Database Performance in the Real World
— TPC-D and SAP R/3 —

Jochen Doppelhammer Thomas Höppler Alfons Kemper Donald Kossmann
Universität Passau
Fakultät für Mathematik und Informatik
D-94030 Passau, Germany
⟨*last name*⟩*@db.fmi.uni-passau.de*
http://www.db.fmi.uni-passau.de

Abstract

Traditionally, database systems have been evaluated in isolation on the basis of standardized benchmarks (e.g., Wisconsin, TPC-C, TPC-D). We argue that very often such a performance analysis does not reflect the actual use of the DBMSs in the "real world." End users typically don't access a stand-alone database system; rather they use a comprehensive application system, in which the database system constitutes an integrated component. In order to derive performance evaluations of practical relevance to the end users, the application system including the database system has to be benchmarked. In this paper, we present TPC-D benchmark results carried out using the SAP R/3 system, an integrated business administration system. Like many other application systems SAP R/3 is based on a commercial relational database system. We compare the SAP R/3 benchmark results with TPC-D results of an isolated database system, the database product that served as SAP R/3's back-end.

1 Introduction

Database performance is typically evaluated by executing a standard or de-facto standard benchmark directly on an *isolated* database system. Examples for this kind of methodology are the results obtained by the Wisconsin Benchmark [BDT83], the TPC benchmark results reported by various database (and hardware) vendors, and the results compiled in the benchmark handbook [Gra93]. Such an approach is very useful for comparing different database systems or particular components of database systems. However, the results derived by such tests do not reflect the performance that a user of a database system can expect in real world applications. Most users do not use an isolated database system; rather they employ an application system with an integrated database system. The data is not accessed directly via the database system interface (e.g., SQL) but rather via the predefined interfaces of the application system. The application system substantially augments the functionality of the database systems and is tailored to the particular application domain of the users.

The purpose of this paper is to make a first step towards getting benchmark results that directly meet the expectations of end users. In this paper, we study the database performance of SAP R/3 using the TPC-D benchmark. SAP is the market leader for integrated business administration systems, and its SAP R/3 product is a comprehensive software system which integrates modules for finance, material management, sales and distribution, etc. The TPC-D benchmark is a standard benchmark for decision support queries in business environments. The TPC-D benchmark was initially designed to study the performance of database systems in isolation, but the queries of the TPC-D benchmark are examples for the kind of queries that SAP R/3 users would ask, and therefore the benchmark can be implemented using SAP R/3. SAP R/3 uses a conventional relational database system as back-end, and it is possible to choose among several commercial systems when installing SAP R/3 (e.g., ADABAS, DB2, Informix, Oracle, SQL Server). To study SAP R/3's database performance, we implemented the TPC-D benchmark in SAP and compared the results with those obtained from running the benchmark directly on the database system that we chose to use in our SAP R/3 installation. What we hope to achieve is the following:

1. Encourage database (and hardware) vendors to do the same as we did and measure the performance of their system in conjunction with very popular, widely used application systems such as SAP R/3. This will help end users to choose the most appropriate database system and hardware platform for their particular applications and workloads. As our performance results indicate, it is not easily possible to deduce from the (isolated) benchmark results currently published by database vendors the actual performance observed by SAP R/3 users so that today users must often guess how well a particular database system would perform for their specific target application.

2. Give some insight into how an application system such as SAP R/3 interfaces with a database management system. As will become clear, application systems sometimes break down a user query into several parts and pass some parts (e.g., joins) down to the database system and execute

other parts (e.g., aggregations) themselves; furthermore, application systems sometimes *translate* queries in such a way that makes it impossible for the optimizer of the database management system to find a good execution plan for the query. Studying these effects will help developers of database systems to judge whether a new feature of their system is really helpful to improve the performance of applications, and it will also be helpful for developers of application systems to improve their query processors in order to take better advantage of today's database technology.

3. Give performance results that indicate the potential benefits and costs of a data warehouse for SAP R/3. This will again be interesting for end users and companies that use SAP R/3. Such companies will definitely use SAP R/3 for their online transaction processing, but they will need to decide whether they want to use the operational SAP database or construct a data warehouse for decision support.

The remainder of this paper is organized as follows: Section 2, gives a general overview of the most important features and characteristics of SAP R/3. Section 3 presents the TPC-D benchmark results; it describes how we implemented the TPC-D benchmark using SAP R/3 and discusses (among others) the results of the TPC-D power test for SAP R/3. We measured two different versions of SAP R/3, 2.2G and 3.0E, because both versions are currently used by many companies. Version 3.0 extends the functionality of Version 2.2 making it possible to achieve significantly better performance in many situations; as we will see, however, it is not easily possible to upgrade an existing 2.2 installation and immediately benefit from the extensions of Version 3.0. Section 4 analyzes the coupling of SAP R/3 with the database system in more detail by evaluating simple queries that isolate certain architectural effects. Section 5 presents experiments that we conducted in order to determine the costs to extract data from SAP R/3 to build a data warehouse. Section 6 concludes this paper.

2 Overview of SAP R/3

SAP R/3 is the market leader for integrated business administration systems. It integrates all business processes of a company and provides modules for finance, human resources, material management, etc. SAP R/3 is based on a (second party) relational database system which serves as an integration platform for all components of SAP R/3. The database system manages the SAP database which stores all business data of a company (e.g., customer and supplier information, orders, ...), all of SAP R/3-internal control data, an SAP R/3 data dictionary, and the code of all application programs. Virtually no data are stored outside this SAP database, thereby avoiding the use of a file system.

Describing the whole system in detail is beyond the scope of this paper. In the following, we will focus on the properties of SAP R/3 which are relevant for the execution of decision support queries: SAP R/3's overall architecture, database schema, query language, and other special features. We will also highlight the major differences between Releases 2.2 and 3.0 (the two different versions of SAP R/3 studies in this paper).

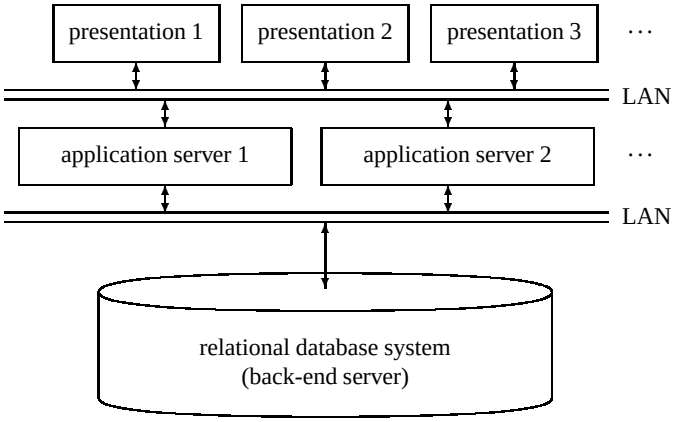

Figure 1: Three-Tier Client/Server-Architecture of SAP R/3

2.1 Architecture of SAP R/3

SAP R/3 [WHSH96, BEG96] is based on a three-tier client/server-architecture with the following layers (see Figure 1):

1. The presentation layer. It provides a graphical user interface (GUI).

2. The application layer. It comprises the business administration "know-how" of the system. It processes predefined and user-defined application programs for, say, OLTP and decision support.

