International

Virtual

Observatory

Alliance

IVOA Astronomical Data Query Language

Version 1.5

IVOA Working Draft 2007 September 3

This version:
1.5-20070903
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, Jeff Lusted, Alexander Szalay, Yuji Shirasaki, Maria A. Nieto-Santisteban, Masatoshi Ohishi, William O'Mullane, 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), especially 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 (:)
- semicolon (;)
- less than operator (<)
- equals operator (=)
- greater than operator (>)
- underscore (_)
- ampersand (&)
- question mark (?)
- vertical bar (|)

2.1.2 Keywords and Identifiers

Besides the character set, the language provides a list of 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. These keywords must be enforced and should be extensive as an escaping mechanism is already in place. We can extend the list of SQL92 reserved keywords to accommodate those useful for astronomical purposes and/or present in a subset of vendor specific languages only (e.g. TOP). This leads to the following list:

- SQL reserved keywords:

ABSOLUTE, ACTION, ADD, ALL, ALLOCATE, ALTER, AND, ANY, ARE,
AS, ASC, ASSERTION, AT, AUTHORIZATION, AVG, BEGIN, BETWEEN, BIT,
BIT_LENGTH, BOTH, BY, CASCADE, CASCADED, CASE, CAST, CATALOG,
CHAR, CHARACTER, CHARACTER_LENGTH, CHAR_LENGTH, CHECK,
CLOSE, COALESCE, COLLATE, COLLATION, COLUMN, COMMIT,
CONNECT, CONNECTION, CONSTRAINT, CONSTRAINTS, CONTINUE,
CONVERT, CORRESPONDING, COUNT, CREATE, CROSS, CURRENT,
CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP,
CURRENT_USER, CURSOR, DATE, DAY, DEALLOCATE, DECIMAL,
DECLARE, DEFAULT, DEFERRABLE, DEFERRED, DELETE, DESC,
DESCRIBE, DESCRIPTOR, DIAGNOSTICS, DISCONNECT, DISTINCT,

DOMAIN, DOUBLE, DROP, ELSE, END, END-EXEC, ESCAPE, EXCEPT, EXCEPTION, EXEC, EXECUTE, EXISTS, EXTERNAL, EXTRACT, FALSE, FETCH, FIRST, FLOAT, FOR, FOREIGN, FOUND, FROM, FULL, GET, GLOBAL, GO, GOTO, GRANT, GROUP, HAVING, HOUR, IDENTITY, IMMEDIATE, IN, INDICATOR, INITIALLY, INNER, INPUT, INSENSITIVE, INSERT, INT, INTEGER, INTERSECT, INTERVAL, INTO, IS, ISOLATION, JOIN, KEY, LANGUAGE, LAST, LEADING, LEFT, LEVEL, LIKE, LOCAL, LOWER, MATCH, MAX, MIN, MINUTE, MODULE, MONTH, NAMES, NATIONAL, NATURAL, NCHAR, NEXT, NO, NOT, NULL, NULLIF, NUMERIC, OCTET_LENGTH, OF, ON, ONLY, OPEN, OPTION, OR, ORDER, OUTER, OUTPUT, OVERLAPS, PAD, PARTIAL, POSITION, PRECISION, PREPARE, PRESERVE, PRIMARY, PRIOR, PRIVILEGES, PROCEDURE, PUBLIC, READ, REAL, REFERENCES, RELATIVE, RESTRICT, REVOKE, RIGHT, ROLLBACK, ROWS, SCHEMA, SCROLL, SECOND, SECTION, SELECT, SESSION, SESSION_USER, SET, SIZE, SMALLINT, SOME, SPACE, SQL, SQLCODE, SQLERROR, SQLSTATE, SUBSTRING, SUM, SYSTEM_USER, TABLE, TEMPORARY, THEN, TIME, TIMESTAMP, TIMEZONE_HOUR, TIMEZONE_MINUTE, TO, TRAILING, TRANSACTION, TRANSLATE, TRANSLATION, TRIM, TRUE, UNION, UNIQUE, UNKNOWN, UPDATE, UPPER, USAGE, USER, USING, VALUE, VALUES, VARCHAR, VARYING, VIEW, WHEN, WHENEVER, WHERE, WITH, WORK, WRITE, YEAR, ZONE

- ADQL reserved keywords:

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

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} as follows:

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

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 and number.

