Application Development on Db2 with Python and Native REST APIs

Sowmya Kameswaran, IBM

New England Db2 Users Group

Platform: Db2 for z/OS

What is python

- Interpreted programming language
- First released in 1991 by Guido van Rossum
- Designed to be simple (See “The Zen of Python”)

```
print('Hello, world!')
```

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
```

Why Python

- Python continuously ranks in the top 3 over popular programming languages
  - TIOBE (May 2022): #1, overtaking C
  - PYPL (May 2022): #1, almost 2x Java
  - Jobs available: #3 behind C and SQL
- It is widely taught at educational institutions
- Most new graduates have probably been exposed to Python

Why Python, cont’ed

- Widely used in
  - Machine Learning
  - Data Science
  - Infrastructure provisioning
    - Ansible
  - Scientific computing
  - Web sites
    - Google, Instagram, Spotify, Netflix, Dropbox, Uber, …

Typical use cases for Python on z/OS

- Productivity aid programs; internal services
  - Replacement for REXX for “young” programmers

- Pipelines, automated testing, ..

…

- Ansible (infrastructure provisioning)
- Machine learning and analytics
- Reporting

Get python for z/OS

- Order IBM Open Enterprise SDK for Python from IBM ShopZ (optional support available)

  or

- Download PAX https://www.ibm.com/products/open-enterprise-python-zos/details

  or

- Rocket OpenApp Dev for z
  - https://www.rocketsoftware.com/openappdev-for-z
  - Conda install or SMP/E (with support)
  - Includes IBM Python
  - Also includes git and many other open source tools for application developers

Python and databases

- Python DB-API v2.0 specification
- “Identical” code to access databases across all platforms & database vendors

- Python_ibmdb project
  - Maintained by Db2 Connect development team
  - Uses ODBC drivers

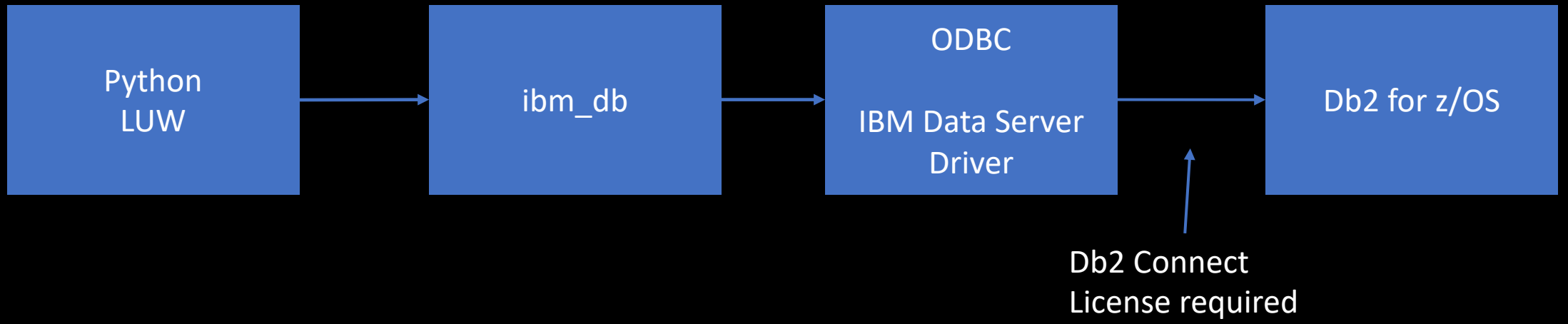
- ibm_db: python drivers for IBM Db2
- ibm_db_dbi: implementation of the DB-API v2.0 specification
- You get both then you install ibm_db

Source code:

API doc

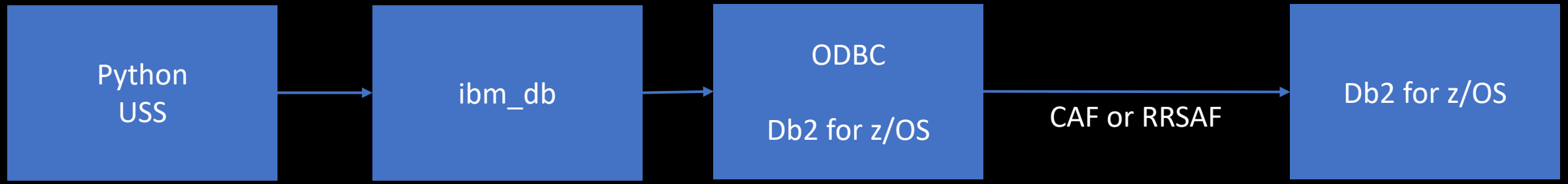
ibm_db – distributed platform

- On distributed platform:
  - Uses Db2 Connect CLI drivers (IBM Data Server Driver)
  - Connect to Db2 for z/OS using DRDA
  - Need Db2 Connect for z/OS license to connect to Db2 for z/OS



ibm_db – z/OS

- On z/OS:
  - CAF (Call Attachment Facility)
  - RRSAF (Resource Recovery Services attachment Facility)
  - Default plan DSNACLI
  - Bind ODBC plans & packages: HLQ.SDSNSAMP(DSNTIJCL)
  - No additional licenses required !



Installing ibm_db on z/OS

- No pre-compiled binaries available like distributed platform
- Pip3 install ibm_db will
  - Download source
  - Compile ibm_db C modules (IBM z/OS C/C++ compilers required)
  - Install into your site-packages library
- Important prereqs:
  - IBM Python V3.8.3+ (from IBM or Rocket z Open App Dev)
  - Db2 V12 PTF UI72859: Python related bugfixes for ODBC driver
    No PTFs required for Db2 V13
- More details here: https://github.com/ibmdb/python-ibmdb/blob/master/install.md

First ibm_db program

- Set STEPLIB so Python can find the ODBC Db2 z/OS drivers
- Create an ODBC ini file
- Set environment variable DSNAOINI to point to the ODBC ini file

