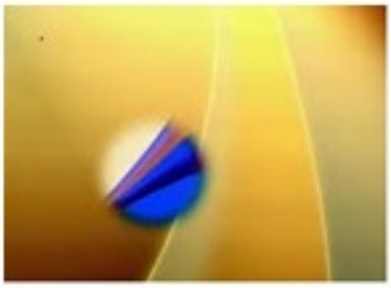
Entity Relationship and Data Flow Tutorial

Visible Systems Corporation
24 School Street, 2nd floor
Boston, MA 02108
617-902-0767

https://www.visiblesystemscorp.com

Email: contact@visiblesystemscorp.com

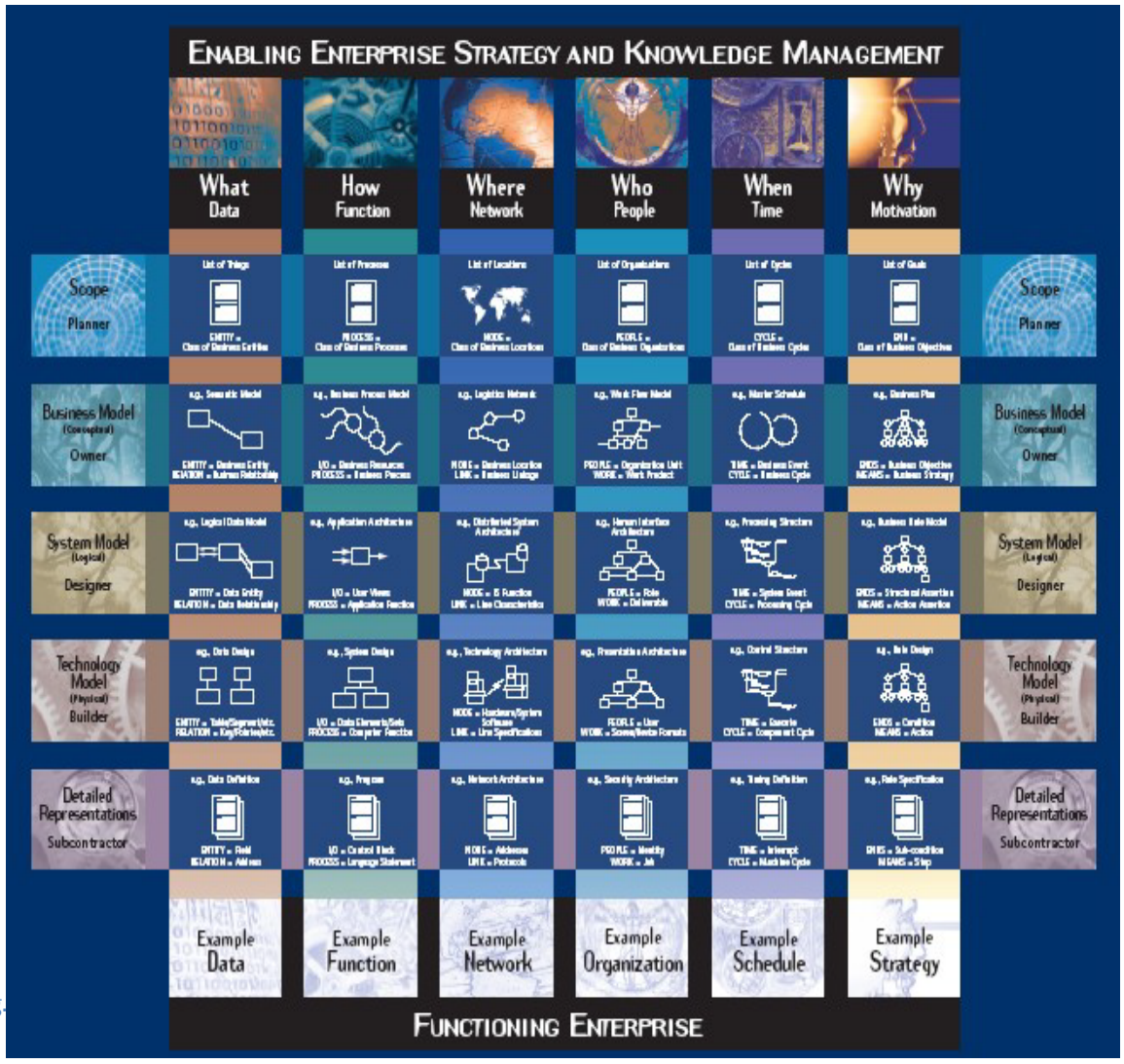


Visible®

Visualize. Align. Transform.™

Visualize patterns, align strategy and transform change into meaningful business outcomes.

Enterprise-wide Analysis, Design and Planning for Improvement.



This manual was prepared using Microsoft Word for Windows.

Visible Analyst
Tutorial on Structured Methods, Repository Management and The Zachman Framework

Lesson
Entity Relationship Diagrams

OVERVIEW

This data modeling technique provides a precise method for detailing and illuminating the interrelationships of the data used by a system. You can depict the ―entities‖ (see definition below) in the data you are modeling and the relationships between them by drawing them onto an entity relationship diagram (ERD). The data model (ERD) shows the major data objects of an application and how they fit together using the relationships. You can define the primary keys for the data entities and the composition of the data attributes of the entities in the Visible Analyst repository. (Defining primary keys and adding data attributes are explained in Lesson 16, Working with the Repository Functions.) The defined components can then be displayed on your ERD diagram by selecting these options from the View menu.

A diagram containing a picture of all or a subset of your data is called a ―view.‖ Each view can show an arbitrarily large or small part of your data model. You can show multiple views of your data model by including different combinations of entities and relationships on various diagrams. However, the entire data model, including the data elements composing each entity, is retained in the repository and can be accessed by creating a global view of the data model. This feature is explained in this lesson.

Definitions

The important diagram constructs in entity relationship data modeling include:

Entity — The entity (or, more properly, the entity type) is nothing more than a real-world object that you want to describe. The most generic type of entity is really a fundamental or independent entity but is usually simply called an entity. It is composed of data elements (also called attributes), and you can describe these in the entity's repository composition field. A fundamental entity is an object or event. It is represented on an entity relationship diagram as a rectangle and is accessed by the first symbol button on the control bar.

Associative Entity

Another type is the associative entity (sometimes called a junction, intersection or concatenated entity, a gerund or a correlation table). This is basically a relationship (see below) about which you want to store information. It can only exist between two other entities that participate in a many-to-many relationship. For example, the relationship between a customer and a product produces as a by-product the associative entity purchase order. A purchase order entity would not exist without the relationship between the other two entities. An associative entity is represented as a rectangle with straight diagonal lines across each corner. It is accessed by the second symbol button on the control bar.

Attributive Entity

The third entity type is the attributive or dependent entity. This is used to show data that is wholly dependent upon the existence of a fundamental entity. It is also used to show repeating subgroups of data. For example, the associative entity purchase order may have a dependent attributive entity named shipment showing the full or partial shipments that fulfill the purchase order. It is represented as a rectangle with rounded lines across each corner and is accessed by the last symbol button on the control bar.

Relationship

A relationship shows how one entity interacts with or can be affiliated with another entity. It appears on a diagram as a line drawn between two entities. Relationship lines ordinarily have two labels, one for each direction. The relationship lines can have terminators that show that the entities relate to each other on a one-to-one, one-to-many, or many-to-many basis (the relationship's cardinality), and whether the relationship is *optional(may), mandatory (must) or optional becoming mandatory (will eventually).* There are four line buttons on the control bar. Line types may be changed after they are drawn on the diagram.

Supertype/Subtypes

Specialized subtype entities can be created that are based on a generalized supertype entity and share common attributes. Only the attributes unique to the specialized entity need to be listed in the subtype entity. This is closely related to the object class inheritance concept. Visible Analyst also provides a detail field for specifying the exact number of relationships, if known. The supertype/subtype button is the fifth line button on the control bar.

Cluster

A cluster is a collection of entities and the relationships between them. It is not truly a part of your data model because it carries no new information. However, it can be very useful when you want to Show very large data models on a single diagram and still have

it comprehensible. You have the ability to cluster groups of entities and show these clusters and the relationships between them in summary fashion on a diagram. This limits the amount of detail on the diagram so that the larger outlines of what is contained in your data model are more visible.

The Automated Rapid Business Change (ARBC) editions of Visible Analyst (the ARBC Corporate Edition, the ARBC University Edition and the ARBC Student Edition) have the unique ability to derive Clusters of entities that are needed by each process.

The last line of each cluster is either an Associative or Intersecting entity (for a Process), a Subtype entity (for a database) or a 5BNF Structure [4] entity (for an Expert System Knowledgebase). These Clusters show the Project Phase Number of each entity.

The Cluster Report is listed in Outline format (similar to a Gantt Chart) to be used as derived Project Plans for Managers to prioritize urgent processes for early delivery.

A cluster is created in the repository and entities are added to its composition field. A cluster view can then be created by Visible Analyst to display the *pseudo*-relationships between clusters rather than real relationships between specific entities. The diagram Visible Analyst generates is an *unstructured* diagram, but the information contained in the diagram pertains to your entity relationship diagrams. For more information on Clusters, see the *Operation Manual* or the online help system.

View Object

A view object can be thought of as a derived or virtual table. It is composed of two components: a list of column names and a select statement used to filter information from the tables in the view. For each view, there is one primary select clause and any number of sub-select and union select clauses. Using the Define View dialog box, you select the tables and columns and define the join relationships, clauses and flags to be used by the view. For more information on view objects, see the *Operation Manual* or the online help. (View objects are not available in the Education Editions of Visible Analyst.)

pg.

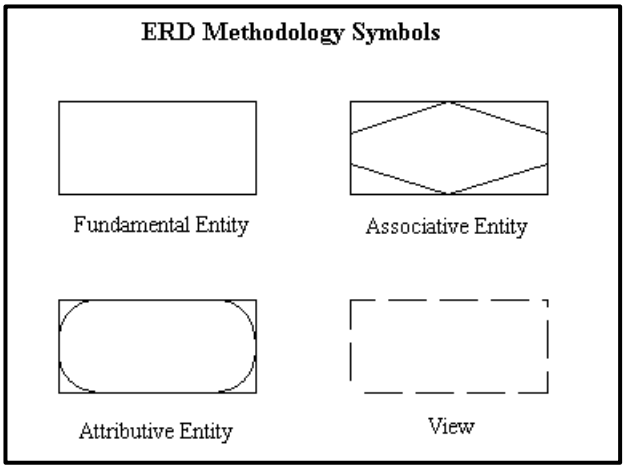


Figure 7-1 Entity Relationship Diagramming Symbols

Relationship Cardinality

Visible Analyst supports four different relationship cardinality notations: IDEF1X, Crowsfoot, Arrow, and Bachman. The type of notation you use is up to you, and you select it when a new project is created. The number of names per relationship line is also your choice. You can indicate one or two names per relationship. For this lesson, we use the standard crows foot notation with two names per relationship.

If you select IDEF1X as the relationship cardinality when creating the project, the default notation is IDEF1X. You would then select crows foot, Arrow or Bachman as an alternate cardinality notation.

THE EVOLUTION OF DATA MODELING

By Clive Finkelstein [1]

Data Modeling today is an integral part of Systems Development and Maintenance. It was not always so. In the 1970's Software Engineering was the preferred development method, and for many organizations it still is. When we were developing Information Engineering (IE) from 1976-1980 we found in project after project a high degree of redundant data versions in data bases developed using Software Engineering. These were very expensive to develop and maintain because any data value changes had to be synchronized across all redundant data versions. This data redundancy cost large organizations hundreds of millions of dollars of ***Systems Development and Systems Maintenance Costs*** and time.

This also required data maintenance programs to be developed redundantly, which resulted in redundant data entry, redundant work, redundant staffing, redundant equipment and redundant floor space. This data redundancy also cost large organizations hundreds of millions of dollars of ***Annual Operating Costs*** and time.

In the late 1970's data modeling methods started to emerge, based on the work by Dr Edgar Codd [2]. This was due to Normalization following his research work on mathematical set theory.

Normalization ensures that attributes are positioned in tables where they are wholly dependent on the primary key. It is based on 5 Normalization Rules: First Normal Form (1NF); Second Normal Form (2NF); Third Normal Form (3NF); Fourth Normal Form (4NF); and Fifth Normal Form (5NF). When correctly applied to data bases in Third Normal Form (3NF) the number of redundant data versions are reduced. This is a data-oriented method, while Software Engineering (SE) is process-oriented.

It took many years for the Industry to recognize that data is more stable than processes. Processes are very

---

[1] Clive Finkelstein is acknowledged World-Wide as the "Father" of Information Engineering (IE) developed by his Australian Company Information Engineering Services Pty Ltd (IES) and himself in Sydney from 1976-1980.
He is also acknowledged as the "Father" of "Enterprise Engineering" (EE) from 1995-2000, developed by him as an enhancement to IE to deliver Enterprise Architecture (EA) projects rapidly into production as data bases and systems in 3-month increments.
From 2014-2020 he developed the Automated Rapid Business Change (ARBC) methods to eliminate Redundant Data Versions and deliver changed data bases and systems in days or weeks.

[2] Dr Codd was an IBM Research Fellow in the IBM Research Labs in San Jose. He died in 2003.

volatile and are changed by many factors: management decisions; new technology; and competition.

Data Modeling was applied by data modelers interviewing business experts one-on-one. If a business expert did not have personal knowledge on how a data element was used; this issue was often left till later to be resolved. In many cases it was not resolved by anyone but was left to the programmer (who was the least knowledgeable person to make a decision about the issue). The programmer then locked the decision in manual code, which was extremely expensive to change.

To avoid this problem, we found that it was more effective to bring all business experts who were knowledgeable about the data, who together could discuss any data issues and make an informed collective decision to resolve the issue.

We found in project after project that business experts had difficulty understanding the Normalization Rules, which were academically worded. We experimented with changing the academic definition of each rule to a business-oriented definition that explained "how" to apply each rule. This was very successful and enabled the business experts to participate actively in the data modeling sessions.

We also found that their active participation eliminated redundant data versions that had been missed by 3NF. We called these business-driven rules "Business Normalization [3]" with 5 Business Normalization Rules: 1BNF; 2BNF; 3BNF; 4BNF; and 5BNF to distinguish from "Traditional Normalization"; 1NF; 2NF; 3NF; 4NF and 5NF.

4BNF identified supertype and subtype entities, for which the business expert data modelers could provide much data attribute detail. 5BNF was unique, as it identified relationships between occurrences of data entities using common knowledge, which was expert knowledge. 5BNF resulted in the development of Knowledge Bases for Expert Systems. 5BNF is very powerful and captures expert knowledge held by a few experienced business experts that could be shared by all business people in the organization.

I documented Business Normalization in a separate chapter for the book that I was co-authoring with James Martin in 1980 [4]. This was a critical chapter. It made IE data-oriented and business-driven in contrast to Software Engineering, which was process- oriented and IT-driven. When the book was published in November 1981 this critical chapter had been excluded and replaced by a chapter on "Data Design". This was a mainframe software package for Normalization developed by James Martin's company: Data Design Inc. (which later was renamed Knowledgeware).

When James Martin subsequently published 3 books on Information Engineering in 1986-1988 there was no mention of Business Normalization and when Knowledgeware subsequently released "Information Engineering Workbench" (IEW) and later "Application Development Workbench" (ADW), the benefits of Business Normalization were missing. The books and modelling tools (IEW and ADW) were clearly process-oriented and IT-driven.
Knowledgeware had obviously used Software Engineering concepts. They did not understand the benefits of Information Engineering and Business Normalization. The problems and costs of redundant data versions were still there. However, the industry accepted these books and modeling tools without question because they

[3] See the Reference Textbook: Clive Finkelstein, *"Enterprise Architecture for Integration: Rapid Delivery Methods and Technologies", Third Edition, IES, (2015).* Chapter 6 describes *"Business Data Modeling. Concepts",* while Chapter 9 describes *"Business Normalization Concepts".*

[4] James Martin and Clive Finkelstein, "Information Engineering", Savant Institute, Carnforth, Lancs UK (Nov 1981).

were associated with James Martin's company [5].

