WiSe03 - NoSQL Datenbankmanagementsysteme

Art	GK
Status	Bereit zur Abgabe
Fach	INSY
Ende	@23. Januar 2025
Erstellt	@13. Januar 2025 14:24
Anfang	@6. Februar 2025
Fertig	@14. Februar 2025 10:32
Formula	Die Deadline ist schon vorbei
Link	https://elearning.tgm.ac.at/mod/assign/view.php?id=129645
Raum	H129

Allgemein:

In der vorliegenden Präsentation findet ihr ein Design, sowie einen Überblick zu den Arten von NoSQL Datenbanksystemen.

Eure Aufgabe ist es nun allein, oder in Teams a maximal 3 Personen, die Arten von NoSQL Datenbanken näher zu beleuchten.

Recherchiert entsprechend im WWW alle grundlegenden Informationen zum jeweiligen Datenbanktyp (Aufbau, Anwendungszweck und Anwendungsbeispiele) und versucht zu **zwei der unterhalb gegebenen Datenbanktypen** ein praktisches Beispiel zu entwickeln, wo ihr entweder die NoSQL Datenbank lokal auf eurem Server o.ä. aufsetzt oder Onlinebeispiele durcharbeitet und dort versucht grundlegende Funktionalitäten aufzuzeigen.

2 Datenbanktypen aus den folgenden auswählen:

- Key-Value-Datenbanken
- Document based
- Spaltenorientierte Datenbanken
- Multi-Value-Datenbanken
- Multi-Modell-Datenbank
- Wide Table Datenbanken

Ergebnisse Eurer Arbeit:

- Stellt zu ALLEN oben angeführten Typen einen Beispieldatenbank vor und bringt vielleicht ein kurzes Beispiel von der Herstellerseite.
- Erstellt eine kurze Ausarbeitung mit Praxisbeispielen der beiden Datenbanksysteme die ihr installiert habt. Dort stellt eure "Forschungsergebnisse" sowie wie oben beschrieben, den Aufbau, den Anwendungszweck sowie Anwendungsbeispiele dar.

Art der Abgabe:

Kurzes Abgabegespräch in dem ihr die Präsentation, sowie die Ausarbeitung vorstellt und erklärt und eure Praxisrealisierungen kurz zeigt.

Bewertung:

Nicht erfüllt - GK überwiegend - GK vollständig

Ausarbeitung der einzelnen Datentypen

▼ Key-Value-Datenbanken

- einfache Key Value Methode zum speichern der Daten
- gut für sehr simple abfragen
- Primary Key('s) - Attribute
- Beispiel: Redis

Insert Movies#

It is time now to add some data into your database, let's insert a few movies, using `redis-cli` or RedisInsight.

Once you are connected to your Redis instance run the following commands:

```
> HSET movie:11002 title "Star Wars: Episode V - The Empire Strikes Back" plot "After the Rebels are brutally overp
> HSET movie:11003 title "The Godfather" plot "The aging patriarch of an organized crime dynasty transfers contro
> HSET movie:11004 title "Heat" plot "A group of professional bank robbers start to feel the heat from police when
> HSET "movie:11005" title "Star Wars: Episode VI - Return of the Jedi" genre "Action" votes 906260 rating 8.3 rele
```

Now it is possible to get information from the hash using the movie ID. For example if you want to get the title, and rating execute the following command:

```
> HMGET movie:11002 title rating

1) "Star Wars: Episode V - The Empire Strikes Back"
2) "8.7"
```

And you can increment the rating of this movie using:

```
> HINCRBYFLOAT movie:11002 rating 0.1
"8.8"
```

But how do you get a movie or list of movies by year of release, rating or title?

One option, would be to read all the movies, check all fields and then return only matching movies; no need to say that this is a really bad idea.

Nevertheless this is where Redis developers often create custom secondary indexes using SET/SORTED SET structures that point back to the movie hash. This needs some heavy design and implementation.

This is where Search and Query in Redis Stack can help, and why it was created.

Quelle: https://redis.io/learn/howtos/moviesdatabase

▼ Document based (document-oriented)

- Documents speichern Daten in field-value pairs
  - kann in Formaten wie JSON, BSON, XML gespeichert werden
- Die Werte können alles sein Strings, [...], arrays, objects
- Beispiel: MongoDB

What Sample Datasets are Available for MongoDB?

There are many datasets to choose from if you are looking for sample or testing data in JSON format for your MongoDB database. But if you have an Atlas account, you don't have to look very far for the data you need. Atlas has *eight* available sample datasets and makes it easy to load that sample data directly into your cluster. With these datasets, you can test your application or skills at writing aggregation queries.

Here are the sample datasets that Atlas provides:

