EJB3 JPA Persistence Performance & Scalability Analysis

BEST IN INDUSTRY RESULTS For Performance And Scalability With Raw Data Volumes and Data Complexity In EJB3 JPA Persistence Applications.

CocoBase is 2 to 4 times on average faster than its closest competitors; Hibernate, OpenJPA and Toplink Essentials.

CocoBase® PURE POJO - Technical Paper
EJB3 JPA Persistence Performance & Scalability Analysis

CocoBase® PURE POJO™ Provides BEST IN INDUSTRY RESULTS For Performance And Scalability With Raw Data Volumes and Data Complexity In EJB3 JPA Persistence Applications.

CocoBase is 2 to 4 times faster on average than its closest competitors; Hibernate, OpenJPA and Toplink Essentials.

INTRODUCTION

Performance and scalability of an ORM technology is based on the ability to manage the persistence operations of the data elements and the relationships between the data elements between the application and the database. This distinguishes the two aspects of performance to be: 1) managing the raw data volume of persisting the data elements or objects, and 2) managing the persistence of the relationships between the data elements or objects which can be described as the “complexity of the data or object model”.

raw data volume - the number of data elements or objects being persisted (amount of data processed)
data complexity - the number of relationships being persisted (complexity of data or object model)

Therefore, it is important to assess an ORM tools’ ability to provide performance and scalability for both raw data volume and then data complexity. The results of this assessment determine if an ORM tool will be effective to use for developing and deploying an enterprise application.

RAW DATA VOLUME DETAILS & PERFORMANCE TEST RESULTS

This part of the test is focused on data volume and so data complexity was kept very low. It is important to understand that this aspect of performance is of less importance than data complexity. While this measure provides an important starting point for an ORM tool, it is not representative of the typical beginning level application that would have a need for using an ORM tool. The typical application that actually has a need for an ORM tool will usually have a medium level of data complexity.

When looking at raw data volume it is important to note that in previous performance tests done on CocoBase® showed that it performed on average twice as fast as the same CRUD operations running without an ORM tool with just a JDBC driver. This is the result of CocoBase's® ability to only persist data that needs to be persisted which can be described as “smart processing”.

The raw data volume performance speeds of the other ORM technologies tested here confirm the common understanding that ORM tools perform 15 to 25 % slower than just using a JDBC driver. This is expected, since the ORM provider is a layer on top of the JDBC driver which would be expected to provide a performance decrease to get the benefit of the extra ORM tool services. It is important to note that the architecture of CocoBase® is the only ORM tool architecture that provides raw data volume performance faster than just using a JDBC driver by an average of 200 to 300%.
Raw Data Volume Performance Results

The results here are the average time it takes the ORM technology to process a CRUD database operation with the focus on high data volume and very low data complexity.

<table>
<thead>
<tr>
<th>TOOL</th>
<th>TIME (MS) FOR AVERAGE CRUD OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CocoBase®</td>
<td>.70 ms</td>
</tr>
<tr>
<td>Hibernate</td>
<td>1.67 ms</td>
</tr>
<tr>
<td>Toplink Essentials</td>
<td>2.49 ms</td>
</tr>
<tr>
<td>OpenJPA</td>
<td>1.526 ms</td>
</tr>
</tbody>
</table>

This result verifies that CocoBase® is much faster than the other four ORM tools by 2 to 21 times (200 - 2,100%). This evidence does a good job of verifying the prior statements that most ORM tools are 15 - 25% slower than just using a JDBC driver with no ORM tool. It is also noted that all four ORM technologies retain this performance average as the data volume is increased. This should be expected and is mostly attributed to the strength of the JDBC driver which for the most part also remains consistent.

DATA COMPLEXITY DETAILS & TEST RESULTS

This part of the test is focused on increasing the level of data complexity which can also be referred to as the object model for an application. The typical application in use has a medium level of data complexity which is the standard beginning point for assessing the effectiveness of an ORM technology.

The statement can be made that the key benefit of an ORM technology is directly attributable to its ability to provide high performance and scalability for the demanding range of data complexity for enterprise applications. This important ORM benefit is where the true value of an ORM technology is provided and becomes the benchmark for determining if an ORM should be used.

