Application Layer NQL Documentation

Ceyoniq

Version 10.1.1508, 09.07.2026

Table of Contents

Nscale Query Language . . . 1
1. Basic Structure . . . 2
2. The select clause . . . 3
2.1. Examples . . . 3
3. The where clause . . . 5
3.1. Examples . . . 6
4. The orderby clause . . . 9
4.1. Examples . . . 9
5. The paging element . . . 10
5.1. Examples . . . 10
6. The scope element . . . 12
6.1. Examples . . . 12
7. The count element . . . 13
7.1. Examples . . . 13
8. The hidden element . . . 14
8.1. Examples . . . 14
9. The timeout element . . . 15
9.1. Examples . . . 15
10. Properties . . . 16
10.1. Scope Repository, Workflow and Business Process (BPMN) . . . 16
10.2. Scope Masterdata, Principal (User Management) and Calendar (Collaboration) . . . 16
10.3. Notations . . . 17
11. Constants . . . 18
11.1. Examples . . . 18
11.2. Convenience constants . . . 19
12. Context Variables . . . 20
12.1. Examples . . . 21
13. Query Functions . . . 23
13.1. Convert Functions . . . 24
13.2. Mathematical Functions . . . 28
13.3. String Functions . . . 34
13.4. Date/Timestamp Functions . . . 45
13.5. List Functions . . . 53
13.6. Other Functions . . . 60
14. Aggregate Searches . . . 77
14.1. The select clause . . . 77
14.2. The where clause . . . 77
14.3. The orderby clause . . . 78

14.4. Examples . . . 78
15. Subqueries . . . 80
15.1. The select clause . . . 80
15.2. The where clause . . . 81
16. Bulk Operations . . . 84
16.1. Bulk Update . . . 84
16.2. Bulk Delete . . . 84

Nscale Query Language

The nscale Query Language (NQL) is a language which can be used to query information stored by *nscale Server Application Layer*.

The *nscale Server Application Layer* API (*Advanced Connector*) provides several search methods which demand an NQL string as parameter to determine the submitted search. Parts of the NQL can also be used to configure the *nscale Server Application Layer*, for example to define visibility rules or formatted properties.

The syntax of NQL is adapted from SQL syntax and is explained in this document. Be aware that NQL is a textual representation of an equivalent object search model (see class *SearchControl* of the *Advanced Connector* API). It is recommended to use the object search model instead of an NQL statement to enhance performance (an NQL statement has to be parsed and transformed into a *SearchControl* before a search can be executed). This document describes the complete NQL of *nscale Server Application Layer* of version 10.1.

Chapter 1. Basic Structure

An NQL instruction is made up of a maximum of the following elements:

- `select` <list of properties>
- `where` <condition>
- `orderby` <sorting list>
- `paging` (number = <long>[, size = <long>])
- `scope onelevel` or `scope subtree`
- `count`
- `hidden`
- `timeout`=<seconds>

All elements are optional (so the empty string is a valid NQL expression), however the order of the elements is prescribed. If an element is specified, it can itself contain required and optional elements. All language elements with the exception of function names, context variables and nscale object names can be provided in lower or upper case. Function names, context variables, and nscale object names such as properties or object class names must match the upper and lower case conventions that are specified or created in the system.



In a first step the NQL parser tries to parse an NQL statement by using the much simpler NQL 6 grammar (in nscale 6 there were no query functions, instant properties, filtered properties, aggregate functions and so on). If this fails (e.g. because query functions are used) in a second step the NQL parser tries to parse the NQL statement by using the much more complex NQL 7 grammar. It is possible though to give the parser a hint to use NQL 7 grammar already in the first step to prevent the needless try with NQL 6 grammar when nscale 7 functionality is contained. This is done by appending the characters `\7` as a prefix of the NQL statement, e.g. `\7 select up(displayname)`.

Chapter 2. The select clause

The properties to be returned are determined by the select clause. The select clause starts with key word `select`, followed by an arbitrary number of comma-separated properties (or alternatively a wildcard *, see examples below). Temporarily calculated expressions, called instant formatted properties or shortened instant properties, can also be included in the select clause. An instant property is determined by a *QueryOperand*, which is one of a property, a constant, a context variable or a function. Note that besides provided properties the returned result set always contains key information for every contained row. So, if no further information is necessary, the select clause may be omitted.

2.1. Examples

- `select *` (strictly disadvised)
- `select creationdate, displayname, objectclass, version`
- `select displayname, yearcreation=year(initialcreation)`

In the last example, the instant property with alias *yearcreation* contains the year of creation of a resource. For an instant property, an alias should be provided, however in *nscale Server Application Layer* of at least version 7.11 this is not mandatory anymore. Nevertheless it is recommended to use an alias in productive code, as clients may fail over a missing alias (e.g. for compatibility reasons). An alias may be omitted for testing purpose, e.g. in the *nscale Query Tester*.



It is strictly disadvised to use *select **, as this results in an overload of the database. It is only provided for convenience and testing purpose and should not be used in productive code. There are even more convenience abbreviations which are meant for testing purpose and should not be used in productive code, for example:

- `select *\(f)` returns all properties except fulltext properties
- `select *\(c)` returns all properties except computed properties
- `select *\(m)` returns all properties except formatted properties
- `select *\(f, c, m)` combination of the three above
- `select *\(mv)` returns all properties except multi-value properties
- `select *\(sv)` returns all properties except single-value properties
- `select *\(sp)` returns all properties except system properties
- `select *\(cp)` returns all properties except custom properties
- `select *\(v)` returns all properties except virtual properties
- `select *\(pv)` returns all properties except pseudo-virtual properties
- `select *\(s)` returns all properties except string properties
- `select *\(i)` returns all properties except integer properties
- `select *\(l)` returns all properties except long properties

- `select *\(d)` returns all properties except double properties
- `select *\(b)` returns all properties except boolean properties
- `select *\(dt)` returns all properties except date properties
- `select *\(ts)` returns all properties except timestamp properties
- `select *\(bt)` returns all properties except blob properties
- `select *\(aq)` returns all properties except area qualified identifier properties
- `select *\(_s)` returns all properties except non-string properties
- `select *\(_i)` returns all properties except non-integer properties
- `select *\(_l)` returns all properties except non-long properties
- `select *\(_d)` returns all properties except non-double properties
- `select *\(_b)` returns all properties except non-boolean properties
- `select *\(_dt)` returns all properties except non-date properties
- `select *\(_ts)` returns all properties except non-timestamp properties
- `select *\(_bt)` returns all properties except non-blob properties
- `select *\(_aq)` returns all properties except non-area qualified identifier properties

Chapter 3. The where clause

The where clause starts with key word `where` followed by a condition which must be met by a resource in *nscale Server Application Layer* in order that it is part of returned result set. The condition may consist of several expressions which are *OR* or *AND* combined. An expression of a condition consists of a left hand side and a right hand side, combined by an operator. The left hand side and the right hand side must be a *QueryOperand*, which is one of a property, a constant, a context variable, a function or a list of these. Supported operators are:

Operator	Description
`=`	The equal operator
`!=`	The not equal operator; alternative representation is `<>`
`>`	The greater operator
`>=`	The greater or equal operator
`<`	The smaller operator
`<=`	The smaller or equal operator
`in`	The *in* operator; note that the right hand side of an *in* operator must be a list; alternative representation is `[]`
`between`	The *between* operator to search for an including interval; note that the right hand side of a *between* operator must be a list which contains exactly two elements; alternative representation is `><`
`like`	The *like* operator to search for matching resources; the wildcard * stands for an arbitrary number of characters while the wildcard ? stands for exactly one character; wildcards are escaped by the \ character; alternative representation is `~`
`is null`	The *is null* operator to search for undefined values; note that the right hand side of a *is null* operator must be omitted; alternative representation is `= #`
`is not null`	The *is not null* operator to search for defined values; note that the right hand side of a *is not null* operator must be omitted; alternative representation is `!= #`
`@=`	The *soundex equals* operator to search for values which sounds similar (Soundex algorithm); alternative representation is `soundexEquals`
`~=`	The *similar to* operator to search for values which accomplish a certain similarity (Levenshtein algorithm); alternative representation is `similarTo`
`%=`	The *equals ignore case* operator to search for case insensitive values; alternative representation is `equalsIgnoreCase`
`%~`	The *like ignore case* operator to search for matching resources (see *like* operator, but ignore case); alternative representation is `likeIgnoreCase`
`*=`	The *matches* operator to search for resources which match a regular expression; alternative representation is `matches`

All elements of a condition may be enclosed in parentheses to improve legibility or to raise the priority of a sub condition. As an *AND* condition has a stronger binding than an *OR* condition, to get expected results, an appropriate parenthesis is necessary. Example:

```
where displayname like 'a*' and (resourcetype = 1 or resourcetype = 2)
```

Note that the left hand side and the right hand side of an expression must have the same data type. However, integers and longs can be mixed together, as well as dates and timestamps. In the latter case, for dates a timestamp midnight is considered. If an expression contains a property of type boolean, the operator and the right hand side may be omitted for convenience and to improve legibility. So instead of `where hasnote = true` it is sufficient to request `where hasnote`. To negate an expression, a *NOT* condition can be used. So instead of `where hasnote != true` it is also possible to request `where not hasnote`.

For multi-value properties, a special handling is necessary. A multi-value property in a condition must be used inside of an *EXISTS* condition. In turn, all properties inside of an *EXISTS* condition must belong to the same multi-value scope. Example:

```
where exists(itemcontenttype = 'text/plain' and itemlength > 100)
```

This condition finds all resources whose multi-value scope contentiteminfo contains an entry with content type *text/plain* and a length > 100.

To sum it up, a condition is a representation of one of the following components:

Name	Format	Remarks
Expression	`<QueryOperand> <Operator> <QueryOperand>`	Note that for operator *is null* and *is not null* the right hand side query operand must be omitted
AND-Condition	`<Condition> and <Condition>`	Alternative representation is `&`
OR-Condition	`<Condition> or <Condition>`	Alternative representation is `|`
NOT-Condition	`not <Condition>`	Alternative representation is `!`
EXISTS-Condition	`exists(<Condition>)`	Alternative representation is `€`

Note that the where clause is optional in principle, but usually it is reasonable to restrict the number of rows of a returned result set, so it should rarely be omitted.

3.1. Examples

In the following examples, we assume that there is a single-value property *lastname* of type string, as well as a multi-value scope *CARS* containing the properties *car* of type string and *color* of type string.

NQL	Description
`where displayname = 'Invoice'`	Finds all resources where property displayname equals *Invoice*

NQL	Description
`where displayname like 'Invoice*'`	Finds all resources where property displayname begins with *Invoice*
`where not displayname like 'Invoice*'`	Finds all resources where property displayname does not begin with *Invoice*
`where displayname is null`	Finds all resources where property displayname is null
`where displayname is not null`	Finds all resources where property displayname is not null
`where lifecyclestate in (1,2,3)`	Finds all resources where lifecyclestate is finalized (1), to be archived (2) or archived (3)
`where lifecyclestate < 1`	Finds all resources where lifecyclestate is indexed (0)
`where creationdate between (2016-01-01, 2016-12-31)`	Finds all resources where current version has been created between January 1st 2016 and December 31th 2016
`where lastname @= 'Smiths'`	Finds all resources where lastname sounds like *Smiths*, e.g. Smith, Smyth, Smythe etc.
`where lastname ~= 'Smiths'`	Finds all resources where lastname is similar to *Smiths*, e.g. Smith, Smyth, Smythe etc.
`where lastname %= 'sMithS'`	Finds all resources where lastname equals ignore case *Smiths*
`where exists(color = 'red')`	Finds all resources where color contains an entry *red*
`where exists(color = 'red' or color = 'green')`	Finds all resources where color contains an entry *red* or an entry *green*
`where exists(color in ('red', 'green'))`	This statement is equivalent to the former one
`where exists(color = 'red') and exists(color = 'green')`	Finds all resources where color contains an entry *red* and an entry *green*.
`where exists(color = 'red' and color = 'green')`	Does not find any resource (an entry in color cannot be *red* and *green* at the same time).
`where exists(color != 'red')`	Finds all resources where color does contain an entry which is not *red*
`where not exists(color = 'red')`	Finds all resources where color does not contain an entry *red*
`where not exists(color != 'red')`	Finds all resources where color does not contain an entry which is not *red*
`where exists(color is not null)`	Finds all resources where color does contain at least one entry

NQL	Description
`where not exists(color is not null)`	Finds all resources where color does not contain any entry
`where exists(car = 'Mercedes' and color = 'blue')`	Finds all resources where multi-value scope *CARS* contains a *blue Mercedes*
`where exists(car = 'Mercedes') and exists (color = 'blue')`	Finds all resources where car contains a *Mercedes* and color contains *blue*



Be aware that the result of a where clause may differ for every database dialect, as every database dialect has a different behavior in detail (e.g. Oracle does not distinguish between *null* and an empty string). This is also true for the orderby clause.



Be additionally aware that the performance of a search is highly dependent on how it is formulated in NQL and whether there are suitable database indexes present. E.g. for case-ignore operators like `%=` and `%~`, the system can not provide suitable database indexes automatically. They have to be generated manually. Note that not all database dialects support the necessary type of database indexes for such an operation.

