Proceedings of the 15th IASTED International Conference on Modelling and Simulation,
March 1-3, 2004, Marina del Rey, California, pp. 157-160

DATABASE ACCESS IN SIMSCRIPT II.5

Stephen V. Rice
Department of Computer and Information Science
The University of Mississippi
201 Weir Hall, P.O. Box 1848, University, MS 38677 USA
tel: (662) 915-5359, fax: (662) 915-5623
email: rice@cs.olemiss.edu

ABSTRACT

Through a new programming interface based on Open Database Connectivity, programs written in the SIMSCRIPT II.5 programming language gain the ability to access relational databases. Retrieving input data, storing simulation results, and sharing data with other software tools are facilitated. A common interface to multiple database management systems, running on local or remote computers, gives the programmer a flexible range of choices.

KEY WORDS

Simulation Tools and Languages, Database Management, SIMSCRIPT, SQL

1. Introduction

The input and output data of a non-trivial simulation program are typically voluminous. Managing this data using external flat files is difficult. A modern database management system (DBMS) offers powerful and convenient features for the organization, storage, and retrieval of simulation data. Access to databases from simulation programs is no longer an option but is a requirement for large modelling projects.

The input data to a simulation program include model parameters and configuration information. A DBMS provides standard and customizable interfaces for data entry and enforces integrity constraints for data validation. Simulation input may come from existing databases; for example, Randell and Bolmsjö used a database describing a factory to simulate the factory [1].

Without database access, a simulation program may choose to summarize its results in a small number of output values. The availability of a database encourages the storage of large quantities of intermediate results. By saving the details of simulation runs, decisions can be made later regarding which summary statistics to compute.

The power of the DBMS query facility can be applied to retrieve and aggregate any portion of the output from one or many simulation runs and models. Tools supplied by the DBMS manufacturer, or by third-party software vendors, can be used for report generation, graphing, animation, statistical analysis, optimization, and data mining. As noted by Standridge and Centeno, the database is the centerpiece of a "modular simulation environment," providing the mechanism by which information is shared among software tools [2]. Simulation models may share inputs, and the output from one model may be input to another.

The DBMS enables concurrent access to simulation data by multiple users and programs; prevents unauthorized access to the data; and provides for backup and recovery of the data.

2. SIMSCRIPT

SIMSCRIPT was among the first programming languages for computer simulation [3]. The RAND Corporation developed SIMSCRIPT I in 1962 [4] and SIMSCRIPT II in 1968 [5] for the U.S. Air Force. SIMSCRIPT II.5 is a commercial version that has been marketed and improved by CACI since the 1970s [6].

A model is described in SIMSCRIPT by its entities and their attributes, and by ordered lists of entities known as sets. Initially, instantaneous events were provided for discrete-event simulation, but time-elapsing processes were added to SIMSCRIPT II.5 in the mid-1970s. The language is known for its self-documenting, English-like syntax, which facilitates the communication and verification of simulation models.

In a 1979 article [7], Harry Markowitz, the principal inventor of SIMSCRIPT, describes the planned functionality of SIMSCRIPT II as a series of "levels." Database entities were to be added to the language in the sixth of seven levels; however, only the first five levels were implemented. He further shows how SIMSCRIPT entities, attributes, and sets can be mapped to and from the network, hierarchical, and relational database models.

In the early 1980s, Markowitz and colleagues at IBM developed an experimental programming language based on SIMSCRIPT II named EAS-E, an acronym for Entities, Attributes, Sets, and Events. The implementation of EAS-E supported database entities using a CODASYL network-model database [8].

Some SIMSCRIPT II.5 programmers have developed "home-brewed" database interfaces using SIMSCRIPT II.5's ability to call non-SIMSCRIPT routines. The language itself, however, did not support database access until 2003.

3. Approach

Open Database Connectivity (ODBC) [9] is a standard programming interface that uses Structured Query Language (SQL) [10] to access relational databases. ODBC provides a common interface to different database management systems.

SIMSCRIPT II.5 Database Connectivity (SDBC) [11] was introduced in 2003. It is a library of routines that enables SIMSCRIPT II.5 programs to create tables in relational databases; to insert, modify, and delete the rows of database tables; and to perform database queries. It is patterned after and utilizes ODBC to provide a common interface to database management systems.

Figure 1 illustrates the communication between a SIMSCRIPT II.5 program and a DBMS. First the program calls an SDBC routine. Using the ODBC application programming interface (API), the SDBC routine calls the ODBC Driver Manager, which is independent of any DBMS. The Driver Manager then calls the ODBC Driver for the intended DBMS. Using a DBMS-specific API, the Driver communicates with the DBMS. After performing the requested operation, the DBMS communicates the results to the Driver, which relays them to the Driver Manager. In turn, the Driver Manager sends the results to the SDBC routine, which returns them to the SIMSCRIPT II.5 program.

