DBS101 Database Systems Fundamentals

Lesson 13

Royal University of Bhutan

Learning Outcomes

1. Understand the structure of JSON objects.
2. Translate schema to document based databases.
3. Basic operations on Document Databases

JSON objects

- JSON stands for JavaScript Object Notation
- JSON is a lightweight format for storing and transporting data.
- Documents in document databases are often represented as JSON objects.

Structure of JSON objects

```
{“customer_id”:187693,
“name”: “Kiera Brown”,
“address” : {
   “street” : “1232 Sandy Blvd.”,
    “city” :  “Vancouver”,
    “state” :  “Washington”,
    “zip” :  “99121”
  },
“first_order” : “01/15/2013”,
“last_order” : ” 06/27/2014”}
```

Structure of JSON objects

JSON objects are constructed using several simple syntax rules

- Data is organized in key-value pairs, similar to key-value databases
- Documents consist of name-value pairs separated by commas
- Documents start with a { and end with a }.
- Names are strings, such as "customer_id" and "address".
- Values can be numbers, strings, Booleans (true or false), arrays, objects, or the NULL value.
- The values of arrays are listed within square brackets, that is [and].
- The values of objects are listed as key-value pairs within curly brackets, that is, { and }.

XML: Extensible Markup Language

- XML (Extensible Markup Language) is a markup language similar to HTML, but without predefined tags to use.
- Documents in document database can also be stored as XML.

XML representation

```
<customer_record>
   <customer_id>187693</customer_id>
   <name>“Kiera Brown”</name>
    <address>
      <street>“1232 Sandy Blvd.”</street>
       <city>“Vancouver”</city>
       <state>“Washington”</state>
       <zip>“99121”</zip>
   </address>
    <first_order>“01/15/2013”</first_order>
    <last_order>“06/27/2014”</last_order>
  </customer_record>
```

Document databases vs Key-Value databases

- Documents manage related attributes within a single object.
- Document databases require less code than key-value data stores to query multiple attributes.
- Emulates relational tables by using a naming convention based on entity names, unique identifiers, and attribute names.
- Like relational tables, documents organize multiple attributes in a single object.
- Simplifies implementation of common requirements, such as filtering based on attributes and retrieving related data.

Documents and Collections

- A **record in MongoDB** is a **document**, which is a data structure composed of field and value pairs.
- MongoDB stores documents in collections. **Collections** are analogous to **tables** in relational databases.
- The full potential of document databases becomes apparent when you work with large numbers of documents. Documents are generally grouped into collections of similar documents.
- Collections can be thought of as lists of documents.

