# Передача больших объектов XML через параметры ODAC в процедуры PLSQL

В жизни каждого программиста иногда возникают не совсем стандартные задачи. Одна из таких задач – передача в функцию или процедуру PLSQL нескольких значений в виде одного текстового упакованного поля (с последующей их распаковкой и обработкой внутри PLSQL).

При этом каждый программист придумывает собственные «велосипеды» форматов упаковки и разбора этих данных.

Обычно для этого используется длинная строка, которая записывается в переменную типа VARCHAR2, передаётся внутрь процедуры PLSQL и там обрабатывается.

Оракул, начиная с версии 9, поддерживает обработку XML, что предоставляет программистам очень удобный механизм упаковки больших структур данных в этот формат.

Для этого, в оракуле существует специальный объектный тип XMLtype, который можно использовать как внутри процедур и функций PLSQL:

declare

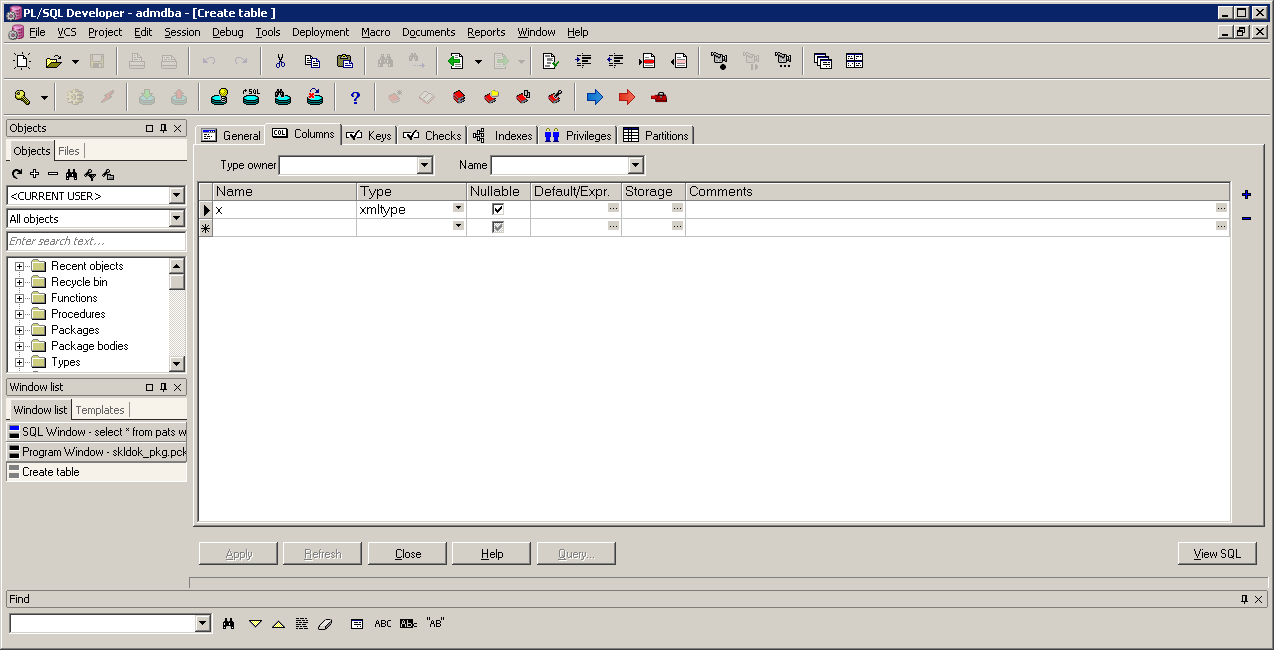
x XMLtype;

begin

null;

end;

так и непосредственно в таблицах для хранения таких данных:



При этом, «сырые» данные XML массива можно передать посредством привычного VARCHAR2 и очень просто преобразовать их в нужный тип

Procedure foo(s varchar2) is

X XMLtype;

Begin

X := XMLtype(s);

End;

Также можно передать в процедуру и сам объект типа XMLtype

Procedure foo(x XMLtype) is

Begin

…

End;

# Работа с объектом XMLtype

Для работы с типом XMLtype в оракле существует несколько функций

1. Преобразование VARCHAR2 в XMLtype делается при помощи конструктора **XMLtype**

