Improve performance of product search in InfoSphere MDM Collaborative Edition

XML search

Maobing Jin (maobing@us.ibm.com)
Lead Architect, IBM MDM Industry Solutions and SaaS
IBM

This tutorial explains how to leverage the XML support in InfoSphere® MDM Collaborative Edition to improve the search performance for a customer solution. We will compare the difference between the various search options in the product and demonstrate how to index XML data of MDM entries to optimize the search query to search MDM entries using XQuery. Sample application source code is included.

Overview

Search performance is a critical measurement for a successful MDM solution. In addition to the best practices for data model and query design, leveraging XML support from the product is another option for improving search performance. InfoSphere MDM Collaborative Edition represents the data entry in XML format and stores the XML documents for the entries in DB2 or Oracle tables. In this tutorial, we will elaborate through a simple MDM solution and demonstrate how to leverage the XML support to improve the performance on product information search in your MDM solutions.

MDM industry solution demo — Retail

InfoSphere MDM Collaborative Edition is designed for product information management (PIM) for retail, finance/banking, and other industries. Check out the demo.

Prerequisites

This tutorial is intended for MDM solution developers. Before proceeding, you will need:

- MDM Collaboration Server (InfoSphere MDM Collaborative Edition) 10.0.0.1 or later
- A sample company abc created on the instance
- The sample solution data model in the attachment has been imported into the company

We will refer to the data in the sample data model and illustrate through the sample solution for search and XML technology in the product.
Introduction

In this tutorial, we will illustrate through a simple MDM solution with the following data model.

**Figure 1. Simple solution data model**

The catalog primary spec: Product Spec includes the following attributes.

**Table 1. Catalog primary spec details: Product Spec**

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Type</th>
<th>Indexed</th>
<th>Internal node ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product spec (spec name)</td>
<td>SPEC</td>
<td></td>
<td>5839 (spec ID)</td>
</tr>
<tr>
<td>ID (primary key)</td>
<td>STRING</td>
<td>Yes</td>
<td>6572</td>
</tr>
<tr>
<td>Name</td>
<td>STRING</td>
<td>No</td>
<td>6575</td>
</tr>
<tr>
<td>Description</td>
<td>STRING</td>
<td>No</td>
<td>6574</td>
</tr>
<tr>
<td>Long description</td>
<td>RICH_TEXT</td>
<td>No</td>
<td>6573</td>
</tr>
<tr>
<td>List price</td>
<td>CURRENCY</td>
<td>No</td>
<td>6576</td>
</tr>
<tr>
<td>Manufactured</td>
<td>GROUPING</td>
<td>No</td>
<td>6578</td>
</tr>
<tr>
<td>Manufactured part number</td>
<td>STRING</td>
<td>No</td>
<td>6570</td>
</tr>
<tr>
<td>Manufacturer name</td>
<td>STRING</td>
<td>No</td>
<td>6569</td>
</tr>
<tr>
<td>Manufactured production date</td>
<td>DATE</td>
<td>No</td>
<td>6571</td>
</tr>
<tr>
<td>Manufactured expiration date</td>
<td>DATE</td>
<td>No</td>
<td>6568</td>
</tr>
<tr>
<td>Thumbnail</td>
<td>THUMBNAIL_IMAGE</td>
<td>No</td>
<td>6577</td>
</tr>
</tbody>
</table>

**Note:** Spec node ID is generated by the system, so it will vary in different system.

In the spec, attribute ID as primary key must be indexed; all other attributes remain as non-indexed. The internal XML document will use internal IDs for spec and spec node as part of the XML element names.

**Understanding the MDM search**

To understand the search feature in an MDM Collaboration Server, let's take a simple example from the UI-based search scenario:
1. Log in to the instance.
2. Click the **Search** tab on top of the left navigation pane to switch to the search pane.
3. Click **New Search** to open the blank search template on the right pane.
4. Select **Product Catalog** in the **Search In Catalog** drop-down box on the right pane.
5. Add category **Women Dresses** for **Search within Categories**.
6. Add attributes **ID** and **Name**, and fill in the search value for each attribute.
7. Select the search options.

As a result, you created an item search on the right pane, as shown below.

**Figure 2. Screenshot for an MDM search UI**

---

**WebSphere Query Language**

WebSphere Query Language (WQL) is an object query language designed for retrieving product business object information from the MDM system. Most of the search-related features are implemented via an WQL query. For more information about WQL, please refer to the reference topic on IBM Support Knowledge Center.