Documents and Collections

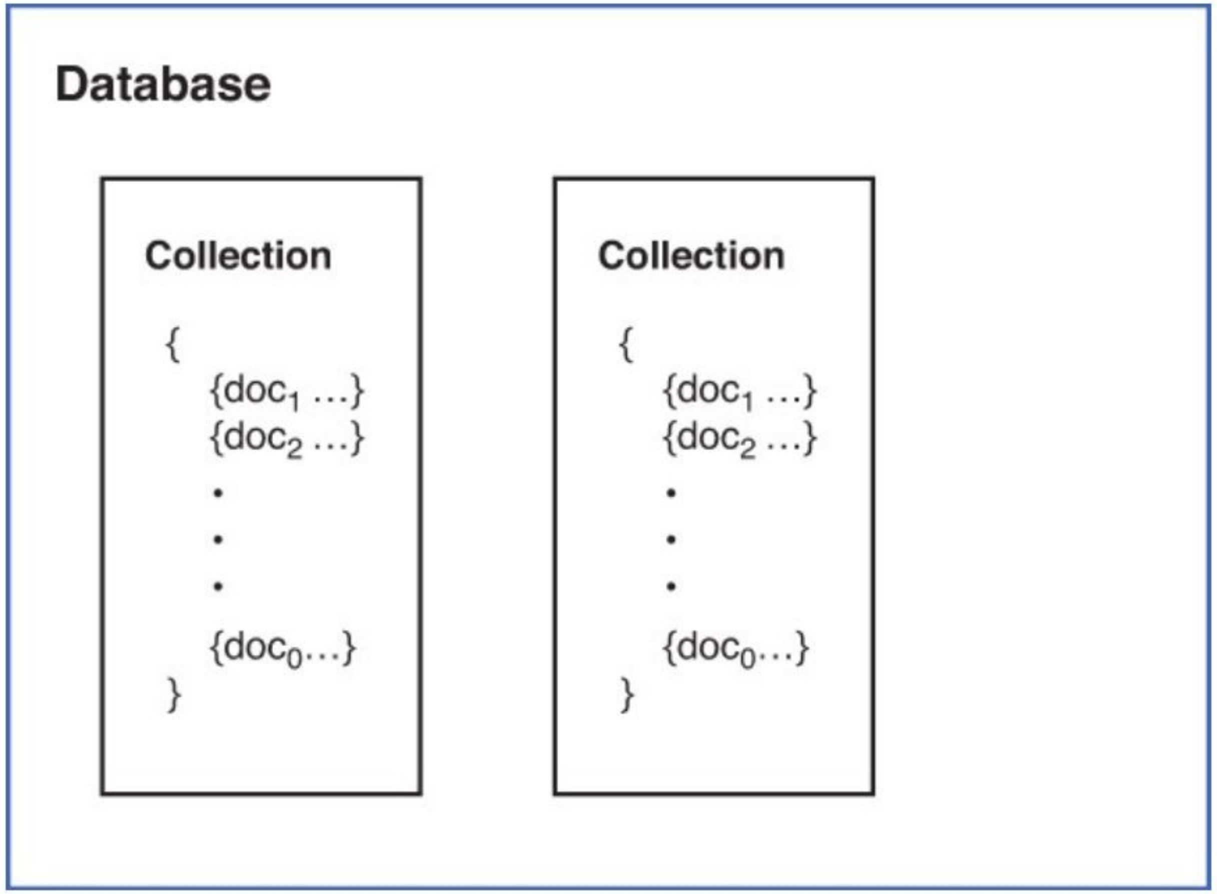

Figure 6.7 *The database is the highest-level logical structure in a document database and contains collections of documents.*

Documents in Mongo

```
{    “customer_id”:187694,
      “name”: “Bob Brown”,
      “address” : {
                    “street” : “1232 Sandy Blvd.”,
                     “city” :  “Vancouver”,
                     “state” :  “WA”,
                     “zip” :  “99121”
                    },
     “first_order” : “02/25/2013”,
    “last_order” : ” 05/12/2014”
     }
```

Documents in Mongo

- Use database

  // Mongodb creates a new database if it does not exist when you use the following command.

```
use TestDB
```

- Create collection

```
db.createCollection(“customers”);
```

Documents in Mongo

Why are we creating a collection here??

Documents in Mongo

```
db.customers.insertOne({
    "customer_id": 1,
    "name": "Bob",
    "address": {
        "street": "2 Norzin lam.",
        "city": "Thimphu",
        "state": "Thimphu",
        "zip": "21001"
    },
    "first_order": "01/02/2024",
    "last_order": "01/05/2024"
});
```

Collections in Mongo

Documents in the same collection do not need to have identical structures, but they should share some common structures.

Example; the following document can also be added to the collection although the structure is a bit different.

Collections in Mongo

```
db.customers.insertOne({
    "customer_id": 2,
    "name": "Dema",
    "address": {
        "street": "2 Norzin lam.",
        "city": "Thimphu",
        "state": "Thimphu",
        "zip": "21001"
    },
    "first_order": "01/28/2022",
    "last_order": "01/31/2024",
    "loyalty_level": "Gold",
    "large_purchase_discount": 0.05,
    "large_purchase_amount": 250.00
})
```

Designing Collections in Mongo

Collections are sets of documents in document databases.

