XMLExport and XMLImport

Author: DBMaker Support Team

凌羣電腦
THE SYSCOM GROUP

Table of Content

1. Introduction

DBMaster supply interfaces for user to export XML data from Database or import XML data to database. We offer two system stored procedures to accomplish this functionality: XMLEXPORT and XMLIMPORT. We will introduce usage of stored procedure arguments with samples.

Notice that in both export and import xml, it can process multiple tables within one call.

1.1 Table schemas

We will test on DBSAMPLE4 with case-insensitive (DB_IDCap = 1) and use the following three tables:

```
create table whole_xml (duty_date date,duty_id char(6),duty_ontime char(8),duty_offtime char(8),xml_file clob);
```

```
create table SYSADM.DUTY_TABLE (
 DUTY_DATE    DATE not null    ,
 DUTY_ID    CHAR(6) not null    ,
 DUTY_ONTIME    CHAR(8) default null ,
 DUTY_OFFTIME    CHAR(8) default null )
 in DEFTABLESPACE    lock mode page    fillfactor 100 ;
```

```
create table SYSADM.EMPLOYEE (
 ID    CHAR(6) not null    ,
 NAME    VARCHAR(30) not null    ,
 PHOTO    LONG VARBINARY default null )
 in DEFTABLESPACE    lock mode page    fillfactor 100 ;
alter table SYSADM.EMPLOYEE primary key ( ID) in DEFTABLESPACE;
```

The contents of two tables are as follows:

```
dmSQL> select * from duty_table;
 DUTY_DATE    DUTY_ID DUTY_ONTIME DUTY_OFFTIME
========== ======= ========== ============
2008-08-05    B00119    08:10:20       18:30:01
2008-08-05    B00120    08:30:12       17:50:15
```

```
2008-08-05    B00121    08:32:02       18:50:00


3 rows selected
```

```
dmSQL> select * from employee;
   ID                      NAME                                PHOTO
====== ============================== ================================
B00119 Wang David                        ffd8ffe000104a46494600010101012c
B00120 Li Linda                         ffd8ffe000104a46494600010101012c
B00121 Zhang Rose                        ffd8ffe000104a46494600010101012c


3 rows selected
```

2. XMLExport

The XMLEXPORT system-stored procedure provides a programmable interface for users to export XML data from DBMaster. Only a SYSADM or a DBA can call these stored procedures. In addition, the execute privilege cannot be granted to other users because XMLEXPORT is a system-stored procedures.

SQL Syntax of the stored procedure is as follows:

```
XMLEXPORT(
            VARCHAR(256)    FILE_PATH,
            VARCHAR(256)    DB_TAG,
            VARCHAR(256)    XML_HEADER,
            VARCHAR(16000) OBJECT_STR,
            VARCHAR(256)    OPTION_STR,
            VARCHAR(256)    LOG_PATH
             )
```

Description of 6 arguments as follows:

Name	Type	Length (bytes)	Description	Case sensitivity
FILE_PATH	varchar	256	Full path of exported xml file	Depends on operating system
DB_TAG	varchar	256	Customized database tag	Yes (output has the same capitalization)
XML_HEADER	varchar	256	Customized xml header	Yes (output has the same capitalization)
OBJECT_STR	varchar	16000	Description string for exported objects	Depends on DBMaster setting
OPTION_FLAG	varchar	256	Description string for option flags	No
LOG_PATH	varchar	256	Full path of error log file on the client	Depends on operating system

2.1 The details on each argument

2.1.1 FILE_PATH

file_path is the full path of xml file exported. Since the generated file will be on server side, the specified file path must be server side file path.

For example, the string “D:\xml\xmlexport.xml” should be used for this argument.

2.1.2 DB_TAG

db_tag is customized a tag as root element. A NULL or empty string means default value (database name) is used.

2.1.3 XML_HEADER

Sometime, xml file is not readable. User might want to add stylesheet in the head of the xml file. So when viewing the xml file via browser, it can be stylish and easier to read. Or user might print some comment in the head of the xml file as well.

➲ **For example**

<?xml-stylesheet type="text/xsl" href="duty.xsl"?>

2.1.4 OBJECT_STR

```
object_str =:
        { <element> [; <element>]...}

<element>=:
        {TABLE_NAME | <select_query>} [#TABLE_TAG]
```

An <element> represents a table. The delimiter used between <element> is semi-colon (;). If the first token from <element> is "select" (case insensitive comparison), this <element> is seen as <select_query> [#TABLE_TAG]. Otherwise, this <element> is seen as TABLE_NAME [#TABLE_TAG].

If <element> = TALBE_NAME [#TABLE_TAG], all columns in this table will be selected and no customized column tag can be specified. That is to say, in the exported XML file, the names of column tags are the same as their corresponding table column names. The TABLE_TAG is for user to specify customized table tag. If no TABLE_TAG is there, table name in the database will be used as table tag name.

If users want to specify any customized column tag name, they can only use <select_query>[#TABLE_TAG] in the <element> string. The customized column tag names are specified by using column alias names in <select_query> statement. That is the user use "AS" in their <select_query>. For example, if the user uses the string , "select c1 as name, c2 as type from t2" as <select_query> statement, column c1 will become "name" tag and column c2 will become "type" tag in exported XML file.

