Graphical Development Environment
for
Postgres Object Oriented Database
(GDEP)

Khaled A. Jababo

July 1994

Forward comments or questions about this report or the GDEP system to: bcdesai@cs.concordia.ca

ABSTRACT

Graphical Development Environment for Postgres Object Oriented Database (GDEP)

A Software Development Environment is a collection of tools that support various phases of Software Development. The environment in which an application is developed may have a larger effect on the programming process than does the language in which the programmer writes an application. Thus, tools of a Software Development Environment are very important; GDEP provides some tools to help Postgres developers be more efficient and more productive.

GDEP provides Postgres users the ability to create, edit, view, import and export classes to and from the Database without knowing the syntax of Postquel, which is the query language for Postgres. Also, GDEP provides developers with a Command Line Interface window allowing them to issue direct Postquel commands.

In this report, we discuss the functionalities of GDEP and how to use it, as well as the two enhancements GDEP adds to Postgres: propagating, or adding a new attribute to all instances of a class, and changing the parent class of an existing class. In addition, GDEP has the ability to show class' properties as a flat class as well as a regular class with inheritance properties. These features are not supported by any available Postgres tools.

Acknowledgments

I would like to express my deepest gratitude to Dr. Bipin C. Desai. His guidance and encouragement made this project work a pleasant and extremely educational experience. More than anyone else, I would like to thank my mother. It was her continued support and encouragement that made this project possible.

This report is a modified version of a Master's project in Concordia University.

Contents

List of Figures

Chapter 1

INTRODUCTION

Since Postgres is public domain software, many development environment packages have been developed to make Postgres an easier database to use. Some packages use a command line interface such as Spog [Spog] to access the database, and others use a graphical user interface to access the database, examples of the latter are Alberi [Alberi] and Geo [Geo]. When Spog was introduced, it was a better tool than monitor [Pg Rel 4.1], which is the standard command line interface for Postgres. Subsequently, Geo was developed with a graphical user interface, but it is not a general tool. It is specialized for geographical maps, which is useful for landscape domain applications. Finally Alberi was developed, which has a nice graphical user interface, and some functionalities that are useful for Postgres users. Alberi was developed on top of OpenWin tools.

In this report, we discuss the GDEP package, which we developed as a graphical development environment for Postgres using the Motif Toolkit. GDEP provides Postgres users all the functionalities that Alberi provides, as well as some others that Alberi does not provide. Also, GDEP adds some enhancements to Postgres which are important with respect to Object Orientation principles. Throughout this report, we describe the functionalities of GDEP and the enhancements of GDEP to Postgres, give a sample Postgres application using GDEP, describe details of how to customize and install GDEP, and finally present our conclusion and talk about future work.

In the rest of this chapter, a description of the organization of the report is introduced. This is followed by an informal description of Postgres, GDEP, and Motif.

1.1 Organization of the Report

The functionalities and features of GDEP are presented in Chapter 2. Enhancements of GDEP to Postgres are described in Chapter 3. In Chapter 4, we give a sample application, with specific features to instruct GDEP users how they can use GDEP to build their own applications. In Chapter 5, we describe how to customize and install

GDEP, in addition to a description of GDEP as a package. Finally, conclusions are given in Chapter 6.

1.2 Overview of Postgres

Postgres is a database research project developed under Professor Michael Stonebraker at the University of California at Berkeley. It is a multi-user Object Oriented Database Management System.

Postgres has a query language called Postquel, which is an incompatible superset of Quel. Names in Postquel are sequences of no more than sixteen characters (alphanumeric), starting with an alphabetic (underscore is considered an alphabetic). Postgres adds an Object Identifier (*Oid*) to all instances automatically. Oid is a unique identifier of an instance, not reused, thirty-two bits long. Postgres supports five types of constants: character, string, integer, floating point, and other constant types defined by Postgres users. It also supports a date type, which is a character string of the following format: 'MMMDD[HH:MM:SS]YYYY'.

The reader may refer to [Pg Rel 4.1] for more details on any subject related to Postgres syntax or semantics used in this report. Postgres consists of three processes:

1.2.1 The Postmaster

This is a process which acts as a clearing-house for requests to the Postgres System. Basically, front end applications connect with the Postmaster, which keeps track of any system error and communication with the backend process. The Postmaster must be running to run any of the Postgres commands.

1.2.2 The Terminal Monitor

This is used for direct access to the database, and is a front-end user interface to the Postgres backend. It sends commands to the Postmaster, which forwards the commands to the backend. GDEP has a command window which plays the same role as the monitor, but the results are displayed in the display area window in GDEP.

1.2.3 The Backend

This is a process which does all the 'real work'. This process is started by the Postmaster when the it receives a connection from the terminal monitor, or from any other interface that talks to the Postmaster, such as GDEP.

1.3 Overview of Motif

Motif is a toolkit from the Open Software Foundation (OSF) that allows programmers to write window-based applications. The Motif toolkit is based on the X Toolkit Intrinsics (Xt), which provides an Object-Oriented Framework for creating reusable, configurable user-interface components called widgets, supporting a convenient interface for creating and manipulating X windows, colormaps, events, and other cosmetic attributes of the display.

Widgets are objects that operate independently of the application; they know how to draw themselves and how to respond to certain events. For example, a PushButton widget knows how to draw itself, highlight itself when it is clicked on with the mouse, and respond to that mouse click. Motif provides a complete set of widgets that follow a Motif Style user-interface designed to implement the application.

Xt and Motif follow the Object-Oriented approach, where the application programmer is completely insulated from the code inside the widgets. A programmer has access to functions to create, manage and destroy widgets, plus certain public widget variables known as resources. If the application programmer wants to keep the user from modifying resources, the values of a widget may be set at creation. An important class of resource, which must be set from the application, is a widget's callback lists. Each widget that expects to interact with an application publishes one or more callback resources. The application must associate with that resource a pointer to the application function, that will be invoked when the widget is selected.

In summary, X Toolkit Intrinsics provides functions for creating and setting resources on widgets. On the other hand, Motif provides the widgets themselves, plus an array of utility and convenience functions for creating groups of widgets that are used collectively as a single type of user-interface element. The reader may also refer to [Motif] for more details on the Motif Toolkit.

1.4 Overview of GDEP

GDEP is a Graphical Development Environment for Postgres users. It has many functionalities in addition to some enhancements made to Postgres. The enhancements made to Postgres are the following: GDEP propagates a new added attribute to a class to all instances of the class, and GDEP allows Postgres users to change the parent class of an existing class. Functionalities and enhancements of GDEP, plus other issues related to GDEP, will be discussed in the following three chapters. GDEP consists of three main parts which are described in the following subsections.

1.4.1 Graphical User Interface

GDEP uses the Motif Toolkit to build the front end Graphical User Interface to the Postgres application. The Graphical User Interface consists of the main window, the main menu bar, the header area, the display area, and finally, the command area. The system is supported with dialog boxes and selection boxes when needed, so the user does not have to remember any Postquel command or the contents of the database to access or manipulate it. The input to the system can be done either by clicking on the mouse or by using the keyboard; both are compatible with Motif and X toolkit functionalities to navigate through the window.

1.4.2 Database

The backend Database Management System (DBMS) used by GDEP is Postgres. GDEP supports most frequently used Postquel queries through the main menu bar. GDEP also has a command area window, which plays the same role as the Postgres terminal monitor. It allows users to issue their queries directly to the database. This is useful in case users like to issue Postquel commands that are not provided through the main menu bar. In addition to the terminal monitor functionalities, GDEP provides users with a selection log option, which will pop up a selection window allowing the user to select a previously executed command. Every time the user executes a command, it will be added to the log file.

1.4.3 Implementation

GDEP is written in C. The code is written to be very general, in that all functions can be reused. This allows the GDEP users to modify the source code in order to build their own applications. Every function is preceded by a short paragraph explaining what it is, and how to use it. Examples of these are the input and output functions. Also, every block in a function is preceded by a short paragraph explaining the functionalities of the block.

The Motif and X Toolkits follow the same style in coding: they both add a special symbol at the beginning of every variable parameter used by their code. For example, X Toolkit adds Xt at the beginning of every variable and parameter used by the X Toolkit, which means 'X Toolkit intrinsic variable'. Also, Motif uses Xm at the beginning of every variable used by the Motif Toolkit, which means 'X motif variable'. In this manner, we followed the same style in coding which Motif and X used in their code. In GDEP, all parameters and variables that belongs to the Graphical User Interface start with Xa, and all other variables and parameters that belong to the main application start with Ma. Also, functions that are concerned with the Graphical User Interface start with XaF, and other functions start with MaF.

GDEP consists of a Makefile file, a header file, and nine C files. The Makefile is used to compile and link all the Object files used by GDEP. The Header file contains all the include files of the external libraries called by GDEP, the define section of all the constants used by GDEP modules, the define section of all the data structures used by GDEP modules, and finally defines all the global variables needed for the application. The libraries called by GDEP are: Motif library (Xm), X Toolkit Intrinsics library (X11), and Postgres Library (libpq and libpq-fe). Each C filename can be considered as a module. These modules are:

1. **mainfile.c**

   This module contains the main function to start the Postgres database application. It initializes the parameters, then builds the main window for the application. This module uses the status file if it exists to initialize the variables of the application. For example, it initializes the width and the height of the main window, in addition to many other parameters. If the status file does not exist, then GDEP uses the default values set by the application to initialize its variables. This is useful if GDEP users would like to reset the application; then all they must do is remove the status file from the directory where GDEP is running.

2. **xaevents.c**

   This module contains all the functions needed to handle the events that occur in the main window with respect to the user request, whether it is an input command, a submenu choice, a menu choice or any other request. Every time the user clicks on a Pushbutton, an event is generated. The generated event sends to this module an event identifier to instruct this module what type of event has occured. Then, this module invokes the right method in module xahandle.c to handle the specific event.

3. **xahandle.c**

   This module contains all the functions needed to respond to the events that the program must take care of in xaevents.c. Every event handler has its own method of handling the event. It might respond to the user directly by displaying or executing the command requested by the user. Otherwise, it might invoke other events such as a dialog box to get more detail needed for the original event to be handled.

4. **xatools.c**

   This module contains all the functions needed as tools to the Graphical User Interface of the project, such as clear a particular window, get contents of a particular window, fill a particular window with a string, and so on. Also, this module contains the functions to set database window values, as well as the class window values that are selected by the user. In addition, this module takes care of setting the X and Y positions of the main, dialog box, and instance windows. It also manages and handles all the dialog boxes created by the application.

5. **matools.c**

   This module contains all the functions needed by the main Postgres database application. It contains functions that are used to open and close the database, and destroy the class tree. It also contains functions that initialize the application, and store the current status of the application. Finally, it contains functions that handle the Postquel queries that are issued to the database. These functions are of different formats: some of them return a string to the calling program, others return specific data structures to meet the need of the clients of this module.

6. **instance.c**

   This module contains all the functions needed by the Postgres database application to respond to the user request to manipulate, instance by instance, a particular class in the database. This module handles different functionalities for the instances such as add a new instance, delete current instance, update current instance, show next instance, show previous instance and cancel. In some cases, previous and next pushbuttons disappear, according to the contents of the database and the position of the current instance in the database. In some other cases, the instance window might disappear by displaying a notice message if there are no more instances for the specified class in the database.

7. **maexport.c**

This module contains all the functions needed to export class definitions and class instances to external text files. The exported text files are compatible with Postgres monitor command files, in that there is a \g at the end of each command. This is useful in case those commands need to be issued to the Postgres database from Postgres Tools other than GDEP. It also contains functions to import text files to the database containing Postquel commands.

8. **function.c**

This module contains all the functions needed by the main application to manipulate strings and linked lists. It contains functions to create and free pointers of a specific data structure. Finally, it contains functions that get size or find a particular string in a linked list data structure.

9. **pattern.c**

This module contains all the functions needed to find a particular pattern in a string. Also, it has the capability of finding a specific pattern in a file. In both cases, the module returns to its client the string which comes right after the pattern if the pattern is found. Otherwise, it returns an empty string. These functions are used to find the value of the parameters in the status and log files, which will be discussed in section 5.2.

Chapter 2

COMPONENTS OF GDEP

In this chapter, we will describe the different parts of GDEP in detail, and describe the functionalities and features of each part individually. In addition, we mention the exceptions of every part when applicable. To clarify our description, we refer to many figures, each representing a particular state of GDEP.

To start GDEP, first the user must install it; this is described in detail in section 5.1. After downloading the GDEP files, the user must type 'make' from the same directory where the GDEP package resides. Make will generate an executable file called 'gdep'. When the user types 'gdep', a GDEP main window will appear, as shown in figure 2.1.

2.1 Main Window

The main window for GDEP is shown in figure 2.1. As we can see in the main window layout, the GDEP header title is shown at the top of the window, which is entitled 'Graphical Development Environment for Postgres'. Then, the GDEP main window is split into four main subwindows:

1. Main Menu Bar
2. Header
3. Display Area
4. Command Area

Each SubWindow will be described in detail in the following sections.

GDEP users can take advantage of Motif built-in features (tools) to resize each subwindow within the main window. The user can do this by clicking with the left button of the mouse on the small rectangle on the line that separates each subwindow from the others, then moving it up or down according to the user's need. Finally, releasing the button at the new position of the subwindow will resize the old windows

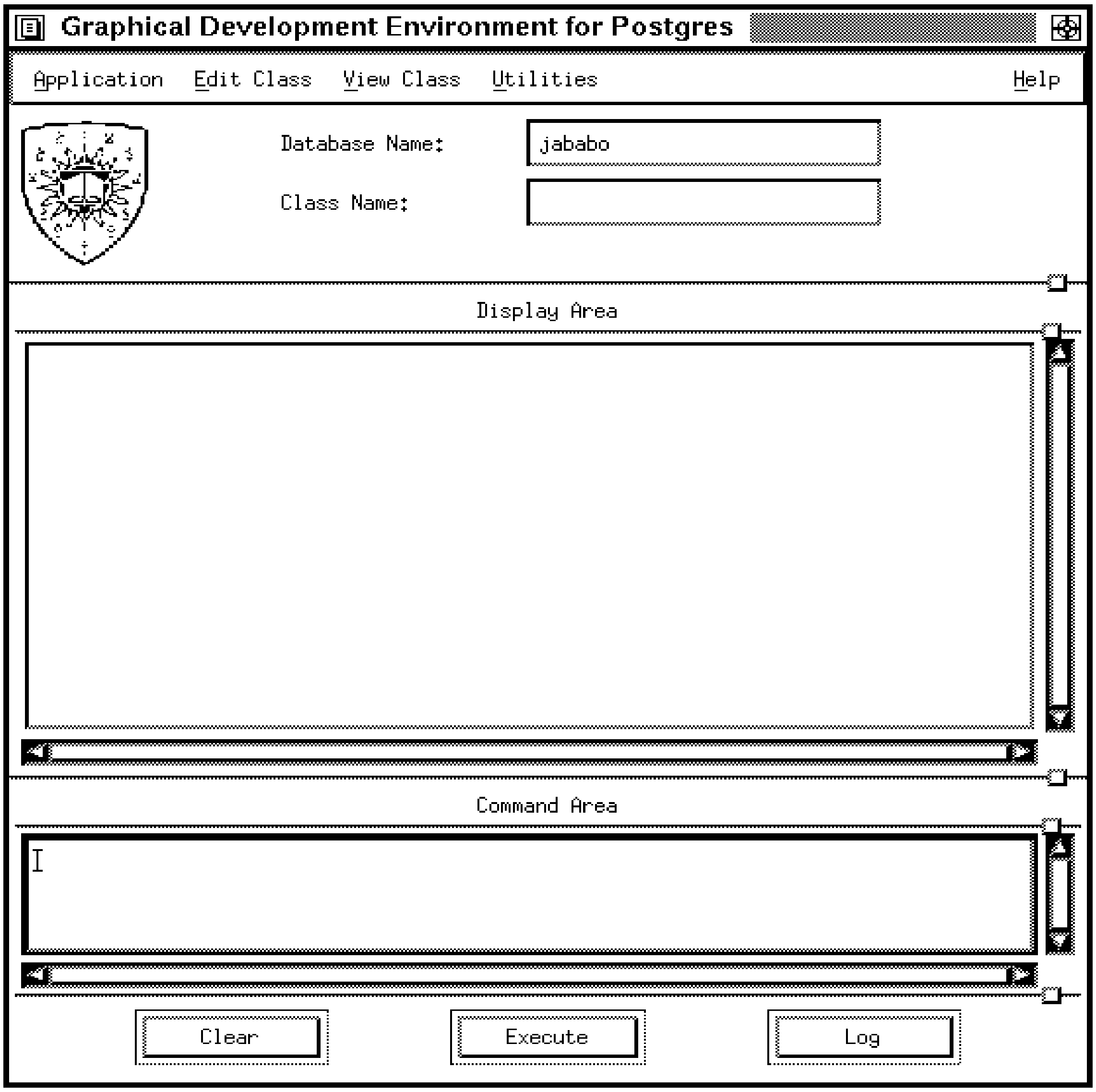

Figure 2.1: GDEP Main Window Components

to the new size desired. This operation can be done with any other window created by the GDEP main window. If the user relocates any window created by GDEP, then its location will be saved by GDEP in the status filename, which will be discussed in section 5.2.2. Otherwise, whatever changes are done to any GDEP window will not change the status of the GDEP application.

2.2 Header Window

In this section, we describe the Header Window with respect to GDEP, and discuss in detail each of the three components of the Header Window. As we can see in figure 2.1, the Header Window is located at the top of the main window, between the main menu bar and the display area. It is composed of three components:

2.2.1 Concordia Logo

This is a drawing area that displays the Concordia University logo. It is located at the left-hand side of the Header Window, and its left, top and buttom borders are attached to the Header Window. This logo's code is embedded within the GDEP application's source code, and it cannot be modified or changed through any external resources for GDEP.

2.2.2 Current Database Name

This area is used to inform GDEP users as to which database they are currently working on. The area is located high on the right-hand side of the Header Window. It contains the field name 'Database Name' and the field contents located to the right of the field name. As we can see, in figure 2.1 the field contents is 'jababo'. How to set or modify this field contents will be discussed later, in section 2.5.1.

2.2.3 Root Class Name

This area is used to inform GDEP users of the current root class. The area is located low on the right-hand side of the Header Window. It contains the field name 'Class Name' and the field contents located at the right of the field name. As we can see in figure 2.1, the field contents is empty. How to set or modify this field contents will be discussed later, in section 2.5.1. In the rest of this report, this field will be referred to as 'Root Class', 'Current Class' or 'Selected Class'.