```
(ibm_python_env) $ export
STEPLIB=PDS1.SDSNEXIT:HLQ.SDSNLOD2:HLQ.SDSNLOAD
(ibm_python_env) $ export DSNAOINI=="$HOME/ODBC_PDS1_CAF"
```

https://www.ibm.com/support/knowledgecenter/SSEPEK_12.0.0/odbc/src/tpc/db2z_hdckeyw.html

Name of ODBC ini file is arbitrary
With lots of SSIDs I just prefer to have SSID in the name

ODBC_PDS1_CAF file:

```
[COMMON]
MVSDEFAULTSSID=RS01PDS1
FLOAT=IEEE
CURRENTAPPENSCH=ASCII
APPLTRACE=0
APPLTRACEFILENAME=/u/xxxxx/odbc_trace
[RS01PDS1]
AUTOCOMMIT=1
MVSATTACHTYPE=CAF
PLANNAME=DSNACLI
```

First ibm_db program, cont'ed

- Simple program using ibm_db

```
import ibm_db
conn=ibm_db.connect('','','')

if conn:
  sql = "SELECT * FROM SYSIBM.SYSTABLES FETCH FIRST 10 ROWS ONLY"
  stmt = ibm_db.exec_immediate(conn, sql)
  result = ibm_db.fetch_both(stmt)
  while( result ):
    print(result[1].strip()+"."+result[0].strip())
    result = ibm_db.fetch_both(stmt)

  ibm_db.close(conn)
```

First ibm_db program, cont’ed

```
(ibm_python_env) $ python3 simple-test.py
SYSIBM.DBDR
SYSIBM.IPLIST
SYSIBM.IPNAMES
SYSIBM.LOCATIONS
SYSIBM.LULIST
SYSIBM.LUMODES
SYSIBM.LUNAMES
SYSIBM.MODESELECT
SYSIBM.SCTR
SYSIBM.SPTR
```

Ok, what if I am running on distributed platform?

- The code is the same! – except for the connect
- On distributed platform you must provide user ID/password as it is not using local attach

```
import ibm_db
conn=ibm_db.connect('DSN=<location>;uid=xxxxx;pwd=yyyyyy','','')
)
```

Bonus slide: what about SSL?

- Of course!
- I haven't configured an ODBC entry for my SSL connection, so I'll use the alternate connection string syntax

```
import ibm_db
conn=ibm_db.connect('database=<location>;hostname=<hostname>;port=3
713;protocol=tcpip;uid=xxxx;pwd=yyyy;security=ssl;sslServerCertific
ate=<path>\\pds1.ca.cert','','')
```

Calling a stored procedure

- Issue a REORG from a python program
- Call SYSPROC.DSNUTILU / SYSPROC.DSNUTILV

- ibm_db.callproc – call procedure
- ibm_db.fetch_assoc – fetch row from result set
- ibm_db.next_result – next result set

Calling a stored procedure, cont’ed

