Java Programming – NoSQL and Nitrite DB

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

- Nitrite NoSQL Database

Today's Lecture

NoSQL Database

- A NoSQL DB does not have tables like a relational DB.
- Can be unstructured (no schemas defining the structure of the data).
- You do not use SQL to query the database.
- SQL DBs are generally normalized while NoSQL DBs are not.
- Relational DB Normalization
  - Removes redundancy (no update, delete, or insert anomalies).
  - Must perform joins to get related data.
  - Joins are an expensive operation on a relational DB.
- NoSQL DBs generally do not need to do join-like operations.

NoSQL Database Software

- MongoDB (local)
- MongoDB Atlas (cloud-based)
- Google Firestore and Google Realtime DB (cloud-based)
- Cassandra (local or cloud-based)
- Amazon Dynamo DB (cloud-based)
- Nitrite (local) ← **We will be using this one**
- There are many others…

Nitrite

- Nitrite is an embedded NoSQL database.
- Why Nitrite?
- NoSQL Object → NO_2 (the formula for nitrite).
- Embedded means there is no server process for the database.
- The application that uses the database imports a library that contains the functionality to manipulate the DB.

- Nitrite is document-based (similar to MongoDB).

- Link for using Nitrite:

https://www.dizitart.org/nitrite-database.html

- A document contains name-value pairs.
- It is similar to JSON in that it can contain any Java objects.
- The value in a name-value pair can be a collection of values (like JSON).
- Here are two documents:

Document

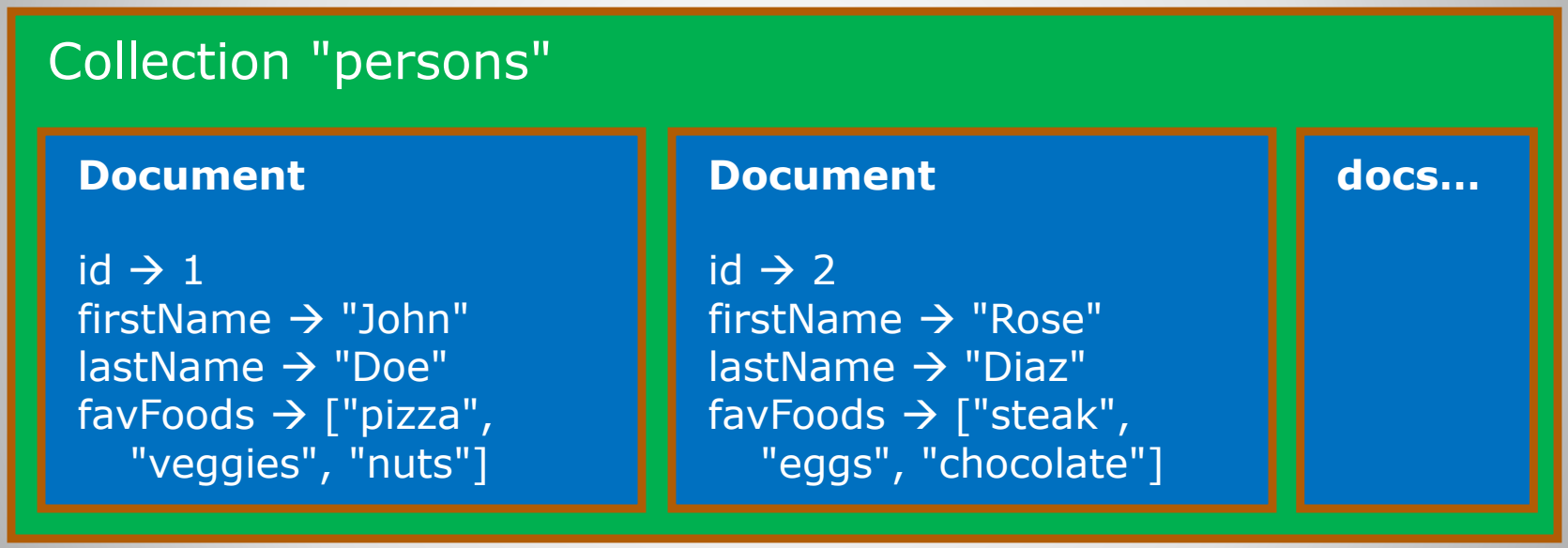
id → 1
firstName → "John"
lastName → "Doe"
Address → "10 Broadway"
favFoods → ["pizza",
"veggies", "nuts"]

