Introduction to Database Systems CSE 414

Lecture 14: SQL++

Announcements

- HW5 + WQ5 released
  - Both due in 1 week
  - Post on piazza ("asterix" / "hw5")
  - Check out the SQL++ tutorial on course website!
    - Authoritative reference on SQL++
    - Written by Don Chamberlin
    - Do not distribute outside of class

Asterix Data Model (ADM)

- Based on the Json standard
- Objects:
  - `{"Name": "Alice", "age": 40}`
  - Fields must be distinct:
    `{"Name": "Alice", "age": 40, ~~"age":50~~}`

Can't have repeated fields

- Ordered arrays:
  - `[1, 3, "Fred", 2, 9]`
  - Can contain values of different types
- Multisets (aka bags):
  - `{{1, 3, "Fred", 2, 9}}`
  - Mostly internal use only but can be used as inputs
  - All multisets are converted into ordered arrays (in arbitrary order) when returned at the end

Datatypes

- Boolean, integer, float (various precisions), geometry (point, line, …), date, time, etc
- UUID = universally unique identifier
  Use it as a system-generated unique key

Closed Types

```
USE myDB;
DROP TYPE PersonType IF EXISTS;
CREATE TYPE PersonType AS CLOSED {
   name: string,
   age: int,
   email: string?
}
```

```
{"name": "Alice", "age": 30, "email": "a@alice.com"}

{"name": "Bob", "age": 40}

-- not OK:
{"name": "Carol", ~~"phone": "123456789"~~}
```


Open Types

```
USE myDB;
DROP TYPE PersonType IF EXISTS;
CREATE TYPE PersonType AS OPEN {
   name: string,
   age: int,
   email: string?
}
```

```
{"name": "Alice", "age": 30, "email": "a@alice.com"}

{"name": "Bob", "age": 40}

-- now it's OK:
{"name": "Carol", "age":20, ~~"phone": "123456789"~~}
```

Types with Nested Collections

```
USE myDB;
DROP TYPE PersonType IF EXISTS;
CREATE TYPE PersonType AS CLOSED {
   Name : string,
   phone: [string]
}
```

```
{"Name": "Carol", "phone": ["1234"]}
{"Name": "David", "phone": ["2345", "6789"]}
{"Name": "Evan",  "phone": []}
```


Datasets

- Dataset = relation
- Must have a type
  - Can be a trivial OPEN type
- Must have a key
  - Can also be a trivial one

Dataset with Existing Key

```
USE myDB;
DROP TYPE PersonType IF EXISTS;
CREATE TYPE PersonType AS CLOSED {
   name: string,
   email: string?
}
```

```
{"name": "Alice"}
{"name": "Bob"}
…
```

```
USE myDB;
DROP DATASET Person IF EXISTS;
CREATE DATASET Person(PersonType) PRIMARY KEY Name;
```


Dataset with Auto Generated Key

```
USE myDB;
DROP TYPE PersonType IF EXISTS;
CREATE TYPE PersonType AS CLOSED {
   myKey: uuid,
   Name : string,
   email: string?
}
```

```
{"name": "Alice"}
{"name": "Bob"}
…
```

Note: no myKey inserted as it is autogenerated

```
USE myDB;
DROP DATASET Person IF EXISTS;
CREATE DATASET Person(PersonType)
    PRIMARY KEY myKey AUTOGENERATED;
```


This is no longer 1NF

- NFNF = Non First Normal Form
- One or more attributes contain a collection
- One extreme: a single row with a huge, nested collection
- Better: multiple rows, reduced number of nested collections

Example from HW5

mondial.adm is totally semi-structured:

```
{"mondial": {"country": [...], "continent":[...], ..., "desert":[...]}}
```

country	continent	organization	sea	...	mountain	desert
[{"name":"Albania",...}, {"name":"Greece",...}, ...]	...	...	...		...	...

Nested objects!

country.adm, sea.adm, mountain.adm are more structured

Country:

-car_code	name	...	ethnicgroups	religions	...	city
AL	Albania	...	[...]	[...]	...	[...]
GR	Greece	...	[...]	[...]	...	[...]
...	...	...	...			

Indexes

- Can declare an index on an attribute of a top-most collection
- Available options:
  - `BTREE`: good for equality and range queries
    E.g., `name="Greece"; 20 < age and age < 40`
  - `RTREE`: good for 2-dimensional range queries
    E.g., `20 < x and x < 40 and 10 < y and y < 50`
  - `KEYWORD`: good for substring search if your dataset contains strings
- Will discuss how they help later in the quarter

Indexes

Cannot index inside a nested collection

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

```
USE myDB;
CREATE INDEX cityname
     ON country(city.name)
     TYPE BTREE;
```

Country:

-car_code	name	...	ethnicgroups	religions	...	city
AL	Albania	...	[...]	[...]	...	[...]
GR	Greece	...	[...]	[...]	...	[...]
...	...	...	...			
BG	Belgium	...				
...						