Recognizing Data-Oriented and Process-Oriented Data Models.

A Data-Oriented, Business-Driven Data Model uses the crows-foot symbol on a relationship line to indicate **many** entity occurrences of the entity touched by the crows-foot symbol. One entity occurrence touched by a relationship line is indicated by the absence of a crows-foot.

A Process-Oriented, IT-Driven Data Model also uses the crows-foot to indicate **many** occurrences of the entities touched by the crows-foot symbol. One occurrence touched by a relationship line is indicated by a vertical bar across the relationship line.

Both of these notations indicate a *one-to-many* relationship between two entities. To indicate *mandatory-one-to-many*, a vertical bar is placed at the *"one"* end of the relationship line. For the Process-Oriented data model this means that two vertical bars are used to indicate *mandatory-one*, while for the data-oriented data model only one vertical bar is needed to show *mandatory-one*.

In Summary: if a data model shows two vertical bars for *mandatory-one*, it is a Process-Oriented data model. If a data model shows only one vertical bar for *mandatory-one*, then it is a Data-Oriented data model.

Business Rules Notation

BUSINESS RULES NOTATION

We will now discuss the symbols used on Relationship lines to represent Business Rules.

Table 7-1: Using Relationship Symbols for Data-Oriented, Business-Driven Business Rules

Symbols in Tables 7-1 and 7-2 are used to show Business Rules and are referenced each by a number.

- The crows-foot symbol (2) on the relationship line indicates many entity occurrences
- The absence of a crows-foot (1) indicates one entity occurrence
- One vertical bar (1) on a relationship line indicates one entity occurrence (must have one)
- The vertical bar (4) indicates mandatory-many (must have many)
- The O (5) indicates optional-one (may have one)
- The O indicates optional-becoming mandatory one symbol (7) (will, eventually have one) (Complex Business Rules)

Table 7-1: Using Relationship Symbols for Data-Oriented, Business-Driven Business Rules

Business Rule Symbol	Cardiinality and Nature	Business Rule Symbol
1		2
One		Many
3	Mandatory (| must)	4
Must Have One		Must Have Many
5	Optional (O may)	6
May Have One		May Have Many
7	Optional Becoming Mandatoy (O| will, eventually)	8
Will (eventually) Have One		Will (eventually) Have Many
	Recursive (5BNF)	Two Entity Occurrences

[5] Tragically, James Martin drowned in 2013, while swimming off his private island in Bermuda.

- The O indicates optional-becoming mandatory many symbol (8) (will, eventually have at least one or many) Complex Business Rules)
- The recursive symbol (9) indicates a 5BNF entity with expert knowledge between entity occurrences.
- When a symbol is added to a Relationship line, the Business Rule associated with that symbol is automatically added as the Relationship name. That name can later be changed if a different name is preferred.

Business Rules Notation

Table 7-2: Using Relationship Symbols for Process-Oriented, IT-Driven Business Rules

Business Rule Symbol	Cardiinality and Nature	Business Rule Symbol
1		2
One		Many
3	Mandatory (| must)	4
Must Have One		Must Have Many
5	Optional (O may)	6
May Have One		May Have Many
7	Optional Becoming Mandatoy (O| will, eventually)	8
Will (eventually) Have One		Will (eventually) Have Many
	Recursive (5BNF)	Two Entity Occurrences

Table 7-2: Using Relationship Symbols for Process-Oriented, IT-Driven Business Rules

Table 7-2 shows symbols used for Process-Oriented Business Rules

- The crows-foot symbol (2) on the relationship line indicates many entity occurrences
- One vertical bar (1) on a relationship line indicates one entity occurrence
- The second vertical bar (3) indicates mandatory-one (must have one)
- The vertical bar (4) indicates mandatory-many (must have many)
- The O (5) indicates optional-one (may have one)
- The O 6 indicates optional-many (may have many)
- The optional-becoming mandatory symbol is not supported (Complex business rules)
- The optional-becoming mandatory symbol is not supported (Complex business rules)
- The recursive symbol (9 indicates a 5BNF entity with expert knowledge between entity occurrences.
- When a symbol is added to a Relationship line, the Business Rule associated with that symbol is automatically added as the Relationship name. That name can later be changed if a different name is preferred.

Cluster Analysis and Cluster Report

Visible Analyst has the capability to automate the analysis of an ERD data model to identify the project plans of sub-projects that can be prioritized and extracted from the data model for early delivery into production. This analysis is called “Cluster Analysis” and the sub-projects that are identified are called “Clusters”. They are documented in a “Cluster Report” that is printed in sub-project priority sequence.

The Relationship Symbols that touch each entity are used to apply relevant Business Rules as follows:

- An entity that is touched only by mandatory-one symbols is a "Phase 1) entity. It appears in the cluster as 1) PROJECT, for example.
- If the other end of the relationship line is a many symbol, that entity is a Phase 2) entity. It appears in the cluster indented one position to the right as 2) PROJECT BUDGET, for example. [PROJECT BUDGET will be an Associative (Intersecting) entity and the cluster represents an Activity or a Process. BUDGET will be another Phase 1) entity. This cluster is shown next.

CLUSTER NAME: PROJECT BUDGET MANAGEMENT ACTIVITY
1) PROJECT
1) BUDGET
2) PROJECT BUDGET (PROJECT BUDGET MANAGEMENT ACTIVITY)

- If the many symbol touching PROJECT BUDGET is mandatory-many, the cluster is a ***Reusable Process*** and is shown in the cluster with all entities in bold as shown above. The automatically generated Cluster Name can later be renamed, if preferred.

If another Phase 1 entity (say BUDGET) touches the PROJECT BUDGET entity in an optional-many or optional-becoming-mandatory-many, the cluster still represents a reusable process. As a general rule of thumb, any two entities related in mandatory-many to an optional-many or optional-becoming-mandatory-many will result in at least one Reusable Process.

A cluster that contains supertypes and subtypes will have separate clusters with each subtype entity as the last line of the cluster (called the "End-Point" entity). These clusters with subtype End Point entities are "data base clusters", which take the Cluster Name from these subtype entities. For example:

Supertype: INSTRUCTOR
_____|_____
| | |
Subtypes: PROFESSOR LECTURER TUTOR

The above example of supertype and subtypes will produce 3 data base clusters:
PROFESSOR database; LECTURER database; and TUTOR database.

If an end-point entity is 5BNF the cluster represents an Expert System Knowledgebase. For example:

5BNF Entity: SALESPERSON will appear as the cluster SALESPERSON Expert System Knowledgebase

Milestone Clusters

Some clusters can be very complex, with many entities. If you find any entities touched by ***one or more mandatory-one symbol and at least one or more many symbols***, that entity is a potential "Milestone" entity. The large cluster in which it resides can be broken at the Milestone entity into two or more smaller "Milestone Clusters" for progressive delivery into production as the Milestone databases and systems are completed.

Moving Clusters into a Different Functional Area

The clusters in the Cluster Report represent systems and databases that are likely needed by different

functional areas of the organization. The data model should be displayed (by File < OPEN DIAGRAM). Click on a Associative (Intersecting) entity to highlight it in a different color. A Select Menu will enable you to choose "Select Cluster". All of the other entities in that cluster will be highlighted in the same color. SChoose the "Move Menu" where all of the entities in the cluster will be listed together with a drop-down list of functional areas that were defined in the FDD. Choose the functional area that the cluster should be moved into and click OK. When the data model for that selected functional area is displayed all of the moved entities in that cluster will be displayed.

Data Model Display Orientation

The data model can be displayed in either a horizontal or vertical orientation.

Horizontal Orientation: All Phase 1 entities are displayed on the top row of the data model diagram; Phase 2 entities are displayed in the second row; Phase 3 entities are displayed in the third row – in a "top-down orientation. Senior Managers will be interested in the top 2 or 3 rows, while detailed operational entities will be displayed in the lower rows.

Vertical Orientation: All Phase 1 entities are displayed vertically on the left side of the diagram; Phase 2 entities are displayed 1 column to the right; with the highest phase number displayed on the right hand side of the diagram. The result is a data model diagram that appears similar to a Pert Chart.

Potential Export Capability to Microsoft Project

The Clusters can potentially be exported to Project Management Software Products such as Microsoft Project. This is described in Chapter 7 of the Reference Textbook.

BUSINESS NORMALIZATION

We will now discuss Business Normalization Principles, which are used by Data-Oriented Business-Driven Data Models to eliminate data redundancy. We will use a Purchase Order Form (see below) as a source document to provide input to Business Normalization in an unnormalized Entity List format.

Purchase Order Entry

PURCHASE ORDER

PO No ____ PO Date ____

Cust No ____ Cust Name _____

Cust Address _____

Supplier No ____ Supplier Name _____

Supplier Address _____

Contract No ____

Product No	Product Name	Qty Ordered	Agreed Unit Price

ENTITY LIST NOTATION FOR BUSINESS NORMALIZATION

ENTITY PURCHASE ORDER

(primary key#,
foreign key#,
[secondary key],
(group attribute),
elemental attribute
((repeating group)),
{derived attribute}

(po no#,
order no#,
[customer no],
(customer address),
po date
((product no, product unit price, qty ordered)),
{po total amount}

Unnormalized Entity List of Attributes for Business Normalization

PURCHASE ORDER (po no#, order no#, [customer no], (customer address), p
date ((product no, product unit price, qty ordered)), {po total amount}

We will extract the field names in the Purchase Order Form above and write them in Entity List Notation as shown in the left of this table.

A data entity name is always shown in all capitals.

- For example, a key attribute is shown with a terminating # (hashtag or Pound sign). A primary key is underlined, while a foreign key is not underlined.
- (A foreign key exists as a primary key in another data entity and links the two entities together.)
- A [secondary key] is written surrounded by square left and right brackets. They are not technically

"keys" (which must be unique) but are more correctly called "selection attributes" as they need not be unique. A good example is "supplier name". Secondary keys or "selection attributes" are implemented as Indexes in a database,

- A (group attribute), such as (supplier address) signifies that it has not yet been decomposed into its elemental attributes, such as "street number, street name, suburb or town, city, state postal code or country.
- A {derived attribute} is surrounded by left and right curly braces, such as an "total amount" to signify that its value is calculated by a formula that is yet to be defined.
- A ((repeating group)) is a number of attributes separated by commas – all surrounded by double left and right curved brackets to signify that all of the attributes occur multiple times.
- The unnormalized entity at the bottom is used as input to Business Normalization to completely eliminate all data redundancy.

DEVELOPING YOUR DATA MODEL

Each entity relationship diagram is complete in and of itself and shows one view of the data model of your project. (Remember that a view is a portion or subset of your entire data model represented on a single diagram.) When beginning your data model, you must manually add new entities and relationships to a view diagram. After this has been done, you can create additional views by using the File menu View function to select existing entities and relationships from the repository. Visible Analyst automatically draws the views for you. Then you can add to or subtract from each view and rearrange it as you wish. Thus you avoid having to draw portions of your data model repeatedly on different views.

Adding Entities to a View

Since the basic building block of the data model is the entity type (or simply, the entity) and since relationships cannot exist except to relate already existing entities, you begin by adding entities to a view.

Set the Zoom Level:	1	From the View menu, select 66% zoom so that you can see all of your needed workspace.
Create a New Diagram:	2	From the File menu, select New Diagram.
	3	Select the diagram type to be Entity Relationship with standard drawing method.
	4	Select the Page Size to be Standard.
	5	Click OK.
Add Entities:	6	Click the first symbol icon, the rectangle. This is a fundamental entity.

7 Place the cursor in the middle of the diagram workspace and click the *left* mouse button. An entity is drawn.

8 Name the entity ―Student Driver‖ and click OK.

9 Add another fundamental entity below the first, and name it ―Driving School.‖

10 Add another fundamental entity below Driving School, and name it ―Driving Lessons.‖

Figure 7-2 New Entities

Save the Diagram: 11 From the File menu, choose Save and name the diagram ―Driving School View.‖

Changing a Symbol Type

In the diagram we have created, the entity Driving Lessons is actually an attributive entity because the entity exists solely because it is an attribute of the fundamental entity Driving School. Since we placed it on the diagram as a fundamental entity, it is necessary to change the symbol type.

Select Symbol to Change:

1 Put the cursor in selection mode by clicking the ↖ button on the control bar.

2 Click the symbol labeled Driving Lessons with the *right* mouse button so that its Object menu appears.

Change the Entity Type:

3 Select Change Item. The Scope must be set to Global change in the Change Object dialog box. This option is important when you change an object's type or label. Selecting Global causes the change to be made on every diagram where that object occurs. If you select Individual, the change is only made to the selected object. A Local change would modify all occurrences on the current diagram. All changes to a symbol type must be Global.

4 Select Change Type.

5 Select Attributive Entity and click OK.

6 Click OK on the Change Object dialog box. The symbol is changed on the diagram.

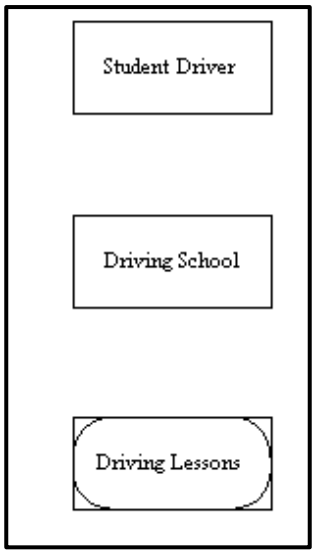


Figure 7-3 Changed Entity Type

Adding Relationship Lines

We need to establish the relationships between the entities on the current diagram.

Draw the Relationship:

1 Click the first line button on the control bar.

2 Draw a line from Driving School to Student Driver. The procedure is the same as that used to draw a line in Lesson 5 - Diagramming. Click and hold the *left* mouse button where you want the line to begin, drag the line to where you want it to end. If you release the button within the symbol, the line is connected automatically. If not, you must double-click the left mouse button to end the line.

Note

When you use an elbow line and the elbow in the line does not bend in the direction that you want it to, click the *right* mouse button while you are still holding the *left* one, and the elbow inverts.

Label the Relationship:

3 Enter ―Instructs‖ for the label of the first relationship. To set the relationship cardinality, click One for the Minimum, and click Many for the Maximum. This means that ―Driving School instructs one or many Student Drivers.‖ If you know the exact maximum number of relationships, you can enter it in the detail box. (See Figure 7-4.)

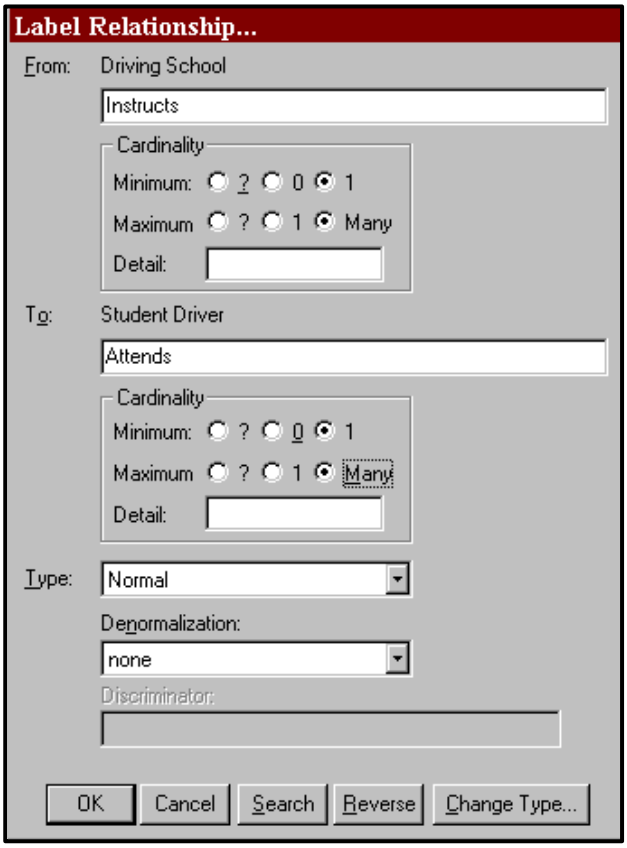


Figure 7-4 Label Relationship Dialog Box

4 Press the TAB key to move the cursor to the next field or click the mouse in the other label field.

5 Enter ―Attends‖ for the reverse relationship name. For the Minimum click One, and for the Maximum click Many. (This deliberate error is added to demonstrate the capabilities of the Analyze function.) It means a ―Student Driver attends one to many Driving School.‖ Both of these relationships are considered mandatory because it is necessary to attend driving school to be a student driver, and it is necessary to have students to be a driving school. Ensure that Type is set to Normal, and click OK.

