ADDITION OF A NEW RDBMS TO THE 'ACCESS FIELD' OF THE CIS/TK SYSTEM

by

Gautam A. Gidwani

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1989

Signature of Author _____
Department of Electrical Engineering and Computer Science
June 5, 1989

Certified by _____
Professor Stuart E. Madnick
Thesis Supervisor

Accepted by _____
Professor Leonard A. Gould
Chairman, Department Committee on Undergraduate Theses

ADDITION OF A NEW RDBMS TO THE 'ACCESS FIELD' OF THE CIS/TK SYSTEM

by

Gautam A. Gidwani

Submitted to the Department of Electrical Engineering and Computer Science on June 5, 1989 in partial fulfillment of the requirements for the degree of Bachelor of Science.

Abstract

Large organizations often have many distinct divisions, each of which uses its own independent Database Management System (DBMS). This does not allow one division to access information electronically from the other divisions in the organisation. Thus, inefficiency in the sharing of information among divisions, and consequently 'divisional boundaries' are created within the organisation. Researchers in management systems have therefore recognized the need for Composite Information Systems (CISs) that will eliminate divisional boundaries by accessing and combining information stored in multiple, disparate databases in remote divisions. Such a system, the 'Composite Information System/Tool Kit' (CIS/TK) is currently in development at the MIT Sloan School of Management. An object-oriented mechanism for data-retieval from multiple remote, disparate databases has been developed as the lowest hierarchic level of the CIS/TK, and is referred to as the Local Query Processor (LQP). This paper outlines the process by which a new remote, disparate, Relational DBMS (RDBMS) can be added to the 'access field' (to be defined in this paper) of the CIS/TK system.

Thesis Supervisor: Professor Stuart E. Madnick
Title: Professor of Management, M.I.T. Sloan School of Management

Dedication

To Prof. Stuart Madnick, who gave me his confidence and advice - despite my ignorance - and agreed to be my thesis Supervisor.... many thanks.

To Prof. Richard Wang, who besides being the backbone of the LQP development process in its early stages, was patient with me for my inexperience.

To Alec Champlin, who I never met, but who implemented the original LQP prototypes, thereby laying the foundation for my own work.... I owe this dedication.

To John Maglio, the man who knows everything there is to know about the IBM 4381 mainframe, who always went out of his way to help, and without whose suggestions and guidance I would never have got the job done... much gratitude.

To Ray Faith, the AT&T counterpart to John Maglio, who was also always there to point me in the right direction when I had a problem.

To everyone who was a part of my life over the last four years - particularly the friends who gave me their understanding and increased my own, and without whom I would not have been able to suffer the MIT factory.

To my family, who have always shared their wisdom with me, for the worries I have caused them, and for their love.

And specially... To my sister, Veena, who has not only made all this possible, but has been my best friend.... my love and deepest gratitude.

Table of Contents

List of Figures

Chapter 1

INTRODUCTION

Today, an increasingly high percentage of organizations use Database Management Systems (DBMSs). DBMSs have greatly improved the efficiency - and thereby the productivity/competitiveness - of organizations by eliminating the time-consuming process of managing large volumes of information (data) manually. In addition, the use of DBMSs has provided organizations greater flexibility in the storage, sorting and retrieval of data. Optimization of information management, in an effort to further increase productivity/competitiveness of organizations, continues to be an ongoing endeavor for both Management Information Systems (MIS) managers as well as researchers of data management systems.

A major drawback of contemporary information management systems is the inevitable presence of 'information boundaries' among disparate systems.[1] Information boundaries are manifest in the inability of divisions/organizations to access data from one another. These boundaries may be 'intra-organizational' (divisional boundaries within an organisation) or 'inter-organizational (boundaries between two or more organizations). MIS managers and executives in business organizations are now becoming increasingly aware of the potential revenue increases (through greater efficiency), and strategic opportunities that can be harnessed by eliminating these information boundaries.

'Composite Information Systems' (CISs) have been recognized by researchers as the future solution to eliminating - or at least reducing - information boundaries towards these profitable ends. As is implied by the name, CISs are information systems capable of

---

[1]Madnick, S.E. and Wang, Y.R., *Integrating Disparate Databases for Composite Answers*, In *Proceedings of the 21st Hawaii International Conference on System Sciences*, Kailu-Kona, Hawaii, January, 1988.

accessing and integrating data scattered among several remote, disparate DBMSs. The 'Composite Information System/Tool Kit' (CIS/TK), currently in development at the MIT Sloan School of Management is such a system.

The CIS/TK, being implemented in an object-oriented framework, intends to access information from as many DBMSs as are included in its 'access field'. The access field, as applied in this paper to the CIS/TK system, is defined as the entire range - or field - of DBMSs that can be accessed by the CIS/TK system. The CIS/TK system allows the user to make queries that would require information from one or any number of DBMSs in its access field. The intelligent upper levels of the system's hierarchy, namely the 'Application Query Processor' (AQP) and the 'Global Query Processor' (GQP) decide which DBMSs are to be accessed to retrieve the required information. The GQP then creates sub-queries from the user's query - known as 'Abstract Local Queries' (ALQs). These ALQs are sent by the GQP to the lowest hierarchic level of the CIS/TK system, namely the 'Local Query Processors' (LQPs). It is important to note that an LQP exists for each DBMS to be accessed by the CIS/TK system. Each LQP receives a single ALQ which it addresses to the distinct DBMS that it represents, and thus retrieves the requested data. Thus, the inclusion of a new DBMS in the access field of the CIS/TK involves the implementation of a new LQP that will access that DBMS.

Figure 1 provides a global view of the CIS/TK hierarchy and identifies the position of each LQP as the interface between the higher levels of the CIS/TK and the distinct DBMS that the LQP accesses.[2]

---

[2]Horton, Dave, *The Design and Implementation of the CIS/TK Query Processor Architecture*, Master's Thesis, Massachusetts Institute of Technology, May, 1988.

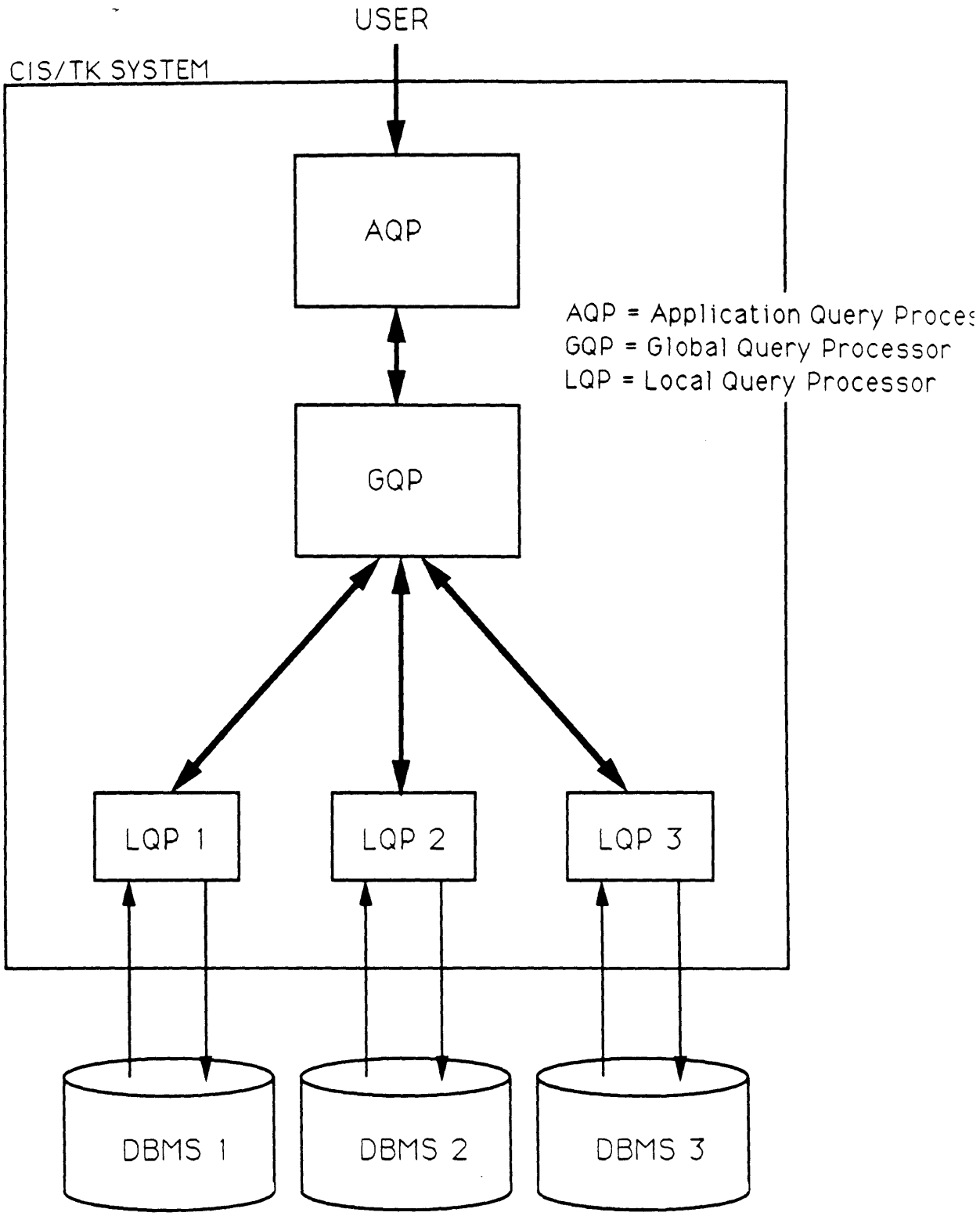


Figure 1-1: CIS/TK Hierarchy and LQP Interfaces

LQPs provide both 'physical' and 'logical'connectivity between the higher levels of the CIS/TK and the respective DBMSs for which they are defined. 'Physical connectivity' refers to the hardware and lower level communication software required to enable the transfer of data between two computer systems. 'Logical connectivity', as applied to CIS systems, refers to the process of relating the information in two seperate DBMSs when there may be differences in syntax and semantics. Each LQP is a *virtual driver*; that is, it reproduces the actions necessary to retrieve information from remote databases as if the query were being processed by a human user. For a thorough understanding of the object-oriented LQP-implementation methodology used herein, it is strongly recommended that the reader study Alec R. Champlin's thesis, "Integrating Multiple Remote Databases in an Object-oriented Framework".[3] The paper provides a detailed description of the specifications of an LQP, and an overview of the object-oriented framework used to implement it.

This paper serves as a practical guide for adding a new remote, Relational DBMS (RDBMS) to the access field of the CIS/TK system using Alec R. Champlin's object-oriented framework. Chapter 2 provides an overview of the internals of an LQP. Chapter 3 defines the problem of adding a new remote DBMS - specifically, the SQL/DS RDBMS on MIT Sloan School of Management's IBM 4381 mainframe - to the CIS/TK access field. The step-by-step implementation procedure followed in the addition of the new LQP is also elaborated in this chapter. Chapter 4 provides a sample session of the working LQP, explaining its functions. Chapter 5 identifies possible future system-specific problems due to the idiosyncracies of the systems involved. Finally, chapter 6 consists of a conclusion, and discussion of future areas of work in the improvement of the LQP implemented herein.

---

[3]Champlin, Alec R., *Integration of Remote Databases in an Object-oriented Framework*, May, 1988.

Chapter 2

OVERVIEW OF A LOCAL QUERY PROCESSOR

This chapter presents an overview of the internal structure of a Local Query Processor (LQP) that is to access a Relational DBMS (RDBMS). The reader is reminded that a distinct LQP is required for each DBMS to be accessed by the CIS/TK system. However, intelligent programming may allow the sharing of program files among several LQPs. Alec R. Champlin uses this program-sharing scheme in the implementation of 3 LQPs in his thesis paper, "Integrating Multiple Remote Databases in an Object-oriented Framework". The 3 LQPs implemented by Alec Champlin use common files to serve as LQP modules that are referred to individually in this paper as LQP-driver modules, and collectively as the LQP-driver.

An LQP may be thought of as 2 parts:

1. The LQP-object
2. The LQP-driver

The functions and internal structure of these two parts of the LQP shall be studied in this chapter. It is imperative that the reader follows, where reference is made, the program files in the appendices or the sample session of chapter 4.

2.1 THE LQP-OBJECT

The LQP object receives a message from the Global Query Processor (GQP), and controls the LQP-driver modules in retrieving the information requested from a DBMS.

The LQP-object contains all the information - stored as *attributes* - regarding file manipulation, and connectivity between the machine on which the 'local' CIS/TK system resides and the 'remote' machine on which the DBMS is to be accessed. The attributes

typically include information such as the name of the remote machine, a user account and a password on the remote machine,and the directories in which files may be accessed on both machines.Also included in the LQP-object is a set of *methods* that are invoked by messages that may be sent to (and are defined in) the LQP-object. It is specifically the methods that control the LQP-driver modules and thereby perform the required data-retrieval functions. There are currently 4 messages that can be supported by the LQP-object. These, along with their respective functions are:

1. **:SELF-INFO** *provides the CIS/TK user with information about the particular DBMS with which the message is associated. The :SELF-INFO message requires no arguments.*
2. **:GET-TABLES** *returns a list of all the tables which can be accessed within the particular DBMS with which the message is associated. The :GET-TABLES message requires no arguments.*
3. **:GET-COLUMNS** *returns a list of all the columns for a specified table within the DBMS with which the message is associated. The :GET-COLUMNS message require a valid 'table name' as its argument.*
4. **:GET-DATA** *returns a list of the requested data from the DBMS with which the message is associated. The :GET-DATA message requires an 'Abstract Local Query' - to be discussed below - as its argument.*

It is important to understand the function of an LQP and how it receives messages prior to discussing Abstract Local Queries (ALQs). The **send-message** function defined in **KOREL**[4] (Knowledge-Object REpresentation Language) is used to pass messages to the LQP(s) from the GQP. If data is required from more than one DBMS, then messages are sent to all the respective LQPs representing distinct DBMSs. The format of the **send-message** function is:

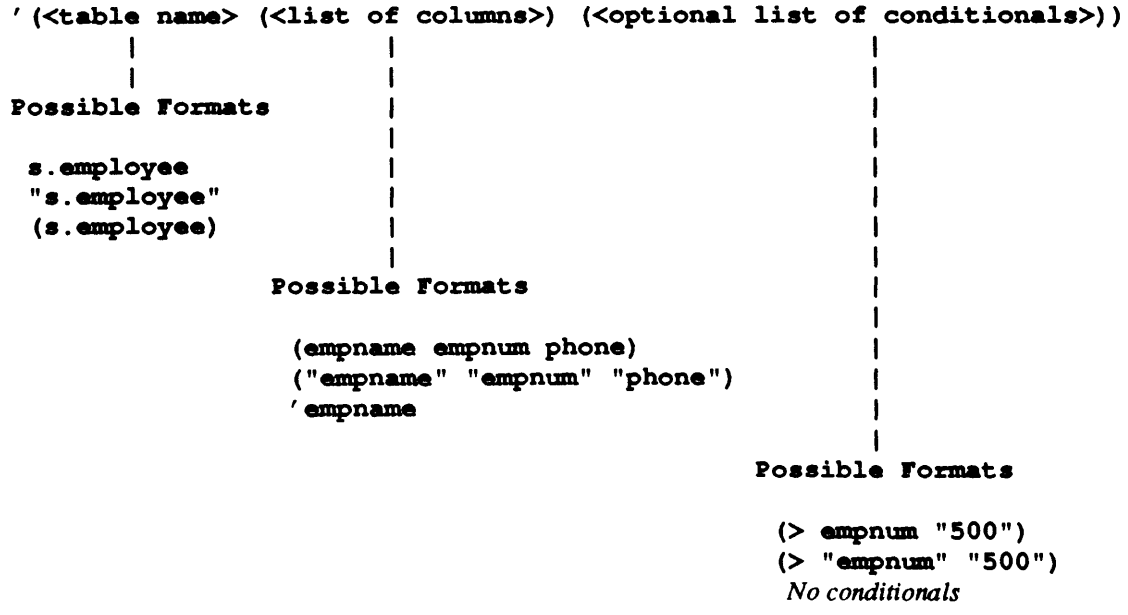
---

[4]Levine, Samuel P., *Interfacing Objects and Databases*, Master's Thesis, Massachusetts Institute of Technology, May, 1987.

(send-message <message> '<lqp-object or instance> <required arguments>)

1. **<message>** *is one of the 4 messages acceptable by the LQP - :SELF-INFO, :GET-TABLES, :GET-COLUMNS, or :GET-DATA.*
2. **<lqp-object or instance>** *is either the name of the LQP-object or its instance, and identifies an LQP. KOREL supports a hierarchic object definition. An* **instance** *of an object inherits all the* **attributes** *of its* **superior**, *the LQP-object. In addition the instance may also have its own set of unique attributes. A message may be sent either to an object, or alternatively to its instance.*
3. **<required arguments>** *may be either a valid table name for the :GET-COLUMNS message, or an ALQ for the :GET-DATA message.*

Now the Abstract Local Query (ALQ) is studied. The format of ALQs is predetermined, and is illustrated below. An ALQ received by an LQP is converted to a query format - known as the 'DBMS Query' - that can be processed by the DBMS to which the message (in this case a :GET-DATA message) is sent. The ALQ format is:



1. **<table name>** *is the name of the table within the DBMS with which the LQP is associated, from which data is to be retrieved.*
2. **<list of columns>** *is a list of columns within the specified table from which rows of data are to be retrieved.*
3. **<optional list of conditionals>** *is an optional list of mathematical and/or logical conditionals. These conditionals specify the rows of data required from the specified columns within the specified table. The conditionals presently supported are:* AND, OR, NOT, NULL, =, !=, >, <, >=, and <=.

Below are a few simple examples of ALQs:

```
EXAMPLE (1)
'(s.employee (empname empnum phone))

  This ALQ represents a request for all the rows of information from the
  'empname', 'empnum', and 'phone' columns in the 's.employee' table.

EXAMPLE (2)
'(s.employee (empname phone) (OR (= empname "name1")
                                 (= empname "name2")))

  This ALQ represents a request for the 'empname' and 'phone' informa-
  tion where the 'empname' is either 'name1' or 'name2'.
  This ALQ provides an illustration of the use of conditionals. Note the
  mandatory use of double quotes for strings to be compared with data
  entries in the DBMS.

EXAMPLE (3)
'("s.employee" ("empname" "empnum" "phone") (< "empnum" "500"))

  This example illustrates that column and table names in ALQs may
  alternatively be formatted as character strings using double quotes -
  a task otherwise performed by the 'Query Translator' module in the
  LQP-driver (discussed later).
```

At this point, the reader is referred to the LQP-object definition in the file **sqlds4381.lsp** (Appendix A.1). Note the attributes defined in the LQP-object "sqlds-4381", its instance "sloandb", message definitions for :SELF-INFO, :GET-TABLES, :GET-

COLUMNS and :GET-DATA, and the respective methods invoked by these messages, *:display-sqlds4381-self-info*, *:get-sqlds4381-tables*, *:get-sqlds4381-tables* and *:get-sqlds4381- data*.

It should be noted that the methods invoked by the messages make calls to the LQP-driver modules. Thus, the important observation should be made that the LQP-object of an LQP 'controls' its LQP-driver. The automated LQP-driver is referred to as the *virtual* LQP-driver.

The following section describes the internal functioning of the LQP-driver.

2.2 THE LQP-DRIVER

The LQP-driver consists of 4 modules which are controlled by the LQP-object. These modules are:

1. The Query Translator Module
2. The Communication Server Module
3. The Data Filter Module
4. The Result Reader Module

The function of these modules is described in the following sections (2.2.1-2.2.4). In order to acquire a better understanding of the functions of each LQP-driver module, the reader should follow the sample session of chapter 4 after the discussion of this chapter. The process of implementing new LQP-driver modules is studied in some depth in chapter 3.

Figure 2 - Internal Structure of LQP-driver and Module Functions - presents an overview of the contents of this section (2.2).