2.3 Display Area Window

This window is located between the Header and Command Area Windows. It consists of two parts: the name of the window, and the scrollable text area where the user can use the up or the down arrows to see other lines of the text simultaneously. The user can also use the left or right arrows to shift the display view in the left or right direction. The name of the window, 'Display Area', is displayed at the middle of the top of this window; the scrollable text area is located right below the name of the window, in a read-only mode. It is used to display the results of the queries executed in the command area of GDEP, as well as the view instances option of the submenu item View Class. If the field length of any instance variable in the result of the query exceeds seventeen characters, then only the first seventeen characters will be displayed, and a star will be displayed at the end of the field to inform the user that there are more data in that particular field. If the user wishes to see the remaining data in that field, 'view instances by tuple' may be selected from the View Class option of the main menu bar. For example, as we can see in figure 2.2, there are stars at the end of the data fields of the instance variables career, hobbies and address. From this we know that more data exists for these particular fields. Also, we realize that the buttom srcollable arrows are half highlighted, which tells us that there is more text at the left side of the Display Area window. Clicking on the left arrow allows us to see the rest of the instance variables and their data for class STUDENT.

2.4 Command Area Window

This window is located at the buttom of the main window; it is also attached to the display area window. This window consists of three parts: the name of the window, the scrollable text area and the command area. The name of the window, 'Command Area', is displayed at the middle of the top of this window; the scrollable text area is located right below the name of the window, in a write-only mode. It is used to input the user's Postquel commands to interact directly with the Postgres backend process. The user can click the mouse within the scrollable text area, and start typing the Postquel commands directly. Finally, the command area consists of three pushbuttons which are used to manipulate the scrollable text area of the Command Area Window, and execute the Postquel commands as well. When any of the three pushbuttons is pushed, it will be highlighted as a feedback to the user. The name and the functionalities of each pushbutton will be presented in the following subsections:

2.4.1 Clear PushButton

This pushbutton is used to clear the command area window. After this button is pushed, the value of the command area text will be an empty string, regardless of what contents it had before it was pushed. This is helpful in case the user wants to issue a new Postquel command to the backend process. Instead of deleting the old Postquel command and retyping the new one, the user can push this button and then type the new Postquel command, or select from previously executed commands, which will be discussed later in section 2.4.3

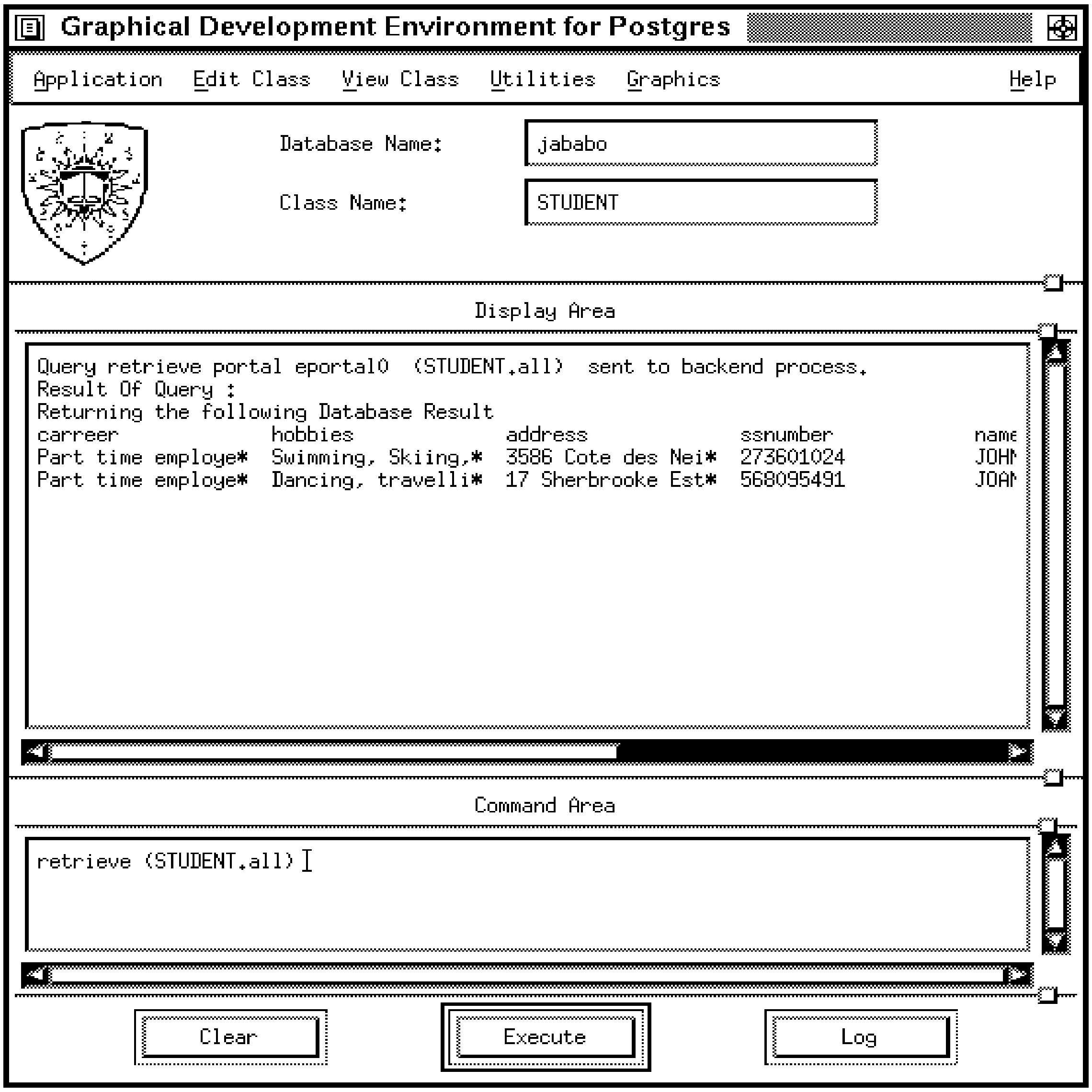

Figure 2.2: Command Area and Display Area of Main Window

2.4.2 Execute PushButton

This pushbutton is used to issue a Postquel command to the backend process of the Postgres database. The contents of the command text area will be the value of the Postquel command. The result of the Postquel command from the backend process will be displayed in the display area window. If the executed command is an invalid Postquel command, then GDEP will display an appropriate message informing the user that the command could not be executed. Otherwise, if the executed command is a query command, then the result of the query as instance variable names and corresponding data values will be displayed in the display area. Else, if the executed command is not a query command then a message stating that the command has been executed successfully will be displayed. The GDEP user is always given a feedback for the executed command, to be informed of its status. For example, if the command is a not a query and could not be executed by the backend process, then GDEP displays a message in the display area stating that the command could not be executed, and the user knows that there is something wrong in the command. Every time a nonempty string is executed, GDEP will append the executed command at the end of the log file 'username.log'. For example, in figure 2.2, after pushing the 'Execute' pushbutton, we see that the button is highlighted. Then GDEP issues command 'retrieve (STUDENT.all)' to the Postgres backend process of 'jababo' database. The command could be entered directly by the user by first clicking in the command area text, followed by typing the command through the keyboard, or it could be selected from the log file through the log pushbutton to be discussed in detail in the next section, 2.4.3. The result of the query is shown in the Display Area window. The first line displays the query that has been issued to the database. The fourth line displays the name of the instance variables, such as 'career', 'hobbies', *etc.* The fifth to the last lines display the instance data values of the result of the executed query.

2.4.3 Log PushButton

This pushbutton is used to help GDEP users look at previously executed commands, and give them the option of selecting any of them. When a user pushes this button, a selection popup window will appear, as shown in figure 2.3, displaying in a scrollable window all the previously executed commands. Every logged command will appear on one line. The user can click once on the selected command, which will copy the log command into the 'selection' field of the selection popup window. At any time, the user can edit the selection field if needed, because it is a text field type. Then, clicking on the 'Ok' pushbutton will copy the data in the 'selection' field into the command area of the GDEP main window, then destroy the selection popup window. Or, the user can select any log command by clicking on it twice, which will copy the selected command into the command area of the GDEP main window, then destroy the selection popup window. Alternatively, clicking on the 'Cancel' pushbutton destroys the selection popup window without changing anything in the GDEP main window. For example, clicking on the last line in the scrollbar text area 'Log Command List' of the selection log command popup window of figure 2.3 causes the log command 'retrieve (STUDENT.all)' to be copied into the selection field at the buttom of the selection popup window. Then, clicking on the 'Ok' pushbutton will copy the contents of the selection field into the command area of the GDEP main window, as shown in figure 2.2, and destroys the selection log command popup window of figure 2.3.

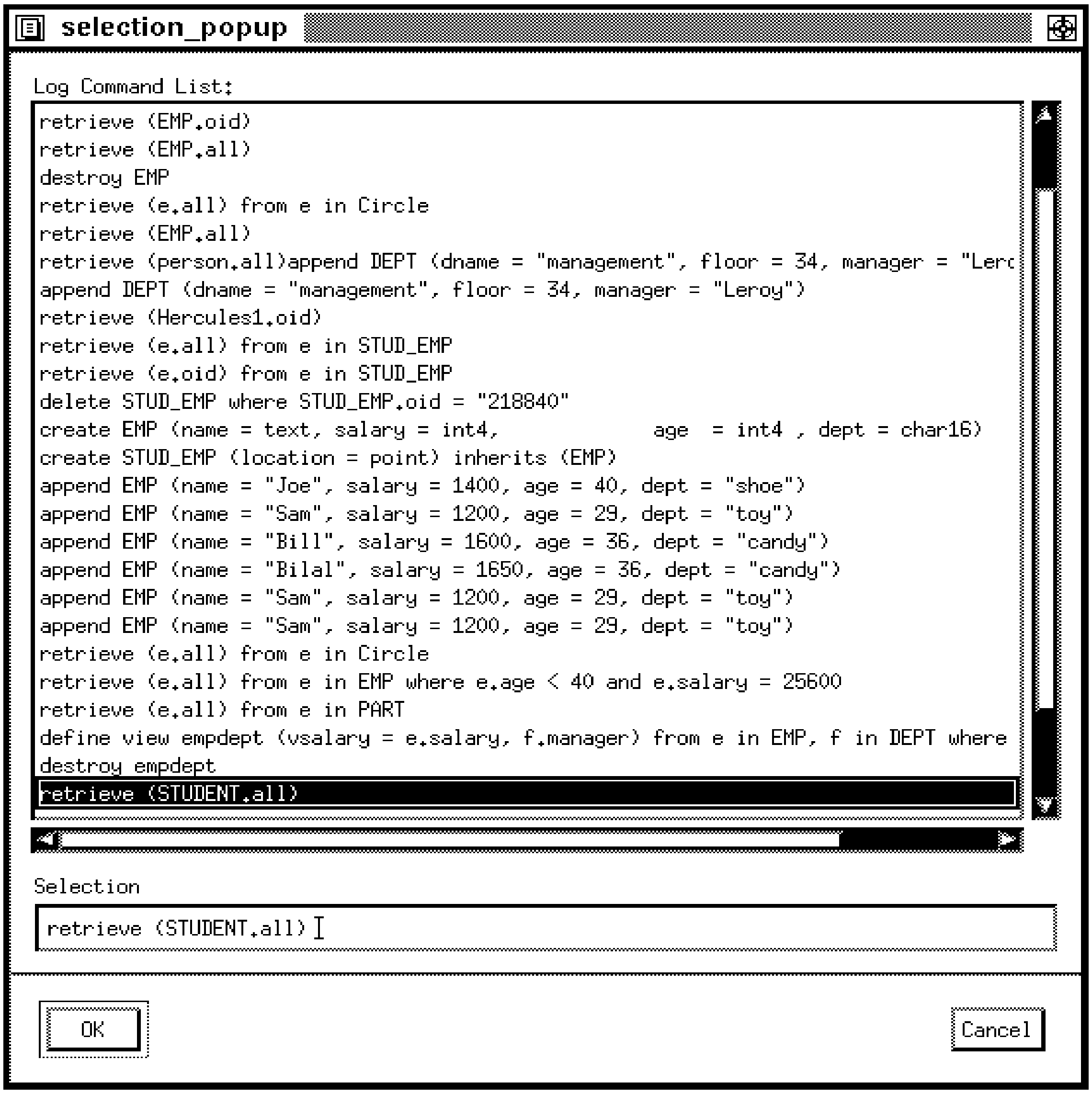

Figure 2.3: Log Command List Window

2.5 Main Menu Bar

As shown in figure 2.1, the main menu bar is located at the top of the GDEP main window, right after the window title, 'Graphical Development Environment for Postgres'. The menubar consists of five menu items:

1. Application
2. Edit Class
3. View Class
4. Utilities
5. Help

Each item is a title for a pulldown submenu. When the user clicks on any of the menu bar items, then a pulldown submenu will be displayed giving the user more options to select from. The user can click on any of the items in the submenu, which will activate a specific task that is attached by GDEP to the selected item. Figure 2.4 shows all the items of the main menu bar, and all the pulldown submenu items of each item in the main menu bar as well. The purpose of each menu bar item, a list of its pulldown menu items, and the functionalities of each one of them will be described in the following subsections.

2.5.1 Application Main Menu Bar Item

This menu bar item consists of four pulldown submenu items:

1. Load Database

   This pulldown submenu item is used to set up and initialize the database for a GDEP application. When the user selects this option, a selection popup window will appear with a list of the available Postgres databases the user can select from. The user must select one of the database names in the list 'Database List', as shown in figure 2.5. If the user selects an item that is not from the list, then a warning message will be displayed stating that the selection must be from the list. After the warning message is acknowledged, GDEP destroys the selection popup window and goes back to the previous state prior to selecting this pushbutton. Otherwise, if the selected item is from the list 'Database List', then GDEP sets the database for the application to the selected database named by the user. Also, it sets the value of the field 'Database Name' in the header window to the value of the selected database name through the selection popup window, then destroys the selection popup window.

2. Set Root Class

   This pulldown submenu item is used to set the root class for a GDEP application. When the user selects this option, a selection popup window will appear with a list of the available classes in the current database. The user must select one of the class names in the list 'Class List', as shown in figure 2.5. If the user selects an item that

