IVOA Astronomical Data Query Language Version 0.91

IVOA Working Draft 2005-02-25

This version:

0.91 http://www.ivoa.net/internal/IVOA/IvoaVOQL/ADQL-0.91.pdf

Previous versions:

0.9 http://www.ivoa.net/internal/IVOA/IvoaVOQL/ADQL-0.9.pdf

0.8.1 http://www.ivoa.net/internal/IVOA/IvoaVOQL/ADQL-0.8.1.pdf

0.8 http://www.ivoa.net/internal/IVOA/IvoaVOQL/ADQL-0.8.pdf

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

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

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 its string representation, ADQL/s . This version is ready for promotion to IVOA standard.

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 It is a draft document and may be updated, replaced, or superseded by other documents at any time.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 full XSD for ADQL/x 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 languages are often expressed in the "Backus Naur Form" BNF[1]. BNF is used by popular tools such as LEX and YACC[2] for

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

producing parsers for a given syntax. Section 6 provides the YACC type grammar for ADQL/s.

The BNF exactly defines the form of SQL92 which is ADQL/s. In essence this is any valid SQL statement. There are some restrictions

2.1.1 Built-in Functions

In ADQL built-in functions which are defined on the server system may be called. The minimum set of supported functions are defined in the SkyNodeInterface specification. These built-in functions are "scalar-valued functions". These would include, e.g., a function to provide great circle distance, converter such as from sexagesimal to decimal, and unit converters. The SkynodeInterface specification also defines a method by which all functions available on the server may be discovered. If a user knows that certain functions exist in the target system (SkyNode etc.), the user may use such functions in ADQL. An example of a function would be (in ADQL/s):

```
Select HEALPIXID(a.ra, a.dec), a.ra, a.dec from photobjall a
```

A concise set of common built-in functions that represent the necessary astronomical functionality, together with their standard function names, will be defined in later versions of the ADQL specification.

2.1.2 INTO clause

INTO is supported for VOSpace. In SQL we may use select ‘into’ to create a new table or insert data ‘into’ an existing table in the system. In ADQL this will probably be a VOSpace endpoint wherein the file/table will be created or appended to. How that is specifed is not part of ADQL. ADQL simply supports syntax to allow to specification of a destination, e.g.:

```
Select g.* into VOS:/JHU/gal from galaxy g where g.redshift > 3.5
```

2.1.3 Comments

Comments will only be supported using the /* */ syntax to delimit comments. Comments are only supported before or after the main query – they may not be interspersed with the actual query.

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
```

---

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

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. See subsection 2.4 for its detailed specification.

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**.

2.2.4 XMATCH

XMATCH implies crossmatch between two or more astronomical catalogues. This will only make sense at a portal or a SkyNode which accepts a table uploaded for matching against. Although the semantic meaning of XMATCH is defined more precisely in the SkyNodeInterface specification, this document only specifies the syntax. XMATCH appears in the where clause and looks like a function. Each parameter is a table to be crossmatched, the final parameter is the sigma value for the chi-square match. Here is an example in ADQL/s:

```
SELECT o.objId, o.ra, o.r, o.type, t.objId
  FROM SDSS:PhotoPrimary o,
       TWOMASS:PhotoPrimary t
  WHERE XMATCH(o,t,3.5)
        AND Region('Circle J2000 181.3 -0.76 6.5')
        AND o.type=3
```

Please note this is a change as of ADQL0.8. Prior to this the sigma value was outside the bracket as in XMATCH(o, t)<3.5.

2.2.5 XPATH for Columns

To support Xquery as well as SQL, and since some of our data formats are described as XSD, it will be possible to express selections and selection criteria as a simple Xpath. Square brackets([,]) and standard operators such as parent are NOT supported. An example of a valid query of this form would be

```
Select /Resource/Contact/Name from Resource where /Resource/Type
like ‘catalog’
```

2.2.6 Returning subset of records – TOP

ADQL supports the top syntax to return only the first N records from a query, e.g.,

```
Select top 10 g.* from galaxy g
```

The semantics of this may vary on different database management systems. In ADQL the assumption is that top returns the first N records satisfying the criteria specified in the query.

2.2.7 Units

ADQL allows units for all constant values specified in the query. These are optional. ADQL does not specify what the units mean, and it simply allows for them syntactically specified, e.g:

```
Select g.* from galaxy g where g.gmag > 100 Jansky
```

2.2.8 Table Names with special chars\

ADQL supports the use of '[ ]' to enclose literal names which may otherwise cause parse errors. For example if a table name starts with a number the parser could not deal with this but the following is valid :

```
Select a.* from [2df] a
```

This is also true for table names with spaces in or tables whose names are reserved words. Many database systems also support this syntax.

2.3 Version information

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/

2.4 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 an ADQL/s string is converted to ADQL/x this Region string will be converted to XML. If the resulting ADQL/x is converted back to ADQL/s 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 ADQL/s might be as follows:

```
SELECT a.objid, a.ra, a.dec
FROM SDSSDR2:Photoprimary a
WHERE Region('CIRCLE J2000 181.3 -0.76 6.5')
```

This would be represented in ADQL/x as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<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/v0.9">
  <SelectionList>
    <Item xsi:type="columnReferenceType" Table="a" Name="objid" />
    <Item xsi:type="columnReferenceType" Table="a" Name="ra" />
  </SelectionList>
  <From>
    <Table xsi:type="archiveTableType" Archive="SDSSDR2"
Name="Photoprimary" Alias="a" />
  </From>
  <Where>
    <Condition xsi:type="regionSearchType">
      <Region xmlns:q1="http://www.ivoa.net/xml/STC/STCregion/v1.10"
xsi:type="q1:circleType" unit="deg">
        <q1:Center>181.3 -0.76</q1:Center>
        <q1:Radius>6.5</q1:Radius>
      </Region>
    </Condition>
  </Where>
</Select>
```

4 Changes from previous versions

- New bnf for ANTLR
- New STC included

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 Grammar

Below is the ANTLR grammar used to produce the parser in C#.

```
options {
    language = "CSharp";
    namespace="net.ivoa.adql";
}


// PARSER
*******************************************************************
*************

class SqlParser extends Parser;
options {
    k = 2;
}
{
    public      ADQLBuilder adqlb = new ADQLBuilder();
    public void init(){
    adqlb.sp = this;
    }
}
// starting rule
sql : selectStatement (SEMICOLON)? EOF;
selectStatement : queryExpression (computeClause)? (forClause)?
(optionClause)?;
```

```
queryExpression : subQueryExpression (unionOperator
subQueryExpression)* (orderByClause)?;

subQueryExpression : querySpecification | LPAREN queryExpression
RPAREN;

querySpecification : {init();adqlb.sql();}

    selectClause (fromClause)? (whereClause)? (groupByClause)?
(havingClause)? ;

selectClause :SELECT (all_distinct)? (restrictClause)? selectList;

restrictClause : {adqlb.AddSelectTop();}

    TOP Integer (PERCENT)? (WITH TIES)?;

all_distinct: ALL | DISTINCT;

whereClause : {adqlb.where();}

    WHERE searchCondition;

orderByClause : ORDER BY expression (orderType)?
{adqlb.AddOrderBy();} (COMMA expression (orderType)?
{adqlb.AddOrderBy();})*;

orderType: {adqlb.StackOrderType();} (ASC | DESC);

groupByClause : GROUP BY (ALL)? expression (COMMA expression)*
(WITH (CUBE | ROLLUP) )? {adqlb.AddGroupBy();};

havingClause : HAVING predicate {adqlb.AddHaving();};

optionClause : OPTION LPAREN queryHint (COMMA queryHint)* RPAREN;

queryHint :

    (HASH | ORDER) GROUP

    | (CONCAT | HASH | MERGE) UNION

    | (LOOP | MERGE | HASH) JOIN

    | FAST Integer

    | FORCE ORDER

    | MAXDOP Integer

    | ROBUST PLAN

    | KEEP PLAN

    | KEEPFIXED PLAN

    | EXPAND VIEWS;

forClause :

FOR (

    BROWSE

    | XML (RAW | AUTO | EXPLICIT) (COMMA XMLDATA)? (COMMA
ELEMENTS)? (COMMA BINARY BASE64)

    );

computeClause :

    COMPUTE

    // only allowed functions are: AVG, COUNT, MAX, MIN, STDEV,
STDEVP, VAR, VARP, SUM

    identifier LPAREN expression RPAREN
    (COMMA identifier LPAREN expression RPAREN)*
    (BY expression (COMMA expression)* )?;
searchCondition :
    subSearchCondition ( {adqlb.setLastSearchType();}(AND | OR)
subSearchCondition)*;
subSearchCondition :
(NOT)? (
    (LPAREN searchCondition RPAREN) => LPAREN searchCondition
RPAREN
    | predicate | xMatch | region
       ){adqlb.AddWhereComparison();};