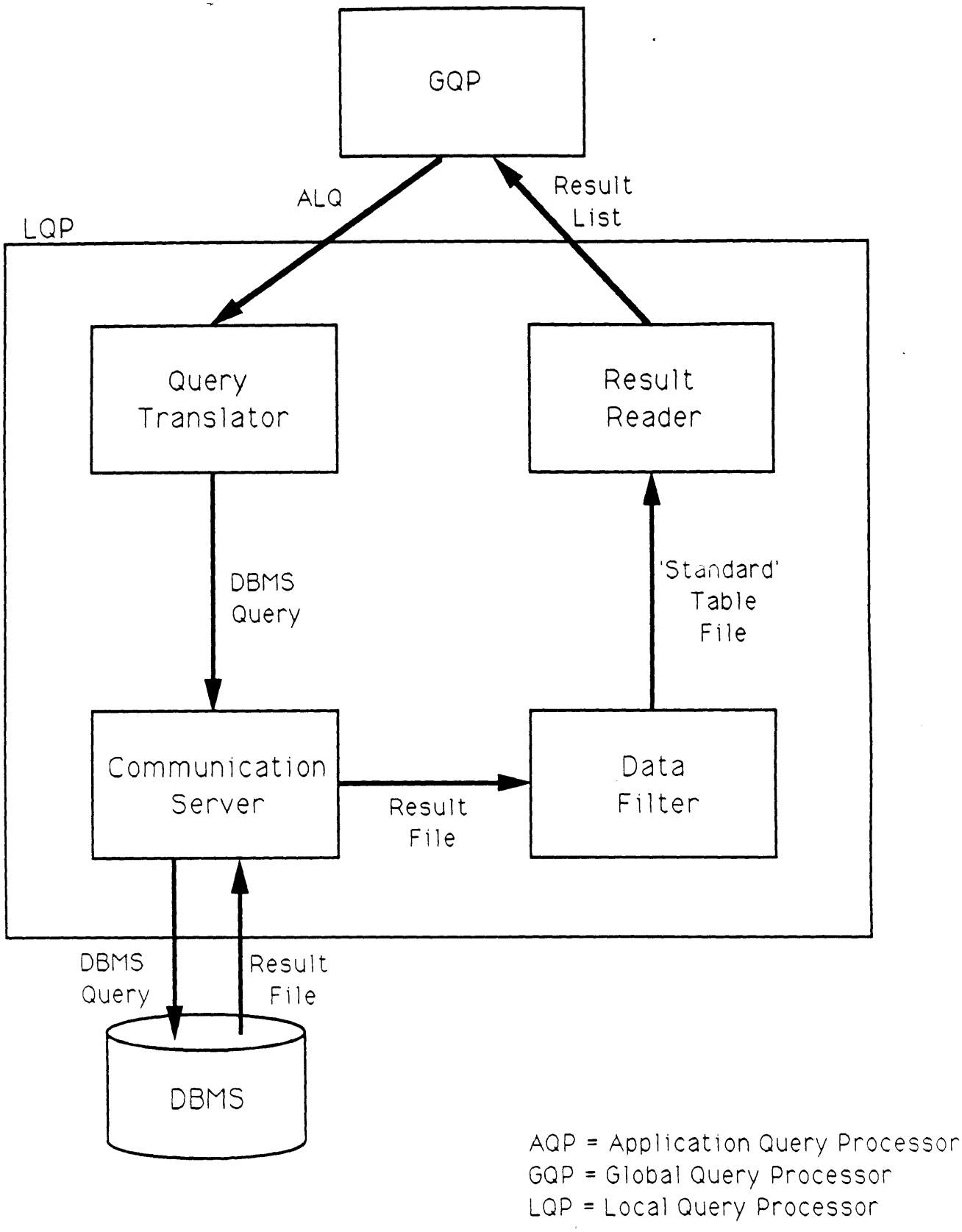


Figure 2-1: Internal Structure of LQP-driver and Module Functions

2.2.1 The Query Translator Module

Each DBMS can only process a database query of a particular specified format. The Query Translator performs the task of converting the ALQ received by the LQP (from the GQP) to this format required by the DBMS to be queried. The query statement format created by the Query Translator is referred to as the 'DBMS Query'. Presently all Relational DBMSs use very similar - if not identical - query formats under the 'Structured Query Language' (SQL) convention. Thus if the SQL statement required by one RDBMS is identical to that required by another, it is possible to use a single Query Translator for the two separate LQPs representing different DBMSs.

SQL provides a 'SELECT' database query statement. Most DBMSs support a 'SELECT' statement format that is identical or very similar to:

SELECT <columns> FROM <table> WHERE <conditionals>

1. **<columns>** *is a list of column names, seperated by commas, within the table from which rows of data are to be retrieved.*
2. **<table>** *is the table from which data is to be retrieved. A table consists of several columns identified by column names, and rows of data within these columns.*
3. **<conditionals>** *are qualifiers that specify which rows of data should be retrieved from the specified columns within a table.*

Recall, for illustrative purposes, the ALQs presented in section 2.1. The ALQs are to be translated to SQL SELECT statements that are acceptable to the RDBMS with the LQP is associated. Below are examples of Query Translator conversions of the ALQs presented earlier, to their respective SQL SELECT statements for an ORACLE RDBMS:

```
EXAMPLE (1)
 ALQ:  '(s.employee (empname empnum phone))
 SQL:  select empname, empnum, phone from s.employee

EXAMPLE (2)
 ALQ:  '(s.employee (empname phone) (OR (= empname "name1")
                                        (= empname "name2")))
 SQL:  select empname, phone from s.employee
         where (empname = 'name1') OR (empname = 'name2)

EXAMPLE (3)
 ALQ:  '("s.employee" ("empname" "empnum" "phone") (> "empname" "500"))
 SQL:  select empname, empnum, phone from s.employee
         where empname > '500'

Note: The ALQ and SQL formats need not be broken up into more than one
      line as they are above.
```

The DBMS Query created by the Query Translator is sent to the DBMS to be accessed by the Communication Server Module.

2.2.2 The Communication Server Module

The Communication Server module provides the required connectivity between the local machine and the machine in which the DBMS resides. It may therefore be considered the heart of the LQP-driver.

This module must first connect to the machine from which data is to be retrieved. Then, it needs to access the DBMS on that machine and provide it with the DBMS Query (SQL statement if an RDBMS) created by the Query Translator. Once the DBMS has been queried, the query result - referred to as the 'Result File' - must be retrieved to the machine on which the CIS/TK system awaits information. This concludes the crucial responsibilities of the Communication Server, and the Result File is then passed to the Data Filter for further processing.

The querying of a remote DBMS involves connecting to the machine on which the

DBMS resides. Then, the DBMS is to be accessed. Depending on how the system is set up, this may or may not involve logging into a user account on the remote machine, and entering a particular directory from which the DBMS may be accessed. The DBMS Query is then sent to the DBMS for processing. Thus, as a result of the idiosyncracies of the systems involved in this process, the Communication Server's task is not always trivial.

The sample session of chapter 4 shows the DBMS query result - the Result File - retrieved by the Communication Server for the new LQP implemented in chapter 3.

2.2.3 The Data Filter

Often, the Result File contains irrelevent communication information - for example, the name of the DBMS, markers at the end of a query result output. The Data Filter is responsible for filtering this irrelevent information from the Result File.

The second, more important function of the Data Filter is to parse the Result File to a standard 'Table File' format. The Table File is standardized to a predetermined format that is acceptable to the Result Reader module. In order to identify the function of the Data Filter, both the Result File and the standardized Table File should be compared for the :GET-TABLES message in the sample session of chapter 4.

The standardized Table File is processed further by the Result Reader module.

2.2.4 The Result Reader

The Result Reader module performs the final LQP task of parsing (reformatting) the Table File returned by the Data Filter to another standardized format - the 'Result List' - required by the Global Query Processor.

The Result List format defined for the CIS/TK LQP implementation methodology used in this paper is:

```
RESULT LIST FORMAT:

  ((column-name1 column-name2 column-name3 ..... column-nameN)
   (data11 data12 data13 ................. data1N)
   (data21 data22 data23 ................. data2N)
   (data31 data32 data33 ................. data3N)
      .       .       .                      .
      .       .       .                      .
      .       .       .                      .
      .       .       .                      .
   (dataM1 dataM2 dataM3 ................. dataMN))
```

The Result List is actually a LISP-readable list of lists to be sent by the LQP to the GQP. The first list within the Result List is a list of the names of the columns from which data was requested. The row of data forms a separate sublist, and each entry within a sublist - column-name1, data12, data23, etc. - is of a string format. Above, the Result List format was exemplified for M rows of data within N specified columns. <dataXY> refers to the data entry in the Xth row of the Yth column.

2.2.5 Summary

1. The functioning if the entire LQP has been presented in this chapter.
2. The LQP is devided into 2 parts, the LQP-object and the LQP-driver.
3. The LQP-object contains all the information required by the LQP-driver in processing data, and controls the LQP-driver.
4. The LQP-driver is made up of 4 modules: The Query Translaotor, the Communication Server, the Data Filter, and the Result Reader.
5. The Query Translator converts an Abstract Local Query (ALQ) from the Global Query Processor (GQP) to a format acceptable for processing by the DBMS to be queried. The Query Translator output format is known as the DBMS Query.
6. The Communication Server connects to the DBMS to be queried, queries the DBMS with the DBMS Query, and retrieves the query result as the Result File.

7. The Data Filter filters irrelevent information from the Result File and converts it to a standardized format known as the Table File.
8. The Result Reader converts the Table File returned by the Data Filter to a LISP-readable format that is acceptable to the GQP.

The reader should now be familiar enough with the structure of an LQP to understand the specific task of adding the new SQL/DS RDBMS on the IBM 4381 mainframe, "sloan", to the access field of the CIS/TK system. The process of defining this task and then implementing the required LQP is discussed in the next chapter.

Chapter 3

ADDITION OF AN LQP FOR THE SQL/DS RDBMS ON 'SLOAN'

In chapter 1 the motivation for Composite Information Systems (CISs) was discussed and the need for LQPs as the interface between remote DBMSs and the CIS/TK system was identified. Chapter 2 presented an overview of the internal structure of an LQP, providing the framework for the addition of a new DBMS to the access field of the CIS/TK system. Specifically, an LQP is to be implemented as the interface between the CIS/TK system on The AT&T machine, "mit2e", and the IBM SQL/DS RDBMS on "sloan".

Section 3.1 defines the problem of implementing the new LQP. In section 3.2, the procedure followed in the implementation of the new LQP is outlined. Generic guidelines for LQP implementation are provided where appropriate.

3.1 LQP PROBLEM DEFINITION

As discussed in chapter 2, the following modules are required in an LQP:

1. The LQP-object
2. The LQP-driver Query Translator Module
3. The LQP-driver Communication Server Module
4. The LQP-driver Data Filter Module
5. The LQP-driver Result Reader Module

Recall that the LQP-object contains all the attributes (information) required to enable the LQP-driver to function as a virtual driver. Thus, it is logical that the conceptualization of the LQP-driver should preceed the definition of the LQP-object. Following this logic, the LQP-driver modules are visited first. Once the LQP-driver modules have been conceptualized, the LQP-object may be implemented concurrently with the LQP-driver.

3.1.1 The Query Translator: Problem Definition

The Query Translator must convert the ALQ to the SQL SELECT statement required by the SQL/DS RDBMS. The SQL format required by the SQL/DS RDBMS is identical to that required by the ORACLE RDBMS discussed in section (2.2.1). A Query Translator that converts an ALQ to the format required by an ORACLE RDBMS (and therefore the SQL/DS RDBMS) already exists. This Query Translator was implemented by Alec R. Champlin in his implementation of the first CIS/TK prototype LQPs. The existent Query Translator may thus be used (shared) by the new LQP for the SQL/DS RDBMS, and this problem is thus voided.

In this case a Query Translator that could perform the task of creating the appropriately formatted SQL statement for the new LQP already existed. This may not be true in general, in which case the implementor of a new LQP must implement a new Query Translator. If an RDBMS's format does not differ from the SQL/DS or ORACLE RDBMS formats by a great deal, it may be possible to use the existing Query Translator as the new Query Translator simply by making minor modifications to its program code.

The reader is referred to the LISP file, **sql.lsp** (Appendix A.2), representing the Query Translator used in the LQP for the SQL/DS RDBMS. The reader intending to implement a new LQP with an SQL format differing from the SQL/DS or ORACLE RDBMS SQL formats is urged to study the program code thoroughly. It is posiible that simple modifications will allow a new LQP to share the Query Translator code in **sql.lsp** with the existent LQPs.

Tasks For Query Translator Implementation

1. Check if any modifications are required to share the existing Query Translator module with the new LQP.

3.1.2 The Communication Server: Problem Definition

The Communication Server is, as stated earlier, the heart of the LQP. The Communication Server for the SQL/DS RDBMS LQP must provide the DBMS Query (SQL statement) created by the Query Translator to the SQL/DS DBMS on the IBM 4381 mainframe, "sloan". This process entails connecting to the IBM machine "sloan", accessing the SQL/DS RDBMS, and querying the RDBMS with the SELECT SQL statement. In addition, the Communication Server must efficiently retrieve the query result - Result File - returned by the SQL/DS RDBMS to the "mit2e", so that it may be processed further by the LQP.

Considering first the problem of querying the remote SQL/DS RDBMS: It is found that a UNIX script file (containing the required commands) may be used to access the VM/IS operating on the remote machine "sloan", thereby establishing the connection between the two machines. Presently, the machine "sloan" may be accessed by using either the **telnet** facility, or the **cu** (call up) facility provided by UNIX on the "mit2e". It is found that the telnet facility takes sub-second times to establish connection between the "mit2e" and "sloan", whereas the cu facility takes approximately 35 seconds, and has a success rate of only about 75 percent. Therefore the telnet facility is chosen for this implementation. Further tinkering with the systems reveals that once connection has been established between the two machines, 0a robust method of accessing the SQL/DS RDBMS is by first logging into a user account on "sloan", and then using the virtual CMS operating system's EXECutive commands to directly access the RDBMS. Specifically the Sloan Information System database, "slinfo", is to be queried using this procedure.

Assuming for now that such a script file can be implemented to access the SQL/DS's "slinfo" database, the next problem is to retrieve the query result from "sloan" to the "mit2e". It is found that a second script file may be used to perform this task.

A reasonable methodology in the implementation of the required script files discussed above, is to first enter the commands to be embedded in the script files manually (on a keyboard) from the "mit2e". Once the required commands have been identified, the script files may be created. The information required by the Communication Server script files as arguments may then be defined as attributes in the LQP-object. The Communication Server can then be automated by providing the script files, along with their required arguments, to the UNIX operating system on "mit2e" using KOREL's message-passing facility. This entails the creation of a Communications Server template that is controlled by the LQP-object and may then be used to control the script files being used to query the RDBMS and retrieve the query result.

A template file for the Communication Server automation has already been implemented by Alec R. Champlin, simplifying the task of implementing the new Communication Server for the SQL/DS RDBMS LQP. This file, **connect.lsp** is found in Appendix A, and the reader is urged to study it closely.

Tasks For Communication Server Implementation

1. Identify the sequence of commands to be automatically supplied by the Communication Server to the remote SQL/DS RDBMS on "sloan". These commands must perform the functions of accessing the SQL/DS RDBMS, querying the "slinfo" database with the DBMS Query, and retrieving the Result File to the local machine, "mit2e".
2. Implement the script file that accesses the remote SQL/DS RDBMS on "sloan" and queries the resident database "slinfo".
3. Implement the script file that is to retrieve the query result from "sloan" to the "mit2e" as the Result File.
4. Check if any modifications are required to share the existing Communication Server template with the new LQP.

Note that the possibility of combining the two script files required here into a single script file should be explored.

3.1.3 The Data Filter: Problem Definition

The Data Filter performs the tasks of filtering irrelevent Communications messages, and parsing the Result File into a standardized Table File format which is readable by the Result Reader.

This module may be implemented by writing a 'C' program that performs both the required filtering and parsing functions. The functionality required of the C program is dependent on two factors: (1) The method used by the Communication Server in accessing the RDBMS and retrieving the Result File - Communications messages are dependent on the sequence of commands used in data retrieval. (2) The format of the query result output from the RDBMS - RDBMS output formats vary. The parsing functionality required depends on the format returned by the SQL/DS RDBMS.

The standard Table File format is found in the sample session of chapter 4. The reader intending to implement a new LQP using the scheme outlined in this paper should study this format as a means of identifying the parsing requirements of his/her Data Filter. Note that C has been chosen to implement the Data Filter by virtue of its speed, and suitability for implementation of filtering and parsing functions.

Tasks For Data Filter Implementation

1. Write a 'C' program to convert the Result File returned by the Communication Server to the standard Table File format that is readable by the Result Reader.

3.1.4 The Result Reader: Problem Definition

The Result Reader, like the Data Filter, performs a parsing function. It converts the standard Table File to a standard Result List that is LISP-readable by the Global Query Processor (GQP).

The Result Reader too has already been implemented by Alec R. Champlin, and performs the required task of converting the Table File to a Result List. Thus, this task is rendered void under the present implementation scheme.

The reader is referred once again to the sample session of chapter 4 as a means of identifying the parsing function performed by the Result Reader. The formats of both the Table File and Result List should be studied closely. The Result Reader implemented by Alec R. Champlin in the LISP file **read.lsp** (Appendix A.4), uses a C program - namely **preREAD.c** (Appendix B.2) - to aid in its task of reformatting the Table File. If the reader intending to implement a new is confident that his/her new Data Filter can return a Table File of the specified format, then he/she can be assured that the existent Result Reader can be shared by his/her new LQP. This is true because the Result Reader has standardized formats both as its input and as its output.

Tasks For Result Reader Implementation

1. Check if any modifications are required to share the existing Result Reader Module with the new LQP.

3.1.5 The LQP-object: Problem Definition

The LQP-driver having been conceptualized, it is now easy to identify the specific requirements of the LQP-object. The LQP-object must control the four LQP-driver modules, thereby creating a virtual LQP-driver.

The LQP-object for the SQL/DS RDBMS must contain as attributes all the information required by the LQP-driver in accessing the SQL/DS RDBMS on "sloan", querying its "slinfo" database, retrieving data to the local "mit2e" machine, and processing data to the final Result List format required by the GQP. In addition, the LQP-object must contain the four 'methods' discussed in section (2.1) that will control the LQP-driver modules in performing the required task of retrieving the requested data from "slinfo" on "sloan".

The reader is now advised to study closely the LQP-object in file **sqlds4381.lsp** (Appendix A.1). The LQP-object definition, along with the sample session of chapter 4, is helpful to the understanding of the LQP-object, and the inheritance of attributes of a

'superior' object by its 'inferior' instance. The reader will notice that the LQP-object has been named "sqlds-4381" and contains all the information required by the LQP-driver modules as attributes. In addition, the four methods alluded to above are also defined in "sqlds-4381". The "sloandb" instance is created for the "sqlds-4381" LQP-object, and it inherits all the attributes and methods defined for "sqlds-4381". The instance of an object can have its own unique attributes and methods in addition to those inherited from its superior object. Further, an examination of the procedures invoked by the messages (having the same names as the methods) sent to the LQP-object will reveal exactly how the LQP-object controls the LQP-driver.

Tasks For LQP-object Implementation

1. Define the LQP-object, with required attributes and methods. The LQP object must contain all the required attributes required by the LQP-driver modules, as well as the methods that may invoke defined procedures by the appropriate messages to the LQP-object.
2. Define procedures (whose names are defined in the methods) that are invoked by the methods when the appropriate message is passed to the LQP-object.

This concludes an identification of the LQP implementation tasks at hand for the addition of the SQL/DS RDBMS to the access-field of the CIS/TK system. The next section discusses the actual implementation procedure followed.

3.2 LQP IMPLEMENTATION

In this section, the procedure followed in the implementation of the LQP for the SQL/DS RDBMS is outlined. Where appropriate, the idiosyncracies encountered during implementation are identified, providing the reader with a realistic view of LQP implementation. A step-by-step account of the actual implementation methodology used by the author follows. Once again, the implementation of LQP-driver modules preceeds the LQP-object implementation.

The implementation tasks identified for the LQP modules are considered one at a time in sections (3.2.1- 3.2.5).

3.2.1 The Query Translator: Implementation

Query Translator File(s):
(1) sql.lsp (Appendix A.2)

TASK 1: Check the existing Query Translator file

The existing Query Translator module - file **sql.lsp** (Appendix A.2) - was to be tested in order to determine whether any modifications were necessary to use it as the new Query Translator for the SQL/DS RDBMS LQP.

The *form-SQL* LISP procedure defined in **sql.lsp** takes an Abstract Local Query (ALQ) as its argument. It uses the *parse-SQL-tname*, *parse-SQL-colmn* and *parse-SQL-conds* procedures respectively, to parse the table name, column list, and conditionals list of the ALQ, and to create the SQL statement format required by SQL/DS. These four procedures were tested thoroughly with appropriate arguments, and it was determined that they flawlessly performed the task of converting the ALQ to the required SQL/DS SQL SELECT statement.

The file **sql.lsp**, implemented by Alec R. Champlin, was thus used unmodified as the Query Translator for the new LQP.

3.2.2 The Communication Server: Implementation

Communication Server File(s):
(1) connect.lsp (Appendix A.3)
(2) 4381FILE (Appendix C.1)
(3) 4381FTP (Appendix C.2)
(4) 4381SEL (Appendix D.1)

TASK 1: Identify all required data retrieval commands

This was found to be the sequence of commands required to access the SQL/DS RDBMS on "sloan", query the "slinfo" database, and retrieve the query result to the local machine, "mit2e":

1. Connect to the remote machine "sloan" by providing the command **telnet**

sloan. This command is provided form the UNIX operating system on the "mit2e". Recall that the telnet facility was chosen over the cu facility, as discussed in the problem definition of section (3.1.2)

2. Provide a **logon <user-account>** command to the VM/IS operating system on "sloan". <user-account> is the name of a user account on "sloan".

3. Similarly, provide the user's **<password>** at the password prompt.

4. Run the virtual CMS system on "sloan's" VM/IS operating system by providing the command **ipl cms**.

5. Create a line mode I/O (input/output) environment by providing the **ac(noprof** and **sysprof3** commands to CMS. Typically, IBM machines use a full-screen (block) mode I/O environment. However, the UNIX operating system on the local AT&T 3B2 machine, "mit2e", is designed to handle streams of data (line mode) as opposed to blocks of data (full-screen mode). Specifically, the **ac(noprof** command accesses the hard disk when the user account has been accessed, but disallows the running of the 'profile' EXEC program which sets the terminal environment. It was found that the environment that 'profile' sets is not appropriate for line mode I/O. The **sysprof3** command also sets environmental variables. Although this command is not necessary, it is useful to ensure that the user (in this case the virtual LQP-driver) is provided with all the 'standard' operating system capabilities.

6. Initialize the SQL/DS database "slinfo" to be queried with the command **dbinit slinfo**.

7. Provide the **RXCASE String** executive (EXEC) command to the virtual CMS operating system to prepare it for a subsequent executive command with an argument of the 'string' format. The RXSELECT executive command, in 8 below, takes an SQL statement of string format as its argument.

8. Query the database "slinfo" directly with the **RXSELECT <SQL/DS SQL statement>** command. The RXSELECT EXEC takes the SQL statement to be passed to the SQL/DS RDBMS as its argument. The SQL statement is of a 'string' format, and the RXCASE EXEC was therefore required above. The RXSELECT provides a direct interface between the VM/CMS operating system and the SQL/DS RDBMS on "sloan". It is therefore preferred over alternative methods of querying the "slinfo" database. Alternative methods of database access include the SQL interactive program, and the QMF (Query Management Facility) interactive program.

9. Save the query result in a temporary file on the remote machine "sloan" with the command **ffile <remote temporary file>**. The name chosen for the remote temporary file is "4381lqp temp". Thus the command **ffile 4381lqp temp** was issued. Note that this working directory requires a write password for the storage of the query result in a temporary file. If a write password is not provided, an error message will be returned, and the query result will not be saved in the temporary file.