- While collections don't enforce a consistent structure on documents, it's best to store documents of the same type in a single collection.

Designing Collections in Mongo

Avoiding Highly Abstract Entity Types

- Collections should ideally store documents about the same type of entity.

1. Filtering collections containing multiple document types can be slower compared to working with separate collections for each document type.
2. Retrieving blocks of data containing mixed document types can adversely impact performance, especially if using disk storage.

Note: A mix of entity types in the same collection may indicate a need to separate them into distinct collections for better performance and organization.

Designing Collections in Mongo

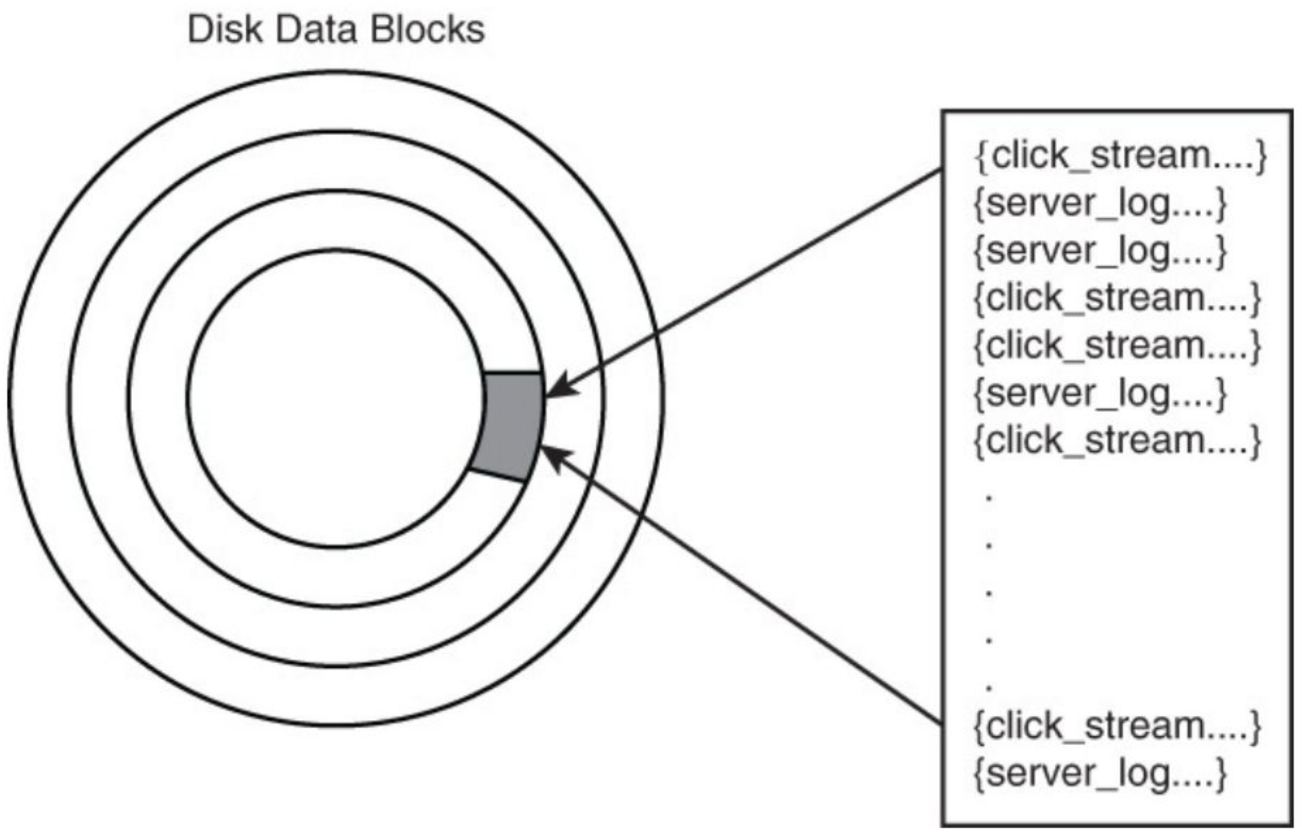

Figure 6.3 *Mixing document types can lead to multiple document types in a disk data block. This can lead to inefficiencies because data is read from disk but not used by the application that filters documents based on type.*

