Oracle9*i* Database

Documentation Release Notes

Release 2 (9.2.0.2)

September 13, 2002

Part No. B10280-01

These Release Notes replace the Documentation Addendum published with Release 2 (9.2.0.1).

The following manuals have been reissued:

Part Number	Title
A96653-02	*Oracle Data Guard Concepts and Administration*
A96673-02	*Oracle Enterprise Manager Configuration Guide*
A96676-02	*Oracle Intelligent Agent User's Guide*
A95961-02	*Oracle9i Data Mining Concepts*
A96530-02	*Oracle9i Database Migration*
A96531-02	*Oracle9i Database New Features*
A96533-02	*Oracle9i Database Performance Tuning Guide and Reference*
A96536-02	*Oracle9i Database Reference*
A96580-02	*Oracle9i Net Services Administrator's Guide*
A96581-02	*Oracle9i Net Services Reference Guide*
A95295-02	*Oracle9i OLAP User's Guide*
A96600-02	*Oracle9i Real Application Clusters Setup and Configuration*
A96540-02	*Oracle9i SQL Reference*
A96571-02	*Oracle9i Streams*
A96620-02	*Oracle9i XML Database Developer's Guide - Oracle XML DB*

These Release Notes contain corrections to the following documents:

- Oracle9i Database Administrator's Guide
- Oracle9i Database Concepts
- Oracle9i Database Globalization Support Guide
- Oracle9i Database Reference
- Oracle9i Data Mining Administrator's Guide
- Oracle9i SQL Reference
- Oracle9i Streams
- Oracle9i Supplied PL/SQL Packages and Types Reference
- Oracle9i XML Database Developer's Guide - Oracle XML DB
- Oracle9i XML Developer's Kits Guide - XDK
- Oracle Call Interface Programmer's Guide
- Oracle Text Reference
- Backup and Recovery Documentation

See Also: Please refer to the product Release Notes for changes in functionality in Release 9.2.0.2.

Oracle9*i* Database Administrator's Guide

These are corrections to the *Oracle9i Database Administrator's Guide*.

Managing Undo Space

In Chapter 13, "Managing Undo Space", in the "Switching Undo Tablespaces" section, the following paragraph is incorrect:

> If the parameter value for `UNDO TABLESPACE` is set to " (two single quotes), the current undo tablespace will be switched out without switching in any other undo tablespace. This can be used, for example, to unassign an undo tablespace in the event that you want to revert to manual undo management mode.

There is no way to dynamically revert to manual undo management mode. The paragraph should read as follows:

> If the parameter value for `UNDO TABLESPACE` is set to " (two single quotes), then the current undo tablespace will be switched out and the next available undo tablespace will be switched in. Use this statement with care, because if there is no available undo tablespace available, the `SYSTEM` rollback segment is used and an alert message is written to the alert file to warn that the system is running without an undo tablespace.

Establishing Security Policies

Chapter 23, "Establishing Security Policies", contains a section titled "A Security Checklist". This checklist is intended to provide guidance for configuring Oracle9*i*, Release 2 (9.2), in a secure manner, thus protecting the database itself from attack. However, the checklist has been updated since the close date for publication of the manual. Please refer to the following URL for the most recent version of the security checklist:

`http://otn.oracle.com/deploy/security/oracle9i/pdf/9iR2_checklist.pdf`

DISPATCHERS Initialization Parameter

In this release, the `SCOPE` clause is not supported for the `DISPATCHERS` initialization parameter in an `ALTER SYSTEM` statement. This fact is not mentioned in Chapter 2, in the section on managing initialization parameters.

To reconfigure the `DISPATCHERS` parameter for the current instance, issue an `ALTER SYSTEM SET DISPATCHERS` statement without the `SCOPE` clause. To make the new configuration persistent, specify a setting for the `DISPATCHERS` parameter in your initialization parameter file before starting the database. If you are using a server parameter file (spfile) to start up the database, then you need to re-create your spfile from the initialization parameter file by issuing a `CREATE SPFILE` statement.

See Also: *Oracle9i Database Administrator's Guide*

Oracle9*i* Database Concepts

In Chapter 1 of *Oracle9i Database Concepts*, the section titled "Oracle Internet File System" no longer pertains to this manual. Oracle Internet File System is not included with the database release 9.2.

See Also: *Oracle9i Database Concepts*

Oracle9*i* Database Globalization Support Guide

Creating a New Character Set Definition with the Oracle Locale Builder When you create a new character set definition for a multibyte character set, base the new character set on an existing 8-bit or multibyte character set. Do not base it on a 7-bit character set because classification verification for multibyte character sets does not apply to 7-bit character sets.

See Also: *Oracle9i Database Globalization Support Guide*

Oracle9*i* Database Reference

In this release, the `SCOPE` clause is not supported for the `DISPATCHERS` initialization parameter in an `ALTER SYSTEM` statement. This fact is not mentioned in Chapter 1, "Initialization Parameters," in the section on `DISPATCHERS`.

To reconfigure the `DISPATCHERS` parameter for the current instance, issue an `ALTER SYSTEM SET DISPATCHERS` statement without the `SCOPE` clause. To make the new configuration persistent, specify a setting for the `DISPATCHERS` parameter in your initialization parameter file before starting the database. If you are using a server parameter file (spfile) to start up the database, then you need to re-create your spfile from the initialization parameter file by issuing a `CREATE SPFILE` statement.

