SQL/Pro

Overview

As the benefits of information systems technology extend to all areas of an organization, the typical IS department becomes increasingly overloaded by requests for services. The average IS backlog is now estimated at 18-24 months for the average firm and growing. The AS/400 IS department is no exception; in fact, it may very well be higher than average due to its deficiency of rapid application development tools such as modern, object-oriented programming languages and low-cost, easy-to-use CASE tools.

To decrease workloads, managers need to look beyond traditional programming languages for products that can accelerate the building of these systems now. The area in which to find and implement products that can reduce most quickly the workload is information retrieval. A large portion of IS job orders consist of requests for reports. This includes the daily barrage of both ad hoc queries and sophisticated management reports-along with everything in between.

There are a wide variety of products designed to easily query and create reports from the AS/400's relational database. Of the many tools available, Structured Query Language (SQL) is fast becoming the standard information retrieval tool-not just for the AS/400, but for all other commercial platforms as well, SQL does more than just retrieve information-it is a full relational database language. There are excellent reasons for this popularity. SQL is a portable, simple-to-use, robust language.

Although this report concentrates on the information retrieval aspect of SQL, since that is its most common use, keep in mind that its power extends to defining, modifying, and controlling your relational database.

But what is SQL, in practical terms? SQL is a very logical, English-like language you use to write a 'sentence' (statement) that tells the computer what relational database operation you want it to perform. To give you a taste of SQL, take a look at the following SQL statement, which will list the customer name (name) and balance due (baldue) for all customers in the customer file (custmast):

```
SELECT name, baldue FROM custmast
```

Granted, that is a simple request. Let's look at a real-life request. The sales manager buzzes you and asks if you can print a report showing all customers who have purchased a widget in the last 90 days, and he needs it as soon as possible. The following six-line statement does it all. In sjust six lines, SQL joins three files, the custom file (custmast), the invoice header file (invhdr) and the invoice detail file (invdtl), by their keys and then selects the invoices that have widget orders within the last 90 days.

```
SELECT cstno, name, phone, orddat, qty
            FROM custmast, invhdr, invdtl
            WHERE custmast.cstno = invhdr.cstno
               AND invhdr.invno = invdtl.invno
              AND DAYS (CURRENT DATE) – DAYS(invhdr.orddat) <= 90
              AND orditm = ‘WIDGET’
```

AS you can see, SQL’s power is simple to access. No programs to write; no compiles to do. Just make your request and see the results. You’ll learn more about SQL’s capabilities in the tutorial section of this report.

Origins

IBM developed SQL as part of the System R project that it sponsored at its San Jose facility starting in the early 1970s. System R was a research project set up to investigate issues involved in the implementation of relational data management concepts on real-world, general purpose computer. About a year before System R project started up, Dr Codd (the father of relational database) wrote the first paper defining the relational data model. The hardware targeted by the System R effort was mainframe gear. The early 1970s was the pre-PC era of computing and there was no ‘midrange’ computing going on. The System/3 line, our pioneering midrange system, was actually a small machine during an era in which there were only big machines or small machines. The System/3 had neither the architecture nor the sheer size to support a relational database management system.

Among the problems tackled by the System R group, one of the more important was defining a straightforward way of precisely specifying the queries that users would want to make against a relational database. The solution adopted by the System R group was to design a new language called SQL.

In 1981 IBM announced its first commercial SQL-based product, SQL/DS. Other large vendors such as Oracle and Relational Technology announced SQL-based relation database management systems in the early 1980s. In 1988 IBM introduced SQL for the AS/400. The popularity of SQL has grown at such a rapid pace that currently hundreds of SQL or SQL-based database management systems are now available on all brands, types and sizes of computers. While SQL’s early implementations were often slow and incomplete, the language is now entrusted with ‘mission-critical’ information management and data processing tasks in major organizations. IBM is seriously promoting SQL by continuing to enhance the language and by designating it as one of the important standards comprising Systems Application Architecture(SAA).

