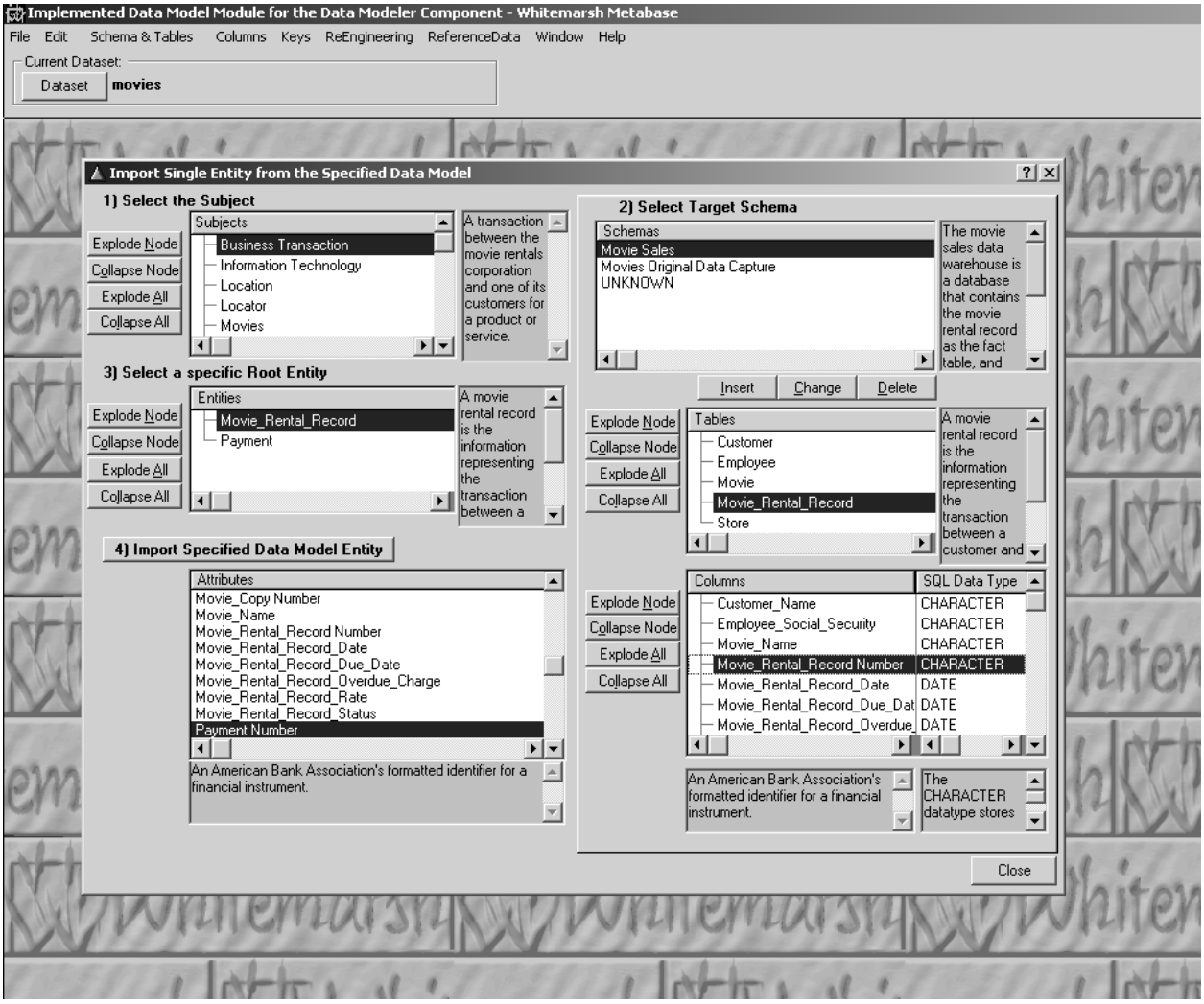

Figure 65. Import single entity.

When a single entity is imported, if it has foreign keys, then the foreign keys are not imported except when the foreign key's columns are part of the primary key. In that one case the foreign key is not imported but its column remains as part of the imported table.

Highlight the subject and the entity. Select the schema that will hold the imported entities. Then press the Import Specified Data Model Entity button. Repeat this process until all the entities are imported. You can check the primary keys. You will see that they all exist.

6.1.3 Add, Delete, or Adjust Columns

The first step in adjusting the columns in the Movie Rental Record is to move the payment columns to it. This is accomplished through the re-engineering process, Move Columns to Tables. The columns that were moved were: Payment Amount, Payment Credit Card Type, Payment Date, and Payment Status. When moving columns, as illustrated in Figure 66, select from the left side, the schema, the table, and then tag the columns to move. Then, on the right side, select the schema and the receiving table. Press the button to move the columns. Now the Movie Rental Record will contain only the columns we want. At this point, delete the Payment table.

In addition to adding or deleting columns, some column names may have to be adjusted. For example the newly added Payment column names should be changed to reflect the context, Movie Rental Record. This process is accomplished by the Column Maintenance menu item which is under Maintenance under the main menu item, Columns. These changes should all be done prior to the creation or modification of foreign keys. If they are not, then the column maintenance screen has buttons that will adjust the data type and picture, and also adjust the column names so that there is a match between the primary key column(s) and the foreign key column(s).

The final action to add "Telephone Number" to three of the dimension tables, Store, Employee, and Customer. This is accomplished through the process depicted in Figure 67. Activate the Import Attributes from the Specified Data Model via Schema & Tables, then Import and Export, then Import from Specified Data Model then Import Attributes from Subject.