3. The database layer. It is implemented on top of a (second party) commercial database product that stores all data of the system, as described above.

In a small company that uses SAP R/3, the application servers and the database system could be installed on the same middle-range machine and users would enter business transactions or issue decision support queries using their PCs. Such a configuration, however, is not practical for large companies with a very high volume of data and transactions. In such companies, all application servers and the database system would be installed on separate dedicated machines. To this end, SAP R/3 has been ported to a large variety of hardware and operating system platforms, and it is also operational on a number of commercial RDBMSs.

2.2 Data Model and Schema of SAP R/3

SAP R/3 is a comprehensive and highly generic business application system that was designed for companies of various organizational structures and different lines of business (e.g., production, retailing, ...). This genericity and comprehensiveness resulted in a very large company data model with

over 8100 logical tables in Version 2.2 and 10055 tables in Version 3.0 of our "vanilla" configuration of SAP R/3. To manage the meta data (e.g., types and interrelationships) of these tables, SAP R/3 maintains its own data dictionary which is (like all other data) stored in SAP's relational database and which can be used by SAP application programs.

There are three different kinds of (logical) SAP tables; they differ in the way they are mapped to (physical) tables of the RDBMS. So-called *transparent* tables are mapped 1:1 onto RDBMS tables. They are registered in the RDBMS's schema and can be accessed directly on the RDBMS without using SAP R/3's query interfaces. While it might be (under certain circumstances) practical to read these tables directly from the RDBMS, it is not reasonable to update these tables without consulting SAP R/3's application programs because, in general, the user cannot anticipate how SAP R/3 would propagate such updates to other SAP tables in order to keep the database in a consistent state.

The other two kinds of SAP tables are so-called *pool* and *cluster* tables. Several SAP pool tables are bundled and stored in a single table of the RDBMS; every tuple of this RDBMS table corresponds to a logical tuple of one of the SAP pool tables. SAP cluster tables are also bundled; here, SAP R/3 aims at storing logically related tuples into a single tuple of the RDBMS table: it is possible to bundle tuples of several different SAP tables, and it is also possible to bundle several tuples of a single cluster table into one tuple of an RDBMS table. In any case, the bundling of tuples and relations makes it impossible to access pool and cluster tables directly on the RDBMS. Pool and cluster tables are, therefore, *encapsulated* by SAP R/3: they can only be accessed using SAP R/3's query interfaces because access to these tables requires retrieving decoding information stored in SAP's data dictionary.

It seems that encapsulated tables are "remains" of times when the functionality and performance of relational database products was insufficient for the specific requirements of SAP R/3. Nowadays, SAP pursues the strategy to move more and more data into transparent tables because—as we will see—transparent tables can be accessed more efficiently than encapsulated tables. This trend is reflected by the schema differences between SAP's Releases 2.2 and 3.0. In Release 2.2, about 6300 of the 8100 SAP tables are pool and cluster tables whereas in Release 3.0, only 2370 of the 10055 SAP tables are encapsulated. Furthermore, SAP R/3 allows users to convert any pool and cluster table into a transparent table in Release 3.0, whereas in Release 2.2 only pool tables can be converted into transparent tables.

In addition to SAP's pre-defined relations, users can define views. These views can, for example, simplify the formulation of (business) queries. Like transparent tables, an SAP view is mapped 1:1 to an RDBMS view.

2.3 ABAP/4

Applications of the SAP R/3 system are coded in the programming language ABAP/4 (**A**dvanced **B**usiness **A**pplication **P**rogramming Language) [Mat96]. Except for a small kernel,

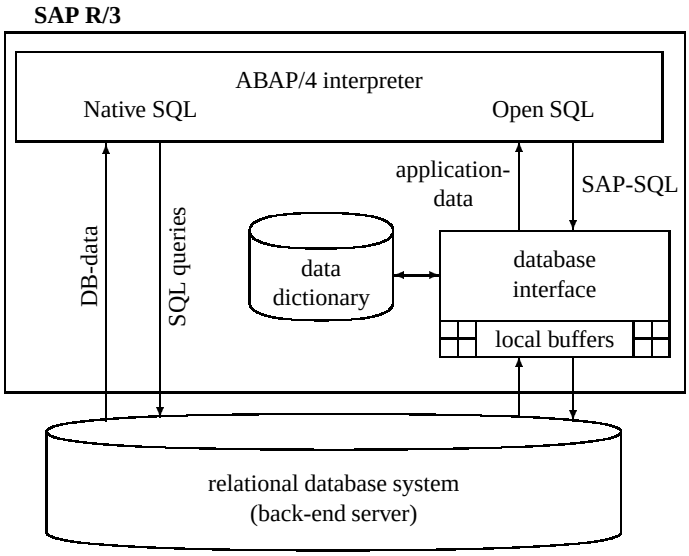

Figure 2: Database Interface of ABAP/4

actually the entire R/3 system is coded in ABAP/4. ABAP/4 is a so-called Fourth Generation Language (4GL) whose origins can be found in report/application generator languages. For this reason, ABAP/4 programs are often called reports. In the course of the R/3 evolution, ABAP/4 was augmented with procedural constructs in order to facilitate the coding of more complex business application programs. For example, ABAP/4 contains language constructs to program so-called "Dynpros" which are dialog programs with a graphical user interface including the logic for validating and processing user entries.

ABAP/4 is an interpreted language, which makes it very easy to integrate new ABAP/4 application programs into the system. Like ordinary data, all ABAP/4 application programs are managed by the R/3 data dictionary and the program code is stored in the SAP database.

As sketched in Figure 2, ABAP/4 provides commands that allow to access the database via two different interfaces: *Native SQL* and *Open SQL*. The Native SQL interface can be addressed by so-called EXEC SQL commands. It allows the user to access the SAP database directly without using the SAP-internal data dictionary. The advantage is, that the database system-specific properties and services (e.g., non-standard SQL statements) can be fully exploited and additional overhead by SAP R/3 is avoided. However, using the Native SQL interface incurs some severe drawbacks: (1) The EXEC SQL commands may be database system-specific which renders non-portable ABAP/4 programs. (2) By circumventing the SAP-internal data dictionary, EXEC SQL commands cannot access encapsulated relations. (3) Native SQL reports are potentially unsafe because Native SQL directly reads database relations, and the implementor of a Native SQL report might overlook intrinsic business process interpretations which are otherwise carried out implicitly by SAP R/3's application programs; that is, bypassing SAP R/3's

data dictionary requires expert knowledge about the rules and dependencies of the system.