Two patents have been issued for the WQL search engine technology:

- Method and system for data retrieval using a product information search engine
- Logical Object Search Framework and Application Programming Interface
The item search consists of:

- Category restrictions — Search within one or more categories from one or more hierarchies that the item is mapped to.
- Attributes — Search the items for the given values of the attributes.
- Search scope — Either in the entire item catalog or a subset (selection).
- Search result sorting — An indexed attribute for sorting search results in ascending or descending order.
- Save the result as a selection — Static or dynamic.

Click **Search** at top of the right pane to conduct the search.

Behind the scenes, the user inputs for the search will be translated into an MDM WQL query, which will then be converted to SQL query and executed against the database through the JDBC connection.

**Listing 1. Sample WQL query for item search**

```wql
-- Sample WQL query for item search: select items in the given catalog matching the given criteria.
-- @author Maobing Jin
-- @since 5.3.0

select item
  from catalog('Product Catalog')
  where (item.category.hierarchy.name = 'Product Category Hierarchy'
    and (item.category.path = path('Product Category Hierarchy/Apparel/Women/Women Dresses', '/')
      or item.category.path like path('Product Category Hierarchy/Apparel/Women/Women Dresses/%', '/'))
  )
  and (item['Product Spec/ID']  = '12345'
    and item['Product Spec/Name']  = 'xyz'
  )
order by  item['Product Spec/ID'] asc
```

**Listing 2. Generated SQL statement for item search WQL query**

```sql
-- Generated SQL statement for Item Search WQL Query.

select distinct ita1.ita_item_id as c1, ita1.ita_value_string as c2
  from cfp cfp1, ctr ctr1, icm icm1, ita ita1, itx itx1
  where ita1.ita_catalog_id = 8206
    and ita1.ita_version_id <= 999999999
    and ita1.ita_next_version_id >= 999999999
    and ((ctr1.ctr_name = 'Product Category Hierarchy'
      and (cfp1.cfp_full_path = 'Product Category Hierarchy/Apparel/Women/Women Dresses'
        or cfp1.cfp_full_path like 'Product Category Hierarchy/Apparel/Women/Women Dresses/\%'))
    )
    and (ita1.ita_value_string = '12345'
      and ita1.ita_node_id = 6572
      and icm1.icm_item_id = ita1.ita_item_id
      and icm1.icm_version_id <= 999999999
      and icm1.icm_next_version_id >= 999999999
      and ctr1.ctr_id = icm1.icm_cat_tree_id
      and cfp1.cfp_full_path_type = 'PATH_FULL_PATH'
      and cfp1.cfp_category_id = icm1.icm_category_id)
```

Improve performance of product search in InfoSphere MDM
Collaborative Edition
and cfpl.cfp_version_id <= 999999999
and cfpl.cfp_next_version_id >= 999999999
and itx1.itx_entry_id = ita1.ita_item_id
and itx1.itx_version_id <= 999999999
and itx1.itx_next_version_id >= 999999999

order by c2 asc

WQL query will be processed in the search engine in MDM Collaboration Server. The item attribute search criteria will be translated into the spec-driven attribute search predicates in WQL, and the search engine will convert indexed and non-indexed attribute WQL predicates into different types of SQL statement, as shown in examples below.

Table 2. Spec attribute search: Indexed vs. Non-indexed

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>WQL predicate</th>
<th>SQL predicate</th>
<th>Search type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product spec/ID</td>
<td>item['Product Spec/ID'] = '12345'</td>
<td>ita1.ita_value_string = '12345'</td>
<td>Relational Search</td>
</tr>
<tr>
<td>Product spec/name</td>
<td>item['Product Spec/Name'] = 'xyz'</td>
<td>XMLEXISTS ($xml/entry/s5839[n6575=&quot;xyz&quot;] passing itx1.itx_entry_content as &quot;xml&quot;)</td>
<td>XML Search</td>
</tr>
</tbody>
</table>

Where, the relational search for attribute ID will query the database table column directly; while the XML search for attribute Name will query the XML document stored in the database table.

When a non-indexed attribute is included in a search, as above, the system will warn users with the following message: At least one non-indexed attribute is specified in the search criteria. This can cause the search to take longer than usual.

Performance limitations on indexed attribute search

For a simple search with one or two indexed attributes, the response time is reasonable. However, the more indexed attributes used in the query, the more time required. The performance could be poor if the query includes multiple indexed attributes and the ITA table volume is large.