NOTE: *make sure column tag name is not reserved word of DBMaster, or else calling the sp will return ERROR (6018) (syntax error or it's reserved word).*

➲ **For example**

We can use the string “select duty_date as DATE,duty_id as ID,duty_ontime as ONTIME,duty_offtime as OFFTIME from duty_table#DUTY;employee#EMPLOYEE” for this argument.

2.1.5 OPTION_FLAG

Users can specify multi-option by option string. Each option is separated by semicolon (;).

```
option_flag =: {[<attribute> [;<attribute>]...]}
```

➲ **For example**

If user wants column names to be treated as attributes and to capitalize all tag names, user can specify "column_as_attribute;capitalize_tag_name" in option string. If user does not specify a certain option, that option is not set.

The option flag string is case-insensitive.

Option flag string	Meaning if set	Meaning if not set
blob_in_separate_file	Blob/Clob column data is exported as a temp file separated from XML file. The name of that temp file is recorded in the exported dtd.	Blob/Clob column data is exported as part of the XML file.
column_as_attribute	Columns are exported as attributes instead of an element in XML file.	Columns are exported as an element in XML file.
capitalize_tag_name	All tag names are capitalized in the XML file.	The capitalization of all tag names stays the same as that of the corresponding names in database.
file_type_as_link	File type data content will not be exported. Only the name of the file is exported in the XML file.	File type data content will be exported as part of the XML file.
no_schema_dtd	Will not generate schema dtd along with the XML file generated.	Will generate corresponding DTD along with the XML file exported.

2.1.6 LOG_PATH

log_path is the full path of the log file where errors are recorded during exporting XML, and the log file is saved on the client machine.

For example, we will use "D:\xml\xmlexport.log" for this argument.

2.2 Examples

2.2.1 XML_HEADER IS EMPTY STRING

XML_HEADER the third argument can be empty string:

```
call XMLEXPORT(
'D:\xml\xmlexport.xml',
'DBSAMPLE4',
```

```
',
'duty_table#DUTY;employee#EMPLOYEE',
'column_as_attribute;capitalize_tag_name',
'D:\xml\xmlexport.log');
```

Two files will be generated in “D:\xml” folder: xmlexport.xml, and xmlexport.dtd.

The content of xmlexport.xml will be:

```
<?xml version="1.0" encoding="GB2312"?>

<!DOCTYPE DBSAMPLE4 SYSTEM "xmlexport.dtd">

<!--
      Generated by DBMaster2XML V1.0
      Database: DBSAMPLE4
-->

<DBSAMPLE4>
<!--   Query string : select * from duty_table            -->
  <DUTY DUTY_DATE="2008-08-05" DUTY_ID="B00119" DUTY_ONTIME="08:10:20"
DUTY_OFFTIME="18:30:01"/>
  <DUTY DUTY_DATE="2008-08-05" DUTY_ID="B00120" DUTY_ONTIME="08:30:12"
DUTY_OFFTIME="17:50:15"/>
  <DUTY DUTY_DATE="2008-08-05" DUTY_ID="B00121" DUTY_ONTIME="08:32:02"
DUTY_OFFTIME="18:50:00"/>
</DBSAMPLE4>
```

The content of xmlexport.dtd will be:

```
<?xml version="1.0" encoding="GB2312"?>

<!ELEMENT DUTY EMPTY>
  <!ATTLIST DUTY
                DUTY_DATE       CDATA #IMPLIED
                  DUTY_ID       CDATA #IMPLIED
             DUTY_ONTIME        CDATA #IMPLIED
            DUTY_OFFTIME        CDATA #IMPLIED
  >
<!ELEMENT DBSAMPLE4 (DUTY*)>
```

2.2.2 OPTION_FLAG DOES NOT SPECIFY OPTION

If user does not specify a certain option in OPTION_FLAG the fifth argument, that option is not set.

```
call XMLEXPORT(
'D:\xml\xmlexport.xml',
'DBSAMPLE4',
'',
'duty_table#DUTY',
'',
'D:\xml\xmlexport.log');
```

Two files will be generated in “D:\xml” folder: xmlexport.xml, and xmlexport.dtd.

Part content of the export file xmlexport.xml will be:

```
…
<DBSAMPLE4>
<!--   Query string : select * from duty_table          -->
  <DUTY>
    <DUTY_DATE>2008-08-05</DUTY_DATE>
    <DUTY_ID>B00119</DUTY_ID>
    <DUTY_ONTIME>08:10:20</DUTY_ONTIME>
    <DUTY_OFFTIME>18:30:01</DUTY_OFFTIME>
  </DUTY>
  <DUTY>
    <DUTY_DATE>2008-08-05</DUTY_DATE>
    <DUTY_ID>B00120</DUTY_ID>
    <DUTY_ONTIME>08:30:12</DUTY_ONTIME>
    <DUTY_OFFTIME>17:50:15</DUTY_OFFTIME>
  </DUTY>
  <DUTY>
    <DUTY_DATE>2008-08-05</DUTY_DATE>
    <DUTY_ID>B00121</DUTY_ID>
    <DUTY_ONTIME>08:32:02</DUTY_ONTIME>
    <DUTY_OFFTIME>18:50:00</DUTY_OFFTIME>
  </DUTY>
</DBSAMPLE4>
```

