The Design and Implementation of System P: A Polygen Database Management System

Yeuk Hong Yuan

May 1990 WP # CIS-90-07

The Design and Implementation of System P: A Polygen Database Management System

Yeuk Hong Yuan

Bachelor of Science Thesis in Computer Science and Engineering
Massachusetts Institute of Technology
Cambridge, MA 02139

ABSTRACT

The Polygen Data Model aims at solving the problem of tagging information from multiple (poly) data sources (gen). The problem is that at present there is no database management system that can support data tags. Data tags are needed to identify where data comes from or how it evolved into its final form. This information is important for various reasons, such as for credibility judgements or for measuring cost and length of data acquisition. The proposed solution, System P, a polygen database management system tags all data from the sources and propagates these tags as necessary as the data is processed, such that when the final result is returned, each datum has a list of sources and a history of how it evolved.

ACKNOWLEDGEMENTS: Thesis advisor: Richard Wang; with assistance from the CISL Research Group. Supported in part by the MIT International Financial Services Research Center, Reuters, and AT&T.

Table of Contents

List of Figures

List of Tables

1. INTRODUCTION

System P, a prototype being developed as part of the Composite Information Systems (CIS) project at the MIT Sloan School of Management[1], aims to solve the problem of data source tagging in a multiple heterogeneous database environment. It is a multiple (poly) source (gen) database management system that tags all data from the sources and propagates these tags as the data is processed. Thus when the solution is returned, each datum has a list of sources and a history of how it evolved. Data tags can be used in various applications such as determining credibility and costs, resolving data conflicts, or in wherever knowledge of data source information plays a role.

1.1. BACKGROUND - THE CISL PROJECT

In a CIS, the complexities of multiple remote databases have been abstracted away to provide the user with a simple, single database environment in which to work. However, in case studies of actual use, it has been found that although users desire the simplicity of a virtual single database, they also want the ability to know where and how the data were collected and transformed. This information is important for various reasons, such as for credibility judgements or in measuring cost and length of data acquisition. In an environment of many hundreds or even thousands of databases these questions take on even greater significance for in such a large pool of databases, conflict and inconsistency also become major problems. Data tags are thus needed to resolve these conflicts and to give users the information they may need to measure reliability of data or time and cost.

System P allows implicit data tagging that is either transparent or visible to the user. It presently supports two types of tags, originating source tags and intermediate source tags; other types may be added simply by modifying the basic six relational operators. To the user who does not need the tags, they are not displayed but are available, because the tags are kept internally.

1.2. GOALS OF THESIS

In order to solve the data source tagging problem, System P must achieve the following goals:

(1) Process SQL queries into operation sequences.
(2) Manipulate the different tags correctly as data is transformed.
(3) Replace present GQP as central processing component of CIS/TK.
(4) Identify issues and problems for future versions of System P.

1.3. OVERVIEW OF THESIS

The focus of this thesis is on the design and implementation of System P: A Polygen Database Management System.

Chapter 2 gives an introduction to the architecture of System P as well as an overview of the previous CIS system. In particular, key components of the old system will be related to this thesis.

Chapter 3 defines the data structures used by System P.

Chapter 4 describes the Polygen Query Processor. We begin with the query translation process in detail and give an example of a query in its intermediate forms as well as its output form. Then we describe the other two components, the LQP interface and the Polygen Operations Engine.

Chapter 5 shows a detailed example of System P in action.

And finally in Chapter 6, we present the conclusion of this thesis as well as suggest some future work.

---

[1]For an overview of the CIS Project, see [WAN88].

2. SYSTEM P ARCHITECTURE

In this chapter we examine the old architecture of the CIS system and then proceed to an overview of the current System P. We finish with descriptions of two other subsystems developed simultaneous with System P and of their interactions with System P.

2.1. CIS/TK V3.0

The previous CIS architecture was built on top of CIS/TK V3.0 and is shown in Figure 1. We briefly summarize two major components of the CIS/TK, the Local Query Processor and the Global Query Processors as they are related to this thesis.[2]

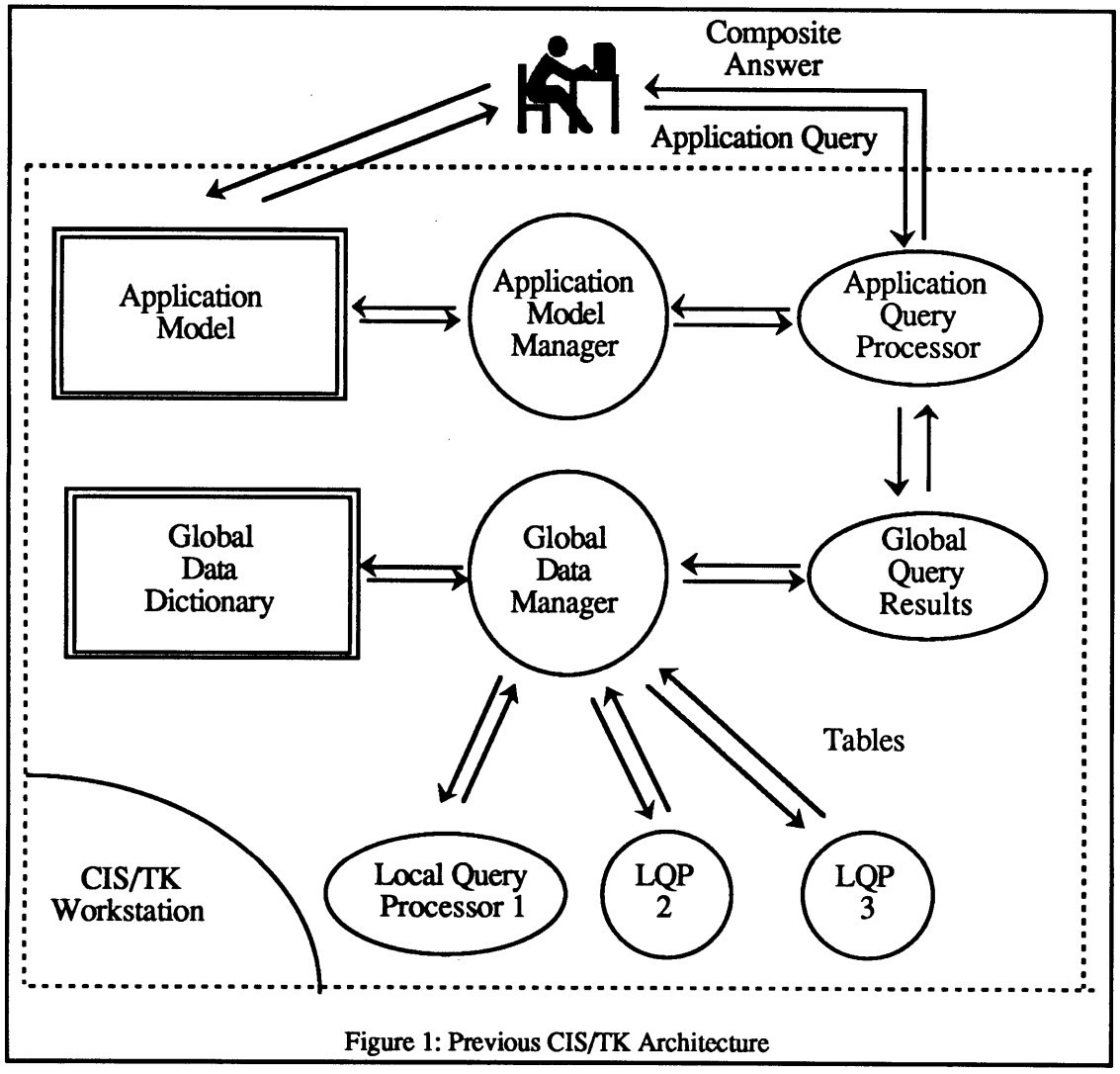

Figure 1: Previous CIS/TK Architecture

2.1.1. Local Query Processors

Local Query Processors (LQP's) are devices that retrieve data from remote databases. For each remote database there is one LQP that is associated with it to handle the communications. The interface between the LQP's and the GQP was a simple pseudo-SQL interface which allowed a query against that remote database.

---

[2]Most recent CIS/TK architecture discussion and overview in [WON88].

2.1.2. Global Query Processor

The Global Query Processor (GQP) was the central processing unit of the CIS/TK system. It took as an input a global query and returned the solution to that query to the user (or higher-level application). Like System P it parsed the query into an execution plan and manipulated the retrieved data to form the query solution.

Reasons to Replace the GQP

(1) The GQP was an adequate solution to problem that it had to handle but it could not solve the data source tagging problem.

(2) System P uses the relational algebra and is theoretically sound; the GQP operations were not based on any theory.

(3) Subsystems, e.g. Synonym Tables, within the GQP were hard to remove and did not allow for easy modifications.

2.2. SYSTEM P OVERVIEW

System P, as shown in Figure 2, consists of two major components, the Polygen Query Processor and the LQP mechanisms, and a set of supporting data structures.

The Polygen Query Processor (PQP) is the central data and query processing unit of System P; it and its components are described in full in Chapter 4. Within the PQP are the Polygen Query Translator and the Polygen Operations Engine. The Polygen Query Translator produces a schedule of operations given a Polygen SQL Query. The Polygen Operations Engine executes these operations, dispatching operations to either the LQP Manager or to the internal Data Engine. These processing units make up the PQP, and the final result of these processes is then returned to the user as the solution to the query.

The LQP subsystem performs the data retrieval for the PQP. It includes the LQP's as described above as well as an LQP Manager and a Data Catalog Bridge System - both were developed simultaneously with System P and are summarized below.

There is also a set of data structures used by System P as repositories of information, i.e. system information, or as workspaces for query and data processing. Some examples of these data structures include the Polygen Schema which represents the virtual database, and the Query Execution Plan, which is the schedule of operations produced by the Polygen Query Translator. All data structures used in System P are described in Chapter 3.

2.2.1. User Input Specifications

When using System P, the user must input three specifications. First, he[3] must input the query or conditions under which the data must fit. Secondly, he must select which, if any, databases to omit from the selection process. Finally, he needs to also determine some specifications for resolving any data conflicts. These inputs are then dispatched to their respective receivers and processed as described below.

The user, when deciding these three inputs, will have at hand or on-screen the Polygen Schema, cost reports, and credibility reports to help him decide what to choose. The Polygen Schema is used to determine what type of information is available from the CIS and from which databases this information is drawn from. Cost reports and credibility reports help the user decide what the most cost-effective and reliable database selection and data conflict rules are.

---

[3]We use the male form to denote both male and female users.

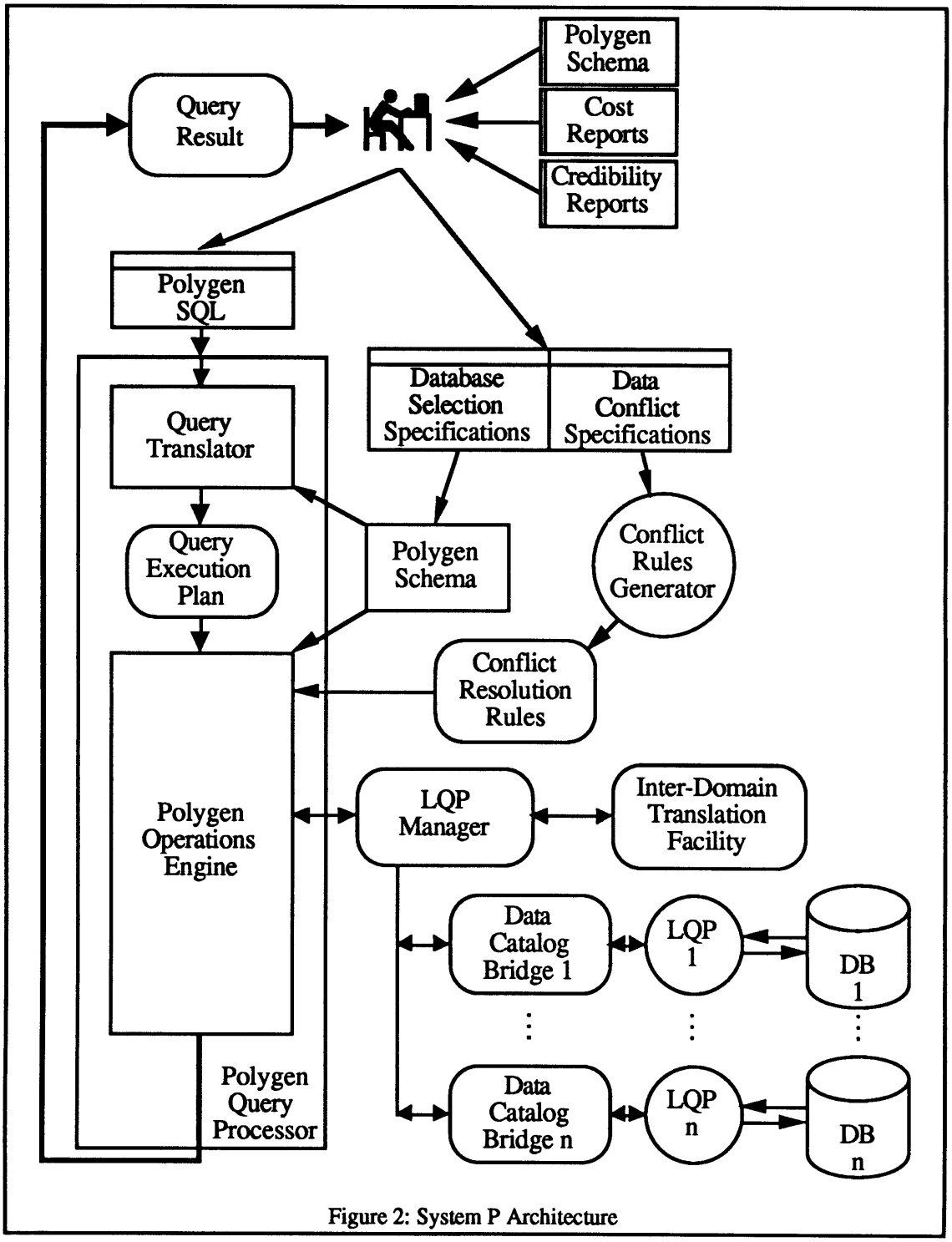

Figure 2: System P Architecture

2.2.2. Data Catalog System

The Data Catalog System [RIG90] solves formatting and scaling problems in a CIS environment. It contains a set of data catalogs, for each LQP and one system-wide. Each data catalog defines what scale, format, and units that the data is to conform with. More specifically, at each LQP there exists a bridge that converts data between the system-wide data catalog and the LQP-specific data catalog. These bridges enforce

a protocol where data inside the PQP follow one defined data catalog and data outside the PQP may fit the data catalog of the LQP where the data is.

The implications of the Data Catalog System is that, within the Polygen Query Processor we need to only consider one set of format, scales and units for each data type and that we do not have to consider the conversions necessary to make data sensible to the LQP's as that is done by the bridges.

2.2.3. LQP Manager

The LQP Manager [TUN90] provides a back-up support for information retrieval in the CIS. This is achieved by using regularly updated local copies of databases when communications to the remote database cannot be established. Thus when a query to an LQP for information is made by the Polygen Operations Engine, we are assured that it will return some data, although it may not be the most recent.

3. SYSTEM P DATA STRUCTURES[4]

3.1. POLYGEN SCHEMA

The Polygen Schema[5] defines the structure of the virtual single database of the CIS. Conceptually and in implementation, the Polygen Schema is a list of polygen relation descriptions. Each polygen relation description is a list of polygen attribute descriptions. Each polygen attribute description consists of a polygen attribute name and a list of local attribute specifications to which the polygen attribute maps to. Thus in the Polygen Schema we store the structure of the virtual single database as well as the mapping information necessary to construct polygen relations from local relations.

The following sample polygen relation definition specifies that (1) porganization is a polygen relation and (2) the polygen attributes are oname, industry, ceo and headquarters. It also specifies for each polygen attribute the local attributes that they correspond to. For example, industry corresponds to two local attributes, the ind attribute in the business relation in the AD database and the trade attribute in the corporation relation in the PD database.

```
(porganization
     ((oname ((ad business bname)
              (cd firm fname)
              (pd corporation cname)))
      (industry ((ad business ind)
                 (pd corporation trade)))
      (ceo ((cd firm ceo)))
      (headquarters ((cd firm hq)
                     (pd corporation state)))))
```

The following components use the polygen schema to look up information:

- User - inspects the schema as the reference for the CIS.
- SQL Parser[6] - in the parser, attribute names are expanded to include the relation names and the Polygen Schema provides the mapping information necessary for this transformation.
- Query Translator[7] - creates a schedule of operations to construct the polygen relations. Since polygen relations are created by collecting local relations and merging them, these operations must specify which relations to retrieve and merge. The Polygen Schema provides this mapping information to convert from polygen attribute names to local attribute names and to coordinate the retrieval processes necessary to construct the polygen relations.
- Polygen Operations Engine - when the Polygen Operations Engine retrieves data from the LQP's, the attribute names of the input relations are returned as local attribute names. The Polygen Schema information is used to transform these local attribute names to their corresponding polygen attribute names.

Database Subset Selection

In a CIS, there may be multiple users, each having their own requirements for a Polygen Schema. In System P, there exists a base polygen schema that provides the mappings to all the remote databases of the CIS, but as we saw in Figure 2 a user's database selection specification affects what the Polygen Schema will be for that user. At present, there is a simple mechanism that filters out attribute mapping information. It matches the database component of the local attribute information to a list of databases that are not to be used. The result of this filter is the Polygen Schema used for that user's session. The default configuration for System P accesses three remote databases but the user may specify which databases not to use and by doing so, create a custom Polygen Schema.

---

[4]Appendix B contains complete BNF's for the System P Data Structures.

[5]Formal definition is in [WAN90].

[6]The SQL Parser is a component of the Polygen Query Processor and is discussed in 4.1.1

[7]within the Polygen Operation Interpreter stage only.

For example, if we do not want to use the information from the Placement Database, then the above PORGANIZATION relation definition becomes:

```
(porganization
     ((oname ((ad business bname)
              (cd firm fname))
      (industry ((ad business ind)))
      (ceo ((cd firm ceo)))
      (headquarters ((cd firm hq)))))
```

3.2. LOCAL SCHEMA

For each underlying database, there is a local schema which describes the structure of the database as a list of column headers. Each column header consists of an attribute name and a data type. The data type structure is unused at present and provides a hook for future development in domain and data type research.

3.3. POLYGEN RELATION

Polygen Relations are the primary vehicles of data in System P, in particular within the PQP. When the data has been collected from the LQP's and when the internal Data Engine returns intermediate forms, they are all polygen relations. As shown in Figure 3, Polygen Relations consist of two parts: (1) a header and (2) the data. The header is a list of column headers, where each is a attribute name and a data-type. The data is represented as a list of tuples in which each tuple is a list of data cells. Each data cell is a triplet consisting of a data value and two tags, (1) originating source tag (2) intermediate source tag.

