Oracle XML DB：
XMLクエリで最適なパフォーマンスを
実現するためのベスト・プラクティス

リリース19cおよび21c、Cloudおよびオンプレミス

2022年2月

目次

はじめに

XQuery言語に対するOracle XML DBのサポートは、SQL/XML関数のXMLQuery、XMLTable、XMLExists、およびXMLCastのネイティブ実装を通じて提供されます。XMLQuery、XMLTable、XMLExists、またはXMLCastを含むSQL文は、リレーショナル・データベースとXQuery固有の最適化テクノロジーの両方を活用して、全体的にコンパイルおよび最適化されます。

XQueryの最適化は、次の2つの広い領域に分けることができます。

- **論理的な最適化とは**、XQueryを、XQueryセマンティックをモデル化するXML演算子で拡張された同等のSQLクエリ・ブロックに変換することです。これらの最適化は、XMLストレージや索引付けモデルから独立した汎用のXQuery最適化です。
- **物理的な最適化とは**、XML演算子、特にXPath演算子を、基礎となる内部ストレージと、XMLストレージおよび索引付けモデルに固有の索引表上で直接同等の演算子に変換することです。XQuery最適化の結果は、XQueryを呼び出す/XMLクエリ文のEXPLAIN PLANによって調べることができます。

本書では、最高のパフォーマンスを実現するためのXQueryのベスト・プラクティスについて説明します。論理的なクエリと物理的なクエリの両方の最適化について説明します。さまざまなXMLストレージと索引付けのオプションについて詳しく説明し、クエリに適切な索引を選択する方法と、XQueryで最高のパフォーマンスを実現する方法について説明します。

SQL/XMLとXQuery

Oracle XML DBは、XQuery言語仕様の最新バージョン、つまりW3C XQuery 1.0勧告をサポートしています。XQuery 1.0は、XMLデータをクエリするために設計されたW3C言語です。これは多くの点でSQLに似ていますが、SQLが構造化されたリレーショナル・データをクエリするために設計されているのと同じように、XQueryはさまざまなデータソースからの半構造化されたXMLデータをクエリするために特に設計されています。XQueryを使用すると、XMLデータがデータベース表に保存されているか、Webサービスを通じて利用できるか、またはその場で作成されるかに関係なく、XMLデータが見つかるどこででもクエリを実行できます。XQuery 1.0について詳しくは、http://www.w3.org/TR/xquery/を参照してください。

W3CのXQuery言語に加えて、SQL標準では、SQL言語とXQuery言語の間の一般的なインタフェースとして、標準SQL/XML関数XMLQuery()、XMLExists()、XMLCast()、および表構成メンバーXMLTable()が定義されています。他のSQL/XML関数の場合と同じように、リレーショナル・データからXMLを生成するために使用されるXMLElement()、XMLAgg()、XMLForest()、XMLConcat()、およびXMLQuery()、XMLExists()、XMLCast()関数、さらにXMLTable()表構成メンバーにより、SQLとXMLの両方の機能と柔軟性を活用できます。これらの関数を使用すると、XMLのクエリと操作、リレーショナル・データを使用したXMLデータの構築、XMLであるかのようにリレーショナル・データをクエリすること、およびXMLデータからのリレーショナル・データの構築ができます。

SQL/XML関数XMLQuery()、XMLExists()、XMLCast()およびXMLTable()構成メンバーはすべて、XMLType入力に対してXQuery式を評価しますが、それらの間でXQueryの結果が使用される方法は異なります。したがって、最高のパフォーマンスを達成するには、SQLのさまざまな句でこれらを使用する必要があります。XQuery言語では、式は常に項目のシーケンスを返します。さまざまなSQLコンテキストで項目のシーケンスがどのように使用されるかは、これらのSQL/XML関数とXMLTable表構成メンバーの適切な用途により、次のように分類されます。

- 結果シーケンス内のすべての項目を単一のXML文書または断片として使用するには、**XMLQuery()**を関数式として（通常はSQLのSELECT句のSELECTリストで）使用し、結果シーケンスをXML文書または断片を表す1つのXMLType値として集計します。たとえば、以下のクエリは、PASSING句を指定した関数XMLQueryを使用して、XMLType列のoe.warehouse_specをコンテキスト項目としてXQueryに渡します。面積が80,000より大きいウェアハウスごとにDetails要素を構築します。/Warehouse/ Area > 80000

  例1：XMLQueryとPASSING句の使用

  ```
  SELECT warehouse_name,
          XMLQuery(
              'for $i in /Warehouse
              where $i/Area > 80000
              return <Details>
                          <Docks num="{$i/Docks}"/>
                          <Rail>{if ($i/RailAccess = "Y")
                                  then "true" else "false"}
                          </Rail>
                      </Details>'
              PASSING warehouse_spec RETURNING CONTENT) big_warehouses
      FROM warehouses;
  ```

- **XMLTable()**構成メンバーは、SQLのFROM句で使用され、XQueryの評価結果を行の表として（結果シーケンス内の各XQuery項目をXMLType値として）返します。ユーザーは、XMLTableを使用して、XMLデータに対するリレーショナル・ビューを生成できます。次のようになります。

  例2：XMLTableを使用して、XMLデータに対するリレーショナル・ビューを生成します。

  ```
  SELECT lines.lineitem, lines.description, lines.partid,
          lines.unitprice, lines.quantity
  ```

```
FROM purchaseorder,
        XMLTable('for $i in /PurchaseOrder/LineItems/LineItem
                    where $i/@ItemNumber >= 8
                      and $i/Part/@UnitPrice > 50
                      and $i/Part/@Quantity > 2
                    return $i'
                  PASSING OBJECT_VALUE
                COLUMNS lineitem      NUMBER        PATH '@ItemNumber',
                        description VARCHAR2(30)    PATH 'Description',
                        partid        NUMBER        PATH 'Part/@Id',
                        unitprice     NUMBER        PATH 'Part/@UnitPrice',
                          quantity      NUMBER        PATH 'Part/@Quantity') lines;
```

- XQueryの結果が空のシーケンスかどうかを判断するために、ブールの結果を持つ**XMLExists()**は通常、SQLのWHERE句またはHAVING句、またはSQL CASE式の条件式で使用されます。以下の例は、SELECTリストでXMLExists()を使用する方法を示しています。

例3：SELECTリストでのXMLExists()とCASE式の使用

```
SELECT
        CASE WHEN XMLEXISTS('$po/PurchaseOrder/LineItems/Part'
        PASSING OBJECT_VALUE AS "po") THEN 1 ELSE 0 END
FROM purchaseorder,
```

-
- シーケンス結果（通常はXMLノードのリーフ値）を、NUMBER、VARCHAR、DATE、TIMESTAMPなどのSQLスカラー・タイプとしてキャストするために、**XMLCast()**は関数式として使用され、SELECT句のSELECTリスト、GROUP BY句のgroup byリスト、またはORDER BY句のorder byリストで使用されるSQLスカラー値項目を結果として得られます。

XQueryをSQL/XML関数とXMLTable構成メンバーで使用して、表やビューからXMLType値をクエリする場合、Oracle XML DBはXQuery式をSQLクエリ・ブロックと演算子のセットにコンパイルし、基礎となるXMLストレージと索引を活用して最適化します。このネイティブなXQuery/SQL/XML最適化モデルは、論理的な最適化と物理的な最適化という2ステップのプロセスによって概念化されています。

1. **論理的な最適化**は、XMLストレージや、基礎となるXMLType値に対する索引付けに依存しません。SQL/XML関数のXMLQuery、XMLExists、XMLCast、およびXMLTable構成メンバーに引数として渡されるXQuery式は、XQueryのセマンティックをモデル化する内部SQLクエリ・ブロックと演算子ツリーにコンパイルされます。一般的な内部演算子の1つは、入力XMLType値をナビゲートするXPath演算子です。XMLQuery、XMLTable、XMLExists、またはXMLCastを含むSQL文は、リレーショナル・データベースとXQuery固有の最適化テクノロジーの両方を活用して、全体的にコンパイルおよび最適化されます。

2. **物理的な最適化**は、基礎となるストレージおよび索引付けモデルに固有です。使用されるXMLストレージおよび索引付け方法に応じて、XPath内部演算子をさらに最適化して、基礎となるXML索引またはストレージに使用される基礎となる物理リレーショナル・ストレージ表上で動作するSQLクエリ・ブロックにすることができます。リレーショナル・オプティマイザは、最適な実行計画を達成するために、結果として得られるSQLクエリ・ブロックと演算子ツリーを最適化します。

結果のクエリ計画は、SQL行ソース・イテレータ・モデルを使用して実行されます。このネイティブXQuery/SQL/XML最適化モデルは、主にクエリ言語としてXQueryを使用した場合の、適切なXMLストレージおよび索引付けモデルを使用してデータベースに保存されたXML文書を検索したり、XMLTable構成メンバーを使用してXMLをリレーショナル結果として表示したりするときのパフォーマンス目標を達成します。'EXPLAIN PLAN'を使用してSQLクエリをチューニングすることが重要であるように、SQL/XMLクエリを理解して'EXPLAIN PLAN'を使用してチューニングすることも重要です。

これについては、本書の後続のいくつかのセクションで詳しく説明し、さまざまなXMLストレージと索引のオプションを取り上げます。

さらに、XQueryは主にXML文書を操作および変換するための言語としても使用できます。入力XMLType値は通常、永続的なXMLまたは一時的なXMLType値から取得された単一のXML文書または断片です。このような場合、XQueryはXML DBで機能的に評価できます。最適なパフォーマンスを実現するには、XML DBでのXQueryの使用法を理解し、分類することが重要です。これについては、この文書の後半にある'XQueryで最高のパフォーマンスを実現する'のセクションで詳しく説明します。

オラクル独自の（XPath1.0ベースの）構文から標準SQL/XML XQueryベースの構文への移行

11gR2以降、オラクルは、以下の表1に示すように、標準ベースのXQuery構文を優先して、主にXPath 1.0ベースの旧式である独自の多くの演算子を非推奨にしてきました。お使いのコードに非推奨の関数または演算子が含まれていない場合には、次のセクションに進むことができます。

表1： 古い構文からXQuery構文への移行

オラクル独自の古い構文	新規XQuery SQL/XMLベースの構文
extract()	XMLQuery()
extractValue	XMLCast(XMLQuery())
existsNode()	XMLExists()
Table (XMLSequence)	XMLTable
ora:instanceof	instanceof
ora:instanceof-only	@xsi:type
getNamespace	fn:namespace-uri
getRootElement	fn:local-name
getStringVal、getBlobVal、getClobVal	XMLSerialize
Xmltype()	XMLParse() for varchar, clob, blob input
DBMS_XMLQUERY	XMLQuery()
DBMS_XMLGEN	SQL/XML演算子
Oracle XML DML演算子	XQUery Update Facility

非推奨の主にXPath 1.0ベースの構文とXQueryベースの構文の間には、重要ないくつかのセマンティックな相違点があります。移行しやすいように、これらは付録Aでリストしています。xmltypeのextract()メソッドやexistsNode()メソッドを呼び出す代わりに、PL/SQL XMLType変数にXQueryを適用する方法については、本書の"XQueryガイドライン6"も参照してください。

