English

Creating and populating a database

Last updated 2023-04-20

This tutorial shows you how to use the Python programming language to create an IBM® Cloudant® for IBM Cloud® database in your IBM Cloud service instance. You also learn how to populate the database with a simple collection of data.

> **Note:** This tutorial doesn't use the most efficient Python code. The intent is to show simple and easy-to-understand working code that you can learn from and apply to your own applications. You must apply normal best practices for checking and handling all warning or error conditions that are encountered by your own applications.

Objectives

This tutorial builds on a series of Python language instructions, suitable for the following tasks:

1. Connecting to an IBM Cloudant service instance on IBM Cloud®.
2. Creating a database within the service instance.
3. Storing a small collection of data as documents within the database.
4. Retrieving data.
5. Deleting the database.
6. Closing the connection to the service instance.

Before you begin

This tutorial provides you with the following options:

- Follow each step as outlined in this tutorial.
- Or execute the Python script, and come back to Step 5. Retrieving data.

Installing Python

Tip: Normally, you don't run commands individually in Python. You usually create a script, which is a list of the commands you want to run, stored in a Python file, with a `py` extension.

1. Set up service credential requirements.

   a. Create a service instance and credentials by following the Getting started tutorial.

   b. Locate your service credentials by following this tutorial.

2. Install the required version of Python.

   You must have a current version of the Python programming language that is installed on your system.

   Check that Python is installed by running the following command at a prompt:

```
$ python3 --version
```

   b. Verify that you get a result similar to the following example:

```
Python 3.8.1
```

3. Verify that your Python Client Library meets the requirement.

Deprecated: The following examples use the deprecated `python-cloudant` client library.

a. Check that the client library installed successfully by running the following command at a prompt:

```
$ pip freeze
```

You get a list of all the Python modules installed on your system.

b. Inspect the list, looking for an IBM Cloudant entry similar to the following example:

```
cloudant==2.14.0
```

Step 1: Connecting to a service instance

You must connect to your service instance before you create a database.

The following components are identified as normal `import` statements.

You can follow steps 1 - 5 to learn about the individual commands, or go to the end of the tutorial to execute the Python script. When you finish, return to Step 5. Retrieving data.

1. Run these `import` statements to connect to the service instance.

```
from cloudant.client import Cloudant
from cloudant.error import CloudantException
from cloudant.result import Result, ResultByKey
```

2. Find `username`, `password`, and `URL` in your Classic service credentials and replace `serviceUsername`, `servicePassword`, and `serviceURL` in the following example.

```
serviceUsername = "apikey-v2-58B528DF5397465BB6673E1B79482A8C"
servicePassword = "49c0c343d225623956157d94b25d574586f26d1211e8e589646b4713d5de4801"
serviceURL = "https://353466e8-47eb-45ce-b125-4a4e1b5a4f7e-bluemix.cloudant.com"
```

3. Establish a connection to the service instance.

```
client = Cloudant(serviceUsername, servicePassword, url=serviceURL)
client.connect()
```

4. Or replace `ACCOUNT_NAME` and `API_KEY` with the values from your IAM API service credentials.

```
client = Cloudant.iam(ACCOUNT_NAME, API_KEY, connect=True)
```

Now, your Python application can access the service instance on IBM Cloud.

Step 2: Creating a database within the service instance

Next, you create a database within the service instance, called `databasedemo`.

1. Create this instance by defining a variable in the Python application.

```
databaseName = "databasedemo"
```

2. Create the database.

```
myDatabaseDemo = client.create_database(databaseName)
```

3. Verify that the database was created successfully.

```
if myDatabaseDemo.exists():
    print("'{0}' successfully created.\n".format(databaseName))
```

Step 3: Storing a small collection of data as documents within the database

You want to store a small, simple collection of data in the database. This data is used in other tutorials, like Using IBM Cloudant Query to find data.

1. Create sample data.

```
sampleData = [
     [1, "one", "boiling", 100],
     [2, "two", "hot", 40],
     [3, "three", "hot", 75],
     [4, "four", "hot", 97],
     [5, "five", "warm", 20],
     [6, "six", "cold", 10],
     [7, "seven", "freezing", 0],
     [8, "eight", "freezing", -5]
]
```

2. Use a `for` statement to retrieve the fields in each row by going through each row in the array.

```
for document in sampleData:
    # Retrieve the fields in each row.
    number = document[0]
    name = document[1]
    description = document[2]
    temperature = document[3]
```

3. Create a JSON document that represents all the data in the row.

```
jsonDocument = {
    "numberField": number,
    "nameField": name,
    "descriptionField": description,
    "temperatureField": temperature
}
```

4. Create a document by using the Database API.

```
newDocument = myDatabaseDemo.create_document(jsonDocument)
```

5. Check that the document exists in the database.

```
if newDocument.exists():
    print("Document '{0}' successfully created.".format(number))
```

Step 4: Retrieving data

To perform a minimal retrieval, you first request a list of all documents within the database. This list is returned as an array. You can then show the content of an element in the array.