Once the Telephone Number columns are added, proceed back through the set of all columns adjusting names as may be appropriate. For example, changing Telephone Number within Customer to Customer Telephone Number.

When you attempt to change a column associated with a foreign key, the metabase system will reject the attempt as the column is not a "natural" column of the table but is a foreign column which really is derived from the source table. Hence Customer Number in Movie Rental Record cannot be changed to, for example, Movie Rental Record Customer Number because Customer Number is a natural column of Customer, not Movie Rental Record.

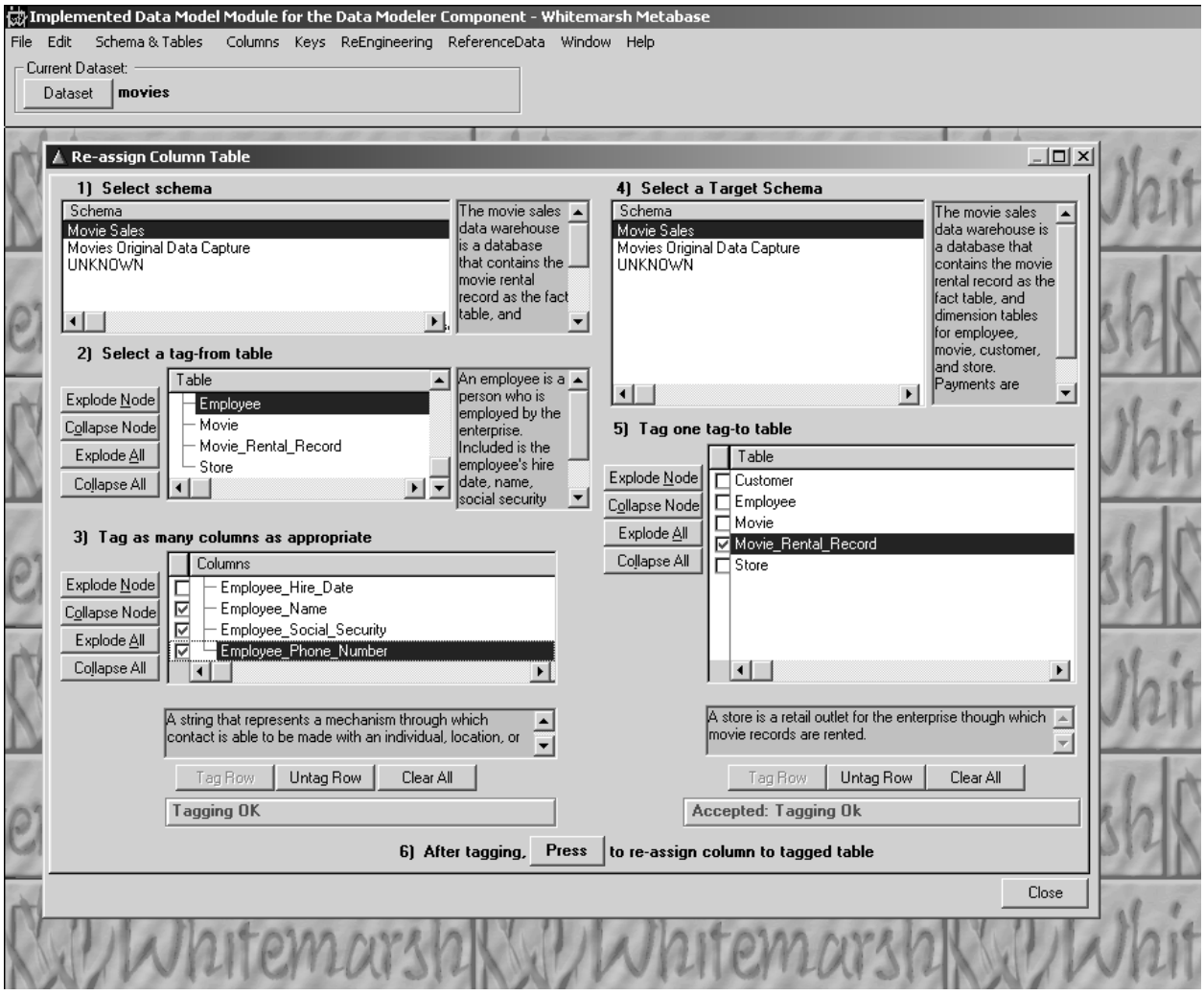

Figure 66. Reassign columns to tables.

