XDB Ease of Use Tools for Structured Storage

This document describes XDB ease of use tools, which includes packages designed to help you control the schema registration options, thus optimizing storage. These packages also contain functions to help with repetitive tasks, such as indexing and creating views. This is useful because XML DB structured storage provides relational performance for queries, but requires that you register the schema in an optimal way to support your particular use case.

This document contains the following topics:

Installation

This section describes the steps you need to take to install XDB ease of use tools.

1. Download `XDBUtilities.zip` from: `http://www.oracle.com/technology/tech/xml/xmldb`.
2. Unzip `dbmsxutil.sql` and `prvtxutil.plb`.
3. Logon to SQL*Plus as `sysdba` and unlock the XDB user by executing this command:

   ```
   ALTER USER xdb IDENTIFIED BY xdb ACCOUNT UNLOCK
   ```

4. Execute `dbmsxutil.sql` as follows:

   ```
   @dbmsxutil.sql
   ```

5. Execute `prvtxutil.plb` as follows:

   ```
   @prvtxutil.plb
   ```

6. OPTIONAL STEPS:

   You can perform the following steps to verify that installation was successful.

```
select OBJECT_NAME, PROCEDURE_NAME
from ALL_PROCEDURES
where OBJECT_NAME ='DBMS_XMLSCHEMA_ANNOTATE';

select OBJECT_NAME, PROCEDURE_NAME
from ALL_PROCEDURES
where OBJECT_NAME ='DBMS_XMLSTORAGE_MANAGE';

select VIEW_NAME
from ALL_VIEWS
where VIEW_NAME LIKE '%ALL_XML_SCHEMA%';
```

Note: All these procedures and views work with Oracle Database releases 11.2, 11.1, and 10.2, unless otherwise stated.

Exceptions are:

- `DBMS_XMLSTORAGE_MANAGE.XPATH2TABCOLMAPPING()` is not available on 10.2 and 11.1.
- `DBMS_XMLSCHEMA_ANNOTATE.GETSCHEMAANNOTATIONS` and `DBMS_XMLSCHEMA_ANNOTATE.SETSCHEMAANNOTATATIONS` are not available on 10.2 (only 11.*x*).
- View `DBA/ALL/XML_SCHEMA_SIMPLE_TYPES` does not contain the column `SQLTYPE` for 10.2 and 11.1.

Overview of Ease of Use Packages

These packages improve manageability and ease of use for Oracle XML DB (XDB), as part of an effort to remove obstacles current users face and increase the user-base.

Specifically, this project makes the structured storage model (also known as object-relational model) easier to use by simplifying common and repetitive tasks.

Note: XDB also other offers storage options for XML data, such as unstructured Binary XML and a hybrid model, but this project focuses on the structured storage aspect.

Structured storage maps XML structures to relational constructs. The mapping is dictated by an XML schema and annotations to the schema, many of which can be set using the `DBMS_XMLSCHEMA_ANNOTATE` package. These annotations provide more fine-grained information about the mapping process such as which elements are stored inline or out-of-line.

This section covers the following topics:

About Structured Storage

Structured storage typically delivers the best query performance for data-centric XML. This is because structured storage stores the XML data, usually generated from structured relational data, by following the general paradigm for relational database design.

Furthermore, structured storage provides `XMLType` abstraction so that users can still query the data using hierarchical XML abstraction with `XQuery/Xpath`. However, using structured storage requires precise planning, just as for relational data. This includes a good E/R model, with regard to indexes, constraints, data types, table partitions, and so on.

Best Practices for Structured Storage

This section contains best practices showing you how to register to get the best performance for structured storage.

The next section, "How to Annotate a Schema and Register It" on page 5, includes an example script that enables you register XML schemas with XML DB, using `DBMSSCHEMA.REGISTERSCHEMA`, as required for structured storage.

Pre-Registration Recommendations

Before you register an XML schema, Oracle recommends that you perform the following, using this xdb namespace `xmlns:xdb="http://xmlns.oracle.com/xdb"`:

- If a top-level element in the XML schema is not used a root element in the actual XML instance, then use `DBMS_XMLSCHEMA_ANNOTATE.DISABLEDEFAULTTABLECREATION()` to avoid creating unnecessary tables.

  This is recommended, because, by default, XML schema registration creates a top-level table for every top-level element in the schema. See "DISABLEDEFAULTTABLECREATION Procedure" on page 14.

- To create top-level tables with user-defined table names, use `DBMS_XMLSCHEMA_ANNOTATE.SETDEFAULTTABLE()`. This makes `EXPLAIN PLAN`s more readable because the table names are not system-generated. See "SETDEFAULTTABLE Procedure" on page 12.

- To disable DOM fidelity use `DBMS_XMLSCHEMA_ANNOTATE.DISABLEMAINTAINDOM()`.

  This is recommended because, by default, XML schema registration generates table structures to store the DOM fidelity specific information. Generally, DOM fidelity is expensive to maintain and not important to most users. See chapter 7 in *Oracle XML DB Developer's Guide*. See "DISABLEMAINTAINDOM Procedure" on page 34.

- If you are using either Oracle9*i* or Oracle 10*i*, set `storeVarrayAsTable = 'true'` to get better performance.

  This directs XML schema registration to generate a separate collection table for elements that may have multi-occurrences, that is, XML schemas where

`maxOccurs >1`. From Oracle Database 11*g* on, this is a default and does not need to be set.

- If the order of collection elements is not important, then set `maintainOrder="false"`. This enables more optimizations in structured rewrite.
- To avoid raising `'1000 column limit'` errors during XML schema registration, when some elements have *complex types with many elements*, use `DBMS_XMLSCHEMA_ANNOTATE.SETOUTOFLINE()`. See "SETOUTOFLINE Procedure" on page 38.

  This procedure stores the elements out-of-line, meaning that the elements are stored in a separate table and the references to each row of the separate table are stored in a link table that is maintained by the main table. After XML schema registration and before loading XML instance data, use `DBMS_XMLSTORAGE_MANAGE.SCOPEXMLREFERENCES()` to make these references scope to the out-of-line table only. This ensures better query performance later on. See "SCOPEXMLREFERENCES Procedure" on page 49.
- To avoid raising `'1000 column limit'` errors when registering an XML schema that has *an inheritance or substitution group*, call `DBMS_XMLSCHEMA_ANNOTATE.SETOUTOFLINE()`.

  This avoid problems where XDB internally generates tables that model horizontal inheritance design, that is, a table with all columns for each inheritance is stored in one table. See "SETOUTOFLINE Procedure" on page 38.
- If the XML schema has a date or time-related data type, such as `xs:time` or `xs:dateTime`, use `DBMS_XMLSCHEMA_ANNOTATE.SETTIMESTAMPWITHTIMEZONE()`. See "SETTIMESTAMPWITHTIMEZONE Procedure" on page 32.
- If the table needs additional properties, such as partition and table space clauses, use `DBMS_XMLSCHEMA_ANNOTATE.SETTABLEPROPS()`. See "SETTABLEPROPS Procedure" on page 18.

Registration Recommendations

If you do not need to access your XML data through the XDB Repository, then register your schema using `genTables=FALSE` and create your own tables for the given `XMLSchema`.

Example 1–3 on page 6 provides a sample schema registration script

Post-Registration Recommendations

After XML schema registration, Oracle recommends that you do the following:

- Use `DBMS_XMLSTORAGE_MANAGE.RENAMECOLLECTIONTABLE()` to assign a proper name to the collection table. See "RENAMECOLLECTIONTABLE Procedure" on page 47.
- If you have followed the suggestion in the previous section, To avoid raising `'1000 column limit'` errors during XML schema registration, then after XML schema registration and before loading XML instance data, use `DBMS_XMLSTORAGE_MANAGE.SCOPEXMLREFERENCES()` to make these references scope to the out-of-line table only. This ensures better query performance later on. See "SCOPEXMLREFERENCES Procedure" on page 49.
- Call `DBMS_XMLSTORAGE_MANAGE.SCOPEXMLREFERENCES` and `DBMS_XMLSTORAGE_MANAGE.INDEXXMLREFERENCES()` for elements stored

out-of-line. You must scope the REFs before you index them. See "SCOPEXMLREFERENCES Procedure" on page 49 and "INDEXXMLREFERENCES Procedure" on page 46.

- Use DBMS_XMLSTORAGE_MANAGE.XPATH2TABCOLMAPPING() to determine the the underlying storage tables to create indexes. See "XPATH2TABCOLMAPPING Function" on page 51.

  By default, the majority of the indexes required are of type B+ tree index. However, when an XML schema is of enumeration type, its value is stored in binary bit form and requires a bitmap index.

- Use DBMS_XMLSTORAGE_MANAGE.XPATH2TABCOLMAPPING() to determine the names of tables and columns in order to create constraints, if needed. See "XPATH2TABCOLMAPPING Function" on page 51.

- Study the EXPLAIN PLAN of the XQuery or SQL/XML query to ensure that the query plan is reasonable.

