

Evaluation of Object-Oriented Database Systems

C. Huemer[†], G. Kappel[‡], S. Rausch-Schott[‡], W. Retschitzegger[‡], A Min Tjoa[†],
S. Vieweg[†], R. Wagner^{*}

[†]Institute of Applied Computer Science and Information Systems. Department of Information Engineering. University of Vienna, Austria

[‡]Institute of Computer Science. Department of Information Systems. University of Linz, Austria

^{*}FAW. Research Institute for Applied Knowledge Processing. University of Linz, Austria

Abstract. Despite the fact that object-oriented database systems (OODBS) have gained potential as promising database technology for non-standard applications such as computer integrated manufacturing there does not yet exist broad experience with the use of OODBS in real-world applications. One reason is that the features of OODBS, both functional and performance, haven't been exposed to a broader audience. The goal of this chapter is to shed some light upon evaluating the features of OODBS. In the first part of the chapter we discuss an extensive evaluation catalogue for advanced database systems which has been developed during the course of a real-world project. A discussion of the pros and cons of using such evaluation catalogues summarizes our experience. The second part of this chapter surveys existing performance benchmarks for OODBS. Special emphasis is given to the requirements a benchmark has to fulfill, and to the practical applicability and usefulness of the proposed benchmarks.

Categories and Subject Descriptors: C.4 [Performance of Systems]; H.2.8 [Database Management] Database Applications; K.6.2 [Management of Computing and Information Systems] Benchmarks

General Terms: Database Systems, Performance Evaluation, Advanced Database Applications

Additional Key Words and Phrases: database requirements for CIM, functional evaluation of object-oriented database systems, application benchmarks

1 Introduction

Advanced database applications such as computer integrated manufacturing have emerged over the past decade [Cattell91, Encarnação90, Gupta91]. Object-oriented database systems are designed to meet their database requirements.

The first prototype implementations of OODBS were hardly usable in production environments. Recent developments and improvements of existing systems increase the usability of OODBS in practice. Consequently, the demand for means to evaluate this new technology has emerged.

In this chapter we describe the functional and performance evaluation of OODBS. The functional evaluation of OODBS concerns the assessment of criteria that form the core of object-oriented database technology whereas performance evaluation provides means to measure database systems with respect to a given workload in a given application domain. In the following we will present approaches to both functional and performance evaluation of OODBS.

This chapter is organized as follows. In the next section we will describe a functional evaluation framework for object-oriented database systems. Section 3.3 gives an overview of performance benchmarks for object-oriented database systems. We conclude with a survey of the most popular approaches.

2 Evaluation Criteria for Object-Oriented Database Systems

Object-oriented database systems were designed to meet the database requirements of advanced database applications. The functionality of OODBS can be characterized by the features depicted in Table 1. These features are part of the evaluation framework as developed by the authors in the course of an evaluation project [Kappel92]. The framework serves as the basis for our further investigations of the functional evaluation of OODBS. The developed criteria catalogue was designed to allow the qualitative and quantitative assessment of object-oriented database systems. The presented criteria have been developed through an extensive study of the literature [Ahmed92, Atkinson89, Encarnacao90, Kappel94, Stonebraker90] and from experience gathered in previous projects. Our analysis of object-oriented database systems is made up of a detailed list of questions. Note that our approach is not restricted to object-oriented technology, but that it provides a framework for evaluating advanced database technology in general. These questions are structured into sections representing the main features of database technology. Each section is further refined into two or more levels of subcriteria. The subcriteria allow a more detailed assessment of the evaluated systems. The evaluation catalogue comprises 20 sections each with about 25 subcriteria. The amount of information items (sections and subcriteria) totals to more than 500 questions.

OODBS Features	
<ul style="list-style-type: none"> • Data Model • Constraints & Triggers • Persistence • Data Dictionary • Tools • Query Management • Host Programming Languages • Schema Evolution • Change Control • Version Management 	<ul style="list-style-type: none"> • Concurrency Control • Recovery • Authorization • Architecture • Storage Management • Query Optimization • Operational Conditions • Distribution • Interfaces • Business Criteria

Table 1. Features of OODBS

A functional evaluation of the OODBS represents the starting point of any database evaluation. We will therefore describe the evaluation features in more detail.

An object-oriented database system is a database system with an object-oriented **Data Model**. At present, there exist several different object-oriented data models. They are either based on existing object-oriented languages, like C++ and Smalltalk, or they have been newly developed. There exists no single object-oriented data model as it was the case for traditional (hierarchical, network, relational) database systems [Maier89]. Nevertheless, there is consensus that a data model to be called object-oriented has to exhibit the following core features [Atkinson89]: *complex object modeling*, *object identity*, *encapsulation*, *types*, *inheritance*, *overriding with late binding*, *extensible types*, and *computational completeness*. Complex object modeling deals with the specification of objects out of other objects which may also be complex by nature. Object identity provides a system-defined unique key and thus, allows the sharing of objects. The principle of encapsulation defines data (= structural information) and accessing operations (= behavioral information) as one unit called object type (or object class). It allows to distinguish the interface of the object type, i.e., the set of operation signatures, from its implementation, i.e., the implementation of the operations and the underlying data structure. An operation signature consists of the name of the operation, the types and names of its input parameters, and the type of its return value. An object type defines structural and behavioral information of a set of objects called *instances* of the object type. Inheritance supports incremental type specification by specializing one or more existing object types to define a new object type. Overriding means redefinition of the implementation of inherited operations, and late binding allows the binding of operation names to implementations at run time. The extensible type requirement permits new object types consisting of structural and behavioral information to be defined by the user and, furthermore, to be treated like any other pre-defined type in the system. Finally, computational completeness requires that any computable algorithm can be expressed in the data definition language (DDL) and/or the data manipulation language (DML) of the database system. The features described above have all been incorporated into the evaluation catalogue.

Constraints and Triggers represent an advanced technique to specify integrity constraints for a database. Constraint and trigger specification is important in the context of active database systems (see chapter „Active Object-Oriented Database Systems for CIM Applications). We therefore included it in our evaluation. The questions mainly focus on how constraints and triggers are incorporated into the database system. This includes the kinds of triggers and their enforcement.

Persistence is one of the main features of any database system. Object-oriented database systems follow different approaches to reach persistence. Persistence can be reached by declaration, by reachability from other persistent objects or by collection membership. The way how objects may become persistent is a crucial issue in the context of porting applications between OODBS and is therefore evaluated in detail. Furthermore we included the deletion semantics, the orthogonality to types, and the homogeneous handling of persistent and transient objects in our investigations.

Usually, advanced database applications operate in a complex environment. A **Data Dictionary** supports the management of such an environment by providing access to schema information. Furthermore, **Tools** for application development, user management, report generation, database archiving, database integrity checking, and for accessing the data dictionary were investigated.

