ON WORKLOAD CHARACTERIZATION OF RELATIONAL DATABASE ENVIRONMENTS

Philip S. Yu, Ming-Syan Chen, Hans-Ulrich Heiss and Sukho Lee
IBM Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Abstract

As relational database systems become increasingly popular, there is a clear need to better understand the workload so systems can be designed or tuned more effectively. A RElational Database Workload AnalyzeR (REDWAR) is developed to characterize the workload in a DB2 environment. This is applied to study a production DB2 system where an SQL trace for a two hour interval and an image copy of the database catalog were obtained. The results of the workload study are summarized. Here we focus on the structure and complexity of SQL statements, the makeup and run-time behavior of transactions/queries, and the composition of relations and views. The results obtained provide the important information needed to build a benchmark workload to evaluate alternative design trade-offs of database systems.

Index Terms: relational databases, workload characterization, DB2.

1 Introduction

As relational database systems grow increasingly popular, throughput and response time requirements are becoming ever more stringent [1]. There is a clear need to understand the workload so systems can be better designed [2]-[6]. The workload has implications for both software and hardware designs. For example, query optimizers often make assumption of uniform distribution of attribute values in access path selection [7] [8]. In the presence of data skew, where certain attribute values are more popular than others, the uniform assumption can lead to non-optimal access path selection and degraded performance [9]. Furthermore, the access plan of a query is often generated before execution time to reduce run time overhead [7]. However, the frequent occurrences of input variables (whose values are determined at run time) in the predicates of SQL statements would support the adaptive approach to access plan selection in [10]. Although a debit-credit type banking workload has often been cited to compare transaction processing systems [11], the workload is very simplistic. Consequently, to examine workloads in other application environments which can better reflect in general the functions and capabilities of a relational system is very important for us to evaluate various design issues of database systems, and is thus taken as the objective of this paper.

Using a RElational Database Workload AnalyzeR (REDWAR), developed at IBM Research, we study in this paper the structure and complexity of SQL statements, the makeup and behavior of transactions/queries, and the composition of relations and views in a DB2 like environment. As SQL statements can be quite complex, we study the SQL statement composition to analyze the percentage of SQL statements having each type of constructs like WHERE, GROUP BY, HAVING,

ORDER BY, subquery, aggregate function, etc. The number of predicates in the WHERE clause, the distribution of the predicate types and operand types, and the number of columns in the SELECT clause are analyzed. The numbers of relations in the FROM, the GROUP BY and the ORDER BY clauses, respectively, are also investigated. The view definition can be similarly analyzed. The transaction statistics gathered include the number of SQL statements of each type executed, the number of tuples scanned and retrieved/updated/inserted/deleted, the number of pages accessed, and the elapsed time for each transaction invocation. This provides all the necessary information to build a benchmark workload to evaluate alternative design trade-offs of database systems. We then apply REDWAR to examine a production DB2 system which runs an accounting type application in a petroleum company. In this environment, a large number of the transactions are to query the database and generate reports. Besides the batch update to refresh the database, there are also on-line updates/inserts/deletes. An SQL trace was collected for a two hour interval during peak period and an image copy of the DB2 catalog was also obtained. Note that although workloads in different environments may vary, this accounting type workload investigated does provide a real-life example of a production DB2 workload in a major application environment which is different from debit credit, and can serve as a good testbed for REDWAR.

Relational database systems like DB2 provide quite a lot of information on the workload. Some of this is stored in the database catalog, providing various types of information on the relations, application plans and SQL statements. In DB2, SQL statements are precompiled into a Database Request Module (DBRM). One or more DBRM's are compiled and bound into an application plan which contains a set of internal control structures representing the complied form of the original SQL statements from which the DBRM's are built [12]. Here, we refer to each commit point in a plan as a transaction. Certain information is available at run time indicating the execution behavior of each SQL statement, such as how many tuples have been scanned on the average for each qualified tuple retrieval. We shall explain the methodology developed in REDWAR so that information scattered in the catalog tables and run-time trace records can be combined in a meaningful way to get insight on SQL statement structure and transaction behavior. Note that to the best of our knowledge, there is no prior work, by developing such a software tool, to study the structure and complexity of SQL statements, the makeup and behavior of transactions/queries, and the composition of relations and views. This distinguishes our work from others [3] [4] [6].

