Data Structures and Databases - Hands On Lab

Outline

Overview

In this lab we will be comparing usage between a traditional relational database and a NoSQL database. Specifically, we will be using MySQL and MongoDB. The choice of these two platforms was made based on current popularity. Currently, MySQL and MongoDB seem to be the open source leaders by a considerable margin in each of their categories.

Why MySQL & MongoDB?[1]

DBMS	Type	License	Popularity Score	
Oracle	RDBMS	Proprietary	21.3	
MySQL	RDBMS	Open	20.9	
Microsoft SQL Server	RDBMS	Proprietary	18.3	
PostgreSQL	RDBMS	Open	5.4	
MongoDB	NoSQL	Open	5	
DB2	RDBMS	Proprietary	2.8	
Microsoft Access	RDBMS	Proprietary	2	
Cassandra	NoSQL	Open	2	
SQLite	RDBMS	Open	1.8	
Redis	NoSQL	Open	1.7	

We will be downloading the data files from the NCBI Taxonomy FTP site and create both databases from this data. The main goals of this lab is to create tables with proper data types, relationships, constraints, and normalization practices.

A database is a collection of data consisting of a physical file residing on a computer. The collection of data in that file is stored in different tables where each row in the table is considered as a record. Every record is broken down into fields that represent single items of data describing a specific thing.

More technically, a database can also be defined as an organized structured object stored on a computer consisting of data and metadata. Data, as previously explained, is the actual information stored in the database, while metadata is data about the data. Metadata describes the structure of the data itself, such as field length or datatype. For example, in a company database the value 6.95 stored in a field is data about the price of a specific product. The information that this is a number data stored to two decimal places and valued in dollars is metadata.

Databases are usually associated with software that allows for the data to be updated and queried. The software that manages the database is called a Relational Database Management System (RDBMS). These systems make storing data and returning results easier and more efficient by allowing different questions and commands to be posed to the database. Popular RDBMS software includes Oracle Database, Microsoft SQL Server, MySQL, and IBM DB2. Commonly, the RDBMS software itself is referred to as a database, although theoretically this would be a slight misnomer. When working with databases we will participate in the design, maintenance and administration of the database that supplies data to our website or application. In order to do this, however,we will need to access that data and also automate the process to allow other users to retrieve and perhaps even modify data without technical knowledge. To achieve this we will need to communicate with the database in a language it can interpret. Structured Query Language (SQL) will allow us to directly communicate with databases and is thus the subject of this book. In this book we will learn the basics of SQL. SQL is composed of commands that enable users to create database and table structures, perform various types of data manipulation and data administration and query the database in order to extract useful information.

SQL

The Structured Query Language (SQL) is the standard language for relational database management systems (RDBMS). The language has standards developed by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO). However, most (if not all) vendors use proprietary keywords and formats which do not allow code to be portable without modifications.

Relational databases heavily borrows concepts from two area of mathematics, set theory and predicate logic. This is reflected in the SQL language. While an explanation of the mathematics is beyond the scope of this course, it is important to highlight one of the more impactful consequences.

Relational databases and SQL utilize ternary logic; values, regardless of it data type, may be set as *unknown*. As a conseeuqnce, conditional expressions may evalue to *true*, *false*, or *unknown*. In SQL the *unknown* value is represented as *NULL*.

Ternary Truth Tables

The following represent how ternary values are evaluated with basic logic operations:

A	**NOT A**
F	T
U	U
T	F

A AND B		**B**		
		F	**U**	**T**
A	**F**	F	F	F
	U	F	U	U
	T	F	U	T

A OR B		**B**		
		F	**U**	**T**
A	**F**	F	U	T
	U	U	U	T
	T	T	T	T

Terminology

There are often slight variances on how technical terms are defined depenent on individual platforms. Since we will be using MySQL the terminologies used in this lab will prefer this context.

- **Database** & **Schema**: Most database vendors define schema as a collection of tables and a database as a collection of schemas. MySQL, however, use database and schema interchangeably with the former definition; a MySQL schema and database refer to a collection of tables. The MySQL documentation uses the plural form of schema (schemas) as its collection.

- Table
- Row
- Column
- Statement
- Query
- Result set
- Predicate: specifies conditions that can be evaluated to one of *true*, *false*, or *NULL*.