Chapter 4. The orderby clause

The orderby clause starts with key word `orderby` or alternatively `order by`. With the orderby clause the sorting of the query results can be determined. This is done by specifying the property according to which the sorting should take place and, optionally, the sort order (asc (ascendant) or desc (descendant)). The property may also be an instant property. By specifying further properties the sub-sorting of results can be specified. The syntax is:

```
orderby <property name> [asc|desc][, <property name> [asc|desc], ...]
```

If sort order is not specified, the result set will be sorted in ascending order.



Note that sorting by a multi-value property is not supported. To overcome this restriction, it is possible though to define an appropriate value property which uses for example query function `listEntry` or `toFlat` on the multi-value property.

4.1. Examples

- `orderby initialcreation desc, identifier`
- `order by displayname`
- `orderby year(creationdate) desc, displayname`
- `orderby length(displayname) asc, displayname`

Chapter 5. The paging element

Paging allows to specify a segment of the actual hits of a search which is returned as result. This is especially useful when a search would return a large number of rows. It enables to iterate through the actual result set by submitting subsequent searches, each with a small number of rows. For the paging element, the page number (starting by 1) and the page size can be specified, e.g. a page number 1 and a page size 100 returns the first 100 hits (the first page) of a search.



Note that the order of the entries of a page is specified by the orderby clause. If the orderby clause is omitted or does not determine a unique order, it is possible that duplicate entries may occur in different pages and on the other hand entries may be missing at all. To overcome this behavior, it is recommended to add an orderby clause which determines a unique order when using paging, e.g. `order by identifier` for the scope *Repository*.

5.1. Examples

NQL	Description
`paging(number=1, size=100)`	Returns the first 100 hits (the first page) of a search
`paging(size=100, number=1)`	This statement is equivalent to the former one
`paging(number=2, size=100)`	Returns the second 100 hits (the second page) of a search
`paging(size=25)`	Returns the first 25 hits (the first page) of a search
`paging(number=1)`	Returns the first 1000 hits (the first page) of a search
`paging(1, 50)`	Returns the first 50 hits (the first page) of a search
`paging(2, 50)`	Returns the second 50 hits (the second page) of a search
`paging(30)`	Returns the first 30 hits (the first page) of a search
`paging()`	Returns the first 100 hits (the first page) of a search
`[0, 100]`	Returns the first 100 hits (the first page) of a search. Caution: for this notation, the page number is zero-based!
`[50]`	Returns the first 50 hits (the first page) of a search.

NQL	Description
`[#1]`	Returns the second 100 hits (the second page) of a search. Caution: for this notation, the page number is zero-based!
`[]`	Returns the first 100 hits (the first page) of a search.

Chapter 6. The scope element

The scope element is only effective for searches in *Repository* and in *UserManagement*.

In *Repository*, the scope element can be used to restrict the search to the current folder or, additionally, expand it to include the subfolders of the current folder. With `scope onelevel` the search is restricted to current folder while with `scope subtree` the search is expanded to all subfolders of the current folder. If no scope is specified, `scope onelevel` takes effect.

In *UserManagement*, a search with `scope onelevel` initiates a search based on *Principal* and join to *PrincipalInfo*, while a search with `scope subtree` initiates a search based on *PrincipalInfo* and join to *Principal*. So, a search with `scope onelevel` finds users, positions and groups of the configured domains, but no additional or orphaned principal info objects. In turn, a search with `scope subtree` finds all existing principal info objects, but no positions (positions do not have additional information) and no principals who do not have default principal info yet.

6.1. Examples

- `select displayname` returns display name of all resources of current folder
- `select displayname scope onelevel` equivalent to the statement before
- `select displayname scope subtree` returns display name of all resources of subtree of current folder

Chapter 7. The count element

The count element is only useful in a search with paging. If this element is used, the total number of results is determined and returned. This can be useful if e.g. only the first 100 hits are requested but the actual number of hits should also be provided.

7.1. Examples

- `paging (1, 50) count`
- `[0, 0] scope subtree count`

In the first example, the rows 1 to 50 and the total number of hits are returned. In the second example an empty result set is returned but the total number of hits is provided. Note that the count element initiates a parallel *select count* search in the database which can be rather time consuming (for the second example only the *select count* search is initiated). So the count element should only be used, when the actual row count is really needed. Note that when the count element is omitted, the result set of a search at least contains information, whether more results are available.

Chapter 8. The hidden element

The hidden element is only effective for searches in *Repository*, *Workflow* and *Business Process*.

Normally, resources for which the system attribute *hidden*, *processhidden* respectively *bp_processhidden* is set are not taken into account when evaluating an NQL query. If the hidden element is specified in an NQL query, however, these objects are taken into account.

8.1. Examples

- `where resourcetype = 2 and hidden scope subtree hidden`
- `select processhidden hidden`

In the first example, all hidden documents are returned. In the second example, hidden as well as not hidden processes are returned, identified by selected property *processhidden*.

Chapter 9. The timeout element

The timeout element can be used to define a timeout for the database search. When a positive number is provided, the timeout is set to given number in seconds. A negative number or 0 results in no timeout. When the timeout element is omitted, the configured value in DbSetting is effective.

9.1. Examples

- `timeout=5`
- `timeout=0`

In the first example, a timeout of 5 seconds for the database search is effective. In the second example, no timeout for the database search is effective.

Chapter 10. Properties

Note that properties are also often called attributes, these two notations are synonymous. For every different scope (like *Repository*, *Workflow*, *Masterdata* etc.) there is a set of properties provided which can be used in NQL. The set of properties may consist of system defined as well as user defined properties. Whether a property can be used in the *select*, *where* or *orderby* clause depends on several settings, like the type of a property or whether it is single-valued or multi-valued.

10.1. Scope Repository, Workflow and Business Process (BPMN)

For the scopes *Repository*, *Workflow* and *Business Process* it is easy to decide, whether a property can be used in the *select*, *where* or *orderby* clause. Every property in these scopes has three boolean flags *Selectable*, *Searchable* and *Sortable*. If a property is *Selectable*, it may be used in the select clause. If a property is *Searchable*, it may be used in the where clause. If a property is *Sortable*, it may be used in the orderby clause. Note that for the flags *Searchable* and *Sortable* there are some restrictions, which means not for every property it is allowed to set these flags to *true*.

A property is not allowed in the where or orderby clause when

- it is of type Blob
- it is a system computed property
- it is a custom computed property which depends on other necessary properties
- it is a formatted property which depends on other non-searchable or non-sortable components
- it is a fulltext property

A property is also not allowed in the orderby clause when

- it is multi-valued

Note that it is possible though to sort over the first entry of a multi-value property by creating a value property with the format `entry(myMultivalueProperty, 0)`. Another option is to create a value property with format `toFlat(myMultivalueProperty)`.

10.2. Scope Masterdata, Principal (User Management) and Calendar (Collaboration)

There are some differences to the scopes *Repository*, *Workflow* and *Business Process*. First, there are no system computed properties in these scopes (there may be custom computed properties in scope *Masterdata* though), and in scopes *Principal* and *Calendar* there are even no formatted properties. Second, in scopes *Principal* and *Calendar* there are no user defined properties, the set of provided properties is fixed and defined by system. In turn, in scope *Masterdata* there are no system properties but only user defined properties. The resulting constraints are:

- a property of type Blob is not sortable; it is only searchable when using operator `is null` or `is`

`not null`

- a multi-value property is not sortable
- a formatted masterdata property may only be searchable respectively sortable when the contained components are searchable respectively sortable

10.3. Notations

The name of a property is case-sensitive, so the use of upper and lower letters is essential. In scopes *Repository*, *Workflow* and *Business Process*, it is possible to create a property which references a masterdata key property. If such a reference is created, all properties of corresponding masterdata scope are provided as virtual properties and it is also possible to copy these properties in actual scope (so called pseudo-virtual properties). The notation for virtual properties is `<masterdata key reference>:<masterdata scope>$<masterdata property>`. The notation for pseudo-virtual properties is `<masterdata key reference>$<masterdata scope>$<masterdata property>`.

Chapter 11. Constants

Constants may be used in the select, where and orderby clause. For the different types, the following syntax is required:

Type	Description
`String`	A constant of type string must be quoted either in single quotes ' or in double quotes ". If the string itself contains the quoting sign, the quoting sign must be escaped by the quoting sign itself (double quoting sign).
`Boolean`	A constant of type boolean must be either `true` or `false`
`Integer`	A constant of type integer is a number in the range of a java Integer
`Long`	A constant of type long is a number in the range of a java Long
`Double`	A constant of type double is a number in the range of a java Double, the decimal divider is a dot, e.g. 3.14
`Date`	A constant of type date must be in format `yyyy-MM-dd`, e.g. 2017-12-31
`Timestamp`	A constant of type timestamp must be in format `yyyy-MM-ddTHH:mm:ss`, e.g. 2017-12-31T12:34:56
`AreaQualifiedIdentifier`	A constant of type area qualified identifier must consist of the name part of the area qualified identifier as string

A list of constants must be set in parentheses, e.g. `(1, 2, 3)`. Note that sorting by a constant is not reasonable and is not supported by many database dialects, so something like `orderby 87` should not be used.

11.1. Examples

NQL	Description
`select 'I was bored before I even began'`	Returns given constant
`select x='I was bored before I even began'`	The same as above but with alias x
`select 'foo', 1, true`	Returns constants foo as string, 1 as long and true as boolean
`where initialcreation >= 2017-01-01`	Finds all resources which have been created in 2017 or later
`where hasnote = true or hasrendition = false`	Finds all resources which have at least one note or no rendition
`where objectclass in ('D1', 'D2', 'D3')`	Finds all resources with objectclass D1, D2 or D3; the property objectclass of type area qualified identifier demands string constants (the name part of the area qualified identifier)
`where resourcetype in (1, 2, 3)`	Finds all folders, documents and links

NQL	Description
`orderby substring(displayname, 0, 1)`	Sorts the result set by the first character of the displayname; in this case, the use of constants 0 and 1 in the orderby clause is reasonable, as they are used in a function which also contains a property

11.2. Convenience constants

For special properties which act like enumerations, NQL provides constants which can be used for convenience:

Property	Constants	Example
resourcetype (scope Repository)	$folder, $document, $link, $annotation, $note, $rendition, $container, $addition	`where resourcetype in ($folder, $document)`
lifecyclestate (scope Repository)	$indexed, $finalized, $toBeArchived, $archived, $error	`where lifecyclestate = $archived`
deletestate (scope Repository)	$none, $logicallyDeleted, $logically, $automaticallyDeleted, $automatically	`where deletestate = $none`
linkreferencetype (scope Repository)	$softlink, $hardlink	`where linkreferencetype = $hardlink`
subtype (scope Repository)	$xsap, $mail, $vcard, $calendar, $partition, $office, $lta, $file, $signature, $sapilm	`where subtype = $xsap`
type (scope UserManagement)	$user, $position, $group	`where type = $user`

Chapter 12. Context Variables

The *nscale Server Application Layer* provides several context variables. The value of a context variable depends on the context of the current request. The following context variables are available:

Context Variable	Type	Description
`%currentUserPrincipalId`	String	Provides the principal id of current user (the user who initiated current request)
`%currentDefaultPositionPrincipalId`	String	Provides the principal id of the default position of current user
`%currentPositionPrincipalIds`	List < String >	Provides the principal ids of all positions of current user
`%currentGroupPrincipalIds`	List < String >	Provides the principal ids of all groups (hierarchical) of current user
`%currentProxiedOrgEntityPrincipalIds`	List < String >	Provides the principal ids of all organizational entities where the current user has proxy competences for; note that proxy competences are transitive
`%currentHeadedOrgEntityPrincipalIds`	List < String >	Provides the principal ids of all organizational entities where the current user has head competences for (flat supervisor competence)
`%currentManagedOrgEntityPrincipalIds`	List < String >	Provides the principal ids of all organizational entities where the current user has manager competences for (hierarchical supervisor competence)
`%currentOwnedOrgEntityPrincipalIds`	List < String >	Provides the principal ids of all organizational entities where the current user has owner competences for; reading access to principal folders of owned organizational entities is granted
`%actAsPositionIds`	List < String >	Provides the principal ids of all positions where the current user has agent competences for; an agent competence on a position allows to login as principal who owns the position
`%agentPrincipalId`	String	Provides the principal id of a principal who acts as an agent in current session, e.g. who logged in as a principal by using agent competences
`%currentWorkPositionPrincipalId`	String	Provides the work position id of current user which must be the id of an existing position of current user or null; if not null, workflow requests are executed in the context of this id by default
`%today`	Date	Provides the current date
`%now`	Timestamp	Provides the current date and time
`%todayAtMidnight`	Timestamp	Provides the current date at midnight

