International

Virtual

Observatory

Alliance

IVOA Astronomical Data Query Language

Version 1.5

IVOA Working Draft 2007 January 26

This version:
1.5-20070126
Latest version:
http://www.ivoa.net/Documents/latest/ADQL.html
Previous version(s):
none

Editor(s):
Pedro Osuna and Inaki Ortiz
Author(s):
Inaki Ortiz, Yuji Shirasaki, Maria A. Nieto-Santisteban, Masatoshi Ohishi, William O'Mullane, Alexander Szalay, Pedro Osuna, the VOQL-TEG and the VOQL Working Group.

Abstract

This document describes the Astronomical Data Query Language (ADQL). ADQL has been developed based on SQL92. This document describes the subset of the SQL grammar supported by ADQL. Special restrictions and extensions to

SQL92 have been defined in order to support generic and astronomy specific operations.

Status of This Document

This is a Working Draft. The first release of this document was 2006 January 22.

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/Documents/.

Contents

1 Introduction

The Astronomical Data Query Language (ADQL) is the language used by the International Virtual Observatory Alliance (IVOA) to represent astronomy queries posted to VO services. The IVOA has developed several standardized protocols to access astronomical data, e.g., SIAP and SSAP for image and spectral data respectively. These protocols might be satisfied using a single table query. However, different VO services have different needs in terms of query complexity and ADQL arises in this context.

The ADQL specification pretends to avoid any distinction between core and advanced or extended functionalities. Hence ADQL has been built according to a single language definition (BNF based [1]). Any service making use of ADQL would then define the level of compliancy to the language. This would allow the notion of core and extension to be service-driven and it would decouple the language from the service specifications.

ADQL is based on the Structured Query Language (SQL), specially on SQL 92. The VO has a number of tabular data sets and many of them are stored in relational databases, making SQL a convenient access means. A subset of the SQL grammar has been extended to support queries which are specific to astronomy.

2 Astronomical Data Query Language (ADQL)

This section describes the ADQL language specification. We will define in subsequent sections the syntax for the special characters, reserved and non-reserved words, identifiers and literals and then, finally, the syntax for the query expression.

The formal notation for syntax of computing languages is often expressed in the “Backus Naur Form” BNF. This syntax is used by popular tools for producing parsers. Appendix A to this document provides the full BNF grammar for ADQL.

The following conventions are used through this document:

- Optional items are enclosed in meta symbols [and]
- A group of items is enclosed in meta symbols { and }
- Repetitive item (zero or more times) are followed by [,…]
- Terminal symbols are enclosed by < and >
- Terminals of meta-symbol characters (=,[,],(,),<,>,*) are surrounded by quotes (“) to distinguish them from meta-symbols
- Case insensitiveness otherwise stated.

2.1 Characters, Keywords, Identifiers and Literals

2.1.1 Characters

The language allows simple Latin letters (lower and upper case, i.e. {aA-zZ}), digits ({0-9}) and the following special characters:

- space
- single quote (’)

- double quote (“)
- percent (%)
- left and right parenthesis
- asterisk (*)
- plus sign (+)
- minus sign (-)
- comma (,)
- period (.)
- solidus (/)
- colon (.)
- less than operator (<)
- equals operator (=)
- greater than operator (>)
- underscore (_)

2.1.2 Keywords and Identifiers

Besides the character set, the language provides a list of reserved and non-reserved keywords plus the syntax description for regular identifiers.

A reserved keyword has a special meaning in ADQL and cannot be used as an identifier. A non-reserved keyword has a special meaning in specific contexts and can be used as an identifier in some other contexts.

The identifiers are used to express, for example, a table or a column reference name.

Both the identifiers and the keywords are case insensitive. They SHALL begin with a letter {aA-zZ}. Subsequent characters shall be letters, underscores or digits {0-9}.

