Use DB2 optimization guidelines with your pureXML applications

Influencing the access plan

Priya Baliga
Anjali Betawadkar-Norwood

December 23, 2009

Optimization guidelines are a powerful tool that enable you to influence key execution plan decisions, including table access methods, index selection, join methods, and join orders. In the IBM® DB2® Version 9.7 for Linux®, UNIX®, and Windows® release, the optimization guidelines infrastructure has been enhanced to support new XML-specific guidelines and to expand the scope of existing relational guidelines to XML operators. This article introduces DB2 for Linux, UNIX, and Windows pureXML® users to the guidelines infrastructure, and guides you through the setup and use of optimization guidelines for your SQL/XML and XQuery workloads.

Introduction

The DB2 optimizer is a highly sophisticated cost-based optimizer. However, optimization decisions are a complex function of various attributes, including the database environment and query characteristics, as well as the data itself. Due to the interplay of these diverse independent factors, in rare instances, the optimizer might select less than optimal execution plans. Since it is not always possible to change the database setup or the inherent characteristics of the data, a tool is needed that can enable users to influence plan choices to compensate for their unique setup characteristics. Optimization guidelines are a powerful mechanism that enable users to influence vital execution plan attributes, including table access methods, index selection, join methods, and join orders in order to overcome any performance issues that may arise. With the most recent DB2 V9.7 for Linux, UNIX, and Windows (DB2 9.7) release, the optimization guidelines infrastructure has been enhanced to introduce new guidelines specific to XML and to expand the scope of existing guidelines to include XML operators.

This article is designed to introduce DB2 pureXML users to the guidelines infrastructure, and to guide them through all the steps required to set up, use, and maintain optimization guidelines for their SQL/XML and XQuery workloads. Various troubleshooting tips, as well as resources, are included to equip users with all the tools and information required for a successful implementation using guidelines.
Database setup used in this article

Transaction Processing over XML (TPoX) is an application-level XML database benchmark based on a financial application scenario, specifically security trading. It uses a real-world XML schema to model the data pertaining to security trading transactions. It is flexible in that it provides various knobs and tuning parameters to modify the data distributions, workload composition, and so on. This article uses a slightly modified form of the TPoX benchmark to show various options available in the XML guidelines infrastructure to influence optimizer plans. For simplicity, the sample queries in this article ignore the namespaces that are present in the TPoX XML data. (For more information about TPoX, see Resources.)

The article uses modified TPoX tables created using the DDL shown in Listing 1:

Listing 1. DDL used to create TPoX tables

```sql
create table security (security_id int, security_type varchar(10), SDOC XML);
create table custacc (CDOC XML);
create table order (security_id int, ODOC XML);
```

The following relational indexes have been created on the tables:

```sql
create index sec_type on security?security_type);
```

The following XML indexes have been created on the tables:

Listing 2. XML indexes created on tables

```sql
create index sec_industry on security(sdoc) generate key using
  xmlpattern '/Security/SecurityInformation/StockInformation/Industry' as SQL varchar(30);
create index sec_symbol on security(sdoc) generate key using
  xmlpattern '/Security/Symbol' as SQL varchar(30);
create index sec_name on security(sdoc) generate key using
  xmlpattern '/Security/Name' as SQL varchar(100);
create index acc_currency on security(sdoc) generate key using
  xmlpattern '/Customer/Accounts/Account/Currency' as SQL varchar(3);
```

Overview of the profiles infrastructure

An optimization profile is an XML document that contains optimization guidelines. There are two categories of optimization guidelines:

- Global guidelines that specify access plan parameters that are to be considered for all the statements while this optimization profile is in effect
- Statement-level guidelines that specify plan attributes that apply to certain statements only

Each statement-level guideline contains:

- A statement key that identifies the statement whose access plan must be influenced
- An action that specifies the desired access plan
The section "Usage scenarios for XML guidelines" includes further details on the anatomy of optimization profiles.

Optimization profiles allow you to influence plans without any application or database configuration changes. The separation of applications and profiles allows both to be developed independent of each other. Users are only required to compose the XML profile document, insert it into the database, and instruct the optimizer to use that profile. Once the profile is enabled, the optimizer automatically matches optimization guidelines to the appropriate statement. The guidelines specified in a profile do not need to include an exhaustive specification of all operators in the plan. You can define guidelines that influence only a few attributes of an execution plan, such as index specification for a certain table in the query or join order for a pair of tables. The optimizer implements the parts of the plan specified in the profile according to the specified guidelines and fills in the rest of the plan intelligently by making cost-based decisions among competing plans.

