

Visual RDBMS Application Development

Ian J. Curington
Advanced Visual Systems, Inc.

Abstract

A library of modules is described using SQL communication with remote data base servers. Application networks using these modules are shown such as GIS map attribute picking and multi-column numerical data.

Introduction

AVS/SQL is a collection of AVS modules that directly connect, query, and extract data from a local or a remote database server. The modules provide an interface between AVS and a relational data base management system (RDBMS) such as ORACLE, INGRES, SYBASE, or INFORMIX. AVS/SQL enables one to combine the sophisticated management facilities available within a RDBMS with AVS's visual tools.

Extending the technology for embedded SQL for client-server database application development, the AVS/SQL system combines SQL constructs through the use of the AVS visual programming environment. The modular design allows high reuse of software components, and allows rapid prototyping and testing by the visual representations of module relationships. The AVS/SQL module suite will operate seamlessly with other non-SQL AVS modules, including those for numerical analysis, data visualization, and 3D graphic presentation and animation. A new class of applications may be constructed employing dynamic 2D or 3D visual object representations of database objects.

Remote Data Access through SQL

The fundamental property of an RDBMS is that it is accessed through commands of the Structured Query Language (SQL). SQL deals with issues such as inserting and retrieving items of data. These actions are modified by operations such as filtration, sorting and comparison. SQL is important because it enables a corporation to organize and maintain complicated data structures; and it allows people to make different uses of a central data store.

However the ability to use SQL is only one reason for using an RDBMS. For example, RDBMS systems usually include control over read/write access for particular users. Data may be distributed over a number of RDBMS servers on a network. Many users can access the same data and RDBMS systems also provide sophisticated "locking" mechanisms to handle situations where more than one user may try to modify the same items of data.

It is quite possible that AVS users would wish to store their data within an RDBMS and AVS/SQL can readily be extended to pass data in this direction as well as retrieving it from a database.

AVS as a front-end to an RDBMS

AVS/SQL provides the communication between the RDBMS and AVS so users can have visual display and interaction with corporate data. The ability to use AVS in this way is very important for three reasons:

- AVS offers a wealth of methods for displaying, manipulating and processing data. There is a great need for visual display and interaction with complicated data from an RDBMS: few users have the time to learn SQL and to explore their data with its low level commands.
- AVS's modular organization means that basic building blocks of AVS/SQL can be assembled rapidly to provide new applications for an end user. This is particularly important since it enables a corporation to evolve in the way it uses its data without incurring a prohibitive cost in the development of new systems.
- An organization can develop new code and use this to enhance the functionality provided by AVS. This is particularly the case with AVS/SQL, because it comes complete with source code.