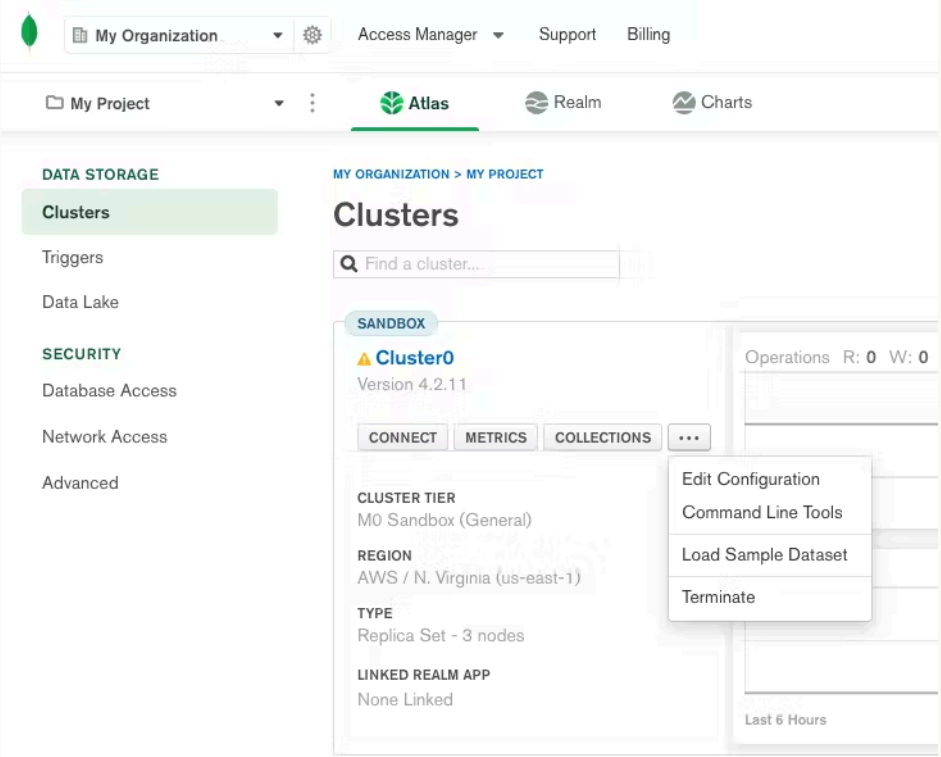
- **AirBnB listings**: This is a randomized compilation of AirBnB listings in one MongoDB collection with documents that include fields like listing_url, address, and name. This collection is from the Inside AirBnB website.
- **Analytics data**: This database contains three collections: accounts, customers, and transactions. It represents data from a typical financial services company.
- **Geospatial data**: This is one collection of data that contains shipwreck data. A single document in this dataset represents an actual shipwreck and contains latitude and longitude data.
- **Movie data**: This database contains five collections: comments, movies, sessions, theaters, and users. It represents the data from a social movie review site.
- **Restaurant data**: This sample dataset contains two collections: restaurants and neighborhoods. It is designed to help users become familiar with GeoJSON data.
- **Supply store data**: This database consists of a single collection called sales. Each document represents a sale from a supply store.
- **Training data**: This is a large dataset used in MongoDB training courses. It contains Crunchbase data, official New York City data, flight data, and Citybike data. It contains seven collections: companies, grades, inspections, posts, routes, trips, and zips.
- **Weather data**: This consists of a single MongoDB collection called data. Each document in this collection represents a weather report.

How to Load Sample Data into Your MongoDB Atlas Cluster

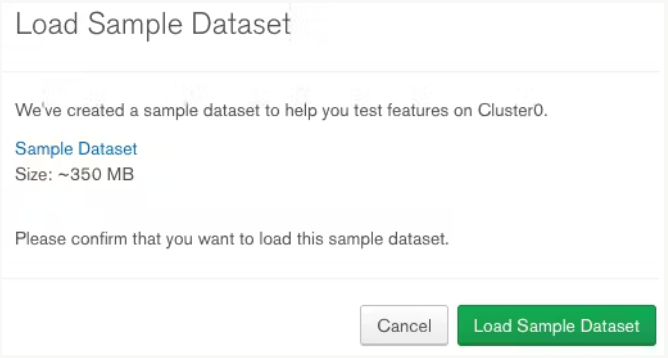
There are two ways to load your sample data into your Atlas Cluster. You can either use the Clusters view or the Data Explorer tutorial below.

Option 1: Loading Sample Data from the Clusters View

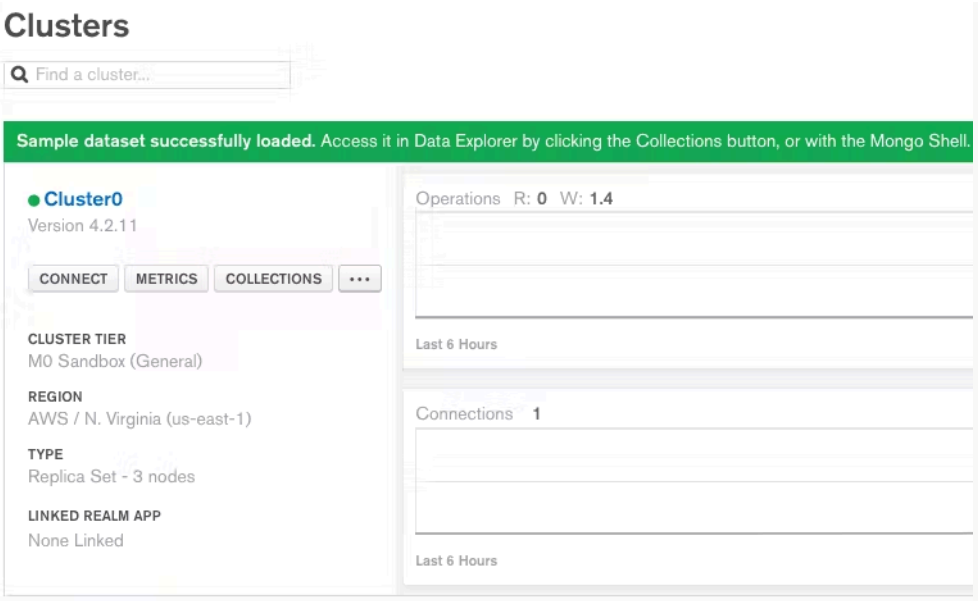
Log into Atlas. The default view is the Clusters view. Click on the ellipses button in your cluster and then click on Load Sample Dataset.



