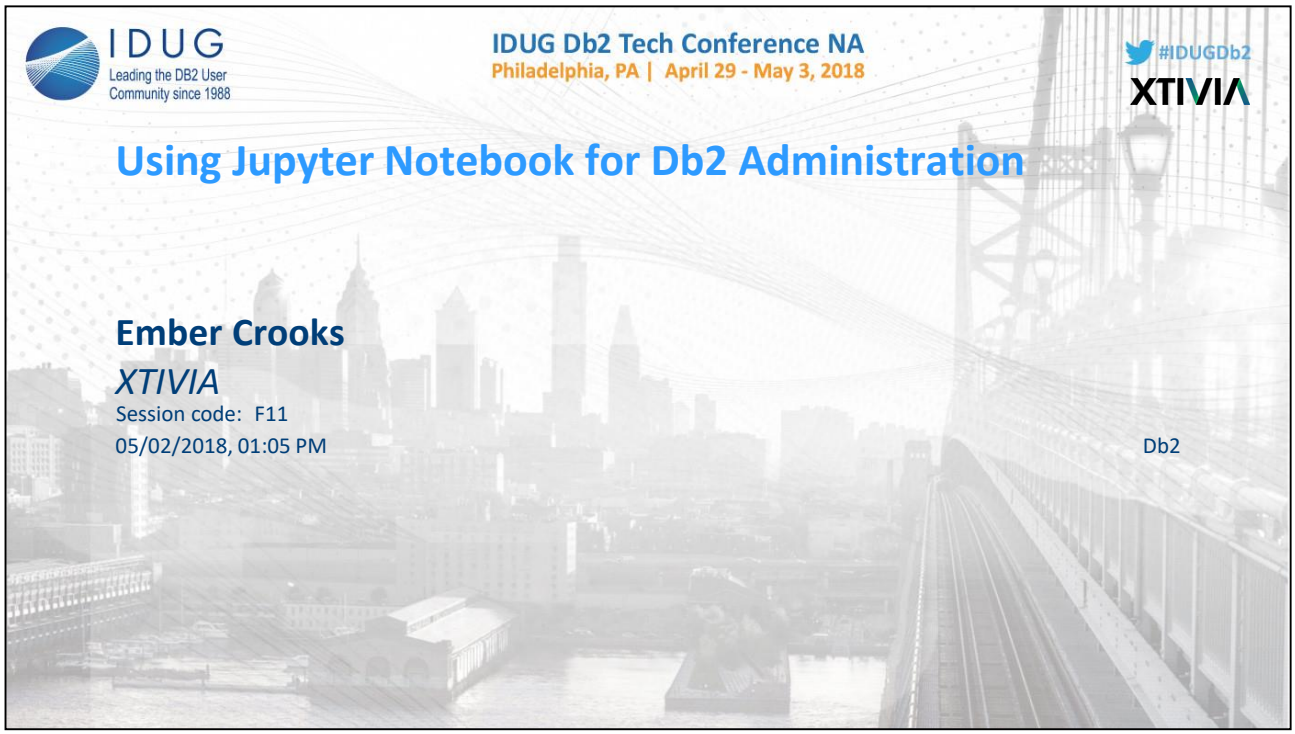


There are so many new technologies in the data world. How can a Db2 DBA keep up with them? We can use some of them to our own advantage. Jupyter Notebook is a tool at the heart of data science. Come to this session to learn how you can use Jupyter Notebook in your day-to-day Db2 DBA tasks, for team documentation, and providing details on DB2 or SQL performance in formats that non-DBAs find convincing and compelling.

Learn how to use Jupyter Notebook to write your own Db2 Snapshot!

The concepts presented are largely cross platform, but the speaker's experience focuses on LUW, and a Db2 LUW database is used for all examples.

Agenda

- What Jupyter Notebook is and how to set it up
- Use SQL Magic to connect to a Db2 database and manipulate data
- Investigate how SQL Magic works with Db2 and what you might need other tools for
- Experience how team documentation or troubleshooting procedures can benefit from a Jupyter Notebook format

Introduction to Jupyter Notebook

What is Jupyter Notebook?