How to set up and use an optimization profile, step by step

You must follow these steps to use optimization profiles:

1. Create the XML profile file.
2. Learn to reference tables from the query in the optimization guideline.
3. Create the OPT_PROFILE table.
4. Load or insert the profiles in the OPT_PROFILE table.
5. Enable a profile.
6. Issue queries.
7. Verify that the profile has been applied and the desired access plan has been achieved.

The following sections examine each of these steps.

Creating the XML profile file

Optimization profiles are written as an XML file that must conform to the "Current Optimization Profile Schema" (COPS) that is published in the DB2 LUW Information Center for each release (see Resources). The optimization profile file contains the set of all SQL queries whose access plans need to be influenced.

Listing 3. Sample optimization profile

```xml
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
  <!--
  Global optimization guidelines section.
  Optional but at most one.
  -->
  <OPTGUIDELINES>
    <REOPT VALUE="ALWAYS"/>
  </OPTGUIDELINES>
  <!--
  Statement profile section.
  Zero or more.
  -->
  <STMTPROFILE ID="Sample Optimization Profile">
    <STMTKEY SCHEMA="TPOX">
      <![CDATA[SELECT *]]>
    </STMTKEY>
  </STMTPROFILE>
</OPTPROFILE>
```
Each XML profile begins with an OPTPROFILE tag that contains metadata, such as an attribute indicating the version of the COPS being used.

Listing 3 shows an example of a single guideline in the global optimization guideline section. The guideline in this example specifies that the REOPT bind option value should be set to ALWAYS. The REOPT bind option affects the optimization of statements that contain parameter markers or host variables. Setting REOPT to ALWAYS causes the statement to be re-compiled for each execution. There can be zero or, at the most, one Global Optimization Section in a valid profile.

The statement profile section is optional. When specified, you can include one or more statement profiles using STMTPROFILE tags, and include guidelines that apply to specific SQL queries. Each statement profile consists of

- A STMTKEY element that identifies the statement to which the guideline must apply
- An OPTGUIDELINES element that describes the guideline itself

The statement defined in the STMTKEY element must exactly match the statement whose access plan must be influenced, although differences in whitespaces are tolerated. There is no support for any wild cards in the STMTKEY to affect groups of statements. Each statement that must be influenced requires a separate STMTPROFILE section. If more than one STMTPROFILE section matches an executing statement, the guideline that occurs first is selected and applied. Listing 3 contains a single optimization guideline that specifies that a XANDOR (XML Index Anding and Oring) plan be selected for the query in the STMTKEY.

Since the statement in the STMTKEY element contains the special XML characters < and >, the query text needs to be enclosed in a CDATA section starting with <![CDATA[ and ending with ]]> as shown in Listing 3 above.

Learn to reference tables from the query in the optimization guideline

The guidelines section in an optimization profile enables the user to specify an action for a specific table, such as the access method for that table, or to indicate that the table participates as the outer or the inner leg of a join, and so on. It is important to correctly identify the table that you want to affect. You can identify a table by its name or by the quantifier number in the optimized SQL statement shown in the execution plan produced by the db2exfmt tool. The use of table names is simpler, but using table identifiers is a more powerful mechanism because it can refer to a derived quantifier, such as an 'xmlexists' predicate.

- Identifying a table by name

Imagine the following simple query:
SELECT * FROM security WHERE XMLEXISTS('__$SDOC/Security/SecurityInformation/StockInformation[Industry= "OfficeSupplies"]')
The exposed table name in this example is 'SECURITY'. A guideline for this query should specify this exposed name. For example, an XML index access using index 'SEC_INDUSTRY' can be specified like this:

```xml
<XISCAN TABLE='SECURITY' INDEX='SEC_INDUSTRY'/>
```

Now consider the following query, which uses the alias 'sec' for the table name:

```
SELECT * FROM security sec WHERE XMLEXISTS('__$SDOC/Security/SecurityInformation/StockInformation[Industry= "OfficeSupplies"]')
```

The exposed table name is 'SEC', and the index guideline needs to be changed to refer to the exposed name, as shown below:

```xml
<XISCAN TABLE='SEC' INDEX='SEC_INDUSTRY'/>
```

### Identifying a table by TABID

When a query is submitted to DB2, it is rewritten in a form that is most suitable for optimization. The rewritten query known as "Optimized SQL" is visible when you explain the statement with the `db2exfmt` command. The optimized SQL is located in the section 'Optimized Statement' of the db2exfmt output. For example, for the query in Listing 4:

#### Listing 4. Query

```
SELECT *
FROM order, security
WHERE order.security_id = security.security_id
AND XMLEXISTS('__$SDOC/Security/Symbol[.="IBM"]');
```

