

A Comparison of Oracle Berkeley DB and Relational Database Management Systems

*An Oracle Technical White Paper
March 2009*

A Comparison of Oracle Berkeley DB and Relational Database Management Systems

Oracle is a well-known leader in the database industry. Oracle's embeddable database engine, Oracle Berkeley DB, is very different from the company's relational database products.

Berkeley DB offers software developers some key advantages over relational databases, the file system, and homegrown storage systems for a large class of applications.

Berkeley DB is a tool for software developers, not IT professionals or DBAs. It is designed to provide reliable low-level storage services invisibly in third-party applications and services.

INTRODUCTION

Oracle is well known for its industry-leading database engine, Oracle Database. Oracle Database is an extremely reliable, highly scalable, client-server, relational database management system (RDBMS). It serves as the information repository for a wide range of applications in a large variety of industries around the world.

In addition to Oracle Database, the company offers a number of other database products for systems and applications that have special performance, resource or run-time requirements. Oracle's TimesTen database is a high-performance in-memory engine that provides relational database services with outstanding responsiveness and throughput. The Oracle Lite database serves the mobile and device markets, providing easy-to-use storage with synchronization services that allow occasionally-connected devices to share information with a centralized Oracle Database server. Oracle Berkeley DB is the company's only non-relational storage engine, and is aimed at applications and devices that need fast local storage without using SQL for data access.

Because Oracle Berkeley DB is non-relational, it is very different from Oracle's other database products. Users familiar with Oracle Database are often surprised by Berkeley DB, and wonder how and where it can best be used. This paper explores both the similarities and the differences between RDBMS products and Berkeley DB, in order to help customers decide when Berkeley DB is, and is not, a good choice for the system that they are building.

WHAT IS BERKELEY DB?

Berkeley DB is a high performance, scalable and reliable non-relational storage system. An outgrowth of research performed initially at the University of California at Berkeley, and later continued at Harvard University, Berkeley DB was designed from the very beginning to be an embeddable database engine, so no separate server is required, and no runtime human administration is needed.

Philosophically, Berkeley DB is a tool for software developers, not for IT professionals or DBAs. It is intended to provide fast, reliable data storage for applications. The only way to use Berkeley DB is to write code. There is no standalone server and no SQL query tool for working with Berkeley DB databases.

Developers often find it easiest to think of Berkeley DB not as a database system, but rather as a library with a good B-tree, hash table and persistent queue for storing records in an application. That is a reasonable simplification. Unlike simple B-trees or hash tables, though, Berkeley DB offers enterprise-grade, concurrent, transactional storage services, so that multiple threads or processes can operate on the same collection of B-trees and hash tables at the same time without risk of data corruption or loss.

The Oracle Berkeley DB product line consists of three different embeddable offerings.

The Berkeley DB product line consists of three distinct offerings: A high-speed data storage library written in the C programming language, a separate implementation in the Java programming language, and a repository for XML documents built on top of the low-level C language storage engine.

Each of these three products is embeddable. Each is a library that links directly into the address space of the application that uses it.