When Common is Good

While different dialects of SQL appear on various machines, the variations are minute, and the language is essentially the same from system to system. This offers tremendous benefits to IS departments that add SQL to their host of languages. It is likely that both your IS staff and non-technical users may already know SQL. In addition, the SQL skills you acquire will be transferable to other computer systems. And, since every commercial platform has an SQL product, if your IS department is ever charged with switching platforms, all of your SQL statements will run on the new system with little or no change! Try that with another language or Query/400!

At One with the Database

One of the rules of relational database system is that it must be able to use one high-level language to structure, query, and change the information in the database. Although, in theory, any number of database languages could fit this bill, SQL is considered **the** relational database management language. The AS/400 is a relational database machine and SQL is the relational database language for the AS/400.

SQL is different from other high level languages like RPG, COBOL or C. It looks at a file, or a set of joined files, and performs an operation against the whole file(s), or a selected subset. You tell SQL what file(s) to look at, what records to select, and what to do with them. SQL works well for jobs such as creating a report of sales broken down by salesperson, branch and department; deleting all item records of an item master file that have had no sales in the previous year; and listing average, minimum and maximum values for customer balances. The uses are endless.

Certainly the IS department could use SQL to perform many maintenance actions against the database, and use it to create many periodic reports. But the beauty of SQL is really apparent when it is in the hands of non-technical users. Because of its ease of use, managers and other users can create their own ad-hoc and permanent reports right from their desks, offloading the work from IS.

A Technical Overview

All SQL operations are expressed as SQL statements. A statement is like a sentence that uses keywords, identifiers, operators, and functions to describe the operation to be performed.

- Keywords are words that have special meaning to SQL
- Identifiers are names of files, fields, and other database objects.
- Operators and functions perform mathematical and computational capabilities.

For clarity, we'll show all keyword in uppercase and other values in lowercase. However, SQL on the AS/400 is not case sensitive.

SQL is used for data manipulation (retrival and modification), data definition, and data administration. Data manipulation can be deivided into two types: data retrieval, which is finding the data; and data modification, which is adding, removing, or changing the data.

Data retrieval operations (often called queries) are the most widely used of SQL statements – especially on the AS/400. Typically, AS/400 installations with SQL create, modify, and administer their database through other hig-level languages such as CL and RPG. But, SQL queries are heavily used to search the database, fetch information, and display or print the results.

The SELECT keyword is used to perform a query. A staement using the SELECT keyword is known as a SELECT statement. In many ways the SELECT statement is the real heart of SQL. Once you become familiar with its syntax, you'll be amazed at what it can do for you.

The SELECT statement begins with this skeleton:

```
SELECT list_of_fields
        FROM file(s)
        WHERE search_conditions
```

Here's a simple SELECT statement that will list all records of the customer file, where the state is California:

```
SELECT *
        FROM custmast
        WHERE state = 'CA'
```

It would produce a report similar to this:

```
cstno  name                          addr                          city              state    ytdpur
------- ----------------------------  ----------------------------  ----------------  -------  --------
9010   ABC Home Loan                 5451 Elm                      Carlsbad          CA       60,000
0542   California Mortage            412 Main St                   Bellflower        CA       45,000
0125   El Camino Title               12 First Ave                  San Diego         CA       30,000
```

As you can see, the statement is very simple. The SELECT list identifies the fields you want to retrieve. The '*' is shorthand for list all fields in the record (that's about the most cryptic thing you'll ever see on an SQL statement). The FROM list specifies what files the fields are in. The WHERE keyword forms the calculations and expressions to subset your data.

Where do you key the statement? SQL products provide an interactive interface where you enter the statement. Once the statement is entered you can execute it interactively and wait for the report to display on the screen.