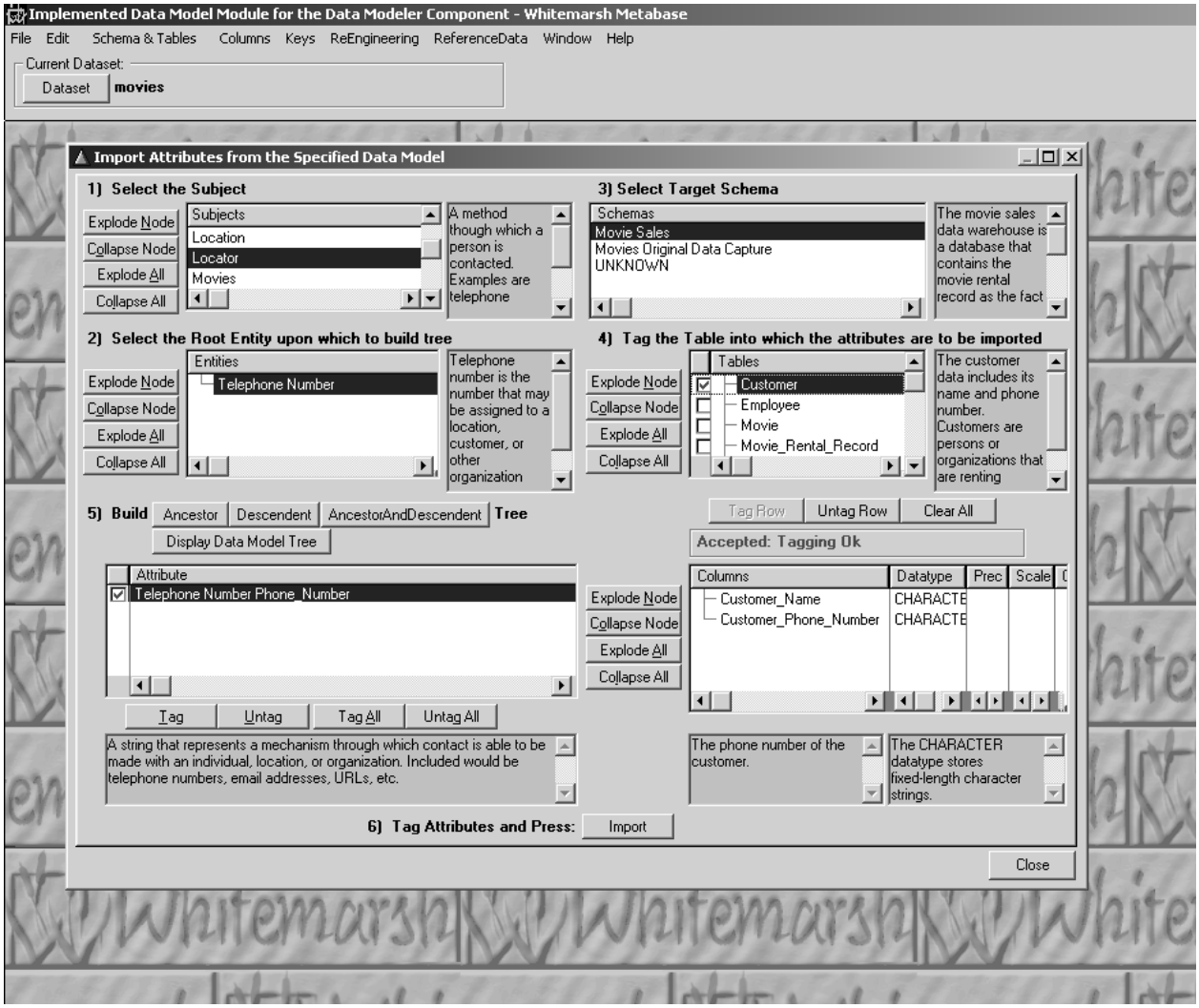

Figure 67. Import attribute from specified data model.

In this example, select subject, then entity and then tag one or more attributes. On the destination side, select the schema and tag the receiving table. Then press the import button. The process copies the attribute across. Repeat twice more, once for employee and then for store. At that point, all three tables have the column, Telephone Number including all its standardized semantics including data type and value domain. Also copied are all the abbreviations.

Once Telephone Number is copied into the tables, perform maintenance on the column to reduce the length of telephone number from 128 characters to just 10 characters. The data type that was assigned to the data element does not have a length more restrictive than the value domain of the value domain, formatted string. Proceed likewise through the rest of the columns in all the tables changing the data type lengths to a more realistic number. When you attempt to adjust the length of some of the columns an error message will surface indicating that such changes are not

allowed for foreign key column. In that case, only change the primary key column, and the press the button to reset the foreign key data types, precision, and scale.

6.1.4 Adjust Column Data Types

A review of the data types for the Implemented Data Model may make you wonder if they were all created by magic. No. Given that a data element is mapped to a value domain, then when a column is created in a table, its source attribute is checked to see if it has a related data element. If it does, and if it has an assigned value domain then the data type of that value domain is employed for the data type of the column. Actually, it's the SQL data type for that value domain's data type that is the actual source. Again, this is another example of the power of data elements: define once and use many times.

Figure 68 presents the data type re-engineering screen. Select the schema, then table, then tag one or more data types that are to change. Then tag the data type that the columns are to reflect and then press the reassignment button.

6.1.5 Create Primary and Foreign keys

The process of creating primary and foreign keys for the Movies Sales database consists of the following processes:

- Create the primary key
- Assign the primary key column
- Create the foreign key

The process of accomplishing this is contained in Sections 4.5.4 and 4.5.5. In this particular example, you need to add four foreign keys:

- Customer to Movie Rental Record
- Employee to Movie Rental Record
- Store to Movie Rental Record
- Movie to Movie Rental Record

6.1.6 Summary

The process of creating the Implemented Data Model is quite simple. This is attested to the brevity of this section of the document compared to the other sections. Simply put, once all the investment has been accomplished in creating the Specified Data and Data Element models then the process of creating a new database design is reduced to analysis and design, with the design activities concentrated on picking and assembling pre-standardized Specified Data Model parts into Implemented Data Models.

