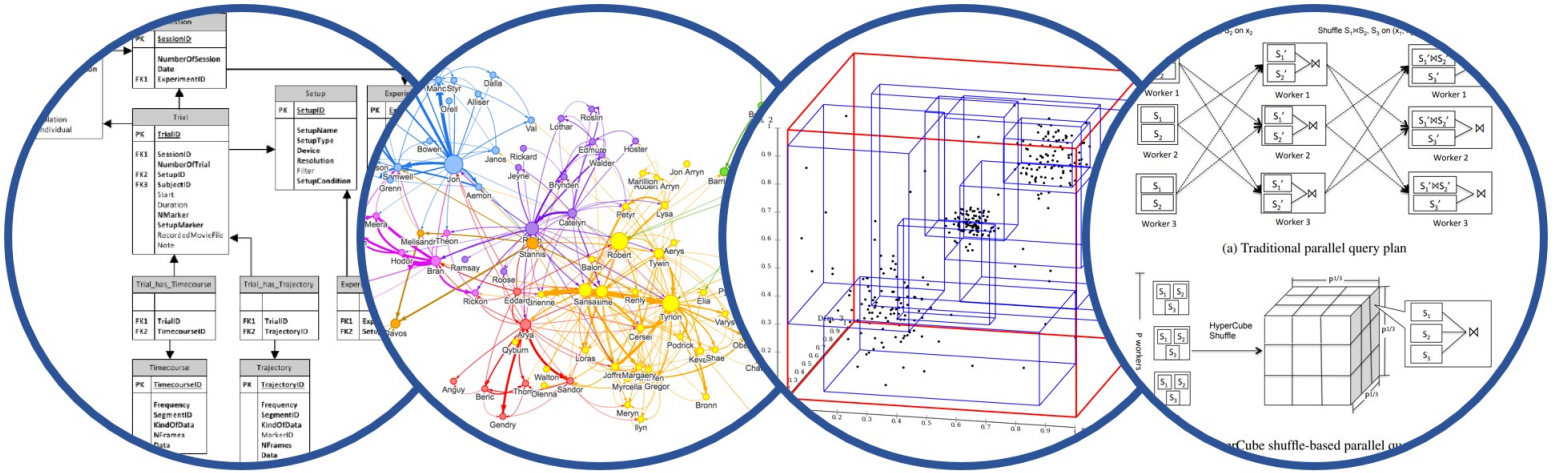


Introduction to Data Management

SQL++

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Announcements

- HW6/2 due tonight (Friday, May 24)
- HW7 to be released on Friday, due on May 31
- **No lecture on Monday, May 27 (Memorial Day)**

Outline

Last time:

- Json, Asterix Data Model (ADM), simple SQL++

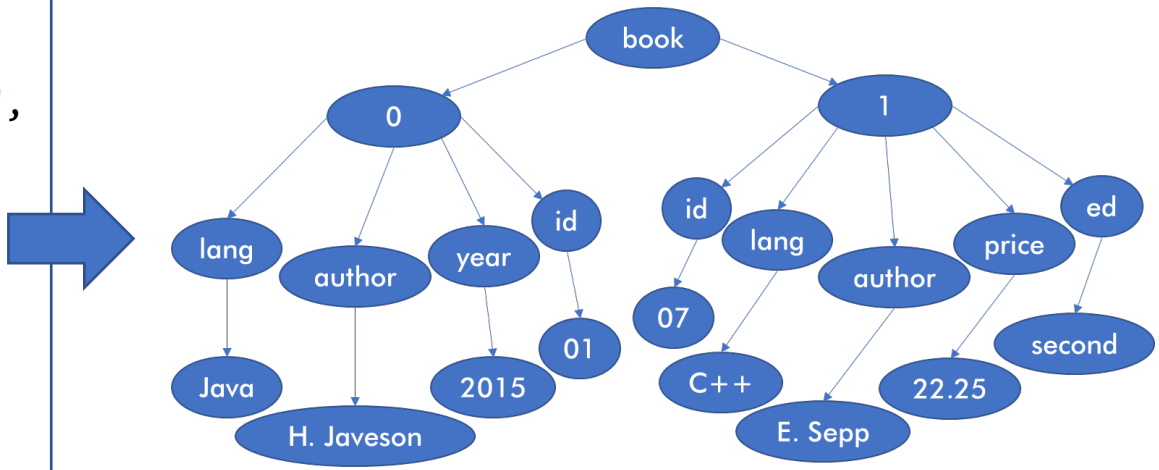
Today:

- Advanced SQL++
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)

Recap: Semi-Structured Data Key Features

- Tree-like data
- Embedded schema

```
{
   "book":[
      {
         "id": "01",
         "language": "Java",
         "author": "H. Javeson",
         "year": 2015
      },
      {
         "author": "E. Sepp",
         "id": "07",
         "language": "C++",
         "edition": "second",
         "price": 22.25
      }
   ]
}
```



Recap: Simple SQL++

```
SELECT x.phone
  FROM [
         {"name": "Dan", "phone": [300, 150]},
         {"name": "Alvin", "phone": 420}
       ] AS x;
```

Recap: Simple SQL++

```
SELECT x.phone
  FROM [
         {"name": "Dan", "phone": [300, 150]},
         {"name": "Alvin", "phone": 420}
       ] AS x;

-- output, same for-loop semantics like in SQL
-- array data
/*
{ "phone": [300, 150] }
{ "phone": 420 }
*/
```

Recap: Simple SQL++

```
SELECT x.phone
  FROM {{
         {"name": "Dan", "phone": [300, 150]},
         {"name": "Alvin", "phone": 420}
       }} AS x;
```