10. Logout of the remote "sloan" machine with the **logoff** command.

The above commands have succeeded in accessing the remote "slinfo" database, querying it, and retrieving data to a remote file on "sloan". In general, UNIX's piping feature enables the piping of the output of a session (such as the one discussed above) to a file on the local machine. However, in this case, the communication level disparities between the AT&T 3B2 and the IBM 4381 machines caused data losses as well as inconsistent formats of retrived data. Specifically, it was found that rows of data were lost in the pipe, or often rows were not seperated by newline characters causing inconsistencies in the data format. Thus a scheme other than a UNIX pipe was considered as an alternative in this process.

It was found that using the **ftp** file transfer protocol was a very robust way of retrieving the data from the remote machine "sloan" to the local "mit2e". This scheme was therefore used as an alternative to the UNIX pipe discussed above, despite the disadvantage of having to use intermediary temporary storage files.

Below is the sequence of additional commands required to retrieve the remote "4381lqp temp" file on "sloan" to the local "mit2e" machine:

1. First, invoke the file transfer protocol facility by providing the **ftp -n** command from the UNIX operating system. This starts up the ftp program, enabling file transfers between the "mit2e" and "sloan". The **-n** (no prompt) option was necessary in this case. The default 'prompt' mode being interactive, it assumes that the <user-account> and <password> (see 3 below) are being typed in by a human user at a keyboard, and therefore it does not support scripted inputs. Thus, the 'no prompt' mode is used.
2. Provide the **open sloan** command to establish ftp connectivity to the remote machine, "sloan".
3. Access the user account containing the temporary data file **4381lqp temp** by providing the command **user <user-account> <password>** to the ftp program. <user-account> and <password> here are the same as in 2 and 3 above.
4. Provide a working directory for the file transfer with the **cd <user-account>**

191. Once again <user-account> is the same account on which the remote query result file had been stored. Note that this working directory needs to be provided with a read password. Also, this is the same working directory as the one provided with a write password in 9 above.

5. Provide the read password for the working directory on the user's account with the **quote acct <working directory password>**. Although the read password may differ from the write password, having the same password for both read and write requires less information (one password instead of two) to be carried as attributes in the LQP-object.
6. Now transfer the remote temporary file with the command **get 4381lqp.temp <local temporary file>** to ftp. This copies the remote temporary file to a local temporary file <local temporary file>.
7. The task of retrieving the data to the local machine is now complete. Thus, the remote temporary file **4381lqp temp** may now be deleted with the command **delete 4381lqp temp** to ftp.
8. Quit the ftp program with the **quit** command.

This completes an identification of the sequence of commands required to query and retrieve data from the remote "slinfo" database on "sloan". Two script files are required to automate these commands - one for the first group of 10 commands (telnet commands), and one for the second group of 8 commands (ftp commands). The ftp commands are required since piping can not be used in this case.

Figure 3 - Communication Server Data Retrieval - provides a clearer picture of how the two script files, 4381FILE and 4381FTP respectively, are used. The first script file, 4381FILE, is responsible for connecting to "sloan", querying the "slinfo" database in the SQL/DS RDBMS, and storing the query result (Result File) in the temporary file, 4381lqp, on the remote machine. The second script file, 4381FTP, then uses the ftp file transfer protocol to retrieve the Result File to the local machine to be passed to the Data Filter.

TASK 2: Implement a script file to query "slinfo" on "sloan"

The script file **4381FILE** (Appendix C.1) automates the process of accessing the "slinfo" database on the SQL/DS RDBMS, querying the database, and storing the query result in a remote temporary file on a user account on "sloan". Thus, the first group of 10 telnet commands is automated by **4381FILE**.

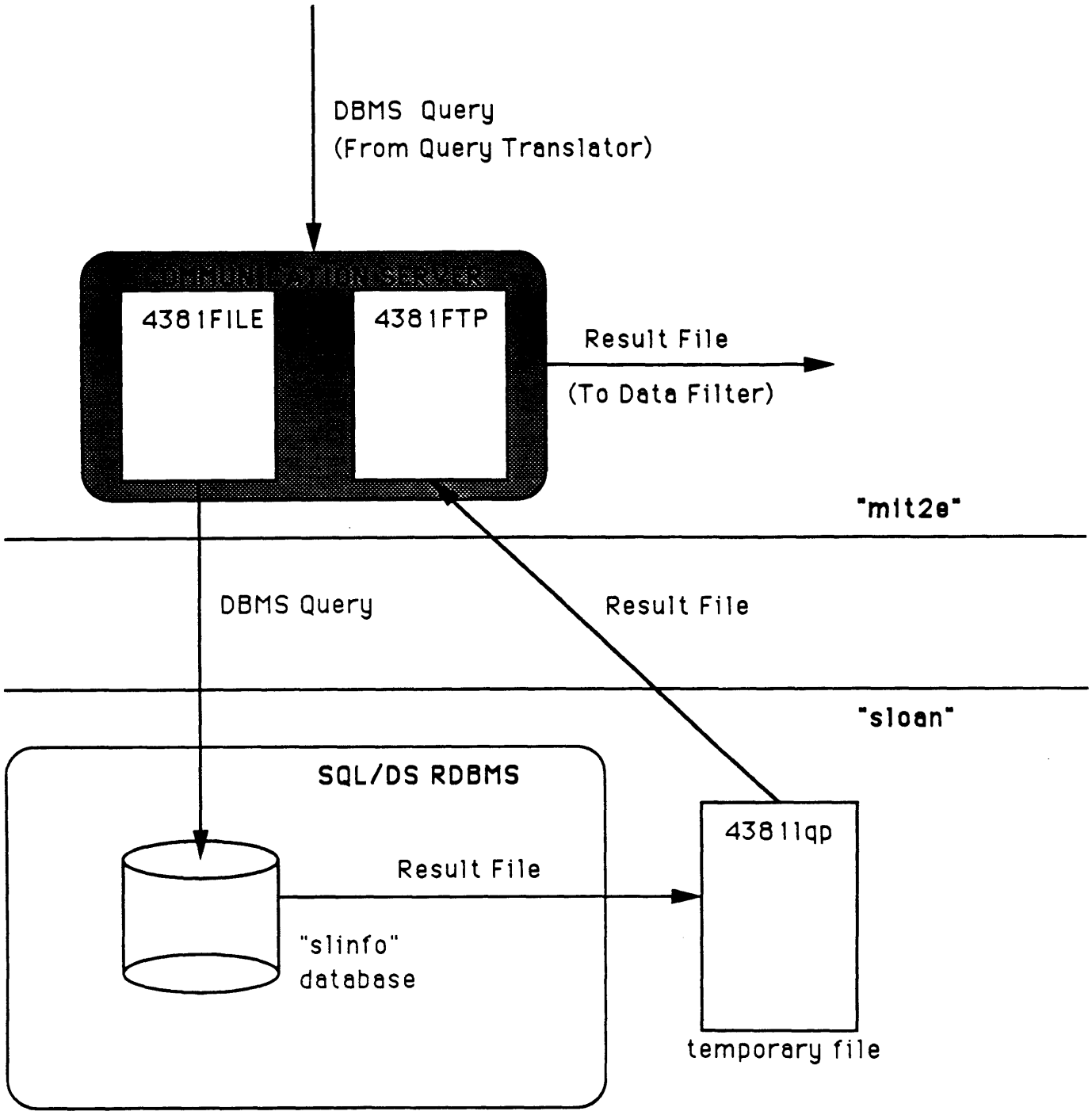


Figure 3-1: Communication Server Data Retrieval

It is important to note that a modified version of the RXSELECT executive program was used instead of the original version. This new version is named the **4381SEL** executive program and may be found in Appendix D.1. The original version of RXSELECT truncated data of more than 254 characters on a single line, and returned blocks of a maximum of 100 rows at a time. The modification to this program increased truncation length to 508 characters, and increased the maximum number of rows returned at one time to 10,000. This was very useful in automating the data retrieval process, since if the original RXSELECT was used, after every 100 rows of data were returned as a block, a **MORE** command would have to be issued to the SQL/DS RDBMS in order to return the next 100 rows of the query result. This process would be very inefficient since only 100 rows of data could be retrieved at a time.

Unlike RXSELECT, the **4381SEL** executive program allows the query result to be returned all at once as a block of no more than 10,000 rows. This is far more efficicient, in terms of data retrieval speed, than issuing **MORE** commands between every 100 rows of data. The modified **4381SEL** is otherwise identical in performance to the RXSELECT executive command.

An additional point to be noted about **4381SEL** is that data will be lost for query results of more than 508 characters on one row, and/or containing more than 10,000 rows of data. These limits can be easily modified by changing the values for the *maxlength* and *maxlns* variables, which are presently set to 508 and 10,000 respectively.

TASK 3: Implement a script file to retrieve data to "mit2e"

Recall that this additional script file is required because UNIX's piping feature can not be used in this case. The script file **4381FTP** (Appendix C.2) uses the **ftp** file transfer protocol to automate the process of retrieving the remote temporary file holding the query result to the local "mit2e".

Although data is retrieved in a robust manner using this ftp script file, the data retrieval process suffers an overhead of 15 seconds due to the **sleep** commands in the script - a great disadvantage as compared with the piping scheme in terms of speed.

TASK 4: Check the existing Communication Server template file

The Communication Server template, **connect.lsp** (Appendix A.3), controls the script files **4381FILE** and **4381FTP**. All the control information required by the Communication Server template - file directories, remote machine name, database name, etc. - are stored as attributes in the LQP-object.

Minor modifications were made to **connect.lsp** to accomodate the new LQP. The new "SQL/DS" database type identifier was included, and the SQL/DS RDBMS invoker was set to "dbinit".

This concludes the implementation of the Communication Server. It should now be apparent, from the function performed by this module, why it is was called the heart of the LQP-driver.

3.2.3 The Data Filter: Implementation

Data Filter File(s):
(1) filt4381.c (Appendix B.1)

TASK 1: Write a 'C' Program to convert the Result File to the Table File

The reader is referred to the Result File and standard Table File formats for the :GET-TABLES message in the sample session (chapter 4).

A C program was written to perform the task of converting the Result File to the Table File. This program, **filt4381.c** is found in Appendix B.1. This C program representing the Data Filter module is controlled by the LQP-object (as are the other LQP-driver modules.

3.2.4 The Result Reader: Implementation

Result Reader File(s):
(1) read.lsp (Appendix A.4)
(2) preREAD.c (Appendix B.2)

The Result Reader defined by Alec R. Champlin performs the task of parsing the Table File to the required LISP-readable Result List format. These two formats are, once again, exemplified in the sample session of chapter 4.

The Result Reader uses the 'C' program preREAD.c (Appendix B.2) - also implemented by Alec R. Champlin - to aid in its task of creating the Result List from the Table File.

TASK 1: Check the existing Result Reader file

On checking whether the existing Result Reader could be shared with the new LQP for the SQL/DS RDBMS, it was found that a minor modification was needed:

The original Result Reader assumed that the standardized Table File was not to be stored in a temporary file, but rather was passed to it internally by LISP from the Data Filter application. The data retrieval methodology used in the new LQP requires the Result Reader to access the Table File from a temporary file in the 'communication server directory' (refer to the LQP-object).

The Result Reader is thus modified to accomodate this new methodology. A conditional statement has been added to the LISP file **read.lsp** (Appendix A.4) to determine whether the LQP making the call to it is the new one being implemented here, or one of the older LQPs. It accepts data from a temporary file in the event that the new LQP makes a call to it, and uses the old methodology for the old LQPs. The modified version of the Result Reader may thus be shared among the LQPs implemented by Alec Champlin and the new LQP implemented here.

3.2.5 The LQP-object: Implementation

LQP-object File(s):
(1) sqlds4381.lsp (Appendix A.1)

The definition of the LQP-object will complete the implementation of the new LQP for the SQL/DS RDBMS on the remote IBM 4381 mainframe "sloan". The LQP-object is defined using KOREL (Knowledge-Object REpresentation Language) developed by Samuel P. Levine.

TASK 1: Define the LQP-object

The LQP-object for the SQL/DS RDBMS on "sloan" is defined using KOREL's *make-object* function and is named "sqlds-4381". An instance, appropriately called "sloandb" since the database to be accessed is on "sloan", is defined for the "sqlds-4381" object using the *create-instance* function.

Since the "sloandb" instance inherits all the attributes of "sqlds-4381", messages for the LQP may now be sent to either "sqlds-4381" or alternatively to "sloandb". Refer to the attributes defined for "sqlds-4381" in the LQP-object file **sqlds4381.lsp** (Appendix A.1). The reader will find that all the information required by the LQP-driver modules is stored in these attributes. These attribute - machine-name, type-of-DBMS, etc. - should be studied in order to understand where and how they are utilized by the LQP-driver.

Also included in the LQP-object definition are the :SELF-INFO, :GET-TABLES, :GET-COLUMNS, and :GET-DATA messages. When these methods are sent to the LQP as messages, they invoke the procedures *display-sqlds4381-self-info*, *get-sqlds4381-tables*, *get-sqlds4381-columns*, and *get-sqlds4381-data* respectively.

TASK 2: Define the procedures invoked by the LQP messages

The procedures *display-sqlds4381-self-info*, *get-sqlds4381-tables*, *get-sqlds4381-columns*, and *get-sqlds4381-data* are also defined in the file **sqlds4381.lsp**. These are the

procedures that control, and thereby automate, the four LQP-driver modules. These procedures should be studied closely in conjunction with the LQP-driver modules that they control.

This completes the implementation of the new LQP for the SQL/DS RDBMS on MIT Sloan School's IBM 4381 mainframe, "sloan". The LQP-driver is now 'virtualized', (automated) and the GQP of the CIS/TK system on "mit2e" may now send data retrieval messages to the LQP-object.

In the next chapter, a sample session of the working LQP is presented. The processing of data from module to module is clearly illustrated for the four messages that the LQP accepts.

Chapter 4

SAMPLE SESSION

This chapter provides a sample session of the working LQP implemented in the previous chapter.

In section (4.1), KOREL's *print-frame* function is used to view the LQP-object "sqlds-4381" and its instance "sloandb". Sections (4.2)-(4.5) describe the working of the LQP in processing the 4 messages :SELF-INFO, :GET-TABLES, :GET-COLUMNS, and :GET-DATA. User input is presented in **bold** typeface, and comments regarding the LQP screen output are presented in text font.

First, the file **demo-gg.lsp** (Appendix A.5) is loaded on the IBCL system.

>(load "/usr/cistk/biggie/ibm4381/demo-gg.lsp")

```
Loading /usr/cistk/biggie/ibm4381/frames.lsp
Finished loading /usr/cistk/biggie/ibm4381/frames.lsp
Loading /usr/cistk/biggie/ibm4381/korel.lsp
Finished loading /usr/cistk/biggie/ibm4381/korel.lsp
Loading /usr/cistk/biggie/ibm4381/sql.lsp
Finished loading /usr/cistk/biggie/ibm4381/sql.lsp
Loading /usr/cistk/biggie/ibm4381/connect.lsp
Finished loading /usr/cistk/biggie/ibm4381/connect.lsp
Loading /usr/cistk/biggie/ibm4381/read.lsp
Finished loading /usr/cistk/biggie/ibm4381/read.lsp
Loading /usr/cistk/biggie/ibm4381/informix2a.lsp
Finished loading /usr/cistk/biggie/ibm4381/informix2a.lsp
Loading /usr/cistk/biggie/ibm4381/informix2c.lsp
Finished loading /usr/cistk/biggie/ibm4381/informix2c.lsp
Loading /usr/cistk/biggie/ibm4381/oracle-rt.lsp
Finished loading /usr/cistk/biggie/ibm4381/oracle-rt.lsp
Loading /usr/cistk/biggie/ibm4381/sqlds4381.lsp
Finished loading /usr/cistk/biggie/ibm4381/sqlds4381.lsp


What level of messages do you want printed?
      --> Quiet, Terse, Normal, or Verbose? verbose

OK...If you change your mind, use 'LQP-MODE'. As in (lqp-mode 'quiet)
Finished loading demo-gg.lsp
T
```

The VERBOSE message level is selected from the 4 possible options - QUIET, TERSE, NORMAL and VERBOSE - for maximum screen display.

4.1 The LQP-object and its Instance

>(print-frame 'sqlds-4381)

```
SQLDS-4381:
    MACHINE-NAME:
        (VALUE sloan)
    TYPE-OF-DBMS:
        (VALUE sqlds)
    LOCAL-DBMS?:
        (VALUE NIL)
    DATABASE:
        (DEFAULT slinfo)
    DATABASE-DIRECTORY:
        (DEFAULT slinfo)
    COMM-SERVER-DIRECTORY:
        (DEFAULT /usr/cistk/biggie/ibm4381)
    COMMUNICATIONS-SCRIPT:
        (VALUE 4381FILE)
    EFFICIENT-COMM-SCRIPT:
        (VALUE 4381FILE)
    FTP-SCRIPT:
        (VALUE 4381FTP)
    TEMPORARY-FILE:
        (VALUE connect1.tmp)
    ACCOUNT:
        (DEFAULT ggidwani)
    PASSWORD:
        (DEFAULT mohina)
    METHODS:
        (MULTIPLE-VALUE-F T)
        (VALUE  (SELF-INFO DISPLAY-SQLDS4381-SELF-INFO)
                (GET-TABLES GET-SQLDS4381-TABLES)
                (GET-COLUMNS GET-SQLDS4381-COLUMNS)
                (GET-DATA GET-SQLDS4381-DATA))
    INSTANCES:
        (MULTIPLE-VALUE-F T)
        (VALUE SLOANDB)
```

The LQP-object "sqlds-4381" contains the required attributes and methods.

>(print-frame 'sloandb)

```
SLOANDB:
    SUPERIORS:
        (MULTIPLE-VALUE-F T)
        (VALUE SQLDS-4381)
    INSTANCE-OF:
        (MULTIPLE-VALUE-F T)
        (VALUE SQLDS-4381)
```

The "sloandb" instance inherits the attributes and methods of "sqlds-4381". Thus sending messages to "sloandb" is equivalent to sending them to "sqlds-4381".

4.2 The :SELF-INFO Message

>(send-message 'sqlds-4381 :self-info)

The :SELF-INFO message is sent to the "sqlds-4381" object.

```
S Q L / D S    O N    S L O A N
-----------------------------------

The SQL/DS Relational Database Management System is a computer program
that manages pieces of information (data) stored in a computer. The data
is contained in tables made up of vertical columns and horizontal rows.
The SQL/DS database uses the SQL query language which is common among
many of today's RDBMS systems.
```

The :SELF-INFO message invokes the *display-sqlds4381-self-info* procedure which takes no arguments. *display-sqlds4381-self-info* simply displays the text defined within its definition. No LQP-driver modules are used here.

4.3 The :GET-TABLES Message

>(send-message 'sloandb :get-tables)

The :GET-TABLES message is sent to the "sloandb" instance of "sqlds-4381". :GET-TABLES takes no arguments.

```
DBMS Query to be sent to machine sloan....

SELECT TNAME, CREATOR, DBSPACENAME, NCOLS FROM SYSTEM.SYSCATALOG
 ORDER BY TNAME, CREATOR


Fetching SQL/DS Table Data Requested....
Connecting to slinfo on machine sloan...
Done.

Transporting Result File to local machine....
Result File Retrieved.
```

The :GET-TABLES message does not use the Query Translator, but rather provides an SQL statement (hard-coded within the procedure), requesting table information, to the Communication Server as the DBMS Query. This is the same scheme used by the older LQPs for the INFORMIX and ORACLE RDBMSs implemented by Alec R. Champlin.