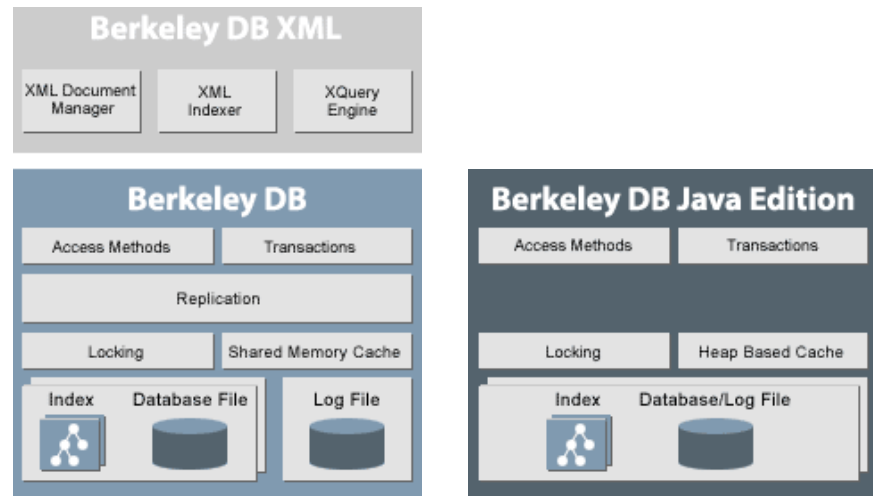


Figure 1: The Oracle Berkeley DB Product Family

In the lower left corner, the product named Berkeley DB is an embeddable library written in the C programming language. It supports different on-disk storage structures, provides full transactional storage services and offers replication for fault tolerance and high availability. The product supports C, C++ and Java language access, as well as a wide variety of popular scripting languages.

Developers building XML-based applications can choose the Berkeley DB XML product, layered on top of the C engine. Berkeley DB XML understands XML schemas, is able to parse and index XML documents, and uses XPATH and XQuery for data retrieval. It supports C++, Java and a variety of popular scripting languages.

Java developers that want the same embeddable storage services as C programmers get, but who need to deploy a 100% pure Java system, can choose Berkeley DB Java Edition. This embeddable engine runs as a single JAR in a Java Virtual Machine. It provides the same transactional storage and retrieval services as the C product, but is implemented entirely in Java.

Some applications need all the power and flexibility of a high-end relational database.

Others just need the low-level read/write services that a file system provides.

Some applications, however, need something in the middle: fast, scalable and transactional storage services, packaged for invisible deployment as a part of a hardware or software product.

Modern relational database systems provide both data storage and data query services. It is useful to think about storage and query support separately when choosing a repository for your application.

CHOOSING THE RIGHT TOOL FOR THE JOB

Sometimes, it is easy to decide how to store data in an application.

For example, large accounting or customer support systems often come already integrated with a relational database product. These systems use the relational engine to store and retrieve the data they manage, and allow administrators and others to query and manipulate that data separately, using SQL and SQL-based reporting tools.

In other cases, a simple file system is all that is needed. Word processing applications, spreadsheets and software development tools like editors and compilers read from, and write to, the file system. Users of these applications work with files by remembering what folders they appear in, and what they are named. No more elaborate query services are required.

Sometimes, though, an application needs something less than the full power of a relational engine, but more than the low-level services of a file system. In those cases, an RDBMS is overkill – it offers too many features, and is often too large and complicated. A file system, by contrast, is underpowered – it offers few or none of the integrity, concurrency and performance advantages of a database system.

Berkeley DB is aimed squarely at that middle ground.

Important Services

When choosing a repository for a new application or service, an architect must think about the data storage and retrieval services that the application requires.

Storage

File systems offer fairly rudimentary storage services, as a rule. They allow applications to read or write any kind of data, but make no guarantees about how simultaneous updates to the same region of a single file are handled. Applications that need to make several interdependent changes to different files – for example, adding a password file entry for a new user, creating a home directory and allocating a mailbox – get no help from the file system, and must handle that interdependency themselves. If the system loses power or suffers a software crash in the middle of a change to a file, most file systems make no promises about what data will survive after restarting.

Most relational databases, by contrast, offer much richer storage services. Competing updates to the same information in a table are handled according to simple rules that guarantee the correct outcome. If an application needs to make several related changes to different tables, it simply starts a transaction, makes the changes, and commits the transaction. The database system guarantees that none of the changes will be permanent until the transaction commits, but once it commits, the changes are all guaranteed to be present, even after a system crash and restart. Most relational systems do impose strict rules about the kinds of

information they will store – applications must turn their data into records in tables, and must translate from between the application’s internal representation, and the database representation, at runtime.

Queries

File systems organize information by filename. Files are stored in folders. In order to use a file, the application must know the name of the file, and of the (likely nested) folder in which it is stored. Of course applications exist that can search a file system for particular file contents, owners and so on, the only built-in query facility in most file systems is name-based lookup.

Relational systems allow applications and users to search for information by typing queries in SQL. These queries can examine the contents of individual records in individual tables, and can combine information from different tables easily. Table names are important, but otherwise, a query in a relational system *describes* the data it wants instead of naming it.