Context Variable	Type	Description
`%tomorrowAtMidnight`	Timestamp	Provides the next date at midnight
`%clientHost`	String	Provides the name of the host which sent current request
`%clientIPAddress`	String	Provides the ip address of the host which sent current request
`%clientApplication`	String	Provides the name of the *nscale* application which sent current request
`%clientLocale`	String	Provides the locale of the client application which sent current request
`%clientToken`	String	Provides a token which has been set in current session by the client application (user defined value without any meaning for the *nscale Server Application Layer*)
`%similarityThreshold`	Double	Provides the configured threshold for a similarity search as a value between 0 and 1; two strings are considered as similar, when the similarity algorithm returns a value greater than or equal to this value
`%systemId`	String	Provides the system id, uniquely identifying an *nscale Server Application Layer* system
`%currentResource`	String	Provides the id of currently affected resource
`%currentWorkflowInstance`	String	Provides the id of currently affected workflow instance
`%currentBusinessProcessInstance`	String	Provides the id of currently affected business process instance
`%webBaseUrl`	String	Provides the web base url of associated *nscale Web Server*
`%activePositionIds`	List < String >	Provides the principal ids of all active positions of current user
`%activeGroupIds`	List < String >	Provides the principal ids of all active groups of current user which are hierarchically accessible through active organizational entities
`%activeProxiedOrgEntityIds`	List < String >	Provides the principal ids of all active organizational entities where the current user has active proxy competences for; note that proxy competences are transitive, transitivity is not disabled by inactive organizational entities but it is disabled by inactive competences

12.1. Examples

NQL	Description
`select %clientLocale`	Returns the locale of the client application which sent current request, e.g. *de* or *en_US*
`where initialcreator = %currentUserPrincipalId`	Finds all resources which have been created by current user
`where exists(involvedpositionids in %currentPositionPrincipalIds)`	Finds all workflow instances where current user has been involved. Note that `%currentPositionPrincipalIds` must *NOT* be set in parentheses as this context variable is already a list, so setting parentheses would result in a list of lists
`where creationdate >= %today`	Finds all resources which have been created today

Chapter 13. Query Functions

The *nscale Server Application Layer* provides 96 query functions which can be used in the select, where and orderby clause. The parameters of a query function are *QueryOperands* which can be one of a property, a constant, a context variable, a query function or a list of these. Note that a parameter of a query function which is parenthesized is automatically converted to a list. For most query functions most parameters support single-values and multi-values. In general, when at least one parameter is multi-valued, the following logic is implemented (there are exceptions from this rule though):

- the return value is multi-valued
- the length of returned list is determined by the length of the longest list of all list parameters
- all list parameters which are shorter are logically filled up with null values
- a single-value parameter behaves like a multi-value parameter where all list entries contain the single value
- the query function is executed line by line for every row



Be aware that the result of a query function in the select clause may differ from the result in the where or orderby clause, as the result of the select clause is calculated in the *nscale Server Application Layer* while the result of the where and orderby clause is calculated in the database. This means, the result of the where and orderby clause depends on the implementation of the function within the database (which actually may be different for every database dialect). Be also aware that the (extensive) use of query functions in the where or orderby clause may decrease the performance of searches drastically, so use them with care. This is in particular the case when nested query functions are used (query functions inside of query functions).

Note that some of the functions are not searchable and/or not sortable, because there is no equivalent function in the database (this may depend on used database dialect). Whenever this is the case, it is mentioned in the descriptions below. However, these restrictions are only effective, when a function contains properties as parameters of the function, otherwise the value of the function results in a constant which can be calculated before the search is executed. In this case, a function is always searchable (it could also be sortable, but sorting by a constant is not reasonable). The provided query functions can be categorized in the following groups:

- convert functions
- mathematical functions
- string functions
- date/timestamp functions
- list functions
- other functions

13.1. Convert Functions

There are 10 convert functions. These functions convert the type of an operand into another type.

13.1.1. ToString

NQL	`toString`, `str`, `s` or `§`
Return type	String
Parameters	`operand` of any type except Blob
Available since	7.1
Description	Converts given operand to string.

Examples

NQL	Description
`select toString(lifecyclestate)`	Converts property lifecyclestate to string
`where s(lifecyclestate) = '1'`	Finds all resources with state finalized
`orderby s(objectclass)`	Sorts by objectclass name

13.1.2. ToInteger

NQL	`toInteger` or `i`
Return type	Integer
Parameters	`operand` of type String, Integer, Long, Double or Boolean
Available since	7.1
Description	Converts given operand to integer.

Examples

NQL	Description
`select i(hasnote)`	Converts property hasnote to integer, all resources having at least one note return 1, the others return 0

13.1.3. ToLong

NQL	`toLong` or `l`

Return type	Long
Parameters	`operand` of type String, Integer, Long, Double, Date, Timestamp or Boolean
Available since	7.1
Description	Converts given operand to long.

Examples

NQL	Description
`select l(deletestate)`	Converts property deletestate to long
`select l(initialcreation)`	Returns the milliseconds of initialcreation since epoch

13.1.4. ToDouble

NQL	`toDouble` or `d`
Return type	Double
Parameters	`operand` of type String, Integer, Long, Double or Boolean
Available since	7.1
Description	Converts given operand to double.

Examples

NQL	Description
`select d(lifecyclestate)`	Converts property lifecyclestate to double

13.1.5. ToBoolean

NQL	`toBoolean` or `b`
Return type	Boolean
Parameters	`operand` of type String, Integer, Long, Double, Boolean or Blob
Available since	7.1
Description	Converts given operand to boolean.

Examples

NQL	Description
`select b(deletestate)`	Converts property deletestate to boolean, all resources which have been deleted logically return true

13.1.6. ToDate

NQL	`toDate`
Return type	Date
Parameters	`operand` of type Date, Timestamp, Long or String
Available since	7.1
Description	Converts given operand to date.

Examples

NQL	Description
`select toDate(creationdate)`	Converts property creationdate to date, which means the time part is cut off
`where toDate(initialcreation) = %today`	Finds all resources which have been created today

13.1.7. ToDatetime

NQL	`toDatetime`
Return type	Timestamp
Parameters	`operand` of type Date, Timestamp, Long or String
Available since	7.1
Description	Converts given operand to timestamp.

Examples

NQL	Description
`select toDatetime(%today)`	Returns today with time midnight (redundant to context variable %todayAtMidnight)

13.1.8. ToBlob

NQL	`toBlob`
Return type	Blob
Parameters	`operand` of type String, Integer, Long, Double or Boolean
Available since	7.5
Description	Converts given operand to blob. This function is not searchable and not sortable.

Examples

NQL	Description
`select toBlob(entries( (1, 2, 3) ))`	Returns a blob (byte array) of length 3, containing entries 1, 2 and 3

13.1.9. ToAQI

NQL	`toAQI`
Return type	AreaQualifiedIdentifier
Parameters	`operand` of type String
	`aqiType` of type Integer, Long or String, must not be a list
Available since	7.10
Description	Converts given operand to area qualified identifier. The second parameter determines the type of area qualified identifier, e.g. `0` or `o` for an ObjectclassName, `1` or `f` for a FormName, `2` or `t` for a FolderTemplateName, `3` or `e` for an ExpirationRuleName, `4` or `r` for a RetentionRuleName and `5` or `p` for a ProcessDefinitionName. This function is not searchable and not sortable.

Examples

NQL	Description
`select toAQI('F0', 0)`	Returns objectclass name F0

13.1.10. BaseConvert

NQL	`baseConvert`
Return type	String
Parameters	`number` of type Integer, Long or String

	`sourceBase` of type Integer or Long (optional, default 10 when number is numeric, else 16)
	`targetBase` of type Integer or Long (optional, default 16 when number is numeric, else 10)
Available since	7.7
Description	Converts given number from source base to target base, e.g. from base 10 to base 16 to get the hexadecimal representation. Supported are any bases between 1 and 36 (including). This function is searchable and sortable only for source base 10 and target base 16 and vice versa.

Examples

NQL	Description
`select baseConvert(substring(storagelayerarcid, 8), 16, 2)`	Converts the property storagelayerarcid from hexadecimal to binary system; the first 8 signs are omitted (otherwise, an overflow would occur)
`where baseConvert(substring(storagelayerarcid, 8)) > 1000`	Finds all resources where the decimal value of property storagelayerarcid is bigger than 1000; the first 8 signs are omitted (otherwise, an overflow would occur)

13.2. Mathematical Functions

There are 15 mathematical functions. Note that a mathematical function always returns a defined value, as null values are replaced by the neutral element of the mathematical function, e.g. 0 for the plus function or 1 for the product function.

13.2.1. Plus

NQL	`plus`, `sum` or `+*`
Return type	Double or Long, depending on type of parameters
Parameters	`operands···` arbitrary number of operands of type Integer, Long or Double
Available since	7.1
Description	Summates given operands.

Examples

NQL	Description
`select plus(1, 2, 3)`	Returns the sum of given operands, which is 6

NQL	Description
`select plus( (1, 2, 3), 7, (5, 2) )`	Returns the sum of given operands, which is a list with entries 13, 11 and 10

13.2.2. Minus

NQL	`minus` or `-*`
Return type	Double or Long, depending on type of parameters
Parameters	`minuend` of type Integer, Long or Double
	`subtrahends…` arbitrary number of operands of type Integer, Long or Double
Available since	7.1
Description	Subtracts given subtrahends from given minuend.

Examples

NQL	Description
`select minus(10, 3, 5)`	Returns 2

13.2.3. Product

NQL	`product`, `times` or `**`
Return type	Double or Long, depending on type of parameters
Parameters	`operands…` arbitrary number of operands of type Integer, Long or Double
Available since	7.1
Description	Multiplies given operands.

Examples

NQL	Description
`where product(i1, i2) > 10`	Finds all resources where the product of integer properties i1 and i2 is bigger than 10
`orderby product(i1, i2)`	Sorts by product of integer properties i1 and i2

13.2.4. Quotient

NQL	`quotient`, `div` or `/`

Return type	Double
Parameters	`dividend` of type Integer, Long or Double
	`divisor` of type Integer, Long or Double
Available since	7.1
Description	Divides given dividend by given divisor.

Examples

NQL	Description
`select div(1, 2)`	Returns 0.5

13.2.5. Modulo

NQL	`modulo`, `mod` or `%`
Return type	Double or Long, depending on type of parameters
Parameters	`dividend` of type Integer, Long or Double
	`divisor` of type Integer, Long or Double
Available since	7.1
Description	Calculates remainder when integer dividing given dividend by given divisor.

Examples

NQL	Description
`select mod(10, 3)`	Returns the remainder of integer dividing 10 by 3 which is 1

13.2.6. Sqrt

NQL	`sqrt` or `√`
Return type	Double
Parameters	`operand` of type Integer, Long or Double
Available since	7.1
Description	Calculates square root of given operand.

Examples

NQL	Description
`select sqrt(9)`	Returns 3

13.2.7. Pow

NQL	`pow` or `^`
Return type	Double
Parameters	`base` of type Integer, Long or Double
	`exponent` of type Integer, Long or Double
Available since	7.1
Description	Calculates the exponential power of given base and given exponent.

Examples

NQL	Description
`select pow(9, 0.5)`	This is equivalent to the former example with function sqrt
`select pow(2, 5)`	Returns 32

13.2.8. Log

NQL	`log`
Return type	Double
Parameters	`operand` of type Integer, Long or Double
	`base` of type Integer, Long or Double (optional, default 10)
Available since	7.12
Description	Calculates the logarithm of given operand and given base.

Examples

NQL	Description
`select log(64, 2)`	Returns 6
`select log(1000)`	Returns 3

13.2.9. Floor

NQL	`floor`
Return type	Long
Parameters	`operand` of type Integer, Long or Double
Available since	7.1
Description	Rounds down given operand.

Examples

NQL	Description
`select floor(5.638)`	Returns 5

13.2.10. Ceiling

NQL	`ceil` or `ceiling`
Return type	Long
Parameters	`operand` of type Integer, Long or Double
Available since	7.5
Description	Rounds up given operand.

Examples

NQL	Description
`select ceil(5.638)`	Returns 6

13.2.11. Round

NQL	`round`
Return type	Double or Long, depending on given precision
Parameters	`operand` of type Integer, Long or Double
	`precision` of type Integer or Long (optional, default 0)
Available since	7.5
Description	Rounds given operand with given precision. A precision of 0 rounds to an integer value; a positive value rounds the decimal part, while a negative value rounds the integer part.

Examples

NQL	Description
`select round(5.638)`	Returns 6
`select round(5.638, 1)`	Returns 5.6
`select round(5.638, 2)`	Returns 5.64
`select round(5.638, -1)`	Returns 10

13.2.12. Abs

NQL	`abs`
Return type	Double or Long, depending on type of operand
Parameters	`operand` of type Integer, Long or Double
Available since	7.1
Description	Eliminates the leading sign of given operand.

Examples

NQL	Description
`select abs(-7)`	Returns 7

13.2.13. Signum

NQL	`signum` or `sign`
Return type	Integer
Parameters	`operand` of type Integer, Long or Double
Available since	7.5
Description	Returns sign of given operand, which is -1 for negative values and 1 for positive values, else 0.

Examples

NQL	Description
`select sign(-7)`	Returns -1

13.2.14. Sinus

NQL	`sinus` or `sin`
Return type	Double
Parameters	`operand` of type Integer, Long or Double
Available since	7.5
Description	Calculates the sinus of given operand.