Recap: Simple SQL++

```
SELECT x.phone
  FROM {{
          {"name": "Dan", "phone": [300, 150]},
          {"name": "Alvin", "phone": 420}
       }} AS x;

-- same output as array data
-- multiset data
```

Recap: Simple SQL++

```
-- error
SELECT x.phone
  FROM {"name": "Dan", "phone": [300, 150]} AS x;

-- output
-- trying to query an object
/*
Type mismatch: function scan-collection expects its
1st input parameter to be type multiset or array,
but the actual input type is object
[TypeMismatchException]
*/
```

Recap: Simple SQL++

```
SELECT x.phone
  FROM [
         {"name": "Dan", "phone": [300, 150]},
         {"name": "Alvin", "phone": null}
       ] AS x;
```

Recap: Simple SQL++

```
SELECT x.phone
  FROM [
         {"name": "Dan", "phone": [300, 150]},
         {"name": "Alvin", "phone": null}
       ] AS x;

-- output, null works like in SQL
-- null values
/*
{ "phone": [300, 150] }
{ "phone": null }
*/
```

Recap: Simple SQL++

```
SELECT x.phone
  FROM [
         {"name": "Dan", "phone": [300, 150]},
         {"name": "Alvin"}
       ] AS x;
```

Recap: Simple SQL++

```
SELECT x.phone
  FROM [
         {"name": "Dan", "phone": [300, 150]},
         {"name": "Alvin"}
       ] AS x;

-- output, missing data is simply passed over (beware of typos!)
-- missing values
/*
{ "phone": [300, 150] }
{ }
*/
```

Recap: Simple SQL++

```
SELECT x.fone -- intentional typo
  FROM [
         {"name": "Dan", "phone": [300, 150]},
         {"name": "Alvin", "phone": 420}
       ] AS x;
```

Recap: Simple SQL++

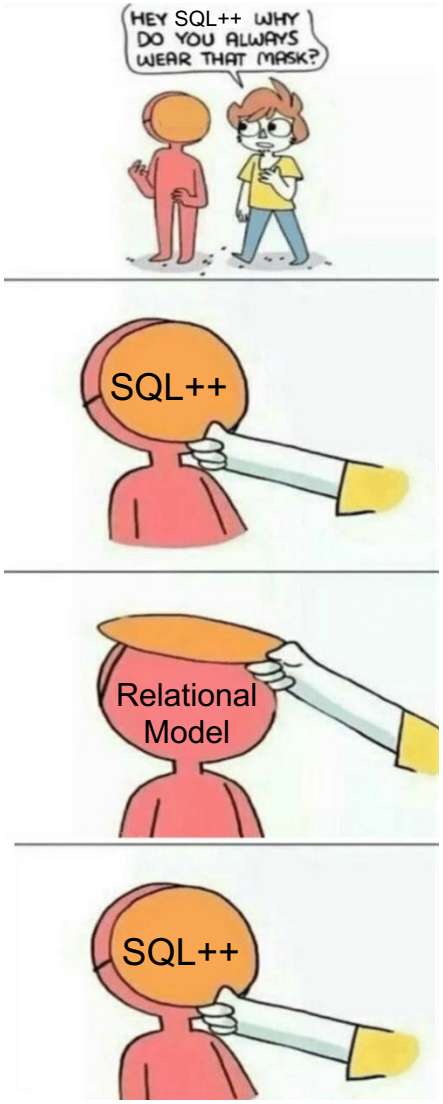
```
SELECT x.fone -- intentional typo
  FROM [
         {"name": "Dan", "phone": [300, 150]},
         {"name": "Alvin", "phone": 420}
       ] AS x;

-- output, beware of typos! No errors are thrown
/*
{ }
{ }
*/
```

Advanced SQL++

SQL++ is like SQL, but plus-plus

- Patterns in querying semi-structured data
- SQL++ behind the mask
- We'll discuss next, but you still need to look up the documentation



DDL and DML

	SQL Examples	SQL++ Examples
Data Description Language (DDL)	CREATE DATABASE… CREATE TABLE… CREATE INDEX… DROP TABLE… ALTER TABLE… (unique)	CREATE DATAVERSE… CREATE TYPE… (unique) CREATE DATASET… CREATE INDEX… DROP DATASET…
Data Manipulation Language (DML)	SELECT…FROM… INSERT INTO… DELETE FROM…	SELECT…FROM… INSERT INTO… DELETE FROM…

DDL and DML

	SQL Examples	SQL++ Examples
Data Description Language (DDL)	C C C D A	C
Data Manipulation Language (DML)	SELECT…FROM… INSERT INTO… DELETE FROM…	SELECT…FROM… INSERT INTO… DELETE FROM…

Schema Manipulation

DDL and DML

	SQL Examples	SQL++ Examples
Data Description Language (DDL)	C C C D A	C
Data Manipulation Language (DML)	S I D	S

Schema Manipulation

Data Manipulation

DDL and DML

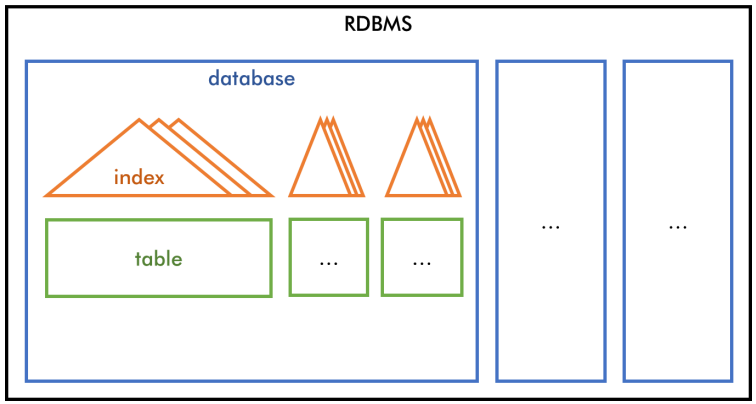
- **Data Definition Language (DDL)**
  - Defining structure beyond self-description
  - Indexing
- Data Manipulation Language (DML)
  - Joins
  - Nesting and Unnesting
- Please see the manual too. It is well written.