It will result in poor performance to search the attribute in the XML document that has not been indexed. Typically, the default XML search can be used for non time-critical searches, such as reports running in the background. It is not recommended to use the default XML search for time-critical searches.

It is recommended to use relational search for high performance queries. The indexed attribute search will use ITA table with database indices on related columns.

However, relational search has its own limitation on the number of the selected attributes. As an alternative, you can enhance the XML search. In the following sections, we will explain how to work around the default behavior of XML search in the out-of-box configuration and leverage XML technology in DB2 or Oracle to improve the search performance in your solution.

MDM entry XML representation

To understand XML search for the non-indexed attributes, we must understand the XML representation of MDM entry, including item and category.
From Version 10.0.0.1 (Fix Pack 1), InfoSphere MDM Collaborative Edition started to adopt the XML format for representing MDM entries and store the XML documents in database tables. With the data model in the above section, take the item WAC007_0701 as example.

**Figure 3. Item WAC007_0701**

When saving the item, the item XML content will be persisted in table TCTG_ITX_ITEM_CONTENT (ITX), as shown below.

**Figure 4. ITX table**

This table is designed for item entry where ITX_ENTRY_CONTENT is an XML column. Similarly, there is another table TCTG_CAX_CATEGORY_CONTENT with XML column CAX_ENTRY_CONTENT is designed for category entry.

**Figure 5. CAX table**
The XML document for MDM entry has the following format.

**Figure 6. MDM entry XML format**

```
<entry>
  <!-- one and only one primary spec -->
  <!-- zero or more category specs -->
  <!-- zero or more location data -->
  <locationdata hierarchyid="(location hierarchy id)"
  <!-- zero or more location specs -->
</locationdata>
</entry>
```

Entry XML document consists of three top-level of elements under the root element `entry`:

- Element for primary spec
- Element(s) for secondary spec(s)
- Element(s) for location data which may has zero or more location spec(s)

The XML elements for primary spec or secondary specs will follow the tree structure of the spec definition, as shown in the XML document below. (Also see Listing 3.)

**Listing 3. XML document for item /tdWAC007_0701/td**

```
<entry>
  <s5839>
    <n6572>WAC007_0701</n6572>
    <n6575>Luigi Valenti Woollen Fedora Hat</n6575>
    <n6574>Blue woollen fedora hat</n6574>
    <n6573>
      Suitable for both men and women, this denim-colored fedora hat is made of light wool
      and presents a professional style.
      Donning an iconic style, this fedora has an indented crown with a matching side feather.
      It can make a fashion statement in the winter.
    </n6573>
    <n6576>19.99</n6576>
    <n6578/>
    <n6577>wac007_0701.jpg</n6577>
  </s5839>
</entry>
```

The XML document keeps the structure of spec, but format the XML element name using internal ID.

**Figure 7. XML element naming rule**

```
's' <spec id> - spec root element name
'n' <node id> - node element name
```

Spec name and spec node names are not good for XML element names as they may include invalid characters.

**Note:** The internal IDs for spec and spec nodes are generated per the local instance of MDM Collaboration Server. Different instances may result in different internal IDs for each spec node. However, you can use the following SQL statement to resolve a spec node name to its internal ID.
Listing 4. SQL query for resolving the node name to its internal ID

```sql
-- =====================================================================================================
-- SQL query for resolving the node name to its internal ID.
--
-- @param: company code
-- @param: spec name
-- @param: spec node name
-- @param:
--
-- @author Maobing Jin
-- @since 5.3.0
--
-- Notes: Developers can run the SQL query on Web UI:
-- 1. Log in to the instance as Admin user.
-- 2. Open menu 'System Administrator | DB Admin'.
-- 3. Copy/Paste the SQL query to the 'SQL Command' field in the right pane.
-- 4. Click the 'Run Query' button on the top of the right Pane.
-- 5. Check the results in the 'Results' field.
--
-- =====================================================================================================

select unique nod_id
from cmp, spc, nod
where cmp_company_id = spc_company_id
and cmp_company_id = nod_company_id
and spc_id = nod_spec_id
and cmp_company_code = 'abc'
and spc_name = 'Product Spec' -- spec name
and nod_name = 'Name'         -- node name
```

Indexing MDM entry XML documents

To leverage XML support for MDM entry search (item and category rich search, for example), you must model the spec attributes as not Indexed.