<Latin letter> [{ <underscore> | {<Latin letter> | <digit>} }]

Reserved keywords are:

ALL, AND, AS, ASC, AVG, BETWEEN, BY, COUNT, DESC, DISTINCT, EXISTS, FALSE, FROM, FULL, GROUP, HAVING, IN, INNER, INTO, IS, JOIN, LEFT, LIKE, MAX, MIN, NATURAL, NOT, NULL, ON, OR, ORDER, OUTER, RIGHT, SELECT, SUM, TOP, TRUE, USING, WHERE.

Non-Reserved keywords are:

ABS, ACOS, ASIN, ATAN, ATAN2, CEILING, COS, DEGREES, EXP, FLOOR, LOG, LOG10, MODE, PI, POWER, RADIANS, REGION, RAND, ROUND, SIN, SQRT, UDF, TAN, TRUNCATE.

For practical purposes the language specification should be able to address reserved keyword and special character conflicts. To do so the language provides a way to escape a non-compliant identifier by using the double quote character as a delimiter.

ADQL allows making use of the same quoting mechanism to handle the case sensitiveness if needed.

2.1.3 Literals

Finally we define the syntax rules for the different data types: string, number, boolean, date and time.

A string literal is a character expression delimited by single quotes.

Literal numbers are expressed in BNF as follows:

```
<unsigned numeric literal> ::=
        <exact numeric literal> | <approximate numeric literal>
<exact numeric literal> ::=
        <unsigned integer> [<period> [<unsigned integer>]]
        | <period><unsigned integer>
<unsigned integer> ::= <digit>
<approximate numeric literal> ::= <mantissa> E <exponent>
<mantissa> ::= <exact numeric literal>
<exponent> ::= <signed integer>
<signed integer> ::= [<sign>] <unsigned integer>
<sign> ::= <plus sign> | <minus sign>
```

A boolean literal expression is either TRUE or FALSE.

The BNF for date and time literals constructs are:

```
<date string> ::= <quote> <date value> <quote>
<date value> ::= <years value> <minus sign> <months value>
        <minus sign>  <days value>
<years value> ::= <datetime value>
<months value> ::= <date time value>
<days value> ::= <date time value>
<datetime value> ::= <unsigned integer>

<time string> ::= <quote> <time value> [<time zone interval>] <quote>
<time value> ::= <hours value> <colon> <minutes value> <colon>
        <seconds value>
<hours value> ::= <datetime value>
<minutes value> ::= <datetime value>
```

```
<seconds value> ::= <seconds integer value> [ <period> [<seconds
        fraction>]]
<seconds integer value> ::= <unsigned integer>
<seconds fraction> ::= <unsigned integer>
<time zone interval> ::= <sign> <hours value> <colon> <minutes value>

<timestamp string> ::= <quote> <date value> <space> <time value>
        [ <time zone interval> ] <quote>
```

2.2 Query syntax

The compact syntax for the SELECT statement shows the main constructs for the query specification:

```
SELECT [ ALL |  DISTINCT ]
        [ TOP <unsigned integer> ]
        { * | COUNT(*) | { <expression> [ AS <name> ] } [,…]  }
        [ INTO <target specification> ]
        FROM <table reference> [ <alias> ] [,…]
        [ { [NATURAL] { { LEFT |  RIGHT }  [OUTER] } | INNER | FULL
            [OUTER] } JOIN <alias> { ON  < search condition> |
            USING(col1, col2,…) } ]
        [ WHERE <search condition> ]
        [ GROUP BY <column> [,…] ]
        [ HAVING <search condition>  ]
        [ ORDER BY { <column> | <unsigned integer> } [ ASC | ESC ] [,…]]
```

The SELECT statement defines a query to some derived table(s) specified in the FROM clause. As a result of this query, a subset of the table(s) is returned. The order of the rows MAY be arbitrary unless ORDER BY clause is specified. The order of columns to return SHOULD be the same as the order as specified in the selection list, or the order defined in the original table if asterisk is specified.