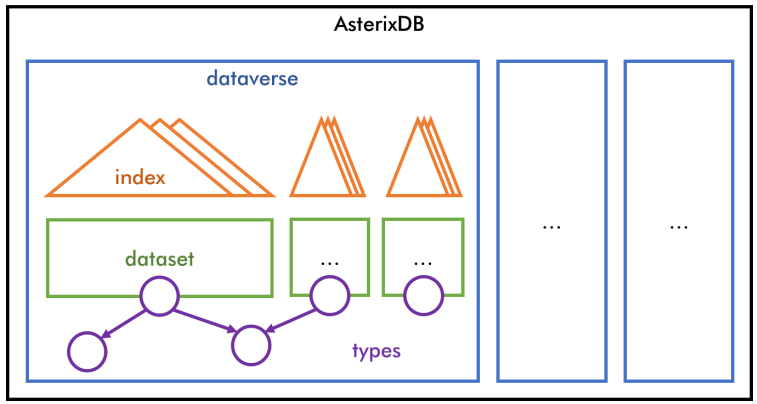
DDL: Key Concepts

- Dataverse (“data universe”): the database
- Type: the type of record
- Dataset: a table
  - It must have a type
  - It must have a primary key (some attribute in that type)
  - Often the key is of type UUID and autogenerated

Data Definition Language (DDL)

- Extremely similar to the relational world





Functionality	RDBMS	AsterixDB
Namespace	Database	Dataverse
Data Collection	Table	Dataset
Data Access	Index	Index

Steps to Load Data

We have an ADM data file: lec27_sqlpp.adm

1. Create a dataverse myDB
2. Create the type used in the ADM file
3. Create the dataset
4. Load the data
5. Query and enjoy…

ADM Data

lec27_sqlpp.adm

```
[
  { "name": "Dan",
    "phone": "555-123-4567",
    "orders":
        [
           { "date": 1997,
             "product": "Furby"
           }
         ]
  },
  { "name": "Alvin",
    "phone": "555-234-5678",
    "orders":
        [
           { "date": 2000,
             "product": "Furby"
           },
           { "date": 2012,
             "product": "Magic8"
           }
         ]
  },
  { "name": "Magda",
    "phone": "555-345-6789"
  }
]
```

1. Dataverse

lec27_sqlpp.adm

```
[
  { "name": "Dan",
    "phone": "555-123-4567",
    "orders":
        [
           { "date": 1997,
             "product": "Furby"
           }
         ]
  },
  { "name": "Alvin",
    "phone": "555-234-5678",
    "orders":
        [
           { "date": 2000,
             "product": "Furby"
           },
           { "date": 2012,
             "product": "Magic8"
           }
         ]
  },
  { "name": "Magda",
    "phone": "555-345-6789"
  }
]
```

```
DROP DATAVERSE myDB IF EXISTS;

CREATE DATAVERSE myDB;
```

2. Types

lec27_sqlpp.adm

```
[
  { "name": "Dan",
    "phone": "555-123-4567",
    "orders":
        [
           { "date": 1997,
             "product": "Furby"
           }
         ]
  },
  { "name": "Alvin",
    "phone": "555-234-5678",
    "orders":
        [
           { "date": 2000,
             "product": "Furby"
           },
           { "date": 2012,
             "product": "Magic8"
           }
         ]
  },
  { "name": "Magda",
    "phone": "555-345-6789"
  }
]
```

```
USE myDB;
DROP TYPE OrderType IF EXISTS;
CREATE TYPE OrderType AS CLOSED {
      date: int,
      product: string
};
```

2. Types

lec27_sqlpp.adm

```
[
  { "name": "Dan",
    "phone": "555-123-4567",
    "orders":
        [
           { "date": 1997,
             "product": "Furby"
           }
         ]
  },
  { "name": "Alvin",
    "phone": "555-234-5678",
    "orders":
        [
           { "date": 2000,
             "product": "Furby"
           },
           { "date": 2012,
             "product": "Magic8"
           }
         ]
  },
  { "name": "Magda",
    "phone": "555-345-6789"
  }
]
```

Specify the dataverse

```
USE myDB;
```

2. Types

lec27_sqlpp.adm

```
[
  { "name": "Dan",
    "phone": "555-123-4567",
    "orders":
        [
          { "date": 1997,
            "product": "Furby"
          }
        ]
  },
  { "name": "Alvin",
    "phone": "555-234-5678",
    "orders":
        [
          { "date": 2000,
            "product": "Furby"
          },
          { "date": 2012,
            "product": "Magic8"
          }
        ]
  },
  { "name": "Magda",
    "phone": "555-345-6789"
  }
]
```

These are Order objects

```
USE myDB;
```

2. Types

lec27_sqlpp.adm

```
[
  { "name": "Dan",
    "phone": "555-123-4567",
    "orders":
        [
          { "date": 1997,
            "product": "Furby"
          }
        ]
  },
  { "name": "Alvin",
    "phone": "555-234-5678",
    "orders":
        [
          { "date": 2000,
            "product": "Furby"
          },
          { "date": 2012,
            "product": "Magic8"
          }
        ]
  },
  { "name": "Magda",
    "phone": "555-345-6789"
  }
]
```