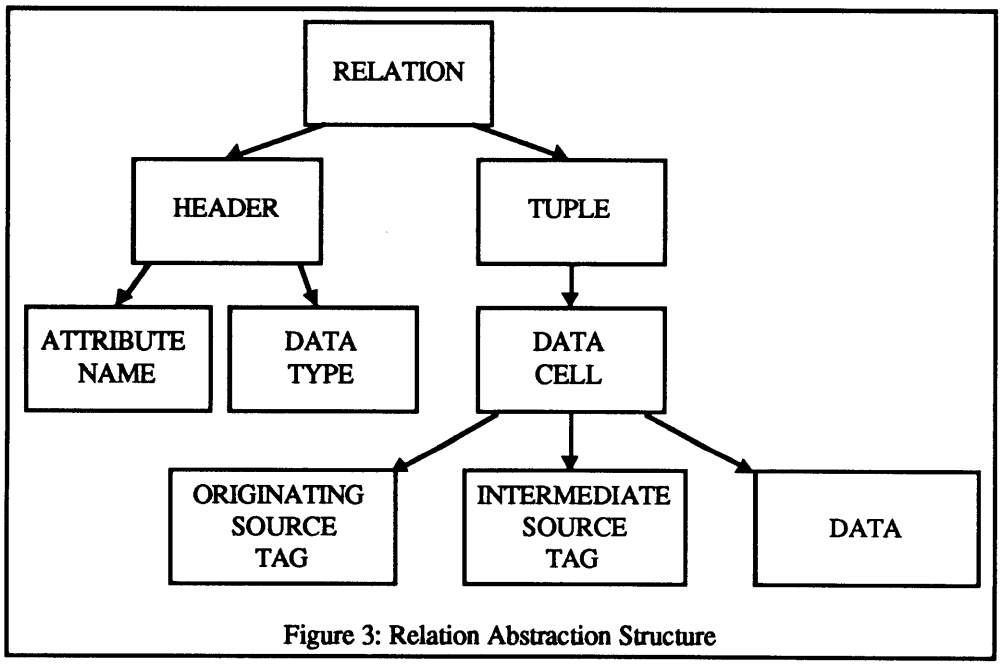

Figure 3: Relation Abstraction Structure

3.4. TAGS

Tags are used in System P to mark where data came from and how the data developed. At present, System P is implemented with two different tags, one to tag data with its sources and the other to track which databases were involved in data development. Adding more tags (e.g. monetary cost tags or time of transaction tags) in the future would require changing only the basic six polygen operators, described below.

Storage of tags is on the data cell level, where each data cell is a single data value and its associated tags. Although this scheme is space-wise inefficient, it is simple to manipulate the tags and thus complexity of tag propagation is reduced. A possible topic of future investigation is how to optimize storage of tags.

3.4.1. Originating Source Tag

Originating Source Tags (OST) mark from where data values originate. At present, the components of an OST are a database name and a relation name, although future versions may have greater level of detail. Data having more than one sources (i.e. if it came from multiple locations), have tags representing each of these sources.

3.4.2. Intermediate Source Tag

Intermediate Source Tags (IST) mark which databases have been involved in the development of the data value. Involvement occurs when the data in question interacts with other data through a polygen operation. For example, in a difference operation the result has been affected by all the data of the second relation and thus by all the originating sources of the data values of that second relation.

3.5. POLYGEN OPERATIONS MATRIX

The Polygen Operations Matrix (POM) represents the schedule of operations to be performed in the virtual single database environment. This implies that all references are to polygen attributes and polygen relations and are thus in the polygen name space. The symbols used in the POM are explained in Table 1 below. The POM is created by the Polygen Algebraic Analyzer[8] in the Query Translator of the PQP.

Table 1: POM Operation Structure Description

PR	The temporary relation name given to the result of the operation of the POM entry
OP	The operation to be performed in this step
LHR	The Left-Hand Relation, i.e. first relation of this operation, and in cases of unary operations, the relation being operated upon.
LHA	The Left-Hand Attribute of the LHR to be operated upon. In the case of a projection operation this contains the projection list, i.e. the attributes to be projected.
THETA	The theta is the comparison operator for the operation, e.g. in a selection, this could be an equal or a greater than, etc.
RHR	The Right-Hand Relation, i.e. second relation operand, for the operation.
RHA	The Right-Hand Attribute of the RHR. In the case of the select operation, this would be a constant value, e.g. an integer or a string.

A sample POM entry would be:

PR	OP	LHR	LHA	THETA	RHA	RHR
R(1)	SELECT	PALUMNUS	DEGREE	=	"MBA"	NIL

where the operation is a SELECTION and the conditional expression is:

(PALUMNUS.DEGREE = "MBA")

3.6. INTERMEDIATE OPERATIONS MATRIX

The Intermediate Operations Matrix (IOM) is the output of Polygen Operation Interpreter[9] in the Query Translator and represents the comprehensive schedule of operations necessary to answer the query. The operations in the IOM include the local query processor calls to retrieve the data and the operations in the internal Data Engine, e.g. merging of data sets from different databases. The structure of the IOM entry is

[8]The Polygen Algebraic Analyzer is discussed in Section 4.1.2.

[9]The Polygen Operation Interpretation is described in Section 4.1.3.

almost identical to that of a POM entry with some subtle differences discussed in Table 2 below and it also keeps an execution location slot.

Table 2: IOM Operation Structure Description

PR	same as POM-PR
OP	same as POM-PR but can also be either a RETRIEVE or a MERGE which are specific to the IOM.
LHR	same as POM-LHR, but in the case of the MERGE operation, the value could be a list of PR values to be merged together.
LHA	same as POM-LHA
THETA	same as POM-THETA
RHR	same as POM-RHR
RHA	same as POM-RHA
EL	The Execution Location of the operation, either a database or the PQP

IOM operations are defined such that operations to be executed at a local database use only local relation (i.e. operations with execution location values not equal to PQP) and attribute names whereas operations in the internal Data Engine are restricted to polygen relation and attribute names. We maintain this naming protocol of having all names in the internal Data Engine in the polygen name space, so that polygen operations are performed as if there is only one database and the abstraction of a single database environment is not violated.

The corresponding IOM entry to the above example would be:

PR	OP	LHR	LHA	THETA	RHA	RHR	EL
R(1)	SELECT	ALUMNUS	DEG	=	"MBA"	NIL	AD

where the LHR and LHA have been converted to the local relation name and local attribute name respectively; and the execution location is Alumni Database (AD).

3.7. QUERY EXECUTION PLAN

The Query Execution Plan (QEP) produced by the Query Optimizer[10] defines the most optimized query plan for the system to execute to resolve the query. The amount of work for the Polygen Operations Engine has been reduced by (1) moving as many operations to the local query databases and (2) projecting away all but the minimal set of data necessary to answer the query. These optimizations reduce the amount of data that needs to be moved to and within the Polygen Operations Engine, thus also reducing the space necessary for the query.

At present the QEP is taken directly from the IOM and no optimizations are made. The effects are minimal for the present CIS system because the number of attributes in the local systems is not high. However, in future systems, an optimized QEP is necessary because database relations could be quite wide by having a large number attributes.

[10]The Query Optimizer is a component of the Query Translator and is discussed in Section 6.2.

4. POLYGEN QUERY PROCESSOR

The Polygen Query Processor (PQP) receives the Polygen SQL query from the user and through a series of processes returns a result to the user. This series of operations is performed by two discrete processes, the Polygen Query Translator and the Polygen Operations Engine.

4.1. POLYGEN QUERY TRANSLATOR

The Polygen Query Translator is a chain of four processes as shown in Figure 4 and outputs the Query Execution Plan to be passed to the Polygen Operations Engine. Each process is discussed below except for the Query Optimizer which is considered future and as such is discussed in Chapter 6.

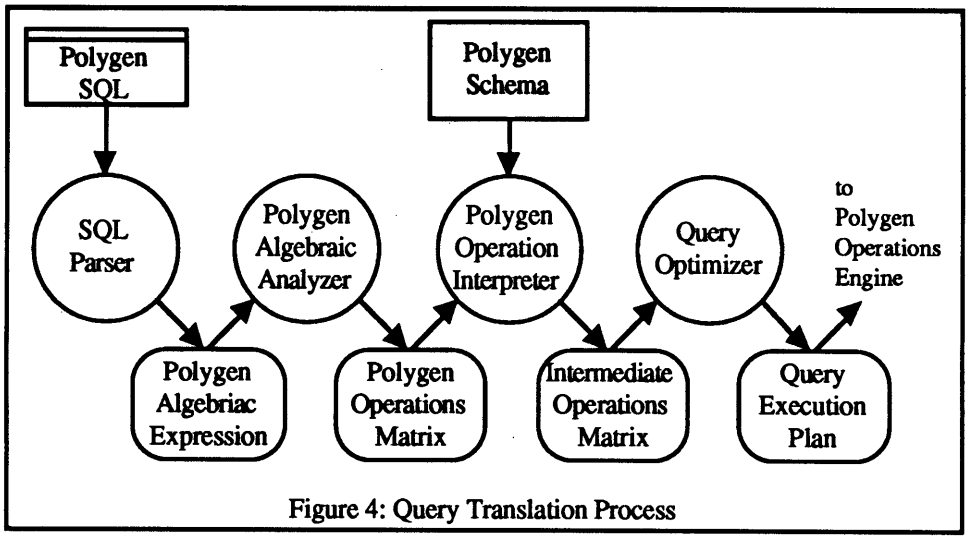

Figure 4: Query Translation Process

4.1.1. SQL Parser

The SQL Parser[11] takes an SQL query based on the Polygen Schema and returns the corresponding polygen algebraic expression. The SQL interface provides a high level query language with which the user may already have some familiarity, since SQL is common in database management systems. The parser performs several transformations on the query: (1) changes all attribute names to full form, by pairing each attribute name with the relation name, (2) simplifies the query to fit into a restricted grammar, and (3) evaluates the query within the restricted grammar into the polygen algebraic expression.

The SQL parser is implemented using the yacc and lex facilities provided in the UNIX environment.

A sample SQL query may be:

```
SELECT ONAME, CEO
FROM PORGANIZATION, PALUMNUS
WHERE CEO = ANAME AND DEGREE = "MBA"
```

and the resulting Polygen Algebraic Expression would be:

```
(((PALUMNUS (DEGREE = "MBA"))
  (ANAME = CEO)
  PORGANIZATION)
 (ONAME CEO))
```

[11]The SQL Parser is being developed independantly by Mervin Chan. See [CHA90] for details.

4.1.2. Polygen Algebraic Analyzer

The Polygen Algebraic Analyzer (PAA) produces the Polygen Operations Matrix by unnesting the Polygen Algebraic Expression[12] (PAE) from the SQL Parser. By assigning each intermediate value of the PAE to a temporary relation name, the PR, the PAA decomposes it into series of discrete polygen operations. The PAA algorithm is given in Figure 5.

```
for each relation operand of the algebraic expression
       if the relation operand is a relation name
       then
              nothing
       else
              /* expand the operand into a POM entry and keep
                 the temporary relation name to be used at the
                 end of this process */
              apply polygen algebraic analyzer to it
              store the result of the application as an POM
                     entry and note the PR value
       endif
compose the POM entry for this operation using temporary values found
from the loop above if any.
```

Figure 5: Polygen Algebraic Analyzer Algorithm

For the above PAE, the corresponding POM would be:

PR	OP	LHR	LHA	THETA	RHA	RHR
R-8	SELECTION	PALUMNUS	DEGREE	=	"MBA"	NIL
R-7	JOIN	R-8	ANAME	=	CEO	PORGANIZATION
FOO	PROJECTION	R-7	(ONAME CEO)	NIL	NIL	NIL

4.1.3. Polygen Operation Interpreter

The Polygen Operation Interpreter takes the Polygen Operations Matrix returned by the Polygen Algebraic Analyzer and through two stages, Pass One and Pass Two, creates the Intermediate Operations Matrix (IOM).

4.1.3.1. Pass One

Pass One transforms the POM into a half-processed IOM. It parses the left side of the POM, e.g. the left-hand relation (LHR) and the left-hand attribute (LHA), expanding LHR's to multiple retrieves if they map to multiple sources and renaming relations and attributes if the that operation step is handled outside the Polygen Operations Engine, i.e. occurs at some Local Query Processor. It creates a basic structure for the IOM for which Pass Two will use to complete the Polygen Operation Interpreter phase. The algorithm for Pass One is given in Figure 6 and the code is in Appendix A.7.

The half-processed IOM for the above POM would be:

PR	OP	LHR	LHA	THETA	RHA	RHR	EL
R-10	SELECTION	ALUMNUS	DEG	=	"MBA"	NIL	AD
R-16	JOIN	R-10	ANAME	=	CEO	PORGANIZATION	PQP
FOO	PROJECTION	R-16	(ONAME CEO)	NIL	NIL	NIL	PQP

[12]The BNF for the Polygen Algebraic Expression is in Appendix B.

4.1.3.2. Pass Two

Pass Two completes the IOM generation process by (1) performing the equivalent of Pass One on the right hand side of the half-processed IOM and (2) adding retrieve operations before the entry thereby resolving entries that have left hand and right hand relations and need to be retrieved from different locations. This completes the Polygen Operation Interpreter phase and the IOM is then passed to the Query Optimizer. The algorithm for Pass Two is given in Figure 7 and the code is in Appendix A.7.

The fully processed IOM for the above POM would be:

PR	OP	LHR	LHA	THETA	RHA	RHR	EL
R-10	SELECTION	ALUMNUS	DEG	=	"MBA"	NIL	AD
R-11	RETRIEVE	BUSINESS	NIL	NIL	NIL	NIL	AD
R-12	RETRIEVE	FIRM	NIL	NIL	NIL	NIL	CD
R-13	RETRIEVE	CORPORATION	NIL	NIL	NIL	NIL	PD
R-14	MERGE	(R-11 R-12 R-13)	NIL	NIL	NIL	NIL	PQP
R-16	JOIN	R-10	ANAME	=	CEO	R-14	PQP
FOO	PROJECTION	R-16	(ONAME CEO)	NIL	NIL	NIL	PQP

```
for each line in POM
        look up LHR in Polygen Schema
        /* LHR is a defined Polygen Relation */
        if LHR in Polygen Schema
        then
                /* Multiple Source Case */
                if LHR maps to MULTIPLE sources
                then
                        for each source in sources
                                retrieve source
                        merge sources
                else
                /* Single Source Case */
                        execute line at local database
                endif
        else
        /* LHR is an intermediate result */
                execute line in PQP
        endif
endfor
```

Figure 6: Pass One Algorithm

```
for each line in IOM /* Half Processed IOM */
       look up RHR in Polygen Schema
       /* RHR is a Polygen defined relation */
       if RHR in Polygen Schema
       then
              if RHR maps to MULTIPLE sources
              then
                     for each source in sources
                            retrieve source
                     merge sources
                     if line execution location is PQP
                     then
                            execute line in PQP with merge result
                     else
                            retrieve LHR of line
                            execute line with the retrieve result LHR
                                   and the merge result as RHR
                     endif
              else
                     if line execution location is PQP
                     then
                            retrieve RHR of line
                            execute line in PQP with retrieve result
                     else
                     /* expand to two retrieves since takes place */
                     /* at two separate databases                 */
                            retrieve LHR of line
                            retrieve RHR of line
                            do line in PQP with two retrieve results
                     endif
              endif
       endif
endfor
```

Figure 7: Pass Two Algorithm

4.2. POLYGEN OPERATIONS ENGINE

The Polygen Operations Engine (POE) as shown in Figure 8 is composed of two major components: (1) the Data Engine and (2) the LQP interface. The Data Engine is composed of a set of high-level and primitive operators that perform the actual manipulation of data and tags. The LQP interface communicates with the LQP's and retrieves data that is then tagged on the way into the POE. A Control Structure reads operations from the Query Execution Plan and invokes either the Data Engine or the LQP interface depending on the execution location requirement of the operation.

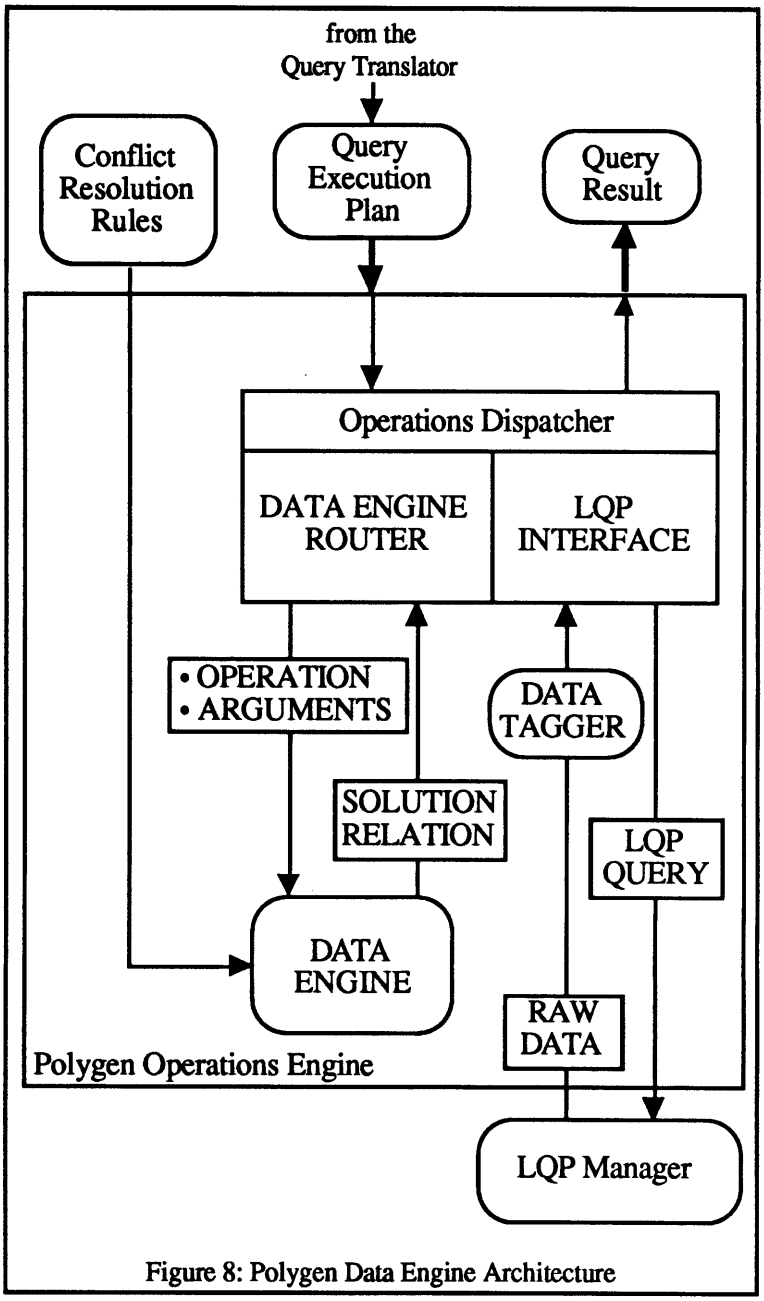

Figure 8: Polygen Data Engine Architecture

4.2.1. Data Engine

The Data Engine is the collection of both high and low level polygen operators as shown in Figure 9. Each invocation to the Data Engine is a call to one of these operator resulting in turn to either a single call or series of calls of primitive operators. All Data Engine invocations require an operator and a set of

arguments, either one or two relations with a set of supporting data, e.g. attribute specifications. Operations return solutions in the form of polygen relations as defined above so that intermediate results can be used as arguments to other Data Engine invocations.