See Also: *Oracle9i Database Reference*

Oracle9*i* Data Mining Administrator’s Guide

Please note changes with regard to the following topics:

- Sharing an Oracle9*i* Data Mining (ODM) Instance
- Stopping an ODM Task
- Starting and Stopping the ODM Task Monitor
- Developer Project for the Sample Programs
- Need for Compatible Character Sets
- Cannot Import Naive Bayes PMML Models

Sharing an ODM Instance

In a classroom situation, you may have several users who need to use ODM at the same time. The simplest solution is for each user to have a separate instance of Oracle9*i* Enterprise Edition with the ODM option installed on his or her system.

If it is necessary for users on different client systems to share one instance of Oracle9*i* and ODM, you must be aware of the following considerations:

1. Ensure that ODM is properly configured: this means unlocking and assigning passwords to the accounts ODM and ODM_MTR in an Enterprise Edition of Oracle9*i*.
2. Install Oracle9*i* JDeveloper on each of the client computers and download and install from Oracle Technology Network (`http://otn.oracle.com`) the JDeveloper project package that contains the ODM sample programs and data, as well as all the required libraries.
3. Each user accesses Oracle9*i* as the user ODM. In each user’s `Sample_Global.property` file, modify the Thin Client information to list the server hostname, port number, and SID; the user name is ODM and password is ODM’s password (as assigned during configuration). The HTTP server is not required.
4. Establish DB Connections from each JDeveloper instance to the schemas ODM and ODM_MTR (using the same connection information that was inserted into the Global property file). This allows the viewing of the schema tables and the execution of SQL*Plus queries from within JDeveloper, so no operating system-level access is needed.
5. Each user must use a unique name for *every* Java object and database table or view created. This is a substantial requirement; as an indication of the extent of this requirement, go to the Oracle9*i* server and look in `$ORACLE_HOME/dm/demo/sample` (UNIX) or `%ORACLE_`

`HOME\dm\demo\sample` (Windows) for all the property files for the ODM sample programs and note how many instances there are of a user-defined object name.

Stopping an ODM Task

If you stop an executing program that uses ODM, you must also stop the ODM task that was initiated by the program. For example, suppose that you start one of the ODM sample programs and then decide that you do not want to run that program after all. If you interrupt the program with Control-C, you will not stop the ODM task that the program started.

Follow these steps to stop an ODM task that was started by a program that is no longer running:

1. Query ODM_MINING_TASK for the Task ID. Suppose the Task ID is 999. (You can verify that the task is still running by querying ODM_MINING_TASK_STATE for that Task ID. If the state is not TERMINATED, the task is still running.)
2. Stop task 999 by setting the value for CLEAN FLAG in ODM_MINING_TASK to 1 where Task ID = 999.
3. Wait a few seconds (at least). Now when you query ODM_MINING_TASK_STATE for the Task ID, you should see the value TERMINATED.

Starting and Stopping the ODM Task Monitor

This corrects the *Oracle9i Data Mining Administrator's Guide*. The commands in step 4 of Section 3.2.1.1, in step 2 of Section 3.2.1.2, and in Section 4.8 all contain a typographical error. The correct commands are described in this section.

To start the ODM task monitor, log in as ODM user and start the ODM Monitor with the following SQL*Plus command:

```
exec ODM_START_MONITOR
```

To stop the ODM task monitor, log in as ODM user and stop the ODM Monitor with the following SQL*Plus command:

```
exec ODM_STOP_MONITOR
```

Developer Project for the Sample Programs

You can download from the Data Mining area in Oracle Technology Network (`http://otn.oracle.com`) an Oracle9*i* JDeveloper project for the ODM sample programs. After you download and install the project, you can modify and execute the ODM sample programs using Oracle9*i* JDeveloper.

The readme file included in the download explains how to install the project and how to compile and execute the ODM sample programs using JDeveloper.

Need for Compatible Character Sets

When using ODM in a shared JVM environment, such as when integrated with a servlet application, all connections made to an ODM server (also known as a Data Mining Server) must be based on databases with compatible character sets. Otherwise, string length tests conducted in the JVM may not recognize these differences, allowing data to pass to the database, which could result in server-side failures.

Cannot Import Naive Bayes PMML Models

Importing Naive-Bayes PMML models fails when TargetValueCount element's count attribute contains a real number. The count attribute's value for the TargetValueCount element in a Naive-Bayes model most commonly contains an integer value; however, the PMML specification allows for real numbers. If the document to be imported contains real numbers for this attribute, the import fails. You should instead convert the values of the count attribute to integers before performing the PMML model import.

See Also: *Oracle9i Data Mining Concepts*

Oracle9*i* SQL Reference

In this release, the `SCOPE` clause is not supported for the `DISPATCHERS` initialization parameter in an `ALTER SYSTEM` statement. This fact is not mentioned in Chapter 10, in the section on the `DISPATCHERS` initialization parameter.

To reconfigure the `DISPATCHERS` parameter for the current instance, issue an `ALTER SYSTEM SET DISPATCHERS` statement without the `SCOPE` clause. To make the new configuration persistent, specify a setting for the `DISPATCHERS` parameter in your initialization parameter file before starting the database. If you are using a server parameter file (spfile) to start up the database, then you need to re-create your spfile from the initialization parameter file by issuing a `CREATE SPFILE` statement.

See Also: *Oracle9i SQL Reference*

Oracle9*i* Streams

The following are corrections to the *Oracle9i Streams* manual (part number A96571-02):

- In Chapter 3, "Streams Staging and Propagation", in the "Queue Buffers" section, the following sentence on page 3-20 is incorrect: "In a single database, all of the queue buffers combined can use up to 10% of SGA memory."