SQL statements may be classified into the following categories:

- **Data Query Language (DQL)**: Statements that query and do not modify data. These statements begin with SELECT.
- **Data Manipulation Language (DML)**: Statements that modify data. These statements generally begin with INSERT, UPDATE, DELETE.
- **Data Definition Languate (DDL)**: Statements that create and modify the schema. DQL and DML process data within the schema, DDL changes the structures which contains that data. These statements usually begin with INSERT, UPDATE, DELETE, OR DROP.

Common SQL Commands

Command	Description
SELECT	extract database data
UPDATE	updates database data
DELETE	deletes database data
INSERT	inserts new database data
CREATE DATABASE	creates new database
ALTER DATABASE	modifies database
CREATE TABLE	creates a table
ALTER TABLE	modifies a table
DROP TABLE	removes or deletes a table
CREATE INDEX	creates search key or index
DROP INDEX	deletes an index

MySQL

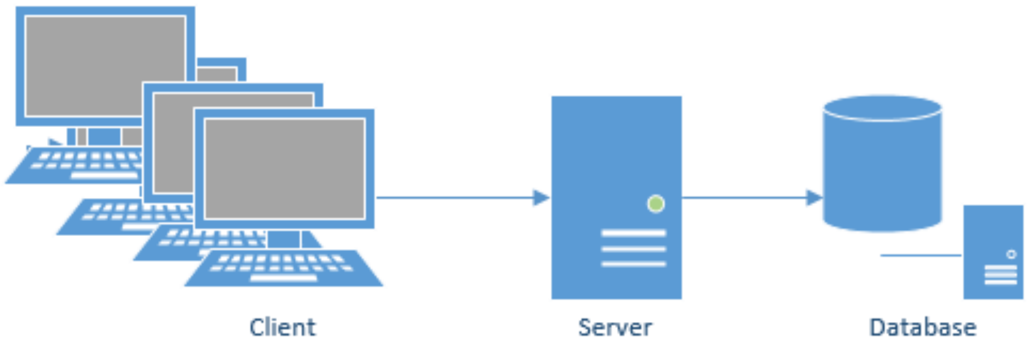
We will be using the MySQL platform for these exercises. Each workstation will have its own local database server to which we'll connect using MySQL Workbench.