1. Retrieve a minimal amount of data.

```
result_collection = Result (myDatabaseDemo.all_docs)
print("Retrieved minimal document:\n{0}\n".format(result_collection[0]))
```

2. See a result similar to the following example.

```
[{
        "id": '60e19edf809418e407fb6791a1d8fec4',
        "key": '60e19edf809418e407fb6791a1d8fec4',
        "value": {
                "rev": '2-3d6dc27627114431c049ddecae9796e0'
        }
}]
```

In a relational database, the first document that is stored in a database is always the first document that is returned in a list of results. This notion doesn't necessarily apply to NoSQL databases, such as IBM Cloudant.

Full retrieval of a document

Additionally, to perform a full retrieval, you request a list of all documents within the database, and specify that the document content must also be returned. You run a full retrieval by using the `include_docs` option. As before, the results are returned as an array. You can then show the details of an element in the array by including the full content of the document this time.

1. Request the first document that is retrieved from the database.

```
result_collection = Result(myDatabaseDemo.all_docs, include_docs=True)
print("Retrieved minimal document:\n{0}\n".format(result_collection[0]))
```

2. See the result, which is similar to the following example:

```
[
    {
        "value": {
          "rev": "1-b2c48b89f48f1dc172d4db3f17ff6b9a"
        },
        "id": "14746fe384c7e2f06f7295403df89187",
        "key": "14746fe384c7e2f06f7295403df89187",
        "doc": {
            "temperatureField": 10,
            "descriptionField": "cold",
            "numberField": 6,
            "nameField": "six",
            "_id": "14746fe384c7e2f06f7295403df89187",
            "_rev": "1-b2c48b89f48f1dc172d4db3f17ff6b9a"
        }
```

Show more

Step 5: Calling an IBM Cloudant API endpoint directly

You can work with the IBM Cloudant API endpoints directly, from within a Python application.

In this example code, you again request a list of all the documents, including their content. However, this time you do so by invoking the IBM Cloudant /_all_docs endpoint.

1. Identify the endpoint to contact and any parameters to supply with it.

```
end_point = '{0}/{1}'.format(serviceURL, databaseName + "/_all_docs")
params = {'include_docs': 'true'}
```

2. Send the request to the service instance and show the results.

```
response = client.r_session.get(end_point, params=params)
print("{0}\n".format(response.json()))
```

3. See the result that is similar to the following *abbreviated* example.

```
{
        "rows": [{
                        "value": {
                                "rev": "1-b2c48b89f48f1dc172d4db3f17ff6b9a"
                        },
                        "id": "14746fe384c7e2f06f7295403df89187",
                        "key": "14746fe384c7e2f06f7295403df89187",
                        "doc": {
                                "temperatureField": 10,
                                "descriptionField": "cold",
                                "numberField": 6,
                                "nameField": "six",
                                "_id": "14746fe384c7e2f06f7295403df89187",
                                "_rev": "1-b2c48b89f48f1dc172d4db3f17ff6b9a"
                        }
```

Show more

Step 6: Deleting the database

1. Delete the database.

```
try :
    client.delete_database(databaseName)
except CloudantException:
    print("There was a problem deleting '{0}'.\n".format(databaseName))
else:
    print("'{0}' successfully deleted.\n".format(databaseName))
```

2. Review the basic error handling that was included to illustrate how problems can be caught and addressed.

Step 7: Closing the connection to the service instance

1. Disconnect the Python client application from the service instance.
2. Run the disconnect command.

```
client.disconnect()
```

Execute the complete Python script

This script is the complete Python script for steps 2, 3, and 4. When you run the script, it connects to your service instance, creates the database, stores a small set of data in the database, and creates JSON documents.

1. In the code example in the next step, replace the values for `serviceUsername`, `servicePassword`, and `serviceURL` with the values from your service credentials.

   For more information about where to find your credentials, see Locating your credentials.

2. Copy the following script into a text editor and name it `demo.py`.

```
#!/usr/bin/env python

# Connect to service instance by running import statements.
from cloudant.client import Cloudant
from cloudant.error import CloudantException
from cloudant.result import Result, ResultByKey

# Add credentials to authenticate to the service instance.
serviceUsername = "apikey-v2-58B528DF5397465BB6673E1B79482A8C"
servicePassword = "680b037145f9dc8ef9e6a6d8b480783cbc1d1c12e71a0f4ced6b1eee30a243cd"
serviceURL = "serviceURL = "https://0c869093-c3ee-4a3f-bcec-00f01c8df8d8-
bluemix.cloudantnosqldb.appdomain.cloud""
databaseName = "databasedemo"

# Define sample data.
```

Show more

3. From the command line, run `demo.py` by typing a command similar to the following one.

```
python3 demo.py
```

Once you run the script, return to Step 5. Retrieving data to complete the tutorial.