Lecture Notes for Lecture 10 of CS 5200 (Database Management System) for the Summer 1, 2019 session at the Northeastern University Silicon Valley Campus.

NoSQL Databases and MongoDB

Philip Gust,
Clinical Instructor
Department of Computer Science

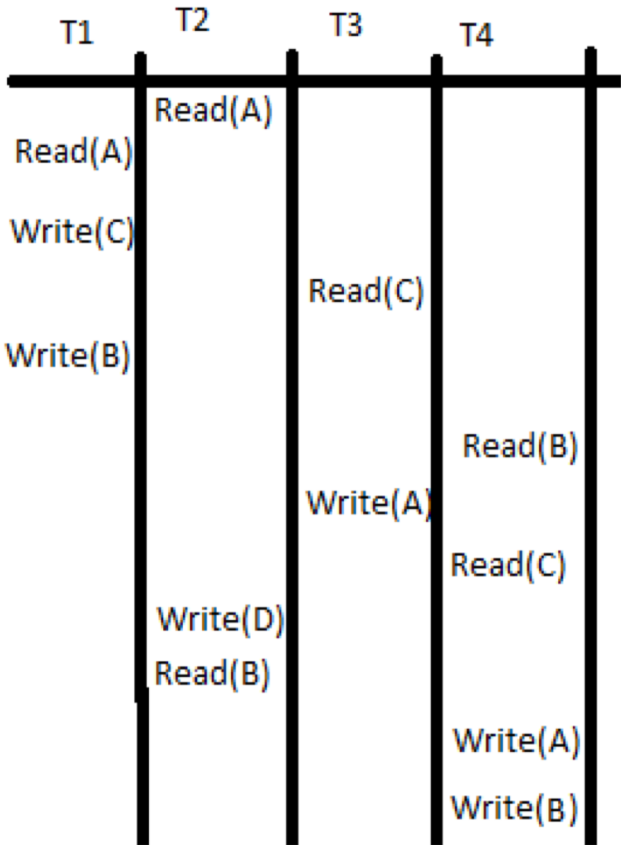
Material contained in this presentation is based in part on and uses content from recommended readings for the course.

Lecture 9 Review

- A transaction groups database operations together so that they their results are not visible outside the transaction until they either all succeed, or all of their effects are rolled back.
- A transaction must maintain a set of properties in order to ensure accuracy, completeness, and data integrity. These properties are referred to by the acronym ACID:
  - Atomicity
  - Consistency
  - Isolation
  - Durability

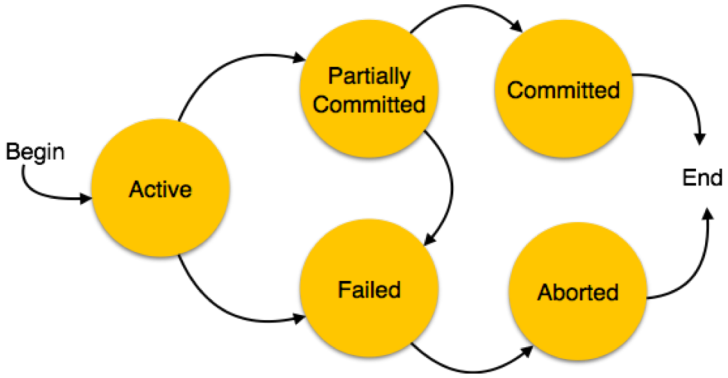
Lecture 8 Review

- When multiple transactions are being executed concurrently, instructions in one transaction are interleaved or serialized with those in other transactions.
- A schedule determines how the instructions are serialized.
- The sequence of operations in a transaction cannot be changed, but the operations across transactions can appear to occur randomly.



