A RELATIONAL QUERY LANGUAGE INTERFACE TO A HIERARCHICAL DATABASE MANAGEMENT SYSTEM

Chin-Wan Chung Kenneth E. McCloskey

Computer Science Department
General Motors Research Laboratories
Warren, Michigan 48090-9057

ABSTRACT

This paper presents an efficient Structured Query Language (SQL) interface to an IMS hierarchical database management system (DBMS). This interface is a part of DATAPLEX, a heterogeneous distributed database management system, which will provide location-transparent access to diverse databases using SQL in engineering and manufacturing environment. The initial target DBMS's to be interfaced are IMS, DB2, and INGRES. Among the three DBMS's, IMS is the hardest to provide an SQL interface. An SQL interface to IMS, called SQL/IMS, was developed. A method which translates IMS data definitions to equivalent relational data definitions is developed. A procedure which decomposes an SQL query into a set of simple IMS processable queries and recomposes partial results is derived. SQL/IMS was tested using a small test database and a large production database. The performance of SQL/IMS is compared with that of another system used to access IMS data.

1. INTRODUCTION

In a computer-aided engineering and manufacturing environment, it is critical to supply necessary data to various engineering and manufacturing processes.

The data to these processes comes from diverse sources. For example, a product assembly plant generally utilizes different DBMS's for the plant controller and the area controller due to the difference of application requirements. The product scheduling database is in the plant controller which typically uses IMS [7], whereas the area controller uses a relational DBMS such as INGRES to manage the product tracking database. In this case, INGRES needs to get data on parts, options, and inventory from IMS. In addition, the plant controller's IMS is connected to the outside IMS managing the product order database which is in turn connected to dealers.

As the database technology evolves, the migration to a new database environment is required where applicable. The bill-of-material and engineering release system has been used for a while in the manufacturing industry, and naturally it has been developed using a non-relational DBMS. To rebuild the system using a relational DBMS, both the new and the old systems must coexist until the migration process is completed because of a vast amount of applications developed on the old system.

Consequently, an interface among different DBMS's is essential. General Motors Research Laboratories has initiated the DATAPLEX project to provide an effective interface among diverse DBMS's. DATAPLEX is a heterogeneous distributed database management system which will provide a common view of diverse databases and a standard query language for Corporate-wide data sharing. The architecture of DATAPLEX is described in [2]. In this architecture, a relational model is used to provide a common view of data and SQL is selected as a standard query language. A prototype DATAPLEX [3] which interfaces an IMS hierarchical DBMS and an INGRES relational DBMS showed the feasibility of the DATAPLEX architecture.

An interface to IMS in DATAPLEX environment is important because IMS has been the most heavily used mainframe DBMS in large corporations. It is estimated that about 90% of DBMS-resident data in General Motors Corporation is stored under IMS. In addition, the techniques to interface IMS can be applied to interface other non-relational DBMS's such as network DBMS's. IMS is based on a hierarchical data model and uses DL/1 as a data manipulation language. While SQL is a high-level query language, DL/1 is a set of subroutine calls that must be embedded in a host language such as PL/1 or COBOL. Since DATAPLEX uses the relational model and SQL, relational DBMS's are relatively easy to interface. On the other hand, an interface to a non-relational DBMS poses a difficult translation problem.

Due to the need for a rapid prototyping, the prototype DATAPLEX used DXT [6] as an SQL to DL/1 translator. DXT is an IBM product which translates a subset of SQL to DL/1. The experience of DXT in the prototype DATAPLEX led to a conclusion that the performance of DXT is not adequate and the subset of SQL supported by DXT is too restrictive for a production DATAPLEX. Therefore, we decided to develop an SQL interface to IMS internally for a full-function DATAPLEX. This interface is called SQL/IMS.

The retrieval part of SQL/IMS has been completed and a driver module was written to run the implemented SQL/IMS as a standalone system under MVS/XA or MVS/ESA. SQL/IMS can execute SQL queries interactively from a TSO session or as a batch job.