Part content of the export file xmlexport.dtd will be:

```
…
<!ELEMENT DUTY (DUTY_DATE, DUTY_ID, DUTY_ONTIME, DUTY_OFFTIME)>
```

```
<!ELEMENT DUTY_DATE (#PCDATA)>
  <!ATTLIST DUTY_DATE
        TYPE CDATA #FIXED "SQL_DATE"
        NAME CDATA #FIXED "DUTY_DATE"
        STORAGE CDATA #FIXED "13"
        ISNULL (true|false) 'false'
        xml:space (default|preserve) 'preserve'
  >
<!ELEMENT DUTY_ID (#PCDATA)>
  <!ATTLIST DUTY_ID
        TYPE CDATA #FIXED "SQL_CHAR"
        NAME CDATA #FIXED "DUTY_ID"
        LENGTH CDATA #FIXED "6"
        ISNULL (true|false) 'false'
        xml:space (default|preserve) 'preserve'
  >
…
<!ELEMENT DBSAMPLE4 (DUTY*)>
```

2.2.3 OPTION_FLAG DOES NOT GENERATE SCHEMA DTD

OPTION_FLAG the fifth argument can specify option, so we will use "no_schema_dtd" for this argument:

```
call XMLEXPORT(
'D:\xml\xmlexport.xml',
'DBSAMPLE4',
'',
'duty_table#DUTY',
'no_schema_dtd',
'D:\xml\xmlexport.log');
```

Only the xmlexport.xml file will be generated in “D:\xml” folder, and its part content is as follows:

```
…
<DBSAMPLE4>
<!--   Query string : select * from duty_table          -->
  <DUTY>
    <DUTY_DATE>2008-08-05</DUTY_DATE>
    <DUTY_ID>B00119</DUTY_ID>
    <DUTY_ONTIME>08:10:20</DUTY_ONTIME>
```

```
    <DUTY_OFFTIME>18:30:01</DUTY_OFFTIME>
  </DUTY>
  <DUTY>
    <DUTY_DATE>2008-08-05</DUTY_DATE>
    <DUTY_ID>B00120</DUTY_ID>
    <DUTY_ONTIME>08:30:12</DUTY_ONTIME>
    <DUTY_OFFTIME>17:50:15</DUTY_OFFTIME>
  </DUTY>
  <DUTY>
    <DUTY_DATE>2008-08-05</DUTY_DATE>
    <DUTY_ID>B00121</DUTY_ID>
    <DUTY_ONTIME>08:32:02</DUTY_ONTIME>
    <DUTY_OFFTIME>18:50:00</DUTY_OFFTIME>
  </DUTY>
</DBSAMPLE4>
```

2.2.4 OBJECT_STR SPECIALS MULTIPLE TABLES

OBJECT_STR the fourth argument can special multiple tables:

```
OBJECT_STR = TALBE_NAME [#TABLE_TAG]; TALBE_NAME [#TABLE_TAG]…
```

In this example, OBJECT_STR = TALBE_NAME [#TABLE_TAG], so all columns in two tables will be selected and no customized column tag can be specified.

```
call XMLEXPORT(
'D:\xml\xmlexport.xml',
'DBSAMPLE4',
'',
'duty_table#DUTY;employee#EMPLOYEE',
'no_schema_dtd',
'D:\xml\xmlexport.log');
```

Only the xmlexport.xml file will be generated in “D:\xml” folder, and its part content is as follows:

```
…
<DBSAMPLE4>
<!--   Query string : select * from duty_table          -->
  <DUTY>
    <DUTY_DATE>2008-08-05</DUTY_DATE>
    <DUTY_ID>B00119</DUTY_ID>
    <DUTY_ONTIME>08:10:20</DUTY_ONTIME>
    <DUTY_OFFTIME>18:30:01</DUTY_OFFTIME>
```

```
  </DUTY>
  <DUTY>
    <DUTY_DATE>2008-08-05</DUTY_DATE>
    <DUTY_ID>B00120</DUTY_ID>
    <DUTY_ONTIME>08:30:12</DUTY_ONTIME>
    <DUTY_OFFTIME>17:50:15</DUTY_OFFTIME>
  </DUTY>
  <DUTY>
    <DUTY_DATE>2008-08-05</DUTY_DATE>
    <DUTY_ID>B00121</DUTY_ID>
    <DUTY_ONTIME>08:32:02</DUTY_ONTIME>
    <DUTY_OFFTIME>18:50:00</DUTY_OFFTIME>
  </DUTY>
<!--   Query string : select * from employee          -->
  <EMPLOYEE>
    <ID>B00119</ID>
    <NAME>Wang David</NAME>
    <PHOTO>ffd8ffe000104a46494600010101012c012c0000ffd…</PHOTO>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>B00120</ID>
    <NAME>Li Linda</NAME>
    <PHOTO>ffd8ffe000104a46494600010101012c012c0000ffdb…</PHOTO>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>B00121</ID>
    <NAME>Zhang Rose</NAME>
    <PHOTO>ffd8ffe000104a46494600010101012c012c0000ffdb0…</PHOTO>
  </EMPLOYEE>
</DBSAMPLE4>
```

2.2.5 OBJECT_STR CUSTOMIZES COLUMN TAG

OBJECT_STR the fourth argument can specify any customized column tag name, you can use OBJECT_STR = <select_query>[#TABLE_TAG].

