Combining queries with subqueries in AQL

Subqueries let you form complex requests and allow you to process more data in with a single query

How to use subqueries

Wherever an expression is allowed in AQL, a subquery can be placed. A subquery is a query part that can introduce its own local variables without affecting variables and values in its outer scope(s).

It is required that subqueries be put inside parentheses `(` and `)` to explicitly mark their start and end points:

```
FOR p IN persons
  LET recommendations = ( // subquery start
    FOR r IN recommendations
      FILTER p.id == r.personId
      SORT p.rank DESC
      LIMIT 10
      RETURN r
  ) // subquery end
  RETURN { person : p, recommendations : recommendations }
```

A subquery's result can be assigned to a variable with `LET` (/stable/aql/high-level-operations/let/) as shown above, so that it can be referenced multiple times or just to improve the query readability.

Function calls also use parentheses and AQL allows you to omit an extra pair if you want to use a subquery as sole argument for a function, e.g. `MAX(<subquery>)` instead of `MAX((<subquery>))`:

```
FOR p IN persons
  COLLECT city = p.city INTO g
  RETURN {
    city : city,
    numPersons : LENGTH(g),
    maxRating: MAX( // subquery start
      FOR r IN g
      RETURN r.p.rating
    ) // subquery end
  }
```

The extra wrapping is required if there is more than one function argument, however, e.g. `NOT_NULL((RETURN "ok"), "fallback")`.

Subqueries may also include other subqueries.

Subquery results and unwinding

Subqueries always return a result **array**, even if there is only a single return value:

```
RETURN ( RETURN 1 )
```

```
[ [ 1 ] ]
```

To avoid such a nested data structure, `FIRST()` (/stable/aql/functions/array/#first) can be used for example:

```
RETURN FIRST( RETURN 1 )
```

```
[ 1 ]
```

To unwind the result array of a subquery so that each element is returned as top-level element in the overall query result, you can use a `FOR` loop:

```
FOR elem IN (RETURN 1..3) // [1,2,3]
  RETURN elem
```

```
[
  1,
  2,
  3
]
```

Without unwinding, the query would be `RETURN (RETURN 1..3)` and the result a nested array `[ [ 1, 2, 3 ] ]` with a single top-level element.

Evaluation of subqueries

From v3.12.1 onward | Up to v3.12.0

Subqueries that are used inside expressions are pulled out of these expressions and conditionally executed beforehand. The effective behavior is short-circuiting evaluation. An example is the ternary operator (/stable/aql/operators/#ternary-operator).

Consider the following query:

```
RETURN RAND() > 0.5 ? (RETURN 1) : 0
```

It get transformed into something more like this, with the calculation of the subquery happening before the evaluation of the condition:

```
LET temp1 = RAND() > 0.5
LET temp2 = (FILTER temp1 RETURN 1)
RETURN temp1 ? temp2 : 0
```

The condition is evaluated separately and stored in a variable, then the subquery is executed conditionally using a `FILTER` operation. Finally, the ternary operator reuses the result of the evaluated condition to determine which value to return.

The short-circuiting behavior allows you to write queries like the following:

```
LET maybe = DOCUMENT("coll/does_not_exist")
LET dependent = maybe ? (
  FOR attr IN ATTRIBUTES(maybe)
    RETURN attr
) : "document not found"
RETURN dependent
```

The `maybe` variable can be `null`, which cannot be iterated over with `FOR` and calling `ATTRIBUTES()` with `null` as argument would raise a query warning. However, the subquery is only executed if `DOCUMENT()` found a document, so this query works without issues.

Similarly, when you use subqueries as sub-expressions that are combined with logical `AND` or `OR`, the subqueries are only executed if the outcome is yet to be determined:

```
RETURN false AND (RETURN ASSERT(false, "executed"))
```

```
RETURN true OR (RETURN ASSERT(false, "executed"))
```

If the first operand of a logical `AND` is `false`, the overall result is `false` regardless of the second operand. If the first operand of a logical `OR` is `true`, the overall result is `true` regardless of the second operand. As the outcome is already determined, there is no need to execute the subqueries in these cases.

‹ Accessing data from collections (/stable/aql/fundamentals/accessing-data-from-collections/)

Query Results (/stable/aql/fundamentals/query-results/) ›