次の表は、オラクル独自のXML DML演算子とそれに相当するXquery Updateの構文の例を示しています。

注：オラクル独自のXMLDMLには、"rename"および"insert as first into"操作はありません。

Update warehouses set warehouse_spec = **appendChildXML**(warehouse_spec, 'Warehouse/Building', XMLType('<Owner>Grandco</Owner>'));	Update warehouses set warehouse_spec = XMLQuery('copy $tmp := . modify (for $i in $tmp/Warehouse/Building return insert node <Owner>Grandco</Owner> as last into $i) return $tmp' passing warehouse_spec returning content);
Update warehouses set warehouse_spec = **deleteXML**(value(po), '/Warehouse/Building');	Update warehouses set warehouse_spec = XMLQuery('copy $tmp := . modify delete node $tmp/Warehouse/Building return $tmp' passing warehouse_spec returning content);
[単一ノードのケース] Update warehouses set warehouse_spec = **insertXML**(warehouse_spec, '/Warehouse/Building/Owner[2]', XMLType('<Owner>ThirdOwner</Owner>');	Update warehouses set warehouse_spec = XMLQuery('copy $tmp := . modify insert node <Owner>ThirdOwner</Owner> into $tmp/Warehouse/Building/Owner[2] return $tmp' passing warehouse_spec returning content);
[単一ノードのケース] Update warehouses set warehouse_spec = **insertXMLBefore**(warehouse_spec, '/Warehouse/Building/Owner[2]', XMLType('<Owner>FirstOwner</Owner>');	Update warehouses set warehouse_spec = XMLQuery('copy $tmp := . modify insert node <Owner>FirstOwner</Owner> before $tmp/Warehouse/Building/Owner[2] return $tmp' passing warehouse_spec returning content);
[単一ノードのケース] Update warehouses set warehouse_spec = **insertXMLAfter**(warehouse_spec, '/Warehouse/Building/Owner[2]', XMLType('<Owner>ThirdOwner</Owner>');	Update warehouses set warehouse_spec = XMLQuery('copy $tmp := . modify insert node <Owner>ThirdOwner</Owner> after $tmp/Warehouse/Building/Owner[2] return $tmp' passing warehouse_spec returning content);
Update warehouses set warehouse_spec = **updateXML**(warehouse_spec, '/Warehouse/Docks/text()', 4);	Update warehouses set warehouse_spec = XMLQuery('copy $tmp := . modify (for $i in $tmp/Warehouse/Docks/text() return replace value of node $i with 4) return $tmp' passing warehouse_spec returning content);
Update warehouses set warehouse_spec = **insertChildXML**(warehouse_spec, '/Warehouse/Building', 'Owner', XMLType('<Owner>LesserCo</Owner>'));	Update warehouses set warehouse_spec = XMLQuery('copy $tmp := . modify (for $i in $tmp/Warehouse/Building return insert node <Owner>LesserCo</Owner> into $i) return $tmp' passing warehouse_spec returning content);
Update warehouses set warehouse_spec = **insertChildXMLBefore**(warehouse_spec, '/Warehouse/Building', 'Owner', XMLType('<Owner>LesserCo</Owner>'));	Update warehouses set warehouse_spec = XMLQuery('copy $tmp := . modify (for $i in $tmp/Warehouse/Building return insert node <Owner>LesserCo</Owner> before $i) return $tmp' passing warehouse_spec returning content);
Update warehouses set warehouse_spec = **insertChildXMLAfter**(warehouse_spec, '/Warehouse/Building', 'Owner', XMLType('<Owner>LesserCo</Owner>'));	Update warehouses set warehouse_spec = XMLQuery('copy $tmp := . modify (for $i in $tmp/Warehouse/Building return insert node <Owner>LesserCo</Owner> after $i) return $tmp' passing warehouse_spec returning content);

[コレクションのケース] Update warehouses set warehouse_spec = **insertXML**(warehouse_spec, '/Warehouse/Building/Owner', XMLType('<Owner>AnotherOwner</Owner>');	Update warehouses set warehouse_spec = XMLQuery('copy $tmp := . modify (for $i in $tmp/Warehouse/Building/Owner return insert node <Owner>AnotherOwner</Owner> into $i) return $tmp' passing warehouse_spec returning content);
[NULLのケース] Update warehouses set warehouse_spec = **updateXML**(warehouse_spec, '/Warehouse/Docks', null);	Update warehouses set warehouse_spec = XMLQuery('copy $tmp := . modify delete node $tmp/Warehouse/Docks return $tmp' passing warehouse_spec returning content);
[空のノードのケース] Update warehouses set warehouse_spec = **updateXML**(warehouse_spec, '/Warehouse/Docks', '');	Update warehouses set warehouse_spec = XMLQuery('copy $tmp := $p1 modify (for $j in $tmp/Warehouse/Docks return replace node $j with $p2) return $i' passing passing warehouse_spec "p1", '' as "p2" returning content) ;
[複数パスのケース] Update warehouses set warehouse_spec = **updateXML**(warehouse_spec, '/Warehouse/Docks/text()', extractValue(warehouse_spec, '/Warehouse/Docks/text()')+4, '/Warehouse/Docks/text()', extractValue(warehouse_spec, '/Warehouse/Docks/text()')+4);	Update warehouses set warehouse_spec = XMLQuery('copy $tmp := . modify ((for $i in $tmp/Warehouse/Docks/text() return replace value of node $i with $i+4), (for $i in $tmp/Warehouse/Docks/text() return replace value of node $i with $i+4)) return $tmp' passing warehouse_spec returning content);

XQueryで最高のパフォーマンスを実現する

XQueryのベスト・プラクティスとパフォーマンス・チューニングは、次の2つの部分に分けることができます。

- XMLTypeストレージ・オプションに依存しないベスト・プラクティス。これらは、"ストレージに依存しないベスト・プラクティス"のセクションにリストされています。
- ユーザーが選択したXMLTypeストレージに固有のベスト・プラクティス。これらには、XQueryを高速化するためにユーザーが作成できるさまざまな索引が含まれます。これらは、"ストレージ依存のパフォーマンス・チューニング"セクションにリストされています。

ストレージに依存しないベストプラクティス

Oracle XML DBでは、XML文書はリレーショナル表のXMLType表またはXMLType列のいずれかに保存されます。XML DBは、大量のXML文書を保存し、XQueryを使用してそれらのXML文書から検索するように設計されています。その目的は、XQueryを使用して操作および変換するための修飾されたXML文書または文書フラグメントを見つけること、XMLTable構成メンバーを使用してXMLに対するリレーショナル・ビューを投影し、リレーショナルにクエリを実行して、成熟したリレーショナル・アプリケーションと統合できるようにすることなどです。

XQueryガイドライン1：XMLExists()およびXMLQuery()を使用してXML DBに保存されているXMLを検索および変換する

XMLType列に保存されたXML文書を検索し、検索結果を操作するSQL文の一般的な記述方法を以下に示します。

例4：検索と変換

```
SELECT XMLQUERY('…' PASSING T.X RETURNING CONTENT)
FROM purchaseorder T
WHERE XMLEXISTS('$p/PurchaseOrder/LineItems/LineItem/Part[@Id="702372"]'
                PASSING T.X AS "p");
```

このSQL文では、文のWHERE句でXMLExists()を使用して、"干し草の山から針を見つける"という典型的なデータベース・タスクを実行します。表purchaseorderには数十億のXML文書が保存される可能性があるため、各XML文書のXMLExists()で使用されるXQueryの機能評価による表スキャンの代わりに、適切な索引を使用することが、クエリ・パフォーマンスを達成するために重要です。最高のパフォーマンスを実現するには、XMLExists()で使用されるXQueryが索引に適したものである必要があります。

XMLExists()で使用されるXQueryが全体として索引に適していない場合は、XQueryを索引に適した式と索引に適さない式に分割し、それらをSQL AND構成メンバーで接続された2つの異なるXMLExists()関数で使用してみてください。このようにして、少なくとも索引に適したXMLExists()は索引を使用して評価でき、索引に適したXMLExists()はポスト索引フィルターとして評価できます。

XQueryガイドライン2：XMLExists()を使用してXML文書を検索してXML DML演算子を介して変更する

XMLType列に保存されたXML文書を検索および変更するSQL文の一般的な記述方法を次に示します。

例5：XMLExists()を使用した検索後にXML文書を更新する

```
UPDATE purchaseorder T SET T.X = DELETEXML(T.X,
                        '/purchaseOrder/LineItems/LineItem[itemName ="TV"]' ) WHERE
XMLEXISTS('$p/PurchaseOrder/LineItems/LineItem/Part[@Id="717951002372"]'
                PASSING T.X AS "p");
```

XQueryガイドライン1で取り上げていたことと同じように、XMLExists()は、"干し草の山から針を見つける"かのように、どのXML文書が変更されるかを識別するために使用されます。UPDATE代入のRHSで使用される関数には、XMLTypeを返す任意の式を使用できます。たとえば、XMLTypeを返すPL/SQL関数呼び出しなどです。セマンティック上は、SQL UPDATE文のRHS式は、XMLType列値全体の文書置換を行うために、LHSのXMLType列に割り当てられたXMLTypeインスタンス文書を返します。

しかし、Oracle XML DBは可能な場合は必ずXML DML演算子のリライト最適化を行い、文書全体を置き換えるのではなく、基礎となるXMLストレージ構造を部分的に更新します。バイナリXMLストレージの場合、ストリーム評価を使用してXPathを評価できる場合、すべてのXML DML演算子に対してXML DML演算子のリライトが行われます。

XML DML演算子のリライトは、/*+NO_XML_DML_REWRITE */ SQLヒントを使用して明示的に無効にできます。これは、XMLストレージ・モデルに関係なく当てはまります。

XQueryガイドライン3：XMLTable構成メンバーを使用してリレーショナル・アクセスでXMLをクエリする

XML文書は本質的に階層構造になっており、典型的なマスター-ディテール・リレーションシップがあります。したがって、以下の例に示すように、XML文書内のマスター-ディテール構成メンバーを、XMLTable構成メンバーを使用して一連のリレーショナル表として射影し、各構成メンバーのリーフ値を検索用のXMLTableの列として射影するのが一般的です。

例6：XMLTableの使用

```
SELECT li.description, li.lineitemFROM purchaseorder T,
        XMLTable('$p/PurchaseOrder/LineItems/LineItem'
    PASSING T.X AS "p"
    COLUMNS lineitem      NUMBER        PATH    '@ItemNumber',
            description   VARCHAR2(30)  PATH    'Description',
            partid        NUMBER        PATH    'Part/@Id',
            unitprice     NUMBER        PATH    'Part/@UnitPrice',
            quantity      NUMBER        PATH    'Part/@Quantity') li
  WHERE li.unitprice > 30 and li.quantity < 20);
```

XMLTable()構成メンバーを効率的に処理するには、XMLTable句でのXQueryの使用はストレージまたは索引に適したものにする必要があります。これにより、ネイティブのXQuery/SQL/XML最適化で、基礎となるXMLストレージおよび索引モデルを活用して最適なクエリ計画を見つけることができます。purchaseorder列がバイナリXMLを使用して保存されている場合、XMLIndexに属する基礎となるリレーショナル表は、結果のクエリ計画で直接アクセスされます。

マルチレベルの階層を横断するには、XMLTableをチェーニング形式で使用できます。

XQueryガイドライン4：GROUP BYおよびORDER BYにXMLCast()構成メンバーとXMLTable()構成メンバーを使用する

SQLスカラー・タイプを操作するGROUP BY句とORDER BY句があります。GROUP BYおよびORDER BYの目的でXQueryの結果をSQLスカラー・タイプにキャストする一般的な方法の1つを次の例に示します。

例7：Using XMLCast() in GROUP BY / ORDER BY

```
SELECT XMLCAST(XMLQUERY('$p/PurchaseOrder/@poDate' PASSING T.X
                                RETURNING CONTENT) AS DATE), COUNT(*)
FROM purchaseorder T
WHERE …
GROUP BY XMLCAST(XMLQUERY('$p/PurchaseOrder/@poDate' PASSING T.X
                RETURNING CONTENT) AS DATE)
ORDER BY XMLCAST(XMLQUERY('$p/PurchaseOrder/@poDate' PASSING T.X
                RETURNING CONTENT) AS DATE);
```

グループ化または順序付けする必要があるスカラー値が複数ある場合は、以下に示すように、順序付けまたはグループ化するすべての列を射影するXMLTable構成メンバーを使用してそれを書き込むことをお勧めします。

例8：GROUP BY / ORDER BYへのXMLTable()構成メンバーの使用

```
SELECT po.DATE, po.poZip, count(*) FROM purchaseorder T,
        XMLTable('$p/PurchaseOrder'
                PASSING T.X AS "p"
                COLUMNS
                        poDate          DATE            PATH '@poDate',
                        poZip           VARCHAR2(8)     PATH 'shipAddress/zipCode',
                ) po
WHERE ….
GROUP BY po.poDate, po.poZip
ORDER BY po.poDate, po.poZip
```