Linear performance characteristics for scaling the range of data complexity is required to effectively provide the needed benefits of ORM. This means that the ORM technology performance is the same for processing a CRUD operation no matter how complex the data or object model is of the application. When an ORM tool is not able to perform in this manner, it greatly impacts the application from development through to deployment. For example, if there the ORM technology cannot provide linear performance and bogs down then tough scalability constraints are required. This will require that the application be written in a manner to segment it into small units of work. This keeps data set sizes to a minimum and does not overload the network throughput; however, this approach directly decreases performance since the number of calls to the database increases dramatically.

When a scalability constraint defines the architecture of the application, you find that it begins to quickly reduce any benefit of using the ORM tool as the performance degrades along with productivity due to the increase in difficulty of designing the application. Additionally, if the ORM tool slows down its’ performance as the data complexity increases, this will require additional computer processing power and throughput just to consistently run the application. It is important to note that this increases hardware costs to a great degree which also increases the software costs as database instances and related software instance licensing increases. Most software tools such as application servers and databases increase their cost as increases in CPU, instances, etc. are increased. It does not take but small increases in these typically quite expensive fees to completely outweigh the ROI benefit of using an ORM tool.
In other words, the linear performance characteristics of CocoBase place little to no scalability constraints upon the development of the application. This makes it entirely easier to write scalable applications, which greatly improves productivity and reduces project costs. Another benefit of linear performance is that the network throughput required is much lower which reduces the amount of computer hardware and software needed to successfully manage running the application. CocoBase easily provides the lowest total cost of application ownership for an ORM tool as an overall result of increased productivity, higher performance with lowered network costs, even when compared to free professional open source alternatives.

Data Complexity Performance Test Results

<table>
<thead>
<tr>
<th>TOOL</th>
<th>TIME (MS) FOR AVERAGE CRUD OPERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CocoBase®</td>
<td>0.186 ms</td>
</tr>
<tr>
<td>Hibernate</td>
<td>0.66 ms</td>
</tr>
<tr>
<td>Toplink Essentials</td>
<td>41.65 ms</td>
</tr>
<tr>
<td>OpenJPA</td>
<td>0.755 ms</td>
</tr>
</tbody>
</table>

In conclusion, this test shows that CocoBase® performance remains the same even with increases in data complexity which can be described as linear scalability of performance. Both Hibernate and OpenJPA remain linearly slower than CocoBase®, whereas Toplink Essentials does not have linear scalability as its performance per relationship decreases rapidly as the complexity of the data increases.

WHY IS RAW DATA PERFORMANCE & DATA COMPLEXITY SCALABILITY SO CRITICAL FOR ENTERPRISE DEVELOPMENT?

"Raw Data Volume Performance" is critical for real enterprise multi-user environments because:

- Less time is required to commit transactions on the client (Java) side, thus allowing a higher number of transactions per minute, which not only saves database server resources (memory, CPU, network, etc.) but also reduces the chances of transaction conflicts/failures. This means that server usage is optimized.

- Less application server and client hardware is needed, as a consequence of the optimal usage of the JVM.

The result is better performance means “fewer resources are required to perform the same operations”, which in ROI terms mean “it costs less to do the same operations”.

"Data Complexity Scalability Performance" is of primary importance for enterprise environments because of the need to provide Linear Performance for increasing amounts of data complexity to retain consistent performance. This makes it more critical than raw data volume performance. The CocoBase® JPA Persistence manager provides "linear" scalability. This means that developers do not need to architecturally rearrange or partition their code in order to obtain the desired scalability.

Most key ORM technologies have linear performance that degrades at a moderate level while CocoBase® retains nearly consistent linear performance. And take note that some ORM technologies have linear performance that decreases exponentially to the number of objects being managed. This will mean that developers will need to write their code with a top heavy focus on scalability that will limit the ability to meet other requirements of performance as well as reduce simplicity. In these worst cases, the developer cannot
allow the number of objects to grow too much in the same persistence manager session. This imposes serious design constraints to application coding so that, from time to time, the application needs to limit access to data, or worse still “clear” the persistence manager cache to avoid scalability problems. As a consequence, not only application coding is more complicated but database objects need to be re-read several times thus consuming additional server resources.

Scalability limitations can also be a problem with some JDBC features such as “batch” updates which will loose part of their value considering the data volume must be partitioned in many different transactions. And there can be situations where developers might end up choosing to write raw JDBC code in order to avoid such scalability issues which reduce productivity, performance and flexibility.