The Communication Server has connected to "sloan", accessed the "slinfo" database, queried it with the SQL statement for table information provided by *get-sqlds4381-tables*, and retrieved the Result File to the local machine "mit2e". The Result File contains data in the same format as on the remote machine "sloan". Notice that communication messages - the SQL SELECT statement and the 'end marker' - are also retrieved in this file. A list of all the tables on the "slinfo" database has been returned:

```
select tname, creator, dbspacename, ncols from system.syscatalog order by
 tname, creator
TNAME            CREATOR  DBSPACENAME        NCOLS
---------------- -------- ----------------- -----
ACCOUNTS         EMILY    PRODUCTION            8
ACTIVITY         S                              4
ACTIVITY         SIS_1    SIS1_ACTIVITY         4
```

ACTLEVEL	S		4
ACTLEVEL	SIS_1	SIS1_MISC	4
ACTTERM	S		5
ACTTERM	SIS_1	SIS1_ACTTERM	5
ACTTYPE	S		3
ACTTYPE	SIS_1	SIS1_MISC	3
ACTWGT	SIS	MISCELLANEOUS	5
ALLOC	ITZKOWIT	ALLOC_TEST	4
APPEDUC	PHD	PHD_APPEDUC	8
APPLICANT	Q	DSQ2STBT	5
APPLICATION	PHD	PHD_APPLICATION	25
APPRDR	PHD	PHD_APPRDR	5
APPRDRCOM	PHD	PHD_APPRDRCOM	5
AUTH_LIST	Q		11
BURDRPT	SIS	SIS_BURDRPT	10
BURDTYPE	SIS	PRODUCTION	2
COLUMN_LIST	Q		8
COMMAND_SYNONYMS	Q	DSQTSSYN	4
CONCAREA	PHD	PHD_MISC	2
COST_TABLE	AMOULTON	EXPLAIN_TABS	3
COUNTRY	PHD	PHD_COUNTRY	2
COUNTRY	SIS		2
COUNTRY_0	SIS	SIS_COUNTRY	2
DBDTPNL1	DBE	DBEDB00	15
DBDTPNL2	DBE	DBEDB00	26
DEPT	EMILY	PRODUCTION	3
DEPTENR	HJACOBY	DEPTENR	7
DEPTENR_DETAIL	HJACOBY	DEPTENR_DETAIL	7
D42	DBE		3
EMPBURD	SIS	SIS_EMPBURD	4
EMPHIST	S		5
EMPHIST	SIS_1	SIS1_EMPHIST	5
EMPLOYEE	S		11
EMPLOYEE	SIS_1	SIS1_EMPLOYEE	11
ENRCAT_GR	HJACOBY	ENRCAT_GR	6
ENRCAT_UG	HJACOBY	ENRCAT_UG	6
ENRPT	S		16
ENRPT	SIS_1	SIS1_ENRPT	16
ENRSIZE	S		3
ENRSIZE	SIS_1	SIS1_MISC	3
ENRWGT	SIS	MISCELLANEOUS	4
ERROR_LOG	Q	DSQTSLOG	5
FLOAD	S		8
FLOAD	SIS_1	SIS1_FLOAD	8
FORM_LIST	Q		4
GR_SUBJ	HJACOBY	JACOBY	1
HJ_ENR_DETAIL	AMOULTON	JACOBY	7
IDS	HJACOBY	JACOBY	4
IDS_DETAIL	HJACOBY	JACOBY	4
IDS2	HJACOBY	DSQTSDEF	4
INVENTORY	SQLDBA	SAMPLE	3
MAJOR	SIS		2
MAJOR_0	SIS	SIS_STUDMISC	2
MAP_STYPE	HJACOBY	JACOBY	3
MAPSUBJ_GR	HJACOBY	JACOBY	3

MAPSUBJ_UG	HJACOBY	JACOBY	3
MERGE	ITZKOWIT		4
MITCOURSE	S		3
MITCOURSE	SIS_1	SIS1_MISC	3
MITDEPT	S		2
MITDEPT	SIS_1	SIS1_MISC	2
OBJECT_DATA	Q	DSQTSCT3	5
OBJECT_DIRECTORY	Q	DSQTSCT1	6
OBJECT_REMARKS	Q	DSQTSCT2	4
OFFERING	S		8
OFFERING	SIS_1	SIS1_OFFERING	8
ORDERS	SQLDBA		4
ORG	DBE	DBEDB00	5
ORG	Q	DSQ2STBT	5
PLAN_TABLE	AMOULTON	EXPLAIN_TABS	12
PLAYTBL3	DBE	DBEDB00	4
POSITION	S		4
POSITION	SIS_1	SIS1_MISC	4
POSTYPE	S		3
POSTYPE	SIS_1	SIS1_MISC	3
PRIME1	SIS	PRODUCTION	6
PRIME2	SIS	PRODUCTION	4
PRIME4	SIS	PRODUCTION	2
PRIME5	SIS	PRODUCTION	2
PRIME6	SIS	PRODUCTION	2
PRIME7	SIS	PRODUCTION	4
PROC_LIST	Q		4
PRODUCTS	Q	DSQ2STBT	4
PROFILES	Q	DSQTSPRO	14
PROG	S		4
PROG	SIS_1	SIS1_MISC	4
PROGTYPE	S		3
PROGTYPE	SIS_1	SIS1_MISC	3
PSETEACH	S		3
PSETEACH	SIS_1	SIS1_PSETEACH	3
PSEWGT	SIS	MISCELLANEOUS	4
QMF_PFKEYS	AMOULTON	PROFILE	4
QMFTABLE_LIST	Q		5
QUERY_LIST	Q		5
QUOTATIONS	SQLDBA	SAMPLE	5
REFERENCE_TABLE	AMOULTON	EXPLAIN_TABS	8
REFLIST	DBE	DBEDB00	9
REGISTRATION	S		8
REGISTRATION	SIS_1	SIS1_REGISTRATION	8
RESOURCE_TABLE	Q	DSQTSGOV	5
RESOURCE_VIEW	Q		5
ROUTINE	EXAMPLE	ISQL	3
ROUTINE	SLINFO	ISQL	4
ROUTINE	SQLDBA	ISQL	4
SALES	Q	DSQ2STBT	5
SAVE_MAP_STYPE	HJACOBY	DSQTSDEF	3
SCHOOLS	SIS		3
SCHOOLS_0	SIS	SIS_SCHOOLS	3
SECTEACH	S		6
SECTEACH	SIS_1	SIS1_SECTEACH	6

```
SECTION            SIS        PRODUCTION          10
SSMAREA            S                               4
SSMAREA            SIS_1      SIS1_MISC            4
SSMGROUP           S                               3
SSMGROUP           SIS_1      SIS1_MISC            3
STAFF              DBE        DBEDB00              7
STAFF              Q          DSQ2STBT             7
STORED QUERIES     SQLDBA     ISQL                 3
STRUCTURE_TABLE    AMOULTON   EXPLAIN_TABS         6
STUDADD            SIS                            11
STUDADD_0          SIS        SIS_STUDADD         11
STUDEDUC           SIS                             5
STUDEDUC_0         SIS        SIS_STUDEDUC         5
STUDENT            SIS                            14
STUDENT_0          SIS        SIS_STUDENT         14
STUDP              SIS        PRODUCTION          16
SUBEVAL            S                              14
SUBEVAL            SIS_1      SIS1_SUBEVAL        14
SUBJECT            S                              16
SUBJECT            SIS_1      SIS1_SUBJECT        16
SUPPLIERS          SQLDBA     SAMPLE               3
SYSACCESS          SYSTEM     SYS0001              9
SYSCATALOG         SYSTEM     SYS0001             19
SYSCHARSETS        SYSTEM     SYS0001              3
SYSCOLAUTH         SYSTEM     SYS0001              6
SYSCOLUMNS         SYSTEM     SYS0001             15
SYSDBSPACES        SYSTEM     SYS0001             12
SYSDROP            SYSTEM     SYS0001              3
SYSINDEXES         SYSTEM     SYS0001             16
SYSOPTIONS         SYSTEM     SYS0001              3
SYSPROGAUTH        SYSTEM     SYS0001              6
SYSSYNONYMS        SYSTEM     SYS0001              4
SYSTABAUTH         SYSTEM     SYS0001             15
SYSTEXT1           SQLDBA     HELPTEXT             2
SYSTEXT2           SQLDBA     HELPTEXT             3
SYSUSAGE           SYSTEM     SYS0001              7
SYSUSERAUTH        SYSTEM     SYS0001              6
SYSUSERLIST        SQLDBA                          5
SYSVIEWS           SYSTEM     SYS0001              4
TABLE_LIST         Q                               4
TAPLAN             S                               6
TAPLAN             SIS_1      SIS1_TAPLAN          6
TERM               S                               8
TERM               SIS_1      SIS1_MISC            8
THESACT            SIS        PRODUCTION           6
UG_SUBJ            HJACOBY    JACOBY               1
VPROFILE           Q                              11
WORKLOAD           SIS        PRODUCTION           8
ZIP                SIS        PRODUCTION           3
************** End-of-Data ***************
```

Note that the Result File is not actually visible on the screen, but is included for the

reader's understanding. All the other screen output presented in this chapter is in fact visible in the VERBOSE mode.

The Data Filter filters the communication messages from the Result File, and converts the Result File to the standard Table File format:

```
Converting Result File to 'standard' Table File...
Done.
Result File after conversion to standard form...

TNAME|CREATOR|DBSPACENAME|NCOLS|
ACCOUNTS|EMILY|PRODUCTION|8|
ACTIVITY|S| |4|
ACTIVITY|SIS_1|SIS1_ACTIVITY|4|
ACTLEVEL|S| |4|
ACTLEVEL|SIS_1|SIS1_MISC|4|
ACTTERM|S| |5|
ACTTERM|SIS_1|SIS1_ACTTERM|5|
ACTTYPE|S| |3|
ACTTYPE|SIS_1|SIS1_MISC|3|
ACTWGT|SIS|MISCELLANEOUS|5|
ALLOC|ITZKOWIT|ALLOC_TEST|4|
APPEDUC|PHD|PHD_APPEDUC|8|
APPLICANT|Q|DSQ2STBT|5|
APPLICATION|PHD|PHD_APPLICATION|25|
APPRDR|PHD|PHD_APPRDR|5|
APPRDRCOM|PHD|PHD_APPRDRCOM|5|
AUTH_LIST|Q| |11|
BURDRPT|SIS|SIS_BURDRPT|10|
BURDTYPE|SIS|PRODUCTION|2|
COLUMN_LIST|Q| |8|
COMMAND_SYNONYMS|Q|DSQTSSYN|4|
CONCAREA|PHD|PHD_MISC|2|
COST_TABLE|AMOULTON|EXPLAIN_TABS|3|
COUNTRY|PHD|PHD_COUNTRY|2|
COUNTRY|SIS| |2|
COUNTRY_0|SIS|SIS_COUNTRY|2|
DBDTPNL1|DBE|DBEDB00|15|
DBDTPNL2|DBE|DBEDB00|26|
DEPT|EMILY|PRODUCTION|3|
DEPTENR|HJACOBY|DEPTENR|7|
DEPTENR_DETAIL|HJACOBY|DEPTENR_DETAIL|7|
D42|DBE| |3|
EMPBURD|SIS|SIS_EMPBURD|4|
EMPHIST|S| |5|
EMPHIST|SIS_1|SIS1_EMPHIST|5|
EMPLOYEE|S| |11|
EMPLOYEE|SIS_1|SIS1_EMPLOYEE|11|
ENRCAT_GR|HJACOBY|ENRCAT_GR|6|
ENRCAT_UG|HJACOBY|ENRCAT_UG|6|
```

```
ENRPT|S|  |16|
ENRPT|SIS_1|SIS1_ENRPT|16|
ENRSIZE|S|  |3|
ENRSIZE|SIS_1|SIS1_MISC|3|
ENRWGT|SIS|MISCELLANEOUS|4|
ERROR_LOG|Q|DSQTSLOG|5|
FLOAD|S|  |8|
FLOAD|SIS_1|SIS1_FLOAD|8|
FORM_LIST|Q|  |4|
GR_SUBJ|HJACOBY|JACOBY|1|
HJ_ENR_DETAIL|AMOULTON|JACOBY|7|
IDS|HJACOBY|JACOBY|4|
IDS_DETAIL|HJACOBY|JACOBY|4|
IDS2|HJACOBY|DSQTSDEF|4|
INVENTORY|SQLDBA|SAMPLE|3|
MAJOR|SIS|  |2|
MAJOR_0|SIS|SIS_STUDMISC|2|
MAP_STYPE|HJACOBY|JACOBY|3|
MAPSUBJ_GR|HJACOBY|JACOBY|3|
MAPSUBJ_UG|HJACOBY|JACOBY|3|
MERGE|ITZKOWIT|  |4|
MITCOURSE|S|  |3|
MITCOURSE|SIS_1|SIS1_MISC|3|
MITDEPT|S|  |2|
MITDEPT|SIS_1|SIS1_MISC|2|
OBJECT_DATA|Q|DSQTSCT3|5|
OBJECT_DIRECTORY|Q|DSQTSCT1|6|
OBJECT_REMARKS|Q|DSQTSCT2|4|
OFFERING|S|  |8|
OFFERING|SIS_1|SIS1_OFFERING|8|
ORDERS|SQLDBA|  |4|
ORG|DBE|DBEDB00|5|
ORG|Q|DSQ2STBT|5|
PLAN_TABLE|AMOULTON|EXPLAIN_TABS|12|
PLAYTBL3|DBE|DBEDB00|4|
POSITION|S|  |4|
POSITION|SIS_1|SIS1_MISC|4|
POSTYPE|S|  |3|
POSTYPE|SIS_1|SIS1_MISC|3|
PRIME1|SIS|PRODUCTION|6|
PRIME2|SIS|PRODUCTION|4|
PRIME4|SIS|PRODUCTION|2|
PRIME5|SIS|PRODUCTION|2|
PRIME6|SIS|PRODUCTION|2|
PRIME7|SIS|PRODUCTION|4|
PROC_LIST|Q|  |4|
PRODUCTS|Q|DSQ2STBT|4|
PROFILES|Q|DSQTSPRO|14|
PROG|S|  |4|
PROG|SIS_1|SIS1_MISC|4|
PROGTYPE|S|  |3|
PROGTYPE|SIS_1|SIS1_MISC|3|
PSETEACH|S|  |3|
PSETEACH|SIS_1|SIS1_PSETEACH|3|
PSEWGT|SIS|MISCELLANEOUS|4|
```

```
QMF_PFKEYS|AMOULTON|PROFILE|4|
QMFTABLE_LIST|Q|  |5|
QUERY_LIST|Q|  |5|
QUOTATIONS|SQLDBA|SAMPLE|5|
REFERENCE_TABLE|AMOULTON|EXPLAIN_TABS|8|
REFLIST|DBE|DBEDB00|9|
REGISTRATION|S|  |8|
REGISTRATION|SIS_1|SIS1_REGISTRATION|8|
RESOURCE_TABLE|Q|DSQTSGOV|5|
RESOURCE_VIEW|Q|  |5|
ROUTINE|EXAMPLE|ISQL|3|
ROUTINE|SLINFO|ISQL|4|
ROUTINE|SQLDBA|ISQL|4|
SALES|Q|DSQ2STBT|5|
SAVE_MAP_STYPE|HJACOBY|DSQTSDEF|3|
SCHOOLS|SIS|  |3|
SCHOOLS_0|SIS|SIS_SCHOOLS|3|
SECTEACH|S|  |6|
SECTEACH|SIS_1|SIS1_SECTEACH|6|
SECTION|SIS|PRODUCTION|10|
SSMAREA|S|  |4|
SSMAREA|SIS_1|SIS1_MISC|4|
SSMGROUP|S|  |3|
SSMGROUP|SIS_1|SIS1_MISC|3|
STAFF|DBE|DBEDB00|7|
STAFF|Q|DSQ2STBT|7|
STORED QUERIES|SQLDBA|ISQL|3|
STRUCTURE_TABLE|AMOULTON|EXPLAIN_TABS|6|
STUDADD|SIS|  |11|
STUDADD_0|SIS|SIS_STUDADD|11|
STUDEDUC|SIS|  |5|
STUDEDUC_0|SIS|SIS_STUDEDUC|5|
STUDENT|SIS|  |14|
STUDENT_0|SIS|SIS_STUDENT|14|
STUDP|SIS|PRODUCTION|16|
SUBEVAL|S|  |14|
SUBEVAL|SIS_1|SIS1_SUBEVAL|14|
SUBJECT|S|  |16|
SUBJECT|SIS_1|SIS1_SUBJECT|16|
SUPPLIERS|SQLDBA|SAMPLE|3|
SYSACCESS|SYSTEM|SYS0001|9|
SYSCATALOG|SYSTEM|SYS0001|19|
SYSCHARSETS|SYSTEM|SYS0001|3|
SYSCOLAUTH|SYSTEM|SYS0001|6|
SYSCOLUMNS|SYSTEM|SYS0001|15|
SYSDBSPACES|SYSTEM|SYS0001|12|
SYSDROP|SYSTEM|SYS0001|3|
SYSINDEXES|SYSTEM|SYS0001|16|
SYSOPTIONS|SYSTEM|SYS0001|3|
SYSPROGAUTH|SYSTEM|SYS0001|6|
SYSSYNONYMS|SYSTEM|SYS0001|4|
SYSTABAUTH|SYSTEM|SYS0001|15|
SYSTEXT1|SQLDBA|HELPTEXT|2|
SYSTEXT2|SQLDBA|HELPTEXT|3|
SYSUSAGE|SYSTEM|SYS0001|7|
```

```
SYSUSERAUTH|SYSTEM|SYS0001|6|
SYSUSERLIST|SQLDBA| |5|
SYSVIEWS|SYSTEM|SYS0001|4|
TABLE_LIST|Q| |4|
TAPLAN|S| |6|
TAPLAN|SIS_1|SIS1_TAPLAN|6|
TERM|S| |8|
TERM|SIS_1|SIS1_MISC|8|
THESACT|SIS|PRODUCTION|6|
UG_SUBJ|HJACOBY|JACOBY|1|
VPROFILE|Q| |11|
WORKLOAD|SIS|PRODUCTION|8|
ZIP|SIS|PRODUCTION|3|
```

Finally, the Result Reader converts the Table File to the Result List:

```
Table File converted to Result List....Done.

(("TNAME" "CREATOR" "DBSPACENAME" "NCOLS")
 ("ACCOUNTS" "EMILY" "PRODUCTION" "8") ("ACTIVITY" "S" " " "4")
 ("ACTIVITY" "SIS_1" "SIS1_ACTIVITY" "4") ("ACTLEVEL" "S" " " "4")
 ("ACTLEVEL" "SIS_1" "SIS1_MISC" "4") ("ACTTERM" "S" " " "5")
 ("ACTTERM" "SIS_1" "SIS1_ACTTERM" "5") ("ACTTYPE" "S" " " "3")
 ("ACTTYPE" "SIS_1" "SIS1_MISC" "3")
 ("ACTWGT" "SIS" "MISCELLANEOUS" "5")
 ("ALLOC" "ITZKOWIT" "ALLOC_TEST" "4")
 ("APPEDUC" "PHD" "PHD_APPEDUC" "8") ("APPLICANT" "Q" "DSQ2STBT" "5")
 ("APPLICATION" "PHD" "PHD_APPLICATION" "25")
 ("APPRDR" "PHD" "PHD_APPRDR" "5")
 ("APPRDRCOM" "PHD" "PHD_APPRDRCOM" "5") ("AUTH_LIST" "Q" " " "11")
 ("BURDRPT" "SIS" "SIS_BURDRPT" "10")
 ("BURDTYPE" "SIS" "PRODUCTION" "2") ("COLUMN_LIST" "Q" " " "8")
 ("COMMAND_SYNONYMS" "Q" "DSQTSSYN" "4")
 ("CONCAREA" "PHD" "PHD_MISC" "2")
 ("COST_TABLE" "AMOULTON" "EXPLAIN_TABS" "3")
 ("COUNTRY" "PHD" "PHD_COUNTRY" "2") ("COUNTRY" "SIS" " " "2")
 ("COUNTRY_0" "SIS" "SIS_COUNTRY" "2")
 ("DBDTPNL1" "DBE" "DBEDB00" "15") ("DBDTPNL2" "DBE" "DBEDB00" "26")
 ("DEPT" "EMILY" "PRODUCTION" "3") ("DEPTENR" "HJACOBY" "DEPTENR" "7")
 ("DEPTENR_DETAIL" "HJACOBY" "DEPTENR_DETAIL" "7")
 ("D42" "DBE" " " "3") ("EMPBURD" "SIS" "SIS_EMPBURD" "4")
 ("EMPHIST" "S" " " "5") ("EMPHIST" "SIS_1" "SIS1_EMPHIST" "5")
 ("EMPLOYEE" "S" " " "11") ("EMPLOYEE" "SIS_1" "SIS1_EMPLOYEE" "11")
 ("ENRCAT_GR" "HJACOBY" "ENRCAT_GR" "6")
 ("ENRCAT_UG" "HJACOBY" "ENRCAT_UG" "6") ("ENRPT" "S" " " "16")
 ("ENRPT" "SIS_1" "SIS1_ENRPT" "16") ("ENRSIZE" "S" " " "3")
 ("ENRSIZE" "SIS_1" "SIS1_MISC" "3")
 ("ENRWGT" "SIS" "MISCELLANEOUS" "4") ("ERROR_LOG" "Q" "DSQTSLOG" "5")
 ("FLOAD" "S" " " "8") ("FLOAD" "SIS_1" "SIS1_FLOAD" "8")
 ("FORM_LIST" "Q" " " "4") ("GR_SUBJ" "HJACOBY" "JACOBY" "1")
 ("HJ_ENR_DETAIL" "AMOULTON" "JACOBY" "7")
```

```
("IDS" "HJACOBY" "JACOBY" "4") ("IDS_DETAIL" "HJACOBY" "JACOBY" "4")
("IDS2" "HJACOBY" "DSQTSDEF" "4") ("INVENTORY" "SQLDBA" "SAMPLE" "3")
("MAJOR" "SIS" " " "2") ("MAJOR_0" "SIS" "SIS_STUDMISC" "2")
("MAP_STYPE" "HJACOBY" "JACOBY" "3")
("MAPSUBJ_GR" "HJACOBY" "JACOBY" "3")
("MAPSUBJ_UG" "HJACOBY" "JACOBY" "3") ("MERGE" "ITZKOWIT" " " "4")
("MITCOURSE" "S" " " "3") ("MITCOURSE" "SIS_1" "SIS1_MISC" "3")
("MITDEPT" "S" " " "2") ("MITDEPT" "SIS_1" "SIS1_MISC" "2")
("OBJECT_DATA" "Q" "DSQTSCT3" "5")
("OBJECT_DIRECTORY" "Q" "DSQTSCT1" "6")
("OBJECT_REMARKS" "Q" "DSQTSCT2" "4") ("OFFERING" "S" " " "8")
("OFFERING" "SIS_1" "SIS1_OFFERING" "8") ("ORDERS" "SQLDBA" " " "4")
("ORG" "DBE" "DBEDB00" "5") ("ORG" "Q" "DSQ2STBT" "5")
("PLAN_TABLE" "AMOULTON" "EXPLAIN_TABS" "12")
("PLAYTBL3" "DBE" "DBEDB00" "4") ("POSITION" "S" " " "4")
("POSITION" "SIS_1" "SIS1_MISC" "4") ("POSTYPE" "S" " " "3")
("POSTYPE" "SIS_1" "SIS1_MISC" "3") ("PRIME1" "SIS" "PRODUCTION" "6")
("PRIME2" "SIS" "PRODUCTION" "4") ("PRIME4" "SIS" "PRODUCTION" "2")
("PRIME5" "SIS" "PRODUCTION" "2") ("PRIME6" "SIS" "PRODUCTION" "2")
("PRIME7" "SIS" "PRODUCTION" "4") ("PROC_LIST" "Q" " " "4")
("PRODUCTS" "Q" "DSQ2STBT" "4") ("PROFILES" "Q" "DSQTSPRO" "14")
("PROG" "S" " " "4") ("PROG" "SIS_1" "SIS1_MISC" "4")
("PROGTYPE" "S" " " "3") ("PROGTYPE" "SIS_1" "SIS1_MISC" "3")
("PSETEACH" "S" " " "3") ("PSETEACH" "SIS_1" "SIS1_PSETEACH" "3")
("PSEWGT" "SIS" "MISCELLANEOUS" "4")
("QMF_PFKEYS" "AMOULTON" "PROFILE" "4") ("QMFTABLE_LIST" "Q" " " "5")
("QUERY_LIST" "Q" " " "5") ("QUOTATIONS" "SQLDBA" "SAMPLE" "5")
("REFERENCE_TABLE" "AMOULTON" "EXPLAIN_TABS" "8")
("REFLIST" "DBE" "DBEDB00" "9") ("REGISTRATION" "S" " " "8")
("REGISTRATION" "SIS_1" "SIS1_REGISTRATION" "8")
("RESOURCE_TABLE" "Q" "DSQTSGOV" "5") ("RESOURCE_VIEW" "Q" " " "5")
("ROUTINE" "EXAMPLE" "ISQL" "3") ("ROUTINE" "SLINFO" "ISQL" "4")
("ROUTINE" "SQLDBA" "ISQL" "4") ("SALES" "Q" "DSQ2STBT" "5")
("SAVE_MAP_STYPE" "HJACOBY" "DSQTSDEF" "3") ("SCHOOLS" "SIS" " " "3")
("SCHOOLS_0" "SIS" "SIS_SCHOOLS" "3") ("SECTEACH" "S" " " "6")
("SECTEACH" "SIS_1" "SIS1_SECTEACH" "6")
("SECTION" "SIS" "PRODUCTION" "10") ("SSMAREA" "S" " " "4")
("SSMAREA" "SIS_1" "SIS1_MISC" "4") ("SSMGROUP" "S" " " "3")
("SSMGROUP" "SIS_1" "SIS1_MISC" "3") ("STAFF" "DBE" "DBEDB00" "7")
("STAFF" "Q" "DSQ2STBT" "7") ("STORED QUERIES" "SQLDBA" "ISQL" "3")
("STRUCTURE_TABLE" "AMOULTON" "EXPLAIN_TABS" "6")
("STUDADD" "SIS" " " "11") ("STUDADD_0" "SIS" "SIS_STUDADD" "11")
("STUDEDUC" "SIS" " " "5") ("STUDEDUC_0" "SIS" "SIS_STUDEDUC" "5")
("STUDENT" "SIS" " " "14") ("STUDENT_0" "SIS" "SIS_STUDENT" "14")
("STUDP" "SIS" "PRODUCTION" "16") ("SUBEVAL" "S" " " "14")
("SUBEVAL" "SIS_1" "SIS1_SUBEVAL" "14") ("SUBJECT" "S" " " "16")
("SUBJECT" "SIS_1" "SIS1_SUBJECT" "16")
("SUPPLIERS" "SQLDBA" "SAMPLE" "3")
("SYSACCESS" "SYSTEM" "SYS0001" "9")
("SYSCATALOG" "SYSTEM" "SYS0001" "19")
("SYSCHARSETS" "SYSTEM" "SYS0001" "3")
("SYSCOLAUTH" "SYSTEM" "SYS0001" "6")
("SYSCOLUMNS" "SYSTEM" "SYS0001" "15")
("SYSDBSPACES" "SYSTEM" "SYS0001" "12")
("SYSDROP" "SYSTEM" "SYS0001" "3")
```

```
("SYSINDEXES" "SYSTEM" "SYS0001" "16")
("SYSOPTIONS" "SYSTEM" "SYS0001" "3")
("SYSPROGAUTH" "SYSTEM" "SYS0001" "6")
("SYSSYNONYMS" "SYSTEM" "SYS0001" "4")
("SYSTABAUTH" "SYSTEM" "SYS0001" "15")
("SYSTEXT1" "SQLDBA" "HELPTEXT" "2")
("SYSTEXT2" "SQLDBA" "HELPTEXT" "3")
("SYSUSAGE" "SYSTEM" "SYS0001" "7")
("SYSUSERAUTH" "SYSTEM" "SYS0001" "6")
("SYSUSERLIST" "SQLDBA" " " "5") ("SYSVIEWS" "SYSTEM" "SYS0001" "4")
("TABLE_LIST" "Q" " " "4") ("TAPLAN" "S" " " "6")
("TAPLAN" "SIS_1" "SIS1_TAPLAN" "6") ("TERM" "S" " " "8")
("TERM" "SIS_1" "SIS1_MISC" "8") ("THESACT" "SIS" "PRODUCTION" "6")
("UG_SUBJ" "HJACOBY" "JACOBY" "1") ("VPROFILE" "Q" " " "11")
("WORKLOAD" "SIS" "PRODUCTION" "8") ("ZIP" "SIS" "PRODUCTION" "3"))
```