The flexible access to the database and thus **Query Management** is one of the most important requirements for advanced database applications. With flexible database access we address the need for varying and demanding database manipulation requirements. The database access strategies differ from task to task. Some of the tasks, such as engineering design may require the access to objects via inter-object references (navigational access) while others such as the manipulation of incoming orders require associative access via the specification of predicates (associative access). In order to allow flexible access to the database, both access strategies must be supported within the same language. [Bancilhon89] discusses several requirements for an advanced (object-oriented) query language. These include ad-hoc facilities to state queries and updates in a high level language without using a programming language. In addition, multi-database queries and recursive query processing were considered important. Thus, the query language must have equal expressive power as conventional programming languages. Therefore, we also investigated the supported **Host Programming Languages**. Object-oriented programming languages such as C++, Smalltalk, Eiffel, and CLOS and non-object-oriented programming languages such as C, Modula-2, and Ada were included in this section.

Besides query facilities a flexible data management should also include the support for **Schema Evolution**. Schema evolution describes the mechanisms to cope with changes in the data definition of a given database. The database schema is the result of requirements specification and conceptual (and logical) database design. It thus reflects the 'real-world' entities elaborated in these design steps. These entities are subject to more or less frequent changes due to the evolving nature of 'real-world' situations. Schema evolution provides a framework to manage these changes in a controlled manner. Our analysis of schema evolution features included conceivable changes in the database schema and invariants in order to preserve the database schema's structural consistency. We assume that the database is populated whenever changes occur. With **Change Control** we denote the mechanisms that are used for the database conversion in order to conform to the new schema resulting from the schema evolution process. Thus, we investigate mechanisms for preserving the behavioral consistency in the course of schema changes. This includes strategies for the adaptation of instances, methods, and queries.

The section **Version Management** investigates how the evolution of database instances is supported. Our analysis of version models includes the specification of the structure and behavior of versioned objects. The structure describes the way in which versionable objects are composed and the basic granularity of the versioning mechanisms. The dynamic component of version models describes the intra- and inter-object relationships of versioned objects. Intra-object relationships define how object versions are created and how they relate to each other. Inter-object relationships charac-

terize the versioned design object as referenced from other objects. The referencing objects may be other versioned objects or non-versioned objects.

Concurrency Control, Recovery, and Authorization provide means for the management of multiple users accessing the same database under restricted resources. The scheduling of these resources requires a measure of consistency and correctness. The concept of transactions provide a framework to ensure this. Transactions are a collection of actions that access the database. They are logical units that group together operations to form a complete task and they are atomic units preserving the consistency of the database. They also serve as unit for recovery to a consistent database state in case of failures. Our analysis in these sections therefore includes lock types, lock granularity, deadlock detection, the logging concept, access control, and authorization.

The achievements in workstation and network technology forced the shift from the host based computing paradigm to the client/server computing paradigm. The **Architecture** of the database system and the **Storage Management** are strongly connected to this computing paradigm. Our analysis is based on the supported client/server concept, disk management, and the memory architecture. We investigated which paradigm is followed in the client/server communication; whether pages, files, or objects are transferred from the client to the server. Furthermore, we investigated the main memory layout, and how disk replicas, indexing and clustering are supported. These issues mainly address performance considerations at the physical level.

Query Optimization addresses the efficient execution of queries. Our investigations in this section mainly focus on the use of the above mentioned features (indexes, clustering) as well as the management of the query optimizer (tuning and interrogating).

In the section **Operational Conditions** we analyzed the hardware and software requirements for each of the database systems. Object-oriented database systems mainly operate in a distributed hardware and software environment. The section **Distribution** addresses how data and control may be distributed. We investigated whether data distribution is transparent to the user. Furthermore, remote-database access, multi-database queries, and the heterogeneous environment were investigated.

Advanced database applications do not operate in an isolated environment. They are integrated into various communication and information services. Therefore, we included the supported **Interfaces** in our analysis. This comprises the database interface to CASE tools, and standardized description languages such as STEP-Express.

Business Criteria such as reference installations, customer support, documentation, pricing, and the support of standards (ANSI C++, ODMG [Cattell94], etc.) are of great importance in a production environment. However, it is quite difficult to assess most of these features.

As mentioned above, all of these features are relevant to the evaluation of any OODBS. The presented evaluation schema is used as a starting point for the evaluation framework. The evaluation is carried out in three phases:

- (a) Functional Evaluation of the OODBS

- (b) Assessment of the Database Requirements
- (c) Rating of the Database System

In the first step the evaluation catalogue is filled in by database experts in order to rate the evaluation features. The functional evaluation is carried out either by means of interviews with product experts or by an extensive study of the product literature. In addition to the 'on-paper' evaluation simple benchmark programs that emphasize a particular feature may be implemented to judge controversial evaluation features. In the second step the evaluation features are assigned weights corresponding to the importance of the evaluation features as rated by the application domain expert. This task is highly dependent on the intended use of the OODBS and eminently relies on the database requirements of the application domain. In the last step, the results of the functional evaluation and of the assessment of the database requirements are joined. This task may be accomplished either with qualitative or quantitative methods. In the former case the decision is taken on the basis of informally stated evaluation ratings such as 'suitable/non-suitable' and 'high/low'. In the latter case the overall rating of the evaluated products is computed by summing up the evaluation ratings weighted with the importance of the evaluation features. In both cases, the result represents a sound basis for the final decision process.

Besides functional requirements performance considerations may influence the decision process as well. In the following we will describe the issues that are relevant to performance evaluation in non-standard application domains. We will then discuss the approaches to benchmark advanced database applications; namely, the *Sun* benchmark, the *HyperModel* benchmark, the *OO1* benchmark, the *OO7* benchmark, and the *SEQUOIA 2000* benchmark.

3 How to Benchmark Object-Oriented Database Systems?

The need for new approaches to database management systems for advanced applications has been widely reported. Besides the advanced functionality, one of the central issues for the acceptance of advanced database systems is *performance*. Thus, measuring performance is an important topic. In this section we will present some of the main issues that effect on the evaluation of OODBS and thus, should be included in any OODBS benchmark.

Domain-specific benchmarks should follow some key criteria. In general, an application benchmark should be

- relevant,
- portable,
- scalable, and
- simple.

Furthermore it should provide a clear measure and both vendors and users should embrace it [Gray91, Gray93]. *Relevance* means that a benchmark should measure the performance of a system when performing typical operations within that problem domain. The benchmark should be *portable*, i.e. easy to implement in different sys-

tems and architectures. The benchmark should apply to computer systems in any size. *Scaling* the benchmark must be possible. The measurement of benchmark operations should result in a clear metric that allows insight into the performance of the system under test. Finally, the benchmark should be *simple* and *understandable* in order to be accepted by the audience. Within this framework, the main characteristics of an application domain must be mapped onto a benchmark specification.