- Use these subprograms from the DBMS_XMLSTORAGE_MANAGE package to improve the performance of bulk load operations (such as IAS, SQL Loader, and so on):

  – "DISABLEINDEXESANDCONSTRAINTS Procedure" on page 55 to disable indexes and constraints before starting the bulk load process.

  – "ENABLEINDEXESANDCONSTRAINTS Procedure" on page 58 to enable indexes and constraints once the bulk load process completes.

How to Annotate a Schema and Register It

To annotate and register a script, there are 4 overall steps:

Step 1: Creating an Annotation Table

The script in Example 1–1 creates a table (annotation_tab) with XMLTYPE columns that you can use to keep track of annotations to the schema.

Example 1–1 Creating an Annotation Table

```
create table annotation_tab (
  inp xmltype,
  out xmltype
);
```

Step 2: Start with an Unannotated schema

You can use the script in Example 1–2 to start a schema with no annotations, then annotate it in Step 3.

Example 1–2 Inserting Values into the Annotation Table

```
insert into annotation_tab (inp) values
('<schema targetNamespace="http://www.example.com/IPO"
         xmlns="http://www.w3.org/2001/XMLSchema"
         xmlns:ipo="http://www.example.com/IPO"
         xmlns:xdb="http://xmlns.oracle.com/xdb">
  <element name="purchaseOrder" xdb:defaultTable="PURCHASEORDER_TAB">
     <complexType>
```

```
        <sequence>
           <element name="shipTo"     type="string"/>
           <element name="billTo"     type="string"/>
           <element ref="ipo:comment" minOccurs="0"/>
           <element name="items"      type="ipo:Items"/>
        </sequence>
        <attribute name="orderDate" type="date"/>
     </complexType>
  </element>
  <element name="comment" type="string"/>
  <complexType name="Items">
     <sequence>
        <element name="item" minOccurs="0" maxOccurs="unbounded">
           <complexType>
              <sequence>
                 <element name="productName" type="string"/>
                 <element name="quantity">
                    <simpleType>
                       <restriction base="positiveInteger">
                          <maxExclusive value="100"/>
                       </restriction>
                    </simpleType>
                 </element>
                 <element name="USPrice"    type="decimal"/>
                 <element ref="ipo:comment" minOccurs="0"/>
                 <element name="shipDate"   type="date" minOccurs="0"/>
              </sequence>
              <attribute name="partNum" type="ipo:SKU" use="required"/>
           </complexType>
        </element>
        <any namespace="http://www.w3.org/1999/xhtml"/>
     </sequence>
  </complexType>
  <simpleType name="SKU">
   <restriction base="string">
    <pattern value="\d{3}-[A-Z]{2}"/>
   </restriction>
  </simpleType>
</schema>');

-- Initially the output schema is the same as input schema
update annotation_tab set out = inp;
```

Step 3: Annotating the Schema

Run the `DBMS_XMLSCHEMA_ANNOTATE` package as described in "Using DBMS_XMLSCHEMA_ANNOTATE" on page 1-8. Each example shown in the package updates the `out` column of the annotation table with new annotations.

Step 4: Registering the Annotated Schema

Finally, Example 1–3 registers the annotated schema with the database.

Example 1–3 Registering an Annotated Schema

```
declare
   xml_schema xmltype;
begin
   -- set the default table for purchaseOrder element
```

```
   SELECT out INTO xml_schema FROM annotation_tab;
   DBMS_XMLSCHEMA_ANNOTATE.SETDEFAULTTABLE(xml_schema,
                                           'purchaseOrder',
    'PURCHASEORDER_TAB',
    TRUE);

   UPDATE annotation_tab SET out = xml_schema;

  -- register the annotated schema
  dbms_xmlschema.registerSchema('http://www.example.com/schemas/ipo.xsd',
  xml_schema,
  TRUE, TRUE, FALSE);
end;
/
```

See Also: *Oracle Database PL/SQL Packages and Types Reference* for details of `DBMSSCHEMA.REGISTERSCHEMA` for more details on this option

Using DBMS_XMLSCHEMA_ANNOTATE

Overview

The `DBMS_XMLSCHEMA_ANNOTATE` package contains procedures to manage and configure the structured storage model, mainly through the use of pre-registration schema annotations. Schema annotations influence the way the XML data is stored. For example, the default table annotation assigns a user-provided name to an XML element instead of allowing the database to generate a system name. Consequently, query plans are more readable and it is easier to create constraints on that table.

Overwriting Schema Annotations

Schema annotations typically add different names to XML schema documents. Many procedures in this package add, set, or reset these attributes. Some procedures provide the following modes to enable overwriting existing names:

- `overwrite=FALSE` directs XDB to ignore elements that have an attribute of the given name. This mode only adds attributes; it does not overwrite them.
- `overwrite=TRUE` directs XDB to add or set attributes and overwrites any existing attributes.

Registering Schemas

After you annotate your schema, using the functions and procedures in this package, you can register it using the registration script in "How to Annotate a Schema and Register It" on page 1-5.

DBMS_XMLSCHEMA_ANNOTATE Subprogram Groups

This section presents subprogram groups that are important to the operation of the `DBMS_XMLSCHEMA_ANNOTATE` package.

Table Naming Annotation Subprograms

This subprogram group contains procedures to set names for tables or allow the system to generate the table names.

Table 1–1 Table Naming Annotation Subprograms

Subprogram	Description
REMOVEDEFAULTTABLE Procedure	Removes any default table attribute given for the element
SETDEFAULTTABLE Procedure	Sets the name of the table for the specified global element

REMOVEDEFAULTTABLE Procedure

This procedure removes any default table attribute given for the element. After calling this procedure, the system generates table names. This procedure always overwrites.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.REMOVEDEFAULTTABLE (
   xmlschema         IN OUT XMLType,
   globalElementName IN VARCHAR2);
```

Parameters

Table 1–2 REMOVEDEFAULTTABLE Procedure Parameters

Parameter	Description
`xmlschema`	The XML schema to be annotated
`globalElementName`	The name of the global element in the schema

Example

Annotations can be verified anytime using `"select out from annotation_tab"`.

```
--The purchaseOrder element will have no annotation for defaultTable.
declare
   xml_schema xmltype;
begin
   SELECT out INTO xml_schema FROM annotation_tab;
   DBMS_XMLSCHEMA_ANNOTATE.REMOVEDEFAULTTABLE(xml_schema,
                                              'purchaseOrder');
   UPDATE annotation_tab SET out = xml_schema;
end;
/
```

SETDEFAULTTABLE Procedure

This procedure sets the name of the table for the specified global element. This is equivalent to using the schema annotation `xdb:defaultTable="<default_table_name>"` for the top-level element.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.SETDEFAULTTABLE (
   xmlschema         IN OUT XMLType,
   globalElementName IN VARCHAR2,
   tableName         IN VARCHAR2,
   overwrite         IN BOOLEAN default TRUE);
```

Parameters

Table 1–3 SETDEFAULTTABLE Procedure Parameters

Parameter	Description
`xmlschema`	The XML schema to be annotated
`globalElementName`	The name of the global element in the schema
`tableName`	The name being assigned to the table
`overwrite`	A boolean that indicates whether or not the procedure overwrites element attributes. The default is `TRUE`.

Default Table Creation Annotation Subprograms

This subprogram group contains procedures to determine whether XDB creates tables for top-level elements or not.

> **Note:** Any elements that appear as a root in an instance document must have a table. Otherwise, the insertion of that instance document fails.

Table 1–4 Top-Level Table Creation Annotation Subprograms

Subprogram	Description
DISABLEDEFAULTTABLECREATION Procedure on page 14	Prevents the creation of a table for the top-level element by adding a default table attribute with an empty value to the element
ENABLEDEFAULTTABLECREATION Procedure on page 15	Removes any existing default table annotations so that tables are created with system-generated names

DISABLEDEFAULTTABLECREATION Procedure

This procedure prevents the creation of a table for the top-level element by adding a default table attribute with an empty value to the element. The first overload applies to a specified top-level element and the second applies to all top-level elements. The procedure always overwrites. This is equivalent to using the schema annotation `xdb:defaultTable=""` for the top-level element or elements.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.DISABLEDEFAULTTABLECREATION (
   xmlschema         IN OUT XMLType,
   globalElementName IN VARCHAR2);


DBMS_XMLSCHEMA_ANNOTATE.DISABLEDEFAULTTABLECREATION (
   xmlschema         IN OUT XMLType);
```

Parameters

Table 1–5 DISABLEDEFAULTTABLECREATION Procedure Parameters

Parameter	Description
`xmlschema`	The XML schema to be annotated
`globalElementName`	The name of the global element in the schema

Example

The `purchaseOrder` element will have an annotation similar to `xdb:defaultTable=""`.

```
declare
   xml_schema xmltype;
begin
   SELECT out INTO xml_schema FROM annotation_tab;
   DBMS_XMLSCHEMA_ANNOTATE.DISABLEDEFAULTTABLECREATION(xml_schema,
                                                'purchaseOrder');
   UPDATE annotation_tab SET out = xml_schema;
end;
/
```

ENABLEDEFAULTTABLECREATION Procedure

This procedure removes any existing default table annotations so that tables are created with system-generated names.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.ENABLEDEFAULTTABLECREATION
   (xmlschema IN OUT XMLType);
```

Parameters

Table 1–6 ENABLEDEFAULTTABLECREATION Procedure Parameters

Parameter	Description
`xmlschema`	The XML schema to be annotated

Usage Notes

This procedure does not affect elements that have a default table name.