Relational systems, by virtue of SQL, provide an important kind of flexibility: They allow arbitrary queries in the future. In a file system, a file name is fixed and is the only way to look up a file. In a relational system, any combination of columns in any combination of tables can be used to search for, and to combine, data. Rather than prescribing search criteria in advance, relational systems support *ad hoc* queries.

Berkeley DB

Berkeley DB offers many of the storage services – transactions, concurrency, and recovery – of relational database engines. It is closer to a file system in that it stores any kind of data in any format, but provides no easy tools for *ad hoc* queries. Developers must write code to search a Berkeley DB database.

Berkeley DB offers the reliable, scalable transactional support of an enterprise-grade relational database engine, but uses much simpler file system-style interfaces to store, fetch and update records.

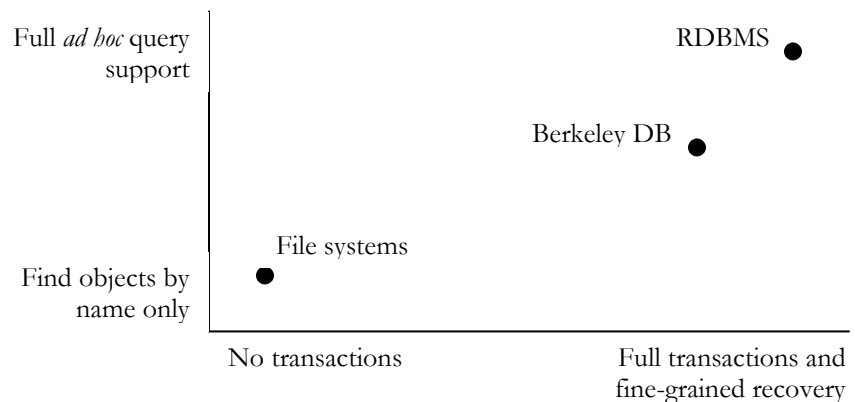


Figure 2: Range of Storage and Query Services

Some applications need transactions and recovery, but not the ability to support arbitrary end-user searches against the data they store. For those applications, Berkeley DB is often a better choice than either the file system or a relational database engine. Berkeley DB combines the simplicity of file system-style data storage and retrieval with the enterprise-grade scalability, reliability and transactional guarantees of a high-end relational system like Oracle Database.

KEY DIFFERENCES

Berkeley DB is different from relational systems in several important ways.

Berkeley DB Is a Library

Most enterprise-grade relational database engines are heavyweight servers. They are often installed on a special purpose machine somewhere in the data center.

Applications that want to store data must connect, usually over a network, to the server, and must read and write data remotely.

Berkeley DB, by contrast, is a library. Each of the three Berkeley DB products links into the same address space as the application that uses it. If multiple applications want to share a single database, they can do so. Each links the Berkeley DB code into its address space, and Berkeley DB uses shared memory and operating system primitives for mutual exclusion to be sure that each thread of control cooperates with the others.

This architecture leads to several benefits.

Ease of Deployment

Because Berkeley DB is embedded directly in the application, no separate server needs to be installed or administered. The application code can be installed on the target system and can create and use Berkeley DB databases immediately.

A large number of end user and infrastructure applications, such as email servers, authentication systems, network management tools and others, need reliable data storage. The people who use these systems, however, are not database administrators, and neither want nor need to know that the application is built on a database. Berkeley DB is entirely invisible to these users.

Outstanding Performance

Certainly, many relational database systems – Oracle Database among them – deliver excellent performance for mission-critical applications. Berkeley DB's in-process architecture, however, gives it some performance advantages for embedding applications.

Because the database engine is literally in the same address space as the application, no database access ever needs to cross a process or network boundary. Context switch time is an important bottleneck in many deployed systems, and moving data

Because it is a library, Berkeley DB is easier for end users to install and use as part of an application. Berkeley DB gets important performance advantages by being in the same address space as the application, too.

across network boundaries on different machines is even more expensive. Berkeley DB eliminates both.

Also, since Berkeley DB runs in process, there are fewer copies required to move the data from disk into the application's data structure. In multi-process or networked database architecture, every process boundary requires at least one new copy of the data. Berkeley DB can move data from the storage device into the buffer cache, and then make a single copy into the application's data structure.