In this example, it only specifies two customized columns in DUTY_TABLE.

```
call XMLEXPORT(
'D:\xml\xmlexport.xml',
```

```
'DBSAMPLE4',
'',
'select duty_date as work_date,duty_id as exmple_id from duty_table#DUTY',
'no_schema_dtd',
'D:\xml\xmlexport.log');
```

Only the xmlexport.xml file will be generated in “D:\xml” folder, and its part content is as follows:

```
…
<DBSAMPLE4>
<!--   Query string : select duty_date as work_date,duty_id as exmple_id from duty_table          -->
  <DUTY>
    <WORK_DATE>2008-08-05</WORK_DATE>
    <EXMPLE_ID>B00119</EXMPLE_ID>
  </DUTY>
  <DUTY>
    <WORK_DATE>2008-08-05</WORK_DATE>
    <EXMPLE_ID>B00120</EXMPLE_ID>
  </DUTY>
  <DUTY>
    <WORK_DATE>2008-08-05</WORK_DATE>
    <EXMPLE_ID>B00121</EXMPLE_ID>
  </DUTY>
<!--   Query string : select * from employee          -->
  <EMPLOYEE>
    <ID>B00119</ID>
    <NAME>Wang David</NAME>
    <PHOTO>ffd8ffe000104a46494600010101012c012c0000ffd…</PHOTO>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>B00120</ID>
    <NAME>Li Linda</NAME>
    <PHOTO>ffd8ffe000104a46494600010101012c012c0000ffdb…</PHOTO>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>B00121</ID>
    <NAME>Zhang Rose</NAME>
    <PHOTO>ffd8ffe000104a46494600010101012c012c0000ffdb0…</PHOTO>
```

```
  </EMPLOYEE>
</DBSAMPLE4>
```

2.2.6 XML_HEADER

User has a predefined xml style sheet file duty.xsl:

```
<?xml version="1.0" encoding="ASCII"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
  <body>
    <h2>Duty List</h2>
    <table border="1">
    <tr bgcolor="#9acd32">
      <th align="left">Date</th>
      <th align="left">ID</th>
      <th align="left">ON</th>
      <th align="left">OFF</th>
    </tr>
    <xsl:for-each select=" DBSAMPLE4/DUTY">
    <tr>
      <td><xsl:value-of select="DUTY_DATE"/></td>
      <td><xsl:value-of select="DUTY_ID"/></td>
      <td><xsl:value-of select="DUTY_ONTIME"/></td>
      <td><xsl:value-of select="DUTY_OFFTIME"/></td>
    </tr>
    </xsl:for-each>
    </table>
    <h2>Employee List</h2>
    <table border="1">
    <tr bgcolor="#888899">
      <th align="left">Employee ID</th>
      <th align="left">Employee Name</th>
    </tr>
    <xsl:for-each select=" DBSAMPLE4/EMPLOYEE">
    <tr>
```

```
        <td><xsl:value-of select="ID"/></td>
        <td><xsl:value-of select="NAME"/></td>
      </tr>
      </xsl:for-each>
      </table>   </body>
   </html>
</xsl:template>
</xsl:stylesheet>
```

Then call XMLExport:

```
call XMLEXPORT(
'D:\xml\duty.xml',
'',
'<?xml-stylesheet type="text/xsl" href="duty.xsl"?>',
'duty_table#DUTY;select id as id,name as name from employee#EMPLOYEE',
'no_schema_dtd',
'D:\xml\duty.log');
```

Part content of output xml file would be duty.xml:

```
…
<?xml-stylesheet type=”text/xsl” href=”duty.xsl”?>
<DBSAMPLE4>
<!--   Query string : select * from duty_table            -->
   <DUTY>
      <DUTY_DATE>2008-08-05</DUTY_DATE>
      <DUTY_ID>B00119</DUTY_ID>
      <DUTY_ONTIME>08:10:20</DUTY_ONTIME>
      <DUTY_OFFTIME>18:30:01</DUTY_OFFTIME>
   </DUTY>
   <DUTY>
      <DUTY_DATE>2008-08-05</DUTY_DATE>
      <DUTY_ID>B00120</DUTY_ID>
      <DUTY_ONTIME>08:30:12</DUTY_ONTIME>
      <DUTY_OFFTIME>17:50:15</DUTY_OFFTIME>
   </DUTY>
   <DUTY>
      <DUTY_DATE>2008-08-05</DUTY_DATE>
      <DUTY_ID>B00121</DUTY_ID>
```

```
    <DUTY_ONTIME>08:32:02</DUTY_ONTIME>
    <DUTY_OFFTIME>18:50:00</DUTY_OFFTIME>
  </DUTY>
<!--  Query string : select id as id,name as name from employee          -->
  <EMPLOYEE>
    <ID>B00119</ID>
    <NAME>Wang David</NAME>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>B00120</ID>
    <NAME>Li Linda</NAME>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>B00121</ID>
    <NAME>Zhang Rose</NAME>
  </EMPLOYEE>
</DBSAMPLE4>
```