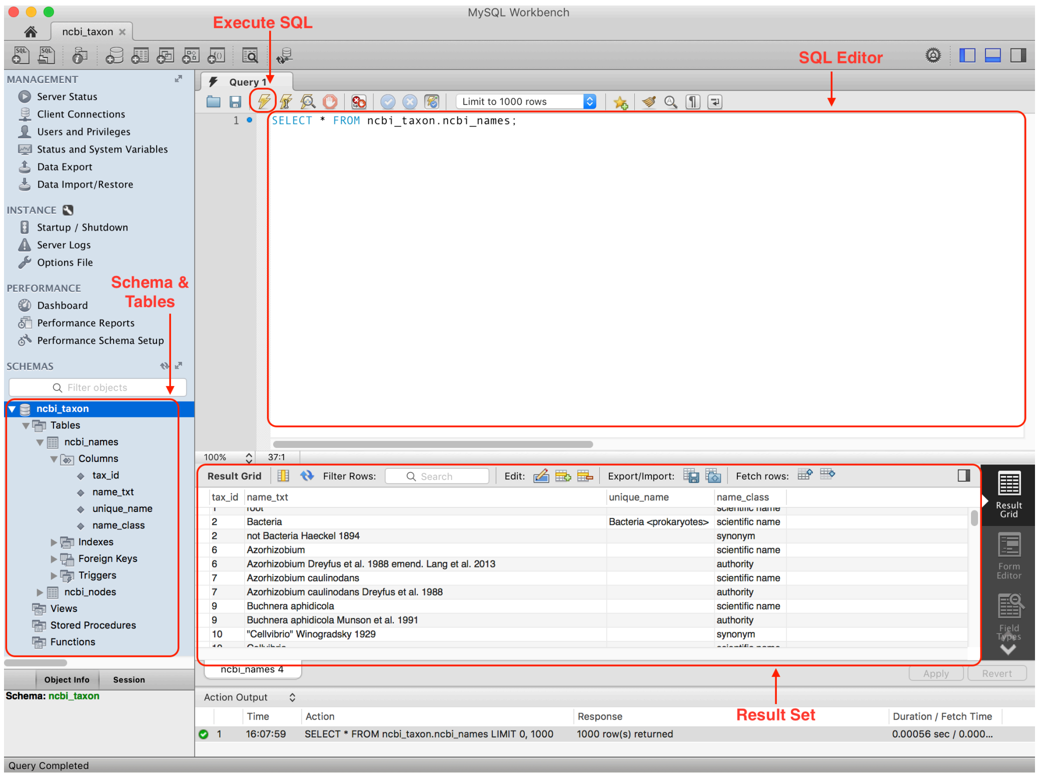
MySQL Workbench

MySQL Workbench is the endorsed intergrated development environment for the MySQL server. It features a graphical SQL client along with design and administrative tools specifically for MySQL.

Client and server

MySQL Workbench and the MySQL database are actually two independently running applications. They may not (usually are not) running on the same machine. The client makes a connection to the server and then sends it commands, such as SQL. If the client sent a query, then it will await a reply then display or process the results. Modern RDBMS servers are designed to handle several client connections simultaneously.





```
SELECT * FROM ncbi_taxon.ncbi_names;
```

tax_id	name_txt	unique_name	name_class
2	Bacteria	Bacteria <prokaryotes>	scientific name
2	not Bacteria Haeckel 1894		synonym
6	Azorhizobium		scientific name
6	Azorhizobium Dreyfus et al. 1988 emend. Lang et al. 2013		authority
7	Azorhizobium caulinodans		scientific name
7	Azorhizobium caulinodans Dreyfus et al. 1988		authority
9	Buchnera aphidicola		scientific name
9	Buchnera aphidicola Munson et al. 1991		authority
10	"Cellvibrio" Winogradsky 1929		synonym

	Time	Action	Response	Duration / Fetch Time
1	16:07:59	SELECT * FROM ncbi_taxon.ncbi_names LIMIT 0, 1000	1000 row(s) returned	0.00056 sec / 0.000...

Useful Online Resources

- The official reference manual for MySQL is available online.
- The Instant SQL Formatter is useful for formatting and cleaning up long SQL statements.

NCBI Taxonomy Data

The datasets used by the subsequent exercises are available for download at the NCBI FTP site. We are mostly interested in the NCBI Taxonomy archive file taxdmp.zip.

Downloading the data

1. Download taxdmp.zip
2. Unzip the file. The contents will be uncompressed to the `taxdmp` directory.
3. Verify that ther files within the `taxdmp` directory include:
    - `names.dmp`
    - `nodes.dmp`
    - `readme.txt`
4. Note the location of the directory `taxdmp` ; we will need this path later.

Examining the data

The first step to creating a database is to gain a familiarity and understanding of the data to be housed.

1. The `readme.txt` contains a short description of the data files. Take a few moments to read the descriptions.
2. Open `names.dmp` and `nodes.dmp` in a text editor to examine the first several rows.

Consider the following questions for discussion:

- Are there any inherent relationships within the data files?
- Which columns within the datasets are good candidates for keys?
- Which data types should we use for various columns – integer, float, date/time, bit, etc.?
- Are there any constraints we might expect to exist within the data, e.g. which columns should always have values, which are optional, which should be unique, are there any implied values?
- Are there opportunities for normalization?

Importing the data

In this exercise, we will be loading the NCI Taxonomy files into a MySQL database (also known as a schema).

We will be using built-in MySQL functions to load the raw flat files into a table. The initial tables will have simple column definitions without any relational features.