These are Order objects

Their type

```
USE myDB;
DROP TYPE OrderType IF EXISTS;
CREATE TYPE OrderType AS CLOSED {
      date: int,
      product: string
};
```

2. Types

lec27_sqlpp.adm

```
[
  { "name": "Dan",
    "phone": "555-123-4567",
    "orders":
        [
           { "date": 1997,
             "product": "Furby"
           }
         ]
  },
  { "name": "Alvin",
    "phone": "555-234-5678",
    "orders":
        [
           { "date": 2000,
             "product": "Furby"
           },
           { "date": 2012,
             "product": "Magic8"
           }
         ]
  },
  { "name": "Magda",
    "phone": "555-345-6789"
  }
]
```

CLOSED means data cannot have extra attrbitutes

```
USE myDB;
DROP TYPE OrderType IF EXIS S;
CREATE TYPE OrderType AS CLOSED {
      date: int,
      product: string
};
```

2. Types

lec27_sqlpp.adm

```
[
  { "name": "Dan",
    "phone": "555-123-4567",
    "orders":
        [
           { "date": 1997,
             "product": "Furby"
           }
         ]
  },
  { "name": "Alvin",
    "phone": "555-234-5678",
    "orders":
        [
           { "date": 2000,
             "product": "Furby"
           },
           { "date": 2012,
             "product": "Magic8”,
             ”weight”: “15lb”
           }
         ]
  },
  { "name": "Magda",
    "phone": "555-345-6789"
  }
]
```

Now it can have

```
USE myDB;
DROP TYPE OrderType IF EXISTS;
CREATE TYPE OrderType AS OPEN {
      date: int,
      product: string
};
```

2. Types

lec27_sqlpp.adm

This is a Person object

```
[
  { "name": "Dan",
    "phone": "555-123-4567",
    "orders":
        [
          { "date": 1997,
            "product": "Furby"
          }
        ]
  },
  { "name": "Alvin",
    "phone": "555-234-5678",
    "orders":
        [
          { "date": 2000,
            "product": "Furby"
          },
          { "date": 2012,
            "product": "Magic8"
          }
        ]
  },
  { "name": "Magda",
    "phone": "555-345-6789"
  }
]
```

```
USE myDB;
DROP TYPE OrderType IF EXISTS;
CREATE TYPE OrderType AS CLOSED {
      date: int,
      product: string
};
```

2. Types

lec27_sqlpp.adm

This is a Person object

```
[
  { "name": "Dan",
    "phone": "555-123-4567",
    "orders":
        [
           { "date": 1997,
             "product": "Furby"
           }
        ]
  },
  { "name": "Alvin",
    "phone": "555-234-5678",
    "orders":
        [
           { "date": 2000,
             "product": "Furby"
           },
           { "date": 2012,
             "product": "Magic8"
           }
        ]
  },
  { "name": "Magda",
    "phone": "555-345-6789"
  }
]
```

This is its type

```
USE myDB;
DROP TYPE OrderType IF EXISTS;
CREATE TYPE OrderType AS CLOSED {
     date: int,
     product: string
};

DROP TYPE PersonType IF EXISTS;
CREATE TYPE PersonType AS CLOSED {
    pid: uuid,
    name:  string,
    phone: string,
    orders: [OrderType]?
};
```

2. Types

lec27_sqlpp.adm

```
[
  { "name": "Dan",
    "phone": "555-123-4567",
    "orders":
        [
           { "date": 1997,
             "product": "Furby"
           }
        ]
  },
  { "name": "Alvin",
    "phone": "555-234-5678",
    "orders":
        [
           { "date": 2000,
             "product": "Furby"
           },
           { "date": 2012,
             "product": "Magic8"
           }
        ]
  },
  { "name": "Magda",
    "phone": "555-345-6789"
  }
]
```

```
USE myDB;
DROP TYPE OrderType IF EXISTS;
CREATE TYPE OrderType AS CLOSED {
     date: int,
     product: string
};

DROP TYPE PersonType IF EXISTS;
CREATE TYPE PersonType AS CLOSED {
    pid: uuid,
    name:  string,
    phone: string,
    orders: [OrderType]?
};
```

Autogenerated

2. Types

lec27_sqlpp.adm

```
[
  { "name": "Dan",
    "phone": "555-123-4567",
    "orders":
        [
           { "date": 1997,
             "product": "Furby"
           }
         ]
  },
  { "name": "Alvin",
    "phone": "555-234-5678",
    "orders":
        [
           { "date": 2000,
             "product": "Furby"
           },
           { "date": 2012,
             "product": "Magic8"
           }
         ]
  },
  { "name": "Magda",
    "phone": "555-345-6789"
  }
]
```

```
USE myDB;
DROP TYPE OrderType IF EXISTS;
CREATE TYPE OrderType AS CLOSED {
     date: int,
     product: string
};

DROP TYPE PersonType IF EXISTS;
CREATE TYPE PersonType AS CLOSED {
    pid: uuid,
    name:  string,
    phone: string,
    orders: [OrderType]?
};
```

Defined above

2. Types

lec27_sqlpp.adm

```
[
  { "name": "Dan",
    "phone": "555-123-4567",
    "orders":
        [
           { "date": 1997,
             "product": "Furby"
           }
        ]
  },
  { "name": "Alvin",
    "phone": "555-234-5678",
    "orders":
        [
           { "date": 2000,
             "product": "Furby"
           },
           { "date": 2012,
             "product": "Magic8"
           }
        ]
  },
  { "name": "Magda",
    "phone": "555-345-6789"
  }
]
```

```
USE myDB;
DROP TYPE OrderType IF EXISTS;
CREATE TYPE OrderType AS CLOSED {
     date: int,
     product: string
};

DROP TYPE PersonType IF EXISTS;
CREATE TYPE PersonType AS CLOSED {
    pid: uuid,
    name:  string,
    phone: string,
    orders: [OrderType]?
};
```