このケースで、purchaseOrder.X列が構造化xmlindexを持つバイナリXMLストレージを使用する場合、クエリ計画は、XMLストレージまたはxmlindexの基礎となるリレーショナル・ストレージ表の列のgroup byおよびorder byを直接使用します。

SQL/XMLのXMLTable使用パターンは、XMLに対してリレーショナル・ビューを作成するためにユーザーによって非常に一般的に採用されており、XMLクエリを既存のリレーショナル・アプリケーション（BIアプリケーションなど）とスムーズに統合できることに注意してください。

XQueryガイドライン5：XQuery拡張式を使用してXQueryの機能評価を示す

XQueryは、XMLの検索と変換の両方を融合した言語です。WHERE句で検索に使用されるXQueryは、基礎となるXMLストレージおよび索引付けモデルを活用するXQueryリライト最適化により適していますが、SELECT句で変換に使用されるXQueryはよりプロシージャ中心であるため、機能評価に適している可能性があります。以下の例に示すように、XQuery拡張式（#ora:xq_proc #）を使用して、XQueryを機能的に評価する必要があることを示すことができます。

例9：機能評価のためのXQuery拡張式

```
SELECT XMLQUERY('(#ora:xq_proc #){…}' PASSING T.X RETURNING CONTENT)
FROM purchaseorder T
WHERE XMLEXISTS('$p/PurchaseOrder/LineItems/LineItem/Part[@Id="717951"]'
                        PASSING T.X AS "p");
```

(#ora:xq_proc#){...}は、"プラグマ"として機能するXQuery拡張式で、中かっこで囲まれたxquery式が関数的に評価される必要があることを示します。Oracle 11gR2 11.2.0.2以降のリリースで利用可能です。

このメカニズムは、/*+ NO_XML_QUERY_REWRITE */ SQLヒントを使用するよりもきめ細かく、したがってさらに柔軟性があります。これはSQL文内で使用されるすべてのXQueryに機能評価を使用することを要求します。これは、SQL文のXMLExists()で使用されるXQueryには望ましくない場合があります。

XQueryガイドライン6：PL/SQLでXQueryを使用してPL/SQL XMLType変数を操作する

PL/SQL XMLTypeメソッドは、XQueryの呼び出しを直接サポートしません。ただし、次の例に示すように、XQueryを使用してSQL/XML関数を呼び出して、XMLType PL/SQL変数をクエリすることができます。PL/SQL XMLType変数値は索引付けされていないため、/*+ NO_XML_QUERY_REWRITE*/ SQLヒントを使用してXQueryを機能的に評価します。

例10：XMLQuery()およびXMLCast()を使用したPL/SQL XMLType変数のクエリ

```
DECLARE
        v_x XMLType;
        NumAcc NUMBER;
BEGIN
         v_x := XMLType(xmlfile(…)); /* xmltype変数を初期化 */
        SELECT /*+ NO_XML_QUERY_REWRITE */
              XMLCAST(XMLQUERY('declare default element namespace
                        "http://custacc";for $cust in $cadoc/Customer return
                        fn:count($cust/Addresses/Address)'
              PASSING v_x AS "cadoc" RETURNING CONTENT) AS NUMBER)
        INTO NumAcc
        FROM DUAL;
END;
```

例11：XMLExists()を使用したPL/SQL XMLType変数のクエリ

```
DECLARE
        v_x XMLType;
        ex NUMBER;
BEGIN
         v_x := XMLType(xmlfile(…)); /* xmltype変数を初期化 */
        SELECT /*+ NO_XML_QUERY_REWRITE */
              CASE WHEN XMLEXISTS('declare default element namespace
                        "http://custacc"; $cadoc/Customer/Addresses/Address)'
              PASSING v_x AS "cadoc")
              THEN 1 ELSE 0 END
        INTO ex
        FROM DUAL;
END;
```

XQueryガイドライン7：適切なXQuery型指定とSQL型指定を使用する

XQuery型システムは、XMLスキーマ型システムに基づいています。XQuery型システムとSQL型システムは正確に一致していませんが、次の表に示すように、各システムの型間には同等のマッピングがあります。xs:date、xs:time、xs:dateTimeにはオプションのタイムゾーン・コンポーネントがあるため、これらは'TIMESTAMP WITH TIMEZONE' SQL型にマップされることに注意してください。タイムゾーン・コンポーネントが使用されていない場合は、DATEまたはTIMESTAMP SQL型にマップできます。

表2：XMLIndexのXMLデータタイプとSQLデータタイプの対応

XMLデータタイプ	SQLデータタイプ
xs:integer,xs:decimal	INTEGER or NUMBER
xs:double	BINARY_DOUBLE
xs:float	BINARY_FLOAT
xs:date	DATE, TIMESTAMP WITH TIMEZONE
xs:time, xs:dateTime	TIMESTAMP, TIMESTAMP WITH TIMEZONE
xs:dayTimeDuration	INTERVAL DAY TO SECOND
xs:yearMonthDuration	INTERVAL YEAR TO MONTH

適切な型認識比較セマンティクスと適切なXML索引の使用を確実にするために、XMLExists()句内で使用されるXQueryでこれらの型を適切にキャストすることをお勧めします。これは次の例のようになります。

例12：XQuery型キャストとSQL型キャストを使用して適切に型指定された値をXMLExists()に渡す

```
SELECT … FROM purchaseOrder T
WHERE XMLEXISTS('$po/purchaseOrder[@id=$id]'
                        PASSING T.X AS "po", CAST(:1 AS NUMBER) as "id" );
```

この例では、SQLバインド変数:1をSQL NUMBER型として明示的にキャストし、それをXMLExists()演算子のXQuery外部変数"$id"にバインドします。

purchaseOrder文書が非XMLスキーマ・ベースの場合、@idのタイプはxs:untypedAtomicです。XQueryの一般的な比較規則では、xs:untypedAtomic値と任意の数値型値（xs:integer、xs:decimal、xs:float、xs:double）の比較は、両方のオペランドをxs:doubleに昇格することで行うと規定しています。これにより、XQueryの@id比較では、SQLバインド変数がxs:decimal型の値として渡されたとしてもxs:double()比較が使用され、内部的にはxs:double型の値にキャストされます。

一方、purchaseOrder文書がXMLスキーマ・ベースの場合、@idはxs:untypedAtomic型ではなく、XMLスキーマで指定された型になります。たとえば、XMLスキーマで@idの型がxs:decimalであると示されている場合、XQueryでの@id比較ではxs:decimal()比較が使用され、xs:decimal型の値として渡されるSQLバインド変数は、内部的にxs:double型の値にキャストされる必要はなくなります。

xs:decimalは正確な数値型用であり、xs:doubleは近似数値型用であることを念頭に置いて、アプリケーションのユーザーは、アプリケーションがどのような型付き比較を必要とするかを決定する必要があります。決定したら、上記の"例12：XQuery型キャストとSQL型キャストを使用して適切に型指定された値をXMLExists()に渡す"クエリを、明示的なXQuery型キャストを使用して、"例13：xs:decimal()型の正確な数値比較の使用"または"例14：xs:double()型の近似数値比較の使用"に従って記述し、xs:decmal()型付き比較またはxs:double()型付き比較のいずれかを取得します。

例13：xs:decimal()型の正確な数値比較の使用

```
SELECT … FROM purchaseOrder T
WHERE XMLEXISTS('$po/purchaseOrder[xs:decimal(@id)=$id]'
                                PASSING T.X AS "po", CAST(:1 AS NUMBER) as "id" );
```

例14：xs:double()型の近似数値比較の使用

```
SELECT … FROM purchaseOrder T
WHERE XMLEXISTS('$po/purchaseOrder[xs:double(@id)=$id]'
                                PASSING T.X AS "po", CAST(:1 AS BINARY_DOUBLE) as "id" );
```

表に保存されているXMLType文書がXMLスキーマ・ベースであるかどうかに関係なく、XQueryが適切な型付き値比較を確実に使用するためには、明示的な型キャストを使用する必要があります。さらに、そうすることでXMLindexの使用が促進されます。

"例13：xs:decmal()型の正確な数値比較の使用"を実行するには、構造化XMLIndexを使用し、"/purchaseOrder/@id"をSQL NUMBER型として索引付けする必要があります。

"例14：xs:double()型の近似数値比較の使用"を実行するには、構造化XMLIndexを使用し、"/purchaseOrder/@id"をSQL TO_BINARY_DOUBLE型として索引付けする必要があります。

数値以外のデータタイプの場合、XQueryの一般的な比較により、xs:untypedAtomic型指定された値を他の値の型にキャストできるため、以下のxs:date()とxs:dateTime()の比較の2つの例に示すように、渡すパラメータにXQuery型キャストを適用するだけで済みます。

例15：日付データタイプの比較でのxs:date()の使用

```
SELECT … FROM purchaseOrder T
WHERE XMLEXISTS('$po/purchaseOrder[@podate =xs:date($d)]'
                                PASSING T.X AS "po", :1 as "d" );
```

ここで、:1は、値（たとえば'2008-07-08'）のSQL varcharにバインドされると予期されます。

例16：タイムスタンプとタイムゾーンのデータタイプの比較でのxs:dateTime()の使用

```
SELECT … FROM purchaseOrder T
WHERE XMLEXISTS('$po/purchaseOrder[@podate =xs:dateTime($d)]'

                                PASSING T.X AS "po", :1 as "d" );
```

ここで、:1は、値（たとえば''2010-01-01T12:00:00Z'）のSQL varcharにバインドされると予期されます。

XQueryガイドライン8：XML索引を使用して最適化できないXQuery式

これらの式は通常、基礎となるXMLストレージや索引構造を活用できないため、大きなサイズのXML文書を処理する場合はパフォーマンス・オーバーヘッドを追加する可能性があります。非常に大きなXML文書をクエリする場合は、そのような式は避けてください。これらは表3に示しています。

表3：サイズの大きいドキュメントで避けるべき式

避けるべき式
次のXPathステップ軸を使用するXQuery式は避けてください。 • ancestor • ancestor-or-self • descendant-or-self • following • following-sibling • parent • preceding • preceding-sibling
<<、>>式は避けてください。

XQueryガイドライン9：適切なXQuery式を使用して上位XQuery内のデータにアクセスする

ピュアXQueryユーザーは、個別のSQL/XML演算子を使用せずにXQueryを作成することを好みます。Oracle XML DBは、ユーザーが次のいずれかを使用してXQuery全体を1つのSQL SELECT文にラップできるようにすることで、このタイプの使用法をサポートします。

*SELECT * FROM XMLTABLE('…') ;*

または

SELECT XMLQuery('…') FROM DUAL;

XQueryの結果がシーケンスとして使用されるか、1つのXML断片として使用されるかによって異なります。SQLがここでは純粋にラップ・メカニズムとして使用されているため、これは"最上位XQuery"と呼ばれます。

11gR2 11.2.0.2リリースより前は、関数fn:collection()およびfn:doc()をora:view()に置き換える必要がありました。

11gR2 11.2.0.2リリースでは、fn:collection()またはfn:doc()を使用して、XMLType表やXMLType列に保存されているXML文書、または純粋なリレーショナル表から仮想的に生成されたXML文書を均一に参照できます。ただし、適切なoradb接頭辞が付いたURLまたはXQuery拡張式を使用する必要があります。以下に例を示します。

最上位XQuery文には、通常のSQL/XML文と同じXQueryリライト最適化が適用されます。SQL文のEXPLAIN PLANを使用してパフォーマンス・チューニングを行うのと同じように、最上位XQuery文のパフォーマンス・チューニングを行うにもEXPLAIN PLANを使用する必要があります。

- ora:view()を使用して、リレーショナル表の内容を仮想XML文書のコレクションとしてマップします。

```
SELECT *
FROM XMLTABLE(
   'for $i in ora:view("SCOTT", "EMP")
     where $i/ROW[EMPNO = 7369 and HIREDATE=xs:date("1980-12-17")]
       return $i' );
```

