IVOA Astronomical Data Query Language Version 0.7.4

IVOA Working Draft 2004-05-17

This version:

0.7.4 http://www.ivoa.net/internal/IVOA/IvoaVOQL/ADQL-0.7.4.pdf

Previous versions:

0.7.1 http://www.ivoa.net/internal/IVOA/IvoaVOQL/ADQL-0.7.1.pdf

0.7 http://skyservice.pha.jhu.edu/develop/vo/adql/ADQL-0.7.pdf

0.6 http://skyservice.pha.jhu.edu/develop/vo/adql/ADQL-0.6.pdf

0.5 http://skyservice.pha.jhu.edu/develop/vo/adql/SkyNodeInterface-0.5.pdf

0.4 http://skyservice.pha.jhu.edu/develop/vo/adql/SkyNodeInterface--0.4.pdf

0.3 http://skyservice.pha.jhu.edu/develop/vo/adql/QueryInterface-2003Aug.pdf

0.2 http://skyservice.pha.jhu.edu/develop/vo/adql/QueryInterface-2003July.pdf

Editors:

Masatoshi Ohishi, Alex Szalay

Authors:

IVOA VOQL Working group

Please send comments to: mailto:voql@ivoa.net

Abstract

This document describes the Astronomical Data Query Language(ADQL) and ADQL/s its string representation,

Status of this document

This is a Working Draft. There are no prior released versions of this document.

This is an IVOA Working Draft for review by IVOA members and other interested parties. It is a draft document and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use IVOA Working Drafts as reference materials or to cite them as other than "work in progress." A list of current IVOA Recommendations and other technical documents can be found at http://www.ivoa.net/docs/.

Acknowledgments

This work is based on discussions at various IVOA meetings and continuing emails on the mailing list.

Contents

1 Introduction

ADQL is an XML language for constructing queries. This is based on Structured Query Language (SQL). We have many tabular data sets in the VO and many are in relational databases, making SQL an interesting first step. This document is a formal agreement of what is contained in ADQL.

The mechanics of passing a query to a node is described in the SkyNode Interface document [6] that also is developed in the VOQL WG of the IVOA. It should be noted that the SkyNode Interface is also related to Data Access Layer WG of the IVOA. To see some current implementations of SkyNodes and the OpenSkyQuery portal go to OpenSkyQuery.net.

2 Astronomical Data Query Language (ADQL)

ADQL is passed as an XML document to the Query Interface. ADQL is based on a subset of SQL plus region with, as a minimum support, for circle(Cone Search). The only SQL command allowed in this version of ADQL is a “select”. The full XSD for ADQL0.7.4 may be found below in Section 7 “ADQL XSD“

Services for translation of SQL to ADQL and back may be found at http://skydev.pha.jhu.edu/develop/vo/adql/

Since ADQL is similar in semantics to SQL the requirements below list differences or special considerations only.

ADQL has two forms:

- ADQL/x : An XML document conforming to the XSD in Section 7.
- ADQL/s : A String form based on SQL92 [1] and conforming to the ADQL grammar in Section 6. Some non standard extensions are added to support distributed astronomical queries.

ADQL/x and ADQL/s are translatable between each other without loss of information.

It is felt the string representation is necessary to ensure the SQL like semantic and structural definition of ADQL within the XML document because many end users of ADQL will ultimately wish to convert to some form of SQL.

2.1 Restrictions on SQL92

The formal notation for syntax of computing languges are often expressed in the “Backus Naur Form” BNF[1]. BNF is used by popular tools such as LEX and YACC[2] for producing parsers for a given syntax. Section 6 provides the YACC type grammar for ADQL/s.

[1] http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html#Johnson75

The BNF exactly defines the form of SQL92 which is ADQL/s. In essence this is any valid Select statement except those containing subqueries .

2.2 Extensions to SQL92

This specification adds requirements on top of SQL92. These are described below.

These extensions to SQL are given with examples in ADQL/s but of course ADQL/x can express any string from ADQL/s.

2.2.1 Aliases

All table names in ADQL must have an alias. Aliasing tables is a part of standard SQL but we are enforcing this in ADQL/s.

This means queries in ADQL/s must take the form

```
Select * from table t
```

This makes substitution of table names much easier as it must be done in only one place to change the alias.

2.2.2 Regions

ADQL supports the region specification as defined by the region.xsd [3] of the IVOA/NVO. For this and RegionXML specified below we shall create some default coordinate systems and units to simplify the regions initially.

2.2.3 Mathematical Funtions

JDBC [4] Mathematical functions shall be allowed in ADQL as follows:

Trigonometric functions: acos, asin, atan, atan2, cos, cot, sin, tan

Math functions: abs, ceiling, degrees, exp, floor, log, log10, mod, pi, power, radians, sqrt, rand, round, truncate.

ADQL/x documents shall contain a version identifier for the version of ADQL. This will start as 1.0. The version number is a dot separated string of numbers. The version number is included in the document solely so the receiving node may decide if it wishes to deal with the document or thrown an exception. This is assumed to only come into use at some later stage when there may be a major version change causing some possible incompatibility between versions. We should strive for backward compatibility i.e. only adding new features not deprecating the old.

