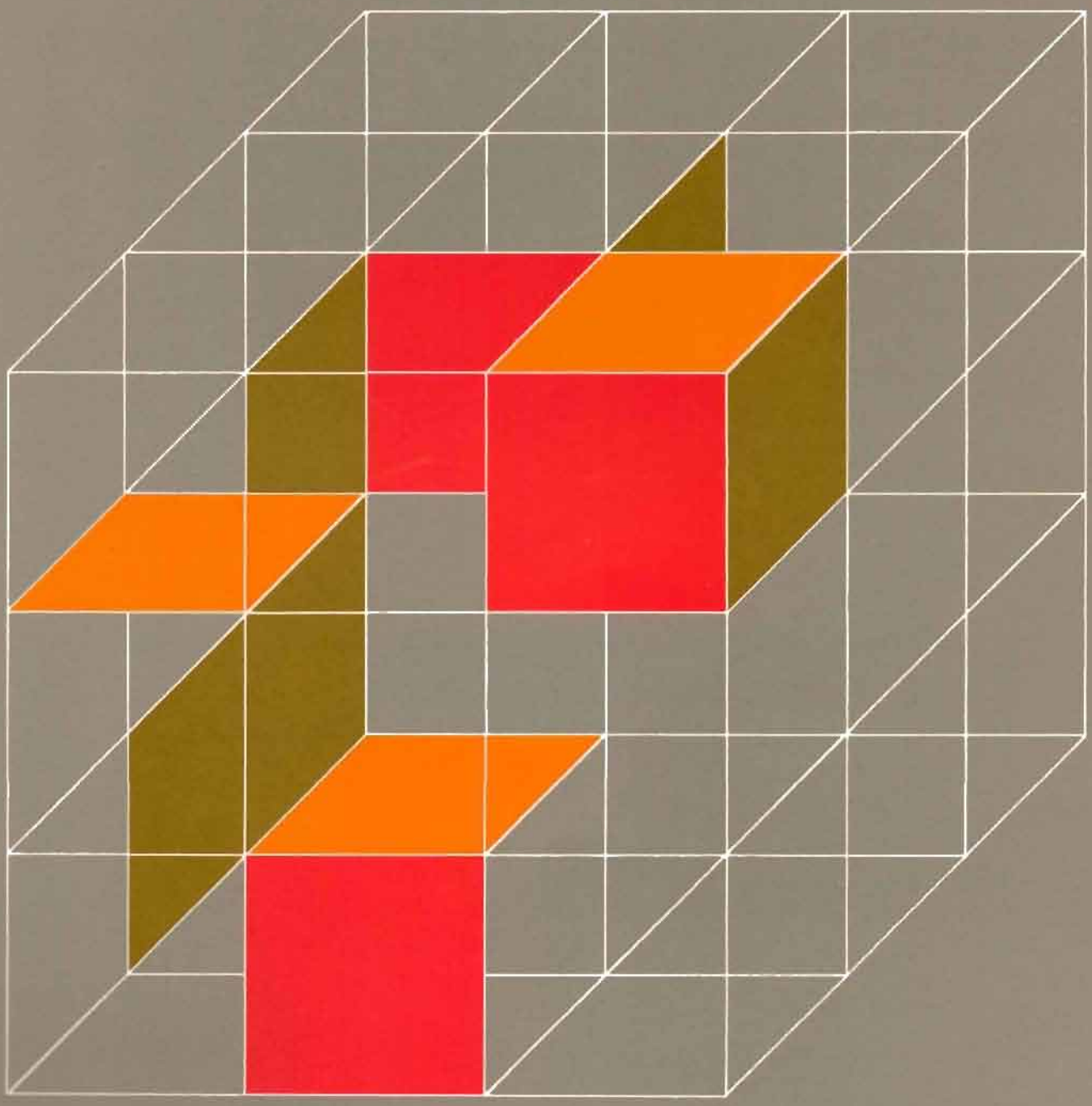
IBM

SQL/ Data System for VSE

A Relational Data System for Application Development



This document is directed to data processing professionals and their management. Its purpose is to demonstrate, through examples, both the simplicity and the power of SQL/DS.

Contents

1 Introduction

As the demands on the data processing department continue to increase, many installations are re-examining their traditional approaches to application development in order to keep pace with the rapidly growing backlog of applications.

For some installations, this has meant providing their data processing professionals with additional tools so that they can become more productive. Other installations have created a new functional organization within data processing, called an Information Center, to provide and support a set of end-user tools. This allows the person with minimal DP skills to develop his own applications with a minimum of dependence on the DP department.

In either case, new technologies in hardware and software have created an application development environment with significantly different characteristics than in the past. Installations have begun to focus on those aspects of application development from which they can achieve significant productivity gains by effectively using machine resources to augment people time—either by increasing the ongoing rate of traditional program development and associated maintenance, or by reducing the number of times that they must choose the "programming alternative."

As a result, many new applications are being developed in a heterogeneous fashion. In other words, while some portions of code are written via traditional programming, other portions are produced by "generators," or by "report writers" and "query" products. With Information Centers, some portions of applications are written and maintained by the end user, not just by the data processing department.

IBM's new Structured Query Language/Data System (SQL/DS) provides the intermediate system installation with a new application development alternative. It is aimed at improving the productivity of both the data processing professional and the end user communities through a wide range of design and implementation approaches.

SQL/DS—An Application Development Alternative for Intermediate Systems

SQL/DS—An Application Development Alternative for Intermediate Systems

A Tool for the DP Professional

SQL/DS is a new IBM program product for the Data Systems Environment; it consists of a relational data base facility, a powerful query language, and a standard programming interface to COBOL, PL/I, and Assembler. In this sense, it is a tool for the DP professional.

A Tool for the End User

In another sense, there are features that allow applications to be designed so that SQL/DS is a very practical tool for the end user. When tailored by the DP professional, or Information Center Specialist, SQL/DS is appropriate for the Information Center environment.

An Integral Data Systems Environment Product

Most importantly however, SQL/DS is an integral product of the Data Systems Environment. It is designed to work in conjunction with other IBM Data Systems Environment products such as CICS/DOS/VS and DL/I DOS/VS.

Providing a Wide Range of Design Approaches

Because SQL/DS is designed to work in conjunction with these products, the installation can use SQL/DS in many ways, depending on the requirements of the application and the capabilities of the end users. These ways are:

- Unplanned end-user query
- Stored query transactions
- Online CICS/DOS/VS transaction programs
- Batch application programs

These effectively provide a "building block" approach to design and implementation. This approach will be discussed later in this publication.

Allowing Interactive Program Development

The relational data base can be shared with an interactive system such as VSE/ICCF. COBOL, PL/I, and Assembler language programs can be compiled, edited, and tested directly from the terminal with the results received back at the terminal. Standard submission to DOS/VSE batch, use of the interactive usability aids, the procedure processor, and the online library system of ICCF are completely supported in this environment.

Offering Control and Administrative Facilities

SQL/DS is flexible in permitting either centralized or decentralized administration of data ...and most administrative tasks can be performed via terminals.

SQL/DS is normally operated in a way that many users and applications can access SQL/DS data concurrently. Data access can be controlled by a comprehensive authorization facility and a set of system catalogs. The system also includes facilities for controlling the security and integrity of its data, even in the event of abnormal termination of application programs, SQL/DS itself, or the operating system.

With Utilities for Data Extraction and Creation

SQL/DS also provides facilities for bulk-loading new data or data from existing systems into its relational data base. For example, it may be desirable to apply the advantages of SQL/DS to existing portions of DL/I applications. The interactive query language includes commands to extract data from the DL/I data base and copy it into SQL/DS.

5 The Relational Approach

The Relational Approach

Objective

An objective of the relational approach is to simplify data base design and processing for programmers and end users. This is achieved through the use of a familiar form of logical data organization—a table—and a high-level language especially designed to take advantage of data in tabular form.

User Perception of Data

Tabular View

Traditionally, data in and out of business is in tabular form; that is, in the form of either tables or reports, which have a title and columns of data. And, of course, in these tables or reports are multiple lines or rows. The relational data structure is a table and allows users to perceive their collection of reports as a collection of tables.

In the figure below, the rows of the INVENTORY table can be thought of as instances of records. The columns INVPART, PNAME, and ONHAND can be thought of as fields for these records. Note that the word "perceived" is essential; how this data is actually stored is not relevant to the relational view.

INVENTORY Table

INVPART	PNAME	ONHAND
105	GEAR	0
106	GEAR	700
124	BOLT	900
125	BOLT	1000

Path-Free Access

In addition, the tabular view of data contains no physical access paths that are visible to the user. This means that access paths, such as links, rings, chains, indexes, etc., do not have to be learned and remembered by the user for purposes of navigating through the data base. Therefore, since the user does not need to consider access paths, the formulation of requests for data is simplified.

Relational Language

Dynamic Relationships

Relationships between rows in one table and rows in another can be established dynamically using the facilities of the relational language. For example, suppose we need to determine the name of the suppliers of part 124 from the two tables shown below.[1]

SUPPLIERS Table

SUPSUPP	NAME	ADDRESS
53	ATLANTIS CO.	8 OCEAN AVE., WASHINGTON, D.C.
57	EAGLE HARDWARE	64 TRANQUILITY PLACE, APOLLO, MN
64	KNIGHT LTD.	256 ARTHUR COURT, CAMELOT, ENGLAND

QUOTATIONS Table

QUOSUPP	QUOPART	PRICE	TIME	ONORD
51	124	1.25	5	400
51	125	0.55	5	0
53	124	1.35	3	500

[1] The names used herein are fictitious; they are used solely for illustrative purposes and are not for identification of any company.

First, the supplier numbers (QUOSUPP) for part 124 need to be identified by examining the QUOTATIONS table. Next, the supplier numbers obtained from this table are compared to the supplier numbers (SUPSUPP) in the SUPPLIERS table. When they match, the system extracts NAMEs from those rows in the SUPPLIERS table.