Examples

NQL	Description
`select sin(3.14159265358)`	Returns 0

13.2.15. Avg

NQL	`avg`
Return type	Double
Parameters	`operands…` arbitrary number of operands of type Integer, Long or Double
Available since	7.3
Description	Calculates the average of given operands. Note that null values are ignored.

Examples

NQL	Description
`select avg(1, null, 4)`	Returns 2.5 (null values are ignored)

13.3. String Functions

There are 20 string functions. These functions operate on query operands of type string.

13.3.1. Concat

NQL	`concat` or `||`
Return type	String
Parameters	`operands…` arbitrary number of operands of type String
Available since	7.1

Description	Concats given operands. A null operand behaves like an empty string.

Examples

NQL	Description
`select ||(displayname, '.', fileextension)`	Returns the concatenation of property displayname, a dot and property fileextension

13.3.2. Substring

NQL	`substring`, `substr` or `§[]`
Return type	String
Parameters	`operand` of type String
	`start` of type Integer or Long, must not be a list
	`len` of type Integer or Long (optional, default is till the end of string), must not be a list
Available since	7.1
Description	Returns a substring within a string from given starting position, optionally with given length. The index is 0-based.

Examples

NQL	Description
`where substr(displayname, 0, 1) = 'A'`	Finds all resources whose displayname begins with an A
`select substr(displayname, 1)`	Cuts off first character from displayname

13.3.3. Instring

NQL	`instring`, `instr` or `§?`
Return type	Integer
Parameters	`operand` of type String
	`searchString` of type String, must not be a List
	`fromIndex` of type Integer or Long (optional, default is 0), must not be a list
	`lastIndex` of type Boolean (optional, default is false), must not be a list
Available since	7.1

Description	Identifies the position of a substring within a string, optionally with given starting position and/or indication whether last occurrance is searched for. The index is 0-based. If substring does not exist, -1 is returned. When lastIndex is true, this function is not searchable and not sortable.

Examples

NQL	Description
`select instr(displayname, 'a')`	Returns the position of first character a in displayname
`select instr(displayname, 'a', 3)`	Returns the position of first character a in displayname, beginning at the fourth character (index is 0-based)
`select instr(displayname, 'a', 0, true)`	Returns the position of last character a in displayname, beginning at the first character (index is 0-based)

13.3.4. Replace

NQL	`replace` or `→`
Return type	String
Parameters	`operand` of type String
	`target` (substring to be replaced) of type String, must not be a list
	`replacement` of type String, must not be a list
	`regularExpressionPolicy` of type Integer or Long (optional, default null), must not be a list
Available since	7.1, third and fourth parameter since 7.5
Description	Replaces a substring within a string by another string. The substring to be replaced (target) may be a regular expression, in this case a regular expression policy must be specified which can be 0 for policy replace first or 1 for policy replace all. When a regular expression policy is specified, this function is not searchable and not sortable.

Examples

NQL	Description
`select replace(displayname, 'xx', 'yyy')`	Replaces characters xx by characters yyy in displayname
`select replace(displayname, '[0-9]', 'x', 1)`	Replaces all numbers in displayname by character x

NQL	Description
`select replace(displayname, '[0-9]', '', 0)`	Replaces first number in displayname by the empty string

13.3.5. ReplaceCharacters

NQL	`replaceChars`
Return type	String
Parameters	`operand` of type String
	`replace` of type String
	`replacement` of type String
Available since	7.12
Description	Replaces characters within a string by other characters. The position of the replace characters (the second parameter) correlates to the position of the replacement characters (the third parameter), e.g. the first character in the second parameter is replaced by the first character in the third parameter. The length of the replace characters should normally equal the length of the replacement characters. If the length of replace characters is longer, then the extra characters are deleted. If the length of replace characters is shorter, then the extra replacement characters are ignored.

Examples

NQL	Description
`select replaceChars(displayname, 'abc', 'xyz')`	Replaces all characters 'a' by 'x', all characters 'b' by 'y' and all characters 'c' by 'z' in displayname
`select replaceChars(displayname, '0123456789', '')`	Replaces all digits in displayname by the empty string, which means removes all digits from displayname
`select replaceChars('Three Chinese With a Double Bass', 'aeiou', 'iiiii')`	Returns Thrii Chinisi With i Diibli Biss
`select replaceChars('1,000.1', ',.', '.,')`	Returns 1.000,1

13.3.6. Trim

NQL	`trim` or `][`
Return type	String
Parameters	`operand` of type String

	`style` of type Integer or Long (optional, default 0), must not be a list
	`character` of type String (optional, default space character (blank)), must not be a list
Available since	7.1, second and third parameter since 7.5
Description	Returns given string operand trimmed. It is possible to execute function trim (0), ltrim (negative number) or rtrim (positive number) by specifying a style. Also the character to be trimmed may be specified. When a character different from blank is specified or style is not a constant, this function is not searchable and not sortable.

Examples

NQL	Description
`select trim(' abc ')`	Returns abc without leading and trailing blanks
`select trim(' abc ', -1)`	Returns abc without leading blanks
`select trim(' abc ', 1)`	Returns abc without trailing blanks
`select trim('aaaHello Worldaa', 0, 'a')`	Returns Hello World

13.3.7. ToLower

NQL	`toLower` or `low`
Return type	String
Parameters	`operand` of type String
	`locale` of type String (optional, default null), must not be a list
Available since	7.1, second parameter since 7.5
Description	Returns given string operand in lower case. It is also possible to specify a locale. When a locale is specified, this function is not searchable and not sortable.

Examples

NQL	Description
`select low('Hello World')`	Returns hello world

13.3.8. ToUpper

NQL	`toUpper` or `up`
Return type	String

Parameters	`operand` of type String
	`locale` of type String (optional, default null), must not be a list
Available since	7.1, second parameter since 7.5
Description	Returns given string operand in upper case. It is also possible to specify a locale. When a locale is specified, this function is not searchable and not sortable.

Examples

NQL	Description
`select up('Hello World')`	Returns HELLO WORLD

13.3.9. Capitalize

NQL	`capitalize`
Return type	String
Parameters	`operand` of type String
Available since	7.12
Description	Returns given string operand where all words are capitalized, which means the first character of all words is in upper case while all following characters are in lower case. A new word is identified by a delimiter which is a blank or a special character.

Examples

NQL	Description
`select capitalize('hello woRLD')`	Returns Hello World
`select capitalize('abc-def/ghi')`	Returns Abc-Def/Ghi

13.3.10. Reverse

NQL	`reverse`
Return type	String
Parameters	`operand` of type String
Available since	7.5
Description	Reverses given operand.

Examples

NQL	Description
`select reverse('abc')`	Returns cba

13.3.11. Replicate

NQL	`replicate`
Return type	String
Parameters	`operand` of type String
	`number` of type Integer or Long
Available since	7.5
Description	Replicates given operand for specified times.

Examples

NQL	Description
`select replicate('please ', 3)`	Returns please please please

13.3.12. Pad

NQL	`pad`
Return type	String
Parameters	`operand` of type String
	`length` of type Integer or Long
	`style` of type Integer or Long (optional, default -1 (lpad-style))
	`pad` of type String (optional, default ' ' (blank))
Available since	7.12
Description	Pads given string operand. The length parameter determines the length of returned string, while the pad parameter determines the string which is padded to given operand. The style operand determines, whether the operand is padded on the left or right hand side. A positive number results in a right padded operand, otherwise the operand is left padded.

Examples

NQL	Description
`select pad('123', 7, 0, '0')`	Returns 0000123

NQL	Description
`select pad('123', 5, 1, '+')`	Returns 123++

13.3.13. Base64

NQL	`base64`
Return type	String
Parameters	`operand` of type String
	`encode` of type Boolean (optional, default true (encoding))
	`urlEncoded` of type Boolean (optional, default false)
Available since	7.14, third parameter since 7.16
Description	Encodes or decodes given string operand to respectively from base 64 encoding. It is also possible to specify, whether base 64 code is url-encoded. This function is not searchable and not sortable.

Examples

NQL	Description
`select base64(displayname)`	Returns base 64 encoded displayname
`select base64('SGVsbG8=', false)`	Returns Hello
`select base64('SGVsbG8%3d', false, true)`	Returns Hello

13.3.14. Hash

NQL	`hash`
Return type	String
Parameters	`operand` of type String
	`algorithm` of type Integer or Long (optional, default 0 (MD5))
Available since	7.14
Description	Executes given hash algorithm on given string operand. The algorithm must be a numeric value between 0 and 4, where 0 represents MD5, 1 represents SHA-1, 2 represents SHA-256, 3 represents SHA-384 and 4 represents SHA-512. This function is not searchable and not sortable.

Examples

NQL	Description
`select hash(displayname)`	Returns MD5 hash of property displayname
`select hash('Hello World', 2)`	Returns a591a6d40bf420404a011733cfb7b190d62c65bf0bcda32b57b277d9ad9f146e

13.3.15. StringFormat

NQL	`stringFormat`
Return type	String
Parameters	`locale` of type String
	`format` of type String
	`parameters…` arbitrary number of parameters of any type
Available since	7.7
Description	Formats the given string format operand in given locale, using given parameters. The string is formatted by using static `format` method in `java.lang.String` class. The locale must be a string in ISO639 format, e.g. 'en_US' or 'de'. If locale is null the default locale takes effect which is the locale of current session if provided, else the locale of the default dictionary if existing, else the default language of the jvm. This function is not searchable and not sortable.

Examples

NQL	Description
`stringFormat(null, '%07d', 123)`	Returns 0000123 (seven-digit number with leading zeros)
`stringFormat('de', '%,d', contentlength)`	Returns property contentlength containing thousands separator in German style
`stringFormat('de', '%,d KB', round(div(contentlength, 1024)))`	Returns property contentlength in KB in German style
`stringFormat(%clientLocale, '%,.2f MB', div(contentlength, ^(1024, 2)))`	Returns property contentlength in MB in style of current locale, rounded to two decimal digits
`stringFormat('en', '%,.3f', myDoubleProperty)`	Returns property myDoubleProperty containing thousands and decimal separator in English style, rounded to three decimal digits

13.3.16. Soundex

NQL	`soundex`
Return type	String
Parameters	`operand` of type String
Available since	7.9
Description	Returns the soundex-code of given operand. The soundex function can be used to search for similar sounding entries. It returns a four character string containing the phonetic representation of its input parameter. Words that are spelled differently, but sound alike, should match to the same return value, which makes it possible to perform a similarity search. Note that the soundex algorithm was invented for the English language but should also produce good results in related languages like German.

Examples

NQL	Description
`where soundex(displayname) = soundex('Smiths')`	Finds resources where displayname sounds like Smiths

13.3.17. SoundexDifference

NQL	`soundexDifference`
Return type	Integer
Parameters	`lhsOperand` of type String
	`rhsOperand` of type String
Available since	7.9
Description	Returns the soundex difference between given operands. The soundex function returns a four character string containing the phonetic representation of its input parameter. The soundex difference function compares two results of a soundex function, for each character that is identical the return value of the soundex difference function is increased by one. Thus, the returned value is a number between 0 and 4 where 4 is the highest similarity. Note that the soundex algorithm was invented for the English language but should also produce good results in related languages like German.

Examples

NQL	Description
`where soundexDifference(displayname, 'Smiths') > 2`	Finds resources with a soundex difference of 3 or 4

13.3.18. EditDistance

NQL	`editDistance`
Return type	Integer
Parameters	`lhsOperand` of type String
	`rhsOperand` of type String
	`caseInsensitive` of type Boolean (optional, default false), must not be a list
Available since	7.10, third parameter since 7.12
Description	Returns the number of steps which are necessary to convert one string into another by using the Levenshtein algorithm, so it can be used to search for entries with a specified similarity. It is possible to specify, whether comparison is case sensitive or not. When third parameter is specified, this function is not searchable and not sortable.

Examples

NQL	Description
`where editDistance(displayname, 'Smiths') <= 2`	Finds resources with an edit distance of 0, 1 or 2

13.3.19. Similarity

NQL	`similarity`
Return type	Double
Parameters	`lhsOperand` of type String
	`rhsOperand` of type String
	`caseInsensitive` of type Boolean (optional, default false), must not be a list
Available since	7.10, third parameter since 7.12

Description	Returns a value between 0 and 1 where 0 means the two given strings are totally different and 1 means the two given strings are identical. It is possible to specify, whether comparison is case sensitive or not. The algorithm to determine similarity of two strings is: `1 - editDistance(string1, string2) / max(len(string1), len(string2))`. This function can be used to search for entries with a specified similarity. Note that for PostgreSQL an internal similarity algorithm can be used instead of the Levenshtein algorithm (see configuration of DbSetting) to increase performance. When third parameter is specified, this function is not searchable and not sortable.

Examples

NQL	Description
`where similarity(displayname, 'Smiths') >= 0.6`	Finds resources with a similarity between 0.6 and 1

13.3.20. Cologne Phonetic

NQL	`cologne Phonetic`
Return type	String
Parameters	`operand` of type String
Available since	7.10
Description	Returns a string containing the phonetic representation of its input parameter. Words that are spelled differently, but sound alike, should match to the same return value, which makes it possible to perform a similarity search. Note that in contrast to the soundex function, the cologne phonetic function is optimized for the German language. This function is not searchable and not sortable (it may be used for a similarity search though by defining appropriate value properties).