Sample applications and tutorials for development and deployment of ADQL services is available at http://skyservice.pha.jhu.edu/develop/vo/adql/

[2] http://epaperpress.com/lexandyacc/

2.3 Regions

ADQL/s shall support the Region keyword. This will be followed by a single quoted string specifying a region in a simple manner similar to the current SDSS cover specification in [5]. This would look something like:

```
Region (‘CIRCLE J2000 19.5 –36.7 0.02’)
```

This is a one way operation. If a SkyQuery string is converted to ADQL this Region string will be converted to XML. If the resulting XML is converted back to a String the region should remain as inlined XML using the RegionXML keyword.
There may be a comment section added to the region xsd. In this comment section the original string should be kept. The comment section will be used for display purposes in certain areas and should contain a summary description (in English) of the region.
Other constructs mentioned in [5] are RECT, POLY, and CHULL are also supported.

As implied above it is possible to inline a region specification as in ADQL/s using the RegionXML keyword e.g. (not a valid region spec)

```
RegionXML (‘<circle><coordsys>ICRS</coordsys><ra>19.5</ra><dec>-
36.7</dec><radius>0.02</readius></circle>’)
```

It is also possible to refer to a region specification as a url in ADQL/s using the RegionURL keyword e.g.

```
RegionURL (‘http://aserver.edu/aregion.xml’)
```

3 ADQL example

An SQL like string for ADQL might be as follows :

```
Select a.* from Tab a where Region('Circle Cartesian 1.2 2.4 3.6
0.2')
```

This would be represented in xml as follows

```
<?xml version="1.0" encoding="utf-16"?>

<Select xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.ivoa.net/xml/ADQL/v0.7.4">

  <SelectionList>

    <Item xsi:type="columnReferenceType" Table="a" Name="*" />

  </SelectionList>

  <From>

    <Table xsi:type="tableType" Name="Tab" Alias="a" />

  </From>

  <Where>

    <Condition xsi:type="regionSearchType">

      <Region xmlns:q1="urn:nvo-region" xsi:type="q1:circleType"
coord_system_id="">

        <q1:Center ID="" coord_system_id="">
```

```
              <Pos3Vector xmlns="urn:nvo-coords">
                <Name>X Y Z</Name>
                <CoordValue>
                  <Value>
                    <double>1.2</double>
                    <double>2.4</double>
                    <double>3.6</double>
                  </Value>
                </CoordValue>
              </Pos3Vector>
            </q1:Center>
            <q1:Radius>0.2</q1:Radius>
          </Region>
        </Condition>
      </Where>
    </Select>
```

4 Changes from previous versions

- Put in BNF and XSD
- Made more of a spec by changing language – no more REQUIREMENTS.

5 References

[1] http://www.contrib.andrew.cmu.edu/%7Eshadow/sql/sql1992.txt

[2] IVOA VOQL Working group; IVOA SkyNode Interface – get latest from www.ivoa.net/voql

[3] Space Time Coordinates for VO; Arnold Rots, May 2003; http://www.ivoa.net/internal/IVOA/InterOpMay2003DataModel/STCdoc.pdf and http://hea-www.harvard.edu/~arots/nvometa/STC_UML.pdf

[4] Java Database Connectivity Specification 3.0 ; download from http://java.sun.com/products/jdbc/index.jsp

[5] SQLServer2000 HTM Interface specification; Alex Szalay, George Fekete, Jim Gray; July 2003 ; http://skyservice.pha.jhu.edu/develop/vo/adql/htmdll_2_0.doc

[6] SkyNode Interface. http://www.ivoa.net/internal/IVOA/IvoaVOQL/SkyNodeInterface-0.7.4.pdf