3. XMLImport

The XMLIMPORT system-stored procedure provides a programmable interface for users to import XML data to DBMaster. Only a SYSADM or a DBA can call these stored procedures. In addition, the execute privilege cannot be granted to other users because XMLIMPORT is a system-stored procedures.

XMLIMPORT will import tables from XML files to tables in DBMaster. When importing from an XML file, users can simply store the whole XML file in the database or analyzing content of the XML file and importing data into tables). The XML file being imported must be on the server and the log file generated during the importing of an XML file is saved on the client machine.

Syntax of the XMLIMPORT stored procedure is as follows:

```
XMLIMPORT(
        VARCHAR(256)   FILE_PATH,
        VARCHAR(16000) OBJECT_STR,
         VARCHAR(256)   OPTION_STR,
         VARCHAR(256)   LOG_PATH)",
```

Description of 4 arguments as follows:

Name	**Type**	**Length (bytes)**	**Description**	**Case sensitivity**
FILE_PATH	varchar	256	Full path of exported xml file	Depends on operating system
OBJECT_STR	varchar	16000	Description string for exported objects	XML tags are case sensitive; table names and table column names depends on DBMaster setting
OPTION_FLAG	varchar	256	Description string for option flags	No
LOG_PATH	varchar	256	Full path of error log file on the client	Depends on operating system

➲ **For example:** analyzing the file content and importing data into tables

Assume that we have a XML file xmlimport.xml in ‘D:\xml’ folder. The file is listed as follows:

```
<ROOT>
   <EMPLOYEE>
        <TITLE>
       <TAG1>2008-08-05</TAG1>
       <TAG2>B00119</TAG2>
```

```
        <TAG3>08:10:20</TAG3>
        <TAG4>18:30:01</TAG4>
      </TITLE>
      <TITLE>
        <TAG1>2008-08-05</TAG1>
        <TAG2>B00120</TAG2>
        <TAG3>08:30:12</TAG3>
        <TAG4>17:50:15</TAG4>
      </TITLE>
      <TITLE>
        <TAG1>2008-08-05</TAG1>
        <TAG2>B00121</TAG2>
        <TAG3>08:32:02</TAG3>
        <TAG4>18:50:00</TAG4>
      </TITLE>
    </EMPLOYEE>
</ROOT>
```

We will import the date recorded in the xmlimport.xml file into DUTY_TABLE table.

From the content of the XML file, under the <EMPLOYEE> element, there is one sub-element. Thinking the imported data architecture, we can map <EMPLOYEE> element as database level, and <TITLE> as table level, so <TAG1>, <TAG2>, <TAG3> and <TAG4> is column tag names.

So execute the following command can import data recorded in the XML file:

```
CALL XMLIMPORT(
'D:\XML\XMLIMPORT.XML',
'/ROOT/EMPLOYEE/TITLE(TAG1,TAG2,TAG3,TAG4)#duty_table(duty_date,duty_id,duty_ontime,
duty_offtime)',
'',
'D:\XML\XMLIMPORT.LOG');
```

Check importing result:

```
dmSQL> select * from duty_table;
DUTY_DATE    DUTY_ID DUTY_ONTIME DUTY_OFFTIME
========== ======= ========== ============
2008-08-05   B00119   08:10:20      18:30:01
2008-08-05   B00120   08:30:12      17:50:15
2008-08-05   B00121   08:32:02      18:50:00

3 rows selected
```

3.1 The details on each argument

Since XMLIMPORT is more complex, we will describe arguments one by one.

3.1.1 FILE_PATH

file_path is the full path of XML file which will be imported, and the XML file being imported must locate in Server side.

3.1.2 OBJECT_STR

object_str argument is used to describe imported objects. This information includes document levels, the mapping between customized column tag names and inserted table column names as well as the mapping between customized table tag name and table name in the database. The format is as follows:

```
object_str =:
        { <table_element> [; <table_element>]...}

<table_element> =
        { <document mapping information>#<table mapping information> }

<document mapping information> =:
        {<document level string>[(<column tag names>)]

<document level string> =: {/<level1> [/<level2>/.....]}

<column tag names> =: {<tag1> [, <tag2>]...}

<table mapping information> =: <table import definition>

<table import definition> =: { <insert sql statement> | <target table name>[(<table column names>)] }

<insert sql statement> =: INSERT INTO <target table name> [(<table column names>)] VALUES
(<value list>)

<table column names> =: {<col1> [, <col2>] ...}

<value list> =: {<insert value>, <insert value>,...}

<insert value> =: {<constant> | <expression>}
```

In addition, if users want to store the whole XML file instead of parsing it and storing the content in tables, they should use special handling in <column tag names>, please refer to chapter 3.2.