The optimized statement may look like:

#### Listing 5. Optimized statement

```
FROM $INTERNAL_FOR$ ((TABLE ($INTERNAL_XPATH$(
  '($INTERNAL_XMLTOXML_NIEO$(Q2.SDOC))[$INTERNAL_EBV_BOOLEAN$(Security/Symbol[(. = "IBM")])](-->$C0:))')) AS Q1), TPOX.SECURITY AS Q2, TPOX.ORDER AS Q3
WHERE (Q3.SECURITY_ID = Q2.SECURITY_ID)
```

The internal number for the SECURITY table is Q2; for ORDER, it is Q3. The `XMLEXISTS` predicate in the original query got converted into a complex expression that can be referred to using the number Q1. A guideline that forces the application of the `XMLEXISTS` predicate to the SECURITY table first, followed by a hash join with the ORDER table, is shown in Listing 6:
Listing 6. Referencing tables by name and by identifier

```xml
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
  <STMTPROFILE ID="Identifying the exposed name of a table">
    <STMTKEY SCHEMA="TPOX">
      SELECT *
      FROM order, security
      WHERE order.security_id = security.security_id
      AND XMLEXISTS('$SDOC/Security/Symbol[.="IBM"]')
    </STMTKEY>
    <OPTGUIDELINES>
      <HSJOIN>
        <NLJOIN FIRST='TRUE'>
          <ACCESS TABLE='SECURITY'/> 
          <ACCESS TABID='Q1'/> 
        </NLJOIN>
        <ACCESS TABID='Q3' /> 
      </HSJOIN>
    </OPTGUIDELINES>
  </STMTPROFILE>
</OPTPROFILE>
```

Listing 6 shows that the guidelines may contain a mixture of table names and table identifiers. Table names are specified with the XML attribute TABLE, while table identifiers are specified with the XML attribute TABID.

Creating the OPT_PROFILE table

Optimization profiles must be stored in the SYSTOOLS.OPT_PROFILE table.

There are two ways to create this table:

- DB2 provides a system-defined stored procedure called SYSINSTALLOBJECTS that can create or drop the OPT_PROFILE table:

  ```sql
  db2 "call sysinstallobjects('opt_profiles', 'c', '', '')"
  ```

  The SYSINSTALLOBJECTS procedure creates or drops database objects that are required for various tools. The first argument specifies that the user is interested in creating the optimization profile table. The second argument specifies that the intended user action is to create the optimization profile table. The third and fourth arguments that specify the schema name and object name must be left blank so that the default schema SYSTOOLS and table name OPT_PROFILE are used. The SYSINSTALLOBJECTS procedure, like all system procedures, resides in the SYSPROC schema.

- Alternatively, you can explicitly issue the following DDL statement to create the OPT_PROFILE table:

  ```sql
  Listing 7. DDL statement to create OPT_PROFILE table
  ```

  ```sql
  create table systools.opt_profile ( 
    schema varchar(128) not null, 
    name varchar(128) not null, 
    profile blob (2m) not null, 
    primary key (schema, name) 
  )
  ```
The *schema* column is used to specify the schema name for the optimization profile. The schema name must be an alphanumeric string and must abide to all the naming rules for schema-names for the corresponding DB2 LUW release.

The *name* column contains the name of the optimization profile and may contain an alphanumeric string of up to 128 characters.

The *profile* column holds the XML document containing the optimization profile.

### Loading the profile

Before enabling an optimization profile, you must associate it with a unique schema-qualified name and store it in the SYSTOOLS.OPT_PROFILE table. You can use an INSERT statement, the IMPORT utility, or the LOAD utility. Listing 8 illustrates how to insert an optimization profile into the OPT_PROFILE table. The profile contains a single XISCAN guideline for the table SECURITY.

#### Listing 8. Inserting an optimization profile

```sql
insert into systools.opt_profile values
('TPOX','PROFILE1',
 CAST('<!--xml version="1.0" encoding="UTF-8"-->
  <OPTPROFILE VERSION="9.7.00">
  <STMTPROFILE ID="Listing 3">
    <STMTKEY><![CDATA[SELECT * FROM security
      WHERE XMLEXISTS("$SDOC/Security/SecurityInformation/
      StockInformation[Industry="OfficeSupplies"]")]]></CDATA><![CDATA[>]]>
    </STMTKEY>
    <OPTGUIDELINES>
      <XISCAN TABLE="SECURITY"/>
    </OPTGUIDELINES>
  </STMTPROFILE>
  </OPTPROFILE>' as blob));
```

Alternatively, users may also use the IMPORT utility to import optimization profiles that have been saved into XML files.