Safe and portable ABAP/4 reports can be written by relying exclusively on ABAP/4's Open SQL commands. In Open SQL reports, access to tables or views of the SAP database can be coded using two very similar ABAP/4 SELECT-constructs:

```
SELECT ⟨attribute list⟩                SELECT SINGLE ⟨attribute list⟩
FROM ⟨one table⟩                       FROM ⟨one table⟩
WHERE ⟨predicate⟩                      WHERE ⟨unique predicate⟩
... process current tuple              ... process the only tuple
ENDSELECT.
```

The basic SELECT command accepts any kind of predicate in its WHERE-clause. The SELECT SINGLE command, on the other hand, requires predicates on *unique* fields of a table so that at most one tuple qualifies and is returned for further processing.

Query Facilities of R/3 Release 2.2 In Release 2.2, both SELECT commands are restricted to a single SAP table or view. That is, unless a (join-)view is defined, it is not possible to implicitly describe a join, as is possible in SQL by referencing several relations in the FROM-clause. Join views can only be formulated over transparent tables and only along primary key/foreign key relationships.

To evaluate a general join within the Open SQL interface, the implementor has to code an ABAP/4 program with nested SELECT...ENDSELECT or SELECT SINGLE loops. This is demonstrated in the following program fragment:

```
SELECT ⟨attribute list⟩
FROM ⟨outer table⟩
WHERE ⟨simple predicate⟩.
      SELECT ⟨attribute list⟩
      FROM ⟨inner table⟩
      WHERE ⟨join predicate⟩.
      ... processing of the current inner (and outer) tuple
      ENDSELECT.
... processing of the current outer tuple
ENDSELECT.
```

Such a program evaluates the join of the tables, without making use of the join methods of the underlying database system. In essence, it corresponds to an (index) nested loops join with the additional overhead of "crossing" the interface between database system and ABAP/4 program for every tuple of the outer relation in order to find the matching tuples of the inner relation.

Furthermore, groupings and aggregations cannot be incorporated into the Open SQL SELECT statements of Release 2.2. As a consequence, all groupings and aggregations have to be performed by the SAP system, thereby possibly transferring huge amounts of data from the RDBMS to the SAP system.

Extended Query Facilities of R/3 Release 3.0 Very recently, SAP has incorporated joins into their Open SQL interface. A join query is formulated as follows:

```
SELECT ⟨attribute list⟩
FROM ⟨table1⟩ JOIN ⟨table2⟩
      ON ⟨join predicate⟩
WHERE ⟨predicate⟩.
```

It is even possible to specify left outer joins in this SQL92-style syntax. However, users of SAP R/3 cannot yet use this feature because not all RDBMSs support outer joins.

SAP has also incorporated grouping and simple aggregation into the Open SQL SELECT statements. The join, grouping and aggregation operations within a SELECT statement are delegated to the underlying RDBMS. Therefore, these operations can benefit from the RDBMS's join and groupby methods, but they can only be applied on transparent tables and not on pool and cluster tables.

Unfortunately, it is only possible to implement simple aggregations on a single attribute of a table with the new Open SQL construct for aggregations; for example, an aggregation cannot contain an arithmetic expression which is needed, for example, to total the discounted price of orders.

Optimization Features of SAP R/3 To optimize query processing, SAP R/3 implements two techniques which take effect if the Open SQL interface is used: (1) *Cursor caching* which reduces the overhead of calls to the RDBMS by using the same cursor for, say, all the queries that retrieve the matching tuples of the inner relation in a nested SELECT statement of a Version 2.2 Open SQL report. Cursor caching is possible because most database systems allow parameterized queries and provide a cursor REOPEN command in their API. (2) *Caching* data in SAP R/3 application servers in order to avoid calls to the RDBMS altogether (cf. Figure 2). For caching, the typical tradeoffs between read and update frequency apply; in addition, SAP R/3 does not fully guarantee cache coherency in a distributed environment as updates are only propagated periodically.

Also, ABAP/4 allows the materialization of query results in internal (i.e., temporary) tables in order to use this data for further processing. For example, it is possible to materialize the inner relation of a nested-loops join of a 2.2 Open SQL report and avoid repeated calls to the RDBMS this way. It is, however, not possible to define indexes on temporary tables.

2.4 Batch Input

In addition to queries, we also studied the performance of data manipulation operations. Typically, users enter transactions (e.g., new orders) interactively using SAP R/3's GUI. To insert (or delete or manipulate) large volumes of data SAP R/3 provides a so-called *batch-input* facility. The procedures of this facility read data records from an external file and "simulate" an interactive entry of data. In particular, the batch-input procedures invoke all SAP R/3 application programs that interpret and check the consistency of the input data. Therefore, the batch-input facility was an ideal mechanism to study the performance of data manipulation operations in SAP R/3.

2.5 Data Warehousing with SAP's EIS

SAP offers a product called *Executive Information System* (EIS) which is designed for evaluating complex, predefined queries. This product uses a data warehouse approach [FS96]; i.e., the information is extracted from the SAP database and inserted into separate data structures. The advantage of this approach is that the data can be pre-processed (e.g., aggregated) into a specialized format and particular operators (like the data cube [GBLP96]) can be used to query the data. The disadvantage is that the data has to be extracted from the SAP database using SAP R/3's query interfaces (i.e., Native or Open SQL). In Section 5 we report on experiments that measure the costs of this data extraction.

3 Benchmarking SAP R/3 with TPC-D

This section presents results of performance experiments with the TPC-D benchmark. We have implemented the TPC-D benchmark in SAP R/3 using both Native SQL and Open SQL: first using the limited features of Release 2.2G and after upgrading our installation, also using the extended features of Release 3.0E. For comparison, we have also implemented the TPC-D benchmark directly on a commercial RDBMS using (standard) SQL. Before presenting the results, we will briefly describe the TPC-D benchmark, and how we implemented it in SAP R/3. SAP has also designed its own set of benchmarks [LM95]; we use TPC-D because it is a standard benchmark widely known in the database community and because the purpose of the SAP benchmarks is to study OLTP-style business processes (e.g., generation of invoices) rather than complex decision support queries.

3.1 Overview of the TPC-D Benchmark

The TPC-D benchmark was designed to evaluate relational database systems for Decision Support in business-oriented applications [TPC95]. The benchmark database has eight tables: REGION, NATION, SUPPLIER, PART, PARTSUPP, CUSTOMER, ORDER, and LINEITEM. The benchmark takes a *scaling factor* as parameter which determines the size of the tables; in all our experiments, we set this scaling factor to 0.2 so that the two largest tables, ORDER and LINEITEM, had 300,000 and 1.2 million tuples.

In addition to the database, the TPC-D benchmark defines 17 queries and 2 update functions. In this study, we chose to carry out the TPC-D power test which specifies to execute all queries and update functions one at a time and measure their running time individually.[1] The TPC-D queries and update functions test a variety of features of a database system. The query suite, for example, ranges from a simple single-table query to a complex eight-way join query. The update functions carry out *insert* as well as *delete* operations. The entire benchmark is motivated and described in full detail in [TPC95].