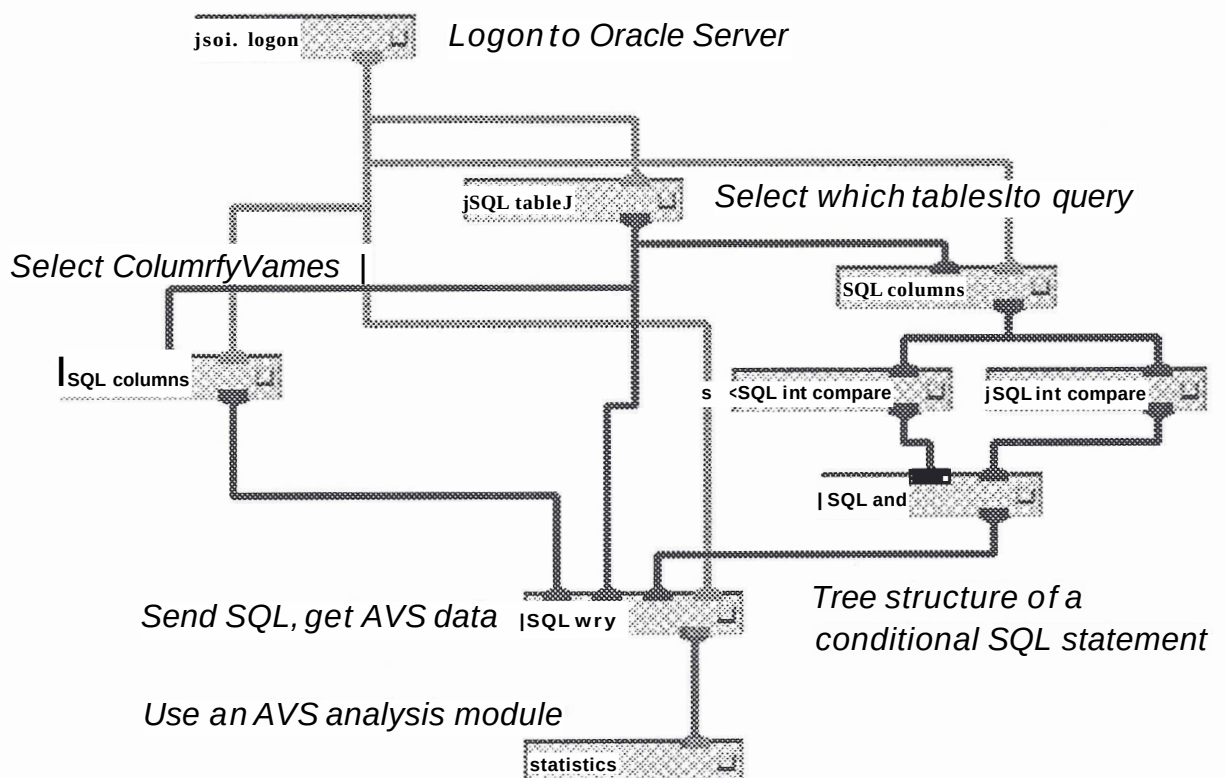


Figure 1: Simple network using the AVS/SQL Modules

Methodology

The AVS/SQL tool kit provides a fully functional collection of AVS modules that interface to an Oracle RDBMS database system. As reusable software modules, they let a user directly connect, query, and extract data from either a local or a remote networked database server.

The AVS/SQL module suite makes use of embedded SQL communication streams, and is developed using the Oracle Pro*C preprocessor development system. The module tool kit is available in source code as well as pre-compiled binary, so that application development teams can take this as a working framework and extend the functionality on top of the core tools provided. A broad range of operations are covered by the pre-compiled modules, so customization of the source code may not be needed in some cases.

The visual object library includes Oracle logon validation, table list browser management, extraction and presentation of columns, SQL statement assembly with error checking, and multiple SQL query objects returning integer or floating point ID or 2D array fields.

The AVS/SQL example demo suite includes two fully working and tested example applications:

- GIS map with pickable markers, picking causes SQL query returning a graph.
- Abstract "City Scape" display, where picking a cell returns attached population statistics attribute data.

In addition several commercial applications have been built using the module tools.

Client-Server Software Architecture

No database is complete without some way to store or retrieve data from the database. The class of tools currently available for the front end or client operations of data access are significantly extended by the use of AVS, using the AVS/SQL interface toolkit. In a client-server environment, a fully developed application must work within a mixed network, with limited bandwidth available between the client and server. Any query to the database should involve some level of data abstraction and selection, such that the information requiring transfer over a wide area network can be minimized.

With the increased complexity of such data structures and access methods, building new front end applications is a challenging task. For full control over the application development design, C or C++ is ideal due to the open nature of these languages. However, they require the programmer to be fully responsible for a very large amount of the application algorithm design and complexity. This is compounded when sophisticated 3D graphics programming is also required in the client application. It is rare to find both RDBMS and 3D Graphics skills in the same development team.

Client-Server with Visual Programming

An alternative approach is to use a modern visual programming environment such as AVS. The AVS network editor shows major pieces of the application as graphical objects, and allows the control,

notification, and data connections between the graphical objects to be specified with a mouse click. The resulting application is shown in a visual schematic form, that can be edited, and the program flow animated.

In addition, a rich suite of 3D data representation and analysis tools are available as visual objects, ranging from graphs and charts, image analysis, contouring, surface terrain, molecular model display, volume rendering, and animation. The resulting high performance graphics-rich client application may itself take advantage of distributed network resources, multi-processor architectures, and parallelism.