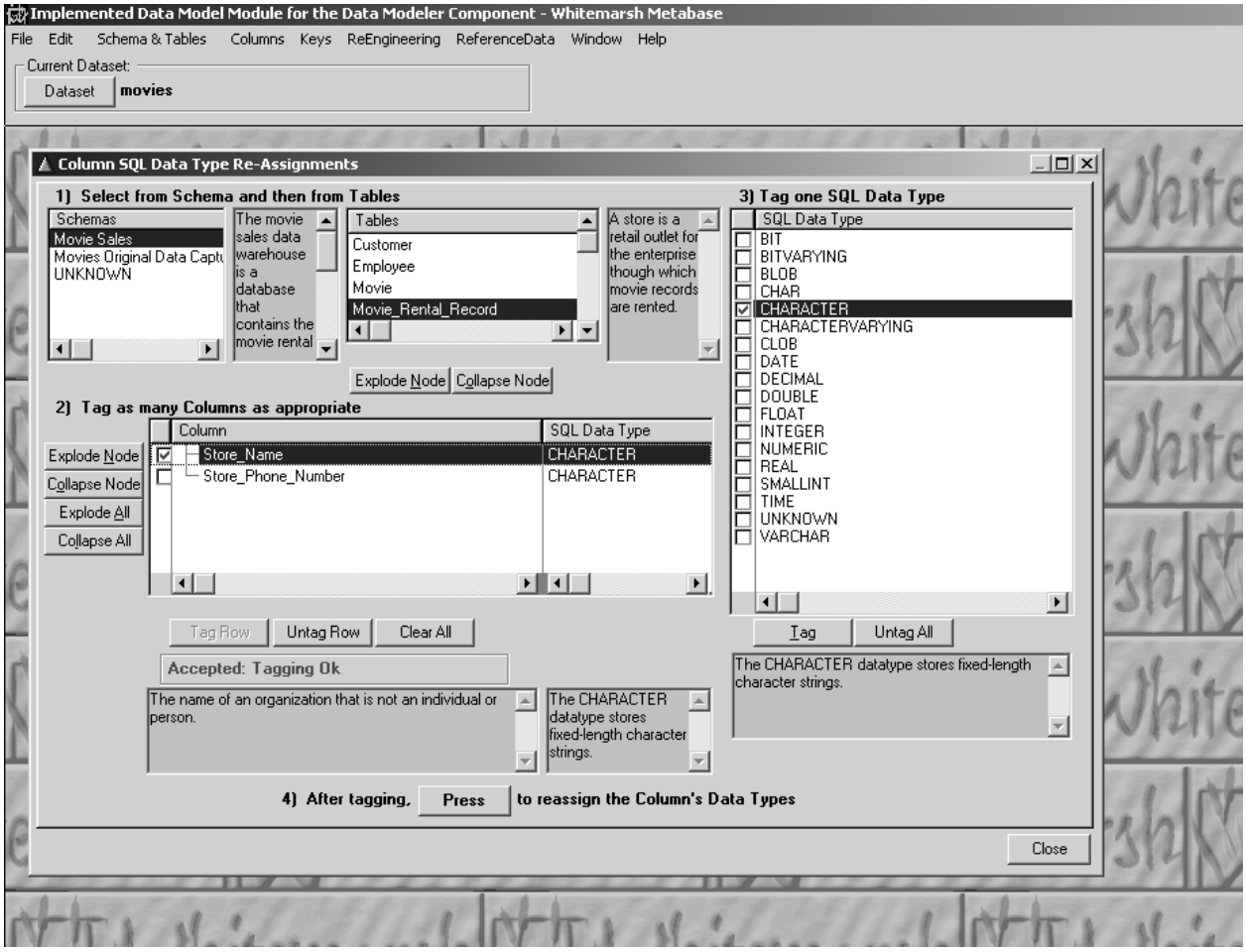

Figure 68. Column data type re-assignment.

6.2 Build an Operational Data Model

The process of building an Operational Data Model consists of the following steps:

- Create a database
- Create a DBMS Schema
- Import Implemented Data Model entities into the DBMS schema
- Add or delete DBMS columns as appropriate
- Adjust DBMS data types as appropriate
- Create primary and foreign keys

The process of building the Operational Data Model is simpler than building the Implemented Data Model. All of these activities occur within the Operational Data Model module.

6.2.1 Create a database

This process is the same as contained in Section 3.1.4.

6.2.2 Create a DBMS Schema

This process is the same as contained in Section 3.1.4. In this particular example, choose Clarion as the DBMS.

6.2.3 Import Implemented Data Model entities into the DBMS schema

This process is similar to that contained in Section 6.1.2 except you would choose the menu item, Import Schema Table Set. As you can see from that list, you can import single tables, a collection of tables that are related through primary and foreign keys, and individual columns. Since the data models are independent of each other, an operational dta model schema can ultimately contain tables from more than one Implemented Data Model schema.

6.2.4 Add or delete DBMS columns as appropriate

This process is the same as contained in Section 6.1.3.

6.2.5 Adjust DBMS data types as appropriate

This process is the same as contained in Section 6.1.4. If in step 6.2.2, above you choose Oracle as the DBMS, then you need to examine if the data types of Oracle are the set necessary to match the newly created Operational Data Model. If they are not, then you will have to do data type adjustment. For example, the data type for Date in Oracle is Datetime. If you do not pick the right data types then when you generate SQL DDL, the required data type will not be present and the generation step will terminate.

6.2.6 Create primary and foreign keys

This process is the same as contained in Section 6.1.5.

6.3 Generate SQL DDL

The process of generating SQL DDL is the same across all three models, that is, Specified, Implemented and Operational. Activate the menu item, Export through Schemas & Tables, Import and Export, SQL DDL, then Export. Figure 69 is then presented.

In this figure, select the database then schema, then press the Select Output File button. Choose the output file to which SQL DDL is to be generated. The next choice is to select how SQL DDL is to be generated for sub-type tables. The choices are: One, that is, all sub-typed tables are folded into one table, or Each which each sub-typed table is its own SQL Table connected by one-to-one relationships. Or SQL 1999 where the SQL DDL conforms to the SQL 1999 standard. That last choice is not implemented because no SQL DBMS has implemented that SQL 1999 feature.

Then you have another choice. Generate a complete set of SQL DDL for all the tables in a Schema (first button choice), or generate the SQL DDL for just a set of tables connected by the primary key of the high lighted table (i.e., Store) and it's connected table, which would be Movie Rental Record.