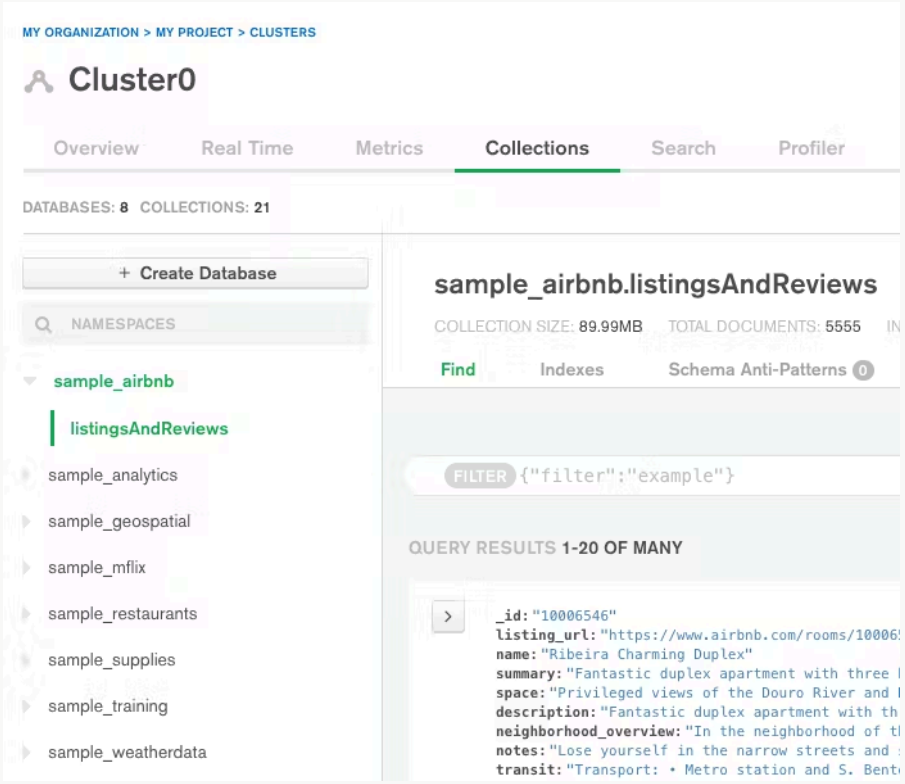
A menu will then pop up to check if you are sure. Click the Load Sample Dataset button.



It will take about a minute for the sample dataset to deploy. Once it does, you will see a message like this:

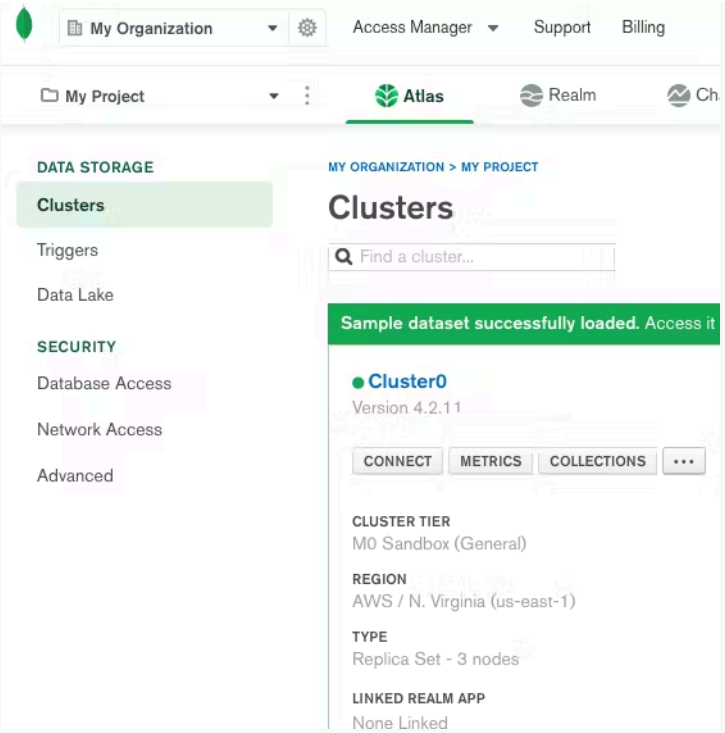


This loaded every one of the Atlas sample databases into your cluster. Clicking on the Collections buttons like the message suggests will take you to them.