[1] There is also a TPC-D throughput test which allows the concurrent execution of queries and update functions. Since we were interested in the basic query processing mechanisms of SAP R/3, we concentrated on the simpler TPC-D power test.

3.2 Implementing the TPC-D Benchmark in SAP R/3

SAP R/3 is capable of managing the data of several business clients (also called business units or mandatory) of a multi-national company. To implement the TPC-D benchmark in SAP R/3, we created a new business client called TPC-D Inc. When a new business client is created in SAP R/3, SAP R/3 implicitly creates a new (logical) SAP database, and it is possible to insert, say, customer, supplier, or order information for that business client into SAP's pre-defined database tables. For the purpose of our experiments, we were able to load all the TPC-D data into those standard pre-defined tables, and then implement the TPC-D queries and update functions using Native SQL, Open SQL, and SAP's batch-input facility as described in Sections 2.3 and 2.4.

Table 1 gives a short characterization of those of the more than 8000 pre-defined SAP tables which were actually used to store the TPC-D data. It becomes apparent that the original TPC-D tables are vertically partitioned in SAP R/3: the TPC-D records are stored in a total of 17 rather than eight SAP tables. Such a partitioning is necessary to take many practical issues of business applications into account. For example, all text fields such as comments and descriptions are stored in separate tables in order to store explanatory text in different languages at the same time. (Keep in mind that SAP R/3 was designed as a global application system for multi-national enterprises.)

SAP Table	Description	Orig. TPC-D Tab.
T005	Country: general info	NATION
T005T	Country: Names	NATION
T005U	Regions	REGION
MARA	Parts: general info	PART
MAKT	Parts: description	PART
A004	Parts: terms	PART
KONP	Terms: positions	PART
LFA1	Supplier: general info	SUPPLIER
EINA	Part-Supplier: general info	PARTSUPP
EINE	Part-Supplier: terms	PARTSUPP
AUSP	properties	PART, SUPP, PARTS
KNA1	Customer: general info	CUSTOMER
VBAK	Order: general info	ORDER
VBAP	Lineitem: position	LINEITEM
VBEP	Lineitem: terms	LINEITEM
KONV	Pricing Terms	LINEITEM
STXL	Text of comments	all

Table 1: SAP Tables used in the TPC-D Benchmark

Of the 17 SAP tables that are used in the TPC-D benchmark, SAP R/3 encapsulates by default the *A004* and *KONV* tables (*A004* is a pool table and *KONV* is a cluster table). *KONV*, for example, is used in many TPC-D queries because it records the *discount* and *tax* of a lineitem. While we could

not convert the *KONV* table into a transparent table in our 2.2G installation, we did convert it in our 3.0E installation because, as stated in Section 2.3, encapsulated tables such as the *KONV* can only be read using Open SQL commands, and queries involving encapsulated tables often show poor performance. The other 16 tables used in our TPC-D experiments were identical in our 2.2 and 3.0 configurations.

In order to avoid the partitioning of data and completely avoid the use of encapsulated tables, it would, of course, have been possible to extend the SAP schema and create a new SAP table for every table of the original TPC-D database. The purpose of this study, however, was to store the business data of the TPC-D benchmark in the same way as, say, a wholesaler would do in the "real world" in order to measure the "real" data processing performance of SAP R/3.

3.3 Details of the Experimental Environment

As mentioned earlier, we carried out the TPC-D power-test using Releases 2.2G and 3.0E of SAP R/3 because both releases are widely used today and because a comparison of the results shows the benefits that users of 2.2G can achieve by upgrading their system and rewriting their reports to exploit the extensions provided by 3.0E. For both sets of experiments, the test environment was essentially the same (with minor exceptions). The whole package of SAP R/3 and relational database system was installed on a Sun SPARC station 20/612MP with two 60 MHz microprocessors and 192 MB main memory. The system software and the test database were stored on four 4 GB Seagate ST15230N disk drives. The operating system was Solaris 2.5. As part of the upgrade from Version 2.2G to 3.0E, we also had to upgrade the hardware to 256 MB main memory and five disk drives. Although possible, we did not use dedicated machines for SAP R/3 and the database management system in any of our experiments because we wanted to avoid communication costs and interference with other users of the network and because we only considered the TPC-D power test in which no benefits from the use of more than one machine can be obtained since all queries and update functions are executed one at a time.

We installed and configured the relational database system as part of the regular SAP R/3 installation procedure[2]. It should be noted that by default SAP R/3 turns off several optimization features of the RDBMS—probably because they are not useful for the kind of workloads initially envisioned by the developers of SAP R/3. We partially turned these optimization features back on because they significantly improved the running time of the TPC-D queries. In general, however, we tried not to change any parameters of SAP's default configuration. In particular, SAP R/3 allocates by default 10 MB of main memory for the buffer of the RDBMS, and reserves the remaining main memory for its own processes; we did not change this parameter and carried out all experiments with 10 MB of database buffer even if we ran queries directly on the RDBMS and the SAP R/3 processes were idle.

We carried out the TPC-D power test following the execution rules of the TPC-D benchmark specification [TPC95]. In particular, we validated the correctness of the implementation of all our programs using a TPC-D test database with scaling factor 0.1, and we executed the queries and update functions in the order specified by the TPC-D power test.

3.4 Results

3.4.1 Size of the Database

As stated above, we performed all our experiments using a TPC-D database with scaling factor SF=0.2. We generated the records of this database using the DBGEN tool provided by TPC. For SF=0.2, the DBGEN tool generates an ASCII file of about 200 MB. We loaded the records of this ASCII file into the SAP database of our TPC-D Inc (at the time using version 2.2G), and for comparison, we also loaded the records directly into the RDBMS (without using SAP R/3) in order to create an *original* TPC-D database (i.e., with eight tables for REGION, NATION, etc.).

Table 2 shows the size of the resulting SAP and *original* TPC-D databases.The SAP database has about 10 times the size of the original TPC-D database for the following three reasons: (1) As mentioned earlier, the SAP database is strongly partitioned in order to support specific requirements of business applications. (2) For a similar reason, the SAP tables contain many fields which are not accounted for in the TPC-D benchmark; in our experiments, these fields were implicitly given default values by SAP R/3. (3) SAP R/3 uses 16 Byte *strings* rather than 4 Byte *integers* to represent key attributes such as, say, *orderkey*.

	Original TPC-D DB		**SAP DB (Version 2.2)**	
	Data	**Indexes**	**Data**	**Indexes**
REGION	16	0	320	400
NATION	16	0	400	400
SUPPLIER	451	120	2.127	1.884
PART	6.144	1.792	79.485	83.525
PARTSUPP	32.310	5.275	102.045	44.455
CUSTOMER	7.929	1.463	37.805	26.355
ORDER	52.578	21.312	399.190	125.243
LINEITEM	171.704	72.860	2.191.844	558.746
Total	271.139	102.822	2.813.216	841.008

Table 2: DB Sizes in KB: Original TPC-D DB and SAP DB

In addition to the raw data, Table 2 also shows the space required to store the indexes that are defined for the original TPC-D DB and SAP DB. Although both databases have an equivalent set of indexes, the SAP indexes require eight times as much space. Again, the increased storage consumption of SAP R/3 is due to the strong vertical data partitioning which results in a large number of primary key indexes and in the use of *strings* instead of *integers* which results in an increased

[2] Because of our license agreement, we are not allowed to publish the name of the vendor and details of the RDBMS.

size of every individual index.