This sentence should be replaced with the following:

> In a single database, all of the queue buffers combined can use up to 10% of shared pool.

- In Chapter 16, "Other Streams Management Tasks", in the "Apply Process Behavior for LCRs Containing LOBs" section, the following sentence should be added on page 16-13:

  Do not modify LOB column data in an LCR. This includes DML handlers, error handlers, and rule-based transformation functions.

Oracle9*i* Supplied PL/SQL Packages and Types Reference

- In Chapter 8, "DBMS_CAPTURE_ADM," replace the description for the `force` parameter in the `STOP_CAPTURE` procedure with the following description:

  This parameter is reserved for future use. In the current release, valid `BOOLEAN` settings are ignored.

- In Chapter 14, "DBMS_DESCRIBE," Table 14-2, the description of the `length` parameter should read as follows: "For `%type` formal arguments, a length of `0` is returned." This is the result of a fix to bug 1402425.

- In Chapter 29, "DBMS_LOGSTDBY," the order of the parameters for the `INSTANTIATE_TABLE` procedure is incorrect. The correct parameter order is as follows:

```
DBMS_LOGSTDBY.INSTANTIATE_TABLE (
    schema_name  IN VARCHAR2,
    table_name   IN VARCHAR2,
    dblink       IN VARCHAR2);
```

  The `DBMS_LOGSTDBY.INSTANTIATE_TABLE` procedure does not support the BLOB datatype even though BLOB datatypes are supported by logical standby databases.

The description should state that the DBMS_LOGSTDBY package is case sensitive for schema and table names. Be careful to use the correct case when supplying schema and tables names to the DBMS_LOGSTDBY package. For example, the following statements show incorrect and correct syntax for a SKIP procedure that skips changes to OE.TEST.

Incorrect statement:

```
execute dbms_logstdby.skip('DML', 'oe', 'test', null);
```

Because the names are specified with lowercase characters, the transactions that update these columns will still be applied to the logical standby database.

Correct statement:

```
execute dbms_logstdby.skip('DML', 'OE', 'TEST', null);
```

- In Chapter 72, "DBMS_STREAMS," the descriptions for the CONVERT_ANYDATA_TO_LCR_DDL function and CONVERT_ANYDATA_TO_LCR_ROW function state that you can specify these functions in rule-based transformations when propagating logical change records (LCRs) from SYS.Anydata queues to typed queues. However, rule-based transformations are not supported currently for such propagations. Therefore, to configure LCR propagations between SYS.AnyData queues and typed queues, you must specify a transformation created using the DBMS_TRANSFORM package.
- In Chapter 73, "DBMS_STREAMS_ADM," Table 73-10, replace the description for the streams_type parameter in the REMOVE_RULE procedure with the following:

  The type of Streams rule, either capture, apply, or propagation.
- In Chapter 107, "JMS Types," the following constants should be listed under "Constants to Support the aq$_jms_message Type.":

  ```
  JMS_TEXT_MESSAGE CONSTANT BINARY_INTEGER;
  JMS_BYTES_MESSAGE CONSTANT BINARY_INTEGER;
  ```

  The following constants are not supported:

  ```
  JMS_STREAM_MESSAGE
  JMS_MAP_MESSAGE
  JMS_OBJECT_MESSAGE
  ```
- In Chapter 107, "JMS Types," the first sentence under "aq$_jms_message Type" should read as follows: "This type is the ADT used to store JMSText and JMSBytes message types." The JMSMap, JMSStream, and JMSObject types should not be included.

- In Chapter 107, "JMS Types," the syntax for the member procedures and functions incorrectly prepends DBMS_AQJMS. The syntax should read as follows:

```
LOOKUP_PROPERTY_NAME(
   new_property_name  IN  VARCHAR);

SET_REPLYTO(
   replyto IN SYS.AQ$_AGENT);

SET_TYPE(
   type IN VARCHAR);

SET_USERID(
   userid IN VARCHAR);

SET_APPID(
   appid IN VARCHAR);

SET_GROUPID(
   groupid IN VARCHAR);

SET_GROUPSEQ(
   groupseq IN INT);

CLEAR_PROPERTIES;

SET_BOOLEAN_PROPERTY(
   property_name  IN VARCHAR,
   property_value IN BOOLEAN);

SET_BYTE_PROPERTY(
   property_name  IN VARCHAR,
   property_value IN INT);

SET_SHORT_PROPERTY(
   property_name  IN VARCHAR,
   property_value IN INT);

SET_INT_PROPERTY(
   property_name  IN VARCHAR,
   property_value IN INT);

SET_LONG_PROPERTY(
   property_name  IN VARCHAR,
   property_value IN NUMBER);

SET_FLOAT_PROPERTY(
   property_name  IN VARCHAR,
   property_value IN FLOAT);
```

```
SET_DOUBLE_PROPERTY(
   property_name  IN VARCHAR,
   property_value IN DOUBLE PRECISION);

SET_STRING_PROPERTY(
   property_name  IN VARCHAR,
   property_value IN VARDHAR);

GET_REPLYTO(
   replyto OUT SYS.AQ$_AGENT);

GET_TYPE(
   type OUT VARCHAR);

GET_USERID(
   userid OUT VARCHAR);

GET_APPID(
   appid OUT VARCHAR);

GET_GROUPID(
   groupid OUT VARCHAR);

GET_GROUPSEQ(
   groupseq OUT INT);

GET_BOOLEAN_PROPERTY(
   property_name IN  VARCHAR)
  RETURN BOOLEAN;

GET_BYTE_PROPERTY(
   property_name IN  VARCHAR)
  RETURN  INT;

GET_SHORT_PROPERTY(
   property_name IN  VARCHAR)
  RETURN INT;

GET_INT_PROPERTY(
   property_name IN  VARCHAR)
  RETURN INT;

GET_LONG_PROPERTY(
   property_name IN  VARCHAR)
  RETURN NUMBER;

GET_FLOAT_PROPERTY(
   property_name IN  VARCHAR)
  RETURN FLOAT;
```

```
GET_DOUBLE_PROPERTY(
   property_name IN  VARCHAR)
  RETURN DOUBLE PRECISION;

GET_STRING_PROPERTY(
   property_name IN  VARCHAR)
  RETURN VARCHAR;

CONSTRUCT(
   mtype  IN INT)
  RETURN aq$_jms_message;

CONSTRUCT RETURN aq$_jms_text_message;

SET_TEXT(
   payload IN VARCHAR2);

SET_TEXT(
   payload IN CLOB);

GET_TEXT(
   payload OUT VARCHAR2);

GET_TEXT(
   payload OUT CLOB);

SET_BYTES(
   payload IN RAW);

SET_BYTES(
   payload IN BLOB);

GET_BYTES(
   payload OUT RAW);

GET_BYTES(
   payload OUT BLOB);
```

