OQL Semantics

Let’s talk algebra

Arūnas Janeliūnas

Object Databases

What is *algebra*?

Simply put, algebra consists of two parts:

- *Data* - what kind of elements we do consider?
- *Operations* - what operations we do perform on this data and what properties those operations have?

What is our *data*?

… or in other words:

So, what kind of elements we will manipulate in OQL algebra?

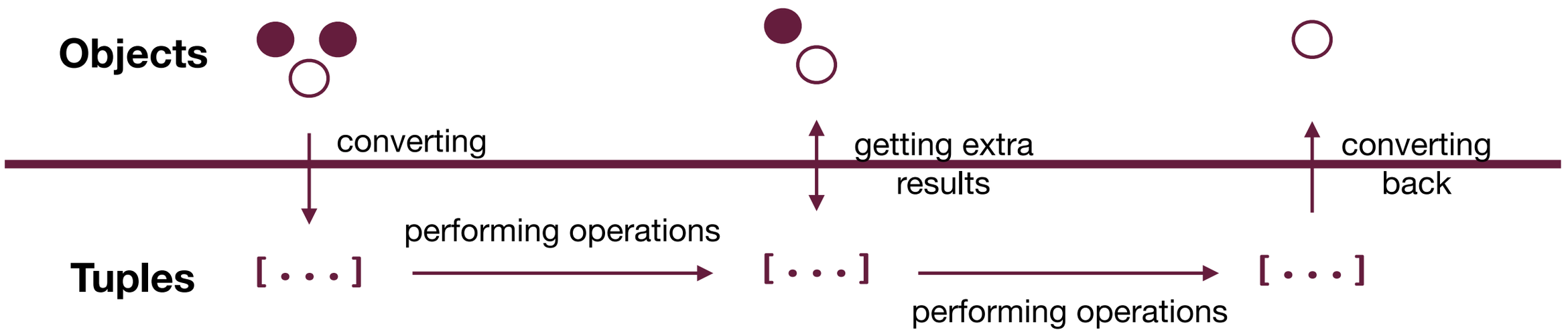
Let it be tuples of the form:

```
[y:some_object, x:some_value, . . .]
```

No methods in algebra…

Principal overview

OQL



Algebra

MAP

```
expr[a] = { [a:x] | x∈expr }
```

```
People[p]
```

gives us

```
{ [p:obj1], [p:obj2], … }
```

MAP

$$MAP_{a:expr2}(expr1) = \{\ x \otimes [a:expr2(x)] \mid x \in expr1\ \}$$

means „concatenate“

$$MAP_{n:p.name}(People[p])$$

gives us

$$\{\ [p:obj1, n:\text{“Adam“}],\ [p:obj2, n:\text{“Eve“}],\ \ldots\ \}$$

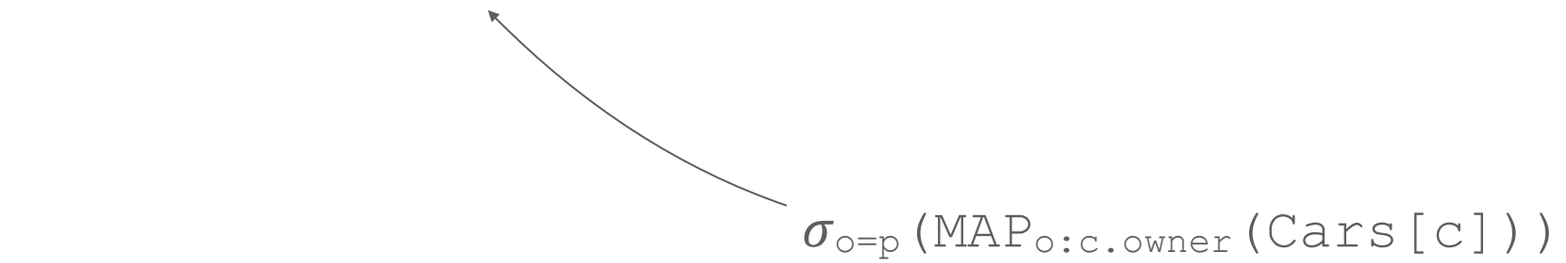
MAP

Calling methods:

$$MAP_{a:p.address.getStreet()}(People[p])$$

Adding sub-queries:

$$MAP_{pc:\ \sigma_{o=p}(MAP_{o:c.owner}(Cars[c]))}(People[p])$$



MAP

$$MAP_{expr2}(expr1) = \{\ expr2(x)\ |\ x \in expr1\ \}$$

$MAP_{n}(MAP_{n:p.name}(People[p]))$

gives us

{ "Adam", "Eve", … }

$MAP_{p.name}(People[p])$

would be quicker, don't you find?

Filtration (selection)

$$\sigma_{p}(\texttt{expr}) = \{\ \texttt{x}\ |\ \texttt{x} \in \texttt{expr} \wedge p(\texttt{x})=\texttt{true}\}$$

$$\sigma_{\texttt{n=„Arunas“}}(\texttt{MAP}_{\texttt{n:c.owner.name}}(\texttt{Cars[c]}))$$

gives us

```
{ [c:obj1,n:“Arunas“], [c:obj2,n:“Arunas“], … }
```

Join

$$expr1 \bowtie_p expr2 = \{ x1 \otimes x2 \mid x1 \in expr1 \wedge x2 \in expr2 \wedge p(x1,x2)=true \}$$

$$(MAP_{n:p.name}(People[p])) \bowtie_{on=n} (MAP_{on:c.owner.name}(Cars[c]))$$

gives us

```
{ [c:obj1,p:obj2,n:"Arunas",on:"Arunas"], … }
```

Dependent Join

$$expr1\langle expr2\rangle \{ x1 \otimes x2 \mid x1 \in expr1 \wedge x2 \in expr2(x2) \}$$

```
People[p]<p.cars[c]>
```

gives us

```
{ [p:obj1,c:obj11],[p:obj2,c:obj21],[p:obj2,c:obj22], … }
```

Sorting

$$\mathrm{Sort}_{A,\theta}(expr) = \{ x_1,\ldots,x_n \mid x_i \in expr \wedge x_i.A_k\ \theta_k\ x_{i+1}.A_k \}$$

$$\mathrm{Sort}_{\{n,a\},\{<,<\}}(\mathrm{MAP}_{a:p.getAge()}(\mathrm{MAP}_{n:p.name}(People[p])))$$

