

Migrate from EJB 2 container-managed persistence to pureQuery for IBM Master Data Management Server, Part 1: Evaluating pureQuery technology and making the decision

Skill Level: Intermediate

[Wei Zheng \(wzheng@ca.ibm.com\)](mailto:wzheng@ca.ibm.com)

Product Architect - MDM Server
IBM

[Paul van Run \(pvanrun@ca.ibm.com\)](mailto:pvanrun@ca.ibm.com)

Senior Technical Staff Member - MDM Architect
IBM

[Kathy Zeidenstein \(krzeide@us.ibm.com\)](mailto:krzeide@us.ibm.com)

Senior Software Engineer, Enablement
IBM

03 Jul 2008

If you've been curious or confused about the new release of WebSphere® Customer Center (now named IBM® InfoSphere™ Master Data Management Server), then this series is for you! This series describes how and why pureQuery technology was used in the new release, the implementation and migration to pureQuery, and the results of performance and capability testing to validate this critical decision. Part 1 focuses on the evaluation of persistence mechanisms and our plan to validate the technology.

Background

In 2007, the IBM Master Data Management team was planning for a major new release of their WebSphere® Customer Center product which was to be renamed IBM InfoSphere Master Data Management Server. One critical architectural decision the team had to make was what to do about the existing persistence mechanism, which was based on a mixture of Enterprise Javabeen (EJB) 2 container-managed

persistence (CMP) entity beans and native JDBC calls, and was becoming outdated. This two-part series describes how and why we made the decision to go with pureQuery technology; our plan for implementing and migrating to pureQuery; and finally, the results of performance and capability testing to validate the decision.

Overview of InfoSphere Master Data Management Server

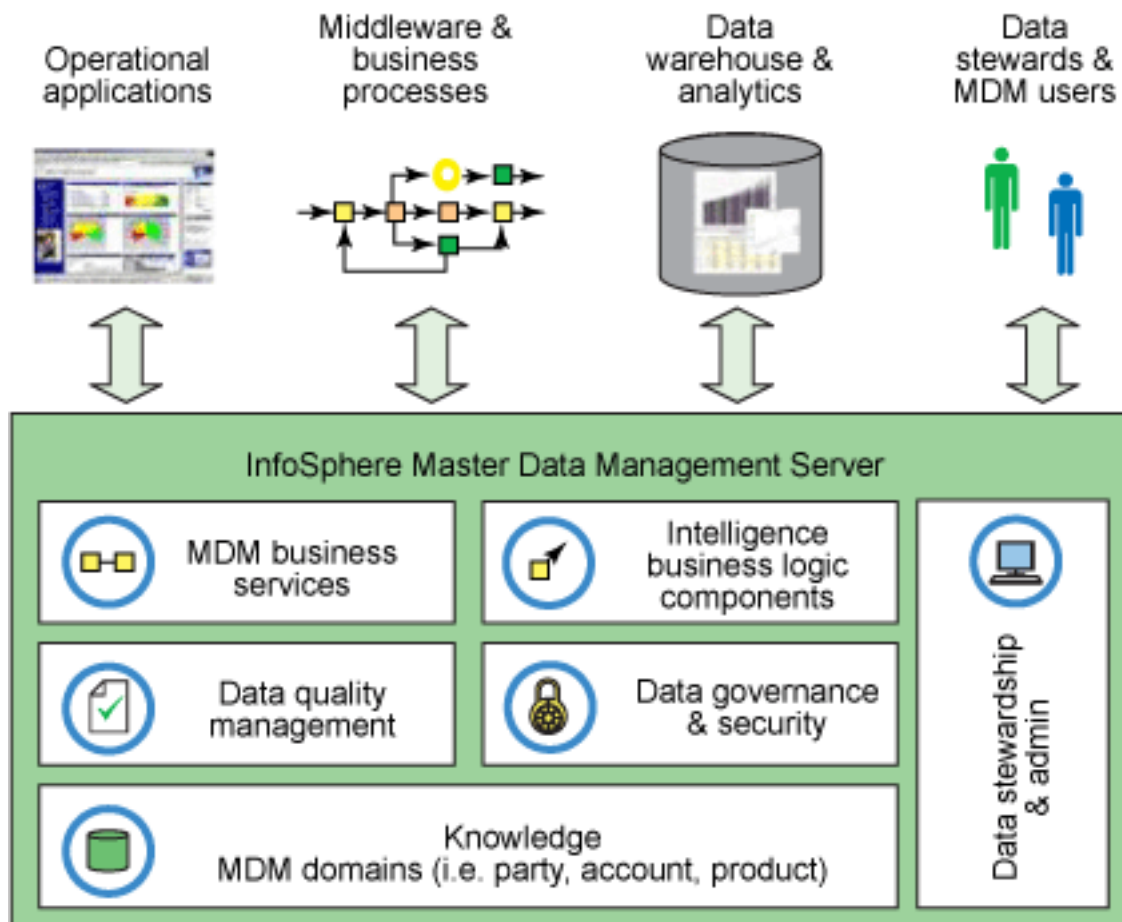
IBM InfoSphere Master Data Management (MDM) Server manages the master data entities that drive the most important business processes within an organization (that is, customer, accounts, and products). IBM delivers a comprehensive MDM solution with significant out-of-the-box functionality serving all master data management approaches. With MDM Server, an organization can centralize its most critical data into a single trusted source -- allowing them to identify their most valuable customers, increase revenues, and reduce costs. This capability enables them to derive more value from their existing systems, improve customer satisfaction, and reach new markets quickly.