Visual Tool for Networked Computation

In most client-server wide area or local network designs, some portion of the program logic is shifted between the client workstation and the remote server. The AVS/SQL system allows for any module in the application to be placed on any host in your network without recompilation, using the AVS visual network editor programming environment. In a flat file environment without a structured database, files or records are returned as requested by the client with little or no interpretation. By contrast in the case of an RDBMS application, the client workstation requests information from the file server, with the server potentially resolving a very complex query.

The client-server model can greatly reduce network traffic because the only data traveling through the network lines are the client's request and the server's reply. In a conventional (flat file) application, the workstation must sift through a lot of data to decide what to accept and what to reject. All that data must use the network bandwidth, even if only a fraction is ultimately required. On a wide area network running at relatively slow line speeds, this can make a conventional application unworkably slow.

Structured Query Language (SQL)

Structured Query Language, or SQL, has become arguably the most popular data base query language. It is available in some form from most database vendors on most computer platforms. SQL is often implemented through some host language that serves as a vehicle for embedding SQL calls. For instance, embedded SQL language systems are in use with ADA, Cobol, Pascal, PL/1, FORTRAN, C, Perl, etc.

An SQL database consists of a group of data tables. Each of these tables contain records and columns. It is also possible to form relations between tables by joining them together logically to form views. A view is a logical representation of the data stored in one or more tables. Similar to a table, it consists of rows and columns. In SQL, a SELECT statement retrieves data. After the SELECT keyword is a list of field names selected from those available using the FROM clause. The AVS/SQL system automates this step by providing the available names in choice browsers. The WHERE clause specifies the restrictions we require for the view.

The AVS/SQL toolkit contain modules supporting the robust construction of SQL statements with interactive selection from the stored names of tables, fields, and columns. Using SQL, a programmer can maintain a more abstract view of the database, without worrying about its physical representation, file formats, etc. The more complex the database the more useful this abstraction becomes. The physical data may reside on a local server, a server 100 km away, the workstation internal hard disk, or some combination of all three. The client application (and thus the client application developer) does not need

to know where the data is, because the database server handles it.

SQL Interoperability or Performance?

It remains, however, a challenge to find portable SQL systems between database vendors, as each has tended to add vendor specific extensions, much like FORTRAN language syntax during the 1970's. Ten years ago, at the time of the first definitions of SQL, it was used within the same vendor environment, and often on the same computer system. Today, interoperability of SQL is severely stretched by competing requirements from hardware vendors, RDBMS core systems vendors, and layered 4GL and application software.

In 1986, the first standard was set by ANSI, and ISO, revised in 1989 in the form of "SQL-89". This has not provided the hoped interoperability, as it lacked essential tools such as scrolling cursors, union joins, and temporary tables. In addition, RDBMS systems vendors have created extensions for performance reasons, such as for "Cluster Objects", with incompatible control mechanisms. This situation has contributed to the growth of "Middleware" environments, that attempt to find the common ground between vendor specific implementations of SQL.

Embedded SQL in a Visual Programming Environment

The idea of mixing visual programming, C program structures, and SQL may seem counter intuitive, but each has complimentary characteristics. The compiled C code, (used to develop the AVS/SQL modules) excels as a general purpose application development language, but has almost no intrinsic facility for the type of data abstraction that SQL allows. On the other hand, SQL is of little use without some general purpose host language, for looping, memory allocation, control flow, and the control of workstation resources. The AVS module architecture allows the C and embedded SQL sections to be kept small and general purpose, and mixed with other parts of the application in a powerful, visual and rapid prototyping system.

Prepackaged Module Suite

The AVS/SQL system is a fully functional collection of reusable modules that let you directly connect, query and extract data from what could be your most valuable corporate asset - your corporate database. Each module is self contained, and may be interconnected in a wide variety of ways with other modules in the AVS/SQL system, other standard AVS modules, or custom modules written using the AVS "Module Generator" source code template system. Each module presents standard input and output ports in the visual network editor, and can be controlled using script files or dynamically with mouse control.