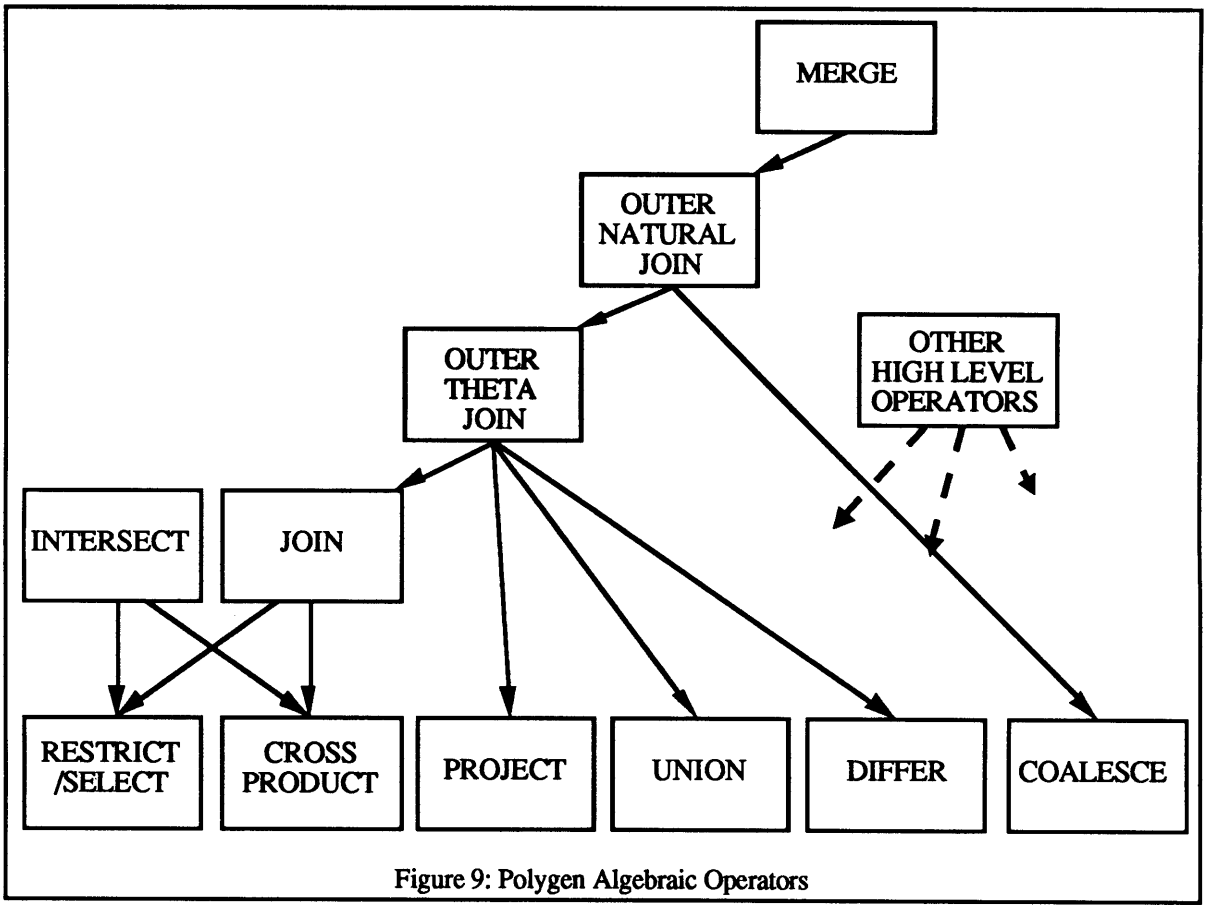

Figure 9: Polygen Algebraic Operators

There are six polygen operators that form the basis set of all operators in our system: restriction/selection, projection, cross-product, union, difference, and coalesce. Since each primitive operator is in itself complete, we can define custom higher level operators as calls to multiple primitive operators thus all higher level operators in System P. For example, intersection, join, and outer-natural-join are based on these primitive operators, in design and implementation.

Each operator performs a specific type of operation on the relation(s) as relational operators but in the polygen environment, the operators also deal with data-tags. In System P, we presently use two data-tags, an originating source tag and an intermediate source tag. The originating source tags specify from which database and relation as well as which column and tuple number; and the intermediate source tags denote which databases have been used to derive the data. These tags are changed only within the six basic operations in specific cases. Additional tags may be added and would involve re-analysis of how the new tags would be changed within the six operators. Present data tag reformulation descriptions are given below within the operations descriptions.

4.2.1.1. Primitive Operators

4.2.1.1.1. Projection

Given a polygen relation, p, a projection of p consists of mapping down the tuples and for each, keep only values for attributes in the specified column list. The result is checked for data-wise duplicates tuples and for those found, one is kept and the others are discarded. However, the data tags of each cell of the tuple being kept are changed as follows: (1) the new value for the originating source tag of the cell is the union

of that cell's old tags and the tags for all of the corresponding cells of the discarded tuples; (2) the same process is used for the intermediate source tags. This method ensures we keep the tag information for discard tuples in our final reduced result.

4.2.1.1.2. Selection/Restriction

Given a relation and a conditional specification, i.e. for a restriction, two attributes and an operator or, for a selection, an attribute, a constant and an operator, test each tuple to determine whether the conditional holds true or not. For the tuples which the conditional is true, the intermediate source tags for the cells of these tuples are modified to be the union of the old intermediate source tags and the originating data sources of the two (for the restriction or one as in the case of the selection) data cells used in the conditional.

4.2.1.1.3. Cartesian Cross-Product

Given two polygen relations, p1 and p2, returns a relation that represents the Cartesian product of the tow relations. The attributes header of each relation is the concatenation of the attributes headers of the p1 and p2.

4.2.1.1.4. Union

Given two union-compatible relations, p1 and p2, copy the tuples of both p1 and p2 into one new relation and search for data-wise duplicates; these duplicates need to be resolved into one tuple. The resultant originating data tags are the union of the duplicates originating data tags and likewise for the resultant intermediate data tags.

4.2.1.1.5. Difference

Given two union-compatible relations, p1 and p2, for each tuple in p1 determine if there is a data-wise duplicate in p2 and if so do not pass it; if not, keep it. For all data-cells of the tuples kept, their intermediate source tags are modified to be the union of the old intermediate source tags and s(p2), which is defined to be the union of all sets of originating source tags in p2.

4.2.1.1.6. Coalesce[13]

Given a relation, p, and two attributes, x and y, for each tuple in p, compare the x and y values to determine which one if any to use. If either one has nil as the value then use the other cell. If they are equal data-wise, then the new cell's originating source tag is the union of the two data cells. Likewise with the intermediate source tag. In the case the data are not equal then either use the conflict resolution rules or signal an error.

4.2.1.2. Compound Operators

Compound operators are combinations of primitive operators. Many benefits to derived from this. They provide a higher-level means to describe commonly used operations, i.e. theta-join vs. selection of a cross-product. This additional layer of abstraction allows simpler operation matrixes to be created and manipulated. Complex operators also take benefit from the fact that all data tag updates are done on the primitive operator level; as such they need not manipulate data tags in their code. Below are descriptions of some operators specific to System P, as others remain the same as in a relational environment.

---

[13]The COALESCE operation is described and formally defined in [DAT83].

4.2.1.2.1. Outer-Natural-Join[14]

Outer-Natural-Join is a powerful operation that is essential in a system with incomplete data. It allows either join keys attribute to have a null value without eliminating that tuple from the results.

4.2.1.2.2. Outer-Primary-Join[15]

The outer-primary-join is a outer natural join with the primary keys used as the join key. This operation is used to create the multiple source polygen relations as the first step of the outer-total-join.

4.2.1.2.3. Outer-Total-Join

Given two polygen based relations, R and S, returns the outer-natural-join of R and S with all associated attributes coalesced. Since local attributes for a multiple source polygen attribute map to the same polygen name in the Polygen Data Engine, associated attributes can be detected by matching attribute names; pairs located are then coalesced.

4.2.1.2.4. Merge

Given multiple polygen based relations, returns a outer-total-join reduction of the relations. The order of reduction is not ordered; the code and operation of merge is independent of the number of argument relations it takes.

4.2.2. Local Query Processor Interface

Local Query Processor (LQP) calls are invoked when the execution location for an operation is not local, i.e. not in the Polygen Operations Engine (POE). The call to the LQP's require the database name, the relation name, a projection list, a conditional expression and the name of the system data catalog. The retrieved data is tagged as it enters the POE, so that all data in the POE has data tags.

---

[14]Outer-Join is described and formally defined in [DAT83].

[15]Outer-Primary-Join, Outer-Total-Join and Merge are formally defined in [WAN90].

5. SYSTEM P APPLICATION EXAMPLE

To use System P, login to MIT2E as syspdemo, the files are automatically loaded into a ibcl[16] process and to start System P, type "(sysp)".

5.1. SAMPLE BASE RELATIONS

Below are the local relations that will be used for this example grouped by their respective local databases. For the purpose of this example all data has been changed to all caps and converted to the LISP type string and comparisons are using the string comparators.

ALUMNI Database

ALUMNUS Relation

AID#	ANAME	DEG	MAJ
012	John McCauley	MBA	IS
123	Bob Swanson	MBA	MGT
234	Stu Madnick	MBA	IS
345	James Yao	BS	EECS
456	Dave Horton	MBA	IS
567	John Reed	MBA	MGT
678	Bob Horton	SF	MGT
789	Ken Olsen	MS	EE

CAREER Relation

AID#	BNAME	POS
012	Citicorp	MIS Director
123	Genentech	CEO
234	Langley Castle	CEO
345	Oracle	Manager
456	Ford	Manager
567	Citicorp	CEO
678	BP	CEO
789	DEC	CEO
234	MIT	Professor

BUSINESS Relation

BNAME	IND
Langley Castle	Hotel
IBM	High Tech
MIT	Education
CitiCorp	Banking
Oracle	High Tech
Ford	Automobile
DEC	High Tech
BP	Energy
Genentech	High Tech

PLACEMENT Database

STUDENT Relation

SID#	SNAME	GPA	MAJOR
01	Forea Wang	3.5	Math
12	Yeuk Yuan	3.9	EECS
23	Rich Bolsky	3.2	Finance
34	John Smith	3.9	Finance
45	Mike Lavine	3.7	IS

INTERVIEW Relation

SID#	CNAME	JOB
01	IBM	System Analyst
12	Oracle	Product Manager
23	Banker's Trust	CFO
34	Citicorp	Far East Manager

CORPORATION Relation

CNAME	TRADE	STATE
Apple	High Tech	CA
Oracle	High Tech	CA
AT&T	High Tech	NY
IBM	High Tech	NY
Citicorp	Banking	NY
DEC	High Tech	MA
Banker's Trust	Finance	NY

FINANCE Database

FIRM Relation

FNAME	CEO	HQ
AT&T	Robert Allen	NY
Langley Castle	Stu Madnick	MA
Banker's Trust	Charles Sanford	NY
CitiCorp	John Reed	NY
Ford	Donald Peterson	MI
IBM	John Ackers	NY
Apple	John Sculley	CA
Oracle	Lawrence Ellison	CA
DEC	Ken Olsen	MA
Genentech	Bob Swanson	CA

FINANCE Relation

FNAME	YR	PROFIT
AT&T	1989	-1.7 bil
Langley Castle	1989	1 mil
Banker's Trust	1989	648 mil
CitiCorp	1989	1.7 bil
Ford	1989	5.3 bil
IBM	1989	5.5 bil
Apple	1989	400 mil
Oracle	1989	43 mil
DEC	1989	1.3 bil
Genentech	1989	21 mil

[16] Ibuki Common Lisp

5.2 SYSTEM P SAMPLE SESSION

In this section , we step through some of the basic features of System P, displaying the various schema information, and processing a query, first with the tags displayed and second without. Then we show a case of database selection where a user who does not desire to use all three databases asks the same query. And although the solution does not change, its evolution, as recorded in its data tags, does change.

Comments for the sample will be shown in this format.

"... skipping some output" denotes that some output was deleted from this example to save space.

```
Welcome to Macintosh Allegro Common LISP Version 1.2.2!
?
;Loading "hard:Rich's ƒ:Yeuk's Stuff:CISL:MAC.LISP"...
;Loading "hard:Rich's ƒ:Yeuk's Stuff:CISL:user.lisp"...
;Loading "hard:Rich's ƒ:Yeuk's Stuff:CISL:misc.lisp"...
;Loading "hard:Rich's ƒ:Yeuk's Stuff:CISL:structs.lisp"...
;Loading "hard:Rich's ƒ:Yeuk's Stuff:CISL:control.lisp"...
;Loading "hard:Rich's ƒ:Yeuk's Stuff:CISL:qtrans.lisp"...
;Loading "hard:Rich's ƒ:Yeuk's Stuff:CISL:ops.lisp"...
;Loading "hard:Rich's ƒ:Yeuk's Stuff:CISL:sim.lisp"...
;Loading "hard:Rich's ƒ:Yeuk's Stuff:CISL:pp.lisp"...
;Loading "hard:Rich's ƒ:Yeuk's Stuff:CISL:data.lisp"...
Load Test Data? (y or n)  y
;Loading "hard:Rich's ƒ:Yeuk's Stuff:CISL:testdata.lisp"...
Test stuff? (y or n)  n
? (sysp)
```

System P Main Menu

```
                System P Top Menu
1.   Query Editor
2.   Schema Displays
3.   System Configuration
4.   Quit

Choice: 3
```

System Configuration Menu

```
                System Configuration
1.   Turn Data Tag Display On
2.   Turn Verbose Mode Off
3.   Change User - (present: YEUK)
4.   Quit to Previous Menu
Choice: 1
```

Turn On Data Tag Display

```
                System Configuration
1.   Turn Data Tag Display Off
2.   Turn Verbose Mode Off
3.   Change User - (present: YEUK)
4.   Quit to Previous Menu
Choice: 4
```

Schema Displays Menu

... skipping some output

```
                    Schema Displays
              1.   Polygen Schema
              2.   Polygen Relation
              3.   Local Schema
              4.   Local Relation
              5.   Local Relation Data
              6.   Quit to Previous Menu
             Choice: 1
```

Shows the Polygen Relations and the Attributes for each.

```
--------------------------------------------------------------------------------
Polygen Relation:       PORGANIZATION
Attributes:       ONAME       INDUSTRY    CEO         HEADQUARTERS
--------------------------------------------------------------------------------
Polygen Relation:       PFINANCE
Attributes:       ONAME       YEAR        PROFIT
--------------------------------------------------------------------------------
Polygen Relation:       PALUMNUS
Attributes:       AID#        ANAME       DEGREE      MAJOR
--------------------------------------------------------------------------------
Polygen Relation:       PCAREER
Attributes:       AID#        ONAME       POSITION
--------------------------------------------------------------------------------
Polygen Relation:       PSTUDENT
Attributes:       SID#        SNAME       GPA         MAJOR
--------------------------------------------------------------------------------
Polygen Relation:       PINTERVIEW
Attributes:       SID#        ONAME       JOB         LOCATION
--------------------------------------------------------------------------------
```

Selecting to view PORGANIZATION

```
                    Schema Displays
              1.   Polygen Schema
              2.   Polygen Relation
              3.   Local Schema
              4.   Local Relation
              5.   Local Relation Data
              6.   Quit to Previous Menu
             Choice: 2
             Relation Name: porganization
--------------------------------------------------------------------------------
Polygen Relation:       PORGANIZATION
Attributes:       ONAME       INDUSTRY    CEO         HEADQUARTERS
--------------------------------------------------------------------------------
            Local Attribute Mapping Information for PORGANIZATION
  Polygen Attribute: ONAME
   --->             DB: AD         Rel: BUSINESS       Attribute: BNAME
   --->             DB: CD         Rel: FIRM           Attribute: FNAME
   --->             DB: PD         Rel: CORPORATION    Attribute: CNAME
  Polygen Attribute: INDUSTRY
   --->             DB: AD         Rel: BUSINESS       Attribute: IND
   --->             DB: PD         Rel: CORPORATION    Attribute: TRADE
  Polygen Attribute: CEO
   --->             DB: CD         Rel: FIRM           Attribute: CEO
  Polygen Attribute: HEADQUARTERS
   --->             DB: CD         Rel: FIRM           Attribute: HQ
   --->             DB: PD         Rel: CORPORATION    Attribute: STATE
```

Viewing Local Schema from the Schema Displays Menu

... skipping some output

```
-------------------------------------------------------------------------------
Local Database:              AD
Local Relation:              BUSINESS
Attributes:             BNAME       IND
-------------------------------------------------------------------------------
Local Database:              AD
Local Relation:              ALUMNUS
Attributes:             AID#        ANAME       DEG         MAJ
-------------------------------------------------------------------------------
Local Database:              AD
Local Relation:              CAREER
Attributes:             AID#        BNAME       POS
-------------------------------------------------------------------------------
Local Database:              CD
Local Relation:              FIRM
Attributes:             FNAME       CEO         HQ
-------------------------------------------------------------------------------
Local Database:              CD
Local Relation:              FINANCE
Attributes:             FNAME       YR          PROFIT
-------------------------------------------------------------------------------
Local Database:              PD
Local Relation:              STUDENT
Attributes:             SID#        SNAME       GPA         MAJOR
-------------------------------------------------------------------------------
Local Database:              PD
Local Relation:              INTERVIEW
Attributes:             SID#        CNAME       JOB
-------------------------------------------------------------------------------
Local Database:              PD
Local Relation:              CORPORATION
Attributes:             CNAME       TRADE       STATE
-------------------------------------------------------------------------------
```

Viewing a particular local relation

... skipping some output

```
              Choice: 4
              Local Database: ad
              Local Relation: alumnus
-------------------------------------------------------------------------------
Local Database:              AD
Local Relation:              ALUMNUS
Attribute          Domain          Polygen Relation       Polygen Attribute
 AID#               INT      --->    AID#                   PALUMNUS
 ANAME              CHAR     --->    ANAME                  PALUMNUS
 DEG                CHAR     --->    DEGREE                 PALUMNUS
 MAJ                CHAR     --->    MAJOR                  PALUMNUS
-------------------------------------------------------------------------------
```

Going to Query Editor from Main Menu

... skipping some output

Loading a sample query from a query file

```
                    Query Editor
1.   Execute Query:
  NIL
2.   Toggle Data Tag Display [ON]
3.   Enter New Query
4.   Save Query to File
5.   Load Query from File (i.e. sampq1)
6.   Quit to Previous Menu
Choice: 5
Filename: sampq1
```

A single query is in the system at any given time and this is the query that we've been following throughout the thesis.

```
                    Query Editor
1.   Execute Query:
  (((PALUMNUS (DEGREE = "MBA")) (ANAME = CEO) PORGANIZATION) (ONAME CEO))
2.   Toggle Data Tag Display [ON]
3.   Enter New Query
4.   Save Query to File
5.   Load Query from File (i.e. sampq1)
6.   Quit to Previous Menu
Choice: 1
```

Stages of the Polygen Query Translator printing values as we proceed...

Notice that in the LHR of R-5 there are three relations being merged, later there will be an example where there is only two.

```
Pass One:
(#S(IOM-LINE PR R-1 OP SELECTION LHR ALUMNUS LHA DEG THETA = RHA MBA RHR NIL EL AD)
 #S(IOM-LINE PR R-0 OP JOIN LHR R-1 LHA ANAME THETA = RHA CEO RHR PORGANIZATION EL PQP)
 #S(IOM-LINE PR FOO OP PROJECTION LHR R-0 LHA (ONAME CEO) THETA NIL RHA NIL RHR NIL EL PQP))
Pass Two:
(#S(IOM-LINE PR R-1 OP SELECTION LHR ALUMNUS LHA DEG THETA = RHA MBA RHR NIL EL AD)
 #S(IOM-LINE PR R-2 OP RETRIEVE LHR BUSINESS LHA NIL THETA NIL RHA NIL RHR NIL EL AD)
 #S(IOM-LINE PR R-3 OP RETRIEVE LHR FIRM LHA NIL THETA NIL RHA NIL RHR NIL EL CD)
 #S(IOM-LINE PR R-4 OP RETRIEVE LHR CORPORATION LHA NIL THETA NIL RHA NIL RHR NIL EL PD)
 #S(IOM-LINE PR R-5 OP MERGE LHR (R-2 R-3 R-4) LHA NIL THETA NIL RHA NIL RHR NIL EL PQP)
 #S(IOM-LINE PR R-7 OP JOIN LHR R-1 LHA ANAME THETA = RHA CEO RHR R-5 EL PQP)
 #S(IOM-LINE PR FOO OP PROJECTION LHR R-7 LHA (ONAME CEO) THETA NIL RHA NIL RHR NIL EL PQP))
```

Now, the Polygen Query Processor begins to start reading from the query execution plan.

```
***LQP_CALL***:Getting Relation from Local-Schema
Doing SELECTION DB: AD        REL: ALUMNUS
Changing to Polygen Attribute Names for ALUMNUS
Database: NIL       Relation: (SELECT ALUMNUS)
```

Result from each line in the QEP is printed. In this case we are printing with the Data Source Tags. For example, under "JOHN MCCAULEY", there is a #(AD ALUMNUS 0 1), this denotes that the datum "JOHN MCCAULEY" comes from first attribute of the zeroeth tuple of the ALUMNUS relation of AD (Alumni Database).