4.4 The :GET-COLUMNS Message

>(send-message 'sqlds-4381 :get-columns 'employee)

The :GET-COLUMNS message takes as its single argument a valid table name - here the table named employee.

```
DBMS Query to be sent to machine sloan....

SELECT DISTINCT CNAME, COLTYPE, NULLS FROM SYSTEM.SYSCOLUMNS
 WHERE TNAME = 'EMPLOYEE' ORDER BY CNAME

Fetching SQL/DS Data Requested....
Connecting to slinfo on machine sloan...
Done.

Transporting Result File to local machine....
Result File Retrieved.
```

The :GET-COLUMNS message also does not use the Query Translator module, but provides the Communication Server with a hard-coded SQL statement (DBMS Query) representing a request for the column names on the specified table.

The Communication Server has connected to "sloan", accessed the "slinfo" database, queried it with the SQL statement for column information provided by *get-sqlds4381-columns*, and retrieved the Result File (not visible as screen output).

```
SELECT DISTINCT CNAME, COLTYPE, NULLS FROM SYSTEM.SYSCOLUMNS
 WHERE TNAME = 'EMPLOYEE' ORDER BY CNAME
CNAME    COLTYPE   NULLS
-------- --------- -----
AREAABB  CHAR      N
COMPID   CHAR      N
EFFTERM  SMALLINT  N
EMPABB   CHAR      N
EMPNAME  CHAR      N
EMPNUM   CHAR      N
EMPTYPE  CHAR      N
ENDTERM  SMALLINT  N
OFFICE   CHAR      N
PHONE    CHAR      N
POSABB   CHAR      N
************** End-of-Data ***************
```

The Data Filter converts the Result File to the standard Table File:

```
Converting Result File to 'standard' Table File...
Done.
Result file after conversion to standard form..

CNAME|COLTYPE|NULLS|
AREAABB|CHAR|N|
COMPID|CHAR|N|
EFFTERM|SMALLINT|N|
EMPABB|CHAR|N|
EMPNAME|CHAR|N|
EMPNUM|CHAR|N|
EMPTYPE|CHAR|N|
ENDTERM|SMALLINT|N|
OFFICE|CHAR|N|
PHONE|CHAR|N|
POSABB|CHAR|N|
```

The Result Reader converts the Table File to the Result List:

```
Table File converted to Result List...Done.

(("CNAME" "COLTYPE" "NULLS") ("AREAABB" "CHAR" "N")
 ("COMPID" "CHAR" "N") ("EFFTERM" "SMALLINT" "N") ("EMPABB" "CHAR" "N")
 ("EMPNAME" "CHAR" "N") ("EMPNUM" "CHAR" "N") ("EMPTYPE" "CHAR" "N")
 ("ENDTERM" "SMALLINT" "N") ("OFFICE" "CHAR" "N") ("PHONE" "CHAR" "N")
```

```
("POSABB" "CHAR" "N"))
```

4.5 The :GET-DATA Message

>(send-message 'sloandb :get-data '(s.employee (empname empnum emptype) (OR (= emptype "X") (= emptype "R"))))

The :GET-DATA message takes an Abstract Local Query (ALQ) as its argument.

```
PARSE-SQL-TNAME -- Converting symbol S.EMPLOYEE into a string.
PARSE-SQL-COLMN -- Converting symbol EMPTYPE into a string.
PARSE-SQL-COLMN -- Converting symbol EMPTYPE into a string.
PARSE-SQL-COLMN -- Converting symbol EMPNAME into a string.
PARSE-SQL-COLMN -- Converting symbol EMPNUM into a string.
PARSE-SQL-COLMN -- Converting symbol EMPTYPE into a string.

DBMS Query to be sent to machine sloan....

SELECT EMPNAME, EMPNUM, EMPTYPE FROM S.EMPLOYEE WHERE (EMPTYPE = 'X')
 OR (EMPTYPE = 'R')
```

It should now be clear how the :GET-DATA allows the user to create arbitrary SQL statements, as opposed to the rigid, 'hard-coded' schemes utilized by the :GET-TABLES and :GET-COLUMNS messages. :GET-DATA uses the Query Translator to convert an ALQ to the SQL SELECT statement that it represents. This, in fact, is the only difference between the :GET-DATA message and the :GET-TABLES and :GET-COLUMNS messages. Note how the names of the column names, the table name, and the contents of the conditionals are converted to 'strings'. Also note how the conditionals perform a conditional search of the "X" and "R" employee types.

```
Fetching SQL/DS Data Requested....
Connecting to slinfo on machine sloan...
Done.

Transporting Result File to local machine....
```

```
Result File Retrieved.
```

The Communication Server has connected to "sloan", accessed the "slinfo" database, queried it with the SQL statement created by the Query Translator, and retrieved the requested data as the Result File (not visible as screen output):

```
select empname, empnum, emptype from s.employee where (emptype = 'X')
 OR (emptype = 'R')
EMPNAME                                  EMPNUM EMPTYPE
---------------------------------------- ------ -------
Alexander, Sidney                        0001   X
Robinson, Richard                        0018   X
Bottiglia, William                       0105   X
Hekimian, J.                             0162   X
Shapiro, Eli                             0236   X
Toong, Hoo-Min                           0251   R
Senge, Peter                             0264   R
Bowman, Edward                           0302   X
Johnson, Howard                          0417   X
Ahuja, Ravindra                          0498   R
Bowles, Edward                           0540   X
Brooks, E                                0541   X
Durand, David                            0542   X
Moore, Leo                               0543   X
Myers, Charles                           0544   X
Bullen, Christine                        0546   R
Davidson, Frank                          0547   R
Egan, Eleanor                            0549   R
Gould, Janet                             0551   R
Graham, Alan                             0552   R
Gupta, Amar                              0553   R
Hollinger, Peter                         0554   R
Katz, Ralph                              0556   R
Pugh, Alexander                          0560   R
Quillard, Judith                         0561   R
Samarov, Alexander                       0562   R
Samuel, Roger                            0563   R
White, Patricia                          0565   R
Wilson, Diane                            0566   R
Invernizzi, E                            0569   X
Stevens, Chandler                        0577   X
Benjamin, Robert                         0578   X
Johansen, Robert                         0579   R
Short, James                             0631   R
Antrim, Lance                            0639   R
Kaminka, Shlomit                         0646   X
Luberto, Gaetano                         0647   X
Martin, Andrew                           0649   R
Yang, Shi-Shen                           0654   X
```

```
Arino, Miguel                          0686   X
Curley, Kathleen                       0689   X
Esteban, Jesus                         0690   X
Gardner, Margaret                      0691   X
Lasserre, Pierre                       0692   X
Triantis, Alexander                    0694   X
Villager, Daniel                       0695   X
Kirsch, John                           0696   R
Martinez, Jon                          0697   R
Patterson, Seymour                     0698   R
Wolfson, A. Mark                       0701   X
Various Faculty                        9998   X
Out of Course Staff                    9999   X
************** End-of-Data **************
```

The Data Filter converts the Result File to the standard Table File:

```
Converting Result File to 'standard' Table File...
Done.

EMPNAME|EMPNUM|EMPTYPE|
Alexander, Sidney|0001|X|
Robinson, Richard|0018|X|
Bottiglia, William|0105|X|
Hekimian, J.|0162|X|
Shapiro, Eli|0236|X|
Toong, Hoo-Min|0251|R|
Senge, Peter|0264|R|
Bowman, Edward|0302|X|
Johnson, Howard|0417|X|
Ahuja, Ravindra|0498|R|
Bowles, Edward|0540|X|
Brooks, E|0541|X|
Durand, David|0542|X|
Moore, Leo|0543|X|
Myers, Charles|0544|X|
Bullen, Christine|0546|R|
Davidson, Frank|0547|R|
Egan, Eleanor|0549|R|
Gould, Janet|0551|R|
Graham, Alan|0552|R|
Gupta, Amar|0553|R|
Hollinger, Peter|0554|R|
Katz, Ralph|0556|R|
Pugh, Alexander|0560|R|
Quillard, Judith|0561|R|
Samarov, Alexander|0562|R|
Samuel, Roger|0563|R|
White, Patricia|0565|R|
Wilson, Diane|0566|R|
Invernizzi, E|0569|X|
```

```
Stevens, Chandler|0577|X|
Benjamin, Robert|0578|X|
Johansen, Robert|0579|R|
Short, James|0631|R|
Antrim, Lance|0639|R|
Kaminka, Shlomit|0646|X|
Luberto, Gaetano|0647|X|
Martin, Andrew|0649|R|
Yang, Shi-Shen|0654|X|
Arino, Miguel|0686|X|
Curley, Kathleen|0689|X|
Esteban, Jesus|0690|X|
Gardner, Margaret|0691|X|
Lasserre, Pierre|0692|X|
Triantis, Alexander|0694|X|
Villager, Daniel|0695|X|
Kirsch, John|0696|R|
Martinez, Jon|0697|R|
Patterson, Seymour|0698|R|
Wolfson, A. Mark|0701|X|
Various Faculty|9998|X|
Out of Course Staff|9999|X|
```

The Result Reader converts the Table File to the Result List:

```
Table File converted to Result List....Done.

(("EMPNAME" "EMPNUM" "EMPTYPE") ("Alexander, Sidney" "0001" "X")
 ("Robinson, Richard" "0018" "X") ("Bottiglia, William" "0105" "X")
 ("Hekimian, J." "0162" "X") ("Shapiro, Eli" "0236" "X")
 ("Toong, Hoo-Min" "0251" "R") ("Senge, Peter" "0264" "R")
 ("Bowman, Edward" "0302" "X") ("Johnson, Howard" "0417" "X")
 ("Ahuja, Ravindra" "0498" "R") ("Bowles, Edward" "0540" "X")
 ("Brooks, E" "0541" "X") ("Durand, David" "0542" "X")
 ("Moore, Leo" "0543" "X") ("Myers, Charles" "0544" "X")
 ("Bullen, Christine" "0546" "R") ("Davidson, Frank" "0547" "R")
 ("Egan, Eleanor" "0549" "R") ("Gould, Janet" "0551" "R")
 ("Graham, Alan" "0552" "R") ("Gupta, Amar" "0553" "R")
 ("Hollinger, Peter" "0554" "R") ("Katz, Ralph" "0556" "R")
 ("Pugh, Alexander" "0560" "R") ("Quillard, Judith" "0561" "R")
 ("Samarov, Alexander" "0562" "R") ("Samuel, Roger" "0563" "R")
 ("White, Patricia" "0565" "R") ("Wilson, Diane" "0566" "R")
 ("Invernizzi, E" "0569" "X") ("Stevens, Chandler" "0577" "X")
 ("Benjamin, Robert" "0578" "X") ("Johansen, Robert" "0579" "R")
 ("Short, James" "0631" "R") ("Antrim, Lance" "0639" "R")
 ("Kaminka, Shlomit" "0646" "X") ("Luberto, Gaetano" "0647" "X")
 ("Martin, Andrew" "0649" "R") ("Yang, Shi-Shen" "0654" "X")
 ("Arino, Miguel" "0686" "X") ("Curley, Kathleen" "0689" "X")
 ("Esteban, Jesus" "0690" "X") ("Gardner, Margaret" "0691" "X")
 ("Lasserre, Pierre" "0692" "X") ("Triantis, Alexander" "0694" "X")
 ("Villager, Daniel" "0695" "X") ("Kirsch, John" "0696" "R")
```

```
("Martinez, Jon" "0697" "R") ("Patterson, Seymour" "0698" "R")
("Wolfson, A. Mark" "0701" "X") ("Various Faculty" "9998" "X")
("Out of Course Staff" "9999" "X"))
```

This concludes the sample session.

Chapter 5

SYSTEM IDIOSYNCRACIES AND POSSIBLE PROBLEMS

A robust LQP has been implemented for the SQL/DS RDBMS on MIT Sloan School's IBM 4381 mainframe. There are, however, idiosyncracies involved with the retrieving of data from the remote IBM 4381 mainframe to the local AT&T 3B2 machine.

It is possible that outside parties - like Systems Administrators - may make inadvertent changes to the systems being used by the new LQP, thereby rendering it non-functional. It is therefore important that CIS/TK system developers take note of these idiosyncracies.

Section (6.1) provides a list of idiosyncracies that could result in the breakdown of the SQL/DS RDBMS LQP. These idiosyncracies are presented in the form of checks to be made, under the assumption that section (5.1) will be particularly important in the event that the LQP is found to fail. Section (5.2) is devoted to a particularly noteworthy idiosyncracy of the SQL/DS SQL format. This section is intendedfor the use of developers of the higher levels of the CIS/TK system.

5.1 SQL/DS LQP Idiosyncracy Check

1. Make sure that a user account is defined on the remote IBM 4381 machine. The **<user-account>** name and the **<password>** for the account should be the same as those stored in the LQP-object as attributes. If it is found that an account has been terminated, contact the IBM 4381 mainframe Systems Administrator.
2. A working directory is required on the IBM 4381 user account. This working directory must be provided with both read and write passwords that are identical to the **<password>** for the user account. The working directories are used by the Communication Server script files. (See section (3.2.2).)
3. The **4381SEL** EXECutive program, a modified version of the RXSELECT EXECutive program, needs to accomodate as many charecters in a line, and

as many rows of data, as are found in the query result output format. The variable *maxlength* which truncates lines after a specified number of charecters, is presently set at 508 charecters. The *maxlns* variable that causes a specified number of rows to be returned at a time, is presently set at 10,000. These values may be changed accordingly if it is found that lines of data are truncated or rows of data are lost. (See section (3.2.2).)

4. The UNIX operating system on the AT&T 3B2 machine can only handle I/O streams. A line mode I/O environment should therefore be specified on the otherwise full-screen (block) mode IBM machine. This is done by issuing the **ac(noprof** and **sysprof3** commands to the IBM 4381 mainframe from the script file **4381FILE**. (see section (3.2.2).)
5. The **-n** option must be used for ftp. This is critical to the correct functioning of the **4381FTP** script file, as described in section (3.2.2).)
6. Timing problems may also cause the LQP script files to fail, depending upon the processing loads on the 2 machines being used. If timing problems are suspected, the **sleep** commands in the script files **4381FILE** and **4381FTP** should be provided with a larger sleep period between commands to be issued to the remote IBM 4381 machine.

5.2 SQL/DS SQL Format Idiosyncracies

The SQL/DS RDBMS uses the concept of a 'creator'. Within the RDBMS, several databases may be defined. Within each of these databases are defined a number of tables. Each table in a database has a creator. Every table is referenced by its creator in an SQL/DS SELECT statement. Specifically, a table **<table>** created by **<creator>** is referred to as **<creator>.<table>** in a SELECT statement. It is, however, referred to simply as **<table>** when the RDBMS uses it to retrieve a list of column names. Thus, for the :GET-DATA message, when an ALQ is sent to the LQP, the table name should have the form **<creator>.<table>**, but for the :GET-COLUMNS message, it should be left as **<table>**.

To illustrate, consider the usage of the table name **employee** created by the creator s, for the :GET-COLUMNS and :GET-DATA messages respectively, below:

```
THE :GET-COLUMNS MESSAGE:

(send-message 'sloandb :get-columns 'employee)
```

The table is referred to simply as **employee** for the :GET-COLUMNS message.

THE :GET-DATA MESSAGE:

```
(send-message 'sloandb :get-data '(s.employee (empname empnum)))
```

The table is referred to as **s.employee** in the :GET-DATA message's ALQ.

Chapter 6

CONCLUSION

The CIS/TK system proposes to eliminate information boundaries by integrating multiple, remote DBMSs. Local Query Processors (LQPs) provide the connectivity/interface between the upper, intelligent levels of the CIS/TK system and the remote DBMSs in its access field.

The object-oriented implementation of an LQP for the addition of a Relational DBMS to the CIS/TK system's access field was studied closely. This involved implementing a LQP-driver and an LQP-object that automates its functioning. Where it was found that existing code performed the same function as that required by an LQP-driver module, this code was shared with the new LQP. The idiosyncracies of the systems exchanging data were studied closely in the implementation process.

In this paper, descriptive guidelines have been established for the inclusion of a new RDBMS in the access field of the CIS/TK system.

6.1 Improvement of the SQL/DS RDBMS LQP

1. Presently, the **ftp** file transfer protocol is used to retrieve the query result output from a temporary file on the remote machine to a temporary file on the local machine. A more elegant approach would be to use UNIX's piping feature to eliminate the need of temporary file storage of data. Although this scheme was attempted in the present implementation, it was found that for reasons indeterminate - buggy software or protocol discrepancies - data was being lost in the pipe. If the reason for this occurence is determined and the problem is solved, then piping can be used successfully.
2. Alternative network protocol may be a more efficient than the **telnet** and **ftp** communication facilities used in the present implementation of the LQP. For example, IBM's LU6.2 APPC (Advanced Program to Program Communication) SNA protocol, being supported by an increasing number of non-SNA architectures and UNIX-based machines, may be used for efficient and flexible communication on a real-time basis.

3. The LQP would do well to automatically handle the idiosyncracies of the SQL/DS SELECT statement format. Specifically, for the ALQ in a :GET-DATA message, a table name <table> created by <creator> should be automatically converted to the format <creator>.<table> at the LQP level. This would be very useful to the upper levels if the CIS/TK system.
4. Lastly, the LQP designed in this paper was only capable of querying a database using the SQL/DS SELECT statement. The functionality of the LQP could be greatly enhanced if the LQP is redesigned to perform updating and alteration functions as well. It would be ideal if the user of the CIS/TK could use every feature of a remote DBMS. This problem must be tackled partially at the LQP level.