The following example shows how to import two profiles from the files weekly_report_profile.xml and scientist_queries_profile.xml into the SYSTOOLS.OPT_PROFILE table. Assume that the two profiles files are in the current directory. A delimited format input file, say ‘profiledata’ for example (see Listing 9), contains the profile schema, profile name, and the XML file name. Note that each profile must appear on a separate line.

#### Listing 9. The input file profiledata

```
"TPOX","PROFILE1","weekly_report_profile.xml"
"TPOX","PROFILE2","scientist_queries_profile.xml"
```

Now, you can use the IMPORT command to import the profiles into the OPT_PROFILE table.

#### Listing 10. Import profiles into the OPT_PROFILE table

```sql
import from profiledata of del
modified by lobsinfile
insert into systools.opt_profile
```

Use DB2 optimization guidelines with your pureXML applications
Enabling the profile

Any number of profiles may be loaded into the OPT_PROFILE table. However, at most, one can be effective at any given time. If you need to affect multiple queries, include multiple statement profiles in a single optimization profile. Then use the OPTPROFILE bind option at the package level, or the CURRENT OPTIMIZATION PROFILE special register at the session level, to enable the optimization profile. CLI applications can use the client configuration option CURRENTOPTIMIZATIONPROFILE to set the special register per connection.

For example, a user may issue the following command:

```
db2 "set current optimization profile='TPOX.PROFILE1'"
```

This enables PROFILE1 in schema TPOX as the effective profile until the end of the current session. At any point, you can modify the value of the special register with the set command.

You can use the following command to tell the optimizer to not use any optimization profile. The optimizer will return the best plan using a cost-based decision.

```
db2 "set current optimization profile=''"
```

The order of precedence for the various options is as follows:

- The OPTPROFILE bind option applies to all static statements, regardless of any other settings.
- For dynamic statements, the value of the CURRENT OPTIMIZATION PROFILE special register is determined by the following order:
  - The last `SET CURRENT OPTIMIZATION PROFILE` statement within the application [highest precedence]
  - The CURRENTOPTIMIZATIONPROFILE client configuration option, if set
  - The OPTPROFILE bind option, if specified

Issuing queries

Once the required profile has been enabled, users may run their workloads without any changes. For queries that match the active profile, the optimizer makes the best possible effort to generate the plan requested by the guideline. For queries that do not match any guideline, the regular cost-based access plan is chosen.

Verifying that the guideline has been applied

The DB2 Explain facility is a useful tool in determining whether a profile has been successfully matched to a query or not. The applicable optimization profile name and statement profile name are indicated in the "Profile Information" section of the explain output.

Listing 11. Profile information

```
OPT_PROF: (Optimization Profile Name)
        TPOX.PROFILE1
STMTPROF: (Statement Profile Name)
```

Listing 3
The "Profile Information" section of the db2exfmt output can be used to verify that the correct optimization profile and statement profile have been applied to the current query.

If the optimizer was not able to apply the guideline, it returns warning 437 with reason code 13:

```
SQL0437W Performance of this complex query may be sub-optimal. Reason code: "13". SQLSTATE=01602
```

You can then explain the query and check the db2exfmt output for more diagnostic information in order to rectify any problems with the guideline. For example, db2exfmt produces the following output if the referenced table name was not found and the guideline was therefore not applied:

**Listing 12. Extended diagnostic information**

```
Diagnostic Identifier:  1
Diagnostic Details:  EXP0009W  Invalid access request. The table reference identified by the TABLE attribute could not be found. Line number "1", character number "60".
```

When no warning message or diagnostic message is issued, it indicates that the optimizer has applied the guideline. You can check the db2exfmt output to see that the requested access/join methods are being used.

**Usage scenarios for XML guidelines**

**Influencing access method / Coercing XML index access**

**XISCAN (XML Index Scan) guideline**

This guideline specifies that a single XML index scan be used to access the given table. You can include the following XISCAN element in an optimization guideline to specify that the table 'SECURITY' should be accessed using index 'SEC_INDUSTRY':

```
<XISCAN TABLE='SECURITY' INDEX='SEC_INDUSTRY' FIRST="TRUE"/>
```

- The TABLE or TABID attribute specifies the table name or tabid (table identifier, as shown in the optimized SQL of the db2exfmt output that is being accessed. This attribute is mandatory).
- The INDEX attribute is optional and may be used to specify an index name. If specified, it requests an access plan using an XISCAN operator over this particular XML index. When the index name is not specified, optimizer picks the cheapest single XML index access plan.
- The FIRST attribute is also optional. If specified, it can only take the value 'TRUE' to indicate that the given table must be the first table accessed in a join query. For all tables listed in a FROM clause, there can be, at most, one access or join request with the 'FIRST' attribute.