1. Open MySQL Workbench and paste the following SQL code into the query tab.
2. Change the paths in lines `17` and `42` to the location of the `taxdmp` directory.

```
 1  DROP SCHEMA IF EXISTS ncbi_taxon;
 2
 3  CREATE SCHEMA ncbi_taxon;
 4
 5  USE ncbi_taxon;
 6
 7
 8  DROP TABLE IF EXISTS `ncbi_names`;
 9
10  CREATE TABLE `ncbi_names` (
11    `tax_id` int unsigned NOT NULL,
12    `name_txt` varchar(255) NOT NULL,
13    `unique_name` varchar(255) NOT NULL DEFAULT '',
14    `name_class` varchar(63) NOT NULL
15  );
16
17  LOAD DATA LOCAL INFILE '/path/to/taxdmp/names.dmp'
18  INTO TABLE `ncbi_names`
19  FIELDS TERMINATED BY '\t|\t'
20  LINES TERMINATED BY '\t|\n'
21  (tax_id, name_txt, unique_name, name_class);
22
23
24  DROP TABLE IF EXISTS `ncbi_nodes`;
25
26  CREATE TABLE `ncbi_nodes` (
27        `tax_id`                          INT UNSIGNED NOT NULL,
28        `parent_tax_id`                   INT UNSIGNED NOT NULL,
29        `rank`                            VARCHAR(32) NOT NULL,
30        `embl_code`                       VARCHAR(16) DEFAULT NULL,
31        `division_id`                     SMALLINT NOT NULL,
32        `inherited_div_flag`              TINYINT NOT NULL,
33        `genetic_code_id`                 SMALLINT NOT NULL,
34        `inherited_gc_flag`               TINYINT NOT NULL,
35        `mitochondrial_genetic_code_id`   TINYINT NOT NULL,
36        `inherited_mgc_flag`              TINYINT NOT NULL,
37        `genbank_hidden_flag`             TINYINT NOT NULL,
38        `hidden_subtree_root_flag`        TINYINT NOT NULL,
    `comments`                      VARCHAR(255) DEFAULT NULL
);

LOAD DATA LOCAL INFILE '/path/to/taxdmp/nodes.dmp'
INTO TABLE `ncbi_nodes`
FIELDS TERMINATED BY '\t|\t'
LINES TERMINATED BY '\t|\n'
(tax_id,parent_tax_id,rank,embl_code,division_id,inherited_div_flag,
genetic_code_id,inherited_GC_flag,mitochondrial_genetic_code_id,
inherited_MGC_flag,GenBank_hidden_flag,hidden_subtree_root_flag,comments);
```

Querying from a table

Introduction

Retrieve all rows from a table and return the specified columns as the result set.

`SELECT c1, c2 FROM t;`

Example:

> `SELECT name_txt, name_class FROM ncbi_names;`

Retrieve all rows from a table and return all columns as the result set.

> `SELECT * FROM <table>;`

Example:

> `SELECT * FROM ncbi_names;`

Retrieve a result set with a condition.

> `SELECT <column> [,<column>...] FROM <table>` `WHERE <condition>;`

Example:

> `SELECT name_txt, name_class FROM ncbi_names`
> `WHERE name_class = 'scientific name';`

Retrieve rows from a table and return a unique result set.

> `SELECT DISTINCT <column> [,<column>...] FROM <table>`

Example:

> `SELECT DISTINCT name_class FROM ncbi_names;`

Retrieve rows from a table and sort the result set.

> `SELECT <column> [,<column>...] FROM <table>`
> `ORDER BY <column> [ASC|DESC] [,<column> [ASC|DESC]...];`

Example:

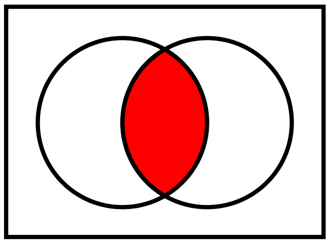
> `SELECT name_txt, name_class FROM ncbi_names` `ORDER BY name_class;`

Querying from multiple tables

Introduction

Retrieve the rows from two tables with an exclusive condition.

> `SELECT <column> [,<column>...]`
> `FROM <table> INNER JOIN <table> ON <condition>;`



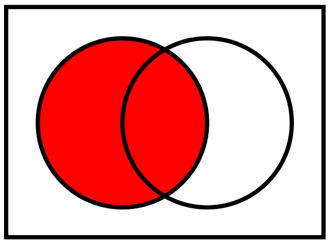
Example:

```sql
SELECT child.tax_id, child.parent_tax_id, child.rank, parent.rank
FROM
ncbi_nodes AS child
INNER JOIN
ncbi_nodes AS parent ON child.parent_tax_id = parent.tax_id;
```

tax_id	parent_tax_id	rank	rank
1	1	no rank	no rank
2	131567	superkingdom	no rank
6	335928	genus	family
7	6	species	genus
9	32199	species	genus
10	1706371	genus	family
11	1707	species	genus
13	203488	genus	family
14	13	species	genus
16	32011	genus	family

Retrieve the rows from a table and include the rows from another table.

```
SELECT c1, c2 FROM t1 LEFT JOIN t2 ON condition;
```