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>
```

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 | DESC ]
            [,…]]
```

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 the columns to return SHOULD be the same as the order 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 value expression.

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

2.2.1 Table subqueries and Joins

Table subqueries are present and can be used by some existing predicates within the search condition (IN and BETWEEN most likely) or as an artifact of building derived tables..

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(x)	double	Basic function. Random value between 0.0 and 1.0. It can take a seed value.
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>
  | <SQL_special_character>

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

<SQL_embedded_language_character> ::=
    <left_bracket>
  | <right_bracket>

<SQL_reserved_word> ::=
    ABSOLUTE | ACTION | ADD | ALL | ALLOCATE | ALTER | AND | ANY | ARE
  | AS | ASC | ASSERTION | AT | AUTHORIZATION | AVG
  | BEGIN | BETWEEN | BIT | BIT_LENGTH | BOTH | BY
  | CASCADE | CASCADED | CASE | CAST | CATALOG | CHAR | CHARACTER
  | CHARACTER_LENGTH
  | CHAR_LENGTH | CHECK | CLOSE | COALESCE | COLLATE | COLLATION
  | COLUMN | COMMIT
  | CONNECT | CONNECTION | CONSTRAINT | CONSTRAINTS | CONTINUE
  | CONVERT | CORRESPONDING | COUNT
  | CREATE | CROSS | CURRENT | CURRENT_DATE | CURRENT_TIME
  | CURRENT_TIMESTAMP
  | CURRENT_USER | CURSOR
  | DATE | DAY | DEALLOCATE | DECIMAL | DECLARE | DEFAULT
  | DEFERRABLE | DEFERRED | DELETE | DESC | DESCRIBE | DESCRIPTOR
  | DIAGNOSTICS
  | DISCONNECT | DISTINCT | DOMAIN | DOUBLE | DROP
  | ELSE | END | END-EXEC | ESCAPE | EXCEPT | EXCEPTION | EXEC
  | EXECUTE
  | EXISTS | EXTERNAL | EXTRACT
  | FALSE | FETCH | FIRST | FLOAT | FOR | FOREIGN | FOUND | FROM | FULL
  | GET | GLOBAL | GO | GOTO | GRANT | GROUP
  | HAVING | HOUR
  | IDENTITY | IMMEDIATE | IN | INDICATOR | INITIALLY | INNER | INPUT
  | INSENSITIVE
  | INSERT | INT | INTEGER | INTERSECT | INTERVAL | INTO | IS
  | ISOLATION
  | JOIN
  | KEY
  | LANGUAGE | LAST | LEADING | LEFT | LEVEL | LIKE | LOCAL | LOWER
  | MATCH | MAX | MIN | MINUTE | MODULE | MONTH
  | NAMES | NATIONAL | NATURAL | NCHAR | NEXT | NO | NOT | NULL
  | NULLIF | NUMERIC
  | OCTET_LENGTH | OF | ON | ONLY | OPEN | OPTION | OR | ORDER | OUTER
  | OUTPUT | OVERLAPS
  | PAD | PARTIAL | POSITION | PRECISION | PREPARE | PRESERVE | PRIMARY
  | PRIOR | PRIVILEGES | PROCEDURE | PUBLIC
  | READ | REAL | REFERENCES | RELATIVE | RESTRICT | REVOKE | RIGHT
  | ROLLBACK | ROWS
  | SCHEMA | SCROLL | SECOND | SECTION | SELECT | SESSION
  | SESSION_USER | SET
  | SIZE | SMALLINT | SOME | SPACE | SQL | SQLCODE | SQLERROR
  | SQLSTATE | SUBSTRING | SUM | SYSTEM_USER
  | TABLE | TEMPORARY | THEN | TIME | TIMESTAMP | TIMEZONE_HOUR
  | TIMEZONE_MINUTE
  | TO | TRAILING | TRANSACTION | TRANSLATE | TRANSLATION | TRIM | TRUE
  | UNION | UNIQUE | UNKNOWN | UPDATE | UPPER | USAGE | USER | USING
  | VALUE | VALUES | VARCHAR | VARYING | VIEW
  | WHEN | WHENEVER | WHERE | WITH | WORK | WRITE
  | YEAR
  | ZONE