Once the SQL DDL is generated a "Finish" message is displayed. Press the View Generation Result button to see the resulting SQL DDL file.

That's it you have now created a new database design from existing metadata starting with the Specified Data Model and then the Implemented Data Model.

To "see" what the database design looks like, you can import the SQL DDL into an ER modeling tool like deZiner (www.datanamic.com). Figure 70 is the five table Movie Sales data warehouse.

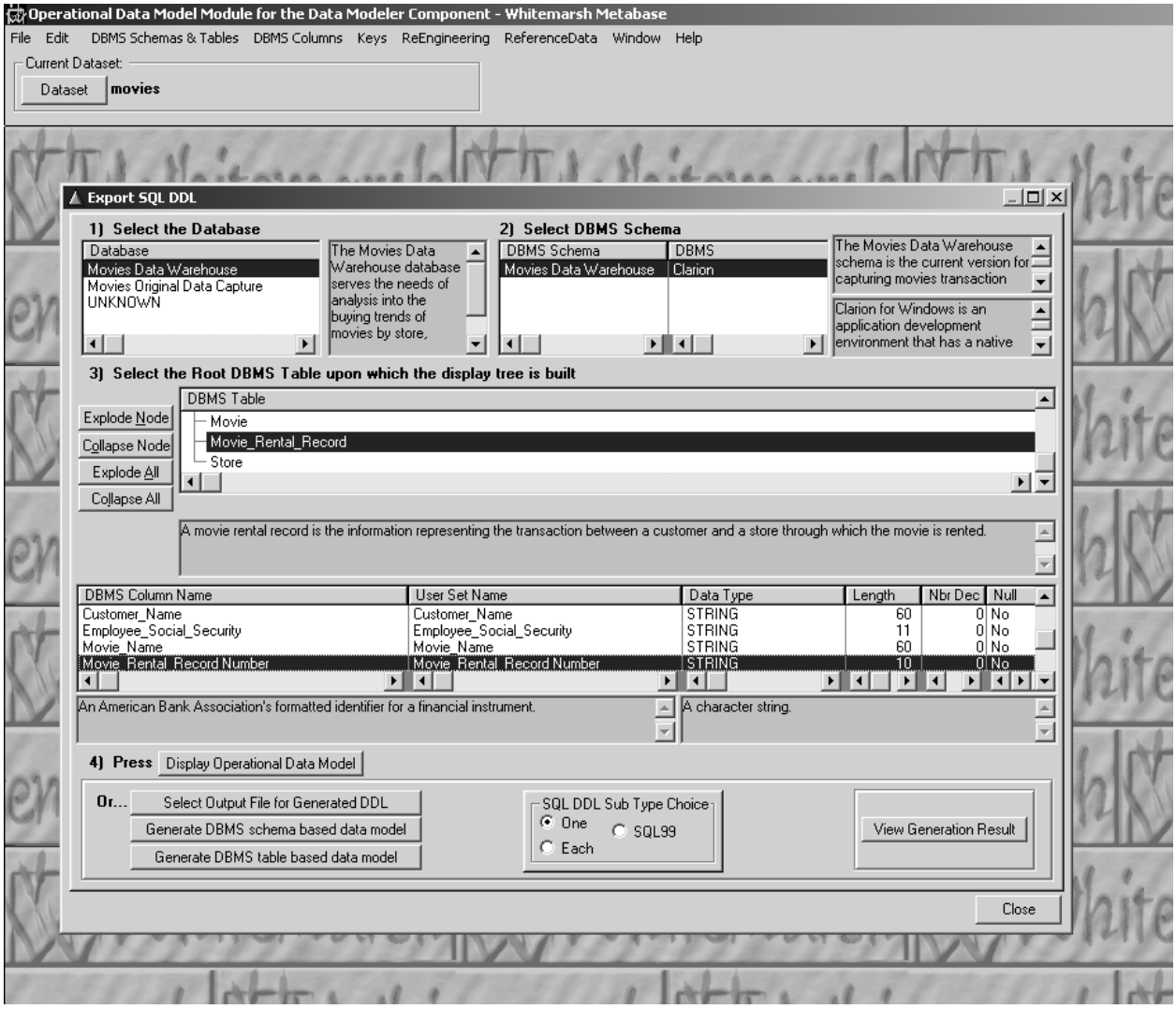

Figure 69. Generate SQL DDL.