There are typically many uses of master data (operational systems, data warehouses, business processes, data governance, and so on) and they have very different requirements. The uses do, however, share some key requirements for an MDM system. Such a system needs to:

- Provide a unified multi-domain (party, account, product) master database -- the knowledge layer
- Provide SOA business services to manipulate and query this data
- Manage quality and correctness of the data
- Provide intelligent, pro-active business logic to make MDM an active participant in the data lifecycle
- Provide data governance capabilities to enforce and audit both transaction and data access security

The following overview of the MDM product illustrates how these needs are met:

Figure 1. MDM product overview



So, in essence, IBM InfoSphere MDM Server is an SOA application with a large set of predefined business services to maintain master data (or master data references) in its database. For example, a request for an update is accepted by the MDM server in XML and parsed into object form; its security is checked; its content is validated, standardized and deduplicated; and then the changes are applied through a persistence layer. The persistence layer is the focus of this article.

IBM InfoSphere Master Data Management Server is the successor to IBM WebSphere Customer Center. It is an enhancement of the functionality in WebSphere Customer Center with the additional master data domains of Account and Product. So, where this article mentions WebSphere Customer Center, it is referring to the previous architecture, and where it uses MDM Server, it is referring to the new one.

Overview of Data Studio Developer and pureQuery

The IBM Data Studio portfolio offers an integrated, modular data management environment to design, develop, deploy and manage data, database, and data-driven applications throughout the entire data management life cycle. The focus

area in this article is the pureQuery technology as delivered in Data Studio Developer and the Data Studio pureQuery Runtime. pureQuery is a high-performance, Java™ data access technology on top of JDBC that makes it easier to develop and deploy database applications and services. pureQuery was developed to bridge the gap between Java and the database. Where these worlds meet is, generally, an area where performance and productivity issues abound.

Coding direct JDBC can be extremely tedious and verbose. In addition, it really helps to be an expert in JDBC and SQL to ensure that your JDBC data access is efficient. For optimal performance, developers must master the JDBC API and exploit features such as batching and result optimizing. Basically, you have to use a lot of code to write a simple query using JDBC.

The tedium of JDBC development led to the creation of the object relational mapping (ORM) framework, which provides a data access abstraction layer. ORMs generally require less initial effort to create the data access layer. However, those frameworks can add overhead and additional complexity when diagnosing runtime performance issues. Tuning and diagnosis become more difficult because the developers no longer control what SQL gets sent to the database; as a result, it's difficult to change the SQL or determine which application issued it. IBM created pureQuery to address these and other issues for Java data access development. pureQuery introduces a new API that is simple and intuitive for Java developers previewing capabilities under consideration for JDBC standardization. Plus, pureQuery lets you use the full power of SQL on Java collections and database caches.