NOTE: *all xml tags are case-sensitive, and the case-sensitivity of table names and table column names depends on DBMaster setting (DB_IDCAP).*

➲ **For example, the object_str can be the following format**

The object_str has several formats, because <table import definition> has two formats and user can specify or not specify <column tag names> in <document mapping information>, and specify or not specify <table column names>.

A. Use <insert sql statement> in <table import definition>:

```
'/DBSAMPLE4/DUTY(DUTY_DATE,DUTY_ID,DUTY_ONTIME,DUTY_OFFTIME)#insert into duty_table(duty_date,duty_id,duty_ontime,duty_offtime) values(?,?,?,?)'

or

'/DBSAMPLE4/DUTY#insert into duty_table(duty_date,duty_id,duty_ontime,duty_offtime) values(?,?,?,?)'
```

B. or use <target table name>[(<table column names>)] in <table import definition>:

```
'/DBSAMPLE4/DUTY(DUTY_DATE,DUTY_ID,DUTY_ONTIME,DUTY_OFFTIME)#duty_table(duty_date,duty_id,duty_ontime,duty_offtime)'
```

or not specify <column tag names>:

```
'/DBSAMPLE4/DUTY#duty_table(duty_date,duty_id,duty_ontime,duty_offtime)'
```

or not specify <column tag names> and <table column names>:

```
'/DBSAMPLE4/DUTY#duty_table'
```

Since object_str is more complex, we will explain syntax one by one.

3.1.2.1 <table_element>

XMLImport also can handle multi-tables. One <table_element> represents a table. Multiple tables are delimited by semicolon (;).

3.1.2.2 <document level string>

In <document level string>, users specify document levels from root to table level. String in <document level string> is case-sensitive, so user must to make sure it is same as tag in XML document.

➲ **For Example**

Assume that XML document is as follows:

```
<root>
    <database>
        <table1>
            <column1>…</column1>
            <column2>…</column2>
        </table1>
        <table2>
```

```
        </table2>
    </database>
</root>
```

In order to import data stored in <table1> tag of <database>, users must specify a <document level string> of "/root/database/table1".

In this case, "/ROOT/DATABASE/TABLE1" is illegal, because <document level string> is case-sensitive.

3.1.2.3 <column tag names>

In <column tag names>, users specify which column tags they want to insert into table. If no <column tag names> is specified, all column tags under a certain table tag are inserted, and DTD file must exist in the same location with imported XML file. String in <column tag names> is case-sensitive.

➲ **For example**

In the above case, <column tag names> is “column1, column2”. And “COLUMN1, COLUMN2” is illegal.

3.1.2.4 <table import definition>

```
<table import definition> =: { <insert sql statement> | <target table name>[(<table column names>)] }
```

In <table import definition>, users are allowed to use either the format of <insert sql statement> or TABLE_NAME [<table column names>].

3.1.2.4.1 When users use <insert sql statement>

```
<insert sql statement> =: INSERT INTO <target table name> [(<table column names>)] VALUES (<value list>)
```

<insert sql statement> is the same as the syntax for ordinary insert SQL statement.

NOTES:

1. If no <table column names> is specified, it is implied that users are trying to insert all columns in the target table.
2. if there is <column tag names> in <document mapping information>, the number of column tags specified in <column tag names> must be equal to the number of host variables in <value list>
3. If there is no <column tag names> in <document mapping information>, it is implied that all column tags under base element are used for insert into target table and the sequence of xml column tags matches the sequence of what users want in <table column names> and the schema information in DTD is used to check whether the number of (all) tags is equal to the number of host variables in <value list>.
4. The mapping among <table column names>, <value list> and <column tag names> in <document mapping information> must be appropriate. The <column tag names> is mapped to host variables in <value list>. The sequence of columns in <table column names> combined with the sequence values in <value list> and the sequence of tags <column tag names> decides what values are inserted into <value list>.
5. <value list> can be <constant> or <expression>.
6. Notice the sequence of <table column names>.

7. Empty tag names are not allowed

➲ **Example for point1**

Assume that object_str is "/root/book/order(tg1, tg2, tg3)#insert into t1 values (?, ?, ?)", it will insert 3 tags into all columns of our target table. The value of tg1 is inserted into c1 of t1, the value of tg2 is inserted into c2 of t1, and the value of tg3 is inserted into c3 of t1.

➲ **Example for point2**

Assume that object_str is "/root/book/order(tg1, tg2)#insert into t1 values (?,?, ?)", and t1 has three columns.

Calling XMLImport with the object_str will return error 5604 which is “no. of imported column and imported column count do not match”.

➲ **For example**

If the <table column names> is (c1, c2, c3), <value list> is (?,?,?) and <column tag names> is (tg1, tg2, tg3), the value in tg1 is used for insert into c1, the value in tg2 is used for insert into c2 and the value in tg3 is used for insert into c3.

