VDP Conformance with Standard SQL

Revision 20240306

Denodo Inc.: 525 University Avenue, Suite 31, Palo Alto, CA 94301. USA www.denodo.com

CONTENTS

1 GOAL....3

2 DATA TYPES....4

3 SQL PREDICATES SUPPORT....6
3.1 ARITHMETIC OPERATORS....6
3.2 PREDICATE OPERATORS....7
3.3 COMPARISON OPERATORS....7

4 SQL FUNCTIONS SUPPORT....8
4.1 CHARACTER STRING OPERATIONS....8
4.2 DATE VALUES FUNCTIONS....10
4.3 TYPE CONVERSION FUNCTIONS....11
4.4 AGGREGATION FUNCTIONS....13

5 QUERY EXPRESSIONS....14
5.1 EXCEPT....14
5.2 GROUP BY....14
5.3 HAVING....15
5.4 INTERSECT....15
5.5 JOIN....15
5.6 ORDER BY....16
5.7 SELECT....16
5.8 UNION....17
5.9 WITH (COMMON TABLE EXPRESSIONS)....17
5.10 SUBQUERIES....18
5.11 QUERY EXPRESSIONS NOT DEFINED IN SQL-92....19

1 GOAL

This document is a reference of the Virtual DataPort conformance with the SQL 92 ANSI standard.
The document is focused on the query capabilities defined in this standard.
This document is aimed at administrators and developers that already have a deep knowledge of the Virtual DataPort Query Language (VQL).

2 DATA TYPES

The following table contains the equivalent type in Virtual DataPort of each SQL type.

In Denodo Versions less than 7.0:

SQL Type Name	Equivalent Type in Virtual DataPort	Remarks
DATE TIME TIME WITH TIME ZONE TIMESTAMP TIMESTAMP WITH TIME ZONE	date	
INTERVAL	Not supported	You can use the functions ADDYEAR, ADDMONTH, ADDDAY, ADDHOUR… to perform similar operations to the ones you can do with the INTERVAL data type.

CHARACTER
CHARACTER VARYING
{
NATIONAL CHAR
| NATIONAL CHARACTER
| NCHAR
}
{
NATIONAL CHAR VARYING
| NATIONAL CHARACTER VARYING
| NCHAR VARYING
	text	The type represents character strings of any character set, including multibyte Unicode ones.

} VARCHAR		
BIT BIT VARYING	blob	
INTEGER NUMERIC SMALLINT	int or long	The precision of these data types is the same as the Java primitives with the same name. The minimum value of an int is -2^{31} and the maximum value is $+2^{31}-1$. The minimum value of a long value is -2^{63} and the maximum value is $+2^{63}-1$.
DECIMAL DOUBLE PRECISION FLOAT REAL	float or double	These data types have the same precision as the Java primitives with the same name. The range of values of float and double values is beyond the scope is this document, but is specified in the Floating-Point Types, Formats and Values section of the Java Language Specification (http://docs.oracle.com/javase/specs/jls/se7/html/jls-4.html#jls-4.2.3)

Addition to the above, support for more data types are added in Denodo 7.0:

SQL Type Name	Equivalent Type in Virtual DataPort	Remarks
DATE	localdate	Date without a timezone
TIME	time	
TIMESTAMP	timestamp	

TIMESTAMP WITH TIMEZONE	timestamptz	
INTERVAL	intervaldaysecond	duration in terms of days, hours, minutes and seconds
INTERVAL	intervaldaymonth	duration in terms of years and months

Type conversion from JDBC sources to Virtual DataPort

Virtual DataPort also supports the following additional types:

- `boolean`, which is not defined by the standard SQL.
- `decimal`, a type to represent decimal numbers with big precision.

The numeric types in Virtual DataPort do not have scale or precision as in the standard SQL so you should use the appropriate type depending on your data requirements.

3 SQL PREDICATES SUPPORT

Predicates are conditions that can be evaluated to return `true` or `false`.
The following sections list the predicates supported and not supported by Virtual DataPort.

3.1 ARITHMETIC OPERATORS

Virtual DataPort supports all the arithmetic operators defined by the standard SQL: `+`, `-`, `*` and `/`.
Virtual DataPort also defines an alias for these operators (see the table below).
These operators can be used with values of the types `int`, `long`, `float`, `double` and `money`.

SQL Operator	Alias in Virtual DataPort
+	SUM (value, value [, value]*)
-	SUBTRACT (value, value [, value]*)
*	MULT (value, value [, value]*)
/	DIV (value, value [, value]*)

Aliases of arithmetic operations

A numeric expression can be prefixed with the sign "-" to reverse the sign of the operand. For example:

```
SELECT – numeric_field_name FROM V
```

If `numeric_field_name` is `1`, (`– numeric_field_name`) is `-1`.