Examples

NQL	Description
`select cologne Phonetic(displayname)`	Returns the cologne phonetic code of property displayname

13.4. Date/Timestamp Functions

There are 12 date/timestamp functions. These functions operate on query operands of type date or timestamp.

13.4.1. Year

NQL	`year`
Return type	Integer
Parameters	`operand` of type Date or Timestamp
Available since	7.1
Description	Extracts the year from given date/datetime operand.

Examples

NQL	Description
`select year(initialcreation)`	Returns the year of creation of a resource
`where year(creationdate) = year(%today)`	Finds resources where current version has been created in current year

13.4.2. Month

NQL	`month`
Return type	Integer
Parameters	`operand` of type Date or Timestamp
Available since	7.1
Description	Extracts the month from given date/datetime operand.

Examples

NQL	Description
`select month(initialcreation)`	Returns the month of creation of a resource as a value between 1 and 12

13.4.3. Day

NQL	`day`
Return type	Integer
Parameters	`operand` of type Date or Timestamp
Available since	7.1
Description	Extracts the day from given date/datetime operand.

Examples

NQL	Description
`select day(initialcreation)`	Returns the day of creation of a resource as a value between 1 and 31

13.4.4. Hour

NQL	`hour`
Return type	Integer
Parameters	`operand` of type Date or Timestamp
Available since	7.1
Description	Extracts the hour from given date/datetime operand.

Examples

NQL	Description
`select hour(%now)`	Returns the hour of current time as a value between 0 and 23

13.4.5. Minute

NQL	`minute`
Return type	Integer
Parameters	`operand` of type Date or Timestamp
Available since	7.1
Description	Extracts the minute from given date/datetime operand.

Examples

NQL	Description
`select minute(%now)`	Returns the minute of current time as a value between 0 and 59

13.4.6. Second

NQL	`second`
Return type	Integer

Parameters	`operand` of type Date or Timestamp
Available since	7.1
Description	Extracts the second from given date/datetime operand.

Examples

NQL	Description
`select second(%now)`	Returns the second of current time as a value between 0 and 59

13.4.7. DateExtract

NQL	`dateExtract`
Return type	Integer
Parameters	`operand` of type Date or Timestamp
	`extract` of type Integer or Long, must not be a list
Available since	7.3
Description	Extracts specified part from given date/datetime operand. The given extract must be a numeric value between 0 and 5 where every number represents a different part of the date/datetime operand. A value 0 extracts the day of the week, e.g. in Germany for Monday the value 1 is returned. A value 1 extracts the day of the week in a month, e.g. value 2 is returned for the second Monday of a month. A value 2 extracts the day of the year, that is the number of days which have passed since the beginning of the year. A value 3 extracts the week of the month, e.g. value 1 is returned for the first week of a month. A value 4 extracts the week of the year, e.g. value 2 is returned for the second week of a year. A value 5 extracts the quarter of the year, e.g. value 3 is returned for months July, August and September. When parameter extract is not a constant, this function is not searchable and not sortable.

Examples

NQL	Description
`select dateExtract(%today, 0)`	Returns the day of the week as a value between 1 and 7
`select dateExtract(%today, 1)`	Returns the day of the week in a month as a value between 1 and 5
`select dateExtract(%today, 2)`	Returns the day of the year as a value between 1 and 366

NQL	Description
`select dateExtract(%today, 3)`	Returns the week of the month as a value between 1 and 5
`select dateExtract(%today, 4)`	Returns the week of the year as a value between 1 and 53
`select dateExtract(%today, 5)`	Returns the quarter of the year as a value between 1 and 4
`where dateExtract(initialcreation, 5) = 1`	Finds resources which have been created in the first quarter

13.4.8. DateRound

NQL	`dateRound`
Return type	Date or Timestamp
Parameters	`date` of type Date or Timestamp
	`roundingPolicy` of type Integer or Long
	`businessCalendar` of type String (optional, default null)
Available since	7.10
Description	Rounds given date/datetime by using specified rounding policy. The rounding policy is represented by a numeric value. If rounding policy is 0 or null, the returned date is not altered. If rounding policy is a negative number, the returned date is rounded down. If rounding policy is a positive number, the returned date is rounded up. The absolute number of the rounding policy determines the kind of rounding, it must lie between 1 and 8, representing rounding by year, month, week, day, hour, minute, second or quarter. When rounding up and the given date/datetime is at the beginning of specified rounding part, the returned date is unaltered, e.g. rounding up by year on January 1st returns given date/datetime unaltered. Note that it is also possible to specify a business calendar used for rounding. When a business calendar is specified, only rounding by day is supported, so a negative rounding policy would result in rounding down to the beginning of the business day and a positive rounding policy would result in rounding up to the end of the business day. This function is not searchable and not sortable.

Examples

NQL	Description
`select dateRound(%today, 1)`	Rounding up current date to the beginning of the next year

NQL	Description
`select dateRound(%today, -1)`	Rounding down current date to the beginning of current year
`select dateRound(%today, 2)`	Rounding up current date to the beginning of the next month
`select dateRound(%today, -2)`	Rounding down current date to the beginning of current month
`select dateRound(%today, 3)`	Rounding up current date to the beginning of the next week
`select dateRound(%now, 4)`	Rounding up current datetime to the beginning of the next day
`select dateRound(%now, 5)`	Rounding up current datetime to the beginning of the next hour
`select dateRound(%now, 6)`	Rounding up current datetime to the beginning of the next minute
`select dateRound(%now, 7)`	Rounding up current datetime to the beginning of the next second
`select dateRound(%today, 8)`	Rounding up current date to the beginning of the next quarter
`select dateRound(%now, -1, 'NordrheinWestfalen')`	Rounding down current datetime to the beginning of current business day, regarding business calendar with name 'NordrheinWestfalen'

13.4.9. DateFormat

NQL	`dateFormat`
Return type	String
Parameters	`operand` of type Date or Timestamp
	`format` of type String, must not be a list
	`locale` of type String (optional, default null), must not be a list
Available since	7.1, third parameter since 7.3
Description	Formats given date/datetime operand in given string format. If format is null the default format 'yyyy-MM-dd HH:mm:ss' takes effect. This function is not searchable and not sortable.

Examples

NQL	Description
`select dateFormat(initialcreation,'dd. MMMM yyyy')`	Returns date of initial creation in given format
`select dateFormat(initialcreation,'EEEE','fi')`	Returns the name of the day of initial creation in Finnish language

13.4.10. LocaleDateFormat

NQL	`localeDateFormat`
Return type	String
Parameters	`operand` of type Date or Timestamp
	`dateStyle` of type Integer or Long (optional, default 2), must not be a list
	`timeStyle` of type Integer or Long (optional, default 2), must not be a list
	`locale` of type String (optional, default null), must not be a list
Available since	7.1
Description	Formats given date/datetime operand in given style and locale. The style must be a numeric value between 0 and 5, where 0 means full style, 1 means long style, 2 means medium style and 3 means short style. Style 4 returns the name of the day and style 5 returns the name of the month. The locale must be a string in ISO639 format, e.g. 'en_US' or 'de'. If style and/or locale are null default values take effect. The default value for date and time style is 2 which means medium style. The default value for locale is the locale of current session if provided, else the locale of the default dictionary if existing, else the default language of the jvm. Note that when time style is specified, the locale must also be specified. This function is not searchable and not sortable.

Examples

NQL	Description
`select localeDateFormat(initialcreation)`	Returns date of initial creation in default format (medium style) and in default language
`select localeDateFormat(initialcreation, 0, 'fr')`	Returns date of initial creation in full style in French language
`select localeDateFormat(initialcreation, 0, 2, 'fi')`	Returns date of initial creation in full date style and medium time style in Finnish language
`select localeDateFormat(initialcreation, 4, 'en')`	Returns the name of the day of initial creation in English language

13.4.11. DateAdd

NQL	`dateAdd`
Return type	Date or Timestamp
Parameters	`date` of type Date or Timestamp
	`summand` of type Integer or Long
	`datePart` of type Integer or Long (optional, default 2), must not be a list
	`businessCalendar` of type String (optional, default null), must not be a list
	`roundingPolicy` of type Integer or Long (optional, default null), must not be a list, see also DateRound function
Available since	7.2, fourth parameter since 7.5, fifth parameter since 7.9
Description	Adds a number to given date part of given date. The date part must be a numeric value between 0 and 5, representing part year, month, day, hour, minute or second. If date part is null the default date part day (represented by value 2) takes effect. If an optional business calendar is specified, only business times and dates are considered. It is also possible to specify a rounding policy. When a business calendar and/or rounding policy is specified, this function is not searchable and not sortable.

Examples

NQL	Description
`select dateAdd(%today, 5, 1)`	Returns a date where 5 months are added to the current date
`select dateAdd(initialcreation, -7)`	Returns a datetime where 7 days are subtracted from the date of initial creation
`select dateAdd(creationdate, 5, 3, 'NordrheinWestfalen')`	Returns a datetime where 5 business hours are added to creation date, regarding business calendar with name 'NordrheinWestfalen'
`where initialcreation >= dateAdd(%today, -7)`	Finds resources which have been created in the last seven days

13.4.12. DateDiff

NQL	`dateDiff`
Return type	Long
Parameters	`startDate` of type Date or Timestamp
	`endDate` of type Date or Timestamp
	`datePart` of type Integer or Long (optional, default 2), must not be a list

	`businessCalendar` of type String (optional, default null), must not be a list
Available since	7.2, fourth parameter since 7.5
Description	Calculates the difference between two dates concerning given date part. The date part must be a numeric value between 0 and 5, representing part year, month, day, hour, minute or second. If date part is null the default date part day (represented by value 2) takes effect. If an optional business calendar is specified, only business times and dates are considered. When a business calendar is specified, this function is not searchable and not sortable.

Examples

NQL	Description
`select dateDiff(initialcreation, %today, 1)`	Returns the difference in months between the date of initial creation and the current date
`select dateDiff(initialcreation, archivedate)`	Returns the difference in days between the date of initial creation and the archive date
`select dateDiff(initialcreation, creationdate, 2, 'NordrheinWestfalen')`	Returns the difference in business days between the date of initial creation and the creation date of the current version, regarding business calendar with name 'NordrheinWestfalen'
`where dateDiff(initialcreation, creationdate) > 10`	Finds resources where the difference of the date of initial creation and the creation date of the current version is greater than 10 days

13.5. List Functions

There are 12 list functions. These functions operate on multi-value query operands.

13.5.1. ConcatLists

NQL	`concatLists` or `++`
Return type	List of any type, depending on type of parameters
Parameters	`lists…` arbitrary number of lists of any (but all the same) type
Available since	7.1
Description	Concatenates all list operands. The resulting list contains all entries of the given lists. This function is not searchable and not sortable.

Examples

NQL	Description
`select ++(%currentGroupPrincipalIds, %currentPositionPrincipalIds, (%currentUserPrincipalId))`	Returns a list which contains the group ids, the position ids and the user id of the current principal; note that the context variable *%currentUserPrincipalId* must be embraced by brackets to cast the single value to a list
`where exists(assignedpooledorgentityid in ++(%currentGroupPrincipalIds, %currentPositionPrincipalIds))`	Finds workflow instances where at least one of current principal's positions or groups is in assigned pool

13.5.2. ListEntry

NQL	`listEntry`, `entry` or `[]`
Return type	Any type, depending on type of first parameter
Parameters	`listOperand` of type list of any type
	`listIndex` of type Integer or Long
Available since	7.1
Description	Extracts an entry from given list operand. The list index is 0-based. If list index does not exist, null is returned. When used in the where clause, this function must not be nested in other functions.

Examples

NQL	Description
`select listEntry(%currentGroupPrincipalIds, 0)`	Returns the id of the first group of current principal
`where entry(itemcontenttype, 0) = 'text/plain'`	Finds resources where first content item is of type text/plain

13.5.3. ListIndex

NQL	`listIndex` or `[]?`
Return type	Integer
Parameters	`listOperand` of type list of any type
	`listEntry` of any type, corresponding to type of list operand
Available since	7.1

Description	Identifies the (first) index of given entry from given list operand. The returned list index is 0-based. If list entry does not exist, -1 is returned. When used in the where clause, this function must not be nested in other functions.

Examples

NQL	Description
`select listIndex(('a','b','c'), 'b')`	Returns 1

13.5.4. ListLength

NQL	`listLength` or `[]!`
Return type	Integer
Parameters	`listOperand` of type list of any type
Available since	7.1
Description	Returns the size of given list operand. When used in the where clause, this function must not be part of a case condition.

Examples

NQL	Description
`select listLength(%currentGroupPrincipalIds)`	Returns the number of groups of current principal
`where listLength(storagelayerarcid) > 0`	Finds resources which reference an *nscale Server Storage Layer* document

13.5.5. Sublist

NQL	`sublist`
Return type	List of any type, depending on type of first parameter
Parameters	`listOperand` of type list of any type
	`start` of type Integer or Long, must not be a list
	`len` of type Integer or Long (optional, default till end of list), must not be a list
Available since	7.1
Description	Returns a sublist of given list operand, determined by given starting index and optionally given length. The index is 0-based. When used in the where clause, this function must not be nested in other functions.