This table-lookup process is accomplished using the operators of the relational language; there is no need to use conventional programming techniques to obtain the information needed.

The important point here is that many interrecord or intertable relationships can be established by the user spontaneously. With conventional approaches, only those relationships that were defined prior to the creation of the data base can be used this simply.

Set Processing

The relational language provides operators that process sets of records at a time, rather than single record-at-a-time processing. A single relational request can be used to selectively retrieve data from multiple rows of multiple tables for presentation to the terminal user. Similarly, a single relational request can be used to selectively update or delete multiple rows of a single table. In conventional language approaches, such operations would require multiple requests to the data manager.

Tabular Output

The result of a relational query, or request, is also presented to the user in the form of a table. By adding some commands to the original query, the user can make the query result a permanent table. In this way, new tables that are (later) needed can be defined dynamically and easily, can be generated online, and become immediately available as part of the entire collection of tables within the system.

Simple Underlying Concept

The tabular view is a primary reason why a relational system is easy to understand. Since the data structure is familiar to most users, there is greater potential for improved communications with the data processing organization when specifying information requirements.

Concise Language

The relational query language used in SQL/DS is called Structured Query Language (SQL). Its data handling capabilities, being geared to operations on sets of records, can mean a significant reduction in the number of statements the user must provide, compared to many existing languages.

Responsive to Changing Requirements

There are many application areas—particularly those involving user analysis, reporting, and planning—where the very nature of the application is constantly changing. Some typical application areas are:

- Financial
  - Budget analysis
  - Profit and loss
  - Risk assessment
- Inventory
  - Vendor performance
  - Buyer performance
- Marketing & sales
  - Tracking & analysis
- Personnel
  - Compliance
  - Skills and job tracking
- Project management
  - Checkpoint/milestone progress
  - Development and test status
- EDP auditing
  - Data verification
  - Installation configuration
- Government/education/health
  - Crime and traffic analysis
  - Admissions/recruiting/research
  - Medical data analysis

These applications typify instances where it is of primary importance to establish interrelationships within the data base and to define new tables.

Coexistent

The fact that the relational approach provides certain advantages over existing systems for specific applications does not mean that it replaces them. SQL/DS is designed to work in conjunction with several other IBM application development facilities in the Data Systems Environment.

This means that the developer is not limited to a single approach to develop an application and can determine the most appropriate development technique for each phase of the application.

Extendable to the End User

In many installations, the key to overall productivity is the ability of DP to offload the appropriate portions of the development and maintenance of an application to the end user.

The flexible design approach mentioned above allows an application to be designed with the end user's capabilities in mind. This could enable the DP professional to implement an application up to the point where the end user could create and execute his own queries, thereby expanding the application on his own and reducing his dependence on the data processing department.

Application Prototyping

All of these characteristics make SQL/DS a powerful prototyping tool. The terminal facilities of SQL/DS can be used to create prototype tables loaded with sample or production data. Online queries can easily be written to demonstrate application usage. End users can see the proposed scheme in operation *prior to* formal DP development. In this prototype approach, people time and computer time are saved while design flaws are easily corrected at an early phase.

Data Management System

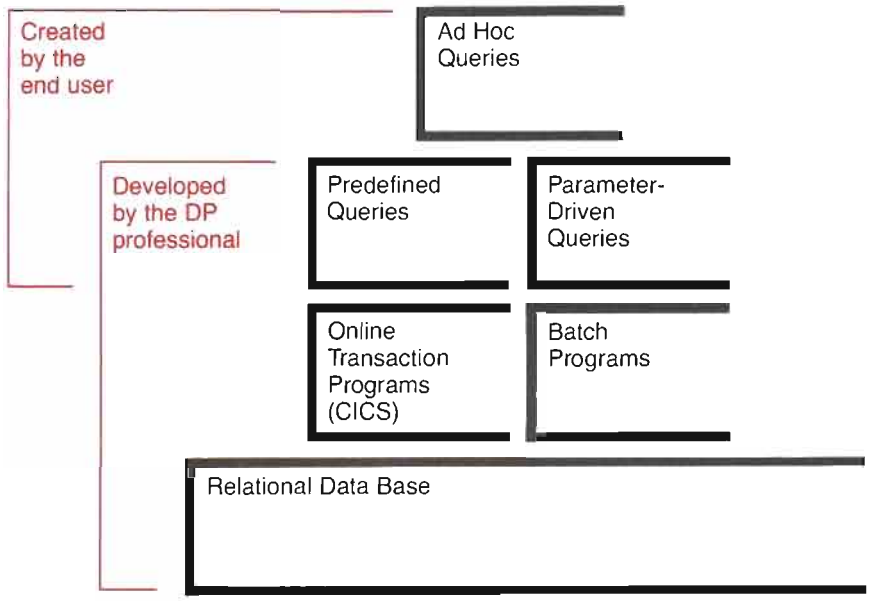
Considering all of the facilities provided by SQL/DS and the fact that it operates in conjunction with CICS/DOS/VS and DL/I DOS/VS, it is obvious that elements of a data systems environment are there. SQL/DS is especially appropriate for non-integrated applications, or for those applications that must be implemented in a relatively short time.

11 Application Design Approaches

Let's take a brief look at how an installation might use SQL/DS for new applications, or for extending existing applications. We'll consider new applications first.

New Applications

A Variety of Building Blocks to Help You Build an Application



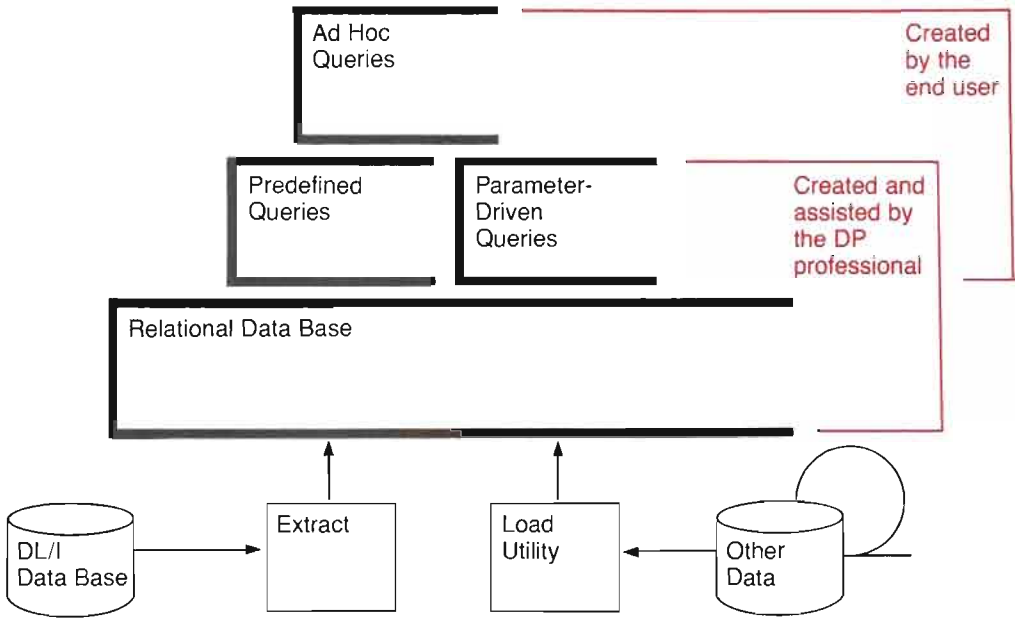
As you can see from the illustration above, there are building blocks for both the DP professional and the end user. Depending on the application, one could initially develop most of the basic functions of the application using the high level SQL language.

Once the data base is created, the end user, depending on his capabilities, could begin to write queries. Repetitively run queries could be predefined and stored for later use. There is even the capability to store queries so that they can be "parameter-driven," just like online transactions.

Of course, some of the more difficult queries will be written by DP, which may also elect to code specific online functions using a programming language. This decision to "hard code" or use a programming alternative may be postponed until after the application is verified.

Application Extensions

Allowing the User to Breathe New Life into Existing Applications



In many of today's applications, significant potential remains unrealized because the data cannot be made more readily available to the end user. In such situations, data is maintained on a regular schedule through some set of operational applications, possibly even in a data base system. Often, however, the typical programming costs to extend the use of that data for planning, reporting, and analysis purposes put a "lock" on the information.

SQL/DS can help "unlock" this data. By using the extraction and loading facilities of SQL/DS, "old data" can be given new life by transferring it to relational form. Both the DP professional (or Information Center Specialist) and the using department could develop the needed improvements using the interactive query language.

Highlights

The interactive query language consists of the Structured Query Language (SQL) and additional commands that provide access to tabular data. Using familiar words like SELECT, FROM, WHERE, and others, you can

- List all or parts of a table
- Sort or sequence the data
- Combine data in one table with data from another
- Perform calculations based on common arithmetic functions
- Invoke various built-in functions such as SUM and COUNT
- Format the output by adding a bottom page title, changing the top page title, and adding subtotal and total lines, all of which can improve the appearance of the report
- Enter, update, and delete data

Several examples of these language capabilities will be shown later in this publication. Before that, however, let's take a closer look at the *structure* of the language.

Structure

Retrieving data is the most fundamental task of SQL and, for this function, the SELECT command is used. The basic form of the SELECT command is:

SELECT	some data (field names)
FROM	some place (table names)
WHERE	certain conditions (if any) are to be met

In some instances, WHERE may not be necessary. This is shown in the first few sample queries. However, many special needs can be expressed in the WHERE part of the query, and these will be shown later.

Around this SELECT...FROM...WHERE structure, the user can place other SQL commands in order to express the many powerful operations of the language.

In all uses of SQL, the user does not have to be concerned with *how* the system should get the data. Rather, the user tells the system *what* he wants. This means that the user only needs to know the meaning of the data, not its physical representation, and this feature can relieve the user from many of the complexities of data access.

Sample Data Base

Most of the examples in this book use three tables that we've created called INVENTORY, QUOTATIONS, and SUPPLIERS. Let's look at the data in these tables by using simple queries.