6 ADQL Grammer

Below is the grammer used to produce the parser in C#.

```
//The SQL(ADQL) Parser Grammar file
```

```
//Author: Vivek Haridas
//Dept of Physics & Astronomy
//Johns Hopkins University
// $Revision: 1.7 $

//Macro section - optional
%macro
{D}                 [0-9];
{E}                 [Ee][\-+]?{D}+;



//Expression section - required
%expression Main
''''                                              %ignore, %push
SQuote;
'[ \n\t\r]+'                                      %ignore;

'[A-Za-z][A-Za-z0-9_]*'
    NAME;
'[\-+]?{D}+'
          INTNUM;
'[\-+]?({D}+"."{D}*({E})?)|({D}+{E})'
    APPROXNUM;



'[aA][lL][lL]'                                    ALL;
'[aA][nN][dD]'                                    AND;
'[aA][vV][gG]'                                    AMMSC;
'[mM][iI][nN]'                                    AMMSC;
'[mM][aA][xX]'                                    AMMSC;
'[sS][uU][mM]'                                    AMMSC;
'[cC][oO][uU][nN][tT]'                            AMMSC;
'[aA][nN][yY]'                                    ANY;
'[aA][sS]'                                              AS;
'[aA][sS][cC]'                                    ASC;
'[aA][uU][tT][hH][oO][rR][iI][zZ][aA][tT][iI][oO][nN]'
    AUTHORIZATION;
'[bB][eE][tT][wW][eE][eE][nN]'                               BETWEEN;
'[bB][yY]'
    BY;
'[cC][hH][aA][rR]([aA][cC][tT][eE][rR])?'          CHARACTER;
'[cC][hH][eE][cC][kK]'
    CHECK;
```

```
'[dD][eE][sS][cC]'                              DESC;
'[dD][iI][sS][tT][iI][nN][cC][tT]'              DISTINCT;
'[dD][oO][uU][bB][lL][eE]'                              DOUBLE;
'[eE][xX][iI][sS][tT][sS]'                      EXISTS;
'[fF][lL][oO][aA][tT]'                          FLOAT;
'[fF][oO][rR]'                                  FOR;
'[fF][rR][oO][mM]'                              FROM;
'[gG][rR][oO][uU][pP]'                          GROUP;
'[hH][aA][vV][iI][nN][gG]'                      HAVING;
'[iI][nN]'                                      IN;
'[iI][nN][tT]([eE][gG][eE][rR])?'              INTEGER;
'[iI][nN][tT][oO]'                              INTO;
'[iI][sS]'                                      IS;
'[kK][eE][yY]'                                                KEY;
'[lL][iI][kK][eE]'                              LIKE;
'[nN][oO][tT]'                                  NOT;
'[nN][uU][lL][lL]'                              NULL;
'[nN][uU][mM][eE][rR][iI][cC]'                  NUMERIC;
'[oO][fF]'                                      OF;
'[oO][nN]'                                      ON;
'[oO][rR]'                                      OR;
'[oO][rR][dD][eE][rR]'                          ORDER;
'[sS][eE][lL][eE][cC][tT]'                      SELECT;
'[sS][eE][tT]'                                  SET;
'[sS][mM][aA][lL][lL][iI][nN][tT]'              SMALLINT;
'[sS][oO][mM][eE]'                              SOME;
'[Tt][aA][bB][lL][eE]'                          TABLE;
'[Tt][oO]'                                      TO;

'[sS][iI][nN]'
   TRIG;
'[cC][oO][sS]'
   TRIG;
'[tT][aA][nN]'
   TRIG;
'[cC][oO][tT]'
   TRIG;
'[aA][sS][iI][nN]'
   TRIG;
'[aA][cC][oO][sS]'
   TRIG;
'[aA][tT][aA][nN]'
   TRIG;
```

```
'[aA][tT][aA][nN][2]'
   TRIG;


'[aA][bB][sS]'
   MATHF;
'[cC][eE][iI][lL][iI][nN][gG]'                              MATHF;
'[dD][eE][gG][rR][eE][eE][sS]'                              MATHF;
'[eE][xX][pP]'
   MATHF;
'[fF][lL][oO][oO][rR]'
   MATHF;
'[lL][oO][gG]'
   MATHF;
'[pP][iI]'
   MATHF;
'[pP][oO][wW][eE][rR]'
   MATHF;
'[rR][aA][dD][iI][aA][nN][sS]'                              MATHF;
'[sS][qQ][rR][tT]'
   MATHF;
'[sS][qQ][uU][aA][rR][eE]'                                  MATHF;
'[lL][oO][gG][1][0]'
   MATHF;
'[rR][aA][nN][dD]'
   MATHF;
'[rR][oO][uU][nN][dD]'
        MATHF;
'[tT][rR][uU][nN][cC][aA][tT][eE]'                          MATHF;


'[uU][nN][iI][oO][nN]'                             UNION;
'[uU][nN][iI][qQ][uU][eE]'                         UNIQUE;
'[uU][sS][eE][rR]'                                      USER;
'[wW][hH][eE][rR][eE]'                             WHERE;
'[wW][iI][tT][hH]'                                 WITH;



'[xX][mM][aA][tT][cC][hH]'                                  XMATCH;
'[rR][eE][gG][iI][oO][nN]'                                  REGION;


'[tT][oO][pP]'
   TOP;


'='
ASSIGN;
".*"                                               ALLFIELDS, '.*';
```

```
"-"                                              UMINUS;
"-"                                              MINUS, '-';
"+"                                              PLUS, '+';
"*"                                              MULT, '*';
'/'                                              DIV, '/';
"("                                              OPAREN, '(';
")"                                              CPAREN, ')';
":"                                              COLON, ':' ;