Example:

```
SELECT
    child.tax_id, child.parent_tax_id, child.rank, parent.rank
FROM
    ncbi_nodes AS child
        LEFT JOIN
    ncbi_nodes AS parent ON child.parent_tax_id = parent.tax_id AND parent.rank = '
```

tax_id	parent_tax_id	rank	rank
1	1	no rank	NULL
2	131567	superkingdom	NULL
6	335928	genus	NULL
7	6	species	genus
9	32199	species	genus
10	1706371	genus	NULL
11	1707	species	genus
13	203488	genus	NULL
14	13	species	genus
16	32011	genus	NULL

Retrieve rows from a table joined to other rows in the same table

```
SELECT c1, c2 FROM t1 AS A LEFT JOIN t1 AS B ON condition;
```

Example:

```
SELECT child.tax_id, child.parent_tax_id, child.rank, parent.rank
FROM ncbi_nodes AS child INNER JOIN
ncbi_nodes AS parent ON child.parent_tax_id = parent.tax_id;
```

Querying aggregates

Aggregate functions compute a result against a column within a result set. These functions are typically statistical functions.

Function	Description
AVG	Return the average value
COUNT	Return the number of rows
MAX	Return the maximum value
MIN	Return the minimum value
SUM	Return the sum of values
STDDEV	Return the standard deviation
VARIANCE	Return the standard variance

Retrieve rows from a table and return an aggregated result set.

```
SELECT {<aggregate>(<column>)|<column>}[,...] FROM <table>
GROUP BY <column>[,...]
```

Example:

```
SELECT name_class, COUNT(tax_id) FROM ncbi_taxon.ncbi_names
GROUP BY name_class;
```

Retrieve rows from a table and return a filtered aggregated result set.

```
SELECT {<aggregate>(<column>)|<column>}[,...] FROM <table>
GROUP BY <column>[,...] HAVING <condition>
```

Example:

```
SELECT name_class, COUNT(tax_id) FROM ncbi_taxon.ncbi_names
GROUP BY name_class HAVING name_class = 'blast name';
```

SQL operators

Introduction

Combine rows from two queries

`SELECT c1, c2 FROM t1` `UNION [ALL]` `SELECT c1, c2 FROM t2;`

Query rows using pattern matching %, _

`SELECT c1, c2 FROM t1` `WHERE c1 [NOT] LIKE pattern;`

Query rows in a list

`SELECT c1, c2 FROM t` `WHERE c1 [NOT] IN (list);`

Query rows between two values

`SELECT c1, c2 FROM t` `WHERE c1 BETWEEN low AND high;`

Check if values in a table is NULL or not

`SELECT c1, c2 FROM t` `WHERE c1 IS [NOT] NULL;`

Modifying data

Introduction

Insert one row into a table

`INSERT INTO t (column_list) VALUES (values);`

Insert rows from t2 into t1

`INSERT INTO t1 (columns)` `SELECT column_list` `FROM t2;`

Update new value in the column c1 for all rows

`UPDATE t` `SET c1 = value;`

Update values in the column c1, c2 that match the condition

`UPDATE t` `SET c1 = value, c2 = value` `WHERE condition;`

Delete all data in a table

`DELETE FROM t;`

Delete subset of rows in a table

`DELETE FROM t` `WHERE condition;`

Managing a schema

Introduction

Create a new table with three columns

```
CREATE TABLE t (
id INT PRIMARY KEY,
name VARCHAR NOT NULL,
price INT DEFAULT 0
);
```

Delete a table from the schema

```
DROP TABLE t;
```

Create an index

```
CREATE INDEX <index name>
ON <table> (<column>);
```

Delete an index

```
DROP INDEX <index name>
ON <table>;
```

MongoDB

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling. MongoDB obviates the need for an Object Relational Mapping (ORM) to facilitate development.