First, we'll list INVENTORY in part number sequence. The commands to do this are:

User

```
SELECT * -
FROM INVENTORY -
ORDER BY INVPART

PRINT
```

Asterisk is shorthand for "all fields"

Name of table from which fields specified in SELECT are obtained

Order the output by part number

Indicates hard copy as well as display

Hard-copy Output

```
1/20/81    SELECT * FROM INVENTORY ORDER BY INVPART    PAGE    1

INVPART   PNAME            ONHAND
-------   ----------   ------------
    105   GEAR                  0
    106   GEAR                700
    124   BOLT                900
    125   BOLT               1000
    134   NUT                 900
    135   NUT                1000
    171   GENERATOR           500
    172   GENERATOR           400
    181   WHEEL              1000
    182   WHEEL              1100
    207   GEAR               7500
    209   CAM                5000
    221   BOLT              65000
    222   BOLT             125000
    231   NUT               70000
    232   NUT              110000
    241   WASHER           600000
    285   WHEEL             35000
    295   BELT               8500
```

The first 50 characters of the query are automatically printed when no page title is specified in the query.

Now let's see all of the data in the QUOTATIONS table and order the output by part within supplier number:

User

Throughout this publication hyphens are used for continuing SQL statements on multiple lines when entered from the terminal.

```
SELECT * -
FROM QUOTATIONS -
ORDER BY QUOSUPP,QUOPART

PRINT
```

Hard-copy Output

Since we did not specify the left-to-right order of the columns in the SELECT request, the columns are presented in the same order defined when the table was created.

```
 1/20/81   SELECT * FROM QUOTATIONS ORDER BY QUOSUPP,QUOPART   PAGE   1

QUOSUPP   QUOPART        PRICE     TIME          ONORD
-------   -------   ----------   ------   ------------
     51       124       1.25        5            400
     51       125       0.55        5              0
     51       134       0.40        5            500
     51       135       0.39        5           1000
     51       221       0.30       10          10000
     51       231       0.10       10           5000
     52       105       7.50       10            200
     52       205       0.15       20              0
     52       206       0.15       20              0
     53       124       1.35        3            500
     53       125       0.58        3              0
     53       134       0.38        3            200
     53       135       0.42        3           1000
     53       222       0.25       15          10000
     53       232       0.10       15          20000
     53       241       0.08       15           6000
     54       134       0.47        4              0
     54       171      21.75       20            200
     54       209      18.00       21            200
     54       221       0.10       30           5000
     54       231       0.04       30          15000
     54       241       0.02       30          10000
     57       172      45.15       25            300
     57       285      21.00       14              0
     57       295       8.50       21           2400
     61       105       9.95        8            400
     61       106       4.35        8            300
     61       221       0.20       21           5000
     61       222       0.20       21          10000
     61       241       0.05       21           4000
     64       106       4.85       10              0
     64       181       5.65       15            400
     64       182       7.05       10            400
     64       207      29.00       14           2000
     64       209      19.50        7            800
```

Finally, let's create a listing of the SUPPLIERS table. In this case, we want a specific left-to-right appearance of the data in *our list.* This differs from the left-to-right appearance in the *data base.* We specify this alteration in the SELECT statement.

User

This is the actual left-to-right appearance that we want the data to have. As stored in the SUPPLIERS table, however, SUPSUPP actually appears before NAME.

```
SELECT NAME,SUPSUPP,ADDRESS -
FROM SUPPLIERS -
ORDER BY NAME

PRINT
```

Hard-copy Output

```
 1/20/81  SELECT NAME,SUPSUPP,ADDRESS FROM SUPPLIERS ORDER BY  PAGE    1

NAME              SUPSUPP  ADDRESS
---------------   -------  ------------------------------------
ATLANTIS CO.           53  8 OCEAN AVE., WASHINGTON DC
DEFECTO PARTS          51  16 JUSTAMERE LANE, TACOMA WA
EAGLE HARDWARE         57  64 TRANQUILITY PLACE, APOLLO MN
KNIGHT LTD.            64  256 ARTHUR COURT, CAMELOT ENGLAND
SKYLAB PARTS           61  128 ORBIT BLVD., SYDNEY AUSTRALIA
TITANIC PARTS          54  32 SINKING ST., ATLANTIC CITY NJ
VESUVIUS, INC.         52  512 ANCIENT BLVD., POMPEII NY
```

NOTE: For ease of reference in using the examples in this book, the data in INVENTORY, QUOTATIONS, and SUPPLIERS is shown on the foldout page attached to the back cover.

Defining a Table

The tabular form of data as stored in SQL/DS is easy to access and understand. There are no embedded pointers or special paths to consider. The user just focuses on the data and its meaning. These same characteristics allow the data definition and creation process to be simple as well.

Before a table can be loaded, it must first be defined. The following command was used to define the INVENTORY table:

User

NOT NULL means each row of the table must have a part number

```
CREATE TABLE INVENTORY (INVPART   SMALLINT NOT NULL, -
                        PNAME     CHAR(10), -
                        ONHAND    INTEGER)
```

The QUOTATIONS table and the SUPPLIERS table were defined by these two commands:

Halfword binary

Fullword binary

```
CREATE TABLE QUOTATIONS (QUOSUPP   SMALLINT NOT NULL, -
                         QUOPART   SMALLINT NOT NULL, -
                         PRICE     DECIMAL(5,2), -
                         TIME      SMALLINT, -
                         ONORD     INTEGER)

CREATE TABLE SUPPLIERS (SUPSUPP    SMALLINT NOT NULL, -
                        NAME       CHAR(15), -
                        ADDRESS    VARCHAR(35))
```

We also defined synonyms for the INVENTORY and QUOTATIONS tables to save keystrokes when entering their names. These are the commands we used:

```
CREATE SYNONYM INV FOR USER1.INVENTORY
CREATE SYNONYM QUO FOR USER1.QUOTATIONS
```

You will see how these tables can be loaded later in this publication.

Simple List

Assume we want a list of only those parts for which the balance-on-hand is between 0 and 1000 units; we want the list in part number sequence.

User

```
SELECT * -
FROM INV -
WHERE ONHAND BETWEEN 0 AND 1000 -
ORDER BY INVPART
```

Display Output

```
INVPART  PNAME           ONHAND
-------  ----------  ------------
    105  GEAR                 0
    106  GEAR               700
    124  BOLT               900
    125  BOLT              1000
    134  NUT                900
    135  NUT               1000
    171  GENERATOR          500
    172  GENERATOR          400
    181  WHEEL             1000
```

Queries Using More Than One Table

What if we wanted to see the supplier numbers for the parts identified in the last example and the quantities ordered from these suppliers. We also want to sequence the output by part number within part name.

Refer to the foldout page of this publication. Notice that the quantity-ordered column (ONORD) and the supplier number column (QUOSUPP) are in the QUOTATIONS table. To answer this request, we will require information from the INVENTORY table *and* the QUOTATIONS table. (This type of operation is often called a *join* of tables).

User

Names of the tables for all fields used in SELECT line

Indicates matching fields between tables

```
SELECT PNAME,INVPART,ONHAND,QUOSUPP,ONORD -
FROM INV,QUO -
WHERE ONHAND BETWEEN 0 AND 1000 -
AND QUOPART=INVPART -
ORDER BY PNAME,INVPART
```

Display Output

PNAME	INVPART	ONHAND	QUOSUPP	ONORD
BOLT	124	900	51	400
BOLT	124	900	53	500
BOLT	125	1000	53	0
BOLT	125	1000	51	0
GEAR	105	0	52	200
GEAR	105	0	61	400
GEAR	106	700	61	300
GEAR	106	700	64	0
GENERATOR	171	500	54	200
GENERATOR	172	400	57	300
NUT	134	900	51	500
NUT	134	900	54	0
NUT	134	900	53	200
NUT	135	1000	53	1000
NUT	135	1000	51	1000
WHEEL	181	1000	64	400

Even from this simple example, some of the power of SQL should become obvious. Consider, from a programming point of view, what would be required to do the same thing:

> A record from one file has to be read, the selection criteria evaluated, and the data fields extracted. One of these fields will then be used to find the corresponding records from the other file. Then the other file has to be searched, using this matching field. When a record is selected, the required data has to be extracted and processed. In most cases, another record has to be read to see if there are any other matching records to process.

These programming considerations tend to get even more complicated as the number of matching files or tables increases.

This is not the case with SQL because it works on *sets* of records at a time, and because the user does not have to tell it how to go about getting the data. In fact, there are search optimizers in the relational access mechanism that attempt to minimize the amount of data search processing for the user.

Formatting a Report

The appearance of the output from the previous query can be improved by making the column headings more readable and adding a title. We can also create subtotal lines of the orders for each part. For example, if we had just executed the query in the previous example, these commands would generate the subsequent formatted report:

User

Changes INVPART to PART

Causes group indication on DESCRIPTION

Subtotals orders

Titles report on top of page

```
FORMAT COLUMN INVPART NAME 'PART'
FORMAT COLUMN PNAME NAME 'DESCRIPTION'
FORMAT COLUMN QUOSUPP NAME 'SUPPLIER'
FORMAT COLUMN ONORD NAME 'ON ORDER'
FORMAT COLUMN ONHAND NAME 'ON HAND'
FORMAT GROUP DESCRIPTION
FORMAT GROUP PART
FORMAT SUBTOTAL 'ON ORDER'
FORMAT TITLE 'INVENTORY REPORT'
PRINT
```

Hard-copy Output

Subtotal lines

```
01/20/81                    INVENTORY REPORT

DESCRIPTION     PART       ON HAND  SUPPLIER      ON ORDER
-----------  ------  ------------  --------  ------------
BOLT            124           900        51           400
                              900        53           500
                                                ------------
                ******                                  900

                125          1000        53             0
                             1000        51             0
                                                ------------
                ******                                  0
                                                ------------
***********                                             900

GEAR            105             0        52           200
                                0        61           400
                                                ------------
                ******                                  600

                106           700        61           300
                              700        64             0
                                                ------------
                ******                                  300
                                                ------------
***********                                             900

GENERATOR       171           500        54           200
                                                ------------
                ******                                  200

                172           400        57           300
                                                ------------
                ******                                  300
                                                -------
***********

NUT             134           900
```

Built-In Functions

The built-in functions of SQL are another indication of the power of the language and can save a significant amount of programming. For example, we can also find the total orders for each of the part numbers in the preceding report by using the built-in function SUM. This time, let's sequence the output by balance-on-hand in descending order.

