PartiQL: Новый язык запросов от компании Amazon

Павел Велихов
Huawei

План доклада

- История PartiQL, N1Q1 и SQL++
- Обзор PartiQL
- Основные отличия от XQuery / JSONiq
- Основные отличия от PostgreSQL

История PartiQL

- 2014: первая версия SQL++ в системе FORWARD, UCSD, Yannis Papakonstantinou
- 2015: AsterixDB поддерживает SQL++ (UC Irvine, Mike Carey)
- 2015: Couchbase выпускает N1QL, далее переходит на SQL++
- 2019: первая имплементация PartiQL в Amazon

Обзор PartiQL

- Модель данных
- Общая структура запросов PartiQL
- Select и Select value
- Pivot
- Запросы на вложенных структурах данных
- Нулевые значения (null, missing)
- Group as
- Снятие ограничений SQL на вложенные подзапросы

PartiQL модель данных

- Своя собственная модель данных PartiQL Data Model (супермножество JSON и SQL)
  - примитивные типы из стандарта SQL
  - массивы значений: [value_1, value_2,…]
  - “мешки” значений: << value_1, value_2, value_1,… >>
  - кортежи (неупорядоченные) : {‘key_1’:value_1, ‘key_2’:value_2, … }
  - При этом реляционные таблицы отображаются в PartiQL:
  - [
    - {‘attribute_1’:value_1, ‘attribute_2’:value_2 … },
    - …
  -]
  - JSON отображается напрямую в PartiQL

Общая структура запроса PartiQL

- SQL-совместимый синтаксис

```
WITH …
SELECT expr_1, … expr_N
FROM table_1 as alias_1 …
WHERE expr
GROUP BY group_by_list
HAVING expr
```

- В отличие от SQL, PartiQL может возвращать кортежи с произвольной структурой
- Также, вместо списка кортежей, PartiQL может возвращать список объектов

Дополнительные конструкции PartiQL и SQL++

- LET clause

```
SELECT e.name
FROM employee e
LET yearly_salary = e.salary * 12
```

- Кванторы существования и всеобщности

```
SELECT e.name
FROM employee e
WHERE EVERY job IN e.jobs
SATISFIES job.employer = 'Amazon'
```

-

Select и Select Value

```
SELECT e.name
FROM employee e


<<{'name':'John'},
 {'name':['Tim','Gates']}>>
```

```
employee = <<
   {'name':'John',
    'salary':10000,
    'department':'IT'},
   {'name':['Tim','Gates'],
    'salary':25000}
>>
```

```
SELECT VALUE e.name
FROM employee e


 <<'John', ['Tim','Gates']>>
```

Pivot и Unpivot

```
PIVOT s.price AT s.symbol
FROM symbols s
```

```
{‘tdc’:313.52, ‘amzn’:840.05}
```

```
symbols:<<
   {‘symbol’:’’tdc','price':31.52},
   {‘symbol':'amzn','price':840.05}
>>
```

```
SELECT symbol as symbol,
       price as price
FROM trades t
UNPIVOT t AS price AT symbol
WHERE symbol != ‘date’
```

```
<< {‘symbol’:’tdc’,’price’:31.52},
   {‘symbol’:’amzn’,’price’:840.05}
 >>
```

```
trades:<<
   {‘date’:’4/1/2019’,’tdc’:31.52,
     ‘amzn’:840.05}
>>
```

PartiQL From Clause

```
SELECT *
FROM employee e,
     dept d
WHERE e.name = d.manager

<< {'e': {'name':'John',…},
    'd':{'name':'IT',…}} >>
```

```
employee = <<
  {'name':'John',
   'salary':10000,
   'department':'IT'},
  {'name':['Tim','Gates'],
   'salary':25000}
>>

dept = <<
  {'name':'IT',
   'manager':'John'},
  {'name':'Accounting',
   'managers':['Scott','Jane']}
>>
```

```
SELECT *
FROM dept AS d AT y

<< {'d': {'name':'IT',…}, y:0},
   {'d': {'name':'Accounting',…}, y:1}
>>
```

Вложенные данные

```
SELECT ms
FROM dept AS d,
     d.managers as ms

<<{'ms':[{'name':'Scott',…},
         {'name':'Jane'}]} >>
```

```
SELECT ms[0] AS m0
FROM dept AS d,
     d.managers as ms

<<{'m0':{'name':'Scott',…}} >>
```

```
dept = <<
  {'name':'IT',
   'manager':'John'},
  {'name':'Accounting',
   'managers':[
     {'name':'Scott',
      'grade':17},
     {'name':Jane'}
   ]
  }
>>
```

Вложенные данные

```
SELECT ms[0].name as n
FROM dept AS d,
     d.managers as ms

<<{'n':'Scott'}>>
```

```
dept = <<
  {'name':'IT',
   'manager':'John'},
  {'name':'Accounting',
   'managers':[
     {'name':'Scott',
      'grade':17},
     {'name':Jane'}
   ]
  }
>>
```

```
SELECT name
FROM dept AS d,
     d.managers[0].name as name

<<{'name':'Scott'}>>
```

Вложенные данные

```
SELECT m
FROM dept AS d,
     d.managers as ms,
     m in ms

<<{m:{'name':'Scott',…}},
  {m:{'name':'Jane'}} >>
```

```
dept = <<
  {'name':'IT',
   'manager':'John'},
  {'name':'Accounting',
   'managers':[
     {'name':'Scott',
      'grade':17},
     {'name':Jane'}
   ]
  }
>>
```