Table Storage Annotation Subprograms

This subprogram group contains procedures to allow you to assign, to an element, table storage properties that are passed in to the corresponding table creation statement.

Table 1–7 Table Storage Annotation Subprograms

Subprogram	Description
REMOVETABLEPROPS Procedure on page 17	Removes the table storage properties from the `CREATE TABLE` statement
SETTABLEPROPS Procedure on page 18	Specifies properties in the `TABLE` storage clause that is appended to the default `CREATE TABLE` statement

REMOVETABLEPROPS Procedure

This procedure removes the table storage properties from the `CREATE TABLE` statement. This procedure is overloaded. Each overload has different parameter requirements as indicated.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.REMOVETABLEPROPS (
   xmlschema         IN OUT XMLType,
   globalElementName IN VARCHAR2);


DBMS_XMLSCHEMA_ANNOTATE.REMOVETABLEPROPS (
   xmlschema         IN OUT XMLType,
   globalObject      IN VARCHAR2,
   globalObjectName  IN VARCHAR2,
   localElementName  IN VARCHAR2);
```

Parameters

Table 1–8 REMOVETABLEPROPS Procedure Parameters

Parameter	Description
`xmlschema`	The XML schema to be annotated
`globalElementName`	The name of the global element in the schema
`globalObject`	The global object (global complex type or global element)
`globalObjectName`	The name of the global object
`localElementName`	The name of a local element that descends from the global element

Usage Notes

This procedure reverses "SETTABLEPROPS Procedure" on page 18.

SETTABLEPROPS Procedure

This procedure specifies properties in the TABLE storage clause that is appended to the default CREATE TABLE statement. There are two overloads with different parameter requirements, as indicated:

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.SETTABLEPROPS (
   xmlschema         IN OUT XMLType,
   globalElementName IN VARCHAR2,
   tableProps        IN VARCHAR2,
   overwrite         IN BOOLEAN default TRUE);


DBMS_XMLSCHEMA_ANNOTATE.SETTABLEPROPS (
   xmlschema          IN OUT XMLType,
   globalObject       IN VARCHAR2,
   globalObjectName   IN VARCHAR2,
   localElementName   IN VARCHAR2,
   tableProps         IN VARCHAR2,
   overwrite          IN BOOLEAN default TRUE);
```

Parameters

Table 1–9 SETTABLEPROPS Procedure Parameters

Parameter	Description
xmlschema	The XML schema to be annotated
globalElementName	The name of the global element in the schema
tableProps	Table properties
globalObject	The global object (global complex type or global element)
globalObjectName	The name of the global object
localElementName	The name of a local element that descends from the global element
overwrite	A boolean that indicates whether or not the procedure overwrites element attributes. The default is TRUE.

Example

The purchaseOrder element will have an annotation similar to xdb:tableProps="CACHE".

```
declare
   xml_schema xmltype;
begin
   SELECT out INTO xml_schema FROM annotation_tab;
   DBMS_XMLSCHEMA_ANNOTATE.SETTABLEPROPS(xml_schema,
                                         'purchaseOrder' ,'CACHE');
   UPDATE annotation_tab SET out = xml_schema;
end;
/
```

SQL Type Assignment Annotation Subprograms

These procedures assign various SQL metadata to schema elements.

Table 1–10 SQL Type Assignment Annotation Subprograms

Subprogram	Description
REMOVEANYSTORAGE Procedure	Removes the setting of the SQL type from the `ANY` child of the complex type with the given name
REMOVESQLCOLLTYPE Procedure	Removes a SQL collection type
REMOVESQLTYPE Procedure	Removes a SQL type
REMOVESQLTYPEMAPPING Procedure	Removes the SQL type mapping for the given schema type
REMOVETIMESTAMPWITHTIMEZONE Procedure	Removes the setting of the `TimeStampWithTimeZone` data type from all `dateTime` typed elements in the XML schema
SETANYSTORAGE Procedure	Assigns a `SQLType` to the `ANY` child of the complex type with the given name
SETSQLCOLLTYPE Procedure	Assigns a SQL type name for a collection
SETSQLNAME Procedure	Assigns a name to the SQL attribute that corresponds to an element or attribute defined in the XML schema
SETSQLTYPE Procedure	Assigns a SQL type to a global object
SETSQLTYPEMAPPING Procedure	Defines a mapping of schema type and SQL type
SETTIMESTAMPWITHTIMEZONE Procedure	Sets the `TimeStampWithTimeZone` data type to all `dateTime` typed elements in the XML schema

REMOVEANYSTORAGE Procedure

This procedure removes the setting of the SQL type from the `ANY` child of the complex type with the given name.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.REMOVEANYSTORAGE (
   xmlschema       IN OUT XMLType,
   complexTypeName IN VARCHAR2);
```

Parameters

Table 1–11 REMOVEANYSTORAGE Procedure Parameters

Parameter	Description
`xmlschema`	The XML schema to be annotated.
`complexTypeName`	The name of the complex type.

Usage Notes

This procedure reverses "SETANYSTORAGE Procedure" on page 25.

REMOVESQLCOLLTYPE Procedure

This procedure removes a SQL collection type. The first overload removes the SQL collection type corresponding to the named element and the second overload removes the type from the XML element inside the complex type.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.REMOVESQLCOLLTYPE (
   xmlschema   IN OUT XMLType,
   elementName IN VARCHAR2);


DBMS_XMLSCHEMA_ANNOTATE.REMOVESQLCOLLTYPE (
   xmlschema        IN OUT XMLType,
   globalObject     IN VARCHAR2,
   globalName       IN VARCHAR2,
   localElementName IN VARCHAR2);
```

Parameters

Table 1–12 REMOVESQLCOLLTYPE Procedure Parameters

Parameter	Description
`xmlschema`	The XML schema to be annotated
`elementName`	The element name
`globalObject`	The global object (global complex type or global element)
`globalName`	The name of the global object
`localElementName`	The name of a local element that descends from the global element

Usage Notes

This procedure reverses "SETSQLCOLLTYPE Procedure" on page 26.

REMOVESQLTYPE Procedure

This procedure removes a SQL type. The first overload removes a SQL type from a global element and the second overload removes the type from a global element inside the complex type.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.REMOVESQLTYPE (xmlschema in out XMLType,
   globalElementName VARCHAR2);


DBMS_XMLSCHEMA_ANNOTATE.REMOVESQLTYPE (
   xmlschema         IN OUT XMLType,
   globalObject      IN VARCHAR2,
   globalObjectName  IN VARCHAR2,
   localObject       IN VARCHAR2,
   localObjectName   IN VARCHAR2);
```

Parameters

Table 1–13 REMOVESQLTYPE Procedure Parameters

Parameter	Description
`xmlschema`	The XML schema to be annotated.
`globalObject`	The global object (global complex type or global element)
`globalElementName`	The name of the global element.
`globalObjectName`	The name of the global object
`localObject`	An object descended from the global object
`localObjectName`	The name of the local object

Usage Notes

This procedure reverses "SETSQLTYPE Procedure" on page 29.

REMOVESQLTYPEMAPPING Procedure

This procedure removes the SQL type mapping for the given schema type.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.REMOVESQLTYPEMAPPING (
   xmlschema        IN OUT XMLType,
   schemaTypeName   IN VARCHAR2);
```

Parameters

Table 1–14 REMOVESQLTYPEMAPPING Procedure Parameters

Parameter	Description
`xmlschema`	The XML schema to be annotated
schemaTypeName	The name of the schema type

Usage Notes

This procedure reverses "SETSQLTYPEMAPPING Procedure" on page 31.

REMOVETIMESTAMPWITHTIMEZONE Procedure

This procedure removes the setting of the `TimeStampWithTimeZone` data type from all `dateTime` typed elements in the XML schema.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.REMOVETIMESTAMPWITHTIMEZONE (
   xmlschema IN OUT xmlType);
```

Parameters

Table 1–15 REMOVETIMESTAMPWITHTIMEZONE Procedure Parameters

Parameter	Description
`xmlschema`	The XML schema to be annotated

Usage Notes

This procedure reverses "SETTIMESTAMPWITHTIMEZONE Procedure" on page 32.

SETANYSTORAGE Procedure

This procedure assigns a SQL data type to the ANY child of the complex type with the given name.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.SETANYSTORAGE )
   xmlschema       IN OUT XMLType,
   complexTypeName IN VARCHAR2,
   sqlTypeName     IN VARCHAR2,
   overwrite       IN BOOLEAN default TRUE);
```

Parameters

Table 1–16 SETANYSTORAGE Procedure Parameters

Parameter	Description
xmlschema	The XML schema to be annotated
complexTypeName	The name of the complex type
sqlTypeName	The name of the SQL type
overwrite	A boolean that indicates whether or not the procedure overwrites element attributes. The default is TRUE.