The upgrade to Release 3.0E inflated the size of the SAP database by another 10%. (The 3.0E database including indexes consumes about 4 GB). Most of this increase was due to the conversion of the *KONV* table which tripled its size from about 200 MB to about 600 MB.

3.4.2 Loading the SAP Database

Table 3 shows how long it took to load the SAP database using version 2.2G. As stated in Section 2.4, SAP provides a batch-input facility for this purpose, and we used this facility to load the records of six out of the eight TPC-D tables. We typed in the data for REGION and NATION interactively because these tables were very small (5 and 25 records), and therefore, do not list loading times for these two tables. ORDERs and their LINEITEMs can only be loaded jointly into the SAP database; so Table 3 lists only one entry for these two tables.

	Loading Time
REGION	—
NATION	—
SUPPLIER	18m
PART	15h 56m
PARTSUPP	30h 24m
CUSTOMER	7h 33m
ORDER+LINEITEM	25d 19h 55m

Table 3: Loading the SAP Database
Two Parallel Batch-Input Processes

In our hardware configuration, it was possible to tune the loading of the database by running two batch-input processes that loaded records in parallel. Nevertheless, it took about a month to load the whole SAP database (including indexes). This extremely high loading time can be explained since SAP R/3 carries out consistency checks for every record of the batch-input individually. These consistency checks are very expensive, and as another consequence, SAP inserts the records a tuple at a time into the database and does not exploit the bulk loading interface of the RDBMS.

We did not have to re-load the database for the 3.0E upgrade so we do not know how long it takes to load a database in that version. In addition to backups, preparations, etc., the upgrade itself with some database reorganization took about two weeks in which the system was not operational and in which SAP R/3 upgrade routines were constantly active.

3.4.3 TPC-D Power Test (Release 2.2G)

Table 4 shows the results of the TPC-D power test using SAP R/3 Release 2.2G. The table contains the running times of Open SQL and Native SQL on the SAP database and, as a baseline, of the isolated RDBMS on the original TPC-D database. Looking at the query performance only (Q1-Q17), the total running time of the Native SQL reports is about four times as high as that of the isolated RDBMS. The most prominent reason for this is the strong partitioning of the data in the SAP database; for example, Query 1 is a single-table query in the original TPC-D DB whereas it is a 5-way join query in the SAP DB. In part, Native SQL also performs poorly because the *KONV* table is an encapsulated table and several queries cannot be fully pushed down to the RDBMS; instead these queries are broken down and joins with the *KONV* table are implemented using nested SELECT statements and thus are evaluated at higher cost by the SAP application server (see Section 2.3).

In Release 2.2G, pure Open SQL reports show significantly worse performance than the Native SQL reports. The reason for this is quite simple: in addition to joins with the *KONV* table, several other joins and aggregations had to be executed by the SAP application server resulting in particularly poor performance for queries Q3, Q6, Q9, and Q12. Recall that using the Open SQL interface in Release 2.2G, joins can only be pushed down to the database system by the means of defining appropriate join views. We made extensive use of this feature; however, it was not always possible to define join views in SAP R/3 because join views could only be defined on transparent tables and only along primary/foreign key relationships.

It should be noted that throughout these experiments we tried to implement all queries and reports in the best possible way. This involved a significant amount of manual tuning because, for example, the optimizer of the RDBMS did not always automatically generate an acceptable query execution plan in all three implementation strategies.

Turning to the running times of the TPC-D update functions: In both SAP variants, we implemented the update functions using SAP R/3's batch-input facility so that these two variants show virtually identical performance. As studied in the previous subsection (Table 3), SAP R/3 carries out expensive, tuple-level consistency checks so that the running time of SAP's batch-input is significantly higher than the running time of a program that directly inserts/deletes tuples into/from the database.

3.4.4 TPC-D Power Test (Release 3.0E)

Table 5 shows the results of the TPC-D power test using SAP R/3 Release 3.0E. As stated before, the upgrade involved a slight upgrade of our hardware and some database reorganization, and it also involved upgrading the RDBMS. As a result, the performance of the isolated RDBMS was slightly better in our 3.0 than in our 2.2 experiments. After the upgrade, we furthermore deleted one index (the index on *shipdate* of *lineitems*) that SAP R/3 creates by default, but which was counterproductive to execute the TPC-D power test in our 3.0 configuration. Despite all these changes, we are convinced that the 3.0E numbers shown in Table 5 are directly comparable to the 2.2G numbers shown in Table 4 and the differences in performance are mostly due to the new features of Release 3.0E; i.e., the extensions to the Open SQL interface and making it possible to convert *KONV* into a transparent table. To take advantage of these features, we had to completely re-code and re-tune all our Native and Open

Query Update	RDBMS (TPCD-DB)	Native SQL (SAP DB)	Open SQL (SAP DB)
Q1	5m 17s	2h 14m 56s	2h 15m 33s
Q2	34s	1m 16s	3m 19s
Q3	5m 55s	19m 42s	3h 12m 57s
Q4	3m 01s	7m 12s	8m 31s
Q5	21m 13s	22m 05s	1h 08m 22s
Q6	1m 18s	8m 22s	10m 52s
Q7	5m 02s	39m 13s	38m 31s
Q8	2m 44s	16m 02s	28m 26s
Q9	9m 14s	36m 06s	2h 31m 36s
Q10	5m 00s	22m 42s	25m 41s
Q11	5s	2m 02s	1m 55s
Q12	2m 59s	36m 35s	1h 17m 25s
Q13	8s	21s	23s
Q14	5m 01s	9m 13s	11m 27s
Q15	3m 46s	12m 24s	19m 18s
Q16	15m 00s	8m 56s	8m 29s
Q17	14s	9m 12s	12m 07s
UF1	1m 59s	44m 26s	44m 26s
UF2	1m 48s	8m 49s	8m 49s
Total (quer.)	1h 26m 31s	6h 26m 19s	13h 14m 52s
Total (all)	1h 30m 18s	7h 19m 34s	14h 08m 07s