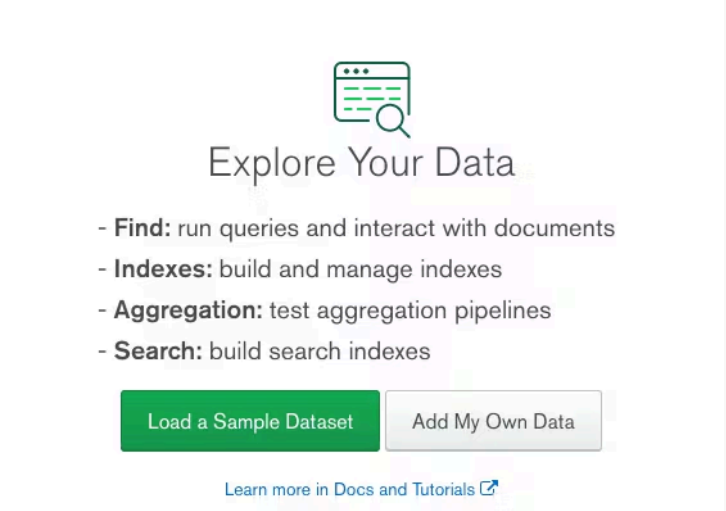


Option 2: Loading Sample Data from the Data Explorer

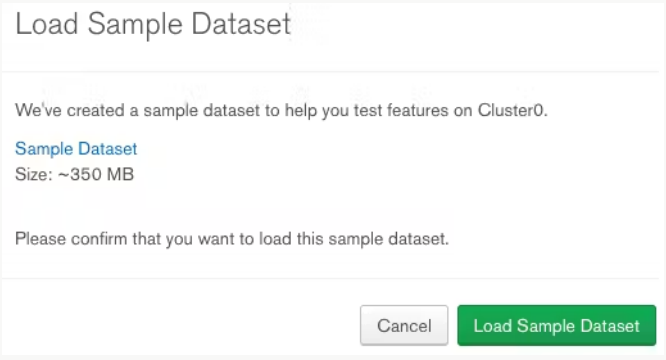
The other way to load these sample datasets into your cluster is to go to the Clusters view in Atlas and instead of clicking the ellipses button, click on the Collections button.



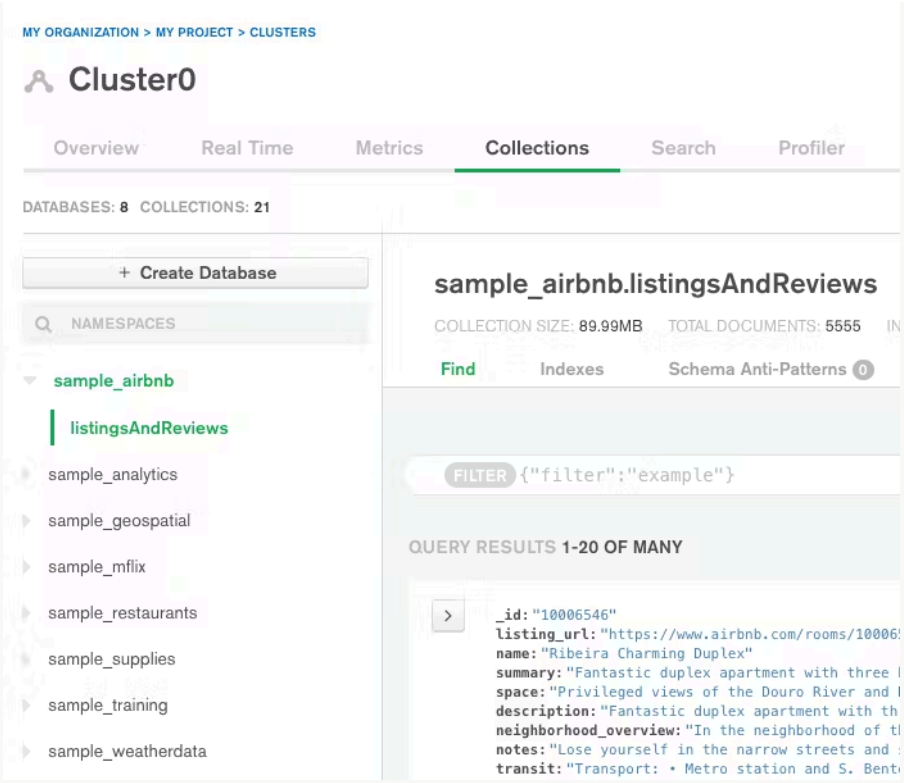
If you have no collections, you will see a Load a Sample Dataset button:



Clicking that button will bring up the same modal. Click the Load Sample Dataset button:



This will take about a minute. Once it is done, you can browse through all of your new collections.



Conclusion

Samples databases are a straightforward concept, but important when building and testing your apps. MongoDB makes it simple to create sample databases with eight different data sets easily imported into MongoDB Atlas.

FAQs

Does MongoDB Provide Sample Datasets?

MongoDB Atlas provides sample datasets that you can load into your own database cluster for testing your application. These available sample datasets include analytics, movie, geospatial, sales, and weather data. You can load this data into your cluster in only a few clicks.

Quelle: https://www.mongodb.com/resources/basics/databases/sample-database

▼ Spaltenorientierte Datenbanken

- Speichert Daten in Spalten denen Speichertypen zugewiesen werden so hat jede Spalte immer den gleichen typen
- gut für große Datenmengen
- Beispiel: MySQL

The Employees database is available from Employees DB on GitHub. You can download a prepackaged archive of the data, or access the information through Git.