When given the complexity of advanced database applications, it is obvious that traditional approaches to benchmarking DBS [Bitton83, Turbyfill91, TPCA92, TPCB92, TPCC92] lack relevance when measuring the performance of advanced database applications. Relevance can be expressed in terms of database requirements for a given application domain. These requirements include both functional requirements and performance requirements. The requirements form the basis of the database schema and the operations included in a benchmark. Consider CIM as an example for a complex application domain. Database systems represent a key technology for the realization of CIM. The database requirements for CIM applications can be grouped into three basic requirement clusters: *data modeling issues*, *querying and manipulation issues*, and *integration issues* (see chapter „Database Requirements of CIM Applications“ in this book). Table 2 gives an overview of these requirements. The data modeling requirements comprise extended attribute domains, complex object support, relationships and dependencies between the modeled objects, and active consistency checking and knowledge-base support. The querying and manipulation requirements are widespread and include advanced transaction management issues, flexible database access structures, change management issues like versioning and schema evolution, and interfaces to various data formats. Due to the distributed nature of manufacturing systems the integration aspect of computer integrated manufacturing deserves particular emphasis. Distributed data management and multi data management are the key requirements for distributed computing in this field. Reverse engineering and integrated data and process modeling are rather concerned with the logical integration of manufacturing and business applications.

<p><i>Data Modeling Requirements</i></p>	<ul style="list-style-type: none"> • Extended Attribute Domains • Complex Object Support • Relationships and Dependencies • Active Consistency Checking and Knowledge-Base Support
<p><i>Querying and Manipulation Requirements</i></p>	<ul style="list-style-type: none"> • Advanced Transaction Management • Flexible Database Access • Change Management • Interfaces

Table 2. Database Requirements for CIM Applications

<i>Integration Requirements</i>	<ul style="list-style-type: none"> • Distributed Data Management • Multi Data Management • Reverse Engineering of Data • Integrated Data and Process Modeling
--	---

Table 2. Database Requirements for CIM Applications

A database requirements analysis as presented above forms the basis of any database application benchmark. The characteristic functional requirements, typical database operations, and the database size requirements are then merged into a benchmark suite. We will now investigate the approaches that have been chosen for advanced database application benchmarks. Therefore, we have identified the following issues as a framework for our analysis:

- Database Schema
- Database and its Generation Process
- Benchmark Operations
- Measurement Guidelines
- Reporting of Benchmark Results

The database schema describes the structure of the database and represents the complexity of the application domain under investigation. With generation process we denote the way how the benchmark database has to be constructed, and the restrictions to be met. Additionally, we describe the database size which is relevant for an application domain. The benchmark operations describe the operations that form the operational basis for the benchmark suite. They usually represent a mix of the most common and most characteristic queries and manipulations of an application domain. Moreover, we describe the measurement guidelines of the benchmark suite. They indicate how the results of the benchmark must be derived and which metric is used. As described above, the full mentioning of the system environment is very important in order to get comparable results from any benchmark run. Therefore, we also included the disclosure requirements for the reporting of the benchmarking results. In the following, each benchmark approach is investigated with respect to the above described characteristics. We will survey the Sun Benchmark [Rubenstein87], the HyperModel benchmark [Anderson90], the OO1 benchmark [Cattell92], the OO7 benchmark [Carey93a], and the SEQUOIA 2000 storage benchmark [Stonebraker93]. For all these approaches we will analyze the underlying database schema, the generation of the database, the operations, how the measurements must be performed and finally, how the benchmark results must be reported.

3.1 The Sun Benchmark

The Sun benchmark is one of the first approaches to measure database performance for OODBS. It thus represents a starting point for further approaches presented below.

Database Schema

The Sun benchmark database schema (Figure 1) describes a simple library data-

base and consists of three basic concepts. The Person class describes persons in terms of `person_ID`, `name`, and `birthdate`. The Document class consists of `document_ID`, `title`, `page_count`, `document_type`, `publication_date`, `publisher`, and a document description. The Authorship relationship associates each person to 0 or more documents and each document to exactly 3 authors.

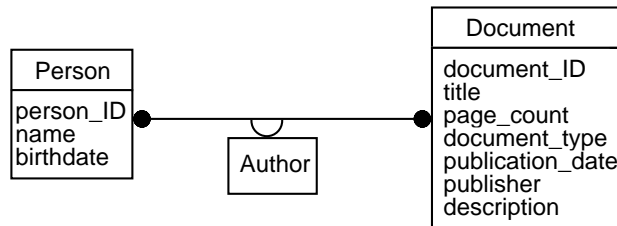


Figure 1. The Sun Benchmark Database Schema

Database Contents and Generation

The benchmark database is generated in a small and a large version. The small database consists of 20000 Persons, 15000 Authors, and 5000 Documents. The large database comprises 200000 Persons, 150000 Authors, and 50000 Documents.

Operations

The Sun benchmark consists of seven operations. These operations can be grouped into lookups (NameLookup, RangeLookup, GroupLookup), inserts, database scan, and database open. The NameLookup operation selects a person with a randomly selected `person_ID`. The RangeLookup operation selects all persons with `birthdate` within a randomly generated range of 10 days. The GroupLookup query finds the person which authored a given document. The Reference Lookup finds a person object that is referenced by a randomly selected author object. RecordInsert inserts a new person in the database. SequentialScan scans all document objects in the database. Furthermore, the time for the database initialization is measured.

Measurements

The measurement for the operations described above work as follows. The operations are repeated 50 times. This operation set is then repeated 10 times. This results in 500 queries and 50 open database operations. The average elapsed time for these operations is then reported.

Disclosure Report

No disclosure report for benchmark results is specified. The authors “hope” that anyone will report the used indexes, the access methods, and the total space required for the database.

OMT is an object-oriented modeling technique [Rumbaugh91]. For further details the interested reader is referred to the literature.

3.2 The HyperModel Benchmark

The HyperModel benchmark is based on the Sun benchmark described above. The extensions include a more complex database schema and additional benchmark operations. As pointed out in [Gray91], the benchmark provides more than a single performance measure. The operations cover a wide spectrum of database operations, each focusing on special queries.

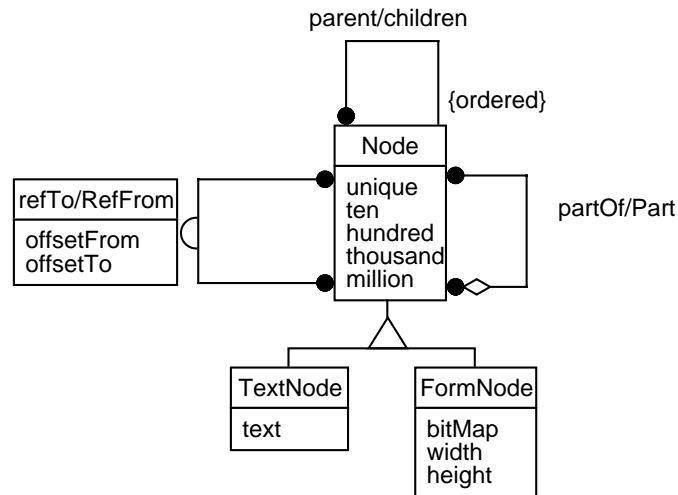


Figure 2. The HyperModel Database Schema

Database Schema

The conceptual schema of the HyperModel Benchmark (Figure 2) is based on an aggregation and generalization hierarchy of HyperModel documents. A HyperModel document consists of a number of sections each of which is represented by an object of type `Node`. `Node` has two sub-types: `TextNode` and `FormNode`. The instances of type `Node` have 5 attributes (`uniqueID`, `ten`, `hundred`, `thousand`, and `million`). In addition, a `TextNode` has a `text` attribute and a `FormNode` has a `bitMap`, `width`, and `height` attribute.

