PROGESS® DATADIRECT XQUERY® PERFORMANCE: GENERATING SQL

Marc Van Cappellen

Jonathan Robie

PROGRESS

TABLE OF CONTENTS

✴ PROGRESS

Implementing XQuery efficiently for relational databases is not trivial, because XQuery and SQL support different operations on quite different data models. At Progress® DataDirect®, we emphasized performance and scalability in the design of our Progress® DataDirect XQuery® engine from the beginning, with a strong focus on relational data. This paper presents some of the techniques we use to generate efficient SQL for relational databases to implement XQuery for these data sources.

These techniques work and give us better performance than other XQuery implementations. Some XQuery implementations we have tested return an entire table for queries where we return only part of a single
row. Others generate the same SQL regardless of the database involved, a strategy that simply can not offer good performance. Some XQuery implementations rely on the least-commondenominator functionality of the least capable JDBC drivers, which limits performance significantly. Some XQuery implementations perform most XQuery functions in the XQuery engine instead of evaluating them in the database.

Progress DataDirect is the leading vendor of database connectivity software, and we know how to measure performance and scalability. We have spent many man-years developing extensive XQuery performance test suites, and we run these suites regularly as part of our standard development cycle. When our support staff identifies interesting customer performance scenarios, these are added to our performance test suites. In our testing, we have been pleased to find that applications written using DataDirect XQuery generally perform better than equivalent applications written using SQL, JDBC, and an XML API. Not all XQuery implementations can offer that kind of performance.

In this paper, we will explain some of the techniques we have developed, showing the SQL that DataDirect XQuery generates for a number of specific XQueries. If you want to see the SQL generated for one of your queries, you can do that using the profiler provided by your database vendor. If you are comparing XQuery products, comparing the SQL they generate can help you understand how these products will perform. Most of the generated SQL shown in this paper is for Oracle 10gR2—SQL generated for other databases may look significantly different.

OVERVIEW

Before we explore individual queries, let's take a high-level look at the techniques we use. Most of the specific techniques we show later are based on the following general principles:

1. Minimize data retrieval.
Moving data is expensive. In DataDirect XQuery, the SQL we generate is as selective as possible, retrieving only the data needed to create the results of a query. Some XQuery implementations we have tested return an entire table for queries where we return only part of a single row.

2. Leverage the database.
In DataDirect XQuery, operations that can be performed in SQL are pushed down into the database, where the relational query optimizer can leverage indexes and other structures. The performance gains this brings are particularly important for joins, Order By clauses, and SQL functions. This also reduces data retrieval, since data need not be retrieved for operations to be done in the database.

PROGRESS

3. **Optimize for each database.**
Today's relational databases support significantly different dialects of SQL, and even when two databases support the same operation, their performance may be quite different. Any given database has enough functionality to support XQuery efficiently, but the constructs needed to do this are different for each database. Some XQuery implementations support only one database; others generate the same SQL regardless of the database involved, which results in poor performance. In contrast, DataDirect XQuery uses a different SQL adaptor for each database, generating SQL specifically optimized for that database, based on our extensive performance testing.

4. **Retrieve data efficiently.**
The underlying database drivers can significantly affect performance. As the leading vendor of JDBC, ODBC, and ADO.NET drivers, Progress DataDirect knows how to make our drivers perform while providing good support for all major relational databases. And because we control both the XQuery implementation and the underlying JDBC drivers, we can add optimizations to the drivers to support our implementation. The added cost of XML construction, which can be considerable in some implementations, is negligible in ours.

5. **Support incremental evaluation.**
In many applications, results are returned to the user as soon as they are available, displaying the first results well before the entire query has been performed. Many XML applications are based on streaming architectures. In DataDirect XQuery, we use lazy evaluation so that streaming APIs can retrieve data as soon as it is available. As data is needed, we retrieve it incrementally from JDBC result sets. Because there is no need to have the entire result in memory at one time, very large documents can be created.

6. **Optimize for XML hierarchies.**
Because XML construction is hierarchical, DataDirect XQuery uses SQL algorithms that optimize retrieving data for building hierarchies. For instance, we make extensive use of merge-joins when translating XQuery to SQL.