Matematikos
ir informatikos
fakultetas

Arūnas Janeliūnas

Object Databases

Grouping

$$\Gamma_{g,A,\theta,f}(\text{expr}) = \{\ x.A\otimes[g:\text{group}]\ |\ x\in\text{expr}\ \wedge\ \text{group}=f(\{y|y\in\text{expr}\ \wedge\ y_i.A_k\ \theta_k\ x_i.A_k\ \}$$

$$\Gamma_{\text{partition},\{n\},\{=\},\text{Id}}(\text{MAP}_{n:p.\text{name}}(\text{People}[p]))$$

gives us

```
{ [n:"Arunas",partition:{[p:obj2,n:"Arunas"],…}],
  [n:"Sigitas",partition:{[p:obj7,n:"Sigitas"],…}],
   … }
```

Query translation

General query form for the „iteration query“:

```
select s
  from x1 in f1, ..., xn in fn
 where p
group by a1:c1, ... , am:cm
having q
order by o1, ... , ok
```

Step 1

```
select s
     from x1 in f1, ..., xn in fn
    where p
 group by a1:c1, ... , am:cm
   having q
 order by o1, ... , ok
```

Step 1: FROM

$F = f_1[x_1] <f_2[x_2]> \ldots <f_n[x_n]>$

Here we use either Dependent Joins or simple Joins depending on the expressions f_i

Step 2

```
select s
    from x1 in f1, ..., xn in fn
   where p
group by a1:c1, ... , am:cm
  having q
order by o1, ... , ok
```

Step 2: WHERE

$$W = \sigma_{p(v_1,\ldots,v_w)}(\mathrm{MAP}_{v_1:m_1,..,v_w:m_w}(F))$$

First we map all sub-queries results as additional attributes v_i and then filter the output set by the predicate p

Step 3

```
select s
    from x1 in f1, ..., xn in fn
   where p
group by a1:c1, ... , am:cm
  having q
order by o1, ... , ok
```

Step 3: GROUP BY

$$G = \Gamma_{\texttt{partition},\{a_1,\ldots,a_w\},\{=,\ldots,=\},\texttt{Id}}(\texttt{MAP}_{a_1:c_1,..,a_w:c_w}(W))$$

First we map all sub-queries results as additional attributes a_i and then filter the output set by the predicate p

Step 4

```
select s
    from x1 in f1, ..., xn in fn
   where p
group by a1:c1, ... , am:cm
  having q
order by o1, ... , ok
```

Step 4: HAVING

$$H = \sigma_{q(h_1,\ldots,h_m)}(\mathrm{MAP}_{h_1:g_1,..,h_m:g_m}(G))$$

Once again, first we map all needed sub-queries as attributes h_i and then filter the output set by the predicate q

Step 5

```
select s
    from x1 in f1, ..., xn in fn
   where p
group by a1:c1, ... , am:cm
  having q
order by o1, ... , ok
```

Step 5: ORDER BY

$$S = \mathrm{Sort}_{\{o_1,\ldots,o_k\},\{\ldots\}}(\mathrm{MAP}_{o_1:s_1,..,o_k:s_k}(H))$$

Matematikos
ir informatikos
fakultetas

Arūnas Janeliūnas

Step 6

```
  select s
      from x1 in f1, ..., xn in fn
     where p
  group by a1:c1, ... , am:cm
    having q
  order by o1, ... , ok
```

Step 5: SELECT

$$\texttt{Result} = \texttt{MAP}_{\texttt{s}}(\texttt{S})$$

Final `MAP` operation is designed to get the desired set of query result

Query translation

Even more straightforward translation for simple queries.

For example:

```
BigBoss.address.getCity()
```

may be translated simply as

$$\texttt{MAP}_{\texttt{bb.address.getCity()}}\texttt{(BigBoss[bb])}$$

Complex example

```
select struct ( age: a,
                cnt: count(partition)
              )
  from l in Lecturers,
       c in l.courses
 where c.title = “ODB”
group by a:l.getAge()
```

Step 1: FROM

```
F = Lecturers[l] <l.courses[c]>
```

gives us

```
{ [l:obj1,c:obj11],[l:obj2,c:obj21],[l:obj2,c:obj22], … }
```

Step 2: WHERE

$$W = \sigma_{t=\text{„ODB“}}(MAP_{t:c.title}(F))$$

gives us

```
{ [l:obj1,c:obj11,t:“ODB“],[l:obj2,c:obj22,t:“ODB“], … }
```

Step 3: GROUP BY

$$G = \Gamma_{\texttt{partition,\{a\},\{=\},Id}}(\texttt{MAP}_{\texttt{a:1.getAge()}}(W))$$

gives us

```
{ [a:42, partition:{[l:obj1,c:obj11,t:"ODB",a:42],…} ,
  [a:53, partition:{[l:obj9,c:obj91,t:"ODB",a:53],…}
  …
}
```

Step 4: SELECT

```
Result = MAP[age:a,cnt:count(partition)](G)
```

gives us

```
{
   [age:42,cnt:2],
   [age:53,cnt:3],
   …
}
```