Introduction to Data Management
CSE 414

Lecture 14: SQL++
Announcements

• Midterm in class on Wednesday, 10/31
  – Covers everything (HW, WQ, lectures, sections, readings) up to beginning of NoSQL lectures.
  – One double-sided sheet of notes allowed
  – Remember to practice your SQL queries

• WQ5 and HW4 due Tuesday 10/30
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": []}
In General

Needs to be an array or multiset (i.e., iterable)

```
SELECT ...
FROM R AS x, S AS y
WHERE x.f1 = y.f2;
```

These cannot evaluate to an array or dataset!

Need to “unnest” the array
Return province and city names

The problem:

```
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";
```

Error: Type mismatch!

- city is an array
- city is an object

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

Error: Type mismatch!
```
Return province and city names

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"...}
   ...
  }, ...
}
The problem:

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

```sql
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);
```
Return province and city names

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!
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 z.city IS missing THEN []
     WHEN IS_ARRAY(z.city) THEN z.city
     ELSE [z.city] END) AS u
WHERE y.name="Greece";
Useful Paradigms

- Unnesting
- Nesting
- Grouping and aggregate
- Joins
- Splitting
We want:

```
[{A:a1, Grp:[{b1, b2}]},
 {A:a2, Grp:[{b1}]}]
```

Using LET syntax:

```
SELECT DISTINCT x.A, g AS Grp
FROM C AS x
LET g = (SELECT y.B FROM C AS y WHERE x.A = y.A)
```
Unnesting Specific Field

A nested collection