Lecture 8 Review

- Transactions have a lifecycle with transition states including Active, Partially Committed, Committed, and Failed



- Commands that control the state of a transaction include COMMIT, ROLLBACK, SAVEPOINT, and SET TRANSACTION.
- SQL supports both statement- and transaction-level runtime rollback when an error occurs, depending on nature and severity of the error.
- A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

Today's Topics

- In this lecture will introduce a different database model that is not based on SQL, which is often referred to as non-relational or NoSQL.
- We will examine the motivations behind NoSQL databases, how they represent data, and what embedded languages they use for data definition, data modeling, and queries.
- We will also introduce MongoDB, a NoSQL database that has become increasingly popular for dynamic web and big-data applications

Today's Topics

- We will also introduce MongoDB, a NoSQL database that has become increasingly popular for dynamic web and big-data applications
- We will review how to install MongoDB and the Java connector that provides a Java API for interacting with MongoDB, and how to create MongoDB applications
- We will also see how to use both the Mongo Shell interactive command interpreter, and NodeJS to perform operations in MongoDB's native JavaScript query language.
- Finally, we will look at how data is represented within MongoDB using JavaScript Object Notation (JSON) and its binary form, BSON.

NoSQL Databases and MongoDB

Non-Relational (NoSQL) Databases

- Another major type of database for managing data is not based on SQL, and is often referred to as non-relational or NoSQL.
- Entities are comprised of attribute/value pairs. The entities are not governed by a schema type. Instead, documents are only related in that they share the same set of attributes.
- Non-relational databases were originally motivated by *Web 2.0* applications, but are new increasingly used in big data and real-time web applications.
- NoSQL systems are also sometimes called "Not only SQL" to emphasize that they may support SQL-like query languages.

NoSQL Databases and MongoDB

Motivations for NoSQL Databases

- Motivations for this approach include:
  - simplicity of design
  - simpler "horizontal" scaling to clusters of machines
  - finer control over availability.
- The data structures used by NoSQL databases are different from those used by default in relational databases, making them more flexible, and for some operations, faster.
- The suitability of a given NoSQL database depends on the problem it must solve.

NoSQL Databases and MongoDB

Challenges for NoSQL Databases

- Barriers to the greater adoption of NoSQL stores include
  - use of low-level query languages
  - lack of standardized interfaces
  - huge previous investments in relational databases.

NoSQL Databases and MongoDB

Challenges for NoSQL Databases

- Many NoSQL stores compromise Consistency in favor of Availability, Partition tolerance (CAP theorem) and speed.
- Most NoSQL stores lack true ACID transactions, although a few databases have made them central to their designs.
- NoSQL databases offer a concept of *eventual consistency*: changes propagated to all nodes eventually (in milliseconds).
- Queries might not return updated data immediately or might result in reading data that is not accurate (*stale reads*).
- Some NoSQL systems exhibit lost writes and other forms of data loss, may use such as write-ahead logging to avoid it.

NoSQL Databases and MongoDB

Popular Models for NoSQL Databases

- ***Key-Value Stores*** use the associative array (also known as a map or dictionary) as their fundamental data model.
  - In this model, data is represented as a collection of key-value pairs, such that each possible key appears at most once in the collection.
  - The key-value model is one of the simplest non-trivial data models, and richer data models are often implemented as an extension of it.
  - Examples include ArangoDB, InfinityDB, Oracle NoSQL Database, Redis, and dbm

NoSQL Databases and MongoDB

Popular Models for NoSQL Databases

- ***Document Stores*** are based on the notion of a "document."
  - Document-oriented database implementation differ on details. They generally assume that documents encapsulate and encode data (or information) in some standard format.
  - Encodings in use include XML, YAML, and JSON as well as binary forms like BSON. Documents are addressed in the database via a unique key that represents that document.
  - In addition to the key lookup performed by a key-value store, document-oriented databases offer an API or query language to retrieve documents based on their content.
  - Examples include MongoDB, IBM Domino, ArangoDB, and Couchbase.

NoSQL Databases and MongoDB

Popular Models for NoSQL Databases

- ***Object Databases*** represent data as objects, and replicate or modify programming language objects to make new objects within the ODBMS.
  - Integrated with the programming language, so the ODBMS and the programming language use the same representation.
  - Relational DBMS projects, by way of contrast, maintain a clearer division between the database model and the application.
  - Some object-oriented databases are designed to work well with object-oriented programming languages; others have their own languages.
  - Examples include ObjectivityDB, Versant Object DB, and GemStone/S.