Table 4: TPC-D Power Test, SAP R/3 Version 2.2G

Query Update	RDBMS (TPCD-DB)	Native SQL (SAP DB)	Open SQL (SAP DB)
Q1	6m 09s	58m 59s	56m 18s
Q2	53s	3m 09s	34s
Q3	4m 03s	9m 02s	11m 51s
Q4	1m 45s	6m 18s	6m 38s
Q5	6m 39s	14m 42s	37m 27s
Q6	1m 20s	7m 28s	14m 06s
Q7	9m 03s	23m 05s	29m 24s
Q8	1m 54s	19m 04s	16m 37s
Q9	8m 42s	31m 33s	1h 7m 14s
Q10	5m 18s	33m 06s	57m 49s
Q11	5s	4m 37s	2m 23s
Q12	3m 15s	9m 48s	9m 36s
Q13	8s	19s	25s
Q14	6m 23s	10m 25s	21m 54s
Q15	3m 25s	13m 51s	28m 31s
Q16	13m 24s	3m 16s	3m 22s
Q17	11s	1m 50s	2m 13s
UF1	1m 40s	1h 46m 54s	1h 46m 54s
UF2	1m 48s	11m 35s	11m 35s
Total (quer.)	1h 12m 37s	4h 10m 32s	6h 06m 22s
Total (all)	1h 16m 05s	6h 09m 01s	8h 04m 51s

Table 5: TPC-D Power Test, SAP R/3 Version 3.0E

SQL reports. It is very important to keep in mind that the old 2.2G Native and Open SQL reports were operational in 3.0E, but they had virtually the same performance in 3.0E as in 2.2G.

Looking at pure query performance (Q1-Q17), the new 3.0E Native SQL reports gained about two hours in total compared to the old 2.2G Native SQL reports; this improvement can be explained because the new Native SQL reports push all the queries completely down to the RDBMS which is possible because now all involved tables (in particular, *KONV*) are transparent. Furthermore, since *KONV* is not in the way anymore, the difference between Native SQL and the isolated RDBMS in Release 3.0E is entirely due to the partitioning of the data in the SAP database and the much higher data volume (Table 2).

Obviously, Open SQL benefited most from the upgrade. The new Open SQL reports for Q1-Q17 outperform the old ones by about 7 hours because of the new Open SQL join construct which allows to delegate all join processing to the RDBMS. Nevertheless, Open SQL is still outperformed by Native SQL for the following three reasons: (1) in some Native SQL reports a special, non-standard SQL string function of the RDBMS was used which could not be used in the Open SQL reports[3]; (2) for several Open SQL reports, the optimizer of the RDBMS did not find the best execution plan; and (3) complex aggregations (e.g., the sum of discounted and taxed prices of lineitems) could not be expressed using Open SQL's new syntax and, therefore, could not be pushed down to the RDBMS. We will study the second and third of these effects in more detail in the next section.

It is interesting to note that for some queries (Q2, Q11, Q16), Open SQL performs better than Native SQL and sometimes even better than the isolated RDBMS. All these queries involve nested sub-queries. In Open SQL, we explicitly unnested the sub-queries because Open SQL's SELECT statement does not allow the coding of nested queries in the FROM and WHERE-clauses. In the Native SQL reports and in the (standard) SQL implementation of the queries on the isolated RDBMS, we did not explicitly unnest the queries because these variants allowed to implement the queries using nested queries as specified by the TPC-D benchmark. It turned out that the RDBMS handled nested queries poorly so that the Open SQL reports with our explicit, manual unnesting showed better performance in these cases. (We also observed this same effect for Q16 in Release 2.2, cf. Table 4).

Turning to the running times of the TPC-D update functions. Again, these were implemented using SAP R/3's batch-input facility for Native and Open SQL. Actually, the reports used here were almost identical with the update reports used in the 2.2G experiments. Amazingly, UF1 (inserts) took more than one hour longer in our 3.0E configuration than in our 2.2G configuration. To date, we have not found a satisfactory explanation for this dramatic performance degradation, but we speculate that the conversion of *KONV* into a transparent table

[3] Using this string function made our Native SQL reports non-portable.

Native SQL:

```
... declarations
EXEC SQL PERFORMING OUTPUT.
SELECT KWENG, ...
FROM VBAP INTO :TAB
WHERE KWMENG < 0
        AND MANDT = '301'
ENDEXEC.
```

Open SQL:

```
... declarations
SELECT KWMENG, ...
FROM VBAP
WHERE KWMENG < 0
ENDSELECT.
```

Figure 3: Native and Open SQL Reports for a Simple Query on a *LINEITEM* Table

might be part of the reason: In our implementation of UF1, we used an SAP standard report which was specifically tuned for a clustered *KONV* table; it is possible that better performance could be achieved by re-writing this SAP standard report.

4 Performance Evaluation With Simple Queries

In the previous section, we saw that Native SQL reports outperform equivalent Open SQL reports in many situations. This observation suggests to use Native SQL as much as possible. Recall, however, that Native SQL is not recommended for many kinds of queries in practice because Native SQL programs may not be safe (in the business sense) and not portable. Pure Open SQL reports, on the other hand, are portable and safe so that it is worth to take a closer look where they loose performance. While the poor performance of the 2.2G Open SQL reports can fairly easily be explained (poor join processing), the performance penalties of the 3.0E Open SQL reports are more subtle. In this section, we will study these penalties by running simple Native and Open SQL reports on SAP R/3 Version 3.0E and by comparing the execution of these reports in detail. Furthermore, we will also study potential performance gains that can be achieved by the means of caching when using Open SQL.

4.1 Finding a Good Plan

In our first experiment, we studied how the translation of Open SQL reports to standard SQL queries impacts the generation of good query plans by the optimizer of the RDBMS. We measured a simple *select-from-where* query on a single table with an index; specifically we asked for lineitems (the *VBAP* table) with a certain maximum quantity (the `KWMENG` field which was indexed in this experiment). Figure 3 shows the corresponding Native and Open SQL reports for this business query. Both reports are equivalent: the (additional) restriction `MANDT=301` is necessary in the Native SQL report; it specifies that we were only interested in results for our TPC-D Inc. business client. Specifying this predicate in the Open SQL report was not necessary because SAP generates this predicate automatically from the application context while translating the Open SQL report into standard SQL queries. The `MANDT=301` predicate, therefore, is a very good example that demonstrates why it is often not safe to write Native SQL reports.

To find out whether the optimizer was always able to decide whether the use of the index on `KWMENG` (quantity) was beneficial and find the best plan for this query, we ran the Native and Open SQL reports of Figure 3 and varied the selectivity of the predicate on `KWMENG`. Table 6 shows the cost of the Native and Open SQL reports in the following two extreme situations:

1. No Result Tuple. This was achieved by restricting the resulting lineitems to have a quantity smaller than 0 (i.e., `KWMENG < 0`, as in Figure 3).

2. 1.2 Mio. Result Tuples (i.e., all lineitems qualify). This was achieved by restricting the resulting lineitems to have a quantity smaller than 9999 (`KWMENG < 9999`).