The example in **Listing 13** shows an XISCAN guideline in the context of a complete optimization profile. The example in **Listing 14** uses the XISCAN guideline without an index name to indicate that the table 'SECURITY' should be accessed using the cheapest available XML index.
Listing 13. An XISCAN guideline to use cheapest XML index

```xml
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
  <STMTPROFILE ID="Example 1">
    <STMTKEY SCHEMA="TPOX">
      SELECT *
      FROM security
      WHERE XMLEXISTS(''$SDOC/Security/SecurityInformation/StockInformation[Industry= "OfficeSupplies"]'')
    </STMTKEY>
    <OPTGUIDELINES>
      <XISCAN TABLE='SECURITY' INDEX='SEC INDUSTRY'/>
    </OPTGUIDELINES>
  </STMTPROFILE>
</OPTPROFILE>
```

Listing 14. An XISCAN guideline to use a specific XML index

```xml
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
  <STMTPROFILE ID="Example 2">
    <STMTKEY SCHEMA="TPOX">
      SELECT *
      FROM security
      WHERE XMLEXISTS(''$SDOC/Security/SecurityInformation/StockInformation[Industry= "OfficeSupplies"]'')
    </STMTKEY>
    <OPTGUIDELINES>
      <XISCAN TABLE='SECURITY'/>
    </OPTGUIDELINES>
  </STMTPROFILE>
</OPTPROFILE>
```

**XANDOR (XML Index Anding and Oring) guideline**

This guideline specifies that all applicable XML indexes be used together by a XANDOR operator to access the given table. The `TABLE` or `TABID` attribute and the `FIRST` attribute have the same meaning, as is described in the XISCAN guideline.

The example guideline in **Listing 15** specifies that the table 'SECURITY' should be accessed using XANDOR of all applicable XML indexes. If there is an XML index for the element 'Industry' and an XML index for the element 'Symbol', DB2 will generate a two-way XANDOR with an XISCAN operator for each of the two indexes. Relational indexes will not be used since a relational index cannot be used in XANDOR method. If only one XML index is available, then DB2 cannot apply this guideline and will instead produce the cheapest alternative plan.
Listing 15. A XANDOR guideline to use all applicable XML indexes

```xml
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Example 3">
    <STMTKEY SCHEMA="TPOX">
        SELECT *
        FROM security
        WHERE security_type = 'Bond Fund'
            AND XMLEXISTS('SSDOC/Security/SecurityInformation
                            /StockInformation[Industry="Software"]')
            AND XMLEXISTS('SSDOC/Security/Symbol[.="IBM"]')
    </STMTKEY>
    <OPTGUIDELINES>
        <XANDOR TABLE='SECURITY' />
    </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

ACCESS guideline

The ACCESS guideline (any access request) allows the optimizer to choose the access method. This guideline is typically used when the join order, not the access method, is of primary concern. The ACCESS element must be used when the target table reference is a derived table.

The ACCESS element can have the following attributes: TABLE, TABID, FIRST, TYPE, INDEX, and ALLINDEXES. The ACCESS element can also have one or multiple child elements named INDEX.

The TABLE or TABID attribute and the FIRST attribute have the same meaning as for the XISCAN guideline discussed previously.

The TYPE attribute is optional and can only take the value 'XMLINDEX'. It indicates that the table must be accessed using one of the XML index access methods such as XISCAN, XANDOR, or IXAND with at least one XML index leg, or RIDSCN (index ORing) with at least one XML index leg. When this attribute is not specified, the optimizer makes a cost-based decision to select an access plan for the given table.

The INDEX attribute is optional and may specify an index name only if the TYPE attribute is specified as 'XMLINDEX'. If this attribute is specified, the optimizer may select one of the following:

- An XISCAN plan using the specified XML index.
- A XANDOR plan with all applicable XML indexes.
- An IXAND plan such that the specified XML index is the leading index of the IXAND. The optimizer may add more indexes to the IXAND plan in a cost-based fashion.
- A cost-based XML index ORing plan. The db2exfmt output for an index ORing plan shows a RIDSCN operator with two or more XISCAN operators as input.

Optional INDEX elements can provide two or more XML index names if the TYPE attribute of the ACCESS element specifies 'XMLINDEX'. If an INDEX element is specified, the optimizer may select one of the following:

- A XANDOR plan with all applicable XML indexes.
• An IXAND plan such that the specified XML indexes are used in the specified order but no additional indexes are included
• A cost-based index ORing (RIDSCN) plan

When the index attribute and one or more index elements are specified, the index attribute is ignored.

The **ALLINDEXES** attribute is optional and can only take the value 'TRUE'. This attribute may be specified only if the **TYPE** is specified as 'XMLINDEX'. Setting the attribute to 'TRUE' indicates that all applicable relational and XML indexes must be used to access the table, irrespective of cost. The optimizer then chooses one of the following plans:

• A XANDOR plan with all applicable XML indexes appearing under the XANDOR operator
• An IXAND plan with all applicable relational and XML indexes appearing under the IXAND operator
• An index ORing plan
• An XISCAN plan if only a single index is defined on the table and it is of type XML

The example guideline in **Listing 16** specifies that some XML index be used to access the table SECURITY. The optimizer may pick an XISCAN, IXAND, XANDOR, or RIDSCN plan using cost-based analysis.

**Listing 16. An ACCESS guideline to use some XML index access plan**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
  <STMTPROFILE ID="Example 4">
    <STMTKEY SCHEMA="TPOX">
      SELECT *
      FROM security
      WHERE XMLEXISTS('SDOC/Security/SecurityInformation/StockInformation[Industry= "OfficeSupplies"]')
    </STMTKEY>
    <OPTGUIDELINES>
    </OPTGUIDELINES>
  </STMTPROFILE>
</OPTPROFILE>
```