ここで、EMPはユーザー"SCOTT"が所有するリレーショナル表です。

11gR2 11.2.0.2リリース以降では、次に示すようにfn:collection()を使用することもできます。

```
SELECT * FROM XMLTABLE(
              'for $i in fn:collection("oradb:/SCOTT/EMP")
               where $i/ROW[EMPNO = 7369 and HIREDATE=xs:date("1980-12-17")]
          return $i' );
```

- ora:view()を使用して、XMLType表の内容をXML文書のコレクションとしてマップします。

```
SELECT * FROM XMLTABLE(
          'for $i in ora:view("PO", "PURCHASEORDER")
          where $i/PurchaseOrder/Id = xs:decimal(789645)
          return $i/PurchaseOrder/LineItems/LineItem[itemName="TV"]')
```

ここで、PURCHASEORDERは、ユーザーPOが所有するXMLType表です。

11gR2 11.2.0.2リリースでは、次に示すようにfn:collection()を使用することもできます。

```
SELECT * FROM XMLTABLE(
          'for $i in fn:collection("oradb:/PO/PURCHASEORDER")
          where $i/PurchaseOrder/Id = xs:decimal(789645)
          return $i/PurchaseOrder/LineItems/LineItem[itemName="TV"]')
```

- fn:collection()を使用して、表のXMLType列をXML文書のコレクションとしてマップします。

  ここで、PURCHASEORDERはユーザーPOが所有するリレーショナル表であり、XMLType列'X'があります。これは、11gR2 11.2.0.2リリース以降で利用可能です。

```
SELECT * FROM XMLTABLE(
        'for $i in fn:collection("oradb:/PO/PURCHASEORDER/ROW/X")
        where $i/PurchaseOrder/Id = xs:decimal(789645)
        return $i/PurchaseOrder/LineItems/LineItem[itemName="TV"]')
```

- ハードコーディングされた検索値を定数としてTop-XQueryに渡すことを避けるために、次の例に示すようにPASSINGバインド変数パラメータを使用できます。

  例17：バインド変数の受け渡し

```
SELECT * FROM XMLTABLE(
                'for $i in fn:collection("oradb:/SCOTT/EMP")
                 where $i/ROW[EMPNO = xs:decimal($empno)]
               return $i'
    PASSING :1 as "empno")
```

XQueryガイドライン10：統計を収集する

よくある問題の1つは、ユーザーが表の統計を収集するのを忘れることです。統計が不正確だと、不適切な実行計画となる可能性があります。したがって、次に示すように、XMLType表および関連索引に関する統計の収集を定期的に実行することをお勧めします。

データが一度ロードされ、数回クエリが実行されるようなユースケースでは、データがロードされた後に、影響を受ける表に対してdbms_stats.gather_table_stats()を実行すれば十分です（以下に概要を示します）。データがかなり頻繁にロードされたり更新されたりするユースケースでは、dbms_stats.gather_schema_stats()またはdbms_stats.gather_table_statsをバックグラウンド・スケジューラー・ジョブ（dbms_schedulerパッケージ）として実行するのが最適です（以下に概要を示します）。gather_table_statsのデフォルトの動作では、統計の収集が表上のすべての索引に伝播されることに注意してください。

- XMLIndexの場合、元表の統計を収集すると、構造化XMLIndex表の統計も自動的に収集されます。したがって、XMLIndexの統計を個別に収集する必要はありません。
- Text索引の場合、元表で統計を収集すると、Text索引表の統計も自動的に収集されます。したがって、Text索引の統計を別途収集する必要はありません。

Oracle 11.2.0.3以降、binary-doubleを2次索引として使用するXML索引が存在する場合、適切な2次索引を選択するためにoptimizer_dynamic_samplingを3に設定することをお勧めします。
たとえば、次の2つのコマンド・スクリプトを使用して、スキーマの統計を収集できます。

```
alter session set optimizer_dynamic_sampling = 3;
exec dbms_stats.gather_schema_stats('USERNAME');
```

XQueryガイドライン11：SET XMLOPT[IMIZATIONCHECK]またはイベントを使用して、クエリ/DMLがリライトされない理由を特定します

クエリのチューニングをすると、SQLのパフォーマンスが向上するのと同じように、XQueryのパフォーマンスも向上します。XMLストレージに適切な索引を選択することで、XQueryのパフォーマンスをチューニングします。一般的なデータベース・クエリと同じように、クエリの実行計画を調べて、チューニングが必要かどうかを判断できます。

一般に、クエリ・パフォーマンスを理解してチューニングするには、SQL文（SQL文にラップされた最上位XQueryを含む）でEXPLAIN PLANを使用します。特に、EXPLAIN PLANに'COLLECTION ITERATOR'と表示される場合、通常、クエリ計画が完全に最適化されていないことを示します。

上級ユーザーは次のものを使用できます：

- XMLOPT[IMIZATIONCHECK] [ON|OFF]メカニズム（Oracle 11gR2リリース11.2.0.2の場合）、またはレベル4096 (0x1000)のイベント19021（11.2.0.2より前のリリースの場合）を使用して、トレース・ファイル内の最適化されリライトされたクエリを取得し、XMLストレージおよび索引モデル用に作成された基礎になる内部表でどのような基礎になるクエリが実行されるかを確認します。
- レベル8192 (0x2000)のイベント19027を使用して、特定の式がリライトされない理由を示すダンプをトレース・ファイルに取得します。

Oracle 11gR2 11.2.0.2以降のリリースの場合：

Oracle 11gR2 11.2.0.2リリース以降では、"SET XMLOPT[IMIZATIONCHECK] [ON|OFF]"メカニズムを使用して、クエリの一部が最適化されていないかどうかを確認することをお勧めします。ONにすると、完全に最適化されたXMLクエリまたはXML操作のみが実行されます。最適ではないXMLクエリまたはDML操作は中断され、次のエラー・メッセージが表示されます。"ORA 19022 - 最適化されていないXML構成メンバーが検出されました"。さらに、クエリまたはDMLが最適ではない理由がトレース・ファイルに出力されます。OFFでは、完全に最適化されたXMLクエリ/DML操作のみが実行されるとは保証されません。このコマンドのデフォルトのオプションはOFFです。XMLOPT[IMIZATIONCHECK] ONは、パフォーマンス・チューニングのためにクエリ/DML操作を開発またはデバッグする場合にのみ使用してください。

Oracle 11gR2 11.2.0.2より前のリリースの場合：

11gR2 11.2.0.2リリースより前のリリースを使用している場合は、SQL文ALTER SESSIONを使用して特定のデータベース・セッションに対してレベル1のイベント19021を設定し、XML操作がリライトされたかどうかを判断できます。XML関数のいずれかがリライトされず、機能的に評価されるときにはいつでもエラーが発生するようにしたい場合は、レベル1のイベント19021をオンにします。このような関数を実行すると、"ORA-19022 - XML XPath functions are disabled"というエラーが発生します。

XQueryガイドライン12：クライアント・プログラムでxmltypeのリソースを適切に解放する

XMLTypeの結果をJDBCプログラムでフェッチする場合、XMLTypeの結果を追跡するためにサーバーによって割り当てられたリソースを解放するために、XMLTypeの結果が使用されたら必ずXMLTypeの結果のclose()メソッドを呼び出してください。次のJDBCコード部分は、XMLTypeの結果に対するclose()メソッドの呼び出しを示しています。

例18：close()メソッドを使用してJDBC内のリソースを解放する

```
XMLType xml2;
while (rset.next())
{
        xml2 = XMLType.createXML(rset.getOPAQUE(1));
        System.out.println("Result: " + xml2.getStringVal());
        xml2.close(); // free the XMLType result tracked by the Server
}
rset.close();
```

XQueryガイドライン13：クライアント・プログラムでxmltypeに対してgetObjectを複数回呼び出さないようにする

JDBCプログラムでは、getObject()を複数回呼び出すことは避けてください。XMLTypeオブジェクトは参照カウントされるため、getObject()を呼び出すたびに参照カウントが1ずつ増加します。XMLTypeのclose()メソッドを呼び出すと、参照カウントが1のときにオブジェクトが解放されます。

例19：getObject()を2回呼び出すことを避ける

そのようにせず、次のようにします。

```
Object res = rset.getObject(j);

 if( res instanceof XMLType)
 {
            xml = (XMLType)rset.getObject(j);
 }
```

We shall do

```
Object res = rset.getObject(j); if( res instanceof XMLType)
 {
        xml = (XMLType)res;
 }
```

ストレージ依存のパフォーマンス・チューニング

Oracle XML DBは、メモリ内にXML文書を構成することなくXML文書に対してXPath式を評価することにより、XMLストレージと索引に基づいて、論理的なリライト最適化を実行し、それに続いて物理的なリライト最適化を実行することを思い出してください。この最適化は、XPathリライト最適化と呼ばれます。これはXMLクエリ最適化の適切なサブセットであり、XPath式ではないFLWOR式などのXQuery式の最適化も含まれます。

XPathリライトには、XMLIndex最適化、バイナリXMLのストリーム評価、およびリレーショナル・データに対するXMLTypeビューの場合の基礎となるオブジェクト・リレーショナル構造またはリレーショナル構造へのリライトが含まれます。

XPathリライトは、次のコンテキスト（またはその組合せ）で行うことができます。

- XMLTypeビューがリレーショナル・データに基づいて構築されている場合。
- XMLIndex索引を使用している場合。
- XMLTypeデータがバイナリXMLとして保存されており、ストリーム評価を使用している場合。

これらすべての項目については、次のサブセクションで説明します。

バイナリXML

バイナリXMLストレージは主に非構造化データに使用されます。標準のデータベース索引（Bツリー、ビットマップ）は、通常、XML文書の特定の部分にアクセスするのには役に立ちません。XMLIndexは、XMLデータの内部構造に索引付けする、一般的なXML固有の索引を提供します。

その主な目的の1つは、バイナリXMLストレージにより生じる索引付け作成の制限を克服することです。クエリで索引を使用できない場合でも、ストリームXPath評価を使用してクエリを最適化できる場合があります。このセクションには、作成する索引と、ストリームXpath評価を使用するクエリの作成方法についてのガイダンスがあります。

バイナリXMLストリームの評価

Xpath評価のストリーム・モードは、バイナリXMLに保存されている文書の最も一般的なタイプのXpathを効率的に評価するために使用されます。これは、まずクエリをリライトして関連するXpathをまとめて収集し、文書の1回のパスで評価できるようにすることで実行されます。このタイプのリライトは、'EXPLAIN PLAN'の出力に'XPATH EVALUATION'として反映されます。たとえば、次のクエリの計画では、SELECTリストとWHERE句のXpathが収集され、'XPATH EVALUATION'ステップの列としてまとめて評価されます。これは、/PurchaseOrder/Referenceに対応する列を参照する条件情報セクションに反映されます。

例20：クエリ計画でのXpath評価

```
SELECT XMLCAST(XMLQuery('$p/PurchaseOrder/@poDate'
                        PASSING OBJECT_VALUE AS "p" RETURNING CONTENT) as DATE)
FROM purchaseorder
WHERE XMLExists('$p/PurchaseOrder[Reference="123456"]'
                        PASSING OBJECT_VALUE AS "p");
```

```
| ID    |  Operation                  | 名前                | Rows    |
|     0 |  SELECT STATEMENT           |                     |       1 |
|     1 |    NESTED LOOPS             |                     |       1 |
|     2 |      TABLE ACCESS FULL      | PURCHASEORDER       |       1 |
|*    3 |      XPATH EVALUATION       |                     |         |
```