Nodes are interrelated by three relationships: the `parent/children` relationship (1:N), the `partOf/parts` relationship (M:N), and the `refTo/refFrom` relationship (M:N). The `parent/children` relationship is used to model the recursive aggregation structure of sections within a document. The children are ordered. The `partOf/parts` relationship is constrained to be hierarchical. Parts may share subparts but the relationship forms an acyclic graph. The `refTo/refFrom` relationship has been designed to model hypertext links, when attributes are attached to each link to describe the offset of each ending point within a node. The two attributes `offsetFrom` and `offsetTo` describe the starting/ending point of these links.

Database Contents and Generation

A HyperModel database consists of a single document composed of a network of nodes with the relationships described.

The parent/children relation forms a hierarchy that is made up of 7 levels (0-6). Each non-leaf node has 5 successors. The nodes are numbered starting with 1 (level 0) and ending with 10471 (last node at level 6). The total amount of nodes is 19531.

The partOf/parts relation is created by relating each node at level k to five randomly chosen (uniformly distributed) nodes of level $k+1$ in the tree structure built up in the parent/children relation.

The refTo/refFrom relation is built up by visiting each node in the parent/children hierarchy and connecting the node to a randomly chosen node of the entire hierarchy. The values of the attributes (`offsetFrom`, `offsetTo`) are initialized randomly (between 0 and 10).

The nodes in the parent/children hierarchy are numbered sequentially. The attributes are initialized with a randomly selected integer value from the interval 0 to the number specified by the "attribute name" (i.e. ten, hundred, thousand, million).

The lowest level in the hierarchy consists of `TextNodes` and `FormNodes`. There is one `FormNode` per 125 `TextNodes` (125 `FormNodes`, 15500 `TextNodes`). `TextNodes` contain a string of a random number (10-100) of words. Words are separated by a space and consist of a random number (1-10) of lowercase alphabetic characters. The first, middle and last word must be "version1". Each `FormNode` has a `bitMap` attribute which is initially white (0s). The size varies between 100x100 and 400x400 (always a square). Clustering (if possible) has to be done on the parent/children relationship.

Operations

The HyperModel benchmark consists of 7 groups of operations (20 operations). The operations can be grouped into Name Lookup Operations (`nameLookup`, `nameOIDLookup`), Range Lookup Operations (`rangeLookupHundred`, `rangeLookupMillion`), Group Lookup Operations (`groupLookup1N`, `groupLookupMN`, `groupLookupMNAtt`), Reference Lookup Operations (`refLookup1N`, `refLookupMN`, `refLookupMNAtt`), Sequential Scan (`seqScan`), Closure Traversal Operations (`closure1N`, `closureMN`, `closureMNAtt`, `closure1NAttSum`, `closure1NAttSet`, `closure1NPred`, `closureMNAttLinkSum`), and Editing Operations (`textNodeEdit`, `formNodeEdit`).

- *Name Lookup Operations*

The `nameLookup` selects a node instance with a randomly chosen `uniqueId` value and returns the value of the hundred attribute. The `nameOIDLookup` operation finds the node instance given a random reference to a node.

- *Range Lookup Operations*

The `rangeLookupHundred` query selects 10 % of the nodes with the hundred attribute in the range of 10. The `rangeLookupMillion` selects 1 % of the nodes with the million attribute in the range of 10000.

- *Group Lookup Operations*

All group lookup operations follow the parent/children relationship, the partOf/parts

relationship and the refTo/refFrom relationship. The groupLookup1N selects the children of a randomly chosen node and returns a set of five objects. The groupLookupMN selects all part nodes of a random node. The groupLookupMNAtt selects the related node of the refTo/refFrom relationship.

- *Reference Lookup Operations*

The reference lookup operations (refLookup1N, refLookupMN, refLookupMNAtt) operate like the group lookup operations, except that they select the nodes in the reverse direction.

- *Sequential Scan*

The seqScan operation visits each node object in the database and accesses the ten attribute value of each node. The sum of all ten attribute values is returned.

- *Closure Traversal Operations*

The closure traversal operations retrieve a node from the database and transitively visit related nodes. All operations start off with a randomly chosen node on level 3 of the node hierarchy.

The closure1N operations selects a random node and follows the parent/children relationship in pre-order to the leaves of the tree. The closureMN selects a random node and traverses the partOf/part relationship recursively to the nodes of the tree. The closureMNAtt selects a random node and follows the refTo/refFrom relationship 25 times. The closure1NAttSum sums the hundred attribute values for all nodes reachable from a random node via the parent/children relationship. The closure1NAttSet visits all nodes reachable from the starting node via the parent/children relationship and updates the hundred attribute value to (99 - the actual value). The closure1NPred visits all nodes reachable by the parent/children relationship. Only those nodes (and their children) are retrieved that are out of the range 10000 of their million attribute values. The closureMNAttLinkSum performs the closureMNAtt operation and sums up the offsetTo attribute values of the nodes.

- *Editing Operations*

The editing operations test the interface to other programming languages and the updating of a node. The textNodeEdit operation selects a random TextNode and updates all three occurrences of the string “version1” to “version-2”. The formNodeEdit operation inverts a 50 x 50 subrectangle (starting at position (25, 25)) in a randomly chosen FormNode. The operation is performed 10 times per node in order to emulate interactive editing.

Measurements

The time of each operation is measured in seconds. In order to achieve correct results in case of extensive cache use of the OODBS the operations are run with empty memory (cold run) and full cache memory (warm run). Basically, the benchmark suite consists of three phases, the setup, the cold run, and the warm run. The following sequence must be followed. First, perform the database setup. This includes the preparation of the inputs to the benchmark operations. Then perform the “cold run”. The operations must be run 50 times. If the operation is an update operation, the changes

must be committed once for all 50 operations. The entire cold run is timed and divided by 50. This number is reported as the “cold run” result. The “warm run” repeats the operations 50 times with the same inputs. If the operation is an update operation, then the changes are committed once for all 50 operations. The total time is divided by 50 and reported as the “warm run” result.

Disclosure Report

The specification of the HyperModel benchmark does not require a disclosure report.

3.3 The OO1 Benchmark

As in the case of the HyperModel benchmark, the OO1 benchmark is based on the Sun benchmark. The improvements in the OO1 approach rather focus on simplifying the earlier approach. The benchmark is designed to measure the performance in the domain of engineering database applications.

