DocuDB – A Document Oriented Database

CS123: Projects in Database Systems
Spencer Strumwasser, Ethan Lo

Supported Operations

Commands:
- Create – Creates a collection of documents
- Insert – Creates a document and places it into a collection
- Select – Selects the documents that satisfy the predicate
- Update/Upsert – Updates all the columns in the document satisfying a predicate
  - Update: if column does not exist in the document, the column does not get added
  - Upsert: If column does not exist in the document, the column gets added to the document
- Delete – Deletes the document from the collection. Also can delete a column from the document
- Drop – Deletes the collection

Operations that affect commands:
- Where – Applies a predicate to the command. Can be used with Select, Update/Upsert and Delete
- Projections – Specifies what select or Delete applies to.

Syntax

❖ < > denotes variable
❖ The where clause is optional in all scenarios
- create <collection>
- insert into <collection> <key_name> {col_name1 : val1, col_name2: val2, ...}
- select <projection> from <collection> where <expression(s)>
- update <collection> set [col name1, colname2] = [value1, value2] where <expressions>
- upsert <collection> set [col name1, colname2] = [value1, value2] where <expressions>
- delete <projection> from <collection> where <expression(s)>
- drop <collection>

Language Parsing

1. Lexer generates token list from query string
2. Parser takes token list and generates Command object (essentially a struct) with the info needed to execute the query
   - Checks for syntax errors during this process
3. Command object parameters are checked to see if they violate any database constraints (column name constraints, value type constraints, etc.).
4. Parser object creates a new instance of the Storage Layer class, and uses the Command object to execute the query.

What is a Document Oriented Database

A document-oriented database is a NoSQL type database where instead of having tables filled with columns and rows that then can be related to a user, it instead has collections of documents where each document contains all the information applicable to it.

Document-oriented databases have become very popular for its ability to deal with data without the use of joins. MongoDB, currently one of the most famous document-oriented databases has customers across many different industries and company sizes.

The thing that separates document-oriented databases from other similar databases , such as key-value store, allow the benefits of fast look-ups on the unique key, however the data is accessible for querying.

Because of all of this, our group has created DocuDB, a document-oriented database built in python.

Comparison With Relational Model

- Data Model
  - Relational
    - Data stored in separate tables with defined schema
    - One object described by several tables
      - Normalized: no redundant data
    - Tables have referential integrity constraints
  - Document
    - Flexible schema. Each document can have different fields
    - One object is usually described by one collection
      - Denormalized: often redundant data
    - Usually doesn't support joins or referential integrity constraints
    - Flexible schema -> faster software development
- Scalability
  - Horizontal scaling: add more servers
    - No downtime and usually cheaper
    - Harder to manage multiple servers
  - Vertical scaling: beef up existing server - add more memory or CPUs
    - Limits on hardware
    - Downtime while adding new resources
  - Relational data has issues scaling horizontally
  - Document oriented data is great for horizontal scaling
- Data Integrity
  - Most document oriented databases are not ACID (Atomicity, Consistency, Isolation, Durability) compliant.
  - Relational databases are.

Document Storage Format

Document Format

Metadata	Row data	Extra padding

Metadata

Filled flag – 1B	Space allocated – 4B	Space currently used – 4B	Key name – 30B

Row-data

Column name length – 1B	Column name – 1~255B	Value type – 1B	Value size – 4B	Value – variable size

Storage Layer

Since one of the important attributes of a document-oriented database is the ease of changing the values of a document, meaning documents constantly change in size. Because of this, the storage layer was one of the most important things to design carefully and efficiently.

The documents are currently stored as a heap file.

Important details about document:
- Document size is power of 2B
- Max document size: 16MiB
- Min document size: 1KiB
- When document needs more space, it doubles in size

Document Traversal in Storage Layer

In order to traverse our storage layer, it is important to understand the metadata. Each document has metadata that contains:
- Filled Flag, denoting whether it is in use or not
- Space Allocated, how much memory is allocated
- Space Currently Used, determines how much of the allocated memory is in use
- Key Name, the document's key

We traverse through the documents by using "Space Allocated" to jump from one document to the next. To avoid loading the whole database into memory at once, we have a "read size" that determines how much gets loaded at one time.

Additionally, each column-value pair in the document contains its own metadata, which allows us to traverse through the data in the document easily.

Repository Link: https://github.com/SpencerStrumwasser/DOCUDB

References

- https://www.mongodb.com/compare/mongodb-mysql
- http://stackoverflow.com/questions/11707879/difference-between-scaling-horizontally-and-vertically-for-databases
- https://en.wikipedia.org/wiki/Document-oriented_database
- https://www.upwork.com/hiring/data/sql-vs-nosql-databases-whats-the-difference/

© Poster Template by Genigraphics® 1.800.790.4001 www.genigraphics.com