

An Oracle White Paper
February 2010

Berkeley DB Java Edition on Android

Introduction.....	3
Performance and Scalability	3
Direct Persistence Layer.....	4
Complex Relationship Modelling	5
Fast Indexed Access	6
True Multi-Threaded Support.....	7
Flexible Many-to-Many Transaction-to-Thread Model	7
Conclusion.....	8

Introduction

Oracle Berkeley DB Java Edition (JE) is an embeddable database implemented in pure Java. It provides a transactional storage engine that reduces the overhead of object persistence, while improving on the flexibility, speed, and scalability of object to relation mapping (ORM) solutions.

“The speed, scalability, and robustness of Berkeley DB Java Edition, compared to other SQL-based solutions, creates new Mobile opportunities on Android” said Chris Eastland of Nebula Software Systems an Essex, Massachusetts developer of business mobile applications. “We are using Berkeley DB to enhance our application framework to take advantage of the Oracle and Google stacks.”

Recently, Oracle certified JE on the Android platform for devices like the Motorola Droid and HTC Eris smartphones. Android breaks new ground in the device category because it is a Java 2 Standard Edition (J2SE) platform, whereas the previous generations of Java-based devices are predominantly Java Micro Edition (Java ME) based. There are significant differences between J2SE and Java ME in terms richness of libraries and APIs and this creates a big opportunity for improved application capabilities. Most notable are the full-featured Java 5 language support, libraries like `java.util.*` and collections, and full multi-threaded support built on the Android Linux kernel.

This paper highlights some of the features and benefits JE offers to the Android application programmer, including performance, scalability, indexing, concurrency control, transactions, and a many-to-many transaction-to-thread model.

Performance and Scalability

Oracle Berkeley DB Java Edition offers the Android application builder significant performance and scalability improvements over the native SQL-based data management software. Because JE is written in pure Java, there is no translation between Java and an underlying C library thereby letting it operate directly on the application's Java objects.

Further, JE does not have the overhead of SQL, so it incurs no penalty for parsing, optimizing, and interpreting requests. The result is that an Android application using JE can realize a 2x or better performance improvement compared to other Android-based data management libraries.

"The Direct Persistence Layer in Oracle Berkeley DB Java Edition has been very valuable to our software development efforts at the Children's Hospital Informatics Program," said Steven Boscarine, Principal Software Engineer, Children's Hospital Boston, a Harvard Medical School Teaching Affiliate. "We found that replacing JPA with DPL yielded a significant performance increase, made it easy to encrypt sensitive patient data, and allowed us to deliver scalable code much faster than we could have done with a traditional RDBMS with an ORM."

For example, using a Motorola Droid, JE can create a database of 100 simulated photos (1 MB each), in 27 seconds, more than three times as fast as the native database software. Tests on the same database demonstrate random access retrievals at 275 ms per record average (cold cache) and 110 ms (warm cache). Using a different database containing one million 100 byte records, JE performs random access reads (fetch) operations in 29 ms (cold cache) and 1.5ms (warm cache). These tests clearly demonstrate JE's scalable performance characteristics for small and large data sets and record sizes. An efficient database translates into longer battery life and better responsiveness.

Direct Persistence Layer

While relational databases are a sophisticated tool available to the developer for data storage and analysis, they are not ideal for storing Java application data since they require (un)marshalling objects to (and from) tuples. For more complex data models, an RDB can create an "impedance mismatch" between objects and database schema. Further, an RDBMS may be overkill for the device-programmer since the analytic capabilities of SQL are generally not needed.

JE's Direct Persistence Layer (DPL) lets the programmer model application data using Plain Old Java Objects (POJO) without worrying about (un)marshalling code or Object Relational Mapping (ORM) tools. Simple Java annotations to the application program's classes are all that are needed to use the DPL. Further, there is no SQL overhead with DPL. The result is faster storage, lower CPU and memory requirements, and a more efficient development process. But despite the lack of a query language, Berkeley DB Java Edition can access Java objects in an ad hoc manner and still provide fast, reliable and scalable data storage in a small, efficient, and easy to manage package.

Consider an Address Book application. To store instances of a `Contact` class using the JE DPL, the programmer simply annotates the class with `@Entity` and the primary and (optional) secondary keys with `@PrimaryKey` and `@SecondaryKey`, respectively:

```

@Entity
class Contact {
    @PrimaryKey
    String name;
    String street;
    String city;
    String state;
    int zipCode;
    @SecondaryKey(related=ONE_TO_ONE)
    String phone;
    private Contact() {}
}

```

Storing the fields in a `Contact` instance is handled transparently by JE without requiring the programmer to implement any special interfaces. The application classes define the schema; the annotations define the metadata. Access to the `Contact` data at runtime is through primary and secondary indices. For example, to create an entry in the address book for George Smith, we write:

```

PrimaryIndex<String, Contact> contactsByName = ... ;
Contact george =
    new Contact("George", "123 Rock Drive",
               "Bedford", "CA", "90222");
contactsByName.put(george);