You also could have specified just which fields you wanted to include in the report, and you could even have added computed fields. For example, let's select just the company name, the YTD total purchases and compute the yearly projection (three times the YTD total for this example). While we're at it, let's sort it by the YTD total purchases.

```
SELECT name, ytdpur, ytdpur * 3
        FROM custmast
        WHERE state = 'CA'
        ORDER BY ytdpur
```

Again, very logical. We've added the ORDER BY clause that tells SQL how to sort the data. This is the result:

```
name                                     ytdpur           ytdpur * 3
--------------------------------------   ---------------  ---------------
El Camino Title                          30,000           90,000
California Mortgage                      45,000           135,000
ABC Home Loan                            60,000           180,000
```

Of course, this unattractive report would not suit business needs beyond ad hoc queries. Fortunately, most AS/400 SQL products provide a report formatter that enables you to customize the output to meet your needs. Generally, these report formatters can do ordering of the columns, positioning of information, meaningful column and page headings, control group summary operations, field edits, headers and footers, and so on.

Most SQL products also allow reports to be sent to the printer or to a database file. Some products allow you to submit your job to batch.

It is beyond the scope of this short report to show you everthing that the SELECT statement can do. Instead, we'll illustrate some of its power with a handful of examples.

Select all customer records whose contact name begins with 'Mc' or 'Mac', either upper-case or lower-case, and whose area code is 619, 714, or 818:

```
SELECT cont, cstno
        FROM custmast
        WHERE (TRANSLATE(cont) LIKE 'MC%'
          or TRANSLATE(cont) LIKE 'MAC%')
          and arcode IN('619', '714', '818')
        ORDER BY cont
```

Output:

```
cont           cstno
-------------  -------
MacGreer       2344
McDonald       9098
Mcfadden       0334
```

The TRANSLATE function converts the contact name into upper case so we can compare for MC or MAC. The LIKE predicate specifies that it selects only records whose value for the cont field begins with 'MC' or 'MAC' (the % in compare value is like an * used in a generic name). The IN predicate compares a value with a set of values. In this case, if the area code is either 619, 714, or 818, the record is selected.

SQL provides aggregate functions to compute summary values. You apply aggregates to sets of records, to all the records in a file, to just those records specified by a WHERE clause, or to groups of records set up in the GROUP BY clause. No matter how you structure the sets, you get a single value for each set of records.

The aggregate functions provide sums, averages, counts, maximum values, minimum values, and many more including trigonometry functions use in forecasting and much more. They are powerful functions, but as simple as anything else to do in SQL. For example, the next statement will report the number of customers, and the minimum, maximum, and average balance due amounts. COUNT (*), MIN, MAX, and AVG are all aggregate functions.

```
SELECT COUNT(*), MIN(baldue), MAX(baldue), AVG(baldue)
      FROM custmast
```

The output would be:

```
COUNT(*)  MIN(value)    MAX(baldue) AVG(baldue)
--------- ----------    ----------- ----------
452       50            289         121
```

The GROUP BY clause provides a way to group records into Groups, working with aggregate functions to produce summary values for each group. Let's enhance the previous SQL statement to report the minimum, maximum, and average balance for customers, grouped by each state:

```
    SELECT state, COUNT(*), MIN(baldue), MAX(baldue), AVG(baldue)
        FROM custmast
        GROUP BY state
```

This would produce a listing like this:

```
state COUNT(*)     MIN(baldue)  MAX(baldue) AVG(baldue)
----- --------     ----------   ----------  ----------
AZ    14           50           189         78
CA    63           56           250         91
…
```

Here's a more sophisticated request. You would like to know the minimum and maximum salaries for each group of employees with the same job code, but only for groups with more than one employee and with a maximum salary greater than or equal to $30,000:

```
    SELECT jobtyp, MIN(salary), MAX(salary)
        FROM employee
        GROUP BY jobtyp
        HAVING count(*) > 1 and MAX(salary) >= 30000
```

Output would be:

```
jobtyp          MIN(salary)        MAX(salary)
------          ----------         ----------
PROG            23,000             42,000
MGR             31,000             76,000
…
```

We limit the employee groups reported with the HAVING clause. This clause looks at the groups the GROUP BY created and checks if the group meets the requirements of the expressions.