TOP n construct is used to return the first n-rows.

The selection list MAY include any numeric, string and datetime value expression.

In the following sections some constructs requiring further description are presented.

2.2.1 Subqueries and joins

A subquery can contain a simple query description or a join statement. Among the different types of joins ADQL supports qualified ones only. These are INNER and OUTER ones (LEFT, RIGHT and FULL). All of these can be NATURAL or not. The join condition does not support embedded sub joins.

2.2.2 Search condition

The search condition can be part of several other clauses: JOIN, HAVING and, obviously, WHERE. Standard logical operators are present in its description (AND, OR and NOT). Six different types of predicates are present in which different types of reserved keywords or characters are used:

- Standard comparison operators: =, !=, <, >, <=, >=
- BETWEEN
- LIKE
- NULL
- EXISTS

[The full syntax for the SELECT statement is present in Appendix A.]

2.3 Functions

ADQL declares a list of non reserved keywords (section 2.1.2) which defines a set of special functions to enhance the astronomical usage of the language. These can be split into three different types:

2.3.1 Mathematical Functions

The next table shows the description of the mathematical functions.

Name	Return type	Comment
acos(x)	double	Basic function. Inverse cosine.
asin(x)	double	Basic function. Inverse sine.
atan(x)	double	Basic function. Inverse tangent.
atan2(x,y)	double	Basic function. Inverse tangent of x/y.
cos(x)	double	Basic function. Cosine.
sin(x)	double	Basic function. Sine.
tan(x)	double	Basic function. Tangent.
abs(x)	double	Basic function. Absolute value.
ceiling(x)	double	Basic function. Smallest integer not less than argument.
degrees(x)	double	Basic function. Radians to degrees.

exp(x)	double	Basic function. Exponential.
floor(x)	double	Basic function. Larger integer not greater than argument.
log(x)	double	Basic function. Natural logarithm.
log10(x)	double	Basic function. Base 10 logarithm.
mod(x, y)	double	Basic function. Remainder of y/x.
pi()	double	Basic function. Pi constant.
power(x, y)	double	Basic function. X raised to the power of Y.
radians(x)	double	Basic function. Degree to radians.
sqrt(x)	double	Basic function. Square root.
rand()	double	Basic function. Random value between 0.0 and 1.0.
round(x, n)	double	Basic function. Round to nearest integer.
truncate(x, n)	double	Basic function. Truncate to n decimal places.

2.3.2 Region

To be discussed.

2.3.3 User Defined Functions

To be discussed.

Appendix A: BNF Grammar

```
--
<ADQL language character> ::=
                <simple Latin letter>
        |       <digit>
        |       <ADQL special character>

<simple Latin letter> ::=
                <simple Latin upper case letter>
        |       <simple Latin lower case letter>

<simple Latin upper case letter> ::=
        A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

<simple Latin lower case letter> ::=
        a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<digit> ::=
        0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<ADQL special character> ::=
                <space>
        |       <double quote>
        |       <percent>
        |       <quote>
        |       <left paren>
        |       <right paren>
        |       <asterisk>
        |       <plus sign>
        |       <comma>
        |       <minus sign>
        |       <period>
        |       <solidus>
        |       <colon>
        |       <less than operator>
        |       <greater than operator>
        |       <equals operator>
        |       <underscore>
<space> ::= !! space character in character set in use

<double quote> ::= "

<percent> ::= %

<quote> ::= '

<left paren> ::= (

<right paren> ::= )

<asterisk> ::= *
```

<plus sign> ::= +

<comma> ::= ,

<minus sign> ::= -

<period> ::= .

<solidus> ::= /

<colon> ::= :

<less than operator> ::= <

<equals operator> ::= =

<greater than operator> ::= >

<underscore> ::= _

--
<token> ::=
<nondelimiter token>
| <delimiter token>

<nondelimiter token> ::=
<regular identifier>
| <keyword>
| <unsigned numeric literal>
| <bit string literal>
| <hex string literal>

<regular identifier> ::= <identifier body>

<identifier body> ::= <identifier start> [{ <underscore> | <identifier part> } ...]