predicate :


(
 expression
    (
     // expression comparisonOperator expression
     comparisonOperator (
     expression
     | (ALL | SOME | ANY) LPAREN selectStatement RPAREN
     )
     | IS (NOT)? NULL
     | (NOT)? (
     LIKE expression (ESCAPE expression)? // only single char
     | BETWEEN expression AND expression
     | IN LPAREN (
         (selectStatement) => selectStatement
         | expression (COMMA expression)*
         ) RPAREN
     )
     | CONTAINS LPAREN (dbObject | STAR) COMMA (stringLiteral |
Variable) RPAREN
     | FREETEXT LPAREN (dbObject | STAR) COMMA (stringLiteral |
Variable) RPAREN
     )
 | EXISTS LPAREN selectStatement RPAREN
);
region: REGION LPAREN regionClause RPAREN {adqlb.StackRegion();};
regionClause: QuotedIdentifier;
xMatch: XMATCH LPAREN xAlias (COMMA xAlias)* (
    (COMMA xSigma RPAREN ) |
    (RPAREN LESSTHAN xSigma)
    ){adqlb.StackXMatch();};
xSigma: number;
selectList : selectItem ( COMMA selectItem )*;
selectItem :
(STAR // "*, *" is a valid select list
| (
    // starts with: "alias = column_name"
    (alias2) => (
                (alias2 dbObject COMMA) => alias2 column
                | (alias2 dbObject (binaryOperator | LPAREN)) =>
alias2 expression
                | (alias2 column) => alias2 column
                | (alias2 expression) => alias2 expression
              )
    // all table columns: "table.*"
    | (tableColumns) => tableColumns
    | (explicitFunction) => explicitFunction
    | (function) => function
    // some shortcuts:
    | (dbObject (alias1)? COMMA) => column (alias1)?

    | (dbObject (binaryOperator | LPAREN) ) => expression (alias1)?

    | (column) => column (alias1)?
    | (expression) => expression (alias1)?
  )
){adqlb.AddSelectItem();};
fromClause : {adqlb.from();}FROM tableSource (COMMA tableSource)*;
tableSource : subTableSource (joinedTable)*;
subTableSource :
(
 LPAREN (
     (joinedTables) => joinedTables RPAREN
     | (queryExpression) => queryExpression RPAREN alias1 //
"derived table", mandatory alias
     )
 | (function) => function (alias1)?
 | (archiveTable)? dbObject (alias1)? ( (WITH)? LPAREN tableHint
(COMMA tableHint)* RPAREN )?
 | Variable (alias1)?
 | (CONTAINSTABLE | FREETEXTTABLE) LPAREN
 dbObject COMMA (dbObject | STAR) COMMA (stringLiteral | Variable)
(COMMA Integer)?
 RPAREN (alias1)?
 | COLON COLON function (alias1)? // built-in function
){adqlb.AddFromItem();};
joinedTable :
CROSS JOIN subTableSource
// "joinHint JOIN" is invalid join expression
| ( (INNER | (LEFT | RIGHT | FULL) (OUTER)? ) (joinHint)? )? JOIN
tableSource ON searchCondition;
joinedTables : subTableSource (joinedTable)+;
joinHint : LOOP | HASH | MERGE | REMOTE;
tableHint :
INDEX (
    LPAREN (identifier | Integer) ( COMMA (identifier | Integer) )*
RPAREN
    | ASSIGNEQUAL identifier    // old index hint syntax
      )
| FASTFIRSTROW
| HOLDLOCK
| NOLOCK
| PAGLOCK
| READCOMMITED
| READPAST
| READUNCOMMITED
| REPEATABLEREAD
| ROWLOCK
| SERIALIZABLE
| TABLOCK
| TABLOCKX
| UPDLOCK
| XLOCK
;
collate : COLLATE identifier
;
alias1 : // alias name can also be single-quoted literal (but not
for table names)
(AS)?
{adqlb.StackAlias();}
(
 identifier
 | stringLiteral
 | keywordAsIdentifier
);
alias2 :
(
 identifier
 | stringLiteral
 | keywordAsIdentifier
)
ASSIGNEQUAL;
xAlias: {adqlb.StackXAlias();} (NOT alias1 | QUESTIONMARK alias1 |
alias1);