To use the Zip archive package, download the archive and unpack it using WinZip or another tool that can read .zip files, then change location into the unpacked package directory. For example, using unzip, execute these commands:

```
$> unzip test_db-master.zip
$> cd test_db-master/
```

The Employees database is compatible with several different storage engines, with the InnoDB engine enabled by default. Edit the employees.sql file and adjust the comments to choose a different storage engine:

```
set storage_engine = InnoDB;
-- set storage_engine = MyISAM;
-- set storage_engine = Falcon;
-- set storage_engine = PBXT;
-- set storage_engine = Maria;
```

To import the data into your MySQL instance, load the data through the mysql command-line tool:

```
$> mysql -t < employees.sql
+-----------------------------+
| INFO                        |
+-----------------------------+
| CREATING DATABASE STRUCTURE |
+-----------------------------+
+------------------------+
| INFO                   |
+------------------------+
| storage engine: InnoDB |
+------------------------+
+--------------------+
| INFO               |
+--------------------+
| LOADING departments |
+--------------------+
+-------------------+
| INFO              |
+-------------------+
| LOADING employees |
+-------------------+
+------------------+
| INFO             |
+------------------+
| LOADING dept_emp |
+------------------+
+----------------------+
| INFO                 |
+----------------------+
| LOADING dept_manager |
+----------------------+
+----------------+
| INFO           |
+----------------+
| LOADING titles |
+----------------+
+------------------+
| INFO             |
+------------------+
```

```
| LOADING salaries |
+------------------+
To test that the data you have loaded matches the expected results, run the test suite. For more information, see Se
```

Quelle: https://dev.mysql.com/doc/employee/en/employees-installation.html

▼ Multi-Value-Datenbanken

- Das speichern von mehreren Werten pro Key
- Bei komplexen oder hierarchischen Strukturen
- Beispiel: Rocket U2

You can install the Circus database as either an account of UniVerse files or as a schema of SQL tables (or both). The two versions of the database are distinguished by a suffix in the file name:

- .F identifies the UniVerse file version.
- .T identifies the SQL table version.

Thus, INVENTORY.F is the UniVerse file version of the inventory data, and INVENTORY.T is the SQL table version of the same data.

Install the version of the files that you prefer. Examples in this manual use the UniVerse SQL table version. Keep in mind that you can issue SQL statements against UniVerse files, and you can issue RetrieVe commands against SQL tables. However, the results may vary slightly, depending on which you use.

Use the following UniVerse commands to generate and remove the Circus database:

Command	Action
SETUP.DEMO.SCHEMA username	Registers username as an SQL user (if not one already) and makes the current UniVerse account into a schema called DEMO_username, which is owned by username. Only an SQL user who is a DBA (database administrator) can run this command.
MAKE.DEMO.TABLES	Creates and loads the Circus database tables into the current account, making the current user the owner of the tables. The user must be a registered SQL user, the account must be an SQL schema, and the tables must not already exist in this schema. Resultant tables have a .T suffix.
REMOVE.DEMO.TABLES	Deletes the Circus database tables from the current schema. The user must be a registered SQL user who is either the owner of the tables or a DBA.
MAKE.DEMO.FILES	Creates and loads the Circus database files into the current account. The files must not already exist in this account. The file names will have an .F suffix, and the contents of the files match those of the corresponding .T tables.
REMOVE.DEMO.FILES	Deletes the Circus database files from the current account.

To install the SQL table version of the Circus database on your system:

1. 1. Create a directory to contain the Circus database and set it up as a UniVerse account.
2. 2. If you are a registered SQL user with RESOURCE privilege, log on to the account and make the account into a schema by entering:

   ```
   > CREATE SCHEMA schemaname;
   ```

   You can use any unique schemaname. You are the owner of the new schema.

   If you are not a registered user with RESOURCE privilege, have your database administrator (DBA) log in to your account and enter:

   ```
   > SETUP.DEMO.SCHEMA username
   ```

   username is your operating system user name. This command registers you as an SQL user, makes the directory into a schema called DEMO_username, and sets you up as the schema's owner.
3. 3. To create the tables and load data into them, enter:

   ```
   MAKE.DEMO.TABLES
   ```

   The table names all have the .T suffix. You are the owner of the tables.

Quelle: https://docs.rocketsoftware.com/bundle/universesqluser_ug_1411/page/jwi1687885291149.html

▼ Multi-Modell-Datenbank

- Erlaubt es mehrere DB-Modelle in einem System speichern zu lassen zB. Dokumente und Graphen
- Beispiel: ArangoDB

IMDB Movie Dataset

This is a movies and actors dataset based on data of the **Internet Movie Database** (IMDB). It was converted into a graph. Also see the **arangodb/example-datasets** repository.

1. Download **imdb_graph_dump_rev2.zip** (6.45 MB)
2. Unpack the downloaded archive
3. Restore the folder `dump` with **arangorestore** into an ArangoDB instance, e.g. `arangorestore --server.endpoint tcp://localhost:8529 --server.database IMDB --create-database --include-system-collections --input-directory dump`
4. Create a View called `imdb` in the IMDB database. You can find various View configuration examples in this chapter.

Quelle: https://docs.arangodb.com/3.12/index-and-search/arangosearch/example-datasets/

▼ Wide Table Datenbanken

optimiert für große Tabellen

gut für große unstrukturierte Datenmengen

Beispiel: Google Bigtable

Beispiele für Lesevorgänge

bookmark_border

Cloud Bigtable-Clientbibliotheken bieten die Möglichkeit, Daten aus einer Tabelle zu lesen. Auf dieser Seite finden Sie Beispiele für die einzelnen Typen von Leseanfragen in mehreren Sprachen. Eine Übersicht finden Sie unter Lesevorgänge.

Die Python-Clientbibliothek für Bigtable bietet zwei APIs: `asyncio` und eine synchrone API. Wenn Ihre Anwendung asynchron ist, verwenden Sie `asyncio`.