The example guideline in **Listing 17** specifies that all applicable relational and XML indexes on the SECURITY table be used. The choice of the access method is left to the optimizer. If there are two XML indexes SEC_INDUSTRY and SEC_SYMBOL for the two predicates in the XMLEXISTS, the optimizer has the choice to either use a XANDOR or an IXAND plan. The optimizer chooses between the two access methods using a cost-based analysis.

**Listing 17. An ACCESS guideline to use all indexes on the SECURITY table**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
  <STMTPROFILE ID="Example 4">
    <STMTKEY SCHEMA="TPOX">
      SELECT *
      FROM security
      WHERE XMLEXISTS('SDOC/Security/SecurityInformation/StockInformation[Industry= "OfficeSupplies"]')
    </STMTKEY>
    <OPTGUIDELINES>
    </OPTGUIDELINES>
  </STMTPROFILE>
</OPTPROFILE>
```
Listing 17. An ACCESS guideline to use all applicable XML indexes

```xml
<?xml version="1.0" encoding="UTF-8"?><OPTPROFILE VERSION="9.7.0.0"><STMTPROFILE ID="Example 5"><STMTKEY SCHEMA="TPOX">SELECT * FROM security WHERE XMLEXISTS('$$DOC/Security/SecurityInformation/StockInformation[Industry= "Software"]') AND XMLEXISTS('$$DOC/Security/Symbol[.="IBM"]')</STMTKEY><OPTGUIDELINES><ACCESS TABLE='SECURITY' TYPE='XMLINDEX' ALLINDEXES='TRUE'/></OPTGUIDELINES></STMTPROFILE></OPTPROFILE>
```

**IXAND (Index Anding) guideline**

The IXAND access request element directs the optimizer to use index anding to access a table. Much like the ACCESS guideline, the IXAND element can have the attributes TABLE, TABID, FIRST, TYPE, INDEX, and ALLINDEXES. The IXAND element can also have one or more child elements named INDEX. The following rules apply to the IXAND guideline:

- The attribute **TYPE** can have the value 'XMLINDEX' to select an index anding access plan that includes one or more XML indexes.
- The **INDEX** attribute is optional and allows you to specify an index name that will be used as the leading index in an 'IXAND' plan. The optimizer will add more indexes to the IXAND based on cost.
- The **INDEX** elements can optionally specify two or more indexes that will be used in the 'IXAND' plan in the specified order and no additional indexes will be added by the optimizer to the IXAND.
- The **INDEX** attribute or the **INDEX** element may specify XML indexes only if the **TYPE** is specified as 'XMLINDEX'.
- The **ALLINDEXES** attribute can be set to 'TRUE' to force the usage of all applicable relational and XML indexes, irrespective of cost.
- When the **INDEX** attribute and one or more **INDEX** elements are both specified, the **INDEX** attribute is ignored.

In the example in Listing 18, the IXAND element in the optimization guideline specifies that the table SECURITY should be accessed using two XML indexes, SEC_INDUSTRY and SEC_SYMBOL. The optimizer generates an IXAND plan with the two indexes as legs of the IXAND plan in the specified order.
Listing 18. An IXAND guideline to use specified multiple XML indexes