Draw Another Relationship:	6	Draw a line from Driving School to Driving Lessons. For the first label, type ―Offers,‖ and set Minimum to Zero and Maximum to Many. For the second label, type ―Are Provided By.‖ Because this is an Identifying relationship, the cardinality is automatically set to 1:1. Click OK.
Save:	7	Press CTRL+S to save the diagram.

Analyzing the Diagram

The Analyze function checks to ensure that the diagram is syntactically correct, meaning that all relationship lines and symbols are labeled. You can also use the Analyze function to check for certain normalization errors.

Start Analyze:	1	Select Analyze from the Diagram menu.
	2	Choose Current Diagram and Syntax Check. Click OK. It tells you that the current diagram is correct.
Insert an Error:	3	Add a symbol to the diagram without naming it.
Analyze Again:	4	Run Analyze again. You see an error message indicating that there is one unnamed entity. Click Cancel to return to the diagram. The unnamed entity can be deleted from the diagram by highlighting it with the cursor in selection mode and pressing Delete.
Analyze Still Again:	5	Run Analyze again, but this time choose Normalization. You see the error message that the relationship ―Driving School [Instructs] Student Driver‖ is not normalized. This is true. The error indicates that the cardinality is 0:many or many:many in both directions. It is flagged as an error because optional:optional and many:many relationships can be difficult to implement. Click Cancel to close the box.
Correct Cardinality Error:	6	To change the cardinality of the relationship Attends, click the relationship line with the *right* button.
	7	Select Change Item. Change the cardinality for Attends from a maximum of Many to a maximum of One.

8 Click OK.

Analyze Once More: 9 Select Analyze from the Diagram menu. Choose Normalization and click OK. The diagram is now correct.

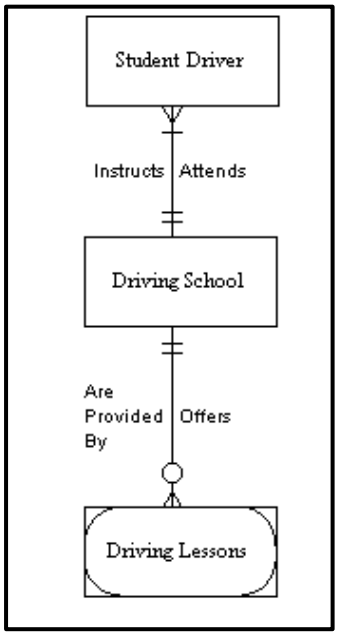


Figure 7-5 Normalized Diagram

Automatically Generating a View of Your Data Model

Another very useful feature of Visible Analyst is the ability to generate new data model views automatically. Since a data model can become very large and sometimes very difficult to decipher with many relationship lines and symbols, generating a specific view of the data model allows you to focus on one portion of your data model without having to redraw all of the symbols and connections that you want to have on the diagram. The function for generating a view is found on the View of Data Model submenu from the File menu.

There are three different options for generating a view from this function.

- There is an option to generate a Global view of your data model. All of the entities and relationships that are in the repository are placed on one diagram. This feature is important when additions are made to one portion of the data model and you would

like to see how those changes have affected the entire model. Another use for this feature is to generate an entity relationship diagram for imported entity information.

- You can generate a New view, allowing you to choose from the entities you have already created on a diagram or in the repository those entities and attached relationships you would like displayed on a new diagram. This allows you to make additions or changes to your entire data model while concentrating on only one portion.
- The other view option from the View of Data Model option is Process. A Process view is an entity relationship diagram that represents a subset of your data model and is based upon a process existing on a data flow diagram or in the repository. Data elements that enter or leave the selected process in data flows and that are also contained in the composition of entities cause those entities to appear in the process view, along with the relationships existing between pairs of entities. A process view allows you to concentrate on the specific portion of your data model that is involved with the selected process. This is the type of view that you now create. The composition information for the entities that appeared, as well as the attribute information of the particular process, has already been entered for you in the sample diagrams we supplied. This is so that you do not have to enter the information necessary to demonstrate this feature of Visible Analyst.

To create the process view:

Start View Generation: 1 Select View of Data Model from the File menu, then choose Process. The Select Process for Views dialog box appears.

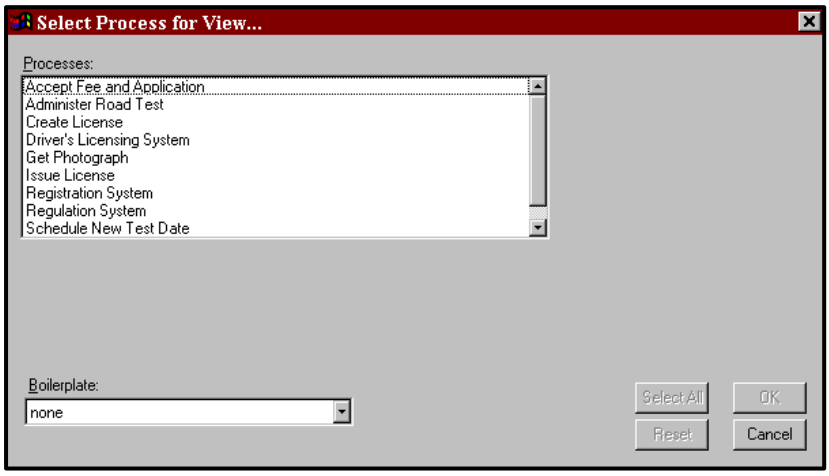


Figure 7-6 Process View Dialog Box

Select the Process:	2	Click the process Issue License and click OK. Visible Analyst searches the repository for entities that contain data elements in common with the data flows that are attached to Issue License and creates a ―View‖ of the data model.
Save the New View:	3	Select Save from the File menu.
	4	Title the diagram ―Process View: Issue License.‖ This diagram is a subset of your entire data model.
	5	Click OK.

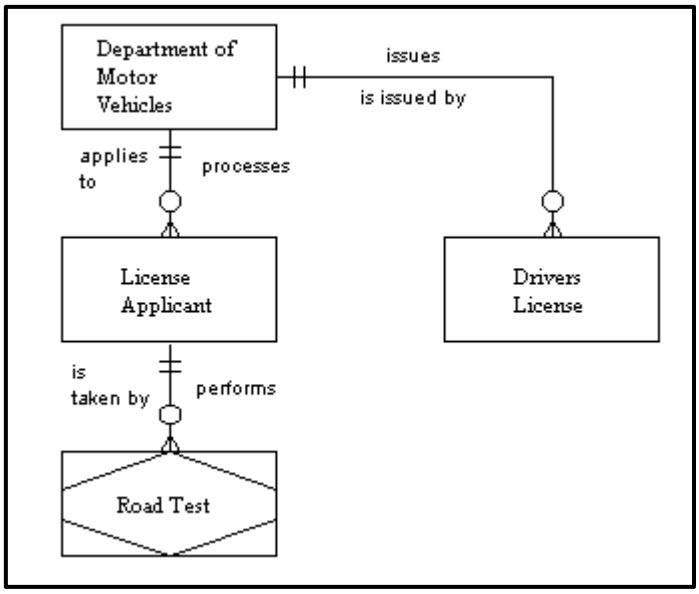


Figure 7-7 The Generated Process View

Lesson
Data Flow Diagrams

OVERVIEW

As described in Lesson 4 – Structured Modeling Techniques, a data flow diagram (DFD) is used for process modeling. This modeling technique shows the flow and transformation of the data without regard to the details of the data structure or type. It clearly represents where the transactions and transformations occur in your system.

A DFD is not the same as a flow chart, although there are certain similarities. A flow chart is much less specific with regard to how pieces of data are broken down, combined, and moved around the system than is a DFD. On the other hand, a flow chart is much more specific and physical than a DFD with regard to how processing is performed. A data flow diagram is more flexible and has a more general applicability than does a flow chart.

Data flow diagramming is not designed to show materials flow, just data. For example, if you were modeling a bookstore, how all of the receipts, invoices, inventory counts and financial transaction items are handled would be shown on your diagrams; but the books themselves would not. The books are materials, and their movement from the publisher to the store‘s loading dock to the shelves to the bag in the customer‘s hand is materials flow and not a part of data flow diagramming.

In any structured analysis methodology, the first task is to draw a top-level diagram, a simple summary of the overall system. It shows the system environment and major inputs and outputs, and is sometimes referred to as the basic problem statement. This is usually much less specific than the way most people picture a system because so many details are omitted. It should involve only one, two, or three processes and a very few external entities (source/sinks). In the example that follows, you use only one process and two source/sinks, though a top-level diagram could contain a few more of each. You break down (decompose) these top-level elements into more specific processes and flows. Some methodologies and analysts like to use a single process to represent the highest level of the data flow diagram. This is called a context diagram, and only one process is allowed on a context diagram to designate the entire system. For child diagrams, though, you can have multiple processes on any diagram.

The idea behind creating a general top-level diagram is twofold:

- To ensure agreement and understanding of the fundamental, overall mission of the system. There is confusion on this more often than is realized, and the details can rarely work well if the overall mission is unclear.
- To make explicit the source and derivation of the more detailed operations of the system. Often it is the second or third level of design that is the taken-for-granted starting point. Making the derivation explicit is important both for the design discipline itself and for the completeness of the resulting documentation. If you start Visible Analyst at the highest level, the tracking of all subsequent derivations automatically results from the data repository documentation.

There are four meaningful objects that appear on data flow diagrams:

Process

The process symbol is accessed with the first symbol button on the control bar. If you are using Yourdon rules, a process is represented by a circle. For Gane & Sarson rules, a process is represented by a rounded-corner square. For SSADM and Métrica rules, a process is represented by a square.

A process signifies that something is happening to transform data. At the highest level you could show the whole bookstore as a single process.

After creating the context (or high-level) diagram, you then break that diagram down into processes representing the various departments of the store, then into processes representing the functions of the departments, then into subdivisions of these processes, and so forth to as fine a level of description as you wish. This is done by ―nesting‖ or decomposing a process and creating a child diagram at a greater level of detail, one that shows all of the inputs and outputs to the parent process and allows you to show what is going on inside it. Processes have numbers, and those numbers reflect the decomposition hierarchy, as shown in Figure 8-1.

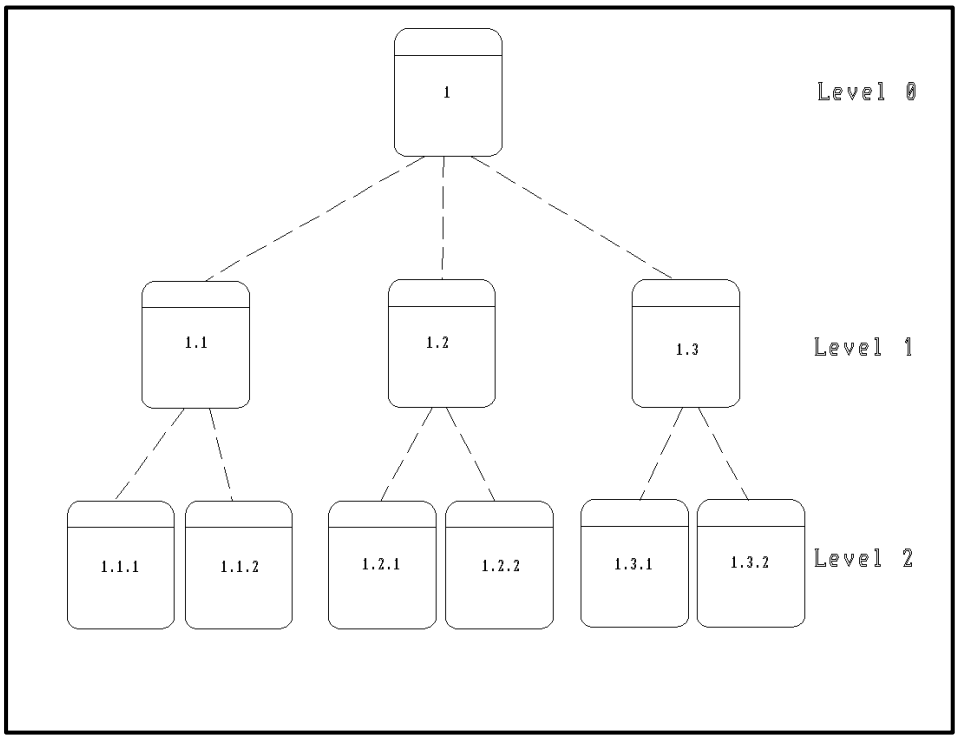


Figure 8-1 A Process Numbering Scheme

Data Store (or File)

A data store or file is accessed using the second symbol button on the control bar. If you are using Yourdon rules, a file is represented by two horizontal parallel lines. For Gane & Sarson , SSADM and Métrica rules, a data store is shown as a rectangle with the right side open, and it has a number. A data store is a place where data is kept while it is not actively being processed. Your process model does not show how it is stored, whether encoded on magnetic disk or scribbled on the back of an envelope, just that it is stored. Data can only enter a data store from a process and can only leave a data store to a process. The detailed data element in a data store can be defined in the Visible Analyst repository.

External Entity (or Source/Sink)

An external entity is accessed using the last symbol button on the control bar. It is represented by a large square under Yourdon rules, or a square drawn with relief under Gane & Sarson rules. For SSADM and Métrica rules, an oval represents an external entity. An external entity is something outside the boundary of the system you are modeling that either sends data to your system or receives data from it. It is effectively a black box, in that what happens inside the external entity is not material to your system description. It is only there to make clear some of the environment in which your system resides. External entities are optional. A net input data flow can just as well be shown coming from nowhere as from an external entity. Note that an external entity has no relation to the entity that is a part of entity relationship modeling. It is simply an unfortunate duplication of terminology.

Data Flow

A data flow depicts the movement of one to many items of data. Data can enter a system from outside, such as the entries that appear on a publisher‘s invoice or a packing list. (The invoice data flow is shown entering a process—it *must* enter a process—where it is examined and acted upon.) This process might send some of the data to be stored, some to be printed, some to be ignored. These invoice data elements may or may not be combined with elements from other input data flows and may then exit the process as parts of other data flows. To draw a data flow line, click on a line type in the control bar.

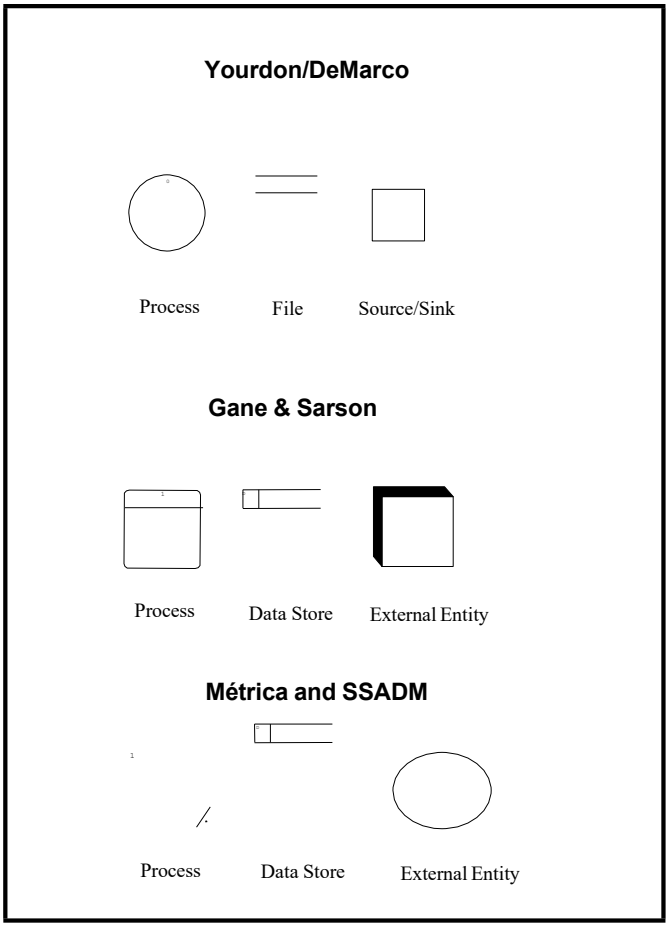


Figure 8-2 Data Flow Diagramming Symbols

Note

In Yourdon methodology, names of data flows contain hyphens instead of spaces. When you enter a space in a data flow name, Visible Analyst uses a hyphen.