An array of orders

2. Types

lec27_sqlpp.adm

```
[
  { "name": "Dan",
    "phone": "555-123-4567",
    "orders":
        [
           { "date": 1997,
             "product": "Furby"
           }
        ]
  },
  { "name": "Alvin",
    "phone": "555-234-5678",
    "orders":
        [
           { "date": 2000,
             "product": "Furby"
           },
           { "date": 2012,
             "product": "Magic8"
           }
        ]
  },
  { "name": "Magda",
    "phone": "555-345-6789"
  }
]
```

No orders

```
USE myDB;
DROP TYPE OrderType IF EXISTS;
CREATE TYPE OrderType AS CLOSED {
     date: int,
     product: string
};

DROP TYPE PersonType IF EXISTS;
CREATE TYPE PersonType AS CLOSED {
    pid: uuid,
    name:  string,
    phone: string,
    orders: [OrderType]?
};
```

…and it may be missing

3. Dataset

lec27_sqlpp.adm

```
[
  { "name": "Dan",
    "phone": "555-123-4567",
    "orders":
        [
           { "date": 1997,
             "product": "Furby"
           }
         ]
  },
  { "name": "Alvin",
    "phone": "555-234-5678",
    "orders":
        [
           { "date": 2000,
             "product": "Furby"
           },
           { "date": 2012,
             "product": "Magic8"
           }
         ]
  },
  { "name": "Magda",
    "phone": "555-345-6789"
  }
]
```

```
CREATE TYPE PersonType AS CLOSED {
    pid: uuid,
    name:  string,
    phone: string,
    orders: [OrderType]
```

Like CREATE TABLE Person

```
USE myDB;
CREATE DATASET Person(PersonType)
PRIMARY KEY pid AUTOGENERATED;
```

3. Dataset

lec27_sqlpp.adm

```
[
  { "name": "Dan",
    "phone": "555-123-4567",
    "orders":
        [
           { "date": 1997,
             "product": "Furby"
           }
         ]
  },
  { "name": "Alvin",
    "phone": "555-234-5678",
    "orders":
        [
           { "date": 2000,
             "product": "Furby"
           },
           { "date": 2012,
             "product": "Magic8"
           }
         ]
  },
  { "name": "Magda",
    "phone": "555-345-6789"
  }
]
```

```
CREATE TYPE PersonType AS CLOSED {
    pid: uuid,
    name:  string,
    phone: string,
    orders: [OrderType]
```

The type of Person

```
USE myDB;
CREATE DATASET Person(PersonType)
PRIMARY KEY pid AUTOGENERATED;
```

3. Dataset

lec27_sqlpp.adm

```
[
  { "name": "Dan",
    "phone": "555-123-4567",
    "orders":
        [
           { "date": 1997,
             "product": "Furby"
           }
         ]
  },
  { "name": "Alvin",
    "phone": "555-234-5678",
    "orders":
        [
           { "date": 2000,
             "product": "Furby"
           },
           { "date": 2012,
             "product": "Magic8"
           }
         ]
  },
  { "name": "Magda",
    "phone": "555-345-6789"
  }
]
```

```
CREATE TYPE PersonType AS CLOSED {
    pid: uuid,
    name:  string,
    phone: string,
    orders: [OrderType]
```

```
USE myDB;
CREATE DATASET Person(PersonType)
PRIMARY KEY pid AUTOGENERATED;
```

The key is pid.
Autogenerated.

Discussion

- Types define the schema of some collection; not necessarily a top-level one
- Specify CLOSED/OPEN
  - CLOSED → no other fields besides the listed ones
  - OPEN → extra fields are allowed
- List all fields
- List optional fields with "?“ (may be missing)

Closed/Open Types

- No additional fields allowed

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

```
[
   {
      "name": "Dan",
      "phone": 5551234567,
      "email": "suciu@cs"
   },
   {
      "name": "Alvin",
      "phone": 5552345678
   }
]
```

Closed/Open Types

- No additional fields allowed

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

```
[
   {
      "name": "Dan",
      "phone": 5551234567,
      "email": "suciu@cs"
   },
   {
      "name": "Alvin",
      "phone": 5552345678,
      "likesBananas": true
   }
]
```

Can't use unspecified fields

Closed/Open Types

- Allows additional fields

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

```
[
   {
      "name": "Dan",
      "phone": 5551234567,
      "email": "suciu@cs"
   },
   {
      "name": "Alvin",
      "phone": 5552345678,
      "likesBananas": true
   }
]
```

Nested Collections

- Data can be a collection

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

```
[
   {
      "name": "Dan",
      "phone": [5551234567]
   },
   {
      "name": "Alvin",
      "phone": [5552345678, 5553456789]
   },
   {
      "name": "Magda",
      "phone": []
   }
]
```

Non-Uniform data

- Types can be undefined and thus non-uniform(!)
- “phone” can be a number, an array, or missing

```
[
   {
      "name": "Dan",
      "phone": 5551234567
   },
   {
      "name": "Alvin",
      "phone": [5552345678, 5553456789]
   },
   {
      "name": "Magda"
   }
]
```

Keys

Each dataset must have a primary key

- For lookup ability
- Secondary indexing
- Sharding/Partitioning

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

DROP DATASET Person IF EXISTS;
CREATE DATASET Person(PersonType)
  PRIMARY KEY name;
```

Keys

Each dataset must have a primary key

- For lookup ability
- Secondary indexing
- Sharding/Partitioning

What if there are no good keys?

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

DROP DATASET Person IF EXISTS;
CREATE DATASET Person(PersonType)
  PRIMARY KEY name;
```

Keys

Each dataset must have a primary key

- For lookup ability
- Secondary indexing
- Sharding/Partitioning

What if there are no good keys?