Example

The xsd:any child of complex type Items is assigned an annotation similar to xdb:SQLType="VARCHAR".

```
declare
   xml_schema xmltype;
begin
   SELECT out INTO xml_schema FROM annotation_tab;
   DBMS_XMLSCHEMA_ANNOTATE.setAnyStorage (xml_schema,
                                          'Items',
                                          'VARCHAR');
   UPDATE annotation_tab SET out = xml_schema;
end;
/
```

SETSQLCOLLTYPE Procedure

This procedure assigns a SQL type name for a collection. A collection is a global or local element with `maxOccurs>1`. Using this procedure, XDB creates `SQLTypes` with the user-defined names provided.

There are two overloads. The first sets the name of the SQL collection type corresponding to an XML element and the second to an XML element inside the specified complex type.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.SETSQLCOLLTYPE (
   xmlschema   IN OUT XMLType,
   elementName IN VARCHAR2,
   sqlCollType IN VARCHAR2,
   overwrite   IN BOOLEAN default TRUE);


DBMS_XMLSCHEMA_ANNOTATE.SETSQLCOLLTYPE (
   xmlschema          IN OUT XMLType,
   globalObject       IN VARCHAR2,
   globalObjectName   IN VARCHAR2,
   localElementName   IN VARCHAR2,
   sqlCollType        IN VARCHAR2,
   overwrite          IN BOOLEAN default TRUE );
```

Parameters

Table 1–17 SETSQLCOLLTYPE Procedure Parameters

Parameter	Description
`xmlschema`	The XML schema to be annotated
`elementName`	The element name
`sqlCollType`	The SQL collection type
`globalObject`	The global object (global complex type or global element)
`globalObjectName`	The name of the global object
`localElementName`	The name of a local element that descends from the global element
`overwrite`	A boolean that indicates whether or not the procedure overwrites element attributes. The default is `TRUE`.

Example

The `item` element will have an annotation similar to `xdb:SQLCollType="ITEM_SQL_COL_TYPE"`.

```
declare
   xml_schema xmltype;
begin
   SELECT out INTO xml_schema FROM annotation_tab;
   DBMS_XMLSCHEMA_ANNOTATE.setSQLCollType (xml_schema,
                                           'item',
                                           'ITEM_SQL_COL_TYPE',TRUE);
   UPDATE annotation_tab SET out = xml_schema;
end;
```

/

SETSQLNAME Procedure

This procedure assigns a name to the SQL attribute that corresponds to an element defined in the XML schema.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.SETSQLNAME (
   xmlschema          IN OUT XMLType,
   globalObject       IN VARCHAR2,
   globalObjectName   IN VARCHAR2,
   localObject        IN VARCHAR2,
   localObjectName    IN VARCHAR2,
   sqlName            IN VARCHAR2,
   overwrite          IN BOOLEAN default TRUE);
```

Parameters

Table 1–18 SETSQLNAME Procedure Parameters

Parameter	Description
`xmlschema`	The XML schema to be annotated
`globalObject`	The global object (global complex type or global element)
`globalObjectName`	The name of the global object
`localObject`	An object descended from the global object
`localObjectName`	The name of the local object
`sqlName`	The name of the SQL attribute that corresponds to the element defined in the XML schema
`overwrite`	A boolean that indicates whether or not the procedure overwrites element attributes. The default is `TRUE`.

Example

The `shipTo` element will have an annotation similar to `xdb:SQLName="SHIPTO_SQLNAME"`.

```
declare
   xml_schema xmltype;
begin
   SELECT out INTO xml_schema FROM annotation_tab;
DBMS_XMLSCHEMA_ANNOTATE.setSQLName (xml_schema,
                                    'element', 'purchaseOrder',
                                    'element', 'shipTo',
                                    'SHIPTO_SQLNAME');
UPDATE annotation_tab SET out = xml_schema;
end;
/
```

SETSQLTYPE Procedure

This procedure assigns a SQL type to a global object.

There are two overloads. The first overload assigns a SQL Type to a global object, such as a global element or global complex type and the second to a local object.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.SETSQLTYPE (
   xmlschema           IN OUT XMLType,
   globalElementName   IN VARCHAR2,
   sqlType             IN VARCHAR2,
   overwrite           IN BOOLEAN DEFAULT TRUE);


DBMS_XMLSCHEMA_ANNOTATE.SETSQLTYPE (
   xmlschema          IN OUT XMLType,
   globalObject       IN VARCHAR2,
   globalObjectName   IN VARCHAR2,
   localObject        IN VARCHAR2,
   localObjectName    IN VARCHAR2,
   sqlType            IN VARCHAR2,
   overwrite          IN BOOLEAN DEFAULT TRUE);
```

Parameters

Table 1–19 SETSQLTYPE Procedure Parameters

Parameter	Description
`xmlschema`	The XML schema to be annotated
`globalObject`	The global object (global complex type or global element)
`globalObjectName`	The name of the global object
`globalElementName`	The name of the global element
`localObject`	An object descended from the global object
`localObjectName`	The name of the local object
`sqlType`	The SQL type assigned to the named global element
`overwrite`	A boolean that indicates whether or not the procedure overwrites element attributes. The default is `TRUE`.

Example

The `purchaseOrder` element will have an annotation similar to `xdb:SQLType="PO_SQLTYPE"` and the `shipTo` element has one similar to `xdb:SQLType="VARCHAR"`.

```
declare
   xml_schema xmltype;
begin
SELECT out INTO xml_schema FROM annotation_tab;
DBMS_XMLSCHEMA_ANNOTATE.setSQLType (xml_schema,
                                    'purchaseOrder',
                                    'PO_SQLTYPE');
UPDATE annotation_tab SET out = xml_schema;
end;
```

```
/

declare
   xml_schema xmltype;
begin
SELECT out INTO xml_schema FROM annotation_tab;
DBMS_XMLSCHEMA_ANNOTATE.setSQLType (xml_schema,
                                    'element','purchaseOrder',
                                    'element' ,'shipTo',
                                    'VARCHAR');
UPDATE annotation_tab SET out = xml_schema;
end;
/
```

SETSQLTYPEMAPPING Procedure

This procedure defines a mapping of schema type and SQL type.

If you use this procedure, you do not need to call the SETSQLTYPE procedure on all instances of the schema type; instead the procedure traverses the schema and assigns the SQL type automatically.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.SETSQLTYPEMAPPING (
   xmlschema        IN OUT XMLType,
   schemaTypeName   IN VARCHAR2,
   sqlTypeName      IN VARCHAR2,
   overwrite        IN BOOLEAN default TRUE);
```

Parameters

Table 1–20 SETSQLTYPEMAPPING Procedure Parameters

Parameter	Description
xmlschema	The XML schema to be annotated
schemaTypeName	The schema type
sqlTypeName	The name of the SQL type
overwrite	A boolean that indicates whether or not the procedure overwrites element attributes. The default is TRUE

Example

The attribute orderDate will have an annotation similar to xdb:SQLType="DATE".

```
declare
   xml_schema xmltype;
begin
SELECT out INTO xml_schema FROM annotation_tab;
DBMS_XMLSCHEMA_ANNOTATE.setSQLTypeMapping (xml_schema,
                                           'date',
                                           'DATE');
UPDATE annotation_tab SET out = xml_schema;
end;
/
```

SETTIMESTAMPWITHTIMEZONE Procedure

This procedure sets the `TimeStampWithTimeZone` data type to all `dateTime` typed elements in the XML schema. This is equivalent to adding `xdb:SQLType="TIMESTAMP WITH TIME ZONE"` to all `dateTime` objects.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.SETTIMESTAMPWITHTIMEZONE (
   xmlschema IN OUT xmlType,
   overwrite IN BOOLEAN default TRUE);
```

Parameters

Table 1–21 SETTIMESTAMPWITHTIMEZONE Procedure Parameters

Parameter	Description
`xmlschema`	The XML schema to be annotated
`overwrite`	A boolean that indicates whether or not the procedure overwrites element attributes. The default is `TRUE`.

Storage Options Subprograms

This subprogram group contains procedures to set storage options, such as DOM fidelity and `SQLInline` attributes.

Table 1–22 Storage Options Subprograms

Subprogram	Description
DISABLEMAINTAINDOM Procedure	Sets the DOM fidelity attribute to `FALSE`
ENABLEMAINTAINDOM Procedure	Sets the DOM fidelity attribute to `TRUE`
REMOVEMAINTAINDOM Procedure	Removes all annotations used to maintain a DOM from a given schema
REMOVEMAKEOUTOFLINE Procedure	Removes any existing `SQLInline` attributes to prevent out-of-line storage
SETOUTOFLINE Procedure	Sets the `SQLInline` attribute to `FALSE`

DISABLEMAINTAINDOM Procedure

This procedure sets the DOM fidelity attribute to `FALSE`.

There are two overloads. The first sets DOM fidelity attribute to `FALSE` for all complex types, and the second sets it to `FALSE` for the named complex type. This is equivalent to adding `xdb:maintainDOM="false"` on all or specified complex types respectively.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.DISABLEMAINTAINDOM (
   xmlschema       IN OUT XMLType,
   overwrite       IN BOOLEAN default TRUE);


DBMS_XMLSCHEMA_ANNOTATE.DISABLEMAINTAINDOM (
   xmlschema       IN OUT XMLType,
   complexTypeName IN VARCHAR2,
   overwrite       IN BOOLEAN default TRUE);
```

Parameters

Table 1–23 DISABLEMAINTAINDOM Procedure Parameters

Parameter	Description
`xmlschema`	The XML schema to be annotated
`complexTypeName`	The name of the complex type
`overwrite`	A boolean that indicates whether or not the procedure overwrites element attributes. The default is `TRUE`

ENABLEMAINTAINDOM Procedure

This procedure sets the DOM fidelity attribute to TRUE.