Declare

X XMLtype;

Begin

X := XMLtype(‘<a><b>111</b><bb>222</bb></a>’);

// или используя статический метод createXML()

X := xmltype.createxml(‘<a><b>111</b><bb>222</bb></a>’);

End;

1. **ExtractValue** - Прочитать значение внутри тега XML. (Работает только на краевых узлах)

Declare

X XMLtype;

S varchar2(100);

Begin X := XMLtype(‘<a><b>111</b><bb>222</bb></a>’);

S := Extractvalue(x, ‘/a/b’); // вернёт ‘111’

S := Extractvalue(x, ‘/a/bb’); // вернёт ‘222’

S := Extractvalue(x, ‘/a/qwerty’); // вернёт NULL

1. Обращение к методам объекта XMLtype

**Xmltype.extract(‘/tag1/tag2’)** – возвращает объект XMLtype, который содержит в себе только «вырезку» указанного тега (tag2) вместе с самим этим тегом

**Xmltype.getstringval()** – возвратит текст XML-объекта (отличается от ExtractValue тем, что возвращает просто текст XML вместе с тегами)

Declare

X,xx XMLtype;

S varchar2(100);

Begin X := XMLtype(‘<a><b>111</b><bb>222</bb></a>’);

xx := X.extract(‘/a/b’); // вернёт объект XMLtype внутри которого будет ‘<b>111</b>’

// чтобы увидеть это, можно сделать так:

S := X.extract(‘/a/b’).GetStringVal(); // вернёт текст ‘<b>111</b>’

End;

1. **XMLsequence** - Преобразование XML в таблицу

select t.column\_value.getstringval() f1,

extractvalue(t.column\_value,'/b') f2

from table(XMLsequence(XMLtype('<a><b>1</b><b>2</b></a>').extract('/a/b'))) t

вернёт:

|  |  |
| --- | --- |
| **f1** | **f2** |
| <b>1</b> | 1 |
| <b>2</b> | 2 |

1. Получить значение атрибута

Если конструкция X.extract(‘/a/b’) возвращает полностью содержимое тега <b>, то конструкция вида **X.extract(‘/a/b/@myattr’)** возвращает значение атрибута myattr из тега <b>

Т.е.

Declare

X,xx XMLtype;

S varchar2(100);

Begin X := XMLtype(‘<a><b myattr=”QWERTY”>111</b><bb>222</bb></a>’);

xx := X.extract(‘/a/b’).GetStringVal(); // вернёт 111

xx := X.extract(‘/a/b/@myattr’); // вернёт QWERTY

End;

1. Перегнать таблицу БД в XMLtype

X XMLtype;

begin

Select XMLtype(cursor(select \* from asu\_zag\_dok where rownum<3)) into x from dual;

End;

Получится что-то такое:

<?xml version="1.0"?>

<ROWSET>

<ROW>

<D\_DOK>26.02.03</D\_DOK>

<WID\_DOK>46</WID\_DOK>

<TYPE>М</TYPE>

<OP>12</OP>

<POST>225,15</POST>

<POTR>39</POTR>

<CEH\_POST>225.15</CEH\_POST>

<CEH\_POTR>39</CEH\_POTR>

<D\_SNAB>26.02.03</D\_SNAB>

<D\_SKL\_POST>05.03.03</D\_SKL\_POST>

<U\_SKL\_POST>GALYA</U\_SKL\_POST>

<DTR>05.03.03</DTR>

<USR>GALYA</USR>

<NZ>28052</NZ>

</ROW>

<ROW>

<D\_DOK>26.02.03</D\_DOK>

<WID\_DOK>6</WID\_DOK>

<TYPE>М</TYPE>

<OP>2</OP>

<N\_SF>ТОВАРНЫЙ</N\_SF>

<POST>2818</POST>

<POTR>222,04</POTR>

<CEH\_POST>2818</CEH\_POST>

<CEH\_POTR>222.04</CEH\_POTR>

<D\_SKL\_POTR>26.02.03</D\_SKL\_POTR>

<U\_SKL\_POTR>MAKAROVA</U\_SKL\_POTR>

<DTR>05.05.03</DTR>

<USR>TEM</USR>

<NZ>28058</NZ>

<S\_NDS>0</S\_NDS>

<N\_GROUP>28057</N\_GROUP>

</ROW>

</ROWSET>

1. Создание XMLtype «на лету», используя XMLelement, AMLAttributes, insertchildXML, appendchildXML