To sum up, Linear Performance is extremely important when dealing with increasing Data Complexity because;

- This directly reduces the number of database server instances, application server instances and client hardware needed for managing the persistence of increasing data complexities which is required to meet budgetary restrictions.

- Otherwise scalability constraints are imposed to the application which increases complexity of the development and is usually in direct conflict with other application requirements. Eliminating scalability restraints with linear performance allows applications to be coded in an elegant and unconstrained way no matter what is the estimated number of objects that might be involved in the server transactions.

Therefore, it is important to think of raw data volume performance and linear performance for increasing Data Complexity not only as a goal but also as a proof of concept that the ORM technology architectural design is lightweight, robust and optimal. CocoBase’s® amazing raw data volume performance as well as necessary linear performance is a consequence of years of research and improvements aiming to provide just such an architecture.

It is important to point out that the test results presented here correspond to the beginning level performance for CocoBase® as these are the “out-of-the-box” results without caching or any other “ORM tricks”. Further optimizations can provide additional performance increases depending on application requirements. For example, CocoBase® is the only tool that allows the developer to actually “design” the SQL that is produced at runtime and without loosing the flexibility provided by the dynamic generation of SQL. This means that even optimization hints could be embedded into the SQL, thus allowing additional optimizations on the server side. And note this is different than “hard-coded fixed” SQL with just a few bindings in it. In CocoBase® the SQL is still generated dynamically which means that it is still optimized at runtime (e.g. only the changed columns are updated in an update SQL statement). This characteristic of CocoBase® is entirely unique and directly improves performance and scalability within enterprise environments.
EJB3 JPA - POJO Sample Application Performance Test Suite Details & Results

The test case we created to analyze JPA performance was designed to specify both light and heavy object models and data volume sizes. The combination of these parameters within the dimensions of testing, very quickly assesses the performance and scaling characteristics of a given JPA Persistence Manager. A basic JPA application was built using the JPA APIs, which allows any JPA persistence manager to be used to manage the persistence. The four ORM tools with JPA support used in this performance test suite are CocoBase® Pure POJO™ Version 5, Hibernate JPA, OpenJPA and Toplink Essentials. Please note that OpenJPA, Hibernate and Toplink Essentials are open-source ORM tools and CocoBase is the only commercially created and supported ORM tool.

The tests suite is designed to show the results of both Raw Data Volumes and Data Complexity levels. The results here are setup as follows;

1) Low Data Complexity (therefore the focus is on Raw Data Volume)
2) Medium Data Complexity (therefore the focus is on Linear Performance)
3) High Data Complexity (therefore the focus is on Linear Performance)

To accomplish this, the test suite allows the application to be made complex by simply increasing the number of parents, children and grandchildren in the object model. The results shown here were created by an outside consultant to help insure an unbiased and accurate performance numbers that are clearly an “apples to apples” comparison.

Details of test platform and test settings are:

Application Running on: Dual AMD Opteron 265 Dual Core Processors, 3.23 GB RAM, Serial-ATA RAID HD, Windows XP Professional, Oracle 9i running remotely.

Database running on: AMD Opteron 265 Dual Core Processors, 1 GB RAM, Serial-ATA RAID HD, Windows 2000 Professional, Oracle 9i running locally.

Test client running on same platform locally: JDK 1.6.0_01-b06
Database server access remotely using pure type-4 jdbc java connection
Object Model sizes varies from Simple to Complex
Number of runs set to 5 best run for each product presented
n parents, n children, n grandchildren, best run chosen

TEST ONE – Low Data Complexity (Allows Focus on Raw Data Volume)

This test is designed to show raw data volume performance with a simple application and low or light data volumes on a simple object model. The test is run with the following parameters; 100 parents, 1 children, 1 grandchildren. In order to support 1000 root nodes, the beta of OpenJPA requires special handling of Connection Pool resources, see the configuration section for details.