CH2806-8/89/0000/0105/$01.00 © 1989 IEEE

Benchmarks were prepared to test various features and the performance of SQL/IMS. The comparison of performance of SQL/IMS and DXT is conducted using a small test database and a large production database.

In Section 2, the architecture of SQL/IMS is presented along with basic design considerations. The translation of IMS data definitions to equivalent relational data definitions is discussed in Section 3. Section 4 explains the translation of an SQL query to a DL/1 program and the execution of the program. Section 5 describes a testbed and the result of benchmark executions.

2. ARCHITECTURE

The major issue in developing an SQL interface to IMS is the incompatibility between SQL and DL/1. SQL is a set (set of records) oriented language suitable for manipulating relationally structured data, whereas DL/1 is a record oriented language for hierarchically structured data.

SQL has more features than DL/1 and consequently there is no DL/1 feature to which certain SQL features can be translated. These SQL features must be implemented without using DL/1 features. However, some of SQL features are seldom used in applications. In fact, most frequently used SQL features are the ones corresponding to relational operations Selection, Projection, and Natural-Join. All these three operations can be translated to DL/1 features, and especially Natural-Join can be processed very efficiently using DL/1 because, in many cases, the relationship between two entities is implemented by pointers in IMS databases.

Based on the above facts, we have defined a subset of SQL to be supported by SQL/IMS with the following guiding principles:

- Include all SQL features that can be translated to DL/1 features to make use of efficient IMS capabilities.
- Include SQL features that are frequently used in applications so that most of the applications can be covered by the defined SQL subset.
- Exclude SQL features that are seldom utilized to avoid the implementation of a relational DBMS and the run time overhead of a large system.

This subset of SQL is defined in Section 4. An SQL query which uses unsupported SQL features is decomposed by DATAPLEX into two queries: Q1 (Query 1) contains supported SQL features, and Q2 is the remaining part of the original query. Then, Q1 is sent to SQL/IMS which translates it to a DL/1 program and submits the program to IMS. Q2 and the result of Q1 are fed to a relational DBMS (which may be at a remote-location). The result of Q2 is the result of the original query.

Our approach to process an SQL query submitted to SQL/IMS is to decompose the query into simpler queries that can be processed by IMS and then perform remaining relational operations on the intermediate results retrieved from IMS to obtain the final result. There are two ways to translate a decomposed simple SQL query to a DL/1 program: (1) to generate DL/1 code on the fly, and (2) to generate parameter values from the SQL query and feed them to a fixed parameterized DL/1 program. We take the second approach for its simplicity and efficiency. The architecture of SQL/IMS consists of the following three modules:

Decomposer/Translator. This module reduces the SQL query into a set of translated subqueries based on the query predicates and the databases referenced. It also generates the recomposition queries required to construct a final result.

Recomposer. This module accepts as input the decomposed queries and recomposition queries. Recomposer first passes each subquery to the DL/1 Engine for processing. When all subqueries are complete, the recomposition queries are issued and a final result is created.

DL/1 Engine. This module is a fixed parameterized DL/1 program that accepts a subquery and generates a series of DL/1 path calls to retrieve the requested data. DL/1 Engine uses the mapping information attached to a subquery to identify the target database paths and format DL/1 segment search arguments. DL/1 Engine performs Selections on non-key attributes and Projections for data retrieved from IMS.

The architecture of SQL/IMS is depicted in Figure 1.

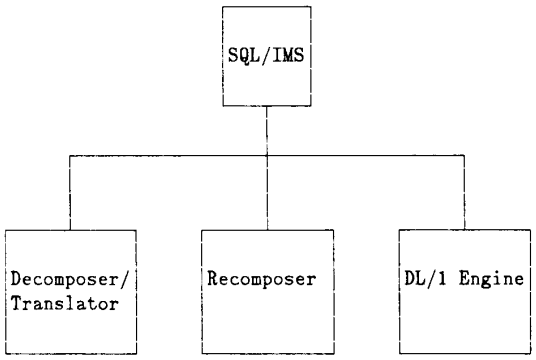

Figure 1. Architecture of SQL/IMS

3. DATA DEFINITION TRANSLATION

3.1. Hierarchy to Relation Mapping

The translation of an IMS hierarchical data definition to an equivalent relational data definition is accomplished by mapping each IMS segment type to a relation. In addition, the relation corresponding to a segment type will include the key attributes of each predecessor of the segment type in the hierarchy (the concatenated key, in IMS parlance). Relations defined on the dependents of non-key segments would not include any attributes from the non-key segments.