Examples

NQL	Description
`select sublist(('a','b','c','d','e'), 2)`	Returns a list with entries 'c', 'd' and 'e'
`select sublist(('a','b','c','d','e'), 2, 2)`	Returns a list with entries 'c' and 'd'
`where exists(sublist(itemcontenttype, 1) = 'text/plain')`	Finds resources with a content item of type text/plain, first content item omitted

13.5.6. ListEntries

NQL	`listEntries` or `entries`
Return type	List of any type, depending on type of parameter
Parameters	`listOperand` of type list of any type
Available since	7.3
Description	May be used as parameter of a function with an arbitrary number of parameters, which means this function must always be nested. The list entries function provides all entries of given list operand. This function is not searchable and not sortable.

Examples

NQL	Description
`select plus(entries(itemlength))`	Returns the sum of all entries of multi-value property itemlength
`select max(entries(itemlength))`	Returns the maximum value of all entries of multi-value property itemlength
`select avg(entries(itemlength))`	Returns the average value of all entries of multi-value property itemlength

13.5.7. ToFlat

NQL	`toFlat` or `_`
Return type	String
Parameters	`listOperand` of type list of any type
	`delimiter` of type String (optional, default ; (semicolon plus blank))
	`lastDelimiter` of type String (optional, default null)
Available since	7.1, third parameter since 7.12

Description	Concatenates the entries of given list operand to a string with given delimiter. If delimiter is omitted, the string '; ' is used as default delimiter. If parameter lastDelimiter is provided, the last two entries are delimited by given string. This function is not searchable and not sortable.

Examples

NQL	Description
`select _(itemlength)`	Returns the length of all content items, delimited by '; '
`select toFlat(%currentGroupPrincipalIds, '#')`	Returns the ids of the groups of current principal, delimited by '#'
`select toFlat(pidResolve(%currentGroupPrincipalIds), ', ', ' and ')`	Returns the names of the groups of current principal, delimited by ', ' respectively ' and ' for the last two entries

13.5.8. Tokenize

NQL	`tokenize`
Return type	List of String when operand is of type String, List of Integer when operand is of type Blob
Parameters	`operand` of type String or Blob
	`delimiter` of type String (optional, default ; (semicolon)), must not be a list, must not be specified when operand is of type Blob
Available since	7.3
Description	Tokenizes given operand by given delimiter to a list. If delimiter is omitted, the string ';' is used as default delimiter. This function is not searchable and not sortable.

Examples

NQL	Description
`select tokenize(%clientIPAddress, '.')`	Returns a list whose entries are the parts of the ip address of current client

13.5.9. UniqueList

NQL	`uniqueList`
Return type	List of any type, depending on type of first parameter

Parameters	`listOperand` of type list of any type
	`excludeNull` of type Boolean (optional, default false), must not be a list
Available since	7.8
Description	Eliminates duplicate entries from a list. It is also possible to specify, whether null values are excluded from the list. This function is not searchable and not sortable.

Examples

NQL	Description
`select uniqueList( ('red', 'yellow', 'red') )`	Returns a list with the two entries 'red' and 'yellow'

13.5.10. SortList

NQL	`sortList`
Return type	List of any type, depending on type of first parameter
Parameters	`listOperand` of type list of any type except blob
	`ascending` of type Boolean (optional, default true), must not be a list
	`nullPosition` of type Integer or Long (optional, default 1), must not be a list
	`caseInsensitive` of type Boolean (optional, default false), must not be a list
Available since	7.8, fourth parameter since 7.12
Description	Sorts given list. It is possible to specify, whether list is sorted ascending or descending and whether sorting is case sensitive or not. It is also possible to specify, where null values are positioned, e.g. at the end or at the beginning of the list or whether null values are filtered. Positive values determine that null values are considered as greatest values. Negative values determine that null values are considered as smallest values. A value of 0 determines that null values aren't sorted (they are omitted). This function is not searchable and not sortable.

Examples

NQL	Description
`select sortList(itemlength)`	Returns multi-value property itemlength sorted in ascending order
`select sortList(itemlength, false)`	Returns multi-value property itemlength sorted in descending order

NQL	Description
`select sortList(itemlength, true, 1)`	Returns multi-value property itemlength sorted in ascending order where null values are positioned at the end of the list

13.5.11. ReduceList

NQL	`reduceList`
Return type	List of any type, depending on type of first parameter
Parameters	`listOperand` of type list of any type
	`excludeOperands…` arbitrary number of operands of the same type as the list entries, none of these operands must be a list
Available since	7.8
Description	Eliminates given operands from given list. This function is not searchable and not sortable.

Examples

NQL	Description
`select reduceList(('red', 'blue', 'green'), 'red', 'yellow')`	Returns a list with the two entries 'blue' and 'green'
`select reduceList(itemlength, entries(itemlength))`	Returns an empty list (all entries from multi-value property itemlength are eliminated from itself)

13.5.12. ToList

NQL	`toList`
Return type	List of any type, depending on type of parameters
Parameters	`operands…` arbitrary number of operands of any (but all the same) type
Available since	7.13
Description	Returns a list which contains all given operands. This function is not searchable and not sortable.

Examples

NQL	Description
`select toList(1, 2, 3)`	Returns a list with the three entries 1, 2 and 3

NQL	Description
`select toList('a', 'b', 'c')`	Returns a list with the three entries a, b and c

13.6. Other Functions

There are 27 more functions which could not be categorized in the sections above.

13.6.1. Null

NQL	`null`
Return type	Null
Parameters	
Available since	7.1
Description	The null function always returns null. Note that the null function is the only function where brackets may be omitted.

Examples

NQL	Description
`select null`	Returns null
`select toLower('Hello World', null)`	Returns hello world

13.6.2. NullIf

NQL	`nullIf`
Return type	Type of left hand side parameter
Parameters	`lhsOperand` of type Integer, Long, Double, Boolean, Date, Timestamp or String
	`rhsOperand` of type Integer, Long, Double, Boolean, Date, Timestamp or String
Available since	7.13
Description	Returns null, if the two parameters are equal, else the left hand side parameter is returned.

Examples

NQL	Description
`select nullIf(1, 1)`	Returns null
`select nullIf(displayname, 'Hello World')`	Returns null, if displayname equals 'Hello World', else the displayname

13.6.3. Coalesce

NQL	`coalesce`
Return type	Any type, depending on type of parameters
Parameters	`operands···` arbitrary number of operands of any (but all the same) type
Available since	7.1
Description	Returns the first operand which is not null.

Examples

NQL	Description
`select coalesce(usercommonname, username, displayname)`	Returns usercommonname if not null, else username if not null, else displayname

13.6.4. Length

NQL	`length`, `len` or `!`
Return type	Integer
Parameters	`operand` of type Integer, Long, Double, Boolean, String or Blob
Available since	7.1
Description	Returns the length of given operand.

Examples

NQL	Description
`select len(displayname)`	Returns the number of characters of the displayname
`where len(displayname) > 10`	Finds resources where the length of the displayname is greater than 10

13.6.5. Max

NQL	`max`
Return type	Any type, depending on type of parameters
Parameters	`operands…` arbitrary number of operands of type Integer, Long, Double, Boolean, Date, Timestamp or String
Available since	7.3
Description	Returns the maximum value of an arbitrary number of operands. An operand which evaluates to null is ignored. If all operands are null, null is returned. Note that all operands must be of same type.

Examples

NQL	Description
`select max(10, 3, 2)`	Returns 10

13.6.6. Min

NQL	`min`
Return type	Any type, depending on type of parameters
Parameters	`operands…` arbitrary number of operands of type Integer, Long, Double, Boolean, Date, Timestamp or String
Available since	7.3
Description	Returns the minimum value of an arbitrary number of operands. An operand which evaluates to null is ignored. If all operands are null, null is returned. Note that all operands must be of same type.

Examples

NQL	Description
`select min(10, 3, 2)`	Returns 2

13.6.7. Case

NQL	`case` or `|=`
Return type	Any type, depending on type of parameters
Parameters	`condition` to be evaluated
	`operandTrue` of any type

	`operandFalse` of any type
Available since	7.1
Description	Evaluates a condition (see where clause) and returns either the first or the second operand, depending on the result of the condition. Note that the true and false operand must be of same type. When the case function is part of the where clause, the result of the function must not be a list.

Examples

NQL	Description
`select case(exists(colour = 'red'), 'I am red', 'I am not red')`	Returns 'I am red' if the multi-value property colour contains the entry 'red', else 'I am not red' is returned
`select case(int1 > 10 and int1 < 20, null, displayname)`	Returns null if user defined property int1 is greater than 10 and smaller than 20, else the display name is returned
`select case int1 > 10 and int1 < 20 then null else displayname`	Alternative representation of the statement above
`select |= int1 > 10 and int1 < 20 ? null : displayname`	Yet another alternative; note that for this notation brackets must be omitted

13.6.8. Switch

NQL	`switch` or `|<`
Return type	Any type, depending on type of parameters
Parameters	`switchOperand` of any type
	`switchOperator` operator (optional, default `=` (the equals-operator))
	`operands…` arbitrary number of operands
Available since	7.5

Description	Evaluates an expression and returns the first value for which the expression is true. The switch function contains three parameters: a switch operand, an optional operator and an array of operands. The switch operand is compared with other values. The operator defines the kind of comparison, the default operator is *equals*. The array of operands consists of tuples (ordered pairs), where the first operand of a tuple is compared with the switch operand and the second operand of a tuple is the value which is returned when a comparison returns true. If the first operand of a tuple is null, this is considered as the default tuple whose value is returned when no other expression returns true. The switch function is rather redundant to the case function, but it is more concise. Note that operator *between* is not supported. When the switch function is part of the where clause, the switch operand and the result of the function must not be a list.

Examples

NQL	Description
`select switch(resourcetype ? 1='Folder', 2='Document', 3='Link')`	Returns 'Folder', 'Document' or 'Link', depending on the type of resource
`select switch(fulltextstate >= ? 16='Fatal', 8='Error', 1='ToBeIndexed', 0='Indexed')`	Returns an aggregated fulltext state
`select switch(year(initialcreation) ? year(%today)='New', null='Old')`	Returns 'New' for resources which have been created in current year, else 'Old'
`select switch(displayname is null ? true='Empty', null=displayname)`	Returns 'Empty' if displayname is null, else the displayname
`select switch(hasnote ? true='I have a note', false='Sorry, no note')`	Returns indication whether resource has a note as text
`select switch(plus(R, L) ? J=true, null=false)`	Returns true if R + L = J, else false

13.6.9. Random

NQL	`random` or `rnd`
Return type	Double
Parameters	`factor` of type Integer or Long (optional, default 100)
	`offset` of type Integer or Long (optional, default 0)
Available since	7.1, first and second parameter since 7.5

Description	Returns a random number between 0 (inclusive) and 100 (exclusive) by default. The upper and lower border can be adjusted by specifying a factor and an offset. The default factor is 100, specifying another value will increase or decrease the upper border. The specified offset is subtracted from factor and added to the random number, e.g. a factor of 50 and an offset of 10 will return a random number between 10 (inclusive) and 50 (exclusive). This function is not searchable and not sortable.

Examples

NQL	Description
`select rnd()`	Returns a random number between 0 and 100
`select random(10)`	Returns a random number between 0 and 10
`select random(10, 5)`	Returns a random number between 5 and 10

13.6.10. Translate

NQL	`translate` or `l10n`
Return type	String
Parameters	`operand` of type String
	`locale` of type String (optional, default null), must not be a list
Available since	7.1
Description	Translates the given operand in given locale. The locale must represent an existing dictionary of the configuration, e.g. 'en_US' or 'de'. If the given locale does not exactly match the name of a dictionary, the next best dictionary is chosen, e.g. when locale 'de_AT' is requested, assuming there is no Austrian dictionary but there is a German dictionary 'de', then 'de' is taken as dictionary (and vice versa). If given locale is null the default locale takes effect which is the locale of current session if provided, else the locale of the default dictionary if existing, else the default language of the jvm. If such a dictionary does not exist, the operand is not translated and is returned unmodified. Otherwise the given operand is taken as a key value of the determined dictionary, the returned value is the corresponding translated phrase. The translate function is not sortable. When it contains any properties, its use in the where clause is restricted by several constraints, e.g. it cannot be used in a NOT-Operation or in a NULL-expression or when the condition contains any OR-operations or in an EXISTS-operation when in-operator is used. Note also that a translation must exist (i.e. an entry in requested dictionary) in order that a row can be found (no fallback to requested key).