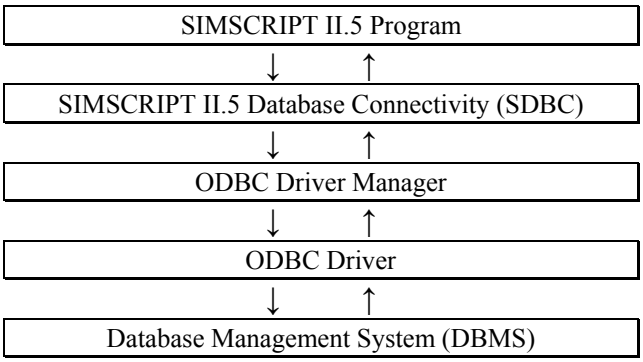

Figure 1. Communication between a SIMSCRIPT II.5 program and a DBMS

SDBC enables SIMSCRIPT II.5 programs to interact with any DBMS providing an ODBC 3.0 Driver. If the DBMS is running on a remote computer, communication over a network is performed implicitly. The SIMSCRIPT II.5 program and the DBMS may be running on different types of computers with different operating systems.

The programmer may choose from a number of available database management systems. To date, SIMSCRIPT II.5 programs have used SDBC to access databases managed by IBM DB2®, Microsoft Access, Microsoft SQL Server, MySQL™, Oracle®, and PostgreSQL. The common interface provided by SDBC means there is only one interface for the programmer to learn, and one DBMS can be replaced by another without rewriting source code.

An effort was made in the design of SDBC to provide an easy-to-use programming interface. The result is a library of 16 routines, whereas the ODBC API consists of approximately 75 routines. The SDBC library offers most of the power of ODBC while hiding ODBC's complexity. The following sections describe the SDBC routines.

4. Connecting to a Database

A SIMSCRIPT II.5 program must first connect to a database. An ODBC data source must have been defined for the database; it associates a data source name with the database, an ODBC Driver, and connection parameters required by the DBMS. The first argument passed to the SDBC routine named DB.CONNECT.R is the data source name; the second and third arguments specify the user name and password for accessing the database. The following statement connects as user STEVE with password SECRET to the database identified by the data source named SIMDB:

```
call DB.CONNECT.R("SIMDB", "STEVE", "SECRET")
```

When the program is finished accessing this database, it calls DB.DISCONNECT.R to disconnect:

```
call DB.DISCONNECT.R
```

5. SQL Updates

After connecting to a database, the SIMSCRIPT II.5 program may pass SQL statements one at a time to the DBMS for processing. An SQL statement that modifies the database is passed to DB.UPDATE.F. This function returns after the SQL statement has been executed by the DBMS on the connected database. The number of rows affected by the operation, if applicable, is returned to the caller.

```
let NUMROWS = DB.UPDATE.F(SQL.COMMAND)
```

An SQL CREATE TABLE statement is passed to DB.UPDATE.F to create a table. For example, suppose a table named RESULT is needed to hold the results of simulation runs, where each row of the table contains a run ID and the maximum and average queue length observed during the run. The following SQL statement creates this table:

```
CREATE TABLE RESULT (RUNID INTEGER,
MAXQLEN INTEGER, AVGQLEN REAL)
```

The program stores the SQL statement in a text variable and then passes it to DB.UPDATE.F for execution by the DBMS:

```
let SQL.COMMAND = CONCAT.F(
"CREATE TABLE RESULT (RUNID INTEGER, ",
"MAXQLEN INTEGER, AVGQLEN REAL)")

let NUMROWS = DB.UPDATE.F(SQL.COMMAND)
```

Upon return from DB.UPDATE.F, the table has been created. The table is destroyed by executing an SQL DROP TABLE statement:

```
let NUMROWS = DB.UPDATE.F("DROP TABLE RESULT")
```

CREATE TABLE and DROP TABLE are examples of SQL Data Definition Language (DDL) statements. Any DDL statement supported by the DBMS may be passed to DB.UPDATE.F for execution.

An SQL INSERT statement inserts one or more rows into a table. Suppose the maximum and average queue length for run #101 were 12 and 2.75, respectively. The following statement inserts a row into the RESULT table to represent this run:

```
let NUMROWS = DB.UPDATE.F(
"INSERT INTO RESULT VALUES (101, 12, 2.75)")
```

An SQL UPDATE statement modifies the value of one or more columns in one or more rows. The following statement changes the average queue length for run #101 to 2.25:

```
let NUMROWS = DB.UPDATE.F(
CONCAT.F("UPDATE RESULT ",
"SET AVGQLEN = 2.25 WHERE RUNID = 101"))
```

An SQL DELETE statement deletes one or more rows. The following statement deletes the row for run #101:

```
let NUMROWS = DB.UPDATE.F(
"DELETE FROM RESULT WHERE RUNID = 101")
```

A database transaction is an atomic sequence of updates to a database in which all or none of the updates are made permanent. If the transaction is *committed*, then all of the changes are made permanent; if the transaction is *rolled back*, then all updates made during the transaction are undone and the database is returned to the state it was in before the transaction was started. Transactions prevent concurrent users from seeing one another's uncommitted changes to the database (i.e., their work in progress) and enable the DBMS to restore the database to a correct state following a system or program failure.