SQL++ Overview

```
SELECT ...
FROM ...
WHERE ...
GROUP BY ...
HAVING ...
ORDER BY ...
```


world

```
{{ {"mondial":
    {"country": [{Albania}, {Greece}, …],
     "continent": […],
     "organization": […],
     ...
     ...
    }
  }
}}
```

Retrieve Everything

A collection of objects

```
SELECT x.mondial FROM world AS x;
```

2. Return `mondial` for each x

1. Bind each object in `world` to x

Answer

```
{"mondial":
    {"country": [ Albania, Greece, …],
     "continent": […],
     "organization": […],
    ...
    ...
     }
}
```


world

```
{{ {"mondial":
    {"country": [{Albania}, {Greece}, …],
     "continent": […],
     "organization": […],
     ...
     ...
    }
  }
}}
```

Retrieve Everything

```
SELECT x.mondial AS ans FROM world AS x;
```

Answer

```
{"ans":
    {"country": [{Albania}, {Greece}, …],
     "continent": […],
     "organization": […],
    ...
    ...
     }
}
```


world

```
{{ {"mondial":
    {"country": [{Albania}, {Greece}, …],
     "continent": […],
     "organization": […],
     ...
     ...
    }
  }
}}
```

Retrieve countries

```
SELECT x.mondial.country FROM world AS x;
```

Answer

```
{"country": [{Albania}, {Greece}, …], ...}
```

world

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

country

```
{{ { “-car_code”:”AL”,
     “gdp_total”:4100,
     ...
   }, ...
}}
```

Find each country’s GDP

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


world

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

country

```
{{ { “-car_code”:”AL”,
     “gdp_total”:4100,
     ...
   }, ...
}}
```

Find each country’s GDP

“-car_code” is an illegal field name
Escape using ` ... `

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


world

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

country

```
{{ { “-car_code”:”AL”,
     “gdp_total”:4100,
     ...
   }, ...
}}
```

Find each country’s GDP

```
SELECT x.mondiao.country.name, c.gdp_total
FROM world AS x, country AS c
WHERE x.mondial.country.`-car_code` = c.`-car_code`;
```

x.mondial.country is an array of objects. No field as -car_code!

```
Error: Type mismatch!
```

Need to “unnest” the array

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"} ]}
```

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

Iterate over each x and bind each object in x.B to y

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"} ]}
```

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

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"}
```


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"} ]}
```

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

Same as before

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"}
```

Find each country's GDP

world

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

country

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

```
SELECT y.name, c.gdp_total
FROM world AS x, x.mondial.country AS y, country AS c
WHERE y.`-car_code` = c.`-car_code`;
```

Answer

```
{ "name": "Albania", "gdp_total": "4100" }
{ "name": "Greece", "gdp_total": "101700" }
...
```


Return province and city names

world

```
{{ {"mondial":
    {"country": [{Albania}, {Greece}, …],
     "continent": […],
     "organization": […],
     ...
     ...
    }
   }
}}
```

```
SELECT z.name AS province_name, u.name AS city_name
FROM world x, x.mondial.country y, y.province z, z.city u
WHERE y.name = "Greece";
```

The problem: Error: Type mismatch!

```
"name": "Greece",
"province": [ ...
    {"name": "Attiki",
     "city": [ {"name": "Athens"...}, {"name": "Pireus"...}, ...]
      ...},
     {"name": "Ipiros",
     "city": {"name": "Ioannia"...}
      ...}, ...
```

city is an array

city is an object

Return province and city names

world

```
{{ {"mondial":
    {"country": [{Albania}, {Greece}, …],
     "continent": […],
     "organization": […],
     ...
     ...
    }
   }
}}
```

```
SELECT z.name AS province_name, u.name AS city_name
FROM world x, x.mondial.country y, y.province z, z.city u
WHERE y.name="Greece" AND IS_ARRAY(z.city);
```

The problem:

```
"name": "Greece",
"province": [ ...
    {"name": "Attiki",
     "city": [ {"name": "Athens"...}, {"name": "Pireus"...}, ...]
      ...},
     {"name": "Ipiros",
     "city": {"name": "Ioannia"...}
      ...}, ...
```

city is an array

city is an object

Return province and city names

world

```
{{ {"mondial":
    {"country": [{Albania}, {Greece}, …],
     "continent": […],
     "organization": […],
     ...
     ...
    }
   }
}}
```

Note: get name directly from z

```
SELECT z.name AS province_name, z.city.name AS city_name
FROM world x, x.mondial.country y, y.province z
WHERE y.name="Greece" AND NOT IS_ARRAY(z.city);
```

The problem:

```
"name": "Greece",
"province": [ ...
    {"name": "Attiki",
     "city": [ {"name": "Athens"...}, {"name": "Pireus"...}, ...]
      ...},
     {"name": "Ipiros",
     "city": {"name": "Ioannia"...}
      ...}, ...
```

city is an array

city is an object

Return province and city names

world

```
{{ {"mondial":
    {"country": [{Albania}, {Greece}, …],
     "continent": […],
     "organization": […],
     ...
     ...
    }
   }
}}
```

```
SELECT z.name AS province_name, u.name AS city_name

FROM world x, x.mondial.country AS y, y.province AS z,

     (CASE WHEN IS_ARRAY(z.city) THEN z.city
      ELSE [z.city] END) AS u

WHERE y.name="Greece";
```

Get both!

Return province and city names

world

```
{{ {"mondial":
    {"country": [{Albania}, {Greece}, …],
     "continent": […],
     "organization": […],
     ...
     ...
    }
   }
}}
```

```
SELECT z.name AS province_name, u.name AS city_name
FROM world x, x.mondial.country y, y.province z,

     (CASE WHEN z.city IS missing THEN []
           WHEN IS_ARRAY(z.city) THEN z.city
           ELSE [z.city] END) AS u

WHERE y.name="Greece";
```

Even better

Useful Functions

- `is_array`
- `is_boolean`
- `is_number`
- `is_object`
- `is_string`
- `is_null`
- `is_missing`
- `is_unknown = is_null or is_missing`