There are two overloads. The first sets DOM fidelity attribute to TRUE for all complex types, and the second sets it to TRUE for the named complex type.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.ENABLEMAINTAINDOM (
   xmlschema       IN OUT XMLType,
   overwrite       IN BOOLEAN default TRUE);


DBMS_XMLSCHEMA_ANNOTATE.ENABLEMAINTAINDOM (
   xmlschema        IN OUT XMLType,
   complexTypeName  IN VARCHAR2,
   overwrite        IN BOOLEAN default TRUE);
```

Parameters

Table 1–24 ENABLEMAINTAINDOM Procedure Parameters

Parameter	Description
xmlschema	The XML schema to be annotated
complexTypeName	The name of the complex type
overwrite	A boolean that indicates whether or not the procedure overwrites element attributes. The default is TRUE

REMOVEMAINTAINDOM Procedure

This procedure removes all annotations used to maintain DOM from the given schema.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.REMOVEMAINTAINDOM (
   xmlschema    IN OUT XMLType,
   );
```

Parameters

Table 1–25 REMOVEMAINTAINDOM Procedure Parameters

Parameter	Description
xmlschema	The XML schema to be annotated

REMOVEMAKEOUTOFLINE Procedure

This procedure removes any existing `SQLInline` attributes to prevent out-of-line storage. There are three overloads.

Syntax

Removes the `SQLInline` attribute for the named element.

```
DBMS_XMLSCHEMA_ANNOTATE.REMOVEOUTOFLINE (
   xmlschema    IN OUT XMLType,
   elementName  IN VARCHAR2,
   elementType  IN VARCHAR2,
   overwrite    IN BOOLEAN default TRUE);
```

Removes the `SQLInline` attribute for the object specified by its global object and local element names.

```
DBMS_XMLSCHEMA_ANNOTATE.REMOVEOUTOFLINE (
   xmlschema        IN OUT XMLType,
   globalObject     IN VARCHAR2,
   globalObjectName IN VARCHAR2,
   localElementName IN VARCHAR2);
```

Removes the `SQLInline` attribute for the referenced global element.

```
DBMS_XMLSCHEMA_ANNOTATE.REMOVEOUTOFLINE (
   xmlschema   IN OUT XMLType,
   reference   IN VARCHAR2);
```

Parameters

Table 1–26 REMOVEMAKEOUTOFLINE Procedure Parameters

Parameter	Description
`xmlschema`	The XML schema to be annotated
`elementName`	The element name
`elementType`	The element type
`globalObject`	The global object (global complex type or global element)
`globalObjectName`	The name of the global object
`localElementName`	The name of a local element that descends from the global element
`reference`	A reference to a global element
`overwrite`	A boolean that indicates whether or not the procedure overwrites element attributes. The default is `TRUE`.

Usage Notes

This procedure reverses "SETOUTOFLINE Procedure" on page 38.

SETOUTOFLINE Procedure

This procedure sets the `SQLInline` attribute to `FALSE`, that is, it sets `xdb:SQLInLine=FALSE`.

This forces XDB to store the corresponding elements in the XML document out-of-line as rows in a separate `XMLType` table. XDB stores references to each row of the `XMLType` table in a link table that is maintained by the main table

This procedure can improve performance in some situations if the out-of-line table acts as the driver for the query. Storing elements in an out-of-line table also reduces the numbers of columns in the base table, thus avoiding `'1000 column limit'` errors during XML schema registration, when some elements have complex types with many elements.

Also See: *Oracle XML DB Developer's Guide*

There are three overloads.

Syntax

Sets the `SQLInline` attribute to `FALSE`, forcing out-of-line storage for the named element.

```
DBMS_XMLSCHEMA_ANNOTATE.SETOUTOFLINE (
   xmlschema          IN OUT XMLType,
   elementName        IN VARCHAR2,
   elementType        IN VARCHAR2,
   defaultTableName   IN VARCHAR2,
   overwrite          IN BOOLEAN default TRUE);
```

Sets the `SQLInline` attribute to `FALSE`, forcing out-of-line storage for the element specified by its local and global name.

```
DBMS_XMLSCHEMA_ANNOTATE.SETOUTOFLINE (
   xmlschema          IN OUT XMLType,
   globalObject       IN VARCHAR2,
   globalObjectName   IN VARCHAR2,
   localElementName   IN VARCHAR2,
   defaultTableName       IN VARCHAR2,
   overwrite          IN BOOLEAN default TRUE);
```

Sets the `SQLInline` attribute to `FALSE` to force out-of-line storage and sets the default table name for all references to a particular global element.

```
DBMS_XMLSCHEMA_ANNOTATE.SETOUTOFLINE (
   xmlschema         IN OUT XMLType,
   reference         IN VARCHAR2,
   defaultTableName      IN VARCHAR2,
   overwrite         IN BOOLEAN default TRUE);
```

Parameters

Table 1–27 SETOUTOFLINE Procedure Parameters

Parameter	Description
`xmlschema`	The XML schema to be annotated.
`elementName`	The element name

Table 1–27 (Cont.) SETOUTOFLINE Procedure Parameters

Parameter	Description
elementType	The element type
defaultTableName	The name of the default table.
globalObject	The global object (global complex type or global element)
globalObjectName	The name of the global object
localElementName	The name of a local element that descends from the global element.
reference	A reference to a global element
overwrite	A boolean that indicates whether or not the procedure overwrites element attributes. The default is TRUE.

Usage Notes

After XML schema registration and before loading XML instance data, use DBMS_XMLSTORAGE_MANAGE.SCOPEXMLREFERENCES() to make these references scope to the out-of-line table only. This ensures better query performance later on. See "SCOPEXMLREFERENCES Procedure" on page 49.

Example

The following example illustrates the third overloaded method. The element comment will have an annotation similar to xdb:defaultTable="CMMNT_DEFAULT_TABLE"

```
declare
   xml_schema xmltype;
begin
SELECT out INTO xml_schema FROM annotation_tab;

DBMS_XMLSCHEMA_ANNOTATE.setOutOfLine (xml_schema,
                                      'ipo:comment',
                                      'CMMNT_DEFAULT_TABLE');
UPDATE annotation_tab SET out = xml_schema;
end;
/
```

Schema Differentiating Annotation Subprograms

These subprograms work together to save the differentiating annotations between an original XML schema and an updated version of it. The annotations can then be applied to other XML schemas.

Table 1–28 Schema Differentiating Annotation Subprograms

Subprogram	Description
GETSCHEMAANNOTATIONS Function	Creates a document containing the differences between the annotated XML schema and the original XML schema
SETSCHEMAANNOTATATIONS Procedure	Takes the annotated differences resulting from a call to `DBMS_XMLSCHEMA_ANNOTATE.GETSCHEMAANNOTATIONS` and patches them into the provided XML schema

GETSCHEMAANNOTATIONS Function

This function creates a document containing the differences between the annotated XML schema and the original XML schema.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.GETSCHEMAANNOTATIONS (
   xmlschema IN xmlType)
  RETURN XMLType;
```

Parameters

Table 1–29 GETSCHEMAANNOTATIONS Function Parameters

Parameter	Description
`xmlschema`	The original XML schema

Return Values

This function returns the document `annotations.xml` as an `XMLType`.

Usage Notes

This function saves all annotations in one document, named `annotations`, and returns it. With this document, you can apply all annotations to a non-annotated schema, using `DBMS_XMLSCHEMA_ANNOTATE.GETSCHEMAANNOTATIONS`.

`DBMS_XMLSCHEMA_ANNOTATE.GETSCHEMAANNOTATIONS` is not available on Oracle Database release 10.2 (only Oracle Database release 11.*x*).

See Also: "SETSCHEMAANNOTATATIONS Procedure" on page 42

Example

For an example of `DBMS_XMLSCHEMA_ANNOTATE.GETSCHEMAANNOTATIONS`, see the example in "SETSCHEMAANNOTATATIONS Procedure" on page 42.

SETSCHEMAANNOTATATIONS Procedure

This procedure takes the annotated differences resulting from a call to DBMS_XMLSCHEMA_ANNOTATE.GETSCHEMAANNOTATIONS and patches them into the provided XML schema.

Syntax

```
DBMS_XMLSCHEMA_ANNOTATE.SETSCHEMAANNOTATIONS (
   xmlschema     IN OUT xmlType,
   annotations   IN VARCHAR2);
```

Parameters

Table 1–30 SETSCHEMAANNOTATIONS Procedure Parameters

Parameter	Description
xmlschema	An XML schema to be patched.
annotations	The differences document produced by calling DBMS_XMLSCHEMA_ANNOTATE.GETSCHEMAANNOTATIONS on the original XML schema and an annotated XML schema.

Usage Notes

DBMS_XMLSCHEMA_ANNOTATE.SETSCHEMAANNOTATATIONS is not available on Oracle Database release 10.2 (only Oracle Database release 11.*x*).

See Also: "GETSCHEMAANNOTATIONS Function" on page 41

Example

The following example illustrates DBMS_XMLSCHEMA_ANNOTATE.SETSCHEMAANNOTATIONS shown here and "GETSCHEMAANNOTATIONS Function" on page 41.