with t as (SELECT XMLElement("REQUEST",

XMLAttributes('meta' AS "xmlns"),

XMLElement("HEADER",

XMLAttributes('TEST' AS "USERID",

'RRUID\_' AS "UNI",

'KF' AS "MODULE")),

XMLElement("COMMAND",

XMLElement("CALC\_GRAPH",

XMLELEMENT ( "DOG\_TYPE\_CODE", null, 'p\_DOG\_TYPE\_CODE'),

XMLELEMENT ( "DOG\_SUM", null, 'p\_DOG\_SUM'),

XMLELEMENT ( "DOG\_TERM", null, 'p\_DOG\_TERM')))

) x

FROM DUAL)

select insertchildxml(x, '/REQUEST/COMMAND/CALC\_GRAPH', 'xxx', xmlelement("xxx",'XXX Value'), 'xmlns=meta')

from t

union all

select appendchildxml(x, '/REQUEST/COMMAND/CALC\_GRAPH', xmlelement("xxx",'XXX Value'), 'xmlns=meta')

from t;

Результат:

<REQUEST xmlns="meta">

<HEADER USERID="TEST" UNI="RRUID\_" MODULE="KF"/>

<COMMAND>

<CALC\_GRAPH>

<DOG\_TYPE\_CODE>p\_DOG\_TYPE\_CODE</DOG\_TYPE\_CODE>

<DOG\_SUM>p\_DOG\_SUM</DOG\_SUM>

<DOG\_TERM>p\_DOG\_TERM</DOG\_TERM>

<xxx xmlns="">XXX Value</xxx>

</CALC\_GRAPH>

</COMMAND>

</REQUEST>

<REQUEST xmlns="meta">

<HEADER USERID="TEST" UNI="RRUID\_" MODULE="KF"/>

<COMMAND>

<CALC\_GRAPH>

<DOG\_TYPE\_CODE>p\_DOG\_TYPE\_CODE</DOG\_TYPE\_CODE>

<DOG\_SUM>p\_DOG\_SUM</DOG\_SUM>

<DOG\_TERM>p\_DOG\_TERM</DOG\_TERM>

<xxx>XXX Value</xxx>

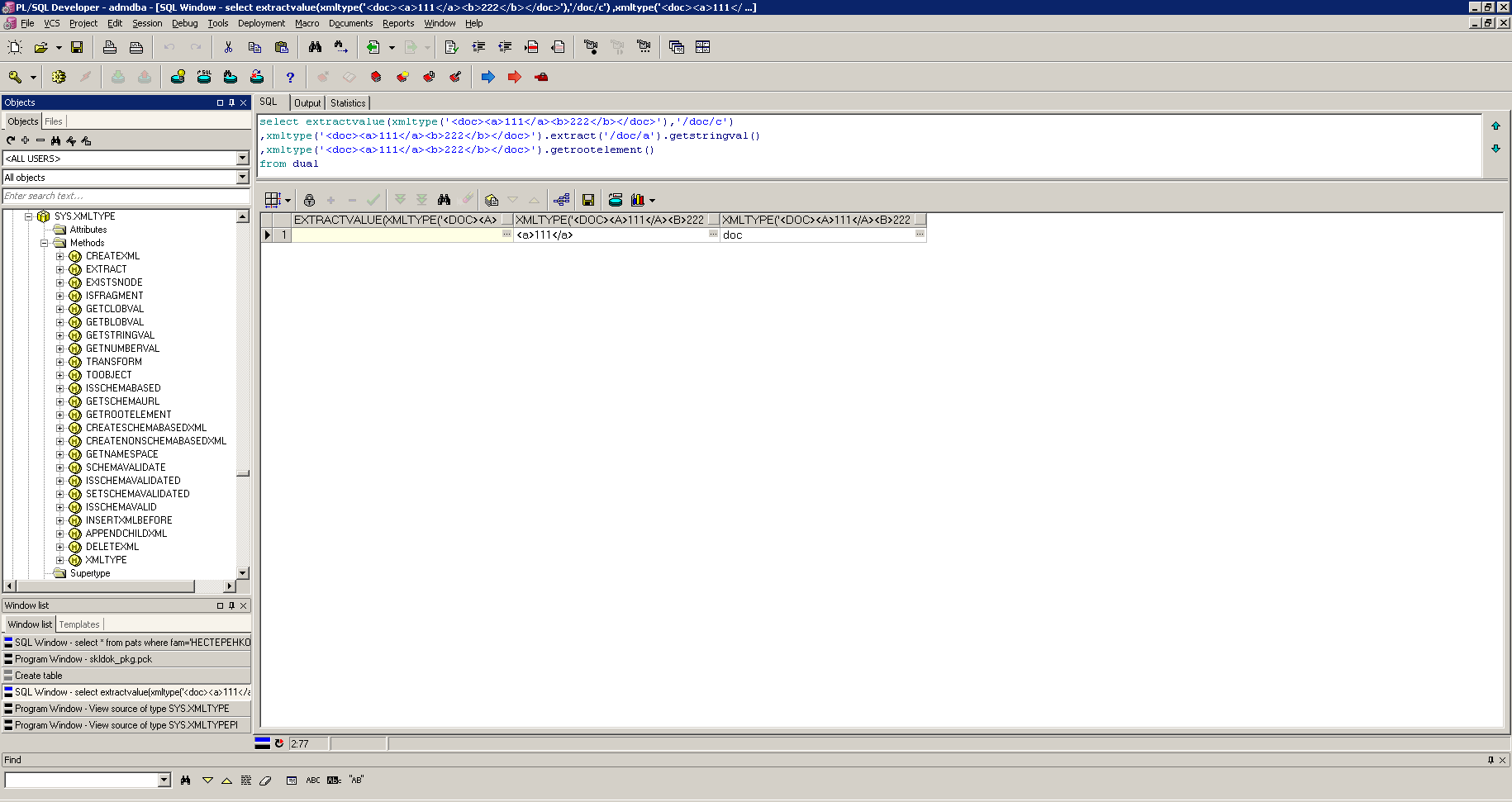
</CALC\_GRAPH>

</COMMAND>

</REQUEST>

Только не спрашивайте в чём разница insertchildXML и appendchildXML

1. А ещё, можно в девелопере найти тип XMLtype и посмотреть как устроено его описание