➲ **For example**

Assume that table t1 has three columns, c1, c2 and c3 and that we have three tags tg1, tg2, tg3 in the xml element we are trying to import.

If object_str is "/root/book/order(tg1, tg2, tg3)#insert into t1 values (?,?, acos(1))", the result of acos(1) is inserted into c3 of t1.

If object_str is "/root/book/order(tg1, tg2, tg3)#insert into t1 values (?, +3, 5)", The column c2 is inserted the value of tag tg2 plus 3 and the column c3 is inserted a constant value of 5.

➲ **For example**

If the object_str is "/root/book/order(tg3, tg2, tg1)#insert into t1 (c2, c1, c3) values (?, ?, ?)", the value of tg3 is inserted into c2, the value of tg2 is inserted into c1 and the value of tg1 is inserted into c3.

➲ **For example**

An object string "/root/book/order(tg1, , tg2)#insert into t2 (c1, c2) values (?, ?, ?)" is illegal.

An object string "/root/book/order(tg1, tg2, tg3)#insert into t2 (c1, c2, c3, c4) values (?, ?, ?,) is legal. What is inserted into c4 of t2 depends on table schema information.

3.1.2.4.2 When users use <target table name>[(<table column names>)]

NOTES:

1. If users use this format, they must specify the table they want to insert in <target table name>. This <target table name> is mapped to the last level in <document level string>.
2. They cannot specify constant value or expression insert as they can when they use <insert sql statement> format.
3. If there is no <column tag names> specified in <document mapping information>, all column tags under base element are used for insert into target table and the sequence of xml column tags matches the sequence of what users want in <table column names> and the schema information in DTD is used to check whether the number of (all) tags is equal to the number of host variables in <value list>.

4. If no <table column names> is specified, all table columns are to be inserted. The schema information in DTD will be used to check whether the number of all tags under base element is equal to the number of all columns in target table.
5. Empty column tag names are not allowed, you must either specify all customized tag names or none of them at all.

➲ **For example**

Assume that table t1 has three columns, c1, c2 and c3 and that we have three tags tg1, tg2, and tg3 in the xml element, and order is table level.

object_str can be "/root/book/order(tg1, tg2, tg3)#t1(c1,c2,c3)", or

can be "/root/book/order#t1", but when using the second format, DTD with the schema information must exist.

3.1.3 OPTION_FLAG

```
option_flag =: {[<attribute> [;<attribute>]...]}
```

The option flag string is case-insensitive.

Option flag string	Meaning if set	Meaning if not set
column_as_attribute	Columns in the imported XML file are treated as attributes.	Columns are treated as elements in XML file.

3.1.4 LOG_PATH

log_path is the full path of the log file where errors are recorded during importing XML, and the log file is saved on the client machine.

For example, we will use "D:\xml\xmlimport.log" for this argument.

3.2 Store the whole XML file

The above description are all importing xml file by parsing the XML file and then storing the data recorded in XML file into tables. XMLIMPORT also can store the whole XML file into columns (data type of column must be large object, such as FILE, CLOB).

Users have to specify a "virtual tag" in <column tag names>. This special "virtual tag" is named "_XML_FILE_". If this "_XML_FILE_" is used as the column tag name, the columns represented by the column tags preceding this special "virtual tag" are used as key value. Also the mapped value in <value list> must be a single host variable without any further calculation.

➲ **For example**

The example is to import xmlexport.xml which is exported in chapter 2.2.3.

If the object_str is "/DBSAMPLE4/DUTY(DUTY_DATE,DUTY_ID,DUTY_ONTIME,DUTY_OFFTIME, _XML_FILE_)#whole_xml(duty_date,duty_id,duty_ontime,duty_offtime, xml_file)", the whole xml file will be inserted into xml_file column.

The part of result is:

```
dmSQL> select * from whole_xml;

 DUTY_DATE    DUTY_ID DUTY_ONTIME DUTY_OFFTIME        XML_FILE

========== ======= ========== ============ ================
```

```
2008-08-05    B00119    08:10:20        18:30:01        <?xml version="1…
…
```

3.3 Examples

3.3.1 IMPORT XML FROM CHAPTER 2.2.2

The example is to import xmlexport.xml (exist the xmlexport.dtd file in 'D:\xml' folder) which is exported in chapter 2.2.2. In this case, because DTD exists, so we do not need to specify <column tag names> in <document mapping information> and <table column names>.