Autogenerate!

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

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

Keys

Each dataset must have a primary key

- For lookup ability
- Secondary indexing
- Sharding/Partitioning

What if there are no good keys?

Autogenerate!

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

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

Each object will have a uuid field named "myKey"

SQL++

- Data Definition Language (DDL)
  - Defining structure beyond self-description
  - Indexing
- **Data Manipulation Language (DML)**
  - Joins
  - Nesting and Unnesting
- Please see the manual too. It is well written.

Nested-loop Semantics

- Same as SQL!

```
SELECT p.name, p.phone, o.date, o.product
  FROM Person AS p, Orders AS o
 WHERE p.name = o.pname;
```

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567"
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678"
   },
   {
      "name": "Magda",
      "phone": "555-345-6789"
   }
}}
```

```
-- Dataset Orders
{{
   {
      "pname": "Dan"
      "date": 1997,
      "product": "Furby"
   },
   {
      "pname": "Alvin"
      "date": 2000,
      "product": "Furby"
   },
   {
      "pname": "Alvin"
      "date": 2012,
      "product": "Magic8"
   }
}}
```

Nested-loop Semantics

- Same as SQL!

```
SELECT p.name, p.phone, o.date, o.product
  FROM Person AS p, Orders AS o
 WHERE p.name = o.pname;
```

```
for each object in p:
    for each object in o:
        if WHERE satisfied:
            …
```

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567"
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678"
   },
   {
      "name": "Magda",
      "phone": "555-345-6789"
   }
}}
```

```
-- Dataset Orders
{{
   {
      "pname": "Dan"
      "date": 1997,
      "product": "Furby"
   },
   {
      "pname": "Alvin"
      "date": 2000,
      "product": "Furby"
   },
   {
      "pname": "Alvin"
      "date": 2012,
      "product": "Magic8"
   }
}}
```

Nested-loop Semantics

- Same as SQL!

```
SELECT p.name, p.phone, o.date, o.product
  FROM Person AS p, Orders AS o
 WHERE p.name = o.pname;

-- Output
/*
{name: Dan,   phone: 555-123-4567, date: 1997, product: Furby}
{name: Alvin, phone: 555-234-5678, date: 2000, product: Furby}
{name: Alvin, phone: 555-234-5678, date: 2012, product: Magic8}
*/
```

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567"
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678"
   },
   {
      "name": "Magda",
      "phone": "555-345-6789"
   }
}}
```

```
-- Dataset Orders
{{
   {
      "pname": "Dan"
      "date": 1997,
      "product": "Furby"
   },
   {
      "pname": "Alvin"
      "date": 2000,
      "product": "Furby"
   },
   {
      "pname": "Alvin"
      "date": 2012,
      "product": "Magic8"
   }
}}
```

Nested Data

- Nested data → Unnested results
- Unnested data → Nested results

Nested Data → Unnested Results

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567",
      "orders": [
         {
            "date": 1997,
            "product": "Furby"
         }
      ]
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678",
      "orders": [
         {
            "date": 2000,
            "product": "Furby"
         },
         {
            "date": 2012,
            "product": "Magic8"
         }
      ]
   },
   {
      "name": "Magda",
      "phone": "555-345-6789",
      "orders": []
   }
}}
```

Nested Data → Unnested Results

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567",
      "orders": [
         {
            "date": 1997,
            "product": "Furby"
         }
      ]
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678",
      "orders": [
         {
            "date": 2000,
            "product": "Furby"
         },
         {
            "date": 2012,
            "product": "Magic8"
         }
      ]
   },
   {
      "name": "Magda",
      "phone": "555-345-6789",
      "orders": []
   }
}}
```

```
SELECT p.name, p.phone,
       p.orders.date, p.orders.product
  FROM Person AS p;

-- ERROR
```

Nested Data → Unnested Results

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567",
      "orders": [
         {
            "date": 1997,
            "product": "Furby"
         }
      ]
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678",
      "orders": [
         {
            "date": 2000,
            "product": "Furby"
         },
         {
            "date": 2012,
            "product": "Magic8"
         }
      ]
   },
   {
      "name": "Magda",
      "phone": "555-345-6789",
      "orders": []
   }
}}
```

```
SELECT p.name, p.phone,
       p.orders.date, p.orders.product
  FROM Person AS p;

-- ERROR
```

Dereferencing can only be done on objects!