The core section of any embedded SQL client application is the logon process. The SQL logon module encapsulates this process in a single reusable module, presenting a validation output flag to any other module requiring access to the SQL communication system. Similarly, table and column name retrieval and SQL SELECT queries each expect the validation as input, and may be connected in various configurations to create a custom application.

To allow client application developers full control over the behavior of the prepackaged modules, the AVS/SQL system modules are supplied in source code, with the intention that they will be used as a

template framework for further application specific modules.

AVS/SQL Module Descriptions

The AVS/SQL system contains a series of SQL specific modules, organized into an AVS library. These are in addition to over 250 standard AVS modules, and hundreds more available from other sources. The following modules are provided in source code, allowing many more variations in a custom application:

Primary Communication Modules

• SQL logon	Validate the user session, establish communication
• SQL tables	Present list of available user tables as choice list
• SQL columns	Present list of available columns within a table
• SQL date	Extracts date lists into an AVS field
• SQL query	Extract data into an AVS field based on SQL clause
• SQL simple query	Simple query engine with no input qualification
• SQL logon query	Combined logon and query for multiple servers

SQL Clause Assembly Subsidiary modules

• SQL and	Build an SQL-style AND phrase into a statement
• SQL or	Build an SQL-style OR phrase into a statement
• SQL not	Build an SQL-style NOT phrase into a statement
• SQL int compare	Build a numerical comparison (=,>,<,!>,>=,<=)
• SQL in	Build a Test for inclusion within a set
• SQL concatenate	Concatenate Table or Column list strings
• SQL make list	Assemble an SQL-style list string from components

Auxiliary Visual Representation and Interaction Modules

• City Scape	Pickable 3D Bar chart system for query generation
• Marker Select	Pickable 2D markers over spatial image data
• Marker Draw	Marker drawing, scan conversion, and filling

Sample Application - River Flow Data

Sample data has been provided by the Institute of Hydrology (UK) for use with AVS/SQL. One table consists of a list of sampling sites with an identifier and a grid reference for each site. The second table consists of river flow rates recorded on particular days together with the date of the sample and the identifier of the site.

One obvious application for an end user would be to mark sample points on a map and pick one of them.

The flow rates for this sample point would be extracted and displayed as a graph plotted against time. The user might also filter the data so that it is extracted between a range of dates.

The AVS/SQL demonstration application was built in just a morning. There are two main aspects to the application:

- AVS/SQL gets a list of sites and their grid references. The AVS Image Viewer displays these as spots on a map. The user can pick a site whereupon it is highlighted. The Image Viewer immediately makes available valuable features such as the ability to pan and zoom the map image.
- AVS/SQL takes paired flow rates and dates from the database. These are sorted according to date and are filtered according to the site identifier and a date range. The site identifier is supplied by the pick on the map while the user controls the date range with two dials. The results are plotted in the Graph Viewer and like the image viewer this provides new facilities for the user: Postscript output, control over plotting styles, etc. Other output facilities may also be used, such as a tabular display with "print field", or summarized with the "statistics" module.

Additional control issues are also addressed within the sample application. There is a login prompt which also enables the user to point towards the particular RDBMS server on which the data are held. There are also choice devices for selecting the tables in which data are stored and for choosing the columns that contain the data. Any of these facilities may be automated.