In Section 2, the relevant workload information available from DB2 is briefly discussed. We describe our methodology employed in REDWAR to obtain and process this information in Section 3. In Section 4, we apply REDWAR to a DB2 environment and present the workload analysis. This paper concludes with Section 5.

2 DB2 Workload Information

DB2 maintains a set of thirty tables referred to as the DB2 catalog [13]. The catalog tables contain information on table spaces (in DB2, to efficiently manage storage, multiple tables can be stored into a single table space), tables, views, columns, indexes application plans, etc. We shall mention a few that are relevant to our study of SQL statements and transactions. The SYSSTMT table contains one or more rows to describe the text of each SQL statement. It also contains information such as statement number, and the name of the application plan. The order of multiple rows describing the same SQL statement is indicated by a sequence number. The SYSVIEWS table contains the name of the view and one or more rows to describe the text of each view statement. The SYSPLAN table gives information of each application plan. SYSTABLES provides descriptions of each table and view, such as the number of columns, the total number of rows (which is not available for

views), the percentage of pages which contains rows of the table, etc. The SYSINDEXES table contains one row for every index and the SYSKEYS table has one row for each column in the index key indicating the numerical positions of the column in the row and in the index key, respectively. SYSCOLUMNS gives information on each column.

DB2 also has a very extensive run-time tracing facility [13] [14]. It can collect all kinds of performance, accounting and statistical data. The tracing facility is invoked through a DB2 START TRACE command. The various parameters in the command specify the information to be traced. In the performance trace, one can specifically request the detailed events to be traced, e.g. subsystem related events, SQL related events, buffer manager IO requests, detailed lock information, sort detail, etc. For the purpose of this study, we shall be concentrating on SQL related events which are provided by the class 3 performance trace. The SQL trace records provide information on start time and end time of each SQL statement, the statement type (e.g. SELECT, FETCH, DELETE, OPEN, CLOSE, etc.), statement number, etc. It also collects run time statistics on tuples scanned and retrieved/updated/inserted/deleted. DB2 consists of various components, such as Relational Data Services (RDS) and Data Manager (DM) [15] RDS is responsible for materializing the external view of data from stored data; DM manages all stored data by providing access to data. A tuple is first scanned by DM applying some search arguments (referred to as SARGABLE predicates) which are simple predicates of the form "column comparison operator value" [7]. If not rejected, the tuple is passed on to RDS where the rest of the predicates are applied. Certainly, overhead can be reduced by rejecting unqualified tuples at the DM level. The SQL trace provides detailed information on the number of tuples scanned by DM, the number of DM qualified tuples, RDS qualified tuples, the number of pages scanned, etc. [16] [17].

3 Methodology

Now we explain the methodology used in REDWAR to analyze SQL statements. There are several issues that need to be addressed. First of all, consider the case of SQL statements involving views. As views are virtual tables which do not exist physically, views appearing in an SQL statement are replaced during execution plan generation by their definitions [12]. In other words, a simple retrieval statement on a view may in fact be a complex join statement in disguise. Secondly, DB2 has the cursor concept associated with the embedded SQL construct [12]. In contrast to COBOL and PL/I like programming languages which deal with one record at a time, an SQL statement can result in a set of records. The cursor concept is introduced to address this issue. A "cursor" is defined on an embedded SQL statement so that one can issue a "FETCH" statement to fetch one record at a time based on the cursor. The FETCH statement is thus an artificial construct which has little to do with analyzing data manipulative SQL statement structures. However, the run time statistics of the FETCH statement reflect that of the corresponding SQL statement assuming the application retrieves all qualified tuples. Finally, not all SQL statements are defined in the Catalog SYSSTMT table. Dynamic SQL [12], constructed during run time, would not appear in the Catalog, but is provided by the class 3 performance trace during run-time [13] [14].