tableColumns :  dbObject DOT_STAR;

column :
(PLUS)* // "++column_name" is valid and updatable column name
(
 dbObject
 // for expression like "(column)" SQL Server returns updatable
column
 | LPAREN column RPAREN
)
(collate)? // it is not well documented but COLLATE can be used
almost anywhere ...
;

expression :
    subExpression (binaryOperator subExpression )*
{adqlb.CloseExpression();};
subExpression :
(unaryOperator)?
(
 constant
 |  Variable
 | (explicitFunction) => explicitFunction
 | (function) => function
 | LPAREN {adqlb.OpenExpression();} (
     (selectStatement) => selectStatement // select statement
returning a single value
     | expression2
     ) RPAREN
 | dbObject    // column
 | parameterlessFunction
 | caseFunction
 | castFunction
)
(collate)?;   // it is not well documented but COLLATE can be used
almost everywhere ...


expression2: expression;
function : // LEFT and RIGHT keywords are also function names
(dbObject | LEFT | RIGHT ) LPAREN (
                        expression (COMMA expression)*
                        | STAR    // aggregate functions like
Count(), Checksum() accept "*" as a parameter
                        | (ALL | DISTINCT) (STAR | expression) //
aggregate function
                        | Variable ASSIGNEQUAL expression (COMMA
Variable ASSIGNEQUAL expression)*
                       )?
RPAREN;
starArg: STAR {adqlb.StackStar();};
//don't use these directly
aggregateFunction:
(AVG|MAX|MIN|SUM|COUNT){adqlb.StackAggregateOp();} LPAREN
(starArg|expression) RPAREN;
trigFunction:
(SIN|COS|TAN|COT|ASIN|ACOS|ATAN|ATAN2){adqlb.StackTrigOp();} LPAREN
expression RPAREN;
mathFunction:
(CEILING|DEGREES|EXP|FLOOR|LOG|PI|POWER|RADIANS|SQRT|SQUARE|LOG10|R
AND|ROUND|TRUNCATE) {adqlb.StackMathOp();}
    LPAREN expression RPAREN ;
//use this one
explicitFunction: (aggregateFunction | trigFunction | mathFunction)
{adqlb.CloseFunction();};


archiveTable : {adqlb.StackArchiveTable();}
    identifier COLON;
caseFunction :
CASE (
    expression (WHEN expression THEN expression)+
    | (WHEN searchCondition THEN expression)+    // boolean
expression
     )
(ELSE expression)? END
;
```

```
castFunction : CAST LPAREN expression AS identifier (LPAREN Integer
(COMMA Integer)? RPAREN)? RPAREN;


dbObject : {adqlb.StackDBObject();}
(identifier | IDENTITYCOL | ROWGUIDCOL | keywordAsIdentifier)
    (
     DOT (identifier | IDENTITYCOL | ROWGUIDCOL |
keywordAsIdentifier)
     | (DOT DOT) => DOT DOT (identifier | IDENTITYCOL | ROWGUIDCOL
| keywordAsIdentifier)
    )*;
parameterlessFunction : // any others ?
CURRENT_TIMESTAMP
| CURRENT_USER
| SESSION_USER
| SYSTEM_USER
;
systemVariable :
F_CONNECTIONS
| F_CPU_BUSY
| F_CURSOR_ROWS
| F_DATEFIRST
| F_DBTS
| F_ERROR
| F_FETCH_STATUS
| F_IDENTITY
| F_IDLE
| F_IO_BUSY
| F_LANGID
| F_LANGUAGE
| F_LOCK_TIMEOUT
| F_MAX_CONNECTIONS
| F_MAX_PRECISION
| F_NESTLEVEL
| F_OPTIONS
| F_PACK_RECEIVED
| F_PACK_SENT
| F_PACKET_ERRORS
| F_PROCID
| F_REMSERVER
| F_ROWCOUNT
| F_SERVERNAME
| F_SERVICENAME
| F_SPID
| F_TEXTSIZE
| F_TIMETICKS
| F_TOTAL_ERRORS
| F_TOTAL_READ
| F_TOTAL_WRITE
| F_TRANCOUNT
| F_VERSION
;
keywordAsIdentifier :
(
 AUTO
 | BASE64
 | BINARY
 | CAST
 | CONCAT
 | CUBE
 | ELEMENTS
 | EXP
 | EXPAND
 | EXPLICIT
 | FAST
 | FASTFIRSTROW
 | FORCE
 | HASH
 | KEEP
 | KEEPFIXED
 | LOOP
 | MAXDOP
 | MIN
 | MAX
 | MERGE
 | NOLOCK
 | PAGLOCK
 | RAW
| READCOMMITED
| READPAST