The colorful graph represents the results of the test. The blue bars represent the time required (ms) for the CocoBase operation, the green bars represent the time required (ms) for the equivalent Hibernate operation, the red bars represent the time required (ms) for the equivalent Toplink Essentials operation and the magenta bars represent the time required (ms) for the equivalent OpenJPA operation.
The TEST ONE - Low Data Complexity test results were provided and or validated by a third party consultant included in this document, and your results should be a graph similar to the following:

![JPA Performance Test Graph](image)

The in-house result values are as follows:

<table>
<thead>
<tr>
<th>Operation</th>
<th>CocoBase Time (ms)</th>
<th>Hibernate Time (ms)</th>
<th>Toplink Essentials Time (ms)</th>
<th>OpenJPA Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td>719</td>
<td>3266</td>
<td>4296</td>
<td>3359</td>
</tr>
<tr>
<td>LOAD</td>
<td>6860</td>
<td>10203</td>
<td>18265</td>
<td>9860</td>
</tr>
<tr>
<td>UPDATE</td>
<td>359</td>
<td>3578</td>
<td>3640</td>
<td>2422</td>
</tr>
<tr>
<td>DELETE</td>
<td>531</td>
<td>3047</td>
<td>3671</td>
<td>2672</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>2117</td>
<td>5023</td>
<td>7468</td>
<td>4578</td>
</tr>
</tbody>
</table>

The results show that for the best run, the **CocoBase® JPA implementation is on average about 2-1/2 times as fast as Hibernate, three times faster than Toplink Essentials and over two times faster than OpenJPA for the same testcase.** The CocoBase runtime shows a distinct performance advantage, even on a simple performance test that does not express a complex data set or complex data model.
SECOND TEST – Medium Level Data Complexity

In this test the parameters are modified so that there are more relationships between data, with a medium complex dataset that involves more children. This details an impressive example of the performance advantages of the CocoBase JPA implementation.

A second run with the settings of 5 parents, 15 children and 25 grandchildren results in the following graph:

The in-house result values are as follows:

<table>
<thead>
<tr>
<th>Operation</th>
<th>CocoBase® JPA Time (ms)</th>
<th>Hibernate Time (ms)</th>
<th>Toplink Essentials Time (ms)</th>
<th>OpenJPA Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td>438</td>
<td>688</td>
<td>8703</td>
<td>1922</td>
</tr>
<tr>
<td>LOAD</td>
<td>688</td>
<td>891</td>
<td>17703</td>
<td>750</td>
</tr>
<tr>
<td>UPDATE</td>
<td>234</td>
<td>2672</td>
<td>9234</td>
<td>1609</td>
</tr>
<tr>
<td>DELETE</td>
<td>281</td>
<td>2500</td>
<td>2359</td>
<td>1719</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>410</td>
<td>1687</td>
<td>9499</td>
<td>1500</td>
</tr>
</tbody>
</table>
The results of the Second Test - Medium Data Complexity shows a significant improvement in CocoBase® JPA performance and the serious degradation in performance of the other JPA implementations. The CocoBase® performance improvement results from enterprise algorithms designed to scale better linearly. The algorithms in CocoBase® are better implemented to scale complex data sets and the JDBC resources are better utilized to reduce overhead in creation and execution of SQL. The CocoBase® SQL maps are compiled just once per VM instance and only when a map is required to perform an operation. Once compiled, query arguments are bound as needed to the buffered SQL strings, and SQL is not regenerated unnecessarily. These approaches are not found in the other JPA implementations.

THIRD TEST – High Data Complexity

This tests modifies the test parameters again so that there are more relationships between the data thus creating a high complexity dataset that involves more children. This shows results that are, once again, a very impressive example of the performance advantages of the CocoBase® JPA implementation.

A third run with the settings of 3 parents, 30 children and 50 grandchildren results in the following graph:
The result values for the **Third Test - High Data Complexity** are as follows:

<table>
<thead>
<tr>
<th>Operation</th>
<th>CocoBase® JPA Time (ms)</th>
<th>Hibernate Time (ms)</th>
<th>Toplink Essentials Time (ms)</th>
<th>OpenJPA Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT</td>
<td>1031</td>
<td>1312</td>
<td>252500</td>
<td>4563</td>
</tr>
<tr>
<td>LOAD</td>
<td>1110</td>
<td>1328</td>
<td>263000</td>
<td>1171</td>
</tr>
<tr>
<td>UPDATE</td>
<td>500</td>
<td>4703</td>
<td>243297</td>
<td>3865</td>
</tr>
<tr>
<td>DELETE</td>
<td>781</td>
<td>4844</td>
<td>6453</td>
<td>4265</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>855</td>
<td>3046</td>
<td>191312</td>
<td>3468</td>
</tr>
</tbody>
</table>

The results of this test run with an increasingly complex data model shows a significant improvement in CocoBase® JPA performance and the serious degradation that can occur with other JPA implementations. If you do not see performance numbers that appear to be consistent with those in this document, please engage our support staff so we can assist you with any configuration, installation or understanding of the results that you may require.