';'                                              SEMICOLON, ';';
"."                                              DOT, '.' ;
','                                              COMMA, ',' ;
"!"                                              NOT, '!';

'<='
   LESSOREQUAL, '<=';
'>='
   GREATOREQUAL, '>=';
'<>'
   NOTEQUAL, '<>';
'>'
   GREATER, '>';
'<'
   LESS, '<';
'='
   EQUAL, '=';



%expression SQuote
'[^''\\\n]+'
     STRINGPART;
'\\\n'
     %ignore;
'\\'
           STRINGPART;
''''
           STRINGEND, %pop;



//Keyword section - optional
//%keyword

//Precedence section - optional
```

```
%prec
1, ';',    %left;
2, WHERE   %left;
3, OR,  %left;
4, AND, %left;
5, ','     %left;
6, ASSIGN, %right;
7, COLON, %right;
14, NOT,  %left;
14, '<',   %left;
14, '>',   %left;
14, '<=',  %left;
14, '>=',  %left;
14, '=',   %left;
16, '+',   %left;
16, '-',   %left;
17, '*',   %left;
17, '/',   %left;
19, UMINUS,%right;
22, '.',   %left;

//Production section - usually required

%production sql
P2 sql -> query_exp ;

P3 query_exp -> query_term;

P4 query_term -> query_spec;

P5 query_spec -> SELECT opt_all_distinct opt_top selection
table_exp opt_order_exp;

P6 opt_all_distinct -> ;
P7 opt_all_distinct -> ALL;
P8 opt_all_distinct -> DISTINCT;


P9  selection -> selection_items_list;
```

```
P10 selection_items_list -> selection_item;
P11 selection_items_list -> selection_items_list COMMA
selection_item;
P11a selection_item -> scalar_exp AS column_alias;
P11b selection_item -> scalar_exp;
P11c selection_item -> *;
p11d column_alias -> NAME;

P12 table_exp -> from_clause  opt_where_clause  opt_group_by_clause
opt_having_clause;

P13 from_clause -> FROM table_ref_commalist;

P14 table_ref_commalist ->  table_ref;
P15 table_ref_commalist ->  table_ref_commalist COMMA table_ref;
P16 table_ref -> table ;

P17 opt_where_clause -> ;
P18 opt_where_clause -> where_clause;

P19 where_clause -> WHERE search_condition ;

P20 opt_group_by_clause ->;
P21 opt_group_by_clause -> GROUP BY column_ref_commalist;

P22 column_ref_commalist -> column_ref;
P23 column_ref_commalist -> column_ref_commalist COMMA column_ref;

P24 opt_having_clause -> ;
P25 opt_having_clause -> HAVING search_condition;

P26 xmatch_condition -> xmatch_predicate;
P27 region_condition ->  region_predicate;

P28 search_condition -> search_condition OR search_condition;
P29 search_condition -> search_condition AND search_condition;
P30 search_condition -> NOT search_condition;
P31 search_condition -> OPAREN search_condition CPAREN;
P32 search_condition -> xmatch_condition;
```

```
P33 search_condition -> region_condition;
P34 search_condition -> predicate;

P35 predicate -> comparison_predicate ;
P36 predicate -> between_predicate ;
P37 predicate -> like_predicate ;

P38 comparison_predicate -> scalar_exp comparison scalar_exp;
P39 between_predicate -> scalar_exp NOT BETWEEN scalar_exp AND
scalar_exp;
P40 between_predicate -> scalar_exp BETWEEN scalar_exp AND
scalar_exp;

P41 like_predicate -> scalar_exp NOT LIKE atom;
P42 like_predicate -> scalar_exp LIKE atom;

P43 xmatch_predicate -> XMATCH OPAREN alias_commalist CPAREN
comparison  number;

P44 region_predicate -> REGION OPAREN string CPAREN;



P45 scalar_exp -> scalar_exp PLUS scalar_exp ;
P46 scalar_exp -> scalar_exp MINUS scalar_exp ;
P47 scalar_exp -> scalar_exp MULT scalar_exp ;
P48 scalar_exp -> scalar_exp DIV scalar_exp ;
P49 scalar_exp -> PLUS scalar_exp ;
P50 scalar_exp -> MINUS scalar_exp ;
P51 scalar_exp -> atom;
P52 scalar_exp -> column_ref;
P53 scalar_exp -> function_ref;
P54 scalar_exp -> OPAREN scalar_exp CPAREN;

P55 atom -> literal;

P56 function_ref -> AMMSC OPAREN MULT CPAREN;
P57 function_ref -> AMMSC OPAREN DISTINCT column_ref CPAREN;
P58 function_ref -> AMMSC OPAREN ALL scalar_exp CPAREN;
P59 function_ref -> AMMSC OPAREN scalar_exp CPAREN;
P59b function_ref -> TRIG OPAREN scalar_exp CPAREN;
P59c function_ref -> MATHF OPAREN scalar_exp CPAREN;
P60 alias_commalist -> alias;
```

```
P61 alias_commalist -> alias_commalist ',' alias;

P62 alias ->  NAME;
P63 alias ->  NOT NAME;

P64 literal -> string ;


P66a literal -> number ;

P67 column_ref -> NAME DOT NAME;
P68 column_ref -> NAME ALLFIELDS;

P69 number -> INTNUM ;
P70 number -> APPROXNUM ;

P71 table ->  NAME ':' NAME NAME;
P71a table -> NAME NAME;

P72 comparison -> '=';
P73 comparison -> '<>';
P74 comparison -> '>=';
P75 comparison -> '<=';
P76 comparison -> '>';
P77 comparison -> '<';

P78 string -> stringList STRINGEND;
P79 stringList  -> stringList STRINGPART;
p80 stringList  -> STRINGPART;

p81 opt_order_exp -> ;
p82 opt_order_exp -> ORDER BY order_comma_list;

p83 order_comma_list -> order_term;
p84 order_comma_list -> order_comma_list ',' order_term;

p85 order_term -> scalar_exp opt_order_direction;

p86 opt_order_direction -> ;
p87 opt_order_direction -> DESC;
```

```
p88 opt_order_direction -> ASC;

P89 opt_top -> ;
P90 opt_top -> TOP INTNUM;
```