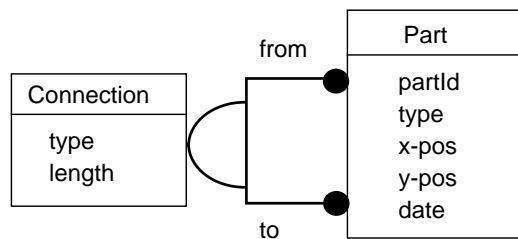


Figure 3. The OO1 Database Schema

Database Schema

The database schema of the OO1 benchmark (Figure 3) consists of two basic components: the part relation and the connection relationship. Parts are built up by `partID`, `type`, `x-pos`, `y-pos`, and `date`. The connection relationship itself contains information about the connected parts (`type` and `length`).

Database Contents and Generation

The database consists of N parts and $3*N$ connections. Parts have a unique identifier (`partID`) ranging from 1 to N . There are exactly three connections to other parts. The random connections between parts are selected to produce some locality of reference (90 % of the connections are randomly selected among the 1 % of the “closest” parts, the remaining connections are made to any randomly selected part. Closeness is defined by using the parts with the numerically closest `partIDs`).

The algorithm for the database generation is provided by the authors. Three benchmark databases are provided: small, large, and huge (see Table 3). The unformatted small database comprises approximately 2 megabytes of data. The authors claim the database to be a good representative of engineering databases. The benchmark results are mostly reported for the small database only. It is assumed that the database resides on a (remote) server and that the application runs on a workstation. There are no

restrictions on indexes, cache-sizes, and network architectures.

<i>Database</i>	<i>Parts</i>	<i>Connections</i>	<i>Scale</i>
<i>small</i>	20000	60000	1
<i>large</i>	200000	600000	10
<i>huge</i>	200000 0	6000000	100

Table 3. The OO1 Database Sizes

Operations

The OO1 benchmark suite consists of three operations: LOOKUP, TRAVERSAL, and INSERT. The LOOKUP operation generates 1000 random Part-IDs and fetches the parts from the database. For each part a null procedure is called (in any host programming language) passing the x,y position and type of the part. The TRAVERSAL operation scheme finds all parts connected to a randomly selected part, or to a part connected to it (7-level closure = 3280 parts with possible duplicates). For each part a null procedure is called passing the x,y position and the type. The time for REVERSE TRAVERSAL (swapping the *from* and *to* directions) is also measured. The INSERT operation inserts 100 parts and three connections from each to other randomly selected parts into the database. Time must be included to update indices or other access structures used in LOOKUP or TRAVERSAL. A null procedure to obtain the x,y position for each insert must be called. The changes must be committed to the disk.

Measurements

The benchmark measures the response time of a single user from the instant when a program calls the database system with a particular query until the results of the query have been placed into the program's variables. Each measure is run 10 times, and the response time is measured for each run to check consistency and caching behavior.

General order of execution

The general execution order in the OO1 benchmark is similar to that of the Hyper-Model benchmark. The benchmark run also consists of a "cold run" and a "warm run". The cold run results report the time of the first execution of the benchmark operations starting off with an empty cache memory. The warm run results are derived from 9 subsequent executions of the operations without clearing the database cache memory. For the insert operation, the database has to be restored to its original state.

Disclosure Report

The work does not define specific requirements. The information which should be included in a benchmark report is listed in Table 4:

Area	Types of reportable information
Hardware	CPU type, amount of memory, controller, disk type/size
Software	O/S version, size of cache
DBMS	Transaction properties (whether atomic, level of supported concurrency, level of read consistency, lock modes used etc.); recovery and logging properties; size of database cache; process architecture (communication protocol, number of processes involved, etc.); security features (or lack thereof); network protocols; access methods used
Benchmark	any discrepancy to the implementation described here; real "wall-clock" run-time, CPU time and disk utilization time are also useful; size of the database

Table 4. The OO1 Disclosure Report

3.4 The OO7 Benchmark

The OO7 benchmark has been designed to overcome the limitations of the OO1 and the HyperModel approach for benchmarking OODBS. This includes complex object operations, associative object access and database reorganization.

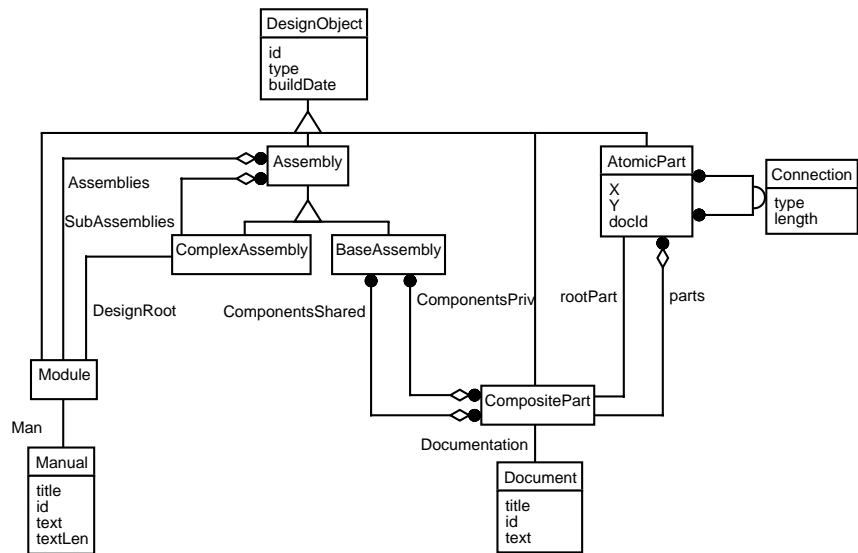


Figure 4. The OO7 Database Schema

Database Schema

The OO7 benchmark [Carey93a, Carey93b] database consists of a set of composite parts. The composite parts have a number of attributes (`id`, `buildDate`, `type`) and are connected to a set of atomic parts. The number of interconnected atomic parts depends on the database parameters. The degree of connectivity varies between 3, 6, and 9 connections to other atomic parts. Furthermore, a document object contains documentation information (`title`, `string`) for each composite part. Composite parts are referred to by base assemblies that form a 7-level complex assembly hierarchy. Complex assembly hierarchies are compiled to modules each of which has an associated manual object that contains additional information. Figure 4 shows the complete database schema of the OO7 benchmark specification.

Database Contents and Generation

The size of the OO7 benchmark database varies between small, medium, and large scale. The connectivity of the atomic parts, the assemblies, and the modules are shown in Table 5.

<i>Parameter</i>	<i>Small</i>	<i>Medium</i>	<i>Large</i>
NumAtomicPerComp	20	200	200
NumConnPerAtomic	3, 6, 9	3, 6, 9	3, 6, 9
DocumentSize (bytes)	2000	20000	20000
ManualSize (bytes)	100K	1M	1M
NumCompPerModule	500	500	500
NumAssmAssm	3	3	3
NumAssmLevels	7	7	7
NumCompPerAssm	3	3	3
NumModules	1	1	10

Table 5. OO7 benchmark database parameters

Operations

The OO7 benchmark includes three clusters of operations; traversals, queries, and structural modifications.

