Feedback on the implementation of ADQL-2.1

Grégory Mantelet[1]

[1]CDS (Centre de Données astronomiques de Strasbourg)

12 October 2019

Table of Contents

A. Implementation status

A.1. ADQL-Lib v2.0 - I

- VOLLT/ADQL-Lib **v2.0** : new major release under development
- implementation of ADQL-2.1
- Possible to test the parsing, checking and translation online:

  `http://cdsportal.u-strasbg.fr/adqltuto/validator.html`

A.1. ADQL-Lib v2.0 - II

ADQL Lib. | Home | Getting started | Documentation | Javadoc | UML | ADQL Validator | Download | What's new?

Query to validate

ADQL query:

```
1 Select main_id AS "Object name", ra || ' - ' || dec as "Position"
2 From basic
3 Where Contains(Point('ICRS', ra, dec), Circle('ICRS', 10, 5, 1)) = 1
4 Order By 1 ASC;
```

***(optional)* ADQL version:** 2.1beta *(PR-2018-01-12 ; partially implemented: missing UNION/INTERSECT/EXCEPT and CAST(...))*

***(optional)* Tables and columns:**

None | *OR* URL: http://simbad.u-strasbg.fr/simbad/sim-tap/tables | *OR* a file: Parcourir... Aucun fichier sélectionné.

***(optional)* ADQL to SQL translator to use:** Postgres+PgSphere *(adql.translator.PgSphereTranslator)*

Parse ADQL !

Parsing result

Correct ADQL !

SQL translation

The same query you have written but in a SQL executable by Postgres+PgSphere

```
1. SELECT "public"."basic"."main_id" AS "Object name" , "public"."basic"."ra" || ' - ' || "public"."basic"."dec" AS "Position"
2. FROM "public"."basic"
3. WHERE (spoint(radians("public"."basic"."ra"),radians("public"."basic"."dec")) @ scircle(spoint(radians(10),radians(5)),radians(1))) = '1'
4. ORDER BY 1 ASC
```

A.1. ADQL-Lib v2.0 - III

- Choice of the **ADQL version**:

```
// 1. CREATE A PARSER:
ADQLParser parser = new ADQLParser(ADQLVersion.V2_1);
// 2. PARSE AN ADQL QUERY:
ADQLQuery query = parser.parseQuery("SELECT ...");
// 3. Do whatever you want with the result:
...
```

- **API altered**: *adaptations will be needed on ADQL-Lib user side ; nothing should change for users of TAP-Lib (and UWS-Lib)*

A.2. Supported features

- Better/Generic support of **optional features**:

Feature	**Supported in the parser?**
LOWER, ILIKE	YES
OFFSET	YES
WITH	YES
~~*Bitwise operators*~~	~~*YES*~~, *but...*
IN_UNIT	YES *(DB impl. required)*
UNION, INTERSECT, EXCEPT	***Not yet***
CAST, TIMESTAMP, INTERVAL	***???***

A.2. Optional feat. vs DBMS

	Postgres	**PgSphere**	**MySQL**	**SQLServer**
Geometries	**NO**	YES	**NO**	**NO**
LOWER	YES	YES	YES	YES
ILIKE	YES	YES	**NO**	**NO**
OFFSET	YES	YES	YES	YES
WITH	YES	YES	YES	YES
~~*Bitwise operators*~~	~~*YES*~~	~~*YES*~~	~~*YES*~~	~~*YES*~~
IN_UNIT	**NO**	**NO**	**NO**	**NO**
UNION, . . .	*should*	*should*	*should*	*should*
CAST, . . .	*???*	*???*	*???*	*???*

B. Raised issues

B.1. REGION

`REGION` removed from the ADQL-2.1's document

- $=>$ no more backward compatibility and still very useful for some existing TAP services
- ***Should come back in ADQL-2.1***
  - the argument **may be an STC-S** (for backward compatibility),
  - but the syntax of the argument **should be the one defined in DALI**

B.2. COOSYS - I

COOSYS

1. **Possible ambiguity with** `POLYGON` and its # of arguments

   Example: *`POLYGON(ra, dec, 3, 4, 5, 6, 7)` is OK according to the parser because `ra`, as a column, could also be a coordinate system*

   - No proposed solution, but should probably be raised somehow in the REC

B.2. COOSYS - II

2. **Mechanism to discover the coordinate system** : in section 4.2.15 about COORDSYS:

   `Details of the coordinate system for a database column are available as part of the service metadata, available via the TAP_SCHEMA tables [...]`

   ....but nowhere in the TAP standard it is mentioned.