- In Chapter 108, "Logical Change Record Types," a new parameter, `use_old`, was added to the following member functions for the `SYS.LCR$_ROW_RECORD` type: `GET_LOB_INFORMATION`, `GET_VALUE`, and `GET_VALUES`. Refer to Chapter 16, "Other Streams Management Tasks," in *Oracle9i Streams* (A96571-02) for detailed information about the `use_old` parameter in these member functions

  See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference*

Oracle9*i* XML Database Developer’s Guide - Oracle XML DB

There are documentation corrections to the following chapters:

- Chapter 5: Structured Mapping of XMLType
- Chapter 7: Searching XMLType Data with Oracle Text
- Chapter 10: Generating XML Data from the Database
- Appendix A: Installing and Configuring Oracle XMLDB

Chapter 5: Structured Mapping of XMLType

The following corrections related to QueryRewrite with XML Schema-Based Structured Storage.

What Is Query Rewrite?

When the XMLType is stored in structured storage (object-relationally) using an XML schema and queries using XPath are used, they can potentially be rewritten directly to the underlying object-relational columns. This rewrite of queries can also potentially happen when queries using XPath are issued on certain non-schema-based XMLType views.

This enables the use of B*Tree or other indexes, if present on the column, to be used in query evaluation by the Optimizer. This query rewrite mechanism is used for XPaths in SQL functions such as existsNode(), extract(), extractValue(), and updateXML(). This enables the XPath to be evaluated against the XML document without having to ever construct the XML document in memory.

Note: Path queries that get rewritten are a subset of the set of supported XPath queries. As far as possible, queries should be written so that the query rewrite advantages are realized.

Example 1 Query Rewrite

For example a query such as:

```
SELECT VALUE(p) FROM MyPOs p
     WHERE extractValue(value(p),'/PurchaseOrder/Company’) = 'Oracle';
```

is trying to get the value of the `Company` element and compare it with the literal ’`Oracle`’. Since the `MyPOs` table has been created with XML schema-based structured storage, the `extractValue` operator gets rewritten to the underlying relational column that stores the company information for the `purchaseorder`.

Thus the preceding query is rewritten to the following:

```
SELECT VALUE(p) FROM MyPOs p
     WHERE p.xmldata.company = ’Oracle’;
```

See Also: Chapter 4, "Using XMLType".

If there was a regular index created on the `Company` column, such as:

```
CREATE INDEX company_index ON MyPos e
      (extractvalue(value(e),’/PurchaseOrder/Company’));
```

then the preceding query would use the index for its evaluation.

Consider the `XMLType` view `employee_xml` created using `SYS_XMLGEN()` over the object type `emp_t`:

```
CREATE TYPE Emp_t AS OBJECT (
  EMPNO NUMBER(4),
  ENAME VARCHAR2(10),
  MGR   NUMBER(4),
  HIREDATE DATE,
  SAL   NUMBER(7,2),
  COMM  NUMBER(7,2));

CREATE VIEW employee_xml OF XMLTYPE
  WITH OBJECT  ID
        (SYS_NC_ROWINFO$.EXTRACT('/ROW/EMPNO/text()').getnumberval()) AS
    SELECT SYS_XMLGEN(
        emp_t(e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal,
e.comm))
    FROM emp e;
```

Consider the following query to retrieve all employees with an employee number of 7934:

```
SELECT VALUE(p) FROM employee_xml p
     WHERE extractValue(value(p),'/ROW/EMPNO') = 7934;
```

This query is similarly trying to match the value of `EMPNO` with 12300. This query is rewritten as follows:

```
SELECT SYS_XMLGEN(
   emp_t(e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm))
“value(p)”
   FROM emp e WHERE emp_t(e.empno, e.ename, e.job, e.mgr, e.hiredate,
e.sal,
        e.comm).empno - 7934;
```

The WHERE clause is further optimized to give the following query:

```
SELECT SYS_XMLGEN(
   emp_t(e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal, e.comm))
“value(p)”
   FROM emp e where e.empno = 7934;
```

An index empno_index created on the empno column is picked up:

```
CREATE INDEX empno_index ON emp (empno);
```

The view may alternately be defined over an object view. The object view may either already exist or be created over the relational data. Consider the following object view created over existing relational data:

```
CREATE VIEW employee_ov OF emp_t WITH OBJECT ID (EMPNO) AS
   SELECT emp_t(e.empno, e.ename, e.job, e.mgr, e.hiredate, e.sal,
                e.comm)
   FROM emp e;
```

A non-schema-based XMLType view may be created as follows :

```
CREATE VIEW employee_xml_xv OF XMLTYPE WITH OBJECT  ID
   (SYS_NC_ROWINFO$.EXTRACT('/ROW/EMPNO/text()').getnumberval()) AS
    SELECT SYS_XMLGEN(value(p)) FROM employee_ov p;
```

When Does Query Rewrite Occur?