6.2 Acessing Additional Databases on the SQL/DS RDBMS

Presently, the LQP-object "sqlds-4381" contains all the attributes and methods required to retrieve data from the database "slinfo". This is a rather limited use of the connectivity that has been established between "mit2e" and "sloan". It would seem reasonable to expect the new LQP to access all the databases on the SQL/DS RDBMS. This can, in fact, be achieved very easily by the procedure outlined below:

Notice that presently, "sloandb" is an instance of the "sqlds-4381" LQP-object. It has not been given any unique attributes, since it intends to access the "slinfo" database, the attributes and methods for which are all contained in "sqlds-4381".

Now consider that we want the LQP to now access two databases - "slinfo", and another database, "pinfo". It is found that the methods of accessing the two databases are identical, except for the fact that the databases are referred to (in the LQP commands) as "slinfo" and "pinfo" respectively. Thus, we can create two new objects, "slinfodb" and "pinfodb". These objects are defined to be instances of "sqlds-4381". This means that whenever required information is not found in these instance objects, the superior "sqlds-4381" will be referenced for the information. Thus, since the only thing unique about the "slinfodb" and "pinfodb" objects are the names of their respective databases. Below are the required instance object definitions:

```
(create-instance 'sqlds-4381 'slinfodb)
(create-instance 'sqlds-4381 'pinfodb)
```

The *create-instance* KOREL function is used to create the "slinfodb" and "pinfodb" instances of "sqlds-4381".

These instances are now defined as objects using KOREL's *make-object* function, and are provided with the unique attributes that they require:

```
(make-object 'slinfodb
             ('database-directory "slinfo")
             ('database "slinfo"))

(make-object 'pinfodb
             ('database-directory "pinfo")
             ('database "pinfo"))
```

Once these instance objects are defined, the LQP messages can be sent to them in addition to the already defined "sqlds-4381" object and the "sloandb" instance.

The following should be noted:

1. If the method to access the "pinfo" database was different from that used to access "slinfo", a new method could be included in the "pinfodb" object to use the new database access procedure.
2. The *'database-directory* and *'database* attributes in "slinfo" and "pinfo" are not *default* values as they are in "sqlds-4381".
3. When a message is sent to an instance object, only if a required attribute is not found in its body will the superior, "sqlds-4381" be referenced.
4. The default database accessed by "sqlds-4381" is "slinfo".
5. The "sloandb" instance does not contain any attributes or methods, and thus is simply another name by which "sqlds-4381" can be referenced. If, hypothetically, attributes and methods were to be defined for "sloandb", it would have to be defined as an object.

Appendix A

Common LISP Files

A.1 sqlds4381.lsp

```
; ****************************
; **  FILE: sqlds4381.lsp  **
; ****************************
; By Gautam A. Gidwani (July, 1988)
; As part of MIT Undergrad. Thesis

; SQL/DS SPECIFIC QUERY PROCESSOR FOR MACHINE 'SLOAN'(AT MIT SLOAN SCHOOL)
;
; The SQL/DS-4381 object can respond to the following messages:
;       =>      :self-info
;       =>      :get-tables
;       =>      :get-columns <table-name>
;       =>      :get-data <cis/tk-single-query>

; The 'get-SQLDS4381-tables', 'get-SQLDS4381-columns', and 'get-SQLDS4381-data'
;       procedures all provide the'FETCH-DATA' function with an SQL statement
;       to be sent to the remote SQL/DS RDBMS.
; FETCH-DATA connects to the remote machine, "sloan", queries the SQL/DS RDBMS,
;       retrieves the required query result and parses it to the final Result
;       List format required.


(defun display-SQLDS4381-self-info  ()
  (lqp-print 'quiet "~%
                  S Q L / D S    O N    S L O A N
                 ------------------------------------~%
The SQL/DS Relational Database Management System is a computer program
that manages pieces of information (data) stored in a computer. The data
is contained in tables made up of vertical columns and horizontal rows.
The SQL/DS database uses the SQL query language which is common among
many of today's RDBMS systems.~%~%"))


(defun get-SQLDS4381-tables ()
  (let* ((SQL (format nil
                      "SELECT TNAME, CREATOR, DBSPACENAME, NCOLS ~
                      FROM SYSTEM.SYSCATALOG ~
                      ORDER BY TNAME, CREATOR~%"))
         (machine (get-self 'machine-name)))
    (FETCH-DATA SQL machine)))


(defun get-SQLDS4381-columns (table)
  (let* ((SQL  (format nil
                  "SELECT DISTINCT CNAME, COLTYPE, NULLS ~
                   FROM SYSTEM.SYSCOLUMNS ~
                   WHERE TNAME = '~A' ~
                   ORDER BY CNAME~%"
```

```
                    (parse-SQL-tname table)))
          (machine (get-self 'machine-name)))
    (FETCH-DATA SQL machine)))

(defun get-SQLDS4381-data (ALQ)
  (let* ((SQL (form-sql ALQ))
         (machine (get-self 'machine-name)))
    (FETCH-DATA SQL machine)))


(defun FETCH-DATA (SQL machine)
  (lqp-print 'normal "~%DBMS Query to be sent to machine ~A...." machine)
  (lqp-print 'normal "~%~A~%" SQL)
  (lqp-print 'terse "Fetching SQL/DS Data Requested....~%~%")
  (connect (get-current-object)
           SQL)
  (lqp-print 'verbose "~%Transporting Result File to local machine....~%")
  (let* ((lqpdir  (get-self 'comm-server-directory))
         (comdir  (get-self 'lqp-common-directory))
         (ftpfile (get-self 'ftp-script))
         (account (get-self 'account))
         (passwrd (get-self 'password))
         (machine (get-self 'machine-name))
         (tmpfile (get-self 'temporary-file)))
    (system (format nil "~A/~A ~A ~A ~A ~A/~A | ftp -n"
                    lqpdir ftpfile account passwrd machine
                    lqpdir tmpfile))
    (lqp-print 'terse "Result File Retrieved.~%~%")
    (lqp-print 'terse "Converting Result File to 'standard' Table File...~%")
    (system (format nil "~A/FILT4381 ~A/~A" lqpdir lqpdir tmpfile))
    (lqp-print 'terse "Done.~%~%")
    (lqp-print 'normal "Table File converted to Result List....~%~%")
    (lqp-print-file 'normal (format nil "~A/~A" lqpdir tmpfile))
    (read-standard-table (format nil "~A/~A" lqpdir tmpfile) comdir)))

;-------------------------------------------------------------------------;
;            DEFINITION OF LQP OBJECT CLASS SQLDS-4381                    ;
;-------------------------------------------------------------------------;

(make-object 'sqlds-4381
             ('machine-name "sloan")
             ('type-of-DBMS "sqlds")
             ('local-DBMS?  nil)
             ('database-directory "slinfo" 'default)
             ('database "slinfo" 'default)
             ('comm-server-directory "/usr/cistk/demo/v2/lqp/ibm4381" 'default)
             ('lqp-common-directory "/usr/cistk/demo/v2/lqp")
             ('communications-script "4381FILE")
             ('efficient-comm-script "4381FILE")
             ('ftp-script "4381FTP")
             ('temporary-file "connect1.tmp")  ;;*DATA PROCESSED HERE*;;
             ('account  "ggidwani" 'default)
             ('password "mohina" 'default)
             ('methods t 'multiple-value-f)
             ('methods '(:self-info display-SQLDS4381-self-info))
             ('methods '(:get-tables get-SQLDS4381-tables))
             ('methods '(:get-columns get-SQLDS4381-columns))
             ('methods '(:get-data get-SQLDS4381-data)))

(create-instance 'sqlds-4381 'sloandb)
```

```
;-------------------------------------------------------------------------;
;       END OF DEFINITION FOR LQP SQLDS-4381  (SQLDS4381.LSP)             ;
;-------------------------------------------------------------------------;
```

A.2 sql.lsp

```
; *************************
; **   FILE:  SQL.LSP   **
; *************************
; By Alec R. Champlin (April, 1988)
; As part of MIT Undergrad. Thesis
; Used UNMODIFIED by Gautam A. Gidwani for implementation of "sqlds-4381"
;   Local Query Processor (June, 1988)

;-------------------------------------------------------------------------;
;               PRINT CONTROL ROUTINES FOR ALL OF LQP CODE                ;
;-------------------------------------------------------------------------;
;
; FOUR PRINT MODES AVAILABLE: QUIET TERSE NORMAL VERBOSE

(defvar *current-lqp-print-mode* 'QUIET)    ; DEFAULT MODE = QUIET

(defun lqp-mode (mode)
  (cond ((and (not (equal mode 'QUIET)) (not (equal mode 'TERSE))
              (not (equal mode 'NORMAL)) (not (equal mode 'VERBOSE)))
         (write-string "VALID MODES: 'quiet 'terse 'normal 'verbose .")
         nil)
        (t (setq *current-lqp-print-mode* mode))))

(defun lqp-print-controlled-apply (mode func &optional args)
  (cond ((and (not (equal mode 'VERBOSE)) (not (equal mode 'NORMAL))
              (not (equal mode 'TERSE)) (not (equal mode 'QUIET)))
         nil)  ; ---May want to make this an error message.---
        ((equal *current-lqp-print-mode* 'VERBOSE)
         (apply func args))
        ((equal *current-lqp-print-mode* 'NORMAL)
         (if (not (equal mode 'VERBOSE))
             (apply func args)))
        ((equal *current-lqp-print-mode* 'TERSE)
         (if (or (equal mode 'TERSE) (equal mode 'QUIET))
             (apply func args)))
        (t nil)))

(defun lqp-print (mode str &rest args)
  (lqp-print-controlled-apply mode
        #'format (cons *standard-output* (cons str args))))

(defun lqp-print-file (mode file)
  (lqp-print-controlled-apply mode
        #'system (list (format nil "cat ~A" file))))

;-------------------------------------------------------------------------;
;   CIS/TK STANDARD LOCAL QUERY TO SQL QUERY STRING TRANSLATION ROUTINES   ;
;-------------------------------------------------------------------------;

(defun form-SQL (query)
  (let ((table (parse-SQL-tname (car query)))
        (conds (if (not (null (caddr query)))
                   (parse-SQL-conds (caddr query))))
        colmn_SQL colmn_LST)
    (multiple-value-setq (colmn_SQL colmn_LST) (parse-SQL-colmn (cadr query)))
    (cond ((or (equal table 'ERROR)
               (equal conds 'ERROR)
```

```
            (equal colmn_SQL 'ERROR))
        (lqp-print 'verbose
                   "FORM-SQL -- Error detected. No query returned.~%")
        (values nil nil))
       ((null conds)
        (values (format nil "SELECT ~A FROM ~A" colmn_SQL table)
                colmn_LST))
       (t (values (format nil "SELECT ~A FROM ~A WHERE ~A"
                                  colmn_SQL table conds)
                  colmn_LST)))))

; NOTE: SQL allows multiple tables, but the protocol doesn't.
(defun parse-SQL-tname (table)
  (cond ((null table)
         (lqp-print 'verbose
                    "PARSE-SQL-TNAME -- No table to parse! ABORTING.~%")
         'ERROR)
        ((atom table)
         (if (stringp table)
             table
             (progn (lqp-print 'verbose
                               "PARSE-SQL-TNAME -- Converting symbol ~A into ~
                                a string.~%" table)
                    (format nil "~A" table))))
        ((listp table)
         (lqp-print 'verbose
                    "PARSE-SQL-TNAME -- Recieved list ~A as argument.~%" table)
         (if (equal 1 (length table))
             (progn (lqp-print 'verbose
                               "PARSE-SQL-TNAME -- Using sole element in list ~
                                as table name.~%")
                    (parse-SQL-tname (car table)))
             (progn (lqp-print 'verbose
                               "PARSE-SQL-TNAME -- Multiple tables not ~
                                accepted. ABORTING.~%")
                    'ERROR)))
        (t (lqp-print 'verbose
                      "PARSE-SQL-TNAME -- Couldn't interpret table ~A. ~
                       ABORTING.~%" table)
           'ERROR)))

; Note: This procedure returns multiple values.
;       1st =>  The SQL relevant column string; e.g., "NAME, ADDRESS, ZIP"
;       2nd =>  A parsed list of columns; e.g., ("NAME" "ADDRESSS" "ZIP")
(defun parse-SQL-colmn (columns)
  (cond ((null columns)
         (lqp-print 'verbose
                    "PARSE-SQL-COLMN -- No columns to parse! ABORTING.~%")
         'ERROR)
        ((or (equal columns 'all) (equal columns '*) (equal columns "all")
             (equal columns "ALL") (equal columns "*"))
         (lqp-print 'verbose
                    "PARSE-SQL-COLMN -- Wildcards not currently supported. ~
                     ABORTING.~%")
         'ERROR)
        ((atom columns)
         (if (stringp columns)
             (values columns (list columns))
             (progn (lqp-print 'verbose
                               "PARSE-SQL-COLMN -- Converting symbol ~A ~
```

```
                                into a string.~%" columns)
                        (values (format nil "~A" columns)
                                (list (format nil "~A" columns))))))
          ((listp columns)
           (if (equal 1 (length columns))
               (parse-SQL-colmn (car columns))
               (let (carx cary cdrx cdry)
                 (multiple-value-setq (carx cary)
                          (parse-SQL-colmn (car columns)))
                 (multiple-value-setq (cdrx cdry)
                          (parse-SQL-colmn (cdr columns)))
                 (values (format nil "~A, ~A" carx cdrx)
                         (append cary cdry)))))
          (t (lqp-print 'verbose
                        "PARSE-SQL-COLMN -- Couldn't interpret columns ~A. ~
                         ABORTING.~%" columns)
             'ERROR)))

(defun parse-SQL-conds (conds)
  (cond ((null conds) (lqp-print 'verbose
                                 "PARSE-SQL-CONDS -- No search condition.~%"))
          ((atom conds) (lqp-print 'verbose
                                   "PARSE-SQL-CONDS -- Search condition ~A in ~
                                    improper form. ABORTING.~%" conds)
           'ERROR)
          ((listp conds)
           (cond ((> (length conds) 3)
                  (lqp-print 'verbose
                             "PARSE-SQL-CONDS -- Search condition ~A in ~
                              improper form. ABORTING.~%" conds)
                  'ERROR)
                 ((or (equal (car conds) 'and) (equal (car conds) "AND")
                      (equal (car conds) "and") (equal (car conds) "And"))
                  (format nil "(~A) AND (~A)"
                          (parse-SQL-conds (second conds))
                          (parse-SQL-conds (third conds))))
                 ((or (equal (car conds) 'or) (equal (car conds) "OR")
                      (equal (car conds) "or") (equal (car conds) "Or"))
                  (format nil "(~A) OR (~A)"
                          (parse-SQL-conds (second conds))
                          (parse-SQL-conds (third conds))))
                 ((or (equal (car conds) 'null) (equal (car conds) "NULL")
                      (equal (car conds) "null") (equal (car conds) "Null"))
                  (format nil "~A IS NULL" (parse-SQL-colmn (second conds))))
                 ((or (equal (car conds) 'not) (equal (car conds) "NOT")
                      (equal (car conds) "not") (equal (car conds) "Not"))
                  (format nil "NOT (~A)" (parse-SQL-conds (second conds))))
                 ((relation-p (car conds))
                  (format nil "~A ~A ~A" (parse-SQL-colmn (second conds))
                                         (parse-SQL-relation (first conds))
                                         (parse-SQL-col-or-lit (third conds))))
                 (t (lqp-print 'verbose
                               "PARSE-SQL-CONDS -- Incorrect form: ~A ~
                                ABORTING.~%" conds)
                    'ERROR)))
          (t (lqp-print 'verbose
                        "PARSE-SQL-CONDS -- Couldn't interpret condition ~A ~
                         ABORTING.~%" conds)
             'ERROR)))
```

```
(defun parse-SQL-relation (relation)
  (cond ((eq '<> relation)
         (format nil "!="))
        ; all the rest are the same
        (t (format nil "~A" relation))))

(defun parse-SQL-col-or-lit (col-or-lit)
  (if (stringp col-or-lit)
      (format nil "'~A'" col-or-lit)
      (format nil "~A" col-or-lit)))

(defun relation-p (relation)
  (or
    (eq relation '=)
    (eq relation '>)
    (eq relation '<)
    (eq relation '>=)
    (eq relation '<=)
    (eq relation '<>)))

#|
;THIS FUNCTION IS NOT CURRENTLY USED. MAY BE USEFUL FOR EFFICIENCY.
(defun parse-SQL-group-relation (relation)
  (cond
    ((eq relation 'average)
     (format nil "AVG"))
    ((eq relation 'sum)
     (format nil "SUM"))
    ((eq relation 'minimum)
     (format nil "MIN"))
    ((eq relation 'maximum)
     (format nil "MAX"))
    ((eq relation 'cardinality)
     (format nil "COUNT"))
    ((eq relation 'variance)
     (format nil "VARIANCE"))
    ((eq relation 'standard_deviation)
     (format nil "STDDEV"))
    ((eq relation 'no_nulls)
     (format nil "NVL"))
    (t (lqp-print 'verbose
                  "PARSE-SQL-GROUP-RELATION -- ~A isn't a valid relation!~%"
                  relation))))
|#

;-----------------------------------------------------------------------;
;              END OF SQL TRANSLATION ROUTINES (SQL.LSP)                ;
;-----------------------------------------------------------------------;
```

A.3 connect.lsp

```
; *************************
; **  FILE: CONNECT.LSP  **
; *************************
; By Alec R. Champlin (April, 1988)
; As part of Undergrad. Thesis

;=============================================================;
;     MODIFIED:  By Gautam A. Gidwani (September, 1988)       ;
;                For use in implementation of the 'sqlds-4381' LQP ;
; MODIFICATION:  Added the SQL/DS 'type-of-DBMS identifier "sqlds" ;
;                to the already present "informix" and "oracle"    ;
;                conditional identifiers                           ;
;=============================================================;

;-------------------------------------------------------------;
;     ROUTINES FOR CONTROLLING THE DBMS COMMUNICATION SCRIPTS  ;
;-------------------------------------------------------------;

(defun connect (DBMS-obj SQL &optional (use-eff? nil))
  (let* ((local    (get-object DBMS-obj 'local-DBMS?))
         (comdir   (get-object DBMS-obj 'comm-server-directory))
         (dbdir    (get-object DBMS-obj 'database-directory))
         (db       (get-object DBMS-obj 'database))
         (dbtype   (get-object DBMS-obj 'type-of-DBMS))
         (machine (get-object DBMS-obj 'machine-name))
         (account (if (not local)
                      (get-object DBMS-obj 'account)
                      "IRRELEVANT"))
         (passwrd (if (not local)
                      (get-object DBMS-obj 'password)
                      "IRRELEVANT"))
         (script   (if use-eff?
                      (get-object DBMS-obj 'efficient-comm-script)
                      (get-object DBMS-obj 'communications-script)))
         (tmpfile1 (si:string-concatenate comdir "/connect1.tmp"))
         (tmpfile2 (si:string-concatenate comdir "/connect2.tmp"))
         invoker)
    (setq script (format nil "~A/~A" comdir script))
    (cond ((or (equal dbtype 'INFORMIX) (equal dbtype "Informix")
               (equal dbtype "INFORMIX") (equal dbtype "informix"))
           (setq invoker (format nil "isql ~A -" db)))
          ((or (equal dbtype 'ORACLE) (equal dbtype "Oracle")
               (equal dbtype "ORACLE") (equal dbtype "oracle")
               (equal dbtype "SQL/RT") (equal dbtype "sql/rt"))
           (setq invoker "sqlcmd"))
          ((or (equal dbtype 'SQLDS) (equal dbtype "SQLDS")
               (equal dbtype 'sqlds) (equal dbtype "sqlds"))
           (setq invoker "dbinit"))
          (t (lqp-print 'terse
                        "CONNECT -- Database type ~A unrecognized. ~
                         Update CONNECT.LSP with new dbtype.~%" dbtype)
             return))
    (lqp-print 'terse "Connecting to ~A on machine ~A...~%" db machine)
    (if (not local)
        (system (unix-format "~A ~A ~A ~A ~A ~A | telnet ~A 1> ~A 2> ~A"
                             script account passwrd dbdir invoker SQL
                             machine tmpfile1 tmpfile2))
```

```
        (system (unix-format "~A ~A ~A ~A ~A 1> ~A 2> ~A"
                             script account passwrd dbdir invoker SQL
                             tmpfile1 tmpfile2)))
    (lqp-print 'terse "Done.~%")
    (values tmpfile1 tmpfile2)))

(defun unix-format (str &rest args)
  (setq args (mapcar #'(lambda (x) (format nil "~C~A~C" #\" x #\"))
                     args))
  (apply #'format (cons nil (cons str args))))

;-------------------------------------------------------------------------;
;     END OF COMMUNICATIONS SCRIPT CONTROL ROUTINES (CONNECT.LSP)         ;
;-------------------------------------------------------------------------;
```