3.2 PREDICATE OPERATORS

Specify a condition that can be evaluated to give a truth-value.

- `<boolean value> OR <boolean value>`
- `<boolean value> AND <boolean value>`
- `NOT (<boolean value>)`
- `IS <boolean value>`

The standard SQL defines the "truth value" `unknown`. Virtual DataPort only supports `true` and `false`.

3.3 COMPARISON OPERATORS

Virtual DataPort supports the following comparison operators defined by the standard SQL:

- = (equal)
- <> (not equal)
- < (less than)
- > (greater than)
- <= (less than or equal)
- >= (greater than or equal)

Other predicates supported by Virtual DataPort:

- `<a> BETWEEN <b> AND <c>`
- `<a> IN <b>, <c>, <d>...`
- `<a> LIKE <pattern>`
  The syntax `<a> LIKE <pattern> ESCAPE <escape character>` is not supported.
- `<a> IS NULL`
- `<a> <operator> { ALL | ANY } <subquery>`
  The operator `SOME` is not supported, but you can use the operator `ANY` instead as it has the same meaning.
- `EXISTS`: see section "Subqueries in the WHERE Clause" in this document.

3.3.1 Unsupported Predicates

The following list contains the predicates not supported by Virtual DataPort.

- `UNIQUE <subquery>`
- `<a> MATCH [ UNIQUE ] [ PARTIAL | FULL ] <subquery>`
- `<a> OVERLAPS <b>`
  As Virtual DataPort does not support date-time intervals, it cannot support this predicate.

4 SQL FUNCTIONS SUPPORT

The following sections list the functions defined by the standard SQL and their equivalent in Virtual DataPort. These sections explain the syntax of each function in Virtual DataPort and when appropriate, the differences with the standard.
Virtual DataPort defines more functions than the standard. For more information about them, read the sections "Functions for Conditions and Derived Attributes" and the appendix "Syntax of Condition Functions" of the Advanced VQL Guide.

4.1 CHARACTER STRING OPERATIONS

4.1.1 CHAR_LENGTH,CHARACTER_LENGTH,OCTET_LENGTH,BIT_LENGTH

These functions return the length of a value

Syntax

SQL Function	Virtual DataPort Equivalent
`CHAR_LENGTH(value)` `CHARACTER_LENGTH(value)`	`LEN (value)`

Remarks
If the value is `NULL`, all the functions return `NULL`.
`OCTET_LENGTH` and `BIT_LENGTH` are currently not supported by Virtual DataPort.

4.1.2 LOWER

Converts text to lowercase.

Syntax
`LOWER( <character value> ) : <character value>`
The syntax of this function conforms to the standard.

Remarks

- `LOWER` returns `NULL` when the input value is `NULL`.

4.1.3 POSITION

Returns the index of a string within another string.

Syntax

SQL Function	Virtual DataPort Equivalent
`POSITION(<string value> IN <string`	`INSTR(<string value>, <text to`

value>)	search:string value>)

Remarks

- For INSTR, the index of the first character is 0 and for POSITION… IN, is 1.
- If any of the parameters is NULL, the function returns NULL.
- INSTR returns -1 if it does not find <string value>. In this case, POSITION… IN returns 0.
- INSTR returns 0 if <string value> has a length of 0. In this case, POSITION… IN returns 1.

4.1.4 SUBSTRING

Returns a substring of a string.

Syntax

```
SUBSTRING(
      <character value> FROM <start position: numeric>
      [ FOR <length : numeric> ] )
```

The syntax of this function conforms to the standard.

Remarks

- If any of the parameters is NULL, the function returns NULL.

4.1.5 TRIM

Returns the input parameter without its leading and trailing space characters.

Syntax

SQL Function	Virtual DataPort Equivalent
TRIM (<character value>)	TRIM(<text value>)
TRIM (BOTH FROM <character value>)	TRIM(<text value>)
TRIM (LEADING FROM <character value>)	LTRIM(<text value>)
TRIM (TRAILING FROM <character value>)	RTRIM(<text value>)
TRIM ({ BOTH | LEADING | TRAILING } <trim character> FROM <character value>)	TRIM([{BOTH | LEADING | TRAILING} [trimcharacter] from | trimcharacter from] value)

Remarks

- The function returns NULL if the input is NULL.

4.1.6 UPPER

Converts text to uppercase.

Syntax
`UPPER( <character value> )`
The syntax of this function conforms to the standard.

Remarks
- `UPPER` returns `NULL` when the input value is `NULL`.