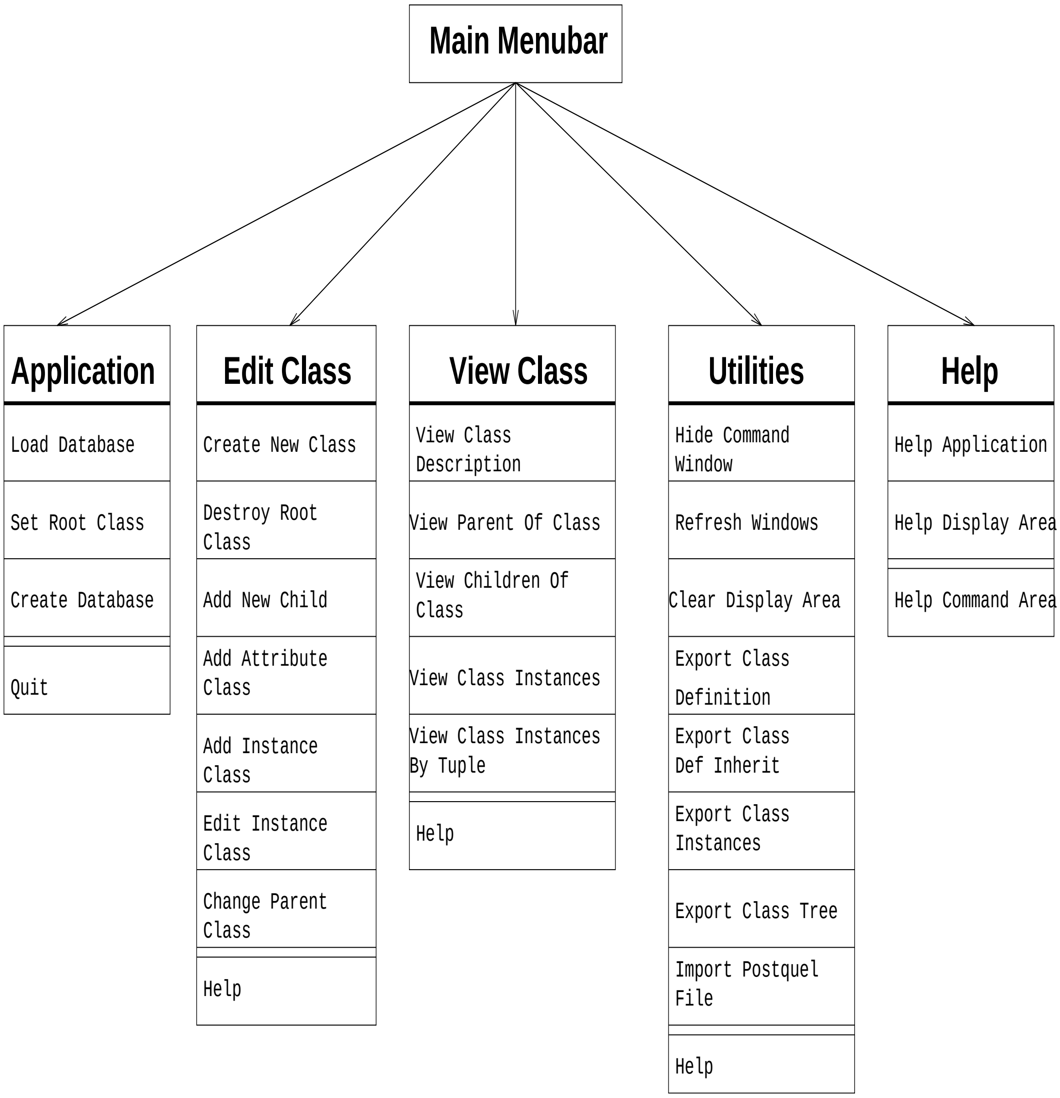

Figure 2.4: Main Menubar with all its Pulldown Submenus

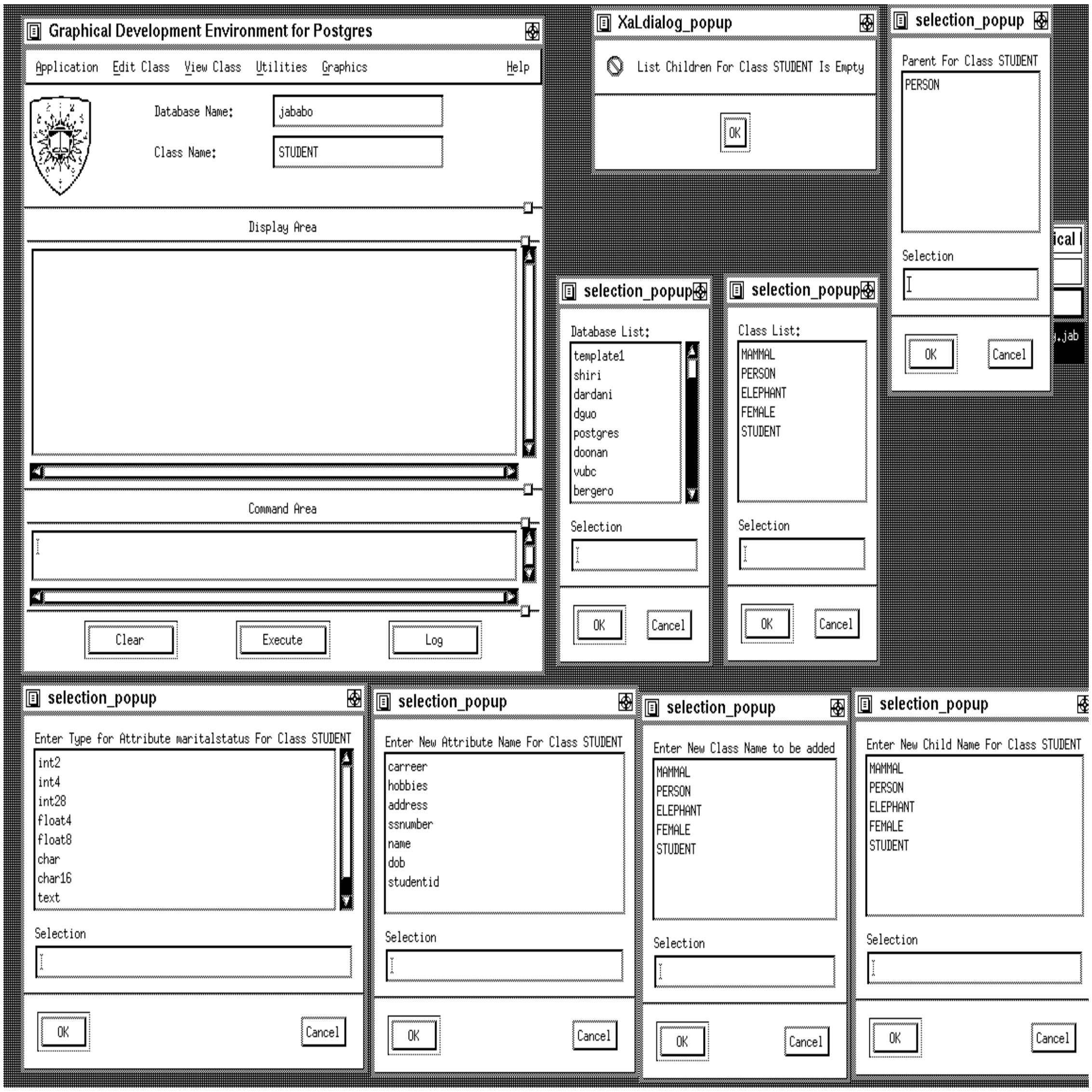

Figure 2.5: Screen Layout of the Application

is not from the list, then a warning message will be displayed stating that the selection must be from the list. After the warning message is acknowledged, GDEP destroys the selection popup window and goes back to the previous state prior to selecting this pushbutton. Otherwise, if the selected item is from the list 'Class List', then GDEP sets the root class for the application to the selected class name by the user. Also, it sets the value of the field 'Class Name' in the header window to the value of the selected class name through the selection popup window, then destroys the selection popup window.

3. Create Database

   This pulldown submenu item is used to create a new Postgres database. If the user selects this option and the value of XaSystemAdministratorPrivilege (discussed in section 5.2.2) is set to zero, then a warning message will be displayed stating that the user does not have the privileges of creating a Postgres database. After the warning message is acknowledged, GDEP destroys the selection popup window, and goes back to the previous state prior to selecting this pushbutton. Otherwise, if the value of XaSystemAdministratorPrivilege is set to one, then a selection popup window will appear with a list of the already existing database names. The user must enter a new database name that does not exist in the list 'Database List', as shown in figure 2.5. If the user selects an item that already exists in the list, then a warning message will be displayed stating that the selection must not be from the list. After the warning message is acknowledged, GDEP destroys the selection popup window and goes back to the previous state prior to selecting this pushbutton. Else, if the selected item is not from the list 'Database List', then GDEP creates a new Postgres database with the name entered by the user, and displays a message informing the user that a new database has been created. After this popup message is acknowledged, GDEP destroys the selection popup window.

4. Quit

   This pulldown submenu item is used to terminate the GDEP application session and return control to the Unix prompt. When the user selects this option, GDEP closes the connection with the backend process of the database. Then, it opens a file called 'username.sta', described in section 5.2.2, and writes the status of the current GDEP application to it. Also, it opens a file called 'username.log', described in section 5.2.2, and writes the log of the current GDEP application to it. Finally, it destroys the GDEP main window and all the windows created by the GDEP application as well, before returning control to the Unix prompt.

2.5.2 Edit Class Main Menu Bar Item

Clicking on this menu bar item reveals a pulldown submenu with eight items. All submenu items of this menu bar item except the 'Create New Class' and 'Help' options require that the root class in the GDEP header window be set to an existing class in the database. If the root class is not set and the user selects an option other than 'Create New Class' or 'Help', then GDEP will respond with a warning message telling the user that the option cannot be performed because the root class is not set. The list of the submenu items and their functionalities is given below.

1. Create New Class

   This pulldown submenu item is used to create a new class in the current database. When the user selects this option, a selection popup window will appear with a list of the available classes in the current database. The user must enter a new class name that does not exist in the list 'Class List', as shown in figure 2.5. If the user selects an item that already exists in the list, then a warning message will be displayed stating that the selection must not be from the list. After the warning message is acknowledged, GDEP destroys the selection popup window and goes back to the previous state prior to selecting this pushbutton. Else, if the selected item is not from the list 'Class List', then GDEP creates a new class in the current database with the name entered by the user; it then asks the user to enter one instance variable name, then its data type, for the new class. Finally, it displays a message informing the user that a new class has been created. After the notice popup message is acknowledged, GDEP destroys the selection popup and notice popup windows.

2. Destroy Root Class

   This pulldown submenu item is used to destroy the root class in the current database. When the user selects this option, a confirmation popup window will appear asking whether the user really wants to delete the root class. If the user responds by pushing the 'No' pushbutton, then GDEP destroys the confirmation popup window and goes back to the previous state prior to selecting this pushbutton. Otherwise, if the user responds by pushing the 'Yes' pushbutton, then GDEP creates a class hierarchy tree consisting of the selected class and all its subclasses, destroys all the classes in the class hierarchy tree starting from the leaf nodes to the root node, sets the root class of the GDEP application and the value of the field 'Class Name' in the header window to an empty string, and finally creates a notice popup window informing the user that the root class has been destroyed successfully. After the notice message is acknowledged, GDEP destroys the notice popup window.

   For example, if the current database contains the class hierarchy represented in figure 2.6, and a GDEP user sets the root class to be PERSON and selects the Destroy Root Class pushbutton, then the current database will consist of the classes represented in figure 2.7. Comparing figures 2.6 and 2.7, we realize that classes PERSON, STUDENT and FEMALE have been removed from the class MAMMAL hierarchy tree and destroyed by executing the Destroy Root Class PERSON. Figure 2.6 is a class tree hierarchy that we built in database jababo. The rectangular boxes contain the class names, and the oval boxes at the leaf nodes of the tree contains the instances of the classes.

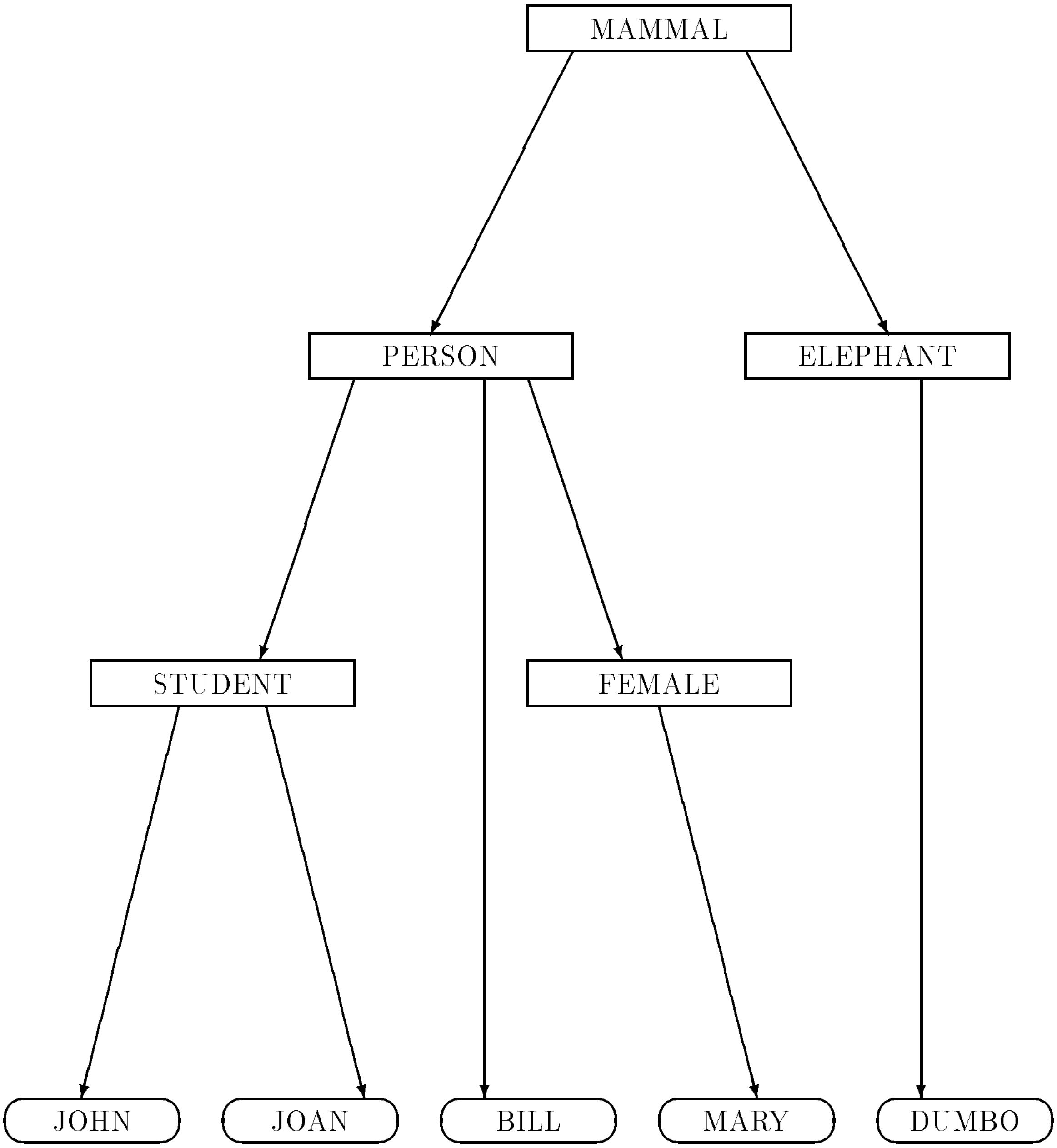

Figure 2.6: MAMMAL Class Hierarchy Tree

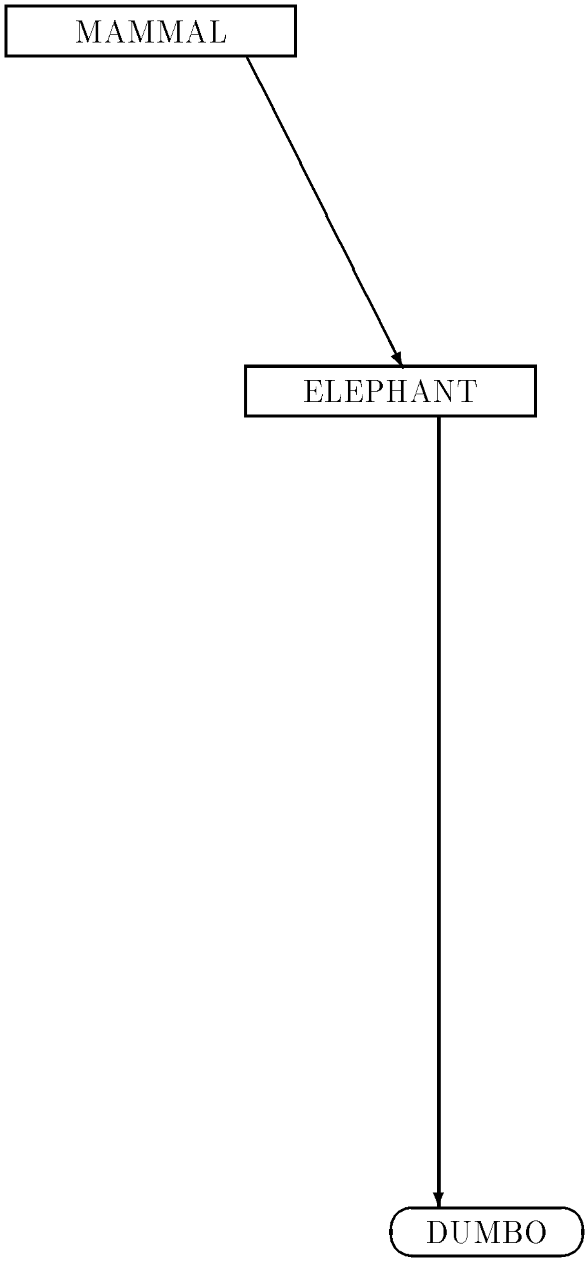

Figure 2.7: MAMMAL Class Hierarchy Tree after Detroy Root Class PERSON

3. Add New Child

   This pulldown submenu item is used to create a new class, which is a child 'subclass' of the root class. After this pushbutton is selected, a selection popup window will appear with a list of the already existing classes in the current database. The user must enter a new class name that does not exist in the list 'Class List', as shown in figure 2.5. If the user selects an item that already exists in the list, then a warning message will be displayed stating that the selection must not be from the list. After the warning message is acknowledged, GDEP destroys the selection popup window and goes back to the previous state prior to selecting this pushbutton. Else, if the selected item is not from the list 'Class List', then GDEP creates a new class with the name entered by the user. The newly-created class inherits all the properties and behavior of the root class. Finally, GDEP displays a message informing the user that a new class has been created successfully. After the notice popup message is acknowledged, GDEP destroys the notice popup window.

   For example, if the current database contains the class hierarchy represented in figure 2.7, and a GDEP user sets the root class to be MAMMAL and selects the Add New Child pushbutton, then the current database will consist of the classes represented in figure 2.8. Comparing figures 2.7 and 2.8, we realize that class MAMMAL has two subclasses, STUDENT and ELEPHANT, after executing the Add New Child operation, while it had only one subclass, ELEPHANT, before. After the Add New Child operation, we added instances JOHN and JOAN to class STUDENT.

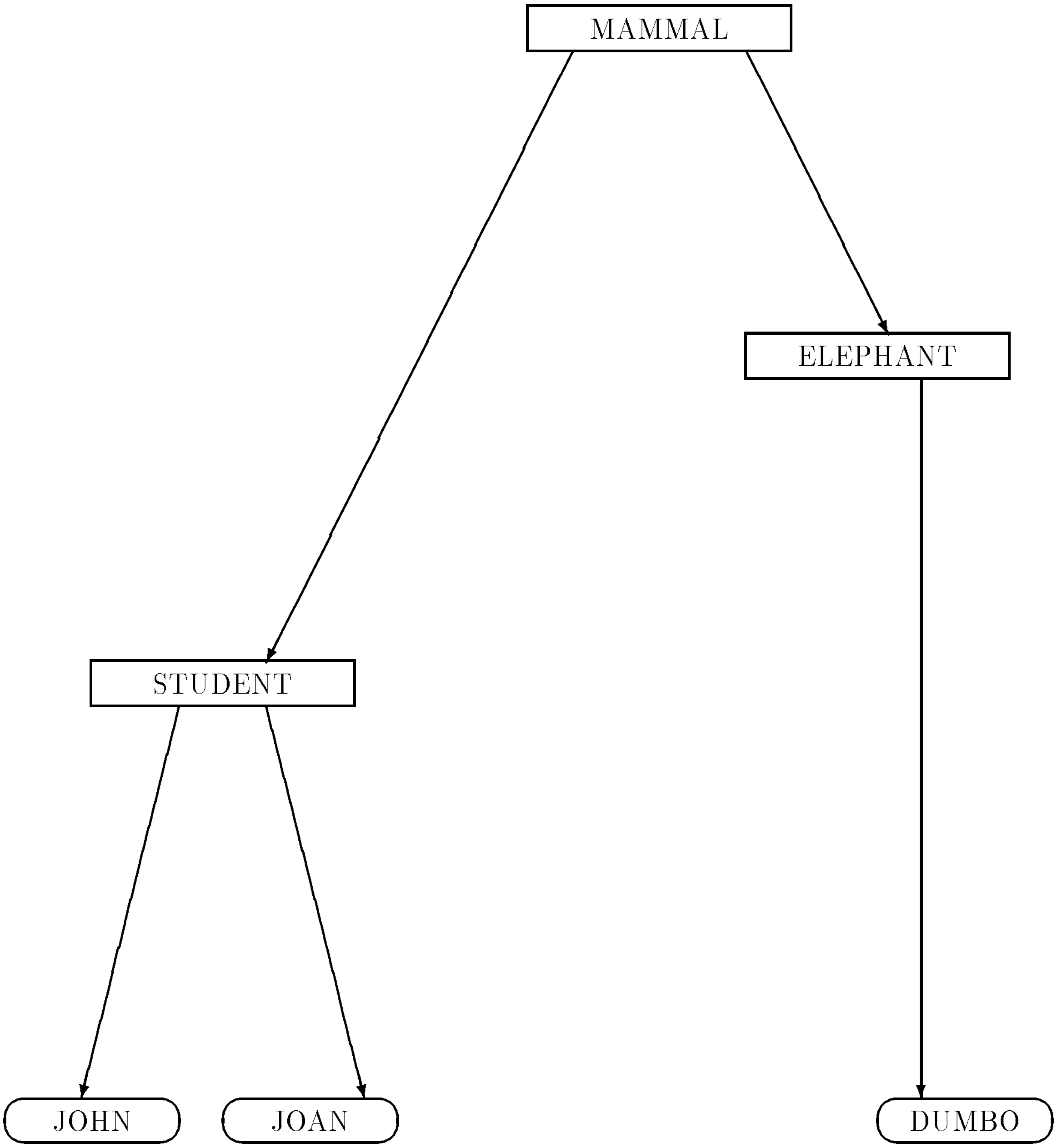

Figure 2.8: MAMMAL Class Hierarchy Tree after Add New Child Class STUDENT

4. Add Attribute Class

   This pulldown submenu item is used to add a new instance variable to the root class. After this pushbutton is selected, a selection popup window will appear with a list of the already existing instance variables for the root class. The user must enter a new instance variable name that does not exist in the list 'Attribute Names', as shown at the buttom of figure 2.5. If the user selects an item that already exists in the list, then a warning message will be displayed stating that the selection must not be from the list. After the warning message is acknowledged, GDEP destroys the selection popup window and goes back to the previous state prior to selecting this pushbutton. Else, if the selected item is not from the list 'Attribute Names', then GDEP creates a new selection popup window with a list of the data types that the new instance variable can have. The user must select one of the data type names in the list 'Type for Attribute', as shown in figure 2.5. If the user selects an item that is not from the list, then a warning message will be displayed stating that the selection must be from the list. After the warning message is acknowledged, GDEP destroys the selection popup window and goes back to the previous state prior to selecting this pushbutton. Else, if the selected item is from the list 'Type for Attribute', then GDEP exports to a temporary external text file the root class hierarchy tree with all the instances of the classes in the tree, then reimports the tree with the new instance variable added to the root class of the tree from the exported temporary external text file. Finally, GDEP displays a message that the new instance variable name has been added successfully to the root class. After the notice popup message is acknowledged, GDEP destroys the notice popup window.