User

```
SELECT QUOPART,PNAME,ONHAND,SUM(ONORD) -
FROM INV,QUO -
WHERE ONHAND BETWEEN 0 AND 1000 -
AND QUOPART = INVPART -
GROUP BY QUOPART,PNAME,ONHAND -
ORDER BY ONHAND DESC
```

The sum is calculated for each unique combination of these fields

Descending order

Display Output

QUOPART	PNAME	ONHAND	SUM(ONORD)
181	WHEEL	1000	400
135	NUT	1000	2000
125	BOLT	1000	0
134	NUT	900	700
124	BOLT	900	900
106	GEAR	700	300
171	GENERATOR	500	200
172	GENERATOR	400	300
105	GEAR	0	600

In this simple use of the SUM function, consider what a programmer would probably have to do in order to accomplish the same thing.

First, a list of parts ordered has to be obtained and sequenced by part number within part name. Then, a subtotal of the orders has to be computed by part number and saved.

Next, a listing for output has to be developed for those parts matching the search criteria (performing all the things we had to do in "Queries Using More Than One Table") and, for each part, the on-order subtotals previously saved must be retrieved. Finally, the listing has to be sorted by quantity-on-hand, in descending sequence.

Almost every one of these operations requires a record to be read, or some data stored and a location posted. And, for each I/O operation, return codes and ancillary logic are usually necessary.

Let's consider another example of the power of built-in functions. For every part in our inventory, we want to list the minimum, maximum, and average prices charged by the various suppliers and we also want to show a count of the number of suppliers that we have for each part. We will use the MIN, MAX, AVG, and COUNT built-in functions to do this.

User

The COUNT(*) allows us to count the number of occurrences within each group defined by the GROUP BY clause.

Indicates matching field between INV and QUO

Indicates the level at which to calculate the built-in functions

Replace heading of column six

```
SELECT QUOPART,PNAME,MIN(PRICE),MAX(PRICE),AVG(PRICE),COUNT(*) -

FROM INV,QUO -
WHERE QUOPART=INVPART -
GROUP BY QUOPART,PNAME -
ORDER BY QUOPART

FORMAT COLUMN 5 PRECISION 3

FORMAT COLUMN 6 NAME 'NO OF SUPPLIERS'
```

Means the fifth column

Reduces the precision to 3 decimal places

Display Output

QUOPART	PNAME	MIN(PRICE)	MAX(PRICE)	AVG(PRICE)	NO OF SUPPLIERS
105	GEAR	7.50	9.95	8.725	2
106	GEAR	4.35	4.85	4.600	2
124	BOLT	1.25	1.35	1.300	2
125	BOLT	0.55	0.58	0.565	2
134	NUT	0.38	0.47	0.417	3
135	NUT	0.39	0.42	0.405	2
171	GENERATOR	21.75	21.75	21.750	1
172	GENERATOR	45.15	45.15	45.150	1
181	WHEEL	5.65	5.65	5.650	1
182	WHEEL	7.05	7.05	7.050	1
207	GEAR	29.00	29.00	29.000	1
209	CAM	18.00	19.50	18.750	2
221	BOLT	0.10	0.30	0.200	3
222	BOLT	0.20	0.25	0.225	2
231	NUT	0.04	0.10	0.070	2
232	NUT	0.10	0.10	0.100	1
241	WASHER	0.02	0.08	0.050	3
285	WHEEL	21.00	21.00	21.000	1
295	BELT	8.50	8.50	8.500	1

Tailoring SQL/DS for the End User

SQL/DS was designed for a broad range of users with varying backgrounds and different capabilities. While many people will be able to learn much of the query language, the DP professional or Information Center Specialist can further simplify its use by creating views, stored queries, and parameter-driven queries.

Tailoring— Creating a View

For some users, a single table may be considerably easier to work with than multiple tables. In addition to tables, SQL supports *views*. A *view* is a logical (or "virtual") table that is derived from one or more tables or other views. In general, views look like, and can be operated on, just as if they were real tables. They can simplify data access requests and can reduce keystrokes and errors.

To create a single-table view of the data from our three tables for the users in the Nuts and Bolts Department (department 17) so that they may retrieve their data as if it were in a single table, we could enter the following:

User

```
CREATE VIEW D17INVENTORY (SUPPNAME,ADDRESS,D17PART,PARTNAME,LEADTIME, -
                          ONHAND,ONORDER,PRICE,TOTALPRICE) AS -
   SELECT NAME,ADDRESS,QUOPART, -
          PNAME,TIME,ONHAND,ONORD,PRICE,PRICE*ONORD -
   FROM INV,QUO,SUPPLIERS -
   WHERE INVPART = QUOPART -
   AND SUPSUPP = QUOSUPP -
   AND PNAME IN ('NUT','BOLT')
```

D17 INVENTORY

SUPPNAME	ADDRESS	D17PART	PARTNAME	LEADTIME	ONHAND	ONORDER	PRICE	TOTALPRICE

Users in Department D17 would think of their data as if it were all in a single table, as shown above. Only the DP department would need to know that the data is actually stored in several tables.

The end user is thus insulated from the actual physical data storage. Moreover, it is possible for the underlying data storage to change (for example, the fields in a table could be rearranged or new fields could be added) and the user would neither know or care.

Now that their data appears as a single table, the users' queries are simpler to write. Consider how easily the following ad hoc requests might be answered by the inventory department using SQL/DS.

How much do we owe TITANIC PARTS?

User

TOTALPRICE is a predefined calculation stored in the view

```
SELECT SUM(TOTALPRICE) -
FROM D17INVENTORY -
WHERE SUPPNAME = 'TITANIC PARTS'
```

Display Output

```
SUM(TOTALPRICE)
---------------
        1100.00
```

How much do we owe our suppliers for BOLTS?

User

```
SELECT SUM(TOTALPRICE) -
FROM D17INVENTORY -
WHERE PARTNAME = 'BOLT'
```

Display Output

```
SUM(TOTALPRICE)
---------------
       10175.00
```

How many different parts do we stock?

User

DISTINCT gives us a count of the part numbers that are distinct, not just a count of part numbers within the D17PART column (which would include duplicates)

```
SELECT COUNT(DISTINCT D17PART) -
FROM D17INVENTORY
```

Display Output

```
COUNT(DISTINCT D17PART)
-----------------------
                      8
```

Do we have a supplier located in TACOMA?

User

The % signs cause a search for the character string anywhere in the ADDRESS field.

```
SELECT DISTINCT SUPPNAME,ADDRESS
FROM D17INVENTORY -
WHERE ADDRESS LIKE '%TACOMA%'
```

Display Output

```
SUPPNAME         ADDRESS
---------------  -----------------------------------
DEFECTO PARTS    16 JUSTAMERE LANE, TACOMA WA
```

Tailoring—Stored Queries

You can save a query for later use without having to re-enter it when it has to be used.

Let's say, for example, that the queries created on the D17INVENTORY view were going to be frequently used by our inventory clerks. The query for the SUM of TOTALPRICE for BOLTs could be made a permanent part of a clerk's query library by entering:

User

HOLD causes the query to be held for further processing before execution

STORE causes the currently held query to be stored for execution by name

```
HOLD SELECT SUM(TOTALPRICE) -
     FROM D17INVENTORY -
     WHERE PARTNAME = 'BOLT'

STORE BOLTS
```

START is used to execute a query that has been stored

Whenever our inventory clerks have to run that report, they would enter:

```
START BOLTS
```

Tailoring—Parameter-Driven Queries

Users can also store a query in such a way that allows it to be tailored to meet specific requests at execution time.

If a user wanted to be able to find the name and address of suppliers from *any* given city, he could store a parameter-driven query that produced the result for a *specific* city by entering:

User

&1 is the parameter for which a city name will be substituted by the user when he "starts" his query.

```
HOLD SELECT DISTINCT SUPPNAME,ADDRESS -
     FROM D17INVENTORY -
     WHERE ADDRESS LIKE '% &1 %'
STORE SUPPCITY
```

To execute this query for CAMELOT, the user would enter:

User

```
START SUPPCITY (CAMELOT)
```

Display Output

```
SUPPNAME          ADDRESS
---------------   -----------------------------------
KNIGHT LTD.       256 ARTHUR COURT, CAMELOT ENGLAND
```

Making Changes to Stored Data

SQL/DS allows users to modify, delete, and insert data into shared or private tables. These operations can be performed interactively from the terminal and can be especially useful to persons who would like to maintain their own records. The following examples will give you an idea of the scope of the system's capabilities in this area.

Modifying Data

Suppose supplier 51 increases his prices by 11%. We have to update QUOTATIONS to reflect this change.

User

```
UPDATE QUO -
SET PRICE = PRICE * 1.11 -
WHERE QUOSUPP = 51
```

A subsequent query on this table would confirm the update.

```
SELECT * FROM QUO ORDER BY QUOSUPP
PRINT
```

Hard-copy Output

```
 1/20/81  SELECT * FROM QUO ORDER BY QUOSUPP     PAGE   1

QUOSUPP  QUOPART      PRICE    TIME         ONORD
-------  -------  ----------  ------  ------------
     51      134        0.44       5           500
     51      124        1.38       5           400
     51      221        0.33      10         10000
     51      231        0.11      10          5000
     51      125        0.61       5             0
     51      135        0.43       5          1000
     52      205        0.15      20             0
     52      206        0.15      20             0
     52      105        7.50      10           200
     53      241        0.08      15          6000
     53      232        0.10      15         20000
     53      222        0.25      15         10000
     53      135        0.42       3          1000
     53      134        0.38       3           200
     53      124        1.35       3           500
     53      125        0.58       3             0
     54      209       18.00      21           200
     54      134        0.47       4             0
     54      171       21.75      20           200
     54      241        0.02      30         10000
     54      231        0.04      30         15000
     54      221        0.10      30
     57      295        8.50      21
     57      285       21.00      14
     57      172       45.15
```

Users of views containing PRICE (such as D17INVENTORY) would automatically have this increase reflected in their data.