For example, consider the following IMS hierarchy, shown in Figure 2, which represents a database for vehicle manufacturing. Each segment type is defined at the bottom of this page.

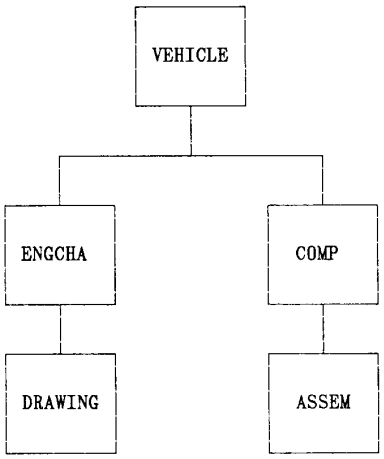

Figure 2. Sample IMS Database Definition

Using the previously described mapping strategy, this sample IMS hierarchy produces the following relations:

Relation	Attributes
VEHICLE	(VID, GROUP, MILEAGE, PRODVOL)
ENGCHA	(VID, DATE, DESIGNER, DESCRIPTION)
DRAWING	(VID, DRAWING#, CONTACT, CADSYS, STATUS)
COMP	(VID, COMP#, CATEGORY, PRODDATE)
ASSEM	(VID, COMP#, PART#, QUANTITY)

Although IMS primarily supports a hierarchical structure, network style structures are possible via IMS logical databases (a misnomer, really, as there is nothing logical about these databases which are actually physical links). Existing physical IMS hierarchies are connected using physical pointers to form new "logical" hierarchies.

This additional structure could cause a mapping problem as not all IMS structures would be hierarchies. Fortunately, there is a simple solution; mapping is always based on the Program Specification Block (PSB). Since all IMS databases described in a PSB present a consistent data structure (a hierarchy), regardless of the underlying physical data structure, this is the most reasonable point from which to map.

The mapping information is stored in two files: the Physical Definition Table (PDT) and the View Definition Table (VDT). The PDT contains all the IMS access information for each attribute. The VDT contains a definition of the relation as seen by the user. This provides three main features: attributes names to be aliased, attribute type coercion and relation subsets to be formed.

4. TRANSACTION TRANSLATION AND EXECUTION

In this section, the level of SQL to be supported is defined. The query translation process is then described followed by a discussion of query execution.

In attempting to develop a relational interface to an older, non-relational system, relational operators would be implemented over the existing single record at a time system. The implementation of a complete set of relational operators would duplicate a large portion of existing relational DBMS's. The optimal strategy is to implement the

Segment	Description	Attributes	Attribute Description
VEHICLE	One occurrence per model, it describes all the model specific data.	*VID	Vehicle ID #
		GROUP	Manufacturing Group
		MILEAGE	Estimated MPG
		PRODVOL	Production volume
ENGCHA	One occurrence per engineering change, it describes a modification to the vehicle design.	DATE	Date of change
		DESIGNER	Name of designer
		DESCRIPTION	Change description
DRAWING	One occurrence per design drawing affected by the engineering change, it provides the reference for the document.	*DRAWING#	Drawing ID #
		CONTACT	Contact person for drawing
		CADSYS	CAD system where stored
		STATUS	Current revision status
COMP	One occurrence per vehicle component, it describes a major vehicle component.	*COMP#	Component ID #
		CATEGORY	Type of component
		PRODDATE	Production date
ASSEM	One occurrence per subcomponent, each describes a portion of a component.	*PART#	Part ID #
		QUANTITY	Quantity used

Attributes that are defined as keys to IMS are preceded by an '*'.

basic relational operators that provide the most useful features of the language. Our SQL subset for retrieval is defined as follows:

SELECT	target attributes, set functions, expressions
FROM	relations
WHERE	predicates
ORDER BY	sort attributes

In the SELECT list, set functions may contain expressions, and vice versa. Predicates are connected using logical AND and OR operators and grouped using parentheses. A predicate can be an equi-join term or a selection term with comparison operators =, >, <, >=, <=, <> which may contain expressions. Nested queries are not supported, however, a nested query can be transformed to an equivalent non-nested query [9].