Figure 8. Screenshot for non-indexed attribute details

XML data, XML indices, and XQuery

For more information about XML supports in DB2, refer to the following books or tutorials:

- "DB2 pureXML Cookbook" offers details about XML indexing in DB2.
- "Query DB2 XML Data with SQL" covers the basic concepts of querying XML data in DB2.
Leave the checkbox **Indexed** unchecked for the **Name** attribute.

The search for non-indexed spec attributes will be converted to XML query for the XML documents in ITX and CAX tables. The default settings for XML document in the database do not include any index on the XML documents.

To improve the XML search performance, you can create XML indices on the attribute used in each search query. For example, to search the content in the **Long Description** attribute, you need to index the following XPath `/entry/s5839/n6573`, using the following script.

### Listing 5. SQL query for creating an XML index on a spec attribute

```sql
-- SQL query for creating XML index on a spec attribute.
-- @author Maobing Jin
-- @since 10.0.0.1
-- Notes: Developers can run the SQL query on Web UI:
-- 1. Log in to the instance as Admin user.
-- 2. Open menu 'System Administrator | DB Admin'.
-- 3. Copy/Paste the SQL query to the 'SQL Command' field in the right pane.
-- 4. Click the 'Run Query' button on the top of the right Pane.
-- 5. Check the results in the 'Results' field.

CREATE INDEX ICTG_ITX_ENTRY_CONTENT_6573
ON ITX(ITX_ENTRY_CONTENT) GENERATE KEY USING XMLPATTERN '/entry/s5839/n6573' AS SQL VARCHAR(3000);
```

This XPath includes internal IDs, which will vary for different MDM Collaboration Server instances. You can use the following script to format your XPath.

### Listing 6. SQL script for formatting XPath

```sql
-- SQL Script for formatting XPath.
-- @param: company code
-- @param: spec name
-- @param: spec node name
-- @author Maobing Jin
-- @since 10.0.0
-- Notes: Developers can run the SQL query on Web UI:
-- 1. Log in to the instance as Admin user.
-- 2. Open menu 'System Administrator | DB Admin'.
-- 3. Copy/Paste the SQL query to the 'SQL Command' field in the right pane.
-- 4. Click the 'Run Query' button on the top of the right Pane.
-- 5. Check the results in the 'Results' field.

select unique concat (concat('/entry/s', nod_spec_id), concat('/n', nod_id))
from nod
```
where nod_name = 'Name'
    and nod_spec_id = (select spc_id from spc where spc_name = 'Product Spec'
    and spc_company_id = (select cmp_company_id from cmp where cmp_company_code = 'abc'));

Listing 7. SQL query for creating an XML index on a spec attribute using spec and node names

```
-- SQL query for creating XML index on a spec attribute using spec and node names.
--
-- @param: spec name
-- @param: spec node name
--
-- @author Maobing Jin
-- @since 10.0.0.1
--
-- Notes: Developers can run the SQL query on Web UI:
-- 1. Log in to the instance as Admin user.
-- 2. Open menu 'System Administrator | DB Admin'.
-- 3. Copy/Paste the SQL query to the 'SQL Command' field in the right pane.
-- 4. Click the 'Run Query' button on the top of the right Pane.
-- 5. Check the results in the 'Results' field.
--
-- select 'CREATE INDEX ICTG_ITX_ENTRY_CONTENT_' || CHAR(NOD_ID) ||
        '    ON ITX(ITX_ENTRY_CONTENT) GENERATE KEY ' ||
        ' USING XMLPATTERN ''/entry/s'|| TRIM(CHAR(NOD_SPEC_ID)) || '/n' || TRIM(CHAR(NOD_ID)) || ''' AS SQL
        VARCHAR(3000); ' ||
    from nod
    where nod_name = 'Description'
    and nod_spec_id = (select spc_id from spc where spc_name = 'Product Spec'
    and spc_company_id = (select cmp_company_id from cmp where cmp_company_code = 'abc'))
    order by nod_name;
```

For more information about XML indexing, refer to "Exploit XML indexes for XML query performance."

In the sample solution data model, all attributes are now marked as non-indexed attributes, so XML search is enabled through the non-indexed attribute. Now let's create XML indices for the following attributes using the script in Listing 7 in the section Indexing MDM entry XML documents.