Nested Data → Unnested Results

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567",
      "orders": [
         {
            "date": 1997,
            "product": "Furby"
         }
      ]
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678",
      "orders": [
         {
            "date": 2000,
            "product": "Furby"
         },
         {
            "date": 2012,
            "product": "Magic8"
         }
      ]
   },
   {
      "name": "Magda",
      "phone": "555-345-6789",
      "orders": []
   }
}}
```

```
SELECT p.name, p.phone, o.date, o.product
  FROM Person AS p, p.orders AS o;

-- output
/*
{name: Dan,   phone: 555-123-4567, date: 1997, product: Furby}
{name: Alvin, phone: 555-234-5678, date: 2000, product: Furby}
{name: Alvin, phone: 555-234-5678, date: 2012, product: Magic8}
*/
```

Parent-child join!

Nested Data → Unnested Results

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567",
      "orders": [
         {
            "date": 1997,
            "product": "Furby"
         }
      ]
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678",
      "orders": [
         {
            "date": 2000,
            "product": "Furby"
         },
         {
            "date": 2012,
            "product": "Magic8"
         }
      ]
   },
   {
      "name": "Magda",
      "phone": "555-345-6789",
      "orders": []
   }
}}
```

```
SELECT p.name, p.phone, o.date, o.product
  FROM Person AS p UNNEST p.orders AS o;

-- output
/*
{name: Dan,   phone: 555-123-4567, date: 1997, product: Furby}
{name: Alvin, phone: 555-234-5678, date: 2000, product: Furby}
{name: Alvin, phone: 555-234-5678, date: 2012, product: Magic8}
*/
```

Same as ","

Nested Data → Unnested Results

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567",
      "orders": [
         {
            "date": 1997,
            "product": "Furby"
         }
      ]
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678",
      "orders": [
         {
            "date": 2000,
            "product": "Furby"
         },
         {
            "date": 2012,
            "product": "Magic8"
         }
      ]
   },
   {
      "name": "Magda",
      "phone": "555-345-6789",
      "orders": []
   }
}}
```

```
SELECT p.name, p.phone, o.date, o.product
  FROM Person AS p, p.orders AS o;

-- output
/*
{name: Dan,   phone: 555-123-4567, date: 1997, product: Furby}
{name: Alvin, phone: 555-234-5678, date: 2000, product: Furby}
{name: Alvin, phone: 555-234-5678, date: 2012, product: Magic8}
*/
```

Non-Uniform Data

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567",
      "orders": {
         "date": 1997,
         "product": "Furby"
      }
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678"
      "orders": [
         {
            "date": 2000,
            "product": "Furby"
         },
         {
            "date": 2012,
            "product": "Magic8"
         }
      ]
   },
   {
      "name": "Magda",
      "phone": "555-345-6789",
      "orders": []
   }
}}
```

- What if data is not uniform?

Non-Uniform Data

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567",
      "orders": {
         "date": 1997,
         "product": "Furby"
      }
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678",
      "orders": [
         {
            "date": 2000,
            "product": "Furby"
         },
         {
            "date": 2012,
            "product": "Magic8"
         }
      ]
   },
   {
      "name": "Magda",
      "phone": "555-345-6789",
      "orders": []
   }
}}
```

- What if data is not uniform?

Non-Uniform Data

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567",
      "orders": {
         "date": 1997,
         "product": "Furby"
      }
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678",
      "orders": [
         {
            "date": 2000,
            "product": "Furby"
         },
         {
            "date": 2012,
            "product": "Magic8"
         }
      ]
   },
   {
      "name": "Magda",
      "phone": "555-345-6789",
      "orders": []
   }
}}
```

- What if data is not uniform?

```
SELECT p.name, p.phone, o.date, o.product
  FROM Person AS p, p.orders AS o;
```

Why is this now invalid?

Non-Uniform Data

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567",
      "orders": {
         "date": 1997,
         "product": "Furby"
      }
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678",
      "orders": [
         {
            "date": 2000,
            "product": "Furby"
         },
         {
            "date": 2012,
            "product": "Magic8"
         }
      ]
   },
   {
      "name": "Magda",
      "phone": "555-345-6789",
      "orders": []
   }
}}
```

- What if data is not uniform?

```
SELECT p.name, p.phone, o.date, o.product
  FROM Person AS p, p.orders AS o;
```

Why is this now invalid?

Can't query an object, only array/multiset!

Non-Uniform Data

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567",
      "orders": {
         "date": 1997,
         "product": "Furby"
      }
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678",
      "orders": [
         {
            "date": 2000,
            "product": "Furby"
         },
         {
            "date": 2012,
            "product": "Magic8"
         }
      ]
   },
   {
      "name": "Magda",
      "phone": "555-345-6789",
      "orders": []
   }
}}
```

- What if data is not uniform?
  - Use built-in functions/keywords

```
SELECT p.name, p.phone, o.date, o.product
  FROM Person AS p,
       (CASE WHEN p.orders IS MISSING
                     THEN []
             WHEN IS_ARRAY(p.orders)
                     THEN p.orders
             ELSE [p.orders]
       ) AS o;
```

Non-Uniform Data

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567",
      "orders": {
         "date": 1997,
         "product": "Furby"
      }
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678",
      "orders": [
         {
            "date": 2000,
            "product": "Furby"
         },
         {
            "date": 2012,
            "product": "Magic8"
         }
      ]
   },
   {
      "name": "Magda",
      "phone": "555-345-6789",
      "orders": []
   }
}}
```

- What if data is not uniform?
  - Use built-in functions/keywords

```
FROM Person AS p
LET ords =
        (CASE WHEN p.orders IS MISSING
                        THEN []
              WHEN IS_ARRAY(p.orders)
                        THEN p.orders
              ELSE [p.orders]
        )
FROM ords AS o;
SELECT p.name, p.phone, o.date, o.product
```

Non-Uniform Data

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567",
      "orders": {
         "date": 1997,
         "product": "Furby"
      }
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678",
      "orders": [
         {
            "date": 2000,
            "product": "Furby"
         },
         {
            "date": 2012,
            "product": "Magic8"
         }
      ]
   },
   {
      "name": "Magda",
      "phone": "555-345-6789",
      "orders": []
   }
}}
```