| READUNCOMMITED
| REMOTE
| REPEATABLEREAD
| ROBUST
| ROLLUP
| ROWLOCK
| SERIALIZABLE
| TABLOCK
| TABLOCKX
| TIES
| UPDLOCK
| VIEWS
| XLOCK
| XML
| XMLDATA
)
;
stringLiteral : UnicodeStringLiteral | ASCIIStringLiteral;
identifier: NonQuotedIdentifier | QuotedIdentifier;
constant :
    {adqlb.StackIntAtom();} Integer |
    {adqlb.StackRealAtom();} Real |
    NULL |
    {adqlb.StackStringAtom();} stringLiteral |
    HexLiteral |
    Currency |
    ODBCDateTime |
    systemVariable;
unaryOperator : PLUS | MINUS | TILDE
;
binaryOperator : arithmeticOperator | bitwiseOperator;
arithmeticOperator : {adqlb.StackArithmeticOperator();} (PLUS |
MINUS | STAR | DIVIDE | MOD);
bitwiseOperator : AMPERSAND | TILDE | BITWISEOR | BITWISEXOR
;
comparisonOperator :{adqlb.StackComparisonOperator();}
(
    ASSIGNEQUAL | NOTEQUAL1 | NOTEQUAL2 | LESSTHANOREQUALTO1 |
LESSTHANOREQUALTO2
    | LESSTHAN | GREATERTHANOREQUALTO1 | GREATERTHANOREQUALTO2 |
GREATERTHAN
);
logicalOperator : ALL | AND | ANY | BETWEEN | EXISTS | IN | LIKE |
NOT | OR | SOME
```

```
;
unionOperator : UNION (ALL)?
;
number: (SIGN)? Number;

// LEXER
*******************************************************************
**************

class SqlLexer extends Lexer;

options {
    testLiterals = false;
    k = 2;
    caseSensitive = false;
    caseSensitiveLiterals = false;
    charVocabulary='\u0000'..'\uFFFE';
}

tokens {
    ADD = "add" ;
    ALL = "all" ;
    ALTER = "alter" ;
    AND = "and" ;
    ANY = "any" ;
    AS = "as" ;
    ASC = "asc" ;
    AUTHORIZATION = "authorization" ;
    AUTO = "auto" ;
    BACKUP = "backup" ;
    BASE64 = "base64" ;
    BEGIN = "begin" ;
    BETWEEN = "between" ;
    BINARY = "binary" ;
    BREAK = "break" ;
    BROWSE = "browse" ;
    BULK = "bulk" ;
    BY = "by" ;
    CASCADE = "cascade" ;
    CASE = "case" ;
    CAST = "cast" ;
```

```
CHECK = "check" ;
CHECKPOINT = "checkpoint" ;
CLOSE = "close" ;
CLUSTERED = "clustered" ;
// COALESCE = "coalesce" ;
COLLATE = "collate" ;
COLUMN = "column" ;
COMMIT = "commit" ;
COMPUTE = "compute" ;
CONCAT = "concat" ;
CONSTRAINT = "constraint" ;
CONTAINS = "contains" ;
CONTAINSTABLE = "containstable" ;
CONTINUE = "continue" ;
// CONVERT = "convert" ;
CREATE = "create" ;
CROSS = "cross" ;
CUBE = "cube" ;
CURRENT = "current" ;
CURRENT_DATE = "current_date" ;
CURRENT_TIME = "current_time" ;
CURRENT_TIMESTAMP = "current_timestamp" ;
CURRENT_USER = "current_user" ;
CURSOR = "cursor" ;
DATABASE = "database" ;
DBCC = "dbcc" ;
DEALLOCATE = "deallocate" ;
DECLARE = "declare" ;
DEFAULT = "default" ;
DELETE = "delete" ;
DENY = "deny" ;
DESC = "desc" ;
DISK = "disk" ;
DISTINCT = "distinct" ;
DISTRIBUTED = "distributed" ;
DOUBLE = "double" ;
DROP = "drop" ;
// DUMMY = "dummy" ;
DUMP = "dump" ;
ELEMENTS = "elements" ;
```

```
ELSE = "else" ;
END = "end" ;
ERRLVL = "errlvl" ;
ESCAPE = "escape" ;
EXCEPT = "except" ;
EXEC = "exec" ;
EXECUTE = "execute" ;
EXISTS = "exists" ;
EXIT = "exit" ;
EXPAND = "expand" ;
EXPLICIT = "explicit" ;
FAST = "fast" ;
FASTFIRSTROW = "fastfirstrow" ;
FETCH = "fetch" ;
FILE = "file" ;
FILLFACTOR = "fillfactor" ;
FOR = "for" ;
FORCE = "force" ;
FOREIGN = "foreign" ;
FREETEXT = "freetext" ;
FREETEXTTABLE = "freetexttable" ;
FROM = "from" ;
FULL = "full" ;
FUNCTION = "function" ;
GOTO = "goto" ;
GRANT = "grant" ;
GROUP = "group" ;
HASH = "hash" ;
HAVING = "having" ;
HOLDLOCK = "holdlock" ;