```xml
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Example 6">
<STMTKEY SCHEMA="TPOX">
SELECT *
FROM security
WHERE security_type = 'Bond Fund'
 AND XMLEXISTS('$/SDOC/Security/Symbol[.="IBM"]')
</STMTKEY>
<OPTGUIDELINES>
<IXAND TABLE='SECURITY' TYPE='XMLINDEX'>
<INDEX IXNAME='SEC_INDUSTRY'/>
<INDEX IXNAME='SEC_SYMBOL'/>
</IXAND>
</OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

In the example in Listing 19, the IXAND element in the optimization guideline specifies that the table SECURITY should be accessed using all applicable relational and XML indexes. Since there is a relational index on the column SECURITY_TYPE and XML indexes on INDUSTRY and SYMBOL, all three indexes are ANDed together in an order chosen by the optimizer.

Listing 19. An IXAND guideline to use all XML indexes

```xml
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Example 7">
<STMTKEY SCHEMA="TPOX">
SELECT *
FROM security
WHERE security_type = 'Bond Fund'
 AND XMLEXISTS('$/SDOC/Security/Symbol[.="IBM"]')
</STMTKEY>
<OPTGUIDELINES>
<IXAND TABLE='SECURITY' TYPE='XMLINDEX' ALLINDEXES='TRUE' />
</OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

In the example in Listing 20, the IXAND guideline specifies that the table SECURITY should be accessed using an IXAND plan and that the XML index SEC_INDUSTRY must be the first index under the IXAND. The optimizer picks additional indexes for the IXAND plan in a cost-based fashion. Since a relational index is available on the column SECURITY_TYPE and an XML index is available on SYMBOL, one or both of these indexes can appear as additional legs of the IXAND, if deemed beneficial by the optimizer's cost-based analysis.
Listing 20. An IXAND guideline to use a specific XML leading index

```xml
<OPTPROFILE VERSION="9.7.0.0">
  <STMTPROFILE ID="Example 8">
    <STMTKEY SCHEMA="TPOX">
      SELECT *
      FROM security
      WHERE security_type = 'Bond Fund'
      AND XMLEXISTS('SDOC/Security/SecurityInformation
                      /StockInformation[Industry= "Software"]')
      AND XMLEXISTS('SDOC/Security/Symbol[.="IBM"]')
    </STMTKEY>
    <OPTGUIDELINES>
      <IXAND TABLE='SECURITY' TYPE='XMLINDEX' INDEX='SEC_INDUSTRY' />
    </OPTGUIDELINES>
  </STMTPROFILE>
</OPTPROFILE>
```

Influencing join orders and join methods in XML queries

All existing optimization guidelines continue to work in the presence of XML data type. Relational guidelines such as JOIN, MSJOIN, NLJOIN, and HSJOIN can be used to specify the join order and the join type. All the new XML access guidelines introduced in this article may be nested within these relational join guidelines to specify the inner or outer leg of the join.

The example guideline in Listing 21 specifies that the XML index SEC_NAME be used to access the ‘SECURITY’ table. A hash join method is requested to perform the join between the ‘SECURITY’ and the ‘ORDER’ tables. Since the XISCAN element with TABLE='SECURITY' is specified before the ACCESS element with TABLE='ORDER', the resulting plan will use the ‘SECURITY’ table as the outer leg of the hash join and the ‘ORDER’ table as the inner leg.

Listing 21. A HSJOIN guideline used to influence join method and join order in an XML query