   - Not sure it should be fixed (COOSYS being now deprecated), but just want to mention it
   - **Solution in ADQL-2.1:** remove this text
   - **But also a suggestion for TAP-1.x's authors:**
     - table `TAP_SCHEMA.COOSYS`
     - and column `coosys_id` in `TAP_SCHEMA.columns`
     - *see http://gaia.ari.uni-heidelberg.de/tap/tables*

B.3. IN_UNIT

`IN_UNIT(value, target_unit)`

- **Origin unit?**
  - ok if value is a column published in `TAP_SCHEMA` with a unit
  - *what about numeric constants and operation results?*
- **Solution 1:** assume the given numeric is in the target unit = no conversion, just set a piece of metadata
- ~~**Solution 2:** another function with a 3rd argument:~~
  1. ~~value to convert~~
  2. ~~origin unit~~
  3. ~~target unit~~

B.4. WITH - I

WITH

1. **# of labels VS # of output columns:**
   - ***Example:***

     ```
     WITH t1 ( ra, dec )
          AS (SELECT col1, col2, col3, col4 FROM ...)
     ```

   - Currently, ADQL-Lib expects a strict equality
   - **Solution:** remove the CTE's labels
     - ***Why?*** *they are really useful only with recursive CTEs, which are not supported in ADQL-2.1 ; so, that would just add useless confusion in ADQL*
   - *(CTE = Common Table Expression ~ a table defined in the WITH clause)*

B.4. WITH - II

2. **Unicity of column names:**
   - ***Example:***

```
WITH cte AS (
    SELECT t1.name, t2.name
    FROM t1 JOIN t2 ON ...
)
SELECT cte.name
 FROM cte
```

   - No problem for databases. . . . *until a duplicated column name has to be used/referenced in the query*
   - *Write a warning in the ADQL-2.1 document?*
     - *. . . but probably a more general issue in ADQL (e.g. output columns, case sensitivity, default column labels, . . .)*

5. Others...

- *See appendices for:*
  - *other minor issues*
  - *the case of bitwise operators and hexadecimal*

C. ADQL-2.1 roadmap

A. Implementation status

B. Raised issues

C. ADQL-2.1 roadmap

1. What changed?
2. Status & Roadmap

X. Appendices

C.1. What changed? - I

- **Changed:**
  - deprecation of the coosys argument in geometries + `COOSYS(...)`
  - `REGION` argument's syntax defined by DALI
  - explicit definition of supported datatypes + add `BOOLEAN`
  - preferred xmatch syntax (i.e. `DISTANCE` instead of `CONTAINS`)
  - fix/clarify text of misc things (e.g. subqueries, `mod(...)`, ...)
  - fix typos

C.1. What changed? - II

- **New:**
  - `LOWER(...)`, `UPPER(...)`
  - `ILIKE`
  - `IN_UNIT(...)`
  - `OFFSET`
  - `WITH` *(i.e. Common Table Expression = CTE)*
  - `UNION`, `INTERSECT`, `EXCEPT`
  - add a comment about the Endorsed Note "Catalogue of ADQL User Defined Functions" - 1.0, Jon Juaristi Campillo, et.al.
  - *`CAST(...)`, `TIMESTAMP(...)`, `INTERVAL(...)`**???***
    - *under discussion*
  - ~~*bitwise operators and hexadecimal*~~

C.2. Status & Roadmap

- Currently **P**roposed **R**ecommendation since Jan-2018
- Known implementations:
  - DACHS
  - VOLLT
- Now, the plan is:
  1. fix the few raised issues in the PR
  2. start RFC, end 2019
  3. Ideally: REC at next May-Interop

X. Appendices

A. Implementation status

B. Raised issues

C. ADQL-2.1 roadmap

X. Appendices

1. Misc. minor issues
2. Bitwise operators

X.1.a. LOWER, ILIKE

- still in <SQL_reserved_word>
- not defined elsewhere in the BNF

```
<lower_function> ::=
    LOWER <left_paren> <character_value_expression> <right_

<string_value_function> ::=
    <string_geometry_function>
    | <lower_function>
    | <user_defined_function>
```

X.1.b. INTERSECT vs UNION,

- INTERSECT has the ***same description as EXCEPT***
- INTERSECT has apparently a precedence.

```
<non_join_query_expression> ::=
   <non_join_query_term>
 | <query_expression> UNION [ ALL ] <query_term>
 | <query_expression> EXCEPT [ ALL ] <query_term>

<non_join_query_primary> ::=
   <query_specification>
 | <left_paren> <non_join_query_expression> <right_paren>

<non_join_query_term> ::=
   <non_join_query_primary>
 | <query_term> INTERSECT [ ALL ] <query_expression>
```

- ***=> Re-writing on its way...***

X.1.c. IN_UNIT

- Missing BNF for `IN_UNIT`

X.1.d. IVOID for opt. feat.