Table 3. Creating XML indices for non-indexed attributes

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Attribute path (including spec)</th>
<th>XPath expression</th>
<th>Index type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Product spec/name</td>
<td>/entry/s5839/n6575</td>
<td>AS SQL VARCHAR(3000)</td>
</tr>
<tr>
<td>Description</td>
<td>Product spec/description</td>
<td>/entry/s5839/n6574</td>
<td>AS SQL VARCHAR(3000)</td>
</tr>
<tr>
<td>Long description</td>
<td>Product Spec/Long Description</td>
<td>/entry/s5839/n6574</td>
<td>AS SQL VARCHAR(3000)</td>
</tr>
<tr>
<td>List price</td>
<td>Product spec/list price</td>
<td>/entry/s5839/n6576</td>
<td>AS SQL DOUBLE</td>
</tr>
<tr>
<td>Part number</td>
<td>Product spec/manufactured/part number</td>
<td>/entry/s5839/n6578/n6570</td>
<td>AS SQL VARCHAR(3000)</td>
</tr>
<tr>
<td>Manufacturer name</td>
<td>Product spec/manufactured/manufacturer name</td>
<td>/entry/s5839/n6578/n6569</td>
<td>AS SQL VARCHAR(3000)</td>
</tr>
<tr>
<td>Production date</td>
<td>Product spec/manufactured/production date</td>
<td>/entry/s5839/n6578/n6571</td>
<td>AS SQL DATE</td>
</tr>
</tbody>
</table>
Using search in the MDM Collaborative Edition

Search is a key component in the product. Many business features are implemented on top of search. For a customer solution, you can use the search features in MDM Collaborative Edition in many business scenarios.

Search scenario No. 1: Search by multiple attributes through rich search UI

User scenario: User wants to run a pre-defined search from the rich search UI screen.

Steps: Same as shown in Understanding the MDM search.

You can define a new item search from the rich search UI, as shown below, including four more attributes in addition to primary key.

Figure 9. XML search for multiple Non-indexed attributes
For this search scenario, the corresponding WQL and SQL queries are listed in Listing 8 and Listing 9.

**Listing 8. WQL query for the specified search**

```sql
-- WQL query for the specified search.
--
-- @author Maobing Jin
-- @since 10.0.0.1
--
-- Notes: Developers can run the WQL query on commandline:
-- 1. Go to $TOP directory.
-- 2. Type command: $JAVA_RT com.ibm.ccd.search.clp.CmdMain
-- 3. Copy/Paste the WQL query as command input in one line.
-- 4. Check the results in the command output.
--
select item
  from catalog('Product Catalog')
  where (upper(item['Product Spec/ID']) like 'WAC'
      AND upper(item['Product Spec/Description']) like '%WOOLLEN'
      AND item['Product Spec/List Price'] < 1000
      AND item['Product Spec/Manufactured/Expiration Date'] > date('Sat Dec 31 10:18:59 PST 2016', 'EEE MMM dd HH:mm:ss zzz yyyy')
      AND upper(item['Product Spec/Thumbnail']) like '%WAC%')
order by item['Product Spec/ID'] asc
```

**Listing 9. SQL query for the specified search**

```sql
-- SQL query for the specified search.
--
-- @author Maobing Jin
-- @since 10.0.0.1
--
-- Notes: Developers can run the SQL query on Web UI:
-- 1. Log in to the instance as Admin user.
-- 2. Open menu 'System Administrator | DB Admin'.
-- 3. Copy/Paste the SQL query to the 'SQL Command' field in the right pane.
-- 4. Click the 'Run Query' button on the top of the right Pane.
-- 5. Check the results in the 'Results' field.
--
select distinct ita1.ita_item_id as c1, ita1.ita_value_string_ignorecase as c2
  from ita ita1, itx itx1
  where ita1.ita_catalog_id = 8206
    and ita1.ita_version_id <= 999999999
    and ita1.ita_next_version_id >= 999999999
    and ita1.ita_value_string_ignorecase like 'WAC'
    and XML EXISTS ('$xml/entry/s5839[fn:contains(fn:upper-case(n6574),"WOOLLEN")
        and (n6576 castable as xs:double and n6576<1000.0)
        and n6578[(n6568 castable as xs:double and n6568>1483208339000)]
        and fn:contains(fn:upper-case(n6577),"WAC")]' passing itx1.itx_entry_content as "xml")
    and ita1.ita_node_id = 6572
    and itx1.itx_entry_id = ita1.ita_item_id
    and itx1.itx_version_id <= 999999999
    and itx1.itx_next_version_id >= 999999999
order by c2 asc
```

**Search scenario No. 2: Run a search query in a script**

**User scenario:** Solution developers want to test a search before they can use it in the solution implementation.
Steps:

1. Log in to the instance as Admin.
2. Open Menu > Data Model Manager > Scripting > Script Sandbox.
3. Copy and paste the script to the Script Pane field in the right pane.
4. Click Run Script on the top of the right pane.
5. Check the results in the Output field.

