Nested subqueries and subquery chaining

Stefan Plantikow
stefan.plantikow@neo4j.org
Petra Selmer
petra.selmer@neo4j.org

What?

Subqueries are self-contained queries running within the scope of an outer query

```
<outer-query>
{
      <inner-query>
}
<next-outer-query> (continued)
```

What?

1. Read-only nested subqueries
2. Read/Write updating subqueries
3. Set operations
4. Chained subqueries

Not in this talk: Subqueries in expressions (scalar, lists, existential)

Relevant CIPs

CIP2017-04-20 - Query combinators for set operations

https://github.com/opencypher/openCypher/pull/227

CIP2016-06-22 - Nested, updating, and chained subqueries

https://github.com/opencypher/openCypher/pull/100

+ CIPs for subqueries in expressions not covered by this talk

Why?

Queries are easier to:

- construct
- maintain
- read

Why?

Subqueries enable:

- Composition of query pipelines
- Programmatic query composition
- Post-processing of results
- Multiple write actions for each record

1. Read-only, nested subqueries

Preliminaries

`<inner-query>`

Any complete, read-only query "`... RETURN`" enclosed within `{}`

May be **uncorrelated** or **correlated**
(`<inner-query>` may use variables from the outer query)

Read-only subqueries may be nested at an arbitrary depth

Variants

Regular subqueries

```
MATCH { <inner-query> }
```

Optional subqueries

```
OPTIONAL MATCH { <inner-query> }
```

Mandatory subqueries

```
MANDATORY MATCH { <inner-query> }
```

Semantics

`<inner-query>` is evaluated for each record from `<outer-query>`

Variables var_{outer} from `<outer-query>` are visible to `<inner-query>`

Variables var_{inner} are introduced and returned by `<inner-query>`
(as output records)

Variables from var_{outer} and var_{inner} are available to `<next-outer-query>`
(as result records)

Semantics

`<inner-query>` may omit variables from var_{outer}

But omitted variables are re-added to the final result records

`<inner-query>` may shadow and thus alter any variable in var_{outer}

But CIP recommends warning if `<inner-query>:`

- Discards a variable v_i in var_{outer}
- Re-introduces v_i

Semantics: Regular MATCH

result records available to the `<next-outer-query>`

All output records returned by the `<inner-query>`
amended with missing variables from var_{outer}

Semantics: Mandatory MATCH

result records available to the `<next-outer-query>`

All output records returned by the `<inner-query>`
amended with missing variables from var_{outer}

Generate an error if no output records are returned by `<inner-query>`

Semantics: Optional MATCH

result records available to the `<next-outer-query>`

`{`

All output records returned by the `<inner-query>` (if at least one of these), **or** a single record with the same fields as the output records, where any v_i in var_{inner} is set to `null`

`}`

and amended with missing variables from var_{outer}

Example: Post-UNION processing

```
MATCH {
  MATCH ...
  RETURN *
  UNION
  MATCH ...
  RETURN *
}
WITH *
WHERE ...
RETURN *
```

Example: Correlated subquery

```
MATCH (u:User {id: $userId})
MATCH (f:Farm {id: $farmId})-[:IS_IN]->(country:Country)
MATCH {
  MATCH (u)-[:LIKES]->(b:Brand)-[:PRODUCES]->(p:Lawnmower)
  RETURN b.name AS name
  UNION
  MATCH (u)-[:LIKES]->(b:Brand)-[:PRODUCES]->(v:Vehicle)
  WHERE v.leftHandDrive = country.leftHandDrive
  RETURN b.name AS name
}
RETURN f, name
```

2. Read/Write updating subqueries

Preliminaries

`<updating-query>`

Can be any updating query, and may not end with **`RETURN`**

(**No** data is returned)

Preliminaries

`<updating-query>`

May be **uncorrelated** or **correlated**

(`<updating-query>` may use variables from the outer query)

Updating subqueries may be nested at an arbitrary depth

Read-only nested subqueries may **not** contain updating subqueries

`FOREACH` removed from Cypher - now obsolete

Variants

Simple updating subqueries

```
DO { <updating-query> }
```

Conditionally-updating subqueries

```
DO
    [WHEN <predicate> THEN { <updating-query> }+
    [ELSE { <updating-query> }]
END
```