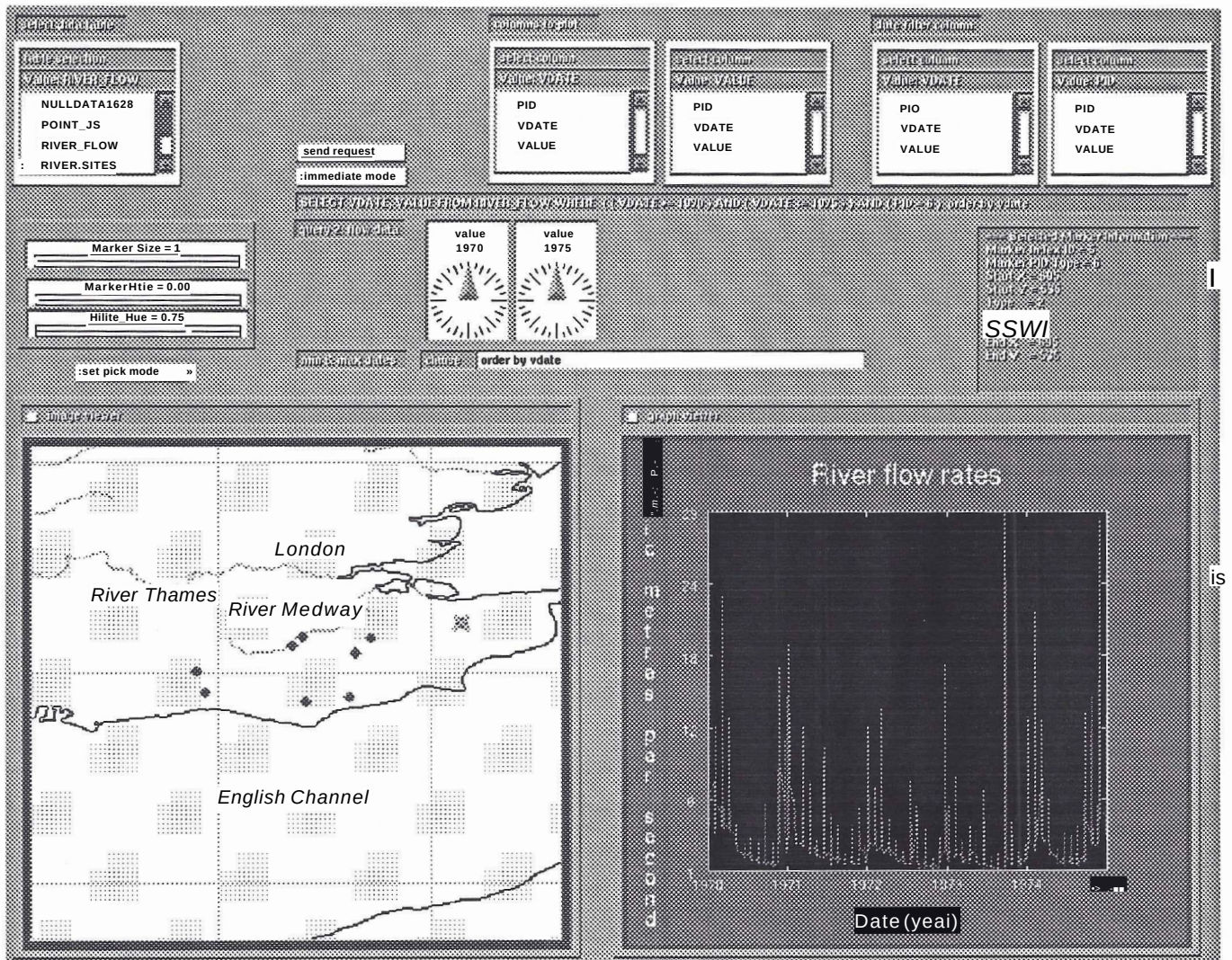


Figure 2: Application interface for interactive flow rate query.

Sample Application - Financial Data Mining

A sample application showing financial data display and mining is also included as a framework application example, using the "City Scape" module. This displays a bar chart arising from a 2D grid and is suitable for data that are classified according to two criteria. The bar chart displays a summary value over each underlying group per bar - usually the mean or a count; it allows considerable control over the axes and other aspects of the image. The summary data for the bar chart are obtained by one SQL command. The user can pick a bar and the I,J locations of the bar form the basis for a second query which can produce more detailed summary statistics or a tabulation of the data for the specified group.

Clearly AVS provides other methods for display and visual interaction with data. In particular it would be possible to store 3D geometric objects in a database and to pick them from within the geometry viewer in order to obtain additional information.