So execute the following command to import xml:

```
CALL XMLIMPORT(
'D:\XML\XMLEXPORT.XML',
'/DBSAMPLE4/DUTY#duty_table',
'',
'D:\XML\XMLIMPORT.LOG');
```

3.3.2 IMPORT XML FROM CHAPTER 2.2.3

The example is to import xmlexport.xml which is exported in chapter 2.2.3. Let us analyze how to compose OBJECT_STR the second argument:

```
<document level string> = '/DBSAMPLE4/DUTY'
<column tag names> = 'DUTY_DATE,DUTY_ID,DUTY_ONTIME,DUTY_OFFTIME'
<table import definition> has two formats, please refer to the following examples.
```

1. <table import definition> =: { <target table name>[(<table column names>)] }

```
<table import definition>='duty_table(duty_date,duty_id,duty_ontime,duty_offtime)'
```

So execute the following command to import xml:

```
CALL XMLIMPORT(
'D:\XML\XMLEXPORT.XML',
'/DBSAMPLE4/DUTY(DUTY_DATE,DUTY_ID,DUTY_ONTIME,DUTY_OFFTIME)#duty_table(
duty_date,duty_id,duty_ontime,duty_offtime)',
'',
'D:\XML\XMLIMPORT.LOG');
```

2. <table import definition> =: { <insert sql statement>}

```
<insert sql statement> =' INSERT INTO duty_table(duty_date,duty_id,duty_ontime,duty_offtime)
VALUES(?,?,?,?)'
```

So execute the following command to import xml:

```
CALL XMLIMPORT(
'D:\XML\XMLEXPORT.XML',
```

```
'/DBSAMPLE4/DUTY(DUTY_DATE,DUTY_ID,DUTY_ONTIME,DUTY_OFFTIME)#INSERT
INTO duty_table(duty_date,duty_id,duty_ontime,duty_offtime) VALUES(?,?,?,?)',
'',
'D:\XML\XMLIMPORT.LOG');
```

3.3.3 IMPORT XML WITH MULTI-TABLE FROM CHAPTER 2.2.4

The example is to import xmlexport.xml which is exported in chapter 2.2.4. Let us analyze how to compose OBJECT_STR the second argument:

The example will import the data recorded in the xmlexport.xml file into the DUTY_TABLE and EMPLOYEE two tables. Assume that we want to import < DUTY> into DUTY_TABLE table and <EMPLOYEE > into EMPLOYEE table.

For < DUTY>:

```
<document level string> = '/DBSAMPLE4/DUTY'
<column tag names> = 'DUTY_DATE,DUTY_ID,DUTY_ONTIME,DUTY_OFFTIME'
<table import definition>='duty_table(duty_date,duty_id,duty_ontime,duty_offtime)'
```

For <EMPLOYEE >:

```
<document level string> = '/DBSAMPLE4/EMPLOYEE'
<column tag names> = 'ID,NAME,PHOTO'
<table import definition>='employee(id,name,photo)'
```

So execute the following command to import xml:

```
CALL XMLIMPORT(
'D:\XML\XMLEXPORT.XML',
'/DBSAMPLE4/DUTY(DUTY_DATE,DUTY_ID,DUTY_ONTIME,DUTY_OFFTIME)#duty_table(
duty_date,duty_id,duty_ontime,duty_offtime);
/DBSAMPLE4/EMPLOYEE(ID,NAME,PHOTO)#employee(id,name,photo)',
'',
'D:\XML\XMLIMPORT.LOG');
```

3.3.4 IMPORT XML FROM CHAPTER 2.2.5

The example is to import xmlexport.xml which is exported in chapter 2.2.5. Let us analyze how to compose OBJECT_STR:

```
<document level string> = '/DBSAMPLE4/DUTY'
<column tag names> = 'WORK_DATE,EXMPLE_ID'
<table import definition>='duty_table(duty_date,duty_id)'
```

1. The command with correct argument

So execute the following command to import xml:

```
CALL XMLIMPORT(
'D:\XML\XMLEXPORT.XML',
```

```
'/DBSAMPLE4/DUTY(WORK_DATE,EXMPLE_ID)#duty_table(duty_date,duty_id)',
'',
'D:\XML\XMLIMPORT.LOG');
```

Result:

```
dmSQL> SELECT * FROM DUTY_TABLE;
 DUTY_DATE    DUTY_ID DUTY_ONTIME DUTY_OFFTIME
========== ======= ========== ============
2008-08-05   B00119       NULL            NULL
2008-08-05   B00120       NULL            NULL
2008-08-05   B00121       NULL            NULL

3 rows selected
```

2. The command with incorrect argument return Error

```
CALL XMLIMPORT(
'D:\XML\XMLEXPORT.XML',
'/DBSAMPLE4/DUTY(work_date,exmple_id)#duty_table(duty_date,duty_id)',
'',
'D:\XML\XMLIMPORT.LOG');
ERROR (6107): [DBMaster] integrity constraint violation (cannot insert NULL value on NON-NULL column)
```

Because XML tags are case sensitive, so XML tags should be capital string which is the same as string in xml file.

3.3.5 WHEN <COLUMN TAG NAMES> HAS INCORRECT TAG NAME

The example is also to import xmlexport.xml which is exported in chapter 2.2.3.

If object_str is “/DBSAMPLE4/DUTY(DUTY_DATE,DUTY_ID,error_tag1,error_tag2)#duty_table(duty_date,duty_id,duty_ontime,duty_offtime)”,

Result is:

```
dmSQL> select * from duty_table;
 DUTY_DATE    DUTY_ID DUTY_ONTIME DUTY_OFFTIME
========== ======= ========== ============
2008-08-05   B00119       NULL            NULL
2008-08-05   B00120       NULL            NULL
2008-08-05   B00121       NULL            NULL

3 rows selected
```