Semantics

`<updating-query>` is run for each incoming record

Executing **`DO`** does not affect the cardinality

Input records are passed on to `<next-outer-query>`

Conditional **`DO`**:
This happens whether or not the incoming record is eligible for processing by `<updating-query>`

Semantics

A query can end with a **DO** subquery in the same way as an updating query currently can end with any update clause

All var_{inner} introduced by `<updating-query>` are suppressed by **DO**

DO can be nested within another **DO**

Semantics: conditional variant

```
DO
   [WHEN <predicate> THEN { <updating-query> }+
   [ELSE { <updating-query> }]
END
```

Conditions in **WHEN** evaluated in order

`<updating-query>` evaluated for the first true condition, falling through to **ELSE** if no true conditions found

Otherwise, no updates will occur

Example: FOREACH vs DO

Old version:

```
MATCH (r:Root)

FOREACH(x IN range(1, 10) |
    MERGE (c:Child {id: x})
    MERGE (r)-[:PARENT]->(c)
)
```

New version:

```
MATCH (r:Root)
UNWIND range(1, 10) AS x
DO {
    MERGE (c:Child {id: x})
    MERGE (r)-[:PARENT]->(c)
}
```

Example: Conditional DO

```
MATCH (r:Root)
UNWIND range(1, 10) AS x
DO WHEN x % 2 = 1 THEN {
    MERGE (c:Odd:Child {id: x})
    MERGE (r)-[:PARENT]->(c)
}
ELSE {
    MERGE (c:Even:Child {id: x})
    MERGE (r)-[:PARENT]->(c)
}
END
```

3. Set operations

What?

Set operations combine results from two queries into one

```
<left-hand-query>
<SET-OPERATION>
<right-hand-query>
```

What?

UNION

UNION ALL

UNION MAX

INTERSECT

INTERSECT ALL

EXCEPT

EXCEPT ALL

EXCLUSIVE UNION

EXCLUSIVE UNION MAX

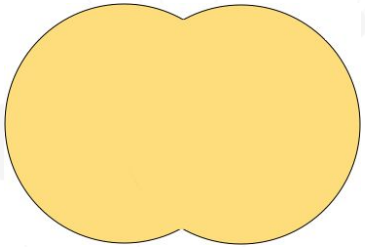
Set operation semantics

- Rule: `<left-hand-query>` and `<right-hand-query>` return same variables (fields) in same order

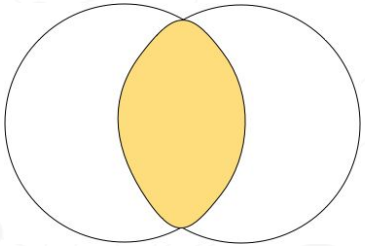
- **`UNION`** set union
- **`INTERSECT`** set intersection
- **`EXCEPT`** set difference
- **`EXCLUSIVE UNION`** set union

Set operation semantics

- UNION

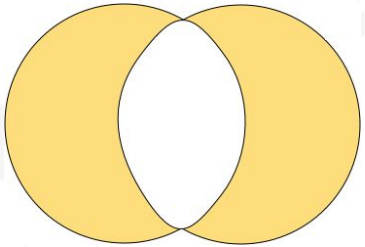


- INTERSECT

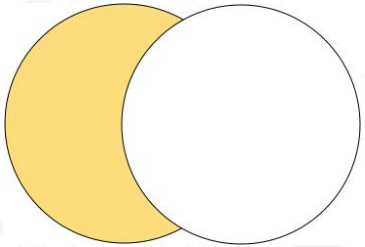


Set operation semantics

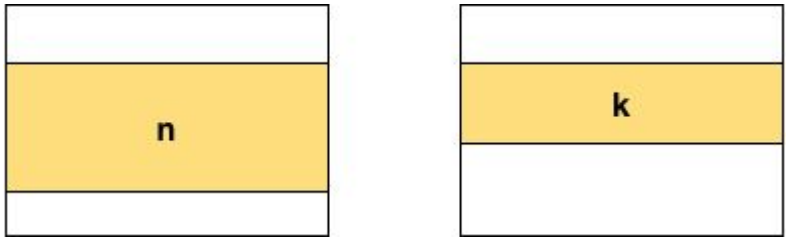
- `EXCLUSIVE UNION`



- `EXCLUDE`



Multiset operations



- `UNION ALL` n + k
- `INTERSECT ALL` min(n, k)
- `EXCLUDE ALL` max(0, n-k)