```python
coll = 
[{A:a1, F:[{B:b1},{B:b2}], G:[{C:c1}]},
 {A:a2, F:[{B:b3},{B:b4},{B:b5}], G:[]},
 {A:a3, F:[{B:b6}], G:[{C:c2},{C:c3}]}]
```
Unnesting Specific Field

A nested collection

coll =
{A:a1, F:[{B:b1},{B:b2}], G:[{C:c1}]},
{A:a2, F:[{B:b3},{B:b4},{B:b5}], G:[]},
{A:a3, F:[{B:b6}], G:[{C:c2},{C:c3}]}

Unnest_F(coll) =
{{A:a1, B:b1, G:[{C:c1}]},
 {A:a1, B:b2, G:[{C:c1}]},
 {A:a2, B:b3, G:[]},
 {A:a2, B:b4, G:[]},
 {A:a2, B:b5, G:[]},
 {A:a3, B:b6, G:[{C:c2},{C:c3}]}]}
Unnesting Specific Field

A nested collection

coll =
{A:a1, F:[{B:b1},{B:b2}], G:[{C:c1}]},
{A:a2, F:[{B:b3},{B:b4},{B:b5}], G:[ ]},
{A:a3, F:[{B:b6}], G:[{C:c2},{C:c3}]}]

Unnest_F(coll) =
{A:a1, B:b1, G:[{C:c1}]},
{A:a1, B:b2, G:[{C:c1}]},
{A:a2, B:b3, G:[ ]},
{A:a2, B:b4, G:[ ]},
{A:a2, B:b5, G:[ ]},
{A:a3, B:b6, G:[{C:c2},{C:c3}]}]
Unnesting Specific Field

A nested collection

coll =

\[\begin{aligned}
\{&A:a1, F:\{B:b1\}, B:b2\}, G:\{C:c1\}], \\
\{&A:a2, F:\{B:b3\}, B:b4\}, B:b5\}, G:[]}, \\
\{&A:a3, F:\{B:b6\}, G:\{C:c2\}, C:c3\}]\]

Unnest\(_F(coll) =

\[\begin{aligned}
\{&A:a1, B:b1, G:\{C:c1\}], \\
\{&A:a1, B:b2, G:\{C:c1\}], \\
\{&A:a2, B:b3, G:[]}, \\
\{&A:a2, B:b4, G:[]}, \\
\{&A:a2, B:b5, G:[]}, \\
\{&A:a3, B:b6, G:\{C:c2\}, C:c3\}]\]

SELECT x.A, y.B, x.G
FROM coll x, x.F y

Nested Relational Algebra

Refers to relations defined on the left
Unnesting Specific Field

A nested collection

\[
\text{coll} = \\
\{\{A:a1, F:[\{B:b1\}, \{B:b2\}], G:[\{C:c1\}]\}, \\
\{A:a2, F:[\{B:b3\}, \{B:b4\}, \{B:b5\}], G:[\ ]\}, \\
\{A:a3, F:[\{B:b6\}], G:[\{C:c2\}, \{C:c3\}]\}\}
\]

Unnest_F(\text{coll}) =
\[
\{\{A:a1, B:b1, G:[\{C:c1\}]\}, \\
\{A:a1, B:b2, G:[\{C:c1\}]\}, \\
\{A:a2, B:b3, G:[\ ]\}, \\
\{A:a2, B:b4, G:[\ ]\}, \\
\{A:a2, B:b5, G:[\ ]\}, \\
\{A:a3, B:b6, G:[\{C:c2\}, \{C:c3\}]\}\}
\]

\[
\text{SELECT x.A, y.B, x.G} \\
\text{FROM coll x, y.F}
\]

\[
= \text{UNNEST coll x}
\]

SELECT x.A, y.B, x.G FROM coll x
UNNEST x.F y
Unnesting Specific Field

A nested collection

\[
\text{coll} = \\
\begin{cases}
\{A:a1, F:\{B:b1, B:b2\}, G:\{C:c1\}\}, \\
\{A:a2, F:\{B:b3, B:b4, B:b5\}, G:[]\}, \\
\{A:a3, F:\{B:b6\}, G:\{C:c2, C:c3\}\}\end{cases}
\]

Unnest\(_F\)(coll) =
\[
\begin{cases}
\{A:a1, B:b1, G:\{C:c1\}\}, \\
\{A:a1, B:b2, G:\{C:c1\}\}, \\
\{A:a2, B:b3, G:[]\}, \\
\{A:a2, B:b4, G:[]\}, \\
\{A:a2, B:b5, G:[]\}, \\
\{A:a3, B:b6, G:\{C:c2, C:c3\}\}\end{cases}
\]

Unnest\(_G\)(coll) =
\[
\begin{cases}
\{A:a1, F:\{B:b1, B:b2\}, C:c1\}, \\