By avoiding context switches and minimizing copies, Berkeley DB gets outstanding performance on commodity hardware. For example, performance tests of Berkeley DB published in a recent white paper¹ documented 90,774 sequential inserts per second, using transactions, in a single thread of control. Read throughput, reading a single record at a time, was 466,623 records per second. Using a high-performance bulk retrieval interface for large sequential scans of the database, the same system was able to read 13,501,800 records per second.

Of course, application developers may build server processes themselves, and those server processes may embed Berkeley DB for data storage. From the point of view of the developer, though, Berkeley DB is embedded directly in the executable file that ships to the end user.

Berkeley DB Offers APIs, not Query Languages

Berkeley DB was designed for software developers, by software developers. Relational database systems generally provide SQL access to the data that they manage, and usually offer some SQL abstraction, like ODBC or JDBC, for use in applications.

Berkeley DB, by contrast, supports a straightforward record-based application programming interface to insert, update, retrieve or delete information. Rather than constructing character strings with SQL and passing those to the database system for interpretation and execution, developers using Berkeley DB make simple function or method calls.

Berkeley DB's interfaces and abstractions are lower level than those of most relational systems. There are interfaces to create, open and close tables. There are interfaces for inserting and deleting single records. There are interfaces that start, commit and abort transactions. Each of these is a single function or method call, and application developers make several of these calls sequentially in order to use a Berkeley DB database.

There is one important exception to the "no query language" story. Berkeley DB XML is an embeddable XML document repository that understands the XPATH and XQuery languages, and that can use XML schemas to store, find and manage documents. Even Berkeley DB XML, though, provides a set of straightforward

Berkeley DB, running on commodity hardware, is able to read nearly half a million records per second in single-record reads. Large sequential reads can use a bulk transfer API to fetch more than thirteen million records per second. Single-record write performance is nearly one hundred thousand records per second.

Berkeley DB's programmatic interfaces were designed by software developers, for software developers. A programmer codes up the data accesses for the application. This means that new coding is required to add new queries later, but it delivers outstanding performance.

¹ "Oracle Berkeley DB: Performance Metrics and Benchmarks"
<http://www.oracle.com/technology/products/berkeley-db/pdf/berkeley-db-perf.pdf>

APIs that allow insert, update and retrieval of these documents. Berkeley DB XML can operate mix XML documents and non-XML data by using the Berkeley DB APIs.

Berkeley DB Uses the Application's Data Model

Relational systems use the relational model, derived from the branch of mathematics known as set theory, to construct and operate on records. The relational model uses a collection of well-defined data types – integer, character, floating-point number and so on – to construct those records. A relational record is formatted in a way that is useful to the database system, but not to the application. In order to move application data, like objects or data structures, to and from the database, the application must translate between the database record format and the application's internal format.

A record in Berkeley DB consists of a key and a data portion. Each of these is an arbitrary bit string, of fixed or variable length, whose contents and meaning are defined by the application. Berkeley DB is able to search for keys in the database, and to retrieve the associated data portion, but knows nothing about the internal structure of either the key or the data portion.

Berkeley DB has built-in collation and hash functions for keys, but developers may override these to enforce a different ordering or exploit a different hash function better suited to their data.

Berkeley DB Can Be Deployed for Zero Administration

Berkeley DB is designed for embedded deployment. It must, in many cases, be entirely invisible to end-users. For example, Berkeley DB is the storage engine used in a popular series of mobile telephone handsets. No mobile phone user wants to handle database administration tasks. Likewise, Berkeley DB is the storage engine for identity management systems, carrier-grade email servers, network management applications and more. In each of these cases, the person responsible for installing and running the service wants to concentrate on user identity, messaging or network monitoring, and not on database backup and checkpointing.

All of the administrative tasks required of a database – backup, checkpointing, reorganization, archival, recovery and so on – are exposed as Berkeley DB APIs. The developer can write application code that handles these tasks invisibly to the end user, and can enforce the practices and policies that make sense for the application. In general, end users are entirely unaware that Berkeley DB is a part of the systems they use.