# Как это выглядит в Builder

**Формирование XML**

Если XML-формат предполагается использовать только для передачи В оракл, то можно обойтись без сторонних библиотек, т.к. XML-формат довольно прост «для писателя»

Для облегчения процесса «написательства» XML,в билдере я придумал для себя вот такой ~~велосипед~~ класс:

class tXML {

private:

AnsiString SectionName;

AnsiString\* str;

int begin\_pos;

public:

map<AnsiString,AnsiString> attr;

AnsiString val;

int cnt;

tXML(AnsiString in\_SectionName, AnsiString &in\_str) {

str = &in\_str;

begin\_pos = str->Length()+1;

SectionName = in\_SectionName;

val = "";

cnt = 1;

}

~tXML() {

\*str = \*str + normalize(val) + "</" + normalize(SectionName) + ">";

\*str = str->Insert("<"+normalize(SectionName)+str\_attr()+">",begin\_pos);

}

AnsiString ReplaceSubString(AnsiString Str, AnsiString SubStrFind, AnsiString SubStrToReplace) const

{

AnsiString out = "";

while (int X = Str.Pos( SubStrFind)) {

out = out + Str.SubString(1,X-1) + SubStrToReplace;

Str = Str.Delete( 1, X -1 + SubStrFind.Length());

}

out = out + Str;

return out;

}

AnsiString normalize(AnsiString str) {

str = ReplaceSubString(str,"&", "&amp;");

str = ReplaceSubString(str,"\"", "&quot;");

str = ReplaceSubString(str,"<", "&lt;");

str = ReplaceSubString(str,">", "&gt;");

str = ReplaceSubString(str,"'", "&apos;");

return str;