7. **Give the programmer the last word.**
Every SQL programmer knows that occasionally you need to use hints to get optimal performance for a specific query. This is also true in XQuery, so DataDirect XQuery allows programmers to influence the SQL we generate. This can significantly improve performance in some cases.

SELECTING DATA

To minimize data retrieval, DataDirect XQuery generates very selective SQL, returning only the data that is needed for a given XQuery. To avoid retrieving rows that are not needed, the conditions in Where clauses and predicates are converted to Where clauses in the generated SQL. To avoid retrieving columns that are not needed, the generated SQL specifies the columns actually needed to evaluate the XQuery. Some XQuery implementations retrieve much more data than is actually needed, which significantly hurts performance.

PROGRESS

EXAMPLE 1. WHERE CLAUSE PUSHDOWN

In this example, the SQL query generated by DataDirect XQuery returns only the rows that are actually needed for the XQuery.

XQuery [with Where clause]

```
for $h in collection('HOLDINGS')/HOLDINGS
  where $h/SHARES < 10000
  return $h/USERID
```

XQuery [with predicate]

```
for $h in collection('HOLDINGS')/HOLDINGS[SHARES <
  10000]
  return $h/USERID
```

Generated SQL [for both forms of the XQuery]

```
SELECT ALL
   nrm4."USERID" AS RACOL1
  FROM
   "PEPPINO"."HOLDINGS" nrm4
  WHERE
   nrm4."SHARES" < 10000
```

EXAMPLE 2. COLUMN PUSHDOWN

The SQL query generated by DataDirect XQuery retrieves only the columns that are actually needed for the XQuery.

XQuery

```
for $u in collec tion('USERS')/USERS
  return <user>{$u/FIRSTNAME,$u/LASTNAME}</user>
```

Generated SQL

```
SELECT ALL
   nrm5."FIRSTNAME" AS RACOL1,
   nrm5."LASTNAME" AS RACOL2

FROM
   "PEPPINO"."USERS" nrm5
```

QUANTIFIERS

Another way DataDirect XQuery minimizes data retrieval is by implementing quantified expressions with SQL in the database. These expressions are generally used in predicates and Where clauses, reducing the number of rows retrieved from the database.

EXAMPLE 3. QUANTIFIER PUSHDOWN

XQuery

```
for $u in collection("USERS")/USERS
  where every $h in collection("HOLDINGS")/
```

✱ PROGRESS

```
HOLDINGS[USERID=$u/USERID]
satisfies $h/SHARES > 1000
return <user id="{$u/LASTNAME}"/>
```

Generated SQL

```
SELECT ALL
   nrm5."LASTNAME" AS RACOL2
 FROM
   "PEPPINO"."USERS" nrm5
 WHERE
   NOT EXISTS(
     SELECT ALL 1 AS RACOL1
     FROM "PEPPINO"."HOLDINGS" nrm10
     WHERE (CASE WHEN nrm10."SHARES" > 1000
     THEN 0 ELSE 1 END) !=0
     AND nrm10."USERID" = nrm5."USERID"
     AND LENGTH(nrm10."USERID") =
     LENGTH(nrm5."USERID")

)
```

JOINS

Relational databases are designed to optimize joins, so DataDirect XQuery leverages the database when an XQuery join involves SQL data. Performing all the joins in the database results in a dramatic performance gain because DataDirect XQuery simply uses the SQL engine in your database.

EXAMPLE 4. JOIN PUSHDOWN

XQuery

```
for $u in collection('USERS')/USERS
  for $h in collection('HOLDINGS')/HOLDINGS
  where $u/USERID = $h/USERID
  return <holding name="{$u/LASTNAME}">{$h/SHARES/
  text()}</holding>
```

Generated SQL

```
SELECT ALL
   nrm5."LASTNAME" AS RACOL1,
   nrm9."SHARES" AS RACOL2
 FROM
   "PEPPINO"."USERS" nrm5,
   "PEPPINO"."HOLDINGS" nrm9
 WHERE
   nrm5."USERID" = nrm9."USERID" AND
   LENGTH(nrm5."USERID") = LENGTH(nrm9."USERID")
```

PROGRESS

SORTING DATA

DataDirect XQuery leverages the database for sorting because the database can leverage indexes to sort much more efficiently.