The results of a third run reflect the benefits of the CocoBase JPA performance when handling a more complex data set or object model. This final example runs a test with the sort of data complexity that is typical of many enterprise applications. Even this third test example shows only relatively moderately complex data and data volumes compared to typical enterprise applications. The overhead of managing these relationships in the other JPA implementations becomes quickly evident with the increased number of relationships. The clear conclusion is that efficiently managing the persistence of complex data with linear scalability is highly critical and thus required.

Fluctuations may occur in the results due to a combination of the following:

- Variations in system speed
- Number of tasks running
- Garbage collection
- Hotspot compiler activation
- JDBC driver implementation
- Fluctuating database load
- Driver and Database Support for optimizations such as statement batch and prepared statement reuse support
- Memory availability
Understanding Why The CocoBase® PURE POJO™ Solution Is So Effective

CocoBase® Pure POJO™ Version 5 is a highly optimized, enterprise requirements focused, Object to Relational Mapping (ORM) platform. It implements data persistence for Plain Old Java Objects (POJO’s) and EJB Entity Beans with Java applications using both the CocoBase Persistence Manager APIs and the industry standard EJB3/JPA Persistence APIs. CocoBase® Pure POJO™ makes it easy for developers to design, store and maintain custom data persistence patterns thus eliminating the need for hand-coded and proprietary persistence implementations.

Persistence patterns designed with CocoBase® Pure POJO™ are stored as either EJB3 or CocoBase Repository maps in EJB3 Class annotations or XML file based repositories. These maps are retrieved and placed into active memory (as required) when application POJO value objects, EJBs or other object model instances are loaded into the JVM. CocoBase OR class mappings can be defined on top of lower level maps called SQL maps. SQL maps are intended to specify the SQL that should be generated dynamically at runtime for each database CRUD operation. A SQL map is different than a simple storage for hard coded SQL. Instead, SQL maps consist of a patented technology that can be used to define SQL “persistence patterns”, and can be totally independent of object class model or relational table model constraints. This gives high flexibility to the developer on fine tuning the SQL that is generated at runtime, while promoting sql map reuse across different application object models (throughout the enterprise). For instance, POJO or EJB3 objects that utilize identical persistence patterns could share a common SQL map instance.

CocoBase maps are very flexible and easily configured using the provided GUI Mapping Workbench. If an object model changes or if additional mappings are required, the CocoBase maps can easily be reconfigured and reloaded with the new or updated application(s).

The CocoBase® Pure POJO™, Version 5 distribution includes an easy to learn Java Swing based Mapping Workbench GUI toolset. This toolset allows rapid map development, reverse engineering and a Magic Mapper facility to intelligently auto-reconcile class and database structural differences. CocoBase also includes a source code generation utility for generating value object, EJB, and other object model source files. Object model definitions require no proprietary code. Persistence is provided through open and extensible proxy classes which utilize the Java Reflection APIs.

The annotation support bundled with CocoBase® Pure POJO™, Version 5 is an implementation of the EJB3 JPA standard. This facility is used to load and map the persistence of all three JPA implementations used in this test. The EJB3 API supports different approaches to mapping JPA - POJO objects. The first is direct use of annotation in EJB3 (almost POJO) classes as this example test case uses. The second is the use of the EJB3 XML mapping format also defined in the EJB3 standard format, which included in Service Release 2 of CocoBase® Pure POJO™, Version 5. The third option is the use of a proprietary or custom mapping repository such as the CocoBase® Pure POJO™, Version 5 project oriented enterprise mapping repository. While the performance tests do not demonstrate the last two options, they are supported in Version 5 Service Release 2 and newer releases of the CocoBase® Pure POJO™, Version 5 JPA implementation.

CocoBase® PURE POJO™ Highlights

Patented, Highly Flexible Dynamic O/R Mapping Layer Architecture

CocoBase provides a powerful yet simple to use approach for persisting data that decouples the object from being hard-coded to the database. CocoBase takes the database specific code and prewritten SQL out of the object and saves this information to the mapping layer (i.e. map). The database specific code and SQL is
dynamically created at runtime. This allows objects to be easily reused over and over enterprise-wide. This dynamic layer provides amazing flexibility as well as a multitude of performance and scaling optimizations that are easily changed by the programmer to best fit their specific requirements. The architecture is protected in landmark patents thus protecting the customer as well.