}

AnsiString str\_attr() {

AnsiString out = "";

for (map<AnsiString,AnsiString>::iterator it=attr.begin(); it!=attr.end(); it++) {

out = out + " " + normalize(it->first) + "=\"" + normalize(it->second) + "\"";

}

return out;

}

};

Для большего удобства работы с этим классом, можно нарисовать вот такой макрос:

#define XML(x) if (false); else for (tXML XML(x,xmlString); XML.cnt>0; XML.cnt--)

Этими инструментами структура XML-документа наглядно отображается через структуру программы на Си

AnsiString xmlString = "";

XML(“doc”) {

XML(“zag”) {

XML.attr[“CAPTION”] = “Заголовок”;

XML(“id”) XML.val=”10”;

XML(“prim”) XML.val=”Примечание”;

}

XML(“sod”) {

XML.attr[“CAPTION”] = “Содержимое”;

XML(“row”) {

XML(“kod”) XML.val=”Код1”;

XML(“kol\_treb”) XML.val=”12”;

}

XML(“row”) {

XML(“kod”) XML.val=”Код2”;

XML(“kol\_treb”) XML.val=”23”;

}

// либо можно в цикле сформировать сразу 10 тегов /row со всем содержимым

for (int I = 0; I < 10; I++) {

XML(“row”) {

XML(“kod”) XML.val=”Код2”;

XML(“kol\_treb”) XML.val=”23”;

}

}

}

}

// Теперь переменная xmlString содержит в себе текст объекта XML

В результате получится вот такой XML-документ (форматирование отступами добавлено для наглядности):

<doc>

<zag CAPTION=”Заголовок”>

<id>10</id>

<prim>Примечание</prim>

</zag>

<sod CAPTION=”Содержимое”>

<row>

<kod>Код1</kod>

<kol\_treb>12</kol\_treb>

</row>

<row>

<kod>Код2</kod>

<kol\_treb>23</kol\_treb>

</row>

</sod>

</doc>

**Передача данных из Builder в Oracle**

Если размер XML данных не слишком большой, то в принципе всё выглядит стандартно. Например так:

std::auto\_ptr<TOraQuery> Q(new TOraQuery(0));

Q->SQL->Text = "begin foo(XMLtype(:1)); end;";

Q->Prepare();

AnsiString S = “<doc><a>111</a><b>222</b></doc>”;

Q->ParamByName("1")->AsString = S;

Q->ExecSQL();

Но есть нюансы.

Через параметр ->AsString не получится передать более 4000 символов за раз. Это ограничение ODAC. Лишние символы просто обрежутся и узнаете вы об этом только в тот момент, когда попытаетесь присвоить этот обрезанный varchar2 объекту XMLtype, который упадёт с ошибкой парсинга данных.

Ситуацию можно немного поправить, если использовать ->AsMemo вместо ->AsString.

Такая простейшая модификация отодвигает максимальную границу до 32к. (точнее 32515). Это ограничение уже не связано с ODAC, а является фундаментальным ограничением типа VARCHAR2.

Q->ParamByName("1")->**AsMemo** = S;

Q->ExecSQL();

Для большинства задач это может быть достаточно. Однако в некоторых случаях может потребоваться ещё больше ~~(«нам нужно больше золота, Милорд»)~~ (не забывайте, что формат XML помимо полезных данных хранит в себе теги и различные спецпоследовательности для возможности хранения спецсимволов)

Поэтому, иногда, при большом количестве тегов, или разветвлённой структуре может потребоваться больше 32к символов за раз. Это тоже можно победить, но требует применения немного дополнительной магии.

Для этой цели можно использовать параметр типа TOraClob.

Но тут есть нюансы. Просто написать:

Q->ParamByName("1")->**AsOraClob->AsString** = S;

Q->ExecSQL();

Недостаточно. Сразу «из коробки» работать не будет. Необходимо перед этим добавить следующие заклинания:

Q->ParamByName("1")->**Options->TemporaryLobUpdate = true;**

Q->ParamByName("1")->**ParamType = ptInput;**

Q->ParamByName("1")->AsOraClob->AsString = S;

Q->ExecSQL();

Теперь можно передавать за раз очень много. Ну очень много. 4ГБ должно хватить всем…

Вот и всё. Детишки ☺