Collections in Mongo

Now should you never mix different document types and use indicators in code to handle different types?

Collections in Mongo

There are times when it makes sense to use document type indicators and have separate code to handle the different types.

Example:

Products Collection:

- Consider a scenario where you need to track various product types like books, CDs, and kitchen appliances.
- Each product type has specific attributes, but they share common fields like name, description, SKU, etc.

Collections in Mongo

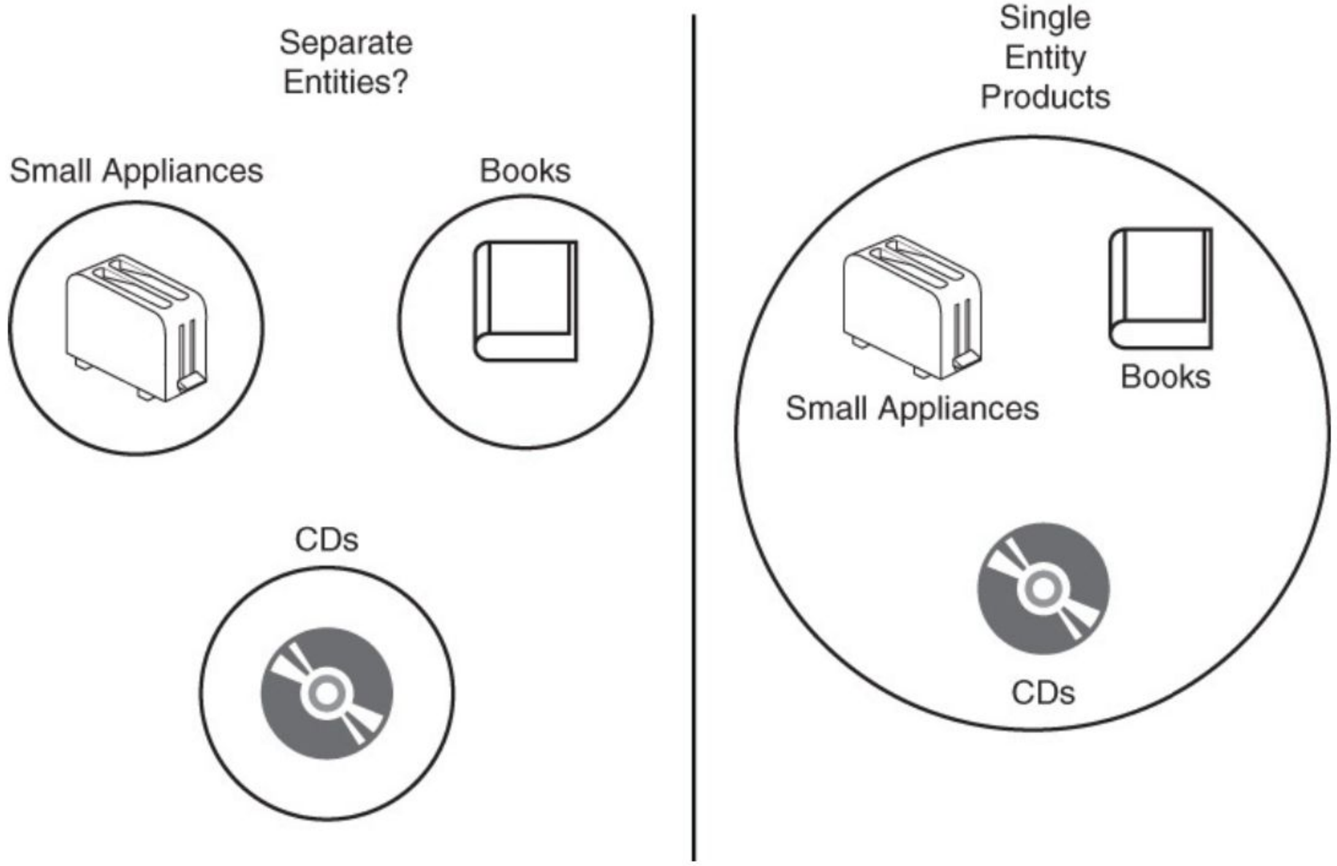

Figure 6.5 *When is a toaster the same as a database design book? When they are treated the same by application queries. Queries can help guide the organization of documents and collections.*

Collections in Mongo

- If queries involve data from all product types and frequently use them together, it's advisable to store them in a single document collection.
- This approach facilitates responding to queries and calculating derived values efficiently.

Collections in Mongo

Favor a single document collection when:

- Queries involve data from all product types.
- The client's product offerings are expected to grow, potentially adding new types.
- Maintaining multiple collections for numerous product types could become unwieldy as the number of types increases.

Collections in Mongo

Avoid schema definitions

Document databases do not require this formal definition step.

Instead, developers can create collections and documents in collections by simply inserting them into the database