CocoBase® is the only apparent O/R Mapping tool that performs faster than raw JDBC with identical JDBC driver and SQL. This is not a ‘trick’ of adding caching or making CocoBase® use optimizations not used in the JDBC code - instead it’s an apples to apples comparison. No caching enabled, same SQL, same JDBC driver, same transaction settings, and same prepared statement handling. This is accomplished through optimized code that in combination with the Java just in time compiler produces a faster ‘executable’ when running. After years of working with the Java JIT, CocoBase® has evolved it's highly optimized code in this way creating a solution unique to CocoBase®. As a result of this optimization, raw SQL operations with CocoBase® execute on average twice as fast as they do with JDBC.

Caching

CocoBase® provides the fastest Java caching architecture currently available. Our patent pending implementations include performance improvements as much as 400-500 times that of raw SQL loads. This is a 40,000-50,000% increase! The CocoBase® caching implementation is on average 10-100 times faster than other typical caching architectures available in Java and provides a unique level of performance and scalability.

Handling of Complex Models Performance is Linear

CocoBase® object graph performance is ‘linear’. The performance cost of managing 1 object or 10,000 objects which have changed is the same per object CPU-wise. We know of no other Object to Relational Mapping tool that retains linear performance for complex models. The analysis contained within this document shows that while some Open Source JPA implementations are better than others resource wise, none retain linear object management scalability. This is but one example of scalability issues that plague o/r tools other than CocoBase®.

Good Use of Combined Calls

CocoBase® supports advanced SQL optimizations such as Cartesian and combined SQL loads that can reduce SQL traffic and database querying. Batch Inserts, stored procedure support, optimization of SQL (including support for database hints and proprietary syntax) all provide a dynamic ability to support complex and optimized solutions for loading and persisting POJO object models.

Extremely Efficient SQL

By default CocoBase® generates SQL based on the context of the operation. So for instance if an update of only a single column is required, that’s what’s issued. If database custom syntax such as hints or proprietary SQL statements are required, the admin facilities allow the developer to edit SQL mapping and create this custom syntax.

Effective JDBC Commands

Out of the box, the CocoBase® runtime uses best practices for managing JDBC connections, prepared statements, cursors, and SQL generation and execution. As a result, the application runs in an optimized and high performance fashion that generally outperforms equivalent hand coded applications by 2-3 times often with orders of magnitude of performance improvement.
Request CocoBase® Online To See Performance Test For Yourself!

Please go now to the Thought Inc.® website (www.thoughtinc.com) and request to evaluate CocoBase® PURE POJO v5. Sales will contact you to start the sales process and during the proof of value step you will be able to see the CocoBase performance test suite for yourself.

See why CocoBase® is chosen for enterprise level development of JPA based applications.

We at Thought Inc.® look forward to working with you.

COMPANY INFORMATION

THOUGHT Inc.®, in business since 1993, has been shipping the CocoBase® Enterprise O/R product since early 1997 and is currently in its’ 5th major release. THOUGHT Inc.® invented and patented repository based Dynamic Object to Relational Mapping™. CocoBase® is by far, the most mature Java based O/R Mapping tool available and leads the industry in technological innovation. This is why so many of our customers rely on CocoBase®! For more information please see the website at www.thoughtinc.com.

LEGAL NOTICES

Copyrights
Copyright 2007, THOUGHT Inc., 5 Third Street suite 815, San Francisco, CA 94103 USA. All rights reserved. Copyright in these (any and all contained herein) documents is owned by THOUGHT Inc. This material is for personal use only. Republication and re-dissemination, including posting to news groups, is expressly prohibited without the prior written consent of THOUGHT Inc. This publication is provided “as is” without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement, to also include any and all technical inaccuracies or typographical errors.

Patents
CocoBase® is a patented product under patent #5857197 as well as patents pending for object caching, object navigation, dynamic object querying technologies, etc.

Trademarks
CocoBase®, THOUGHT Inc.®, Dynamic O/R Mapping™, Dynamic Transparent Persistence™, Pure POJO™ and others are all trademarks of THOUGHT Inc.® All other trademarks are owned by their respective owners.