NoSQL Databases and MongoDB

Introduction to MongoDB

- We will study MongoDB as an example of a non-relational or NoSQL database, and compare and contrast it with relational or SQL databases such as Derby.
- MongoDB is a cross-platform, document-oriented database that provides, high performance, high availability, and a simple model of scalability.
- MongoDB is one of the most widely-used NoSQL databases for web-based and big-data applications.
- MongoDB has official drivers for over a dozen programming languages and development environments, with unofficial or community-supported drivers for others.

NoSQL Databases and MongoDB

MongoDB Features

- Ad hoc queries
  - MongoDB supports field, range queries, regular expression searches.
  - Queries can return specific fields of documents and also include user-defined JavaScript functions.
  - Queries can also be configured to return a random sample of results of a given size.
- Indexing
  - Fields in a MongoDB document can be indexed with primary and secondary indices.

NoSQL Databases and MongoDB

MongoDB Features

- Replication
  - MongoDB provides high availability with replica sets. A replica set consists of two or more copies of the data.
  - Each replica set member may act in the role of primary or secondary replica at any time.
  - All writes and reads are done on the primary replica by default. Secondary replicas maintain a copy of the data of the primary using built-in replication.
  - When a primary replica fails, the replica set automatically conducts an election process to determine which secondary should become the primary.
  - Secondaries can optionally serve read operations, but that data is only eventually consistent by default.

NoSQL Databases and MongoDB

MongoDB Features

- Load balancing
  - MongoDB scales horizontally using sharding.
  - The user chooses a shard key, which determines how the data in a collection will be distributed.
  - The data is split into ranges based on the shard key and distributed across multiple shards. A shard is a master with one or more slaves.
  - Alternatively, the shard key can be hashed to map to a shard – enabling an even data distribution.
  - MongoDB can run over multiple servers, balancing the load or duplicating data to keep the system up and running in case of hardware failure.

NoSQL Databases and MongoDB

MongoDB Features

- File storage
  - MongoDB can be used as a file system with load balancing and data replication features over multiple machines for storing files.
  - This function, called grid file system, is included with MongoDB drivers. MongoDB exposes functions for file manipulation and content to developers.
  - GridFS is used in plugins for Nginx and lighttpd. GridFS divides a file into parts, or chunks, and stores each of those chunks as a separate document.

NoSQL Databases and MongoDB

MongoDB Features

- Aggregation
  - MapReduce can be used for batch processing of data and aggregation operations.
  - The aggregation framework enables users to obtain the kind of results for which the SQL GROUP BY clause is used.
  - Aggregation operators can be strung together to form a pipeline – analogous to Unix pipes.
  - The aggregation framework includes the $lookup operator which can join documents from multiple documents, as well as statistical operators such as standard deviation.

NoSQL Databases and MongoDB

Mongo-DB Features

- Server-side JavaScript Execution
  - JavaScript can be used in queries, *aggregation functions* (such as MapReduce), and sent directly to the database to be executed.
- Capped collections
  - MongoDB supports fixed-size collections called *capped collections*.
  - This type of collection maintains insertion order and, once the specified size has been reached, behaves like a circular queue.

NoSQL Databases and MongoDB

MongoDB *Documents*, *Collections,* and *Databases*

- Document
  - A document is a set of key-value pairs.
  - Documents have dynamic schema.
  - Documents in the same collection do not need to have the same set of fields or structure.
  - Common fields in a collection's documents may hold different types of data.

NoSQL Databases and MongoDB

MongoDB *Documents*, *Collections,* and *Databases*

- Collection
  - Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table.
  - A collection exists within a single database. Collections do not enforce a schema.
  - Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose
- Database
  - Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

NoSQL Databases and MongoDB

This table compares RDBMS terms with MongoDB.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
Column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (default key ***_id*** is provided by MongoDB itself)
Database Server and Client	
startNetworkServer /mysqld	mongod
Ij /mysql	mongo

NoSQL Databases and MongoDB

MongoDB Document Structure

- The MongoDB execution engine is JavaScript, and the document representation is *JavaScript Object Notation* (*JSON*).
  - JSON grew out of a need for stateful, real-time server-to-browser communication protocol without using browser plugins such as Flash or Java applets, the dominant methods in the early 2000s.
  - JSON was originally intended to be a subset of the JavaScript scripting language and is commonly used with Javascript, but it is a language-independent data format.
  - Code for parsing and generating JSON data is readily available in many programming languages. JSON's Web site lists JSON libraries by language.
  - JSON itself became an ECMA international standard in 2013 as the ECMA-404 standard.