We've only scratched the surface of selecting, ordering, and grouping. There is still an incredible amount of functionality that we have not touched upon. But let's move on to joining, where the real relational power of SQL is found.

Since relational databases follow the rules of normalization, the related information is broken up into multiple files. For instance, an A/R system might have, at a minimum, a customer master file, an invoice header file and an invoice detail file. For most reporting needs, you'll need to join the information from the related files together.

SQL allows joining and accomplishes it implicitly through the WHERE clause of SELECT statement. Like everything else in SQL, it's a simple clause that specifies which fields of each file tie the files together. In our A/R scenario, the customer number would tie the customer master and invoice files together.

Let's take a look at how you would create a report that would list each invoice for customers in New York, showing the customer name, alphabetized by customer name. A join is necessary since the invoice file, normalized to reduce redundancy, does not carry the customer name:

```
SELECT name, invno, invamt
     FROM custmast, invhdr
     WHERE custmast.cstno=invhdr.cstno
          And state = 'NY'
     ORDER BY name
```

The FROM clause identifies the names of the files that are to be joined. The WHERE clause identifies not only which records to select from file custmast (NY state only), but also which fields from each file to use for the joining process. In our example, field cstno from file custmast and field cstno from file invhr are used to join records from the two files.

You can even do a self-join, joining a file to itself. Why would you want to do such a thing? AS one example, you can use a self-join to find duplicate records in a database. Here, we check for duplicates by requesting that all records with the same phone number be listed:

```
SELECT c1.cstno, c1.name, c2.cstno, c2.name
     FROM custmast c1, custmast c2
     WHERE c1.phone = c2.phone
     And c1.cstno <> c2.cstno
```

The custmast file appears to SQL as two files. We've accomplished this by assigning different aliases – c1 and c2 – to the custmast file in the FROM clause. The aliases are also used to qualify the field names in the rest of the query. The WHERE statement says to report any records which have equal phone numbers, but are not the same customer number. The final line checks that the matching records are not the same customer number (if this is not done, every customer's phone number would match its own phone number). There is, of course, much more to joining. In fact, you may accomplish virtually any file joins that are required.

You may find that you have to nest a SELECT statement within a SELECT statement to produce the desired result. This is known as a subquery. A statement that includes a subquery operates on records from one file based upon its evaluation of the subquery's SELECT statement. The subquery can refer to the same file as the outer query, or to a different file.

Subquerires sidestep the need to run one query to get a result and then run another query that uses that result. For example, what if you wanted to produce a report that showed all the customers whose balance was greater than the average balance for all customers? This subquery provides the results:

```
SELECT name, baldue
        FROM custmast
        WHERE baldue >
                (SELECT avg(baldue)
                        FROM custmast)
```

In a subquery, the inside SELECT statement is processed first. It results in a value that is passed to the outer SELECT statement. The outer SELECT statement than processes, substituting the returned value of the inner SELECT. In our example, if the inner SELECT produces a value of $243, the WHERE clause becomes ‘WHERE baldue> $243.’ Many other variations of subqueries can be performed.

SQL provides what is called a UNION. With a UNION you derive a result table from combining two other result tables. When UNION ALL is specified, the result consists of all records from both tables. When just UNION is specified, without ALL, duplicate records are omitted.

What good is this? Let’s say you wanted to list all employee numbers from the employee file that are specified as managers (jobtyp = ‘MGR’) or are enrolled in the manager training program (a record in the program file with a prgtyp of ‘MGRTRN’):

```
SELECT empno
        FROM employee
        WHERE jobtyp = ‘MGR’
UNION
SELECT empno
        FROM program
        WHERE prgtyp = ‘MGRTRN
```

All the managers are selected from the employee file, and employees enrolled in the manager training program are selected from the program file. Both are placed together into one table and the result would be a list of the employee numbers selected.