```
utility_id = 'REORGDEF'
restart = 'NO'
utstmt = 'TEMPLATE FCOPY DSN RSTEST.IC.&SSID..&SN..P&PA..&UNIQ. UNIT 3390 TEMPLATE
REC1 DSN &USERID..&SSID..UNLD.&DB..&TS. UNI
T 3390 DISP(OLD,CATLG,CATLG) TEMPLATE WORK1 DSN &USERID..&SSID..SYSUT1 UNIT 3390
TEMPLATE WORK2 DSN &USERID..&SSID..SORTOUT  U
NIT 3390 REORG TABLESPACE TS5941.GLWSDPT LOG NO SORTDATA SHRLEVEL CHANGE
KEEPDICTIONARY STATISTICS TABLE(ALL) SAMPLE 60 INDEX(
ALL) COPYDDN(FCOPY) UNLDDN(REC1), WORKDDN(WORK1,WORK2)'
retcode = 0

stmt, utility_id,restart,utstmt,retcode = \
ibm_db.callproc(conn, 'SYSPROC.DSNUTILU', (utility_id,restart,utstmt,retcode) )
```

Calling a stored procedure, cont’ed

```
while stmt1:
   # result set available

   row = ibm_db.fetch_assoc(stmt1)
   while row:
      print(row[“TEXT”])
      row = ibm_db.fetch_assoc(stmt1)

   stmt1 = ibm_db.next_result(stmt)
```

Calling a stored procedure, cont’ed

```
(test2) $ python call-sp.py RS01PDS1 TS5941.GLWSDPT
Running REORG for tablespace TS5941.GLWSDPT on DSN=RS01PDS1
Successfully connected to DSN=RS01PDS1
1DSNU000I    297 11:43:16.65 DSNUGUTC - OUTPUT START FOR UTILITY, UTILID = REORGDEF
 DSNU1045I   297 11:43:16.66 DSNUGTIS - PROCESSING SYSIN AS UNICODE UTF-8
0DSNU050I    297 11:43:16.66 DSNUGUTC -  TEMPLATE FCOPY DSN RSTEST.IC.&SSID..&SN..P&PA..&UNIQ. UNIT 3390
 DSNU1035I   297 11:43:16.67 DSNUJTDR - TEMPLATE STATEMENT PROCESSED SUCCESSFULLY
[snip]
 DSNU610I  !PCA1 297 11:43:18.33 DSNUSUCD - SYSCOLDIST CATALOG UPDATE FOR TS5941.GLWXDPT2 SUCCESSFUL
 DSNU620I  !PCA1 297 11:43:18.33 DSNUSEOF - RUNSTATS CATALOG TIMESTAMP = 2020-10-23-11.43.17.393547
 DSNU3357I   297 11:43:18.71 DSNUGUTC - MAXIMUM SORT AMOUNT ESTIMATION VARIATION WAS 0 PERCENT
 DSNU3355I   297 11:43:18.71 DSNUGUTC - TOTAL SORT MEMORY BELOW THE BAR: OPTIMAL 24 MB, USED 24 MB
 DSNU010I    297 11:43:18.73 DSNUGBAC - UTILITY EXECUTION COMPLETE, HIGHEST RETURN CODE=4
```

Calling a stored procedure – ADMIN_COMMAND_DB2

```
Issue command on DSN=RS01PDS1: -DISPLAY GROUP
Succesfully connected to DSN=RS01PDS1
Result set
DSN7100I  !PCA1 DSN7GCMD
*** BEGIN DISPLAY OF GROUP(PDS1    ) CATALOG LEVEL(V12R1M509)
                 CURRENT FUNCTION LEVEL(V12R1M510)
                 HIGHEST ACTIVATED FUNCTION LEVEL(V12R1M510)
                 HIGHEST POSSIBLE FUNCTION LEVEL(V12R1M510)
                 PROTOCOL LEVEL(2)
                 GROUP ATTACH NAME(PDS1)
------------------------------------------------------------------
DB2          SUB                       DB2    SYSTEM    IRLM
MEMBER   ID  SYS  CMDPREF   STATUS     LVL    NAME      SUBSYS IRLMPROC
-------- --- ---- --------  -------- ------ --------  ----   --------
PCA1       1 PCA1 !PCA1     ACTIVE   121510 RS01      IR05   PCA1IRLM
PCB1       2 PCB1 !PCB1     QUIESCED 121510 RS02      IR06   PCB1IRLM
------------------------------------------------------------------
SCA   STRUCTURE SIZE:    16384 KB, STATUS= AC,   SCA IN USE:     7 %
LOCK1 STRUCTURE SIZE:    64512 KB
NUMBER  LOCK ENTRIES:    16777216
NUMBER  LIST ENTRIES:       36345, LIST ENTRIES  IN USE:         15
SPT01 INLINE LENGTH:        32138
*** END DISPLAY OF GROUP(PDS1    )
DSN9022I  !PCA1 DSN7GCMD 'DISPLAY GROUP ' NORMAL COMPLETION
```