7 ADQL XSD

```
<?xml version="1.0" encoding="utf-8" ?>

- <xs:schema
    targetNamespace="http://www.ivoa.net/xml/ADQL/v0.7.4"
    xmlns:reg="urn:nvo region"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://www.ivoa.net/xml/ADQL/v0.7.4"
    elementFormDefault="qualified">
    <xs:import namespace="urn:nvo region"
      schemaLocation="http://hea
      www.harvard.edu/~arots/nvometa/region.xsd" />
  - <xs:complexType name="selectionItemType" abstract="true">
    - <xs:annotation>
        <xs:documentation>The base type for any of items to be
          selected in a query</xs:documentation>
      </xs:annotation>
    </xs:complexType>
  - <xs:complexType name="scalarExpressionType" abstract="true"
      mixed="false">
    - <xs:annotation>
        <xs:documentation>The base type for a scalar
          expression</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
        <xs:extension base="tns:selectionItemType" />
      </xs:complexContent>
    </xs:complexType>
  - <xs:complexType name="closedExprType" mixed="false">
    - <xs:annotation>
        <xs:documentation>Represents an expression inside a
          bracket</xs:documentation>
      </xs:annotation>
  - <xs:complexContent mixed="false">
    - <xs:extension base="tns:scalarExpressionType">
      - <xs:sequence>
          <xs:element name="Arg"
            type="tns:scalarExpressionType" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="binaryExprType" mixed="false">
  - <xs:annotation>
      <xs:documentation>Represents a binary expression such
        as a+b</xs:documentation>
    </xs:annotation>
  - <xs:complexContent mixed="false">
    - <xs:extension base="tns:scalarExpressionType">
      - <xs:sequence>
          <xs:element name="Arg"
            type="tns:scalarExpressionType"
            minOccurs="2" maxOccurs="2" />
        </xs:sequence>
        <xs:attribute name="Oper"
          type="tns:binaryOperatorType" use="required" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
- <xs:simpleType name="binaryOperatorType">
  - <xs:annotation>
      <xs:documentation>Used for expressing operations like
        A+B</xs:documentation>
    </xs:annotation>
  - <xs:restriction base="xs:string">
      <xs:enumeration value="+" />
      <xs:enumeration value=" " />
      <xs:enumeration value="*" />
      <xs:enumeration value="/" />
    </xs:restriction>
  </xs:simpleType>
- <xs:complexType name="unaryExprType" mixed="false">
  - <xs:annotation>
      <xs:documentation>Represents an unary expression such
        as (a.ra)</xs:documentation>
    </xs:annotation>
  - <xs:complexContent mixed="false">
    - <xs:extension base="tns:scalarExpressionType">
      - <xs:sequence>
          <xs:element name="Arg"
            type="tns:scalarExpressionType" />
        </xs:sequence>
        <xs:attribute name="Oper"
          type="tns:unaryOperatorType" use="required" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
- <xs:simpleType name="unaryOperatorType">
  - <xs:annotation>
      <xs:documentation>Operators for expressing a single
        element operation</xs:documentation>
    </xs:annotation>
  - <xs:restriction base="xs:string">
      <xs:enumeration value="+" />
      <xs:enumeration value=" " />
    </xs:restriction>
  </xs:simpleType>
- <xs:complexType name="columnReferenceType" mixed="false">
  - <xs:annotation>
      <xs:documentation>Represents a
        column</xs:documentation>
    </xs:annotation>
  - <xs:complexContent mixed="false">
    - <xs:extension base="tns:scalarExpressionType">
        <xs:attribute name="Table" type="xs:string"
          use="required" />
        <xs:attribute name="Name" type="xs:string"
           use="required" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="atomType" mixed="false">
  - <xs:annotation>
      <xs:documentation>Encapsulates basic literals such as
         Strings, Integers and Real
         numbers</xs:documentation>
    </xs:annotation>
  - <xs:complexContent mixed="false">
    - <xs:extension base="tns:scalarExpressionType">
      - <xs:sequence>
          <xs:element name="Literal"
             type="tns:literalType" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="literalType" abstract="true">
  - <xs:annotation>
      <xs:documentation>The base type for all
         literals</xs:documentation>
    </xs:annotation>
  </xs:complexType>
- <xs:complexType name="numberType" abstract="true"
     mixed="false">
  - <xs:annotation>
      <xs:documentation>The base type for all
         numbers</xs:documentation>
    </xs:annotation>
  - <xs:complexContent mixed="false">
      <xs:extension base="tns:literalType" />
    </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="realType" mixed="false">
```

```
  - <xs:annotation>
      <xs:documentation>Represents a real
         number</xs:documentation>
    </xs:annotation>
  - <xs:complexContent mixed="false">
    - <xs:extension base="tns:numberType">
        <xs:attribute name="Value" type="xs:double"
           use="required" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="integerType" mixed="false">
  - <xs:annotation>
      <xs:documentation>Represents an
         integer</xs:documentation>
    </xs:annotation>
  - <xs:complexContent mixed="false">
    - <xs:extension base="tns:numberType">
        <xs:attribute name="Value" type="xs:long"
           use="required" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="stringType" mixed="false">
  - <xs:annotation>
      <xs:documentation>Represents a string
         literal</xs:documentation>
    </xs:annotation>
  - <xs:complexContent mixed="false">
    - <xs:extension base="tns:literalType">
        <xs:attribute name="Value" type="xs:string"
           use="required" />
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="functionType" abstract="true"
     mixed="false">
    - <xs:annotation>
        <xs:documentation>The base type for a
          function</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