Query rewrite happens for the following SQL functions:

- extract()
- existsNode()
- extractValue
- updateXML

The rewrite happens when these SQL functions are present in any expression in a query, DML, or DDL statement. For example, you can use extractValue() to create indexes on the underlying relational columns.

What XPath Expressions Are Rewritten?

XPath expressions involving simple expressions with no wild cards or descendant axes get rewritten. The XPath may select an element or an attribute node. Predicates are supported and get rewritten into SQL predicates. In addition, wild cards and descendant axes are rewritten and supported with certain restrictions for XML schema-based XMLType tables and views when it can be determined that it can be mapped to a single unique XPath.

Table 1 lists the kinds of XPath expressions that can be translated into underlying SQL queries in this release.

Table 1 Supported XPath Expressions for Translation to Underlying SQL Queries

XPath Expression for Translation	Description
Simple XPath expressions: /PurchaseOrder/@PurchaseDate /PurchaseOrder/Company	Involves traversals over object type attributes only, where the attributes are simple scalar or object types themselves. The only axes supported are the child and the attribute axes.
Collection traversal expressions: /PurchaseOrder/Item/Part	Involves traversal of collection expressions. The only axes supported are child and attribute axes. Collection traversal is not supported if the SQL operator is used during `CREATE INDEX` or `updateXML()`.
Predicates: [Company="Oracle"] /ROW[DEPTNAME="ACCOUNTING"]	Predicates in the XPath are rewritten into SQL predicates. Predicates are not rewritten for `updateXML()`
List indexe: lineitem[1] /ROW/EMPLOYEES[1]	Indexes are rewritten to access the n'th item in a collection. These are not rewritten for `updateXML()`.

Supported XPath Constructs for Query Rewrites

The following XPath constructs get rewritten:

- Simple XPath traversals
- Predicates
- Descendant axes (XML schema-based only): Rewrites over descendant axis (//) are supported if the following conditions are met:
  - There is at least one XPath child or attribute access following the //
  - Only one descendant of the children can potentially match the XPath child or attribute name following the //. If the schema indicates that multiple descendants children can potentially match, and there is no unique path the // can be expanded to, then no rewrite is done.
  - None of the descendants have an element of type `xsi:anyType`

    - There is no substitution group that has the same element name at any descendant.

- Wildcards (XML schema-based only): Rewrites over wildcard axis (/*) are supported if the following conditions are met:
    - There is at least one XPath child or attribute access following the /*
    - Only one of the grandchildren can potentially match the XPath child or attribute name following the /*. If the schema indicates that multiple grandchildren can potentially match, and there is no unique path the /* can be expanded to, then no rewrite is done.
    - None of the children or grandchildren of the node before the /* have an element of type `xsi:anyType`
    - There is no substitution group that has the same element name for any child of the node before the /*.

Supported Non-Schema-Based XMLType Views

Queries over non-schema-based `XMLType` views using `SYS_XMLGEN` can potentially get rewritten to operate on the arguments to `SYS_XMLGEN`.

- The arguments to `SYS_XMLGEN` can be of any object type, and can themselves be columns of an object view. If the object type of the argument to `SYS_XMLGEN` contains embedded collections, then rewrite happens only if there is a single XML operator, such as `existsnode()`, `extract()`, or `extractvalue()` in the query. If there are multiple operators, then no rewrite is done.
- `SYS_XMLGEN` must not take any additional parameters, such as `XMLFormat`. The major advantage of non-schema-based `XMLType` views is that the data does not have to be migrated into new tables, so the data can be directly exposed as XML and queried as XML. A disadvantage is that a rich set of mappings to XML are best captured using XML schema, and only the canonical mapping of object types to XML is supported with non-schema-based `XMLType` views.

How are the XPaths Rewritten For Non-Schema-Based XMLType Views?

Consider the following `XMLType` view definition. This `XMLType` view definition is used as the running example for this subsection. This subsection explains how XPaths are rewritten for non-schema-based `XMLType` views.

```
create type emp_t as object (
  EMPNO NUMBER(4), ename VARCHAR2(10), job VARCHAR2(9), mgr NUMBER(4),
  HIREDATE DATE);
```

```
create type emp_list is varray(100) of emp_t;

create or replace type dept_t as object (
 "@DEPTNO"   NUMBER(2), DeptNAME VARCHAR2(14),  LOC VARCHAR2(13),
  employees emp_list);

create view dept_ov of dept_t with object id (deptname) as
       select deptno, dname, loc, CAST( MULTISET(
              select emp_t(empno, ename, job, mgr, hiredate)
              from emp e where e.deptno = d.deptno) AS emp_list)
 from dept d;

create view dept_xv of xmltype
    with object id(SYS_NC_
ROWINFO$.extract('/ROW/@DEPTNO').getnumberval()) as
       select  SYS_XMLGEN(VALUE(p)) FROM dept_ov p ;
```

Non-Schema-Based XMLType Views: Mapping for a Simple XPath

A rewrite for a simple XPath involves accessing the attribute corresponding to the XPath expression. Thus, rewrite of the expressions '/ROW/DEPTNAME', '/ROW/@DEPTNO' and '/ROW/EMPLOYEES' map to dname, deptno, and CAST(MULTISET(select emp_t(empno, …) from emp e where …) AS emplist) respectively.