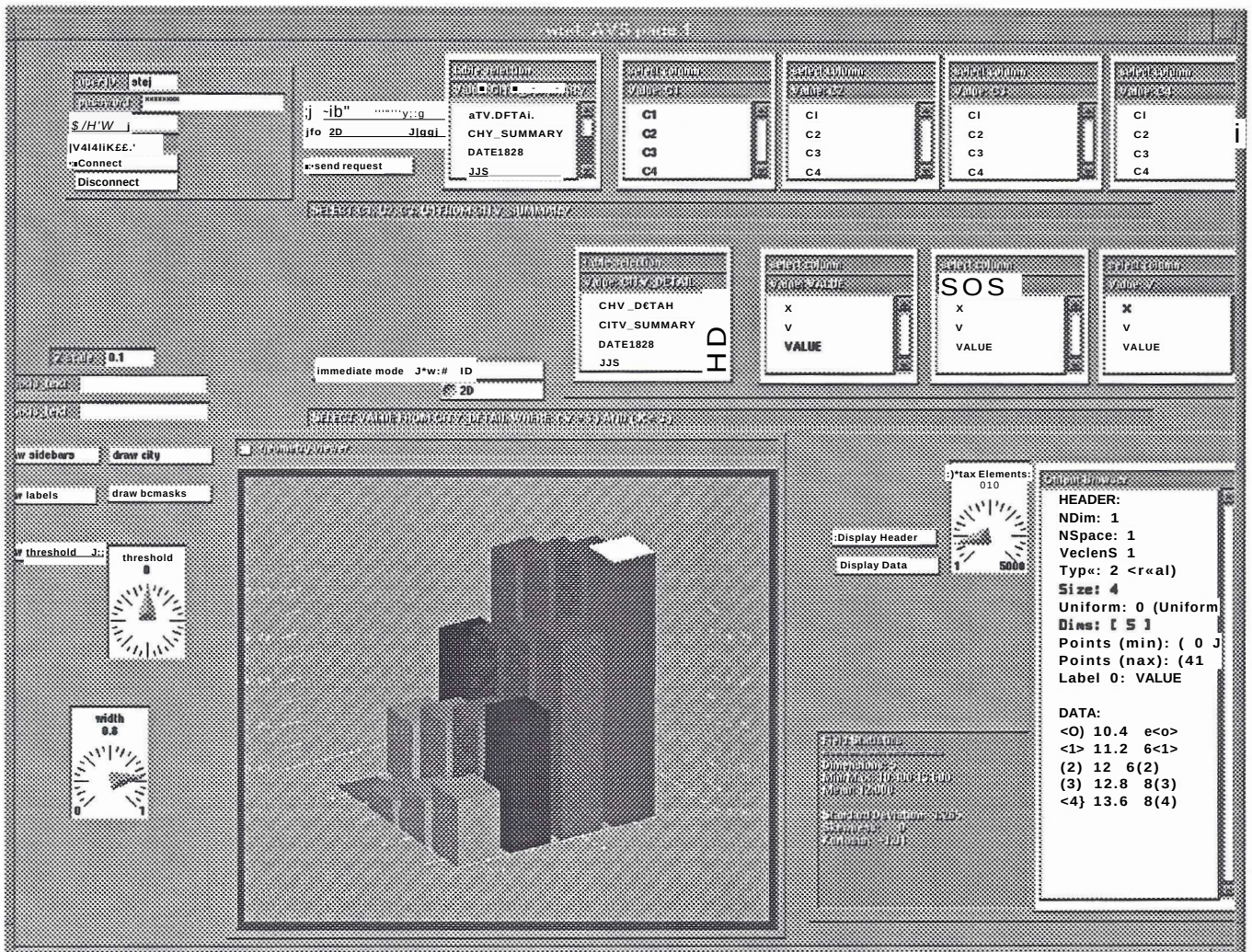


Figure 3: Application interface for city scape data mining.

Sample Application - GIS Data

A further use of the the AVS/SQL kit has been applied to creating a geographical information system (GIS). Spatial data representing the elevation of gridded locations are stored in a database. Sections of a large map are retrieved from the database and displayed in a 3D interactive window as a perspective terrain elevation.

By interactively selecting a subregion of a 2D map by mouse rubber band actions, a query is generated to retrieve the elevation data for this region. By using this data to construct a 3D sheet object, the region can be viewed as a solid 3D land-form, with perspective, shading, and lighting. Selecting any other area of the original map will immediately send a new query, resulting in the display of the new region in the 3D window.