RDBMs vs Document DBs

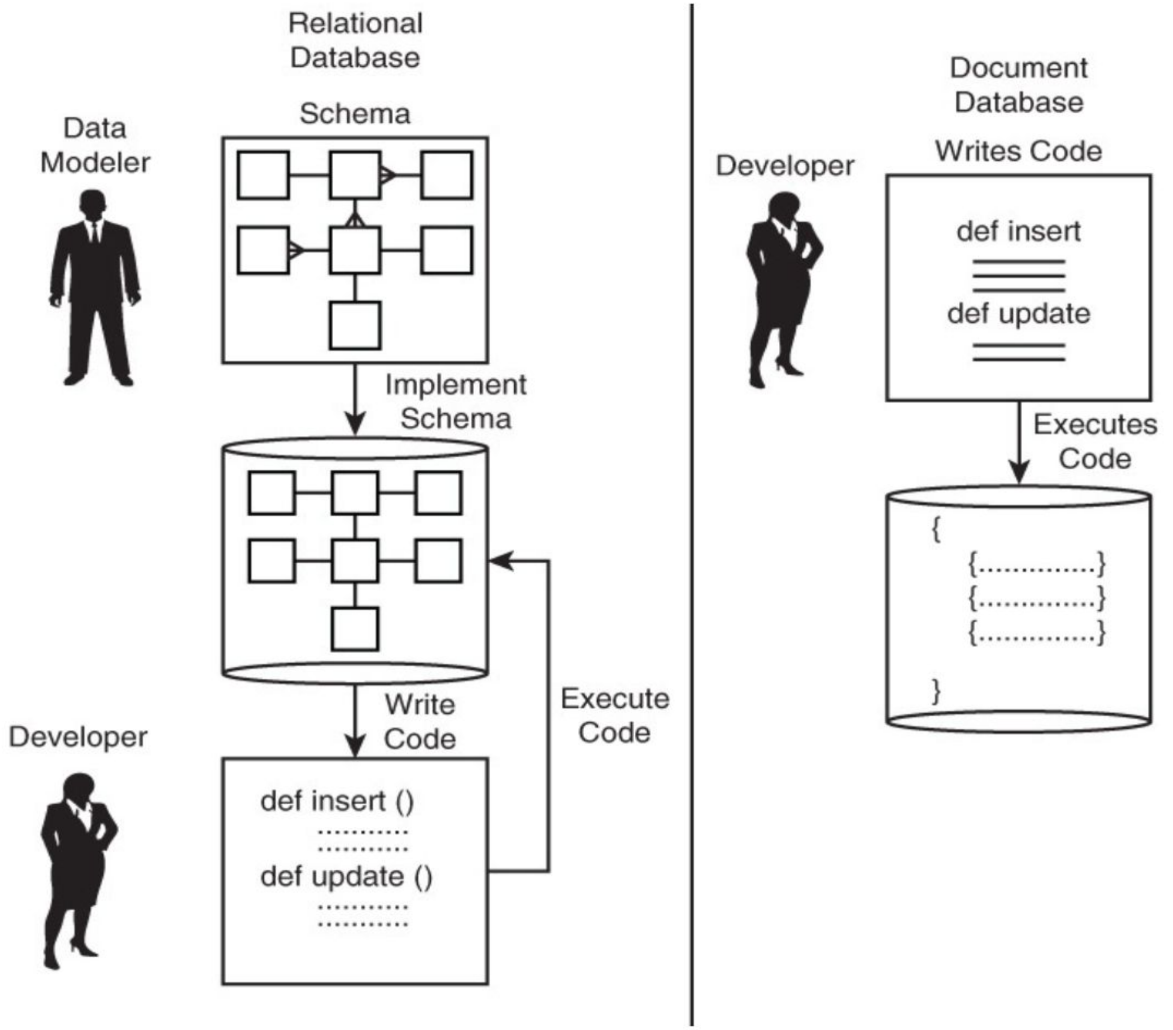

Figure 6.6 *Relational databases require an explicitly defined schema, but document databases do not.*

RDBMs vs Document DBs

1. Schemaless Nature:
   - Document databases are schemaless, meaning data modelers aren't required to specify all possible document fields before building and populating the database.
   - This flexibility allows for variations in documents within a collection, accommodating diverse data structures.

RDBMs vs Document DBs

2. Implicit Organization:

- While there's no need to define a schema beforehand, an implicit organization emerges from the set of documents inserted into the database.
- The organization becomes apparent in the code that manipulates the documents, where specific fields are set based on document type or purpose.

Basic operations in Document Databases

use Books – Create database

Inserting

- InsertOne - Insert one record
- InsertMany - Insert many records

Note: Collections can also be created when inserting records.

Basic operations in Document Databases

```
db.books.insertMany(
        [
{“book_id”: 1,
 “title”:“Mother Night”,
 “author”: “Kurt, Jr.”},

{“book_id”: 2,
 “title”:“Science and the Modern World”,