The workload analyzer consists of two parts:

- the SQL analyzer for analyzing the SQL statements and the catalog information,
- the trace analyzer for processing the trace and gathering various run time statistics.

Both of them use information from each other to provide an integrated and comprehensive picture of the workload.

From the catalog tables, starting with processing the SYSVIEWS table, we analyze the SQL statements in SYSSTMT. The view information is incorporated into the SQL statement analysis. As views can be defined in terms of other views, several expansions or substitutions may be required before we can express a view in terms of base tables and obtain summary information. Each time a table name is encountered in the FROM clause of an SQL, we first check whether it is a view. If yes, the information obtained from the view analysis is retrieved to be incorporated into the SQL statement analysis. For example, assume the following view from [12] is defined in the SYSVIEWS table.

CREATE VIEW CITYPAIRS
(SCITY, PCITY)
AS SELECT S.CITY, P.CITY
FROM S, SP, P
WHERE (S.S# = SP.S#) **AND** (SP.P# = P.P#)

This is the usual suppliers (S), parts (P), and shipment (SP) relations used in [12]. The view tries to create all the supplier city and parts city pairs related to the same shipment. In the SYSSTMT table, assume we have the following statement.

SELECT PCITY
FROM CITYPAIRS
WHERE SCITY = London

When processing the above statement, we need to realize that relation CITYPAIRS is a view, so this is not a simple retrieve statement with one predicate. It is in fact a three way join involving 3 predicates.

Whether in SYSVIEWS or SYSSTMT, the SQL statement can span multiple rows. We first sort the catalog tables so the rows related to the same SQL statement are contiguous. We then develop the SQL analyzer to collect structural information on SQL statements. Our analysis here will concentrate on SQL statements for data manipulation. The data manipulation SQL statements are classified into five types: (1) "Singleton" SELECT, (2) UPDATE, (3) DELETE, (4) INSERT, (5) SELECT involving CURSOR (w/wo UPDATE OF). For each SQL statement type, we collect statistics on the number of relations involved, the number of columns selected, the number of predicates appearing, whether subqueries appear, and whether aggregate functions are used. We also examine how often constructs like ORDER BY, GROUP BY and HAVING are included.

The predicates are further classified according to their types [13]: basic, quantified, BETWEEN, NULL, LIKE, EXISTS, and IN. A basic predicate is used to compare two values. It takes the form of an expression followed by a comparison operator and another expression or a subquery (without ANY or ALL). A quantified predicate compares a value with a collection of values. It has the same form as a basic predicate type except that the second operand is a subquery preceded by ANY or ALL. An IN predicate is another way to compare a value with a collection of values. An EXISTS predicate tests for the existence of certain rows. A BETWEEN predicate is used to compare a value with a range of values, whereas a LIKE predicate is used to search for strings that have a certain pattern. A NULL predicate tests for NULL values. The aggregate functions used are also classified according to their types: AVG, MAX, MIN, SUM, COUNT(*), or COUNT(DISTINCT).

The trace analyzer consists of two phases. In the first phase the trace analyzer filters out all dynamic SQL statements and sends them to the SQL analyzer for structure analysis as described above. Having analyzed both the static and the dynamic statements, the SQL analyzer provides the trace analyzer with some structure information, e.g., whether a particular statement implies a join-operation or if there are views involved. In the second phase the trace records are examined to collect run time statistics. Statistics are gathered on either a per plan or per statement basis. Selection

of a particular plan-ID, connection-ID, or authorization-ID is also possible. The identification of each static SQL statement and its run time records is through its statement number, which appears in the trace records as well as in the catalog table SYSSTMT. For dynamic statements, a unique number is generated during phase 1 to make identification possible during phase 2. For a FETCH, we need to use the cursor name to identify the corresponding SQL statement. The program can be parameterized to provide a variety of reports of different extent and detail.