Notice that this is the result from the first operation

```
  #S(IOM-LINE PR R-1 OP SELECTION LHR ALUMNUS LHA DEG THETA = RHA MBA RHR NIL EL AD)
```

specifying a selection such that DEG = "MBA".

Also notice that the attribute name has been changed to the polygen one in the solution as opposed to the local one in the IOM entry, e.g. DEG (local name) vs. DEGREE (polygen name).

AID# - INT	ANAME - CHAR	DEGREE - CHAR	MAJOR - CHAR
12	JOHN MCCAULEY	MBA	IS
#(AD ALUMNUS 0 0)	#(AD ALUMNUS 0 1)	#(AD ALUMNUS 0 2)	#(AD ALUMNUS 0 3)
AD	AD	AD	AD
123	BOB SWANSON	MBA	MGT
#(AD ALUMNUS 1 0)	#(AD ALUMNUS 1 1)	#(AD ALUMNUS 1 2)	#(AD ALUMNUS 1 3)
AD	AD	AD	AD
234	STU MADNICK	MBA	IS
#(AD ALUMNUS 2 0)	#(AD ALUMNUS 2 1)	#(AD ALUMNUS 2 2)	#(AD ALUMNUS 2 3)
AD	AD	AD	AD
456	DAVE HORTON	MBA	IS
#(AD ALUMNUS 4 0)	#(AD ALUMNUS 4 1)	#(AD ALUMNUS 4 2)	#(AD ALUMNUS 4 3)
AD	AD	AD	AD
567	JOHN REED	MBA	MGT
#(AD ALUMNUS 5 0)	#(AD ALUMNUS 5 1)	#(AD ALUMNUS 5 2)	#(AD ALUMNUS 5 3)
AD	AD	AD	AD

Performing the RETRIEVE operation as it begins to gather the data to construct the PORGANIZATION relation.

```
***LQP_CALL***:Getting Relation from Local-Schema
Retrieving BUSINESS from local database...
Changing to Polygen Attribute Names for BUSINESS
Database: AD        Relation: BUSINESS
```

ONAME - CHAR	INDUSTRY - CHAR
GENENTECH	HIGH TECH
#(AD BUSINESS 8 0)	#(AD BUSINESS 8 1)
BP	ENERGY
#(AD BUSINESS 7 0)	#(AD BUSINESS 7 1)
DEC	HIGH TECH

#(AD BUSINESS 6 0)	#(AD BUSINESS 6 1)
FORD	AUTOMOBILE
#(AD BUSINESS 5 0)	#(AD BUSINESS 5 1)
ORACLE	HIGH TECH
#(AD BUSINESS 4 0)	#(AD BUSINESS 4 1)
CITICORP	BANKING
#(AD BUSINESS 3 0)	#(AD BUSINESS 3 1)
MIT	EDUCATION
#(AD BUSINESS 2 0)	#(AD BUSINESS 2 1)
IBM	HIGH TECH
#(AD BUSINESS 1 0)	#(AD BUSINESS 1 1)
LANGLEY CASTLE	HOTEL
#(AD BUSINESS 0 0)	#(AD BUSINESS 0 1)

Second RETRIEVE operation...

```
***LQP_CALL***:Getting Relation from Local-Schema
Retrieving FIRM from local database...
Changing to Polygen Attribute Names for FIRM
Database: CD        Relation: FIRM
```

ONAME - CHAR	CEO - CHAR	HEADQUARTERS - CHAR
GENENTECH	BOB SWANSON	CA
#(CD FIRM 9 0)	#(CD FIRM 9 1)	#(CD FIRM 9 2)
DEC	KEN OLSEN	MA
#(CD FIRM 8 0)	#(CD FIRM 8 1)	#(CD FIRM 8 2)
ORACLE	LAWRENCE ELLISON	CA
#(CD FIRM 7 0)	#(CD FIRM 7 1)	#(CD FIRM 7 2)
APPLE	JOHN SCULLEY	CA
#(CD FIRM 6 0)	#(CD FIRM 6 1)	#(CD FIRM 6 2)

```
| IBM               | JOHN ACKERS        | NY
-------------------------------------------------------------------------------
| #(CD FIRM 5 0)    | #(CD FIRM 5 1)     | #(CD FIRM 5 2)
-------------------------------------------------------------------------------
===============================================================================
| FORD              | DONALD PETERSON    | MI
-------------------------------------------------------------------------------
| #(CD FIRM 4 0)    | #(CD FIRM 4 1)     | #(CD FIRM 4 2)
-------------------------------------------------------------------------------
===============================================================================
| CITICORP          | JOHN REED          | NY
-------------------------------------------------------------------------------
| #(CD FIRM 3 0)    | #(CD FIRM 3 1)     | #(CD FIRM 3 2)
-------------------------------------------------------------------------------
===============================================================================
| BANKER'S TRUST    | CHARLES SANFORD    | NY
-------------------------------------------------------------------------------
| #(CD FIRM 2 0)    | #(CD FIRM 2 1)     | #(CD FIRM 2 2)
-------------------------------------------------------------------------------
===============================================================================
| LANGLEY CASTLE    | STU MADNICK        | MA
-------------------------------------------------------------------------------
| #(CD FIRM 1 0)    | #(CD FIRM 1 1)     | #(CD FIRM 1 2)
-------------------------------------------------------------------------------
===============================================================================
| AT&T              | ROBERT ALLEN       | NY
-------------------------------------------------------------------------------
| #(CD FIRM 0 0)    | #(CD FIRM 0 1)     | #(CD FIRM 0 2)
-------------------------------------------------------------------------------
===============================================================================
```

Third RETRIEVE Operation...

```
***LQP_CALL***:Getting Relation from Local-Schema
Retrieving CORPORATION from local database...
Changing to Polygen Attribute Names for CORPORATION
Database: PD        Relation: CORPORATION
-------------------------------------------------------------------------------
| ONAME - CHAR      | INDUSTRY - CHAR    | HEADQUARTERS - CHAR
===============================================================================
| BANKER'S TRUST    | FINANCE            | NY
-------------------------------------------------------------------------------
| #(PD CORPORATION 6 0)                  | #(PD CORPORATION 6 1)                  | #(PD
CORPORATION 6 2
)
-------------------------------------------------------------------------------
===============================================================================
| DEC               | HIGH TECH          | MA
-------------------------------------------------------------------------------
| #(PD CORPORATION 5 0)                  | #(PD CORPORATION 5 1)                  | #(PD
CORPORATION 5 2
)
-------------------------------------------------------------------------------
===============================================================================
| CITICORP          | BANKING            | NY
-------------------------------------------------------------------------------
| #(PD CORPORATION 4 0)                  | #(PD CORPORATION 4 1)                  | #(PD
CORPORATION 4 2
)
-------------------------------------------------------------------------------
===============================================================================
| IBM               | HIGH TECH          | NY
-------------------------------------------------------------------------------
```

```
| #(PD CORPORATION 3 0)                | #(PD CORPORATION 3 1)                | #(PD
CORPORATION 3 2
)
-------------------------------------------------------------------------------
===============================================================================
| AT&T              | HIGH TECH         | NY
-------------------------------------------------------------------------------
| #(PD CORPORATION 2 0)                | #(PD CORPORATION 2 1)                | #(PD
CORPORATION 2 2
)
-------------------------------------------------------------------------------
===============================================================================
| ORACLE            | HIGH TECH         | CA
-------------------------------------------------------------------------------
| #(PD CORPORATION 1 0)                | #(PD CORPORATION 1 1)                | #(PD
CORPORATION 1 2
)
-------------------------------------------------------------------------------
===============================================================================
| APPLE             | HIGH TECH         | CA
-------------------------------------------------------------------------------
| #(PD CORPORATION 0 0)                | #(PD CORPORATION 0 1)                | #(PD
CORPORATION 0 2
)
-------------------------------------------------------------------------------
===============================================================================
```

Going into the MERGE Operation... we demonstrate here that all high-level operations are based upon the six primitive polygen operators. We see here the first Outer-Total-Join operation being performed on the first two components.

NOTE: the descriptive messages have been truncated as to save space in this sample.

```
Doing OTJ on BUSINESS and FIRM...
Taking X-PRODUCT DB: AD       REL: BUSINESS
            WITH DB: CD       REL: FIRM
Doing RESTRICTION DB: NIL     REL: (X-PROD BUSINESS FIRM)
Doing PROJECTION DB: NIL      REL: (RESTRICT (X-PROD BUSINESS FIRM))
Doing RDIFFERENCE DB: AD      REL: BUSINESS
             WITH DB: NIL     REL: (PROJECT (RESTRICT (X-PROD BUSINESS FIRM)))
Doing PROJECTION DB: NIL      REL: (RESTRICT (X-PROD BUSINESS FIRM))
Doing RDIFFERENCE DB: CD      REL: FIRM
             WITH DB: NIL     REL: (PROJECT (RESTRICT (X-PROD BUSINESS FIRM)))
Taking X-PRODUCT DB: NIL      REL: (DIFFERENCE BUSINESS (PROJECT (RESTRICT (X-
            WITH DB: NIL      REL: NIL-REL
Taking X-PRODUCT DB: NIL      REL: NIL-REL
            WITH DB: NIL      REL: (DIFFERENCE FIRM (PROJECT (RESTRICT (X-PROD
Doing RUNION DB: NIL          REL: (X-PROD (DIFFERENCE BUSINESS (PROJECT (REST
        WITH DB: NIL          REL: (X-PROD NIL-REL (DIFFERENCE FIRM (PROJECT (
Doing RUNION DB: NIL          REL: (RESTRICT (X-PROD BUSINESS FIRM))
        WITH DB: NIL          REL: (UNION (X-PROD (DIFFERENCE BUSINESS (PROJEC
Doing COALESCE DB: NIL        REL: (UNION (RESTRICT (X-PROD BUSINESS FIRM)) (U
```

Here begins the second Outer-Total-Join between the result from the above OTJ and with the third component of PORGANIZATION, the CORPORATION relation.

```
Doing OTJ on (COALESCE (UNION (RESTRICT (X-PROD BUSINESS FIRM)) (UNION (X-PROD
Taking X-PRODUCT DB: NIL      REL: (COALESCE (UNION (RESTRICT (X-PROD BUSINESS
            WITH DB: PD       REL: CORPORATION
Doing RESTRICTION DB: NIL     REL: (X-PROD (COALESCE (UNION (RESTRICT (X-PROD
```

```
Doing PROJECTION DB: NIL      REL: (RESTRICT (X-PROD (COALESCE (UNION (RESTRIC
Doing RDIFFERENCE DB: NIL     REL: (COALESCE (UNION (RESTRICT (X-PROD BUSINESS
            WITH DB: NIL      REL: (PROJECT (RESTRICT (X-PROD (COALESCE (UNION
Doing PROJECTION DB: NIL      REL: (RESTRICT (X-PROD (COALESCE (UNION (RESTRIC
Doing RDIFFERENCE DB: PD      REL: CORPORATION
            WITH DB: NIL      REL: (PROJECT (RESTRICT (X-PROD (COALESCE (UNION
Taking X-PRODUCT DB: NIL      REL: (DIFFERENCE (COALESCE (UNION (RESTRICT (X-P
          WITH DB: NIL        REL: NIL-REL
Taking X-PRODUCT DB: NIL      REL: NIL-REL
          WITH DB: NIL        REL: (DIFFERENCE CORPORATION (PROJECT (RESTRICT
Doing RUNION DB: NIL          REL: (X-PROD (DIFFERENCE (COALESCE (UNION (RESTR
       WITH DB: NIL           REL: (X-PROD NIL-REL (DIFFERENCE CORPORATION (PR
Doing RUNION DB: NIL          REL: (RESTRICT (X-PROD (COALESCE (UNION (RESTRIC
       WITH DB: NIL           REL: (UNION (X-PROD (DIFFERENCE (COALESCE (UNION
Doing COALESCE DB: NIL        REL: (UNION (RESTRICT (X-PROD (COALESCE (UNION (
Doing COALESCE DB: NIL        REL: (COALESCE (UNION (RESTRICT (X-PROD (COALESC
Doing COALESCE DB: NIL        REL: (COALESCE (COALESCE (UNION (RESTRICT (X-PRO
```

The PORGANIZATION relation as returned from the MERGE Operation.

Notice at this stage, we have the first Intermediate Data Source Tags. For example, under ONAME, the datum "DEC" has three originating data source tags:

#(CD FIRM 8 0), #(AD BUSINESS 6 0), and #(PD CORPORATION 5 0)

and three intermediate dat source tags: CD, AD and PD.

This temporary relation display is difficult to read because the originating data source tags are wide.

Database: NIL Relation: (COALESCE (COALESCE (COALESCE (UNION (RE

ONAME - CHAR	INDUSTRY - CHAR	CEO - CHAR	HEADQUARTERS - CHAR
DEC	HIGH TECH	KEN OLSEN	MA
#(CD FIRM 8 0)	#(AD BUSINESS 6 1)	#(CD FIRM 8 1)	#(CD FIRM 8 2)
#(AD BUSINESS 6 0)	#(PD CORPORATION 5 1)		NIL
#(PD CORPORATION 5 2)			
#(PD CORPORATION 5 0)	NIL	NIL	NIL
CD	CD	CD	CD
AD	AD	AD	AD
PD	PD	PD	PD
ORACLE	HIGH TECH	LAWRENCE ELLISON	CA
#(CD FIRM 7 0)	#(AD BUSINESS 4 1)	#(CD FIRM 7 1)	#(CD FIRM 7 2)
#(AD BUSINESS 4 0)	#(PD CORPORATION 1 1)		NIL
#(PD CORPORATION 1 2)			
#(PD CORPORATION 1 0)	NIL	NIL	NIL
CD	CD	CD	CD
AD	AD	AD	AD
PD	PD	PD	PD
CITICORP	BANKING	JOHN REED	NY
#(CD FIRM 3 0)	#(AD BUSINESS 3 1)	#(CD FIRM 3 1)	#(CD FIRM 3 2)
#(AD BUSINESS 3 0)	#(PD CORPORATION 4 1)		NIL
#(PD CORPORATION 4 2)			
#(PD CORPORATION 4 0)	NIL	NIL	NIL
CD	CD	CD	CD
AD	AD	AD	AD
PD	PD	PD	PD
IBM	HIGH TECH	JOHN ACKERS	NY

```
-------------------------------------------------------------------------------
| #(CD FIRM 5 0)     | #(AD BUSINESS 1 1)                  | #(CD FIRM 5 1)     | #(CD FIRM 5 2)
| #(AD BUSINESS 1 0)                    | #(PD CORPORATION 3 1)                  | NIL                |
#(PD CORPORATION 3 2)
| #(PD CORPORATION 3 0)                 | NIL                | NIL                | NIL
-------------------------------------------------------------------------------
| CD                 | CD                 | CD                 | CD
| AD                 | AD                 | AD                 | AD
| PD                 | PD                 | PD                 | PD
===============================================================================
| AT&T               | HIGH TECH          | ROBERT ALLEN       | NY
-------------------------------------------------------------------------------
| #(CD FIRM 0 0)     | #(PD CORPORATION 2 1)                  | #(CD FIRM 0 1)     | #(CD FIRM 0 2)
| #(PD CORPORATION 2 0)                 | NIL                | NIL                | #(PD CORPORATION 2 2
)
-------------------------------------------------------------------------------
| CD                 | CD                 | CD                 | CD
| PD                 | PD                 | PD                 | PD
===============================================================================
| BANKER'S TRUST     | FINANCE            | CHARLES SANFORD    | NY
-------------------------------------------------------------------------------
| #(CD FIRM 2 0)     | #(PD CORPORATION 6 1)                  | #(CD FIRM 2 1)     | #(CD FIRM 2 2)
| #(PD CORPORATION 6 0)                 | NIL                | NIL                | #(PD CORPORATION 6 2
)
-------------------------------------------------------------------------------
| CD                 | CD                 | CD                 | CD
| PD                 | PD                 | PD                 | PD
===============================================================================
| APPLE              | HIGH TECH          | JOHN SCULLEY       | CA
-------------------------------------------------------------------------------
| #(CD FIRM 6 0)     | #(PD CORPORATION 0 1)                  | #(CD FIRM 6 1)     | #(CD FIRM 6 2)
| #(PD CORPORATION 0 0)                 | NIL                | NIL                | #(PD CORPORATION 0 2
)
-------------------------------------------------------------------------------
| CD                 | CD                 | CD                 | CD
| PD                 | PD                 | PD                 | PD
===============================================================================
| BP                 | ENERGY             | NIL                | NIL
-------------------------------------------------------------------------------
| #(AD BUSINESS 7 0)                    | #(AD BUSINESS 7 1)                     | #(NIL NIL-REL 1 1)
                     | #(NIL NIL-REL 1 2)
| NIL                | NIL                | #(NIL NIL-REL 0 1)                     | #(NIL NIL-REL 0 2)

| NIL                | NIL                | #(NIL NIL-REL 2 1)                     | #(NIL NIL-REL 2 2)

-------------------------------------------------------------------------------
| AD                 | AD                 | AD                 | NIL
| NIL                | NIL                | NIL                | NIL
| CD                 | CD                 | CD                 | NIL
===============================================================================
| MIT                | EDUCATION          | NIL                | NIL
-------------------------------------------------------------------------------
| #(AD BUSINESS 2 0)                    | #(AD BUSINESS 2 1)                     | #(NIL NIL-REL 1 1)
                     | #(NIL NIL-REL 1 2)
| NIL                | NIL                | #(NIL NIL-REL 0 1)                     | #(NIL NIL-REL 0 2)

| NIL                | NIL                | #(NIL NIL-REL 2 1)                     | #(NIL NIL-REL 2 2)

-------------------------------------------------------------------------------
| AD                 | AD                 | AD                 | NIL
| NIL                | NIL                | NIL                | NIL
| CD                 | CD                 | CD                 | NIL
===============================================================================
| LANGLEY CASTLE     | HOTEL              | STU MADNICK        | MA
-------------------------------------------------------------------------------
| #(AD BUSINESS 0 0)                    | #(AD BUSINESS 0 1)                     | #(CD FIRM 1 1)     |
#(CD FIRM 1 2)
| #(CD FIRM 1 0)     | NIL                | NIL                | NIL
-------------------------------------------------------------------------------
| AD                 | AD                 | AD                 | AD
| NIL                | NIL                | NIL                | NIL
| CD                 | CD                 | CD                 | CD
```

```
=============================================================================
| FORD              | AUTOMOBILE        | DONALD PETERSON   | MI
-----------------------------------------------------------------------------
| #(AD BUSINESS 5 0)                    | #(AD BUSINESS 5 1)                  | #(CD FIRM 4 1)    |
#(CD FIRM 4 2)
| #(CD FIRM 4 0)    | NIL               | NIL               | NIL
-----------------------------------------------------------------------------
| AD                | AD                | AD                | AD
| NIL               | NIL               | NIL               | NIL
| CD                | CD                | CD                | CD
=============================================================================
| GENENTECH         | HIGH TECH         | BOB SWANSON       | CA
-----------------------------------------------------------------------------
| #(AD BUSINESS 8 0)                    | #(AD BUSINESS 8 1)                  | #(CD FIRM 9 1)    |
#(CD FIRM 9 2)
| #(CD FIRM 9 0)    | NIL               | NIL               | NIL
-----------------------------------------------------------------------------
| AD                | AD                | AD                | AD
| NIL               | NIL               | NIL               | NIL
| CD                | CD                | CD                | CD
=============================================================================
```