 “author”: “Alfred”},
{“book_id”: 3,
 “title”:“Foundation and Empire”,
 “author”: “Isaac”}  ])
```

Basic operations in Document Databases

```
db.books.insert(
        [
{“book_id”: 1,
 “title”:“Mother Night”,
 “author”: “Kurt, Jr.”},

{“book_id”: 2,
 “title”:“Science and the Modern World”,
 “author”: “Alfred”},
{“book_id”: 3,
 “title”:“Foundation and Empire”,
 “author”: “Isaac”}  ])
```

The [and] in the parameter list delimit an array of documents to insert.

Basic operations in Document Databases

Deleting

- **db.books.deleteOne({“book_id”: 1})** – Deletes one record that matches the query/
- **deleteMany** – Deletes more than one record associated with the query.

Basic operations in Document Databases

Updating

```
- db.books.updateOne ({“book_id”: 2},
                             {$set  {“quantity” : 10 }})
- updateMany
- findOneAndUpdate
- replaceOne
```

Basic operations in Document Databases

Retrieving

```
db.books.findOne({“author”: “Kurt, Jr.”},
                 {“title” : 1} )
```

- **find** – Retrieves all records associated with the query.

Cheat sheet

Next class:
Document Databases
- Types of Partitions
- Data Modeling and Query Processor
    - Normalization and Denormalization