Erforderliche Rollen

Bitten Sie Ihren Administrator, Ihnen für die Tabelle die IAM-Rolle **Bigtable Reader** (`roles/bigtable.reader`) zuzuweisen, damit Sie die Berechtigungen erhalten, die Sie zum Lesen von Daten aus einer Tabelle benötigen. Weitere Informationen zum Zuweisen von Rollen finden Sie unter Zugriff auf Projekte, Ordner und Organisationen verwalten.

Sie können die erforderlichen Berechtigungen auch über benutzerdefinierte Rollen oder andere vordefinierte Rollen erhalten.

Einzelne Zeile lesen

Die folgenden Codebeispiele zeigen, wie Sie eine **einzelne Datenzeile** mit dem Zeilenschlüssel abrufen.

GoHBaseJavaPython asyncioPythonC#C++Node.jsPHPRuby

Informationen zum Installieren und Verwenden der Clientbibliothek für Bigtable finden Sie unter Bigtable-Clientbibliotheken.

Richten Sie zur Authentifizierung bei Bigtable die Standardanmeldedaten für Anwendungen ein. Weitere Informationen finden Sie unter Authentifizierung für Clientbibliotheken einrichten.

```
const rowkey = 'phone#4c410523#20190501';

const [row] = await table.row(rowkey).get();
printRow(rowkey, row.data);
```

Teilzeile lesen

Die folgenden Codebeispiele zeigen, wie Sie **bestimmte Spalten** aus einer Zeile abrufen.

GoHBaseJavaPython asyncioPythonC#C++Node.jsPHPRuby

Informationen zum Installieren und Verwenden der Clientbibliothek für Bigtable finden Sie unter Bigtable-Clientbibliotheken.

Richten Sie zur Authentifizierung bei Bigtable die Standardanmeldedaten für Anwendungen ein. Weitere Informationen finden Sie unter Authentifizierung für Clientbibliotheken einrichten.

```
const COLUMN_FAMILY = 'stats_summary';
const COLUMN_QUALIFIER = 'os_build';
const rowkey = 'phone#4c410523#20190501';

const [row] = await table
  .row(rowkey)
  .get([`${COLUMN_FAMILY}:${COLUMN_QUALIFIER}`]);

printRow(rowkey, row);
```

Mehrere Zeilen lesen

Die folgenden Codebeispiele zeigen, wie Sie **mehrere Datenzeilen** mit einer Gruppe von Zeilenschlüsseln abrufen.

GoHBaseJavaPython asyncioPythonC#C++Node.jsPHPRuby

Informationen zum Installieren und Verwenden der Clientbibliothek für Bigtable finden Sie unter Bigtable-Clientbibliotheken.

Richten Sie zur Authentifizierung bei Bigtable die Standardanmeldedaten für Anwendungen ein. Weitere Informationen finden Sie unter Authentifizierung für Clientbibliotheken einrichten.

```
const rowKeys = ['phone#4c410523#20190501', 'phone#4c410523#20190502'];
const [rows] = await table.getRows({keys: rowKeys});
rows.forEach(row => printRow(row.id, row.data));
```

Bereich von Zeilen lesen

Die folgenden Codebeispiele zeigen, wie Sie **mehrere Datenzeilen** mit einem **Start- und einem Endschlüssel** abrufen.

GoHBaseJavaPython asyncioPythonC#C++Node.jsPHPRuby

Informationen zum Installieren und Verwenden der Clientbibliothek für Bigtable finden Sie unter Bigtable-Clientbibliotheken.

Richten Sie zur Authentifizierung bei Bigtable die Standardanmeldedaten für Anwendungen ein. Weitere Informationen finden Sie unter Authentifizierung für Clientbibliotheken einrichten.

```
const start = 'phone#4c410523#20190501';
const end = 'phone#4c410523#201906201';

await table
  .createReadStream({
    start,
    end,
  })
  .on('error', err => {
    // Handle the error.
    console.log(err);
  })
  .on('data', row => printRow(row.id, row.data))
  .on('end', () => {
    // All rows retrieved.
  });
```

Mehrere Bereiche von Zeilen lesen

Die folgenden Codebeispiele zeigen, wie Sie **mehrere Datenzeilen** mit **mehreren Start- und Endschlüsseln** abrufen. Beachten Sie, dass sich das Anfordern einer großen Anzahl von Zeilenbereichen in einer einzelnen Anfrage auf die Leseleistung auswirken kann.

GoHBaseJavaPythonC#C++Node.jsPHPRuby

Informationen zum Installieren und Verwenden der Clientbibliothek für Bigtable finden Sie unter Bigtable-Clientbibliotheken.

Richten Sie zur Authentifizierung bei Bigtable die Standardanmeldedaten für Anwendungen ein. Weitere Informationen finden Sie unter Authentifizierung für Clientbibliotheken einrichten.

```
await table
  .createReadStream({
    ranges: [
      {
        start: 'phone#4c410523#20190501',
        end: 'phone#4c410523#20190601',
      },
      {
        start: 'phone#5c10102#20190501',
```

```
      end: 'phone#5c10102#20190601',
    },
  ],
})
.on('error', err => {
  // Handle the error.
  console.log(err);
})
.on('data', row => printRow(row.id, row.data))
.on('end', () => {
  // All rows retrieved.
});
```

Mehrere Zeilen mithilfe eines Zeilenschlüsselpräfixes lesen

Die folgenden Codebeispiele zeigen, wie Sie **mehrere Datenzeilen** mit einem **Zeilenschlüsselpräfix** abrufen.