IDENTITY = "identity" ;
IDENTITY_INSERT = "identity_insert" ;
IDENTITYCOL = "identitycol" ;
IF = "if" ;
IN = "in" ;
INDEX = "index" ;
INNER = "inner" ;
INSERT = "insert" ;
INTERSECT = "intersect" ;
INTO = "into" ;
```

```
IS = "is" ;
JOIN = "join" ;
KEEP = "keep" ;
KEEPFIXED = "keepfixed" ;
KEY = "key" ;
KILL = "kill" ;
LEFT = "left" ;
LIKE = "like" ;
LINENO = "lineno" ;
LOAD = "load" ;
LOOP = "loop" ;
MAXDOP = "maxdop" ;
MERGE = "merge" ;
NATIONAL = "national" ;
NOCHECK = "nocheck" ;
NOLOCK = "nolock" ;
NONCLUSTERED = "nonclustered" ;
NOT = "not" ;
NULL = "null" ;
// NULLIF = "nullif" ;
OF = "of" ;
OFF = "off" ;
OFFSETS = "offsets" ;
ON = "on" ;
OPEN = "open" ;
OPENDATASOURCE = "opendatasource" ;
OPENQUERY = "openquery" ;
OPENROWSET = "openrowset" ;
OPENXML = "openxml" ;
OPTION = "option" ;
OR = "or" ;
ORDER = "order" ;
OUTER = "outer" ;
OVER = "over" ;
PAGLOCK = "paglock" ;
PERCENT = "percent" ;
PLAN = "plan" ;
PRECISION = "precision" ;
PRIMARY = "primary" ;
PRINT = "print" ;
```

```
PROC = "proc" ;
PROCEDURE = "procedure" ;
PUBLIC = "public" ;
RAISERROR = "raiserror" ;
RAW = "raw" ;
READ = "read" ;
READCOMMITED = "readcommited" ;
READPAST = "readpast" ;
READTEXT = "readtext" ;
READUNCOMMITED = "readuncommited" ;
RECONFIGURE = "reconfigure" ;
REFERENCES = "references" ;
REGION = "region";
REMOTE = "remote" ;
REPEATABLEREAD = "repeatableread" ;
REPLICATION = "replication" ;
RESTORE = "restore" ;
RESTRICT = "restrict" ;
RETURN = "return" ;
REVOKE = "revoke" ;
RIGHT = "right" ;
ROBUST = "robust" ;
ROLLBACK = "rollback" ;
ROLLUP = "rollup" ;
ROWCOUNT = "rowcount" ;
ROWGUIDCOL = "rowguidcol" ;
ROWLOCK = "rowlock" ;
RULE = "rule" ;
SAVE = "save" ;
SCHEMA = "schema" ;
SELECT = "select" ;
SERIALIZABLE = "serializable" ;
SESSION_USER = "session_user" ;
SET = "set" ;
SETUSER = "setuser" ;
SHUTDOWN = "shutdown" ;
SOME = "some" ;
STATISTICS = "statistics" ;
SYSTEM_USER = "system_user" ;
TABLE = "table" ;
```

```
TABLOCK = "tablock" ;
TABLOCKX = "tablockx" ;
TEXTSIZE = "textsize" ;
THEN = "then" ;
TIES = "ties" ;
TO = "to" ;
TOP = "top" ;
TRAN = "tran" ;
TRANSACTION = "transaction" ;
TRIGGER = "trigger" ;
TSEQUAL = "tsequal" ;
UNION = "union" ;
UNIQUE = "unique" ;
UPDATE = "update" ;
UPDATETEXT = "updatetext" ;
UPDLOCK = "updlock" ;
USE = "use" ;
USER = "user" ;
VALUES = "values" ;
VARYING = "varying" ;
VIEW = "view" ;
VIEWS = "views" ;
WAITFOR = "waitfor" ;
WHEN = "when" ;
WHERE = "where" ;
WHILE = "while" ;
WITH = "with" ;
WRITETEXT = "writetext" ;
XLOCK = "xlock" ;
XMATCH = "xmatch";
XML = "xml" ;
XMLDATA = "xmldata" ;

// system variables
F_CONNECTIONS = "@@connections" ;
F_CPU_BUSY = "@@cpu_busy" ;
F_CURSOR_ROWS = "@@cursor_rows" ;
F_DATEFIRST = "@@datefirst" ;
F_DBTS = "@@dbts" ;
F_ERROR = "@@error" ;
```

```
F_FETCH_STATUS = "@@fetch_status" ;
F_IDENTITY = "@@identity" ;
F_IDLE = "@@idle" ;
F_IO_BUSY = "@@io_busy" ;
F_LANGID = "@@langid" ;
F_LANGUAGE = "@@language" ;
F_LOCK_TIMEOUT = "@@lock_timeout" ;
F_MAX_CONNECTIONS = "@@max_connections" ;
F_MAX_PRECISION = "@@max_precision" ;
F_NESTLEVEL = "@@nestlevel" ;
F_OPTIONS = "@@options" ;
F_PACK_RECEIVED = "@@pack_received" ;
F_PACK_SENT = "@@pack_sent" ;
F_PACKET_ERRORS = "@@packet_errors" ;
F_PROCID = "@@procid" ;
F_REMSERVER = "@@remserver" ;
F_ROWCOUNT = "@@rowcount" ;
F_SERVERNAME = "@@servername" ;
F_SERVICENAME = "@@servicename" ;
F_SPID = "@@spid" ;
F_TEXTSIZE = "@@textsize" ;