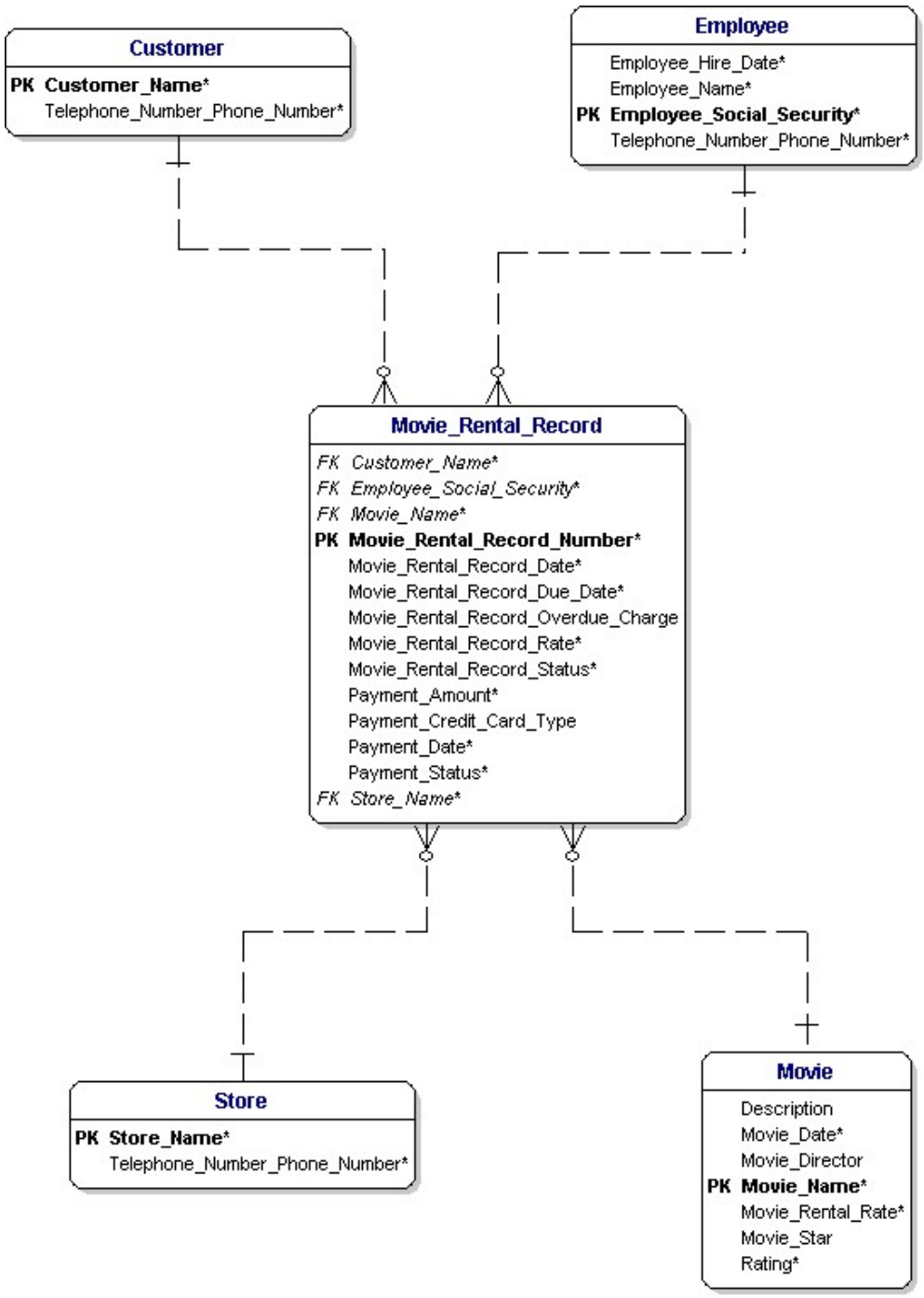

Figure 70. Movie Sales data warehouse data model diagram.

6.4 The Payoff

Table 16 that follows shows the benefit of having metadata from data element concepts all the way down to Operational Data Model DBMS columns. From this table you can see the counts in the table immediately below. Clearly there is reuse. And, that's the whole point of this effort

ISO 11179 Data Element Model			Specified Data Model	Implemented Data Model
Concepts	**Data Element Concepts**	**Data Elements**	**Entity Attributes**	**Database Columns**
10	15	21	35	93

In this very trivial and small example there is significant reuse. In general, every data element was re-used about 4.5 times. As the quantity of operational data capture databases grows, the quantity of concepts, data element concepts, and data elements will NOT grow in proportion. Rather they will converge ultimately on a high degree of re-use. The low estimate is about 30:1. This ratio comes from a study of a data analysis effort that was done on a class of U.S. Department of Defense applications in the early 1990s. That study showed the following statistics:

Activity	Quantity	Cost for standardization
Starting quantity of columns/fields	19,000	$6.75 million
Elimination of closely named columns and fields reduced the quantity to	3,000	$1.06 million
Elimination of same concept but very differently named columns and fields reduced the quantity to	560	$200,000

Concept	Data Element Concept	Data Element	Attribute	Column
Abstract Asset	Abstract Asset Characteristics	Product Description	Description	DESCRIPTION
				Movie Description
Communications Components	Communications Locator	Communications Locator String	Telephone Number Phone Number	CUSTOMER PHONE NUMBER
				Customer Phone Number
				DISTRIBUTOR PHONE NUMBER
				EMPLOYEE PHONE NUMBER
				Employee Phone Number
				STORE PHONE NUMBER
				Store Phone Number
Event Component	Event Date	Event Date	Employee Hire Date	EMPLOYEE HIRE DATE
				Employee Hire Date
			Movie Date	MOVIE DATE
				Movie Date
			Movie Rental Record Due Date	MOVIE RENTAL RECORD DUE DATE
				Movie Rental Record Due Date
			Payment Credit Card Expiration	PAYMENT CREDIT CARD EXPIRATION
			Payment Date	Movie Rental Record Payment Date
				PAYMENT DATE
	Event Name	Assessment	General Condition	GENERAL CONDITION
			Rating	Movie Rating
				RATING
Finance Components	Financial Instrument Identifier	Financial Instrument Identifier	Check Number	CHECK NUMBER
			Movie Rental Record Number	Movie Rental Record Number
		Payment Mechanism Name	Payment Credit Card Type	Movie Rental Record Payment Credit Card Type
				PAYMENT CREDIT CARD TYPE