4.1. Query Decomposition/Translation

The process of query decomposition/translation involves breaking the query down into units simple enough to be executed by an IMS application program and converting them to an acceptable format. For the sake of brevity this process will be referred to as translation. There are eight steps of translation all queries must pass through prior to execution. Each step either removes functions that are not processable by IMS or splits the query into subqueries that reference a smaller range of data. For each act of translation imposed on a query, a complementary recomposition query is generated to be executed once the data has been retrieved from IMS. The eight steps of translation are now described.

Step 1. Syntax Check. The SQL query is scanned to verify that the query is syntactically correct. All attribute and relation references are tested for existence in the VDT.

Step 2. Remove Nonsupported Operations. Since IMS is a simple record-oriented DBMS, any functions provided in the SQL subset that perform set-oriented operations must be removed. Expressions, set functions and the DISTINCT option (if specified) are removed from the target list. Sort instructions (i.e., ORDER BY) are also removed at this time. Each of the removed operations is saved in a recomposition query that is executed after the data is retrieved.

Step 3. Conversion to Disjunctive Normal Form. The query is then transformed into a set of conjunctive ('AND' connected) subqueries. This is a prerequisite for the Steps 6 and 7 of translation where the query will be decomposed based on the range of the query. A recomposition query will be generated to union the results from each subquery.

Step 4. Remove Expressions from Predicates. Each of these new conjunctive queries then have any expressions removed from their respective search conditions (i.e., WHERE clause). These expressions are saved in a recomposition query that will be executed once the data has been retrieved.

Step 5. Conversion to Graphical Representation. Each conjunctive query is translated into a query graph to perform the following two steps of translation. Using the information stored in the PDT, each relation is graphed as a node and the search conditions are represented as arcs on the nodes. All information on the attributes and hierarchies is included in the graph.

Step 6. Decompose by Database. Each query graph is then examined for the location of the data to be retrieved. An IMS query (DL/1 call) may reference only one database. Therefore queries that reference data in multiple IMS databases are decomposed into queries that reference data in a single IMS database by cutting the connecting arcs between the relations.

Step 7. Decompose by Path. The strength of the hierarchical data structure used by IMS lies in its physical storage technique. IMS (usually) stores the occurrence of a root physically adjacent to its dependents and connects them into the hierarchical form using pointers. This structure allows very fast access to dependent segments from the root segment.

A by-product of this structure is an efficient way to perform equijoins. A relation mapped from a segment type will include the key fields of all its predecessors (including the root). Therefore, an SQL command that specifies an equijoin between relations that map to different IMS segment types, using the common root key field may be retrieved using a special type of IMS operation known as a path call. This is called a pathjoin.

A query referencing multiple segment types is tested for the presence of a pathjoin. If there are arcs which do not correspond to the pathjoin, then the query is further decomposed by cutting the connecting arcs between nodes.

Step 8. Final Graph. The query graph is now regenerated based upon the results from the last two steps of translation. All decomposed graphs are generated into independent graphs with the removed arcs sent to recomposition join queries.

At this point the subqueries are processable by the IMS interface program.

4.2. Query Execution/Recomposition

After translation, the query is passed to the query execution/recomposition module, the Recomposer. It passes the individual subqueries to the IMS interface program for execution. Once all the subqueries have executed the recomposition queries are applied against the retrieved data in the reverse order in which they were created during translation.

The IMS interface module is called the DL/1 Engine (DLE). The DLE is a standard IMS application program that uses the normal application call interface to retrieve IMS segments. The DLE search module uses the information contained in the query graph to format the DL/1 call and screen the

segments returned. For segments that meet the query qualifications, the query target attributes are formatted into the result row and returned to the Recomposer.