Examples

NQL	Description
`select translate('Decision', 'de_AT')`	Returns 'Entscheidung' (the translation for key 'Decision' of dictionary 'de_AT' if existing, else 'de')

13.6.11. PrincipalIdResolver

NQL	`pidResolve`
Return type	String
Parameters	`operand` of type String
	`style` of type Integer or Long (optional, default 3), must not be a list
Available since	7.1, styles added in different versions
Description	Formats one or more principal ids in given style. The style must be a numeric value between 0 and 9, where 0 means name style (e.g. 'admin'), 1 means full name style (e.g. 'admin@nscale'), 2 means common name style (e.g. 'nscale Administrator') and 3 means common name style with fallback to name style. The latter one means, the common name is returned if it is not null, else the name is returned. Style 4 is the same as style 3 except that for a non existing principal id the principal id is returned instead of 'unknown'. Style 5 returns the domain name and style 6 returns the type of the principal. Style 7 returns the referenced principal which is the default position in case of a user and the associated user in case of a position. Also style 7 resolves a virtual principal. Style 8 resolves a full name to a principal id (priority is user before position before group). Style 9 resolves a virtual principal, for non-virtual principals the given value is returned unaffected. If style is null or an undefined value, the common name with fallback style takes effect as default. The principal id resolver function is not allowed in sort order items. When it contains any properties, its use in a condition is restricted by several constraints, e.g. it cannot be used in a NOT-Operation or in a NULL-expression or when the condition contains any OR-operations or in an EXISTS-operation when in-operator is used. Note also that deleted principals cannot be found.

Examples

NQL	Description
`select pidResolve(initialcreator)`	Returns the initial creator in default style
`select pidResolve(initialcreator, 1)`	Returns the initial creator in full name style

NQL	Description
`where pidResolve(%currentUserPrincipalId, 5) = 'nscale'`	Finds all resources when current principal belongs to domain 'nscale'
`where pidResolve('admin@nscale', 8) = %currentUserPrincipalId`	Finds all resources when current principal is 'admin@nscale'

13.6.12. GroupResolver

NQL	`groupResolve`
Return type	List of String
Parameters	`operand` of type String, must not be a list
	`flat` of type Boolean (optional, default false), must not be a list
	`activeOnly` of type Boolean (optional, default false), must not be a list
Available since	7.17
Description	Returns all group ids of given principal id, representing a user, position or group. It is also possible to determine whether groups are considered flat or hierarchical and whether all or only active groups are returned. This function is not searchable and not sortable.

Examples

NQL	Description
`select groupResolve(%currentUserPrincipalId)`	Returns ids of all groups of current user
`select pidResolve(groupResolve(%currentUserPrincipalId, true))`	Returns names of all explicit assigned groups of all positions of current user
`select groupResolve(%currentUserPrincipalId, true, true)`	Returns ids of all explicit assigned groups of all positions of current user which are active

13.6.13. CompetenceResolver

NQL	`competenceResolve`
Return type	List of String
Parameters	`operand` of type String, must not be a list
	`type` of type Integer or Long, must not be a list
	`implicit` of type Boolean (optional, default false), must not be a list
	`activeOnly` of type Boolean (optional, default false), must not be a list
Available since	7.17

Description	Returns all principal ids where given principal has a competence of given type for. Given principal id may represent a user, position or group. If given principal id represents a user, all positions of the user are considered (when active flag is set to true, only active positions are considered). The type must be a numeric value between 1 and 6 where 1 means head, 2 means proxy, 3 means owner, 4 means local group admin, 5 means manager and 6 means agent competence. For competences of type head, proxy and manager it is possible to determine, whether implicit competences are considered. If set to true, for type proxy also transitive proxies are returned. For type head and manager also members of groups are returned (flat for head and hierarchical for manager). It is also possible to determine whether only active competences are considered. This function is not searchable and not sortable.

Examples

NQL	Description
`select competenceResolve(%currentUserPrincipalId, 2)`	Returns ids of all explicit proxied principals of current user
`select pidResolve(competenceResolve(%currentUserPrincipalId, 1, true))`	Returns names of all implicit headed principals of current user
`select competenceResolve(%currentDefaultPositionPrincipalId, 5, true, true)`	Returns ids of all implicit managed principals where current user's default position has an active competence for

13.6.14. CompetenceReferenceResolver

NQL	`competenceReferenceResolve`
Return type	List of String
Parameters	`operand` of type String, must not be a list
	`type` of type Integer or Long, must not be a list
	`activeOnly` of type Boolean (optional, default false), must not be a list
Available since	8.4

Description	Returns all ids of principals who own a competence of given type for given principal. That means, this function is the inverse function of the competence resolver function. Given principal id may represent a user, position or group. If given principal id represents a user, all positions of the user are considered (when active flag is set to true, only active positions are considered). The type must be a numeric value between 1 and 6 where 1 means head, 2 means proxy, 3 means owner, 4 means local group admin, 5 means manager and 6 means agent competence. It is also possible to determine whether only active competences are considered. This function is not searchable and not sortable.

Examples

NQL	Description
`select competenceReferenceResolve(%currentUserPrincipalId, 2)`	Returns ids of all proxies of current user
`select pidResolve(competenceReferenceResolve(%currentUserPrincipalId, 1, true))`	Returns names of all active head principals of current user
`select competenceResolve(%currentDefaultPositionPrincipalId, 5, true)`	Returns ids of all active managers of current user

13.6.15. ReferenceResolver

NQL	`refResolve`
Return type	Any type, dependent on type of resolved property
Parameters	`referenceId` of type Integer, Long or String
	`category` of type Integer or Long, must not be a list
	`propertyName` of type String, must not be a list, should be a constant
	`areaName` of type String (optional, default null), must not be a list
Available since	7.7, fourth parameter since 7.15

Description	Resolves a property of a referenced resource. This function is determined by a reference id, a category and a property name. A referenced resource may be a repository resource, a workflow instance, a masterdata, a principal(-info), a calendar, or a business process instance. The reference id may be a numeric id or a string representation of a referenced resource, e.g. a resource id or a workflow instance id etc. The category must be a numeric value between 1 and 6, where 1 means repository, 2 means workflow, 3 means business process, 4 means masterdata, 5 means principal or principal info and 6 means calendar. The property must be the name of a property which exists in given category or a query operand which contains at least one property which exists in given category. Providing an optional area name is only supported for categories repository, workflow, business process and masterdata. In categories repository, workflow and business process, the area name must be the name of an existing document area. In category masterdata, the area name must be the name of an existing masterdata scope. Note that resolving of resources may cause performance problems, as the read permissions are evaluated for every resource of a result set. If the referenced id is an id of a principal and only the name is supposed to be resolved, use function PrincipalIdResolver instead (does not need any validation of permissions). Caution: the property to be resolved should be deployed as a constant and *NOT* a property itself, as this produces unpredictable results. This function is not searchable and not sortable.

Examples

NQL	Description
`select refResolve(parentidentifier, 1, 'displayname')`	Returns the display name of the parent folder of a resource
`select refResolve(%currentUserPrincipalId, 5, 'emails')`	Returns the email addresses of current principal
`select refResolve(42, 1, 'displayname', 'AnotherDocumentArea')`	Returns the display name of resource with identifier 42 in document area with name 'AnotherDocumentArea'

13.6.16. ValueSetResolver

NQL	`valueSetResolve`
Return type	String
Parameters	`valueSet` of type String
	`key` of type String
Available since	7.14

Description	Resolves an entry in given value set. If data of given value set is in expected format (e.g. the format used by *nscale Cockpit* or *nscale Web*) the entry with given key is considered and corresponding value is returned. If given key does not exist, the key is returned as value. If given value set does not exist, the *nscale Server Application Layer* tries to find a value set which is represented by given name, e.g. if given value set name is the suffix of an existing value set where the prefix represents a locale, the locale of the client application is taken into account, see examples below. If no value set can be found, the key is returned as value. This function is not searchable and not sortable.

Examples

NQL	Description
`select valueSetResolve('default_useraddresstitle', 'useraddresstitle_mr')`	Returns Mr or Herr, depending on locale of the client application (note that this value set is provided when user scenario is enabled)
`select valueSetResolve('de_default_useraddresstitle', 'useraddresstitle_mr')`	Returns Herr (note that this value set is provided when user scenario is enabled)

13.6.17. KeyGenerator

NQL	`keyGen`
Return type	String or Long, depending on whether a uuid or a sequential number is requested
Parameters	`keyGenerator` of type String (optional, default null)
	`lowerBound` of type Integer or Long (optional, default null)
	`upperBound` of type Integer or Long (optional, default null)
Available since	7.3, second and third parameter since 7.7

Description	Generates either a uuid or a numeric id. If the function is called without any parameter, a uuid of type String is returned. If a key generator is specified, a numeric id of type Long is generated by given key generator. If given key generator is null, the default (plugins) key generator is considered. Note that referenced key generator must exist (either a KeyGeneratorDefinition or a RangeKeyGeneratorDefinition of given name), else an exception is thrown. For a numeric id, a lower and an upper bound may be specified, to determine the interval from which a number is generated. When an interval is specified, the referenced key generator should be a range key generator, specifying upper and lower bounds for a non range key generator is not reasonable. When an interval is specified for a range key generator and there is no more free number left in this interval, null is returned. This function is not searchable and not sortable.

Examples

NQL	Description
`select keyGen()`	Returns a uuid
`select keyGen(null)`	Returns a numeric id from default (plugins) key generator
`select keyGen('myKeyGenerator')`	Returns a numeric id, generated by key generator with name 'myKeyGenerator'
`select keyGen('myRangeKeyGenerator', 1000, 2000)`	Returns a numeric id between 1000 and 2000 (inclusive), generated by range key generator with name 'myRangeKeyGenerator'

13.6.18. BitAnd

NQL	`bitAnd`
Return type	Long
Parameters	`lhsOperand` of type Integer or Long
	`rhsOperand` of type Integer or Long
Available since	7.5
Description	Calculates a bitwise AND of two numeric operands. An operand which evaluates to null is replaced by value 0.

Examples

NQL	Description
`select bitAnd(15, 44)`	Returns 12

NQL	Description
`where bitAnd(fulltextstate, 8) = 8`	Finds resources where fulltext indexing caused an error (not classified permanent)

13.6.19. BitOr

NQL	`bitOr`
Return type	Long
Parameters	`lhsOperand` of type Integer or Long
	`rhsOperand` of type Integer or Long
Available since	7.5
Description	Calculates a bitwise OR of two numeric operands. An operand which evaluates to null is replaced by value 0.

Examples

NQL	Description
`select bitOr(15, 44)`	Returns 47

13.6.20. LogicalAnd

NQL	`logicalAnd`
Return type	Boolean
Parameters	`operands…` arbitrary number of operands of type Boolean
Available since	7.16
Description	Returns true when all given parameters resolve to true, else false is returned.

Examples

NQL	Description
`select logicalAnd(true, true, true)`	Returns true

13.6.21. LogicalOr

NQL	`logicalOr`
Return type	Boolean

Parameters	`operands···` arbitrary number of operands of type Boolean
Available since	7.16
Description	Returns true when at least one of given parameters resolves to true, else false is returned.

Examples

NQL	Description
`select logicalOr(false, true, false)`	Returns true

13.6.22. LogicalXOr

NQL	`logicalXOr`
Return type	Boolean
Parameters	`operands···` arbitrary number of operands of type Boolean
Available since	7.16
Description	Returns true when exactly one of given parameters resolves to true, else false is returned.

Examples

NQL	Description
`select logicalXOr(false, true, true)`	Returns false

13.6.23. MatchesNumeric

NQL	`matchesNumeric`, `isNumeric` or `_n`
Return type	Boolean
Parameters	`operand` of type String
Available since	7.10
Description	Evaluates whether a string represents a numeric value (could be converted to a number).

Examples

NQL	Description
`select isNumeric('123')`	Returns true
`select isNumeric('abc')`	Returns false

NQL	Description
`orderby toLong(case(isNumeric(displayname), displayname, '0'))`	Sorts a result set numerically, if displayname contains values which can be converted to numbers

13.6.24. Matches

NQL	`matches`
Return type	Boolean
Parameters	`operand` of type String
	`regularExpression` of type String
Available since	7.10
Description	Evaluates whether a string matches a regular expression.

Examples

NQL	Description
`select matches('123', '[0-9]+')`	Returns true
`select matches('abc', '[0-9]+')`	Returns false

13.6.25. Ascii

NQL	`ascii`
Return type	Integer
Parameters	`operand` of type String or Blob
Available since	7.5
Description	Returns the ascii code of a character (the first character of a string).

Examples

NQL	Description
`select ascii('A')`	Returns 65
`select ascii(tokenize(displayname, ''))`	Returns ascii codes of all characters of property displayname

13.6.26. Char

NQL	`char`
Return type	String
Parameters	`operand` of type Integer, Long or Blob
Available since	7.5
Description	Returns the character represented by an ascii code.

Examples

NQL	Description
`select char(65)`	Returns A
`select toFlat(char(ascii(tokenize(displayname, ''))),'')`	Returns the displayname

13.6.27. Custom

NQL	`customFunction`, `#` or `#<name of custom function>`
Return type	Defined by custom function
Parameters	Defined by custom function
Available since	7.10
Description	Executes a previously defined custom function. A custom function can be defined in two ways, either by configuring a CustomFunctionDefinition or by implementing a plugin of type CustomComputedIndexingPropertyDefinitions whose handler class implements interface CustomFunctionsHandler. The number and type of parameters as well as the type of returned value depends on the definition of the custom function.