We also made an attempt to analyze the number of tuples scanned before a qualified tuple can be retrieved. We are especially interested in the case when the SQL is a retrieval type statement (not join) and index scan is used to access the tuples. The number of tuples scanned per qualified tuple implies the efficiency of the access path. Toward this end, the trace analyzer needs the information from the SQL analyzer to find out whether a large index scan is to be interpreted as resulting from retrievals with very selective predicates, or is caused by performing a join.

4 Sample Results

We now present some of the information obtained after applying REDWAR to study the DB2 workload mentioned before. The following subsections examine the table characteristics, transaction plan description, SQL statement structure, view definition, and transaction run-time behavior.

Table Characteristics

The characteristics of the relations are first examined. The environment consists of 323 base tables and 567 views. Table 1 presents the average number of columns and rows, and the maximum number of columns and rows in a base table. Also shown is the average number and maximum number of columns in a view. The average number of columns over all base tables and views is 32.9. As views are derived dynamically at run time, statistics on the number of rows are not available in the catalog. Among the base tables, 38.4% have no index, 33.1% have one index, 17.0% have two indexes, 9.6% have three index and 1.9% have four indexes. The average number of indexes per table is 1.03. The indexes are generally based on multiple columns. Fig. 1 shows the distribution of the column width in bytes over all base tables and views, where the average column width is 9.5 bytes and the maximum column width is 4006 bytes. Most of the columns have a width less than 20 bytes.

Application Plan Description

There are 305 application plans defined in the catalog. The average number of SQL statements appearing in a transaction plan is 17, whereas the maximum number of statements appearing is 551. We present the distribution of the number of statement appearances in Fig. 2. As we can see, 90% of the application plans contain less than 20 SQL statements, with 51.6% of the plans having less than 5 SQL's and there are a few with a large number of SQL statements. In Table 2, the SQL statements are classified according to their types. Both static SQL statements from the catalog, and dynamic SQL statements from the run time trace are considered. There are 5381 static SQL statements and 459 dynamic SQL statements. Note that dynamic statements generated from each invocation of a plan are counted as different statements. In static SQL close to 30% of the SQL statements are data manipulative type statements, which are the SELECT, DELETE, UPDATE, and INSERT statements. Among those SQL statements, 75.9 % are SELECT, 4.7% are DELETE, 7.7% are UPDATE and 11.7% are INSERT. Here the SELECT includes both the singleton SELECT and cursor SELECT. There are also a large number of OPEN, CLOSE and FETCH statements, which are the constructs used to retrieve tuples from cursor SELECT statements. Another 8% of statements are related to preparing and executing dynamic SQL statements. Note that the DECLARE statements here only include DECLARE TABLE and DECLARE STATEMENT, but

not DECLARE CURSOR. In dynamic SQL, 96% are data manipulative type statements, and among those SQL statements, 96.8 % are SELECT, 1.4% are DELETE, 0.9% are UPDATE and 0.9% are INSERT. Among the data manipulative SQL statements, 47 (i.e. 2.9%) static SQL statements and 82 (i.e. 18.6%) dynamic SQL statements reference views and all reference just one view.