• Traversals

T1: Raw traversal: This traversal query scans each assembly and visits the associated composite parts. A depth-first search is performed on the atomic parts. The number of the visited atomic parts is returned.

T2: Traversal with updates: The traversal query T2 operates like traversal T1. In addition, the objects are updated during the traversal. Three types of

updates are specified: (T2a) update one atomic part per composite part, (T2b) update every atomic part, (T2c) update each atomic part of a composite part four times. The traversal query returns the number of update operations that were performed.

T3: Traversal with indexed field updates: T3 performs the same steps as T2, except that the updates are performed on the indexed field `buildDate` (increment if odd, decrement if even).

T6: Sparse traversal speed: This traversal query scans each assembly and visits the associated composite parts. The root atomic parts are visited. The number of the visited atomic parts is returned.

T8: Operations on manual: T8 scans the manual object and counts the number of occurrences of the character “I”.

T9: Operations on manuals: T9 checks the manual text if the first and the last characters are the same.

TCU: Cached update: This traversal operation first performs T1, then T2a. Both traversal operations are performed in a single transaction. The total time minus the T1 hot time minus the T1 cold time is reported.

- *Queries*

Q1: Exact match lookup: The exact match lookup query selects atomic parts by a lookup of their randomly generated id fields. An index can be used for the lookup. The number of atomic parts processed is returned.

Q2: Range query: Q2 performs a 1 % selection of the atomic parts via the `buildDate` field.

Q3: Range query: The range query Q3 performs a 10 % selection of the atomic parts via the `buildDate` field.

Q7: Range query: Query Q7 scans all atomic parts.

Q4: Path lookup: Query Q4 generates 100 randomly selected document titles and performs the following lookup query. For each title retrieve all base assemblies that belong to the composite part associated with the corresponding document object. The number of base assemblies is reported.

Q5: Single-level make: Q5 performs a selection of base assemblies that have a component part of a more recent `buildDate` than that of the base assembly. The number of qualifying base assemblies is reported.

Q8: Ad-hoc join: Q8 performs a join over document `ids` between documents and atomic parts.

- *Structural Modification Operations*

SM1: Insert: The insert operations create five new composite parts (with the corresponding number of atomic parts) and inserts them into the database. References from base assemblies to these composite parts are randomly generated.

SM2: Delete: SM2 deletes the five previously created composite parts (and its associated atomic parts and document objects) from the database.

SM3: Database reorganization: All composite parts are scanned. For each composite part 50 % of its atomic parts are deleted and then newly inserted.

SM4: Database reorganization: This reorganization operation deletes and re-inserts all composite parts and their associated atomic parts.

Measurements

The benchmark measures the elapsed time for each operation. Measurement include a “cold” run with empty cache memory and the average execution time of three further “warm” runs.

Disclosure Report

The authors do not give an explicit procedure for a disclosure report, but they describe the testbed configuration of the hardware and software that were being used in the benchmark runs.

3.5 The SEQUOIA 2000 Benchmark

The SEQUOIA 2000 Storage Benchmark addresses the application domain of engineering and scientific databases. It has been developed in the SEQUOIA 2000 research project that searches to investigate DBS support for Earth Scientists. Earth Scientists are mainly investigating issues that have effects on the condition of our environment. These investigations can be divided into three areas: field studies, remote sensing, and simulation. The SEQUOIA 2000 benchmark has evolved from those areas and addresses these application domains by specifying a set of databases and queries in order to measure the performance of databases for this application domain. Note that the benchmark data is not synthetic data as in the previous approaches but real data collected for scientific use. In the following we will describe the SEQUOIA 2000 benchmark according to the benchmark characteristics we identified above.

Database Schema

The kind of data that is mainly used by Earth Scientists can be divided into four categories: raster data, point data, polygon data, and directed graph data. The benchmark database is thus made up of these kinds of data sets. Figure 5 shows an OMT diagram of the SEQUOIA 2000 benchmark database schema; Figure 6 shows the corresponding Postgres schema.

The raster data represents data from the Advanced Very High Resolution Radiometer (AVHRR) sensor on NOAA satellites. The observed data is divided into so-called tiles, each of which is 1 square km. Two integers represent the relative position of a point within the corresponding tile. The satellite sensors observe 5 wavelength bands for each tile. The benchmark database consists of 26 observations per year.

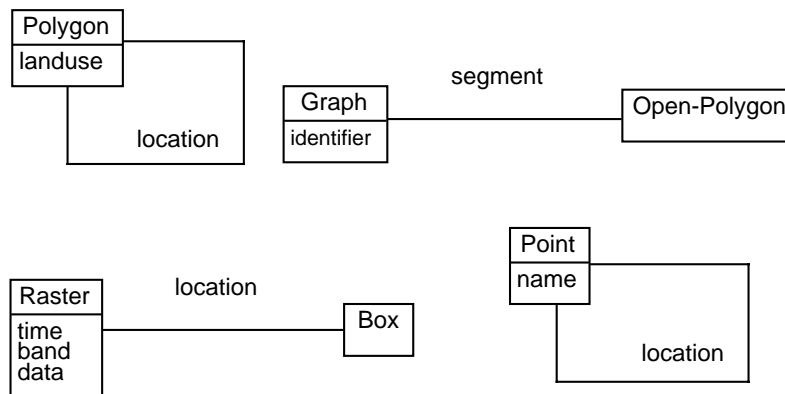


Figure 5. The SEQUOIA 2000 Benchmark Database Schema

The point data consists of names and locations taken from the United States Geological Survey (USGS) and from the Geographic Names Information System (GNIS). Each name in the database (with an average of 16 bytes) and its location (two 4-byte integers) are recorded.

The polygon data consists of homogeneous landuse/landcover data (available from USGS). Each item in the polygon database consists of a variable number of points (two 4-byte integers) and a landuse/landcover type (4-byte integer). An average polygon has 50 sides.

The graph data represents USGS information about drainage networks. Each river is represented as a collection of segments. Each segment is a non-closed polygon. For each segment the segment geometry and the segment identifier are recorded.

```

Raster(time=int4,      location=box,      band=int4,
data=int2[][]])
Point(name=char[], location=point)
Polygon(landuse=int4, location=polygon)
Graph(identifier=int4, segment=open-polygon)
  
```

Figure 6. Postgres Schema of the SEQUOIA 2000 Benchmark

Database Contents and Generation

Usually, geological, geographical, and environmental database systems operate on a huge amount of data. This fact was taken into account in the SEQUOIA 2000 benchmark. The scales of the benchmark data cover the regional database, the national database, and the world database. Table 6 gives an overview of the benchmark sizes for each of the benchmark databases. The regional database consists of data of a 1280km x 800km rectangle. The national database comprises the benchmark data for the whole United States (5500km x 3000km). The world database covers all the world data for each of the types of data described above. The overall database size for the regional database is 1.1 GBytes, for the national database 18.4 GBytes, and more than 200

TBytes for the world database. The authors claim this approach to be durable with respect to technological progress in hardware. Up to now, SEQUOIA 2000 benchmark results are only available for the regional database [Stonebraker93].