EXAMPLE 5. ORDER BY PUSHDOWN

The XQuery uses the Order By clause to sort database data in the database.

XQuery

```
for $u in collection('USERS')/USERS
  order by $u/LASTNAME
  return <user name="{$u/LASTNAME}"/>
```

Generated SQL

```
SELECT ALL
    nrm5."LASTNAME" AS RACOL1
  FROM
    "PEPPINO"."USERS" nrm5
  ORDER BY
    nrm5."LASTNAME" ASC
```

EXAMPLE 6. ORDER BY PUSHDOWN—EMPTY LEAST

DataDirect XQuery supports all variants of Order By in XQuery, pushing them down to the database. For instance, here is the SQL generated for a query that uses the empty least clause.

XQuery

```
for $u in collection('USERS')/USERS
  order by $u/OTHERNAME descending empty least
  return <user name="{$u/LASTNAME}"/>
```

Generated SQL [Oracle 10gR2]

```
SELECT ALL
    nrm5."LASTNAME" AS RACOL1
  FROM
    "PEPPINO"."USERS" nrm5
  ORDER BY
    nrm5."OTHERNAME" DESC NULLS LAST
```

EXAMPLE 7. SORTING EMPTY GREATEST ON MICROSOFT SQL SERVER

In some cases, supporting an XQuery sort order requires some ingenuity. For instance, Microsoft SQL Server does not support sorting NULL high, so we use a simple trick to implement empty greatest for this database efficiently while still correctly implementing the semantics of XQuery.

XQuery

```
for $u in collection('USERS')/USERS
  order by $u/OTHERNAME descending empty greatest
  return <user name="{$u/LASTNAME}"/>
```

PROGRESS

Generated SQL (SQL Server)

```
SELECT ALL
    nrm5."LASTNAME" AS RACOL1
  FROM
    "PEPPINO"."USERS" nrm5
  ORDER BY
    (CASE WHEN nrm5."OTHERNAME" IS NULL THEN 0
    ELSE 1 END) ASC, nrm5."OTHERNAME" DESC
```

BUILDING XML HIERARCHIES

XQuery is all about XML, which is based on hierarchy and sequence. If an XQuery is to be fast, the XQuery implementation must handle hierarchy efficiently. Choosing the correct SQL algorithm is important, so our performance testing of DataDirect XQuery included a variety of algorithms. You can choose any of these algorithms for a given query.

The sort-merge join algorithm is an efficient way to obtain data for building hierarchies in all queries. In many cases it is the fastest, and it has very good worst-case performance, so this is the default algorithm in DataDirect XQuery. For each level of hierarchy, SQL result sets are generated using the same sort order. These result sets are then merged in the XQuery engine, which creates the XML structures that represent the hierarchy.

EXAMPLE 8. SORT-MERGE JOIN

Using the sort-merge algorithm, the inner and outer FLWOR expressions are each translated into a SQL query, sorted by row id, and the results are merged in the XQuery engine.

XQuery

```
for $u in collection("USERS")/USERS
  return
    <user name="{$u/LASTNAME}">{
    for $h in collection("HOLDINGS")/HOLDINGS
    where $h/USERID = $u/USERID
    return
      <holding id="{$h/STOCKTICKER}">{data($h/
      SHARES)}</holding>
    }</user>
```

SQL (for users)

```
SELECT ALL
    nrm5."LASTNAME" AS RACOL1,
    nrm5."ROWID" AS RACOL2
  FROM
    "PEPPINO"."USERS" nrm5
  ORDER BY
    nrm5."ROWID" ASC
```

SQL (for holdings)

```
SELECT ALL
    nrm5."ROWID" AS RACOL2,
    nrm9."STOCKTICKER" AS RACOL3,
```

PROGRESS

```
      nrm9."SHARES" AS RACOL4
    FROM
      "PEPPINO"."USERS" nrm5,
      "PEPPINO"."HOLDINGS" nrm9
    WHERE
      nrm9."USERID" = nrm5."USERID" AND
      LENGTH(nrm9."USERID") = LENGTH(nrm5."USERID")
    ORDER BY
      nrm5."ROWID" ASC
```

In some cases, such as when the XML nesting level is limited to four or less, you get better performance using the outer join algorithm, which uses only a single SQL statement. However, when the XML nesting level is greater than four, this single SQL statement can be very complex, so this is not the default algorithm in DataDirect XQuery. The following query shows how to choose the outer join algorithm.