Documents

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

```
{
    "_id": ObjectId("58cf0175efd4b08ca194b20a"),
    "tax_id": 1,
    "parent_tax_id": 1,
    "rank": "no rank",
    "name_txt": "root"
},
{
    "_id": ObjectId("58cf0175efd4b08ca194b20b"),
    "tax_id": 2,
    "parent_tax_id": 131567,
    "rank": "superkingdom",
    "name_txt": "Bacteria"
},
{
    "_id": ObjectId("58cf0175efd4b08ca194b20c"),
    "tax_id": 6,
    "parent_tax_id": 335928,
    "rank": "genus",
    "name_txt": "Azorhizobium"
},
{
    "_id": ObjectId("58cf0175efd4b08ca194b20d"),
    "tax_id": 7,
    "parent_tax_id": 6,
    "rank": "species",
    "name_txt": "Azorhizobium caulinodans"
},
{
    "_id": ObjectId("58cf0175efd4b08ca194b20e"),
    "tax_id": 9,
    "parent_tax_id": 32199,
    "rank": "species",
    "name_txt": "Buchnera aphidicola"
},
{
    "_id": ObjectId("58cf0175efd4b08ca194b20f"),
    "tax_id": 10,
    "parent_tax_id": 1706371,
    "rank": "genus",
    "name_txt": "Cellvibrio"
}
```

Collections

MongoDB stores documents in collections. Collections are analogous to tables in relational databases. Unlike a table, however, a collection does not require its documents to have the same schema.

In MongoDB, documents stored in a collection must have a unique _id field that acts as a primary key.

Import Example Dataset

Overview

The examples in this guide use the nodes collection in the taxon database. The following is a sample document in the nodes collection:

```
{
    "tax_id" : 2,
    "parent_tax_id" : 131567,
    "rank" : "superkingdom",
    "name_txt" : "Bacteria"
}
```

Use the following procedure to populate the nodes collection.

Prerequisites

You must have a running mongod instance in order to import data into the database.

Procedure

1. Retrieve the nodes data.

   Retrieve the dataset from https://s3.amazonaws.com/niehs/ncbi_nodes.json.zip and unzip.

2. Import data into the collection.

   In the system shell or command prompt, use mongoimport to insert the documents into the nodes collection in the taxon database. If the collection already exists in the taxon database, the operation will drop the nodes collection first.

```
mongoimport --db taxon --collection nodes --drop --file ~/downloads/ncbi_nodes.
```

The mongoimport connects to a mongod instance running on localhost on port number 27017. The --file option provides the path to the data to import, in this case, ~/downloads/ncbi_nodes.json.

To import data into a mongod instance running on a different host or port, specify the hostname or port by including the --host and the --port options in your mongoimport command.

MongoDB Shell

Start mongo

Once you have installed and have started MongoDB, connect the mongo shell to your running MongoDB instance. Ensure that MongoDB is running before attempting to launch the mongo shell.

On the same system where the MongoDB is running, open a terminal window (or a command prompt for Windows) and run the mongo shell with the following command:

```
mongo
```

On Windows systems, add .exe as follows:

```
mongo.exe
```

You may need to specify the path as appropriate.

When you run mongo without any arguments, the mongo shell will attempt to connect to the MongoDB instance running on the localhost interface on port 27017.

Help in mongo Shell

Type **help** in the mongo shell for a list of available commands and their descriptions:

```
help
```

The mongo shell also provides key completion as well as keyboard shortcuts similar to those found in the bash shell or in Emacs. For example, you can use the and the to retrieve operations from its history.

Query Data

Overview

You can use the **find()** method to issue a query to retrieve data from a collection in MongoDB. All queries in MongoDB have the scope of a single collection.

Queries can return all documents in a collection or only the documents that match a specified filter or criteria. You can specify the filter or criteria in a document and pass as a parameter to the **find()** method.

The **find()** method returns query results in a cursor, which is an iterable object that yields documents.

Prerequisites

The examples in this section use the nodes collection in the taxon database. For instructions on populating the collection with the sample dataset, see Import Example Dataset.

In the mongo shell connected to a running mongod instance, switch to the taxon database.

```
use taxon
```

Query for All Documents in a Collection

To return all documents in a collection, call the **find()** method without a criteria document. For example, the following operation queries for all documents in the nodes collection.

```
db.nodes.find()
```

The result set contains all documents in the nodes collection.

Specify Equality Conditions

The query condition for an equality match on a field has the following form:

```
{ <field1>: <value1>, <field2>: <value2>, ... }
```

If the is a top-level field and not a field in an embedded document or an array, you can either enclose the field name in quotes or omit the quotes.

If the is in an embedded document or an array, use dot notation to access the field. With dot notation, you must enclose the dotted name in quotes.

Query by a Top Level Field

The following operation finds documents whose rank field equals "species".

```
db.nodes.find( { "rank": "species" } )
```

The result set includes only the matching documents.