\{A:a2, F:\{B:b3, B:b4, B:b5\}, C:[]\}, \\
\{A:a3, F:\{B:b6\}, C:c2, C:c3\}\end{cases}
\]

\[
\text{SELECT } x.A, y.B, x.G \\
\text{FROM } \text{coll} x, x.F y
\]
Unnesting Specific Field

A nested collection

coll =
[A:a1, F:[{B:b1},{B:b2}], G:[{C:c1}]],
{A:a2, F:[{B:b3},{B:b4},{B:b5}], G:[]},
{A:a3, F:[{B:b6}], G:[{C:c2},{C:c3}]]

Unnest_F(coll) =
[{A:a1, B:b1, G:[{C:c1}]},
{A:a1, B:b2, G:[{C:c1}]},
{A:a2, B:b3, G:[]},
{A:a2, B:b4, G:[]},
{A:a2, B:b5, G:[]},
{A:a3, B:b6, G:[{C:c2},{C:c3}]}]

Unnest_G(coll) =
[{A:a1, F:[{B:b1},{B:b2}], C:c1},
{A:a3, F:[{B:b6}], C:c2},
{A:a3, F:[{B:b6}], C:c3}]

SELECT x.A, y.B, x.G
FROM coll x, x.F y

SELECT x.A, x.F, z.C
FROM coll x, x.G z

Nested Relational Algebra

SQL++
Nesting (like group-by)

A flat collection

coll =
[{A:a1, B:b1}, {A:a1, B:b2}, {A:a2, B:b1}]
Nesting (like group-by)

A flat collection

\[
\text{coll} = \\
[{A:a1, B:b1}, {A:a1, B:b2}, {A:a2, B:b1}]
\]

\[
\text{Nest}_A(\text{coll}) = \\
[{A:a1, GRP:[{B:b1},{B:b2}]}, \\
[{A:a2, GRP:[{B:b2}]}]
\]
Nesting (like group-by)

A flat collection

\[
\text{coll} = \\
\{\{A:a1, B:b1\}, \{A:a1, B:b2\}, \{A:a2, B:b1\}\}
\]

\[
\text{Nest}_A(\text{coll}) = \\
\{\{A:a1, \text{GRP:}[[B:b1],[B:b2]]\} \\
\{A:a2, \text{GRP:}[[B:b2]]\}\}
\]
Nesting (like group-by)

A flat collection

\[
\text{coll} = \\
\{A:a1, B:b1\}, \{A:a1, B:b2\}, \{A:a2, B:b1\}
\]

\[
\text{Nest}_A(\text{coll}) = \\
\{A:a1, \text{GRP:}\{B:b1, B:b2\}\}, \\
\{A:a2, \text{GRP:}\{B:b2\}\}
\]

\[
\text{Nest}_B(\text{coll}) = \\
\{B:b1, \text{GRP:}\{A:a1, A:a2\}\}, \\
\{B:b2, \text{GRP:}\{A:a1\}\}
\]
Nesting (like group-by)

A flat collection

\[
\text{coll} = \\
\{\{A:a1, B:b1\}, \{A:a1, B:b2\}, \{A:a2, B:b1\}\}
\]

\[
\text{Nest}_A(\text{coll}) = \\
\{\{A:a1, \text{GRP:}\{\{B:b1\}, \{B:b2\}\}\} \\
\{\{A:a2, \text{GRP:}\{\{B:b2\}\}\}\}
\]

\[
\text{Nest}_B(\text{coll}) = \\
\{\{B:b1, \text{GRP:}\{\{A:a1\}, \{A:a2\}\}\}, \\
\{B:b2, \text{GRP:}\{\{A:a1\}\}\}\}
\]

\[
\text{SELECT DISTINCT } x.A, \\
\quad (\text{SELECT } y.B \text{ FROM coll } y \text{ WHERE } x.A = y.A) \text{ as GRP} \\
\text{FROM coll } x
\]
Nesting (like group-by)

A flat collection

\[
\text{coll} = \{\{A:a1, B:b1\}, \{A:a1, B:b2\}, \{A:a2, B:b1\}\}
\]

\[
\text{Nest}_A(\text{coll}) = \{\{A:a1, \text{GRP}:[\{B:b1\},\{B:b2\}]\}, \{A:a2, \text{GRP}:[\{B:b2\}]\}\}
\]

\[
\text{Nest}_B(\text{coll}) = \{\{B:b1, \text{GRP}:[\{A:a1\},\{A:a2\}]\}, \{B:b2, \text{GRP}:[\{A:a1\}]\}\}
\]