Adding Columns to a Table

Suppose that we require new columns in the QUOTATIONS table for DISCOUNTRATE and DISCOUNTQTY. These columns may be added to the table using the following SQL commands:

User

```
ALTER TABLE QUOTATIONS -
ADD DISCOUNTRATE DECIMAL (2,2)

ALTER TABLE QUOTATIONS -
ADD DISCOUNTQTY INTEGER
```

With SQL/DS, these data definitions are dynamically executed. All the existing rows of the QUOTATIONS table are effectively expanded and stored with an initial "null" value and the user need not be concerned with "reorganizing" the data.

Another feature of SQL/DS is that most queries and views that referred to the QUOTATIONS table before these columns were added do not have to be modified. For example, the D17INVENTORY view is not affected by this modification.

Actual data for these new columns would be entered using a series of UPDATE commands. For example, assume that TITANIC PARTS (supplier number 54) has a discount rate of 10 percent on parts over 10 dollars and a discount rate of 5 percent on parts 10 dollars or less. Furthermore, assume their discount quantity is 20 for the former and 100 for the latter sets of parts. This information can be added to the QUOTATIONS table using the following UPDATE commands:

User

```
UPDATE QUOTATIONS -
  SET DISCOUNTRATE=.10,DISCOUNTQTY=20 -
  WHERE QUOSUPP=54 -
  AND PRICE > 10

UPDATE QUOTATIONS -
  SET DISCOUNTRATE=.05,DISCOUNTQTY=100 -
  WHERE QUOSUPP= 54 -
  AND PRICE <= 10
```

To verify that updates were made, we can query the (newly expanded) QUOTATIONS table. Notice that the quotations for which no discount rate or discount quantity has been entered have "null" (?) entries in the corresponding columns.

User

```
SELECT QUOSUPP,QUOPART,PRICE,DISCOUNTRATE,DISCOUNTQTY -
FROM QUOTATIONS -
WHERE QUOSUPP > 53 -
ORDER BY QUOSUPP,QUOPART
```

Display Output

QUOSUPP	QUOPART	PRICE	DISCOUNTRATE	DISCOUNTQTY
54	134	0.47	0.05	100
54	171	21.75	0.10	20
54	209	18.00	0.10	20
54	221	0.10	0.05	100
54	231	0.04	0.05	100
54	241	0.02	0.05	100
57	172	45.15	?	?
57	285	21.00	?	?
57	295	8.50	?	?
61	105	9.95	?	?
61	106	4.35	?	?
61	221	0.20	?	?
61	222	0.20	?	?
61	241	0.05	?	?
64	106	4.85	?	?
64	181	5.65	?	?
64	182	7.05	?	?
64	207	29.00	?	?
64	209	19.50	?	?

Each user can set his own "null" character for reporting purposes.

Entering Data from a Terminal

Entire rows of data, or portions of rows of data, can be entered by the user directly from the terminal. Suppose the purchasing department was negotiating contracts with new suppliers. If supplier numbers were assigned, we could add new rows to the SUPPLIERS table as follows:

User

To insert a single row into a table

```
INSERT INTO SUPPLIERS (NAME,ADDRESS,SUPSUPP) -
VALUES ('OLYMPUS CORP','12 KROE ST., COS COB CT',66)
```

Or

To insert multiple rows into a table

```
INPUT SUPPLIERS (NAME,ADDRESS,SUPSUPP)
'VILLAGE PARTS','347 HILLSBORO AVE., PHILA PA',67
'PLASTICS INC.','16 HIGHMEADOW RD., ST.LOUIS MO',72
'SUPERIOR METALS','160 SANDPIPER AVE., MOUNTAIN VIEW CA',70
END
```

Data and System Administration

Data and System Administration

In every business, someone is responsible for such assets as cash, inventory, and production, and for the standards and procedures that make a business run. Within a data processing department, there is a similar need to control, audit, and protect *its* resources.

This requires documentation about data, programs, queries, and users. It also means that the data definitions and relationships between these items must be documented so that they can be made available whenever changes occur. Frequently asked questions are:

- Who accesses the data and which programs use it?
- Which programmers are responsible for which applications?
- What will be affected by changing a particular data field?

All too often, the initial effort for creating and maintaining this information is greater than a department can afford. And, as a result, this important information is scattered throughout the DP department: in program listings, in desk drawers, or in people's memories.

The System Catalogs of SQL/DS

Data and system administration is aided by use of the SQL/DS system catalogs. These catalogs are a special set of tables, which are totally integrated into the system, and which contain descriptive information on data, programs, users, and other "objects" in the SQL/DS system.

These tables look like and can be queried like any other tables in SQL/DS, except that their contents are *dynamic* and maintained through the use of the SQL Data Definition commands, rather than through UPDATE, INSERT, and DELETE commands. Some of the items described by the SQL/DS catalogs, and the specific catalog tables that describe them are shown below:

Item	Catalog Name	Comments
Tables	SYSCATALOG	Includes views
Columns	SYSCOLUMNS	Includes view columns
Programs	SYSACCESS	Information on programs
Programs	SYSUSERAUTH	More information on programs
Users	SYSUSERAUTH	User authorities
Synonyms	SYSSYNONYMS	User synonyms
Table usage	SYSUSAGE	Tables used by programs
Queries	STMTS	Stored queries

Managing a Typical DP Department Problem

Let's look at how these catalogs can be used to help manage a typical DP department problem.

A senior programmer (GEORGE) is transferred and we wish to find out which programs/queries he wrote and what tables these programs/queries used. We also want to know what tables he created.

The following queries will give us this information:

• Find Programs Written by GEORGE

To find the programs written by senior programmer GEORGE, we would query the SYSACCESS catalog:

User

Since all the catalogs are "owned" by the system, references to catalog names must be preceded by "SYSTEM."

```
SELECT TNAME,TIMESTAMP -
FROM SYSTEM.SYSACCESS -
WHERE CREATOR = 'GEORGE' -
AND TABTYPE = 'X'

FORMAT COLUMN TNAME NAME PROGRAM

FORMAT TTITLE 'GEORGES PROGRAMS'

PRINT
```

Hard-copy Output

```
1/20/81          GEORGES PROGRAMS          PAGE   1

 PROGRAM              TIMESTAMP
 ------------------   -----------------
 ORDERPLAN            06/16/80.16:55:17
 ORDERANALYSIS        06/13/80.18:54:32
 PARTSBUDGET          06/16/80.17:15:23
```

• Find Tables Used by GEORGE's Programs

Tables used by GEORGE's programs can be found by querying SYSUSAGE:

User

Can be a real table or a view

Using objects are programs

GEORGE's programs

```
SELECT DNAME,BCREATOR,BNAME,BTYPE -
FROM SYSTEM.SYSUSAGE -
WHERE BTYPE IN ('R','V') -
AND DTYPE = 'X' -
AND DCREATOR = 'GEORGE' -
ORDER BY DNAME,BCREATOR

FORMAT COLUMN DNAME NAME PROGRAM

FORMAT COLUMN BCREATOR NAME 'TABLE OWNER'

FORMAT COLUMN BNAME NAME 'TABLE NAME'

FORMAT COLUMN BTYPE NAME 'TABLE TYPE'

FORMAT TTITLE 'TABLES USED BY GEORGES PROGRAMS'

PRINT
```

Hard-copy Output

```
1/20/80  TABLES USED BY GEORGES PROGRAMS  PAGE   1

 PROGRAM              TABLE OWNER  TABLE NAME          TABLE TYPE
 ------------------   -----------  ------------------  ----------
 ORDERANALYSIS        JONESDA      SUPPLIERS           R
 ORDERANALYSIS        GEORGE       D17INVENTORY        V
 ORDERPLAN            JONESDA      INVENTORY           R
 ORDERPLAN            JONESDA      QUOTATIONS          R
 PARTSBUDGET          JONESDA      QUOTATIONS          R
```

• Find Tables Created by GEORGE

To find the tables created by GEORGE, we would query SYSCATALOG:

User

Table name and type

To separate views and real tables

```
SELECT TNAME,TABTYPE -
FROM SYSTEM.SYSCATALOG -
WHERE CREATOR = 'GEORGE' -
ORDER BY TABTYPE,TNAME

FORMAT COLUMN TNAME NAME 'TABLE NAME'

FORMAT COLUMN TABTYPE NAME 'TYPE'

FORMAT TTITLE 'GEORGES TABLES'

PRINT
```

Hard-copy Output

```
1/20/81   GEORGES TABLES  PAGE   1

TABLE NAME          TYPE
------------------  ---------
CUSTOMERS           R
DEPT                R
EMPLOYEES           R
COMPOSITE           V
DEPT PAY            V
D17INVENTORY        V
INV                 V
ORGANIZATION        V
PAYCHECKS           V
QUO                 V
```

Imagine how much work it would take to accumulate this information manually?

Securing the Data Base

Another aspect of data administration and control is data security. Although not all data is considered to be "sensitive", most end-user applications require some level of data authorization to control the reading and/or writing of data across the data base. The ability to share access on tables and views within SQL/DS is provided by the GRANT and REVOKE commands. These commands can be used only by those who have been authorized to use them or by the owners of the tables and views.

Let's look at some examples

Assume that the department manager has decided to allow all users to have only retrieval (read) access to the SUPPLIERS table. He would grant their access privilege by issuing the following command:

User

```
GRANT SELECT -
ON SUPPLIERS -
TO PUBLIC
```

As another example, the administrator could further authorize USER1 and USER2 to update the ADDRESS column in SUPPLIERS:

```
GRANT UPDATE (ADDRESS) -
ON SUPPLIERS -
TO USER1,USER2
```

JONES is in the Purchasing Department and negotiates contracts with suppliers numbered 51 and 53. Let's allow JONES to have update privileges only on these suppliers for the PRICE and TIME columns in the QUOTATIONS table.