Translating between programming language objects and relational records is cumbersome and expensive at runtime. Berkeley DB stores application data in its native format, eliminating data marshalling and reducing copies.

Most end users of Berkeley DB-based applications and devices have no idea that they are using a database.

Applications and hardware platforms vary enormously. Berkeley DB is flexible enough to run large server applications on multiprocessor systems, and single-user services like address books on PDAs and mobile phones.

Berkeley DB is Highly Configurable

Multi-processor trading systems, network switches, rack-mounted email servers and mobile telephone handsets are all very different hardware platforms. They have different amounts of memory and disk. An equity trading application usually needs absolute transactional reliability, and possibly replication for high availability. Read-only caching on a Web server can often tolerate temporary inconsistencies without a problem.

Berkeley DB runs well in all of these examples. It offers a wide range of configuration parameters to the developer. Systems designers can configure Berkeley DB to run on the hardware that they use, and to enforce the policies and guarantees that their applications require.

For example, developers can configure Berkeley DB for very small or multi-gigabyte caches. They can choose to enable concurrency, locking and transactions, or not, as the application requires. If fault tolerance, load balancing or high availability is important, developers can turn on replication. If not, they can leave it off.

The list of configuration parameters in Berkeley DB is very long. It includes page sizes for individual tables, policies on duplicate keys in tables, buffer pool replacement policy, mechanism to use for detection and resolution of deadlocks, whether individual tables reside on disk or in main memory, the choice of optimistic or pessimistic concurrency control, whether to force synchronous disk writes at transaction boundaries, how to distribute data to standby machines in a replication group and much, much more. Developers can configure Berkeley DB not just for the hardware they use, but for the specific needs of the application or service they build.

Berkeley DB can make use of very large memories, but occupies half a megabyte – or less, depending on the product chosen and compile-time settings – of code space in the deployed application.

USE CASES

One of the easiest ways to distinguish applications that need a relational database engine like Oracle Database from those that are better served by Berkeley DB is to look at those that have already chosen Berkeley DB.

Infrastructure

Many critical infrastructure services need fast, reliable data storage and retrieval, but do not need the flexibility of dynamic SQL queries. Many of these services rely on Oracle Berkeley DB.

Identity Management

With the proliferation of Web-based services, many applications need to authenticate people when they sign onto systems, and to manage user identity.

Core network infrastructure services – identity management, messaging, storage, packet routing and others – need reliable embedded data management.

These applications generally need to store user names and passwords, as well as information about past behavior and preferences. Public-key infrastructure, directory servers and single sign-on engines are all examples of identity management components.

A number of successful commercial and open source identity offerings use Berkeley DB for storage. Data access patterns are static and predictable in advance. Responsiveness and throughput are critical. Berkeley DB provides the right level of data abstraction, without the overhead of a round trip and a declarative query for every operation.

Messaging

Messaging systems – including email servers, instant messaging and mobile text messaging – all need to store data reliably. They need to keep track of the sender and recipient of a message, its length, creation time, subject and other metadata, and the message body. Enterprise-grade systems need to deliver each message exactly once, which demands transactional reliability.

Berkeley DB serves as the metadata storage engine for email servers from a variety of vendors. It is also a popular choice to store message text and metadata for text messaging and instant messaging systems.

Storage Systems

Storage systems provide the space that file systems and relational database engines need for the information they manage. Storage systems themselves, though, have some strong data management requirements, and many use Berkeley DB to satisfy those requirements.

Storage systems are not only the spinning disks on which files are written. Those disks are managed by software, including file systems and storage switches, which decide how and where to write out user data. In addition, large storage systems often include pooling software that allows many disks to be viewed as a single large repository, with the details hidden behind another layer of software. Finally, the administrative tools that an IT professional uses to monitor and control the storage pool are sophisticated pieces of software themselves.

All of these – the single volume manager, the pool manager and the administrative tools for managing the information life cycle – need to store and retrieve data reliably. For a single file, the information may include its owner, size and creation time. For a pool, the information may include the address, capacity and current usage for each device. For the management dashboard, the information may include policies, user identities and permissions. Berkeley DB is an excellent choice for data management in each of these applications.