SELECT DISTINCT x.A,
    (SELECT y.B FROM coll y WHERE x.A = y.A) as GRP
FROM coll x

SELECT DISTINCT x.A, g as GRP
FROM coll x
LET g = (SELECT y.B FROM coll y WHERE x.A = y.A)
Grouping and Aggregates

Count the number of elements in the G array for each A

SELECT x.A, COLL_COUNT(x.G) AS cnt
FROM C AS x

SELECT x.A, COUNT(*) AS cnt
FROM C AS x, x.F AS y
GROUP BY x.A

These are NOT equivalent!
### Grouping and Aggregates

<table>
<thead>
<tr>
<th>Function</th>
<th>NULL</th>
<th>MISSING</th>
<th>Empty Collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLL_COUNT</td>
<td>counted</td>
<td>counted</td>
<td>0</td>
</tr>
<tr>
<td>COLL_SUM</td>
<td>returns NULL</td>
<td>returns NULL</td>
<td>returns NULL</td>
</tr>
<tr>
<td>COLL_MAX</td>
<td>returns NULL</td>
<td>returns NULL</td>
<td>returns NULL</td>
</tr>
<tr>
<td>COLL_MIN</td>
<td>returns NULL</td>
<td>returns NULL</td>
<td>returns NULL</td>
</tr>
<tr>
<td>COLL_AVG</td>
<td>returns NULL</td>
<td>returns NULL</td>
<td>returns NULL</td>
</tr>
<tr>
<td>ARRAY_COUNT</td>
<td>not counted</td>
<td>not counted</td>
<td>0</td>
</tr>
<tr>
<td>ARRAY_SUM</td>
<td>ignores NULL</td>
<td>ignores NULL</td>
<td>returns NULL</td>
</tr>
<tr>
<td>ARRAY_MAX</td>
<td>ignores NULL</td>
<td>ignores NULL</td>
<td>returns NULL</td>
</tr>
<tr>
<td>ARRAY_MIN</td>
<td>ignores NULL</td>
<td>ignores NULL</td>
<td>returns NULL</td>
</tr>
<tr>
<td>ARRAY_AVG</td>
<td>ignores NULL</td>
<td>ignores NULL</td>
<td>returns NULL</td>
</tr>
</tbody>
</table>
Grouping and Aggregates

\[
\begin{align*}
\text{C} &\quad \text{G}\\
\{\text{A:a1, F:GU\{B:b1, B:b2\}}, \ G:\{\text{C:c1}\}\}, \\
\{\text{A:a2, F:GU\{B:b3, B:b4, B:NULL\}}, \ G:\{\}\}, \\
\{\text{A:a3, F:GU\{B:b6\}}, \ G:\{\text{C:c2, C:c3}\}\}\end{align*}
\]

Count the number of elements in the G array for each A

\[
\begin{align*}
\text{SELECT x.A, COLL_COUNT(x.G) AS cnt} \\
\text{FROM C AS x}
\end{align*}
\]

\[
\begin{align*}
\text{SELECT x.A, COUNT(*) AS cnt} \\
\text{FROM C AS x, x.F AS y} \\
\text{GROUP BY x.A}
\end{align*}
\]

These are NOT equivalent!
Joins

Two flat collection

coll1 = [{A:a1, B:b1}, {A:a1, B:b2}, {A:a2, B:b1}]
coll2 = [{B:b1, C:c1}, {B:b1, C:c2}, {B:b3, C:c3}]

SELECT x.A, x.B, y.C
FROM coll1 AS x, coll2 AS y
WHERE x.B = y.B

Answer

SELECT x.A, x.B, y.C
FROM coll1 AS x JOIN coll2 AS y ON x.B = y.B

[{{A:a1, B:b1, C:c1},
  {A:a1, B:b1, C:c2},
  {A:a2, B:b1, C:c1},
  {A:a2, B:b1, C:c2}}]
Outer Joins

Two flat collection