条件情報（オペレーションIDで識別）：

```
   3 - filter("P"."C_01$"='123456')
```

次のクエリでは、XMLTableのすべての列が'XPATH EVALUATION'ステップの一部としてまとめて評価されます。

例21：XMLTableクエリのストリームXpath評価

```
SELECT li.description, li.lineitem
FROM
   purchaseorder T,
   XMLTable('$p/PurchaseOrder/LineItems/LineItem'
         PASSING OBJECT_VALUE AS "p"
         COLUMNS      lineitem        NUMBER          PATH '@ItemNumber',
                      description      VARCHAR2(30)  PATH 'Description',
                      partid          NUMBER          PATH 'Part/@Id',
                      unitprice       NUMBER          PATH 'Part/@UnitPrice',
                      quantity        NUMBER          PATH 'Part/@Quantity') li
WHERE li.unitprice > 30 and li.quantity < 20;
```

```
| Id | Operation                    | Name             | Rows    |
|      0 | SELECT STATEMENT       |                  |       1 |
|      1 |    NESTED LOOPS        |                  |       1 |
|      2 |       TABLE ACCESS FULL  | PURCHASEORDER |       1 |
|*     3 |       XPATH EVALUATION  |                  |         |
条件情報（オペレーションIDで識別）：

   3 – filter(CAST("P"."C_01$" AS NUMBER)>30 AND
                  CAST("P"."C_02$" AS NUMBER)<20)
```

一般に、子軸や子孫軸を含むXpathはこのモードで評価できますが、逆方向軸（祖先軸など）を含むXpathは評価できません。Xpathのほとんどの位置ベースの条件は、11gR2のストリーム・モードで評価されます。11gR2より前のリリースでは、位置条件を含むXpathはこのモードでは評価できません。すべてのリリースで、last()を含む条件を持つXpathや、同じステップ内に位置ベースの条件と非位置ベースの条件を含むXpathは、これらのXpathがストリーム・モードで評価されないため、避けるべきです。

ストリームXpath評価から最良の結果を得るためのクエリの作成方法に関するいくつかのガイドラインを示します。

ストリーム評価ガイドライン1：可能であればXpath逆方向軸を順方向軸に変換する

多くの場合、逆方向軸を使用するXpathを、逆方向軸を使用しない同等のXpath（つまり、順方向軸のみを使用する）に変換するのは簡単です。たとえば、次のクエリでは、親軸（'..'ステップ)を使用して、名前が'a'という子と、値が'abc1'である属性IDを持つノードを選択します。以下に示すように、条件に（別個のパス・ステップとしてではなく）'a'を含めることで、親軸を使用しない同等のクエリにリライトできます。

例22：逆方向軸から順方向軸への変換

```
-- 逆方向軸でのクエリ（ストリーム・モードでは評価できない）
SELECT XMLQuery('$p/PurchaseOrder/*/a[@id="abc1"]/..'
                        PASSING OBJECT_VALUE AS "p" RETURNING CONTENT)
FROM purchaseorder
WHERE XMLExists('$p/PurchaseOrder[Reference="123456"]'
                                PASSING OBJECT_VALUE AS "p");

-- 逆方向軸を使用しない同等のクエリ（ストリーム・モードでは評価できない）
SELECT XMLQuery('$p/PurchaseOrder/*[a/@id="abc1"]'
                        PASSING OBJECT_VALUE AS "p" RETURNING CONTENT)
FROM purchaseorder
WHERE XMLExists('$p/PurchaseOrder[Reference="123456"]'
                              PASSING OBJECT_VALUE AS "p");
```

ストリーム評価ガイドライン2：大きな文書の場合、正確な（名前付き）パス・ステップを使用できるのであれば子孫軸とワイルド・カードは避ける

子孫軸とワイルド・カードを含むXpathはストリーム・モードで評価できますが、子軸と名前付きパス・ステップのみを使用する場合ほど効率的ではありません。たとえば、特定の発注書の数量が5より大きいすべての明細項目を取得するには、//LineItemの代わりに/PurchaseOrder/LineItemを、以下に示すように使用します。

例23：子軸の回避

```
-- Query with descendant axis
SELECT XMLQuery('$p//LineItem[@quantity > 5]'
                PASSING OBJECT_VALUE AS "p" RETURNING CONTENT)
FROM purchaseorder
WHERE XMLExists('$p/PurchaseOrder[Reference="123456"]'
                  PASSING OBJECT_VALUE AS "p")

-- Query with named path steps (avoiding descendant axis)
SELECT XMLQuery('$p/PurchaseOrder/LineItem[@quantity > 5]'
                PASSING OBJECT_VALUE AS "p" RETURNING CONTENT)
FROM purchaseorder
WHERE XMLExists('$p/PurchaseOrder[Reference="123456"]'
                PASSING OBJECT_VALUE AS "p")
```

ストリーム評価ガイドライン3：DML負荷の高いワークロードの場合は、基礎となるLOB列への書込みのキャッシュを有効にします。

バイナリXML表は、エンコードされたXML文書を保存するために'xmldata'という名前の非表示BLOB列を使用することに注意してください。大量のDMLを伴うワークロードでは、このLOB列への書込みのキャッシュを有効にすることで、影響を受ける文書に対するストリーム評価を使用する後続のクエリを高速化できます。次のSQL文は、purchaseorder表のBLOB列への書込みのキャッシュを有効にします。

例24：バイナリXML表でのDMLのキャッシュの有効化

```
ALTER TABLE purchaseorder modify lob (xmldata) (cache);
```

バイナリXMLの索引付け

上述のとおり、リレーショナルな領域での索引付けソリューションはXMLの索引付けには適していないため、XMLのユースケースには異なる索引のセットを用意しています。サポートされているさまざまな索引を以下に示します。

- XMLIndex構造化コンポーネント、または表ベースの索引。これは略して"構造化XMLIndex"と呼ばれます。
- ConText索引

大量のXMLデータを扱う場合は、オラクルが提供する並列処理およびパーティション化機能の利用を検討できます。基本XMLのパーティション化が範囲パーティション化方法またはリスト・パーティション化方法で実施されている場合、キーワードLOCALを使用して、対応するXMLIndexをXML上に作成できます。これが完了すると、XMLIndexは元表と等分割されます。XMLIndexの各パーティションは、ベースXMLのパーティションと1対1で対応します。XMLIndexパーティション化は、範囲パーティション化またはリスト・パーティション化された表でのみサポートされることに注意してください。

XMLIndex索引を作成または変更するときにPARALLEL句（度数は任意）を使用することで、索引の作成と保守が並列処理されるようにできます。元表がパーティション化されているか、並列処理が有効になっている場合、これにより、DML操作（INSERT、UPDATE、DELETE）および索引DDL操作（CREATE、ALTER、REBUILD）の両方のパフォーマンスが向上します。XMLIndexレベルで指定された並列度（DOP）値も、XMLIndexの各内部表に設定されます。

パス式の条件、FLWOR式のWHERE句、XMLExists()またはXMLTable構成メンバーを持つSQL/XML文のWHERE句は、以下の例で示されます。このような条件評価は、適切なXMLIndexを使用することで大幅に高速化できます。XMLIndexを使用すると、文書間検索（表に保存されているXML文書行のフィルタリング）と文書内検索（表の各行に保存されているXML文書のXML文書フラグメントのフィルタリング）の両方を行うことができます。

例25：XMLIndexを使用できる例

```
/* SQL WHERE句のパス式に条件を含むXMLExists()：

  * 索引を使用して表purchaseOrderの行をフィルタリングできます */
  SELECT XMLQuery('$po/PurchaseOrder/Requestor'

          PASSING OBJECT_VALUE AS "po" RETURNING CONTENT)
FROM purchaseorder
WHERE

XMLExists('$poPurchaseOrder/LineItems/LineItem/Part[@Quantity = 1]'
          PASSING OBJECT_VALUE AS "po");

/* Xquery式のWHERE句。

  * 索引を使用して表purchaseOrderの行をフィルタリングできます */

SELECT *
FROM XMLTABLE(
      'for $po in ora:view("purchaseorder")/PurchaseOrder
      where $po/LineItems/LineItem/@ItemNumber="1"
      return $po/Requestor);
```

```
/* SQL WHERE句のXMLTable列。*/
SELECT li.description, li.lineitem
FROM purchaseorder,
        XMLTable('/PurchaseOrder/LineItems/LineItem'
    PASSING OBJECT_VALUE
    COLUMNS     lineitem        NUMBER          PATH    '@ItemNumber',
                description     VARCHAR2(30)    PATH    'Description',
                partid          NUMBER          PATH    'Part/@Id',
                unitprice       NUMBER          PATH    'Part/@UnitPrice',
                quantity        NUMBER          PATH    'Part/@Quantity') li
WHERE Lineitem = 4567;

/* パス式内の条件。
 * 索引を使用して文書断片を識別できます
 * 索引はpurchaseorderの行の識別には使用できません
 * SQL WHERE句がないためすべての行が返されるからです
 * 索引を使用して説明断片を識別できます
 * これはpurchasorderの各行からのパス条件を満たします
 * このクエリはSQL文のSELECTリストでのスカラー・サブクエリの
 * 使用に似ており、そこではスカラー・サブクエリは
 * 索引を利用できる独自のwhere句を持ちます */

SELECT XMLCAST(XMLQUERY(
    '/PurchaseOrder/LineItems/LineItem[@ItemNumber=1]/Description'
    PASSING object_value RETURNING CONTENT) AS VARCHAR2(4000))
FROM purchaseorder p;
```

ユースケースに応じてどの索引を選択するかに関するいくつかのガイドラインを示します。

索引選択ガイドライン1：Xpathが静的な場合は構造化XMLIndexを使用して条件に応答する

Xpathが事前にわかっている場合は、XMLIndexの構造化コンポーネントがユースケースに最適です。これは、構造化XMLIndexを持つXpathのXQueryでリレーショナル・パフォーマンスを達成するのに役立ちます。

構造化XMLIndexを使用して、条件に表示されるXpathを索引付けすることで、最適なパフォーマンスを実現できます。これらの条件は、"例25：XMLIndexを使用できる例"に示すように、SQL文、where句の条件、またはXquery自体に含めることができます。

"例27：構造化コンポーネントを使用したXMLIndexの作成"に示すように構造化索引を作成すると、上記のすべてのクエリを最適化できます。

索引選択ガイドライン2：全文検索要件にテキスト索引を使用する

アプリケーションに全文検索の要件がある場合は、SQL contains()演算子の使用を検討し、ベースXMLType列にテキスト索引を作成します。

例26：SQL contains()を使用して全文検索を実行する

```
create table po of xmltype;
create index po_otext_ix on po (object_value) indextype is

                    ctxsys.context;
call dbms_stats.gather_table_stats(USER, 'PO');

select distinct
XMLCast(XMLQuery('$p/PurchaseOrder/ShippingInstructions/address'
                    passing po.object_value as "p" returning content)

          as varchar2(256)) "Address"
from po po
where contains(po.object_value, '$(Fortieth) INPATH
                    (PurchaseOrder/ShippingInstructions/address)') > 0;
```

索引選択ガイドライン3：断片抽出

XML断片を射影するクエリが存在する場合、索引付けのアプローチは文書の平均サイズによって異なります。

- データセットが小規模から中規模の文書で構成されている場合は、次のいずれかを使用する必要があります。
    - 1つは、Xquery拡張式（#ora:xq_proc #）を使用して、Xqueryが機能的に評価されることを示します。注：*ora:xq_proc*はきめ細かな制御ができます。条件の*Xpath*が*XMLIndex*から除外されない限り、断片抽出に*xq_proc*を使用させ、条件に*XMLIndex*を使用させることができます。
    - または、バイナリXMLストリーム評価を使用します。

索引選択ガイドライン4：必要に応じてさまざまな索引を組み合わせる

さまざまな索引を組み合わせて使用できます。たとえば、技術文書の表がある場合、タイトル、作成者、および日付フィールドの構造化コンポーネントを含むXMLIndexを作成し、テキスト検索クエリに応答するためのOracle Text索引を作成できます。