```xml
<OPTPROFILE VERSION="9.7.0.0">
  <STMTPROFILE ID="Example 9">
    <STMTKEY SCHEMA="TPOX">
      SELECT *
      FROM order, security
      WHERE order.security_id = security.security_id
      AND XMLEXISTS('SDOC/Security/Name[.="International Business Machines"]')
    </STMTKEY>
    <OPTGUIDELINES>
      <HSJOIN>
        <XISCAN TABLE='SECURITY' INDEX='SEC_NAME'/>
        <ACCESS TABLE='ORDER' />  
      </HSJOIN>
    </OPTGUIDELINES>
  </STMTPROFILE>
</OPTPROFILE>
```

The example optimization guideline in Listing 22 contains two elements, ACCESS and XANDOR. The attribute FIRST in the ACCESS guideline element specifies that the table CUSTACC must appear as the outermost table when the tables in the FROM clause are joined. The ACCESS element further specifies that some XML index must be used to access the table CUSTACC. The XANDOR element
specifies that the table ORDER should be accessed using a XANDOR plan. The optimizer uses all applicable XML indexes in the XANDOR plan. The order of indexes is chosen by the optimizer.

**Listing 22. Using the FIRST attribute to influence join order in an XML query**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
<STMTPROFILE ID="Example 10">
  <STMTKEY SCHEMA="TPOX">
    SELECT ordqty, orddate, ordid, security, lasttrade
    FROM order, security, custacc,
    XMLTABLE('$/DOC/FIXML/Order'
      COLUMNS ordid VARCHAR(10) PATH '@ID',
      orddate date PATH '@TrdDt',
      ordqty float PATH 'OrdQty/@Qty') AS T1,
    XMLTABLE( '/$DOC/Security'
      COLUMNS security varchar(50) PATH 'Name',
      lasttrade float PATH 'Price/LastTrade') AS T2
    WHERE XMLEXISTS('$_DOC/Security[Symbol/fn:string(.)
      = $ODOC/FIXML/Order/Instrmt/@Sym/fn:string(.)]')
    and XMLEXISTS('$_DOC/FIXML/Order[@Acct/fn:string(.)
      = $CADOC/Customer/Accounts/Account/@id/fn:string(.)]')
    and XMLEXISTS('$CADOC/Customer[@id = 1011]')
    ORDER BY ordqty desc
  </STMTKEY>
  <OPTGUIDELINES>
    <ACCESS TABLE='CUSTACC' TYPE='XMLINDEX' FIRST='TRUE' />
    <XANDOR TABLE='ORDER' />
  </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

**XQuery support**

All optimization guidelines are also applicable to queries written in the XQuery language.

The optimization profile in **Listing 23** contains an XQuery expression that returns customer account information for companies that deal in software and trade in Yen currency. The guideline specifies that table SECURITY should be the outer table in the join and the XML index SEC_INDUSTRY should be used to access the SECURITY table. The guideline also specifies that the table CUSTACC should be the inner table of the join and the XML index ACC_CURRENCY should be used to access the CUSTACC table.
Listing 23. A guideline for an XQuery expression

```xml
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0.0">
  <STMTPROFILE ID="Example 11">
    <STMTKEY SCHEMA="TPOX">
      <![CDATA[
        xquery
    for $s in db2-fn:xmlcolumn("SECURITY.SDOC")/Security[
      SecurityInformation/StockInformation/Industry="Software"]
    for $c in db2-fn:xmlcolumn("CUSTACC.CADOC")/Customer
    /Accounts/Account[Currency="YEN"]
    where $s/Symbol = $c/Holdings/Position/Symbol
    return <yenaccount> {$c} </yenaccount> ]]>  
    </STMTKEY>
    <OPTGUIDELINES>
      <JOIN>
        <ACCESS TABLE='SECURITY' TYPE='XMLINDEX' INDEX='SEC_INDUSTRY'/>
        <ACCESS TABLE='CUSTACC'  TYPE='XMLINDEX' INDEX='ACC_CURRENCY'/>
      </JOIN>
    </OPTGUIDELINES>
  </STMTPROFILE>
</OPTPROFILE>
```

Troubleshooting tips and tricks

- If you have enabled a profile following all the steps in this article, but you get neither the requested plan nor any SQL0437W warning message, then:
  - Make sure you have enabled the correct optimization profile using the special register, client configuration option, or bind option, as appropriate.
  - Make sure that the profile name contains alphanumeric characters only and does not exceed 128 characters.
- If you are using an IXAND or ACCESS guideline to specify a particular leading index, but are not getting the requested plan, check the following:
  - Make sure that you have not specified both INDEX attribute and INDEX elements within the guideline. If the INDEX attribute and one or more INDEX elements are specified at the same time, the INDEX attribute is ignored.
- If you are not getting the requested plan in a query that uses aliases, then check the following:
  - Make sure that the alias and not the base table name is specified in the TABLE attribute in the optimization guideline. For example, to request an XML index access for the following query:
    ```sql
    SELECT * FROM security sec WHERE ...
    ```
    the guideline must specify
    ```xml
    <XISCAN TABLE='SEC' INDEX='SEC_INDUSTRY'/>
    ```
    and not
    ```xml
    <XISCAN TABLE='SECURITY' INDEX='SEC_INDUSTRY'/>
    ```

Conclusion

XML optimization guidelines are a powerful tool that enables you to influence plan selection and compensate for any performance degradations due to the unique characteristics of your database environments. It is important to use guidelines wisely, and not as an alternative to a robust database design. You are advised to follow the best practices recommended for DB2
pureXML. If the execution plan problems cannot be avoided, guidelines can be used to affect the optimizer plan to mitigate the performance problems.

© Copyright IBM Corporation 2009
 Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)