This lesson leads you through the diagram creation process for a Gane & Sarson-based process model. Basic drawing and decomposing a process into a subordinate ―child‖ diagram

are shown. Also, you see how the system is validated using the rules capabilities of Visible Analyst. You build errors into your diagram to demonstrate the types of errors that can be identified by the Analyze function.

CREATING AND POPULATING A TOP-LEVEL DIAGRAM

The basic procedure for creating a top-level DFD is the same as creating a new diagram for the unstructured diagram type. The only difference is that if you choose a context diagram, by clicking the box at the bottom of the New Diagram screen, a process symbol number 0 is automatically placed on the diagram; and you are prompted for its name. A context diagram is permitted only one process symbol. You can add data flows and other symbols to the diagram.

This diagram has already been created for you so that you do not have to draw the diagrams and enter repository information. It is named DMV System and is shown in Figure 8-9, the top-level diagram of the DFDs you spawned from your FDD. This diagram also has one child diagram called Driver‘s Licensing System. You can display a list of diagrams by selecting Open Diagram from the File menu, or by clicking the Open button on the control bar. When a diagram type has a plus sign next to it, it means that diagrams of that type have been created. Click the plus sign to display the list of existing diagrams, and then double-click the diagram you would like to open. (You can hide the list again by clicking the minus sign next to the diagram type name.)

To close a diagram, click the control button in the top left corner of the diagram window and select Close, double-click the control button, or choose Close Diagram from the File menu.

NESTING A PROCESS

In this unit, you structurally decompose a process symbol. This is also called ―nesting‖ or ―exploding‖ a process. The File menu contains the Nest function for this purpose; the submenu contains the Explode function. Explode can also be found by clicking the *right* mouse button on a process symbol that you want to model in more detail to display its Object menu, and then selecting Explode.

If the process has not previously been decomposed, this generates a ―child‖ diagram from this ―parent‖ process. All of the data flows attached to the parent process are automatically ―dragged down‖ to the child diagram by the Nest function. These flows can be attached to the lower-level processes that you create on the child diagram. Those lower-level processes can then be nested further to increase the level of detail. In the current example, the child diagram was created by the Spawn function that you executed in Lesson 6 – Functional Decomposition Diagrams, and the processes you added to the FDD were placed on it.

Open the Diagram: 1 From the File menu or the open diagram button on the

control bar, open the data flow diagram DMV System, if it is not still open from a previous lesson. This is the context diagram for this project.

Select a Process: 2 Click the right mouse button on the process Driver‘s Licensing System to open its Object menu, and choose Explode. This opens the existing child diagram Driver‘s Licensing System and is an alternate way to navigate between the diagrams of your project, avoiding the File menu.

Explode It: 3 Click the *right* mouse button on the process labeled Issue License and choose Explode. The flows attached to the parent process are dragged down to the spawn-created diagram entitled Issue License, where the three process symbols from the functional decomposition diagram were placed. Maximize the diagram. The dragged-down flows are lined up on the sides of the child diagram, input flows on the left, output flows on the right. (See Figure 8-3.) Since you did not move the symbols on the diagram before nesting, it is possible that the dragged-down flows were drawn over a symbol.

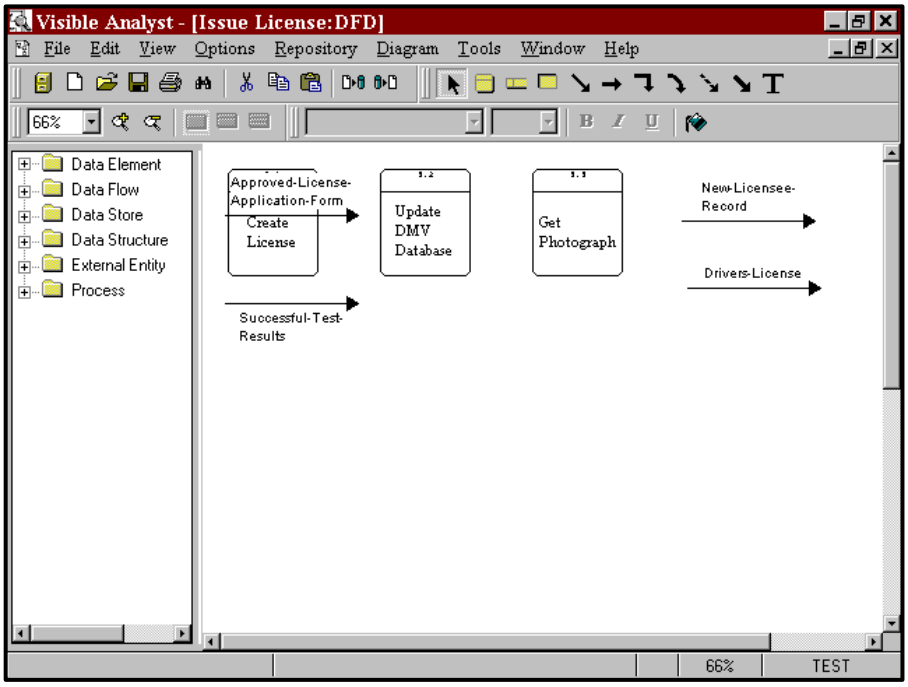


Figure 8-3 Child Diagram With Dragged-Down Flows

Edit the Diagram: 4 Move the symbols and attach the flows as shown in Figure 8-4. To move a symbol, click and drag it with the *left* mouse button. To attach the lines, click one endpoint. Then click the *left* mouse button on the middle of the data flow and drag the line so that it is positioned correctly. When the data flow is selected, it changes color; and the line becomes a dashed line as it is moved on the diagram. Do the same for the other data flows. (Ignore for now the other flows you see in Figure 8-4; you add them later in this section.)

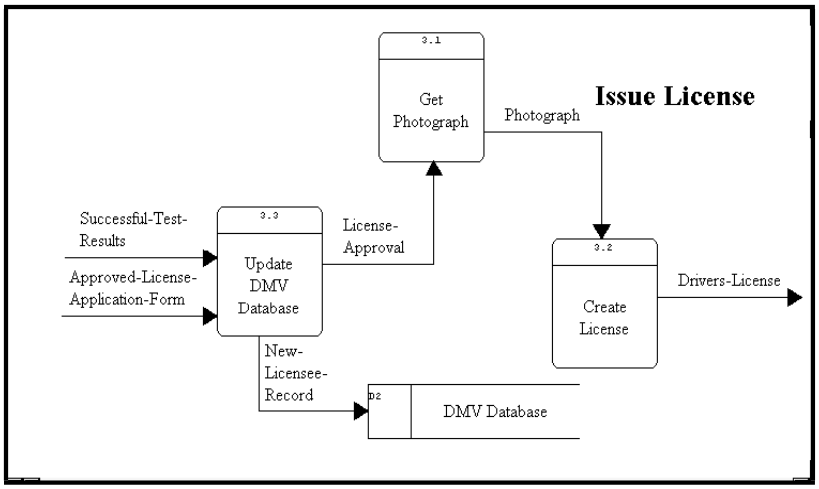


Figure 8-4 The Completed Diagram

Note

When you want to show a data flow line (or another type of line) as attached to a process (or another type of symbol), you must drag the end of the line into contact with the symbol. With the Auto Connect option on, Visible Analyst redraws your connection at the outer edge of the symbol.

Add Flows and Text: 5 Add two new data flows: License-Approval and Photograph. Since these are internal flows, as opposed to net input or net output data flows[7] appearing on the parent, the Nest function could not create them.

Change Line Format: 6 You may want to change the data flow lines from straight to elbow. This can be done by highlighting the line and then selecting Line Settings from the Options menu and changing the line orientation to elbow. Or click the elbow line button on the control bar before drawing the lines.

To change the orientation of an elbow, position the cursor over the line segment handle to change and click, but do not release, the left mouse button. Move the mouse slightly until the line changes from solid to dashed, and

[7]For a full explanation of net input and output flows, please see the Visible Analyst *Operation Manual* or the online help.

		then press the right mouse button. Release the left mouse button to save the change.
	7	Click the T button on the control bar to add the caption text ―Issue License‖ to display the diagram title on the diagram. Note that there is a way to do this automatically by using boilerplates. You can read about this in the Visible Analyst *Operation Manual* or in the online help system. (Boilerplates are not available in the Education Editions of Visible Analyst.)
Add a File:	8	If you wish, you can add the file DMV Database to the diagram. Since it appears on the context diagram, this is not necessary, but some people feel that showing it on a lower-level diagram adds clarity. Move the flow New Licensee-Record to attach it to the file DMV Database. A symbol is considered attached to a line when the endpoint of the line is touching the edge of the symbol. (It does not automatically connect to the symbol.)
Save:	9	Select Save from the File menu.

CREATING A NEW DIAGRAM

In previous sections of this lesson, you worked with diagrams that were either supplied by Visible Systems or created by the Spawn function. In this unit, you create and populate a new diagram yourself and practice more data flow diagramming techniques. You structurally decompose the process Administer Road Test (drawing a level-three diagram) that details what occurs within the process Administer Road Test.

If you have any other diagrams open, you should maximize the window by clicking the Maximize button in the upper right corner of the window.

Set the Zoom Level:	1	Set the zoom level to 66% from the View menu.
Open the Parent Diagram:	2	Select Nest from the File menu.
	3	Select Parent from the submenu. You move up the diagram tree to display the diagram Driver‘s Licensing System.
Nest a Process:	4	With the *left* mouse button, select the process symbol

Administer Road Test by clicking on it. It is highlighted as the current object.

5 From the File menu, select Nest and then Explode.

6 Choose Create New Diagram. If you had previously nested this process, the child diagram would have displayed automatically. This option is useful to drag down new data flows that you may have drawn on the parent diagram to child diagrams after the child diagram has been created. A new diagram is drawn with your input flows in the upper left corner of the diagram, and the output flows in the upper right corner. If you cannot see the flows, select 33% zoom from the View menu and your diagram, shown in Figure 8-5, scales down so that you can see more of it.

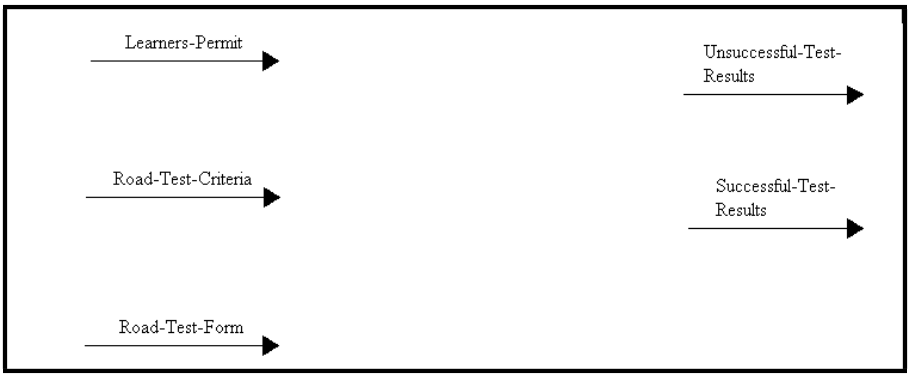


Figure 8-5 Exploded Diagram with Flows

Save:

7 Select Save from the File menu and click OK. The title of your diagram defaults to the name of the parent process. Visible Analyst indicates that it is saving *both* diagrams. This is because they are involved in a nest relationship and both ends of the nest relationship must be saved in the repository.

Adding Processes to a Child Diagram

Now add processes to the child diagram named Administer Road Test. The processes contained in this diagram are the individual processes that make up the parent process Administer Road Test. This diagram is a more detailed representation of the transformations and interactions that occur to the data flows within the parent process.

Add Processes:	1	Click the first symbol button, process, in the control bar.
	2	Add and label three processes: Validate Applicant, Test Vehicle Knowledge, and Test Driving Capabilities.
Save:	3	Select Save from the File menu.

Attaching Data Flows to Symbols

The input data flows on the left side of the diagram and the output data flows on the right side of the diagram were dragged down to the child diagram with the Nest function. It is necessary to attach the data flows to the appropriate processes on the child diagram. To attach a flow to a symbol:

Select a Line:	1	Put the cursor in selection mode by clicking the ↖ button on the control bar.
	2	Select the data flow Learners-Permit. The line handles appear.
Drag It Into Position:	3	Drag it to the edge of the process symbol labeled Validate Applicant, as shown in Figure 8-6.
Repeat for Other Flows:	4	Attach the other dragged-down flows as shown in Figure 8-6.
Add New Flows:	5	Click the first line button on the control bar.
	6	Add a flow from process Validate Applicant to Test Vehicle Knowledge and label it ―Valid-Applicant.‖
	7	Click the straight line button on the control bar and add an input flow into the process Test Driving Capabilities and label it ―Test-Criteria.‖
	8	Add a flow from process Test Vehicle Knowledge to process Test Driving Capabilities, but leave it unlabeled

by clicking Cancel or pressing ESC when you are prompted to enter a name. (This deliberate error is added to demonstrate the capabilities of the Analyze function.)

9 Add an unattached data flow labeled —Driving-Criteria.‖ (Remember, you must double-click to end the line when it is not attached to a symbol.) This demonstrates the ability to select an existing flow from the diagram when a flow is split.

Save: 10 From the File menu select Save.

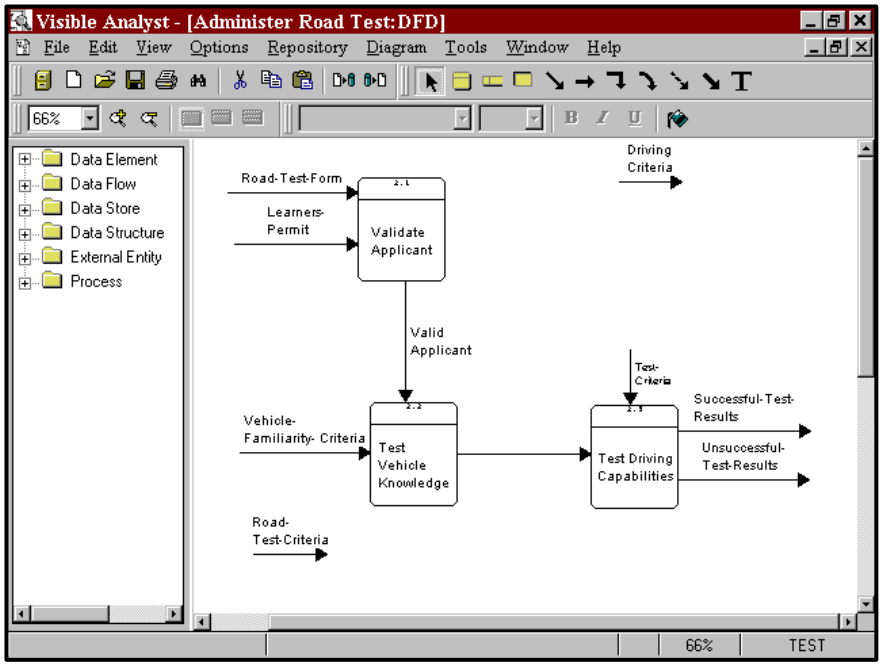


Figure 8-6 Child Diagram with Processes and Flows

Splitting Data Flows

Decomposing, or ―splitting,‖ a data flow offers the capability to divide net input or net output data flows into subflows, creating more detailed representation on lower-level diagrams. The concept is illustrated in Figure 8-7. This capability greatly aids in the system analysis process by showing more complex data flows at high levels of the structured specification and smaller or even atomic data flows at the lower levels of the structured specification. This decomposition capability provides a better understanding of the entire system and its parts. Whenever a dragged-down data flow is split into subflows, the original flow is erased from the current diagram and replaced by the selected or created subflows.