<identifier start> ::= <simple Latin letter>

<identifier part> ::= <identifier start> | <digit>

<keyword> ::= <reserved word> | <non-reserved word>

<reserved word> ::=
ALL | AND | AS | ASC | AVG | BETWEEN | BY | COUNT | DESC | DISTINCT | EXISTS
| FALSE | FROM | FULL | GROUP | HAVING | IN | INNER | INTO | IS | JOIN | LEFT |
LIKE | MAX | MIN | NATURAL | NOT | NULL | ON | OR | ORDER | OUTER | RIGHT |
SELECT | SUM | TOP | TRUE | USING | WHERE

<non-reserved word> ::=
ABS | ACOS | ASIN | ATAN | ATAN2 | CEILING | COS | DEGREES | EXP | FLOOR |
LOG | LOG10 | MODE | PI | POWER | RADIANS | REGION | RAND | ROUND | SIN |
SQRT | UDF | TAN | TRUNCATE

<unsigned numeric literal> ::=
<exact numeric literal>

```
        |       <approximate numeric literal>

<exact numeric literal> ::=
                <unsigned integer> [ <period> [ <unsigned integer> ] ]
        |       <period> <unsigned integer>

<unsigned integer> ::= <digit> ...

<approximate numeric literal> ::=
                <mantissa> E <exponent>

<mantissa> ::= <exact numeric literal>

<exponent> ::= <signed integer>

<signed integer> ::= [ <sign> ] <unsigned integer>

<sign> ::= <plus sign> | <minus sign>

<character representation> ::= <nonquote character> | <quote symbol>

<nonquote character> ::= !! See the Syntax rules

<quote symbol> ::= <quote> <quote>

<separator> ::= { <comment> | <space> | <newline> }...

<comment> ::= <comment introducer> [ <comment character>... ] <newline>

<comment introducer> ::= <minus sign><minus sign> [<minus sign>...]

<comment character> ::= <nonquote character> | <quote>

<newline> ::= !! implementation defined end of line indicator

<bit string literal> ::=
                B <quote> [ <bit> ... ] <quote> [ { <separator>... <quote> [ <bit>... ] <quote> }... ]

<bit> ::= 0 | 1

<hex string literal> ::=
                X <quote> [ <hexit> ... ] <quote> [ { <separator>... <quote> [ <hexit>... ]
<quote> }... ]

<hexit> ::= <digit> | A | B | C | D | E | F | a | b | c | d | e | f
--
<delimiter token> ::=
                <character string literal>
        |       <boolean literal>
        |       <date string>
        |       <time string>
        |       <timestamp string>
        |       <ADQL special character>
        |       <not equals operator>
        |       <greater than or equals operator>
        |       <less than or equals operator>
```

| <concatenation operator>
| <double period>
| <left bracket>
| <right bracket>

<character string literal> ::= <quote> [<character representation>...] <quote> [{ <separator>... <quote> [<character representation>...] <quote> }...]

<boolean literal> ::= TRUE | FALSE

<date string> ::= <quote> <date value> <quote>

<date value> ::= <years value> <minus sign> <months value> <minus sign> <days value>

<years value> ::= <datetime value>

<datetime value> ::= <unsigned integer>

<months value> ::= <datetime value>

<days value> ::= <datetime value>

<time string> ::= <quote> <time value> [<time zone interval>] <quote>

<time value> ::= <hours value> <colon> <minutes value> <colon> <seconds value>

<hours value> ::= <datetime value>

<minutes value> ::= <datetime value>

<seconds value> ::= <seconds integer value> [<period> [<seconds fraction>]]

<seconds integer value> ::= <unsigned integer>

<seconds fraction> ::= <unsigned integer>

<time zone interval> ::= <sign> <hours value> <colon> <minutes value>

<timestamp string> ::= <quote> <date value> <space> <time value> [<time zone interval>] <quote>

<not equals operator> ::= <>

<greater than or equals operator> ::= >=

<less than or equals operator> ::= <=

<concatenation operator> ::= ||

<double period> ::= ..

