Customizing InfoSphere MDM Server using pluggable business object query and pluggable SQL framework

Guidelines and instructions

Prashanta Chandramohan

Overview

IBM® InfoSphere® Master Data Management Server is a highly customizable product that can meet stringent performance and scalability requirements. To meet specific performance requirements of clients, the product offers a pluggable business object query and pluggable SQL framework. These features help in modifying an existing query used by the MDM server, helping to optimize the application. This article focuses on the recommended practices for implementing InfoSphere MDM Server pluggable business object queries and pluggable SQL.

IBM® InfoSphere® Master Data Management Server (MDM Server) is a leading MDM solution from IBM in the market. MDM Server is a physical master repository that delivers a single version of truth about critical data entities like customer, product, account, and more. It is designed to meet high performance and scalability requirements. The product offers flexible customizations to data model and the business services offered to manage master data.

Based on customer requirements, the product can be customized. To serve diverse a customer base with heterogeneous software and hardware environments, customer business requirements, and data characteristics, there are a number of customizations for every implementation. It is important that these implementations follow best practices around every customization so these changes don't affect the MDM Server behavior. This will help the customer in easier maintenance of the system.

Our objective is to show the best way to write pluggable business object queries in MDM Server. A pluggable query framework allows us to modify an existing database query used by an inquiry service. The new query change may be required to make sure the service is optimized and can run faster to meet a specific performance objective. This article takes an example scenario where this customization is required and shows a step-by-step approach. Since MDM is built on the J2EE framework, it follows Java™ standards and is designed with re-usability in mind, and there are certain best practices recommended for better system maintenance. Also, the article discusses inquiry-level enhancements introduced in MDM Server V9.0.x.
Introduction to the MDM Server business objects query

An important customization option in InfoSphere MDM Server is to use the pluggable business object query framework. This feature helps in modifying an existing query used by the MDM service and replaces it with an optimized query. Very often during implementations, we have seen customers use this feature to configure fast-running queries to meet performance objectives.

MDM documentation provides high-level steps to be performed to implement pluggable queries. This article helps identify a requirement when pluggable query framework is required and explains detailed steps in doing this product modification with an example scenario.

Since MDM business object classes are designed keeping re-usability in mind, and classes related to a particular domain are grouped together, a wrong implementation of these classes may result, affecting the behavior of the product and may cause undesired results. The best practices discussed soon will help in easier maintenance of pluggable queries without affecting the core product functionality.

System prerequisites and configuration information

The following components and version of IBM InfoSphere MDM Server are used here:

- InfoSphere MDM Server V9.0.1
- An application server running WebSphere® Application Server V6.1.0.25
- A database server running DB2® V9.5 Fixpack 4

When to use the business object query extension framework

The InfoSphere MDM Server business object query extension framework allows us to modify an existing MDM Server query to achieve better performance. This customization may be required as MDM Server is designed to operate in many client environments to meet different data requirements. If a particular inquiry transaction is not meeting required performance expectations, customers can consider modifying the database query executed by the product.

All product inquiry transactions have pluggable query support except services from the module DWLCommonServices, DWLAdminServices, and Code Table Services.

Sample business object query implementation

We will take an example of business object query implementation suggested in the "Severe degradation of SQL performance with complicated "OR" clauses on DB2 z/OS platforms" Technote.

Introduction to sample implementation

The Technote mentioned above is a modification suggested to InfoSphere MDM Server while using DB2 on z/OS® as the database server. While invoking the getPartyRelationship service, DB2 z/OS exhibits higher I/O and CPU costs due to a table scan on a frequently executed SQL statements. This is because of the usage of a complicated or clause in the default query provided out of the box.
To address this, we can modify this default query with an optimized SQL statement by implementing the pluggable business object query feature of MDM Server. The next section covers detailed steps and best practices to follow.

**Detailed steps to implement business object queries**

In this sample scenario, we need to customize the existing pluggable query since we will only be modifying the query run during the `getPartyRelationship` transaction:

1. Extend the `BObjQuery` class
2. Override the existing query
3. Extend the `BObjQueryFactory` implementation class
4. Register the new factory implementation class

Now that we have identified the required high-level customization tasks, let's go through each step in detail.

**1. Extend the `BObjQuery` class**

To extend the core business object's query implementation, let's create a new class called `CustomPartyRelationshipExtBObjQuery`. This will override `provideSQLStatement()` so it can return a correct query when the `getPartyRelationship` transaction is executed.

**Listing 1. Sample code showing extended `BObjQuery` class**

```java
package com.custom.tcrm.coreParty.bobj.query;

public class CustomPartyRelationshipExtBObjQuery extends PartyRelationshipBObjQuery {
    public CustomPartyRelationshipExtBObjQuery(String queryName, DWLControl control) {
        super(queryName, control);
    }

    public CustomPartyRelationshipExtBObjQuery(String persistenceStrategyName, DWLCommon objectToPersist) {
        super(persistenceStrategyName, objectToPersist);
    }

    @Override
    protected String provideSQLStatement() throws BObjQueryException {
        return super.provideSQLStatement();
    }
}
```

**2. Override the existing query**

Next, we define a string parameter to hold the new SQL statement. Define a new string variable with the following SQL statement.