First, we define a view on QUO that subsets the data for suppliers 51 and 53 only. (JONES already has SELECT access to QUO.)

```
CREATE VIEW JONESQUO -
AS   SELECT * -
     FROM QUO -
     WHERE QUOSUPP IN (51, 53)
```

Then we grant update access on this view:

```
GRANT UPDATE (PRICE,TIME) -
ON JONESQUO -
TO JONES
```

The authorization capability of SQL/DS is quite thorough and can be used to establish similar authorizations for other operations such as ALTER, DELETE, and INSERT.

Quite often the information used in analysis and planning applications is derived directly from data stored in existing production files. Generally, the user performing the analysis requires the data from different points of view and arranged rather differently than it may have been structured for the production applications.

By loading their data into relational tables, these users can obtain the freedom necessary to do various queries across these tables in ways which perhaps could not have been easily anticipated and predefined.

Finally, the interactive query language gives users the opportunity to directly access their data with less dependence on the data processing professional to write a program for each of their requests.

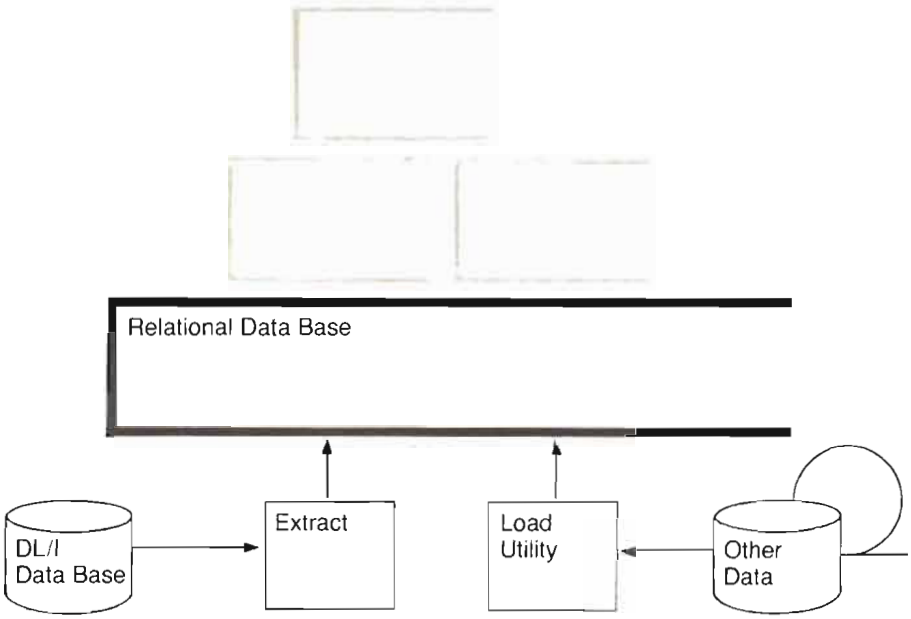
Although the user could directly enter his data from the terminal, to meet *bulk* loading requirements SQL/DS provides various facilities for loading large volumes of data into relational tables from existing files.

Loading Data from Sequential Files

Typically, many information needs can be satisfied simply by putting the data online and allowing users to directly access it.

SQL/DS provides a batch utility program, called Data Base Services (DBS), that provides several supportive functions for maintaining the system. The DATALOAD command of the DBS Utility allows you to load rows into a previously defined SQL/DS table from data contained in a user-created sequential file (SAM).

We mentioned earlier that SQL/DS can be used in conjunction with an interactive system such as IBM's VSE/ICCF. In this way, even batch programs can be invoked by users from their terminals. Invocation of the DBS utility program is a good example of this "batch" mode of execution.



Load the "Equipment" File into SQL/DS

Many installations maintain a capital equipment file to keep track of depreciated assets, delivery schedules, and costs. If this data were loaded into SQL/DS, then the system's easy-to-use query facilities would allow the DP department to make timely use of this information and keep abreast of changes readily.

Let's assume that the "Equipment" file is sequential (SAM) and stored on DASD. The ICCF user can load this data into SQL/DS using the ICCF library support and the SQL/DS DBS utility in the following way.

First, using the ICCF editor, he could build a DBS job in an ICCF library member (call it EQIPLOAD). The resulting job stream would look like this:

This statement obtains the DBS utility

This statement says the DBS command cards follow

This card tells DBS who the user is

```
/LOAD ARIDBS
/FILE NAME=EQUIP,SER=xxxxxx
/OPTION GETVIS=AUTO
/DATA
CONNECT GEORGE IDENTIFIED BY PSWRD
CREATE TABLE EQUIPMENT (EQUIPTYP SMALLINT,
                        NAME CHAR(15),
                        SERIALNO INTEGER,
                        MODELNO CHAR(10),
                        FUNDNGCD SMALLINT,
                        LOCATION CHAR(11),
                        ACQDATE CHAR(6),
                        PVALUE INTEGER,
                        LVALUE INTEGER);
DATALOAD TABLE (EQUIPMENT)
    EQUIPTYP      2-3
    NAME          5-19
    SERIALNO     21-28
    MODELNO      31-40
    FUNDNGCD     41-42
    LOCATION     44-55
    ACQDATE      58-63
    PVALUE       68-77
    LVALUE       78-87
INFILE (EQUIP PDEV(DASD))
.
.
.
```

This job could then be run by executing the library member:

```
/EXEC EQIPLOAD
```

The DBS utility messages could be transmitted to the terminal so the user would see when his table was loaded. At that point, he could sign on to SQL/DS and immediately begin to query the new table.

Extracting and Loading Data from DL/I Data Bases

The online DL/I EXTRACT facility allows users to issue requests for DL/I data from their terminals, yet provides the system administrator with a high degree of central control. This arrangement allows the Data Base Administrator to determine the proper time to actually execute the data transfer. The most appropriate time, for example, may be at off-peak hours.

All phases of the extraction process are invoked by the interactive query language and a typical application is described in the following examples.

Let's assume that the INVENTORY table data comes from our DL/I data base. Before the SQL/DS EXTRACT facility can communicate with DL/I, the system administrator must first describe to SQL/DS the DL/I data needed by INVENTORY.

This is done online, using DEFINE commands of the interactive query language (not shown). These definitions are stored in the SQL/DS catalogs, and refers to special kinds of tables called *external data tables* (EDTs).

In our example, let's say that the system administrator called this external data table INVDB_PARTS, and that USER1 and others want to access it in order to load their tables.

Using the following commands, the administrator gives three users the authority to issue EXTRACT commands against INVDB_PARTS:

Administrator

```
GRANT EXTRACT ON INVDB_PARTS TO USER1
GRANT EXTRACT ON INVDB_PARTS TO USER2
GRANT EXTRACT ON INVDB_PARTS TO USER3
```

Once a user has been granted "extract" privileges, he can request that the DL/I data be copied to the SQL/DS tables for which he has been authorized for insert operations.

User (USER1)

```
EXTRACT INTO INV(INVPART,PNAME,ONHAND) -
  SELECT(PARTS_NUMBER,PARTS_DESCR,PARTS_BALANCE) -
FROM SA.INVDB_PARTS
```

The SA owns INVDB and is sharing it with these users

The system will put the request on the EXTRACT queue, and assign an EXTRACT ID (extract identification number) for the request. An information message is issued to the user (USER1) giving him the EXTRACT ID (for example, 56).

System

```
EXTRACT REQUEST NOW IS WAITING TO BE SUBMITTED.
EXTRACT ID IS 56.
```

At the appropriate time, the administrator determines if there were any outstanding extract requests and submits the queue of INVDB requests for execution as follows:

Administrator

The Administrator is asking for all outstanding requests

```
READQ *
```

System

```
EXTRACTID  STATUS   EXTRACTOR  EDTNAME      DEFINER  FNAME
---------  -------  ---------  -----------  -------  -----------
       56  WAITING  USER1      INVDB_PARTS  SA       INVDB
```

Administrator

The Administrator is submitting all requests against the INVDB data base

```
SUBMIT INVDB
```

Users can determine the status of their extracts by entering:

User (USER1)

The user is asking for the status of all his requests

```
READQ *
```

The status of the user's requests will be displayed. If the extract run has been completed, the system responds as follows:

System

```
EXTRACTID  STATUS   TNAME      CREATOR   EDTNAME      DEFINER
---------  -------  ---------  --------  -----------  --------
       56  DONE     INV        USER1     INVDB_PARTS  SA
```

With the run completed, USER1 would then want to delete the entry in the EXTRACT queue. This would be done by issuing the DELETEQ request for EXTRACTID 56:

User (USER1)

```
DELETEQ 56
```

The user-to-system dialogue facility automates many of the manual activities typical of data extraction procedures. It allows users and the system administrator to carry on their regular activities without needing to be in constant touch with one another.

SQL/DS is not just for end-user access. One can also use its facilities in a normal application program.

In order to provide special functions, or to access other data (e.g. DL/I) along with SQL/DS data, or to improve the efficiency of highly repetitive operations, it may be more appropriate to develop online transaction programs instead of writing interactive SQL queries.

Online Transactions Via COBOL, PL/I, or Assembler

As an application programmer, you can develop online transactions in one of these host programming languages by using SQL statements inline in your program.

Let's see how we might do this in a COBOL program that will be executed as a CICS transaction.

Suppose the purchasing department wants to determine the suppliers of "overpriced" parts so that it can renegotiate orders.