Here we perform the JOIN operation form the QEP. Note that the JOIN is being translated into a restriction of a cross-product.

```
Taking X-PRODUCT DB: NIL      REL: (SELECT ALUMNUS)
            WITH DB: NIL      REL: (COALESCE (COALESCE (COALESCE (UNION (RESTR
Doing RESTRICTION DB: NIL     REL: (X-PROD (SELECT ALUMNUS) (COALESCE (COALESC
Database: NIL      Relation: (RESTRICT (X-PROD (SELECT ALUMNUS) (COAL
-----------------------------------------------------------------------------
| AID# - INT        | ANAME - CHAR      | DEGREE - CHAR     | MAJOR - CHAR      | ONAME - CHAR      |
INDUSTRY - CHAR     | CEO - CHAR        | HEADQUARTERS - CHAR
=============================================================================
| 123               | BOB SWANSON       | MBA               | MGT               | GENENTECH         |
HIGH TECH           | BOB SWANSON       | CA
-----------------------------------------------------------------------------
| #(AD ALUMNUS 1 0) | #(AD ALUMNUS 1 1) | #(AD ALUMNUS 1 2) | #(AD ALUMNUS 1 3) | #(AD BUSINESS 8 0)
                    | #(AD BUSINESS 8 1)  | #(CD FIRM 9 1)    | #(CD FIRM 9 2)
| NIL               | NIL               | NIL               | NIL               | #(CD FIRM 9 0)    |
NIL                 | NIL               | NIL
-----------------------------------------------------------------------------
| AD                | AD                | AD                | AD                | NIL               |
NIL                 | NIL               | NIL
| CD                | CD                | CD                | CD                | AD                |
AD                  | AD                | AD
| NIL               | NIL               | NIL               | NIL               | CD                |
CD                  | CD                | CD
=============================================================================
| 234               | STU MADNICK       | MBA               | IS                | LANGLEY CASTLE    |
HOTEL               | STU MADNICK       | MA
-----------------------------------------------------------------------------
| #(AD ALUMNUS 2 0) | #(AD ALUMNUS 2 1) | #(AD ALUMNUS 2 2) | #(AD ALUMNUS 2 3) | #(AD BUSINESS 0 0)
                    | #(AD BUSINESS 0 1)  | #(CD FIRM 1 1)    | #(CD FIRM 1 2)
| NIL               | NIL               | NIL               | NIL               | #(CD FIRM 1 0)    |
NIL                 | NIL               | NIL
-----------------------------------------------------------------------------
| AD                | AD                | AD                | AD                | NIL               |
NIL                 | NIL               | NIL
| CD                | CD                | CD                | CD                | AD                |
AD                  | AD                | AD
| NIL               | NIL               | NIL               | NIL               | CD                |
CD                  | CD                | CD
=============================================================================
| 567               | JOHN REED         | MBA               | MGT               | CITICORP          |
BANKING             | JOHN REED         | NY
-----------------------------------------------------------------------------
| #(AD ALUMNUS 5 0) | #(AD ALUMNUS 5 1) | #(AD ALUMNUS 5 2) | #(AD ALUMNUS 5 3) | #(CD FIRM 3 0)    |
#(AD BUSINESS 3 1)  | #(CD FIRM 3 1)    | #(CD FIRM 3 2)
| NIL               | NIL               | NIL               | NIL               | #(AD BUSINESS 3 0)
```

```
                | #(PD CORPORATION 4 1)                | NIL              | #(PD CORPORATION 4 2
)
| NIL             | NIL              | NIL               | NIL              | #(PD CORPORATION 4 0
)                 | NIL              | NIL               | NIL
-------------------------------------------------------------------------
| AD              | AD               | AD                | AD               | PD              |
PD                | PD               | PD
| CD              | CD               | CD                | CD               | AD              |
AD                | AD               | AD
| NIL             | NIL              | NIL               | NIL              | CD              |
CD                | CD               | CD
=========================================================================
```

Performing the last operation of the QEP, the PROJECTION of our solution, we have the result to our query with both sets of data tags displayed.

```
Doing PROJECTION DB: NIL      REL: (RESTRICT (X-PROD (SELECT ALUMNUS) (COALESC
Database: NIL       Relation: (PROJECT (RESTRICT (X-PROD (SELECT ALUMN
-------------------------------------------------------------------------
| ONAME - CHAR      | CEO - CHAR
=========================================================================
| GENENTECH         | BOB SWANSON
-------------------------------------------------------------------------
| #(AD BUSINESS 8 0)                    | #(CD FIRM 9 1)
| #(CD FIRM 9 0)    | NIL
-------------------------------------------------------------------------
| NIL               | NIL
| AD                | AD
| CD                | CD
=========================================================================
| LANGLEY CASTLE    | STU MADNICK
-------------------------------------------------------------------------
| #(AD BUSINESS 0 0)                    | #(CD FIRM 1 1)
| #(CD FIRM 1 0)    | NIL
-------------------------------------------------------------------------
| NIL               | NIL
| AD                | AD
| CD                | CD
=========================================================================
| CITICORP          | JOHN REED
-------------------------------------------------------------------------
| #(CD FIRM 3 0)    | #(CD FIRM 3 1)
| #(AD BUSINESS 3 0)                    | NIL
| #(PD CORPORATION 4 0)                 | NIL
-------------------------------------------------------------------------
| PD                | PD
| AD                | AD
| CD                | CD
=========================================================================
```

We execute the exact same query without the data tags display for a simple display.

... skipping some output

```
Doing PROJECTION DB: NIL      REL: (RESTRICT (X-PROD (SELECT ALUMNUS) (COALESC
Database: NIL       Relation: (PROJECT (RESTRICT (X-PROD (SELECT ALUMN
-------------------------------------------------------------------------
| ONAME - CHAR      | CEO - CHAR
=========================================================================
| GENENTECH         | BOB SWANSON
-------------------------------------------------------------------------
| LANGLEY CASTLE    | STU MADNICK
-------------------------------------------------------------------------
```

```
| CITICORP          | JOHN REED
-------------------------------------------------------------------------
```

Changing the USER also changes the Polygen Schema. In this case YEUK used all three databases but RICH prefers to use only Alumni Database and Corporate Database.

... skipping some output

```
3.   Change User - (present: YEUK)
Choice: 3
User Name: RICH
Logging in RICH ...  setting up custom polygen schema...

                System Configuration
1.   Turn Data Tag Display On
2.   Turn Verbose Mode Off
3.   Change User - (present: RICH)
4.   Quit to Previous Menu
Choice: 4
```

Displaying PORGANIZATION reveals the changes to the Polygen Schema

```
-------------------------------------------------------------------------
Polygen Relation:       PORGANIZATION
Attributes:       ONAME       INDUSTRY    CEO         HEADQUARTERS
-------------------------------------------------------------------------
            Local Attribute Mapping Information for PORGANIZATION
  Polygen Attribute: ONAME
   --->             DB: AD         Rel: BUSINESS       Attribute: BNAME
   --->             DB: CD         Rel: FIRM           Attribute: FNAME
  Polygen Attribute: INDUSTRY
   --->             DB: AD         Rel: BUSINESS       Attribute: IND
  Polygen Attribute: CEO
   --->             DB: CD         Rel: FIRM           Attribute: CEO
  Polygen Attribute: HEADQUARTERS
   --->             DB: CD         Rel: FIRM           Attribute: HQ
```

We verify this change by examining the result to the same query used by Yeuk.

First notice that the IOM result is different in that it does not contain the FIRM relation and the MERGE is only for two relations.

```
Pass Two:
(#S(IOM-LINE PR R-1 OP SELECTION LHR ALUMNUS LHA DEG THETA = RHA MBA RHR NIL EL AD)
 #S(IOM-LINE PR R-2  OP RETRIEVE LHR BUSINESS LHA NIL THETA NIL RHA NIL RHR NIL EL AD)
 #S(IOM-LINE PR R-3 OP RETRIEVE LHR  FIRM LHA NIL THETA NIL RHA NIL RHR NIL EL CD)
 #S(IOM-LINE PR R-4 OP MERGE LHR (R-2 R-3) LHA NIL THETA  NIL RHA NIL RHR NIL EL PQP)
 #S(IOM-LINE PR R-6 OP JOIN LHR R-1 LHA ANAME THETA = RHA CEO RHR R-4 EL PQP)
 #S(IOM-LINE PR FOO OP PROJECTION LHR R-6 LHA (ONAME CEO) THETA NIL RHA NIL RHR NIL EL PQP))
```

And second, notice that the solution does not use any data outside of the two selected databases, AD and CD.

```
Doing PROJECTION DB: NIL      REL: (RESTRICT (X-PROD (SELECT ALUMNUS) (COALESC
Database: NIL       Relation: (PROJECT (RESTRICT (X-PROD (SELECT ALUMN
-------------------------------------------------------------------------
| ONAME - CHAR      | CEO - CHAR
=========================================================================
```

GENENTECH	BOB SWANSON
#(AD BUSINESS 8 0)	#(CD FIRM 9 1)
#(CD FIRM 9 0)	NIL
AD	AD
CD	CD

LANGLEY CASTLE	STU MADNICK
#(AD BUSINESS 0 0)	#(CD FIRM 1 1)
#(CD FIRM 1 0)	NIL
AD	AD
CD	CD

CITICORP	JOHN REED
#(AD BUSINESS 3 0)	#(CD FIRM 3 1)
#(CD FIRM 3 0)	NIL
AD	AD
CD	CD

6. CONCLUDING REMARKS

6.1. CONCLUSIONS

In this thesis we presented a polygen database management system called System P. The motivation for System P was to develop a system that could solve the problems of data conflicts and data reliability. As discussed in [WAN90], these problems could be solved by a system that had knowledge of where the data came from and how the data was being manipulated, i.e. a system that tagged its data. In the context of the CIS system this led to System P, a data tag handling database management system that also provided the capability of creating a virtual single database environment from a distributed heterogeneous database network.

The present implementation of System P solves the data source tagging problem. It schedules and executes operations to collect and manipulate data to answer the query with a result that is tagged so that further analysis of the data for reliability, cost, etc. can be done.

6.2. FUTURE WORK

In the design and development of System P, other issues, outlined below, arose which merit future investigation and research.

Query Optimization - Since commercial or even private databases may become very large and communication and retrieval costs are high, it is important that we can determine the minimal set of data necessary to solve a query, so that less time and costs are used to retrieve data from the remote databases.

LQP Interface Refinement - Upon completion of this phase of development for the LQP's, in particular, the LQP Manager, a review of the LQP interface needs to be performed. At present there is the simple interface outlined in 4.2.2.; but as more features are added to the LQP modules, the interface may have to change to allow the PQP to access these features.

Data Conflict Resolution - Due to the lack of standards in real world databases data consistency across databases becomes a problem. Thus in a CIS, mechanisms for resolving these inconsistencies are required.

Inter-Domain Translation - A CIS needs to be able to modify queries to fit to the data provided by the remote databases, because queries may contain forms that are not found as is in a database but there exists some semantically close form that which it may map. To bridge this gap, a CIS needs to be able to swap not only units, scales as in the Data Catalog, but also between domains.

6.2.1. Query Optimization

At present the Query Execution Plan is taken directly from the IOM but this is not optimal since an overwhelming large amount of data may be retrieved from the LQP invocation even though only a fraction of it may be used. These optimizations will require a substantial subsystem that can generate the parse tree, examine it, and find the applicable optimizations to be made.

Two of the optimizations to be made include:

- Move as many filtering conditional expressions to the LQP, i.e. form complex conditional expressions for the LQP invocation.
- Find the minimal set of data necessary for the query and project away useless data at the LQP level.i.e. before it is retrieved.

6.2.2. Conflict Resolution Rules

In System P, conflicting data can occur when the coalesce operation is executed; and an error is signalled. A more robust CIS system would be able to resolve this conflict with information given by the user, i.e. to trust the data from database A and not B. At present this problem has not been fully investigated and has been left for future work.

A preliminary proposal is to query the user for a set of specifications on conflict resolution and from this information, formulate rules that would be used in the POE to handle conflicts when they occur.

6.2.3. Inter-Domain Translation

The Inter-Domain Translator (IDT) facility is necessary for any robust CIS but would be quite complex in nature and as such is beyond the scope of this thesis. The IDT would convert between data types or domains, e.g. zip code -> state, or state -> capital; not change the syntax or units of a data cell, which is the function of the Data Catalog Subsystem (Intra-Domain Translation). The IDT would be necessary because different databases may not have data as specified in the query but may have some other information which is semantically close to it, e.g. querying for New York City as opposed to a 10003 zip code.

A Data Translation Facility from CIS/TK 3.0 which functioned as described above was not be used for System P because (1) it was quite complicated and poorly organized and (2) a more well designed one will be needed once the CIS begins to add more LQP's as each will require its own set of synonyms and conversions. For these reasons the translations facility has been omitted from the present version of System P.

7. REFERENCES

[CHA90] Chan, Merwin. An SQL to Relational Algebra Translator. MIT, 1990.

[DAT83] Date, C. J. The outer join. In *Proceedings of the 2nd International Conference on Databases* (Cambridge, England, September, 1983).

[ELM89] Elmasri, R. & Navathe. S.B. Fundamentals of database systems. 1989.

[RIG90] Rigaldies, B. Technologies and Policies for the Development of Composite Information Systems in Decentralized Organizations. S.M. Thesis in Technology and Policy (Sloan School of Management, MIT, Cambridge, MA. May 1990).

[MAD88] Madnick, S. & Wang, Y. R. Integrating disparate databases for composite answers. In *Proceedings of the 21st Annual Hawaii International Conference on System Sciences* (January, 1988).

[TUN90] Tung, M. Local Query Processor Manager for the Composite Information System / Tool Kit. S.B. Thesis. (MIT, Cambridge, MA. May 1990).

[WAN88] Wang, Y. R. & Madnick, S. Connectivity among information systems. *Composite Information Systems (CIS) Project 1* (1988).

[WAN89] Wang, Y. R. & Madnick, S. A polygen data model for data source tagging in composite information systems. *WP # 3100-89 MSA.* (Sloan School of Management, MIT, Cambridge, MA. November 1989).

[WAN90] Wang, Y. R. & Madnick, S. A polygen model for heterogeneous database systems: the source tagging perspective. In the *Proceedings of the International Conference on Very Large Databases.* (Brisbane, Australia. August, 1990). Also WP#*CIS-90-01* (Sloan School of Management, MIT, Cambridge, MA. February 1990).

[WON89] Wong, T. K. Data connectivity for the composite information system/tool kit. *WP # CIS-89-03* (Sloan School of Management, MIT, Cambridge, MA. 1989), CISL Project.

APPENDIX A: SYSTEM P CODE LISTINGS

FILE ORGANIZATION

The files are maintained in the /usr/cistk/syspdemo/code directory on the MIT2E. Organization of procedures and files are shown in Table 3 below.

Table 3: Code Files Organization Table

Appendix	Filename	Description
A.1	control.lisp	procedures related to the routing and reading of operations in the PQP
A.2	data.lisp	contains the Polygen and Local Schema definitions as well as some test data
A.3	misc.lisp	procedures that did not fit well logically into any of the other files
A.4	ops.lisp	procedures that comprised the data engine of the POE, including the six primitive polygen operators and the higher-level ones.as well as a set of tag manipulation procedures that are called from the polygen operators
A.5	pdbms.lisp	the initial load up file
A.6	pp.lisp	procedures that perform various pretty printing functions, including the relation printing functions as well as some border printing functions
A.7	qtrans.lisp	procedures related to the query translation process, including the polygen algebra analyzer and the polygen operation interpreter
A.8	sim.lisp	a set of random relation generation procedures that were written to simulate extensive System P use. Never used.
A.9	structs.lisp	all the structure definitions essential to System P operations as well as structure related operations, i.e. some recursive structure copying procedures.
A.10	user.lisp	all components of the menu interface including the struct definitions, variable settings and menu declarations

A.1 CONTROL.LISP

```
;; EXECUTE-QUERY takes a polygen algebra expression, runs it
;; through the query translator and the data engine, and returns
;; the result relation

(defun execute-query (alg)
  (iom-lisp (alg-iom alg)))

;; IOM-LISP - changes iom table to lisp commands and executes them
;; return the result relation

;; temprels is used to hold temporary relations generated
;; by intermediate steps in the processing - entries in
;; temprels are ordered pairs where the car is the relation's
;; PR name and the cadr is the relation (of type struct lr-relation)

;; source-lr is used to get relations if the relation is a
;; temp, if not, will return the name of the relation to be
;; loaded if necessary

(defun iom-lisp (iom)
  (let* (temprels)
    (mapc #'(lambda (iom-entry)
              (let* ((el (ioml-el iom-entry))
                     (lr (source-lr (ioml-lhr iom-entry)
                                    el temprels))
                     (rr (source-lr (ioml-rhr iom-entry)
                                    el temprels))
                     (la (ioml-lha iom-entry))
                     (theta (convert-theta (ioml-theta iom-entry)))
                     (ra (ioml-rha iom-entry))
                     (op (ioml-op iom-entry))
                     (tempname (ioml-pr iom-entry))
                     rslt)
                (setf rslt
                      (if (equal el 'pqp)
                        ;; Operation to take place in PQP -
                        ;; PQP operations must be added here
                        (case op
                              ('restriction
                               (restriction lr la ra theta))
                              ('selection
                               (selection lr la ra theta))
                              ('projection
                               (projection lr la))
                              ('join
                               (theta-join lr rr la ra #'equalp))
                              ('merge
                               (apply #'rmerge lr))
                              (t
                               (eprint " - Unknown Operator - iom-lisp")))
                        ;; non-PQP operation - get values into newlr
                        ;; then rename attributes to Polygen ones
                        (let ((newlr (execute-lqp-op op lr la theta ra el)))
                          (change-attrs newlr lr el))))
                (print-lr rslt)
                (push (cons tempname rslt)
                      temprels)))
          iom)
    (cdar temprels)))

;; source-lr determines what the nature of the LHR
;; argument it, i.e. if it's a list then it's a merge
```

```
;; list else if it's an intermediate then get that
;; intermediate relation or return its name - in cases
;; where the lhr is a relation that needs to be retrieved
;; from the LQP's

(defun source-lr (lhr trels)
  (cond ((listp lhr) (mapcar #'(lambda (l)
                                 (source-lr l trels))
                             lhr))
        ((intermediate-p lhr)
         (cdr (assoc lhr trels)))
        (t lhr)))

(defun execute-lqp-op (op lr la theta ra el)
  (let ((newrel (get-lr el lr)))
    (case op
      ('restriction
       (restriction newrel la ra theta))
      ('selection
       (selection newrel la ra theta))
      ('projection
       (projection newrel la))
      ('retrieve
       (retrieve el newrel))
      (t
       (eprint " - Unknown Operator - EXECUTE-LQP-OP")))))

;; tests if an rname is a temporary by looking in a list
;; of all relations *all-rels*

(defun intermediate-p (rname)
  (not (member rname *all-rels* :test #'equal)))

;; poly-a-name takes a db name and a local attribute name
;; and tries to find the polygen attribute name in the
;; polygen schema; returns column name and relation name

(defun poly-a-name (db rel lname)
  (block match
    ;; if does not find polygen name returns original
    (dolist (psch *p-schema* lname)
      ;; att is list of polygen attributes
      (dolist (att (cadr psch))
        ;; lschs is list of local attributes for a polygen one
        (let ((lschs (cadr att)))
          ;; checks if db, rel and att names match
          (if (member (list db rel lname) lschs :test #'equalp)
            ;; return global attribute name
            (return-from match (values (car att) (car psch)))))))))

;; CHANGE-ATTRS - takes a relation a database name and
;; a relation name and for each column header of that
;; relation changes the column name to the polygen one

(defun change-attrs (rel r db)
  (vprint
   (format nil
           "Changing to Polygen Attribute Names for ~A~%"
           r))
  (mapc #'(lambda (cspec)
            (setf (colhead-cname cspec)
                  (poly-a-name db r (colhead-cname cspec))))
        (lr-cols rel))
  rel)
```

```
;; CONVERT-THETA takes an operation from the IOM or QUERY-PLAN
;; and selects a LISP usable operator

(defun convert-theta (theta)
  ;; more operators may be needed, also generalized operators
  ;; may be preferred to basic lisp ones.
  (case theta
    ('= #'equalp)
    ('())
    (t (eprint " - Unknown Operator - CONVERT-THETA"))))
```