ユースケースに適した索引を選択したら、これらの索引から最高のパフォーマンスを得る方法に関するガイドラインについて、対応するセクションを参照してください。

XMLIndex構造化コンポーネント

バイナリXML内のデータが構造化されていない場合でも、予測可能な構造化データのアイランドが含まれていることがあります。例としては、タイトル、作成者、日付のフィールドがある技術文書などがあります。
周囲のデータが比較的構造化されていない場合でも、XMLコンテンツの固定構造化アイランドを射影するクエリには、XMLIndex索引の構造化コンポーネントを作成して使用します。
構造化XMLIndexコンポーネントは、そのようなアイランドをリレーショナル形式で編成します。これはSQL/XML関数のXMLTableに似ており、構造化コンポーネントを定義するために使用する構文は、この類似度を反映します。索引付けデータの保存に使用されるリレーショナル表はデータタイプを認識し、各列は異なるスカラー・データタイプにすることができます。したがって、XMLIndex索引の構造化コンポーネントを作成することは、XMLデータの構造化部分をリレーショナル形式に分解することであると考えることができます。

構造化コンポーネントは対象を絞った索引であるため、索引付けするXpathとそのデータタイプを慎重に指定する必要があります。ただし、このような索引を使用する利点は、静的に既知のXpathを使用するクエリには重要です。

構造化コンポーネントを使用する利点のいくつかを以下に示します。

1. 型認識でリレーショナルスタイルの検索 - XMLIndexの構造化コンポーネントには、値を型別とパス別に異なる列に分離する機能があるため、かなり固有であるリレーショナルスタイルの統計を、XMLIndexを構築できるリレーショナル・コスト・ベース・オプティマイザに提供できます。
2. 複合*B*ツリーおよびビットマップ索引のサポート - 構造化XMLIndexの内部表は、異なるXPathからの値を個別の列に保存できるため、複合Bツリーとビットマップ索引を作成できます。
3. *SQL*条件にサブクエリがない - 構造化XMLIndexが使用される場合、WHERE句の条件は構造化XMLIndex表の列レベルのチェックになります。
4. *BI*スタイル・クエリの索引付け - order-by、group-by、windowなどのSQL構文により、リレーショナル・データに対する強力なビジネス・インテリジェンス・クエリが可能になります。XMLデータ内の値に対してorder-by、group-by、windowなどを使用するアプリケーションは、リレーショナル表の列に対してクエリをorder-by、group-by、windowなどにリライトできるため、構造化XMLIndexを使用することでリレーショナル・パフォーマンスを達成できます。これは次のようにして実現されます。XMLTableを使用すると、XML内の値を仮想表として射影できます。XMLTable関数を使用するクエリは、構造化XMLIndexのリレーショナル表に簡単にアクセスできるようにリライトできます。これは、仮想表の列を操作するorder-by、group-by、windowなどが、構造化XMLIndex表の対応する物理列を操作するorder-by、group-by、windowなどに変換されることを意味します。

次の例は、構造化XMLIndexを作成する方法を示しています。これは、"LineItem"と呼ばれるコレクションを持つPurchase Orderスキーマを使用します。行パターン/PurchaseOrder/LineItems/LineItemに一致するXMLノードごとに、このXMLIndexはリレーショナル索引表に5列を射影します。これらのノードの値は、相対XPathの@ItemNumber、Description、Part/@Id、Part/@UnitPrice、およびPart/@Quantityに一致するノードの値です。内部索引表には、XML文書ごとに、文書内のLineItemノードの数と同じ数の行が含まれます。索引DDLは、表の名前（この場合はlineitem_tab）、5つの列の名前、およびこれら5つの列のSQLデータタイプを指定します。

例27：構造化コンポーネントを使用したXMLIndexの作成

```
CREATE INDEX po_struct ON purchaseorder (OBJECT_VALUE)
INDEXTYPE IS XDB.XMLIndex
PARAMETERS (
```

'XMLTable lineitem_tab ''/PurchaseOrder/LineItems/LineItem''

```
        COLUMNS  lineitem     NUMBER          PATH    '@ItemNumber',
                 description  VARCHAR2(30)    PATH    'Description',
                 partid       NUMBER          PATH    'Part/@Id',
                 unitprice    NUMBER          PATH    'Part/@UnitPrice',
                 quantity     NUMBER          PATH    'Part/@Quantity') li
```

以下に、構造化XMLIndexで最高のパフォーマンスを実現する方法に関するガイドラインを示します。

構造化索引ガイドライン1：複数の関数索引や仮想列の代わりに構造化索引を使用する

XMLの複数のリレーショナル・キー列を射影して、それらの列で迅速な検索を行えるようにBツリー索引を構築するXMLユースケースでは、構造化XMLIndexが最適です。構造化XMLIndexは、効率の悪い複数の仮想列（VC）に依存するのではなく、効率的な検索のためにすべての主要なリレーショナル列をキャプチャする1つのリレーショナル表を射影します。

これらの構造化XMLIndex列は、入力ベース文書の1回のスキャンで効率的に移入されます。これは、仮想列では実行できません。構造化XMLIndexベースのアプローチは、XMLがコレクションを持つ場合に機能しますが、VCベースのアプローチは、射影された値がXMLコレクション内にある場合には使用できません。

構造化索引ガイドライン2：索引とクエリのデータタイプを対応させる

XMLIndex構造化コンポーネントに使用されるリレーショナル表は、SQLデータタイプを使用します。クエリで使用されるXQuery式は、XMLデータタイプ（XMLスキーマ・データタイプおよびXQueryデータタイプ）を使用します。XQuery型指定ルールは、一貫性と型チェックを確実にするために、副次式のデータタイプを自動的に変更できます。たとえば、XPath式 /PurchaseOrder/LineItem[@ItemNumber = 25]を使用してクエリされた文書がXMLスキーマ・ベースではない場合、副次式@ItemNumberはxs:untypedAtomicとなり、XQuery =比較演算子によってxs:doubleに自動的にキャストされます。XMLIndex構造化コンポーネントを使用してこのデータを索引付けするには、SQLデータタイプとしてBINARY_DOUBLEを使用する必要があります。

これは一般的なルールです。構造化コンポーネントを含むXMLIndex索引をクエリに適用するには、データタイプが対応している必要があります。ガイドライン7の表2："適切なXQuery型指定とSQL型指定の使用"

関連するXMLデータタイプとSQLデータタイプに1対1対応が組み込まれていない場合は、クエリに対して索引が選択されるように、（表2に従って）それらを対応させる必要があります。これを行うには2とおりの方法があります。

- 索引をクエリに対応させる - XMLデータタイプに対応するように、構造化索引コンポーネント内の列を定義（または再定義）します。たとえば、索引付けするクエリでXMLデータタイプxs:doubleを使用する場合は、対応するSQLデータタイプBINARY_DOUBLEを使用するように索引を定義します。
- クエリを索引に対応させる - クエリ内で、XQuery式の関連部分を、索引コンテンツ表で使用されるSQLデータタイプに対応するデータタイプに明示的にキャストします。

構造化索引ガイドライン3：XMLTableビューを対応する索引（BIスタイルのクエリなど）とともに使用する

XMLIndexの構造化コンポーネントはXMLTableの考え方に基づいて構築されているため、このような索引は、このリレーショナル・パラダイムが適用されるユースケースにうまく適合します。リレーショナル・アクセス・パラダイムを必要とするアプリケーション開発者の場合、XMLTableに基づいて構築された1つ以上のリレーショナル・ビューを作成する必要があります。XMLTable関数は、XML内のキー値をリレーショナル列として公開する方法を提供します。多くのユースケースでのXMLのクエリは、XMLTable関数を使用するリレーショナル・ビューの定義内に隠すことができるので、SQLに精通しているもののXPath/XQueryの複雑さは回避したいアプリケーション開発者の間にXMLは浸透しやすくなります。このような場合、構造化XMLIndex定義はリレーショナル・ビューの定義と一致します。

以下の例は、XMLTable()がXMLに対するリレーショナル表の抽象化をどのように提供するかを示します。次の例は、対応するビューを作成する方法を示し、例27はそれに対応する索引を示します。

例28：XMLTableによるXMLに対する仮想表の抽象化の提供

```
SELECT    lines.lineitem ,
          lines.description,
          lines.partid ,
          lines.unitprice ,
          lines.quantity
FROM      purchaseorder,
          XMLTable('/PurchaseOrder/LineItems/LineItem'
          PASSING OBJECT_VALUE
          COLUMNS  lineitem          NUMBER         PATH    '@ItemNumber',
          description  VARCHAR2(30)    PATH     'Description',
          partid       NUMBER          PATH     'Part/@Id',
          unitprice    NUMBER          PATH     'Part/@UnitPrice',
          quantity     NUMBER          PATH     'Part/@Quantity') lines;

LINEITEM  DESCRIPTION  PARTID  UNITPRICE  QUANTITY
--------  -----------  -------  ---------  --------
11    Orphic Trilogy   37429148327   80  3
22    Dreyer Box Set   37429158425   80  4
11    Dreyer Box Set   37429158425   80  3
```

例29：XMLTableを使用したリレーショナル・ビュー、および対応する構造化XMLIndex

```
CREATE VIEW lineitems_v
(lineitem, description, partid, unitprice, quantity)
AS SELECT
          lines.lineitem, lines.description, lines.partid,
          lines.unitprice, lines.quantity
FROM purchaseorder,
XMLTable('/PurchaseOrder/LineItems/LineItem'
PASSING OBJECT_VALUE
COLUMNS     lineitem        NUMBER          PATH     '@ItemNumber',
            description     VARCHAR2(30)    PATH     'Description',
            partid          NUMBER          PATH     'Part/@Id',
            unitprice       NUMBER          PATH     'Part/@UnitPrice',
            quantity        NUMBER          PATH     'Part/@Quantity') li
) lines;
```

この一般的な使用例の1つは、BIスタイルのクエリです。order-by、group-by、windowなどのSQL構文により、リレーショナル・データに対する強力なビジネス・インテリジェンス・クエリが可能になります。XMLTableを使用すると、XML内の値を仮想表として射影できます。order-by、group-by、windowなどは、仮想表の列に対して操作できます。構造化XMLIndexは、XMLTableによって公開される仮想表を反映する方法でストレージ表を内部的に編成します。したがって、構造化XMLIndexは、このようなXMLTableベースのクエリを非常に効率的に行う方法でXMLデータに索引付けするのに適しています。XMLTable関数を使用するクエリは、構造化XMLIndexのリレーショナル表に簡単にアクセスできるようにリライトできます。これは、仮想表の

列を操作するorder-by、group-by、windowなどが、構造化XMLIndex表の対応する物理列を操作するorder-by、group-by、windowなどに変換されることを意味します。

XMLTableを使用して、XMLに対してリレーショナル・ビューを作成することをお勧めします。ビューは、関心対象となるすべての列をBIアプリケーションに射影します。アプリケーション・クエリは、これらのリレーショナル・ビューに対して作成する必要があります。構造化XMLIndexがこれらのビューに1対1対応で作成される場合、Oracle RDBMSによって確実に、ビューに対するクエリが構造化XMLIndexのリレーショナル表に対するクエリにシームレスに変換されるので、リレーショナル・パフォーマンスを達成できます。

構造化索引ガイドライン4：2次索引の作成（特に条件用）

"例27：構造化コンポーネントを使用したXMLIndexの作成"では、内部でリレーショナル表lineitem_tabが作成されます。値ベースの検索で良好なパフォーマンスを得るには、索引表に2次索引を作成することが重要です。これは次の例のようになります。

例30：構造化XMLIndex表での2次索引の作成

```
CREATE INDEX li_itemnum_idx     ON lineitem_tab(lineitem); CREATE INDEX
li_desc_idx                     ON lineitem_tab(description); CREATE
INDEX li_partid_idx             ON lineitem_tab(partid); CREATE INDEX
li_uprice_idx                   ON lineitem_tab(unitprice); CREATE INDEX
li_quantity_idx                 ON lineitem_tab(quantity);
```

複合Bツリー索引、ビットマップ索引、およびドメイン索引（Oracle Textなど）も索引表に作成できます。

例31：構造化XMLIndex表でのOracle Text索引の作成

```
CREATE INDEX li_desc_ctx_idx ON lineitem_tab(description)
indextype is ctxsys.context;
```