Switching and Routing

The fabric of the Internet consists of a large number of switching and routing systems connected to one another. Getting data from a video server or search engine to your laptop computer these routers to find you, find the video server, and find one another, and then to transmit the data from end to end, often over several hops.

These devices need to keep track of their own locations in the network – for example, their IP addresses – and the systems that are connected directly to them. In addition, each must be able to look up the address for an arbitrary machine (your laptop, for example), and find another server that is likely to be closer to you.

This application needs extremely fast, but fairly simple, data management. Systems that keep track of network addresses and routes often rely on Berkeley DB.

Edge Applications

Infrastructure applications usually run inside the network. Email and routing are core services. Another collection of services, however, runs closer to the edge of the network, near your desktop or server machine. Many of these edge services use Berkeley DB for reliable data storage.

Distributed systems, including service-oriented architectures and Web-based applications, often need low-latency, high-throughput data storage at the edge of the network. Berkeley DB is a popular choice for these systems.

Logging, Auditing and Compliance

Many businesses need to share information with their partners. Vendors, for example, exchange purchase orders and invoices with their suppliers and their customers. Sharing this information often means exporting data from the financial systems at one company, sending it across the Internet to another, and loading it into a different financial system at the other end.

It is often important to keep logs of this shared information, for auditing and compliance purposes, or simply to speed up processing if you have to send it again. Berkeley DB is often used as a cache for information exported from, or imported into, legacy databases and applications. It sits at the edge of the network, captures incoming and outgoing traffic, and makes a permanent record of it for later review.

Data is read from and written to these logs using simple and predictable queries. They do not need the flexibility of SQL, and often cannot tolerate the overhead of a client/server RDBMS.

Caching for Scalable Web Applications

Major e-commerce and software-as-a-service sites on the Web need to find the right information for delivery to users, quickly and reliably. Many of these sites use Oracle Database as the backend repository for their transaction processing systems, including order management and billing. However, they often use Berkeley DB as a high-performance read/write cache to deliver data to Web browsers quickly.

In this case, Berkeley DB acts as a cache for information stored in the Oracle Database database. That cache may be augmented with other data – user preferences and identity, for example. When a user visits the Web site to make a purchase or fetch an article, the Web server generates the page if it does not already exist, and then caches the content in Berkeley DB for fast delivery the next time it is requested.

Devices

Consumer devices like mobile phones are increasingly powerful, and run more sophisticated applications than before.

These devices need the reliability, configurability and small footprint of Berkeley DB.

Berkeley DB is also used in some consumer devices. For example, a popular mobile telephone that can store and play back business documents, music and videos uses Berkeley DB as its repository. The phone also needs to store contact information, call logs, preferences and settings, and more. In this case, the data managed is often quite complex, but the access patterns are fairly predictable. Berkeley DB's small footprint – under a half a megabyte of code – is an important advantage in the constrained world of consumer devices.

Dynamic Data, Static Queries

In general, Berkeley DB excels in applications that have predictable data access patterns. The data it stores may change quickly, but Berkeley DB is best suited to applications where the queries themselves are static. Since developers need to write the code that fetches the information they need, they must be able to predict access patterns at design time, and write their application software to store and fetch the data properly in Berkeley DB.

By contrast, relational systems are extremely good at answering new questions. Since they include SQL processors, they can execute arbitrary queries. The developer need not predict the query patterns in advance. Even if the data they manage does not change very quickly, relational systems are outstanding at dynamic queries.

CONCLUSION

Fast, reliable and scalable data storage services are critical for a wide range of applications today. In the past, many developers were forced to choose between a large, full-featured and complex relational database engine and a low-level file system for data storage.

Berkeley DB introduces a third option. It provides the same enterprise-grade storage services as high-end relational engines like Oracle Database, but uses a much simpler and more flexible design that is better suited to embedded use.

To learn more about Berkeley DB, please visit Oracle's Web site at <http://www.oracle.com/database/berkeley-db>.



**A Comparison of Oracle Berkeley DB and Relational Database Management Systems
March 2009**

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Copyright © 2006, Oracle. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission. Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.