```
-- test getannotations and apply them
declare
 xml_schema xmltype;
 xml_schema2 xmltype;
 annotations xmltype;
begin
   select out into xml_schema from annotation_tab;

   -- get the annotations from the schema
   annotations := DBMS_XMLSCHEMA_ANNOTATE.getSchemaAnnotations (xml_schema);

   -- apply the annotations to the schema
   select inp into xml_schema2 from annotation_tab;

   DBMS_XMLSCHEMA_ANNOTATE.setSchemaAnnotations(xml_schema2, annotations);

   update annotation_tab t set t.out = xml_schema2;
end;
/
```

Using DBMS_XMLSTORAGE_MANAGE

`DBMS_XMLSTORAGE_MANAGE` contains procedures to manage and modify XML storage after schema registration has been completed. Use subprograms from this package to improve the performance of bulk load operations. You can disable indexes and constraints before doing a bulk load process and enable them afterwards.

DBMS_XMLSTORAGE_MANAGE Subprogram Groups

This section presents subprogram groups that are important to operation of the `DBMS_XMLSTORAGE_MANAGE` package.

- XML Table Modification Subprograms
- XPath to Tables or Columns Subprograms
- XML Data Bulk Loading Subprograms

XML Table Modification Subprograms

These subprograms enable you to modify underlying XML tables after schema registration.

> **Note:** These procedures are limited to the structured storage model.

Table 1–31 XML Table Modification Subprograms

Subprogram	Description
INDEXXMLREFERENCES Procedure on page 46	Creates unique indexes on the `REF` columns of the given XML type table or the XML type column of a given table
RENAMECOLLECTIONTABLE Procedure on page 47	Renames a collection table to the given table name
SCOPEXMLREFERENCES Procedure on page 49	Scopes all XML references

INDEXXMLREFERENCES Procedure

This procedure creates unique indexes on the REF columns of the given XML type table or the XML type column of a given table.

If the procedure creates multiple REF columns, it appends _1, _2, and so on to their names.

Syntax

```
DBMS_XMLSTORAGE_MANAGE.INDEXXMLREFERENCES (
   owner_name    IN VARCHAR2 DEFAULT USER,
   table_name    IN VARCHAR2,
   column_name   IN VARCHAR2 default NULL),
   index_name    IN VARCHAR2;
```

Parameters

Table 1–32 INDEXXMLREFERENCES Procedure Parameters

Parameter	Description
owner_name	The owner's name
table_name	The table being indexed
column_name	A column name. Not needed for XML type tables.
index_name	The name of the newly created index

Usage Notes

This procedure is only used if the REFs are scoped. See SCOPEXMLREFERENCES Procedure on page 49.

Indexed REFs lead to better performance when joins between the base table and a child table occur in the query plan.

- If the base table has a higher selectivity than the child table, there is no need to index the REFs.
- If the selectivity of the child table is higher than that of the base table and if no indexes are present, then the join of one row in the child table with the base table leads to a full table scan of the base table.

INDEXXMLREFERENCES does not index REFs recursively in child tables of a table it is called on. To do this, Oracle recommends calling the procedure from within a loop over the XML_OUT_OF_LINE_TABLES or XML_NESTED_TABLES view. This creates the index names from the current value of a column in the view.

Note: This procedure is limited to the structured storage model.

RENAMECOLLECTIONTABLE Procedure

This procedure renames a collection table to the given table name.

An XPath expression specifies the collection table, starting from the `XMLtype` base table or an `XMLType` column of the base table.

This procedure provides the only way to derive a collection table name from the corresponding collection type name because there is no direct schema annotation for the purpose.

Syntax

```
DBMS_XMLSTORAGE_MANAGE.RENAMECOLLECTIONTABLE (
   owner_name            IN VARCHAR2 DEFAULT USER,
   table_name            IN VARCHAR2,
   column_name           IN VARCHAR2 DEFAULT NULL,
   xpath                 IN VARCHAR2,
   collection_table_name IN VARCHAR2
   namespaces IN VARCHAR2 default NULL  //for release 11.2 only
);
```

Parameters

Table 1–33 RENAMECOLLECTIONTABLE Procedure Parameters

Parameter	Description
`owner_name`	The name of the owner
table_name	The name of a base table that can be used as the starting point for specifying the collection table
`column_name`	An `XMLType` column that can be the starting point for specifying the collection table
`xpath`	The XPath expression that specifies the collection table
`collection_table_name`	The name of the collection table
`namespaces`	For Oracle Database 11*g* Release 2 (11.2) and higher. The namespaces used in XPath.

Usage Notes

Call this procedure after registering the XML schema.

The table name serves as a prefix to the index names.

Oracle recommends using this function because it makes query execution plans more readable.

Report errors that occur while this procedure runs to the user that called the procedure.

> **Note:** This procedure is limited to the structured storage model.

For Oracle Database 11*g* Release 2 (11.2) and higher, only, this function accepts XPath notation as well as dot notation. If XPath notation is used, a `namespaces` parameter may also be required.

Example

The collection table name will be `EMP_TAB_NAMELIST`. You can verify this using `select * from user_nested_tables`.

Using Dot Notation:

```
call  XDB.DBMS_XMLSTORAGE_MANAGE.renameCollectionTable(
           USER,
           'EMP_TAB',
           NULL,
           '"XMLDATA"."EMPLOYEE"."NAME"',
          ''EMP_TAB_NAMELIST');
```

Using XPath Notation:

XPath notation is available with Oracle Database 11g Release 2 (11.2) and higher.

```
call  XDB.DBMS_XMLSTORAGE_MANAGE.renameCollectionTable(
              USER,
              'EMP_TAB',
              NULL,
              '/e:Employee/Name',
              'EMP_TAB_NAMELIST',
              '''http://www.oracle.com/emp.xsd'' as "e"');
```

SCOPEXMLREFERENCES Procedure

This procedure scopes all XML references. Scoped `REF` types require less storage space and allow more efficient access than unscoped `REF` types.

Syntax

```
DBMS_XMLSTORAGE_MANAGE.SCOPEXMLREFERENCES;
```

Usage Notes

If you have used "SETOUTOFLINE Procedure" on page 38 to avoid raising `'1000 column limit'` errors during XML schema registration, you should also use "SCOPEXMLREFERENCES Procedure" on page 49.

Using `DBMS_XMLSTORAGE_MANAGE.SCOPEXMLREFERENCES()` after XML schema registration and before loading XML instance data, makes these reference scoped to the out-of-line table only.

Note: This procedure is limited to the structured storage model.

XPath to Tables or Columns Subprograms

This subprogram helps you to determine the names of underlying tables and columns in secondary tables where the names correspond to element types in the XML schema or to an XPath expression. The subprogram uses these names to create indexes or constraints on secondary tables.

This is useful because schema registration creates many secondary tables.

Table 1–34 XPath to Tables or Columns Subprograms

Subprogram	Description
XPATH2TABCOLMAPPING Function on page 51	Maps a user-provided XPath expression and returns the names of the underlying tables and columns

XPATH2TABCOLMAPPING Function

This function maps a user-provided XPath expression and returns the names of the underlying tables and columns. You can use these names to create B-Tree indexes, Bitmap-indexes, or constraints.

Syntax

```
DBMS_XMLSTORAGE_MANAGE.XPATH2TABCOLMAPPING (
   owner_name    IN VARCHAR2,
   table_name    IN VARCHAR2,
   column_name   IN VARCHAR2 DEFAULT NULL,
   xpath         IN VARCHAR2,
   namespaces    IN VARCHAR2)
  RETURN XMLTYpe;
```

Parameters

Table 1–35 XPATH2TABCOLMAPPING Function Parameters

Parameter	Description
`owner_name`	The owner's name
`table_name`	The name of the topmost base table
`column_name`	A column name that the function may be called on
`xpath`	An XPath expression that can contain only the child and descendant axis, but no predicates. The wildcard `nodetest*` is valid
`namespaces`	The namespaces used in XPath. Namespaces have the same syntax as the `XMLNAMESPACES` clause for the SQL/XML function `XMLTable`. See section SQL/XML Functions `XMLQUERY` and `XMLTABLE` in Chapter 5 of *Oracle XML DB Developer's Guide* for further details.

Return Values

An `XMLType` containing the table name or names and the column name or names of the internal storage tables.

Usage Notes

This function can be called on either an XML type table or an XML type column. In either case, the table must be the topmost base table. The function should not be called on internal XML type tables or columns. If the parameter `column_name` is not specified (null), then an XML type table is assumed.

This function only supports structured storage. Calling the function on binary or `CLOB` storage causes an error.

`DBMS_XMLSTORAGE_MANAGE.XPATH2TABCOLMAPPING()` is not available on Oracle Database releases 10.2 and 11.1.

Example

The following examples demonstrate how to map XPath to a table or column.

Example 1: Demonstrates a Manual Examination of Result

This is a 2 step process:

Step 1:

```
select XDB.DBMS_XMLSTORAGE_MANAGE.xpath2TabColMapping(USER,
  'PURCHASEORDER_TAB',NULL, '/ipo:purchaseOrder/shipTo', '
  ''http://www.example.com/IPO'' as "ipo"') from  dual;
```

The result should look like the following with differing values for `SYS_NC`...

