Advanced Topics in Databases
Database and Software Engineering Working Group

Magdeburg, June 2019

Summer Term 2019

Marcus Pinnecke, M.Sc.
Prof. Dr. Gunter Saake

Task 1 The Case for Document Stores? **7** (1 + 2 + 3 + 1) **Points**

Document stores manage denormalized records that must not typically match a predefined schema.

1. Name and describe at least two application scenarios that imply or require semi-structured data!
2. Imagine an application scenario where a schema can be defined up-front and does not change too much afterwards. Assume further that many entities have relationships to each other. Discuss the claim: "*A semi-structured data model is a reasonable choice for this use case*"!
3. Denormalization lead to the risk of low data integrity due to *data anomalies* resulting from data redundancy and update, insertion or deletion operations that do not affect each copy of the data. Discuss *update*, *insertion*, and *deletion anomalies* in context of semi-structured data!
4. List at least two use cases that benefit from semi-structured data despite the fact of data anomalies, and give one example each!

Task 2 JSON and the Document Database Model **3** (1 + 1 + 1) **Points**

The data model of document stores centers around records that are JSON-like.

1. Explain the terms *document* and *collection* (resp. database)! For documents, include statements on the schema, data normalization, object identification, nesting and referencing in your explanation. For collections, include statements on schema across records contained in the collection.
2. State whether the following statements are *true* or *false*! Give an explanation!
   a. JSON is a human-readable markup language similar to XML.
   b. JSON is designed for applications having unspecific knowledge of their data.
   c. JSON is not a general serialization format and language independent.
   d. JSON represents primitive, and structured data types.
   e. The string `[[{answer:"no"}]]` is a valid JSON file according to the latest specification.
   f. JSON is self-describing; there's no mechanism in JSON for schema verification.
3. JSON Pointers is a concept to enable references to specific value within a JSON document. Construct a minimal example for a JSON file for which the JSON pointer `/x/1/y/0/4` evaluates to a numeric value of `42` !

Task 3 Hands on Document Stores **4** (1 + 3) **Points**

Document stores are database system that store, retrieve and manage semi-structured data. This system class defines one of the main categories of modern NoSQL databases and is trending in popularity.

1. MongoDB and CouchDB are two prominent implementation for document stores. Compare both systems with respect to their storage engine design! Include the following in your comparison:

   a. Which concurrency control mechanism is used?

   b. What is about consistency, availability, and/or partition tolerance (cf. CAP theorem)?

   c. How is the support for mapreduce, filter operations, and further aggregation queries?

   Afterwards, explain the master-master architecture of CouchDB and the sharding architecture of MongoDB!

2. Download and setup *either* an instance of MongoDB and CouchDB on your system. Additionally, clone the *libcarbon* repository from GitHub[1], and checkout the branch `teaching/atdb/2019`. In this branch you will find the directory `ds/`, which contains excerpts of pre-processed datasets. The *GitHub Repository API Excerpt* dataset is the one you will work with.

   Create either a new database in CouchDB or a new collection in MongoDB for this dataset, and import the file `ds/github-repo-api/snapshot-excerpt.json`!

   > **Tip**: For MongoDB look for a tool called `mongoimport` and use the flag `--jsonArray`
   >
   > **Tip**: For CouchDB you may want to import the dataset as a bulk, see http://docs.couchdb.org/en/2.3.1/api/database/bulk-api.html#inserting-documents-in-bulk for documentation. Further tip: the github dataset must be wrapped with some additional text to match CouchDBs importer syntax.

   Afterwards, implement the following queries in the database of your choice (either MongoDB or CouchDB):

   a. Give the value for the key `"html_url"` for the research paper having the property `"name": "Python"` by executing a database query! Additionally, give your query statement!

   b. Give the set of key names (1st level properties (no nested object), no duplicate key names) for all research papers stored in the database! You may use mapreduce for this purpose! Additionally, give your query statement.

[1] Type the following in your bash

```
$ git clone https://github.com/protolabs/libcarbon.git && cd libcarbon && git checkout -b
  teaching/atdb/2019 origin/teaching/atdb/2019
```

Task 4 Document Stores under the Hood **2** (1 + 1) **Points**

Despite differences in their data model, document stores share many conceptual ideas and internal structures with relational systems. However, due to a focus on scalability that heavily benefit from the denormalization of stored records, document stores typically have significant differences at storage level.