You can embed a search query in a script and run it in the Script Sandbox; see Listing 10.

**Figure 10. Screenshot for multiple non-indexed attribute search in Script Sandbox**

```
BEGIN=======---------------------------------------------------------------
 Mon Jan 12 13:06:03:927 PST 2015: parsing query ... 
 Mon Jan 12 13:06:04:002 PST 2015: executing query ... 
 Mon Jan 12 13:06:04:052 PST 2015: counting the resultset ... 
 Mon Jan 12 13:06:04:130 PST 2015: reporting ... size = 1
END=======---------------------------------------------------------------
```

**Listing 10. Script for XML search**

```
// =====================================================================================================
// Script for XML Search.
// #author: Macheng Jin
// #since 5.3.0
// Notes: Developers can run the script on Web UI:
// 1. Log in to the instance as Admin user.
// 2. Open menu 'Data Model Manager | Scripting | Script Sandbox'.
// 3. Copy/Paste the script to the 'Script Pane' field in the right pane.
// 4. Click the 'Run Script' button on the top of the right Pane.
// 5. Check the results in the 'Output' field.
```

// Use out.write(...) to output script objects or variables
Search scenario No. 3: Run a search query in a Java extension

User scenario: Solution developers want to implement a report using Java™ code.

Steps:

1. Log in to the instance.
2. Open Menu > Product Manager > Reports Console.
3. Click New and follow the wizard to create a new report (Sample Product Report, for example).

In Sample Product Report, the report script (see Figure 11) is calling a report extension point implemented in Java technology.
Figure 11. Screenshot of an XML search for multiple non-indexed attributes in report script

You can embed a search query in the Java code and run it as part of report job execution, see the Java code in Listing 11.

Listing 11. Sample Java code for a report

```java
package com.ibm.mdm.samples;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import com.ibm.pim.attribute.AttributeDefinition;
import com.ibm.pim.catalog.item.Item;
import com.ibm.pim.common.exceptions.PIMSearchException;
import com.ibm.pim.context.Context;
import com.ibm.pim.context.PIMContextFactory;
import com.ibm.pim.search.SearchQuery;
import com.ibm.pim.spec.Spec;
import com.ibm.pim.utils.Logger;
```
{
    private static Logger logger =
        PIMContextFactory.getCurrentContext().getLogger(SampleProductReport.class.getName());

    @Override
    public void reportGenerate(ReportGenerateFunctionArguments inArgs)
    {
        // receive input parameters
        HashMap<String, String> inputs = (HashMap<String, String>)inArgs.getCustomParameter("inputs");
        String idBeginsWith = inputs.get("IDBeginsWith");
        String descriptionContains = inputs.get("DescriptionContains");
        String priceUpLimit = inputs.get("PriceUpLimit");
        String expirationAfter = inputs.get("ExpirationAfter");
        String thumbnailContains = inputs.get("ThumbnailContains");

        // verify date
        if  (expirationAfter == null || expirationAfter.isEmpty())
        {
            logger.logDebug("Default to current date/time.");

            expirationAfter = formatDateFunction(new Date(), "yyyy-MM-dd-hh.mm.ss");
        }

        // initial context
        Context currentContext = PIMContextFactory.getCurrentContext();