```

To look up his neighbor Bill:

```

Contact bill = contactsByName.get("Bill");

```

Complex Relationship Modelling

JE's storage capabilities are even more important when there are multiple related objects in the schema. Consider a mobile sales order entry system in which salespersons enter orders on their Android handset. An order consists of an `Order` object with references to a `Customer` object; multiple `Parts` with SKUs; and a series of strings containing special instructions (notes) for the order. Modeling this using the JE DPL is quite simple:

```

@Entity
class Order {
    @PrimaryKey(sequence="ID")
    long orderId;
    @SecondaryKey(related=MANY_TO_ONE,
                 relatedEntity=Customer.class)
    String customerName;
    @SecondaryKey(related=MANY_TO_MANY,
                 relatedEntity=Part.class,
                 onRelatedEntityDelete=NULLIFY)
    Set<Long> skus = new HashSet<Long>();
    List<String> notes = new ArrayList<String>();
    private Order() {}
}

@Entity
class Customer {
    @PrimaryKey
    String name;
    String address;
    ...
}

@Entity
class Part {
    @PrimaryKey(sequence="ID")
    long partSku;
    ...
}

```

JE maintains consistency of all the relationships between the various entities as well as the storage of the fields, thereby simplifying the code and the programmer's task.

Fast Indexed Access

JE's indexing capabilities are valuable in a mobile environment, not only because they provide relationship capabilities like the ones shown in the above sample schema, but also because they provide fast access to the data. Our sample mobile sales application keeps a local read-only copy of the widget catalog on the device and utilizes JE's indices for fast search and retrieval without going over the web to a backend server. JE supports primary and secondary indices, composite keys, keys based on complex classes, lazy index population, and key prefixing. On Android, JE databases can reside on the flash memory card (for example, up to 32 GB on a Motorola Droid, 8GB on HTC Eris) facilitating large local data sets.

On a Motorola Droid, a random access read of a 1 KB record in a 100,000 record (100MB) database is about 1 ms using JE and about 5 ms using the native Android SQL-based database.

True Multi-Threaded Support

JE provides versatile multi-threaded concurrency control, an important feature for many applications on the Android platform. For example, consider an application which uploads and downloads to (and from) a server in a background thread while the user is entering or viewing data on the screen. In this scenario, the application may benefit from loosening locking requirements to permit these background updates. JE lets the programmer adjust locking and concurrency by supporting all four levels of ANSI serialization. Some of an application's data may require strict two-phase locking, which JE enforces by default. On the other end of the spectrum, there may be a set of data for which strict inter-thread locking is not required, and for that, JE allows data-access using dirty reads.

Flexible Many-to-Many Transaction-to-Thread Model

Programs can make use of JE's transaction capabilities by wrapping one or more operations with `beginTransaction()` and `commit()` (or `abort()`) method calls. Transactions might seem overkill on a device like a smartphone, but consider a mobile sales order entry application where a user enters multi-component orders into the device and then uploads them to the server for fulfillment processing: Atomicity and consistency over the orders in the local storage is important. For example, depending on the outcome of a large order upload -- success or network failure in the middle of uploading -- the application may want to delete all of the local order elements in a single transaction. Or, the application may want to read several objects from a device-local database in a transactionally consistent manner while a background task is concurrently updating data received from a server (e.g. updates to email/text messages or 'friend' presence status). This is a scenario where the availability of different serialization options is useful. Further, JE has a many-to-many transaction-to-thread model; multiple threads may use a transaction and one thread may use multiple transactions, allowing you the greatest flexibility over the ACID characteristics of the application.

Conclusion

Android has created the next generation of mobile device technology by implementing a J2SE stack capable of supporting sophisticated multi-threaded database applications. In turn, Berkeley DB Java Edition and its Direct Persistence Layer provide scalable, transactional data management to the new breed of Android applications and services.

For more information, see:

Oracle Berkeley DB Java Edition

[\(<http://www.oracle.com/database/berkeley-db/je/index.html>\)](http://www.oracle.com/database/berkeley-db/je/index.html)

Oracle Berkeley DB Product Family

[\(<http://www.oracle.com/database/berkeley-db/index.html>\)](http://www.oracle.com/database/berkeley-db/index.html)

Oracle Berkeley DB Blog (<http://blogs.oracle.com/berkeleydb/>)

Charles Lamb's Blog (<http://blogs.oracle.com/charlesLamb/>)



Berkeley DB Java Edition on Android
February 2010
Author: Charles Lamb

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



Oracle is committed to developing practices and products that help protect the environment

Copyright © 2010, Oracle and/or its affiliates. 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 and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. UNIX is a registered trademark licensed through X/Open Company, Ltd. 0110