```
python3 disgroup.py
Issue command: -DISPLAY GROUP
Successfully connected to Db2
Member Status   Version LPAR
PCA1   ACTIVE   121510  RS01
PCB1   QUIESCED 121510  RS02
```

Python and all the libraries

- Big advantage of python is 1000s of libraries
  - Many work out-of-the-box
  - Some require porting of C-code (like ibm_db)

- Interesting libraries
  - ibm_db – of course ☺
  - Requests – for making REST calls
  - Flask – for making web apps
  - Jupyter / pandas / matplotlib
  - XLSX – for creating Excel spreadsheets

Calling a Db2 native REST service

```
import requests
import sys

if len(sys.argv) < 3:
   print("Usage: %s <userid> <password>
   exit(-1)

userid=sys.argv[1]
password=sys.argv[2]

collection='TS5941'
service='TESTIDC4'

url = 'https://<hostname>:<port>/service
auth = requests.auth.HTTPBasicAuth(useri

resp = requests.get(url,auth=auth,verify
print(resp)
print(resp.json())
```

```
<Response [200]>
{'TESTIDC4': {'serviceName': 'TESTIDC4', 'serviceCollectionID':
'TS5941', 'version': 'V1', 'isDefaultVersion': True,
'serviceProvider': 'db2service-1.0', 'serviceDescription': 'TEST
IDENTICALLY NAMED COLUMNS 4', 'alternateServiceURL':
'https://rs01.rocketsoftware.com:3713/services/TS5941/TESTIDC4',
'serviceURL':
'https://rs01.rocketsoftware.com:3713/services/TS5941/TESTIDC4/V1
', 'serviceStatus': 'started', 'RequestSchema': {},
'ResponseSchema': {'$schema': 'http://json-schema.org/draft-
04/schema#', 'type': 'object', 'properties': {'ResultSet Output':
{'type': 'array', 'items': {'type': 'object', 'properties':
{'CREATOR': {'type': 'string', 'maxLength': 128, 'description':
'VARCHAR(128)'}, 'NAME': {'type': 'string', 'maxLength': 128,
'description': 'VARCHAR(128)'}}, 'required': ['CREATOR', 'NAME',
'NAME'], 'description': 'ResultSet Row'}}, 'StatusDescription':
{'type': 'string', 'description': 'Service invocation status
description'}, 'StatusCode': {'type': 'integer', 'multipleOf': 1,
'minimum': 100, 'maximum': 600, 'description': 'Service
invocation HTTP status code'}}, 'required': ['StatusDescription',
'StatusCode', 'ResultSet Output'], 'description': 'Service
TESTIDC4 invocation HTTP response body'}}}
```

Tune a query using IBM Db2 Tuning Services

- Db2 Accessories Suite for z/OS – Database Services Expansion Pack Feature (no-charge, available from IBM ShopZ)
- UI available through VS Code IBM Db2 Developer Extension and web-based IBM Db2 Administration Foundation

- But you can also call tuning services from a program or pipeline
- This example
  - Logon to Tuning Services
  - Submit Runstats advisor “job” (job = async task in the Tuning Service server)
  - Wait for “job” to complete
  - Show result

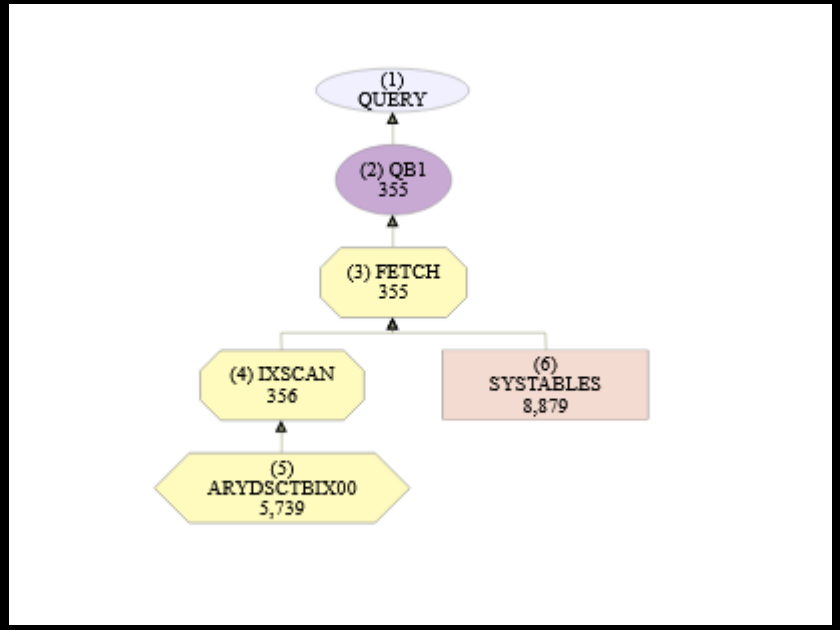
Tune a query using IBM Db2 Tuning Services