The following IVOIDs are not defined in TAPRegExt (1.x):

- `ivo://ivoa.net/std/TAPRegExt#features-adql-string`
- `ivo://ivoa.net/std/TAPRegExt#features-adql-sets`
- `ivo://ivoa.net/std/TAPRegExt#features-adql-common-table`
- `ivo://ivoa.net/std/TAPRegExt#features-adql-type`
- `ivo://ivoa.net/std/TAPRegExt#features-adql-unit`
- `ivo://ivoa.net/std/TAPRegExt#features-adql-bitwise`
- `ivo://ivoa.net/std/TAPRegExt#features-adql-offset`
- => ***TAPRegExt should be updated***

X.2.a. Un-/signed

Integer encoding/sign specification for bitwise operations:

	SQL	Result
PostgreSQL	`~3`	`-4`
SQLServer	`~3`	`-4`
MySQL	`~3`	`18446744073709551612`
MySQL	`CAST(~3 AS signed)`	`-4`

- *Something should be written about the integer encoding expected when using bitwise operations in the ADQL-2.1 document*

X.2.b. Hexadecimal. - I

Hexadecimal values (e.g. `0x42`)

1. **Smallint, integer or bigint?**

   Signed or unsigned? Should it be determined automatically from the number of hexadecimal digits? If less digits, are they the strong or weak bits?

   - ***Example:*** *`0xFFFF` = `-1` if interpreted as smallint, `0xFFFF` = 32768 when interpreted as an integer or long.*

X.2.b. Hexadecimal - I

- **Currently, in VOLLT:**
  - **Default:**
    - if # of hexa. digits $>$ 8, then BIGINT, else, INTEGER
    - given digits represent the weak bits ; ***e.g.*** *`0xFF` = `0x00FF` as an integer, and = `0x000000FF` as a long*
  - **Exception:** VOLLT forbids hexadecimal literals with MySQL
    - ***Why?*** *MySQL can not interpret `0xFFFFFFFF` as an integer (i.e. -1). It always represents it as a bigint (i.e. 4294967295)....even with any combination of `CAST(... AS signed integer)` and `CONV('...', -16, 10)`.*

X.2.c. Hexadecimal II

2. Is it really a good idea to **consider hexadecimal values as numeric**?
   - In all DBMS, it is considered as a binary string.

Conclusion on hexadecimal: no hexadecimal values in ADQL-2.1

X.2.d. BIT_NOT - I

2. **Incorrect BNF syntax of `BIT_NOT` (~)**:
   - ***Example:*** `~3|2 = ~(3|2) = -4`

- Instead of **applying on a single term** as in any DBMS:
  - ***Example:*** `~3|2 = (~3)|2 = -2`

X.2.d. BIT_NOT - II

- **Current BNF:**

```
<bitwise_expression> ::=
  **<bitwise_not> <numeric_value_expression>**
  | <numeric_value_expression>
    (<bitwise_and> | <bitwise_or> | <bitwise_xor>)
    <numeric_value_expression>
```

- **Suggested correction:**

```
<factor> ::= [ <sign> | **<bitwise_not>** ]
             <numeric_primary>

<bitwise_expression> ::=
  <numeric_value_expression>
  (<bitwise_and> | <bitwise_or> | <bitwise_xor>)
  <numeric_value_expression>
```

X.2.e. Precedence - I

3. **Precedence** of all numeric/bitwise operators together?

   Example: *Interpretation of ~3-1|2*5^6/1+2:*

DBMS	Interpretation	Result
PostgreSQL	((~(3-1))|(2*5))^((6/1)+2)	-9
MySQL	((~3)-1)|(((2*(5^6))/1)+2)	-5
SQLServer	((((~3)-1)|(2*5))^(6/1))+2	-1
BNF ADQL-2.1	~(((3-1)|(2*5))^((6/1)+2))	-3
C/Java	((~3)-1)|((2*5)^((6/1)+2))	-5
VOLLT	((~3)-1)|((2*5)^((6/1)+2))	-5

X.2.e. Precedence - II

- **Precedence rules:**

DBMS	Precedence rule
PostgreSQL	`[-+] >> [*/] >> [+-] >> [~] >> [^&|]`
MySQL	`[~-+] >> [^] >> [+-*/] >> [&] >> [|]`
SQLServer	`[~-+] >> [*/] >> [+-^&|]`
BNF ADQL-2.1	`[-+] >> [*/] >> [+-] >> [^&|] >> [~]`
C/Java	`[~-+] >> [*/] >> [+-] >> [&] >> [^] >> [|]`
VOLLT	`[~-+] >> [*/] >> [+-] >> [^&|]`

See also
https://rosettacode.org/wiki/Operator_precedence

- **Alternative solution:** use functions instead of operators *(after all, users will write parenthesis anyway)*
- **Chosen solution:** no bitwise operators in ADQL-2.1