SQL Statement Structure

We next examine the structural characteristics of the data manipulative type SQL's. For static SQL, Table 3a shows aggregate statistics for each statement type, where the column labelled C-SELECT means SELECT with cursor, and the column labelled S-SELECT means singleton SELECT. The total number of statements for each SQL type is given in parentheses. The aggregate statistics include the percentage of statements with WHERE clause, GROUP BY, HAVING, ORDER BY, and subquery constructs. The percentage of join queries are indicated after taking the view definition into consideration. We also indicate the percentage of multiple-record INSERT statements (INSERT statements with a SELECT clause), the percentage of cursor select with FOR UPDATE, the percentage of DELETE and UPDATE with cursor control, and the percentage of SELECT queries involving UNION. Note that in INSERT statements, only multiple-record INSERT statements can have a WHERE clause, FROM clause, GROUP BY, HAVING, ORDER BY, or subquery construct. Thus all these statistics (except the one on INSERT with SELECT) in the INSERT column are presented relative to the mutiple-record INSERTs instead of to all INSERTs. Furthermore, in Table 3a, an "X" means not applicable. Table 3b shows similar statistics for dynamic SQL. Note that by definition dynamic SELECT must be associated with a cursor. The percentage of dynamic SQL with a WHERE clause appears to be low, as one of the frequently executed plans generates a dynamic SQL with no WHERE clause to reference a small table. In Table 4a we show the distribution of the number of columns in the SELECT clause. For INSERT, we only consider multiple-record INSERT, i.e. those with a SELECT clause. Also included are the averages and maximum values. The number of relations in the FROM clause is shown in Table 4b. When more than one relation appears in the FROM clause, join operations will occur. As indicated by Table 4b, there are very few joins, mainly two-way joins and a few 3-way and 4-way joins. Table 4c shows the distribution of the number of predicates in the WHERE clause. For static SQL, more than 90% of the WHERE clauses have less than 5 predicates and one predicate is generally most prevalent. Occasionally, a large number (> 10) of predicates may occur. Dynamic SQL seems to have a larger tail distribution. The distribution of the number of columns in the ORDER BY clauses are shown in Table 5. The number of statements with the ORDER BY constructs are indicated in parentheses. GROUP BY construct only occurs in 8 SELECTs with cursor and 28 INSERTs in static SQL. Among these, 75% is on one column and 25% is on two columns for SELECT with cursor and 100% is on one column for INSERT. Table 6 shows the distributions of the predicates types. Clearly, the basic predicate type is used most often. For static SQL, the first operand in a predicate is always a column name, whereas the second operand is generally (more than 90% of the time) a host variable as shown in Table 7. For dynamic SQL, the first operand is again always a column name, but the second operand is equally likely to be either a constant or a parameter to be specified at execution time. Aggregate functions are not used in dynamic SQL and are used only occasionally in static SQL. For example, COUNT(*) occurs 6 times in CURSOR SELECT, MAX 34 times in singleton SELECT, SUM 76 times in INSERT and 2 times in CURSOR SELECT.

View Definition

We next consider the view definitions. Among the 567 views, only 4 of them are defined in terms of views: two with one view and the other two with two views. The structural characteristics can be analyzed in a similar way as the base table and is omitted here.

Transaction Run-time Behavior

We now examine the execution behavior of SQL statements and transactions. In the 2 hour tracing interval, 24,364 static SQL statements and 125,428 dynamic SQL statements were executed. The average response time per static SQL statement is 27 msec while that per dynamic SQL statement is 32 msec. There are 2438 transactions (plan invocations) out of 34 different plans. Among these, 88.56% are read-only transactions. Table 8 shows the executed SQL statement distributions based on SQL statement types for static SQL and dynamic SQL. We distinguish between PREPARE-c, which is PREPARE for SELECT with cursor, and PREPARE-nc, which is PREPARE for other statement types. None of the UPDATEs are cursor controlled as indicated by UPDATE-nc. For CREATE and DROP, we use ts, t and ix to denote the target: table space, table, and index, respectively. We next consider the transaction/plan execution frequency. There are 7 "popular" transaction plans, each getting more than a hundred executions during the two hour period, while more than half of the plans get less than ten executions. The average number of executions per plan in the measured interval is 71.7. Most of the transactions have a response time less than a second, but occasionally there are large transactions with a response time of more than a hundred seconds. The average transaction response time is about 4.5 seconds. Furthermore, more than 80% of the transaction executions issue less than 10 SQL statements. The average number of SQL statements occurring per transaction execution is a lot larger (61.4), due to occasional transaction execution with a large number of repeated FETCH statements.