<SQL_special_character> ::=
    <ampersand>
  | <asterisk>
  | <colon>
  | <comma>
  | <double_quote>
  | <equals_operator>
  | <greater_than_operator>
  | <left_paren>
  | <less_than_operator>
  | <minus_sign>
  | <percent>
  | <period>
  | <plus_sign>
  | <question_mark>
  | <quote>
  | <right_paren>
  |  <semicolon>
  | <solidus>
  |  <space>
  | <underscore>
  | <vertical_bar>


<ampersand> ::= &

<approximate_numeric_literal> ::= <mantissa> E <exponent>

<as_clause> ::= [ AS ] <column_name>

<asterisk> ::= *

<between_predicate> ::=
   <value_expression> [ NOT ] BETWEEN <value_expression> AND <value_expression>

<boolean_factor> ::= [ NOT ] <boolean_primary>

<boolean_primary> ::=
   <predicate> | <left_paren> <search_condition> <right_paren>

<boolean_term> ::= <boolean_factor> | <boolean_term> AND <boolean_factor>

<catalog_name> ::= <identifier>

<character_factor> ::= <character_primary>

<character_primary> ::= <value_expression_primary> | <user_defined_function>

<character_representation> ::= <nonquote_character> | <quote_symbol>

<character_string_literal> ::=
   <quote> [ <character_representation>... ] <quote>
   [ {<separator>... <quote> [ <character_representation>...    ]<quote>}... ]

<character_value_expression> ::= <concatenation> | <character_factor>

<colon> ::= :

<column_name> ::= <identifier>

<column_name_list> ::= <column_name> [ { <comma> <column_name> }... ]

<column_reference> ::= [ <qualifier> <period> ] <column_name>

<comma> ::= ,

<comment> ::= <comment_introducer> [ <comment_character>... ] <newline>

<comment_character> ::= <nonquote_character> | <quote>

<comment_introducer> ::= <minus_sign><minus_sign> [<minus_sign>...]

<comp_operator> ::=
    <equals_operator>
  | <not_equals_operator>
  | <less_than_operator>
  | <greater_than_operator>
  | <less_than_or_equals_operator>
  | <greater_than_or_equals_operator>

<comparison_predicate> ::= <value_expression> <comp_operator> <value_expression>

<concatenation> ::=
    <character_value_expression> <concatenation_operator>  <character_factor>

<concatenation_operator> ::= ||

<correlation_name> ::= <identifier>

<correlation_specification> ::= [ AS ] <correlation_name>

<default_function_prefix> ::=
!! The prefix is set by default to “udf_.”
!! It should be possible to change the default prefix to accommodate local preferences

<delimited_identifier> ::= <double_quote> <delimited_identifier_body> <double_quote>

<delimited_identifier_body> ::= <delimited_identifier_part>…

<delimited_identifier_part> ::= <nondoublequote_character> | <doublequote_symbol>

<delimiter_token> ::=
    <character_string_literal>
  | <delimited_identifier>
  | <ADQL_special_character>
  | <SQL_special_character>
  | <not_equals_operator>
  | <greater_than_or_equals_operator>
  | <less_than_or_equals_operator>
  | <concatenation_operator>
  | <double_period>
  | <left_bracket>
  | <right_bracket>

<derived_column> ::= <value_expression> [ <as_clause> ]

<derived_table> ::= <table_subquery>

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

<double_period> ::= ..