- What if data is not uniform?
  - Use built-in functions/keywords

Iterate over each person p…

```
FROM Person AS p
LET ords =
        (CASE WHEN p.orders IS MISSING
                        THEN []
              WHEN IS_ARRAY(p.orders)
                        THEN p.orders
              ELSE [p.orders]
        )
FROM ords AS o;
SELECT p.name, p.phone, o.date, o.product
```

Non-Uniform Data

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567",
      "orders": {
         "date": 1997,
         "product": "Furby"
      }
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678",
      "orders": [
         {
            "date": 2000,
            "product": "Furby"
         },
         {
            "date": 2012,
            "product": "Magic8"
         }
      ]
   },
   {
      "name": "Magda",
      "phone": "555-345-6789",
      "orders": []
   }
}}
```

- What if data is not uniform?
- ...ilt-in functions/keywords

This is done for each p!

Compute ords which is always an array

```
FROM Person AS p
LET ords =
        (CASE WHEN p.orders IS MISSING
                    THEN []
              WHEN IS_ARRAY(p.orders)
                    THEN p.orders
              ELSE [p.orders]
        )
FROM ords AS o;
SELECT p.name, p.phone, o.date, o.product
```

Non-Uniform Data

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567",
      "orders": {
         "date": 1997,
         "product": "Furby"
      }
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678",
      "orders": [
         {
            "date": 2000,
            "product": "Furby"
         },
         {
            "date": 2012,
            "product": "Magic8"
         }
      ]
   },
   {
      "name": "Magda",
      "phone": "555-345-6789",
      "orders": []
   }
}}
```

- What if data is not uniform?
  - Use built-in functions/keywords

```
FROM Person AS p
LET ords =
        (CASE WHEN p.orders IS MISSING
                        THEN []
              WHEN IS_ARRAY(p.orders)
                        THEN p.orders
              ELSE [p.orders]
        )
FROM ords AS o;
SELECT p.name, p.phone, o.date, o.product
```

Now iterate over ords

Non-Uniform Data

```
-- Dataset Person
{{
   {
      "name": "Dan",
      "phone": "555-123-4567",
      "orders": {
         "date": 1997,
         "product": "Furby"
      }
   },
   {
      "name": "Alvin",
      "phone": "555-234-5678",
      "orders": [
         {
            "date": 2000,
            "product": "Furby"
         },
         {
            "date": 2012,
            "product": "Magic8"
         }
      ]
   },
   {
      "name": "Magda",
      "phone": "555-345-6789",
      "orders": []
   }
}}
```

- What if data is not uniform?
  - Use built-in functions/keywords

```
FROM Person AS p
LET ords =
        (CASE WHEN p.orders IS MISSING
                        THEN []
              WHEN IS_ARRAY(p.orders)
                        THEN p.orders
              ELSE [p.orders]
        )
FROM ords AS o;
SELECT p.name, p.phone, o.date, o.product
```

Finally, return what we need

Useful functions

- IS_ARRAY(…)
- IS_OBJECT(…)
- IS_BOOLEAN(…)
- IS_STRING(…)
- IS_NUMBER(…)
- IS_NULL(…)
- IS_MISSING(…)
- IS_UNKNOWN(…)

Unnested Data → Nested Results

Different query!

Return an object for each product and a list of people who bought that product.

```
-- Dataset Orders
{{
   {
      "pname": "Dan"
      "date": 1997,
      "product": "Furby"
   },
   {
      "pname": "Alvin"
      "date": 2000,
      "product": "Furby"
   },
   {
      "pname": "Alvin"
      "date": 2012,
      "product": "Magic8"
   }
}}
```

Unnested Data → Nested Results

Different query!

Return an object for each product and a list of people who bought that product.

```
SELECT DISTINCT o.product,
       (SELECT u.pname
          FROM Orders AS u
         WHERE o.product = u.product) AS names
  FROM Orders AS o;
```

```
-- Dataset Orders
{{
   {
      "pname": "Dan"
      "date": 1997,
      "product": "Furby"
   },
   {
      "pname": "Alvin"
      "date": 2000,
      "product": "Furby"
   },
   {
      "pname": "Alvin"
      "date": 2012,
      "product": "Magic8"
   }
}}
```

Unnested Data → Nested Results

Different query!

Return an object for each product and a list of people who bought that product.

```
FROM Orders AS o
LET n = (SELECT u.pname
         FROM Orders AS u
         WHERE o.product = u.product)
SELECT DISTINCT o.product, n AS names;
```

Cleaner

```
-- Dataset Orders
{{
   {
      "pname": "Dan"
      "date": 1997,
      "product": "Furby"
   },
   {
      "pname": "Alvin"
      "date": 2000,
      "product": "Furby"
   },
   {
      "pname": "Alvin"
      "date": 2012,
      "product": "Magic8"
   }
}}
```

Unnested Data → Nested Results

Different query!

Return an object for each product and a list of people who bought that product.

```
-- Dataset Orders
{{
   {
      "pname": "Dan"
      "date": 1997,
      "product": "Furby"
   },
   {
      "pname": "Alvin"
      "date": 2000,
      "product": "Furby"
   },
   {
      "pname": "Alvin"
      "date": 2012,
      "product": "Magic8"
   }
}}
```

```
FROM Orders AS o
LET n = (SELECT u.pname
         FROM Orders AS u
         WHERE o.product = u.product)
SELECT DISTINCT o.product, n AS names;

-- Output
/*
{product: Furby, names:[{pname: Dan}, {pname: Alvin}]}
{product: Magic8, names:[{pname: Alvin}]}
*/
```

Takeaways for SQL++

Much like SQL, but…

- Any order of SELECT, FROM, WHERE
- Also LET (same as WITH)
- Can query nested collections w/ nested iterators
- Non-uniform data is a pain to deal with