```
coll1      [{A:a1, B:b1}, {A:a1, B:b2}, {A:a2, B:b1}]
coll2      [{B:b1, C:c1}, {B:b1, C:c2}, {B:b3, C:c3}]
```

```
SELECT x.A, x.B, y.C
FROM coll1 AS x RIGHT OUTER JOIN coll2 AS y
ON x.B = y.B
```

Answer

```
[{A:a1, B:b1, C:c1},
 {A:a1, B:b1, C:c2},
 {A:a2, B:b1, C:c1},
 {A:a2, B:b1, C:c2},
 {B:b3, C:c3}]
```
Ordering

\[
\text{coll1} = \{\{A:a1, B:b1\}, \{A:a1, B:b2\}, \{A:a2, B:b1\}\}
\]

\[
\text{SELECT } x.A, x.B \\
\text{FROM } \text{coll AS } x \\
\text{ORDER BY } x.A
\]

Be careful of data type
Nesting allows us to keep collections of references in the place of a single value.

Sometimes these references are kept in a single string:

- The reference is a string of keys separated by space
- Need to use split(string, separator) to split it into a collection of foreign keys
Splitting

river

```
["name": "Donau", "-country": "SRB A D H HR SK BG RO MD UA"],
{"name": "Colorado", "-country": "MEX USA"},
... ]
```

```
SELECT ...
FROM country AS x, river AS y,
    split(y. `-country`, " ") AS z
WHERE x.-car_code` = z
```

```
split("MEX USA", " ") = ['"MEX", "USA"]
```
Behind the Scenes

Query Processing on NFNF data:

• Option 1: give up on query plans, use standard java/python-like execution

• Option 2: represent the data as a collection of flat tables, convert SQL++ to a standard relational query plan
Flattening SQL++ Queries

A nested collection

coll =
[{A:a1, F:[{B:b1},{B:b2}], G:[{C:c1}]},
{A:a2, F:[{B:b3},{B:b4},{B:b5}], G:[ ]},
{A:a1, F:[{B:b6}], G:[{C:c2},{C:c3}]}]
A nested collection

coll = [
{A:a1, F:[{B:b1},{B:b2}], G:[{C:c1}]},
{A:a2, F:[{B:b3},{B:b4},{B:b5}], G:[ ]},
{A:a1, F:[{B:b6}], G:[{C:c2},{C:c3}]}]

Relational representation

<table>
<thead>
<tr>
<th>id</th>
<th>A</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a1</td>
<td>1</td>
<td>b1</td>
</tr>
<tr>
<td>1</td>
<td>a1</td>
<td>1</td>
<td>b2</td>
</tr>
<tr>
<td>1</td>
<td>a1</td>
<td>1</td>
<td>b3</td>
</tr>
<tr>
<td>2</td>
<td>a1</td>
<td>1</td>
<td>b4</td>
</tr>
<tr>
<td>3</td>
<td>a1</td>
<td>1</td>
<td>b5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>parent</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b1</td>
</tr>
<tr>
<td>2</td>
<td>b3</td>
</tr>
<tr>
<td>3</td>
<td>b6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>parent</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c1</td>
</tr>
<tr>
<td>3</td>
<td>c2</td>
</tr>
<tr>
<td>3</td>
<td>c3</td>
</tr>
</tbody>
</table>
Flattening SQL++ Queries

A nested collection

```
coll = 
[{A:a1, F:[{B:b1},{B:b2}], G:[{C:c1}]}],
{A:a2, F:[{B:b3},{B:b4},{B:b5}], G:[ ]},
{A:a1, F:[{B:b6}], G:[{C:c2},{C:c3}]}]
```

RELATIONAL REPRESENTATION

<table>
<thead>
<tr>
<th>id</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td>2</td>
<td>a2</td>
<td>b2</td>
<td>c2</td>
</tr>
<tr>
<td>3</td>
<td>a1</td>
<td>b3</td>
<td>c3</td>
</tr>
</tbody>
</table>

SQL++

```
SELECT x.A, y.B
FROM coll AS x, x.F AS y
WHERE x.A = "a1"
```
Flattening SQL++ Queries

A nested collection

```
coll =

[A:a1, F:[{B:b1},{B:b2}, G:[{C:c1}]},
 {A:a2, F:[{B:b3},{B:b4},{B:b5}, G:[]},
 {A:a1, F:[{B:b6}, G:[{C:c2},{C:c3}]]}
```

Relational representation

```

<table>
<thead>
<tr>
<th>id</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a1</td>
<td>b1</td>
</tr>
<tr>
<td>2</td>
<td>a2</td>
<td>b2</td>
</tr>
<tr>
<td>3</td>
<td>a1</td>
<td>b3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>parent</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c1</td>
</tr>
<tr>
<td>3</td>
<td>c2</td>
</tr>
<tr>
<td>3</td>
<td>c3</td>
</tr>
</tbody>
</table>
```

SQL++