5. Add Instance Class

   This pulldown submenu item is used to add a new instance to the root class. When the user selects this option, a popup window will be displayed on the screen that consists of four parts: the header name 'Add Instance Class', the Class Name that contains the name of the class that we are adding the instance to, the instance variable names and their data values, and the menu bar option at the buttom. The instance variable part is split into two major parts. As shown in figure 2.9, there are some instance variable fields at the top of this display which contain data fields of twenty characters length; these fields are of any data types except text. And, at the buttom of the display there are scrollable text fields of two rows each, representing text data type fields. In the menu bar, we see three pushbuttons labeled: Clear, Add and Cancel. The 'Clear' pushbutton is used to clear all the fields of the current instance, by. assigning an empty string value to all the fields of the current instance. The 'Add' pushbutton is used to add a new instance to the current class, with the values in the instance variable fields assigned as data values to the new added instance. The new added instance could be a duplicate or a modified instance of the previous added instance, or a new instance. After the instance is added to the database, GDEP displays a notice popup window informing the user that the new instance is added successfully. After the message is acknowledged, GDEP destroys the notice popup

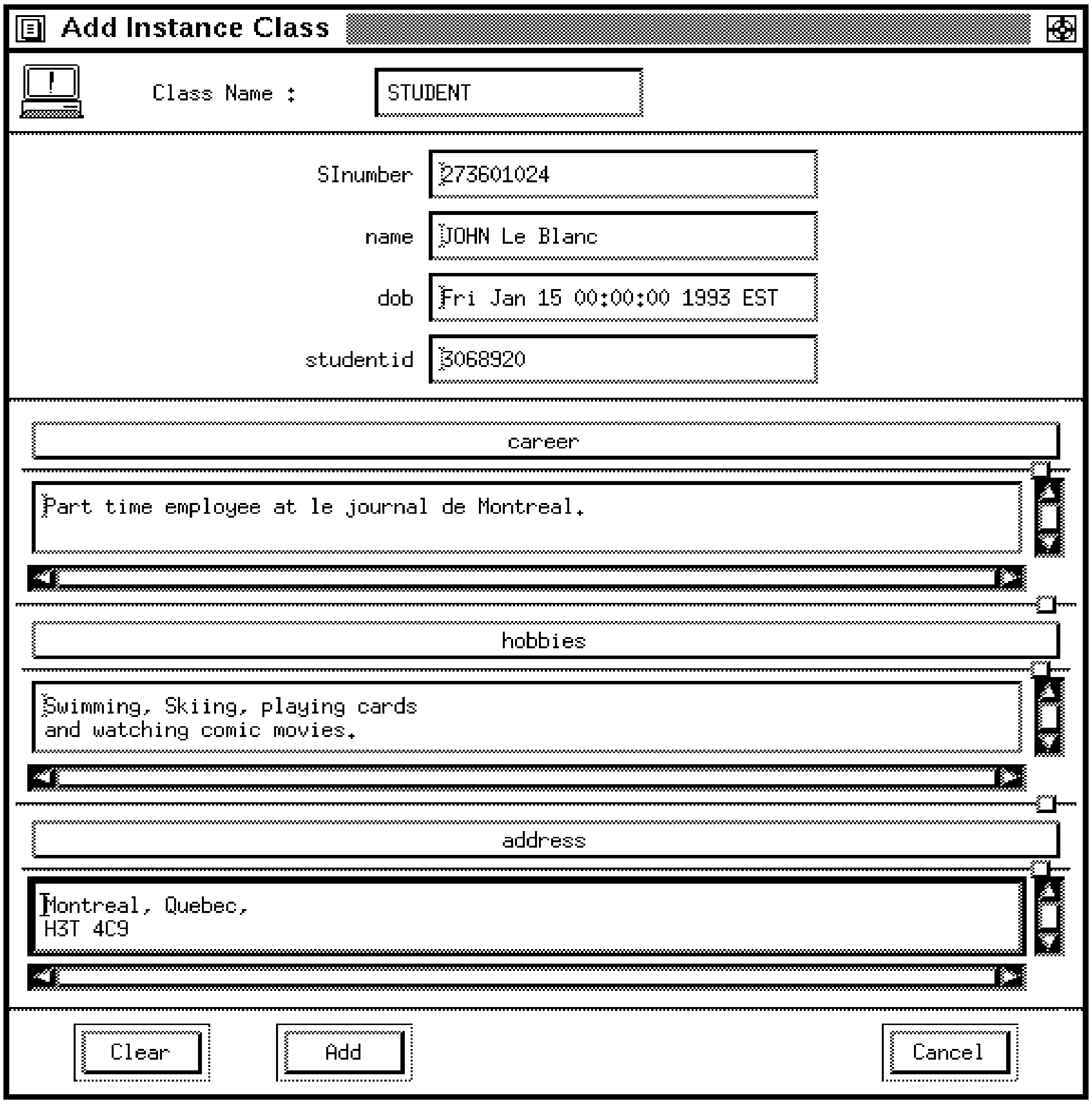

Figure 2.9: Add instance to class STUDENT

window and the user can continue to add new instances or perform any other task. The 'Cancel' pushbutton destroys the 'Add Instance Class' window.

6. Edit Instance Class

   This pulldown submenu item is used to edit existing instances in the root class. When the user selects this option, a popup window will be displayed on the screen that consists of four parts: the header name 'Edit Instance Class', the Class Name that contains the name of the class whose instances we are editing, the instance variable name and their data values, and the menu bar option at the buttom. The instance variable part is split into two major parts. As shown in figure 2.10, there are some instance variable fields at the top of this display which contains data fields of twenty characters length; these fields are of any data types other than text data type. And at the buttom of the display there are scrollable text fields of two rows each to represent the text data type fields. For example, in figure 2.10, the fields SInumber, name, dob and studentid are regular fields of twenty characters length each; the fields career, hobbies and address are text fields that have a scrollable text area each. In the menu bar, we see five pushbuttons labeled: Previous, Next, Update, Delete and Cancel. The 'Previous' pushbutton is used to show the instance which comes right before the current instance of the current class in the database. If the current instance is the first instance for the current class in the database, then the 'Previous' pushbutton will disappear. The 'Next' pushbutton is used to show the instance which comes right after the current instance for the current class in the database. If the current instance is the last instance for the current class in the database, then the 'Next' pushbutton will disappear. The 'Update' pushbutton is used to replace the data values for the current instance of the current class with the values in the instance variable fields assigned as data values to the current instance. After the instance is modified in the database, GDEP displays a notice popup window informing the user that the current instance is modified successfully. After the message is acknowledged, GDEP destroys the notice popup window and the user can continue editing new instances or performing any other task. The 'Delete' pushbutton is used to delete the current instance of the current class in the database. When the user selects this pushbutton, GDEP displays a confirmation popup window asking whether the user really wants to delete the current instance. If the user responds 'No', then the confirmation popup window will be destroyed and the user can continue editing the current instance. Else, if the user responds by 'Yes', then GDEP destroys the current instance in the database and informs the user through a notice popup window with a message stating that the current instance has been destroyed successfully. Then the next instance will be displayed as the current instance if it exists. Otherwise, the first instance will be displayed as the current instance if there is at least one instance for the current class. Otherwise, GDEP destroys the 'Edit Instance Class' window and displays a notice popup window stating that there are no more instances in the database for the current class. The 'Cancel' pushbutton destroys the 'Edit Instance Class' window.

7. Change Parent Class

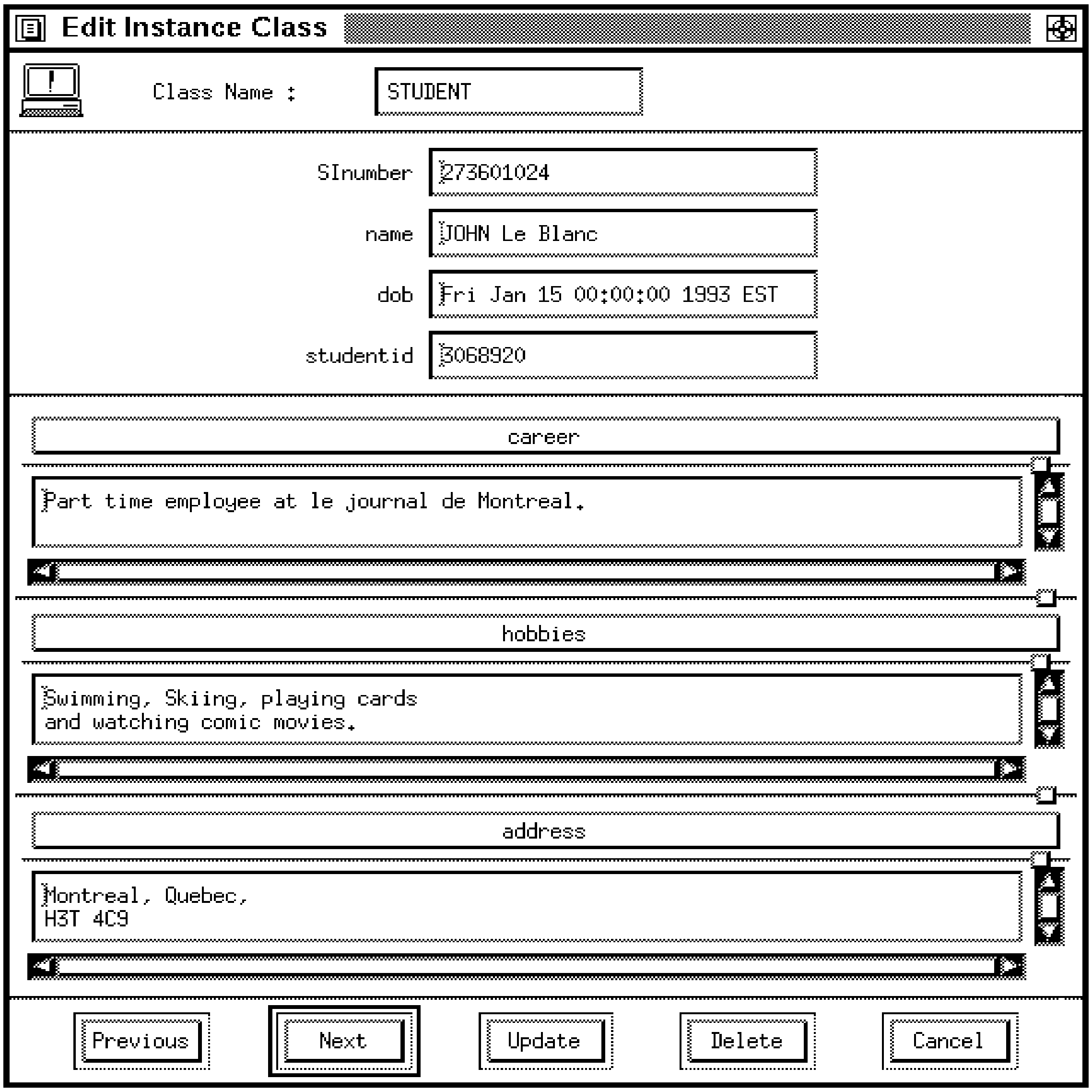

Figure 2.10: Edit instance of class STUDENT

This pulldown submenu item is used to set the superclass 'Parent Class' of the selected class to a specific existing class. When the user selects this option, a selection popup window will appear with a list of the valid parent classes in the current database. A valid parent class is a class that exists in the current database, which is not a subclass of the root class. If the user selects an item that is not from the list, then a warning message will be displayed stating that the selection must be from the list. After the warning message is acknowledged, GDEP destroys the popup window and goes back to the previous state prior to selecting this pushbutton. Else, if the selected item is from the list 'Class List', then GDEP checks if there is a conflict in data type between the root class and the new parent class, then displays a notice popup window about the conflict, stating that the root class cannot be a subclass of the new parent class. After the notice message is acknowledged, GDEP destroys the notice popup window and goes back to the previous state prior to selecting this pushbutton. Otherwise, if there is no conflict, then GDEP exports the whole hierarchy rooted at the current class with all its instances to a temporary external file, destroys the root class tree, imports the tree with the root class inheriting the new parent class, erases the external file and finally displays a notice popup window stating that the operation has been done successfully. After the notice message is acknowledged, GDEP destroys the notice popup window.

8. Help

   This pulldown submenu item is used to provide help on the 'Edit Class' menu item of the main menu bar. When the user selects this option, a help popup window will be displayed on the screen providing the GDEP user with the appropriate information on how to use any item in the pulldown submenu of the menu item 'Edit Class'. The help popup window will be destroyed after being acknowledged by the GDEP user. Figure 2.11 shows the Help Edit Popup Dialog Box. All other help Popup dialog boxes offered by GDEP have the same window layout as the edit help Popup dialog box, but differ in the contexts of the help text.

2.5.3 View Class Main Menu Bar Item

This menu bar item consists of six pulldown submenu items. All submenu items of this menu bar item except 'Help' require that the root class in the GDEP header window be set to an existing class in the database. If the root class is not set and the user select an item other than 'Help', then GDEP will respond with a warning message telling the user that the root class name is empty. The submenu items and their functionalities are:

1. View Class Definition

   This pulldown submenu item is used to show the instance variable names and their data types for the root class. When the user selects this option, as shown in figure 2.12, a popup window will be displayed on the screen that consists of four parts: the header name 'View

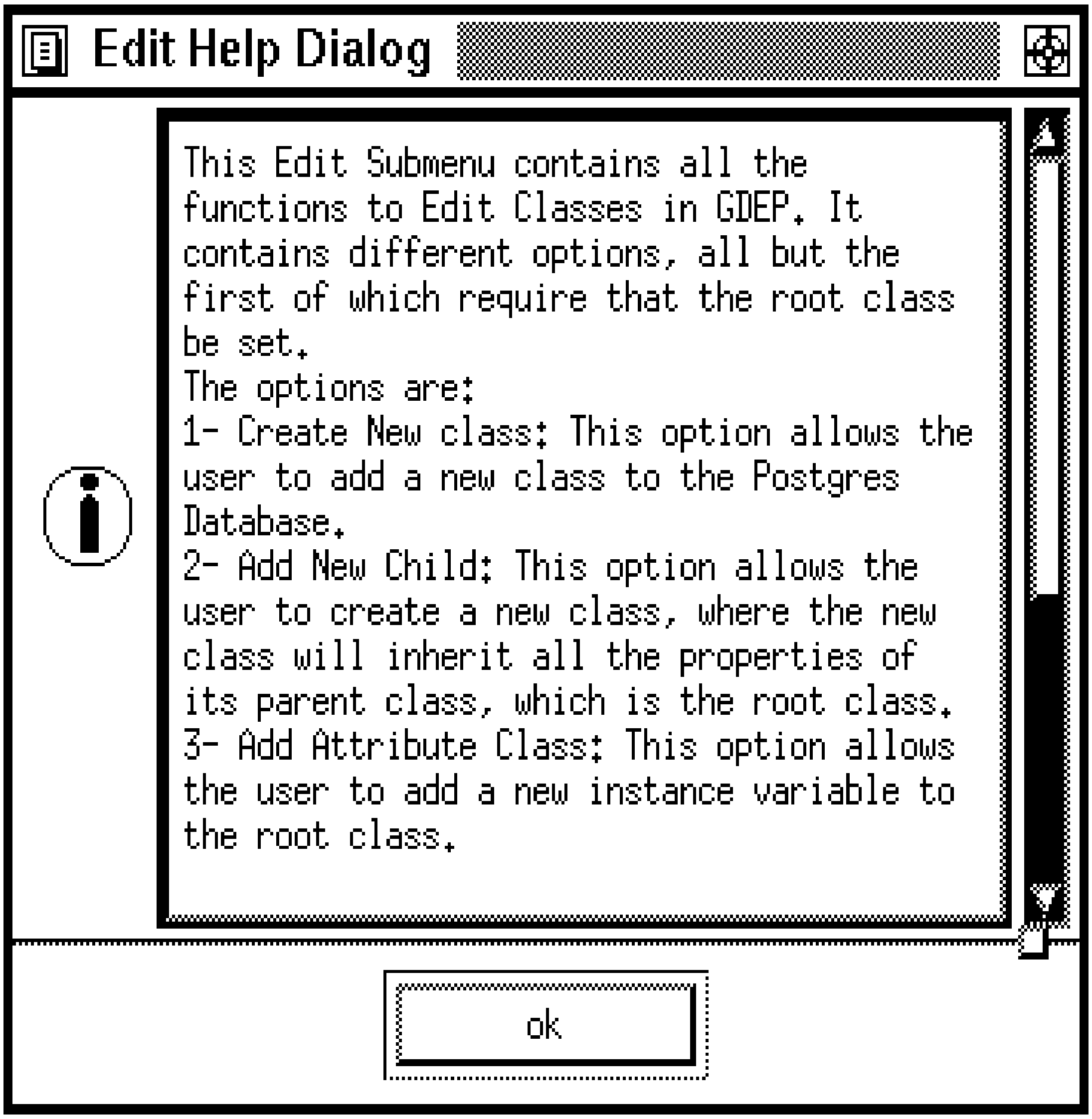

Figure 2.11: Edit Help Popup Dialog Box

Class Definition', the Class Name that contains the name of the class whose definition is being viewed, the instance variable names and their data types, and the menu bar option. In the menu bar there is only one pushbutton, labeled 'Cancel', which destroys the 'View Class Definition' window.