Вложенные данные

```
SELECT d, m
FROM dept AS d,
     LEFT JOIN d.managers as ms,
     LEFT JOIN m in ms
```

```
dept = <<
   {'name':'IT',
    'manager':'John'},
   {'name':'Accounting',
    'managers':[
      {'name':'Scott',
       'grade':17},
      {'name':Jane'}
    ]
   }
>>
```

```
<< {d:{'name':'IT',…},
    {d:{'name':'IT',…}, m:{'name':'Scott',…}},
    {d:{'name':'IT',…}, m:{'name':'Jane'}} >>
```

Создание вложенных данных

```
SELECT VALUE
    {'emp-dept':
        {'employee':e},
        {'dept':d}}
FROM employee AS e
     dept as d
WHERE e.name = dept.name
```

```
<< {'emp-dept':
     {'employee':
       {'name':'John',…},
      'dept':
       {'name':'IT,…},
     }
    }
>>
```

```
employee = <<
  {'name':'John',
   'salary':10000,
   'department':'IT'},
  {'name':['Tim','Gates'],
   'salary':25000}
 >>

dept = <<
  {'name':'IT',
   'manager':'John'},
  {'name':'Accounting',
   'managers':['Scott','Jane']}
>>
```

NULL и Missing

- Кроме традиционного NULL в PartiQL модель данных вводится MISSING
- Из путевых выражений и при доступе к массивам мы можем получить MISSING значение
- Также, операции с неправильными типами данных приводят к MISSING значениям
- MISSING ведет себя почти как NULL, кроме отдельных предикатов и конструкции новый объектов

NULL и MISSING

- Обычные операции над значениями идентичны семантики NULL значений в SQL, например следующие выражения возвращают MISSING:
  - `5+missing, 5 > ‘a’, NOT {a:1}`
- Оператор = в PartiQL глубокий, и при сравнении массивов NULL приравнивается к MISSING
- Если имя атрибута или его значение MISSING - то этот атрибут не попадает в кортеж при его создании в SELECT
- При этом при создании массивов MISSING значения попадают в новый массив

Примеры MISSING при создании элементов

```
SELECT t.x + 1 AS y
FROM [{‘x’:1}, {‘x’:’a’}, {‘z’:1}] t
```

```
<< {‘y’:2} >>
```

```
SELECT VALUE [t.x + 1]
FROM [{‘x’:1}, {‘x’:’a’}, {‘z’:1}] t
```

```
<< [2], [MISSING], [MISSING] >>
```

Group AS

```
SELECT VALUE g
FROM customers AS c, orders AS o
WHERE c.cust_id = o.cust_id
GROUP BY c.address.zipcode
GROUP AS g
```

```
<< [{‘c’:c1, ‘o’:o1}, {‘c’:c2, ‘o’,o2}],
   [{‘c’:c3, ‘o’:o3}],
   …
>>
```

Group AS

```
SELECT zip, `best rating`, `best customers`
FROM customers AS c
GROUP BY c.address.zipcode AS zip
GROUP AS g
LET `best rating` = MAX(c.rating),
    `best customers` =
       (SELECT gi.c.custid, gi.c.name
        FROM g AS gi
        WHERE gi.c.rating = `best rating`
        ORDER BY gi.c.custid)
ORDER BY zip;
```

Снятие ограничений вложенных запросов в SQL

- В SQL коррелированные подзапросы ограничены - они не могут использовать данные друг-друга
- В PartiQL и SQL++ эти ограничения сняты
- Например:

```
SELECT c, o
FROM customer as c,
   (SELECT o
    FROM order as o
    WHERE o.id = c.id ) AS o
```

Правила коэрции вложенных запросов

- Вложенный запрос PartiQL приводится к атомарному типу или типу массив

```
SELECT v.foo,(SELECT w.bar
              FROM someDataSet w
              WHERE w.sth = v.sthelse) AS bar
 FROM anotherDataSet v
```

```
SELECT VALUE {'foo': v.foo'bar': COLL_TO_SCALAR(
                                 SELECT VALUE {'bar': w.bar}
                                 FROM someDataSet w
                                 WHERE w.sth = v.sthelse)}
FROM anoterDataSet v
```

Сравнение с XQuery

- Преимущества XQuery:
  - Рекурсивные путевые выражения
  - Исключения и их обработка
    - Отсутсвие понятия NULL и связанной с ними семантики
    - Более чистая семантика GROUP BY
- Преимущества PartiQL:
  - Совместимость с SQL
  - Независимость семантики от схемы данных

Сравнение с PostgreSQL

- В PostgreSQL имплементированы только путевые выражения
- Основные операторы -> и ->> возвращают JSON объекты или текст, работают как сверху массивов так и объектов
-

->	[{"a":"foo"},{"b":"bar"},	{"c":"baz"}
->	{"a": {"b":"foo"}}'::json->'a'	{"b":"foo"}
->>	[1,2,3]'::json->>2	3
->>	{"a":1,"b":2}'::json->>'b'	2
#>	{"a": {“b":{"c": "foo"}}}'::json#>'{a,b}'	{"c": "foo"}
#>>	{"a":[1,2,3],"b":[4,5,6]}'::json#>>'{a,2}'	3

Сравнение с PostgreSQL

- Дополнительные операторы: @>, <@, ?, ?|, ?&, ||, -, #-
- 23 функции для различных операций над JSON, включая конструкцию объектов
- Все это можно сделать в PartiQL с помощью общего языка запросов

Другие No-SQL языки

- GraphQL
- MongoDB
- Jaql