The pureQuery tooling simplifies the most common tasks, including out-of-the-box support for storing and retrieving beans and maps to and from the database. The SQL is fully customizable, so there are no surprises. And it's extensible which means you can plug in custom-result processing patterns. The extended Java editor includes an integrated SQL editor, giving developers the same level of code-completion, validation, and execution assistance for SQL as they have for Java. It facilitates use of Java and SQL best practices.

A key differentiator for pureQuery programming against DB2 is the ability to program in a single API and to deploy using either static or dynamic execution of SQL. Static SQL against a DB2 database has long been recognized for its ability to improve performance, stability, and security, but has historically required specialized coding to utilize it. Although we have not yet used this capability, we are aware of its advantages, and it's something we will consider for future enhancements.

In an upcoming release, pureQuery will also enable administrators or developers to be able to trace SQL back to the originating application, thus shortening problem determination and assisting impact analysis.

The [Resources](#) section includes many articles in which you can learn more about pureQuery and Data Studio Developer.

IBM pureQuery is not a full-blown persistence layer. It does not provide managed object support. If you need a full-blown persistence layer and/or managed object support, you will likely need to use something like openJPA. An upcoming release of pureQuery is planned to enable many of its benefits for all Java applications, not just those written using the pureQuery APIs.

Persistence mechanisms previously used in WebSphere Customer Center

In WebSphere Customer Center, two persistence mechanisms are used. Because of performance problems using EJB2 queries, we decided a long time ago to use direct JDBC calls for query operations and EJB2 CMP for create, retrieve, update, and delete (CRUD) operations.

The ORM layer in EJB2 CMP entity beans hides the details of JDBC operations from object operations. This layer also helps generate the appropriate SQL statements for retrieving, changing, and persisting a record in a table. In our case, this ORM layer is simple and lightweight because we did not need all its features:

- It only maps a one-to-one relationship between a CMP entity bean and a table.
- It does not map table relationship to CMP entity bean relationship.
- It does not map table inheritance to CMP entity bean inheritance.

We used EJB2 CMP entity bean life-cycle events before inserting and updating a record into a table.

- Before inserting the record, the primary key is set with the `ejbCreate` method.
- Before updating the record, we used optimistic concurrency control with the `set entity object` method.
- The logic to set common entity fields occurs in `ejbCreate` and `set entity object` method before inserting or updating the record.

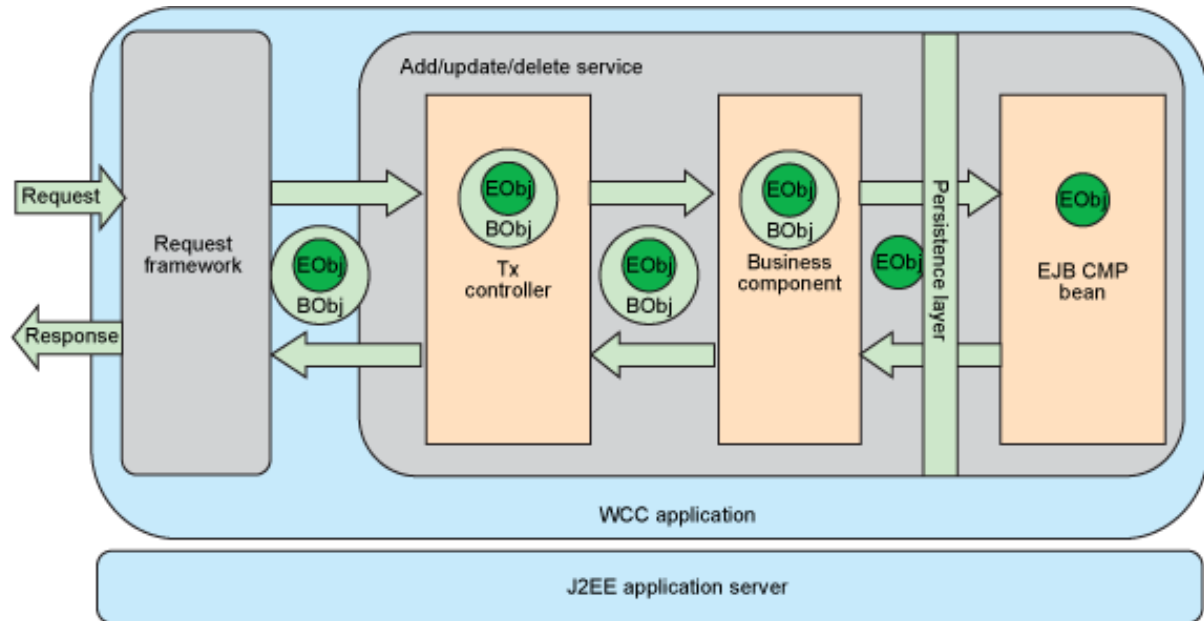
As stated earlier, we used direct JDBC calls for our complex query operations to avoid the performance overhead in EJB2 query. Unfortunately, because of this, we could not reuse the code we were using from EJB2 CMP ORM, so we needed to manually code the object-relational mapping for the JDBC queries.