A.4 read.lsp

```
; ************************
; **  FILE: read.lsp  **
; ************************
; By Alec R. Champlin (April, 1988)
; As part of MIT Undergrad. Thesis

: MODIFICATIONS:
;     Modified by Gautam A. Gidwani for use in the new 'sqlds-4381'
;     LQP.
;     The second 'if' statement now checks (using 'probe-file') for
;     a file both in the current working directory or in a directory
;     specified by 'comdir'. Previously, the function assumed that
;     the file was in the current directory.

;-------------------------------------------------------------;
;     ROUTINE FOR READING "STANDARDIZED" DBMS OUTPUT FILES     ;
;-------------------------------------------------------------;

(defun read-standard-table (file comdir &aux tmp info)
  (if (not (probe-file (format nil "~A/preREAD" comdir)))
      (lqp-print 'terse "READ-STANDARD-TABLE -- File '~A' needed and ~
                         not found!" (format nil "~A/preREAD")))

;;; This 'if' statement modified - gg ;;;

  (if (or (probe-file file) (probe-file (format nil
                                                "~A/~A"
                                                comdir file)))
      (progn (lqp-print 'terse "~%Reading DBMS output file...")
             (if (probe-file file)
                 (system (format nil "~A/preREAD ~A" comdir file)))
             (if (probe-file (format nil
                                     "~A/~A"
                                     comdir file))
                 (system (format nil "~A/preREAD ~A/~A" comdir comdir file)))
             (with-open-file (data file :direction :input)
                (loop (let ((line (read-line data nil 'EOF)))
                        (cond ((equal line 'EOF)
                               (lqp-print 'terse "Done.~%~%")
                               (return (remove-if #'null (reverse info))))
                              ((equal line "")
                               (setq info (cons (reverse tmp) info))
                               (setq tmp '()))
                              (t (setq tmp (cons line tmp))))))))))
      (lqp-print 'terse "READ-STANDARD-TABLE -- File '~A' not found!" file)))

#|

     THIS IS THE OLD SET OF ROUTINES FOR READING "STANDARD" TABLES

(defun read-standard-table (file &aux info)
  (if (probe-file file)
      (with-open-file (data file :direction :input)
        (inf2c-print "READ-STANDARD-TABLE -- Now reading table file...")
        (loop (let* ((row (read-line data nil "EOF"))
                     (entries (get-entries row)))
                (if (equal row "EOF")
```

```
                    (progn
                      (inf2c-print "Done.~%~%")
                      (return (reverse (remove-if #'null info))))
                    (setq info (cons entries info))))))
      (inf2c-print "READ-STANDARD-TABLE -- Error: No table to read!~%")))

(defun get-entries (str)
  (form-entries (coerce str 'list) NIL ""))

(defun form-entries (lst entry-lst temp-str)
  (cond ((null lst)
         (remove-if #'(lambda (x) (equal x ""))
                    (reverse (cons temp-str entry-lst))))
        ((equal (car lst) #\|)
         (form-entries (cdr lst) (cons temp-str entry-lst) ""))
        (t
         (form-entries (cdr lst)
                       entry-lst
                       (si:string-concatenate temp-str (car lst))))))

|#

;-------------------------------------------------------------------;
;      END OF "STANDARDIZED" TABLE READING ROUTINES (READ.LSP)      ;
;-------------------------------------------------------------------;
```

A.5 demo-gg.lsp

```
; ************************
; **  FILE: DEMO.LSP  **
; ************************
; By Gautam A. Gidwani (Feb., 1989)
; As part of MIT Undergrad. Thesis

;-------------------------------------------------------------------------;
;             LOCAL QUERY PROCESSOR DEMO LOADING COMMANDS                 ;
;-------------------------------------------------------------------------;

(let* ((sqlds4381-dir "/usr/cistk/biggie/ibm4381"))

  (if (not (probe-file (format nil "~A/4381FILE" sqlds4381-dir)))
      (format t "4381FILE missing! ~% --> LQP object SQLDS-4381 will not ~
             process messages 'GET-TABLES', ~%  'GET-COLUMNS' and ~
             'GET-DATA' correctly.~%"))
   (if (not (probe-file (format nil "~A/4381FTP" sqlds4381-dir)))
      (format t "4381FTP missing! ~% --> LQP object SQLDS-4381 will not ~
             process messages 'GET-TABLES', ~%  'GET-COLUMNS' and ~
             'GET-DATA' correctly.~%"))
   (if (not (probe-file (format nil "~A/4381SEL" sqlds4381-dir)))
      (format t "4381SEL missing! ~% --> LQP object SQLDS-4381 will not ~
             process messages 'GET-TABLES', ~% 'GET-COLUMNS' and ~
             'GET-DATA' correctly.~%"))

   (if (not (probe-file (format nil "~A/preREAD" sqlds4381-dir)))
      (format t "preREAD missing! ~%  --> Most LQP messages will not work.~%"))
   (if (not (probe-file (format nil "~A/filt4381" sqlds4381-dir)))
      (format t "FILT4381 missing! ~% --> SQLDS-4381 messages will not work.~%"))

   (load (format nil "~A/frames.lsp" sqlds4381-dir))
   (load (format nil "~A/korel.lsp" sqlds4381-dir))

   (load (format nil "~A/sql.lsp" sqlds4381-dir))
   (load (format nil "~A/connect.lsp" sqlds4381-dir))
   (load (format nil "~A/read.lsp" sqlds4381-dir))

   (load (format nil "~A/sqlds4381.lsp" sqlds4381-dir)))

(defun get-users-lqp-print-preference ()
  (format t "~%
What level of messages do you want printed?
      --> Quiet, Terse, Normal, or Verbose? ")
  (let ((input (read *terminal-io*)))
    (cond ((and (not (equal input 'QUIET)) (not (equal input 'TERSE))
                (not (equal input 'NORMAL)) (not (equal input 'VERBOSE)))
           (get-users-lqp-print-preference))
          (t (setq *current-lqp-print-mode* input)
             (format t "~%OK...If you change your mind, use 'LQP-MODE'. ~
                        As in (lqp-mode 'quiet)~%"))))))

(get-users-lqp-print-preference)

;-------------------------------------------------------------------------;
;                      END OF LOAD INSTRUCTIONS                           ;
;-------------------------------------------------------------------------;
```

A.6 korel.lsp

```
;FILE:  KOREL.LSP
;
; Copyright (C) 1987 by Sam Levine
;
;       o  Modified by Alec R. Champlin (May, 1988)
;          "Added the "send self" functionality to the MESSAGE-
;           PASSING system, so that methods could make references
;           to the object instances that requested/invoked them."
;
;*************************************************************************
;
;  PACKAGE-NAME:  KOREL.LSP  by Sam Levine (SPL)
;  USES-PACKAGES: FRAMES.LSP
;  PACKAGE-DESCRIPTION: Object-oriented Knowledge-representation Language
;*************************************************************************

;*************************************************************************
;OBJECTS, CLASSES, and INSTANCES
;*************************************************************************

;  An object is either a class or an instance.
;  There is a heirarchy of classes, with the CLASS class being at the
;  root of the tree.  Every class is, directly or indirectly, a member
;  of the CLASS class.  Each object (except CLASS) has some non-zero number of
;  superiors.  An object inherits properties from its superiors.
;  A class can spawn subordinate classes as well as instances.
;  A subordinate class inherits all the properties of the superior, but
;  usually adds additional information.
;  An instance represents a specific physical object.  It inherits
;  information from the hierarchy of classes above it.
;
;  Here is what the system looks like now:
;  CLASS:
;    SUPERIORS:  Those classes of which the class is a specialization.
;      For example, mammal is a superior of human.  list.
;    SUBORDINATES:  Those classes of which the class is a parent.  For
;      example, mammal is subordinate to living-things.  list.
;    METHODS:  Contains pairs.  The first of each pair is
;      a message to match against.  The second of each pair is the
;      appropriate function to execute.  list.
;    INSTANCES:  This contains a list of instances of this class. list.
;    SLOT1:
;    ...
;    SLOTn:
;
;  INSTANCE:
;    INSTANCE-OF:  The class which created this instance.
;    SLOT1:
;    ...
;    SLOTn:


;*************************************************************************
;FACETS
;*************************************************************************
;
;SLOT-FACETS
```

```
;  VALUE: {values}
;    the value for this slot.  single or list
;  DEFAULT: {values}
;    the default value for this slot.  single or list
;  IF-NEEDED: {procedure-names}
;    a list of procedures to try if there is no value or default value
;    for this slot.  list.
;  IF-ADDED:  {demons}
;    a list of demons which are executed whenever a value is placed into
;    the value facet of this slot.  list.
;  IF-REMOVED:  {demons}
;    a list of demons which are executed whenever a value is removed from
;    the value facet of this slot.  list.
;  VALUE-TYPE: {integer, string, fraction, real}
;    Whenever a value is placed into choices, value, or default facets,
;    it is first checked to ensure that it is of the appropriate value-type.
;    nil => no constraint on value-type.  Single.
;  CONSTRAIN-OTHERS: {rule-names}
;    whenever a value is placed into the value or default facets, this
;    rule set is executed sequentially.  list.
;  SELF-CONSTRAINTS: {constraints}
;    whenever a value is placed into the value, or default facets, this
;    set of constraints is checked to ensure that none are violated.
;    Whenever a new constraint is added to this list, the choices, value and
;    default facets are checked to ensure consistency.  Inconsistent defaults
;    and choices are eliminated, while the user is asked how to resolve
;    inconsistent values.
;    Constraints are lisp procedures which take three arguments:  a frame
;    and slot name (usually the current frame and slot) and a value.  list.
;  CHOICES: {values}
;    list of valid choices for this slot. list.
;  QUERY: {multi-query}
;    executes the contained multi-query to return a value for this slot
;    from a database.
;
;  These facets are flags
;  MULTIPLE-VALUE-F: {t or nil}
;    if T, signifies that the slot accepts multiple values and defaults.
;    default is nil. (ie: single valued)
;  NO-INHERIT-F: {t or nil}
;    if T, signifies that the the facets of the slot aren't inherited.
;    otherwise, they can be.  Default is nil. (ie: inheritence).
;
;  These facets are extensions to allow a database interface
;  RETRIEVAL-PATTERN: {database-retrieval-pattern} see
;    DBMS.LSP for a specification of this pattern.
;  DBMS: {the name of a physical DBMS} contains the name of the
;    database that the value for this slot may be found in.


;*****************************************************************************
;OBJECT FUNCTIONS:
;*****************************************************************************
;external:
;  PUT-OBJECT
;  GET-OBJECT
;  REMOVE-OBJECT
;internal:
;  GET-ALL-SUPERIORS
;  GET-ALL-SUBORDINATES
```

```
;  GET-ALL-INSTANCES
;
;*********************************************************************
;SLOT FUNCTIONS:
;*********************************************************************
;external:
;  A consistent interface to the slots is provided by the following
;  3 functions.  The facet defaults to 'VALUE.  The facet is checked to
;  ensure that it is one of the ones supported (see above).  If it is, then
;  the appropriate sequence of actions is performed.  Otherwise, an error
;  is signalled.
;
;  PUT-SLOT (frame slot VALUE &optional FACET)
;  GET-SLOT (frame slot &optional FACET)
;  REMOVE-SLOT (frame slot VALUE &optional FACET)
;
;internal:
;  CHECK-VALUE-TYPE (FRAME SLOT VALUE): returns t if value is of the type.
;                To be used before adding a value.
;  FIRE-CONSTRAIN-OTHERS (FRAME SLOT) :
;                To be used after adding a value
;  CHECK-SELF-CONSTRAINTS (FRAME SLOT VALUE) : returns t if the value is
;                acceptable according to the set of self constraints, nil
;                otherwise.
;                To be used before adding a value.
;  TEST-SELF-CONSTRAINTS (FRAME SLOT) :  tries all the self constraints on
;                the value, asking the user to resolve inconsistencies.
;                Deletes entries from the default and choices facets to
;                make them consistent with the self constraints.
;                To be used before adding a constraint.


;*********************************************************************
; OBJECT PROCEDURES
;*********************************************************************
;external

(defun put-object (frame slot value &optional facet)
  (if (null facet) (setq facet 'value))
  (case slot
    (superiors
      (freplace frame slot 'multiple-value-F t)
      (fput frame slot facet value)
      (fput value 'subordinates facet frame))
    (subordinates
      (freplace frame slot 'multiple-value-F t)
      (fput frame slot facet value)
      (fput value 'superiors facet frame))
    (instances
      (freplace frame slot 'multiple-value-F t)
      (create-instance frame value))
    (t (put-slot frame slot value facet))))

(defun get-object (frame slot &optional option)
  (case slot
    (superiors
      (if (eq option 'all)
          (cdr (reverse (get-all-superiors (list frame) nil)))
          (fget frame slot 'value)))
    (subordinates
```

```
      (if (eq option 'all)
          (cdr (reverse (get-all-subordinates (list frame) nil)))
          (fget frame slot 'value)))
    (instances
      (if (eq option 'all)
          (reverse (get-all-instances (list frame) nil))
          (fget frame slot 'value)))
    (t (get-slot frame slot option))))


(defun remove-object (frame slot value &optional facet)
  (if (null facet) (setq facet 'value))
  (fremove frame slot facet value))


(defmacro make-object (object-name &rest slots)
  (dolist (slot slots)
    (eval
      `(put-object ,object-name ,@slot)))
  t)

;internal

(defun get-all-superiors (queue classes)
  (cond ((null queue) classes)
        (t (get-all-superiors
             (append (get-object (car queue) 'superiors)
                     (cdr queue))
             (if (member (car queue) classes)
                 classes
                 (cons (car queue) classes))))))

(defun get-all-subordinates (queue classes)
  (cond ((null queue) classes)
        (t (get-all-subordinates
             (append (get-object (car queue) 'subordinates)
                     (cdr queue))
             (if (member (car queue) classes)
                 classes
                 (cons (car queue) classes))))))

(defun create-instance (class-frame-name new-instance-name)
  (if (get-object class-frame-name 'instance-of)
      (format t
              "~%~A is not an appropriate class to create an instance of."
              class-frame-name)
      (progn
        (freplace class-frame-name 'instances 'multiple-value-F t)
        (freplace new-instance-name 'superiors 'multiple-value-F t)
        (freplace new-instance-name 'instance-of 'multiple-value-F t)
        (fput class-frame-name 'instances 'value new-instance-name)
        (fput new-instance-name 'superiors 'value class-frame-name)
        (fput new-instance-name 'instance-of 'value class-frame-name)
        new-instance-name)))

(defun get-all-instances (queue instances)
  (cond ((null queue) instances)
        (t (get-all-instances
             (append (get-object (car queue) 'subordinates)
                     (cdr queue))
```

```
            (if (member (get-object (car queue) 'subordinates) instances)
                instances
                (append (get-object (car queue) 'subordinates)
                        instances))))))

;*********************************************************************
; SLOT PROCEDURES
;*********************************************************************
;external

(defun get-slot (frame slot &optional facet &aux ret-val)
  (if (null facet) (setq facet 'value*))
  (case facet
    (value* ;uses Z-inheritance (breadth-first search)
      (setq ret-val
            (if (get-slot frame slot 'multiple-value-F)
                (fget-z1 slot (cons frame (get-object frame 'superiors 'all)))
              (car (fget-z1
                     slot
                     (cons frame (get-object frame 'superiors 'all))))))
      (if (and (null ret-val)
               (get-slot frame slot 'query))
          (setq ret-val (send-multi-query (get-slot frame slot 'query))))
        ret-val)
    ((value default)
     (if (get-slot frame slot 'multiple-value-F)
         (fget-i frame slot facet)
         (car (fget-i frame slot facet))))
    ((value-type multiple-value-F no-inherit-F query)
      (car (fget-i frame slot facet)))
    ((if-needed if-added if-removed constrain-others self-constraints
          choices)
          (fget-i frame slot facet))
    (t (format t "~%~A is not a valid facet for a slot." facet))))

(defun remove-slot (frame slot value &optional facet)
  (if (null facet) (setq facet 'value))
  (case facet
    (value
      (fremove-p frame slot facet value))
    ((default if-needed if-added if-removed value-type constrain-others
        self-constraints choices internal-choices query)
         (fremove frame slot facet value))
    (t (format t "~%~A is not a valid facet for a slot." facet))))

(defun put-slot (frame slot value &optional facet)
  (if (null facet) (setq facet 'value))
  (case facet
    ((value default) ;single or multiple
      (cond
        ((not (check-self-constraints frame slot value))
         (format t "~%Self Constraints would be violated by adding ~A to ~A."
                 value facet))
        ((not (check-value-type frame slot value))
         (format t "~%~A is of wrong type." value))
        ((not (check-choices frame slot value))
         (format t "~%~A is not a valid choice for slot." value))
        ((get-slot frame slot 'multiple-value-F)
         (fput-p frame slot facet value)
         (fire-constrain-others frame slot)
```

```
          value)
         (t (freplace-p frame slot facet value)
            (fire-constrain-others frame slot)
            value)))
    ((if-needed if-added if-removed) ;demons
      (fput frame slot facet value))
    ((multiple-value-F no-inherit-F query)
      (freplace frame slot facet value))
    (value-type
      (if (member value '(integer string real))
          (freplace frame slot facet value)
          (format t "~%~A is not a valid type.~
                  ~%Valid types are: INTEGER STRING REAL~%"
                    value)))
    (constrain-others
      (fput frame slot 'constrain-others value)
      (fire-constrain-others frame slot)
      value)
    (self-constraints
      (fput frame slot 'self-constraints value)
      (test-self-constraints frame slot)) ; and checks to ensure consistent
    (choices        ;choices takes a list of choices
      (let ((old-values (follow-path (list slot facet)
                                     (fget-frame frame))))
        (delete old-values old-values))
      (if (atom value) (setq value (list value)))
      (dolist
        (choice value t)
        (cond ((not (check-self-constraints frame slot choice))
               (format t "~%~A doesn't satisfy self constraints."
                       choice))
              ((not (check-value-type frame slot choice))
               (format t "~%~A is of wrong type." choice))
              (t (fput frame slot facet choice))))
      value)
    (t (format t "~%~A is not a valid facet for a slot." facet))))


;internal

(defun check-value-type (frame slot value)
  (let ((type (get-slot frame slot 'value-type)))
    (cond ((null type))
          ((and (eq 'integer type) (integerp value)))
          ((and (eq 'string type) (stringp value)))
          ((and (eq 'real type)
                (or (typep value 'single-float)
                    (typep value 'double-float)))))))

;this procedure sequentially fires the rules in the constrain-others facet
;it will use the Winston expert system shell for this.
(defun fire-constrain-others (frame slot)
  (do ((rules-to-try
         (get-slot frame slot 'constrain-others)
         (cdr rules-to-try)))
      ((null rules-to-try))
    (cond ((use-rule (car rules-to-try))
           (setq rules-to-try (get-slot frame slot 'constrain-others))))))

; returns t if all constraints check out okay for the value
```

```
(defun check-self-constraints (frame slot values-list)
  (let ((constraints-list (get-slot frame slot 'self-constraints)))
    (if (not (listp values-list)) (setq values-list (list values-list)))
    (dolist
      (value values-list t)
      (dolist
        (constraint constraints-list t)
        (if (not (funcall constraint frame slot value))
            (return nil))))))  ; a test failed


; returns t if all constraints check out okay
(defun test-self-constraints (frame slot)
  (let ((value-list (get-slot frame slot))
        (default-list (get-slot frame slot 'default)))
    (if (not (null value-list))
        (check-self-constraints frame slot value-list))
    (if (not (null default-list))
        (check-self-constraints frame slot default-list))))

; returns t is the value is an acceptable choice for the slot, nil otherwise
(defun check-choices (frame slot value)
  (or (null (get-slot frame slot 'choices))
      (member value (get-slot frame slot 'choices) :test #'special-equal)))

(defun special-equal (x y)
  (or (equal x y)
      (equal x (and (listp y) (car y)))))

;*****************************************************************
; MESSAGE PASSING SYSTEM
;*****************************************************************
(defvar *korel-current-object-stack* '())      ;Alec Champlin -- Added. 5/88

(defun push-current-object (object)            ;Alec Champlin -- Added. 5/88
  (setq *korel-current-object-stack*
        (cons object *korel-current-object-stack*)))

(defun get-current-object ()                   ;Alec Champlin -- Added. 5/88
  (car *korel-current-object-stack*))

(defun pop-current-object (&aux result)        ;Alec Champlin -- Added. 5/88
  (setq result (get-current-object))
  (setq *korel-current-object-stack* (cdr *korel-current-object-stack*))
  result)

(defun get-self (slot &optional facet)         ;Alec Champlin -- Added. 5/88
  (if (null facet)
      (get-object (get-current-object) slot)
      (get-object (get-current-object) slot facet)))

(defun put-self (slot value &optional facet) ;Alec Champlin -- Added. 5/88
  (if (null facet)
      (put-object (get-current-object) slot value)
      (put-object (get-current-object) slot value facet)))

;Alec Champlin -- Modified to support "send self" concept. 5/88
(defun send-message (object message &rest arg-list &aux result)
  (if (equal object 'self)
      (if (not (null arg-list))
```

```
          (send-message (get-current-object) message arg-list)
          (send-message (get-current-object) message))
    (progn
      (push-current-object object)
      (dolist (pair (fget-i object 'methods 'value))
        (if (equal (first pair) message)
            (return (progn (setq result (apply (second pair) arg-list))
                           (pop-current-object)
                           result)))))))

(defun message-exists-p (object message)
  (dolist (pair (fget-i object 'methods))
    (if (equal (first pair) message)
        (return t))))

;*****************************************************************************
; USER INTERFACE DEFINITIONS
;*****************************************************************************

;(defvar *user-messages-window* t) ;here's where the error messages are sent.
;(setq *user-messages-window* t)   ;default is t.

;*****************************************************************************
; DISPLAY UTILITIES
;*****************************************************************************

(defun display-classes ()
  (format t "~%CLASS~%")
  (do* ((classes-to-show (list 'class))
        (current-class (first classes-to-show) (first classes-to-show))
        (current-subs (get-object current-class 'subordinates)
                      (get-object current-class 'subordinates)))
       ((null classes-to-show))
    (if current-subs
        (format t "SUBS of ~A:    ~A~%" current-class current-subs))
    (setq classes-to-show (append (cdr classes-to-show) current-subs))))

(defun remove-classes (&optional class-to-start-at)
  (if (null class-to-start-at)
      (setq class-to-start-at 'CLASS))
  (do* ((classes-to-kill (list class-to-start-at))
        (current-class (first classes-to-kill) (first classes-to-kill))
        (subordinates (append (get-object current-class 'subordinates)
                              (get-object current-class 'instances))
                      (append (get-object current-class 'subordinates)
                              (get-object current-class 'instances))))
       ((null classes-to-kill))
    (reset-frame current-class)
    (setq classes-to-kill (append (cdr classes-to-kill) subordinates)))
  t)


(defun print-object (frame-name)
  (let* ((frame (fget-frame frame-name))
         (slots (cdr frame)))
    (format t "~%~%~A:~%" frame-name)
    (dolist (slot slots)
      (cond ((or (eq (first slot) 'superiors)
                 (eq (first slot) 'subordinates)
                 (eq (first slot) 'instances)
```

```
                (eq (first slot) 'queries)))
            (t (format t "    ~A:~%" (car slot))
               (dolist (facet (cdr slot))
                 (format t "        ~A~%" facet))))
            (terpri))))

(defun select-query (object &aux queries chosen-query)
  (setq queries (get-object object 'queries))
  (if queries
      (progn
        (setq
          chosen-query
          (choose-list-with-prompt
            "Choose a query to execute"
            (mapcar #'(lambda(x)
                        (first x))
                    queries)
            1))
        (dolist (query queries)
          (if (eq (first query)
                  (first chosen-query))
              (progn
                (print-multi-query (second (second query)))
                (return (send-multi-query (second (second query))))))))))
    (format t "~%~A object has no associated class queries.~%"
            object)))
```

Appendix B
'C' Program Files

B.1 filt4381.c

This program has been commented to help in understanding the conversion of the Result File to the 'standard' Table File. Viewing the Result File and the Table File would also be helpful.