        // searching ...
        logger.logDebug("BEGIN=======================================");
        String wqlQuery = ""
        + "select item "
        + "  from catalog('Product Catalog') "
        + "  where (upper(item['Product Spec/ID']) like '%" + idBeginsWith.toUpperCase() + '%'
        + "  AND upper(item['Product Spec/Description']) like '%" + descriptionContains.toUpperCase() + '%'
        + "  AND item['Product Spec/List Price'] < " + priceUpLimit + ""
        + "  AND item['Product Spec/Manufactured/Expiration Date'] > " + "date('" + expirationAfter + '",
        'yyyy-MM-dd-hh.mm.ss') "
        + "  AND upper(item['Product Spec/Thumbnail']) like '%" + thumbnailContains.toUpperCase() + '%')
        + "order by item['Product Spec/ID'] asc "
        + ""
        + new Date().toString() + ": parsing query ... ");
        SearchQuery searchQuery = currentContext.createSearchQuery(wqlQuery);
        logger.logDebug("" + new Date().toString() + ": executing query ... ");
        SearchResultSet resultSet = searchQuery.execute();
        logger.logDebug("" + new Date().toString() + ": counting the resultset ... ");
        int size = resultSet.size();
        logger.logDebug("" + new Date().toString() + ": reporting ... start");
        logger.logDebug("\tSize = " + size);
        logger.logDebug("==========================================");
        while (resultSet.next())
        {
            try {
                // get next record in the result set
                Item item = resultSet.getItem(1);

                // retrieving item details
                StringBuffer buf = new StringBuffer();
                Spec spec = currentContext.getSpecManager().getSpec("Product Spec");
                List<AttributeDefinition> attributeDefinitions = spec.getAttributeDefinitions();
                for (AttributeDefinition attributeDefinition:attributeDefinitions)
                {
                    if (attributeDefinition.isLeaf())
                    {
                        String nameValuePair = formatAttributeNameValuePair(item, attributeDefinition.getPath());
                        nameValuePair = escapeSpecialChars(nameValuePair);
Measuring the search performance

For a quick testing, let's compare the performance between XML search and relational SQL search:

1. Set up a blank instance. Run $TOP/bin/db/create_schema.sh to ensure a blank instance.
2. Create a blank company abc. Run $TOP/bin/db/create_cmp.sh --code=abc to create a blank company on the instance.
3. Import the attached data model. Open Menu > System Administrator > Import Environment to import the attached ZIP file.
4. Edit Product Spec to ensure that the attributes Description, List Price, Expiration Date, Thumbnail in Product Spec are marked Indexed.
5. Create test data in the instance. Run the script in Listing 12 below to populate 12,500 items.

Listing 12. Script for creating performance testing data

```java
//
// Script for creating performance testing data.
//
// @author Maobing Jin
```
// @since 5.3.0
//
// Notes: Developers can run the script on Web UI:
// 1. Log in to the instance as Admin user.
// 2. Open menu 'Data Model Manager | Scripting | Script Sandbox'.
// 3. Copy/Paste the script to the 'Script Pane' field in the right pane.
// 4. Click the 'Run Script' button on the top of the right Pane.
// 5. Check the results in the 'Output' field.
//
var e;
catchError(e)
{
    var ATTR_PATH_PRIMARY_KEY = "Product Spec/ID";

    var CATALOG_NAME = "Product Catalog";
    var catalog = getCTGByName(CATALOG_NAME);
    var hierarchy = catalog.getPrimaryCategoryTree();

    var spec = getSpecByName("Product Spec");
    var nodes = spec.getSpecNodes();

    var attrStrVals = [];
    attrStrVals[0] = "test0";
    attrStrVals[1] = "test1";
    attrStrVals[2] = "test2";
    attrStrVals[3] = "test3";
    attrStrVals[4] = "test4";
    attrStrVals[5] = "test5";
    attrStrVals[6] = "test6";
    attrStrVals[7] = "test7";
    attrStrVals[8] = "test8";
    attrStrVals[9] = "test9";

    var attrNumVals = [];
    attrNumVals[0] = 12;
    attrNumVals[1] = 25;
    attrNumVals[2] = 7;
    attrNumVals[3] = 96;
    attrNumVals[4] = 74;
    attrNumVals[5] = 11;
    attrNumVals[6] = 134;
    attrNumVals[7] = 67;
    attrNumVals[8] = 37;
    attrNumVals[9] = 8;

    var startNum = 0;
    var endNum = 12500;
    for (i = startNum; i < endNum; i++)
    {
        var itemKey = "TEST_" + i;

        // create item
        var item = new CtgItem(catalog, false, false, false);

        // set PK
        item.setCtgItemAttrib(ATTR_PATH_PRIMARY_KEY, itemKey);

        // fill each attribute
        var k = 0;
        forEachHmElement(nodes, key, node)
        {
            nodeType = node.getNodeAttributeValue("TYPE");
            nodePath = node.getNodePath();
6. Test 1: Run the script in Listing 10 to perform search on attributes ID, Description, List Price, Expiration Date, Thumbnail using ITA table. Record the search elapsed time.