**Listing 2. Modified SQL statement declared in `BObjQuery` class**

```java
private static final String CUSTOM_PARTY_RELATIONSHIPS_ACTIVE_QUERY =
    "SELECT CONTACTREL.CONT_REL_ID, CONTACTREL.REL_TP_CD,
     CONTACTREL.REL_DESC, CONTACTREL.START_DT, CONTACTREL.END_DT,
     CONTACTREL.TO_CONT_ID, CONTACTREL.FROM_CONT_ID,
     CONTACTREL.LAST_UPDATE_DT, CONTACTREL.LAST_UPDATE_USER,
    ";
```
Also, modify the `provideSQLStatement()` function in the `CustomPartyRelationshipExtBObjQuery` class to return the new SQL statement, as shown in Listing 3.

We have to be careful selecting the query name to compare to. To find out the correct name of the query, you can visit MDM API documentation and look for appropriate `BObjQuery` class. In our sample scenario, we will look into the `PartyRelationshipBObjQuery` class and choose `PARTY_RELATIONSHIP_ACTIVE_QUERY`.

Listing 3. Return the modified SQL statement from `provideSQLStatement` function

```java
@Override
protected String provideSQLStatement() throws BObjQueryException {
  if (queryName.equals(PARTY_RELATIONSHIPS_ACTIVE_QUERY)) {
    return CUSTOM_PARTY_RELATIONSHIPS_ACTIVE_QUERY;
  }
  else {
    return super.provideSQLStatement();
  }
}
```

3. Extend the `BObjQueryFactory` implementation class

For this sample scenario, we need to extend the party query factory class to pick up the extended `BObjQuery` class. Since we are customizing party relationship query, this class will be `PartyModuleBObjQueryFactoryImpl`. This has methods to create `BObjQuery` instances for each of the business objects under `PartyModule`.

For our scenario, we can define the class as shown below.
Listing 4. Extend the PartyModuleBObjQueryFactoryImpl class

```java
public class CustomPartyModuleBObjQueryFactoryImpl extends PartyModuleBObjQueryFactoryImpl {
    public CustomPartyModuleBObjQueryFactoryImpl() {
        super();
    }

    public BObjQuery createPartyRelationshipBObjQuery(String queryName, DWLControl dwlControl) {
        if ((queryName == null) || (queryName.trim().equals("")))
            throw new IllegalArgumentException("Query Name cannot be empty or null.");
        return new CustomPartyRelationshipExtBObjQuery(queryName, dwlControl);
    }
}
```

4. Register the new factory implementation class

Registering the new BObjQueryFactory class is important. This configuration allows us to let MDM Server know which class to execute so an instance of CustomPartyRelationshipExtBObjQuery is created, instead of the out-of-the-box BObjQuery class.

To perform this configuration, modify the TCRM.properties file to change the Party.BObjQueryFactory configuration. Provide the newly created class name as shown below, instead of the default factory class.

Listing 5. Update the TCRM.properties file so the BObjQueryFactory implementation is changed to new class

```properties
Party.BObjQueryFactory=
com.hnb.tcrm.coreParty.bobj.query.CustomPartyModuleBObjQueryFactoryImpl
```

Once these steps are performed, publish your code changes to the application server where MDM is running. A restart of the server is needed to make sure the property file change we performed takes effect.

You can now test the service by running a `getPartyRelationship` service.

Inquiry-level enhancements: Pluggable SQL

InfoSphere MDM Server provides an option to use inquiry levels in search and inquiry transactions. These inquiry levels let users choose the level of details for objects being returned from MDM. For example, an inquiry level ranging from 0 to 4 can be provided when a `getPerson` service is run. This inquiry level controls a different type of additional detail information returned for the person party. The higher the inquiry level the more information we get.

Along with using default inquiry levels provided out of the box, customers have an option to configure custom inquiry levels to fetch new object combinations. Party and contract domains support this, and below are some of the services that support selective retrieval of child objects.
For more details on inquiry levels supported for these services, please refer to MDM Transaction Reference Guide.

1. getParty
2. getPerson
3. getOrganization
4. getContract
5. getProductInstance

The pluggable SQL, also referred to as Optimized Transparent SQL (OTS), introduced in MDM Server V9.0.x, helps in reducing the response time of coarse-grained inquiry transactions, thus achieving improved performance. Figure 1 illustrates the usage of the pluggable SQL feature.

**Figure 1. Using pluggable SQL feature**

Taking the getPerson inquiry service as an example, when this service is executed, there will be more than one database call made to retrieve each child object of PersonBObj. If you are using a higher inquiry level, there will be more database interactions, as this transaction has to fetch AddressBObj, PersonNameBObj, ContactMethodBObj, PartyIdentificationBObj, PartyPrivPrefBObj, etc.