Architectural view of CMP entity beans in WebSphere Customer Center

Figure 2 shows an architectural view of the persistence mechanism for CRUD

operations using CMP entity beans in WebSphere Customer Center.

Figure 2. CMP entity beans in Add/Update/Delete services



WebSphere Customer Center is a J2EE application that runs inside an application server. It has the following four layers (moving from left to right in the picture):

- The *Request framework* manipulates and transforms request and response messages.
- The *Tx controller* handles any transaction-level logic and may call one or more business components for component-level logic.
- The *Business component* handles any component-level logic and can be used to create different transactions.
- The *Persistence layer (CMP entity beans)* is called by the business component for any CRUD operations.

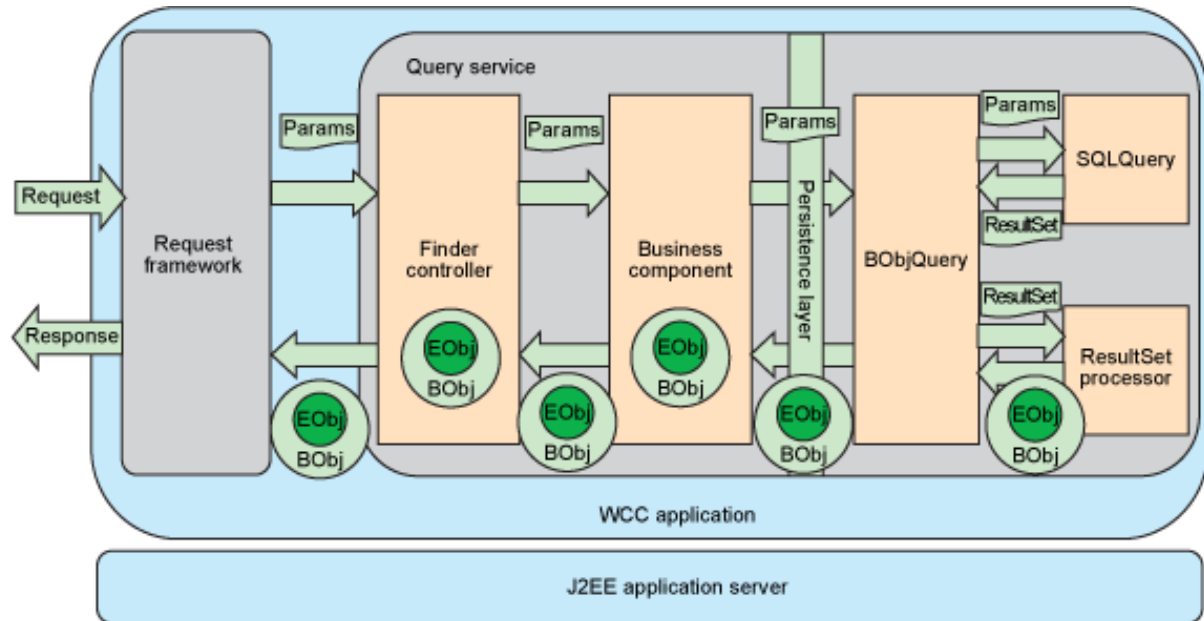
The entity object (EObj) is retrieved from the business object (BObj) in the Business Component layer and is sent to the persistence layer (CMP entity beans) for CRUD operations. Once it is returned from the persistence layer, the entity object (EObj) is wrapped again by the business object. The entity object has a one-to-one mapping to the table and is the transfer object used to pass data between the persistence layer and Business Component layer.