4.1.7 UNSUPPORTED TEXT VALUES FUNCTIONS

The following functions are not supported by Virtual DataPort:
- `CONVERT… USING`
- `TRANSLATE … USING`: the Virtual DataPort functions `REPLACE` and `REPLACEMAP` provide a similar result.

4.2 DATE VALUES FUNCTIONS

SQL Function	Virtual DataPort Equivalent	Remarks
`CURRENT_DATE`	`CURRENT_DATE`	This function returns a "date" value with the fields hour, minute, second and milliseconds set to zero.
`CURRENT_TIME`		Although this function is not supported, a similar result can be achieved with the functions `EXTRACT(…)` and `NOW()`.
`CURRENT_TIMESTAMP`	`CURRENT_TIMESTAMP`	This function returns a "`date`" value.

`EXTRACT (… FROM <value> )`	`EXTRACT (`
	`{ YEAR`
	`| MONTH`
	`| DAY`
	`| HOUR`
	`| MINUTE`
	`| SECOND`
	`}`
	`FROM <value:date> )`
		Returns `NULL` if value is `NULL`.

`OVERLAPS`	Not supported	Virtual DataPort does not support the type "interval of dates".

Date processing functions

4.3 TYPE CONVERSION FUNCTIONS

4.3.1 CAST

Syntax
`CAST ( <value> AS <type> )`

Remarks
The syntax of the function in Virtual DataPort is the same as in the SQL-99. The value of `<type>` can be the name of a Virtual DataPort type or a standard SQL type:

<type> Value	Output Type
`array`	`array`
`text, blob`	`blob`
`text, int, long, float, double, boolean`	`boolean`
`text, date, long`	`date`
`text, int, long, float, double, money`	`double`
`text, enumerated`	`enumerated`
`text, int, long, float, double, money`	`float`
`text, int, long, float, double, money`	`int`
`text, int, long, float, double, money`	`long`
`text, int, long, float, double, money, date`	`money`
`xml, register`	`register`
`text, int, long, float, double, boolean, date, xml, money, link, blob, enumerated, register, array`	`text`
`text, blob, xml, register, array`	`xml`

Type conversion performed by the function `CAST` (Virtual DataPort types)

<type> value SQL Type	Output type
`CHAR (n)`	`text`
`CHARACTER (n)`	`text`

CHARACTER VARYING (n)	text
VARCHAR (n)	text
NCHAR (n)	text
NVARCHAR (n)	text
BIT (n)	blob
BIT VARYING (n)	blob
INTEGER	int
SMALLINT	int
FLOAT	float
REAL	float
DOUBLE PRECISION	double
NUMERIC	double
NUMERIC (n)	double
DECIMAL	double
DECIMAL (n)	double
DATE	date
TIMESTAMP	date timestamp(in Denodo versions >= 7.0)
TIMESTAMP WITH TIME ZONE	Date timestamp(in Denodo versions >= 7.0)

Type conversions performed by the function CAST (standard SQL types)

In some scenarios, Virtual DataPort converts a value to the required type. E.g., Let us say that we have a view `internet_inc` with a field `iinc_id` of type `int`.
The following query works because Virtual DataPort automatically converts `<string value>` into the appropriate data type (`int`).

```
SELECT * FROM internet_inc WHERE iinc_id = '<string value>'
```

According to the standard, to round a number you have to use the `CAST` function. For example, `CAST(0.9681 as INTEGER)` returns `1` and `CAST(0.9681 AS NUMERIC(9,2))` returns `0.93`.
To do this in Virtual DataPort, instead of using `CAST`, you have to use the `ROUND` function. See more about this function in the appendix **Round** of the Advanced VQL Guide.

4.4 AGGREGATION FUNCTIONS

The following table lists the aggregation functions defined by the standard SQL. Virtual DataPort supports these functions and others. See more information about them in the appendix **Aggregation functions** of the Advanced VQL Guide.

SQL Function	Description
`AVG`	Returns the average of the non-null values of an attribute of the table
`COUNT`	Returns the number of non-null values of an attribute of the table
`MAX`	Returns the highest value of an attribute for each group of values
`MIN`	Returns the lowest value of an attribute for each group of values
`SUM`	Returns the sum of all non-null values of an attribute for each group of values

Standard SQL aggregation functions supported by Virtual DataPort

5 QUERY EXPRESSIONS

The following sections list the expressions defined by the standard SQL and their equivalent in Virtual DataPort. These sections explain the syntax of each expression in Virtual DataPort and when appropriate, the differences with the standard.

5.1 EXCEPT

In Virtual DataPort, the expression `EXCEPT` is called `MINUS`. This operation returns all the rows returned by the first query except the rows that are also returned by the second query.