Once all the subqueries have been executed the results must be massaged into the final result requested by the original query. This process is called recomposition. Recomposition is comprised mainly of performing all the functions that cannot be processed within IMS. These operations are performed in the reverse order of translation and are as follows:

1) Non-pathjoin equijoins.
2) WHERE clause expression.
3) Union of all subquery results, from each conjunctive subquery.
4) DISTINCT.
5) Set Function evaluation.
6) SELECT list expression evaluation.
7) Sorting, i.e., ORDER BY.

5. TESTING

SQL/IMS is mostly in C with some assembler. SQL/IMS is installed on an IBM 4381 S91E (Model 23) which runs MVS/ESA. The IBM 4381 is a heavily loaded departmental test machine. Two IMS databases are used to test SQL/IMS. One is a small test database used to test the prototype DATAPLEX [3]. The other is a part of the production Maintenance Management Information System (MMIS) obtained from a plant automation development group. SQL queries were formulated to test various features of SQL and to retrieve diverse sizes of results from different parts of the IMS databases.

Since the test database is small, it is used to check the correctness of the results produced by SQL/IMS. The data structure diagram and the number of occurrences for each segment type of the test database are shown in Figure 3. Fourteen SQL queries were executed against the test database. SQL/IMS produced correct results for all the test queries. The equivalent relational data definition of the test database and the fourteen SQL queries can be found in Appendix A.

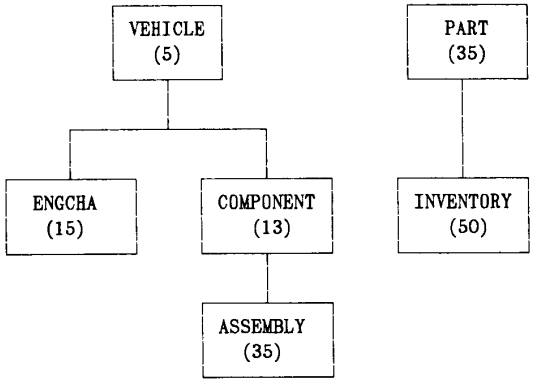

Figure 3. Structure of Small Test Database (number of occurrences)

The performance of SQL/IMS against a small IMS database is also tested using the test database. In addition, it is compared with the performance of DXT. DXT [6] is an IBM product which was used as an SQL to DL/1 translator for the prototype DATAPLEX. Table 1 shows the CPU time for SQL queries. Due to the lack of functionality, DXT couldn't process some of the queries. The corresponding entries are marked with '*'.

TABLE 1

Comparison of Performance of SQL/IMS and DXT Against a Small Test Database

Request	CPU Time (in seconds)		# Records in Result
	SQL/IMS	DXT	
1	0.414	3.576	2
2	0.335	3.502	2
3	0.512	3.877	49
4	0.418	3.770	7
5	1.358	3.804	2
6	0.582	*	1
7	0.585	*	1
8	0.574	*	1
9	0.461	*	1
10	0.896	*	5
11	0.902	*	5
12	0.878	3.608	16
13	0.343	3.626	13
14	0.338	3.628	13

* indicates the query contains syntax and/or features not supported by DXT.

The MMIS database is used to test the performance of SQL/IMS against a large production IMS database. The MMIS database contains maintenance scheduling information for plant equipment. The data structure diagram of the MMIS database is shown in Figure 4 with the number of occurrences for each segment type. The equivalent relational data definition of the MMIS database is provided in Appendix B.

Eight SQL queries are formulated to test the performance for various types of access to IMS databases. These queries are listed in Appendix B. The comparison of the CPU time for SQL/IMS and DXT is shown in Table 2. For the queries that are successfully executed by both of the systems, the two systems produced the same number of result records for each request.