Architectural view of a direct JDBC call in WebSphere Customer Center

Figure 3 is an architectural view of the persistence mechanism for query operations which use direct JDBC calls in WebSphere Customer Center. Code layers are the same as shown in [Figure 2](#), except the persistence layer is supported by some of

our own framework code to do direct JDBC calls.

Figure 3. Direct JDBC calls in query services



- The *Request framework* manipulates and transforms the request and response messages
- The *Finder controller* handles any transaction-level logic and may call one or more business components for component-level logic
- The *Business component* handles any component-level logic and can be used to create different transactions.
- The *Persistence layer (Direct JDBC)* is called by the business component for any query operations.

Query parameters are passed through the different code layers to BObjQuery. BObjQuery class has different implementations for different business components. BObjQuery calls a common SQLQuery class to get a SQL ResultSet. This ResultSet is passed to ResultSetProcessor for creating business objects. Each business component has a specific ResultSetProcessor implementation.

Issues with the WebSphere Customer Center persistence mechanisms

Our product has been using the persistence mechanisms described above for several years and they served us well. We were able to scale the WebSphere Customer Center application to large user, transaction, and data volumes and offered our clients many extension mechanisms. Over time, however, we did encounter several drawbacks we wanted to make you aware of.

EJB2 CMP entity bean issues

Here are the CMP entity bean issues we found:

- CMP only solves part of our ORM needs. Entity beans are not just plain old Java objects (POJOs) and cannot be passed around, so we created our own entity object. This forced us to manually move attribute values to and from the CMP entity bean and our own entity object.
- As stated earlier, the ORM done in CMP is not reusable in JDBC queries. The ORM in CMP generates mapping code between the CMP entity bean and tables. Because you cannot treat a CMP entity bean as a POJO, the mappings are useless in a JDBC query.
- CMP development is still quite complex even with the help of an IDE. For just one CMP entity bean, you need to create six classes: Local Home, Remote Home, Remote Interface, Local Interface, Key Class, and Bean Class. In addition, you need to have EJB entity bean deployment descriptors and a mapping file and database schema. Even a simple invocation of the CMP entity bean involves many steps. For example, for an update operation, you need to get JNDI context, look up EJB home, find the EJB entity bean and then update the bean.
- CMP entity beans are J2EE server dependent. Different servers require a different set of deployment code and deployment descriptors. This all causes extra work in code maintenance and product distribution.

Direct JDBC call issue

When writing JDBC queries, ORM relies on manual coding. This is tedious, error-prone, and it slows down development productivity.

Problems combining JDBC with EJB2 CMP

Accessing the database using both JDBC and EJB2 CMP can cause data synchronization issues when update and query are performed together within one transaction. During a transaction, the update through the EJB2 CMP entity bean isn't sent to the database until the end of the transaction which means the updated data is not visible to the subsequent JDBC query that is part of the same transaction.

Performance issues

Here is a list of performance concerns we had with our existing persistence mechanisms:

- CMP EJB query doesn't perform well in a query scenario, and that is why we used direct JDBC calls for queries.

- CMP entity bean update is not efficient because it requires a read from the database and then an update in the database. This behavior is because the CMP entity beans are managed by the container.
- Container-management of entity beans generates lots of code. This is necessary to isolate the relational layer from the object layer, but this convenience comes at the cost of performance. With our programming model, we deal with the relational layer for complex native SQL and have no need to hide the SQL from the programmer. This basically made the entity management useless for us.

Technology roadmap issues

It was time for us to evaluate our roadmap against the technology we were using. We found some gaps that made it imperative to rethink our persistence mechanism.