EXAMPLE 9. OUTER JOIN

This example shows how to use a declaration option to tell DataDirect XQuery to use the outer join algorithm when generating SQL for building hierarchies.

XQuery

```
declare option ddtek:sql-rewrite-algorithm "outer-join";
  for $u in collection("USERS")/USERS
  return
    <user name="{$u/LASTNAME}">{
    for $h in collection("HOLDINGS")/HOLDINGS
    where $h/USERID = $u/USERID
    return
      <holding id="{$h/STOCKTICKER}">{data($h/
      SHARES)}</holding>
  }</user>
```

Generated SQL

```
SELECT ALL
    RAREL3.RACOL3 AS RACOL3,
    RAREL3.RACOL4 AS RACOL4,
    RAREL3.RAOJ1 AS RAOJ1,
    nrm5."LASTNAME" AS RACOL1,
    nrm5."ROWID" AS RACOL2
  FROM
    {oj
      "PEPPINO"."USERS" nrm5
    LEFT OUTER JOIN
      (SELECT ALL
    nrm9."STOCKTICKER" AS RACOL3,
    nrm9."SHARES" AS RACOL4,
    nrm9."USERID" AS USERID,
    2 AS RAOJ1
    FROM "PEPPINO"."HOLDINGS" nrm9) RAREL3
  ON
    RAREL3.USERID = nrm5."USERID" AND
```

```
    LENGTH(RAREL3.USERID) = LENGTH(nrm5."USERID")}
    ORDER BY RACOL2 ASC
```

EXTERNAL VARIABLES

XQJ prepared queries are analogous to SQL prepared statements; they use XQuery external variables in the same way that SQL uses parameter markers. The DataDirect XQuery SQL Adapter converts queries with external variables into SQL prepared statements [except when an external variable contains a sequence—this case is discussed in Example 11].

EXAMPLE 10. XQUERY EXTERNAL VARIABLES

XQuery

```
declare variable $shares as xs:integer external;
  for $h in collection('HOLDINGS')/HOLDINGS
  where $h/SHARES < $shares
  return $h/USERID
```

Generated SQL

```
SELECT ALL
    nrm4."USERID" AS RACOL1
  FROM
    "PEPPINO"."HOLDINGS" nrm4
  WHERE
    nrm4."SHARES" < ?
```

EXAMPLE 11. SEQUENCES AS EXTERNAL VARIABLES

XQuery supports sequences as external variables, but in SQL, a parameter is a single value. DataDirect XQuery uses temporary tables in the database to represent XQuery external variables that contain sequences. The sequence is inserted into the temporary table using batch insert in order to minimize the number of database round trips.

XQuery

```
declare variable $shares as xs:int* external;
  for $h in collection('HOLDINGS')/HOLDINGS
  where $h/SHARES = $shares
  return $h/USERID
```

Generated SQL

```
INSERT INTO RATEMP009368710001( RAVAR, RAINT)
  VALUES(0, ?);
  SELECT ALL
    nrm4."USERID" AS RACOL2
  FROM
    "PEPPINO"."HOLDINGS" nrm4
  WHERE
    EXISTS(SELECT ALL
      1 AS RACOL1
    FROM
      RATEMP009368710001 nrm7
```

✳ PROGRESS

```
WHERE
  nrm7.RAVAR = 0 AND nrm4."SHARES" =
  nrm7.RAINT)
;
```

```
DELETE FROM RATEMP009368710001 WHERE RAVAR IN (0)
```

The cost of temporary tables and execution of three SQL statements is greatly outweighed by the ability to evaluate the Where clause in the SQL database. Most implementations do not push sequences of external variables into the database, so they are not able to push the Where clause into the database and must retrieve the entire table, evaluating the Where clause in the XQuery engine.