A.2 DATA.LISP

```
;; local database base definitions - grouped by database

(defconstant *base-l-schema*
  `((AD
     ,(cons 'business (m-l-rel 'AD
                               'business
                               '((bname char) (ind char))
                              '(("LANGLEY CASTLE" "HOTEL" )
                               ("IBM" "HIGH TECH" )
                               ("MIT" "EDUCATION" )
                               ("CITICORP" "BANKING" )
                               ("ORACLE" "HIGH TECH" )
                               ("FORD" "AUTOMOBILE" )
                               ("DEC" "HIGH TECH" )
                               ("BP" "ENERGY" )
                               ("GENENTECH" "HIGH TECH" ))))
     ,(cons 'alumnus (m-l-rel 'AD
                               'alumnus
                               '((aid# int) (aname char) (deg char) (maj char))
                             '((012 "JOHN MCCAULEY" "MBA" "IS" )
                               (123 "BOB SWANSON" "MBA" "MGT" )
                               (234 "STU MADNICK" "MBA" "IS" )
                               (345 "JAMES YAO" "BS" "EECS" )
                               (456 "DAVE HORTON" "MBA" "IS" )
                               (567 "JOHN REED" "MBA" "MGT" )
                               (678 "BOB HORTON" "SF" "MGT" )
                               (789 "KEN OLSEN" "MS" "EE" ))))
     ,(cons 'career (m-l-rel 'AD
                               'career
                               '((aid# int) (bname char) (pos char))
                             '((012 "CITICORP" "MIS DIRECTOR" )
                               (123 "GENENTECH" "CEO" )
                               (234 "LANGLEY CASTLE" "CEO" )
                               (345 "ORACLE" "MANAGER" )
                               (456 "FORD" "MANAGER" )
                               (567 "CITICORP" "CEO" )
                               (678 "BP" "CEO" )
                               (789 "DEC" "CEO" )
                               (234 "MIT" "PROFESSOR" )))))
    (CD
     ,(cons 'firm  (m-l-rel 'CD
                            'firm
                            '((fname char) (ceo char) (hq char))
                           '(("AT&T" "ROBERT ALLEN" "NY" )
                            ("LANGLEY CASTLE" "STU MADNICK" "MA" )
                            ("BANKER'S TRUST" "CHARLES SANFORD" "NY" )
                            ("CITICORP" "JOHN REED" "NY" )
                            ("FORD" "DONALD PETERSON" "MI" )
                            ("IBM" "JOHN ACKERS" "NY" )
                            ("APPLE" "JOHN SCULLEY" "CA" )
                            ("ORACLE" "LAWRENCE ELLISON" "CA" )
                            ("DEC" "KEN OLSEN" "MA" )
                            ("GENENTECH" "BOB SWANSON" "CA" ))))
     ,(cons 'finance (m-l-rel 'CD
                               'finance
                               '((fname char) (yr int) (profit int))
                             '(("AT&T" 1989 "-1.7 BIL" )
                               ("LANGLEY CASTLE" 1989 "1 MIL" )
                               ("BANKER'S TRUST" 1989 "648 MIL" )
                               ("CITICORP" 1989 "1.7 BIL" )
                               ("FORD" 1989 "5.3 BIL" )
                               ("IBM" 1989 "5.5 BIL" )
```

```
                          ("APPLE" 1989 "400 MIL" )
                          ("ORACLE" 1989 "43 MIL" )
                          ("DEC" 1989 "1.3 BIL" )
                          ("GENENTECH" 1989 "21 MIL" )))))
    (PD
     ,(cons 'student (m-l-rel 'pd
                              'student
                              '((sid# int) (sname char) (gpa int) (major char))
                            '((01 "FOREA WANG" "3.5" "MATH" )
                              (12 "YEUK YUAN" "3.9" "EECS" )
                              (23 "RICH BOLSKY" "3.2" "FINANCE" )
                              (34 "JOHN SMITH" "3.9" "FINANCE" )
                              (45 "MIKE LAVINE" "3.7" "IS" ))))
     ,(cons 'interview (m-l-rel 'pd
                                'interview
                                '((sid# int) (cname char) (job char))
                              '((01 "IBM" "SYSTEM ANALYST" )
                                (12 "ORACLE" "PRODUCT MANAGER" )
                                (23 "BANKER'S TRUST" "CFO" )
                                (34 "CITICORP" "FAR EAST MANAGER" ))))
     ,(cons 'corporation (m-l-rel 'pd
                                  'corporation
                                  '((cname char) (trade char) (state char))
                                '(("APPLE" "HIGH TECH" "CA" )
                                  ("ORACLE" "HIGH TECH" "CA" )
                                  ("AT&T" "HIGH TECH" "NY" )
                                  ("IBM" "HIGH TECH" "NY" )
                                  ("CITICORP" "BANKING" "NY" )
                                  ("DEC" "HIGH TECH" "MA" )
                                  ("BANKER'S TRUST" "FINANCE" "NY")))))))

;; Polygen Relation Base Definitions

(defconstant *base-p-schema*
  '((porganization
     ((oname ((ad business bname)
              (cd firm fname)
              (pd corporation cname)))
      (industry ((ad business ind)
                 (pd corporation trade)))
      (ceo ((cd firm ceo)))
      (headquarters ((cd firm hq)
                     (pd corporation state)))))
    (pfinance
     ((oname ((cd finance fname)))
      (year ((cd finance yr)))
      (profit ((cd finance profit)))))
    (palumnus
     ((aid# ((ad alumnus aid#)))
      (aname ((ad alumnus aname)))
      (degree ((ad alumnus deg)))
      (major ((ad alumnus maj)))))
    (pcareer
     ((aid# ((ad career aid#)))
      (oname ((ad career bname)))
      (position ((ad career pos)))))
    (pstudent
     ((sid# ((pd student sid#)))
      (sname ((pd student sname)))
      (gpa ((pd student gpa)))
      (major ((pd student major)))))
    (pinterview
     ((sid# ((pd interview sid#)))
      (oname ((pd interview cname)))
```

```
      (job ((pd interview job)))
      (location ((pd interview loc)))))))

;; User Schemes

(defvar *bad-dbs* '())
(defvar *l-schema* *base-l-schema*)
(defvar *p-schema* (user-filter *bad-dbs* *base-p-schema*))

(defvar *all-rels*
  (append (mapcar #'car
                  *p-schema*)
          (flatten
           (mapcar #'(lambda (x)
                       (mapcar #'car (cdr x)))
                   *l-schema*))))
```

A.3 MISC.LISP

```
;; flatten takes any list of any depth and returns a one-level list
(defun flatten (lst)
  (if (null lst)
      nil
    (if (atom lst)
       (list lst)
      (apply 'append (mapcar 'flatten lst)))))

;; nsubseq takes (1) a sequence (2) start index and
;; (3) end index and returns that part of the sequence
;; NOTE: We use this because if subseq gets an end spec
;; that is greater than the length of the sequence
;; it errors and we don't want that error

(defun nsubseq (str st end)
  (if (> end (length str))
    str
    (subseq str st end)))

;; new-pr returns a new symbol with R- prefix
(defun new-pr ()
  (gentemp "R-"))

;; range returns a list of numbers starting from
;; st to en omitting numbers in drplst
;; USED IN: Projection Operations
(defun range (st en &optional (drplst '()))
  (let (ans '())
    (dotimes (n (- en st) ans)
      (let ((x (+ n st)))
        (if (not (member x drplst))
          (setf ans (append ans (list x))))))))

;; sequal is equal test but nil is NOT equal to nil
(defun sequal (op1 op2)
  (if (and (null op1) (null op2))
    nil
    (equal op1 op2)))

;; src-equal - checks if srctags are equal
;; This procedure is not necessary on some implementations
;; of LISP, but the MAC DEFINITELY needs it.

(defun src-equal (tg1 tg2)
  (and (equal (srctag-db tg1) (srctag-db tg2))
       (equal (srctag-rel tg1) (srctag-rel tg2))
       (equal (srctag-row tg1) (srctag-row tg2))
       (equal (srctag-col tg1) (srctag-col tg2))))

;; erase removes ALL data occurence of tple in lst
(defun erase (tple lst)
  (if (endp lst)
      '()
    (if (equalp (mapcar #'cell-data tple)
                (mapcar #'cell-data (car lst)))
      (cdr lst)
      (cons (car lst) (erase tple (cdr lst))))))

;; COLLOOKUP checks if the col is a number and if
;; not, does a lookup in the colhead to find the position number
;; else checks numbers validity and in both sets the col spec
;; to the number as necessary.
```

```
;; COLLOOKUP2 is a two col version that handles ambiguities if the
;; other col spec is the duplicate

(defmacro collookup (collst col1)
  `(if (numberp ,col1)
     (check-bounds ,collst ,col1)
     (let ((ans (position ,col1 ,collst :key #'colhead-cname
                          :test #'equalp)))
       (cond ((not ans)
              (eprint " - Column name not found - COLLOOKUP"))
             ((> (count ,col1 ,collst :key #'colhead-cname
                        :test #'equalp)
                 1)
              ;; ignore and return leftmost
              (setf ,col1 ans))
              ;;(eprint " - Ambiguous Column Name - COLLOOKUP"))
             (t (setf ,col1 ans))))))

(defmacro collookup2 (collst col1 col2)
  `(let* ((ans1a (if (numberp ,col1)
                   (check-bounds ,collst ,col1)
                   (position ,col1 ,collst :key #'colhead-cname
                             :test #'equalp)))
          (ans1b (if (numberp ,col1) ,col1
                     (position ,col1 ,collst :key #'colhead-cname
                               :test #'equalp :from-end t)))
          (ans2 (if (numberp ,col2)
                  (check-bounds ,collst ,col2)
                  (position ,col2 ,collst :key #'colhead-cname
                            :test #'equalp :from-end t))))
     ;; if col1 is found and either col2 was not bound
     ;; or col2 was found
     (if (and ans1a ans2)
       ;; if there is a unique ans1a and ans2
       ;; or there are two cols by same name - ans1b = ans2
       ;; and they are both specified
       ;;
       ;; ignore and return leftmost
       ;;(if (or (= ans1a ans1b) (= ans1b ans2))
       (and (setf ,col1 ans1a)
            (setf ,col2 ans2))
         ;; else all three are different or ans1a = ans2
         ;; (eprint " - Ambiguous Column Name - COLLOOKUP"))
       ;; if either no col1 match was found or
       ;; no col2 match could not be found
       (eprint " - Column name not found - COLLOOKUP"))))

(defmacro check-bounds (clst c)
  `(if (numberp ,c)
     (cond ((< ,c 0)
            (eprint " - Colnum less than zero - COLLOOKUP"))
           ((>= ,c (list-length ,clst))
            (eprint " - Colnum greater than no.  of cols - COLLOOKUP"))
           (t ,c))))
```

A.4 OPS.LISP

```
;; ========================================
;; Six Basic Relational Operators
;; ========================================

;; Projection, Restriction, Selection, Coalesce
;; allow colnumbers or colnames
;; checks both x-ool and y-col separately so they do not
;; have to be the same type
;; note: location of colnumber from -col will find
;; only LEFTMOST column named attribute
;; collookup is a macro - see misc.lisp

(defun projection (rel collist)
  (vprint (format nil "Doing PROJECTION DB: ~A~30,10TREL: ~A~%"
                  (lr-db rel) (lr-rel rel)))
  (let* ((relname (format nil "(PROJECT ~A)" (lr-rel rel)))
         (olddata (lr-data rel))
         (oldhead (lr-cols rel))
         (ncollist (mapcar #'(lambda (colspec)
                               (collookup oldhead colspec))
                           collist))
         (newdata (mapcar #'(lambda (tuple)
                              (mapcar #'(lambda (idx)
                                          (nth idx tuple))
                                      ncollist))
                          olddata)))
    (make-l-relation
     :rel relname
     :cols (mapcar #'(lambda (idx)
                       (copy-colhead (nth idx oldhead)))
                   ncollist)
     :data (join-duplicate-tuples newdata))))

(defun cross-product (rel1 rel2)
  (vprint
   (format nil "Taking X-PRODUCT DB: ~A~30,10TREL: ~A~%"
           (lr-db rel1) (lr-rel rel1))
   (format nil "~12,0TWITH DB: ~A~30,10TREL: ~A~%"
           (lr-db rel2) (lr-rel rel2)))
  (let* ((relname (format nil "(X-PROD ~A ~A)" (lr-rel rel1) (lr-rel rel2)))
         (r1 (copy-l-relation rel1))
         (r2 (copy-l-relation rel2))
         (olddata1 (lr-data r1))
         (olddata2 (lr-data r2))
         (tempdata '()))
    (mapc #'(lambda (tuple1)
              (mapc #'(lambda (tuple2)
                        (push (append tuple1 tuple2)
                              tempdata))
                    olddata2))
          olddata1)
    (make-l-relation
     :rel relname
     :cols (append (lr-cols r1)
                   (lr-cols r2))
     :data tempdata)))

(defun restriction (rel x-col y-col theta)
  (vprint (format nil "Doing RESTRICTION DB: ~A~30,10TREL: ~A~%"
                  (lr-db rel) (lr-rel rel)))
  (let* ((relname (format nil "(RESTRICT ~A)" (lr-rel rel)))
         (r (copy-l-relation rel))
```

```
        (oldcols (lr-cols r))
         (olddata (lr-data r))
         (tempdata '()))
    (collookup2 oldcols x-col y-col)
    (mapc #'(lambda (tuple)
              (let ((x-cell (nth x-col tuple))
                    (y-cell (nth y-col tuple)))
                (if (apply theta
                           (list (cell-data x-cell)
                                 (cell-data y-cell)))
                    (progn
                      (let ((srcs (mapcar #'srctag-db
                                          (union (cell-srctags x-cell)
                                                 (cell-srctags y-cell)
                                                 :test #'src-equal))))
                        (mapc #'(lambda (cell)
                                  (setf (cell-inttags cell)
                                        (union (cell-inttags cell) srcs)))
                              tuple)
                        (push tuple tempdata))))))
          olddata)
    ;; returns a relation even if its empty
    (make-l-relation :rel relname
                     :cols oldcols
                     :data tempdata)))

(defun selection (rel x-col const theta)
  (vprint (format nil "Doing SELECTION DB: ~A~30,10TREL: ~A~%"
                  (lr-db rel) (lr-rel rel)))
  (let* ((relname (format nil "(SELECT ~A)" (lr-rel rel)))
         (r (copy-l-relation rel))
         (olddata (lr-data r))
         (oldcols (lr-cols r))
         (tempdata '()))
    (collookup oldcols x-col)
    (mapc #'(lambda (tuple)
              (let ((x-cell (nth x-col tuple)))
                (if (apply theta (list (cell-data x-cell)
                                       const))
                    (progn
                      (let ((srcs (mapcar #'srctag-db
                                          (cell-srctags x-cell))))
                        (mapc #'(lambda (cell)
                                  (setf (cell-inttags cell)
                                        (union (cell-inttags cell) srcs)))
                              tuple)
                        (push tuple tempdata))))))
          olddata)
    ;; returns a relation even if its empty
    (make-l-relation :rel relname
                     :cols oldcols
                     :data tempdata)))

(defun runion (rel1 rel2)
  (vprint
   (format nil "Doing RUNION DB: ~A~30,10TREL: ~A~%"
           (lr-db rel1) (lr-rel rel1))
   (format nil "~8,0TWITH DB: ~A~30,10TREL: ~A~%"
           (lr-db rel2) (lr-rel rel2)))
  (if (compatible rel1 rel2)
      (make-l-relation
       :rel (format nil "(UNION ~A ~A)" (lr-rel rel1) (lr-rel rel2))
       :cols (copy-cols (lr-cols rel1))
       :data (join-duplicate-tuples
```

```
            (append (copy-data (lr-data rel1))
                    (copy-data (lr-data rel2)))))
    (eprint " - Relations not union-compatible - runion")))

(defun rdifference (rel1 rel2)
  (vprint
   (format nil "Doing RDIFFERENCE DB: ~A~30,10TREL: ~A~%"
           (lr-db rel1) (lr-rel rel1))
   (format nil "~13,0TWITH DB: ~A~30,10TREL: ~A~%"
           (lr-db rel2) (lr-rel rel2)))
  (if (compatible rel1 rel2)
      (let* ((relname (format nil "(DIFFERENCE ~A ~A)"
                              (lr-rel rel1) (lr-rel rel2)))
             (allsrcs (get-all-srcs rel2))
             (tempdata (copy-data (lr-data rel1))))
        (mapc #'(lambda (tuple)
                  (setf tempdata
                        (erase tuple tempdata)))
              (lr-data rel2))
        (make-l-relation :rel relname
                         :cols (copy-cols (lr-cols rel1))
                         :data (addinttags tempdata allsrcs)))
    (eprint " - Relations not union-compatible - rdifference")))

(defun coalesce (rel col1 col2)
  (vprint (format nil "Doing COALESCE DB: ~A~30,10TREL: ~A~%"
                  (lr-db rel) (lr-rel rel)))
  (let* ((nrel (copy-l-relation rel))
         (relname (format nil "(COALESCE ~A)" (lr-rel nrel)))
         (rellen (rel-cols nrel))
         (relcnt (rel-rows nrel))
         (oldcols (lr-cols nrel))
         (olddata (lr-data nrel))
         newdata
         newcols)
    (collookup2 oldcols col1 col2)
    (setf newcols (range 0 rellen (list col2)))
    (dotimes (rownum relcnt)
             (let* ((cell1 (nth col1 (nth rownum olddata)))
                    (data1 (cell-data cell1))
                    (cell2 (nth col2 (nth rownum olddata)))
                    (data2 (cell-data cell2)))
               (cond ((null data1)
                      ;; cell level copy - when x is nil
                      (setf (nth col1 (nth rownum olddata))
                            (nth col2 (nth rownum olddata))))
                     ((equal data1 data2)
                      (setf (cell-srctags cell1)
                            (union (cell-srctags cell1)
                                   (cell-srctags cell2)
                                   :test #'src-equal))
                      (setf (cell-inttags cell1)
                            (union (cell-inttags cell1)
                                   (cell-inttags cell2)
                                   :test #'equal)))
                     ((null data2))
                     (t
                      (eprint " - Data not compatible - Coalesce")))))
    (make-l-relation
     :rel relname
     :data (join-duplicate-tuples
            (mapcar #'(lambda (tuple)
                        (mapcar #'(lambda (idx)
                                    (nth idx tuple))
```

```
                              newcols))
                    olddata))
     :cols (mapcar #'(lambda (idx)
                       (copy-colhead (nth idx oldcols)))
                   newcols))))

(defun retrieve (db rel)
  (vprint (format nil "Retrieving ~A from local database...~%"
                  (lr-rel rel)))
  (copy-l-relation rel))

;; ============================================
;; Compound Relational Operators
;; ============================================

;; All compound operators here allow column specifications
;; either (a) as a colnum for the respective relation
;;        (b) as a column name for the respective relation

;; RINTERSECTION
(defun rintersection (rel1 rel2)
  (vprint
   (format nil "Doing RINTERSECTION DB: ~A~30,10TREL: ~A~%"
           (lr-db rel1) (lr-rel rel1))
   (format nil "~15,0TWITH DB: ~A~30,10TREL: ~A~%"
           (lr-db rel2) (lr-rel rel2)))
  (if (compatible rel1 rel2)
      (let* ((relname (format nil "(INTERSECTION ~A ~A)"
                              (lr-rel rel1) (lr-rel rel2)))
             (mtchlst1 (lr-cols rel1))
             (mtchlst2 (lr-cols rel2))
             (nocols (length mtchlst2))
             (temprel (cross-product rel1 rel2)))
        (dotimes (n (length mtchlst1))
          (setf temprel
                (restriction temprel n (+ nocols n) #'equalp)))
        (dotimes (n (length mtchlst1))
          (setf temprel
                (coalesce temprel n nocols)))
        (make-l-relation :rel relname
                         :cols (lr-cols temprel)
                         :data (lr-data temprel)))
    (eprint " - Relations not union-compatible - rintersection")))

(defun theta-join (rel1 rel2 col1 col2 op)
  (restriction (cross-product rel1 rel2)
               (collookup (lr-cols rel1) col1)
               (+ (rel-cols rel1)
                  (collookup (lr-cols rel2) col2))
               op))

;; Natural Join can only be an EQUI-join

(defun natural-join (rel1 rel2 col1 col2)
  (let* ((temp (theta-join rel1
                           rel2
                           (collookup (lr-cols rel1) col1)
                           (collookup (lr-cols rel2) col2)
                           #'equalp))
         (xlen (+ (rel-cols rel1) col2)))
    (projection temp
                (range 0
                       (rel-cols temp)
                       (list xlen)))))
```

```
(defun outer-theta-join (R S col1 col2 op)
  (let* ((T1 (theta-join R
                         S
                         (collookup (lr-cols R) col1)
                         (collookup (lr-cols S) col2)
                         op))
         (Rlen (rel-cols R))
         (Slen (rel-cols S))
         (R1 (rdifference
              R
              (projection T1 (range 0 Rlen))))
         (S1 (rdifference
              S
              (projection T1 (range Rlen (+ Rlen Slen))))))
    (runion
     T1
     (runion
      (cross-product R1 (mnr (lr-cols S)))
      (cross-product (mnr (lr-cols R)) S1)))))

(defun outer-natural-join (R S r-col s-col &optional (theta #'sequal))
  (coalesce (outer-theta-join R
                              S
                              (collookup (lr-cols R) r-col)
                              (collookup (lr-cols S) s-col)
                              theta)
            r-col
            (+ s-col (rel-cols R))))

;; OUTER-TOTAL-JOIN - both relations are in polygen based form
;; since all attributes are renamed upon entry into PQP
;; and this is only called after retrieves are executed.

(defun outer-total-join (R S)
  (vprint
   (format nil "Doing OTJ on ~A and ~A...~%" (lr-rel R) (lr-rel S)))
  (let* (temprel
         (Scols (lr-cols S))
         (Rcols (lr-cols R)))
    (block matchcols
      ;; find first match from R and S to be join attributes
      (dolist (Rc Rcols (eprint " - No matching columns - OTJoin"))
        (let ((n (position Rc Scols :test #'equalp)))
          ;; if found return minimal info to OTJ operation
          ;; e.g. a column name specifier and a column no.  spec
          (if n (return-from
                 matchcols
                 (setf temprel
                       (outer-natural-join R S (colhead-cname Rc) n)))))))
    (dolist (Rc Rcols temprel)
      ;; n first occurence of rc - from R
      ;; m second attribute - from S
      (let* ((cols (lr-cols temprel))
             (n (position Rc cols :test #'equalp))
             (m (position Rc cols :test #'equalp :start (+ n 1))))
        (if (and n m)
          (setf temprel
                (coalesce temprel n m)))))))

(defun rmerge (&rest rels)
  (reduce #'outer-total-join rels))

;; ========================================
```

```
;; Utility Functions
;; ==========================================

(defun nulllist (n)
  (let ((st '()))
    (dotimes (x n st)
            (push nil st))))

;; make nil relation

(defun mnr (collst)
  (m-l-rel '()
          'nil-rel
          collst
          (nulllist (list-length collst))))

;; COMPATIBLE - checks for union-compatibility
(defun compatible (rel1 rel2)
  (equalp (lr-cols rel1) (lr-cols rel2)))

;; JOIN-DUPLICATE-TUPLES
(defun join-duplicate-tuples (datalist)
  (let ((dataonly (mapcar #'(lambda (tuple)
                              (mapcar #'cell-data tuple))
                          datalist)))
    (if (equal (remove-duplicates dataonly :test #'equal) dataonly)
       datalist
      (let ((i 0))
       (loop
        (let ((idx (position (nth i dataonly)
                             (nthcdr (+ 1 i) dataonly) :test #'equal)))
          (if idx
              (progn
               (setf (nth (+ 1 idx i) datalist)
                     (join-tuple (nth i datalist)
                                 (nth (+ 1 idx i) datalist)))
               (setf dataonly (remove (nth i dataonly) dataonly))
               (setf datalist (remove (nth i datalist) datalist :count 1))
               (setf i (- i 1))))
          (setf i (+ i 1))
          (if (> i (length dataonly))
              (return datalist))))))))

(defun join-tuple (t1 t2)
  (mapcar #'(lambda (c1 c2)
              (make-cell :srctags (union (cell-srctags c1)
                                         (cell-srctags c2)
                                         :test #'src-equal)
                         :inttags (union (cell-inttags c1)
                                         (cell-inttags c2)
                                         :test #'equal)
                         :data (cell-data c1)))
          t1 t2))

;; ADDINTTAGS - changes all cells in the data matrix to have the
;; inttags unioned to their existing ones

(defun addinttags (datmat tags)
  (mapcar #'(lambda (tple)
              (mapcar #'(lambda (cell)
                          (setf (cell-inttags cell)
                                (union (cell-inttags cell)
                                       tags :test #'equal))
                          cell)
```

```
                    tple))
        datmat))

;; GET-ALL-SRCS - takes a relation and returns a list of all sources
;; for all the cells of the relation - only db names

(defun get-all-srcs (rel)
  (remove-duplicates
   (mapcar #'srctag-db
           (flatten
            (mapcar #'(lambda (tple)
                        (mapcar #'cell-srctags tple))
                    (lr-data rel))))
   :test #'equal))
```