NoSQL Databases and MongoDB

MongoDB Document Structure

- JSON's basic data types are:
  - **Number**: a signed decimal number not including non-numbers such as NaN. JavaScript uses a double-precision floating-point format, but in other languages JSON may encode numbers differently.
  - **String**: a sequence of zero or more Unicode characters. Strings are delimited with double-quotation marks and support a backslash escaping syntax.
  - **Boolean**: either of the values **true** or **false**.
  - **Array**: an ordered list of zero or more values, each of which may be of any type. Arrays use square bracket notation and elements are comma-separated.

NoSQL Databases and MongoDB

MongoDB Document Structure

- JSON's basic data types are:
  - **Object**: an unordered collection of name–value pairs where the names (also called keys) are strings and the values are basic data types, objects, and arrays of basic data types and objects.

    Since objects represent associative arrays, it is recommended, though not required, that each key is unique within an object.

    Objects are delimited with curly brackets and use commas to separate each pair, while within each pair the colon ':' character separates the key or name from its value.
  - **null**: An empty value, using the word null

NoSQL Databases and MongoDB

MongoDB Document Structure

- Here is a sample MongoDB document for a blog site in JSON format.

```
{
    _id: ObjectId(7df78ad8902c)
    title: Workload Isolation with Atlas and Charts',
    by: 'Tom Hollander',
    url: 'https://www.mongodb.com/blog/post/workload-isolation-with-atlas-and-charts',
    dateCreated: new Date(2019,6,5,13,59),
    tags: ['mongodb', 'database', workloads'],
    likes: 100,
    comments: [
        {
            user:'user1',
            message: 'My first comment',
            dateCreated: new Date(2019,6,7,2,15),
        },
        {
            user:'user2',
            message: 'My second comments',
            dateCreated: new Date(2019,6,8,7,45),
        }
    ]
}
```

NoSQL Databases and MongoDB

MongoDB Document Structure

- **_id** is a 12 bytes hexadecimal ObjectID which assures the uniqueness of every document.
- You can provide _id while inserting the document. If you do not provide then MongoDB provides a unique id for every document.
  - The first 4 bytes for the current timestamp,
  - The next 3 bytes for machine id,
  - The next 2 bytes for process id of MongoDB server
  - The remaining 3 bytes are simple incremental VALUE.

NoSQL Databases and MongoDB

Connect to Database Using Java API

```
import com.mongodb.MongoClient;
import com.mongodb.client.MongoDatabase;

public class ConnectToDB {
   public static void main( String args[] ) {
      // Creating a Mongo client
      MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
      System.out.println("Connected to the server successfully");

      // Accessing the database
      MongoDatabase database = mongoClient.getDatabase("myDb");
         System.out.printf("Database %s opened", database.getName());
      mongoClient.close();
   }
}
```

NoSQL Databases and MongoDB

Create a Collection Using Java API

```
import com.mongodb.MongoClient;
import com.mongodb.client.MongoDatabase;

public class CreateCollection {
   public static void main( String args[] ) {
      // Creating a Mongo client
      MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
      System.out.println("Connected to the server");

      // Accessing the database
      MongoDatabase database = mongoClient.getDatabase("myDb");
      System.out.printf("Database %s opened", database.getName());

      //Creating a collection
      database.createCollection("sampleCollection");
      System.out.println("Collection sampleCollection created");

      mongoClient.close();
   }
}
```