```
#Submit Runstats advisor
url = baseUrl + '/tuningservice/v1/sa'
resp = requests.post(url,json=vebody,verify=verify,headers=headers)
job_id=resp.json()['job_id']
```

```
Logon TMS
<Response [200]>
Submit Runstats advisor
<Response [202]>
TMSJOB0002I: The job has been submitted with Job_ID
17143072140705177 60.
Job ID 17143072140705177 60 status RUNNING
Job ID 17143072140705177 60 status COMPLETED
Runstats advisor results
<Response [200]>
Runstats recommended:
 RUNSTATS TABLESPACE DSNDB06.SYSTSTAB TABLE(SYSIBM.SYSTABLES)
COLUMN(ALTEREDTS,ARCHIVING_SCHEMA,ARCHIVING_TABLE,AUDITING,CARD,
CHECKFLAG,CHECKRID,CLUSTERTYPE,COLCOUNT,CONTROL,
CREATEDBY,CREATEDTS,CREATOR,DBID,DBNAME, …
```

Tune a query using IBM Db2 Tuning Services

```
Logon TMS
<Response [200]>
Submit Visual Explain advisor
<Response [202]>
TMSJOB0002I: The job has been submitted with Job_ID
17438416726476759O4.
Job ID 17438416726476759O4 status RUNNING
Job ID 17438416726476759O4 status COMPLETED
Visual Explain advisor results
<Response [200]>
URL for Visual Explain:
https://rs01.rocketsoftware.com:9944/tuningservice/v1/ve/viewer/1e7df
f89-d2f1-46ad-b6f2-1412f500637e
```



Upgrade EXPLAIN tables – inspired from IDUG-L

- Db2 for z/OS ships stored procedure ADMIN_EXPLAIN_MAINT to create, upgrade and create alias for explain tables

- Alternatively use the Db2 Tuning Service REST endpoint

```
Logon TMS
<Response [200]>
Run Explain table maintenance
<Response [200]>
{'code': 200, 'status': 'success', 'payload': {'result':
[{'ALIAS_CREATED': '0', 'TB_NOT_UNICODE': '0', 'TB_EXAMINED':
'0', 'TS_DROPPED': '0', 'AUX_CREATED': '5', 'TB_DROPPED': '0',
'TB_CREATED': '23', 'DB_CREATED': '1', 'TB_ALTERED': '0',
'IX_CREATED': '26', 'TS_CREATED': '28', …
```

```
explaintbbody = {
  "action": "CREATE",
  "auth_id": "TS5941",
  "bp16kblob": "BP16K0",
  "bp32kblob": "BP32K",
  "bp4kblob": "BP0",
  "bp8kblob": "BP8K0",
  "connection": "profile1",
  "database_name": "TS5941EE",
  "ixbufferpool": "BP0",
  "managealias": "NO",
  "mode": "RUN",
  "schema_alias": "",
  "schema_name": "TS5941EE",
  "storagegroup": "DB2EA",
  "storagegroup_idx": "DB2EA",
  "tableset": "ALL",
  "ts16kbufferpool": "BP16K0",
  "ts32kbufferpool": "BP32K",
  "ts4kbufferpool": "BP0",
  "ts8kbufferpool": "BP8K0"
}
url = baseUrl + '/tuningservice/v1/explaintb'
resp =
requests.post(url,json=explaintbbody,verify=verify,h
eaders=headers)
```

Manage Db2 Analytics Accelerator

- Uses IBM Db2 Analytics Accelerator Administration Services
  - Available from IBM Fix Central https://ibm.biz/BdfWns
- UI available through web-based IBM Db2 Administration Foundation
- But you can also call tuning services from a program or pipeline
- This example
  - Logon to Accelerator Admin Services
  - Load a table

Load table in Db2 Analytics Accelerator