      - <xs:extension base="tns:scalarExpressionType">
        - <xs:sequence>
            <xs:element name="Allow"
              type="tns:selectionOptionType" minOccurs="0"
              />
            <xs:element name="Arg"
              type="tns:selectionItemType" />
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  - <xs:complexType name="selectionOptionType">
    - <xs:annotation>
        <xs:documentation>Option of selecting all or distinct
          elements in a query</xs:documentation>
      </xs:annotation>
      <xs:attribute name="Option" type="tns:allOrDistinctType"
        use="required" />
    </xs:complexType>
  - <xs:simpleType name="allOrDistinctType">
    - <xs:annotation>
        <xs:documentation>Enumeration for All and Distinct
          options</xs:documentation>
      </xs:annotation>
    - <xs:restriction base="xs:string">
        <xs:enumeration value="All" />
        <xs:enumeration value="DISTINCT" />
      </xs:restriction>
    </xs:simpleType>
  - <xs:complexType name="trigonometricFunctionType"
      mixed="false">
    - <xs:annotation>
```

```
        <xs:documentation>Represents a trigonometric
          function</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
      - <xs:extension base="tns:functionType">
          <xs:attribute name="Name"
            type="tns:trigonometricFunctionNameType"
            use="required" />
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  - <xs:simpleType name="trigonometricFunctionNameType">
    - <xs:annotation>
        <xs:documentation>Enumeration of allowed
          trigonometric functions</xs:documentation>
      </xs:annotation>
    - <xs:restriction base="xs:string">
        <xs:enumeration value="SIN" />
        <xs:enumeration value="COS" />
        <xs:enumeration value="TAN" />
        <xs:enumeration value="COT" />
        <xs:enumeration value="ASIN" />
        <xs:enumeration value="ACOS" />
        <xs:enumeration value="ATAN" />
        <xs:enumeration value="ATAN2" />
      </xs:restriction>
    </xs:simpleType>
  - <xs:complexType name="mathFunctionType" mixed="false">
    - <xs:annotation>
        <xs:documentation>Represents a math
          function</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
      - <xs:extension base="tns:functionType">
          <xs:attribute name="Name"
            type="tns:mathFunctionNameType"
            use="required" />
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  - <xs:simpleType name="mathFunctionNameType">
    - <xs:annotation>
        <xs:documentation>Enumeration of allowed math
          functions</xs:documentation>
      </xs:annotation>
    - <xs:restriction base="xs:string">
        <xs:enumeration value="ABS" />
        <xs:enumeration value="CEILING" />
        <xs:enumeration value="DEGREES" />
        <xs:enumeration value="EXP" />
        <xs:enumeration value="FLOOR" />
        <xs:enumeration value="LOG" />
        <xs:enumeration value="PI" />
        <xs:enumeration value="POWER" />
        <xs:enumeration value="RADIANS" />
        <xs:enumeration value="SQRT" />
        <xs:enumeration value="SQUARE" />
        <xs:enumeration value="LOG10" />
        <xs:enumeration value="RAND" />
        <xs:enumeration value="ROUND" />
        <xs:enumeration value="TRUNCATE" />
      </xs:restriction>
    </xs:simpleType>
  - <xs:complexType name="aggregateFunctionType" mixed="false">
    - <xs:annotation>
        <xs:documentation>Represents an aggregate
          function</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
        - <xs:extension base="tns:functionType">
          <xs:attribute name="Name"
            type="tns:aggregateFunctionNameType"
            use="required" />
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  - <xs:simpleType name="aggregateFunctionNameType">
    - <xs:annotation>
        <xs:documentation>Enumeration of allowed aggregate
          functions</xs:documentation>
      </xs:annotation>
    - <xs:restriction base="xs:string">
        <xs:enumeration value="AVG" />
        <xs:enumeration value="MIN" />
        <xs:enumeration value="MAX" />
        <xs:enumeration value="SUM" />
        <xs:enumeration value="COUNT" />
      </xs:restriction>
    </xs:simpleType>
  - <xs:complexType name="aliasSelectionItemType" mixed="false">
    - <xs:annotation>
        <xs:documentation>Used to select an expression as a
          new alias column</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
        - <xs:extension base="tns:selectionItemType">
          - <xs:sequence>
              <xs:element name="Expression"
                type="tns:scalarExpressionType" />
          </xs:sequence>
          <xs:attribute name="As" type="xs:string"
            use="required" />
        </xs:extension>
        </xs:complexContent>
    </xs:complexType>
  - <xs:complexType name="allSelectionItemType" mixed="false">
    - <xs:annotation>
        <xs:documentation>Represent all columns as in Select *
          query</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
        <xs:extension base="tns:selectionItemType" />
      </xs:complexContent>
    </xs:complexType>
  - <xs:simpleType name="comparisonType">
    - <xs:annotation>
        <xs:documentation>The Comparison operators such as
          Less than or More than, etc</xs:documentation>
      </xs:annotation>
    - <xs:restriction base="xs:string">
        <xs:enumeration value="=" />
        <xs:enumeration value="<>" />
        <xs:enumeration value=">" />
        <xs:enumeration value=">=" />
        <xs:enumeration value="<" />
        <xs:enumeration value="<=" />
      </xs:restriction>
    </xs:simpleType>
  - <xs:complexType name="fromTableType" abstract="true">
    - <xs:annotation>
        <xs:documentation>The base type for all tables used in
          the From clause of the query</xs:documentation>
      </xs:annotation>
    </xs:complexType>