F_TIMETICKS = "@@timeticks" ;
F_TOTAL_ERRORS = "@@total_errors" ;
F_TOTAL_READ = "@@total_read" ;
F_TOTAL_WRITE = "@@total_write" ;
F_TRANCOUNT = "@@trancount" ;
F_VERSION = "@@version" ;

//math
CEILING = "ceiling";
DEGREES = "degrees";
EXP = "exp";
FLOOR = "floor";
LOG = "log";
PI = "pi";
POWER = "power";
RADIANS ="radians";
SQRT = "sqrt";
SQUARE = "square";
LOG10 = "log10";
```

```
    RAND = "rand";
    ROUND = "round";
    TRUNCATE ="truncate";

    //trig
    SIN = "sin";
    COS = "cos";
    TAN = "tan";
    COT = "cot";
    ASIN = "asin";
    ACOS = "acos";
    ATAN = "atan";
    ATAN2 = "atan2";

    //aggregate
    MAX = "max";
    MIN = "min";
    SUM = "sum";
    AVG = "avg";
    COUNT = "count";
}

// Operators

protected DOT:; // generated as a part of Number rule
COLON : ':' ;
COMMA : ',' ;
SEMICOLON : ';' ;

LPAREN : '(' ;
RPAREN : ')' ;
//LSQUARE : '[' ;
//RSQUARE : ']' ;

ASSIGNEQUAL : '=' ;
NOTEQUAL1 : "<>" ;
NOTEQUAL2 : "!=" ;
LESSTHANOREQUALTO1 : "<=" ;
LESSTHANOREQUALTO2 : "!>" ;
LESSTHAN : "<" ;
```

```
GREATERTHANOREQUALTO1 : ">=" ;
GREATERTHANOREQUALTO2 : "!<" ;
GREATERTHAN : ">" ;

DIVIDE : '/' ;
PLUS : '+' ;
MINUS : '-' ;
STAR : '*' ;
MOD : '%' ;

AMPERSAND : '&' ;
TILDE : '~' ;
BITWISEOR : '|' ;
BITWISEXOR : '^' ;
DOT_STAR : ".*" ;

NOT : '!';
QUESTIONMARK : '?';

Whitespace
    : (' ' | '\t' | '\n' | '\r')
    { _ttype = Token.SKIP; }
    ;

// COMMENTS
SingleLineComment
    : "--"( ~('\r' | '\n') )*
    { _ttype = Token.SKIP; }
    ;

MultiLineComment
    : "/*" (~'*')* '*' ('*' | ( ~('*' | '/') (~'*')* '*') )* '/'
    { _ttype = Token.SKIP; }
    ;

// LITERALS

protected Letter : 'a'..'z' | '_' | '#' | '@' | '\u0080'..'\uffe';
protected Digit : '0'..'9';
protected Integer :;
protected Real :;
protected Exponent : 'e' (Sign)? (Digit)+ ;
protected Sign : (PLUS | MINUS);
Number :
      ( (Digit)+ ('.' | 'e') ) => (Digit)+ ( '.' (Digit)*
(Exponent)? | Exponent) { _ttype = Real; }
    | '.' { _ttype = DOT; } ( (Digit)+ (Exponent)? { _ttype = Real;
} )?
    |  (Digit)+ { _ttype = Integer; }
    | "0x" ('a'..'f' | Digit)* { _ttype = HexLiteral; } // "0x" is
valid hex literal
;
protected Currency : // generated as a part of NonQuotedIdentifier
rule
    ('$' | '\u00a3'..'\u00a5' | '\u09f2'..'\u09f3' | '\u0e3f' |
'\u20a0'..'\u20a4' | '\u20a6'..'\u20ab')
    ((Digit)+ ('.' (Digit)* )? | '.' (Digit)+)
;
ODBCDateTime :
    '{' (Whitespace)? ("ts" | 't' | 'd') (Whitespace)?
    ('n')? '\'' (~'\'')* '\'' ( '\'' (~'\'')* '\'' )* (Whitespace)?
'}'
;


NonQuotedIdentifier options { testLiterals = true; } :
(
 (Currency) => Currency { _ttype = Currency; }
 | ('a'..'z' | '_' | '#' | '\u0080'..'\ufffe') (Letter | Digit)* //
first char other than '@'
);


QuotedIdentifier
    :
    (
      '[' (~']')* ']' (']' (~']')* ']')*
    | '"' (~'"')* '"' ('"' (~'"')* '"')*
    )
    ;


Variable
    // test for literals in case of a function begining with '@'
(eg. "@@ERROR")
    options { testLiterals = true; }
```

```
        : '@' (Letter | Digit)+
        ;

    ASCIIStringLiteral
        :
        '\'' (~'\'')* '\'' ( '\'' (~'\'')* '\'' )*
        ;

    UnicodeStringLiteral
        :
        'n' '\'' (~'\'')* '\'' ( '\'' (~'\'')* '\'' )*
        ;

    // Numeric Constants

    protected
    HexLiteral // generated as a part of Number rule
        : // "0x" ('0'..'9' | 'a'..'f')*
      ;
```