```
# Generate JWT auth token
print("Logon Accel Admin Svc")
url = baseUrl + '/token'
resp =
requests.post(url,json=loginbody,verify=v
erify)
print(resp)
token=resp.json()['result']['token']
#print(token)
headers={'Authorization':'Bearer '+token}


print(f"Load table {schema}.{table}")
url = baseUrl + '/table/partition/load'
resp =
requests.post(url,json=loadbody,verify=ve
rify,headers=headers)
print(resp)
print(resp.json())
```

```
Logon Accel Admin Svc
<Response [201]>
Load table TS5941.TESTMD
<Response [200]>
{'result': {'name': 'IDAAZ30'}, 'message': [], 'status':
'SUCCESS'}
```

Jupyter install on z/OS

```
$ conda create -n jupyter_env notebook pip xlc-wrapper
cython "panda*" matplotlib matplotlib
$ conda activate jupyter_env

$ pip install ibm_db

$ jupyter notebook --generate-config

[edit config – I added my SSL cert and change port#]

$ mkdir my-notebooks
$ cd my-notesbooks
$ jupyter notebook
```


```
[I 02:31:15.307 NotebookApp] Notebook authentication will be used.  The hashed password should be in the configuration
file.
[I 02:31:15.977 NotebookApp] Serving notebooks from local directory: /u/userid/devel/jupyter-notebooks
[I 02:31:15.977 NotebookApp] The Jupyter Notebook is running at:
[I 02:31:15.977 NotebookApp]
https://rs01.rocketsoftware.com:8888/?token=a21db5d98377c28af0527d3157ac9444e0d9688549f693dc
[I 02:31:15.977 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 02:31:15.983 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///u/userid /.local/share/jupyter/runtime/nbserver-131433-open.html
    Or copy and paste one of these URLs:
        https://rs01.rocketsoftware.com:8888/?token=a21db5d98377c28af0527d3157ac9444e0d9688549f693dc
```

jupyter Db2z notebook Last Checkpoint: 03/22/2021 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help Trusted Python 3

Run Code

In [1]:
```
%run db2.ipynb
```

```
Db2 Extensions Loaded.
```

In [2]:
```
%sql CONNECT TO RS01PDS1
```

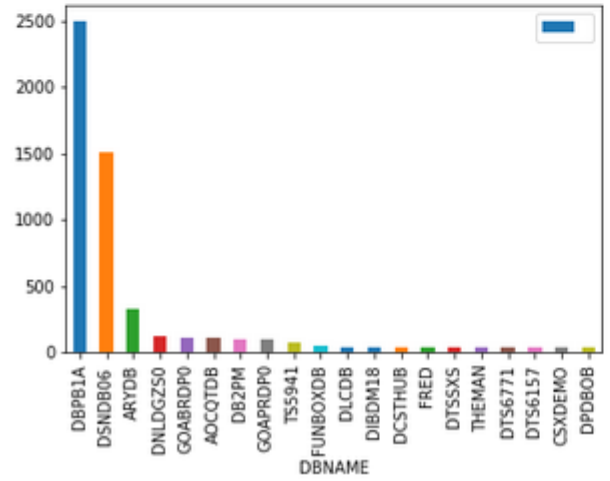
```
Connection successful.
```

In [3]:
```
%sql select getvariable('SYSIBM.SSID') as ssid,  getvariable('SYSIBM.DATA_SHARING_GROUP_NAME') as dsg from sysibm.
```

Out[3]:

	SSID	DSG
0	PCA1	PDS1

In [4]:
```
%sql -pb SELECT dbname,count(*) FROM SYSIBM.SYSTABLES group by dbname order by 2 desc fetch first 20 rows only
```



Get db2.ipynb from https://github.com/IBM/db2-jupyter/blob/master/db2.ipynb

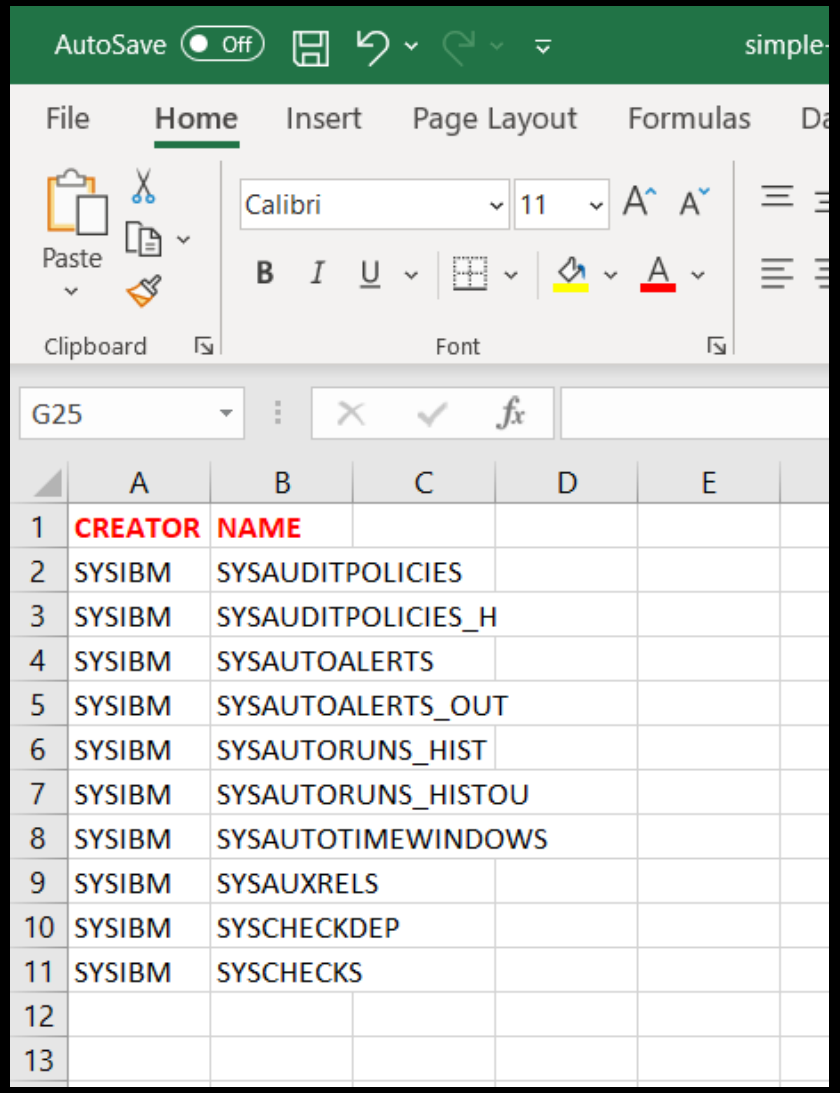
Write excel file from Db2 query

- Report generation from SQL statement
- Output written as XLSX

- Uses xlsxwriter library