  - <xs:complexType name="archiveTableType" mixed="false">
    - <xs:annotation>
        <xs:documentation>Same as a tableType with an
          additional archive name</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
        - <xs:extension base="tns:fromTableType">
          <xs:attribute name="Archive" type="xs:string"
            use="required" />
          <xs:attribute name="Name" type="xs:string"
            use="required" />
          <xs:attribute name="Alias" type="xs:string"
            use="required" />
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  - <xs:complexType name="tableType" mixed="false">
    - <xs:annotation>
        <xs:documentation>Represents a table with its name and
          its alias name</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
        - <xs:extension base="tns:fromTableType">
          <xs:attribute name="Name" type="xs:string"
            use="required" />
          <xs:attribute name="Alias" type="xs:string"
            use="required" />
        </xs:extension>
        </xs:complexContent>
    </xs:complexType>
  - <xs:complexType name="xMatchTableAliasType" abstract="true">
    - <xs:annotation>
        <xs:documentation>The base type for all table inclusion
          or drop types used in a cross match
          expression</xs:documentation>
        </xs:annotation>
    </xs:complexType>
  - <xs:complexType name="includeTableType" mixed="false">
    - <xs:annotation>
        <xs:documentation>Used for adding a table for the
          Xmatch operation</xs:documentation>
        </xs:annotation>
    - <xs:complexContent mixed="false">
        - <xs:extension base="tns:xMatchTableAliasType">
          <xs:attribute name="Name" type="xs:string"
            use="required" />
        </xs:extension>
        </xs:complexContent>
    </xs:complexType>
  - <xs:complexType name="dropTableType" mixed="false">
    - <xs:annotation>
        <xs:documentation>Used for avoiding a table in
          Xmatch</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
        - <xs:extension base="tns:xMatchTableAliasType">
          <xs:attribute name="Name" type="xs:string"
            use="required" />
        </xs:extension>
        </xs:complexContent>
    </xs:complexType>
  - <xs:complexType name="searchType" abstract="true">
    - <xs:annotation>
        <xs:documentation>The base type for searches in Where
          and Having clauses of the query</xs:documentation>
        </xs:annotation>
    </xs:complexType>
  - <xs:complexType name="intersectionSearchType" mixed="false">
    - <xs:annotation>
        <xs:documentation>Represents expressions like A And
          B</xs:documentation>
        </xs:annotation>
    - <xs:complexContent mixed="false">
        - <xs:extension base="tns:searchType">
          - <xs:sequence>
            <xs:element name="Condition"
              type="tns:searchType" minOccurs="2"
              maxOccurs="2" />
          </xs:sequence>
        </xs:extension>
        </xs:complexContent>
    </xs:complexType>
  - <xs:complexType name="unionSearchType" mixed="false">
    - <xs:annotation>
```

```
        <xs:documentation>Represents expressions like A Or
          B</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
      - <xs:extension base="tns:searchType">
        - <xs:sequence>
            <xs:element name="Condition"
              type="tns:searchType" minOccurs="2"
              maxOccurs="2" />
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  - <xs:complexType name="xMatchType" mixed="false">
    - <xs:annotation>
        <xs:documentation>A cross match
          expression</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
      - <xs:extension base="tns:searchType">
        - <xs:sequence>
            <xs:element name="Table"
              type="tns:xMatchTableAliasType"
              minOccurs="2" maxOccurs="unbounded" />
            <xs:element name="Nature"
              type="tns:comparisonType" />
            <xs:element name="Sigma"
              type="tns:numberType" />
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  - <xs:complexType name="likePredType" mixed="false">
    - <xs:annotation>
        <xs:documentation>The Like expression of a
          query</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
      - <xs:extension base="tns:searchType">
        - <xs:sequence>
            <xs:element name="Arg"
              type="tns:scalarExpressionType" />
            <xs:element name="Pattern"
              type="tns:atomType" />
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  - <xs:complexType name="notLikePredType" mixed="false">
    - <xs:annotation>
        <xs:documentation>The Not Like expression of a
          query</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
        <xs:extension base="tns:likePredType" />
      </xs:complexContent>
    </xs:complexType>
  - <xs:complexType name="closedSearchType" mixed="false">
    - <xs:annotation>
        <xs:documentation>Represents expressions like
          (A)</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
      - <xs:extension base="tns:searchType">
        - <xs:sequence>
            <xs:element name="Condition"
              type="tns:searchType" />
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  - <xs:complexType name="comparisonPredType" mixed="false">
    - <xs:annotation>
        <xs:documentation>Represents the Comparison of two
          expressions</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
      - <xs:extension base="tns:searchType">
        - <xs:sequence>
            <xs:element name="Arg"
              type="tns:scalarExpressionType"
              minOccurs="2" maxOccurs="2" />
          </xs:sequence>
          <xs:attribute name="Comparison"
            type="tns:comparisonType" use="required" />
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  - <xs:complexType name="betweenPredType" mixed="false">