XQUERY GLOBAL VARIABLES

SQL does not have a construct equivalent to XQuery's global variables. When an XQuery global variable is set to values from the database, DataDirect XQuery generally inlines the variable in the statement created for the query body.

EXAMPLE 12. GLOBAL VARIABLES

XQuery

```
declare variable $users := collection('USERS')/
  USERS;
  for $u in $users
  return $u/LASTNAME
```

Generated SQL

```
SELECT ALL
  nrm5."LASTNAME" AS RACOL1
  FROM
  "PEPPINO"."USERS" nrm5
  WHERE
  nrm5."LASTNAME" IS NOT NULL
```

XQUERY BUILT-IN FUNCTIONS

XQuery has a large library of built-in functions, which are all supported by DataDirect XQuery. The vast majority of these are translated into SQL. For some of these functions, the translation is straightforward; for others, the translation requires more thought. We do not know of another XQuery implementation in which such a high proportion of XQuery functions are executed directly in the database.

EXAMPLE 13. A STRAIGHTFORWARD BUILT-IN FUNCTION

XQuery

```
for $u in collection('USERS')/USERS
  return concat($u/FIRSTNAME,$u/LASTNAME)
```

Generated SQL

★ PROGRESS

```
SELECT ALL
  {fn CONCAT(nrm5."FIRSTNAME",nrm5."LASTNAME")} AS
  RACOL1
  FROM
  "PEPPINO"."USERS" nrm5
```

For most of the functions in the XQuery built-in library, though, translating to efficient SQL is nontrivial. DataDirect XQuery does this without changing the semantics of the XQuery function.

EXAMPLE 14. A MORE COMPLEX XQUERY BUILT-IN FUNCTION

XQuery

```
for $u in collection('USERS')/USERS
  return substring-before($u/FIRSTNAME, 'lo')
```

Generated SQL

```
SELECT ALL
    (CASE WHEN
      {fn LOCATE('lo',nrm5."FIRSTNAME")} > 0
    THEN
      {fn LEFT( nrm5."FIRSTNAME", {fn
      LOCATE('lo',nrm5."FIRSTNAME")} -1)}
    ELSE
      '' END) AS RACOL1
    FROM
      "PEPPINO"."USERS" nrm5
```

This translation uses features found only in Oracle databases along with JDBC escapes that are standard, but not supported by all drivers—these features are needed to implement substring-before() efficiently. Different features are used on other databases to achieve an efficient and conformant implementation.

USER-DECLARED FUNCTIONS

DataDirect XQuery uses intelligent function inlining so that user-declared functions can be executed in the database.

EXAMPLE 15. INLINING USER-DECLARED FUNCTIONS

XQuery

```
declare function local:popularShares($quantity) {
  $quantity > 10000
  };
  collection('HOLDINGS')/HOLDINGS
  SHARES[local:popularShares(.)]
```

SQL

```
SELECT ALL
    nrm5."SHARES" AS RACOL1
  FROM
    "PEPPINO"."HOLDINGS" nrm5
```

✳ PROGRESS

```
WHERE
   nrm5."SHARES" > 10000
```

CALLING DATABASE FUNCTIONS

DataDirect XQuery lets you call any function defined in your database directly as an XQuery external function. Both built-in database functions and user-defined SQL functions are supported. External Java functions can also be called, and these can invoke database functions. Our support for external functions provides a way to invoke SQL functions or Java functions with virtually no performance overhead. Because this support is a central part of our open design, you can use DataDirect XQuery in just about any architecture.

EXAMPLE 16. CALLING DATABASE FUNCTIONS

XQuery

```
declare function ddtek-sql:RTRIM($s as xs:string) as
  xs:string external;
  collection('USERS')/USERS/ddtek-sql:RTRIM(LASTNAME)
```

Generated SQL

```
SELECT ALL
   RTRIM(nrm7."LASTNAME") AS RACOL1
   FROM
   "PEPPINO"."USERS" nrm7
```

CONSISTENT SQL GENERATION

In XQuery, many queries may be semantically equivalent. DataDirect XQuery recognizes common equivalences in queries and ensures that efficient SQL is generated for all the different ways of expressing a query.