GoHBaseJavaPython asyncioPythonC#C++Node.jsPHPRuby

Informationen zum Installieren und Verwenden der Clientbibliothek für Bigtable finden Sie unter Bigtable-Clientbibliotheken.

Richten Sie zur Authentifizierung bei Bigtable die Standardanmeldedaten für Anwendungen ein. Weitere Informationen finden Sie unter Authentifizierung für Clientbibliotheken einrichten.

```
const prefix = 'phone#';

await table
  .createReadStream({
    prefix,
  })
  .on('error', err => {
    // Handle the error.
    console.log(err);
  })
  .on('data', row => printRow(row.id, row.data))
  .on('end', () => {
    // All rows retrieved.
  });
```

In umgekehrter Reihenfolge scannen

Die folgenden Codebeispiele zeigen, wie Sie rückwärts scannen. Weitere Informationen finden Sie unter Rückwärtssuche. Begrenzen Sie einen umgekehrten Scan auf etwa 50 Zeilen, um Ineffizienzen zu vermeiden.

GoHBaseJavaC++

Informationen zum Installieren und Verwenden der Clientbibliothek für Bigtable finden Sie unter Bigtable-Clientbibliotheken.

Richten Sie zur Authentifizierung bei Bigtable die Standardanmeldedaten für Anwendungen ein. Weitere Informationen finden Sie unter Authentifizierung für Clientbibliotheken einrichten.

```
namespace cbt = ::google::cloud::bigtable;
using ::google::cloud::Options;
using ::google::cloud::StatusOr;
[](cbt::Table table) {
  // Read and print the rows.
  auto reader = table.ReadRows(
      cbt::RowRange::RightOpen("phone#5c10102", "phone#5c10103"), 3,
      cbt::Filter::PassAllFilter(),
      Options{}.set<cbt::ReverseScanOption>(true));
  for (StatusOr<cbt::Row>& row : reader) {
```

```
    if (!row) throw std::move(row).status();
    PrintRow(*row);
  }
}
```

Mit Filtern lesen

Die folgenden Codebeispiele zeigen, wie Sie **mehrere Datenzeilen** mit einem **Zeilenfilter** abrufen. Weitere Informationen zu den Filtertypen, die Sie in Leseanfragen verwenden können, finden Sie der Filterübersicht. Weitere Codebeispiele, die das Implementieren verschiedener Filtertypen in mehreren Sprachen zeigen, sind ebenfalls verfügbar.

Hinweis: Der HBase-Client für Java verwendet Filter APIs von HBase anstelle der Bigtable Data API. Informationen zu Filter APIs von HBase finden Sie in der HBase-Dokumentation.

GoHBaseJavaPython asyncioPythonC#C++Node.jsPHPRuby

Informationen zum Installieren und Verwenden der Clientbibliothek für Bigtable finden Sie unter Bigtable-Clientbibliotheken.

Richten Sie zur Authentifizierung bei Bigtable die Standardanmeldedaten für Anwendungen ein. Weitere Informationen finden Sie unter Authentifizierung für Clientbibliotheken einrichten.

```
const filter = {
  value: /PQ2A.*$/,
};

await table
  .createReadStream({
    filter,
  })
  .on('error', err => {
    // Handle the error.
    console.log(err);
  })
  .on('data', row => printRow(row.id, row.data))
  .on('end', () => {
    // All rows retrieved.
  });
```

Aus einer autorisierten Ansicht lesen

In den folgenden Beispielen wird gezeigt, wie eine Leseanfrage an eine autorisierte Ansicht gesendet wird. Die Syntax ähnelt dem Lesen aus einer Tabelle, mit der Ausnahme, dass Sie auch die ID der autorisierten Ansicht angeben müssen.

Java

In diesem Beispiel wird gezeigt, wie eine einzelne Zeile aus einer autorisierten Ansicht gelesen wird.

Informationen zum Installieren und Verwenden der Clientbibliothek für Bigtable finden Sie unter Bigtable-Clientbibliotheken.

Richten Sie zur Authentifizierung bei Bigtable die Standardanmeldedaten für Anwendungen ein. Weitere Informationen finden Sie unter Authentifizierung für Clientbibliotheken einrichten.

```
try {
  System.out.println("\nReading a single row by row key from an authorized view");
  Row row =
      dataClient.readRow(AuthorizedViewId.of(tableId, authorizedViewId), ROW_KEY_PREFIX + 0);
  System.out.println("Row: " + row.getKey().toStringUtf8());
  for (RowCell cell : row.getCells()) {
    System.out.printf(
        "Family: %s    Qualifier: %s    Value: %s%n",
        cell.getFamily(), cell.getQualifier().toStringUtf8(), cell.getValue().toStringUtf8());
```

```
    }
    return row;
  } catch (NotFoundException e) {
    System.err.println("Failed to read from a non-existent authorized view: " + e.getMessage());
    return null;
  }
```

In diesem Beispiel wird gezeigt, wie Daten aus einer autorisierten Datenansicht mit einem Filter gelesen werden.

Informationen zum Installieren und Verwenden der Clientbibliothek für Bigtable finden Sie unter Bigtable-Clientbibliotheken.