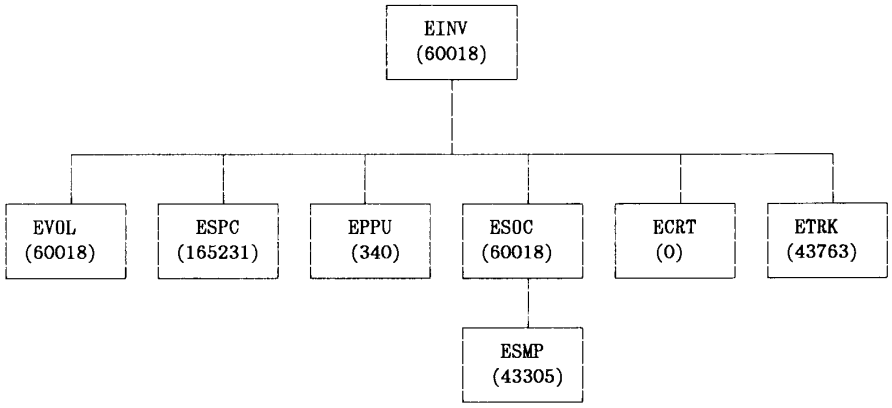

Figure 4. Structure of MMIS Database (number of occurrences)

TABLE 2

Comparison of Performance of SQL/IMS and DXT Against a Large Production Database

Request	CPU Time (in seconds)		# Records in Result
	SQL/IMS	DXT	
1	36.049	35.382	5
2	35.823	36.893	539
3	0.741	4.142	165
4	65.263	99.028	5882
5	46.666	82.687	978
6	1.584	6.087	244
7	0.802	*	91
8	59.763	*	5033

* indicates the query contains syntax and/or features not supported by DXT.

6. CONCLUSIONS

An SQL interface to IMS, called SQL/IMS, was developed. Users can submit SQL queries to SQL/IMS to retrieve IMS data interactively from a TSO terminal or as a batch job. A testbed was created on IBM 4381 running MVS/ESA. The performance of SQL/IMS was analyzed using two IMS databases: a small test database used to test the prototype DATAPLEX, and a large production database which is a part of the Maintenance Management Information System (MMIS). Fourteen test queries were executed against the small test database and eight queries against the large production database.

The performance of SQL/IMS was compared with that of DXT which is IBM's SQL to DL/1 translator. The performance and the functionality of SQL/IMS were superior to those of DXT. While the performance of SQL/IMS is satisfactory, it can be further improved because we have only optimized each step of SQL/IMS processing. By adding a simple global optimizer, the efficiency of SQL/IMS can be increased especially for unusual queries.

ACKNOWLEDGEMENT

We acknowledge the contributions of Relational Technology, Inc. personnel during numerous discussions about the design of the SQL interface to IMS. We would also like to thank Jeff Hollar and Mike McGreevy for their participation in the development of SQL/IMS.

APPENDIX A

Relational Data Definition and SQL Queries for the Test Database

(1) An equivalent relational data definition

VEHICLE (VID, CARGROUP, MILEAGE, PRODVOL)

ENGCHA (VID, CHGNUM, CHGDATE, DESIGNER, DESCRIPT)

COMPONENT (VID, CNUM, CATEGORY, PRODDATE)

ASSEMBLY (VID, CNUM, PNUM, QTY)

PART (PNUM, MTRL, PRICE)

INVENTORY (PNUM, WHNUM, QOH)

(2) SQL queries to the test database

```
1. select *
   from   vehicle
   where  mileage > 30

2. select whnum, qoh
   from   inventory
   where  pnum = 'P44'

3. select p.pnum, price, whnum, qoh
   from   part p, inventory i
   where  p.pnum = i.pnum
   and    qoh > 10000
```

```
4.  select e.vid, descript, cnum, proddate
    from   engcha e, component c
    where  chgdate > '85-12-31'
    and    e.vid = c.vid
    and    category = 'ELECTRICAL'

5.  select p.pnum, price, qty
    from   part p, assembly a
    where  p.pnum = a.pnum
    and    vid = 'V20'
    and    qty > 10

6.  select min(price), max(price)
    from   part

7.  select avg(price)
    from   part
    where  mtrl = 'ALUMINUM'

8.  select sum(qoh)
    from   inventory
    where  whnum = 'W19'

9.  select count(cnum)
    from   component
    where  proddate > '84-01-01'

10. select distinct *
    from   vehicle
    order by prodvol

11. select distinct *
    from   vehicle
    order by mileage desc

12. select pnum, price
    from   part
    where  price > 100
    or     price < 10

13. select chgnum, designer, descript
    from   engcha
    where  not designer = 'BROWN, A.G.'

14. select chgnum, designer, descript
    from   engcha
    where  designer <> 'BROWN, A.G.'
```

APPENDIX B

Relational Data Definition and SQL Queries for the MMIS Database

(1) An equivalent relational data definition.