EXAMPLE 17. SQL GENERATION FOR EQUIVALENT QUERIES

XQuery (all of these queries are equivalent)

```
collection("USERS")/USERS/FIRSTNAME[.='John']

collection("USERS")//FIRSTNAME[.='John']

collection("USERS")/USERS[FIRSTNAME='John']/FIRSTNAME

for $u in collection("USERS")/USERS

where $u/FIRSTNAME = 'John'

return $u/FIRSTNAME

declare function local:testUser($firstname){$firstname
  = 'John'};

for $u in collection("USERS")/USERS

where local:testUser($u/FIRSTNAME)
```

```
return $u/FIRSTNAME

for $u in collection("USERS")/USERS

return if ($u/FIRSTNAME = 'John') then $u/FIRSTNAME
  else ()
```

Generated SQL (generated for all the above queries)

```
SELECT ALL
   nrm6."FIRSTNAME" AS RACOL1
  FROM
   "PEPPINO"."USERS" nrm6
  WHERE
   nrm6."FIRSTNAME" = 'John' AND
   LENGTH(nrm6."FIRSTNAME") = LENGTH('John')
```

RELAXING XQUERY SEMANTICS

When translating XQuery to SQL, DataDirect XQuery is careful to preserve XQuery semantics, even if this complicates the generated SQL. For example, unlike XQuery string comparison, SQL character comparison is not sensitive to differences in trailing spaces. To accommodate this semantic difference, the SQL statements that DataDirect XQuery executes compare both the strings and the length of the strings.

EXAMPLE 18. STRING COMPARISON AND TRAILING SPACES (XQUERY SEMANTICS)

XQuery

```
for $u in collection('USERS')/USERS
  where $u/FIRSTNAME = 'John'
  return <user name="{$u/LASTNAME}"/>
```

SQL

```
SELECT ALL
   nrm5."LASTNAME" AS RACOL1
  FROM
   "PEPPINO"."USERS" nrm5
  WHERE
   nrm5."FIRSTNAME" = 'John' AND
   LENGTH(nrm5."FIRSTNAME") = LENGTH('John')
  ORDER BY
   nrm5."ROWID" ASC
```

Note the additional comparison on the string length, which is needed to fully conform to the XQuery specification. But in most cases VARCHAR columns don't have trailing spaces, and trailing spaces in a CHAR column are not significant. Using a declaration option, you can instruct DataDirect XQuery to ignore the trailing spaces for character comparisons, which corresponds to the semantics of SQL string comparisons and, thus, simplifies the generated SQL.

PROGRESS

EXAMPLE 19. STRING COMPARISON AND TRAILING SPACES (IGNORING TRAILING SPACES)

XQuery

```
declare option ddtek:sql-options "ignore-trailing-
  spaces=yes";
  for $u in collection('USERS')/USERS
  where $u/FIRSTNAME = 'John'
  return <user name="{$u/LASTNAME}"/>
```

SQL

```
SELECT ALL
   nrm5."LASTNAME" AS RACOL1
  FROM
   "PEPPINO"."USERS" nrm5
  WHERE
   nrm5."FIRSTNAME" = 'John'
```

The DataDirect XQuery User's Guide lists more declaration options that can be used to simplify the generated SQL.

SUMMARY

DataDirect XQuery was designed for performance and scalability, which are essential in business-critical applications. Because of our careful attention to the generating the best possible SQL to implement XQuery on any given database, as well as using the best available JDBC technology, DataDirect XQuery has exceptional performance. It generally outperforms both Java applications that create XML from database data using JDBC, SQL, and XML APIs. It also generally outperforms other XQuery implementations.

Implementing XQuery efficiently for SQL databases requires sophisticated query optimization and SQL generation algorithms. In this document, we have outlined some of the techniques DataDirect XQuery uses to generate efficient SQL. Our performance suites and our customers' experience show that these techniques are very effective. When you need more control over the SQL that is used, you can call SQL functions as XQuery external functions or use declaration options to influence the algorithms we use in generated SQL.

PROGRESS