A.5 PDBMS.LISP

```
(setf *load-verbose* t)
(setf *print-pretty* t)
(setf *line-width* 80)

;; ====================
;; System Esentials
;; ====================
(load "user.lisp")
(load "misc.lisp")
(load "structs.lisp")
(load "control.lisp")

;; ====================
;; Parsers
;; ====================
(load "qtrans.lisp")

;; ====================
;; Engine
;; ====================
(load "ops.lisp")
(load "sim.lisp")
(load "pp.lisp")

;; ====================
;; Data Loading
;; ====================
(load "data.lisp")

;; ====================
;; Testing Stuff
;; ====================
(if (y-or-n-p "Load Test Data?")
  (load "testdata.lisp"))
(if (y-or-n-p "Test stuff?")
    (load "debug.lisp"))
```

A.6 PP.LISP

```
;; ============================================
;; User Interface
;; ============================================

;; Schema and Relation Display Functions

;; Printing Polygen Schemes and Relations
;; la-flg controls printing of local attributes

(defun print-p-schema (ps &optional la-flg)
  (mapc #'(lambda (rel) (print-p-relation rel la-flg)) ps)
  (values))

(defun print-p-relation (poly-rel &optional la-flg)
  (printline)
  (format t "Polygen Relation:~24,0T~A~%" (car poly-rel))
  (format t "Attributes: ~{~18,12T~A~}~%"
          (mapcar #'car (cadr poly-rel)))
  (printline)
  (if la-flg
    (progn
      (format t "~79:@<Local Attribute Mapping Information for ~A~>~%"
              (car poly-rel))
      (mapc
       #'(lambda (mapinfo)
           (format t "  Polygen Attribute: ~A~%" (car mapinfo))
           (format t "~:{   --->~20,0TDB: ~A~35,5TRel: ~A~55,5TAttribute: ~A~%~}"
                   (cadr mapinfo)))
       (cadr poly-rel))))
  (values))

;; Printing Local Schemes and Relation and Data
;; d-flg controls printing of data
;; pa-flg controls printing of polygen attributes

(defun print-l-schema (ls &optional pa-flg d-flg)
  (mapc #'(lambda (rel) (print-l-relation rel pa-flg d-flg))
        (mapcar #'cdr (reduce #'append (mapcar #'cdr ls))))
  (values))

(defun print-l-relation (r &optional pa-flg d-flg)
  (let* ((db (lr-db r))
         (rel (lr-rel r)))
    (printline)
    (if d-flg
      (pp r)
      (progn
        (format t "Local Database:~29,0T~A~%" db)
        (format t "Local Relation:~29,0T~A~%" rel)
        (if pa-flg
          (format t "Attribute~20,0TDomain~37,0TPolygen Relation~60,0TPolygen Attribute~%~
                     ~:{ ~A~21,0T~A~30,0T--->    ~A~61,0T~A~%~}"
                  (mapcar #'(lambda (pr)
                              (let* ((cname (colhead-cname pr)))
                                (multiple-value-bind
                                    (pname rname)
                                    (poly-a-name db rel cname)
                                    (list cname
                                          (colhead-domain pr)
                                          pname
                                          rname))))
                          (lr-cols r)))
```

```
          (format t "Attributes: ~{~0,12T~A~}~%"
                  (mapcar #'colhead-cname (lr-cols r))))))
    (printline)
  (values)))

;; Main Menu

;; ============================================
;; Pretty printing operations
;; ============================================

(defun pp (rel)
  (let ((cols (lr-cols rel))
        (data (lr-data rel)))
    (format t "Database: ~A ~20,0TRelation: ~A~%"
            (lr-db rel)
            (nsubseq (string (lr-rel rel))
                     0 40))
    (printline)
    ;; print header information
    (if cols
       (progn
         (format t "~:{| ~A - ~A~0,10T~}~%"
                 (mapcar #'(lambda (chead)
                             (list (colhead-cname chead)
                                   (colhead-domain chead)))
                         cols))
         (printdline)
         (if data
             ;; print the data - nested iteration forms in format
             (mapc #'(lambda (tple)
                       (format t "~{| ~A~0,20T~}~%"
                               (mapcar #'cell-data tple))
                       (printline))
                   data)
           (format t "PP: No data in relation~%"))
         (values))
      (format t "PP: No attributes in relation~%"))
  (values)))

(defun ppp (rel)
  (let ((cols (lr-cols rel))
        (data (lr-data rel)))
    (format t "Database: ~A ~20,0TRelation: ~A~%"
            (lr-db rel)
            (nsubseq (string (lr-rel rel))
                     0 40))
    (printline)
    ;; print header information
    (if cols
       (progn
         (format t "~:{| ~A - ~A~0,10T~}~%"
                 (mapcar #'(lambda (chead) (list (colhead-cname chead)
                                                 (colhead-domain chead)))
                         cols))
         ;; print the data - nested iteration forms in format
         (if data
             (progn
              (mapc #'(lambda (tple)
                        (let* ((data (mapcar #'cell-data tple))
                               (stags (mapcar #'cell-srctags tple))
                               (nostgs
                                (apply #'max (mapcar #'list-length stags)))
                               (itags (mapcar #'cell-inttags tple))
```

```
                            (noitgs
                             (apply #'max (mapcar #'list-length itags))))
                        (printdline)
                        (format t "~{| ~A~0,20T~}~%" data)
                        (mapc #'(lambda (lst cnt)
                                  (printline)
                                  (dotimes (no cnt)
                                           (format t "~{| ~A~0,20T~}~%"
                                                   (mapcar #'(lambda (tags)
                                                               (nth no tags))
                                                           lst))))
                              (list stags itags)
                              (list nostgs noitgs))))
                  data)
              (printdline))
            (format t "PPP - No data in relation~%"))
          (values))
      (format t "PPP - No attributes in relation~%"))
  (values)))

;; VPRINT - printing trace info - checks *verbose-mode*
;; if strlst is a list of strings then chops and prints
;; each line in strlst else does it once only

(defun vprint (&rest strlst)
  (if *verbose-mode*
      (mapc #'(lambda (line)
                (let ((l (length line)))
                  (if (> l 79)
                    (format t "~A~%"
                            (subseq line 0 78))
                    (format t "~A" line))))
            strlst)))

;; TPRINT

(defun tprint (str)
  (let ((s (make-string (length str) :initial-element '#\-)))
    (format t "~A~%~A~%~A~%" s str s))
  (values))

;; TTPRINT

(defun ttprint (str)
  (let ((s (make-string (length str) :initial-element '#\=)))
    (format t "~A~%~A~%~A~%" s str s))
  (values))

;; EPRINT

(defun eprint (str)
  (format t "~%System P ERROR: ~A~%" str)
  (throw 'sysperror nil))

;; printline

(defun printline ()
  (format t "~A~%"
          (make-string 79 :initial-element '#\-)))

(defun printdline ()
  (format t "~A~%"
          (make-string 79 :initial-element '#\=)))
```

A.7 QTRANS.LISP

```
;;; Syntax Analyzer

(defmacro qprint ()
  `(if *verbose-mode*
     (progn
      (format t "~%Pass One:~%~A" pass-one)
      (format t "~%Pass Two:~%~A~%" pass-two))))

;; gen-pom makes a single line of pom for the algebra that it is inputted,
;; if there are any nested expressions, they are moved either into the lhr
;; location or the rhr depending on the query.  The output is a pom-line
;; that parses only the top level algebra expression.
;; NOTES: only handles the following operations:
;;         restriction selection join projection
;; more operations can be added as deemed necessary

(defun gen-pom (rel-alg &optional (pr 'finis))
  (let* ((condition (cadr rel-alg))
         (lhr (car rel-alg))
         (op (cadr condition))
         (lha (car condition))
         (rha (caddr condition))
         (rhr (caddr rel-alg)))
    (if (= (list-length rel-alg) 2)
      (if (member (cadr condition) *ops*)
        (if (or (stringp rha) (numberp rha))
          (make-pom-line :pr pr :op 'selection :lhr lhr
                         :lha (car condition) :theta (cadr condition)
                         :rha (caddr condition) :rhr '())
          (make-pom-line :pr pr :op 'restriction :lhr lhr
                         :lha (car condition) :theta (cadr condition)
                         :rha (caddr condition) :rhr '()))
        (make-pom-line :pr pr :op 'projection :lhr lhr
                         :lha condition :theta '() :rha '() :rhr '()))
      (make-pom-line :pr pr :op 'join :lhr lhr
                     :lha (car condition) :theta (cadr condition)
                     :rha (caddr condition) :rhr (caddr rel-alg)))))

;; parse-alg takes a general relation albegra query and recursively parses
;; it to its lowest level, i.e. to the base polygen relation level, and
;; returns a list of pom entries in the necessary order for execution.

(defun parse-alg (alg &optional (pr 'foo))
  (let* ((base-line (gen-pom alg pr))
         lhr-line rhr-line
         (lhr (poml-lhr base-line))
         (rhr (poml-rhr base-line)))
    (if (consp lhr)
      (let ((tempr (new-pr)))
        (setf lhr-line (parse-alg lhr tempr))
        (setf (poml-lhr base-line) tempr)))
    (if (consp rhr)
      (let ((tempr (new-pr)))
        (setf rhr-line (parse-alg rhr tempr))
        (setf (poml-rhr base-line) tempr)))
    (append lhr-line rhr-line (list base-line))))

;; Polygen Operation Interpreter

;;; pass-one takes a list of POM-lines and outputs the half-processed
;;; IOM table (as a list of IOM-lines)
```

```
(defun parse-pom (POM)
  (let* ((r-pairs (mapcar #'(lambda (pomline)
                              (let ((pr (poml-pr pomline)))
                                (list pr pr)))
                          POM))
         ;; r-pairs tracks potential changes to the PR values for the case
         ;; where r(#) has to be changed if there is a merge operation performed
         ;; on a previous r(#) and the
         (pass-one (flatten
          (mapcar #'(lambda (line)
                      (let* ((oldpr (poml-pr line))
                             (lhr (poml-lhr line))
                             (lha (poml-lha line))
                             (schme (assoc lhr *p-schema*))
                             (attinschme (assoc lha (cadr schme)))
                             (mcol (cadr (find-if #'(lambda (x)
                                                      (> (list-length (cadr x)) 1))
                                                  (cadr schme))))
                             (lattlst (cadr attinschme))
                             mrgeprs)
                        (if (and schme attinschme)
                          ;; if the lhr is a polygen relation and the attribute
                          ;; is in the attributes list ---
                          (if mcol
                            ;; if there is more than one local attribute that the
                            ;; polygen attribute corresponds to, then we need to
                            ;; retrieve the local relations and perform joins as
                            ;; necessary.
                            (let ((retlines
                                   (mapcar #'(lambda (triplet)
                                               (let ((retpr (new-pr)))
                                                 (setf mrgeprs (append mrgeprs
                                                                       (list retpr)))
                                                 (make-iom-line :pr retpr
                                                                :op 'retrieve
                                                                :lhr (cadr triplet)
                                                                :el (car triplet))))
                                           mcol)))
                              (append retlines
                                      (let ((mergepr (new-pr)))
                                        (list (make-iom-line :pr mergepr
                                                             :op 'merge
                                                             :lhr mrgeprs
                                                             :el 'pqp)
                                              (let ((finpr (new-pr)))
                                                (rplacd (assoc oldpr r-pairs) (list finpr))
                                                ;; changes references to oldpr to the
                                                ;; post-merge result
                                                (make-iom-line :pr finpr
                                                               :op (poml-op line)
                                                               :lhr mergepr
                                                               :lha lha
                                                               :theta (poml-theta line)
                                                               :rha (poml-rha line)
                                                               :rhr (poml-rhr line)
                                                               :el 'pqp))))))
                            ;; case where the polygen schema maps directly to a
                            ;; single local attribute
                            (make-iom-line :pr oldpr
                                           :op (poml-op line)
                                           :lhr (cadar lattlst)
                                           :lha (caddar lattlst)
                                           :theta (poml-theta line)
                                           :rha (poml-rha line)
```

```
                                    :rhr (poml-rhr line)
                                    :el (caar lattlst)))
                    ;; case where the lhr is an intermediate, i.e. an
                    ;; result from a operation that left the result in the PQP
                    (make-iom-line :pr oldpr
                                   :op (poml-op line)
                                   :lhr (cadr (assoc lhr r-pairs))
                                   :lha lha
                                   :theta (poml-theta line)
                                   :rha (poml-rha line)
                                   :rhr (poml-rhr line)
                                   :el 'pqp))))
          POM)))
(pass-two (flatten
 (mapcar #'(lambda (line)
             (let* ((oldpr (ioml-pr line))
                    (lhr (ioml-lhr line))
                    (lha (ioml-lha line))
                    (rhr (ioml-rhr line))
                    (rha (ioml-rha line))
                    (el (ioml-el line))
                    (schme (assoc rhr *p-schema*))
                    (attinschme (assoc rha (cadr schme)))
                    (mcol (cadr (find-if #'(lambda (x)
                                             (> (list-length (cadr x)) 1))
                                         (cadr schme))))
                    (lattlst (cadr attinschme))
                    mrgeprs)
               (if (and schme attinschme)
                 ;; if the rhr is a polygen relation and the attribute
                 ;; is in the attributes list ---
                 (if mcol
                   ;; if there is more than one local attribute that the
                   ;; polygen attribute corresponds to, then we need to
                   ;; retrieve the local relations and perform joins as
                   ;; necessary.
                   (let ((retlines
                          (mapcar #'(lambda (triplet)
                                      (let ((retpr (new-pr)))
                                        (setf mrgeprs (append mrgeprs
                                                              (list retpr)))
                                        (make-iom-line :pr retpr
                                                       :op 'retrieve
                                                       :lhr (cadr triplet)
                                                       :el (car triplet))))
                                  mcol)))
                     (append retlines
                             (let ((mergepr (new-pr))
                                   (intpr (new-pr))
                                   (finpr (new-pr)))
                               ;(print mergepr)
                               ;(print intpr)
                               ;(print finpr)
                               (rplacd (assoc oldpr r-pairs) (list finpr))
                               (list (make-iom-line :pr mergepr
                                                    :op 'merge
                                                    :lhr mrgeprs
                                                    :el 'pqp)
                                     ;; changes references to oldpr to the
                                     ;; post-merge result
                                     (if (equal el 'pqp)
                                       (make-iom-line :pr finpr
                                                      :op (ioml-op line)
                                                      :lhr lhr
```

```
                                                    :lha lha
                                                    :theta (ioml-theta line)
                                                    :rha rha
                                                    :rhr mergepr
                                                    :el 'pqp)
                                          (list (make-iom-line :pr intpr
                                                               :op 'retrieve
                                                               :lhr lhr
                                                               :el el)
                                                (make-iom-line :pr finpr
                                                               :op (ioml-op line)
                                                               :lhr intpr
                                                               :lha (poly-a-name el lhr
lha)
                                                               :theta (ioml-theta line)
                                                               :rha rha
                                                               :rhr mergepr
                                                               :el 'pqp)))))))
                                  ;; case where the polygen schema maps directly to a
                                  ;; single local attribute
                                  (let ((ltmppr (new-pr))
                                        (rtmppr (new-pr)))
                                    (if (equal el 'pqp)
                                      (list (make-iom-line :pr rtmppr
                                                           :op 'retrieve
                                                           :lhr (cadar lattlst)
                                                           :el (caar lattlst))
                                            (make-iom-line :pr oldpr
                                                           :op (ioml-op line)
                                                           :lhr (cadr (assoc lhr r-pairs))
                                                           :lha lha
                                                           :theta (ioml-theta line)
                                                           :rha rha
                                                           :rhr rtmppr
                                                           :el 'pqp))
                                      (list (make-iom-line :pr ltmppr
                                                           :op 'retrieve
                                                           :lhr lhr
                                                           :el el)
                                            (make-iom-line :pr rtmppr
                                                           :op 'retrieve
                                                           :lhr (cadar lattlst)
                                                           :el (caar lattlst))
                                            (make-iom-line :pr oldpr
                                                           :op (ioml-op line)
                                                           :lhr ltmppr
                                                           :lha (poly-a-name el lhr lha)
                                                           :theta (ioml-theta line)
                                                           :rha rha
                                                           :rhr rtmppr
                                                           :el 'pqp)))))
                                ;; case where the rhr is an intermediate, i.e. an
                                ;; result from a operation that left the result in the PQP
                                (make-iom-line :pr oldpr
                                               :op (ioml-op line)
                                               ;; list check allows special forms in lhr i.e. merge
                                               ;; to be passed through
                                               :lhr (if (intermediate-p lhr)
                                                        (cadr (assoc lhr r-pairs))
                                                        lhr)
                                               :lha lha
                                               :theta (ioml-theta line)
                                               :rhr (cadr (assoc rhr r-pairs))
                                               :rha rha
```

```
                                        :el el))))
                pass-one))))
    (qprint)
    pass-two))

(defun alg-iom (algexp)
  (parse-pom (parse-alg algexp)))
```