2. View Parent of Class

   This pulldown submenu item is used to show the Parent Class list of the root class. When the user selects this option, as shown in the upper right corner of figure 2.5, a popup window will be displayed on the screen that contains the list of the names of the parent classes for the root class, if the root class has at least one superclass. If the user selects any of the parent classes, then the selected parent class name will be the root class for the GDEP application, and its name will replace the child class name in the Class Name field of the header area in the main window. Otherwise, if the root class does not have any parent class, then a warning dialog popup window will be displayed informing the user that the root class does not have any parent class. The notice dialog window will be destroyed after being acknowledged by the user.

3. View Children of Class

   This pulldown submenu item is used to show the Children Class list for the root class. When the user selects this option, as shown in the middle row of figure 2.5, a popup window will be displayed on the screen that contains the list of the names of the children classes 'subclasses' for the root class if the root class has at least one child class 'subclass'. If the user selects any of the children classes, then the selected child class name will be the root class for the GDEP application, and its name will replace the parent class name in the Class Name field of the header area in the main window. Otherwise, if the root class does not have any subclass, then a warning dialog popup window will be displayed informing the user that the root class does not have any subclass. The notice dialog window will be destroyed after being acknowledged by the user.

4. View Class Instances

   This pulldown submenu item is used to view existing instances in the root class. As shown in figure 2.2, The instance variable names and their data values are displayed in the 'Display Area' region of the main window, in the form of a table. Every instance is displayed in one row. Each instance variable value is displayed within eighteen characters. If the data value exceeds eighteen characters, then a star is displayed at the eighteenth character location, to inform the user that there are more characters for that particular field.

5. View Class Instances by Tuple

   This pulldown submenu item is used to view existing instances in the root class one tuple at a time. It has almost the same functionalities as the submenu item 'Edit Instance Class' of the menu bar item 'Edit Class', described earlier in section 2.5.1. The only difference is that

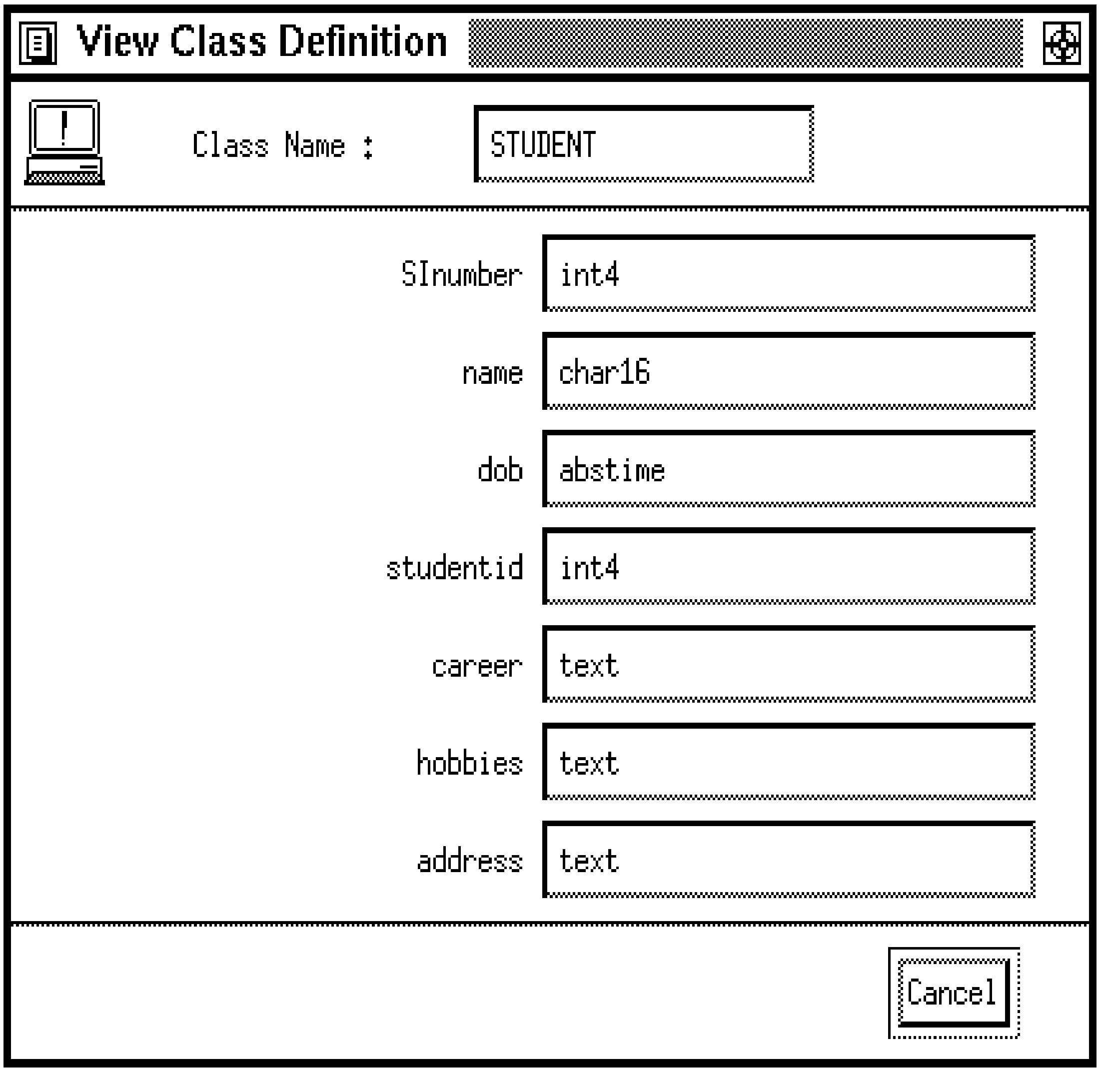

Figure 2.12: Class Description of class STUDENT

this option does not have the 'Update' or 'Delete' pushbuttons in the menu bar section of the 'View Instance' window.

6. Help

   This pulldown submenu item is used to provide help on the 'View Class' menu item of the main menubar. When the user selects this option, a help popup window will be displayed on the screen, providing the GDEP user with the appropriate information as to how to use any item in the pulldown submenu of the menu item 'View Class'. The help popup window will be destroyed after being acknowledged by the GDEP user.

2.5.4 Utilities Main Menu Bar Item

This menubar item consists of ten pulldown submenu items:

1. Hide Command Window

   This pulldown submenu item is used to hide the Command Area in the main window of the GDEP application. When the user selects this option, the command area will disappear and the display area will be extended to occupy the previous space used by the command area. Then the 'Hide Command Window' option will be replaced by 'Show Command Window' in the submenu of the 'Utilities' menu bar item. The size of the main window will not be affected by this operation.

2. Show Command Window

   This pulldown submenu item is used to show the Command Area in the main window of the GDEP application. When the user selects this option, the command area will be redisplayed and the display area will be decreased to allow the command area window to be displayed. Then the 'Show Command Window' option will be replaced by the 'Hide Command Window' in the submenu of the 'Utilities' menu bar item. The size of the main window will not be affected by this operation.

3. Refresh Windows

   This pulldown submenu item is used to reset and refresh the display view of the main window. When the user selects this option, GDEP destroys the main window, then recreates it again. The state of GDEP before the execution of this operation and after is the same; only the view display of the subwindows of the main window might be changed after the execution of this operation.

4. Clear Display Area

   This pulldown submenu item is used to clear and reset the display area of the main window. When the user selects this option, the value of the display area text will be an empty string, regardless of what contents it had before the execution of this operation.

5. Export Class Definition

   This pulldown submenu item is used to export the class definition of the root class as a flat class to an external .cdf file in Postquel format. For example, if we select this pulldown submenu item and the root class is set to FULL-TIME of the class hierarchy tree shown in figure 3.3, then GDEP creates a file called FULL-TIME.cdf that contains the postquel command 'create FULL-TIME (name = char16, salary = float)'. The exported text file can be imported later by any Postgres tool that has the ability to do so. After the class is exported, GDEP displays a notice popup window informing the user of the name of the external file to which the class definition has been exported. The notice popup window will be destroyed after being acknowledged by the GDEP user.

6. Export Class Def Inherit

   This pulldown submenu item is used to export the class definition in an inheritance form of the root class. It exports the root class name, the instance variable names and their types that are not inherited from the parent class of the root class, and finally the word 'inherits' and a list of the parent classes of the root class if the root class has at least one parent class, as a class that inherits its parent class, to an external .cdi file in Postquel format. For example, if we select this pulldown submenu item and the root class is set to FULL-TIME of the class hierarchy tree shown in figure 3.3, then GDEP creates a file called FULL-TIME.cdi that contains the postquel command 'create FULL-TIME (salary = float) inherits (EMPLOYEE)'. The exported text file can be imported later by any Postgres tool that has the ability to do so. After the class is exported, GDEP displays a notice popup window informing the user of the name of the external file to which the class has been exported. The notice popup window will be destroyed after being acknowledged by the GDEP user.

7. Export Class Instances

   This pulldown submenu item is used to export the class instances of the root class to an external .cin file in Postquel format. The exported text file can be imported later by any Postgres tool that has the ability to do so. After the class is exported, GDEP displays a notice popup window informing the user of the name of the external file to which the class has been exported. The notice popup window will be destroyed after being acknowledged by the GDEP user.

8. Export Class Tree

   This pulldown submenu item is used to export the class hierarchy tree and the instances of the classes in the tree of the selected class. It exports each class's definition and then its instances, before starting with another class. It exports the class hierarchy tree, from root to leaf nodes, to an external filename in a Postquel format. The exported text file can be imported later by any Postgres tool that has the ability to import Postquel files. After the class is exported, GDEP displays a notice popup window informing the user of the name of the external file to which the class has been exported. The notice popup window will be destroyed after being acknowledged by the GDEP user.

9. Import Postquel File

   This pulldown submenu item is used to import an external text file into the current database. When the user selects this option, GDEP creates a selection file popup window allowing the user to select an external file from a list of existing files. After a selection is made, GDEP tries to import the selected file to the current database. If it succeeds, then a notice popup window will be displayed, informing the user that the selected external file has been imported to the current database. Otherwise, a notice popup window will be displayed, informing the user of the selected external file could not be opened to be imported to the current database. The notice popup window will be destroyed after being acknowledged by the GDEP user. Failure to open a file is not the only possible hindrance to importing; for example, the file might have lost its Postquel format. This issue is not dealt with here, as it is peripheral of our present field of interest.

   GDEP imports the selected file by opening the file in a read mode and reading its contents line by line, issues a Postquel command to the backend process of the database, consisting of each line read from the imported file, and finally closes the selected file. GDEP does not parse the contents of the lines read from the selected file; it just assumes that they are Postquel commands. It is the backend process that parses the Postquel commands issued by GDEP, and decides whether to accept them or not. In all cases, a feedback message is displayed in the main window by Postgres, informing the user of the status of each command issued to the database by GDEP.

10. Help

   This pulldown submenu item is used to provide help on the 'Utilities' menu item of the main menu bar. When the user selects this option, a help popup window will be displayed on the screen, providing the GDEP user with the appropriate information as to how to use any item in the pulldown submenu of the menu item 'Utilities'. The help popup window will be destroyed after being acknowledged by GDEP user.

2.5.5 Help Main Menu Bar Item

This menu bar item consists of three pulldown submenu items:

1. Help Application

   This pulldown submenu item is used to provide help on the whole GDEP application. When the user selects this option, a help popup window will be displayed on the screen, providing the GDEP user with the appropriate information as to the purpose and functionalities of GDEP as a Postgres tool. The help popup window will be destroyed after being acknowledged by GDEP user.

2. Help Display Area

    This pulldown submenu item is used to provide help on the purpose of the 'Display Area' window and its role within the GDEP application. When the user selects this option, a help popup window will be displayed on the screen providing the GDEP user with the appropriate information as to how to use and clear the 'Display Area' window. The help popup window will be destroyed after being acknowledged by the GDEP user.

3. Help Command Area

    This pulldown submenu item is used to provide help on the purpose of the 'Command Area' window, and its role within the GDEP application. When the user selects this option, a help popup window will be displayed on the screen providing the GDEP user with the appropriate information as to how to use the 'Display Area' window, and the functionalities of the three pushbutton items in its menu bar. The help popup window will be destroyed after being acknowledged by the GDEP user.

Chapter 3

GDEP ENHANCEMENTS TO POSTGRES

In this chapter, we will describe the two enhancements that we added to Postgres: Add attribute to a class, and Change parents of a class. These added features are not supported yet by any existing Postgres tools. In the following sections, we will describe the effects of using each enhancement to the database. Also, we will discuss the importance of these enhancements with respect to Object Orientation.

3.1 Add Attribute to a Class

Inheritance allows us to reuse the behavior of a class in the definition of new classes. Subclasses of a class inherit the operations of their parent class, and may add new operations and new instance variables. Figure 3.1 describes EMPLOYEE by an inheritance hierarchy of classes (representing behaviors). The class EMPLOYEE has FULL-TIME and PART-TIME as its subclasses. The class FULL-TIME has EMPLOYEE as its superclass. The instances JOHN, JOAN, BILL, MARY and LARRY each have a unique base class. For more details on inheritance refer to [Wegner90].

According to [Desai], with respect to Class-Hierarchy and the IS-A relationship, an instance of a class belongs to its superclasses and it supersuperclass because of the IS-A relationship.

In Postgres, if we add a new instance variable to a class, the new instance variable does not propagate to its subclasses. For example, in figure 3.1, if we add a new instance variable called address to class EMPLOYEE, then instance BILL will have the instance variable address, as shown in figure 3.2. But all other instances, such as JOHN, JOAN, MARY and LARRY, will not have the instance variable address after the add attribute address operation in Postgres. Since those instances are instances of classes FULL-TIME and PART-TIME, and since FULL-TIME and PART-TIME are EMPLOYEEs according to the class hierarchy, then class EMPLOYEE is a supersuperclass of those instances. As we just mentioned, an instance of a class belongs to its superclasses and its supersuperclass; so these instances should belong to class EMPLOYEE, and contain the instance variable address.

To overcome this problem and keep the database hierarchy as was mentioned in [Desai], GDEP modified the Postquel add attribute to a class function, and propagates the new added attribute to all instances of the modified class. The GDEP user does not have a choice not to propagate the new added instance variable to its subclasses. The value of the new instance variable in the already defined instances will be assigned to NULL. For example, if a user adds an instance variable 'address' to class EMPLOYEE through GDEP, then all instances will have the instance variable address, as shown in figure 3.3. Also, all new added instances to classes FULL-TIME and PART-TIME will have the instance variable address, which would not be true with the regular add attribute in Postgres. The value of address for instance LARRY is NULL, because it has not been modified after we added the new instance variable address to class EMPLOYEE.

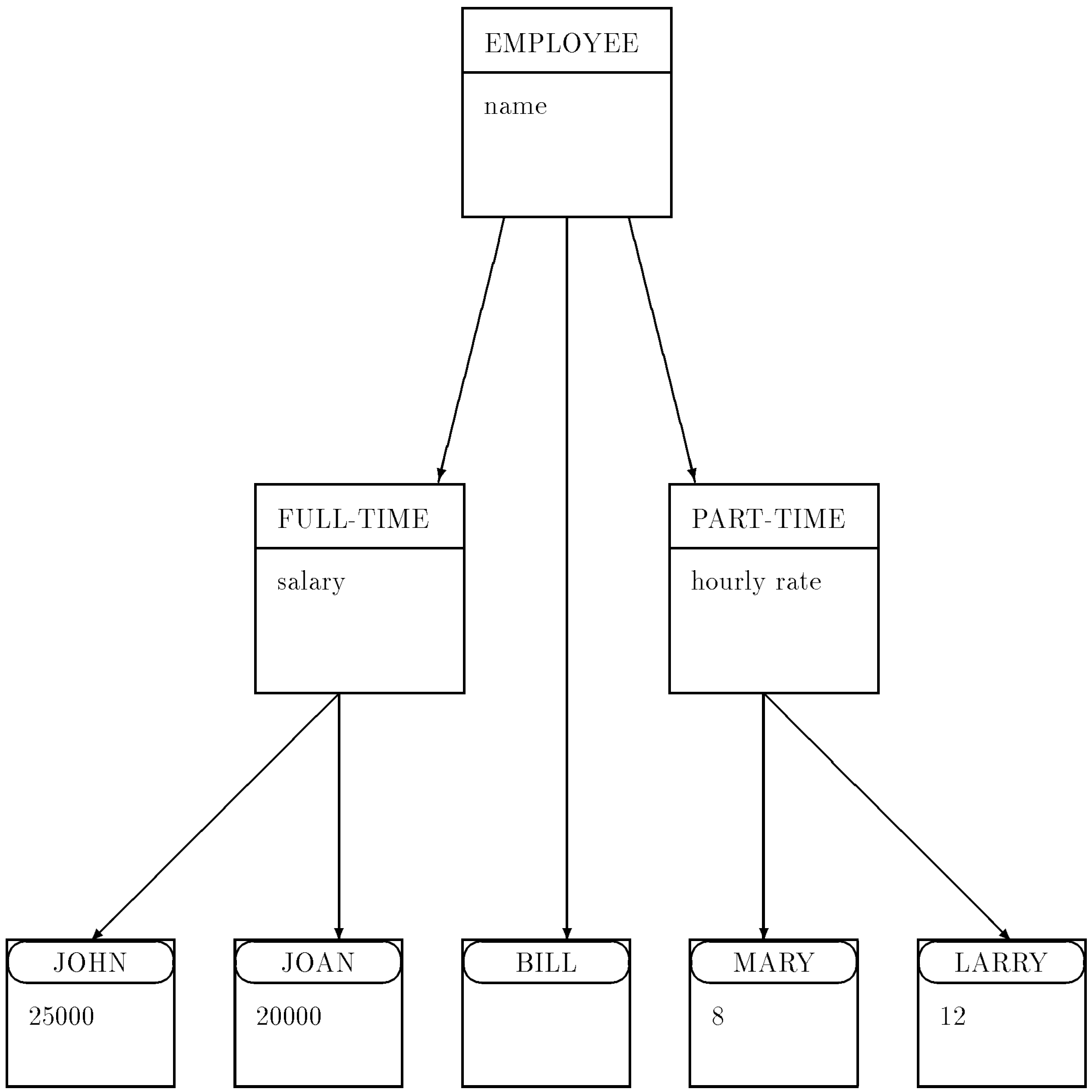

Figure 3.1: Employee class inheritance graph.