```
<Result>
  <Mapping TableName="PURCHASEORDER_TAB" ColumnName="SYS_NC00009$"/>
</Result>
```

Step 2:

Create an index or constraint by manually extracting the required information.

```
create index shipto_idx on PURCHASEORDER_TAB (SYS_NC00009$);
```

Example 2: Pointing One XPath to Multiple Table-Column-pairs

You can map one XPath to multiple table-column-pairs using a wildcard or descendant axis in the XPath by following this 2 step process:

Step 1:

```
select XDB.DBMS_XMLSTORAGE_MANAGE.xpath2TabColMapping(USER,
  'PURCHASEORDER_TAB',NULL, '/ipo:purchaseOrder/items/item/*',
  '''http://www.example.com/IPO'' as "ipo"') from  dual;
```

The results should look like the following:

```
<Result>
  <Mapping TableName="SYS_NTbp8aCRZa/mzgQESYQmUt/Q==" ColumnName="productName"/>
  <Mapping TableName="SYS_NTbp8aCRZa/mzgQESYQmUt/Q==" ColumnName="quantity"/>
  <Mapping TableName="SYS_NTbp8aCRZa/mzgQESYQmUt/Q==" ColumnName="USPrice"/>
  <Mapping TableName="SYS_NTbp8aCRZa/mzgQESYQmUt/Q==" ColumnName="comment"/>
  <Mapping TableName="SYS_NTbp8aCRZa/mzgQESYQmUt/Q==" ColumnName="shipDate"/>
</Result>
```

Step 2:

Create an index or constraint by manually extracting the required information.

```
create index shipto_idx on "SYS_NTbp8aCRZa/mzgQESYQmUt/Q" (productName);
```

Repeat this command for the remaining `Result` items.

Example 3: PL/SQL Program Creates Index For Each XPath Expression Mapping

Manual extraction, as shown in the previous examples, cannot be used with a script because table names and column names are system-generated. If a script is needed, the following code unifies both steps into one script.

This PL/SQL program automatically creates one index for each mapping of an XPath expression.

```
DECLARE
  index_name VARCHAR2(32) := 'po_idx';
  tab_name   VARCHAR2(32);
  col_name   VARCHAR2(32);
```

```
   CURSOR idx_cursor IS
     SELECT "tab_name", "col_name"
     FROM XMLTable('/Result/Mapping'
             passing  (
                SELECT XDB.DBMS_XMLSTORAGE_MANAGE.xpath2TabColMapping(USER,
                   PURCHASEORDER_TAB',NULL,
                   '/ipo:purchaseOrder/items/item/*',
                   '''http://www.example.com/IPO'' as "ipo"')
                FROM  dual)
           COLUMNS "tab_name" VARCHAR2(32) PATH '/Mapping/@TableName',
                   "col_name" VARCHAR2(32) PATH '/Mapping/@ColumnName');
BEGIN
  FOR entry IN idx_cursor LOOP
     EXECUTE IMMEDIATE 'CREATE INDEX ' || index_name || '_' ||  entry."col_name"
         || ' ON "' || entry."tab_name" ||'"("'|| entry."col_name"||'")';
  END LOOP;
END;/
```

XML Data Bulk Loading Subprograms

These procedures work together to improve the performance of bulk load operations. One procedure disables indexes and constraints. This makes the next step, bulk data loading, quicker because rows do not have to be inserted into indexes and checked for damage to constraints. After the bulk load completes, another procedure recreates the indexes and reactivates the constraints.

You can perform a bulk load operation using SQL Loader, Data Pump Import, or `Insert... Select` statements. However, SQL Loader is likely to perform best as it implicitly disables triggers before starting the bulk load.

The bulk load procedures recursively disable and enable indexes and constraints on all the child and out-of-line tables associated with the input `XMLTYPE` table or column.

Flow of Bulk Load Process

- Disable indexes and constraints "DISABLEINDEXESANDCONSTRAINTS Procedure" on page 55
- Load data using bulk load utilities and processes
- Enable indexes and constraints "ENABLEINDEXESANDCONSTRAINTS Procedure" on page 58

About Bulk Load Procedures

The bulk load procedures follow these guidelines:

- They restrict usage to only structured storage, that is, the procedures assume that the `XMLTYPE` tables and columns are stored object-relationally.
- They disable and enable all system indexes (such as the `nested_table_id` index).
- They disable and enable all user indexes including domain indexes such as `XMLIndex` and text index.
- They disable and enable all constraints.
- They do not disable or enable any triggers.

Table 1–36 XML Data Bulk Loading Subprograms

Subprogram	Description
DISABLEINDEXESANDCONSTRAINTS Procedure on page 55	Disable the indexes and constraints for `XMLType` tables and `XMLType` columns
ENABLEINDEXESANDCONSTRAINTS Procedure on page 58	Rebuilds all indexes and enables the constraints on an `XMLType` table including its child tables and out-of-line tables.

DISABLEINDEXESANDCONSTRAINTS Procedure

This procedure disables the indexes and constraints for XMLType tables and XMLType columns.

Syntax

```
DBMS_XMLSTORAGE_MANAGE.DISABLEINDEXESANDCONSTRAINTS (
   owner_name    IN VARCHAR2 DEFAULT USER,
   table_name    IN VARCHAR2,
   column_name   IN VARCHAR2 DEFAULT NULL)
   clear BOOLEAN IN DEFAULT FALSE);
```

Parameters

Table 1–37 DISABLEINDEXESANDCONSTRAINTS Procedure Parameters

Parameter	Description
owner_name	The owner's name
table_name	The name of the XMLType table that the procedure is being performed on
column_name	An XMLType column name
clear	A boolean that when set to TRUE clears all stored index and constraint data for the table before the procedure executes. The default is FALSE, which does not clear them. See "Using clear to Enable and Disable Indexes and Constraints" on page 1-55

Usage Notes

Passing XMLTYPE tables

For XMLType tables, you must pass the XMLType table name on which the bulk load operation is to be performed. For XMLType columns, you must pass the relational table name and the corresponding XMLType column name.

Using clear to Enable and Disable Indexes and Constraints

> **Note:** If the DISABLEINDEXESANDCONTRAINTS procedure is called with clear set to TRUE, it removes any index or constraint information about the XMLTYPE table or column memorized during earlier executions of the procedure.

Therefore, you must ensure that all disabled indexes and constraints are re-enabled on the table or column before you call the DISABLEINDEXESANDCONTRAINTS procedure with clear set to TRUE.

Ideally, it is recommended that you set clear set to TRUE for the first execution. For any subsequent executions (due to errors while disabling or enabling indexes) clear should be set to FALSE, the default value. Once you have successfully re-enabled all the indexes and constraints following the bulk load operation, you can call this procedure again with clear set to TRUE for the next bulk load operation.

Example

The following example illustrates the use of `clear` in the `DISABLEINDEXESANDCONSTRAINTS` procedure and the "ENABLEINDEXESANDCONSTRAINTS Procedure" on page 58:

Example 1–4 Use of clear with Indexes and Constraints

First, add a not null constraint on `comment` element of the `PURCHASEORDER_TAB` table

```
alter table PURCHASEORDER_TAB add constraint c1 check
            ("XMLDATA"."comment" is not null);
```

Then, disable all the indexes and constraints by passing the `clear` as `TRUE`, by calling the `DISABLEINDEXESANDCONSTRAINTS` procedure.

```
begin
     xdb.DBMS_XMLSTORAGE_MANAGE.DISABLEINDEXESANDCONSTRAINTS
                                ( USER,'PURCHASEORDER_TAB',NULL,TRUE );
end;
/
```

Next, perform a bulk load operation (such as datapump import) which violates constraint `c1` in the `ALTER` table statement. This does not raise an error because the constraint is disabled.