When "auto-commit" is "on," each SQL statement is executed as its own transaction. That is, either the statement completes successfully and all changes made by the statement are made permanent (the transaction is committed), or the statement fails and all changes are undone (the transaction is rolled back). When auto-commit is "off," a sequence of SQL statements may be executed within the same transaction. The current transaction is terminated by calling DB.COMMIT.R or DB.ROLLBACK.R and a new transaction is begun. Auto-commit is turned on or off by calling DB.AUTOCOMMIT.R and is on by default.

6. SQL Queries and Parameters

A SIMSCRIPT II.5 program may submit queries to the DBMS for processing. Any SQL SELECT statement may be passed to DB.QUERY.R. The rows returned by the query are retrieved one at a time by calling DB.FETCH.F for each row. DB.FETCH.F returns 1 if it has successfully retrieved the next row of the query result and returns 0 when there are no more rows to retrieve. The value of each column of a retrieved row is obtained by calling DB.GETINT.F for an integer column, DB.GETREAL.F for a floating-point column, and DB.GETTEXT.F for a character-string column, where each function accepts a column number as its argument.

In the following example, each row with an ID between 100 and 200 is retrieved from the RESULT table and displayed:

```
let SQL.QUERY = CONCAT.F(
"SELECT RUNID, MAXQLEN, AVGQLEN FROM RESULT",
" WHERE RUNID BETWEEN 100 AND 200")

call DB.QUERY.R(SQL.QUERY)

while DB.FETCH.F = 1
do
  write DB.GETINT.F(1) as "Run #", i 3
  write DB.GETINT.F(2) as
  " had a maximum queue length of ", i 3
  write DB.GETREAL.F(3) as
  " and an average queue length of ", d(5,1),/
loop
```

An SQL parameter is a question mark (?) appearing in an SQL statement which is replaced by the value of a program variable or expression. Suppose integer variables named MIN.ID and MAX.ID contain the smallest and largest run IDs to be retrieved. The preceding example can be rephrased using SQL parameters that take their values from these variables:

```
let SQL.QUERY = CONCAT.F(
"SELECT RUNID, MAXQLEN, AVGQLEN FROM RESULT",
" WHERE RUNID BETWEEN ? AND ?")

call DB.SETINT.R(1, MIN.ID)
call DB.SETINT.R(2, MAX.ID)
call DB.QUERY.R(SQL.QUERY)
```

The value of each SQL parameter must be set before the statement containing the parameters is executed. DB.SETINT.R, DB.SETREAL.R, and DB.SETTEXT.R set the value of an integer, floating-point, and character-string parameter, respectively.

7. Conclusion

SIMSCRIPT II.5 Database Connectivity brings the power of relational database systems to SIMSCRIPT II.5 programs. By utilizing Open Database Connectivity, a common interface is provided to multiple database management systems. The union of SIMSCRIPT II.5 and modern database technology has been anticipated for many years and is now a reality.

8. Acknowledgement

The SDBC project was suggested to the author by Ana Marjanski and was sponsored by CACI. The author expresses his sincere appreciation to Ana Marjanski and Harry Markowitz for their helpful comments.

References

[1] L.G. Randell and G.S. Bolmsjö, Database driven factory simulation: A proof-of-concept demonstrator, *Proc. 2001 Winter Simulation Conference*, Arlington, VA, USA, 2001, 977-983.

[2] C.R. Standridge and M.A. Centeno, Concepts for modular simulation environments, *Proc. 1994 Winter Simulation Conference*, Lake Buena Vista, FL, USA, 1994, 657-663.

[3] R.E. Nance, A history of discrete event simulation programming languages, in T.J. Bergin and R.G. Gibson (Eds.), *History of programming languages*, 8 (New York: ACM Press, 1996) 369-427.

[4] H.M. Markowitz, B. Hausner, and H.W. Karr, *SIMSCRIPT: A simulation programming language* (Englewood Cliffs, NJ: Prentice-Hall, 1963).

[5] P.J. Kiviat, R. Villanueva, and H.M. Markowitz, *The SIMSCRIPT II programming language* (Englewood Cliffs, NJ: Prentice-Hall, 1968).

[6] CACI Products Company, *SIMSCRIPT II.5 reference handbook* (La Jolla, CA: CACI Products Company, 1997).

[7] H.M. Markowitz, SIMSCRIPT, in J. Belzer, A.G. Holzman, and A. Kent (Eds.), *Encyclopedia of computer science and technology*, 13 (New York: Marcel Dekker, 1979) 79-136.

[8] A. Malhotra, H.M. Markowitz, and D.P. Pazel, EAS-E: An integrated approach to application development, *ACM Transactions on Database Systems*, *8*(4), 1983, 515-542.

[9] Microsoft Corporation, *Microsoft ODBC 3.0 programmer's reference* (Redmond, WA: Microsoft Press, 1997).

[10] C.J. Date and H. Darwen, *A guide to the SQL standard* (Boston: Addison-Wesley, 2000).

[11] CACI Products Company, *SIMSCRIPT II.5 database connectivity user's manual* (San Diego, CA: CACI Products Company, 2002).