これらの2次索引を作成するのはユーザーの責任です。構造化XMLIndexコンポーネントに対してシステムが自動的に2次索引を作成することはありません。どの2次索引がニーズに最も適しているかについて最善の判断ができるのはユーザーであるからです。2次索引が作成されたら、オプティマイザが索引を選択できるように、元表の統計を収集する必要があります。

クエリで条件（SQL WHERE句など）において特定のXPathを使用する場合は、構造化XMLIndex表の対応する列に2次索引を作成することを強くお勧めします。

構造化索引ガイドライン5：実行計画をチェックして構造化索引が使用されているかを確認する

クエリを高速化するために必要な索引を作成した後、実行計画が実際に索引を選択していることを確認する必要があります。たとえば、"例27：構造化コンポーネントを使用したXMLIndexの作成"に示すように、構造化XMLIndexを作成し、例30に示すように2次索引を作成したとします。次に、以下の例に示すように、クエリに対してEXPLAIN PLANを実行する必要があります。

例32：EXPLAIN PLANを使用して索引が選択されているかどうかを判別する

```
EXPLAIN PLAN FOR
SELECT XMLCAST(XMLQUERY(  '/PurchaseOrder/Requestor' PASSING object_value RETURNING
CONTENT) AS VARCHAR2(4000))
FROM purchaseorder p
WHERE XMLExists('/PurchaseOrder/LineItems/LineItem[xs:decimal(@ItemNumber)=1]' PASSING
object_value);

Explained.

SQL> select Id, Operation, Name from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------
Plan hash value:2801523227
--------------------------------------------------------------------
| Id | Operation                                | Name                |
--------------------------------------------------------------------
|      0 | SELECT STATEMENT                       |                     |
|      1 |    NESTED LOOPS SEMI                   |                     |
|      2 |       TABLE ACCESS FULL                | PURCHASEORDER       |
|*     3 |       TABLE ACCESS BY INDEX ROWID      | LINEITEM_TAB        |
|*     4 |          INDEX RANGE SCAN              | LI_ITEMNUM_IDX      |
--------------------------------------------------------------------
```

実行計画は、構造化索引ストレージ表を使用するようにクエリがリライトされたことを示しています。LINEITEM_TABと2次索引LI_ITEMNUM_IDX。

構造化索引ガイドライン6：マスター-ディテール・リレーションシップの索引付け

構造化アイランドにマスター-ディテール的なリレーションシップがある場合、構造化XMLIndexは、表間のプライマリ・キー-外部キーの関係を使用して、各構造化アイランドをリレーショナル表としてキャプチャする方法を提供します。このようなマスター-ディテール・ビューの定義と、それに対応する構造化XMLIndexを次に示します。

例33：マスター-ディテール・リレーションシップがあるリレーショナル・ビュー

```
CREATE OR REPLACE VIEW purchaseorder_detail_view AS
        SELECT po.reference, li.*
        FROM purchaseorder p,
        XMLTable('/PurchaseOrder' PASSING p.OBJECT_VALUE

COLUMNS

        reference VARCHAR2(30) PATH 'Reference',
        lineitem XMLType PATH 'LineItems/LineItem') po,
        XMLTable('/LineItem' PASSING po.lineitem

COLUMNS

        itemno          NUMBER(38)      PATH    '@ItemNumber',
        description     VARCHAR2(256)   PATH    'Description',
        partno          VARCHAR2(14)    PATH    'Part/@Id',
        quantity        NUMBER(12, 2)   PATH    'Part/@Quantity',
unitprice       NUMBER(8, 4)    PATH    'Part/@UnitPrice') li;
```

例34：マスター-ディテール・リレーションシップを索引付けする構造化XMLIndex

```
CREATE INDEX po_struct ON po_tab (OBJECT_VALUE)

INDEXTYPE IS XDB.XMLIndex
PARAMETERS ('XMLTable po_ptab
            XMLNAMESPACES(DEFAULT "http://www.example.com/po"),
                "/purchaseOrder"
            COLUMNS orderdate DATE                  PATH "@orderDate",
                    Id          BINARY_DOUBLE PATH "@id",
                    items       XMLType             PATH "items/item" VIRTUAL
            XMLTable li_tab
      XMLNAMESPACES(DEFAULT "http://www.example.com/po"),
                "/item" PASSING items
        COLUMNS partnum     VARCHAR2(15)        PATH    "@partNum",
                description CLOB                PATH    "productName",
                usprice     BINARY_DOUBLE       PATH    "USPrice",
                shipdat     DATE                PATH    "shipDate"');
```

構造化索引ガイドライン7：SELECT句とWHERE句の間で断片抽出と値検索を分割する

断片抽出と値検索に単一のXQueryを使用する代わりに、断片抽出にはSELECT句でXMLQuery()を使用し、値検索にはWHERE句でXMLExists()を使用します。この分離を行うことで、値検索には構造化xmlindexを選択でき、断片抽出にはバイナリXMLストリームを使用できます。

例35：断片抽出と値検索の分割

この例では、次のXMLIndexが存在する場合、クエリ1はクエリ2よりも適切な計算式です。

索引定義：

```
CREATE TABLE XML_TEST (XML_DOC XMLType)
            XMLType XML_DOC STORE AS BINARY XML;

CREATE INDEX XML_TEST_IX ON XML_TEST (XML_DOC) INDEXTYPE
            IS XDB.XMLIndex
PARAMETERS ('GROUP XML_TEST_G XMLTable XML_TEST_X
          XMLNAMESPACES("http://example.com/metadata" as "m"),
          "/m:object" COLUMNS
          TENANT VARCHAR(100) PATH "m:meta/m:tenant",
          ID VARCHAR(250) PATH "m:meta/m:id"');

CREATE INDEX XML_TEST_IX_1 ON XML_TEST_X(TENANT, ID);
```

クエリ1：良い例

```
SELECT
XMLQUERY('declare namespace m="http://example.com/metadata";
                  for $obj in $doc/m:object
                  return <m:object>

                                    {$obj/m:meta/m:id}{$obj/m:meta/m:tenant}
                              </m:object>'
                passing T.XML_DOC as "doc" returning content)
FROM XML_TEST T
WHERE

XMLEXISTS('declare namespace m="http://example.com/metadata";
                   $doc/m:object[m:meta/m:tenant=$tenant
                                              and m:meta/m:id=$id]'
                  passing T.XML_DOC as "doc",
                              'tenant5' as "tenant",
```

```
                    'id_555' as "id");
```

クエリ2：避けるべき例

```
SELECT X.XML_DOC
FROM XML_TEST T,
        XMLTABLE(
          XMLNAMESPACES('http://example.com/metadata' as "m"),
           'for $obj in /m:object
           where $obj/m:meta/m:tenant="tenant5" and
                    $obj/m:meta/m:id="id_5555"
           return <m:object>
                        {$obj/m:meta/m:id}{$obj/m:meta/m:tenant}
                    </m:object>'
          PASSING T.XML_DOC COLUMNS XML_DOC XMLTYPE PATH '.') X;
```

構造化索引ガイドライン8：クエリ結果の順序付けのためにXMLTableとともにSQL ORDER BYを使用する

XQuery ORDER BY句を使用する代わりに、XMLTableを使用して順序付けの基準となるキーを射影し、SQL ORDER BYを使用します。以下の例では、クエリは断片抽出を、値の検索とともに示しています。
断片は、XMLTABLE()句に射影されるテナント、IDによって順序付けされます。

例36：SQL order byの使用

```
SELECT XMLQUERY('declare namespace
        m="http://example.com/metadata";
            for $obj in $doc/m:object
            return <m:object>
                        {$obj/m:meta/m:id} {$obj/m:meta/m:tenant}
                    </m:object>'
            passing T.XML_DOC as "doc" returning content)
FROM XML_TEST T,
    XMLTABLE(XMLNAMESPACES('http://example.com/metadata'
                                    as "m"),
                '$doc/m:object' PASSING T.XML_DOC as "doc"
                 COLUMNS
                        tenant VARCHAR(100) PATH 'm:meta/m:tenant',
                        id VARCHAR(250) PATH 'm:meta/m:id'
          ) tt
WHERE
    XMLEXISTS('declare namespace m="http://example.com/metadata";
            $doc/m:object[m:meta/m:tenant=$tenant]'
            passing T.XML_DOC as "doc", 'tenant5' as "tenant")
ORDER BY tt.tenant, tt.id;
```

Text索引

要素や属性などのXMLノードにアクセスすることに加えて、XMLテキスト・ノード内のテキストの特定のパッセージへの高速アクセスを提供することが重要になる場合があります。これがOracle Text索引の目的であり、全文文字列に索引付けします。全文索引付けは、XML要素とテキスト・ノード・コンテンツが混在することが多い文書中心のアプリケーションに特に役立ちます。全文検索は、多くの場合、構造化XML検索と組み合わせることで、つまり、XPath式を使用して識別されるXML文書の特定の部分に制限することによって、より強力でより焦点を絞ったものにできます。

XMLType列に作成されたOracle TextのCONTEXT索引により、SQL関数contains()が有効になり、XQuery関数ora:contains()でのXMLの全文検索が容易になります。次の例は、XMLType列にOracle Text索引を作成する方法を示しています。

例37：Oracle Text索引の作成

```
CREATE INDEX po_otext_ix ON po_clob (OBJECT_VALUE)
        INDEXTYPE IS CTXSYS.CONTEXT;
Index created.
```

Oracle Textの索引付けは、他のタイプの索引付けとは完全に直交しています。SQL関数contains()またはXPath関数ora:contains()を使用するときは常に、Oracle Text索引を全文検索に使用できます。次の例は、XMLIndex索引とOracle Text索引の両方が同じXMLデータに対して定義されている場合を示しています。Oracle Text索引は、XMLIndexパス表のVALUE列に対して作成されます。

例38：他の索引と併用してのOracle Text索引の使用

```
CREATE INDEX po_otext_ix ON my_path_table (VALUE)
        INDEXTYPE IS CTXSYS.CONTEXT;
Index created.

EXPLAIN PLAN FOR
   SELECT DISTINCT XMLCAST(XMLQUERY(
                '/PurchaseOrder/ShippingInstructions/address' PASSING object_value RETURNING
CONTENT) AS VARCHAR2(4000)) "Address"
      FROM po_clob
      WHERE contains (OBJECT_VALUE, '$(Fortieth) INPATH
                        (PurchaseOrder/ShippingInstructions/address)') > 0;

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------
| Id | Operation                          | Name                           |
-----------------------------------------------------------------------------
|    0 | SELECT STATEMENT                 |                                |
|*   1 |  TABLE ACCESS BY INDEX ROWID     | MY_PATH_TABLE                  |
|*   2 |    INDEX RANGE SCAN              | SYS78942_PO_XMLINDE_ORDKEY_IX  |
|    3 | HASH UNIQUE                      |                                |
|*   4 |   TABLE ACCESS FULL              | PO_CLOB                        |
-----------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("SYS_P0"."PATHID"=HEXTORAW('35EF580A') AND
SYS_XMLI_LOC_ISNODE("SYS_P0"."LOCATOR")=1)
```

```
   2 - access("SYS_P0"."RID"=:B1)
       filter("SYS_P0"."RID"=:B1)
   4 - filter("CTXSYS"."CONTAINS"(SYS_MAKEXML("SYS_ALIAS_1"."XMLDATA"),
     '$(Fortieth) INPATH (PurchaseOrder/ShippingInstructions/address)')>0)
```

上記の例の実行計画は、XMLIndex索引とOracle Text索引の両方を参照しており、両方が使用されていることを示しています。XMLIndex索引は、そのパス表MY_PATH_TABLEと、その順序キー索引SYS78942_PO_XMLINDE_ORDKEY_IXによって示されます。
Oracle Text索引は、条件情報内のSQL関数containsへの参照によって示されます。

xmltypeの全文検索は、SQLでcontains()関数を使用するか、XPathまたはxquery式内でora:contains()を使用することで実行できます。各機能の詳細は以下のとおりです。

contains()を使用したXMLデータの検索

XMLType列に対して、*contains*や*score*などのOracle Text操作を実行できます。containsを実行するには、xmltype列に対してOracle Text索引（ctxsys.context）を作成する必要があります。*contains*演算子はXML名前空間を認識しないことに注意してください。次の例は、SQL関数containsを使用したOracle Text検索を示しています。