<left bracket> ::= [

<right bracket> ::=]

--

<search condition> ::=
<boolean term> | <search condition> OR <boolean term>

<boolean term> ::=
<boolean factor> | <boolean term> AND <boolean factor>

<boolean factor> ::= [NOT] <boolean test>

<boolean test> ::= <boolean primary> [IS [NOT] <truth value>]

<boolean primary> ::= <predicate> | <left paren> <search condition> <right paren>

<predicate> ::=
<comparison predicate>
| <between predicate>
| <in predicate>
| <like predicate>
| <null predicate>
| <exists predicate>

<comparison predicate> ::= <row value constructor> <comp op> <row value constructor>

<row value constructor> ::=

<row value constructor element>
| <left paren> <row value constructor list> <right paren>
| <row subquery>

<row value constructor element> ::=
<value expression>
| <null specification>

<value expression> ::=
<numeric value expression>
| <string value expression>
| <datetime value expression>
| <interval value expression>

<numeric value expression> ::=

<term>
| <numeric value expression> <plus sign> <term>
| <numeric value expression> <minus sign> <term>

<term> ::=
<factor>
| <term> <asterisk> <factor>
| <term> <solidus> <factor>

<factor> ::= [<sign>] <numeric primary>

<numeric primary> ::= <value expression primary> | <numeric value function>

<value expression primary> ::=
<unsigned value specification>
| <column reference>
| <set function specification>
| <scalar subquery>

```
        |   <left paren> <value expression> <right paren>

<unsigned value specification> ::= <unsigned literal>
<unsigned literal> ::= <unsigned numeric literal>

<column reference> ::= [ <qualifier> <period> ] <column name>

<qualifier> ::= <table name> | <correlation name>

<correlation name> ::= <identifier>

<set function specification> ::=
        COUNT <left paren> <asterisk> <right paren>
        |   <general set function>

<general set function> ::=
        <set function type> <left paren> [ <set quantifier> ] <value expression> <right paren>

<set function type> ::= AVG | MAX | MIN | SUM | COUNT

<set quantifier> ::= DISTINCT | ALL

--
<scalar subquery> ::= <subquery>

<subquery> ::= <left paren> <query expression> <right paren>

<query expression> ::= <non-join query expression> | <joined table>

<non-join query expression> ::=
        <non-join query term>


<non-join query term> ::=
        <non-join query primary>

<non-join query primary> ::= <simple table> | <left paren> <non-join query expression> <right
paren>

<simple table> ::=
        <query specification>

<query specification> ::=
        SELECT [ <set quantifier> ] <select list> <table expression>

<select list> ::=
        <asterisk>
        | <select sublist> [ { <comma> <select sublist> }... ]

<select sublist> ::= <derived column> | <qualifier> <period> <asterisk>

<derived column> ::= <value expression> [ <as clause> ]

<as clause> ::= [ AS ] <column name>

<table expression> ::=
```

<from clause>
[<where clause>]
[<group by clause>]
[<having clause>]
[<order by clause>]

<from clause> ::= FROM <table reference> [{ <comma> <table reference> }...]