    - <xs:annotation>
        <xs:documentation>Represents the Between expression
          of a query</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
      - <xs:extension base="tns:searchType">
        - <xs:sequence>
            <xs:element name="Arg"
              type="tns:scalarExpressionType"
              minOccurs="3" maxOccurs="3" />
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  - <xs:complexType name="notBetweenPredType" mixed="false">
    - <xs:annotation>
        <xs:documentation>Represents the Not Between
          expression of a query</xs:documentation>
      </xs:annotation>
    - <xs:complexContent mixed="false">
        <xs:extension base="tns:betweenPredType" />
      </xs:complexContent>
    </xs:complexType>
```

```
- <xs:complexType name="inverseSearchType" mixed="false">
  - <xs:annotation>
      <xs:documentation>Represents expressions like Not
        A</xs:documentation>
    </xs:annotation>
  - <xs:complexContent mixed="false">
    - <xs:extension base="tns:searchType">
      - <xs:sequence>
          <xs:element name="Condition"
            type="tns:searchType" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="regionSearchType" mixed="false">
  - <xs:annotation>
      <xs:documentation>Represents the Regions such as circle
        in Where clause</xs:documentation>
    </xs:annotation>
  - <xs:complexContent mixed="false">
    - <xs:extension base="tns:searchType">
      - <xs:sequence>
          <xs:element name="Region"
            type="reg:regionType" />
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
- <xs:complexType name="havingType">
  - <xs:annotation>
      <xs:documentation>Represents the Having expression
        part of a query</xs:documentation>
    </xs:annotation>
  - <xs:sequence>
      <xs:element name="Condition" type="tns:searchType" />
    </xs:sequence>
  </xs:complexType>
- <xs:complexType name="groupByType">
  - <xs:annotation>
      <xs:documentation>Represents the Group By expression
        part of a query</xs:documentation>
    </xs:annotation>
  - <xs:sequence>
      <xs:element name="Column"
        type="tns:columnReferenceType"
        maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
- <xs:complexType name="whereType">
  - <xs:annotation>
      <xs:documentation>Represents the Where part of the
        query</xs:documentation>
    </xs:annotation>
  - <xs:sequence>
      <xs:element name="Condition" type="tns:searchType" />
    </xs:sequence>
  </xs:complexType>
- <xs:complexType name="fromType">
  - <xs:annotation>
      <xs:documentation>Represents the From part of the
        query</xs:documentation>
    </xs:annotation>
  - <xs:sequence>
      <xs:element name="Table" type="tns:fromTableType"
        maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
- <xs:complexType name="selectionListType">
  - <xs:annotation>
      <xs:documentation>List of items to be selected in the
        Query</xs:documentation>
    </xs:annotation>
  - <xs:sequence>
```

```
        <xs:element name="Item" type="tns:selectionItemType"
          maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  - <xs:complexType name="selectionLimitType">
    - <xs:annotation>
        <xs:documentation>Represents the TOP part of a
          query</xs:documentation>
      </xs:annotation>
      <xs:attribute name="Top" type="xs:unsignedInt" />
    </xs:complexType>
  - <xs:simpleType name="orderDirectionType">
    - <xs:annotation>
        <xs:documentation>Ascending or Descending order of an
          Order by term</xs:documentation>
      </xs:annotation>
    - <xs:restriction base="xs:string">
        <xs:enumeration value="ASC" />
        <xs:enumeration value="DESC" />
      </xs:restriction>
    </xs:simpleType>
  - <xs:complexType name="orderOptionType">
    - <xs:annotation>
        <xs:documentation>Option for setting the direction for
          Order By</xs:documentation>
      </xs:annotation>
      <xs:attribute name="Direction" type="tns:orderDirectionType"
        use="required" />
    </xs:complexType>
  - <xs:complexType name="orderType">
    - <xs:annotation>
        <xs:documentation>Represents the ORDER BY part of a
          query</xs:documentation>
      </xs:annotation>
    - <xs:sequence>
        <xs:element name="Expression"
          type="tns:scalarExpressionType" />
      <xs:element name="Order" type="tns:orderOptionType"
         minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
_ <xs:complexType name="orderExpressionType">
  _ <xs:annotation>
      <xs:documentation>List of expressions in which order the
         results should be provided</xs:documentation>
    </xs:annotation>
  _ <xs:sequence>
      <xs:element name="Item" type="tns:orderType"
         maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
_ <xs:element name="Select" type="tns:selectType">
  _ <xs:annotation>
      <xs:documentation>The only permitted root element of a
         query, the SELECT element</xs:documentation>
    </xs:annotation>
  </xs:element>
_ <xs:complexType name="selectType">
  _ <xs:annotation>
      <xs:documentation>The SELECT part of a
         query</xs:documentation>
    </xs:annotation>
  _ <xs:sequence>
      <xs:element name="Allow"
         type="tns:selectionOptionType" minOccurs="0" />
      <xs:element name="Restrict"
         type="tns:selectionLimitType" minOccurs="0" />
      <xs:element name="SelectionList"
         type="tns:selectionListType" />
      <xs:element name="From" type="tns:fromType"
         minOccurs="0" />
      <xs:element name="Where" type="tns:whereType"
         minOccurs="0" />
      <xs:element name="GroupBy" type="tns:groupByType"
         minOccurs="0" />
```

```
        <xs:element name="Having" type="tns:havingType"
          minOccurs="0" />
        <xs:element name="OrderBy"
          type="tns:orderExpressionType" minOccurs="0" />
      </xs:sequence>
    </xs:complexType>
  </xs:schema>
```