Querying, Integrating and Extracting Data

Faculty of Computer Science
Workflow Systems and Technologies

Interoperability SS 2023

Amolkirat Singh Mangat
Matthias Ehrendorfer
Florian Stertz

Introduction

Can we treat the Web as a database with a schema?

Can we search the Web for specific information?

Data Schema

- A data model or schema imposes a clear structure on information.
- Typical example: relational database management systems (RDBMS) that enforce a horizontal schema
  - Each column (attribute) of a relation (table) represents a specific data type.
  - Each row represents a tuple of data elements.
  - Structured Query Language (SQL) enables to query against relations using elements of schema.

```
CREATE Table foo(
  bar text,
  primary key(bar)
);
```

```
SELECT *
FROM foo
WHERE foo.bar = "foobar";
ORDER BY foo.bar
```

Classification of Data

- Data can be classified into *unstructured, semi-structured and structured* data in terms of the *ability to search* for specific information.
- **Unstructured Data**
  - Does not follow any particular arrangement of information that would be enforced by a data schema.
  - Internally it may be composed of structured building blocks.
  - Typical examples include text, images and videos.
  - Search in unstructured data is more challenging.

    E.g.: *Search for all images which have tea cups and shortbread biscuits*.

Classification of Data

- **Structured Data**
  - In contrast to unstructured data, structured data is strictly arranged with regards to a data schema (e.g. a relational schema).
  - Lends itself to structured search against elements defined in the schema.
- **Semi-Structured Data**
  - May not be constrained by a schema; if a schema is present it may only be loosely imposed. This form of flexibility can be highly desirable!
  - Schema usually *inherent part of data*; thus it is *"self-describing"* [2].

Dimensions of Data Integration

- Data is being generated and collected at an unprecedented scale:
  - E.g. 1 *petabytes (1.000.000.000.000.000 bytes = 10^15 bytes = 1000 terabytes)* of collision data is produced by CERN's LHC per second! [4]
- Goal is to **analyze** and **extract** value from data for data-driven decision making that impacts various aspects of society. [3]
- Value of data sources increases if they can be **related** and **unified** into *a **shared representation***.

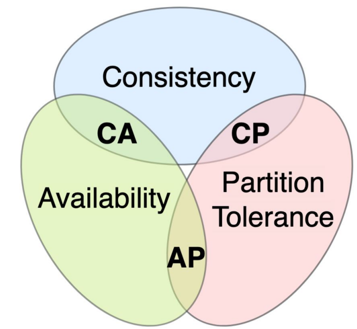
Dimensions of Data Integration

Key dimensions of data integration to consider are [3]:

- **Volume** Data sources can contain huge volumes of data.
- **Velocity** The growth in volume is determined by the *rate at which data is collected* and made available. Data sources are dynamic; they *evolve* and *increase* over time.
- **Variety** Data sources can be heterogeneous at the *schema level* with regards to how data is structured but also in terms of how an entity is described at the *instance level*.
- **Veracity** Quality of data sources may vary significantly (also when they are part of the same domain) in terms of *coverage, accuracy and timeliness* of the provided data.

Non-Relational Databases

- In addition to relational databases (e.g. PostgreSQL) a number of alternative database types are known under the umbrella term NoSQL (not only SQL) [5].
- NoSQL databases can be classified into
  - Document databases (e.g. MongoDB)
  - Key-Value databases (e.g. Redis)
  - Column-oriented databases (e.g. Cassandra)
  - Graph databases (e.g. Neo4j)
  - (Object-oriented databases)
- Usually based on the **BASE** principle (basically-available, soft-state, eventual consistency) in contrast to the - traditional - **ACID** model (atomicity, consistency, isolation, durability). Also see **CAP** theorem.