Implementation

AVS/SQL is a toolkit comprising source code and as such it is readily extensible. It was impossible to

provide a shrink wrapped application because of the infinite variety of database structures and the diversity of application requirements.

The number of platforms that can be envisaged is also great: it approaches the product of the number of operating systems and the number of RDBMS products. The potential complexity is increased because of the use of multiple languages. The ORACLE implementation uses "C" and "SQL" in a mixed language environment, where SQL statements are interleaved within "C" code for use with the ORACLE Pro*C compiler.

AVS/SQL provided modules allow the user to connect to a database and to extract information from it. The primary forms of extraction are from columns of data within database tables into AVS fields. Currently AVS/SQL handles database data types of "number" and "date". Numbers can be directed into integer or real fields which can be 1D or 2D. (1D fields preserve column names but are unsuitable for X-Y plots in the Graph Viewer.) Dates can be extracted in one of two formats and can include time information. There are two other groups of data types which users might eventually need to extract into AVS: "long" and other forms for holding vast quantities of amorphous information; and "string" with related containers for textual information. Modules for inserting or modifying data have not been provided as yet, but these are part of the AVS/SQL future upgrade designs.

Null values can exist within databases and in AVS/Express, but not currently within AVS5. Therefore the extraction modules will substitute a numerical value (under user's control) for a null data item.

There are also secondary forms of extraction from the database which greatly improve ease of use. Several modules are included that present a list of table names and a list of column names within a table. These may be connected to choice list interfaces, for automated query selection, with their output connected to other modules to derive application specific query functions.

An SQL command to extract data from a table could be something like:

```
SELECT pressure, temperature FROM results WHERE time > 10 ORDER BY time;
```

which takes two columns from the results table and organizes them by a third (time) so that times of 10 or less are excluded and the output is sorted according to time. AVS/SQL modules issue these commands to the RDBMS. The user can type the entire query string in a module parameter. However, the supporting modules are helpful in providing necessary table and column names. There are also a wealth of string manipulation modules which aid in the assembly of the command. For example one module helps to build a clause for comparing data items in a column with a numerical value. It takes in a column name (in this case "time"), and provides an integer dial and choice device for the type of comparison (e.g. >, >=, !=) and outputs a string such as "time > 10".

The modules are able to parse the SQL string to a limited extent to obtain administrative information. For example they can determine the number of rows and columns of data within a table so that memory allocation is automatic; they can determine the list of column names so that scalars within an AVS field have meaningful names. Of course SQL strings can be of great complexity: in order to use more involved commands it is possible to avoid parsing the command and to provide statement construction in other AVS modules which could contain application specific statements.

Database systems protect the user by returning error and warning messages as necessary. AVS/SQL detects these conditions and displays the message. Additional protection has been provided in the modules. For example if no checks are made, it is possible to filter out all rows in a table so that an AVS field of length zero would be created. Under these circumstances the module builds a null field and issues a warning message.

Conclusion

AVS/SQL provides a basic toolkit whereby AVS can exchange data with a database system. Many corporations have compelling reasons to store their data in relational databases and to display and interact with their data. It is expected that AVS/SQL will make a valuable contribution towards the creation of applications for fields including the oil, gas and water businesses; the financial sector; industries concerned with demography, logistics and communications.

The AVS/SQL system enhances program development efficiency by the visual assembly of modular SQL communication tools. Interactive 3D application construction with visual representation of database objects is possible in minutes by combining a wide range of tools. A source code generator allows ultimate flexibility with access to detailed control structures. The AVS/SQL system advances the range of possible client applications, and provides a rich rapid prototyping environment for creation of these new applications.

References

1. Stephen, John, "AVS/SQL" AVS UK User Group Meeting, Canterbury, May 1994.
2. "AVS/SQL Technical Manual" Advanced Visual Systems, Inc. November 1994.

Acknowledgements

ORACLE, Pro*C are registered trademarks of Oracle Corporation, Belmont, California.