For example, consider a query that extracts the department number dno from the department view dept_xv:

```
SELECT extractvalue(value(p),'/ROW/@DEPTNO') dno from dept_xv p ;
```

This gets rewritten into the following query:

```
SELECT SYS_ALIAS_1.DNO "DNO" FROM DEPT SYS_ALIAS_1;
```

Non-Schema-Based XMLType Views: Mapping for Scalar Nodes

An Xpath expression containing a text() operator may be rewritten into the underlying relational column. For example, consider a query that extracts the department name dname from the department view dept_xv:

```
SELECT extract(value(p),'/ROW/DEPTNAME'/text()) dname from dept_xv p ;
```

This gets rewritten into the following query :

```
SELECT SYS_ALIAS_1.DNAME "DNAME" FROM DEPT SYS_ALIAS_1;
```

Non-Schema-Based XMLType Views: Mapping of Predicates

Predicates are mapped to SQL predicate expressions. For example, consider a query that gets the number of departments called ACCOUNTING:

```
select count(*)  from dept_xv e where existsnode(value(e),
'/ROW[DEPTNAME="ACCOUNTING"]')  = 1;
```

This query gets rewritten to the following :

```
SELECT COUNT(*) "COUNT(*)" FROM DEPT "SYS_ALIAS_1"
WHERE "SYS_ALIAS_1"."DNAME"='ACCOUNTING' AND DEPT_T(….) IS NOT NULL;
```

Non-Schema-Based XMLType Views: Simple Collection Traversals

Simple traversals through a collection are rewritten using XMLAGG() to aggregate over the elements of the collection. Consider the following query extracts all the employee numbers from the department view. Since it needs to iterate over the elements of the collection of employees for each department, an XMLAGG() is added to aggregate over the final result.

```
SELECT extract(value(e), '/ROW/EMPLOYEES/EMP_T/EMPNO') empno from  dept_
xv e;
```

This is rewritten using XMLAGG() as follows :

```
SELECT (SELECT XMLAGG(SYS_XMLGEN(TO_CHAR(EMPNO))) “VALUE(P)”
     FROM EMP E WHERE EMPNO IS NOT NULL AND EMP_T(…) IS NOT NULL
     AND E.DEPTNO = “SYS_ALIAS1”.DEPTNO) “EMPNO”
   FROM DEPT  “SYS_ALIAS1”;
```

Non-Schema-Based XMLType Views: Collection Indexing

An XPath expression can also access a particular index of a collection For example, ‘/ROW/EMPLOYEES/EMP_T[1]/EMPNO’ is rewritten to get the first element in the employees collection of the department view, and then to get the employee number from the first element.

Non-Schema-Based XMLType Views: Predicates in Collections

An XPath expression that traverses through a collection might involve predicates. For example, the expression extract(value(p), ‘/ROW[EMPLOYEES/EMP_T/EMPNO > 7900]’) involves a comparison predicate (> 7900). XPath 1.0 defines such XPaths to be satisfied if any of the items of the collection satisfies the predicate. Thus, this expression is rewritten to retrieve all those rows that have an employee collection such that at least one employee in the collection has an employee number greater than 7900. An existential check using the EXISTS() clause gets the list of all such rows.

Chapter 7: Searching XMLType Data with Oracle Text

Highlighting Support for INPATH and HASPATH Operators Page 7-34 states that the `CTX_DOC.HIGHLIGHT` procedure does not support highlighting of XML documents with `INPATH` and `HASPATH` operators. While this is true for release 9.2, highlighting support has been added for these operators in 9.2.0.2. This added support is documented with examples on page 7-49 *Oracle9i XML Database Developer's Guide - Oracle XML DB*, release 9.2.0.2., under the heading "Using Oracle Text: Advanced Techniques".

Chapter 10: Generating XML Data from the Database

The following functions were not previously documented:

- `PROCEDURE setNullHandling(ctx IN ctxHandle, flag IN NUMBER);`
- `PROCEDURE useNullAttributeIndicator(ctx IN ctxHandle, attrind IN boolean := TRUE);`

setNullHandling The values for the flag of `setNullHandling` are:

- `DROP_NULLS CONSTANT NUMBER := 0`; This is the default setting.
- `NULL_ATTR CONSTANT NUMBER := 1;`
- `EMPTY_TAG CONSTANT NUMBER := 2;`
- `DROP_NULLS` leaves out the tag for null elements.
- `NULL_ATTR` sets `xsi:nil="true"`.
- `EMPTY_TAG` sets, for example, `<foo/>.`

useNullAttributeIndicator This is a short-cut for `setNullHandling(ctx, NULL_ATTR)`.

Appendix A: Installing and Configuring Oracle XMLDB

To re-install Oracle XML DB follow these steps:

1. Remove the dispatcher.
2. Drop user xdb.
3. Drop tablespace xdb.
4. Recreate tablespace xdb.
5. Execute `catnoqm.sql`.