CAP Theorem [6]

Data Extraction via SQL/XML and SQL/JSON

- SQL/XML and SQL/JSON enable to extract relational data as XML and JSON respectively.
- Both are standardised, SQL/XML by ISO/IEC 9075 and SQL/JSON by ISO/IEC 19075-6.
- Check the documentation of databases for the coverage of the standards.


```xml
<ROWS>
  <ROW id="1">
    <COUNTRY_ID>AU</COUNTRY_ID>
    <COUNTRY_NAME>Australia</COUNTRY_NAME>
  </ROW>
  <ROW id="5">
    <COUNTRY_ID>JP</COUNTRY_ID>
    <COUNTRY_NAME>Japan</COUNTRY_NAME>
    <PREMIER_NAME>Shinzo Abe</PREMIER_NAME>
    <SIZE unit="sq_mi">145935</SIZE>
  </ROW>
  <ROW id="6">
    <COUNTRY_ID>SG</COUNTRY_ID>
    <COUNTRY_NAME>Singapore</COUNTRY_NAME>
    <SIZE unit="sq_km">697</SIZE>
  </ROW>
</ROWS>
```

```json
{
  "track": {
    "segments": [
      {
        "location":   [ 47.763, 13.4034 ],
        "start time": "2018-10-14 10:05:14",
        "HR": 73
      },
      {
        "location":   [ 47.706, 13.2635 ],
        "start time": "2018-10-14 10:39:21",
        "HR": 135
      }
    ]
  }
}
```

Search, Integrate and Extract Data with

- PostgreSQL - an object-relational database
- Supports storage of document data (XML, JSON) and key-value pairs as a single attribute value, in addition to the traditional primitive types normally associated with a relational database.
- Supports transformation of tabular data into XML and JSON.
- Also provides means to search in documents in combination with SQL via XPATH and SQL/JSON path language.
- See documentation for SQL/XML and SQL/JSON for further details.

Example - Data Source in PostgreSQL

```
CREATE TABLE beings(
  being_name      text,
  power_level     bigint,
  address         xml,
  powerfoods      jsonb,

  primary key (being_name)
);


CREATE TABLE being_traits(
  being_name      text,
  trait_code      text,
  trait_score     int,        -- 1 lowest and 5 highest score

  foreign key (being_name) references beings(being_name),
  primary key (trait_code, being_name)
);
```

Example - Data Source in PostgreSQL

```
insert into beings values
  ('Storm', 9900,
   '<address><location city=""/><planet>Earth</planet></address>',
   '[]'),
  ('Batman', 1000,
   '<address><location city="Gotham City"/><planet>Earth</planet></address>',
   '[{"food":"Oatmeal"}, {"food":"Grilled Chicken"}]'),
  ('Black Widow', 800,
   '<address><location city=""/><planet>Earth</planet></address>',
   '[{"food":"Medium Rare Steak"}, {"food":"Raspberry Ice Cream"}]'),
  ('Popeye', 199,
   '<address><location city="Mellieha"/><planet>Earth</planet></address>',
   '[{"food":"spinach"}]');
```

Example - Data Source in PostgreSQL

```
insert into being_traits values
  ('Storm',       'stealth', 3),
  ('Batman',      'stealth', 4),
  ('Black Widow', 'stealth', 5),
  ('Popeye',      'stealth', 1),

  ('Storm',       'charisma', 4),
  ('Batman',      'charisma', 1),
  ('Black Widow', 'charisma', 2),
  ('Popeye',      'charisma', 1),

  ('Storm',       'ruthlessness', 4),
  ('Batman',      'ruthlessness', 5),
  ('Black Widow', 'ruthlessness', 3),
  ('Popeye',      'ruthlessness', 2);
```

SQL/XML

Query

```
select
  xmlelement(
    name being,
    xmlattributes(power_level as powlvl),
    being_name)
from
  beings;
```

Output (4 rows)

<being powlvl="9900">Storm</being>
<being powlvl="1000">Batman</being>
<being powlvl="800">Black Widow</being>
<being powlvl="199">Popeye</being>

SQL/XML

Query

```
select
  xmlelement(
    name being,
    xmlattributes(power_level as powlvl),
    being_name) as being
from
  beings;
```

Output (4 rows)

<being powlvl="9900">Storm</being>
<being powlvl="1000">Batman</being>
<being powlvl="800">Black Widow</being>
<being powlvl="199">Popeye</being>

`xmlelement` expression creates an XML element.