- We are planning to use the XML data type that is not a standard JDBC SQL type. Our product can run on DB2 and Oracle, each of which interprets the XML data type differently. This means we will modify the basic CRUD SQL statements for each database. However, the basic CRUD SQL statements generated by CMP entity beans cannot be modified, which would make it impossible for us to adopt the XML data type.
- EJB2/CMP has been deprecated by JEE5 and is being replaced by EJB3/Java Persistence API (JPA). EJB3/JPA is designed to simplify the development and provide more flexibility for complex query scenarios. EJB3/JPA is still based on the entity-management concept.

Expectations for a new persistence mechanism

Given the pains we had and our plans for moving forward, we knew that we needed a new persistence mechanism. Below is a list of the requirements needed for our mechanism:

- Need only simple object to relational mapping capabilities, specifically:
 - Mapping of the Java bean attribute name to the table column name, but not mapping of entity relationship or inheritance
 - Super class mapping support; attribute to column mapping defined in a super class should be available for its child class. However, the super class itself doesn't map to a table. This is to save the mapping effort required for many common attributes and columns.
 - SQL generation for basic entity CRUD operations

- Entity life cycle events callback to handle some of the logic before and after insert, update, and delete
- All generated code should be portable to different application servers.
- The development process to implement the persistence mechanism must be easy to reduce the development costs associated with the upgrade. This includes support for development tools to handle both “bottom up” and “meet-in-the-middle” approaches.
 - The “bottom up” approach assumes you already have database table schema and need to develop an object layer and object relational mapping from the table schema.
 - The “meet-in-the-middle” approach assumes you already have database table schema plus an object layer and need to develop an ORM between the two.
- Extensibility and flexibility, including access to the full power of SQL and the ability to change any generated SQL CRUD statements.
- The persistence mechanism we chose must perform well when compared to our existing persistence mechanism.
- Smooth migration path for our existing code
- Clients can extend our product with additional entities and attributes, and the mechanism is backward-compatible with our clients’ existing code.
- The technology must be available to match our next version product delivery schedule and the risks to adopting it must be minimized.
- The technology needs to have a strong technology roadmap that aligns with our directions and will give us benefits down the road. In addition, the technology must be available for at least 5 years.

Comparing managed persistence with pureQuery

With all the expectations of the new persistence mechanism in mind, we investigated three persistence mechanisms for comparison: 1) our existing persistence mechanism -- EJB2 CMP, 2) EJB3/JPA and 3) pureQuery. EJB2/CMP and EJB3/JPA are managed persistence solutions, whereas, as stated earlier, pureQuery is not. See [Table 1](#) for our assessment of the technologies against our requirements.

Table 1. Persistence mechanism comparison

	EJB 2/CMP (no change)	EJB3/JPA	pureQuery
--	-----------------------	----------	-----------

Simple O/R mapping	Half support	Full support with POJO style	Full support with POJO style
Ease of development / Runs on different application servers	Complex Different generated code for different CMP implementations on different application servers	Simple Different generated code for different JPA implementations on different application servers	Simple SQL assistance integrated in Java editor. Single code generation for different application servers
Flexibility and extensibility	CRUD SQL generated can't be changed	INSERT SQL generated can't be changed	CRUD SQL generated can be changed
Performance	Baseline <ul style="list-style-type: none"> • Query through CMP does not perform well • Read and update for entity update scenario • It manages entities with extra cost 	Better <ul style="list-style-type: none"> • Query through JPA query API performs • Read and update for entity update scenario • It manages entities with extra cost 	Best <ul style="list-style-type: none"> • Query API or method style query performs well • One update for entity update scenario • No entity management, which reduces cost
Migration and backward compatibility	No migration, this is our starting point	Migration required. Should be backward compatible based on JEE specification.	Migration required. Should be backward compatible based on the design.
Technology availability	Available	Available	Available (although we were working with pre-release code)
Technology roadmap	Deprecated	<ul style="list-style-type: none"> • Roadmap and direction in the hands of JCP • Concern that new features may have slower adoption 	<ul style="list-style-type: none"> • Clear roadmap aligns to IBM persistence strategy • Quick to add or adopt new features

Risk	It will be costly later if the application server drops the support. Potential performance problems due to entity management	Potential performance problems due to entity management	Minimal
-------------	---	---	---------

Making the decision and developing an execution plan

It appeared to us that a managed persistence solution was not required for our programming model, and we wanted to unleash the power of SQL without hiding it from the developer. For us, entity management adds a layer of overhead that we do not require and which can negatively affect performance.

