Appendix A ADQL Grammar (maria's version)

We have used the following conventions to describe the BNF representation of ADQL.

BOLD	font denotes ADQL keywords.
italic	font denotes terminal elements which do not require further specification.
< >	Angle brackets enclose non-terminal elements (also know as syntax rules identifiers).
:: =	Definition operator is used to separate the non-terminal element being defined (on the left) from its definition (on the right).
{ }	Brace brackets enclose required elements. (Do not type {}).
[ ]	Square brackets enclose optional elements. (Do not type [ ]).
[...n]	Preceding element may be repeated n times.
[, …n]	Preceding element may be repeated n times. Elements are separated by commas.
|	The vertical bar is used to separate syntactic elements. One of the elements must be chosen.
ADQL	is case insensitive.

A- 1 Core ADQL Syntax

A-1-1 Construction

```
<core_select_query> ::=
 {SELECT {[TOP n] <selection_list> |
          <core_aggregate_function>}
  FROM <core_table_source>}
  [WHERE <core_search_condition>]
```

- **TOP** specifies that only the first n rows of the result set are returned.

```
<core_aggregate_function> ::=
  {COUNT( {<table_alias>.* | <qualified_column>} )}
```

- **COUNT** returns the number of rows in the result set.

<core_table_source> ::=
{<table_name> [**AS**] <table_alias>}

- Only one table is allowed in CORE ADQL.
- Aliasing a table is compulsory

<selection_list> ::=
{<table_alias>.* | <column_list>}

- * denotes all columns.

<column_list> ::=
{<column>[,...n]}

<column> ::=
{<expression> | <qualified_column>} [[**AS**] <column_alias>]

<expression> ::=
{<expression_primary> | (<expression_primary>)}

<expression_primary> ::=
{<string_expression> | <numeric_expression>}

<string_expression> ::=
{*'string_value'* | <string_value_function>}

- Strings are delimited by single quotes (‘’)

<string_value_function>} ::= ????
(What functions should be defined at the Core ADQL level, if any?)

<numeric_expression> ::=
{*numeric_value* | <operand> <arithmetic_operator> <operand> |
<numeric_expression> <arithmetic_operator> <numeric_expression>}

<operand> ::=
{*numeric_value* | <qualified_column> | <numeric_expression>}

<qualified_column> ::=
{<table_alias>.<column_name>}

<arithmetic_operator> :: = { + | - | * | / }

<core_search_condition> ::=

{<spatial_condition> [**AND** <non_spatial_condition>] |

<non_spatial_condition> [**AND** <spatial_condition>]}

- Only one spatial condition is allowed in CORE ADQL.
- Only the **AND** operator is allowed in order to combine spatial and non-spatial conditions.

<spatial_condition> ::=

{**REGION** ('<spatial_predicate>')}

- Region predicates are enclosed between single quotes.

<spatial_predicate> ::=

{<box_constraint> | <circle_constraint>}

- Only BOX and CIRCLE constraints are allowed in CORE ADQL.

<box_constraint> ::=

{**BOX** ???????}

<circle_constraint>} ::=

{**CIRCLE** ??????}

<non_spatial_condition> ::=

{[**NOT**] <predicate> | (<non_spatial_condition>) }

[{ **AND** | **OR** } [**NOT**] { <predicate> |

(<non_spatial_condition>)}][,...n]

<predicate> ::=

{<expression> <comparison_operator> <expression> |

<qualified_column> [**NOT**] **LIKE** '*string_pattern*'} |

<between_predicate> | <in_predicate>}

- string_pattern is the sequence of characters to search for in the content of the column. The pattern can include wildcard characters as:
  - '%' to denote any string of zero or more characters.
  - '-' to denote any single character.

```
<comparison_operator> ::=
   { = | <> | != | > | >= | < | <= }

<between_predicate> ::=
  <value_expression> [NOT] BETWEEN <value_expression> AND
                                   <value_expression>
<in_predicate> ::=
  <value_expression> [NOT] IN (<value_expression>[,...n])

<value_expression> ::=
 {<numeric_expression> | <string_expression>}

<table_name>   ::= {identifier | "string_value"}
<table_alias>  ::= {identifier | "string_value"}
<column_name>  ::= {identifier | "string_value"}
<column_alias> ::= {identifier | "string_value"}
```

  - Identifiers follow common coding rules as an Identifier cannot start with a number nor can include special characters. Identifiers which don't follow this rule shall be delimited by double quotes.

- See data type section to see which operators are supported for each data type.
  - In my opinion trying to specify the data types in the BNF is an over killing.
- If a Boolean value expression that is not supported is specified, it should be evaluated as true rather than throwing an exception.
  - I don't understand the point of this bullet

A-1-2 Specification number

- **QL-C01 [Core]** All services SHALL implement the SELECT core syntax.
- **QL-C02 [Core]** All services SHALL support numeric, string data types.
- **QL-C03 [Core]** All services SHALL support count(*) aggregate function.

[These three requirements are already explicit in the grammar. It has already been said that all VO Services using ADQL SHALL implement CORE ADQL.]