A.8 SIM.LISP

```
;; ============================================
;; Relation Constructors
;; ============================================

(defun mrr (rws cls &optional
                (db (format nil "~A~A" 'db- (random *dbmax*)))
                (rel (string (gentemp "rel-"))))
  (let (tempcols tempdata)
    (dotimes (i cls)
      (push (make-colhead :cname (random-colname i)
                          :domain (random-domain))
            tempcols))
    (setf tempcols (reverse tempcols))
    (dotimes (j rws)
      (push (mapcar #'random-data
                    (mapcar #'colhead-domain tempcols))
            tempdata))
    (m-l-rel db rel tempcols tempdata)))

;; ============================================
;; Internal Procedures
;; ============================================

(defun random-domain ()
  (nth (random (list-length *domains*)) *domains*))

(defun random-colname (cls)
  (format nil "col-~A" cls))

(defun random-data (dom)
  (if (> (random 1.0) *nil-prob*)
    (cond ((equal dom 'int) (random 20))
          ((equal dom 'char) (string (code-char (+ 65 (random 26)))))
          (t (eprint " - illegal domain - random-data")))
    nil))
```

A.9 STRUCTS.LISP

```
;; ==========================================
;; Data Structure Definitions and Operations
;; ==========================================

;;; RELATIONS

(defstruct (l-relation (:conc-name lr-))
  db
  rel
  (cols () :type list)
  (data () :type list))

(defun rel-rows (rel)
  (list-length (lr-data rel)))

(defun rel-cols (rel)
  (list-length (lr-cols rel)))

;; m-l-rel makes creating relations easier
;; The nested loop below tags each data as it is passed in
;; so that all data is tagged at the end of this procedure
;; NOTE: collst can be a list of colheads of a list of pairs where
;; the car is a cname and the cadr is the domain

(defun m-l-rel (db rel collst datalst)
  (let ((norows (list-length datalst))
        (nocols (list-length collst)))
    (make-l-relation
     :db db
     :rel rel
     :cols (mapcar #'(lambda (x)
                       (if (colhead-p x)
                           (make-colhead :cname (colhead-cname x)
                                         :domain (colhead-domain x))
                           (make-colhead :cname (car x)
                                         :domain (cadr x))))
                   collst)
     :data (do* ((rnum 0 (1+ rnum))
                 (rslt '() rslt))
                ((= rnum norows) rslt)
              (push (do* ((cnum 0 (1+ cnum))
                          (tple '() tple))
                         ((= cnum nocols) (reverse tple))
                         (push (make-cell
                                :data (nth cnum (nth rnum datalst))
                                :srctags (list (make-srctag
                                                :db db
                                                :rel rel
                                                :row rnum
                                                :col cnum))
                                :inttags '())
                               tple))
                    rslt)))))

;; Recursive full copying copy-l-relation - all cells are copied
;; not just the pointers

(defun copy-l-relation (rel)
  (make-l-relation :db (lr-db rel)
                   :rel (lr-rel rel)
                   :cols (copy-cols (lr-cols rel))
                   :data (copy-data (lr-data rel))))
```

```
;; PRINT-LR prints a relation using *verbose-mode* to determine
;; whether or not to print tags

(defun print-lr (rel)
  (if *ptags*
      (ppp rel)
    (pp rel)))


;; GET-LR retrieves the relation itself based on a database
;; and a relation name

(defun get-lr (db rel)
  (format t "***LQP_CALL***:Getting Relation from Local-Schema~%")
  (cdr (assoc rel (cdr (assoc db *l-schema*)))))

;; Recursive copy of data matrix only

(defun copy-data (datmat)
  (mapcar #'(lambda (tple)
              (mapcar #'copy-cell tple))
          datmat))

;;; CELL - cells

(defstruct cell
  data
  (srctags () :type list)
  (inttags () :type list))

;; Copies cells all the way down - i.e. tags are copied

(defun copy-cell (cl)
  (make-cell :data (cell-data cl)
             :srctags (mapcar #'copy-srctag (cell-srctags cl))
             :inttags (copy-tree (cell-inttags cl))))

;;; SRCTAG - source tag

(defstruct (srctag (:type vector))
  db rel row col)

;;; COLHEAD - column header

(defstruct colhead
  cname domain)

;; Copies cols of relations such that all colheads are copied

(defun copy-cols (chead)
  (mapcar #'copy-colhead chead))

;;; POM-LINE - Polygen Operations Matrix

(defstruct (pom-line (:conc-name poml-))
  pr op lhr lha theta rha rhr)

;;; IOM-LINE - Intermediate Operations Matrix

(defstruct (iom-line (:conc-name ioml-)
                     (:include pom-line))
  el)
```

A.10 USER.LISP

```
;; ==========================================
;; Constants and Variables
;; ==========================================

;;; *OPS* - Operators for thetas in relational algebra expression

(defconstant *ops* '(= != >= > < <=))

;;; *DOMAINS* - list of valid domains

(defconstant *domains* '(int char))

;;; *DBMAX* - max number for random dbname genereation

(defconstant *dbmax* 3)

;;; *NIL-PROB* - probability of random nil being generated

(defvar *nil-prob* 0.1)

;;; *PTAGS* - flag to print tags in pretty print or not

(defvar *ptags* nil)

;;; *VERBOSE-MODE* - flag to print messages as operations are done

(defvar *verbose-mode* t)

;;; *USER* - present user of system P

(defvar *user* "YEUK")

;;; *USERS* - users that are recognized by system

(defvar *users*
  '(("RICH" (pd))
    ("STU" ())
    ("YEUK" ())))

;;; *QUERY* - in memory query

(defvar *query* nil)

;;; Menues

(defstruct menu
  ;; header - string on top of menu
  header
  ;; list of pairs where car is string and
  ;; cadr is the procedure to execute
  choices
  ;; help string
  help)

(defun sysp ()
  (reset-p-schema)
  (catch 'exit (exec-menu top-menu)))

(defun exec-menu (menu)
  (let* (in
         (choices (menu-choices menu))
         (nochoices (+ 1 (length choices))))
```

```
    (catch 'l-exit
      (loop
        (format t "~8%~79:@<~A~>~4%" (menu-header menu))
        (format t "~:{~15T~D.   ~A~2%~}"
                (mapcar #'(lambda (pr n)
                            (list n (eval (car pr))))
                        choices
                        (range 1 nochoices)))
        (format t "~%~15TChoice: ")
        (setf in (read))
        (cond ((not (numberp in))
               (format t "~%~A~%" (menu-help menu)))
              ((or (>= in nochoices)
                   (< in 1))
               (format t "~4%~79:@<Not a legal selection - Try again~%~>")
               (sleep 3))
              (t
               (eval (cadr (nth (- in 1) choices)))))))))

(setf top-menu
      (make-menu
       :header "System P Top Menu"
       :choices '(("Query Editor" (exec-menu q-editor))
                  ("Schema Displays" (exec-menu schema-disp))
                  ("System Configuration" (exec-menu sysp-config))
                  ("Quit" (quit-menu)))
       :help "this is helpful"))

(defun get-rel-spec-user ()
  (let (db rel)
    (format t "~%~15TLocal Database: ")
    (setf db (read))
    (format t "~%~15TLocal Relation: ")
    (setf rel (read))
    (get-lr db rel)))

(defun get-filename-user (suffix)
  (format t "~%~15TFilename: ")
  (concatenate 'string (read-line) suffix))

(defun get-query-filename-user ()
  (get-filename-user ".qry"))

(defun quit-menu ()
  (throw 'l-exit nil))

(setf schema-disp
      (make-menu
       :header "Schema Displays"
       :choices '(("Polygen Schema"
                   (print-p-schema *p-schema*))
                  ("Polygen Relation"
                   (progn
                     (format t "~%~15TRelation Name: ")
                     (setf ans (read))
                     (let ((rans (assoc ans *p-schema*)))
                       (if rans
                         (print-p-relation rans t)
                         (format t "~%~15A no found." ans)))))
                  ("Local Schema"
                   (print-l-schema *l-schema*))
                  ("Local Relation"
                   (let ((rans (get-rel-spec-user)))
                     (if rans
```

```
                    (print-l-relation rans t)
                    (format t "~%~15TNot found."))))
                ("Local Relation Data"
                 (let ((rans (get-rel-spec-user)))
                   (if rans
                     (print-l-relation rans t t)
                     (format t "~%~15TNot found."))))
                ("Quit to Previous Menu"
                 (quit-menu)))
       :help "Schema disp help"))

(setf sysp-config
      (make-menu
       :header "System Configuration"
       :choices '(((if *ptags*
                     "Turn Data Tag Display Off"
                     "Turn Data Tag Display On")
                   (setf *ptags* (not *ptags*)))
                  ((if *verbose-mode*
                     "Turn Verbose Mode Off"
                     "Turn Verbose Mode On")
                   (setf *verbose-mode*
                         (not *verbose-mode*)))
                  ((format nil "Change User - (present: ~A)" *user*)
                   (progn
                     (format t "~%~15TUser Name: ")
                     (setf *user* (read-line))
                     (new-user-environment *user*)))
                  ("Quit to Previous Menu"
                   (quit-menu)))
       :help "System P help"))

(setf q-editor
      (make-menu
       :header "Query Editor"
       :choices '(((format nil "Execute Query:~2%~17T~S"
                           *query*)
                   (catch 'sysperror
                     (execute-query *query*)))
                  ((if *ptags*
                     "Toggle Data Tag Display [ON]"
                     "Toggle Data Tag Display [OFF]")
                   (setf *ptags* (not *ptags*)))
                  ("Enter New Query"
                   (progn
                     (format t "~%~15TNew Query:~%~16T")
                     (setf *query* (read))))
                  ("Save Query to File"
                   (let ((fname (get-query-filename-user)))
                     (save-query *query* fname)))
                  ("Load Query from File (i.e. sampql)"
                   (let ((fname (get-query-filename-user)))
                     (load-query *query* fname)))
                  ("Quit to Previous Menu"
                   (quit-menu)))
       :help "Q-editor help"))

(defun user-filter (bad-dbs in-schema)
  (let ((temp-p-schema in-schema))
    (dolist (db bad-dbs temp-p-schema)
      (setf temp-p-schema
            (mapcar
             #'(lambda (poly-rel)
                 (list (car poly-rel)
```

```
                    (mapcar
                     #'(lambda (att)
                         (list (car att)
                               (remove db
                                       (cadr att)
                                       :key #'car)))
                     (cadr poly-rel))))
            temp-p-schema)))))

(defun new-user-environment (user)
  (let ((specs (assoc user *users* :test #'equal)))
    (if specs
      (login-user user)
      (format t "~%~15TUnknown User...  using full system~%"))
    (setf *bad-dbs* (cadr specs))
    (reset-p-schema)))

(defun login-user (uname)
  (format t "~%~15TLogging in ~A ...  ~
             setting up custom polygen schema...~%"
          uname))

(defun reset-p-schema ()
  (setf *p-schema* (user-filter *bad-dbs* *base-p-schema*)))

(defun save-query (qry fname)
  (let ((outst
         (open fname :direction :output
               :if-does-not-exist :create
               :if-exists :supersede)))
    (format outst "~S" qry)
    (close outst)))

(defmacro load-query (qry fname)
  `(let ((inst (open ,fname)))
     (setf ,qry
           (read-from-string (read-line inst)))
     (close inst)))
```

APPENDIX B: BNF'S FOR SYSTEM P STRUCTURES

The BNF's in this chapter follow standard BNF listing specifications and use the additional following base definitions:

<??_name> denotes a string that is a name of a ??. <??_name> symbols are also terminating symbols.

<datum> represents a single unit of data.

<??_list> denotes a list of elements of type ?? of length greater than 0. Each list is also surrounded by a set of parentheses.

B.1 POLYGEN SCHEMA

<Polygen_Schema> ::= <Polygen_Relation_Descr_list>

<Polygen_Relation_Descr> ::= (<Polygen_Relation_name> <Polygen_Attribute_Descr_list>)

<Polygen_Attribute_Descr> ::= (<Polygen_Attribute_name> <Local_Attribute_Descr_list>)

<Local_Attribute_Descr> ::= (<database_name> <local_relation_name> <local_attribute_name>)

B.2 POLYGEN RELATION

<Polygen_Relation> ::= (<Header> <Polygen_Data>)

<Header> ::= <Column_Head_list>

<Column_Head> ::= (<Polygen_Attribute_name> <Data_Type_name>)

<Polygen_Data> ::= <Data_Tuple_list>

<Data_Tuple> ::= <Data_Cell_list>

<Data_Cell> ::= (<datum> <Originating_Source_Tag_list> <Intermediate_Source_Tag_list>)

B.3 TAGS

<Originating_Source_Tag> ::= (<database_name> <relation_name>)

<Intermediate_Source_Tag> ::= (<database_name>)

B.4 LOCAL SCHEMA

<Local_Schema> ::= <Local_Relation_list>

<Local_Relation> ::= <Local_Column_list>

<Local_Column> ::= (<Local_Attribute_name> <Data_Type_name>)

B.5 POLYGEN OPERATIONS MATRIX

<Polygen_Operations_Matrix> ::= <POM_Operation_list>

<POM_Operation> ::= (<PR> <POM-OP> <POM-LHR> <POM-LHA> <THETA> <POM-RHA>
<POM-RHR>)

<PR> ::= <temporary_relation_name> | FOO

<POM-OP> ::= SELECT | RESTRICT | JOIN | PROJECTION

<POM-LHR> ::= <temporary_relation_name> | <Polygen_Relation_name>

<POM-LHA> ::= <Polygen_Attribute_name> | <Polygen_Attribute_name_list> | NIL

<THETA> ::= > | < | = | NIL

<POM-RHR> ::= <temporary_relation_name> | <Polygen_Relation_name> | NIL

<POM-RHA> ::= <Polygen_Attribute_name> | NIL

B.6 INTERMEDIATE OPERATIONS MATRIX

<Intermediate_Operations_Matrix> ::= <IOM_Operation_list>

<IOM_Operation> ::= (<IOM-PR> <IOM-OP> <IOM-LHR> <IOM-LHA> <THETA> <IOM-RHA> <IOM-RHR> <EL>)

<IOM-OP> ::= <POM-OP> | MERGE | RETRIEVE

<IOM-LHR> ::= <POM-LHR> | <Local_Relation_name> | <PR_list>

<IOM-LHA> ::= <POM-LHA> | <Local_Attribute_name>

<IOM-RHR> ::= <POM-LHR> | <Local_Relation_name>

<IOM-RHA> ::= <IOM-LHA>

<EL> ::= <database_name> | PQP

B.7 POLYGEN ALGEBRAIC EXPRESSION

<Polygen_Algebraic_Expression> ::= <Selection_Expr> | <Restriction_Expr> | <Projection_Expr> | <Join_Expr>

<Selection_Expr> ::= (<Relation_Expr> <Selection_Conditional_Expr>)

<Restriction_Expr> ::= (<Relation_Expr> <Restriction_Conditional_Expr>)

<Projection_Expr> ::= (<Relation_Expr> <Polygen_Attribute_name_list>)

<Join_Expr> ::= (<Relation_Expr> <Selection_Conditional_Expr> <Relation_Expr>)

<Relation_Expr> ::= <Polygen_Algebraic_Expression> | <Polygen_Relation_name>

<Selection_Conditional_Expr> ::= (<Polygen_Attribute_name> <THETA> <Polygen_Attribute_name>)

<Restriction_Conditional_Expr> ::= (<Polygen_Attribute_name> <THETA> <datum>)