Richten Sie zur Authentifizierung bei Bigtable die Standardanmeldedaten für Anwendungen ein. Weitere Informationen finden Sie unter Authentifizierung für Clientbibliotheken einrichten.

```
try {
  // A filter that matches only the most recent cell within each column
  Filter filter = FILTERS.limit().cellsPerColumn(1);
  System.out.println("\nScanning authorized view with filter");
  Query query = Query.create(AuthorizedViewId.of(tableId, authorizedViewId)).filter(filter);
  ServerStream<Row> rowStream = dataClient.readRows(query);
  List<Row> authorizedViewRows = new ArrayList<>();
  for (Row r : rowStream) {
    System.out.println("Row Key: " + r.getKey().toStringUtf8());
    authorizedViewRows.add(r);
    for (RowCell cell : r.getCells()) {
      System.out.printf(
          "Family: %s    Qualifier: %s    Value: %s%n",
          cell.getFamily(), cell.getQualifier().toStringUtf8(), cell.getValue().toStringUtf8());
    }
  }
  return authorizedViewRows;
} catch (NotFoundException e) {
  System.err.println("Failed to read a non-existent authorized view: " + e.getMessage());
  return null;
}
```

Quelle: https://cloud.google.com/bigtable/docs/reading-data?hl=de#node.js

Anwendungsbeispiele:

Key-Value DB

> 💡 Hierfür nutze ich die oben genannte Redis.
> Als Sprache nutze ich NodeJS. Alles wird mit Docker realisiert.

Wie immer werde ich erstmal eine `docker-compose.yml` erstellen:

```
services:
  database:
    image: redis:latest
    ports:
      - "6379:6379"
    volumes:
      - database:/data

  app:
```

```
    build: ./app
    volumes:
      - ./app:/app
      - app:/app/node_modules
    depends_on:
      - database
    environment:
      DATABASE_URL: redis://database:6379/
volumes:
  app:
  database:
```

Ich habe einen `database` Container mit der Datenbank erstellt und gleich das Beispiel, welches ich im `app/` Ordner erstellt habe, als `app` Container in die docker-compose aufgenommen.

Daraufhin habe ich im `app/` ein `node.js` Projekt erstellt:

```
npm init
npm i redis dotenv
```

Eine `index.js` mit dem Beispiel abfragen erstellt:

```
const redis = require('redis');
require('dotenv').config();

const client = redis.createClient({ url: process.env.DATABASE_URL });

client.on('connect', () => {
  console.log('Connected to Redis!');
});

client.on('error', (err) => {
  console.error('Error connecting to Redis:', err);
});

(async () => {
  await client.connect();

  await client.set('user:1', 'John Doe');
  console.log('Stored user:1:', await client.get('user:1'));

  await client.setEx('session:12345', 3600, 'active');
  console.log('Stored session:12345:', await client.get('session:12345'));

  await client.disconnect();
})();
```

Schlussendlich rufe ich das Projekt mit einem kräftigen `docker-compose up` ins leben und erhalte von dem app Container folgende Konsolenausgabe:

```
app-1      |
app-1      | > @ start /app
app-1      | > node ./index.js
app-1      |
app-1      | Connected to Redis!
app-1      | Stored user:1: John Doe
app-1      | Stored session:12345: active
app-1 exited with code 0
```

Key-Value DB

Hierfür nutze ich die oben genannte MongoDB.
Als Sprache nutze ich NodeJS. Alles wird mit Docker realisiert.

Wie immer werde ich erstmal eine `docker-compose.yml` erstellen:

```
services:
  database:
    image: mongo:latest
    ports:
      - "27017:27017"
    volumes:
      - database:/data/db

  app:
    build: ./app
    volumes:
      - app:/app
    depends_on:
      - database
    environment:
      DATABASE_URL: mongodb://database:27017/
volumes:
  app:
  database:
```

Ich habe einen `database` Container mit der Datenbank erstellt und gleich das Beispiel, welches ich im `app/` Ordner erstellt habe, als `app` Container in die docker-compose aufgenommen.

Daraufhin habe ich im `app/` ein `node.js` Projekt erstellt:

```
npm init
npm i mongodb dotenv
```

Eine `index.js` mit dem Beispiel abfragen erstellt:

```
const { MongoClient } = require("mongodb");
require("dotenv").config();

const url = process.env.DATABASE_URL;
const client = new MongoClient(url);

try {
  if (!url) {
    throw new Error("DATABASE_URL is not defined in .env");
  }

  await client.connect();
  console.log("Connected to MongoDB!");

  const db = client.db("ecommerce");
  const products = db.collection("products");

  const product = {
    product_id: 101,
    name: "Laptop",
    category: "Electronics",
```

```
    price: 1200,
    stock: 15,
  };

  const result = await products.insertOne(product);
  console.log("Inserted product:", result.insertedId);

  const queriedProduct = await products.findOne({ product_id: 101 });
  console.log("Queried product:", queriedProduct);
} catch (err) {
  console.error("Error:", err);
} finally {
  await client.close();
}
```

Schlussendlich rufe ich das Projekt mit einem kräftigen `docker-compose up` ins leben und erhalte von dem app Container folgende Konsolenausgabe:

```
app-1  |
app-1  | > start
app-1  | > node ./index.js
app-1  |
app-1  | Connected to MongoDB!
app-1  | Inserted product: new ObjectId('67af0a8eedeb93e9d8b35ebb')
app-1  | Queried product: {
app-1  |   _id: new ObjectId('67af0096679507bcdc53e003'),
app-1  |   product_id: 101,
app-1  |   name: 'Laptop',
app-1  |   category: 'Electronics',
app-1  |   price: 1200,
app-1  |   stock: 15
app-1  | }
app-1 exited with code 0
```