Database Scale	Benchmark Data				
	raster data	point data	polygon data	directed graph data	
regional database	1 GB	1.83 MB	19.1 MB	47.8 MB	1.1 GB
national database	17 GB	27.5 MB	286 MB	1.1 GB	18.4 GB
world database	200 TB	numbers not available			

Table 6. The SEQUOIA 2000 Database Size

There are no restrictions on the layout of the database schema in the target DBS as long as AVHRR elements are 16 bit objects and point objects are pairs of 32 bit objects. In addition, the users are free to decompose the data for storage needs. Any necessary indexing or clustering technique may be used.

Operations

The SEQUOIA benchmark queries can be grouped into 5 collections of benchmark queries. Data loading, raster queries, polygon and point queries, spatial joins, and recursion. We will now describe each of these queries that form the basis of the benchmark suite.

- *Data Load*

Q1: Create and load the data base and build any necessary secondary indexes:
This benchmark measure includes the loading of the data from the disk into the database system and the construction of the necessary indexes. It provides a measure for the efficient loading of bulk data into the DBS.

- *Raster Queries*

Q2: Select AVHRR data of a given wavelength band and rectangular region ordered by ascending time

Q3: Select AVHRR data of a given time and geographic rectangle and then calculate an arithmetic function of the five wavelength band values for each cell in the study rectangle.

Q4: Select AVHRR data of a given time, wavelength band, and geographic rectangle. Lower the resolution of the image by a factor of 64 to a cell size of 4

square km and store it persistently as a new object.

This type of queries is dedicated to the analysis of raster data. Q2 retrieves raster data of a specified region within a specified time. Q3 emphasizes the retrieval of raster data followed by an arithmetic analysis of the data. Q4 retrieves raster data, computes a lower resolution image of the data, and stores it in the database.

- *Polygon and Point Queries*

Q5: Find the POINT record that has a specific name.

Q6: Find all polygons that intersect a specific rectangle and store them in the DBS.

Q7: Find all polygons of more than a specific size and within a specific circle.

The polygon and point queries represent queries about geographic points or polygons. Q5 represents a simple lookup operation given a point name. Q6 computes the intersection of polygons. Q7 combines spatial and non-spatial queries; by retrieving the polygons that satisfy spatial and non-spatial restrictions.

- *Spatial Joins*

Q8: Show the landuse/landcover in a 50 km quadrangle surrounding a given point.

Q9: Find the raster data for a given landuse type in a study rectangle for a given wavelength band and time.

Q10: Find the names of all points within polygons of a specific vegetation type and store them as new DBS objects.

The collection of spatial joins emphasizes the database system's ability to perform joins to different spatial data types. Q8 joins polygon and point data. Q9 joins raster data and polygon data. Q10 combines point and polygon data.

- *Recursion*

Q11: Find all segments of any waterway within 20 km downstream of a specific geographic point.

Q11 represents a restricted recursive query of the graph data. It computes all affected regions of a waterway that are 20 km downstream of a specific point.

Measurements

The metric of the benchmark results is the elapsed time in seconds for each of the queries described above. In addition, a performance/price ratio is reported. The overall performance of the database system is computed as the total elapsed time divided by the retail price of hardware. Software and maintenance costs etc. are neglected.

The elapsed time of the benchmark operations does not include the display of the retrieved data on the screen. The measurement includes the time from the start of the query until the placement of the results in the application memory. There are no restrictions as to the language in which the benchmark queries are coded. In case of using low-level DBS interface routines, the results must also be reported for the high-

level interface. Furthermore, the DBS and the application must operate within different system domains in order to ensure the minimum security requirements.

Disclosure Report

The authors do not explicitly give a procedure for a disclosure report, but describe the testbed configuration of the hardware and software that were being used in the benchmark runs.

4 Conclusion

In the previous sections we described approaches to the evaluation of OODBS. We presented an approach to the *functional evaluation* based on the presented criteria catalogue. This catalogue comprises about 500 criteria that allow the functional assessment of OODBS. We also surveyed the most popular *object-oriented database benchmarks*, namely the Sun Benchmark, the HyperModel benchmark, the OO1 benchmark, the OO7 benchmark, and the SEQUOIA 2000 benchmark. We classified the approaches according to the database schema, the database generation, the benchmark operations, the measurement guidelines, and the reporting of the results. A summary of the approaches described above is presented in Table 10.

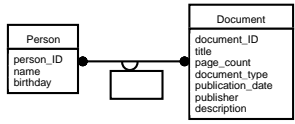
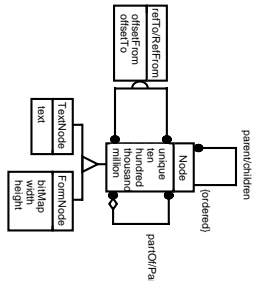
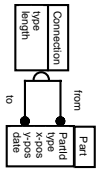
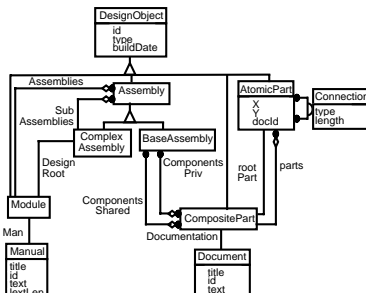
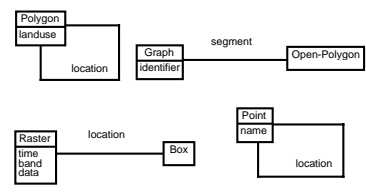
Sun Benchmark	HyperModel Benchmark	OO1 Benchmark	OO7 Benchmark	Sequoia 2000 Benchmark	
<p>Database Schema</p> 	<p>Benchmark Description</p> <p>The Sun benchmark database describes a simple library database. It consists of a person class that describes authors and a document class that describes documents. The authorship concept describes the relationship between documents and persons. In the small version the database consists of 20000 persons, 15 000 authors, and 5000 documents. The large database scales these numbers by the factor ten.</p>	<p>The HyperModel database consists of an aggregation hierarchy and a generalization hierarchy of Nodes. Each Node has 5 successors (7 levels); thus forming a hierarchy of 19531 Nodes. The lowest level consists of 15500 TextNodes and 125 FormNodes (subtypes of Node). In addition to the parent/children relationship, a partOf/parts and a refTo/refFrom relation is generated.</p> 	<p>N parts form the basis of the OO1 database. Each part is connected with exactly 3 other parts (N*3 connections). Small (20000 parts, 60000 connections), medium (200000 parts, 600000 connections), and large (2000000 parts, 6000000 connections) database size is specified.</p> 	<p>The OO1 database consists of composite parts which are connected to a document object and a set of interconnected atomic parts. The composite parts form the leaves of a complex assembly hierarchy. Each of the connections is based on the benchmark database parameters. Small, medium, and large database size is specified.</p> 	<p>The SEQUOIA 2000 database consists of four sets of data: Raster Data, Point Data, Polygon Data, and Graph Data. The data represents geological and geographical data. Three scales of the database sets are specified: regional database (1.1 GB), national database (18.4 GB), and world database (over 200 TB)</p> 