The program code is in smaller, bold text while the comments have been *italicized.*

```
/*****************************************************************************/
/**                                                                         **/
/**   FILE:  FILT4381.c            By Gautam A. Gidwani (Feb., 1989)       **/
/**   ================           As part of MIT Undergrad. Thesis           **/
/**                                                                         **/
/**   THIS ROUTINE CONVERTS THE DATA FILE FROM THE SQL/DS DATABASE ON       **/
/**      THE IBM-4381 MAINFRAME TO THE 'STANDARD' TABLE FILE TO BE          **/
/**             READ BY THE RESULT READER PROGRAM 'READ.LSP'                **/
/**                                                                         **/
/*****************************************************************************/

#include <stdio.h>

#define MAX_COLS 200        /* Allows a maximum of 200 data columns */
#define DELIM '|'

main (argc, argv)
     int argc;
     char *argv[];
{
  int             c, i1 = 0, i2 = 0, i3 = 0;
  int             loop, col_num, col_size, size_cnt;
  int             col_data[MAX_COLS];
  char            tmp[45];
  static char     end[] = {"************** End-of-Data **************\n"};
  FILE            *in_file, *tmp_file, *out_file;


  if (argc != 2)          /* One argument, the file to be filtered, is reqd. */
    {                     /* This file is the Result List */
      printf ("FILT4381 -- Expecting one argument, <FILENAME>.\n");
      exit (1);
    }
  else if ((in_file = fopen (argv[1], "r+")) == NULL)
    {                     /* This input file is called in_file */
      printf ("FILT4381 -- Couldn't open file \"%s\".\n", argv[1]);
      exit (2);
    }
```

```
  else if ((tmp_file = tmpfile()) == NULL)
    {                       /* A temporary file, tmp_file, is created */
      printf ("FILT4381 -- Couldn't open temporary file.\n");
      exit (3);
    }

  rewind (in_file);      /* Rewinds to the beginning of in_file */

 /* Loop = 0 skips the SQL SELECT statement */
/*  in the Result List                     */

  while (loop == 0)
    {
      if ((c = getc (in_file)) != EOF)/* Get a charecter from in_file   */
        {                                 /* Check if char. is alphanumeric, */
          if (c > 0x20 && c < 0x7F)       /*  space, tab or newline          */
            i1 = 1;
          if (i1 == 1 && c == '\n')       /* Go to loop = 1 when newline    */
            {                             /*  after the SQL statement found */
              loop = 1;
              i1 = 0;
            }
        }
      else
        {
          rewind (in_file);                   /* If no newline, then ERROR */
          printf ("FILT4381 -- No data found in file \"%s\".\n");
          exit (4);
        }
    }

/* Loop = 1 copies the column names to tmp_file */

  while (loop == 1)
    {
      if ((c = getc (in_file)) != EOF)            /* Gets a char., c */
        {
          if (i1 == 0 && c > 0x20 && c < 0x7F)/* Checks for alphanumeric,*/
            i1 = 1;                                 /*  space, tab or newline */
          if (i1 == 1)
            {
              if ((fputc (c, tmp_file)) == EOF)/* YES => put c in tmp_file
                {
                  printf ("FILT4381 --'FPUTC' Error.\n");
                  exit (5);   /* Checks that c put in tmp_file correctly */
                }
              if (c == '\n')  /* Go to loop = 2 when newline after   */
                {             /*  column names found */
                  loop = 2;
                  i1 = 0;
                }
            }
        }
      else
        {
          rewind (in_file);       /* Otherwise rewind in_file and ERROR */
          printf ("FILT4381 -- No data found in file \"%s\".\n");
          exit (6);
        }
    }
```

```
  col_num  = 0       /* col_num => column number  */
  col_size = 0;      /* col_size => size of a column */

/* Loop = 2 counts the dashes in the Result List. Determines */
/*  number of columns and their respective sizes    */

  while (loop == 2)
    {
      if ((c = getc (in_file)) != EOF)   /* Get the next char. c */
        {
          if (i1 == 0 && c > 0x20 && c < 0x7F)
            i1 = 1;
          if (i1 == 1)
            {
              if (c == '-')   /* Looks for a dash */
                ++col_size;   /* YES => increment size of present column */
              else if (c == ' ')   /* Looks for space between dashes  */
                {
                  col_data[col_num] = col_size;/* YES => store size of  */
                  col_size = 0;       /* present column in col_data array, */
                  ++col_num;          /* count the column, reset  */
                }                     /* column size counter */
              else if (c == '\n' || c == '\r')   /* Check for newline at  */
                {                                /* end of dashes */
                  loop = 3;                      /* YES => Go to loop = 3 */
                  col_data[col_num] = col_size;  /* Store size of last */
                  col_size = 0;                  /*  column, reset col_size */
                  ++col_num;                     /*  count the last column, */
                  col_data[col_num] = '\0';      /*  end col_data w/ '\0'  */
                  i1 = 0;
                }
              else          /* Otherwise ERROR */
                {
                  printf ("FILT4381 -- Unexpected input format in file..
                                                \"%s\".\n", argv[1]);
                  exit (7);
                }
            }
        }
      else
        {
          rewind (in_file);
          printf ("FILT4381 -- No data found in file \"%s\".\n", argv[1]);
          exit (8);
        }
    }

  i2 = strlen (end);  /* i2 = length of string array 'end' defined above */

/* Loop = 3 copies everything after the dashes and everything before the */
/*  end of file marker (identical to 'end' is found.                  */)

  for (loop = 3, i1 = 0; loop == 3 && i1 < i2; )
    {
      if ((c = getc (in_file)) == EOF)          /* Check for end of file */
        {
          for (i1 = 0; tmp[i1] != '\0'; ++i1) /* If not '\0', ++i1 */
            if ((fputc (tmp[i1], tmp_file)) == EOF)
              {
                printf ("FILT4381 -- 'FPUTC' Error.\n");
```

```
                exit (9);               /* End marker not found here */
              }
            printf ("FILT4381 -- End marker not found in file \"%s\".\n", argv[1]);
          }
        tmp[i1] = c;          /* Puts c in tmp[i1] */
        tmp[i1+1] = '\0';     /* Ends 'tmp' string array with required '\0' */
        if (c == end[i1])     /* Compares char. c with end[i1] */
          ++i1;               /* YES => increment i1 (thus compares next char */
        else                  /*  with the next element of 'end'          */
          {
            for (i1 = 0; tmp[i1] != '\0'; ++i1)       /* Otherwise, ERROR */
              if ((fputc (tmp[i1], tmp_file)) == EOF)
                  {
                    printf ("FILT4381 -- 'FPUTC' Error.\n");
                    exit (10);
                  }
            i1 = 0;
          }
      }


    if ((out_file = freopen (argv[1], "w", in_file)) == NULL)
      {           /* opens an output file, out_file */
        printf ("FILT4381 -- 'FREOPEN' Error.\n");
        exit (11);
      }

    rewind (tmp_file);   /* Goes to the beginning of tmp_file, which should */
                         /*  now contain column names and data only - no    */
                         /*  SQL statement, dashes and end marker           */
    loop = 4;                       /* Goes to loop = 4 */
    i1 = 0; i2 = 0;                 /* i1 = 1 for space, i2 = 1 for alphanumeric */
    i3 = col_num;                      /* Number of columns counted */
    col_num = 0; size_cnt = 0;      /* Resets col_num and col_size */
                                    /* Note that the first column is represented */
                                    /*  by col_data[0]  */

/* Loop = 4 put delimiters between the columns and creates the required  */
/*  'standard' Table File format                                        */

    while (loop == 4)
      {
        if ((c = getc (tmp_file)) == EOF)   /* Gets character from tmp_file */
          break;                            /*  and checks for EOF */
        else if (c == '\n')     /* If newline, put a delimiter in out_file */
          {
            if ((fputc (DELIM, out_file)) == EOF)
              {
                printf ("FILT4381 -- 'FPUTC' Error.\n");
                exit (12);
              }
            for (i2 = col_num; i2 < (i3-1); ++i2) /* Check if the newline */
              {           /* was found before the last column */
                if ((fputc (' ', out_file)) == EOF)
                  {                         /* YES => put space in out_file */
                    printf ("FILT4381 -- 'FPUTC' Error.\n");
                    exit (13);
                  }
                if ((fputc (DELIM, out_file)) == EOF)/*Also put a delimiter*/
                  {                                  /* in out_file*/
```

```
            printf ("FILT4381 -- 'FPUTC' Error.\n");
            exit (14);
          }
      }
    if ((fputc (c, out_file)) == EOF)     /* Put in the newline */
      {
        printf ("FILT4381 -- 'FPUTC' Error.\n");
        exit (15);
      }
    col_num = 0; size_cnt = 0;          /* Reset for next row of data */
    i1 = 0; i2 = 0;
  }
else if (size_cnt < col_data[col_num])
  {           /* Checks if not at end of column */
    if (c > 0x20 && c < 0x7F)   /* Checks for alphanumeric, space, */
      {                                 /* tab or newline */
        if (i1 == 1 && i2 == 1) /* Checks for space between words */
          if ((fputc (' ', out_file)) == EOF) /* Puts in the space */
            {
              printf ("FILT4381 -- 'FPUTC' Error.\n");
              exit (16);
            }
        if ((fputc (c, out_file)) == EOF)/* Puts in present char.*/
          {
            printf ("FILT4381 -- 'FPUTC' Error.\n");
            exit (17);
          }
        ++size_cnt;
        i1 = 0; i2 = 1;   /* i2 = 1 => last char. alphanumeric */
      }
    else if (c == ' ') /* if space, then ignore it but flag i1 = 1 */
      {
        ++size_cnt;
        i1 = 1;
      }
  }
else if (size_cnt >= col_data[col_num])
  {         /* Check if at end of a column */
    if (i2 == 0)   /* Check if no characters found in column */
      if ((fputc (' ', out_file)) == EOF)   /* YES => put in space */
        {
          printf ("FILT4381 -- 'FPUTC' Error.\n");
          exit (18);
        }
    if ((fputc (DELIM, out_file)) == EOF) /* Then put in delimiter */
      {
        printf ("FILT4381 -- 'FPUTC' Error.\n");
        exit (19);
      }
    ++col_num; size_cnt = 0;
    i1 = 0; i2 = 0;                 /* Reset for new column */
  }
}
```

B.2 preREAD.c

```
/*************************************************************************/
/**                                                                     **/
/**   FILE: preREAD.c       By Alec R. Champlin (April, 1988)         **/
/**  ================       As part of MIT Undergrad. Thesis          **/
/**                                                                     **/
/**  THIS ROUTINE WAS WRITTEN TO TAKE SOME OF THE BURDEN OFF OF THE     **/
/**  LISP "READ-STANDARD-TABLE" PROCEDURE, SINCE LISP COMPILER WAS      **/
/**                        NOT AVAILABLE                                **/
/**                                                                     **/
/*************************************************************************/

#include <stdio.h>

main (argc, argv)
     int  argc;
     char *argv[];
{
  int     c;
  FILE    *file, *tmp_file;

  if (argc != 2)
    {
      printf ("preREAD -- Expecting one argument, <FILENAME>.\n");
      exit (1);
    }
  if ( (file = fopen (argv[1], "r")) == NULL )
    {
      printf ("preREAD -- Couldn't open file \"%s\".\n", argv[1]);
      exit (2);
    }
  if ( (tmp_file = tmpfile ()) == NULL )
    {
      printf ("preREAD -- Couldn't open temporary file.\n");
      exit (3);
    }
  while ( (c = fgetc (file)) != EOF )
    {
      if ( c == '|' ) fputc ('\n', tmp_file);
      else fputc (c, tmp_file);
    }
  rewind(tmp_file);
  if ( freopen (argv[1], "w", file) == NULL)
    {
      printf ("preREAD -- Couldn't re-open file \"%s\".\n", argv[1]);
      exit (4);
    }
  while ( (c = fgetc (tmp_file)) != EOF ) fputc (c, file);
}
```

Appendix C
UNIX Script Files

C.1 4381FILE

```
sleep 1
echo
echo logon $1
sleep 1
echo $2
sleep 1
echo ipl cms
sleep 1
echo 'ac(noprof'
sleep 1
echo sysprof3
sleep 1
echo "$4 $3"
sleep 1
echo erase 4381lqp temp
sleep 1
echo EXEC RXCASE String
sleep 1
echo "EXEC 4381SEL $5"
sleep 10
echo ffile 4381lqp temp
sleep 2
echo logoff
```

C.2 4381FTP

```
echo "open $3"
sleep 1
echo "user $1 $2"
sleep 1
echo "cd $1 191"
sleep 1
echo "quote acct $2"
sleep 1
echo "get 4381lqp.temp $4"
sleep 10
echo delete 4381lqp.temp
sleep 1
echo quit
```

Appendix D
EXECutive Program File

D.1 4381SEL

This program is identical to the RXSELECT executive program except for the modification of the values of the *maxlength* and *maxlns* variables. These modifications have been commented below. For a better understanding of the function of the program, refer to IBM's EXEC manual.

```
/* 4381SEL sql-stmt : A modified version of the IBM RXSELECT exec */
/*   5798-DXT (C) COPYRIGHT IBM CORP. 1986       */
/*   Licensed material - Program Property of IBM */
Address 'COMMAND'
Parse Arg stmt
If stmt = '' | stmt = '?' then Do
   Say 'Format is: RXSELECT select-stmt'
   Say ''
   Say "    Where: select-stmt is any valid SQL/DS 'SELECT' statement."
   Say '           See SQL/DS documentation for more information.'
   Say ''
   Say 'The rows returned by the SELECT will be displayed using XEDIT.'
   Say 'A maximum of 100 rows will be returned.'
   Say 'If there are more rows in the result, the MORE command can be'
   Say 'used from within RXSELECT to display them.'
   Say ''
   Say 'All of the SELECT statement will be converted to upper case'
   Say 'unless the RXCASE exec has been issued with the STRING option.'
   Say 'See the RXCASE exec for information.'
   Exit 100
End
iotype = 0
said = 0
writelog = 'EXECIO 1 DISKW S$Q$L E$L$O$G A 0 V (STRING'
'QUERY CMSTYPE (LIFO'
Parse Pull . . rt .
open = 0
nextln = 0
lines = 0

NEWSEL:
'ERASE S$Q$L E$L$O$G'
maxlength = 508     /*** Modified to truncate after 508 characters ***/
maxlns = 10000      /*** Modified to hold a maximum of 10,000 rows ***/
If "TRANSLATE"("WORD"(stmt,1)) = 'SELECT' then selstmt = stmt
Else selstmt = 'Select' stmt
'GLOBALV SELECT $select GET CASE'
If case ^= 'STRING' then Upper selstmt
Call EXSQL 'PREP SELSTMT' selstmt
If rc > 4 then Signal 'FINWRT'
```

```
Call EXSQL 'DESCRIBE SELSTMT ANY'
If rc ^= 0 then Signal 'FINWRT'
fields = sqldan.0
Do i = 1 to fields
   /*Say ' Sqlda.'i "'"sqldan.i"'" sqldat.i*/
   var.i = sqldan.i
   If "INDEX"('SIFD',"LEFT"(sqldat.i,1)) ^= 0 then lr.i = 'RIGHT'
   Else lr.i = 'LEFT'
   nulls.i = "RIGHT"(sqldat.i,1) = 'N'
End
Call EXSQL 'OPEN SELSTMT'
If rc > 4 then Signal 'FINWRT'
open = 1
nextln = 1
width. = 1
MORE:
wmod = 0
If sqlcode = 0 then Do
   Do ln= nextln to nextln+maxlns-1
      Call EXSQL 'FETCH SELSTMT lnv.ln.'
      If rc > 4 then Signal 'FINWRT'
      If sqlcode = 100 then Do
         open = 0
         Call EXSQL 'CLOSE SELSTMT'
         If rc ^= 0 then Signal 'FINWRT'
         Call EXSQL 'COMMIT'
         If rc ^= 0 then Signal 'FINWRT'
         Leave
      End
      Do j = 1 to fields
         If nulls.j & "SYMBOL"('lnv.ln.j') ^= 'VAR' then lnv.ln.j = '?'
         Else Do
            iv = "LENGTH"(lnv.ln.j)
            If width.j < iv then Do
               width.j = iv
               wmod = 1
            End
         End
      End
   End
   lines = ln - 1
End
Else lines = 0
If wmod then nextln = 1
If nextln = 1 then Do
   lin = ''
   ulin = ''
   Do j = 1 to fields
      If width.j < "LENGTH"(var.j) then Do
         width.j = "LENGTH"(var.j)
      End
      If lr.j = 'LEFT' then
         lin = lin "LEFT"(var.j,width.j)
      Else
         lin = lin "RIGHT"(var.j,width.j)
      ulin = ulin "COPIES"('-',width.j)
   End
   'ERASE S$Q$L S$T$M$T'
   Call WRITE selstmt
   Call WRITE "SUBSTR"(lin,2)
```

```
   Call WRITE "SUBSTR"(ulin,2)
End
Do i = nextln to lines
   lin = ''
   Do j = 1 to fields
      If lr.j = 'LEFT' then
         lin = lin "LEFT"(lnv.i.j,width.j)
      Else
         lin = lin "RIGHT"(lnv.i.j,width.j)
   End
   Call WRITE "SUBSTR"(lin,2)
End
If ^open then Call WRITE '************** End-of-Data ***************'
FINWRT:
'FINIS S$Q$L S$T$M$T'
'FINIS S$Q$L E$L$O$G'
XED:
'GLOBALV SELECT $select SET SELECT'
If open then Push 'COMMAND MSG Enter MORE to get more rows of data.'
'ESTATEW S$Q$L E$L$O$G A'
If rc = 0 then Push 'XEDIT S$Q$L E$L$O$G'
Push 'COMMAND :'nextln+3
Push 'COMMAND SET CASE M I'
Push 'COMMAND SET SYNONYM SELECT 6 MACRO RXSELECT'
Push 'COMMAND SET SYNONYM MORE 4 MACRO RXMORE'
Push 'COMMAND SET SYNONYM SQLHELP 7 MACRO RXSQLHLP'
'XEDIT S$Q$L S$T$M$T (WIDTH' maxlength
nextln = lines + 1
AFTHELP:
'GLOBALV SELECT $select GET SELECT'
If select ^= '' then Do
   Parse Var select cmd stmt
   Upper cmd
   If cmd = 'SELECT' then Do
      'RXSQL PURGE SELSTMT'
      Signal 'NEWSEL'
   End
   Else If cmd = 'MORE' then Do
      If stmt ^= '' & "DATATYPE"(stmt,'W') then maxlns = stmt+0
      If open then Signal 'MORE'
      Else Signal 'XED'
   End
   Else If cmd = 'SQLHELP' then Do
      Upper stmt
      'EXEC RXSQLHLP' stmt
      Signal 'AFTHELP'
   End
End
If open then Do
   Call EXSQL 'CLOSE SELSTMT'
   Call EXSQL 'COMMIT'
End
'ERASE S$Q$L S$T$M$T'
'GLOBALV SELECT $select SET SELECT'
'RXSQL PURGE SELSTMT'
Exit

EXSQL: Parse Arg cmd
 'RXSQL' cmd
 If rc = 0 then Return
```

```
 If rc >= 100 then Do
    r = rc
    writelog 'RXSQL' cmd
    writelog '+++('r')+++' rxsqlmsg
    rc = r
    Return
 End
 If rc = 8 then Do
    r = rc
    writelog selstmt
    writelog 'RXSQL' cmd
    writelog '  Sqlcode:' sqlcode
    Do errd = 1 to 6
       If sqlerrd.errd ^= 0 then
          writelog '  Sqlerrd.'errd':' sqlerrd.errd
    End
    If sqlerrp ^= '' then writelog '  Sqlerrp:' sqlerrp
    If sqlerrm ^= '' then writelog '  Sqlerrm:' sqlerrm
    If sqlwarn ^= '' then writelog '  Sqlwarn:' sqlwarn
    If sqlcode ^= 0 then
       Push 'COMMAND EMSG Enter RXSQLHLP' sqlcode,
            'to get more information.'
    If "INDEX"('WS',"LEFT"(sqlwarn,1)) = 0 then 'RXSQL ROLLBACK'
    rc = r
 End
Return

WRITE: Parse Arg wrtlin
linlen = "LENGTH"(wrtlin)
If iotype = 0 then Do
   'SET CMSTYPE HT'
   'EXECIO 1 DISKW S$Q$L S$T$M$T A 0 V (VAR WRTLIN'
   r = rc
   'SET CMSTYPE' rt
   If r = 0 then Do
      If linlen > maxlength then maxlength = linlen
      Return
   End
   iotype = 1
End
If iotype = 1 then Do
   If ^said & linlen > 254 then Do
      said = 1
      Say 'Line truncated' linlen-254 'characters by EXECIO'
   End
   'EXECIO 1 DISKW S$Q$L S$T$M$T A 0 V (STRING' "LEFT"(wrtlin,254)
   maxlength = 254
End
Return
```