The make up and average behavior of a transaction is shown in Table 9. For each statement type, the number of executions, the mean response time per execution, and the number of rows processed and examined, which are the number of rows in the table space scanned and the number of rows in the table scanned, respectively, are presented. The invocation of each CURSOR SELECT statement consists of OPEN CURSOR, FETCHes, and CLOSE CURSOR. Regarding an OPEN/CLOSE pair as the representation of an invocation of a CURSOR SELECT we have in Table 4.10 on the average 0.69 CURSOR SELECT invocations each of which comprises about 80 FETCHes. The "scan type" indicates what kind of scan has been performed by the Data Manager: index (INDX), table space (SEQD), or work file (SEQW). The average number of rows qualified by the DM and RDS, the number of rows inserted/updated/deleted and the average number of pages scanned per SQL execution are also presented for each type of scan. During each SQL execution, not all scan types get invoked. Still, the average is taken over all executions of the statement type, not just the executions invoking the particular scan type. For FETCH and (singleton) SELECT type statements, when each execution results in one RDS qualified tuple, the number in the RDS column reflects the fraction of the corresponding scan type executions. For (singleton) SELECT, 72% of the executions use index scan with an average of 1.06 (= 0.76/0.72) tuples examined per (RDS) qualified tuple, while 28% of the executions use table space scan with an average of 36.07 (= 10.10/0.28) tuples examined per qualified tuple. For FETCH, only 20% of the executions use index scan with an average of 453.2 tuples examined per qualified tuple, 77% of the executions use table space scan with an average of 64.79 tuples examined per qualified tuple, and 3% use work file scan with an average of 1147.67 tuples examined per qualified tuple. Thus as we can see, the singleton SELECT and cursor SELECT show very different run time behavior. From Table 9, the filtering factors of the predicates at DM and RDS can be calculated. For example, under index scan, the DM selects one tuple out of every 40.1 (=90.64/2.26) tuples examined based on the SARGABLE predicates and the RDS selects one tuple out of every 11.3 (=2.26/0.20) DM qualified tuples. From the "pages scan" column, we can derive that on the average two hundred some pages need to be accessed per transaction execution.

For the number of relations involved in the SQL statements executed, we find that about

95% access just one relation. Almost 5%, however, are fetching result tuples from some four-way join operation. We next consider the effect of selectivity variations on the optimality of access path selection. Within the tracing period, there were a few situations where for a single FETCH statement more than 1,000,000 rows (i.e. almost the entire table) were examined using index scan. On the average, 87% of the operations using an index examine less than 10 rows before they find the qualified tuple, 99.6% examine less than 100 rows. The remaining 0.4%, however, can lead to very long searches. Finally, we examine the execution behavior of the cursor SELECT statements, i.e. the number of FETCHes executed under a cursor SELECT. It is found that close to 87% of the time less than 10 tuples satisfy the predicates, but there are times (around 0.18%) more than 10,000 tuples are qualified.

In the particular environment we studied, the following observations can be made. The tables (including both base tables and views) have on the average 32.9 columns with an average column width of 9.4 bytes. The base tables have on the average of 14 K rows, while the maximum is around 1.9 M rows. About half of these tables have one or two indexes while nearly 40% have no index. Close to 90% of the transactions are read-only. A majority of the application plans contain only a few SQL statements (less than 10). Each application executes on the average about 60 SQL's, and most of them are repeated executions of FETCH. There are substantial variations on the number of FETCHes executed even for different invocations of the same transaction plan. On the average, two hundred some pages are accessed per transaction execution. This workload is thus considerably more complex and shows more variations on transaction execution time than the IMS transaction workload examined in [18]. Almost 85% of the SQL statements executed are dynamic statements. Index scan is prevalent (72%) for singleton SELECTs and table space scan is prevalent (77%) for cursor SELECTs, which accounts for more than 90% of the SQL statements executed. The number of tuples examined under index scan to get a qualified tuple can vary from a few to occasionally more than a million, which is more than half of the maximum table size. This clearly indicates the effect of selectivity variations on the optimality of access path selection. We further analyze the structure of the data manipulative SQL statements. Few of the SQL's involve GROUP BY, HAVING, subquery, and aggregate functions, etc., while ORDER BY appears in about one fifth of the static SQL statements. The WHERE clause consists on the average of 2 to 5 predicates, depending on the statement type, with a maximum of 22. In static SQL, each predicate almost always has an input variable as the second operand, while in dynamic SQL parameter appears as the second operand for about half of the time. About 5% of the SQL's executed are FETCHes into results of join type queries.