`xmlattributes` expression creates attributes for the XML element.

SQL/XML

Query

```
select xmlagg(beings.being)
from
(select
  xmlelement(
    name being,
    xmlattributes(power_level as powlvl),
    being_name) as being
from
  beings) beings;
```

Output (1 row)

<being powlvl="9900">Storm</being>
<being powlvl="1000">Batman</being>
<being powlvl="800">Black Widow</being>
<being powlvl="199">Popeye</being>

SQL/XML

Query

```
select xmlagg(beings.being)
from
(select
  xmlelement(
    name being,
    xmlattributes(power_level as powlvl),
    being_name) as being
from
  beings) beings;
```

Output (1 row)

<being powlvl="9900">Storm</being>
<being powlvl="1000">Batman</being>
<being powlvl="800">Black Widow</being>
<being powlvl="199">Popeye</being>

`xmlagg` an aggregate functions that concatenates XML fragments across rows.

SQL/XML

Query

```
select
  xmlelement(
    name being,
    xmlforest(
      being_name as name,
      power_level,
      address as whereabouts))
from beings
where being_name = 'Storm';
```

Output (1 row)

```
<being>
  <name>Storm</name>
  <power_level>9900</power_level>
  <whereabouts>
    <address>
      <location city=""/>
        <planet>Earth</planet>
    </address>
  </whereabouts>
</being>
```

SQL/XML

Query

```
select
  xmlelement(
    name being,
    xmlforest(
      being_name as name,
      power_level,
      address as whereabouts))
from beings
where being_name = 'Storm';
```

Output (1 row)

```
<being>
  <name>Storm</name>
  <power_level>9900</power_level>
  <whereabouts>
    <address>
      <location city=""/>
        <planet>Earth</planet>
    </address>
  </whereabouts>
</being>
```

`xmlforest` expression produces a sequence of XML elements.

SQL/XML

Query

```
select
  xmlelement(
    name beings,
    xmlagg(btraits.btraits))
from
  (select
    xmlelement(
      name being,
      xmlattributes(
        rhs.being_name as name,
        rhs.power_level as level),
      xmlagg(
        xmlelement(
          name trait,
          xmlattributes(
            trait_code as name,
            trait_score as score)))) btraits
  from being_traits lhs right join beings rhs
       on lhs.being_name = rhs.being_name
  where rhs.power_level >= 1000
  group by rhs.being_name) btraits;
```

Output (1 row)

```
<beings>
  <being name="Batman" level="1000">
    <trait name="stealth" score="4"/>
    <trait name="charisma" score="1"/>
    <trait name="ruthlessness" score="5"/>
  </being>
  <being name="Storm" level="9900">
    <trait name="stealth" score="3"/>
    <trait name="charisma" score="4"/>
    <trait name="ruthlessness" score="4"/>
  </being>
</beings>
```

References

[1] Agrawal, Rakesh, Amit Somani, and Yirong Xu. "Storage and querying of e-commerce data." VLDB. Vol. 1. 2001.

[2] Buneman, Peter. "Semistructured data." Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems. 1997.

[3] Dong, Xin Luna, and Divesh Srivastava. "Big data integration." 2013 IEEE 29th international conference on data engineering (ICDE). IEEE, 2013.

[4] https://home.cern/news/news/computing/cern-data-centre-passes-200-petabyte-milestone, last accessed 15.03.2023

[5] Han, Jing, et al. "Survey on NoSQL database." 2011 6th international conference on pervasive computing and applications. IEEE, 2011.

[6] https://en.wikipedia.org/wiki/CAP_theorem#/media/File:CAP_Theorem_Venn_Diagram.png, last accessed 16.03.2023