Examples

NQL	Description
`select #FooBar(87, %today, displayname, true)`	Executes previously defined custom function with name FooBar
`select customFunction('FooBar', 87, %today, displayname, true)`	Equivalent to the former statement
`select #(FooBar, 87, %today, displayname, true)`	Equivalent to the former statement

Chapter 14. Aggregate Searches

NQL also supports aggregate searches. The basic structure of an aggregate search is the same as for a normal search with two exceptions. First, the count clause is not supported. Second, the select clause is mandatory. This is evident as an aggregate search does not return key information, so an aggregate search without a select clause would return nothing. The following aggregate functions are supported:

Function	Description
`count`	Returns number of values which are not null
`max`	Returns maximum of all values
`min`	Returns minimum of all values
`avg`	Returns average of all values
`sum`	Returns sum of all values
`countDistinct`	Returns number of different values which are not null
`distinct`	Returns all different values (duplicates are omitted)

Every aggregate function allows exactly one *QueryOperand* as parameter, where the *QueryOperand* must be either a property or a query function. If a query function is used, there must be at least one property used inside of the (maybe nested) parameters of the query function. Note that the aggregate functions max, min, avg and sum are ambiguous to query functions of same name. If they are used in an aggregate search, the aggregate function is presumed, when the function only has one parameter, else the query function is presumed (the aggregate function only allows one parameter while the use of a query function with only one parameter is senseless). Note also that aggregate function distinct may only be used for the first property of the select clause. Also, when using distinct, the use of other aggregate functions is useless.

14.1. The select clause

There are only a few deviating characteristics for the select clause of an aggregate search, which are described here. NQL allows an arbitrary number of aggregate functions in the select clause as well as properties which do not use an aggregate function. Every property which does not use an aggregate function is grouped, which means it is part of the groupby clause. Note that NQL does not have an explicit groupby clause but the groupby clause is set implicitly. Be aware that the resulting values of properties of the select clause are calculated in the database, so in an aggregate search all used properties and query functions must be searchable. Additionally, computed properties are allowed in the select clause which only need exactly one necessary property to be calculated. In this case, the grouping takes place on the necessary property. When using aggregate function `count` it is advised to count over a property which does not have null values (like `identifier` in Repository or `processidentifier` in Workflow etc.), as using other properties may produce unpredictable results.

14.2. The where clause

There are only a few deviating characteristics for the where clause of an aggregate search, which

are described here. Aggregate functions may also be used in the where clause. In SQL, this would result in a having clause, but NQL does not have an explicit having clause but the having clause is set implicitly. Note that when aggregate functions are used in the where clause, all used properties in the where clause which are not used in an aggregate function must also be used in the select clause.

14.3. The orderby clause

There are only a few deviating characteristics for the orderby clause of an aggregate search, which are described here. Aggregate functions may also be used in the orderby clause. Note that in the orderby clause only properties may be used which are also part of the select clause.

14.4. Examples

NQL	Description
`select count(identifier)`	Returns the number of all resources in a folder (of course only visible resources are considered)
`select count(identifier) scope subtree`	Returns the number of all resources in the sub-tree of a folder
`select count(identifier), resourcetype`	Returns the number of all resources in a folder, grouped by their resource type. So, assuming there is at least one folder, document and link inside of the folder, the returned result set contains three rows, containing the number of documents, folders and links in the first column and the corresponding resource type in the second column
`select count(identifier), resourcetype orderby resourcetype desc`	Returns the same result as the search before, but sorts it by resource type in descending order
`select count(identifier), resourcetype orderby count(identifier)`	Returns the same result as the search before, but sorts it by the number of resources per resource type
`select count(identifier), resourcetype where count(identifier) > 10`	Returns the number of all resources in a folder, grouped by their resource type, where the number of resources per resource type is greater than 10
`select count(identifier), resourcetype where count(identifier) > 10 and resourcetype > 1 orderby count(identifier) desc, resourcetype`	Returns the number of all documents and links in a folder, grouped by their resource type, where the number of resources per resource type is greater than 10, ordered by number and resource type

NQL	Description
`select count(identifier), resourcetype, year(initialcreation)`	Returns the number of all resources in a folder, grouped by their resource type and the year of creation
`select count(identifier), up(substring(displayname, 0, 1)) scope subtree`	Returns the number of resources grouped by the first character of the display name in upper case
`select count(identifier), parentidentifier scope subtree`	Returns the number of resources per folder
`select max(displayname), min(displayname), avg(lifecyclestate) scope subtree`	Returns the maximum display name, the minimum display name and the average of all lifecycle states
`select max(displayname), min(displayname), resourcetype scope subtree`	Returns the maximum display name and the minimum display name per resource type
`select max(initialcreation), min(initialcreation), max(year(initialcreation)), min(year(initialcreation)) scope subtree`	Returns the maximum creation date, the minimum creation date and the maximum and minimum year of creation
`select sum(itemlength) where resourcetype = 2 scope subtree`	Returns the sum of the length of all content items of all documents
`select sum(itemlength), itemcontenttype where resourcetype = 2 scope subtree`	Returns the sum of the length of all content items of all documents, grouped by their content type
`select count(identifier), displayname where count(identifier) > 1 orderby count(identifier) desc, displayname scope subtree`	Returns duplicate display names and the number of their occurrences
`select countDistinct(displayname) scope subtree`	Returns the number of different display names
`select distinct(displayname) scope subtree`	Returns all different display names

Chapter 15. Subqueries

NQL also supports subqueries in form of instant filtered properties (shortened filtered properties). A filtered property may be used in the list of requested properties (select clause) and in the condition (where clause).

15.1. The select clause

In the select clause, a filtered property may be used in two ways. Either it is used to restrict the returned values of a multi-value property or it is used to retrieve the value of an aggregate property by building a subquery. In the first case, a filtered property consists of a target property and a condition, separated by the \ character. The target property must be a multi-value property and the condition must only contain properties of the same multi-value scope as the target property. The effect is, that the returned values for the multi-value property are filtered by specified condition. Note that the condition must not contain itself a filtered property. As for instant formatted properties, an alias may be used for a filtered property. So the syntax for a filtered property to restrict the returned values of a multi-value property is:

```
[alias=]targetProperty\condition
```

15.1.1. Examples

NQL	Description
`select itemcontenttype\itemcontenttype = 'text/plain'`	Returns the multi-value property itemcontenttype, where all entries are omitted which are not text/plain (the returned lists are filtered)
`select text=itemcontenttype\itemcontenttype = 'text/plain'`	The same search as above but with an alias
`select itemcontenttype\itemlength >= 1000`	Returns the multi-value property itemcontenttype, where all entries are omitted which are smaller than 1000 bytes
`select x=itemcontenttype\itemcontenttype in ('text/plain', 'image/tiff') and itemlength > 100, itemcontenttype, itemlength`	Returns three properties, column two contains multi-value property itemcontenttype, column three contains multi-value property itemlength and column one contains filtered property x, which contains all entries of itemcontenttype where the type is text/plain or image/tiff and the length is greater than 100

The second case to use a filtered property in the select clause is to retrieve the value of an aggregate property. The basic structure of a filtered property for this case is the same as for the first case, so there is a target property and a condition. In contrast to the first case however, the target property may be a single-value property, the condition is optional and the filtered property must always be

parenthesized. The target property must be an aggregate property, as the subquery must return a unique value. Furthermore, an additional component may be used in a filtered property, a bind property. A bind property may be specified to bind a property of the subquery with the same property of the superior select (a kind of join of the two statements). The bind property starts with the `@` character. When bind is used without specifying a property (by just declaring the `@` character), the primary key is bound. So the syntax for a filtered property to retrieve the value of an aggregate property is:

```
[alias=](targetProperty[@[bindProperty]]\[condition])
```

15.1.2. Examples

NQL	Description
`select (max(displayname)\)`	Returns the maximum displayname of all entries of returned result set, the returned value is identical in all rows
`select maxlen=(max(len(displayname))\)`	Returns the maximum length of all displaynames of returned result set, the returned value is identical in all rows
`select maxlen=(max(len(displayname))@resourcetype\)`	Returns the maximum length of all displaynames of returned result set per resourcetype, the returned value is identical for all rows with identical resourcetype
`select pad(displayname, (max(len(displayname))\))`	Returns the padded displayname, the length of all returned values is determined by the maximum length of all displaynames of returned result set
`select contentlength, (sum(contentlength)\), round(product(div(contentlength, (sum(contentlength)\)), 100), 2)`	Returns the content length, the sum of all content lengths and the percentage of content length compared to the overall length of all contents, rounded to two decimal digits

15.2. The where clause

A filtered property in the where clause behaves like a subselect. The basic structure of a filtered property in the where clause is the same as in the second case for the select clause, so there is a target property, a condition and a bind property. In contrast to the select clause however, an alias must not be used and the condition may contain itself filtered properties (a subselect inside of a subselect).

In addition, there is another optional component in a filtered property in the where clause, a list of filter facet properties. Besides the target property, all other components are optional. Filter facet properties may be used, when the target property is an aggregate property. The effect is, that the filter facet properties are grouped (they are part of the implicit groupby clause). The filter facet

properties start with the | character and are comma-separated.

In general, a subselect always returns a list, so for a subselect only operator `in` is supported. Sole exception is when the target property is an aggregate property and there are no filter facet properties used. In this case, the return value of the subselect is unique, so more operators may be used like `=`, `!=`, `<`, `<=`, `>` or `>=`. A subselect in the where clause must always be parenthesized. So the syntax for a filtered property in the where clause is:

```
(targetProperty[|filterFacetProperty[,...]][@[bindProperty]]\[condition])
```

15.2.1. Examples

NQL	Description
`where parentidentifier in (identifier\parentidentifier = -1) scope subtree`	Finds all resources of level 2
`where parentidentifier in (identifier\parentidentifier in (identifier\parentidentifier = -1)) scope subtree`	Finds all resources of level 3
`where not identifier in (parentidentifier\) and resourcetype = 1 scope subtree`	Finds all empty folders
`where int1 > (avg(int1)\) scope subtree`	Finds all resources where the value of integer property int1 is greater than the average of int1. Note that the use of operator `>` is allowed, as the target property is an aggregate property and no filter facet properties are specified
`where int1 > (avg(int1)@resourcetype\) scope subtree`	Finds all resources where the value of integer property int1 is greater than the average of int1, regarding the own resource type (in other words: finds all folders where the value of integer property int1 is greater than the average of all folders and analogous for documents and links)
`where displayname in (max(displayname)|resourcetype\) scope subtree`	Finds all resources where the display name is equal to the maximum display name of either folder, document or link

NQL	Description
`where displayname in (max(displayname)@resourcetype\) scope subtree`	Finds all resources where the display name is equal to the maximum display name, regarding the own resource type. The difference to the former statement is as follows: assuming, the maximum name of all folders is x, the maximum name of all documents is y and the maximum name of all links is z, then the former statement would also find documents with name x and links with name x and y. In contrast, this statement binds the resource type and therefore only folder x, document y and link z are found
`where displayname in (max(displayname)|resourcetype, year(initialcreation)\) scope subtree`	This is an example for a subselect with more than one filter facet property; the subselect finds the maximum display name per resource type and year
`select case(int1 > (avg(int1)\), 'yes', 'no')`	Returns yes for all resources where the value of int1 is greater than the average of int1, else no is returned
`select displayname, identifier, version where not version in (max(version)@identifier\) orderby identifier, version`	When searching for versions this query filters the maximum version per resource

Chapter 16. Bulk Operations

Writing operations are not supported by NQL which means there is no equivalent to SQL's `insert`, `update` or `delete` statements. However, the *nscale Query Tester* does support bulk operations. For the sake of completeness, the syntax of these bulk operations is also described here, but be aware that this is not part of NQL!

16.1. Bulk Update

The *nscale Query Tester* supports bulk updates. A bulk update statement consists of the same components as an NQL statement except that the select clause is replaced by an update clause. The update clause consists of the key word `update` followed by a comma-separated list of key-value pairs, where the key is a property and the value is a *QueryOperand* (a constant, context variable, property, query function or a list of these), separated by the `=` character.

16.1.1. Examples

Update statement	Description
`update displayname = 'abc' where displayname is null`	Updates property displayname for all resources in current folder where displayname is null
`update displayname = 'abc' where displayname is null scope subtree`	Updates property displayname for all resources in subtree of current folder where displayname is null
`update int1 = 7, int2 = 10 where plus(int1, int2) < 3`	Updates properties int1 and int2 where the sum of int1 and int2 is less than 3
`update filename = concat(displayname, '.', fileextension) where contenttype is not null`	Updates property filename for all resources having content with the value of property displayname, a dot and the value of property fileextension
`update color = ('green', 'red')`	Updates multi-value property color for all resources

16.2. Bulk Delete

The *nscale Query Tester* supports bulk deletes. A bulk delete statement consists of the same components as an NQL statement except that the select clause is replaced by a delete clause. The delete clause simply consists of the key word `delete`.

16.2.1. Examples

Delete statement	Description
`delete where displayname = 'xyz'`	Deletes all resources in current folder where displayname equals xyz

Delete statement	Description
`delete where resourcetype = 2 scope subtree`	Deletes all documents
`delete`	Deletes all resources