7. Edit Product Spec to uncheck the Indexed mark for the attributes Description, List Price, Expiration Date, Thumbnail in Product Spec to ensure that those attributes are non-indexed.

8. Test 2: Run the script in Listing 10 to perform search on attributes ID, Description, List Price, Expiration Date, Thumbnail, using XML document in ITX table. Record the search elapsed time.

9. Run the SQL query in Listing 7 to create XML index on an attribute. Set nod_name = the attribute name (Description, List Price, Thumbnail). For grouping attribute Manufactured/Expiration Date, you need to modify the query and insert the grouping node ID in the path. Use the script in Listing 4 to find the node ID.

10. Test 3: Run the script in Listing 10 to perform a search on attributes ID, Description, List Price, Expiration Date, Thumbnail, using XML document in ITX table with XML indices. Record the search elapsed time.

The table below lists the performance data collected in a local test.
Table 4. Performance comparison for multiple attribute search

<table>
<thead>
<tr>
<th>Test number</th>
<th>Search type</th>
<th>Search attributes</th>
<th>Total number of entries</th>
<th>Query execution time (ms)</th>
<th>Size of resultset (entries)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Attribute search using relational table</td>
<td>• ID[STRING, Indexed, PK] • Description[STRING, Indexed] • List Price[CURRENCY, Indexed] • Expiration Date[DATE, Indexed] • Thumbnail[THUMBNAIL_URL, Indexed]</td>
<td>12517</td>
<td>655</td>
<td>1</td>
<td>Marking the attributes as Indexed will trigger the search using ITA table.</td>
</tr>
<tr>
<td>2</td>
<td>Attribute search using XML documents</td>
<td>• ID[STRING, Indexed, PK] • Description[STRING] • List Price[CURRENCY] • Expiration Date[DATE] • Thumbnail[THUMBNAIL_URL]</td>
<td>12517</td>
<td>905</td>
<td>1</td>
<td>Not marking the attributes as Indexed will trigger the search using XML document in ITX table.</td>
</tr>
<tr>
<td>3</td>
<td>Attribute search using XML documents with DB indices</td>
<td>• ID[STRING, Indexed, PK] • Description[STRING] • List Price[CURRENCY] • Expiration Date[DATE] • Thumbnail[THUMBNAIL_URL]</td>
<td>12517</td>
<td>203</td>
<td>1</td>
<td>Not marking the attributes as Indexed will trigger the search using XML document in ITX table. Creating XML indices for all attributes will improve the XQuery performance.</td>
</tr>
</tbody>
</table>

Note: The data shown above may vary from system to system. For formal performance benchmarking testing, you can use those professional tools such as Rational® Performance Tester or JMeter.

With XML indices, the XML search performance improved significantly. When the data volume increases, the relational search will take longer, but the XML search will have constant performance.

Summary

XML search is a useful feature in the product. Because there is no direct tool on the Web-based UI application and the command line, it has not been well known to many developers. This tutorial has illustrated a quick way to enable the feature and improve search performance. More importantly, this XML approach overcomes the performance limitation in the indexed attribute search using relational table.

MDM scripts, WQL/SQL scripts, and Java extension code used in this tutorial are included in the attached Eclipse project for MDM solution. You can open the project in Eclipse or other Java...
IDE tools and test the sample solution against the attached data model, which is packaged in the attached ZIP file.
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample data model</td>
<td>SampleMDMModel.zip</td>
<td>20KB</td>
</tr>
<tr>
<td>Sample Eclipse project for MDM solution</td>
<td>SampleMDMProject.zip</td>
<td>50KB</td>
</tr>
</tbody>
</table>
Resources

- "pureXML in DB2 9: Which way to query your XML data?" has more examples and details for XML query in DB2.
- The DB2 area on developerWorks provides resources for architects, developers, and engineers.
- Stay current with developer technical events and webcasts focused on a variety of IBM products and IT industry topics.
- Follow developerWorks on Twitter.
- Watch developerWorks demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.
- Get involved in the developerWorks Community. Connect with other developerWorks users while you explore developer-driven blogs, forums, groups, and wikis.
About the author

Maobing Jin

Maobing Jin is a senior software architect in the InfoSphere MDM division at IBM, responsible for overall product architecture of MDM Collaborative Edition. He has been working on MDM product development since 2004, and designed and implemented the core components in MDM product. Recently, he has been leading the MDM industry solution development and the SaaS offering.