Document

id → 2
firstName → "Rose"
lastName → "Diaz"
Address → "5 Maple St"
favFoods → ["steak",
"eggs", "chocolate"]

Nitrite Document

- A Nitrite collection contains documents.
- Nitrite collections have a unique name. The collection below is named "persons".



Nitrite Collection

Relational vs NoSQL

- Relational DB
  - Store data in tables
  - Try to eliminate redundancy of data using normalization
  - Use joins to retrieve related data from different tables
- NoSQL DB
  - Uses some type of document to store data (specifics differ depending on the NoSQL DB)
  - Use key-value pairs (in general)
  - Keep collections of documents
  - Can have redundant data

- Add the following dependency to Maven to use Nitrite:

```
<!-- https://mvnrepository.com/artifact/org.dizitart/nitrite -->
<dependency>
    <groupId>org.dizitart</groupId>
    <artifactId>nitrite</artifactId>
    <version>3.4.4</version>
</dependency>
```

Maven Nitrite Dependency

Open or Create Database

- Use the Nitrite class to create or open a database.

```
// Create DB in default directory
Nitrite db = Nitrite.builder()
    .filePath("./hello.db")
    .openOrCreate();

// Create DB in another directory
Nitrite db = Nitrite.builder()
    .filePath("/tmp/hello.db")
    .openOrCreate();
```

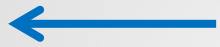
The name of the database is hello. It is created in the current working directory (default directory)

The name of the database is hello. The database will be created in the tmp directory. The tmp must exist for this to work otherwise an exception is thrown

Close Database

- You should close the database when you are done using it.
- For example (assumes db is declared with Nitrite as the data type):

```
db.close();
```



- NitriteCollection - Kind of similar to a table in a relationalDB.
- Document – Data for one entity in the database (like one object).

Create collection using getCollection (db is an instance of Nitrite)

```
NitriteCollection collection = db.getCollection("persons");
```

persons is the name of the collection

```
Document doc = createDocument("id", 1)
  .put("firstName", "John")
  .put("lastName", "Doe");
```

Create document using createDocument (new is called inside createDocument)

```
// insert a document into the collection
collection.insert(doc);
```

Insert the document into the collection

Add Data to a NitriteCollection

Query the DB (all documents)

- Use **find** method to query the Nitrite DB.
- Find returns documents from a collection.
- The documents are returned in a Cursor.
- You can write code to iterate over the Cursor results.
- Cursor import. Make sure you import the following for the Cursor: import org.dizitart.no2.Cursor;

Find returns a Cursor

```
Cursor results = collection.find();

for (Document currDoc : results) {
    System.out.println(currDoc.toString());
}
```

Iterate overall results

- Use find with a filter to retrieve records by a given criteria.

Use a Filters class to set the criteria of documents to retrieve

```
Cursor results = collection.find(Filters.eq("firstName", "Jane"));

for (Document currDoc : results) {
        System.out.println(currDoc.toString());;
}
```

Query the DB (using a criteria)

Removing Documents from a Collection

- Use **remove** to delete documents from a collection.
- You can specify documents to remove using specific criteria (remove all records that have a certain value).
- You can remove all documents from the collection.

Removes all documents with "John" as first name from the collection

```
collection.remove(eq("firstName", "John"));
```

Removes all documents

```
collection.remove(Filters.ALL);
```

- End of Slides

End of Slides