Since relations contain a large number of attributes, only the relations and attributes referenced in queries are listed.

EINV (EINVKEY, PLANTSI1, SYSTEM, EQUIP, COMP, DESCRIPTOR, PROPERTY, PARTS_SET, ORIG_COST, BOOK_VALUE, DRAWING_SET)

EVOL (EINVKEY, PRD_DOWNYTD)

ESOC (EINVKEY, START_CDATE)

ESMP (EINVKEY, COUNTDUE, RUNDUE, DATEDUE, TASK, UNITDUE)

ETRK (EINVKEY, DESIGN_COST, BUILD_COST, REJECT_CODE, GAGE_CODE, DISP, DISP_DATE, NEW)

(2) SQL queries to the MMIS database

```
1.  select plantsi1, system, equip, comp
    from   einv
    where  parts_set <> '    '
    and    property = 'repair'

2.  select plantsi1, equip, comp, descriptor,
           property
    from   einv
    where  system = 'scrap'

3.  select disp, design_cost, build_cost,
           reject_code
    from   etrk
    where  einvkey <= 'to 0000300'

4.  select einvkey, countdue, rundue
    from   esmp
    where  datedue = 90365

5.  select i.einvkey, equip, gage_code,
           disp_date
    from   einv i, etrk t
    where  disp_date >= 880301
    and    new ='y'
    and    disp = 'ac'
    and    i.einvkey = t.einvkey

6.  select i.einvkey, start_cdate, task,
           unitdue
    from   einv i, esoc s, esmp p
    where  i.einvkey <= 'T0 0001331'
    and    i.einvkey = s.einvkey
    and    s.einvkey = p.einvkey

7.  select plantsi1, system, equip, comp,
           avg(orig_cost)
    from   einv
    where  einvkey >= 'T0 0000010'
    and    einvkey <= 'T0 0000100'

8.  select equip, orig_cost - book_value,
           prd_downytd
    from   einv i, evol v
    where  i.einvkey      = v.einvkey
    and    drawing_set >= '0015640'
    order by 2
```

REFERENCES

[1] ANSI X3H2-86-141, Database Language SQL2, November 1986.

[2] Chung, C. W., "DATAPLEX: A Heterogeneous Distributed Database Management System," to appear in Communications of the ACM.

[3] Chung, C. W., "Design and Implementation of a Heterogeneous Distributed Databases Management System," Proc. of the IEEE INFOCOM '89, April 1989, pp. 356-362.

[4] Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM, Vol. 13, June 1970, pp. 377-387.

[5] International Business Machines Corporation, C Language Manual (SC19-1128-0) for IBM 'C' for 370 (Product offering 5713-AAG), 1986.

[6] International Business Machines Corporation, Data Extract Version 2.3: General Information (GC26-4241), May 1988.

[7] International Business Machines Corporation, IMS/VS 1.3 General Information Manual (GH20-1260), March 1984.

[8] ISO/JTC 1/SC 2/WG 3N, Information Systems - Open Systems - Generic Remote Database Access Service and Protocol, September 1988.

[9] Kim, W., "On Optimizing an SQL-Like Nested Query," ACM TODS, Vol. 7, No. 3, September 1982, pp. 443-469.

[10] Landers, T. and R. L. Rosenberg, "An Overview of Multibase," Distributed Databases, North-Holland, 1982, pp. 153-184.

[11] Zaniolo, C., "Design of Relational Views Over Network Schemes," Proc. of the ACM SIGMOD Conference, May 1979, pp. 179-190.