<double_quote> ::= "

<double_quote_symbol> ::= <double_quote><double_quote>

<equals_operator> ::= =

<exact_numeric_literal> ::=
    <unsigned_integer> [ <period> [ <unsigned_integer> ] ]
  | <period> <unsigned_integer>

<exists_predicate> ::= EXISTS <table_subquery>

<exponent> ::= <signed_integer>

<factor> ::= [ <sign> ] <numeric_primary>

<from_clause> ::= FROM <table_reference> [ { <comma> <table_reference> }... ]

<general_literal> ::= <character_string_literal>

<general_set_function> ::=
    <set_function_type> <left_paren> [ <set_quantifier> ] <value_expression> <right_paren>

<greater_than_operator> ::= >

<greater_than_or_equals_operator> ::= >=

<group_by_clause> ::= GROUP BY <grouping_column_reference_list>

<grouping_column_reference> ::= <column_reference>

<grouping_column_reference_list> ::=
    <grouping_column_reference> [ { <comma> <grouping_column_reference>  }... ]

<having_clause> ::= HAVING <search_condition>

<identifier> ::= <regular_identifier> | <delimited_identifier>

<in_predicate> ::= <value_expression> [ 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> } ...

<join_column_list> ::= <column_name_list>

<join_condition> ::= ON <search_condition>

<join_specification> ::= <join_condition> | <named_columns_join>

<join_type> ::=
    INNER
  | <outer_join_type> [ OUTER ]

<joined_table> ::=
    <qualified_join>
  | <left_paren> <joined_table> <right_paren>

<keyword> ::= <SQL_reserved_word> | <ADQL_reserved_word>

<left_bracket> ::= [
<left_paren> ::= (

<less_than_operator> ::= <

<less_than_or_equals_operator> ::= <=

<like_predicate> ::=
    <match_value> [ NOT ] LIKE   <pattern>

<mantissa> ::= <exact_numeric_literal>

<match_value> ::= <character_value_expression>

<math_function> ::=
    ABS <left_paren> <numeric_value_expression> <right_paren>
  | CEILING <left_paren> <numeric_value_expression> <right_paren>
  | DEGREES <left_paren> <numeric_value_expression> <right_paren>
  | EXP <left_paren> <numeric_value_expression> <right_paren>
  | FLOOR <left_paren> <numeric_value_expression> <right_paren>
  | LOG <left_paren> <numeric_value_expression> <right_paren>
  | PI <left_paren><right_paren>
  | POWER <left_paren> <numeric_value_expression> <comma> <unsigned_integer>
<right_paren>
  | RADIANS <left_paren> <numeric_value_expression> <right_paren>
  | SQUARE <left_paren> <numeric_value_expression> <right_paren>
  | SQRT <left_paren> <numeric_value_expression> <right_paren>
  | LOG10 <left_paren> <numeric_value_expression> <right_paren>
  | RAND <left_paren> [ <numeric_value_expression> ] <right_paren>
  | ROUND <left_paren> <numeric_value_expression> <right_paren>
  | TRUNCATE <left_paren> <numeric_value_expression> <right_paren>

<minus_sign> ::= -

<named_columns_join> ::= USING <left_paren> <join_column_list> <right_paren>

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

<nondelimiter_token> ::=
    <regular_identifier>
  | <keyword>
  | <unsigned_numeric_literal>

<nondoublequote_character> ::= !! See syntax rules

<nonquote_character> ::= !! One ASCII character

<not_equals_operator> ::= <not_equals_operator1>  |  <not_equals_operator2>

<not_equals_operator1> ::= <>

<not_equals_operator2> ::= !=

<null_predicate> ::= <column_reference> IS [ NOT ] NULL

<numeric_primary> ::= <value_expression_primary> | <numeric_value_function>
```

```
<numeric_value_expression> ::=
    <term>
  | <numeric_value_expression> <plus_sign> <term>
  | <numeric_value_expression> <minus_sign> <term>

<numeric_value_function> ::=
    <trigonometric_function>
  | <math_function>
  | <user_defined_function>

<order_by_clause> ::= ORDER BY <sort_specification_list>

<ordering_specification> ::= ASC | DESC

<outer_join_type> ::= LEFT | RIGHT | FULL

<pattern> ::= <character_value_expression>

<percent> ::= %

<period> ::= .

<plus_sign> ::= +

<predicate> ::=
    <comparison_predicate>
  | <between_predicate>
  | <in_predicate>
  | <like_predicate>
  | <null_predicate>
  | <exists_predicate>
  | <region_predicate>

<qualified_join> ::=
    <table_reference> [ NATURAL ] [ <join_type> ] JOIN <table_reference> [ <join_specification> ]

<qualifier> ::= <table_name> | <correlation_name>

<query_expression> ::=
    <query_specification>
  | <joined_table>

<query_specification> ::=
    SELECT [ <set_quantifier> ] [ <set_limit> ] <select_list> <table_expression>

<question_mark> ::= ?