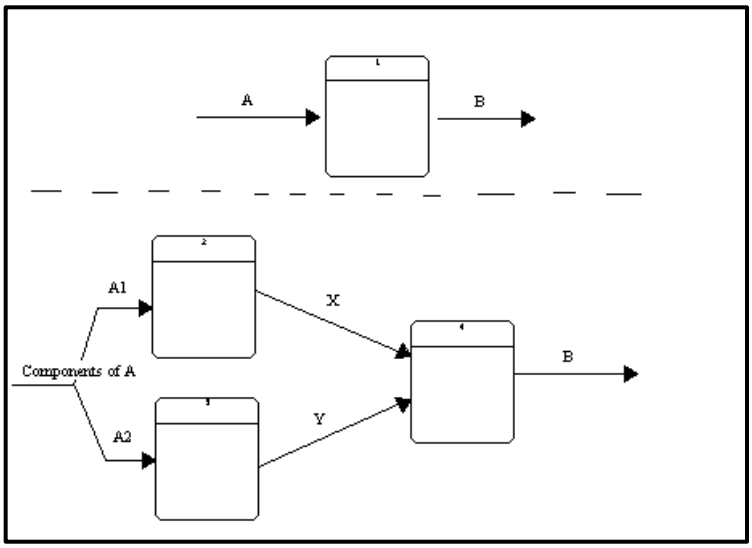


Figure 8-7 Splitting a Data Flow

Select a Flow to Split:	1	Click the ↖ button on the control bar to put the cursor into selection mode.
	2	Display the Object menu for the data flow labeled Road-Test-Criteria by clicking on one end of it with the *right* mouse button.
Start the Split:	3	Choose Split Data Flow.
	4	In the box labeled Enter Subflows, type in ―Vehicle-Familiarity-Criteria.‖ This option draws a new flow (a subflow of Road-Test-Criteria) on the diagram with this

label. See Figure 8-8. If you want to add more than one name in this box, press ENTER to place the cursor on a new line.

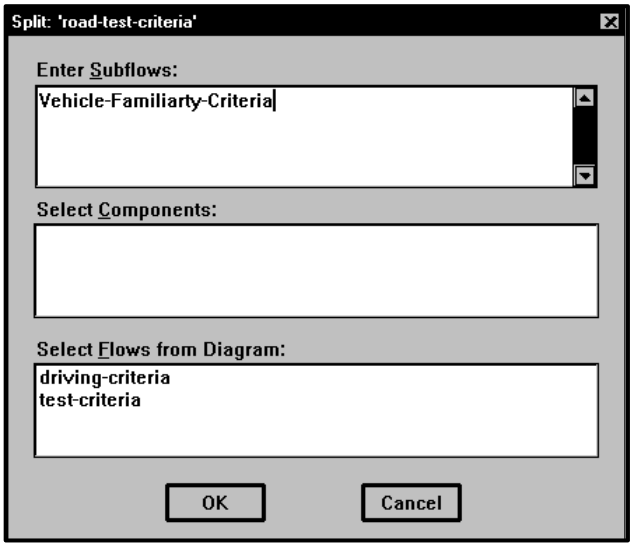


Figure 8-8 Split Data Flow Dialog Box

5 In the box marked Select Flows from Diagram, click Driving-Criteria. This option allows you to select an existing flow to be a subflow of Road-Test-Criteria.

6 Click OK. Notice that the flow Road-Test-Criteria is no longer on the diagram and that the flow Vehicle-Familiarity-Criteria has been added to the diagram.

7 Attach the flow Driving-Criteria to the process Test Driving Capabilities with the arrow pointing *away* from the process symbol. (This error is made deliberately; it is explained during the discussion of Analyze.)

8 Ignore the data flow Vehicle-Familiarity-Criteria, as another test for Analyze.

ANALYZING FOR BALANCE AND COMPLETENESS

As a project goes through a number of nested decompositions (nests), data flow splits, various object moves and other edit procedures, there is a significant possibility that various data flows are incorrectly used, or that objects are forgotten, etc. For a large project with many symbols and flows, this is a real probability; and the errors are not easily detected by visually checking the diagrams yourself. The Analyze function, found on the Diagram menu, is designed to warn you of completeness and logic errors. The function checks diagrams for:

- Labels on all objects.
- Unattached objects.
- At least one input flow and one output flow for each process.
- Data flow balance, which implies that an input flow is used everywhere as an input flow rather than an output flow and that data flows are properly accounted for at all levels of the diagram hierarchy.

The diagram is now analyzed for adherence to the rules of the Gane & Sarson methodology. Those rules are outlined in the Visible Analyst *Operation Manual* and in the online help.

Analyze the Diagram:

1 Select Analyze from the Diagram menu.

2 Select Current Diagram and click OK.

Visible Analyst displays the errors found. To display the errors full screen, click the Maximize button in the upper right corner of the error window. If an error message extends beyond the box, use the scroll bar at the bottom of the box to scroll the text to the left. There should be five messages.

Data Flow labeled 'Vehicle-Familiarity-Criteria' is dangling.	This indicates that Vehicle-Familiarity-Criteria is not attached to a process.
There are 1 unnamed Data Flow(s).	This is the data flow that you left unlabeled on the diagram.
Net input Data Flow 'Test-Criteria' is not shown attached to parent Process.	This indicates that the data flow Test-Criteria has been added to the child diagram but is not accounted for on the parent diagram.
'Driving-Criteria' be shown as a net Flow.	This indicates that Driving-Criteria is being used as a net output flow on the diagram, while it is used as a net input flow on the parent. *should input Data*

Input Data Flow 'Road-Test-Criteria' on parent is not shown.	This message is a result of the fact that Vehicle-Familiarity-Criteria, a child flow of Road-Test-Criteria is not *attached* to a process as a net input flow, even though it appears on the diagram.

Note

- Analysis error dialog boxes allow you to keep them on the screen while you carry on various Visible Analyst activities. This is to make it easier for you to correct the errors found by Analyze. If you don't want to keep the box open, press ESC or click Cancel to close it.

Fixing the Errors

Correct the Data Flows:	3	Attach the data flow Vehicle-Familiarity-Criteria to the process Test Vehicle Knowledge, as shown in Figure 8-9.
	4	Reverse the direction of Driving-Criteria, so that it becomes an input flow to Test Driving Capabilities (see Figure 8-9), by dragging the endpoints.
	5	Delete Test-Criteria by clicking on the line and pressing the DELETE key.
	6	Label the unlabeled data flow ―Vehicle-Knowledge‖ by clicking on the line with the *right* mouse button and selecting Change Item from the Object menu. Then enter the label and click OK
Analyze Again:	7	Select Analyze from the Diagram menu again.
	8	Choose Current Diagram and click OK. The diagram should now be correct.

Note

- It is unnecessary to save a diagram after Analyze has been performed because Visible Analyst automatically saves it for you before analysis begins.

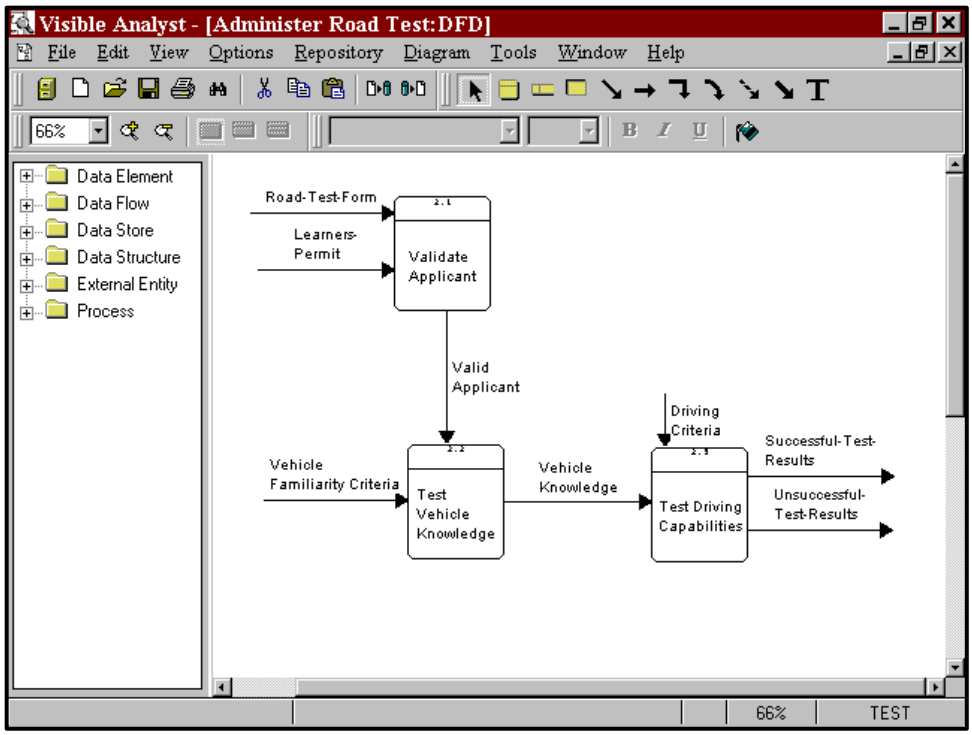


Figure 8-9 The Completed Diagram

GENERATING A PROCESS DECOMPOSITION MODEL

A process decomposition model for a process shows you the hierarchical structure of a decomposed process that has been nested.

Note

A *process* decomposition diagram is very different from a *functional* decomposition diagram. The former is simply an unstructured diagram displaying the hierarchy of processes that are descendants of an indicated process. The latter, discussed in Lesson 4 -Functional Decomposition Diagrams, is a full diagramming methodology for performing business planning.

Open the Diagram: 1 From the Window menu, click —DMV System: DFD.‖

Select a Process: 2 Click the process Driver‘s Licensing System with the

right mouse button.

Create the Decomposition:	3	Select Decompose. An unstructured diagram is generated showing the hierarchical structure of the process.
Save the New Diagram:	4	Select Save from the File menu. Label the diagram —Process Decomposition‖ and click OK.

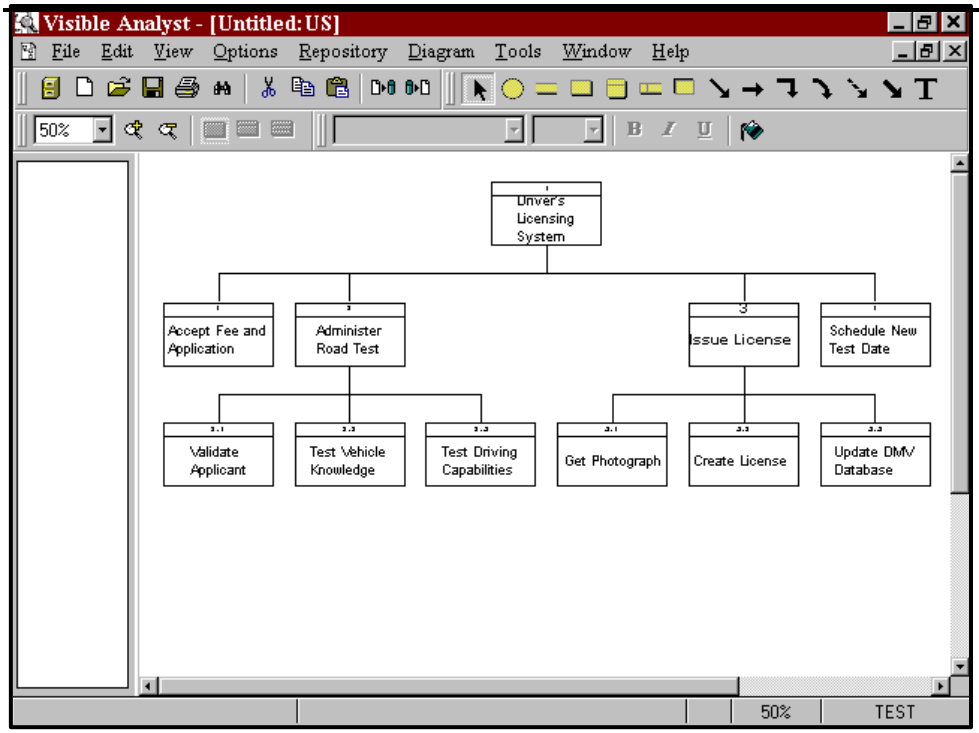


Figure 8-10 Process Decomposition Diagram

Lesson
Working with Repository Functions

OVERVIEW

This unit helps familiarize you with the operation of the Visible Analyst repository and shows you the power of an online, interactive database for systems analysis, design and data modeling. The TEST project used in the previous lessons is used as the basis for your exercises.

The repository is a powerful tool for creating and managing the narrative portions of a system‘s specification. A project repository is used to provide an entry location for all project documentation. Each graphical entry on your diagrams has an automatically created corresponding entry in the project repository, as do any items entered into a Composition or Alias field.

You have the ability to thoroughly define all of your graphical entries in the repository or to simply enter notes about them in the Notes field. As an integrated part of Visible Analyst, the repository operates in parallel with the diagramming functions to accomplish data decomposition logically. It contains powerful data management, text editing, import/export, and report facilities. By using it, meaning can be ascribed to diagrams and an asset of ever-increasing value can be created. After defining items, changing entries and entering notes, you can generate reports from this information in many different forms.

When you finish defining your data and processing, the repository also allows you to put it into an ASCII file and export it. The ASCII file can then be sorted to move data specifications to your database and process specifications to your text editor for writing code. (The Shell Code Generation utility can also be used for this purpose.)

Note

- Users of the Educational and Demonstration versions of Visible Analyst cannot add items directly into the repository. First add the object to the diagram, and then edit it into the repository.

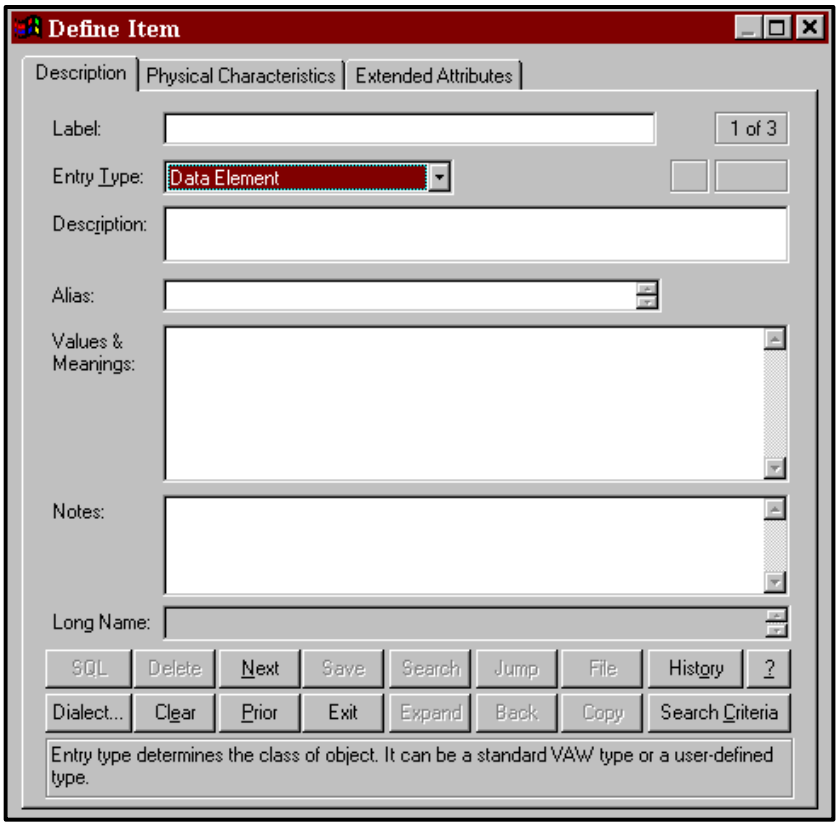


Figure 17-1 Blank Repository Dialog Box, Page One

REPOSITORY BASICS

Repository Control Buttons

The repository control buttons (see Figure 17-2) are always displayed at the bottom of the repository dialog box. Each of the button functions is accessed by clicking on the button or, as is customary in Windows, by using its keyboard shortcut by holding down the ALT key and pressing the underlined letter to execute the button function. Only the functions available to you at a given time are active; the others are grayed. The button functions are:

Figure 17-2 Repository Dialog Box Control Buttons

SQL	This button opens the Generated SQL for View dialog box. This dialog box displays the SQL generated for the current view object based on the view table and column specifications selected when creating the view, as well as the current SQL dialect. This button is active only when the entry type is View.
Dialect	This activates the RDBMS SQL Dialect dialog box. From there, you can change the current SQL dialect.
Delete	This deletes the current repository entry from the database. An entry can only be deleted when it has no location references, meaning that it does not appear on a diagram nor as an attribute of another repository item.
Clear	This clears the display of an entry and displays a blank repository dialog box. This allows you to Search for an existing entry or add a new entry. If you have made changes, you are prompted to save them before clearing. Your current location in the repository remains unchanged.
Next	This displays the next sequential repository entry that meets the repository search criteria (see below).
Prior	This displays the previous sequential repository entry that meets the repository search criteria.
Save	This saves all changes made to an entry.
Exit	This or the ESC key exits from the repository.
Search	This initiates a search for a particular entry in the repository. The procedure is explained in the section on Search Capabilities.
Expand *Contract*	This allows you to expand or contract the display size of some fields. The fields that normally display four lines can expand to display 15.