```
import ibm_db
import sys
import xlsxwriter

print(ibm_db.__version__)
conn=ibm_db.connect('','','')

workbook = xlsxwriter.Workbook('simple-excel.xlsx')
worksheet = workbook.add_worksheet()

cell_format0 = workbook.add_format({'bold': True, 'font_color':
'red'})

row = 0
worksheet.write(row,0,"CREATOR",cell_format0)
worksheet.write(row,1,"NAME",cell_format0)

row = row + 1

if conn:
  sql = "SELECT CREATOR,NAME FROM SYSIBM.SYSTABLES WHERE CREATOR =
'SYSIBM' AND NAME LIKE 'SYS%' FETCH FIRST 10 ROWS ONLY"
  stmt = ibm_db.exec_immediate(conn, sql)
  result = ibm_db.fetch_both(stmt)
  while( result ):
    worksheet.write(row,0,result[0].strip())
    worksheet.write(row,1,result[1].strip())
    row = row + 1

    result = ibm_db.fetch_both(stmt)

  ibm_db.commit(conn)
  ibm_db.close(conn)

workbook.close()
```

Write excel file from Db2 query



	A	B	C	D	E
1	CREATOR	NAME			
2	SYSIBM	SYSAUDITPOLICIES			
3	SYSIBM	SYSAUDITPOLICIES_H			
4	SYSIBM	SYSAUTOALERTS			
5	SYSIBM	SYSAUTOALERTS_OUT			
6	SYSIBM	SYSAUTORUNS_HIST			
7	SYSIBM	SYSAUTORUNS_HISTOU			
8	SYSIBM	SYSAUTOTIMEWINDOWS			
9	SYSIBM	SYSAUXRELS			
10	SYSIBM	SYSCHECKDEP			
11	SYSIBM	SYSCHECKS			
12					
13					

Bonus topic – interact with z/OS

- Datasets
- Jobs
- Other z/OS services

- Very little native support
- IBM Z Open Automation Utilities

Interact with z/OS

- Use os.system, os.popen, subprocess.open

```
import os

stream = os.popen('uname')
output = stream.read()
if output.startswith("OS/390"):
   print("Yeah, running on z/OS!")
```