7 ADQL XSD

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="http://www.ivoa.net/xml/ADQL/v0.9"
xmlns:tns="http://www.ivoa.net/xml/ADQL/v0.9"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:reg="http://www.ivoa.net/xml/STC/STCregion/v1.10"
elementFormDefault="qualified">
      <xs:import
namespace="http://www.ivoa.net/xml/STC/STCregion/v1.10"
schemaLocation="region-v1.10.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:attribute name="xpathName"
type="xs:string"/>
                  </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:element name="Unit" type="xs:string"
minOccurs="0"/>
                        </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="optional"/>
                  </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="&lt;&gt;"/>
                  <xs:enumeration value="&gt;"/>
                  <xs:enumeration value="&gt;="/>
                  <xs:enumeration value="&lt;"/>
                  <xs:enumeration value="&lt;="/>
            </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="optional"/>
                  </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="alias"/>
                        <xs:attribute name="xpathName"
type="xs:string"/>
                  </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="exclusiveSearchType" mixed="false">
            <xs:annotation>
                  <xs:documentation>Represents SQL NOT IN
expression</xs:documentation>
            </xs:annotation>
            <xs:complexContent mixed="false">
                  <xs:extension base="tns:searchType">
                        <xs:sequence>
                              <xs:element name="Expression"
type="tns:scalarExpressionType"/>
                              <xs:element name="Set"
type="tns:inclusionSetType"/>
                        </xs:sequence>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="inclusionSetType" abstract="true">
            <xs:annotation>
                  <xs:documentation>The base type for selection set in
a SQL IN expression</xs:documentation>
            </xs:annotation>
      </xs:complexType>
      <xs:complexType name="subQuerySet" mixed="false">
            <xs:annotation>
                  <xs:documentation>Represents the subquery in a SQL IN
expression</xs:documentation>
            </xs:annotation>
            <xs:complexContent mixed="false">
                  <xs:extension base="tns:inclusionSetType">
                        <xs:sequence>
                              <xs:element name="selection"
type="tns:selectType"/>
                        </xs:sequence>
                  </xs:extension>
            </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:complexType name="intoType">
            <xs:annotation>
                  <xs:documentation>Represents the SQL INTO
expression</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                  <xs:element name="TableName" type="xs:string"/>
            </xs:sequence>
      </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:complexType name="ConstantListSet" mixed="false">
            <xs:annotation>
```

```
                  <xs:documentation>Represents a list of constants
provided for a SQL IN expression</xs:documentation>
            </xs:annotation>
            <xs:complexContent mixed="false">
                  <xs:extension base="tns:inclusionSetType">
                        <xs:sequence>
                              <xs:element name="Item"
type="tns:literalType" maxOccurs="unbounded"/>
                        </xs:sequence>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="inclusiveSearchType" mixed="false">
            <xs:annotation>
                  <xs:documentation>Represents SQL IN
expression</xs:documentation>
            </xs:annotation>
            <xs:complexContent mixed="false">
                  <xs:extension base="tns:searchType">
                        <xs:sequence>
                              <xs:element name="Expression"
type="tns:scalarExpressionType"/>
                              <xs:element name="Set"
type="tns:inclusionSetType"/>
                        </xs:sequence>
                  </xs:extension>
            </xs:complexContent>
      </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="InTo" type="tns:intoType"
minOccurs="0"/>
                  <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:element name="StartComment" type="xs:string"
minOccurs="0"/>
                  <xs:element name="EndComment" type="xs:string"
minOccurs="0"/>
            </xs:sequence>
      </xs:complexType>
      <xs:complexType name="userDefinedFunctionType" mixed="false">
            <xs:annotation>
                  <xs:documentation>Represents user defined function
expressions</xs:documentation>
            </xs:annotation>
            <xs:complexContent mixed="false">
                  <xs:extension base="tns:scalarExpressionType">
                        <xs:sequence>
                              <xs:element name="Name"
type="xs:string"/>
                              <xs:element name="Params"
type="tns:scalarExpressionType" minOccurs="0" maxOccurs="unbounded"/>
                        </xs:sequence>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
      <xs:simpleType name="jointTableQualifierType">
            <xs:annotation>
                  <xs:documentation>Denotes the type of a Join
operation</xs:documentation>
            </xs:annotation>
            <xs:restriction base="xs:string">
                  <xs:enumeration value="LEFT_OUTER"/>
                  <xs:enumeration value="RIGHT_OUTER"/>
                  <xs:enumeration value="FULL_OUTER"/>
                  <xs:enumeration value="INNER"/>
                  <xs:enumeration value="CROSS"/>
            </xs:restriction>
      </xs:simpleType>
      <xs:complexType name="joinTableType" mixed="false">
            <xs:annotation>
                  <xs:documentation>Represents SQL JOIN
expression</xs:documentation>
            </xs:annotation>
            <xs:complexContent mixed="false">
                  <xs:extension base="tns:fromTableType">
                        <xs:sequence>
                              <xs:element name="Qualifier"
type="tns:jointTableQualifierType"/>
                              <xs:element name="Tables"
type="tns:ArrayOfFromTableType"/>
                              <xs:element name="Condition"
type="tns:comparisonPredType"/>
                        </xs:sequence>
                  </xs:extension>
            </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="ArrayOfFromTableType">
            <xs:annotation>
```

```
                  <xs:documentation>Represents an array of tables in
the from expression</xs:documentation>
            </xs:annotation>
            <xs:sequence>
                  <xs:element name="fromTableType"
type="tns:fromTableType" nillable="true" maxOccurs="unbounded"/>
            </xs:sequence>
      </xs:complexType>
</xs:schema>
```