Jump — This allows you to jump immediately to another entry that is referred to in the current one. This feature is described in the section on Navigation Capabilities.

Back — This button provides a means of jumping to the previous repository entry. You can then continue to move backwards displaying previous repository entries.

File — This allows you to insert text from a DOS file at the cursor position or to copy highlighted text to a DOS file. It is explained in further detail in the Visible Analyst *Operation Manual* and in the online help system.

Copy — This button provides a means of copying the current object.

History — This provides a means of jumping back to a previously displayed repository object. A list is kept of every object definition that has been displayed. If you choose this button, the History dialog box appears and you can jump between entries by double-clicking on an entry. The maximum is 500 objects.

Help(?) — This displays context sensitive help about the repository. You can also press F1 to activate the help system.

Search Criteria — This allows you to specify how the repository is to be searched. It is explained in the section on Search Capabilities.

Other buttons that may be displayed on the Define dialog box are:

Primary Key — If the current object being examined is an entity type, the primary key button is displayed to the left of the Composition/Attributes field.

Attributes Details — This button provides a means of populating the composition of a repository entry with components and physical information. This button is displayed to the left of the Composition/Attributes field.

When the Entry Type is a Class, or when the Classic User Interface is turned off, the Add button displayed beneath the Attributes Details button is active. You can use this button to add details. When you begin typing in the field next to the Add button, the

button is enabled. Click the Add button to add the attributes to the Attributes field.

Editing Keys

Because the Edit menu is not accessible from the repository, you can use the right-click menu that is available when an object (text) is highlighted and you click the right mouse button. Using the right-click menu, you can Cut, Copy, Paste, or Delete the selected object.

Field Types

The data repository of a Visible Analyst project is displayed using Define dialog box variations corresponding to different diagram objects. You see and work with some of these variations during the course of this lesson. The basic dialog box, shown in Figure 17-1, is for data elements, aliases, miscellaneous objects and external entities or source/sinks. Other objects, such as data stores, processes, functions, entities, relationships, modules, data flows information clusters, etc., have variations in individual fields and tabs of the Repository dialog box to accommodate the specific needs of those items. Some of these differences are seen later in the lesson.

Label Field

This is the name of the repository item. The names of items drawn on diagrams are automatically entered here.

Entry Type Field

This tells Visible Analyst what kind of object the item is: process, data flow, entity, etc. The entry type can be entered manually, or you can select the type from the scroll box accessed by clicking the down arrow at the end of the entry type field.

Note

- You can edit the Entry Type and Label fields of data elements and data structures that do not appear on diagrams. The entry type for a data element cannot be changed if physical information for that element has been entered.

Description Field

The Description field is a two-line field that provides a convenient place to enter a somewhat more extensive descriptive title of the object than the Label field allows. The contents of this field are used for the Comment on Column (data elements) and Comment on Table (entities) when SQL DDL is generated if the selected SQL dialect supports this syntax.

Alias Field

The Alias field contains 10 lines of 128 characters each. It allows for the entry of alternative labels to the one used as the object label. This is most commonly used for indicating the cryptic abbreviations that are sometimes used in the actual coding of a software program, as opposed to the plain English names that are desirable for reference. The Alias field is an intelligent field. Data names entered into it establish new repository entries for these aliases.

Attributes Field

The purpose of the Attributes field is to accumulate the collection of data elements that you wish to define as constituting a data flow, entity, data store, etc. The Attributes field is an intelligent field. Data names entered into it establish new data element repository entries or update existing ones. These new data elements can then be used for further definition. Data flows, data structures and couples can also appear in some Attributes fields.

When you click the Attributes Details button, the Add Attributes dialog box appears. Using this dialog box, you can define up to 12 components and some of their properties. As you enter items, the dialog box automatically scrolls as necessary to allow you to enter more items until you reach 12. When you complete the entries, click OK to add them to the Attributes field. If you need to add more than 12 components, click the Attributes Details button again; and a new dialog box opens so that you can add additional attributes.

Use the Add button at the bottom of the Attributes field to add components one at a time. When you begin typing in the field next to it, the Add button becomes active. Complete your entry, and then click Add to enter the component in the Attributes field.

Values & Meanings Field

The Values & Meanings field allows an unlimited number of lines. The maximum number of characters that can be contained in the field is 64K. This field allows the entry of specific information about the value(s) the item can take.

Discriminator Values & Meanings Field

If the current object is a data element that is used as a discriminator, this field contains a list of values to identify the subtype entities. For each subtype, a value can be entered that will uniquely identify it. By default, these values are numbers starting with 0 for the supertype. To change the value, click the value until an edit control appears, make your changes, then press ENTER.

Notes Field

The Notes field is also a field that allows you to enter any pertinent information about the object. The maximum number of characters that can be contained in the field is 64K. This is the logical field to use when creating hyperlinks to external documents, web pages or other application files.

Location Field

This field displays two types of usage information. The field can contain the diagram name (and, for DFDs, the diagram number) of every diagram where the item appears. The field can also tell you if the item appears as an attribute of another item. This second kind of location entry has the entry type of the parent item, followed by an arrow and the name of the parent item.

Other Pages and Fields

Other pages of the Define dialog box contain additional information. For example, pages 2 and 3 of the basic repository form provide location and relationship information and specifications for PowerBuilder/VISION extended attributes. These two pages are similar for most entry types. For some entry types, additional pages will be displayed:

- When the entry type is an entity, the next five pages contain keys, foreign keys, triggers, check constraints, and physical information.
- For views, the next five pages provide table, column, join, clause, and option information.
- When the entry type is a relationship, there are additional pages that contain foreign key and cardinality information.
- When the entry type is a tablespace, an additional page contains property information.

A full list and complete descriptions of pages and fields can be found in the *Operation Manual* and in the online help.

Object Repository

The Visible Analyst repository provides several additional forms and data input components for supporting the object-oriented concepts. The object repository components are detailed below.

Attributes

The Attributes field replaces the Values & Meanings field whenever the Repository dialog box displays a class. The field contains a list of the data members for the class showing the local data element and type. To add, change, or remove local data elements, click the Attributes Details button or select Add/Change from the Repository Object menu. For each attribute, the following information can be defined:

- **Name**. The name of the attribute. Each attribute of a class has a separate entry in the repository with a type of local data element. This is an optional field. The search button can be used to find other local data elements in the repository.
- **Type**. The attribute type can be a class, data element, or data structure. If the type does not exist in the repository, a new class is created. The location field of the attribute type contains a reference to the current class. This is a mandatory field. The Search button can be used to display a list of valid types. If the attribute type is a data element or elemental

class, its physical characteristics are displayed. Entries added to the Type field are saved as data elements for an entity or data flow, and class/subtype element when the object is a class.

- **Limit**. The number of occurrences of the attribute. If this field is blank, the attribute occurs once.
- **Reference**. A qualifier to indicate the access method for an attribute. *Value* indicates the object defined in the *Type* field is used; *Address* indicates a pointer to the object is to be used; and *Reference* indicates a reference to the object is to be used. The default is Value.
- **Visibility.** *Public* members have global visibility. *Private* members are only accessible to member functions and friends. *Protected* members are accessible to derived classes and friends. *Implementation* members are only accessible to the class itself. The default is Private.
- **Qualification**. *Constant* indicates a member's value cannot be changed. *Volatile* indicates the member can be modified by something other than the program, either the operating system or hardware. *Static* indicates there is only one instance of the member regardless of the number of times a class is instantiated. The default is None.
- **Physical Characteristics**. If the attribute type is elemental, the physical characteristics can be set.

For every item entered into the Type field, Visible Analyst creates a repository entry (if one with the same name does not already exist) and updates that entry's location field. If an item is removed, this field is updated to reflect this. These repository entries are generally created as classes unless a data element already exists with the same name or the physical characteristics are defined

As you enter items, the dialog box automatically scrolls as necessary to allow you to enter more items until you have finished. Insert is used to insert a new attribute into the list at the current position, while Delete removes the current attribute (the current position is indicated by ➢➢). When you have completed the entries, click OK to add them to the Attributes field.

Item names entered into this field may contain up to 128 characters each and may consist of any upper or lower case letters, numbers, spaces, periods, underscore characters and hyphens; but the first character must always be a letter.

Attached Entities/Classes

The attached entities/classes for the currently displayed relationship are listed in this field. When an inheritance relationship is displayed, the characteristics of that relationship can be changed (see changing Inheritance Characteristics later in this chapter). Otherwise, the information cannot be edited from within the repository; and all changes must be made on a diagram. The field lists the two entities or classes attached to this relationship. Below the second entity name is listed the reverse of the current relationship. If either direction of the relationship has not been named, the name of the relationship in the reverse direction is

displayed as —reverse of (opposite relationship name).‖ This field allows you to jump to the repository entries for any of these entities or relationships, as described above.

Relations

For an entity or class, the Relations field displays the relationship name followed by the name of the entity or class on the other end of this relationship for each relationship attached to this entry. These sets are ordered alphabetically by the opposite entry name. When an inheritance relationship is displayed, the characteristics of that relationship can be changed (see Changing Inheritance Characteristics later in this chapter); otherwise, the information cannot be edited from within the repository; and all changes must be made on a diagram.

This field allows you to jump to the repository entries for any of these entities, classes, or relationships by positioning the cursor on the line containing an entity, class, or relationship name and clicking the Jump button.

Long Name

When a repository entry, either a local data element or a module, belongs to a class, the full name of the entry includes the class name. The Long Name field displays this name and, in the case of modules, includes the argument list (the argument list is required to differentiate overloaded member functions). If you want to change the argument list for a class method, click the right mouse button on the Long Name field and select Change (see the Methods section later in this chapter for details). If you want to change the class to which the method belongs, select Class from the Repository Object menu. To display the class definition, click the Jump button.

Class Characteristics

Concurrency, displayed on the Methods/Friends tab, is a class property that distinguishes an active object from inactive object. An *active* object may represent a separate thread of control. A *sequential* object is a passive object whose semantics are guaranteed only in the presence of a single thread of control. A *guarded* object is a passive object whose semantics are guaranteed in the presence of multiple threads of control.

A persistent class exists beyond the lifetime of an executable program. This means it must be stored on a non-transitory storage device. If the subtype of a class is set to either entity (associative or attributive) and the class is used on an entity relationship diagram, this field cannot be changed.

An abstract (or virtual) class cannot be instantiated because it contains pure virtual methods. If pure virtual methods exist for a class, Abstract is checked. If you attempt to uncheck this field, all pure virtual methods are reset to virtual. If you attempt to check it and virtual methods exist, they are converted to pure virtual methods.

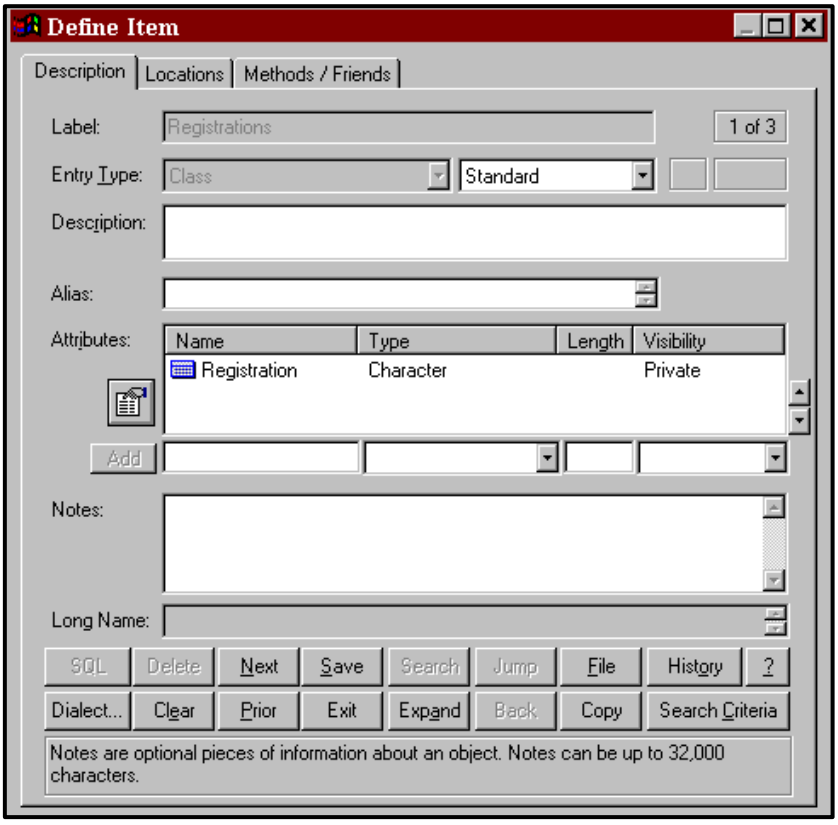


Figure 17-3 Class Attributes

Methods

Methods (or Member Functions) are the operations that are defined for accessing a class. The Methods field contains a list of the functions for a class showing the name, return value, argument list, and flags to indicate its visibility. To add, change, or remove methods, click on the Methods field and click the Attributes Details button or select Add/Change from the Repository Object menu. To add a new method for a class, click the New button and type the name of method you wish to add. To search for methods that have already been defined in the repository, click the Search button. The list contains all modules that have previously been defined in the repository. If the module already belongs to a class, the class name is displayed. Note that when you select a module that already exists, the complete definition for that module is used including return value and argument list. Click OK to add the method name to the list of methods for the current class. For each method, the following information can be defined:

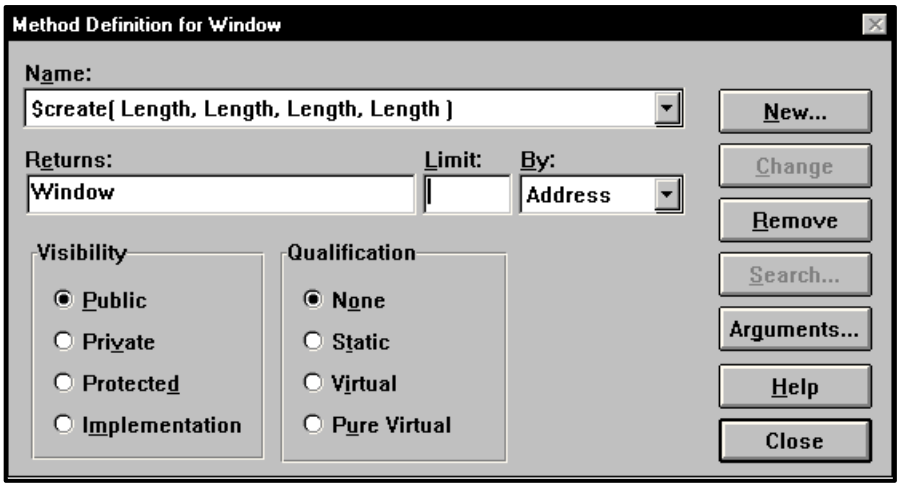


Figure 17-4 Class Methods

- **Returns**. The return type can be a class or data element. If the type does not exist in the repository, a new class is created. The Location field of the attribute type contains a reference to the method. This is an optional field. Click the Search button to display a list of valid types.
- **Limit**. The number or size of the parameter. If this field is blank, it occurs once.
- **By.** A qualifier to indicate how the return value is passed. *Value* indicates a copy of the parameter is passed; *Address* indicates a pointer to the object is to be used; and *Reference* indicates a reference to an object is to be used.
- **Visibility**. *Public* methods have global visibility. *Private* methods are only accessible to other member functions within the same class and friends. *Protected* methods are accessible to derived classes and friends. *Implementation* methods are only accessible to the class itself. The default is Public.
- **Qualification**. *Static* indicates a method can be used without a specific instance of an object (it can only be used with static attributes (data members)). A *Virtual* method is one that you expect to be redefined in a derived class. A *Pure Virtual* method has no definition and must be defined in a derived class. A class with any pure virtual functions is an abstract (or virtual) class. The default is None.
- **Arguments**. A list of parameters to be used by the method. This is an optional field. If a method appears more than once with the same name within a class, it must have a different argument list for each definition. This is known as function overloading. See the next section for defining arguments.