Concept	Data Element Concept	Data Element	Attribute	Column
		Payment Mechanism Number	Credit Card Number	CREDIT CARD NUMBER
Information Technology Components	Information Technology Components Integer Identifiers	Identifier	Information Technology Attributes Identifier	CUSTOMER NUMBER
				CUSTOMER NUMBER
				CUSTOMER NUMBER
				DISTRIBUTOR ID
				DISTRIBUTOR ID
				EMPLOYEE ID
				EMPLOYEE ID
				EMPLOYEE ID
				MOVIE COPY NUMBER
				MOVIE COPY NUMBER
				MOVIE NUMBER
				MOVIE NUMBER
				PAYMENT TRANSACTION NUMBER
				PAYMENT TRANSACTION NUMBER
				RENTAL RECORD NUMBER
				STORE ID
				STORE ID
		Identifier	Information Technology Attributes Identifier	SUPERVISOR
Location Components	Location Address	Geopolitical Name	Location Address City	CUSTOMER CITY
				DISTRIBUTOR CITY
				EMPLOYEE CITY
			Location Address State	CUSTOMER STATE
				DISTRIBUTOR STATE
				EMPLOYEE STATE
		Postal Code	Location Address Zip Code	CUSTOMER ZIP CODE
				DISTRIBUTOR ZIP CODE

Concept	Data Element Concept	Data Element	Attribute	Column
				EMPLOYEE ZIP CODE
		Street Address	Location Address Street Address	CUSTOMER STREET ADDRESS
				DISTRIBUTOR STREET ADDRESS
				EMPLOYEE STREET ADDRESS
				STORE STREET ADDRESS
Organization Components	Organization Name	Organization Name	Distributor Name	DISTRIBUTOR NAME
		Organization Name	Store Name	STORE NAME
		Organization Name	Store Name	Store Name
	Organizational Identifier	Financial Institution Identifier	Check Bank Number	CHECK BANK NUMBER
		Government Institution Identifier	Employee Social Security	EMPLOYEE SOCIAL SECURITY #
				Employee Social Security
Person Components	Person Name Part	Person Name	Customer Name	CUSTOMER NAME
				Customer Name
				Customer Name
			Employee Name	EMPLOYEE NAME
				Employee Name
			Movie Director	MOVIE DIRECTOR
				Movie Director
			Movie Star	MOVIE STAR
				Movie Star
Real Asset	Real Asset Name	Product Name	Movie Name	MOVIE NAME
				Movie Name
				Movie Name
				Store Name
Transaction Components	Transaction Amount	Financial Amount	Movie Rental Rate	MOVIE RENTAL RATE
				Movie Rental Rate
		Transaction Amount	Payment Amount	Movie Rental Record Payment Amount

Concept	Data Element Concept	Data Element	Attribute	Column
				PAYMENT AMOUNT
	Transaction Date	Transaction Date	Movie Rental Record Date	MOVIE RENTAL RECORD DATE
				Movie Rental Record Date
			Movie Rental Record Overdue Charge	MOVIE RENTAL RECORD OVERDUE CHARGE
				Movie Rental Record Overdue Charge
			Movie Rental Record Rate	MOVIE RENTAL RECORD RATE
				Movie Rental Record Rate
	Transaction Name	Payment Mechanism Type	Payment Type	PAYMENT TYPE
	Transaction State	Transaction Status	Movie Rental Record Status	MOVIE RENTAL RECORD STATUS
				Movie Rental Record Status
			Payment Status	Movie Rental Record Payment Status
				PAYMENT STATUS

Table 16. Multiple use of concepts down through Implemented data model columns.