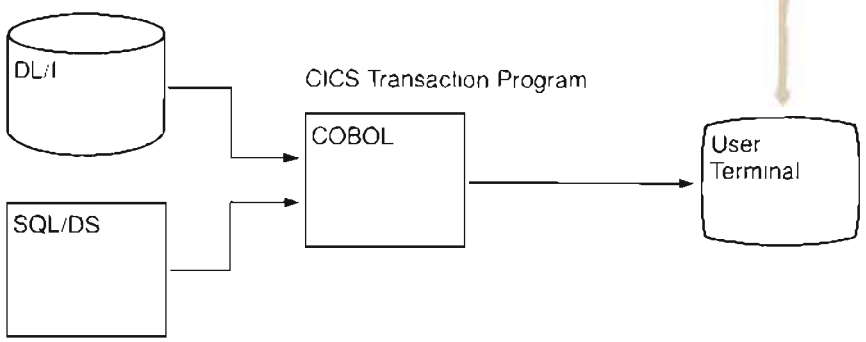
The transaction program will have to develop a list of suppliers whose price for a part exceeds the average price for that part.

For each of these suppliers, the program also has to retrieve the part numbers, the quantities on hand for these parts, and the quantities our customers are ordering *from us* for the parts.

All but the customer order information can be retrieved from the SQL/DS data base. However, our customer order entry system, which is on DL/I, uniquely contains the customer orders. Therefore, the program has to access *both* the SQL/DS and the DL/I data bases.

We want the program to develop screens similar to the one shown below, giving the purchasing agent the information needed.

FLAG	SUPPLIER #	PART #	SUPPLIER INV STATUS	STATUS QUANTITY ON ORDER FROM SUPPLIER	SUPPLIER PRICE	AVERAGE PRICE	TOTAL CUSTOMER ORDER
*	51	221	65,000	10,000	.30	.20	12,450
	51	231	70,000	5,000	.10	.07	55,000
*	53	241	60,000	6,000	.08	.05	26,000
*	64	209	5,000	800	19.50	18.75	975



Portions of a COBOL program to retrieve this information are shown below.

This causes the SQL return code structure to be automatically generated in the program.

This section identifies the program variables to be used in SQL statements.

This statement identifies where to find error-handling logic for SQL errors.

This statement defines a cursor for stepping through the query results one row at a time.

The FETCH statement retrieves one row of the query result and puts the field values in the program variables identified by the INTO clause.

This routine builds the output screen (code not shown).

For each new part, a DL/I call is executed to retrieve the customer order directly from the DL/I data base.

The program could also make an evaluation of the inventory status and the customer orders, based on other indicative information, and flag those supply orders that appear cancelable. The purchasing agent, of course, would make the decision. (This code is not shown.)

```
WORKING STORAGE SECTION.
     EXEC SQL INCLUDE SQLCA END-EXEC
      .
      .
      .
     EXEC SQL BEGIN DECLARE SECTION END-EXEC.
01    SUPP PIC S9(9) COMP.
01    PART PIC S9(9) COMP.
01    PRICE PIC S9(3)V9(2) COMP-3.
01    ONORD PIC S9(9) COMP.
      AVPRC PIC S9(3)V9(2) COMP-3
      INVOH PIC S9(9) COMP


PROCEDURE DIVISION:
      .
      .
      .
     EXEC SQL WHENEVER SQLERROR GOTO
          ERROR-RECOVERY END-EXEC.
      .
      .
      .
     EXEC SQL DECLARE C1 CURSOR FOR
              SELECT QUOSUPP,QUOPART,PRICE,ONORD,AVPRC,INVOH
              FROM QUO, TEMP
              WHERE QUOPART = TEMPPART
              AND PRICE > AVPRC
    END-EXEC.
      .
      .
      .
GETNEXT.

     EXEC SQL FETCH C1
              INTO :SUPP,:PART,:PRICE,:ONORD,:AVPRC,:INVOH
    END-EXEC.
      .
      .
      .
CALCORDER.
      .
      .
      .
BUILDMAP.
      .
      .
      .
     EXEC DL/I GET UNIQUE USING PCB(1)
        SEGMENT (PARTSEG) WHERE (KEY = PART-NUMBER) INTO
        (PART-SEG-IN) END-EXEC.
      .
      .
      .
    EXEC CICS RETURN END-EXEC.
```

The SQL language exploits the power of relational operations and allows the user to clearly state the solution to solve the problem

The examples that follow are more complex than the earlier examples of the SQL language shown. These examples are presented so that you can appreciate the power and scope of the SQL relational operations, although, at this point, you may not completely understand the language.

At the very least, these sample queries should make it clear that an SQL user, in relatively few statements, can define operations that could take several pages of statements using traditional programming languages and perhaps several days to debug.

Using Multiple Queries Together

Creating a Temporary Table Querying Three Tables

Assume that our Inventory department is instituting a new ordering policy: "NEW ORDERS MUST EXCEED TWICE THE REMAINING BALANCE." Let's see those parts, with supplier and price, when this rule is not in effect

As you can see from the output in the "min/max" example of the "Built-in Functions" section, more than one supplier usually supplies each part, and there may be multiple orders for a part. However, in our data base, there is no single field that contains the total orders for each part (This often happens in a data base. One does not always store computed fields for every requirement.)

One simple approach to solving our problem when we need to know the total orders is shown below.

First Step: First we build a temporary table creating the field we need (TOTORDER), and then we query against that field.

User

```
CREATE TABLE TEMP (TEMPPART INTEGER,TOTORDER INTEGER)

INSERT INTO TEMP VALUES SELECT QUOPART,SUM(ONORD) -
                    FROM QUO -
                    GROUP BY QUOPART
```

Result

TEMP Table

TEMPPART	TOTORDER
106	300
124	900
125	0
.	.
.	.
.	.
295	2400

Second Step: Now we can access the new table, along with the QUOTATIONS and INVENTORY tables, to get our answer.

User

Matching fields between QUO and INV

Matching fields between QUO and TEMP

```
SELECT QUOPART,PNAME,QUOSUPP,PRICE,ONHAND,ONORD -
FROM QUO,INV,TEMP -
WHERE QUOPART=INVPART -
AND QUOPART=TEMPPART -
AND TOTORDER < 2 * ONHAND -
ORDER BY QUOPART
```

Display Output

QUOPART	PNAME	QUOSUPP	PRICE	ONHAND	ONORD
106	GEAR	61	4.35	700	300
106	GEAR	64	4.85	700	0
124	BOLT	51	1.25	900	400
124	BOLT	53	1.35	900	500
125	BOLT	51	0.55	1000	0
125	BOLT	53	0.58	1000	0
134	NUT	51	0.40	900	500
134	NUT	53	0.38	900	200
134	NUT	54	0.47	900	0
171	GENERATOR	54	21.75	500	200
172	GENERATOR	57	45.15	400	300
181	WHEEL	64	5.65	1000	400
182	WHEEL	64	7.05	1100	400
207	GEAR	64	29.00	7500	2000
209	CAM	64	19.50	5000	800
209	CAM	54	18.00	5000	200
221	BOLT	51	0.30	65000	10000
221	BOLT	54	0.10	65000	5000
221	BOLT	61	0.20	65000	5000
222	BOLT	61	0.20	125000	10000
222	BOLT	53	0.25	125000	10000
231	NUT	54	0.04	70000	15000
231	NUT	51	0.10	70000	5000
232	NUT	53	0.10	110000	20000
241	WASHER	53	0.08	600000	6000
241	WASHER	54	0.02	600000	10000
241	WASHER	61	0.05	600000	4000
285	WHEEL	57	21.00	35000	0
295	BELT	57	8.50	8500	2400

Third Step: Since we no longer need the data in the TEMP table, we can elect to DROP (erase) it.

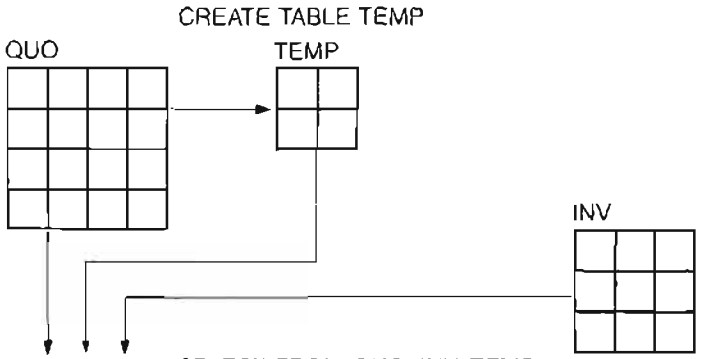
User

```
DROP TABLE TEMP
```

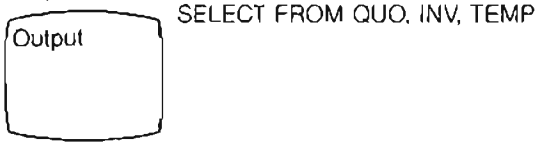
You will see how all of the preceding can be done in a single query later. Again, think of how much more effort would have been required to do this in a non-SQL/DS environment.

Summary

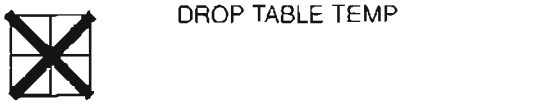
First Step:



Second Step:



Third Step:



Subqueries

When one query is embedded in another, it is called a *subquery*. One reason why SQL is so powerful is that the user can build complex queries by an assembly of many simple queries.

Subqueries are usually evaluated once during the processing of the overall query, and their resulting "answer list" is substituted directly into the main query.

Depending on how the user structures his subquery, it can operate in a different way The following examples show various ways of writing a subquery.

To find those quotations for part 134 in which the price is greater than the minimum price for that part, we could enter:

User

This is the main part of the query

This is the subquery

```
SELECT QUOSUPP,PRICE -
FROM QUO -
WHERE QUOPART = 134 -
AND PRICE > -
    ( SELECT MIN(PRICE) -
      FROM QUO -
      WHERE QUOPART = 134)
```

Display Output

```
QUOSUPP       PRICE
-------  ----------
     51        0.40
     54        0.47
```

Now to find those quotations for part 105 in which the price is greater than the average unit price for all the parts in our inventory, we enter:

User

```
SELECT QUOSUPP,PRICE -
FROM QUO -
WHERE QUOPART = 105 -
AND PRICE > -
    ( SELECT AVG(PRICE) -
      FROM QUO)
```

Display Output

```
QUOSUPP       PRICE
-------  ----------
     61        9.95
     52        7.50
```

Repeating Subqueries

In the previous two examples of subqueries, the subquery was processed once and the resulting value substituted into the main query. The following example shows a subquery that is executed repeatedly, once for each row in the table.