6. Execute `catqm.sql`.

7. Execute `catxdbj.sql`.

> **See Also:** *Oracle9i XML Database Developer's Guide* - *Oracle XML DB*

Oracle9*i* XML Developer's Kits Guide - XDK

Please note the corrections to documentation of the following topics:

- XML-SQL Utility (XSU)
- XMLGEN API Has Been Deprecated

XML-SQL Utility (XSU)

In Chapter 8, "XML SQL Utility (XSU)", Example 3 should be replaced as it is unnecessarily complex. You must first call XSU, then position the first element's cursor. Also, there is no need for a scrollable cursor. XSU processes all rows following the current row in the resultset. If you position the cursor at the first row of the resultset, then XSU starts with row #2. To position the cursor before the first row use the `beforeFirst()` method. XSU scrolls the resultset on its own so you do not have to worry about breaking.

The following example should replace the existing Example 3. It shows how you can use the XSU for Java API to generate an XML page:

```
----------------- b.java -------------
import oracle.sql.*;
import oracle.jdbc.driver.*;
import oracle.xml.sql.*;
import oracle.xml.sql.query.*;
import oracle.xml.sql.dataset.*;
import oracle.xml.sql.docgen.*;

import java.sql.*;
import java.io.*;

public class b
{
  public static void main(String[] args) throws Exception
  {

@    DriverManager.registerDriver(new
oracle.jdbc.driver.OracleDriver());

    Connection conn =
      DriverManager.getConnection("jdbc:oracle:oci8:scott/tiger@");

    Statement stmt =
conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                          ResultSet.CONCUR_READ_ONLY);

    String sCmd = "SELECT ENAME FROM SCOTT.EMP";
    ResultSet rs = stmt.executeQuery(sCmd);

    OracleXMLQuery xmlQry = new OracleXMLQuery(conn, rs);
    xmlQry.keepObjectOpen(true);
    //xmlQry.setRowIdAttrName("");
    xmlQry.setRowsetTag("ROWSET");
    xmlQry.setRowTag("ROW");
    xmlQry.setMaxRows(20);

    //rs.beforeFirst();
    String sXML = xmlQry.getXMLString();
    System.out.println(sXML);
  }
}
```

XSU Dependencies

XML SQL Utility (XSU) depends on the following components:

- Database connectivity -- JDBC drivers. XSU can work with any JDBC driver but it is optimized for Oracle JDBC drivers. Oracle Corporation does not make any guarantee or provide support for the XSU running against non-Oracle databases.
- Oracle XML Parser, Version2 -- `xmlparserv2.jar`. `xmlparserv2.jar` is included in the Oracle9*i* installations. `xmlparserv2.jar` is also part of the XDK for java archive downloadable from Oracle Technology Network (OTN) Web site.
- XSU also depends on the classes included in `xdb.jar` and `servlet.jar`. These are present in Oracle9*i* installations. These are also included in the XDK for java archive downloadable from OTN.

Installing the XSU

XML SQL Utility (XSU) ships with the Oracle9*i* software CD, and is also part of XDK for Java available from OTN. XSU is comprised of the following two files:

- `$ORACLE_HOME/lib/xsu12.jar`: This contains the Java classes that constitute XSU. xsu12 requires a minimum of JDK1.2.x and JDBC2.x.

- `$ORACLE_HOME/rdbms/admin/dbmsxsu.sql`: This is the SQL script that builds the XSU PL/SQL API. Load `xsu12.jar` into the database before executing `dbmsxsu.sql`.

By default Oracle Universal Installer installs XSU on your hard drive at the locations specified in the previous paragraph. It also loads XSU into the database. If XSU is not installed during the initial Oracle installation, it can be installed later. You can either use Oracle Universal Installer to install the XSU and its dependent components, or you can download the latest XDK for Java from OTN.

To load the XSU into the database you must perform one the following steps, depending on how you installed XSU:

- Oracle Universal Installer installation: change to your `ORACLE_HOME` directory, then to `rdbms/admin`. Run `initxml.sql`.
- OTN download installation: change to the `bin` directory of the downloaded/expanded XDK archive. Run the `xdkload` script. If you are using Windows run `xdkload.bat`.

XMLGEN API Has Been Deprecated

Before the first XSU production release, that is, before Oracle8*i* Release 3 (8.1.7), XSU for PL/SQL API was named "XMLGEN". This must not be confused with a) the XML generation SQL function `SYS_XMLGEN`, b) the XML generation PL/SQL supplied package `DBMS_XMLGEN`, or c) `XMLGen()` the SQLX standard function. Note that when XSU was first offered as a production release in Oracle8*i* Release 3 (8.1.7), the "`XMLGEN`" package was deprecated. In other words, "`XMLGEN`" was never offered as part of Oracle8*i* Release 3 (8.1.7) production code although it continued to be shipped with the Oracle software. It was never documented.

"`XMLGEN`" replacements are `DBMS_XMLQuery`, used for XML generation, and `DBMS_XMLSave` used for DML and data manipulation. Oracle9*i* Release 2 (9.2) and higher no longer include "`XMLGEN`" with the Oracle software.

Although for this release, the "`XMLGEN`" "deprecated" package can still be downloaded from OTN as part of the XSU download, which in turn is part of the XDK download, you are recommended to migrate to the latest production packages `DBMS_XMLQuery` and `DBMS_XMLSave`. Migration is simple as the method names are identical. The difference is that the new XSU for PL/SQL API contains more methods. Note that all take the context handle as the first argument.