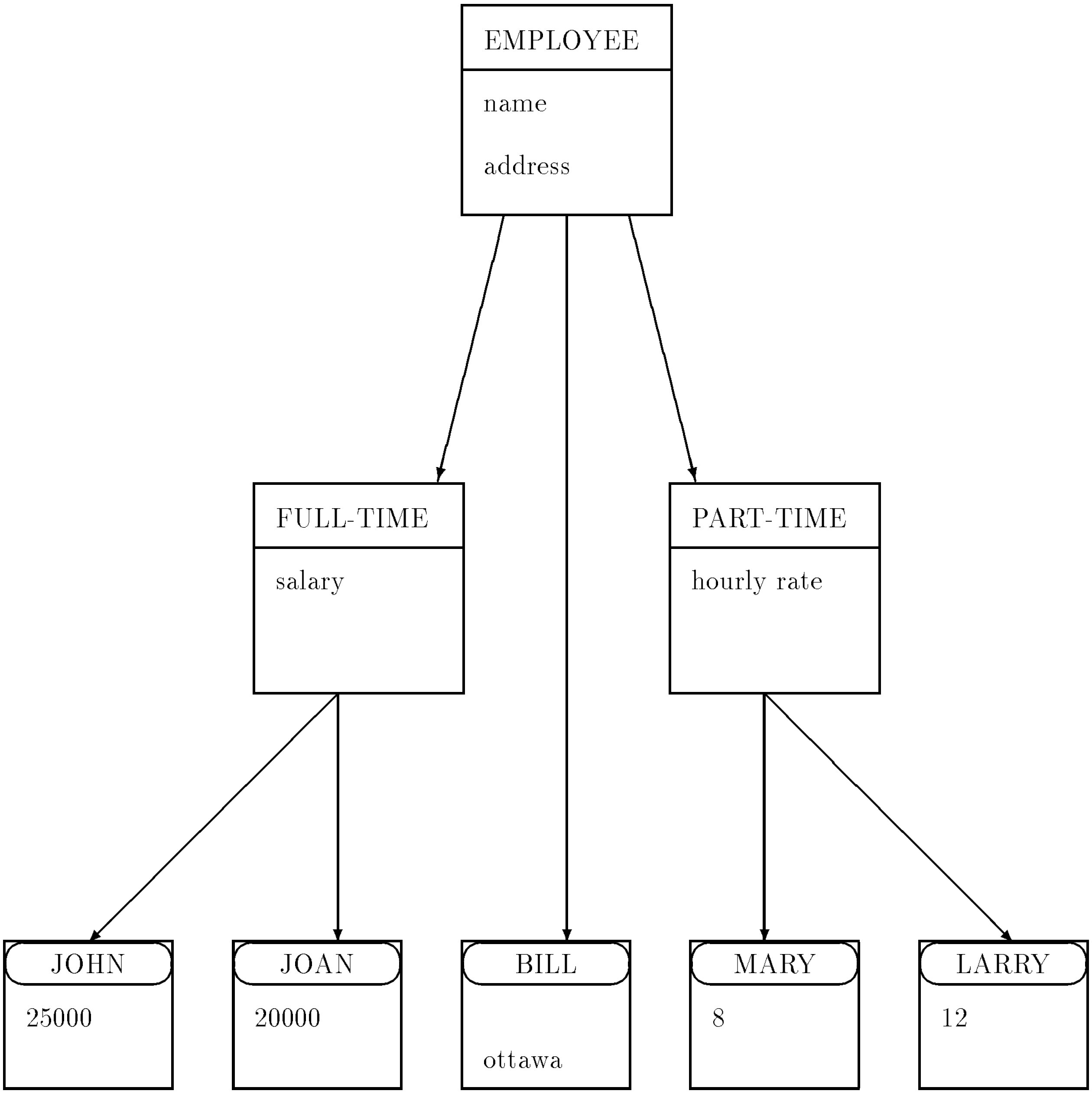

Figure 3.2: Employee class inheritance graph after Add Attribute address in monitor.

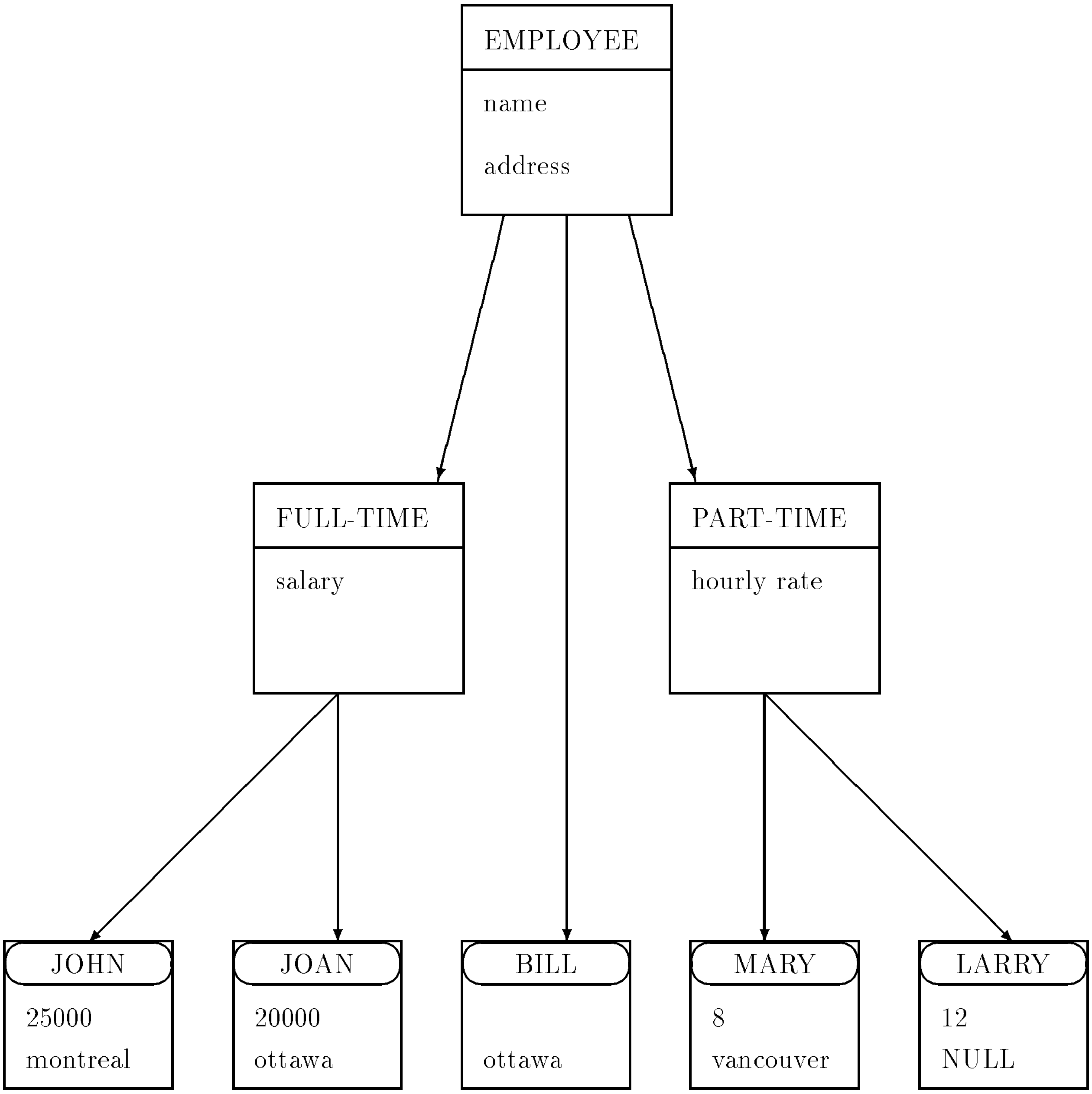

Figure 3.3: Employee class inheritance graph after Add Attribute address in GDEP.

3.2 Change Parents of a Class

In some applications, Postgres users may need to change the parents of an existing class that inherits properties from a particular class, to a new parent class; Postgres does not provide this feature. To make this feature available to Postgres users, GDEP added this option. In this section, we will discuss the change class parents enhancement feature that GDEP offers to Postgres users, in addition to the effects of executing this option to the database.

A user can select the 'change parent class' option whether the current class has any parent class or not. In both cases, GDEP considers the selected class as a flat class, and adds the inheritance feature of the selected class to inherit the new parent class. But in the first case, where a class already has a parent class, GDEP takes the selected class from the children list of the previous parents, so that the previous parents will no longer have the selected class as a child class. After this operation is completed, the selected class will have as parents the new parent class only. Also, the new parent class will have as children the selected class, in addition to the previous children classes it had before the operation was performed. All instances in the database that inherit new instance variables from the new parent will assign a null value to those new instance variables at the execution of this operation. When performing this operation, two exception cases might occur: circularity in the database, and compatibility between instance variable names and types between the selected class and the new parent class. In the next section, 3.2.1, we discuss how GDEP handles these two exception cases.

To clarify this operation, take as example a database that consists of four classes: STUDENT, EMPLOYEE, GRAD STUD and COOP STUD. As shown in figure 3.4, class STUDENT has two instance variables, name and stud id; it has two subclasses, GRAD STUD and COOP STUD; and it has one direct instance, BILL, in addition to other instances of its subclasses. Class COOP STUD has parent class STUDENT, and two instance variables, name and stud id, that it inherits from its parent class STUDENT. It also has two instances, MARY and LARRY, each with its own stud id. Also, we have class EMPLOYEE that has two instance variables, name and salary, but no child or instance. If through GDEP we set the root class to be COOP STUD, then we could select the 'change parents' option and select EMPLOYEE to be the new parent class. Then the state of the database would change, and after modifying the instance variable values of class COOP STUD wil be as in fig 3.5. EMPLOYEE will have COOP STUD as a subclass. COOP STUD will have three instance variables: stud id and name that it previously inherited from class STUDENT before this operation, and instance variable salary that it inherited from its new parent class EMPLOYEE. Finally, MARY and LARRY instances will have the instance variables name, salary and stud id. If we compare figure 3.4 and figure 3.5, we realize that these two instances had two instance variables name and student id before the operation. But after this operation they have a new instance variable salary added to the previous instance variables. The default value for the new instance variable inherited from the new parent class is NULL. As we see in figure 3.5, the value of the inherited instance variable salary for LARRY is NULL. Also, MARY has a 24,000 salary after modifying the value of the inherited instance variable. In section 3.2.2 below, we defend against type clash between inherited fields of the same name. Complete rigour would demand in addition a check for meaning clash; but this is beyond the scope of this project.

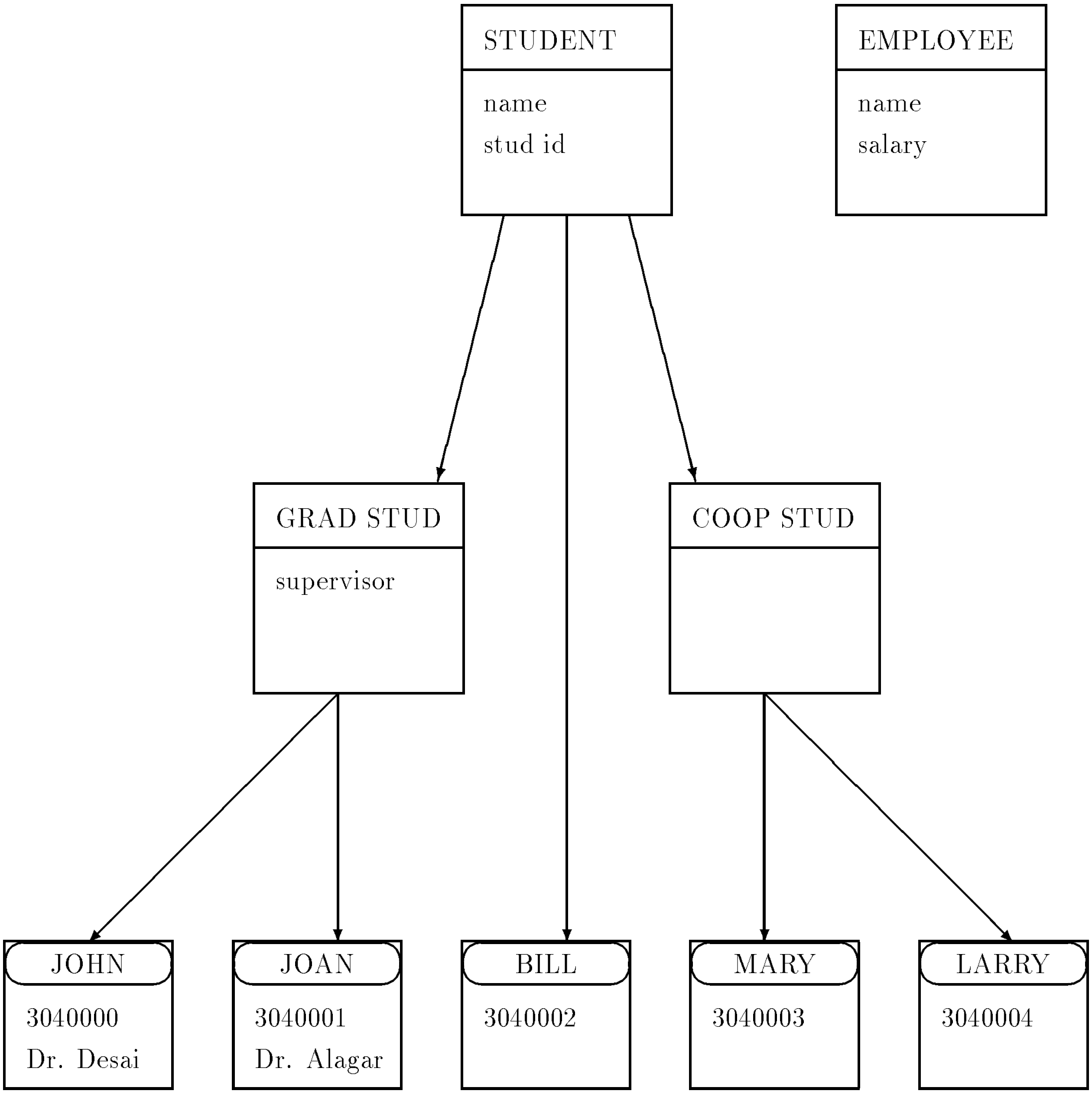

Figure 3.4: STUDENT and Employee class inheritance graph.

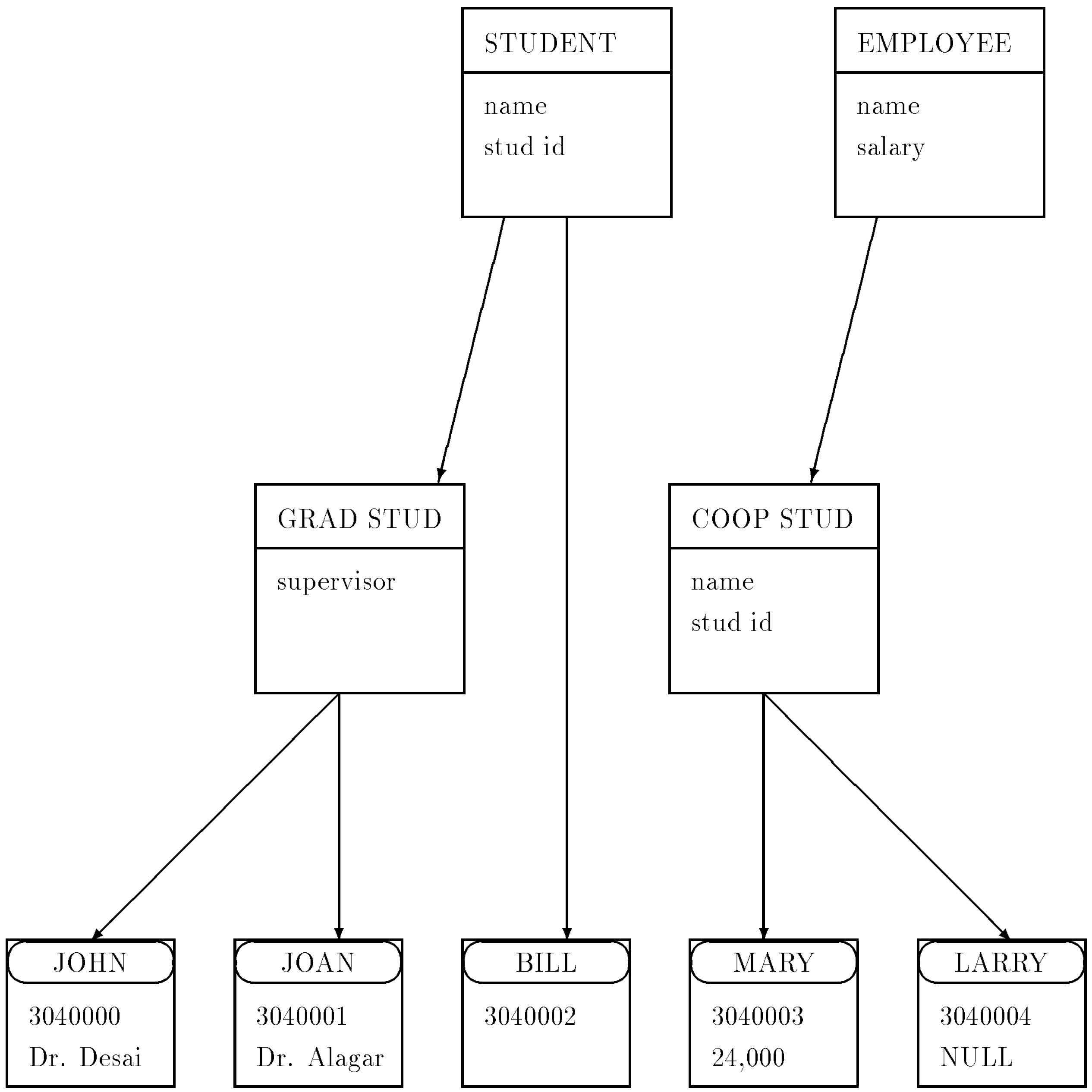

Figure 3.5: COOP STUD class with new Parent Class EMPLOYEE.

3.2.1 Exceptions

When the user selects this option, GDEP will handle some exceptions, which are listed and discussed in this subsection. The exceptions are:

1. **Circularity:**

   If the new parent class is a child of the selected class, then there is a circularity. In this case, GDEP will warn the user and give an appropriate message, then cancel the operation. If the user still likes to change the parent class of the same class, the same operation must be performed again. But this time, a parent class should be selected that is not a subclass of the current class, because GDEP does not allow Postgres users to create circularity in their database through any of its submenu options.

2. **Conflict in instance variables' name:**

   In case the selected class and the new parent class have the same instance variable name, then two possibilities might occur: the two instance variables may have the same or different domains. In the first case, if the two instance variables have the same domain, then everything will be normal and those instance variables will have the same values as they had before the execution of this operation. But if the two instance variables have different domains, then if the old instance variable domain is a subset of number type[1] and the new instance variable domain is a subset of a string type[2], then GDEP will transform those instances into the new types. Otherwise, those instance variables will have null values.