5 Conclusion

In this paper a relational database workload analyzer, REDWAR, is presented. The objective is to study the structure and complexity of SQL statements, the makeup and behavior of transactions, and the composition of tables and views. This is applied to study a DB2 production environment, where an SQL trace for a two hour interval during peak period and an image copy of the database catalog were obtained. The results obtained provide the important information needed to build a benchmark workload to evaluate alternative design trade-offs of database systems.

ACKNOWLEDGEMENT

The authors are indebted to Y.-H. Lee for his contributions at the early stages of this study. They are grateful to Union Texas Petroleum for providing the DB2 workload. Also, they would like to

thank L. Dalton, S. Degrange, G. Flatow, S. Lakshmi, A. Shibamiya, P. Selinger and J. Wolf for their help and advice.

REFERENCES

[1] C. J. Date, *Relational Databases: Selected Writings*, MA.: Addison-Wesley, 1986.

[2] W. Alexander, T. W. Keller and E. E. Boughter, "A Workload Characterization Pipeline for Models of Parallel Systems," *Performance Eval. Rev.,* vol. 15, no. 1, pp. 186-194, May 1989.

[3] R. B. Ashton, "DB2 Workload Analysis," *Proceedings of Computer Measurement Group Conf.*, pp. 71-73, 1986.

[4] P. R. Chintamaneni and L. W. Dowdy, "Workload Characterization and Synthetic Workload Generation: An Application Study," *Comput. Syst. Sci. Eng.,* vol. 4, no. 4, pp. 205-215, Oct. 1989.

[5] D. Ferrari, "Workload Characterization for Tightly Coupled and Loosely Coupled Systems," *Performance Eval. Rev.,* vol. 17, no. 1, pp. 210-211, May 1989.

[6] G. M. King, "Workload Characterization," *Proceedings of Computer Measurement Group Conf.*, pp. 789-801, 1986.

[7] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access Path Selection in a Relational Database Management System", *Proc. of ACM SIGMOD 20,* pp. 23-34, 1979.

[8] S. Christodoulakis, "Implications of Certain Assumptions in Database Performance Evaluation", *ACM Trans. on Database Systems,* vol. 9, no. 2, pp. 163-186, June 1984.

[9] M. S. Lakshmi, and P. S. Yu, "Effectiveness Parallel Joins", *IEEE Trans. on Knowledge and Data Eng.,* vol. 2, no. 4, pp. 410-424, Dec. 1990.

[10] Y.-H. Lee, and P. S. Yu, "Adaptive Selection of Access Plan and Join Methods", *Proc. 13th Intl. Computer Software and Applications Conf.,* pp. 250-256, Sept. 1989.

[11] R. W. Horst, and T. C. K. Chow, "An Architecture for High Volume Transaction Processing", *Proc. 12th Intl. Symposium on Computer Architecture,* Boston, MA, pp. 240-245. June 1985,

[12] C. J. Date, "A Guide to DB2", MA.: Addison-Wesley, 1988.

[13] "IBM Database 2 References", SC26-4078, IBM Corp.

[14] "IBM System Planning and Administration Guide", SC26-4085, IBM Corp.

[15] D. J. Haderle, and R. D. Jackson, "IBM Database 2 Overview", *IBM System Journal,* vol. 23, no. 2, pp. 112-125. 1984.

[16] "Database 2 Performance Monitor General Information", GH20-6856-01, IBM Corp.

[17] "Database 2 Performance Monitor: User's Guide", SH20-6857, IBM Corp.

[18] P. S. Yu, D. M. Dias, J. T. Robinson, B. R. Iyer and D. W. Cornell, "Distributed Concurrency Control Analysis for Data Sharing", *Proceedings of Computer Measurement Group Conf.,* pp. 13-20, 1985.