1. Consider the append-only storage of CouchDB, and the update-in-place storage of MongoDB. Explain how database modifications (inserts and updates) are handled, and discuss benefits and drawbacks of each approach.
2. Recap the default storage engine of MongoDB since version 3.2, WiredTiger. Explain where B+-tree structures are used.

Task 5 Semi-Structured Data by Bits and Bytes **9** (3 + 3 + 3) **Points**

Document records are physically organized by a variety of data formats, that result from a diversity of applications (such as fast parsability, understandability, or expressibility) .

1. Justify the statement "*There is no 'One-Size-Fits-All' format for representation of semi-structured data*" w.r.t. the diversity of application requirements, and discuss this statement for at least two formats not already mentioned in the lecture.
2. Clone the *libcarbon* repository from GitHub[2], and checkout the branch `teaching/atdb/2019`. In this branch you will find the directory `ds/`, which contains excerpts of pre-processed datasets. The *GitHub Repository API Excerpt* dataset is the one you will work with. Analyze the (disk) size requirements for the following formats on the GitHub Repository API dataset snapshot (that you must download from our FTP server) by converting the snapshot into each format, and compare them to the plain-text JSON format.

   (a) Universal Binary JSON (UBJSON)
   Tip: Find a library or tool under ubjson.org/libraries/ that matches your preferences

   (b) Binary JSON (BSON)
   Tip: Study the toolchain of MongoDB, which provides an export to BSON

   (c) Columnar Binary JSON (CARBON)
   Tip: Build `carbon-tool` from sources in the branch *teaching/atdb/2019* from our repository github.com/protolabs/libcarbon (see README.md), and run in your bash:

```
$ build/carbon-tool convert --size-optimized --no-string-id-index github-repo-api.carbon
        ds/github-repo-api/snapshot-excerpt.json
```

[2] Type the following in your bash
```
$ git clone https://github.com/protolabs/libcarbon.git && cd libcarbon && git checkout -b
  teaching/atdb/2019 origin/teaching/atdb/2019
```

(conversion with `carbon-tool` may take XXX min for this dataset)

Tip: Use a Linux distribution or macOS as operating system to match the building tools and tool chains.

3. Revisit your results from sub task 2 for each format (including the given plain-text JSON format). Speculate on the reason why your see differences in the file sizes for each format.

Task 6 Semi-Structure for a Structured Query Language **9** (2 + 2 + 1 + 1 + 3) **Points**

SQL is the de-facto industry standard for the lingua franca of (relational) database systems. In its last 45 years of existence, SQL evolved heavily. By the latest standard SQL:2016, support for JSON data was added to the language.

1. Consider the groups of new functionality in SQL: *validation*, *construction*, *querying* functionality and the *SQL/JSON Path Language*. For each group, sketch roughly its purpose in your own word.
2. Recap the concepts of *strict* and *lax* mode in SQL/JSON Path Language. Explain the purpose of both modes, and their differences. Afterwards, provide one example query strings that only differs in their mode, and roughly explain how the mode affects the query evaluation.
3. Recap the concepts of *context variable* and *named variables* in the SQL/JSON Path Language. Explain the purpose of both concepts with your own words, and provide an example each.
4. State the evaluation semantics of the following
   a. Member access
   b. Element access
   c. Filter expressions
5. Explain the SQL/JSON Path Language item functions `type()` and `keyvalue()` in your own words!
6. Assume you the following two facts in a database

   F_1 *Jane is a woman but her age is not known (null).*

   F_2 *Joe is a man but his age is not known (null).*

   Recap the semantics for equality comparison of *unknown (null)* values in SQL/JSON Path Language and SQL. Now, it is asked *whether Jane and Joe have the* same *age*. What is the answer to this query in SQL/JSON Path Language and what is the answer in SQL? Give an explanation!

   A_1 *Jane and Joe have the same age.*

   A_2 *Jane and Joe doesn't have the same age.*

A_3 *It is not known whether Jane and Joe have the same age.*

Task 7 Hands on Evaluation in SQL/JSON Path Language **3** (3) **Points**

The SQL/JSON Path Language is a dedicated language built into SQL. This new language comes with its own semantics that slightly differ from the host language SQL, e.g., equality comparison results for *Unknown (Null)*.

Recap the example of property access that does not exists for all array entries (given in the lecture). Evaluate the following SQLJ/SON Path Language query step by step. For each step, show intermediate results and give a comment explaining this step. Is the filter expression required for this query?

```
lax $.authors[*] ? (exists (@.org)).org
```

Finally, explain the differences to the queries

```
strict $.authors[*] ? (exists (@.org)).org
```

and

```
lax $.authors[*].org
```