Syntax

```
<query 1> MINUS <query 2>
```

Remarks

The modifier `ALL` in the operation `MINUS` is not supported.

5.2 GROUP BY

The `GROUP BY` operation groups the results of a query by the values of the fields of the view, generating a row for each group. The attributes and expressions with which the `GROUP BY` operation is performed are specified in the `GROUP BY` clause.
The queries with `GROUP BY` can only project the attributes specified in the `GROUP BY`. The other fields of the view can be used only as parameters of aggregation functions.

Syntax

```
SELECT…
FROM…
[ GROUP BY <group by field> [ , <group by field> ]* ]
<group by field> ::= { <field name> | <expression> }
```

When the `GROUP BY` clause is missing, the entire result of the query is considered a single group.

Remarks

The standard defines that <group by field> is a field name. In Virtual DataPort, it can be also one of the following:

- An alias of a projected field. For example,

  ```
  SELECT (field1 + 1) AS alias1, COUNT(*)
  FROM view1
  GROUP BY alias1
  ```

  In this example, the `GROUP BY` operation groups by "`field1 + 1`".

- A position of a field in the `SELECT` clause of the query. For example,

  ```
  SELECT f1, f2, f3
  FROM view1
  GROUP BY 2
  ```

In this example, the GROUP BY operation groups by the field f2.

5.3 HAVING

The HAVING clause is used with GROUP BY to remove the results returned by the GROUP BY that do not meet the <search condition>.

Syntax

```
SELECT…
FROM…
GROUP BY…
HAVING <search condition>
```

Remarks

In Virtual DataPort, <search condition> cannot be a subquery.

5.4 INTERSECT

The intersect operation returns the common elements of the result of two or more input queries.

Syntax

```
<query 1> INTERSECT <query 2>
```

Remarks

The modifier ALL in the operation INTERSECT is not supported.

5.5 JOIN

The JOIN operation combines rows from two views.

Syntax

```
SELECT…
FROM
      <view 1> [ <join type> ] JOIN <view 2> ON <join condition>
    | <view 1> NATURAL JOIN <view 2>
    | <view 1> JOIN <view 2> USING ( <field> [, <field> ]* )
    | <view 1> CROSS JOIN <view2>

<join type> ::= INNER | LEFT [ OUTER ] | RIGHT [ OUTER ] | FULL [ OUTER ]
```

<view1> and <view2> can be either a view or a subquery.

Extended syntax

The following syntax includes the clauses to set implementation details of the join operation:

- Algorithm used to execute the join. The query engine of Virtual DataPort tries to select the most appropriate algorithm for each scenario. However, you can force it to use a different algorithm. See the Virtual DataPort Administration Guide for more details about this.
  - HASH
  - MERGE
  - NESTED

    - NESTED PARALLEL
- <order> is only relevant when the JOIN is a LEFT OUTER JOIN or RIGHT OUTER JOIN.
  If ORDERED, the “left view” is <view A>.
  If REVERSEORDER, the “left view” is <view B>.

```
SELECT…
FROM
      <view 1> [ <algorithm> ] [ <order> ] [ <join type> ]
      JOIN <view 2> ON <condition>
    | <view1> NESTED PARALLEL [ <order> ] [ <join type> ]
      JOIN [ <parallel number:integer> ] <view 2> ON <condition>
    | <view1> [ <algorithm> ] [ <order> ]
      NATURAL [ <join type> ] JOIN <view 2>
    | <view 1> NESTED PARALLEL [ <order> ]
      NATURAL [ <join type> ] JOIN [ <parallel number:integer> ]
      <view 2>
    | <view 1> [ <algorithm> ] [ <order> ] [ <join type> ]
      JOIN <view 2> USING ( <field> [, <field> ]* )
    | <view 1> NESTED PARALLEL [ <order> ] [ <join type> ]
      JOIN [ <parallel number:integer> ] <view 2>
      USING ( <field> [, <field>]* )
    | <view 1> CROSS JOIN <view 2>

<algorithm> ::= HASH | NESTED | MERGE
<order>  ::= ORDERED | REVERSEORDER
```

5.6 ORDER BY

ORDER BY sorts the results by one or more of their fields.

Syntax

```
SELECT…
FROM…
[ ORDER BY <field name> [ ASC | DESC ]
           [, <field name> [ ASC | DESC ] ]* ]
```

When the sort direction (ASC or DESC) is not present, ORDER BY sorts in ascending order.