<table reference> ::=
<table name> [<correlation specification>]
| <derived table> <correlation specification>
| <joined table>

<correlation specification> ::=
[AS] <correlation name> [<left paren> <derived column list> <right paren>]

<derived column list> ::= <column name list>

<derived table> ::= <table subquery>

<table subquery> ::= <subquery>

<joined table> ::=
<cross join>
| <qualified join>
| <left paren> <joined table> <right paren>

<cross join> ::=
<table reference> CROSS JOIN <table reference>

<qualified join> ::=
<table reference> [NATURAL] [<join type>] JOIN <table reference> [<join specification>]

<join type> ::=
INNER
<outer join type> [OUTER]

<outer join type> ::= LEFT | RIGHT | FULL

<join specification> ::= <join condition> | <named columns join>

<join condition> ::= ON <search condition>

<named columns join> ::= USING <left paren> <join column list> <right paren>

<join column list> ::= <column name list>

<where clause> ::= WHERE <search condition>

<group by clause> ::= GROUP BY <grouping column reference list>

<grouping column reference list> ::=
<grouping column reference> [{ <comma> <grouping column reference> }...]

<grouping column reference> ::= <column reference>

<having clause> ::= HAVING <search condition>

<query term> ::= <non-join query term> | <joined table>

<query primary> ::= <non-join query primary> | <joined table>

--
<character value expression> ::= <concatenation> | <character factor>


ıcatenation> ::= <character value expression> <concatenation operator> <character factor>

<character factor> ::= <character primary>

<character primary> ::= <value expression primary> |

<datetime value expression> ::=

<datetime term>
| <interval value expression> <plus sign> <datetime term>
| <datetime value expression> <plus sign> <interval term>
| <datetime value expression> <minus sign> <interval term>

<interval term> ::=

<interval factor>
| <interval term 2> <asterisk> <factor>

| <interval term 2> <solidus> <factor>
| <term> <asterisk> <interval factor>

<interval factor> ::= [<sign>] <interval primary>

<interval primary> ::= <value expression primary> [<interval qualifier>]

<interval term 2> ::= <interval term>

<interval value expression> ::=
<interval term>
| <interval value expression 1> <plus sign> <interval term 1>
| <interval value expression 1> <minus sign> <interval term 1>
| <left paren> <datetime value expression> <minus sign> <datetime term> <right paren>
<interval qualifier>

<interval value expression 1> ::= <interval value expression>

<interval term 1> ::= <interval term>

<datetime term> ::= <datetime factor>

<datetime factor> ::= <datetime primary>

<datetime primary> ::= <value expression primary> | <datetime value function>

<string value expression> ::= <character value expression>

<null specification> ::= NULL

<row value constructor list> ::= <row value constructor element> [{ <comma> <row value constructor element> } ...]

<row subquery> ::= <subquery>

<comp op> ::=
<equals operator>
| <not equals operator>
| <less than operator>
| <greater than operator>
| <less than or equals operator>
| <greater than or equals operator>

<between predicate> ::=
<row value constructor> [NOT] BETWEEN <row value constructor> AND <row value constructor>

<in predicate> ::= <row value constructor> [NOT] IN <in predicate value>

<in predicate value> ::= <table subquery> | <left paren> <in value list> <right paren>

<in value list> ::= <value expression> { <comma> <value expression> } ...

<like predicate> ::= <match value> [NOT] LIKE <pattern>

<match value> ::= <character value expression>

<pattern> ::= <character value expression>

<escape character> ::= <character value expression>

<null predicate> ::= IS [NOT] NULL

<exists predicate> ::= EXISTS <table subquery>

<row value constructor 1> ::= <row value constructor>

<row value constructor 2> ::= <row value constructor>

<truth value> ::= TRUE | FALSE

<order by clause> ::= ORDER BY <sort specification list>

<sort specification list> ::= <sort specification> [{ <comma> <sort specification> }...]

<sort specification> ::= <sort key> [<ordering specification>]

<sort key> ::= <column name> | <unsigned integer>

<ordering specification> ::= ASC | DESC

References

[1] BNF Grammar for ISO/IEC 9075:1992 – Database Language SQL (SQL-92)
http://savage.net.au/SQL/sql-92.bnf.html
[2] BNF for SQL92
http://www.sqlzoo.net/sql92.html