NoSQL Databases and MongoDB

List Collections Using Java API

```
import com.mongodb.MongoClient;
import com.mongodb.client.MongoDatabase;

public class ListOfCollections {
    public static void main( String args[] ) {
        // Creating a Mongo client
        MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
        System.out.println("Connected to the server");

        // Accessing the database
        MongoDatabase database = mongoClient.getDatabase("myDb");
        System.out.printf("Database %s opened", database.getName());

        // Accessing collections in database
        System.out.printf("Collections:\n");
        for (String name : database.listCollectionNames()) {
            System.out.printf("  %s\n", name);
        }
        mongoClient.close();
}
```

NoSQL Databases and MongoDB

Select a Collection Using Java API

```
import com.mongodb.MongoClient;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.MongoCollection;
import org.bson.Document;

public class SelectCollection {
   public static void main( String args[] ) {
      // Creating a Mongo client
      MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
      System.out.println("Connected to the server");

      // Accessing the database
      MongoDatabase database = mongoClient.getDatabase("myDb");
      System.out.printf("Database %s opened\n", database.getName());

      // Retrieving a collection
      MongoCollection<Document> collection = database.getCollection(”sampleCollection");
      System.out.println("Collection sampleCollection selected");

      mongoClient.close();
   }
}
```

NoSQL Databases and MongoDB

Insert One Document Using Java API

```
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.MongoClient;
import org.bson.Document;

public class InsertDocumentsOne {
   public static void main( String args[] ) {
      // Creating a Mongo client
      MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
      System.out.println("Connected to the server successfully");

      // Accessing the database
      MongoDatabase database = mongoClient.getDatabase("myDb");
      System.out.printf("Database %s opened\n", database.getName());
```

NoSQL Databases and MongoDB

Insert One Document Using Java API

```
        // Retrieving a collection
        MongoCollection<Document> collection = database.getCollection("sampleCollection");
        System.out.println("Collection sampleCollection selected");

        Document document = new Document()
                .append("title", "MongoDB Architecture”)
                .append("id", 1)
                .append("description", "database")
                .append("likes", 100)
                .append("url", "https://www.mongodb.com/mongodb-architecture")
                .append("by", ”MongoDB");
        collection.insertOne(document);
        System.out.println("Document inserted");

        mongoClient.close();
    }
}
```

NoSQL Databases and MongoDB

Update One Document Using Java API

```
import com.mongodb.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;
import com.mongodb.client.result.UpdateResult;
import org.bson.Document;

public class UpdateDocumentsOne {

  public static void main( String args[] ) {
    // Creating a Mongo client
    MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
    System.out.println("Connected to the server");
```

NoSQL Databases and MongoDB

Update One Document Using Java API

```
        // Accessing the database
        MongoDatabase database = mongoClient.getDatabase("myDb");
        System.out.printf("Database %s opened\n", database.getName());

        // Retrieving a collection
        MongoCollection<Document> collection = database.getCollection("sampleCollection");
        System.out.println("Collection myCollection selected”);

        UpdateResult result = collection.updateOne(Filters.eq("id", 1), Updates.set("likes", 150));
        System.out.printf("Documents updated: %d\n”, result.getModifiedCount());

        mongoClient.close();
    }
}
```

Using the MongoDB Java API

Delete One Document Using Java API

```
import com.mongodb.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Filters;
import com.mongodb.client.result.DeleteResult;
import org.bson.Document;

public class DeleteDocumentsOne {

  public static void main( String args[] ) {
    // Creating a Mongo client
    MongoClient mongoClient = new MongoClient( "localhost" , 27017 );
    System.out.println("Connected to the server successfully");
```

Using the MongoDB Java API

Delete One Document Using Java API

```
        // Accessing the database
        MongoDatabase database = mongoClient.getDatabase("myDb");
        System.out.printf("Database %s opened\n", database.getName());

        // Retrieving a collection
        MongoCollection<Document> collection = database.getCollection("sampleCollection");
        System.out.println("Collection sampleCollection selected");

        // Deleting the documents
        DeleteResult result = collection.deleteOne(Filters.eq("id", 1));
        System.out.printf("Documents deleted: %d\n", result.getDeletedCount());

        mongoClient.close();
    }
}
```