Remarks

Virtual DataPort supports using the ORDER BY expression in a nested query. For example:

```
SELECT v.*, ROWNUM()
FROM
    (SELECT DISTINCT id, description, ttime, state
     FROM incidents
     ORDER BY cif
) AS v
```

5.7 SELECT

Selects the fields to project from the result obtained by the query.

Syntax

```
SELECT [ DISTINCT ] <select fields>
FROM…
```

```
<select fields> ::=
    {
      *
    | <select sublist> <as clause>
      [, <select sublist> <as clause> ]*

<select sublist> ::= <derived column> | <view name>.*

<derived column> ::= { <expression> | <field name> } [ <as clause> ]

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

Remarks
The modifier ALL is not supported.
In Virtual DataPort, you can not place subqueries in the SELECT clause.

5.8 UNION

The UNION operation returns the union of the tuples returned by two queries.

Syntax
```
<query 1> UNION [ ALL ] <query2>
```

Remarks
The ALL modifier of the UNION operation is ignored, which means that the result of the operation may return repeated rows.
The schema of <query 1> does not have to match the schema of <query 2>. If it does not, the schema of the result is formed by the fields of <query 1> and the fields of <query 2> that are not present in <query 1>.

5.9 WITH (COMMON TABLE EXPRESSIONS)

A *common table expression* (CTE) is a temporary result set that is defined and used for the duration of an SQL statement.
The main advantage of a CTE is that it improves the readability and maintenance of complex queries. The query can be divided into separate “building blocks”, which can then be used to build more complex queries.

Syntax
```
WITH <expression name> [ ( <column name> [, <column name> ]+ ) ]
AS ( <query> )
SELECT <column list>
FROM <expression name>
```

If <column name> is not specified, the names of the fields of the expression are the names of the schema of <query>.

For example, the following statement finds the department with the lowest total pay.

```
WITH department_salary (deptno, totalpay) AS
    (SELECT deptno, SUM(salary)
     FROM EMP
     GROUP BY deptno)
```

```
SELECT deptno
FROM department_salary
WHERE totalpay = (
    SELECT max(totalpay)
    FROM department_salary
)
```

Remarks
Virtual DataPort supports using the `WITH` clause in the `SELECT` and the `CREATE VIEW` statements. It is not allowed in subqueries or in `INSERT` statements.

5.10 SUBQUERIES

The following sections explain how to include subqueries in the `FROM` and `WHERE` clauses of a query.
Note that, as explained in this document, Virtual DataPort does not currently support subqueries in the clauses `SELECT` and `HAVING`.

5.10.1 SUBQUERIES IN THE FROM CLAUSE

In a query, the elements that form the `FROM` clause can be either the name of a view or a query.

Example
```
SELECT * FROM (SELECT * FROM a)
```

5.10.2 SUBQUERIES IN THE WHERE CLAUSE

In a query, the conditions of the `WHERE` clause can be built using the result of a subquery.
See more about this in the section "*Subqueries in the WHERE Clause of the Query*" of the Advanced VQL Guide.

Example 1
```
SELECT * FROM incidences
WHERE taxid IN
    (SELECT taxid
     FROM flat_revenue
     WHERE revenue > 2500)
```

Example 2
```
SELECT *
FROM internet_inc AS a
WHERE EXISTS
    (SELECT b.PINC_ID
     FROM PHONE_INC AS b
     WHERE a.iinc_id = b.pinc_id)
```

In Example 2, the query uses the alias of the main query (`internet_inc AS a`) in the `WHERE` clause of the subquery.

5.11 QUERY EXPRESSIONS NOT DEFINED IN SQL-92

5.11.1 FETCH AND OFFSET

The `OFFSET` clause skips the first N rows of a query's result.
`OFFSET` is usually used with `FETCH` to obtain a subset of rows of a table sorted by a primary key. It is used in scenarios where we need to paginate the contents of the query.

Syntax

```
SELECT…
FROM…
ORDER BY <primary key fields>
[ OFFSET <number> { ROW | ROWS } ]
[ FETCH { FIRST | NEXT } [ <number> ] { ROW | ROWS } ONLY
```

The modifiers `FIRST` and `NEXT` and `ROW` and `ROWS` can be used interchangeably.

Remarks

Although sorting by the primary key fields is not required, it is recommended to obtain consistent results when paginating the rows of a query.
In Virtual DataPort, `FETCH` and `OFFSET` can only be used when querying a view. Creating a derived view with `FETCH` and/or `OFFSET` is not valid.
SQL-92 does not define these clauses, but they are defined in the SQL-2011 standard. Many database management systems have provided similar clauses to these (`TOP`, `LIMIT`, `ROWNUM`…). However, unlike `FETCH` and `OFFSET`, these clauses are not standard.