Query by a Field in an Embedded Document

To specify a condition on a field within an embedded document, use the dot notation. Dot notation requires quotes around the whole dotted field name. The following operation specifies an equality condition on the scientific name field in the names embedded document.

```
db.nodes.find( { "names.scientific": "Dictyoglomus" } )
```

The result set includes only the matching documents.

For more information on querying on fields within an embedded document, see Query on Embedded/Nested Documents.

Query by a Field in an Array

The grades array contains embedded documents as its elements. To specify a condition on a field in these documents, use the dot notation. Dot notation requires quotes around the whole dotted field name. The following queries for documents whose synonym names array contains an embedded document with a field value equal to "Enterococcus coli".

```
db.nodes.find( { "names.synonym": "Enterococcus coli" } )
```

The result set includes only the matching documents.

For more information on querying on arrays, such as specifying multiple conditions on array elements, see Query an Array and $elemMatch.

Specify Conditions with Operators

MongoDB provides operators to specify query conditions, such as comparison operators. Although there are some exceptions, such as the $or and $and conditional operators, query conditions using operators generally have the following form:

```
{ <field1>: { <operator1>: <value1> } }
```

For a complete list of the operators, see query operators.

Greater Than Operator ($gt)

Query for documents whose synonym names array contains an embedded document with a name that starts with a letter after D.

```
db.nodes.find( { "names.synonym": { $gt: "D" } } )
```

The result set includes only the matching documents.

Less Than Operator ($lt)

Query for documents whose grades array contains an embedded document with a name that starts with a letter before D.

```
db.nodes.find( { "names.synonym": { $lt: "D" } } )
```

The result set includes only the matching documents.

Combine Conditions

You can combine multiple query conditions in logical conjunction (AND) and logical disjunctions (OR).

Logical AND

You can specify a logical conjunction (AND) for a list of query conditions by separating the conditions with a comma in the conditions document.

```
db.nodes.find( { "rank": "superkingdom", "name.scientific": "Bacteria" } )
```

The result set includes only the documents that matched all specified criteria.

Logical OR

You can specify a logical disjunction (OR) for a list of query conditions by using the $or query operator.

```
db.nodes.find(
   { $or: [ { "rank": "superkingdom" }, { "name.scientific": "Bacteria" } ] }
)
```

The result set includes only the documents that match either conditions.

Sort Query Results

To specify an order for the result set, append the **sort()** method to the query. Pass to **sort()** method a document which contains the field(s) to sort by and the corresponding sort type, e.g. 1 for ascending and -1 for descending.

For example, the following operation returns all documents in the nodes collection, sorted first by the parenttaxid field in ascending order, and then, within each set with the same parent, by the "name.scientific" field in ascending order:

```
db.nodes.find().sort( { "parent_tax_id": 1, "name.scientific": 1 } )
```

The operation returns the results sorted in the specified order.

Insert Data

You can use the insert() method to add documents to a collection in MongoDB. If you attempt to add documents to a collection that does not exist, MongoDB will create the collection for you.

Prerequisites

In the mongo shell connected to a running mongod instance, switch to the taxon database.

use nodes

Insert a Document

Insert a document into a collection named nodess. The operation will create the collection if the collection does not currently exist.

```
db.nodes.insert(
    {
        "tax_id": 22,
        "parent_tax_id": 267890,
        "rank": "genus",
        "name_txt": "Shewanella"
    }
)
```

The method returns a WriteResult object with the status of the operation.

```
WriteResult({ "nInserted" : 1 })
```

If the document passed to the insert() method does not contain the _id field, the mongo shell automatically adds the field to the document and sets the field’s value to a generated ObjectId.

Update Data

You can use the update() method to update documents of a collection. The method accepts as its parameters:

- a filter document to match the documents to update,
- an update document to specify the modification to perform, and
- an options parameter (optional).

To specify the filter, use the same structure and syntax as the query conditions. See Find or Query Data with the mongo Shell for an introduction to query conditions.

By default, the update() method updates a single document. Use the multi option to update all documents that match the criteria.

You cannot update the _id field.

Prerequisites

The examples in this section use the nodes collection in the taxon database. For instructions on populating the collection with the sample dataset, see Import Example Dataset.

In the mongo shell connected to a running mongod instance, switch to the taxon database.

```
use taxon
```

Update Specific Fields

To change a field value, MongoDB provides update operators, such as $set to modify values. Some update operators, such as $set, will create the field if the field does not exist. See the individual update operators reference.