```
import os

stream = os.popen('tsocmd "alloc da(from.python) space(1,1) new"')
output = stream.read()

print(output)
```

Interact with z/OS

- IBM Z Open Automation Utilities
  - No charge
  - Wraps MVS utilities like IEBCOPY, IDCAMS, IKJEFT01, etc
  - Wraps SDSF REXX API
  - ZOAU 1.2 rewrote some functions in C for better performance
  - Supports shell scripts and Python

  - https://www.ibm.com/docs/en/zoau/1.2.0

IBM Z Open Automation Utilities

Example: run query and write to sequential dataset


```
import ibm_db
import sys
from zoautil_py import datasets

print(ibm_db.__version__)
conn=ibm_db.connect('','','')
print("Connected",conn)

datasetname = "TS5941.PYTHON.REPORT.TXT"

if datasets.exists(datasetname):
   datasets.delete(datasetname)

datasets.create(datasetname, type="SEQ",
   record_length=133, record_format="FB",
   primary_space="10k")

content = ""
if conn:
  sql = "SELECT * FROM SYSIBM.SYSTABLES …"
  stmt = ibm_db.exec_immediate(conn, sql)
  result = ibm_db.fetch_both(stmt)
  while( result ):
    content = content +
       result[1].strip()+"."+result[0].strip() + "\n"
    result = ibm_db.fetch_both(stmt)

  ibm_db.close(conn)

datasets.write(datasetname,content=content)
```

Interact with z/OS

- z/OSMF REST APIs
- Also expensive to call
- Will work on both z/OS and distributed

```
import requests
url =
'https://rs01.rocketsoftware.com:11443/zosmf/restjobs/jobs?prefix=IZPS*&owner=*'
auth = requests.auth.HTTPBasicAuth('userid', 'password')
print(auth)

resp = requests.get(url,auth=auth,verify=False)
if resp.ok:
   print("%8s %8s %8s %8s %8s" % \
      ("Jobname", "JobId", "Owner", "Status","Retcode"))
   for j in resp.json():
      print("%8s %8s %8s %8s %8s" % \
         ( j["jobname"], j["jobid"], j["owner"],
j["status"], j["retcode"] ))
```

Interact with z/OS

- z/OSMF REST APIs
- Also expensive to call
- Will work on both z/OS and distributed

```
import requests
url =
'https://rs01.rocketsoftware.com:11443/zosmf/restjobs/jobs?prefix=IZPS*&owner=*'
auth = requests.auth.HTTPBasicAuth('userid', 'password')
print(auth)

resp = requests.get(url,auth=auth,verify=False)
                                                    \
                                                    ", "Status","Retcode"))

                                                     % \
                                                    "], j["owner"],
```

```
Jobname    JobId    Owner    Status  Retcode
 IZPSRV STC01178   DOESTC   OUTPUT  CC 0000
 IZPSRV STC04386   DOESTC   OUTPUT  CC 0000
IZPSRVP STC04120   DOESTC   OUTPUT  CC 0000
IZPSRVP STC04121   DOESTC   OUTPUT  CC 0000
IZPSRVP STC04113   DOESTC   OUTPUT  CC 0000
IZPSRVP STC04119   DOESTC   OUTPUT  CC 0000
IZPSRVP STC04112   DOESTC   OUTPUT  CC 0000
IZPSRVP STC04111   DOESTC   OUTPUT  CC 0000
IZPSRVP STC04110   DOESTC   OUTPUT  CC 0000
 IZPSRV STC04387   DOESTC   ACTIVE     None
IZPSRVP STC04122   DOESTC   ACTIVE     None
IZPSRV2 STC07395   DOESTC   ACTIVE     None
```

Interact with z/OS

- CFFI
  - Create python wrapper over C libraries, e.g., “access()” to check if file exists

Conclusion

- Python is popular
- Python can talk to Db2 for z/OS
- Python can run on z/OS (read: USS)
- Python can run on z/OS *and* now also talk to Db2 for z/OS

Install on LPAR w/o internet access

- Pip download ibm_db – on your laptop
- Transfer to USS; unzip
- cd python-ibmdb-v.r.m/IBM_DB/ibm_db
- chtag –R –tc ISO8859-1 .
- Pip install .

Troubleshooting (1)

- “Exception” on connect without detailed error messages
- Most likely an encoding error with the ODBC ini file
- MUST BE encoded in IBM-1047
- MUST BE tagged IBM-1047
- “text” tag must be off
- Check file contents, e.g., [and] (hex 0xAD and 0xBD)

```
Traceback (most recent call last):
  File "simple-test.py", line 5, in <module>
    conn=ibm_db.connect('','','')
Exception
```

Thank You

Speaker: Sowmya Kameswaran

Company: IBM

Email Address: skamesw@us.ibm.com

Session Title: Application Development on Db2 with Python and Native REST APIs