```
SELECT x.A, y.B
FROM coll AS x, x.F AS y
WHERE x.A = "a1"
```

SQL

```
SELECT x.A, y.B
FROM coll AS x, F AS y
WHERE x.id = y.parent AND x.A = "a1"
```
Flattening SQL++ Queries

A nested collection

```
coll = 
[{A:a1, F:{B:b1},{B:b2}, G:{C:c1}}],
{A:a2, F:{B:b3},{B:b4},{B:b5}, G:[]},
{A:a1, F:{B:b6}, G:{C:c2},{C:c3}}]
```

Relational representation

<table>
<thead>
<tr>
<th>id</th>
<th>A</th>
<th>parent</th>
<th>B</th>
<th>parent</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a1</td>
<td>1</td>
<td>b1</td>
<td></td>
<td>c1</td>
</tr>
<tr>
<td>2</td>
<td>a2</td>
<td>1</td>
<td>b2</td>
<td>3</td>
<td>c2</td>
</tr>
<tr>
<td>3</td>
<td>a1</td>
<td>2</td>
<td>b3</td>
<td>3</td>
<td>c3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>b4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>b5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>b6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SQL++

```
SELECT x.A, y.B
FROM coll AS x, x.F AS y
WHERE x.A = "a1"
```

SQL

```
SELECT x.A, y.B
FROM coll AS x, F AS y
WHERE x.id = y.parent AND x.A = "a1"
```

```
SELECT x.A, y.B
FROM coll AS x, x.F AS y, x.G AS z
WHERE y.B = z.C
```
Flattening SQL++ Queries

**A nested collection**

coll =
{A:a1, F:[{B:b1},{B:b2}], G:[{C:c1}]},
{A:a2, F:[{B:b3},{B:b4},{B:b5}], G:[ ]},
{A:a1, F:[{B:b6}], G:[{C:c2},{C:c3}]}

**Relational representation**

coll:

<table>
<thead>
<tr>
<th>id</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a1</td>
</tr>
<tr>
<td>2</td>
<td>a2</td>
</tr>
<tr>
<td>3</td>
<td>a1</td>
</tr>
</tbody>
</table>

F:

<table>
<thead>
<tr>
<th>parent</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>b1</td>
</tr>
<tr>
<td>1</td>
<td>b2</td>
</tr>
<tr>
<td>2</td>
<td>b3</td>
</tr>
<tr>
<td>2</td>
<td>b4</td>
</tr>
<tr>
<td>2</td>
<td>b5</td>
</tr>
<tr>
<td>3</td>
<td>b6</td>
</tr>
</tbody>
</table>

G:

<table>
<thead>
<tr>
<th>parent</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>c1</td>
</tr>
<tr>
<td>3</td>
<td>c2</td>
</tr>
<tr>
<td>3</td>
<td>c3</td>
</tr>
</tbody>
</table>

**SQL++**

SELECT x.A, y.B
FROM coll AS x, x.F AS y
WHERE x.A = 'a1'

**SQL**

SELECT x.A, y.B
FROM coll AS x, F AS y
WHERE x.id = y.parent AND x.A = 'a1'

SELECT x.A, y.B
FROM coll AS x, F AS y, G AS z
WHERE y.B = z.C
WHERE x.id = y.parent AND x.id = z.parent
AND y.B = z.C
Semistructured Data Model

• Several file formats: JSON, protobuf, XML
• The data model is a tree
• They differ in how they handle structure:
  – Open or closed
  – Ordered or unordered
• Query language needs to take NFNF into account
  – Various “extra” constructs introduced as a result
Conclusion

• Semi-structured data best suited for *data exchange*

• “General” guidelines:
  – For quick, ad-hoc data analysis, use a “native” query language: SQL++, or AQL, or Xquery
    • Where “native” = how data is stored
  – Modern, advanced query processors like AsterixDB / SQL++ can process semi-structured data as efficiently as RDBMS
  – For long term data analysis: spend the time and effort to normalize it, then store in a RDBMS