[1] Integer and float types are subsets of a number type.

[2] Char16 and text types are subsets of a string type.

Chapter 4

SAMPLE APPLICATION

In this chapter, we discuss the sample application (Mosaic) that we added to the standard features of GDEP. The purpose of this sample application is to show GDEP users how they can extend GDEP to handle specific tasks such as searching a document and launching an application.

4.1 Overview of NCSA Mosaic

NCSA Mosaic [Mosaic] is an Internet information browser and World Wide Web client. It was developed at the National Center for Supercomputing Applications at the University of Illinois, Urbana-Champaign. Mosaic comes in three flavours:

1. Mosaic for the X Window System
2. Mosaic for Microsoft Windows
3. Mosaic for Macintosh

It provides different types of information to the World Wide Web:

1. Serve Information to the Web

    Mosaic provides a hypertext interface to the global Internet. Hypertext is text which contains highlighted links, called hyperlinks or anchors, to other texts. Each highlighted phrase (in color or underlined) is a hyperlink to another document or information resource somewhere on the Net. A single click with the left mouse button on any highlighted phrase is used to follow the link, which means that Mosaic will retrieve the document associated with the selected hyperlink and display it.

The Mosaic client communicates with HTTP servers. HTTP is the HyperText Transfer Protocol of the WWW (World Wide Web). Mosaic can also communicate with more traditional Internet protocols such as FTP, Gopher, WAIS, NNTP, *etc.*

2. Create HTML Documents

   The hypertext documents viewed with Mosaic are written in HTML (HyperText Markup Language), which is a subset of SGML (Standard Generalized Markup Language). Among the many formatting features, HTML allows Mosaic to display inline images. (In fact, an inlined image can serve as a hyperlink, just as a word or phrase can).

3. Provide Complex Features

   Mosaic also features unlimited multimedia capabilities. File types that Mosaic cannot handle internally, such as mpeg movies, sound files, Postscript documents, and JPEG images, are automatically sent to external viewers (or players).

For more details on the NCSA Mosaic package, the user can refer to [Mosaic].

4.2 Mosaic Sample Application in GDEP

We extended the standard features of GDEP by adding a new menu item called Mosaic, shown in figure 4.1, to the main menu bar described in section 2.5. This is helpful to GDEP users, because it can be used as an example of how they can extend GDEP to do specific tasks. In the next subsection, we describe the pulldown submenu items of the newly-added main menu bar item, Mosaic, and their functionalities.

4.2.1 Mosaic Main Menu Bar Item

This menu bar item consists of three pulldown submenu items:

1. Launch Main Application

   This pulldown submenu item is used to launch a Mosaic application. Once the user selects this submenu item, the system command 'system /pkg/Mosaic/bin/Mosaic &' will be issued by GDEP, and will create a process in the background starting a Mosaic application, which causes the Mosaic main window application shown in figure 4.2 to pop up at the top left corner of the screen. Since the created process is in the background, it is totally independent of the GDEP application. In other word, quitting GDEP will not cause the Mosaic window to be destroyed.

2. Launch Selected Document

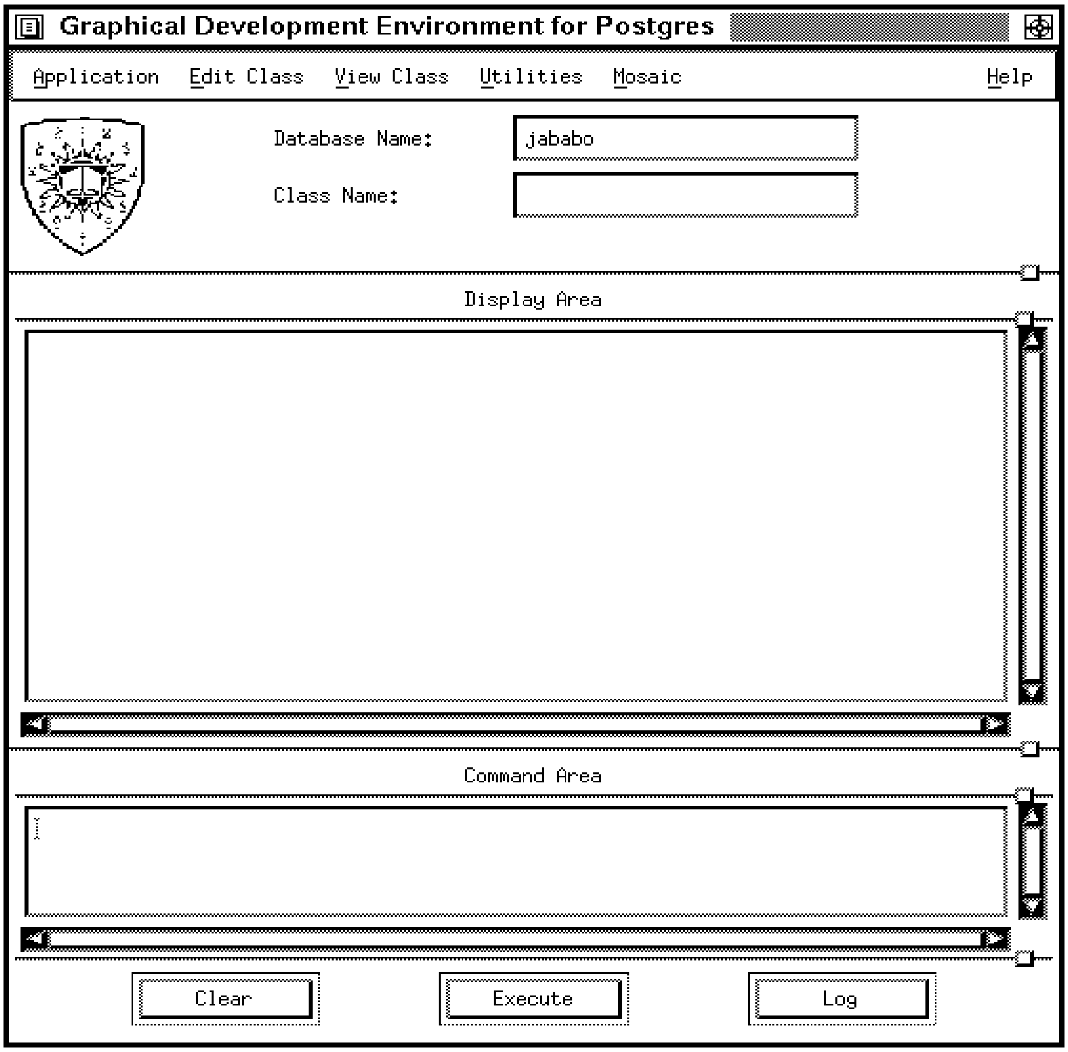

Figure 4.1: GDEP main window with Mosaic menubar option.

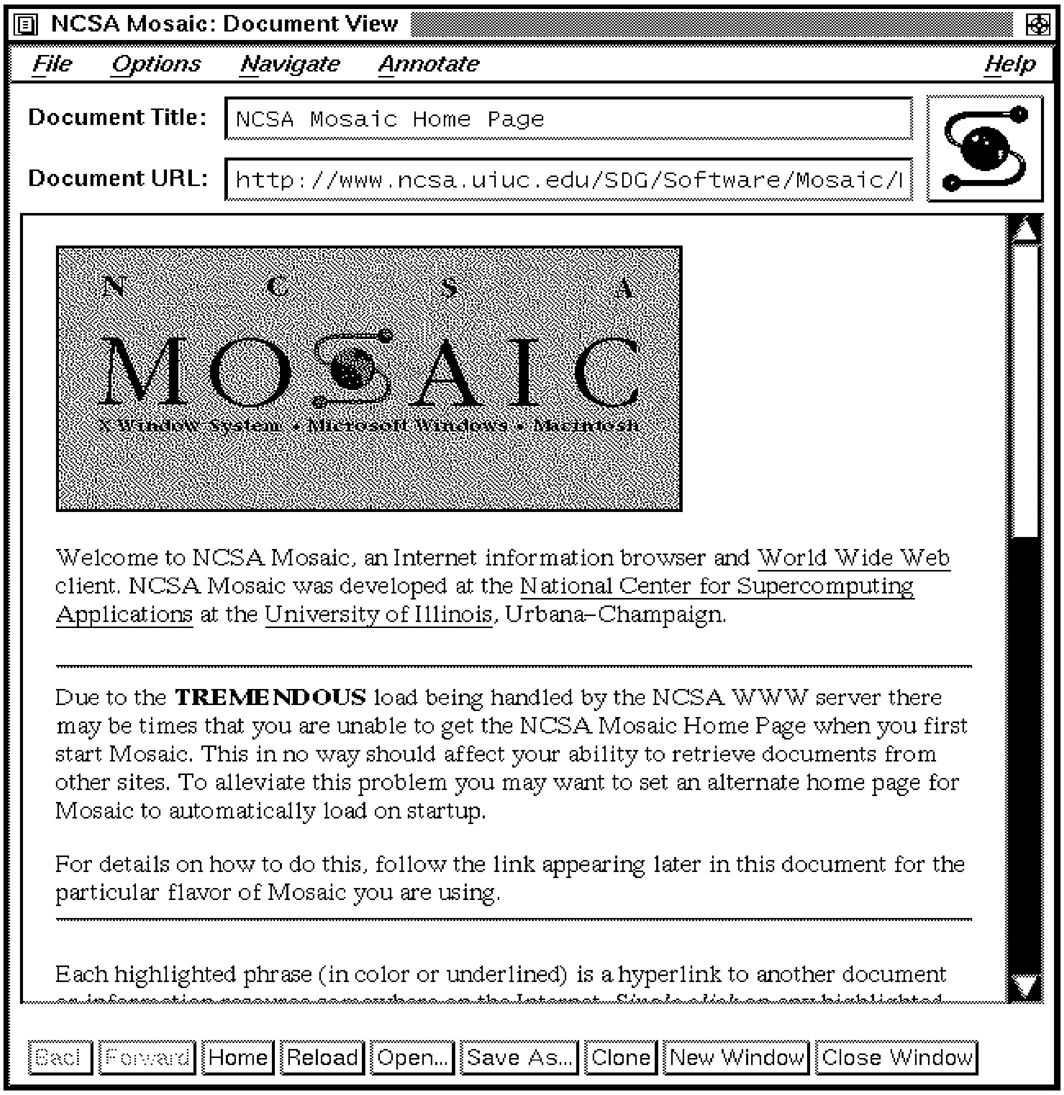

Figure 4.2: NCSA Mosaic: Main Application

This pulldown submenu item is used to launch a selected document using the Mosaic application. Once the user selects this submenu item, GDEP will check whether class Mosaic exists in the current database. If not, GDEP creates a popup dialog window warning the user that the current pulldown submenu selection item cannot be performed because class Mosaic does not exist in the current database. Else, if class Mosaic exists in the current database, then GDEP will create a popup dialog box requesting the user to enter the subject he is interested in searching documents about. After the subject is entered, GDEP will search in the Mosaic class of the current database for instances with a subject instance variable having a data value which is the same as the one entered by the user. If there are no instances in the database matching the subject entered by the user, then GDEP displays a notice popup message to that effect; if there are instances matching the user request, then GDEP will create a selection popup window containing the list of the available instances in the database. When the user selects any item in the list, the system command '/pkg/Mosaic/bin/Mosaic URL_VALUE_FIELD &' will be issued by GDEP. The URL_VALUE_FIELD is the data value of the URL instance variable field corresponding to the selected instance by the user. This system command will create a process in the background starting a Mosaic application which causes the Mosaic selected document view to pop up at the top left corner of the screen. Since the created process is in the background, it is totally independent of the GDEP application. In other words, quitting GDEP will not cause the NCSA Mosaic window to be destroyed.

For example, if the value of URL in the selected instance is 'http://www.cs.concordia.ca/local.html', then the system command '/pkg/Mosaic/bin/Mosaic http://www.cs.concordia.ca/local.html &' will be issued by GDEP, causing the window shown in figure 4.3 to pop up at the top left corner of the screen. If the URL value is an invalid value such as 'http://www.cs.concordia.ca/local/html', then the system command '/pkg/Mosaic/bin/Mosaic http://www.cs.concordia.ca/local/html &' will be issued by GDEP, causing the window shown in figure 4.4 to pop up at the top left corner of the screen.

3. Help

This pulldown submenu item is used to provide help on the 'Mosaic' menu item of the main menu bar. When the user selects this option, a help popup window will be displayed on the screen providing the GDEP user with the appropriate information as to how to use any item in the pulldown submenu of the menu item 'Mosaic'. The help popup window will be destroyed after being acknowledged by the GDEP user.

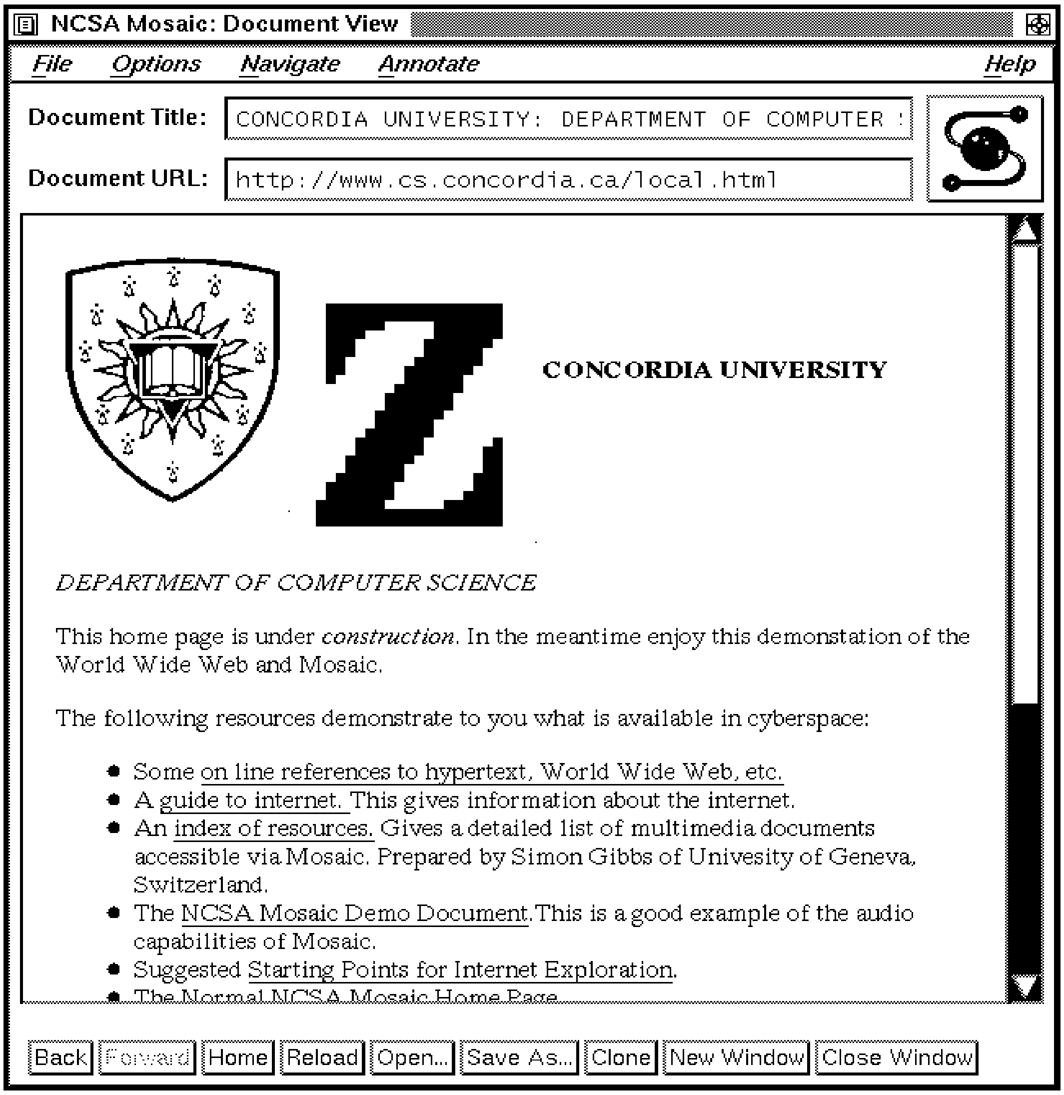

Figure 4.3: NCSA Mosaic: Document View

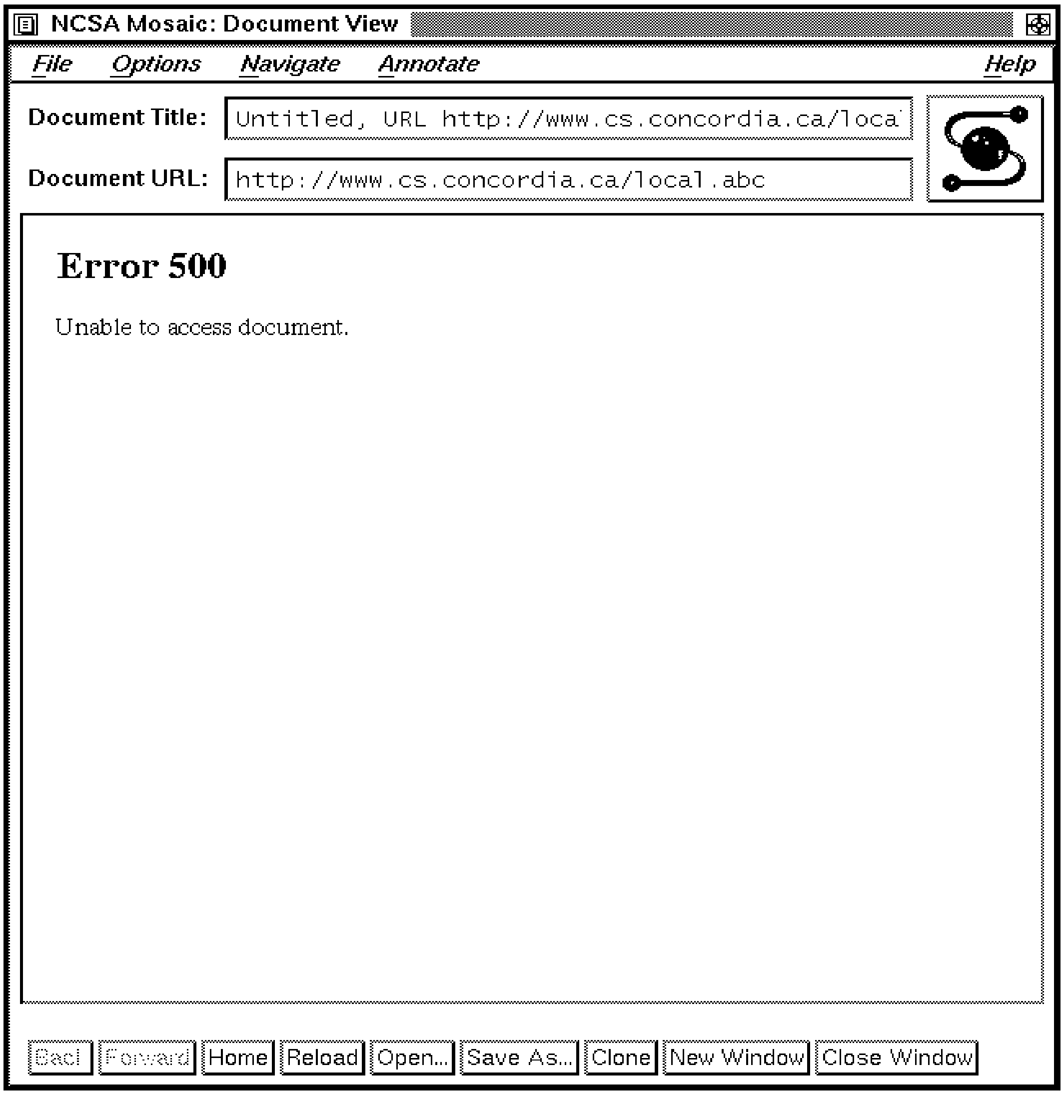

Figure 4.4: NCSA Mosaic: Invalid Document View

Chapter 5

INSTALLATION AND CUSTOMIZATION

This chapter is split into two sections: Installation and Customization. First, in the Installation section, we instruct GDEP users how to install GDEP, as well as the environment under which GDEP can be installed. Also, we talk about the documentation that is provided to GDEP users. Second, we describe to GDEP users the parameters that they are allowed to manipulate in order to customize GDEP to best suit their needs. Also, we describe how to manipulate those parameters to customize GDEP applications.