Table 10. An Overview of Advanced DBS Application Benchmarks

	Sun Benchmark	HyperModel Benchmark	OO1 Benchmark	OO7 Benchmark	Sequoia 2000 Benchmark
Restrictions	There are no restrictions on the database schema, indexing, clustering for the benchmark measurements.	There are no restrictions on the location of the database (local or remote). Clustering has to be done on the parent/children relationship (if possible).	The database is assumed to reside on a remote server; the application runs on a workstation. There are no restrictions on indexing, clustering, cache-sizes etc.	There are no restrictions on the database schema, indexing, clustering for the benchmark measurements.	There are no restrictions on the database schema, indexing, clustering for the benchmark measurements.
Operations	7 operations: <ul style="list-style-type: none"> • Name Lookup • Range Lookup • Group Lookup • Reference Lookup • Record Insert • Sequential Scan • Database Open 	7 groups of operations (20 operations): <ul style="list-style-type: none"> • NameLookup • RangeLookup • GroupLookup • Referencelookup • Sequential Scan • Closure Traversal • Editing Operations 	3 operations: <ul style="list-style-type: none"> • Lookup • Traversals • Inserts 	3 groups of operations: <ul style="list-style-type: none"> • Traversals • Queries • Structural Modification Operations 	5 groups of operations (11 operations): <ul style="list-style-type: none"> • Database Loading • Raster Queries • Polygon and Point Queries • Spatial Joins • Recursion
Measurements	The time (response-time) is measured for each operation. Cold and warm times are reported.	The time (response-time) is measured for each operation. Cold and warm times are reported.	The time (response-time) is measured for each operation. Cold and warm times are reported.	The time (response-time) is measured for each operation. Cold and warm times are reported.	The elapsed time for each benchmark operation is reported. An overall performance number may be reported (elapsed time for all benchmark queries / retail cost of hardware).
Reporting	no disclosure report required	no disclosure report required	The authors specify types of reportable information.	no disclosure report required	no disclosure report required

Table 10. An Overview of Advanced DBS Application Benchmarks

References

- [Ahmed92] S. Ahmed, A. Wong, D. Sriram, R. Logcher; Object-oriented database management systems for engineering: A Comparison; JOOP, June 1992
- [Anderson90] T. Anderson, A. Berre, M. Mallison, H. Porter III, B. Schneider; The HyperModel Benchmark; Proc. of the EDBT Conf., 1990
- [Atkinson89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, S. Zdonik; The object-oriented database manifesto; Proc. of the Conf. on Deductive and Object-Oriented Databases; 1989
- [Banc89] A. F. Bancilhon; Query Languages for Object-Oriented Database Systems: Analysis and a Proposal; Datenbanksysteme in Büro, Technik und Wissenschaft, Springer, IFB 204, Zürich, March 1989
- [Bitton83] D. Bitton, D. DeWitt, C. Turbyfill; Benchmarking Database Systems, A Systematic Approach; Proc. of the VLDB Conf.; 1983
- [Carey93a] M. Carey, D. DeWitt, J. Naughton; The OO7 Benchmark; Proc. of the ACM SIGMOD Conf. 1993
- [Carey93b] M. Carey, D. DeWitt, J. Naughton; The OO7 Benchmark; Tech. Report, CS Dept., Univ. of Wisconsin-Madison; 1993
- [Cattell91] R. Cattell; Object data management: object-oriented and extended database systems; Addison-Wesley; 1991
- [Cattell92] R. Cattell, J. Skeen; Object Operations Benchmark; ACM TODS; Vol. 17, No. 1; 1992
- [Cattell94] R. Cattell, The Object Database Standard: ODMG-93, Morgan Kaufmann Publishers, San Mateo, California, 1994
- [Encarnação90] J. Encarnação, P. Lockemann; Engineering Databases, Connecting Islands of Automation Through Databases; Springer Verlag; 1990
- [Gray91] J. Gray; Standards are a Prerequisite for Interoperability and Portability; Tutorial Notes; held at the EDBT Summer School, Alghero/Italy; 1991
- [Gray93] J. Gray; A Tour of Popular DB and TP Benchmarks; Tutorial Notes, held at the ACM SIGMETRICS Conf.; 1993
- [Gupta91] R. Gupta, E. Horowitz (eds.); Object-Oriented Databases with Applications to CASE, Networks, and VLSI CAD; Prentice-Hall; 1991
- [Kappel92] G. Kappel, S. Rausch-Schott, W. Retschitzegger, M. Schrefl, U. Schreier, M. Stumptner, S. Vieweg; Object-Oriented Database Management Systems - An Evaluation; Tech. Rep. ODB/TR 92-21; Institute of Applied Computer Science and Information Systems; Univ. of Vienna; 1992
- [Kappel94] G. Kappel, S. Vieweg; Database Requirements of CIM Applications; in this book; 1994
- [Maier89] D. Maier; Why isn't there an object-oriented data model? Information Processing 89 - IFIP World Computer Congress; G.X. Ritter; North-Holland; 1989
- [Rubenstein87] W. Rubenstein, M. Kubicar, R. Cattell; Benchmarking Simple Database Operations; Proc. of the ACM SIGMOD Conf., 1987

- [Rumbaugh91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen; Object-Oriented Modeling and Design; Prentice Hall; 1991
- [Stonebraker90] M. Stonebraker, L. Rowe, B. Lindsay, J. Gray, M. Carey, M. Brodie, P. Bernstein, D. Beech; Third-Generation Database System Manifesto; SIGMOD Record, Vol. 19, No. 3; 1990
- [Stonebraker93] M. Stonebraker, J. Frew, K. Gardels, J. Meredith; The Sequoia 2000 Storage Benchmark; Proc. fo the ACM SIGMOD Conf.; 1993
- [TPCA92] TPC Benchmark™ A, Standard Specification, Revision 1.1, Transaction Processing Performance Council (TPC); March 1, 1992
- [TPCB92] TPC Benchmark™ B, Standard Specification, Revision 1.1, Transaction Processing Performance Council (TPC); March 1, 1992
- [TPCC92] TPC Benchmark™ C, Standard Specification, Revision 1.1, Transaction Processing Performance Council (TPC); August 13, 1992
- [Trapp93] G. Trapp; The emerging Step standard for production-model data exchange; IEEE Computer; Vol. 26, No. 2; 1993
- [Turbyfill91] C. Turbyfill, C. Orji, D. Bitton; AS³AP: An ANSI SQL Standard Scaleable and Portable Benchmark for Relational Database Systems