In the first case (high selectivity, no result tuples), the use of the index resulted in best performance; only the index needed to be consulted to find out that no tuple qualifies. In the execution of both reports, Native and Open SQL, the index was used so that both reports have a very low response time of less than a second in this case.

selectivity	**Native SQL**	**Open SQL**
high (0 result tuples)	<1s	<1s
low (1,2 mio. result tuples)	4m 56s	1h 50m 02s

Table 6: Cost of a One-Table Query
Index on `KWMENG` Available

If, however, the selection predicate is not selective the (unclustered) index on the `KWMENG` attribute should not be used because the use of the index results in random disk I/O to fetch the result tuples. To execute the Native SQL report, the entire query inside the EXEC SQL...ENDEXEC-delimiters is directly passed to the database system. The query optimizer of the database system finds out that, in this case, the usage of the index is counterproductive and generates an (optimal) query evaluation plan based on a full table scan which has a running time of about 5 minutes. Prior to execution, the Open SQL report, on the other hand, is *translated* by the SAP R/3 query processor. Due to this translation, the optimizer of the RDBMS cannot estimate the selectivity of the predicate of the translated query and thus *blindly* generates a plan. In this particular case, the optimizer chose to use the index, and as a result, the execution of the Open SQL report took almost 2 hours.

Precisely, SAP translated the Open SQL report as follows:

```
SELECT ...                    SELECT ...
FROM VBAP              ⇝      FROM VBAP
WHERE KWMENG < 9999           WHERE KWMENG < ?
ENDSELECT.
```

That is, the query is translated into a parameterized query, and "?" denotes the query parameter. SAP R/3 translates Open SQL reports in this generic way in order to carry out *cursor*

Native SQL:

```
... declarations
EXEC SQL PERFORMING EXTRACT.
SELECT KPOSN,
        AVG(KAWRT * (1 + KBETR/1000))
INTO :SUMME FROM KONV
WHERE MANDT = '301'
        AND STUNR = '040
        AND ZAEHK = '01'
        AND KSCHL = 'DISC'
GROUP BY KPOSN
ORDER BY KPOSN
ENDEXEC.
```

Open SQL:

```
... declarations
FIELD-GROUPS: HEADER, LINE.
INSERT TABELLE-KPOSN INTO HEADER.
INSERT CHARGE INTO LINE.
SELECT KPOSN KBETR KAWRT
INTO TABELLE
FROM KONV
WHERE STUNR = '040'
        AND ZAEHK = '01'
        AND KSCHL = 'DISC'
ORDER BY KPOSN.
CHARGE = TABELLE-KAWRT * (1 + TABELLE-KBETR/1000).
EXTRACT LINE.
ENDSELECT.
SORT. LOOP.
        COUNT = COUNT +1.
        AT END OF TABELLE-KPOSN.
                AVG = SUM(CHARGE) / COUNT.
                WRITE : / TABELLE-KPOSN, AVG.
                COUNT = 0.
        ENDAT.
ENDLOOP.
```

Figure 4: Native and Open SQL Reports for a Query with Grouping, Sorting and Aggregation

caching as described in Section 2.3 (i.e., reduce the overhead of the repeated execution of similar queries).

Of course, we made sure in a separate set of experiments, which are not shown here, that the difference in performance was really due to the choice of the access path by the optimizer of the RDBMS and not due to inefficiencies in transferring result tuples from the RDBMS to the SAP application server. The performance of shipping result tuples from the RDBMS to SAP is the same regardless of whether the Native or Open SQL interface is used.

4.2 Complex Aggregations

In the second experiment, we study a group-by query with a complex aggregation; i.e., an aggregation with an arithmetic expression. As stated in Section 2.3, the syntax of the Open SQL SELECT and SELECT SINGLE statements currently does not allow to express such complex aggregations so that complex aggregations cannot be pushed down to the RDBMS and must be coded and executed in SAP instead. The simple query we used to study the implications of this limitation of Open SQL lists the average discounted volumes of order positions. We used this rather contrived query because its cost is dominated by the grouping operation (on a lineitem table) and we could, thus, isolate the effects of inefficient group-by processing. More realistic queries with complex aggregations can be found in the TPC-D query suite—all the TPC-D queries with complex aggregations, however, involve expensive joins. The Native and Open SQL reports for our example query are shown in Figure 4, and the running times of these reports are given in Table 7.

	Native SQL	**Open SQL**
cost	4m 11s	13m 48s

Table 7: Costs for Grouping Tuples

In this experiment, the cost of the Open SQL report is more than three times as high as that of the Native SQL report. There are two reasons why executing aggregations in SAP is more expensive than pushing them down to the RDBMS: (1) to compute the aggregation in SAP, all the required *KONV* tuples must be shipped from the RDBMS to SAP, whereas only the few aggregation values of the resulting groups need to be shipped if the aggregation is computed by the RDBMS. (2) Aggregations in SAP proceed in two separate steps: first, sorting and writing the sorted result to secondary storage, and then re-reading the sorted table to perform the grouping. The RDBMS, on the other hand, does not require to write intermediate results to secondary storage after sorting because sorting and grouping can be carried out in a pipelined fashion.

We expect that extensions provided by future releases of SAP R/3 will make it possible to express complex aggregations as part of an Open SQL SELECT or SELECT SINGLE statement. As a result, Open SQL will perform just as well as Native SQL for queries with complex aggregations and, in addition, it will become much simpler to implement Open SQL reports for such queries.

4.3 Caching

The previous experiments showed that Native SQL reports often show better performance than Open SQL reports because the RDBMS can effectively optimize queries of Native SQL's

EXEC SQL statement and because complex aggregations can usually be carried out more efficiently in the RDBMS. In this section, we will study potential performance benefits of Open SQL reports that can be achieved by the means of caching data in SAP R/3 application servers. (This kind of caching cannot be exploited in the execution of Native SQL reports; Section 2.3.) Caching is particularly effective if mostly "small" queries are executed; for example, when a sales person enters orders, repeatedly queries to retrieve information about a specific part are issued. If this part is cached, these queries can be executed with no interaction with the database system at all.

```
... declarations
SELECT * FROM VBAP.
    SELECT SINGLE *
    FROM MARA
    WHERE MATNR = VBAP.MATNR.
ENDSELECT.
```

Figure 5: Open SQL Report to Study Caching

To examine the impact of caching on the performance of "small" queries, we executed the Open SQL report of Figure 5. This report carries out a join between *VBAP* (LINEITEM) and *MARA* (PART). The report is programmed in such a way that with every tuple from *VBAP* a separate query to find the matching tuple of *MARA* is evaluated; in all, 1.2 million "small" queries to *MARA* are evaluated. We executed the report in the following three different configurations:

1. Caching of *MARA* not activated: In this case, every *MARA* query is processed by the RDBMS.

2. Caching of *MARA* activated, 2 MB cache: For every *MARA* query, SAP R/3 first inspects the cache to find out whether the requested *MARA* tuple can be obtained from the cache. If this is the case (hit), the tuple is read from the cache without interacting with the RDBMS, otherwise the *MARA* query is passed to the RDBMS, as in 1.

3. Caching of *MARA* activated, 20 MB cache: Processing the query proceeds, as in 2. The difference is that a much higher hit ratio is achieved due to the larger cache.