Let's find those suppliers whose price for a part exceeds the average price for that part and show the supplier, part, and price.

User

The 'X' indicates that the subquery should be processed for each selected row in QUO. 'X' is a character string the user creates of his own choosing. (e.g. 'Y', 'Each', etc.)

```
SELECT QUOSUPP,QUOPART,PRICE -
FROM QUO X -
WHERE PRICE > -
      ( SELECT AVG(PRICE) -
        FROM QUO -
        WHERE X.QUOPART = QUOPART )
ORDER BY QUOSUPP
```

Means "find the average price for each part we SELECT from the QUOTATIONS table"

Display Output

```
QUOSUPP   QUOPART       PRICE
-------   -------   ----------
     51       221        0.30
     51       231        0.10
     53       241        0.08
     53       222        0.25
     53       135        0.42
     53       124        1.35
     53       125        0.58
     54       134        0.47
     61       105        9.95
     61       221        0.20
     61       241        0.05
     64       106        4.85
     64       209       19.50
```

Looking over these last three examples, you can see the subtle power of subqueries. Within the basic SELECT..FROM..WHERE structure, the user can concisely direct SQL/DS to do precisely what is needed, just by a slight modification of the query. And it doesn't require learning different commands to perform each of the variations of these special operations.

Repeating Subqueries with Multiple Tables

Let's go back to the example where we wanted to know about those parts on order that were less than twice the balance-on-hand. Our previous solution involved building a temporary table.

We can obtain our answer in another way by using a repeating subquery. This technique is shown below.

Let's see the parts where the total quantity ordered is less than twice this balance.

User

We want to calculate the total orders each time we SELECT another part number from the INV table.

```
SELECT QUOPART,PNAME,QUOSUPP,PRICE,ONHAND,ONORD -
FROM INV, QUO X -
WHERE QUOPART=INVPART -
AND 2 * ONHAND > -
   (SELECT SUM(ONORD) -
    FROM QUO -
    WHERE X.QUOPART = QUOPART) -
ORDER BY QUOPART
```

Display Output

QUOPART	PNAME	QUOSUPP	PRICE	ONHAND	ONORD
106	GEAR	64	4.85	700	0
106	GEAR	61	4.35	700	300
124	BOLT	53	1.35	900	500
124	BOLT	51	1.25	900	400
125	BOLT	51	0.55	1000	0
125	BOLT	53	0.58	1000	0
134	NUT	53	0.38	900	200
134	NUT	54	0.47	900	0
134	NUT	51	0.40	900	500
171	GENERATOR	54	21.75	500	200
172	GENERATOR	57	45.15	400	300
181	WHEEL	64	5.65	1000	400
182	WHEEL	64	7.05	1100	400
207	GEAR	64	29.00	7500	2000
209	CAM	64	19.50	5000	800
209	CAM	54	18.00	5000	200
221	BOLT	51	0.30	65000	10000
221	BOLT	61	0.20	65000	5000
221	BOLT	54	0.10	65000	5000
222	BOLT	61	0.20	125000	10000
222	BOLT	53	0.25	125000	10000
231	NUT	51	0.10	70000	5000
231	NUT	54	0.04	70000	15000
232	NUT	53	0.10	110000	20000
241	WASHER	53	0.08	600000	6000
241	WASHER	61	0.05	600000	4000
241	WASHER	54	0.02	600000	10000
285	WHEEL	57	21.00	35000	0
295	BELT	57	8.50	8500	2400

The technique above and the approach using temporary tables are both valid solutions. An advantage of the SQL language is that one can choose the way that feels most comfortable.

The query above used the QUOTATIONS table twice: once in the main part of the query to get the detail information (supplier number, price, quantity ordered), and then again in the subquery, to get the total ordered (SUM(ONORD))

Now we're going to show how you can conceptually access a single table to get information from two different rows in the same table.

Working on Two Rows at a Time

Let's display the maximum and minimum prices for each part in our data base, and show the corresponding supplier numbers. (The maximum and minimum prices are associated with different suppliers, and therefore are in two different rows of the QUOTATIONS table.)

We're going to imagine that we have two "virtual" copies of the QUOTATIONS table. From one of these we're going to get the maximum price and supplier number for each part, and from the other, we're going to get the minimum price and supplier number.

Just as we did earlier, we'll make up character strings MX and MN and call the two copies of the QUO table QUO MX and QUO MN. This will allow us to distinguish between similarly named fields from the two tables.

User

```
SELECT MX.QUOPART,MAX(MX.PRICE),MX.QUOSUPP,MIN(MN.PRICE),MN.QUOSUPP -
FROM QUO MX, QUO MN -
WHERE MX.QUOPART = MN.QUOPART -
AND MX.PRICE = (SELECT MAX(PRICE) -
                FROM QUO -
                WHERE QUOPART = MX.QUOPART) -
AND MN.PRICE = (SELECT MIN(PRICE) -
                FROM QUO -
                WHERE QUOPART = MN.QUOPART) -
GROUP BY MX.QUOPART,MX.QUOSUPP,MN.QUOSUPP -
ORDER BY MX.QUOPART
```

Display Output

QUOPART	MAX(PRICE)	QUOSUPP	MIN(PRICE)	QUOSUPP
105	9.95	61	7.50	52
106	4.85	64	4.35	61
124	1.35	53	1.25	51
125	0.58	53	0.55	51
134	0.47	54	0.38	53
135	0.42	53	0.39	51
171	21.75	54	21.75	54
172	45.15	57	45.15	57
181	5.65	64	5.65	64
182	7.05	64	7.05	64
205	0.15	52	0.15	52
206	0.15	52	0.15	52
207	29.00	64	29.00	64
209	19.50	64	18.00	54
221	0.30	51	0.10	54
222	0.25	53	0.20	61
231	0.10	51	0.04	54
232	0.10	53	0.10	53
241	0.08	53	0.02	54
285	21.00	57	21.00	57
295	8.50	57	8.50	57

Using this approach, a rather complicated inquiry is simplified by a two-table query.

	SQL/DS can provide your installation with a comprehensive development capability that can span a wide range of users and applications.
An Application Development Tool That is Easy to Use	The underlying relational data structure and concise interactive language permeate all aspects of SQL/DS and provide the DP department with a powerful yet easy-to-use alternative to add to its repertoire of application development tools.
Improves Productivity for the DP Professional and the End User	For the DP department, the productivity gains obtained can be considerable, either by using SQL directly, or by offloading much of the reporting, analysis, and planning aspects of applications to the user department. On the other hand, end-user departments can begin using the data they have long needed. They can gain the benefits from its increased availability, getting more of the right data at the right time.
Allows for Faster Implementation of New Systems	From application specification to production operation, the productivity aspects of SQL/DS affect virtually every phase of development...from the initial discussions with the user department...to building the data base...to validating the results with prototype queries.
Cost Effective Alternative	If you have applications that are constantly changing, have a short life cycle, or involve stand-alone non-integrated data, SQL/DS may offer the best development alternative at a lower cost.
	Whatever your current application development approach is, SQL/DS may be the appropriate addition you need to complement your existing development and end-user strategies in order to satisfy the growing demands of the 80's.

INVENTORY Table

INVPART	PNAME	ONHAND
105	GEAR	0
106	GEAR	700
124	BOLT	900
125	BOLT	1000
134	NUT	900
135	NUT	1000
171	GENERATOR	500
172	GENERATOR	400
181	WHEEL	1000
182	WHEEL	1100
207	GEAR	7500
209	CAM	5000
221	BOLT	65000
222	BOLT	125000
231	NUT	70000
232	NUT	110000
241	WASHER	600000
285	WHEEL	35000
295	BELT	8500

QUOTATIONS Table

QUOSUPP	QUOPART	PRICE	TIME	ONORD
51	124	1.25	5	400
51	125	0.55	5	0
51	134	0.40	5	500
51	135	0.39	5	1000
51	221	0.30	10	10000
51	231	0.10	10	5000
52	105	7.50	10	200
52	205	0.15	20	0
52	206	0.15	20	0
53	124	1.35	3	500
53	125	0.58	3	0
53	134	0.38	3	200
53	135	0.42	3	1000
53	222	0.25	15	10000
53	232	0.10	15	20000
53	241	0.08	15	6000
54	134	0.47	4	0
54	171	21.75	20	200
54	209	18.00	21	200
54	221	0.10	30	5000
54	231	0.04	30	15000
54	241	0.02	30	10000
57	172	45.15	25	300
57	285	21.00	14	0
57	295	8.50	21	2400
61	105	9.95	8	400
61	106	4.35	8	300
61	221	0.20	21	5000
61	222	0.20	21	10000
61	241	0.05	21	4000
64	106	4.85	10	0
64	181	5.65	15	400
64	182	7.05	10	400
64	207	29.00	14	2000
64	209	19.50	7	800

SUPPLIERS Table

SUPSUPP	NAME	ADDRESS
51	DEFECTO PARTS	16 JUSTAMERE LANE, TACOMA WA
52	VESUVIUS, INC.	512 ANCIENT BLVD., POMPEII NY
53	ATLANTIS CO.	8 OCEAN AVE., WASHINGTON DC
54	TITANIC PARTS	32 SINKING ST., ATLANTIC CITY NJ
57	EAGLE HARDWARE	64 TRANQUILITY PLACE, APOLLO MN
61	SKYLAB PARTS	128 ORBIT BLVD., SIDNEY AUSTRALIA
64	KNIGHT LTD.	256 ARTHUR COURT, CAMELOT ENGLAND

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604 USA

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y. 10591 USA

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y. 10601 USA

2-81 Printed in USA G320-6590