<quote> ::= '

<quote_symbol> ::= <quote> <quote>

<region_predicate> ::= !! To be defined

<region_specification> ::= !! Region definition from STC schema here
```

```
<regular_identifier> ::=
    <simple_Latin_letter>... [ { <underscore> | <simple_Latin_letter> | <digit> } ... ]

<right_bracket> ::= ]

<right_paren> ::= )

<schema_name> ::= [ <catalog_name> <period> ] <unqualified_schema_name>

<search_condition> ::=
    <boolean_term>
  | <search_condition> OR <boolean_term>

<select_list> ::=
    <asterisk>
  | <select_sublist> [ { <comma> <select_sublist> }... ]

<select_sublist> ::= <derived_column> | <qualifier> <period> <asterisk>

<semicolon> ::= ;

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

<set_function_specification> ::=
    COUNT <left_paren> <asterisk> <right_paren>
  | <general_set_function>

<set_function_type> ::= AVG | MAX | MIN | SUM | COUNT

<set_limit> ::= TOP <unsigned_integer>

<set_quantifier> ::= DISTINCT | ALL

<sign> ::= <plus_sign> | <minus_sign>

<signed_integer> ::= [ <sign> ] <unsigned_integer>

<simple_Latin_letter> ::=
    <simple_Latin_upper_case_letter>
  | <simple_Latin_lower_case_letter>

<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

<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

<solidus> ::= /

<sort_key> ::= <column_name> | <unsigned_integer>

<sort_specification> ::= <sort_key> [ <ordering_specification> ]

<sort_specification_list> ::= <sort_specification> [ { <comma> <sort_specification> }... ]
```

```
<space> ::= !! space character here

<string_value_expression> ::= <character_value_expression>

<subquery> ::=
    <left_paren> <query_expression> <right_paren>

<table_expression> ::=
    <from_clause>
    [ <where_clause> ]
    [ <group_by_clause> ]
    [ <having_clause> ]
    [ <order_by_clause> ]

<table_name> ::= [ <schema_name> <period> ] <identifier>

<table_reference> ::=
    <table_name> [ <correlation_specification> ]
  | <derived_table> <correlation specification>
  | <joined_table>

<table_subquery> ::=  <subquery>

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

<token> ::=
    <nondelimiter_token>
  | <delimiter_token>

<trigonometric_function> ::=
    ACOS <left_paren> <numeric_value_expression> <right_paren>
  | ASIN <left_paren> <numeric_value_expression> <right_paren>
  | ATAN <left_paren> <numeric_value_expression> <right_paren>
  | ATAN2 <left_paren> <numeric_value_expression> <comma> <numeric_value_expression>
<right_paren>
  | COS <left_paren> <numeric_value_expression> <right_paren>
  | COT <left_paren> <numeric_value_expression> <right_paren>
  | SIN <left_paren> <numeric_value_expression> <right_paren>
  | TAN <left_paren> <numeric_value_expression> <right_paren>

<underscore> ::= _

<unqualified_schema_name> ::= <identifier>

<unsigned_integer> ::= <digit> ...

<unsigned_literal> ::= <unsigned_numeric_literal>  | <general_literal>

<unsigned_numeric literal> ::=
    <exact_numeric_literal>
  | <approximate_numeric_literal>
```

```
<unsigned_value_specification> ::= <unsigned_literal>

<user_defined_function> ::=
    <user_defined_function_name>
      <left_paren>
        [ <numeric_value_expression> [  { <comma> <numeric_value_expression> } …]  ]
     <right_paren>

<user_defined_function_name> ::=
    [ <default_function_prefix> ] <regular_identifier>

<value_expression> ::=
    <numeric_value_expression>
  | <string_value_expression>

<value_expression_primary> ::=
    <unsigned_value_specification>
  | <column_reference>
  | <set_function_specification>
  | <left_paren> <value_expression> <right_paren>

<vertical_bar> ::= |

<where_clause> ::= WHERE <search_condition>
```

References

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