Aggregations

CIP2017-04-13
(CIR-2017-188 and CIR-2017-219)

Tobias Lindaaker @ neo4j
The problem with Aggregations today

MATCH (x:L) RETURN x.a + count(x) + x.b + count(x) + x.c;

vs

MATCH (x:L) RETURN x.a + x.b + x.c + count(x) + count(x);
The problem with Aggregations today

MATCH (x:L) RETURN x.a + \texttt{count}(x) + x.b + \texttt{count}(x) + x.c;

\texttt{vs}

MATCH (x:L) RETURN x.a + x.b + x.c + \texttt{count}(x) + \texttt{count}(x);

1. How does a reader of the query know \textit{what is a function} and \textit{what is aggregation}?

2. If aggregation is part of an expression, \textit{how do we know what the grouping key is?}
How do we know what the grouping key is?

Given:

```sql
CREATE (:L {a:1,b:2,c:3}), (:L {a:2,b:3,c:1}), (:L {a:3,b:1,c:2});
```

<table>
<thead>
<tr>
<th>x.a + count(x) + x.b + count(x) + x.c</th>
<th>8</th>
<th>8</th>
<th>8</th>
<th>8</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>x.a + x.b + x.c + count(x) + count(x)</th>
<th>12</th>
</tr>
</thead>
</table>
Knowing what the grouping key is!

Force aggregation to be a top-level expression:

```sql
MATCH (x:L) WITH x.a + x.b + x.c AS v, count(x) AS c RETURN v + c + c
```

<table>
<thead>
<tr>
<th>v + c + c</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

```sql
MATCH (x:L) WITH x, count(x) AS c RETURN x.a + x.b + x.c + c
```

<table>
<thead>
<tr>
<th>x.a + x.b + x.c + c + c</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>8</td>
</tr>
</tbody>
</table>

+-------------------------+
Can we force aggregation to be top-level syntactically?

Turning the problem of knowing what the grouping key is into a problem of knowing what is a function and what is an aggregation
Syntactic differentiation for aggregation

Return = 'RETURN', ['DISTINCT'], ReturnBody, Filter;
With = 'WITH', ['DISTINCT'], ReturnBody, Filter;

ReturnBody = ('*' | ReturnItem), {',', ReturnItem};
ReturnItem = (Expression, ['AS', Variable])
                | (Aggregation, ['AS', Variable]);

Aggregation = Aggregator, 'OF', ['DISTINCT'], (Expression | '*');
Aggregator = {SymbolicName, '.'}, SymbolicName, [AggregatorParams];
AggregatorParams = '(', Expression, {',', Expression}, ')';

Filter = [Where], [Order], [Skip], [Limit];
Syntactic differentiation for aggregation

\[
\text{MATCH } (x:L) \text{ WITH } x.a + x.b + x.c \text{ AS } v, \text{ count OF } x \text{ AS } c \text{ RETURN } v + c + c
\]

Instead of:
\[
\text{MATCH } (p:Person) \text{ RETURN percentileCont}(p.age, 0.4)
\]
You would write:
\[
\text{MATCH } (p:Person) \text{ RETURN percentileCont}(0.4) \text{ OF } p.age
\]

Instead of:
\[
\text{MATCH } (p:Person)-[:KNOWS]-\{f\} \text{ RETURN } p, \text{ collect}(f) \text{ AS friends}
\]
You would write:
\[
\text{MATCH } (p:Person)-[:KNOWS]-\{f\} \text{ RETURN } p, \text{ list OF } f \text{ AS friends}
\]
Pros and Cons

- Deviation from familiar syntax from SQL
- Improvement in terms of readability

- A first step is to at least force aggregations to be top-level even if we still retain their syntax equal to `FunctionInvocation`
- If we want to *also* differentiate the syntax, the OF-syntax is one idea