More Than Just Queries

Although the most common usage of SQL is to perform data retrieval, SQL does so much more. Since SQL is the AS/400’s relational database language, you can do any relational database function with SQL. You can create and delete physical and logical files; grant and revoke authority; add, update and delete records; and commit and rollback transactions.

The statements for these functions (CREATE, DROP, GRANT, REVOKE, INSERT, UPDATE, DELETE, COMMIT, and ROLLBACK) follow the same English-like format, as does SELECT. Only each has its own set of clauses. INSERT, UPDATE, and DELETE use the same WHERE clause as does SELECT.

While this report is concentrating mostly on the reporting of data, examples of SQL’s other abilities are in order. The statement below shows off the types of sophisticated data manipulation that SQL can perform. The DELETE statement will delete any record in itemmast (item master file) that is not used in invdtl (invoice detail file). In other words, it deletes obsolete or otherwise unused item master records.

```
DELETE FROM itemmast a
        WHERE a.itemno NOT IN
                (SELECT b.itemno FROM invdtl b)
```

First, the subquery builds a list of item numbers used in file invdtl. The main query deletes any records in itemmast whose item nbumber is not found in the subquery list. Aliases ‘a’ and ‘b’ are used to identify the files.

Here’s another, more complex, example: It’s a new year and time to adjust the prices of your company’s products. You need to reduce by 15% the prices of all items that did not sell over 1000 units last year. The item price is in item master file and you’ll have to count up the number of units sold in the prior year by reading the invoice detail file. You could write a program to count each invoice’s charges and update the item cost field in the invoice header record. Or you could use SQL. The following UPDATE statement will easily accomplish the task:

```
UPDATE item
        SET itmprc = itmprc * .85
                WHERE 1001 >
                        (SELECT sum(invdtl.itmqty)
                                FROM invdtl,
                                WHERE invdtl.itmeno = item.itemno)
```

The subselect adds the quantities of the invoice line items that have the item number that matches the item the UPDATE is currently processing. This number becomes the second half of the WHERE clause. The SET clause multiplies the item price by .85%(a 15% reduction) and sets the field to the calculated sum.

SQL can perform any host-allowed relational database function. There is no other language on the AS/400 that offers so much power, with so much simplicity.

SQL vs. Query/400

Perhaps you have IBM’s Query/400 or are thinking of purchasing Query/400. Why should you consider using SQL instead? QUERY/400 also selects, joins, sorts, groups, and otherwise reports your data. So why SQL? Because SQL possesses many important advantages over IBM’s Query/400 tool.

- SQL can perform even more powerful queries than Query/400.
- Although Query/400 has extensive reporting capabilities, SQL’s report formatting functions(the Query Manager in IBM’s SQL/400 and third-party products’ formatters) offer comprehensive reporting as well.
- Query/400 joining is limited.
- With Query/400 you cannot use parentheses in AND/Ors for joining and record selection.
- Query/400 does not have subquery support.
- SQL is a relational database language – Query/400 is not. As such, SQL processes all of the functions required for a relational database. You can create files and views; add, update, and

delete records; and perform other database functions such as the granting of authority and committing and rolling back transactions.

- Some SQL products allow you to embed SQL in high-level languages. This means that you can call SQL statements directly from within a program, such as RPG.
- SQL is a universal language, supported by every hardware platform and operating system. Your knowledge of the language is portable. Plus, because it is so prevalent on other systems, many of your users may already be familiar with SQL and can begin using it with little or no training. Since Query/400 is a proprietary AS/400 product, it is not a widely used language.
- The English-like language structure is simple enough to put directly in the hands of users to create their own reports. (But don't worry, anyone can be secured out of the data manipulation statements.)
- Many third-party software vendors are beginning to incorporate SQL interfaces in their products.
- SQL fits more into IBM's strategic plans than does Query/400, which is not SAA compatible.
- There is no choice between Query/400 and SQL. By far, SQL offers the most power and best ease-of-use. If you already have Query/400, you should still consider SQL. It dramatically assists in reducing your IS department's workload.