- `UNION MAX` max(n, k)
- `EXCLUSIVE UNION MAX` max(n, k) - min(n,k)

Multiset operation semantics

- Rule: `<left-hand-query>` and `<right-hand-query>` return same variables (fields) in same order

- **`UNION ALL`** multiset union
- **`INTERSECT ALL`** multiset intersection
- **`EXCEPT ALL`** multiset difference (0-bounded)

Multiset operation semantics

- Rule: `<left-hand-query>` and `<right-hand-query>` return same variables (fields) in same order

- **`UNION MAX`** max-bounded multiset union (largest number of duplicates from either input query)

- **`EXCLUSIVE UNION MAX`** exclusive union (excess duplicates from either input query over the other)

Using multiple set operations

All set operations are left-associative

```
Q1 UNION Q2 INTERSECT Q3
```

is equivalent to

```
((Q1 UNION Q2) INTERSECT Q3)
```

OTHERWISE [ALL]

- **`OTHERWISE [ALL]`** computes the logical choice between `<left-hand-query>` and `<right-hand-query>`

- Becomes `<left-hand-query>` if non-empty, `<right-hand-query>` otherwise

- **OTHERWISE** does not preserve duplicates
- **OTHERWISE ALL** preserves duplicates

- Rule: `<left-hand-query>` and `<right-hand-query>` return same variables (fields) in same order

CROSS

- **CROSS** computes the cross product (cartesian product) between `<left-hand-query>` and `<right-hand-query>`

- Rule: `<left-hand-query>` and `<right-hand-query>` must return different variables

4. Chained subqueries

What?

Chain arbitrary queries:

```
<query1> composedWith <query2> composedWith <query3> ...
```

Returned variables from `<queryN>` are passed on to `<queryN+1>`

Why?

- Query pipeline construction:
  - Factor out subqueries
  - Post-union processing
  - Execute read-only query after updating query

- Programmatic composition:
  - `result.cypher("WITH a")`

What won't work

```
MATCH {
  MATCH {
    MATCH {
      ...
      // death by curly
      ...
    }
  }
}
```

- Requires deep nesting
- Would require putting updating queries inside read-only queries (instead of after them)

Our proposal

```
MATCH ...RETURN ...
UNION
MATCH ...RETURN ...
WITH
...
MATCH ...RETURN ...
```

Recast **WITH** after **RETURN** as query combinator (similar to **UNION**)

- Linear flow of subqueries
- Post-union processing

Our proposal

```
<query-without-combinator>
<query-combinator>
<query-without-combinator>
...
```

Integrate query combinators and set operations

- Halve level of nesting

Data-dependent composition

```
<query1>
WITH ...
<query2>
WITH ...
<query3>
...
```

Pass variables and rows
from one query to the next

Data-independent composition

```
<query1>
THEN
<query2>
THEN
<query3>
...
```

Passes no variables and records
from one query to the next

Instead:
Reduces cardinality to single empty record

WITH and THEN at start of query

```
WITH ... <query>
```

Declares expected input variables

Useful for programmatic composition: `result.cypher("WITH a")`

```
THEN ... <query>
```

Drops incoming records
(Reduces cardinality to single empty record)

Complex chained queries

Like set operations, chained subqueries are left-associative!

```
<query1> WITH ... <query2> THEN <query3> ...
```

is equivalent to

```
(((<query1> WITH ... <query2>) THEN <query3>) ...)
```

Complex chained queries

`<queryN>` cannot contain set operations or query combinators

But it can contain nested subqueries again!

```
MATCH { ... } RETURN ...
UNION
MATCH { ... } RETURN ...
WITH ...
RETURN ...
```

Discussion

Extension to multiple graphs

Multiple Graphs add returning of graphs besides variables

Set operations are defined over sets of records

Set operations may also be defined for graphs

=> This extends to Cypher with support for multiple graphs

Chaining is about passing variables to the follow-up query

=> This extends to Cypher with support for multiple graphs

Summary

Adding subqueries to Cypher will massively increase expressivity

Chained subqueries provide a powerful composition mechanism

Subqueries naturally extends to Cypher for multiple graphs

Thank you