Attachment 1
Movie Rentals SQL DDL

```
CREATE TABLE MOVIE_COPY (
    MOVIE_COPY_NUMBER    INTEGER NOT NULL,
    GENERAL_CONDITION    INTEGER
);


ALTER TABLE MOVIE_COPY
    ADD PRIMARY KEY (MOVIE_COPY_NUMBER);


CREATE TABLE MOVIE (
    MOVIE_NUMBER        INTEGER NOT NULL,
    MOVIE_NAME          CHARACTER(25),
    MOVIE_DIRECTOR      CHARACTER(25),
    DESCRIPTION         INTEGER,
    MOVIE_STAR          CHARACTER(25),
    RATING              INTEGER,
    MOVIE_RENTAL_RATE    DECIMAL(7,2),
    MOVIE_DATE          INTEGER
);


ALTER TABLE MOVIE
    ADD PRIMARY KEY (MOVIE_NUMBER);


CREATE TABLE DISTRIBUTOR (
    DISTRIBUTOR_ID      INTEGER NOT NULL,
    DISTRIBUTOR_NAME    CHARACTER(25),
    DISTRIBUTOR_STREET_ADDRESS CHARACTER(45),
    DISTRIBUTOR_CITY    CHARACTER(20),
    DISTRIBUTOR_STATE    CHARACTER(2),
    DISTRIBUTOR_ZIP_CODE INTEGER,
    DISTRIBUTOR_PHONE_NUMBER INTEGER
);


ALTER TABLE DISTRIBUTOR
    ADD PRIMARY KEY (DISTRIBUTOR_ID);


CREATE TABLE STORE (
    STORE_ID            INTEGER NOT NULL,
    DISTRIBUTOR_ID      INTEGER,
    STORE_STREET_ADDRESS CHARACTER(45),
    STORE_PHONE_NUMBER   INTEGER,
    STORE_NAME          CHARACTER(25)
);
```

```
ALTER TABLE STORE
    ADD PRIMARY KEY (STORE_ID);


CREATE TABLE EMPLOYEE (
    EMPLOYEE_ID         INTEGER NOT NULL,
    STORE_ID            INTEGER,
    EMPLOYEE_NAME       CHARACTER(25),
    SUPERVISOR          INTEGER NOT NULL,
    EMPLOYEE_STREET_ADDRESS CHARACTER(45),
    EMPLOYEE_CITY       CHARACTER(20),
    EMPLOYEE_STATE      CHARACTER(2),
    EMPLOYEE_ZIP_CODE   INTEGER,
    EMPLOYEE_PHONE_NUMBER INTEGER,
    EMPLOYEE_SOCIAL_SECURITY_# INTEGER,
    EMPLOYEE_HIRE_DATE  INTEGER
);


ALTER TABLE EMPLOYEE
    ADD PRIMARY KEY (EMPLOYEE_ID);


CREATE TABLE CUSTOMER (
    CUSTOMER_NUMBER     INTEGER NOT NULL,
    CUSTOMER_NAME       CHARACTER(25),
    CUSTOMER_STREET_ADDRESS CHARACTER(45),
    CUSTOMER_CITY       CHARACTER(20),
    CUSTOMER_STATE      CHARACTER(2),
    CUSTOMER_ZIP_CODE   INTEGER,
    CUSTOMER_PHONE_NUMBER INTEGER
);


ALTER TABLE CUSTOMER
    ADD PRIMARY KEY (CUSTOMER_NUMBER);


CREATE TABLE PAYMENT (
    PAYMENT_TRANSACTION_NUMBER INTEGER NOT NULL,
    CUSTOMER_NUMBER     INTEGER,
    EMPLOYEE_ID         INTEGER NOT NULL,
    PAYMENT_TYPE        CHARACTER(20),
    PAYMENT_AMOUNT      DECIMAL(7,2),
    PAYMENT_DATE        INTEGER,
    PAYMENT_STATUS      CHARACTER(2),
    CHECK_BANK_NUMBER   INTEGER,
    CHECK_NUMBER        INTEGER,
    CREDIT_CARD_NUMBER  INTEGER,
    PAYMENT_CREDIT_CARD_EXPIRATION INTEGER,
    PAYMENT_CREDIT_CARD_TYPE CHARACTER(20)
);
```

```
ALTER TABLE PAYMENT
    ADD PRIMARY KEY (PAYMENT_TRANSACTION_NUMBER);


CREATE TABLE MOVIE_RENTAL_RECORD (
    EMPLOYEE_ID         INTEGER NOT NULL,
    RENTAL_RECORD_NUMBER INTEGER NOT NULL,
    MOVIE_COPY_NUMBER    INTEGER NOT NULL,
    MOVIE_NUMBER         INTEGER NOT NULL,
    CUSTOMER_NUMBER      INTEGER NOT NULL,
    PAYMENT_TRANSACTION_NUMBER INTEGER,
    MOVIE_RENTAL_RECORD_DATE INTEGER,
    MOVIE_RENTAL_RECORD_DUE_DATE INTEGER,
    MOVIE_RENTAL_RECORD_STATUS CHARACTER(2),
    MOVIE_RENTAL_RECORD_RATE DECIMAL(7,2),
    MOVIE_RENTAL_RECORD_OVERDUE_CHARGE DECIMAL(7,2)
);


ALTER TABLE MOVIE_RENTAL_RECORD
    ADD PRIMARY KEY (EMPLOYEE_ID, RENTAL_RECORD_NUMBER,
        MOVIE_COPY_NUMBER, MOVIE_NUMBER, CUSTOMER_NUMBER);


ALTER TABLE STORE
    ADD FOREIGN KEY (DISTRIBUTOR_ID)
                REFERENCES DISTRIBUTOR  (DISTRIBUTOR_ID)
                ON DELETE RESTRICT
                ON UPDATE RESTRICT;


ALTER TABLE EMPLOYEE
    ADD FOREIGN KEY (STORE_ID)
                REFERENCES STORE  (STORE_ID)
                ON DELETE RESTRICT
                ON UPDATE RESTRICT;


ALTER TABLE EMPLOYEE
    ADD FOREIGN KEY (SUPERVISOR)
                REFERENCES EMPLOYEE  (EMPLOYEE_ID)
                ON DELETE RESTRICT
                ON UPDATE RESTRICT;


ALTER TABLE PAYMENT
    ADD FOREIGN KEY (EMPLOYEE_ID)
                REFERENCES EMPLOYEE  (EMPLOYEE_ID)
                ON DELETE RESTRICT
                ON UPDATE RESTRICT;


ALTER TABLE PAYMENT
```

```
    ADD FOREIGN KEY (CUSTOMER_NUMBER)
                REFERENCES CUSTOMER  (CUSTOMER_NUMBER)
                ON DELETE RESTRICT
                ON UPDATE RESTRICT;


ALTER TABLE MOVIE_RENTAL_RECORD
    ADD FOREIGN KEY (PAYMENT_TRANSACTION_NUMBER)
                REFERENCES PAYMENT  (
          PAYMENT_TRANSACTION_NUMBER)
                ON DELETE RESTRICT
                ON UPDATE RESTRICT;


ALTER TABLE MOVIE_RENTAL_RECORD
    ADD FOREIGN KEY (MOVIE_NUMBER)
                REFERENCES MOVIE  (MOVIE_NUMBER)
                ON DELETE RESTRICT
                ON UPDATE RESTRICT;


ALTER TABLE MOVIE_RENTAL_RECORD
    ADD FOREIGN KEY (MOVIE_COPY_NUMBER)
                REFERENCES MOVIE_COPY  (
          MOVIE_COPY_NUMBER)
                ON DELETE RESTRICT
                ON UPDATE RESTRICT;


ALTER TABLE MOVIE_RENTAL_RECORD
    ADD FOREIGN KEY (CUSTOMER_NUMBER)
                REFERENCES CUSTOMER  (CUSTOMER_NUMBER)
                ON DELETE RESTRICT
                ON UPDATE RESTRICT;
```