When a method is added to a class definition, an entry of type module is created in the repository. The long name includes the class name and the argument list. The argument list is needed to differentiate between overloaded functions.

Note

Because the same name can be used for more than one method, there may be duplicate module entries in the repository, each belonging to a different class.

Arguments for Methods

When defining methods (member functions) for a class, the parameters to the function need to be specified. To add, change, or remove arguments, click the Arguments button on the Method Definition dialog box. For each argument, the following can be defined:

- **Name**. The name of the parameter. This is an optional field.
- **Type**. The parameter type can be a class or data element. If the type does not exist in the repository, a new class is created. This is a mandatory field. The Search button can be used to display a list of valid types. If the parameter type is a data element or elemental class, its physical characteristics are displayed.
- **Limit**. The number or size of the parameter. If this field is blank, it occurs once.
- **Pass By**. A qualifier to indicate the how the parameter is passed. *Value* indicates a copy of the parameter is passed; *Address* indicates a pointer to the object is to be used; and *Reference* indicates a reference to an object is to be used. The default is Value.
- **Qualification**. *Constant* indicates a parameter‘s value cannot be changed. *Volatile* indicates the parameter can be modified by something other than the program, either the operating system or hardware. The default is None.
- **Physical Characteristics**. If the parameter type is elemental, the physical characteristics can be set.

For every item entered into the Type field, Visible Analyst creates a repository entry (if one with the same name does not already exist). These repository entries are generally created as classes unless a data element already exists with the same name or the physical characteristics are defined.

As you enter items, the dialog box automatically scrolls as necessary to allow you to enter more items until you have finished. INSERT is used to insert a new parameter into the list at the current position, while the DELETE key removes the current parameter (the current position is indicated by ➢➢). When you have completed the entries, click OK to update the method name field. Item names entered into this field may contain up to 128 characters each and may consist of any upper or lower case letters, numbers, spaces, periods, underscore characters and hyphens; but the first character must always be a letter.

Friends

The Friends field displays a list of both friend classes and methods (or functions). A friend is allowed access to the private data members of a class. To add friends, click on the Friends field and click the Search button, select Add from the Repository Object menu, or double-click on the Friends field while pressing CTRL key. A list of classes and member functions is displayed in the Search list box. Locate each repository item you want to place in the Friends field and click the Search button; the item is added to the Select list box at the bottom. When you have found all of the entries you want, click the Select button and they are entered into the Friends field.

To remove a friend, highlight the desired item and press the DELETE key or select Cut or Delete from the Repository Object menu.

Navigation Capabilities

In this section, you change the displayed repository entry using Next, Prior, and Jump.

Note

The repository saves some internal settings for the duration of a Visible Analyst session. If these are set incorrectly, they may interfere with the smooth flow of this lesson. Therefore, we suggest that if you or another user worked in the repository during the current session, you should exit to Windows and restart Visible Analyst. In this way, you have a clean slate on which to run this lesson.

Open the Repository:	1	Access the repository using either Define from the Repository menu or CTRL+D. A blank Define dialog box is displayed.
Access an Entry:	2	Type ―Person Information‖ in the Label field and press ENTER twice. (Pressing ENTER once brings up the Search dialog box. Pressing it a second time displays the entry found. If you press ENTER twice quickly, you get the same result without displaying the search box.) The repository entry for Person Information displays with all of the information that has been entered into the repository for this entry.
Move Around:	3	Click Next. The next entry in alphabetical order is displayed.
	4	Click Prior. Person Information is again displayed.

Jump to Other Entries:

5 Click the element Name in the Attributes field. (It may be necessary to scroll the contents of the field to bring Name into view.) Click Jump. (Click Yes if you are asked if you want to save Person Information.) The repository entry for the data element Name is displayed.

6 Move to page two by clicking the Physical Information tab at the top of the dialog box. (The current page number is displayed in the upper right corner of the Define window.) This displays more information about the current entry, including the Location information that indicates where the current entry is used.

7 Click the line in the Location field containing Person Information. This highlights the line.

8 Click Jump. The entry for Person Information is once again displayed. The Locations tab (page 2) is currently displayed. An alternative to selecting Jump to switch to another repository entry is to double-click the entry name in the Location field or to click the Back button.

9 Move to page one by clicking the Description tab.

Search Capabilities

Searching for entries in the repository is an easy procedure. It can also be a very useful feature because you can set the Search Criteria to display only certain entry types as you move from one repository entry to the next. To search for an entry in the repository:

Access an Entry:

1 Click Clear. This clears the dialog box but does not delete the entry.

2 Type ―Road Test‖ and press ENTER twice. The repository entry for Road Test is displayed with all of the information that has been entered into the repository for this entry. (This was done for you in the samples included with the TEST project.)

3 Click Clear.

Search for the Entry:

4 Click the Search button to open the search box to select from the repository. Type —r‖ and entries that begin with —r‖ appear in the list box. If you now type an —o,‖ you see that the repository searches incrementally as you type, getting closer to the entry you want.

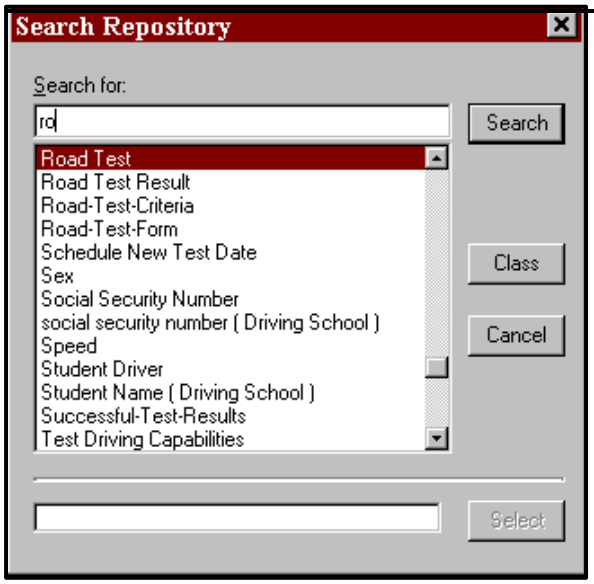


Figure 17-5 Repository Search Dialog Box

5 Click Road Test and then click Search. The repository entry for Road Test is displayed.

Setting the Search Criteria

Search criteria set the scope of the entries that are displayed as you search through the repository.

Clear the Dialog Box: 1 Click Clear to clear the dialog box.

Set the Criteria: 2 Click Search Criteria. You see a dialog box entitled Set Search Criteria, as shown in Figure 17-6.

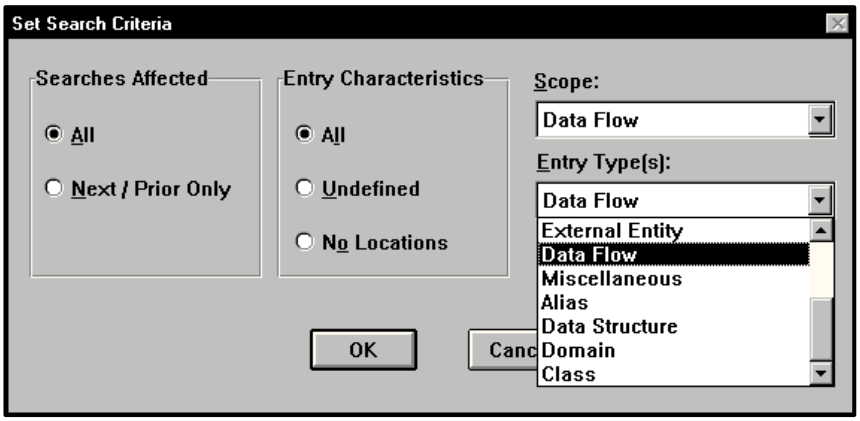


Figure 17-6 Setting Repository Search Criteria

3 In the box entitled Searches Affected, select All. This is the method used to limit the scope of the entries displayed when navigating the repository using Next and Prior, as well as the entries that are displayed when you select Search.

4 In the box labeled Entry Characteristics, select All. This tells Visible Analyst to search all items in the repository, rather than only those entries that are Undefined or entries that have No Locations. No Location entries are typically those that have been entered directly into the repository rather than added to the repository by being placed on a diagram.

5 Click the down arrow on the right side of the field marked Scope. This allows you to choose the diagram type to which you wish to limit your search. Select Data Flow.

6 Click the down arrow on the right side of the field marked Entry Type(s). This allows you to be very specific about the type of entry to which you wish to limit your search. You can choose individual types and some combination types.

7 Select Data Flow, then click OK.

Try Out the Criteria:	8	At the blank Define dialog box, click Search. Because your search criteria limits searches to data flows, the list displays only the entries in the repository of the type data flow. Select Road-Test-Criteria and then click Search.
	9	Now click Next. The next entry displayed is the next *data flow* in alphabetical order, rather than simply the next *entry* in alphabetical order. If you click Next a few more times, you notice that only data flow entries are displayed.
	10	Click Search Criteria again and set Scope back to Entire Repository. Be sure that Entry Type(s) is set to All. Click OK.

Using Search to Add Items to a Field

The Search feature can also be used to add repository entries to a field without retyping them. This option is very useful for adding multiple data elements to an Attributes field. Instead of typing the name into the field, you can select it using the Search function.

Clear the Dialog Box:	1	Click Clear.
Find an Entry:	2	Type —V‖ in the Label field and click Search. Valid-Applicant should be the first entry on the list. Click on it and it appears in the Search For field. Click Search and the repository entry for Valid-Applicant appears.
Select Attributes:	3	Click on the Attributes field.
	4	Click the Search button. The available data elements are displayed. Double-click on Address, Birth Date, Name, and Social Security Number. All the selected elements are displayed at the bottom of the Search dialog box as shown in Figure 17-7.

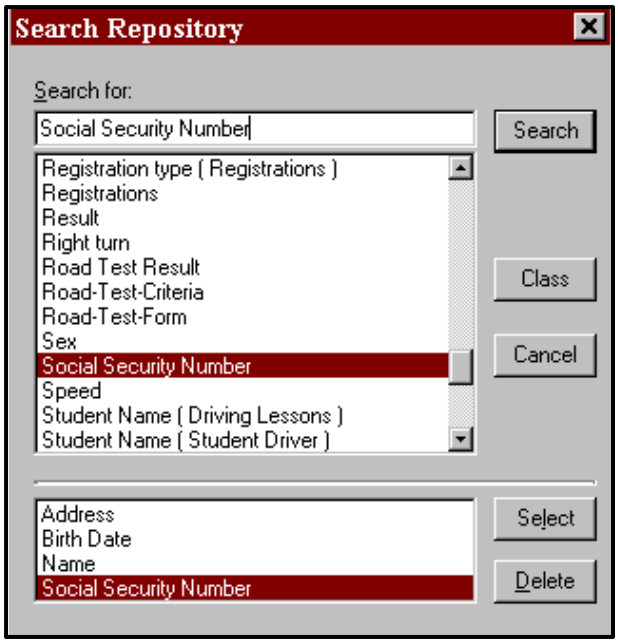


Figure 17-7 Add Information with Search

Add Attributes and Save:	5	Click Select. All the selected elements are added to the Attributes field. Click Save and then click Exit.

ADVANCED REPOSITORY FEATURES

Adding Information to the Repository

In this unit, you add attribute information to an entity; the attributes consist of the data elements that make up the entity. You also add the primary key information, so that you can demonstrate Key Analysis and Key Synchronization to migrate foreign keys across relationships automatically. All of the key information relates to the method for accessing tables in a database. We assume that each entity corresponds to one table.

Open a Diagram:	1	Open the entity relationship diagram —Driving School View‖.

Display a Repository Entry:	2	Click the ↖ button on the control bar.
	3	With the *left* mouse button, double-click the entity Student Driver. Its repository entry is displayed.
Enter Attribute Data:	4	Place the text entry cursor in the field immediately to the right of the Add button under the Attributes field. Type ―Student Name‖ and click Add. Add ―Home Address‖ and ―Age‖ in the same manner. Since the data elements you just added to the Attributes field are not already in the repository, entries for each are automatically added when you click Save.
Save the Entries:	5	Click Save to save the attributes you entered.
Enter Key Information:	6	Click the key button to display the Primary Key dialog box. Select Student Name to move it from the Columns in Table box to Columns in Key box. Click OK to return to the Define dialog box. The key notation by Student Name indicates that Student Name is the primary key for this entity.

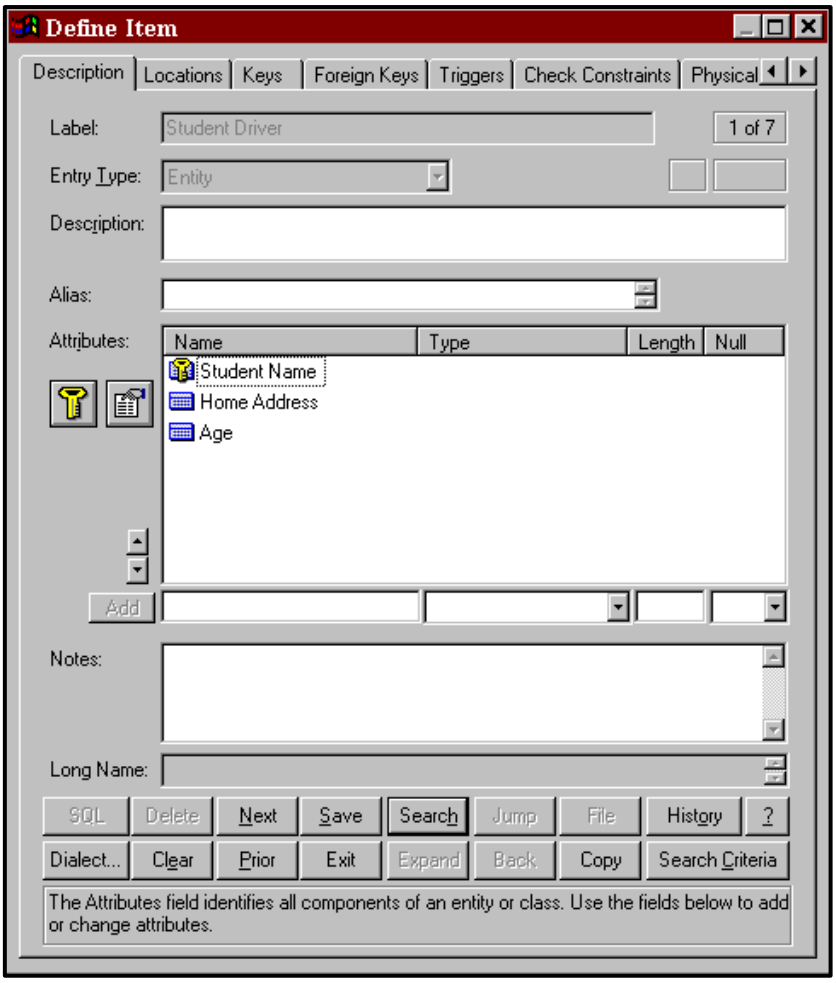


Figure 17-8 Student Driver Attribute Information

Clear the Dialog Box:	7	Click Clear. This clears the repository dialog box but does not delete the entry from the repository.
Access Another Entry:	8	Type —Driving School‖ in the Label field and press ENTER.