With all the SQL development help from Data Studio Developer tools and its focus on data-centric development, it made pureQuery a natural choice for us. To support our persistence mechanism decision, we developed a plan to prove the technology in the following areas:

- Identify pureQuery development and delivery environment
 - Investigate programming model
 - Investigate software platform for product delivery
 - Investigate software platform for product development
 - Investigate development tool for object to relational mapping
 - Investigate optimistic control implementation
- Estimate the amount of work to migrate CMP and direct JDBC call to pureQuery
 - Investigate how to migrate existing WebSphere Customer Center EJB CMP beans
 - Investigate how to migrate existing direct JDBC calls
- Functional Test to address our backward compatibility concern
 - Investigate the co-existence of pureQuery with EJB2/CMP and direct JDBC call.
 - Investigate the support of Global Transaction (XA).
- Benchmark Test to address our performance concerns and ensure the performance of the new mechanism is better than our existing mechanism.

Acknowledgements

We would like to thank the pureQuery development team for their help during our initial investigation and review of this article.

Conclusion

This article gives an overview of InfoSphere Master Data Management Server and Data Studio Developer/pureQuery. It then dives into the persistence mechanism previously used in WebSphere Customer Center. Several issues with the persistence mechanism are listed for improvement. With our expectation set forth on persistence mechanism, we did a comparison on EJB2/CMP, EJB3/JPA and pureQuery. pureQuery stands out as our choice, and we developed an execution plan to further validate our decision.

In Part 2 of this article, we will detail the results of our proof of technology, including developer productivity in migrating entity beans to pureQuery, performance results for both query and CRUD operations, and transactional consistency. We'll also include some lessons learned from our experience that will hopefully help you if you decide to embark on a similar project.

Resources

Learn

- IBM Master Data Management Server [product page](#): See how the IBM InfoSphere Master Data Management Server can benefit your business by managing your critical master data across customer, product and account domains.
- IBM Data Studio [product page](#): Learn how IBM Data Studio Gives you a comprehensive data management solution to design, develop, deploy and manage data-driven applications.
- "[The Easy Way to Quick Data Access](#)" (IBM Database Magazine, Q3 2007): Get a great overview of pureQuery technology and learn how it relates to existing frameworks.
- View these [demos](#) to see the developer tools in action.
- [Data Studio pureQuery tools series](#) (developerWorks): Discover how pureQuery tools make Java programming with SQL more productive than ever before.
- [Introducing pureQuery Annotated Method Style](#) (developerWorks, April 08): Get an introduction to the pureQuery annotated method coding style -- a simple, flexible style falling under the named-query paradigm, capable of executing SQL statically or dynamically.
- [Introducing pureQuery Inline Style](#) (developerWorks, May 08): Explore the benefits as well as some of the key features of using the inline method programming style.
- Browse the [technology bookstore](#) for books on these and other technical topics.

Get products and technologies

- Download [IBM Data Studio development tools](#).

Discuss

- [WebSphere Customer Center Forum](#) : Chat with other developers about the WebSphere Customer Center.
- [Data Studio Forum](#): Collaborate with other Data Studio users.
- Visit the [Data Studio Community Space](#) on developerWorks to participate in a community devoted to Data Studio.

About the authors

Wei Zheng

Wei Zheng is a member of the Master Data Management Server Architecture team. He joined IBM in 2005 from acquisition of DWL and works on various areas of the product. He graduated from Zhejiang University, China with a Master Degree in Electronic Engineering.

Paul van Run

Paul van Run is a lead architect and STSM on the IBM Master Data Management team. He came to IBM through the DWL acquisition on 2005. Paul leads the architecture teams for MDM Server and WPC. He has a Master degree in Mathematics (Artificial Intelligence) from the University of Waterloo (Canada) and a Masters degree from the Technical University of Eindhoven in the Netherlands.

Kathy Zeidenstein

Kathy Zeidenstein has worked at IBM for a bazillion years. She currently works on the IBM Data Studio enablement team focusing on community development. Before taking on this role, she was the product marketing manager for IBM OmniFind Analytics Edition.