See Also: *Oracle10i XML Developer's Kit Guide* - *XDK*

Oracle Call Interface Programmer’s Guide

- In Chapter 2, "OCI Programming Basics", in the "Using PL/SQL in an OCI Program" section, the following note is added:

  When binding a PL/SQL `VARCHAR2` variable in OCI, the maximum size of the bind variable is 32512, because of the overhead of control structures.

- In Chapter 6, "Describing Schema Metadata", in the "Character Length Semantics Support in Describing" section, the following sentence is added:

  Calling `OCIAttrGet()` with attribute `OCI_ATTR_CHAR_SIZE` does not return data on stored procedure parameters because stored procedure parameters are not bounded.

- Chapter 13, "Object Cache Navigation" is rewritten to satisfy the following bug report:

  The object cache is logically partitioned by "connection", that is, service context. As a result of this the OCI code is written to expect that all object types (TDOs) and table definitions are fetched for each service context that uses them. Unless the cache is low and objects are being aged out, programs that fetch all TDOs and tables on one service context but use them on others, generally work as you would like. However once TDOs and tables are aged out of the cache, unexpected behavior can occur. This can lead to internal errors.

  The section "The Object Cache and Memory Management" should explain from the start how the cache is logically partitioned and the importance of the service context. It does briefly mention that you have one copy of an object in the cache for each connection; but the rest of the chapter makes no reference to this again. For instance, under the pinning section, it says if the object is already in the cache it is retrieved. This is true only if it is in your logical partition of the cache, otherwise it is fetched from the database again.

  This has confused a number of customers in the past when they try to share objects between threads but end up with multiple copies.

  `OCITypeByName()`, `OCITypeArrayByName()`, and `OCIObjectPinTable()` definitions should be updated to explain that:

- The service context should correspond to that of the logical partition in which the TDO or table definition is used.

- If TDOs or tables are used across logical partitions, then the behavior is not known and may change between releases.

See Also: *Oracle Call Interface Programmer's Guide*

Oracle Text Reference

- The descriptions for the INPATH, HASPATH, and CTX_DOC.HIGHLIGHT, CTX_DOC.MARKUP do not mention that highlighting is not supported for INPATH and HASPATH in 9.2.

  In 9.2.0.2 this limitation has been removed. You can use CTX_DOC.HIGHLIGHT and CTX_DOC.MARKUP to highlight XML documents that have been queried with the INPATH and HASPATH operators. For more information, see Chapter 7 of the *Oracle9i XML Database Developer's Guide* - *Oracle XML DB*, release 9.2.0.2.
- The ctxsrv index maintenance utility and its related procedures and views are no longer supported. Specifically, the related procedure of CTX_ADM.SHUTDOWN is not supported as documented in Chapter 5, nor is the querying of the CTX_SERVERS view as documented in Appendix G.
- In Chapter 6, "CTX_CLS Package", in the "Syntax" section, the following columns currently support only nonnegative integer values:

  doc_id

  catdocid

  catid

  rescatid

  See Also: *Oracle Text Reference*

Backup and Recovery Documentation

The following are corrections to *Oracle9i Recovery Manager Reference*:

- In the description of the BLKSIZE parameter in allocOperandList, the choice of BLKSIZE must also be a multiple of the database initialization parameter DB_BLOCK_SIZE. This restriction holds true regardless of the block sizes of the datafiles being backed up.

  If you set BLKSIZE to something that is not a multiple of DB_BLOCK_SIZE, then RMAN issues errors when you try to allocate the channel.
- In "Performing Differential Incremental Backups: Example" in the examples section for the BACKUP command, the user is instructed to make a new backup after an ADD DATAFILE. You do not need to take a level 0 backup after adding a datafile. RMAN detects that a datafile was added, and when you take the next incremental backup, it takes the

incremental backup for all of the files except the one that was recently added, and for that file it automatically takes a level 0 backup.

The following notes are corrections to *Oracle9i Recovery Manager User's Guide*:

- In Chapter 5, "RMAN Concepts I: Channels, Backups, and Copies," the section "Detection of Physical Block Corruption" contains a misleading note that explains that RMAN cannot detect all types of block corruption. If you run RMAN with the following configuration, then it detects all types of corruption that are possible to detect:
    - In the initialization parameter file, set `DB_BLOCK_CHECKSUM=TRUE`
    - In the RMAN `BACKUP` and `RESTORE` commands, do *not* specify the `MAXCORRUPT` option, do *not* specify the `NOCHECKSUM` option, but *do* specify the `CHECK LOGICAL` option
- In Chapter 5, "RMAN Concepts I: Channels, Backups, and Copies," the section "Backup Retention Policies" is unclear with respect to incremental backups. The `REPORT OBSOLETE` and `DELETE OBSOLETE` commands work in two steps:
    - They identify the oldest full or level 0 backup or copy of every datafile that is not obsolete as defined by the retention policy (either `REDUNDANCY` or `RECOVERY WINDOW`). Every older full or level 0 backup is obsolete.
    - RMAN considers the following files obsolete: archived logs and incremental backups at level 1 or higher that are older than the oldest non-obsolete full backup. They are obsolete because there is no longer any full or level 0 backup to which they can possibly apply. Note that incremental backups at a level 1 or higher perform much the same function as archived logs (causing a full backup to move forward in time), so they are treated as archived logs for the purposes of the retention policy.

    See Also:

    - *Oracle9i Recovery Manager Reference*
    - *Oracle9i Recovery Manager User's Guide*

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.