With the OTS feature turned on, only one call to the database is made, as there will only be one SQL to execute.

Customers can use pluggable SQL statements in two ways:

- To enhance existing coarse-grained inquiry services with SQLs more suitable for the environment
- To create optimized pluggable SQL statements for custom inquiry levels

**Turning OTS on**

To turn on the pluggable SQL feature, set the flag optimized.sql to true in tcrm_extension.properties, as shown in Listing 6. This is a global setting and applies to all domains and inquiry levels.

**Listing 6. Turning OTS on in tcrm_extension.properties**

```bash
optimized.sql=true
```
To enhance the existing inquiry services, customers can activate entries in the INQLVLQUERY table corresponding to the domain to which the service belongs. For example, `getPerson` belongs to `group_name = 'Person'` in the INQLVL table. To enhance this transaction when run with inquiry level 4, you will be running below SQL statement.

**Listing 7. Tuning existing course-grained inquiry service**

```sql
update INQLVLQUERY
set END_DT = null
where INQLVL_ID
in (select INQLVL_ID
 from INQLVL
 where INQ_LVL = 4
 and GROUP_NAME = 'Person')
```

Alternatively, customers can add new custom inquiry levels using `addInqLevelQuery` and `updateInqLevelQuery` admin services:

- Create a custom inquiry level with inquiry level type 500.
- Create a pluggable SQL for new inquiry level 500 using the `addInqLevelQuery` service.
- Tune the SQL statement generated and use the `updateInqLevelQuery` admin service to persist the optimized SQL statement.

Pluggable SQL allows enhancements to coarse-grained inquiry services along with retaining existing functionality of MDM Server. They can help in avoiding multiple database calls and enhance the inquiry services with one single optimized SQL call. With OTS in place, database administrators can view, analyze, and tune this SQL statement based on the data model used in MDM Server implementation, helping reduce the response time and increase throughput.

**Summary of best practices**

InfoSphere MDM Server is physical master data repository that can manage multiple data domains and can be implemented to meet a wide set of business requirements. Accordingly, each MDM implementation is different from others and need careful observation in the way the MDM data model is used and services are configured to achieve the client's performance requirements.

This article discussed pluggable business object query implementation and the pluggable SQL framework used by MDM Server. The example discussed will give you good hands-on knowledge of this feature and cover some best practices to follow.

Additionally, the recommendations below will help you use this feature in tandem with MDM design. These tips will help you use the customizations efficiently without harming core MDM functionalities:

- We often introduce new inquiry levels to support specific inquiry requests coming to MDM Server. It's recommended we use custom inquiry levels starting from 100, keeping upgradability of the product in mind.
- Take the help of the database administrator in carefully verifying, analyzing, and tuning the SQL statement you plan to use during OTS. This feature is provided to enhance the
performance of the inquiry services. A wrong SQL will result in degraded performance of the product.

- During pluggable business object query implementation, identifying the core business object query implementation is key. The MDM Developers Guide provides good documentation in identifying the right classes to modify. Please refer to the section "Pluggable Business Object Queries" for more details.
- The out-of-the-box query names follow standard naming conventions as described in the MDM Developers Guide. For example, `PARTY_RELATIONSHIPS_ACTIVE_QUERY` is the name of the query used by the `getPartyRelationship` service to fetch all active relationships between two parties. When you are overriding this to provide your own custom query, you can name it `CUSTOM_PARTY_RELATIONSHIPS_ACTIVE_QUERY` as shown in our sample scenario.
- MDM API documentation provides useful information about out-of-the-box `BObjQuery` classes and a list of query parameters used by the class. Since you will be overriding one of these parameters, it's important to get familiar with API documentation. Also, refer to the MDM Developers Guide for standard naming conventions used for naming the query parameters.

**Conclusion**

IBM InfoSphere Master Data Management Server helps companies gain control over their master data entities by enabling them to manage and maintain a complete and accurate view of master data. InfoSphere MDM Server enables companies to extract maximum value from master data by centralizing multiple data domains and providing a comprehensive set of pre-built business services that support a full range of master data management functionality.

Highly flexible in nature, MDM Server offers a wide range of customization capabilities that help you develop services that can meet customers' demands. Due to the highly customizable nature, you need to understand the impact that each tier has on overall functionality of the product. This article contains practical information to help you understand InfoSphere MDM Server and the ways you can modify it to meet customers' performance goals and business needs. It provides valuable information about customizing MDM using the pluggable business object query and pluggable SQL framework, which can be used in real customer implementations.

**Acknowledgments**

I would like to thank Henk Albals, InfoSphere MDM, product manager, for his feedback and contributions in creating this article. And I would like to extend my sincere thanks to numerous developerWorks authors who have written similar narratives on InfoSphere MDM Server best practices. I have used this content as reference during creation of this article and to effectively implement InfoSphere MDM Server for several customers.