5.1 Installation

This section consists of two main subsections:

5.1.1 Hardware

GDEP can run on any machine that both Postgres and Motif Toolkit can run on. So far, Postgres has been tested by the Postgres development team on Sun Microsystems Sparc architecture machines running SunOS 4.1 and higher. Postgres is also supported on DECstation 3100's and 5000's running Ultrix 4.1 and higher. In order to use Postgres, your machine should have at least 8 megabytes of memory and you will require at least 45 megabytes of disk space to hold source, binaries, and user databases.

The Ultrix version requires a kernel which allows 4 megabytes of shared memory. Also, the original release of Ultrix 4.3 has a kernel bug that causes the operating system to hang when running Postgres.

Motif can run on any of the previously-mentioned machines. We tested GDEP on Sun Operating System version 4.0.1, and compiled it under the SUN C++ compiler. We believe that GDEP can run on any machine, assuming that Postgres and Motif are properly installed on the machine; but this is not guaranteed. Further testing for GDEP is needed on machines other than the one we tested it on, to prove our assumptions. For more information on how to install Postgres, refer to [Pg Rel 4.1]. Also, for more information on how to install Motif, refer to [Motif].

5.1.2 Software

GDEP is a package that consists of a Makefile, nine C files that contains the source code for the application, and a document file that contains the documentation for the application. To compile GDEP, you need to have the SUN C++ compiler, the Makefile and all the source code files under one directory. When the user types 'make', the Makefile will generate nine object files, one for every C file in the application. Finally, it will generate an executable file called gdep that is three hundred and twenty kilobytes in size. Typing gdep will start GDEP by displaying the GDEP main window. GDEP can be started from any directory, provided Postgres is well installed. No path needs to be set to run GDEP.

Postgres and Motif have a type clash in declaring a typedef String. To overcome this problem, you can modifiy file /usr/postgres/include/tmp/c.h on line 442 by replacing the following statement:

typedef char *String;

by the following block:

#ifndef XINCLUDE
typedef char *String;
#endif

As we can see in the Makefile, we have -DXINCLUDE at the end of the CFLAGS line; this will eliminate the redefinition of the String in Postgres. In addition to the standard C library, GDEP uses tmp/libpq.h and tmp/libpq-fe.h files from the /usr/postgres/lib Postgres library. Also, GDEP uses the Xm Motif, Xt X Toolkit Intrinsic, and X11 X libraries.

5.2 Customization

In this section, we describe the two files generated and used by GDEP to initialize the setup for the GDEP environment. The two files, Log file and Status file, are discussed in the following subsections.

5.2.1 Log File

When GDEP is started, a username is retrieved using getenv and used to set the log filename that resides on the directory when GDEP was executed. Then GDEP tries to read the log file, which is called username.log . If the file exists, then GDEP reads the Path to be used by the project from the log file. Otherwise GDEP uses the default Path, which is the current directory where GDEP was started.

The Path that is set by GDEP is used by the whole application for many purposes. The Path is used for:

1. Setting the Path to write the Log filename.
2. Setting the Path to read the Status filename.
3. Setting the Path to write the Status filename.
4. Setting the Path to Export class definitions.
5. Setting the Path to Export class definition inheritance.
6. Setting the Path to Export class instances.
7. Setting the Path to Export the class Tree.

For example, if we start GDEP from an account having username jababo, then GDEP looks for a file called jababo.log. If file jababo.log exists in the current directory, then GDEP opens the file and looks in the file for pattern MaProjectPath. If it finds the pattern MaProjectPath, then it sets the value of the global variable MaProjectPath to the value followed by pattern MaProjectPath in file jababo.log. Otherwise, if it does not find the pattern MaProjectPath, then it sets it to its default value './'. In the latter case, the path for all GDEP external files for read and write operations will be the current directory from which the user started GDEP.

5.2.2 Status File

After initializing the variables to their default values to set the environment, GDEP tries to open status filename username.sta, to set up the variables customized to meet user needs. If the status filename is opened in read mode successfully, then GDEP reads the values of its parameters from the status filename and sets its environment; otherwise, GDEP keeps the default values as the values to set the environment. When the user elects to quit GDEP, GDEP will open the status filename in write mode and write all the environment setup parameters to the file, so when the user runs GDEP later, GDEP will use the same environment the user was last using.

There are many parameters to customize GDEP; the user can go to the status filename and modify its parameter values as needed, or some of the parameters can be set directly through the GDEP window. All these parameters are listed and described below:

1. **XaSystemAdministratorPrivilege:** This is to set the system administrator privileges for the application. The user can only create a new database through GDEP if XaSystemAdministratorPrivilege is set to True. The default value for XaSystemAdministratorPrivilege is 0.

2. **XaMainWindowX:** This is to set the X position of the main window for GDEP. The user can move the GDEP window with the mouse, which will update the XaMainWindowX value in the application, or XaMainWindowX can be set in the status filename. The default value for XaMainWindowX is 0.

3. **XaMainWindowY:** This is to set the Y position of the main window for GDEP. The user can move the GDEP window with the mouse, which will update the XaMainWindowY value in the application, or XaMainWindowY can be set in the status filename. The default value for XaMainWindowY is 0.

4. **XaMainWindowWidth:** This is to set the Width value for the main window for GDEP. The user can reset it by modifying its value in the status filename. The default value for XaMainWindowWidth is 550.

5. **XaMainWindowHeight:** This is to set the Height value for the main window for GDEP. The user can reset it by modifying its value in the status filename. The default value for XaMainWindowHeight is 800.

6. **XaWorkAreaDisplayRows:** This is to set the Number of Rows value for the Display Area for GDEP. The user can reset it by modifying its value in the status filename. The default value for XaWorkAreaDisplayRows is 16.

7. **XaWorkAreaCommandRows:** This is to set the Number of Rows value for the Command Area for GDEP. The user can reset it by modifying its value in the status filename. The default value for XaWorkAreaCommandRows is 4.

8. **XaShowCommandAreaStatus:** This is to set the status value for the Command Area for GDEP. The user can reset it by modifying its value in the status filename. If XaShowCommandAreaStatus value is one, the command area is shown on the screen. Otherwise, the command area will be hidden by the display area. The default value for XaShowCommandAreaStatus is 1.

9. **XaHelpDialogBoxTextWidth:** This is to set the Width value for the Help Dialog Box for the GDEP application. The user can reset it by modifying its value in the status filename. The default value for XaHelpDialogBoxTextWidth is 300.

10. **XaHelpDialogBoxTextRows:** This is to set the Number of Rows value for the Help Dialog Box for the GDEP application. The user can reset it by modifying its value in the status filename. The default value for XaHelpDialogBox-TextRows is 6.

11. **XaHelpDialogBoxX:** This is to set the X position value for the Help Dialog Box for the GDEP application. The user can reset it by modifying its value in the status filename. The default value for XaHelpDialogBoxX is 0.

12. **XaHelpDialogBoxY:** This is to set the Y position value for the Help Dialog Box for the GDEP application. The user can reset it by modifying its value in the status filename. The default value for XaHelpDialogBoxY is 0.

13. **XaInstanceShellBoxX:** This is to set the X position value for the instance display popup shell for the GDEP application. It can be reset by modifying its value in the status filename. The default value for XaInstanceShellBoxX is 0.

14. **XaInstanceShellBoxY:** This is to set the Y position value for the instance display popup shell for the GDEP application. It can be reset by modifying its value in the status filename. The default value for XaInstanceShellBoxX is 0.

15. **XaInstanceAreaDisplayRows:** This is to set the number of rows value for the text fields in the instance display popup shell for the GDEP application. The user can reset it by modifying its value in the status filename. The default value for XaInstanceAreaDisplayRows is 2.

Chapter 6

CONCLUSION

In this chapter, we conclude our report with a description of the advantages of using GDEP for Postgres users. We suggest future work concerning GDEP, and describe what can be done to extend GDEP and make it a better Graphical Development Environment for Postgres users. Also, we propose a scheme which allows the sample application described in chapter 4 to be extended, in order to make GDEP more practical and more efficient for searching a document before launching an external application such as Mosaic.

6.1 Advantages of Using GDEP

Throughout this report, we discussed the different features and functionalities of GDEP. Comparing GDEP to other Postgres tools such as Alberi [Alberi], Geo [Geo] and Spog [Spog], we find that GDEP provides many features not provided by any other available Postgres tools. Some of the features help Postgres users to issue Postquel commands to the Postgres backend process without knowing the syntax of Postquel commands. We saw this in the functionalities of the pulldown submenu items of the main menu bar items in section 2.5. Other features introduce new functionalities to Postquel, and enhance its query language for Postgres databases. The two enhancements are:

1. Add Attribute to a Class, discussed in section 3.1.
2. Change Parents of a Class, discussed in section 3.2.

These features make GDEP a better Graphical Development Environment for Postgres users, allowing them to spend more time on designing their database instead of thinking about the syntax of Postquel.

6.2 Future Work

Many features can be added to GDEP to make it a better Graphical Development Environment for Postgres users. We chose two features that can be used as examples for future work:

1. Designing a Graphical Representation for the Class Hierarchy Tree.

2. Enhancing the sample application described in chapter 4.

In the following subsections, we describe in detail the two features that we think are important for Postgres users.

6.2.1 Graphical Representation for Class Hierarchy Tree

It would be helpful for Postgres Users if GDEP could provide a Graphical Representation Hierarchical Tree for a selected class, showing the class and all its descendants in a tree structure. For example, viewres tools installed under Sun Unix allow its users to see the class hierarchy tree for class objects. As we can see in figure 6.1, viewres shows us the root class object and all its descendants.

6.2.2 Enhancing the Sample Application

Postgres does not allow two classes to have the same instance, where the two classes are not related to each other by a parent-child class relationship. Because of that, we could not build a class hierarchy tree in a Postgres database to let us search a document URL from the database, and launch a Mosaic Application with the fetched URL. What we did in the sample application in section 4.2 is to search a document only by subject. We describe what can be done in order to enhance this feature and allow GDEP users to fetch a document by any stored characteristics of the document, such as title, author, *etc.* First, we need to create a flat class called Mosaic in the database, that contains all the instances needed to hold all the references needed to search for any document. The class Mosaic will have as instance variables: Author, Title, Subject, Call Number, ISBN, DocType, Publisher, Place and URL. Second, we need to create an abstract class called SearchDocument with one instance variable called ObjectId of oid data type, and another instance variable called name of char16 data type. Third, we need to create eight classes as children of class SearchDocument: Author, Title, Subject, Call Number, ISBN, DocType, Publisher and Place. We next need to design a procedure that reads all the instances in the Mosaic class, takes one instance at a time, takes every instance variable in that instance and appends its value and its Object Id as a new instance of the appropriate class. The appropriate class is the one with the same name as the instance variable name to be appended. After finishing this procedure, the user can fetch the SearchDocument tree by any desired type, take the object Ids of the fetched instances and look for their corresponding URL in the Mosaic Class.

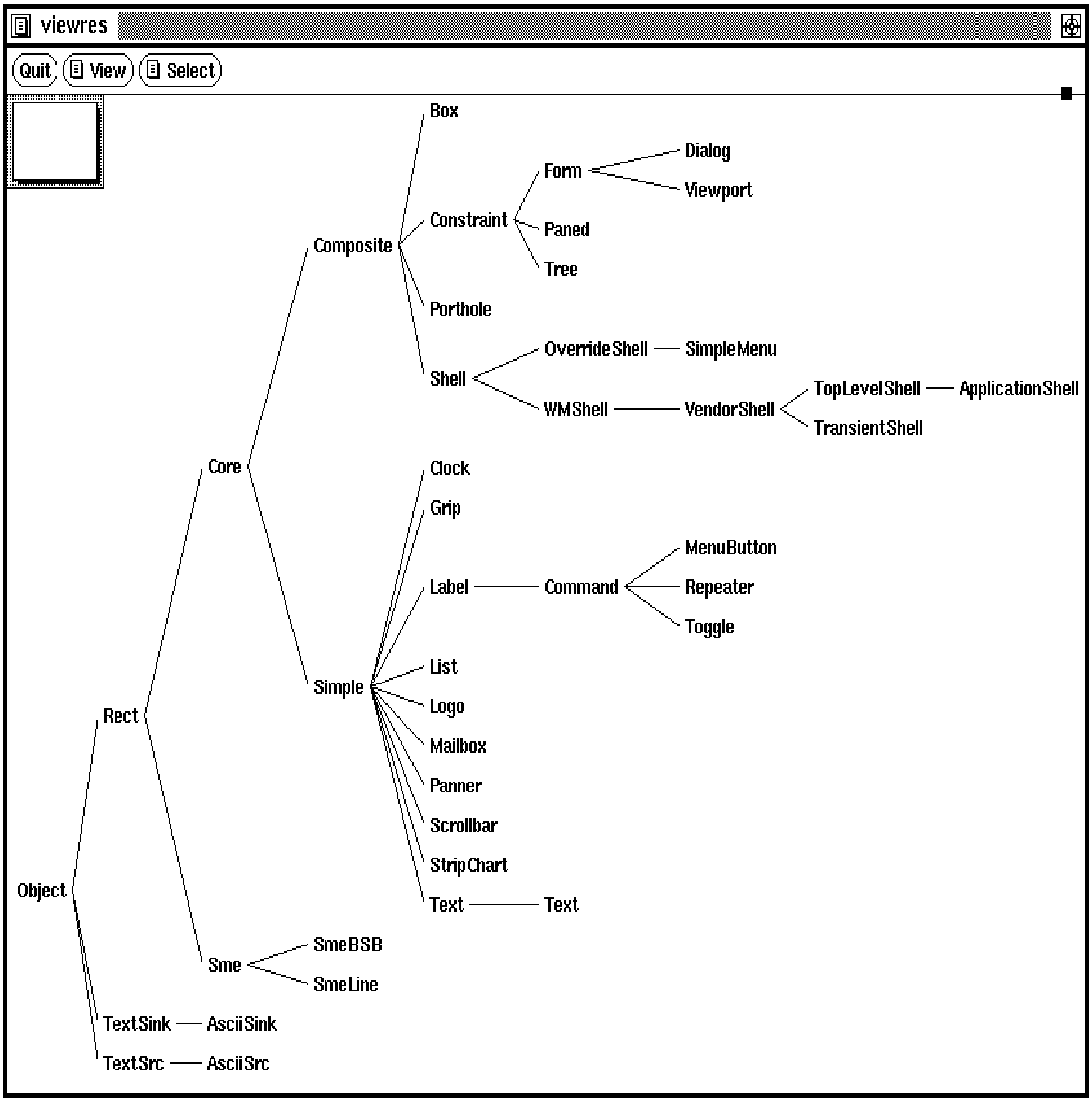

Figure 6.1: Graphical Hierarchy Tree For Class Object

Bibliography

[ACM90] Rebecca J. Wirfs-Brock and Ralph E. Johnson; "Current Research in Object Oriented Design"; Communications of the ACM; Sept 1990; Vol 33, Number 9.

[Alberi] "Alberi Graphical User Interface for Postgres Release 0.91";
ftp site: athena.cs.uga.edu;
Directory: /pub/dist;
Tar file: alberi.0.91.tar.Z

[Desai] Bipin C. Desai; "Object Orientation"; Technical Report; Concordia University; Montréal, Canada.
ftp site: ftp.cs.concordia.ca;
Postcript File: /pub/www/bcd/oodbms.ps

[Geo] "Graphical User Interface for Geographical Environment using Postgres Release 1.37";
ftp site: s2k-ftp.cs.berkeley.edu;
Tar File: /usr/postgres/geo.tar.Z

[Grogono] Peter Grogono; "Object Oriented Design"; Technical Report; Concordia University; Montréal, Canada.

[Grogono91] Peter Grogono; "Issues in the Design of an Object-Oriented Programming Language"; Structured Programming; Springer-Verlag; New York Inc.; January 1991.

[Kim90] Won Kim; "Introduction to Object-Oriented Databases"; Cambridge, Massachusetts; 1990 MIT Press.

[Mosaic] "NCSA Mosaic World Wide Web browser Release 2.4"; Dec 1993.
ftp site: ftp.ncsa.uiuc.edu
Directory: /pkg/Mosaic

[Motif] Dan Heller; "Motif Programming Manual for OSF/Motif Release 1.2, Edition 6"; O'Reilly and Associates, Inc.;
USA; Feb 1994.

[Pg Rel 4.1] "Postgres Installation Instructions Release 4.1"; ftp site: epoch.cs.berkeley.edu; Directory: /usr/postgres; Feb 1993.

[Spog] "Command line Interface for Postgres"; ftp site: epoch.cs.berkeley.edu; Directory: /usr/postgres/src/contrib.

[Wegner90] Peter Wegner; "Concepts and Paradigms of Object-Oriented Programming"; OOPSLA-89 Keynote Talk; Brown University; Expansion of Oct 4.