SQL++

Friday, April 26, 2019 1:28 PM

- AsterixDB
  - ADM Derived Types
    - No repeated fields for objects.
    - Ordered array, and can be heterogenous.

      [1, 2, "kaiser", 3]
    - Multisets (bags)
      - {{1, 3, "k", 4}}
      - are converted into ordered arrays (in arbitrary order) when returned at the end.
- Datatypes
  - Boolean, integer, float, geometry, UUID (universally unique identifier)
  - NULL, Missing

    {"attrib": missing} = {} ⇒ really missing

    {"age": null} = NULL like null in SQL.
  - Derived types
    - object {"attr_name1": val, "attr_n2": val ...}

- (Ordered) Array [elet1 , elet2 , elet3 ...]
- (Unordered) Multiset {{ elet1, elect2, ... }}

e.g., select x.phone
from [{"name": "Alice", "phone":[30, 150] }] as x

⇐> from {{ { } }} as x

error! because can't put a single object.
from { } as x

- SQL++ {
  - Data Definition Language (create table, ...) Type, Dataset, Dataverse, Index...
  - Data Manipulation Language (select, from, ...)

★ SQL++ As a data definition language

- Dataverse is a Database.
  - create dataverse myDB [if not exists];
    drop dataverse myDB [if exists];
    use myDB;
- Type : defines the schema of a relation; lists all required fields
  - field_name ? is optional field
  - [CLOSED] type ... the fields allowed

- CLOSED type = no other fields allowed

  OPEN type = other fields allowed

  e.g.,

  ```
  use myDB;
  DROP TYPE PersonType IF EXISTS;
  CREATE TYPE PersonType AS CLOSED or OPEN {
          name : string,
          age : int,
          email : string?,
          address : [string]
  }
  ```

  - If closed, the following is not allowed; If open, it's allowed.

    {"name": "Carol", "address": ["k", "Pea"], "phone": "123456"}

  - A PersonType can either have an email.

- Types within Types

  - The field in Type1 may be of type Type2

    but remember to declare Type2 earlier.

- **Datasets = relation/table**

  - must have a type (can be OPEN) and a key.

  - follow previous type code

    ```
    DROP DATASET Person IF EXISTS;

    CREATE DATASET Person(PersonType) PRIMARY KEY Name;
    ```

  - automatically generated key: uuid

    ```
    CREATE TYPE PersonType AS CLOSED{
        myKey : uuid,
    ```

} Don't need to insert mykey when insert values.

```
CREATE DATASET Person(PersonType) PRIMARY KEY mykey AUTOGENERATED;
```

- JSON is NFNF (Non-first-Normal-form)
  - JSON has nests, is not necessarily flat

- Indexes
  - A way to access our data efficiently
  - Can declare an index on an top-level type attribute

    i.e. the type used by the data set.
  - BTREE: Good for equality and range queries

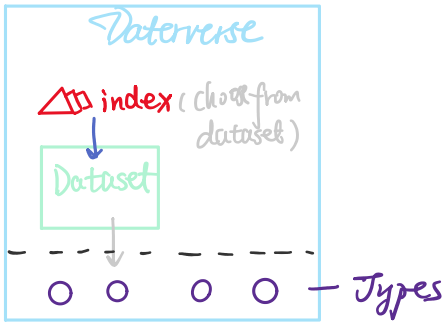
    RTREE: Good for 2-dimensional range queries

    KEYWORD good for substring search
  - e.g.,

```
USE myDB;
CREATE INDEX countryID
ON country('-car_code')
TYPE BTREE;
```

  - cannot index inside a nested collection.

- AsterixDB Data Model Recap



- Data Manipulation

Array is ordered: "[]"
Multiset is not: "{}"

Unnesting collections

mydata

```
{"A": "a1",   "B": [{"C": "c1", "D": "d1"}, {"C": "c2",  "D": "d2"} ]}
{"A": "a2",   "B": [{"C": "c3", "D": "d3"}] }
{"A": "a3",   "B": [{"C": "c4", "D": "d4"}, {"C": "c5",  "D": "d5"} ]}
{"A": "a4"}
```

```
SELECT x.A, y.C, y.D
FROM mydata AS x, x.B AS y;
```

Remember that they need to match

Equivalent to UNNEST

Form cross product between each x and its x.B

Answer

```
{"A": "a1", "C": "c1", "D": "d1"}
{"A": "a1", "C": "c2", "D": "d2"}
{"A": "a2", "C": "c3", "D": "d3"}
{"A": "a3", "C": "c4", "D": "d4"}
{"A": "a3", "C": "c5", "D": "d5"}
```

- select x
  from name1 as x1, name1.name2 as x2 (multiset or array to be iterated)
  where x1.y = x2.p (can not evaluate to arrays)

- Find each country's GDP

world

```
{{ {"mondial":
    {"country":
     [{"-car_code":"AL", …}
       {"name":"Albania"}, …
       ], ...
    }, ...
}}
```

country

```
{{ { "-car_code":"AL",
     "gdp_total":4100,
     ...
   }, ...
}}
```

```
SELECT x.mondial.country.name, c.gdp_total
```

```
FROM world AS x, country AS c
WHERE x.mondial.country.`-car_code` = c.`-car_code`;
```

a type of array

can't access "field" -car_code

Right:

```
select y.name, c.gdp_total
from world as x, world.mondial.country as y, country as c
where y.'-car_code' = c.'car_code';
```

unnest an array

- Type mismatch

↓

for the same attribute, it might have a value of array or object.

- IS_ARRAY(attribute_name) // Check if it's array

↓

- Useful functions:

Q: if is_multiset() ?

is_array()

is_boolean()

is_number()

is_object()

is_string()

is_null()

is_missing()

is_unknown = is_null or is_missing

- if-else in SQL++

```
(CASE WHEN condition1 THEN operation1
      ⋮
      WHEN           THEN
      ELSE operation END)
```

e.g.
```
select
from world x x.mondial.country as y, y.province as z
    (CASE WHEN z.city IS missing then []
          WHEN IS_ARRAY(z.city) THEN z.city
          ELSE [z.city]) AS u
where y.name = "Greece";
```

- Unnesting

  - Basic unnesting

    e.g. arr = [[a,b],[],[b,c,d]]

    unnest(arr) = [a,b,b,c,d]

```
select y
from arr x, x y;
```

e.g.2., coll = [{A:a1, F:[{B:b1},{B:b2}]}, ···]

unnest(coll) = [{A:a1, B:b1}, {A:a1, B:b2} ··]

the F here matches with from

```
select x.A, y.B, x.G
from coll x, x.F y.
```

- Nesting

LET relation_name = (subquery) Q: with ··· as ?

- Grouping and Aggregates Notes

  - Count the number of elements in array F for each A.

```
select x.A, strict_count(x.F) as cnt
from C as x
```

Method I. A is not a key, then output multiple times

Method II. F is empty, return 0 times

```
---------
select x.A, count(*) as cnt
from C as x, x.F as y
group by x.A
```

- Joins : works as SQL
- Ordering → atomic value

  e.g., "900" > "8000" but 900 < 8000.

  - Datatype matter.

- Splitting

```
split(string_name, " s ")   // return an array, which is formed by string s.
```

- Behind the Scenes Notes

  Query Processing on NFNF data