Add Composition:	9	Click the Attributes Details button and type ―Driving School Number‖ and ―Driving School Name,‖ each on a separate line. Click OK.
Create Primary Key:	10	Click the Key button next to the Attributes field to display the Primary Key dialog box.
	12	Click Driving School Number in the Columns in Table box to move it to the Columns in Key box. Click OK to return to the Define dialog box.
Save:	12	Click Save to save your changes, then click Clear.
Access Another Entry:	13	Type ―Driving Lessons‖ in the Label field and press ENTER.
Add Attributes:	14	Click the Attributes Details button, place the cursor in the Type field, and then click Search.
	16	Scroll the search box until Driving School Number appears. Click Driving School Number and then click Search to enter it on the Attributes dialog box. Move the cursor to Type field of the next line. Add Student Name in the same manner.
	17	Move the cursor to the Type field of the next line, and type ―Lesson Number.‖ Click the cursor in the Limit field to enable the Physical Characteristics pane at the bottom of the dialog box. Select Integer as the Data Type.
	18	Click OK to add the attributes to the Define dialog box.
Create Primary Key:	19	Click the Key button next to the Attributes field to display the Primary Key dialog box. Click Lesson Number in the Columns in Table box to move it to the Columns in Key box. Click OK to return to the Define dialog box.
Save and Exit:	20	Click Save and then click Exit.

Key Analysis and Key Synchronization

The Key Analysis and Key Synchronization functions, found on the Repository menu, can help you set up a consistent relational database key structure. There are three types of keys used in a data model: primary, foreign, and alternate keys. All keys are designated in the Attributes field of an entity in the project repository. A primary key is one or more attributes or data elements that uniquely identify an entity. To designate a data element as a primary key, the yellow key notation is used in the Define dialog box. On the diagram, primary keys are displayed in the area immediately under the entity name when the primary key level is selected from the control bar or the View/Entity Display Options menu. A foreign key is a non-key attribute in one relation that appears as the primary key (or part of a compound primary key) in another relation. The gray key notation in the Attributes field of an entity designates a foreign key. The FK notation is shown when the entity on the diagram is displayed at the attribute view level.

Key Analysis verifies that the key structure for your data model is complete, checking that all key information is correctly identified for the data model. Key Synchronization analyzes the key structure and migrates data elements that you designate as keys, or parts of compound keys, across relationships to their associated entities, and creates the resulting foreign keys. Using associator element names in relationship repository entries makes this process work better. (Please check the Visible Analyst manual or online help system for an explanation of associator elements.)

Key Analysis and Key Synchronization both involve analyzing the primary key [PK] and foreign key [FK] designations in the TEST project repository. A primary key is an attribute or data element that uniquely identifies a record.

Run Key Analysis:	1	Select Key Analysis from the Repository menu. Visible Analyst scans the entire repository and indicates any errors it finds.

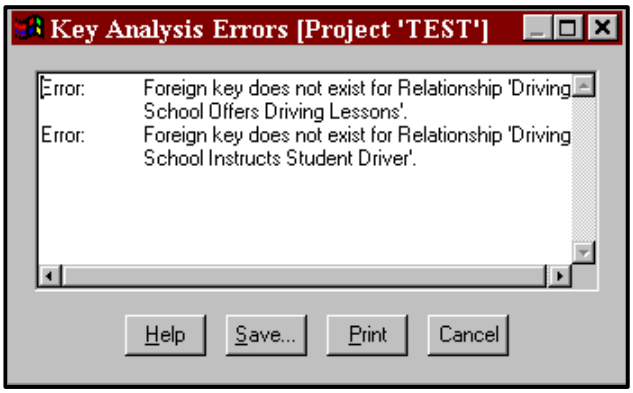


Figure 17-9 Key Analysis Error Messages

View the Errors:

2 Click the Maximize button in the upper right corner of the errors dialog box. Scroll through the messages. You see that there are error messages indicating missing foreign keys for the entities on the current diagram.

Note

You can keep analysis error dialog boxes on the screen while you carry on various Visible Analyst activities. This is to make it easier for you to correct the errors found by Analyze. The same holds true for SQL Schema Generation, Shell Code Generation, etc.

.

3 Click Cancel.

4 Select Key Synchronization from the Repository menu. Visible Analyst first analyzes for key errors and then migrates the foreign keys across relationships.

5 Maximize the Key Synchronization Messages dialog box. Key Analysis messages appear first, followed by Key Synchronization messages. You should notice the Key Synchronization messages, indicating the foreign keys that have been migrated.

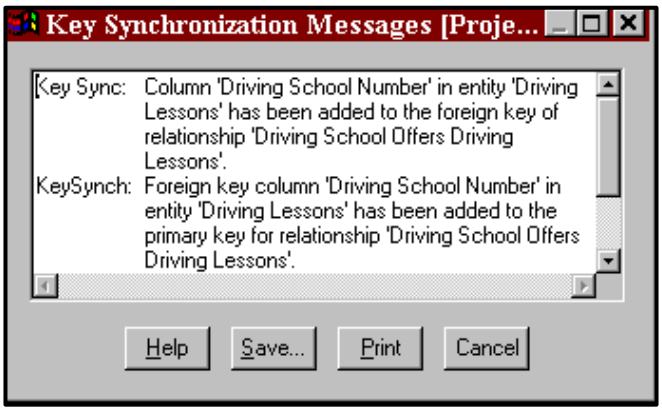


Figure 17-10 Key Synchronization Messages

6 Click Cancel.

Examine the Migrated Key:

7 Double-click Student Driver. Notice the foreign key Driving School Number that has been added. This was done by Key Synchronization. It saves you from migrating all of the foreign keys manually.

Note also that Analyze added text describing the key. All text following an asterisk is considered a comment and is ignored by the repository. (When the object interface is enabled, comments are not displayed.)

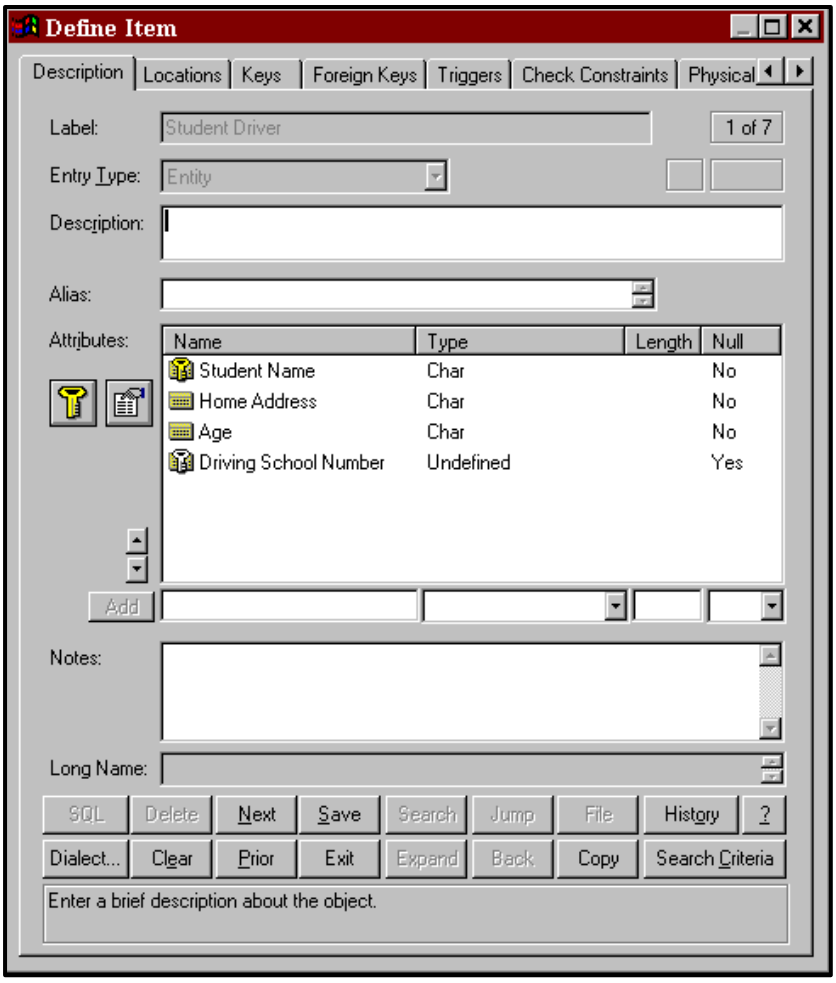


Figure 17-11 New Foreign Key Information

8 Click Exit.

9 Deselect Student Driver on the diagram.

View Objects

Visible Analyst Corporate and Zachman Editions support the concept of an SQL view, which can be thought of as a derived or virtual entity. A view is similar to an entity in that it has a

composition, but the items that appear in the composition of a view must belong to other entities or be expressions based on data elements used by another entity.

An SQL view is made up of two major components: a list of column names and a select statement that is used to filter information from the tables in the view. The select statement can contain not only the primary select clause, but also any number of sub-selects and union selects. When view is selected as the entry type, view-specific Define dialog box pages are displayed. Using these pages, you can select tables, columns, join relationships, clauses, and other options for the view. An expression builder is available to help you create the expressions to be used in the filter, group by, having, start with, connect by, or join expression controls.

Detailed information about views can be found in the *Operation Manual* and in the online help system.

Note

Views are not available in the Education Editions of Visible Analyst.

Generate Database Schema

The Corporate and Zachman Editions of Visible Analyst generates SQL DDL (Structured Query Language – Data Definition Language) schema from the information contained in the repository. In the Corporate and Zachman Editions, you can select from several different dialects of SQL, including a User Defined type, to allow the use of a dialect not currently supported by Visible Analyst. For more information on the custom feature, see the *Operation Manual* or the online help system. The statements that are supported include CREATE TABLE, CREATE INDEX, and COMMENT ON. More information is contained in the *Operation Manual* or in the online help system.

The Education Editions of Visible Analyst allow you to generate SQL for Microsoft Access and Oracle only. To generate SQL:

Choose Access Dialect:	1	Choose SQL Dialect from the Options menu, then choose Access.
Generate SQL Schema:	2	Select Generate Database Schema from the Repository menu to generate the schema. When the dialog box appears, click OK. (Refer to the *Operation Manual* or online help system for details of the SQL Schema Generation dialog box.) If errors are found, they along with the generated schema will be displayed.
View the Schema:	3	Maximize the SQL generation dialog box.

4 Click the Schema button to display the generated schema. See Figure 17-12. If Visible Analyst does not have the information to generate the schema, a list of errors is displayed; but no Select box is present. Click the Errors button to view any errors. (If too many errors are generated, the Schema button is not displayed.)

```
CREATE TABLE Department_of_Motor_Vehicles
(
   DMV_Number             INTEGER NOT NULL,
   Address                CHAR(30),
   Number_of_Evaluators   INTEGER,
   Number_of_Evaluators   INTEGER,
   Evaluator_Number       NUMBER,

   CONSTRAINT PKC_Department_of_Motor_Vehicles0000 PRIMARY KEY ( DMV_Numb
);

CREATE TABLE DMV_Evaluator
(
   Evaluator_Number         NUMBER NOT NULL,
   DMV_Number               INTEGER NOT NULL,
   Evaluator_Name           VARCHAR(40),
   Evaluator_Address        VARCHAR(60),
   Evaluator_Phone_Number   VARCHAR(12),

   CONSTRAINT PKC_DMV_Evaluator0001 PRIMARY KEY
      ( Evaluator_Number, DMV_Number ),

   CONSTRAINT FKC_employs0002 FOREIGN KEY ( DMV_Number ) REFERENCES
      Department_of_Motor_Vehicles
```

Figure 17-12 Generated SQL Schema

Shell Code Generation

The Corporate and Zachman Editions can generate C and COBOL shell code. The code that is generated encompasses the sequence of functions or paragraphs that make up a program, including global definitions, descriptive comments, function call/PERFORM statements, and passed parameters. Information entered in text fields in the repository entry for a program item or a structure chart module produces comments that describe these items within the generated code. Also, actual source code can be entered in the module description field of a module or macro, and this code is placed in-line with the function calls or PERFORM statements that are generated by invocations. Couples or ITRs used with invocation lines generate parameters for C code. There is also an option to customize the code to be

generated. (See the online help for other generation options supported by Visible Analyst, such as AS/400 DDS, Visual Basic, PowerBuilder, etc.)

XML Generation

Visible Analyst can generate the XML Schema based on the W3C standard by selecting the Tools | Export | XML Schema (XSD) menu option. The XML file is generated for the entities and (optionally) classes developed in the project. The XML file is written to the Visible Analyst TRANS folder.

XML DCD code can also be generated based on the data models. This is similar to SQL schema generation. XML can be selected as the generation option when you select SQL Dialect from the Options menu. The procedure is similar to the SQL DDL generation. See the *Operation Manual* or the online help for more information.

Repository Reports

Now you practice generating a report on the data contained in the repository. This is a basic report containing a detailed listing of all entries contained in the repository. For detailed information about Reports and Report Queries (Custom Reports), see the *Operation Manual* or the online help system.

First set the font for the report you want to generate.

Set the Report Font:	1	From the Options menu select Text Settings.
	2	Under Text Type, select Report Body.
	3	Select a typeface and point size, and click OK.
Set the Report Criteria:	4	Select Reports from the Repository menu. The Repository Reports dialog box appears (see Figure 17-13).

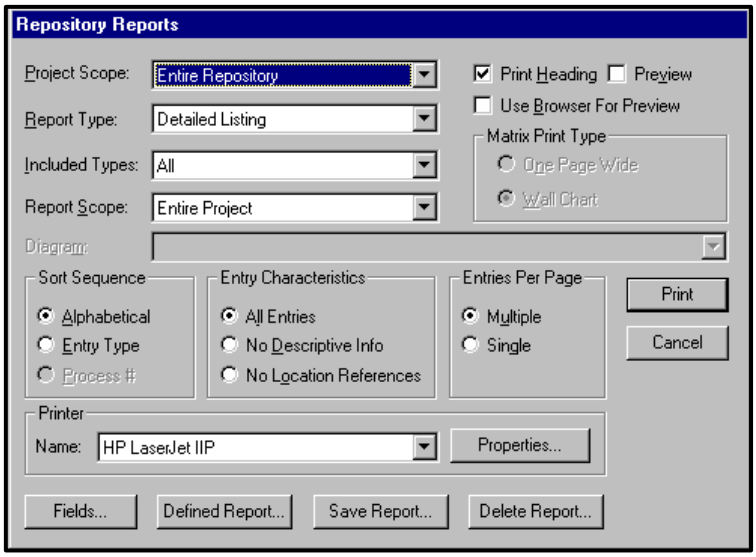


Figure 17-13 Repository Reports Dialog Box

5 Under Project Scope, select Entire Repository.

6 Under Report Type, select Detailed Listing.

7 Under Included Types, select All.

8 Under Report Scope, Entire Project is selected.

9 In the box labeled Sort Sequence, select Alphabetical. This determines the entry order in your report printout.

10 In the box labeled Entry Characteristics, select All Entries.

11 In the box entitled Entries Per Page, select Multiple Entries Per Page. You can select Single Entry Per Page to reorder the pages of your report once they have been printed.

Run the Report:

12 Click Print; the information is sent to the printer. Select Preview to view the report first.