```
host impdp orexample/orexample directory=dir dumpfile=dmp.txt
   tables=OREXAMPLE.PURCHASEORDER_TAB content = DATA_ONLY;
```

NOTE: To view the disabled constraints and indexes use:

```
select constraint_name,table_name,status from all_constraints
  where owner = user;
```

Finally, try to enable the constraint using the `ENABLEINDEXESANDCONSTRAINTS` procedure. It raises an error because `c1`, the not null constraint, is violated by the bulk load operation.

```
begin
     xdb.DBMS_XMLSTORAGE_MANAGE.ENABLEINDEXESANDCONSTRAINTS
                                          ( USER,'PURCHASEORDER_TAB');
end;
/
```

To disable all the indexes and constraints, again use `DISABLEINDEXESANDCONSTRAINTS`, but set `clear= FALSE` (because the `ENABLEINDEXESANDCONSTRAINTS` failed to complete successfully). Note: `clear = FALSE` by default, so we do not need to pass it explicitly in the next call.

```
begin
     xdb.DBMS_XMLSTORAGE_MANAGE.DISABLEINDEXESANDCONSTRAINTS
                                ( USER,'PURCHASEORDER_TAB');
end;
/
```

Then, delete the incorrect rows entered into the table

```
delete from purchaseorder_tab p
     where p.xmldata."comment"
              is NULL;
```

Re-enable the indexes and constraints using ENABLEINDEXESANDCONSTRAINTS, which completes successfully.

```
begin
     xdb.DBMS_XMLSTORAGE_MANAGE.ENABLEINDEXESANDCONSTRAINTS
                                          ( USER,'PURCHASEORDER_TAB');
end;
/
```

ENABLEINDEXESANDCONSTRAINTS Procedure

This procedure rebuilds all indexes and enables the constraints on an `XMLType` table including its child tables and out-of-line tables. When `column_name` is passed, it does the same for this `XMLType` column.

Syntax

```
DBMS_XMLSTORAGE_MANAGE.ENABLEINDEXESANDCONSTRAINTS (
   owner_name   IN VARCHAR2 DEFAULT USER,
   table_name   IN VARCHAR2,
   column_name  IN VARCHAR2 DEFAULT NULL);
```

Parameters

Table 1–38 ENABLEINDEXESANDCONSTRAINTS Procedure Parameters

Parameter	Description
`owner_user`	The owner's name
`table_name`	The name of the table that the indexes and constraints are being removed from
`column_name`	A column name

Usage Notes

This procedure reverses "DISABLEINDEXESANDCONSTRAINTS Procedure" on page 55.

Example

See Example 1–4 on page 56.

Schema Views for XDB

For all the XDB schema views, `DBA`, `ALL`, and `USER` versions apply.

After you register the schema, you can use these views to query tables and type structures related to the underlying table.

The following is a list of XDB views.

XML_SCHEMA_ATTRIBUTES

XML_SCHEMA_COMPLEX_TYPES

XML_SCHEMA_ELEMENTS

XML_SCHEMA_NAMESPACES

XML_NESTED_TABLES

XML_OUT_OF_LINE_TABLES

XML_SCHEMA_SIMPLE_TYPES

XML_SCHEMA_SUBSTGRP_HEAD

XML_SCHEMA_SUBSTGRP_MBRS

XML_SCHEMA_ATTRIBUTES

XML_SCHEMA_ATTRIBUTES view displays all the attributes and their properties.

Table 1–39 XML_SCHEMA_ATTRIBUTES View

Column	Description
OWNER	User who owns the attribute
XML_SCHEMA_URL	URL of schema within which the attribute is defined
TARGET_NAMESPACE	Namespace of the attribute
ATTRIBUTE_NAME	Name of the attribute
TYPE_NAME	Name of type of the attribute
GLOBAL	1 if attribute is global, otherwise 0
ATTRIBUTE	Actual XMLType for the attribute
ELEMENT_ID	Element ID of the element to which the attribute belongs

XML_SCHEMA_COMPLEX_TYPES

XML_SCHEMA_COMPLEX_TYPES view displays all complex type.

Table 1–40 XML_SCHEMA_COMPLEX_TYPES View

Column	Description
OWNER	User who owns the type
XML_SCHEMA_URL	URL of schema that contains the type definition
TARGET_NAMESPACE	Namespace of the type
COMPLEX_TYPE_NAME	Name of the complex type
COMPLEX_TYPE	Actual XMLType of the type
MAINTAIN_DOM	XDB annotation for maintainDOM
SQL_TYPE	XDB annotation for sqlType
SQL_SCHEMA	XDB annotation for sqlSchema
DEFAULT_TABLE	XDB annotation for defaultTable
SQL_NAME	XDB annotation for sqlName
SQL_COL_TYPE	XDB annotation for sqlColType
STORE_VARRAY_AS_TABLE	XDB annotation for storeVarrayAsTable

XML_SCHEMA_ELEMENTS

XML_SCHEMA_ELEMENTS view displays all the elements and their properties.

Table 1–41 XML_SCHEMA_ELEMENTS View

Column	Description
OWNER	User who owns the element
XML_SCHEMA_URL	URL of schema that contains the element
TARGET_NAMESPACE	Namespace of the element
ELEMENT_NAME	Name of the element
TYPE_NAME	Name of the type of the element
GLOBAL	1 if the element is global, otherwise 0
ELEMENT	Actual XML fragment of the element
SQL_TYPE	XDB annotation value for SQLType
SQL_SCHEMA	XDB annotation value for SQLSchema
DEFAULT_TABLE	XDB annotation value for default table
SQL_NAME	XDB annotation value for SQLName
SQL_INLINE	XDB annotation for SQLInline
SQL_COL_TYPE	XDB annotation value for SQLColType
ELEMENT_ID	Unique identifier for the element
PARENT_ELEMENT_ID	Identifies the parent of the element

XML_SCHEMA_NAMESPACES

XML_SCHEMA_NAMESPACES view is self-describing. It displays all the available namespaces.

Table 1–42 XML_SCHEMA_NAMESPACES View

Column	Description
OWNER	User who owns the namespace
TARGET_NAMESPACE	Target namespace
XML_SCHEMA_URL	URL of the schema

XML_NESTED_TABLES

XML_NESTED_TABLES view displays all the tables and their corresponding nested tables.

Table 1–43 XML_NESTED_TABLES View

Column	Description
OWNER	Owner of the table
TABLE_NAME	Name of the table
NESTED_TABLE_NAME	Name of the nested table

XML_OUT_OF_LINE_TABLES

XML_OUT_OF_LINE_TABLES view returns all the out-of-line tables connected to a given root table for the same schema.

Note: this returns the root_table, also as a part of table_name.

Table 1–44 XML_OUT_OF_LINE_TABLES View

Column	Description
ROOT_TABLE_NAME	Name of the root table
ROOT_TABLE_OWNER	Owner of the root table
TABLE_NAME	Name of out of line table
TABLE_OWNER	Owner of the out of line table

XML_SCHEMA_SIMPLE_TYPES

XML_SCHEMA_SIMPLE_TYPES view displays all simple types.

XML_SCHEMA_SIMPLE_TYPES view does not contain the column SQLTYPE for Oracle Database releases 10.2 and 11.1.

Table 1–45 XML_SCHEMA_SIMPLE_TYPES View

Column	Description
OWNER	User who owns the type
XML_SCHEMA_URL	URL of schema within which the type is defined
TARGET_NAMESPACE	Namespace of the type
SIMPLE_TYPE_NAME	Name of the simple type
SIMPLE_TYPE	Actual XMLType of the type
MAINTAIN_DOM	XDB annotation for maintainDOM
SQL_TYPE	XDB annotation for sqlType
SQL_SCHEMA	XDB annotation for sqlSchema
DEFAULT_TABLE	XDB annotation for defaultTable
SQL_NAME	XDB annotation for sqlName
SQL_COL_TYPE	XDB annotation for sqlColType
STORE_VARRAY_AS_TABLE	XDB annotation for storeVarrayAsTable

XML_SCHEMA_SUBSTGRP_HEAD

XML_SCHEMA_SUBSTGRP_HEAD view displays the heads of substitution groups.

Table 1–46 XML_SCHEMA_SUBSTGRP_HEAD View

Column	Description
OWNER	User who owns the element
XML_SCHEMA_URL	URL of schema that contains the element
TARGET_NAMESPACE	Namespace of the element
ELEMENT_NAME	Name of the element
ELEMENT	Actual XML fragment of the element

XML_SCHEMA_SUBSTGRP_MBRS

XML_SCHEMA_SUBSTGRP_MBRS view displays all members of substitution groups.

Table 1–47 XML_SCHEMA_SUBSTGRP_MBRS View

Column	Description
OWNER	User who owns the element
SCHEMA_URL	URL of schema that contains the element
TARGET_NAMESPACE	Namespace of the element
ELEMENT_NAME	Name of the element
ELEMENT	Actual XML fragment of the element
HEAD_OWNER	User who owns the head element for the current element
HEAD_XML_SCHEMA_URL	URL of schema that contains the head element
HEAD_TARGET_NAMESPACE	Namespace of the head element
HEAD_ELEMENT_NAME	Name of the head element
HEAD_ELEMENT	Actual XML type of the head element