例39：SQL関数CONTAINSを使用したXMLデータの検索

```
SELECT DISTINCT XMLCast(XMLQuery('$p/PurchaseOrder/ShippingInstructions/address'
                PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
                AS VARCHAR2(256)) "Address"
  FROM po_clob po
  WHERE contains(po.OBJECT_VALUE,
                      '$(Fortieth) INPATH
          (PurchaseOrder/ShippingInstructions/address)') > 0;

住所
--------------------------------
1200 East Forty Seventh Avenue New York
NY
10024
USA
1 row selected.
```

このクエリの実行計画は、Oracle Text CONTEXT索引が使用される以下の2とおりの方法を示しています。

1. 索引をドメイン索引として明示的に参照します。
2. 条件情報内のSQL関数containsを指します。

```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
| Id | Operation                                | Name            | Rows | Bytes |
-------------------------------------------------------------------------------
|     0 | SELECT STATEMENT                      |                 |      7 | 14098 |
|     1 |  HASH UNIQUE                          |                 |      7 | 14098 |
|     2 |    TABLE ACCESS BY INDEX ROWID        | PO_CLOB         |      7 | 14098 |
|*    3 |      DOMAIN INDEX                     | PO_OTEXT_IX     |        |       |
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("CTXSYS"."CONTAINS"(SYS_MAKEXML('…..... ',523
```

```
                3,"XMLDATA"),'$(Fortieth)  INPATH
            (PurchaseOrder/ShippingInstructions/address)')>0)
```

ora:contains()を使用したXMLデータの検索

XQuery関数のora:contains()を使用すると、ユーザーは特定のXPathまたはxqueryコンテキスト内でキーワードを検索できます。ora:contains()の評価の機能的な実行にOracle Text索引（ctxsys.context）は必要ありませんが、パフォーマンスのために必要になる場合があります。

可能な場合、Oracleは内部でora:contains()演算子をcontains()演算子にリライトします。これは、次の条件が両方とも満たされる場合に起きます。

1. ora:contains()のXPathまたはxqueryコンテキストは、構造化xmlindexのユーザー定義列にリライトできます。
2. 列に対するTRANSACTIONAL Oracle Text索引があります。

上記の条件が両方ともtrueの場合、ora:contains()は列のcontains()にリライトされます。列のOracle Text索引がTRANSACTIONALでない場合、ora:contains()は機能的に評価されます（索引なし）。次の例は、そのような索引を作成する方法を示しています。

例40：ora:contains()を使用したXMLデータの検索

```
create table myemp of xmltype tablespace sysaux;

create index emp_xtidx on myemp (object_value)
        indextype is xdb.xmlindex parameters('
        GROUP gp1
                XMLTABLE ETAB
                XMLNamespaces(DEFAULT "http://www.oracle.com/tkxmsch1.xsd"),
                "/Employee"
                columns "eid" integer PATH "EmployeeId",
                        "fname" varchar2(70) PATH "FirstName",
                        "lname" varchar2(70) PATH "LastName",
                        "jdesc" varchar2(70) PATH "JobDesc"');

create index jdctxidx on ETAB (jdesc)
    indextype is ctxsys.context parameters ('transactional');

select xmlcast(xmlquery('
                    declare default element namespace
                    "http://www.oracle.com/tkxmsch1.xsd";(::)
                    /Employee/FirstName' passing value(e) returning content) as varchar2(50) )
from myemp e
where xmlexists( '
                    declare default element namespace
                    "http://www.oracle.com/tkxmsch1.xsd";(::)
                    /Employee[ora:contains(JobDesc, "program")>0]'
                    passing value(e))
/
```

全文クエリで最高のパフォーマンスを実現するには、以下のガイドラインに従います。

Text索引ガイドライン1：バイナリXMLストレージ：contains()を使用する

ストレージがバイナリXMLの場合は、xmltypeに対してOracle Text索引を作成し、contains()を使用します。これは、バイナリXMLに対する全文検索の推奨アプローチです。ただし、Oracle Text索引はXML名前空間を認識しないことに注意してください。

Text索引ガイドライン2：バイナリXMLストレージ：構造化XMLIndex列でのText索引の作成

ストレージがバイナリXMLの場合には、ガイドライン2が使用できないときにのみ、構造化XMLIndexのユーザー定義列にOracle Text索引を作成することを検討してください。

構造化XMLIndexのユーザー定義列をCLOBとして定義すると、ノード値の切捨てを回避できます。ただし、CLOB列があると、構造化XMLIndexのロード・パフォーマンスに劇的な影響を与えます。

結論

XQuery言語に対するOracle XML DBのサポートは、SQL/XML関数のXMLQuery、XMLTable、XMLExists、およびXMLCastのネイティブ実装を通じて提供されます。本書ではまず、ストレージに依存しないXQueryのベスト・プラクティスについて説明し、次にさまざまなストレージ/索引付けオプションで最高のパフォーマンスを実現するためのガイドラインへと説明が移りました。

付録A：非推奨の主にXPath 1.0ベースの関数と標準SQL/XML XQueryベースの関数との間のセマンティックな相違点

非推奨の構文とXQueryベースの構文の間には重要ないくつかの相違点があり、移行の助けとなるようにそれらの相違点を以下に示しています。

サポートが終了したextract()、existsNode()、table(xmlsequence())、extractValue()では、パス式でXPath 1.0のみを使用できます。SQL/XML標準演算子のXMLQuery()、XMLExists()、XMLTable、XMLCast()は、クエリ式でXQuery 1.0を使用します。この重要な違い以外にも、サポートが終了した演算子には標準ではないいくつかの動作があり、ユーザーは標準ベースの演算子を使用するように移行時には特別な注意を払う必要があります。

- スキーマ・ベースのデータタイプの比較：サポートが終了した演算子がスキーマ・ベースのバイナリXMLType列に適用される場合、スキーマ・ベースのデータタイプ比較セマンティクスが適用されます。たとえば、非文字列型と文字列の比較はキャストとデータタイプ固有の比較となります。ただし、標準の演算子では、**XQuery日付型キャスト関数を使用する必要があります。**そうしない場合は、エラーが発生します。XQueryガイドライン7を参照してください。

  例：

  @podateがxs:date型、@poidがxs:integer型で、purchaseOrderがスキーマ・ベースのpurchaseOrder XML文書インスタンスを保存するXMLType表であると仮定します。

  サポートが終了した構文：

  ```
  Select 1 from purchaseOrder p
  where existsNode(value(p), '/PurchaseOrder[@podate > "1998-09-02"]') = 1
  ```

  標準ベースの構文：

  ```
  Select 1 from purchaseOrder p where xmlexists( 'declare namespace po =
  http://www.po.com;/PurchaseOrder[@podate >xs:date( "1998-09-02")]' passing
  value(p))
  ```

  次のクエリでは型エラーが発生します

  ```
  Select 1 from purchaseOrder p where xmlexists( 'declare namespace po =
  http://www.po.com;/PurchaseOrder[@podate > "1998-09-02"]' passing value(p))
  ```

  サポートが終了した構文：

  ```
  Select 1 from purchaseOrder p
  where existsNode(value(p), '/PurchaseOrder[@poid = "3456"]') = 1
  ```

  標準ベースの構文：

  ```
  Select 1 from purchaseOrder p where xmlexists( 'declare namespace po =
  http://www.po.com;/PurchaseOrder[@poid = 3456]' passing value(p))
  ```

- 名前空間へのパッチ適用：上に示した例のように、**XML文書に名前空間がない場合を除き、名前空間宣言を指定する必要があります。**一方、サポートが終了した構文では、演算子の3番目のパラメータとして指定されていない場合でも、名前空間がパッチされる可能性があります。
- existsNodeは0または1を返しますが、XMLExistsはブールを返すため、SQL WHERE句で新しい構文を直接使用できます。SELECTリストで使用する場合は、"例3：SELECTリストでのXMLExists()とCASE式の使用"を参照してください。
- バインド変数：サポートが終了した構文のように、文字列連結演算子||を使用してXPath文字列を構築し、バインド変数を埋め込む必要はありません。代わりに、PASSING句を使用してバインド変数をXQueryベースの関数に渡します。

  例：

  サポートが終了した構文：

  ```
  Select value(p) from purchaseOrder p
  where existsNode(value(p), '/PurchaseOrder[@podate >' || :1: ']') = 1
  ```

  標準構文：

```
Select value(p) from purchaseOrder p
where xmlexists( 'declare namespace po = http://www.po.com;/PurchaseOrder[@podate >
xs:date($bindvar)]' passing value(p), :1 as "bindvar")
```

- ora:instanceof()およびora:instanceof-only()は、サポートが終了した構文のXPathでのみ使用できます。標準構文では、XQueryの'instance of'式と'@xsi:type ='をそれぞれ使用します。

例：ora:instanceof()

サポートが終了した構文：

```
select extract(value(r),'/N2:R1[ora:instanceof(.,"N1:superType1")]',
'xmlns:N1="http://www.oracle.com/xdb/N1" xmlns:N2="http://www.oracle.com/xdb/N2"
xmlns:ora="http://xmlns.oracle.com/xdb"') from R1 r;
```

標準構文：

```
select XMLQuery('declare namespace N1="http://www.oracle.com/xdb/N1";
declare namespace N2="http://www.oracle.com/xdb/N2";
/N2:R1[. instance of element(N2:R1, N1:superType1)]'
 passing object_value returning content) from R1 r ;
```

例：ora:instanceof-only()

サポートが終了した構文：

```
select extract(value(r),'/N2:R1[ora:instanceof-only(.,"N1:superType1")]',
'xmlns:N1="http://www.oracle.com/xdb/N1" xmlns:N2="http://www.oracle.com/xdb/N2"
xmlns:ora="http://xmlns.oracle.com/xdb"') from R1 r;
```

標準構文：

```
select XMLQuery('declare namespace N1="http://www.oracle.com/xdb/N1"; declare
namespace N2="http://www.oracle.com/xdb/N2";
/N2:R1[@xsi:type="N1:superType1"]' passing object_value returning content)
from R1 r ;
```

xsi:type条件もXPathでサポートされていることに注意してください。つまり、次のクエリは上記の2つと同じように機能します。

```
select extract(value(r),'/N2:R1[@xsi:type="N1:superType1"]',
'xmlns:N1="http://www.oracle.com/xdb/N1" xmlns:N2="http://www.oracle.com/xdb/N2"
xmlns:ora="http://xmlns.oracle.com/xdb"') from R1 r;
```

ora:upper()、ora:lower()、ora:to_number()、ora:to_date()のみが、サポートが終了した構文のXPathで使用可能です。標準構文では、対応するXQuery F&O関数fn:upper-case()、fn:lower-case()、xs:Decimal()、xs:date()をそれぞれ使用します。

例：DBMS_XMLGEN：

サポートが終了した構文：

```
SELECT sys_XMLGen(km_t(kid,
                       kname,
                       knum,
                       CAST(MULTISET (SELECT kid, kdid, kdname
                                        FROM ktest_d d
                                       WHERE d.kid = m.kid) AS kdlist_t))).getclobval() AS detail
FROM ktest_m m;
```

標準構文：

```
select XMLSERIALIZE
        (
      document

      XMLELEMENT
      (
            "KD_LIST",
            XMLAGG
              (
              (
            SELECT XMLAGG
                  (
                  XMLELEMENT
                  (
                  "KD_T",
                  XMLELEMENT("KID",KID),
                  XMLELEMENT("KDID",KDID),
                  XMLELEMENT("KDNAME",KDNAME)
                  )
                  )
            from KTEST_D d
            where d.kid = m.kid
            )
      )
      )
      as clob indent size=2
      )
from KTEST_M m;
```

Connect with us

+1.800.ORACLE1までご連絡いただくか、oracle.comをご覧ください。北米以外の地域では、oracle.com/contactで最寄りの営業所をご確認いただけます。

blogs.oracle.com　facebook.com/oracle　twitter.com/oracle