Update Top-Level Fields

The following operation updates the first document with scientific name equal to "Methylophilus methylotrophus", using the $set operator to update the comment field and the $currentDate operator to update the lastModified field with the current date.

```
db.nodes.update(
    { "name.scientific" : "Methylophilus methylotrophus" },
    {
      $set: { "comment": "See https://www.wikigenes.org/e/mesh/e/19198.html" },
      $currentDate: { "lastModified": true }
    }
)
```

The update operation returns a WriteResult object which contains the status of the operation.

Update an Embedded Field

To update a field within an embedded document, use the dot notation. When using the dot notation, enclose the whole dotted field name in quotes. The following updates the scientific field in the embedded name document.

```
db.nodes.update(
  { "tax_id" : "17" },
  { $set: { "name.scientific": "Methylophilus methylotrophus" } }
)
```

The update operation returns a WriteResult object which contains the status of the operation.

Update Multiple Documents

By default, the update() method updates a single document. To update multiple documents, use the multi option in the update() method. The following operation updates all documents that have a parent taxon id field equal to 22, setting the comment field to "See https://www.wikigenes.org/e/mesh/e/19198.html" and the lastModified field to the current date.

```
db.nodes.update(
  { "parent_tax_id": "22" },
  {
    $set: { "comment": "See https://www.wikigenes.org/e/mesh/e/19198.html" },
    $currentDate: { "lastModified": true }
  },
  { multi: true}
)
```

The update operation returns a WriteResult object which contains the status of the operation.

Replace a Document

To replace the entire document except for the _id field, pass an entirely new document as the second argument to the update() method. The replacement document can have different fields from the original document. In the replacement document, you can omit the _id field since the _id field is immutable. If you do include the _id field, it must be the same value as the existing value.

IMPORTANT After the update, the document only contains the field or fields in the replacement document. After the following update, the modified document will only contain the taxid field, parenttax_id field, and the scientific name field. i.e. the document will not contain the comment, and the other name fields.

```
db.nodes.update(
   { "tax_id" : 22 },
   {
     "parent_tax_id": 1,
     "name.scientific" : "Methylophilus methylotrophus"
   }
)
```

The update operation returns a WriteResult object which contains the status of the operation.

Remove Data

You can use the remove() method to remove documents from a collection. The method takes a conditions document that determines the documents to remove.

To specify a remove condition, use the same structure and syntax as the query conditions. See Find or Query Data with the mongo Shell for an introduction to query conditions.

Prerequisites

The examples in this section use the nodes collection in the taxon database. For instructions on populating the collection with the sample dataset, see Import Example Dataset.

In the mongo shell connected to a running mongod instance, switch to the taxon database.

```
use taxon
```

Procedures

Remove All Documents That Match a Condition

The following operation removes all documents that match the specified condition.

```
db.nodes.remove( { "tax_id": "31" } )
```

The remove operation returns a WriteResult object which contains the status of the operation. nRemoved field specifies the number of documents removed.

Use the justOne Option

By default, the remove() method removes all documents that match the remove condition. Use the justOne option to limit the remove operation to only one of the matching documents.

```
db.nodes.remove( { "parent_tax_id": "37" }, { justOne: true } )
```

Successful operation should return the following WriteResult object.

```
WriteResult({ "nRemoved" : 1 })
```

nRemoved field specifies the number of documents removed, in this case 1.

Remove All Documents

To remove all documents from a collection, pass an empty conditions document {} to the remove() method.

```
db.nodes.remove( { } )
```

The remove operation returns a WriteResult object which contains the status of the operation. nRemoved field specifies the number of documents removed.

Drop a Collection

The remove all operation only removes the documents from the collection. The collection itself, as well as any indexes for the collection, remain. To remove all documents from a collection, it may be more efficient to drop the entire collection, including the indexes, and then recreate the collection and rebuild the indexes. Use the drop() method to drop a collection, including any indexes. `db.nodes.drop()`

Upon successful drop of the collection, the operation returns true.

```
true
```

If the collection to drop does not exist, the operation will return false.

Further Reading

- Ranking of Relational DBMS
- Performance of Graph vs. Relational Databases | Database Zone
- Graph theory | Wikipedia

References

1. DB-Engines Ranking March 2017