	No Caching	**2 MB Cache**	**20 MB Cache**
hit ratio	0%	11%	85%
costs for querying *MARA*	1h 48m 34s	1h 50m 51s	35m 41s

Table 8: Effectiveness of Caching
Open SQL Report of Fig. 5; Caching of *MARA* (PART) Varied

Table 8 lists the hit ratio (on tuples of *MARA*) and the costs of the 1.2 million small *MARA* queries[4] for the three different configurations. For the small cache of 2 MB, the overhead of cache management and the testing whether or not a required tuple was resident were about as high as the gains that were achieved by using the cache because only very few hits could be achieved. With a large cache of 20 MB, on the other hand, nearly all tuples of *MARA* could be cached and, therefore, the costs for querying *MARA* 1.2 million times were reduced by a factor of 3.

[4]The cost for querying *MARA* was computed by subtracting the cost to process the *VBAP* tuples from the total cost of the report shown in Figure 5.

5 Construction of a Data Warehouse

Let's go back to the TPC-D power-test results shown in Tables 4 and 5 of Section 3.4. It was seen that decision support queries cannot be evaluated in the most efficient way using the SAP database, regardless of whether Native or Open SQL is used. To achieve the same (or even better) performance as can be obtained by using an RDBMS on the original TPC-D database, one would have to construct a so-called data warehouse [FS96]. For this purpose, the data is extracted from the SAP database and stored in a separate database, which is under control of an RDBMS or a specialized data warehouse system [Col96]. To decide whether a data warehouse approach pays off for SAP customers (e.g., the use of the SAP product EIS), the customer needs to consider the initial cost for constructing the data warehouse and the maintenance costs for incrementally propagating updates (insertions, deletions and modifications) to the data warehouse. Extracting the data from the SAP database requires the execution of extremely complex (Native or Open SQL) reports.

To measure the cost of extracting data, Table 9 lists the costs of Open SQL reports that given our SAP database (SF=0.2) reconstruct the original TPC-D database into ASCII files, again using version 3.0E. (The running time of equivalent Native SQL reports is almost identical). The total running time of these reports was more than 6 hours and is, thus, about as high as the cost to execute all queries of the TPC-D power test—when using Open SQL reports in version 3.0E. Consequently, the construction of a data warehouse would only pay off if many more and/or more complex queries are issued against the data warehouse than those of the TPC-D power test.

	running time
REGION	13s
NATION	4s
SUPPLIER	41s
PART	12m 31s
PARTSUPP	11m 08s
CUSTOMER	5m 55s
ORDER	57m 31s
LINEITEM	4h 37m 02s
total	6h 05m 05s

Table 9: Costs for Constructing an SAP Data Warehouse
Open SQL Reports, SAP R/3 Version 3.0E

6 Conclusion

In this work, we have shown that the performance analysis of isolated database systems can have only limited relevance in the "real world" of data processing. The end users typically employ a comprehensive application system, in which the database system is an integrated component. We showed that the standard TPC-D benchmark for decision support is much more complex in the business reality than in its "synthetic" form: The data volume of the business processes modeled by the TPC-D benchmark is in reality (i.e., in SAP R/3) inflated by a factor of 10. Furthermore, the data is strongly partitioned so that an *n*-way join query of the synthetic TPC-D benchmark becomes in reality an $(x * n)$-way join query.

Based on these observations, we encourage hardware and database vendors to benchmark the performance of particular hardware/database/application system-configurations in the way we have done for SAP R/3 in this work. For this purpose, all our benchmark applications have been made public on our web server [DHKK]. We think that performance results obtained in this way better meet the expectations of end users who employ the integrated application system and not the isolated database system.

We also showed how application systems such as SAP R/3 exploit the advanced query processing facilities of state-of-the-art RDBMSs. The query interfaces of Release 2.2 forces users to implement reports in such a way that joins and aggregations are carried out at unacceptable high cost by the SAP R/3 application servers. Release 3.0 allows aggressive push-down of joins and certain aggregations to the RDBMS which resulted in drastically improved performance in our TPC-D experiments. While this is very good news, it should be noted that (1) even Release 3.0 does not allow to fully exploit all the features of today's database systems (not even all the standardized features), and (2) after an upgrade, current users of Release 2.2 need to re-write all their reports in order to take advantage of the new features of Release 3.0.

In this work, we also presented some initial experimental results to study a data warehouse approach for SAP R/3. The construction and maintenance of a data warehouse from the SAP database incurs a high cost because the initial and incremental extraction of data from the SAP database requires the execution of very complex reports (in addition to the actual data warehouse construction costs). In future work, we will study the tradeoffs of a data warehouse approach for SAP R/3 more comprehensively; in particular, we will study the performance that can be achieved by using SAP's data warehouse product EIS.

Acknowledgments We thank SAP for providing us with a free installation of R/3 and for helpful support during installation and customizing of our configuration. In particular, we would like to thank M. Härtig, U. Koch, B. Lober, and B. Schiefer of SAP for very detailed comments on a draft of this paper. K. Peithner and H. Zorn carried out the initial installation, and P. Kleinschmidt and his group at the University of Passau provided valuable help for the configuration and the upgrade of the system. This work was partially supported by the German National Research Council (DFG) under contract Ke 401/6-2.

References

[BDT83] D. Bitton, D. DeWitt, and C. Turbyfill. Benchmarking database systems: A systematic approach. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, 1983.

[BEG96] R. Buck-Emden and J. Galimow. *Die Client/Server-Technologie des SAP-Systems R/3*. Addison-Wesley, Reading, MA, USA, 3. edition, 1996.

[Col96] G. Colliat. OLAP, relational, and multidimensional database systems. *ACM SIGMOD Record*, 25(3):64–69, Sep 1996.

[DHKK] J. Doppelhammer, T. Höppler, A. Kemper, and D. Kossmann. Database performance of SAP System R/3. http://www.db.fmi.uni-passau.de/projects/SAP.

[FS96] P. Fernandez and D. Schneider. The ins and outs (and everything in between) of data warehousing. Tutorial handouts for the SIGMOD Conference, 1996.

[GBLP96] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *Proc. IEEE Conf. on Data Engineering*, pages 152–159, 1996.

[Gra93] J. Gray. *The Benchmark Handbook for Database and Transaction Processing Systems*. Morgan-Kaufmann Publishers, San Mateo, CA, USA, 2. edition, 1993.

[LM95] B. Lober and U. Marquard. *Standard Application Benchmarks*. SAP AG, 69185 Walldorf, Germany, Dec 1995.

[Mat96] B. Matzke. *ABAP/4 - Die Programmiersprache des SAP-Systems R/3*. Addison-Wesley, Reading, MA, USA, 1996.

[SAP] SAP AG. R/3 system overview. http://www.sap.com/r3/r3_over.htm.

[TPC95] Transaction Processing Performance Council TPC. TPC benchmark D (decision support). Standard Specification 1.0, May 1995.

[WHSH96] L. Will, C. Hienger, F. Straßenburg, and R. Himmer. *R/3-Administration*. Addison-Wesley, Reading, MA, USA, 1996.