- Open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and explanatory text.
- Heavily used in Data Science
- Supports a large number of programming languages, including SQL
- Requires Python
- Easiest to install by installing Anaconda

4

http://jupyter.org/
Anaconda: https://www.continuum.io/downloads

Why Use Jupyter Notebook when Administering Db2?

- Experience with tools that developers and data scientists are using enhances communication and our skill sets as DBAs
- Combination of explanatory text and in-line execution of code (SQL) very powerful for learning and understanding
- Easy visualization of query results for analysis particularly powerful
  - Just one line after a query can generate a line, pie, or bar graph

5

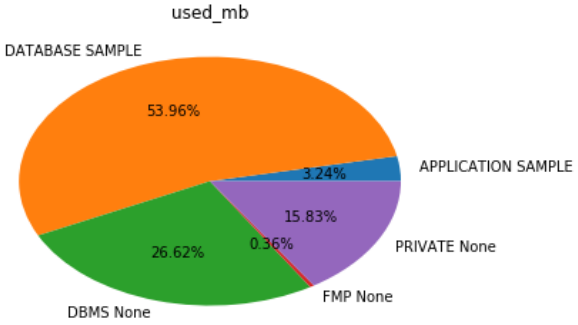
As a database administrator, I sometimes find it hard to learn the hot new tools. Why bother, when the good old command line is always there? However, it's nice to keep skills up to date and learn some of the development techniques. This is my number one reason for working with Jupyter Notebook.

As a blogger and a speaker, I find the combination of formatted explanatory text and in-line executable code very useful. I love providing people with the SQL to play with themselves from my presentations.

Sometimes, visualizing the data is very useful, even for someone used to working at the command line. A pie chart can be very useful for visualizing the balance of current memory allocation, for example. Adding charts an graphs using Jupyter Notebook can be as simple as a single line or two of code.

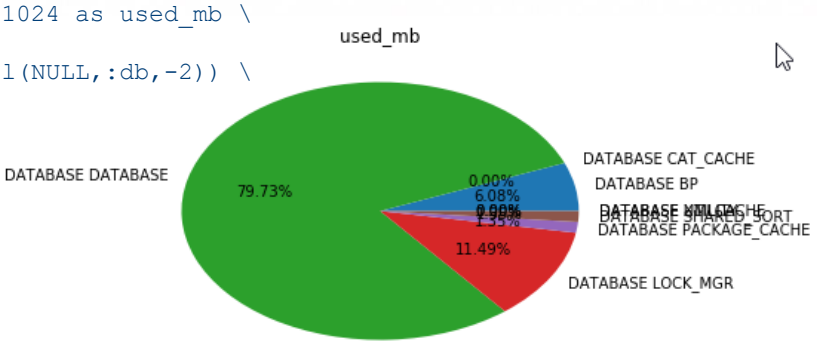
Jupyter Notebook Example: Viewing DB2 Memory Allocation (Instance-Level)

```
inst_memory=%sql select \
  memory_set_type \
  , db_name \
  , sum(memory_set_used)/1024 as used_mb \
from \
  table(mon_get_memory_set(NULL,NULL,-2)) \
group by
  memory_set_type
  , db_name

inst_memory.pie()
```



Jupyter Notebook Example: Viewing DB2 Memory Allocation (Database-Level)

```
db_memory=%sql select \
  memory_set_type \
  , memory_pool_type \
  , sum(memory_pool_used)/1024 as used_mb \
from
  table(mon_get_memory_pool(NULL,:db,-2)) \
where db_name=:db \
group by \
  memory_set_type \
  , memory_pool_type

db_memory.pie()
```



Jupyter Notebook Terms (1)

- Application
  - Web application that allows editing and running notebook document via web browser
  - Can run on a local laptop/desktop or a remote server
- Kernel
  - Computational engine that execute the code contained in the notebook
  - ipython kernel executes python – kernels for other languages exist
  - Each running notebook has a different instance of a Kernel

These slides are available for reference.

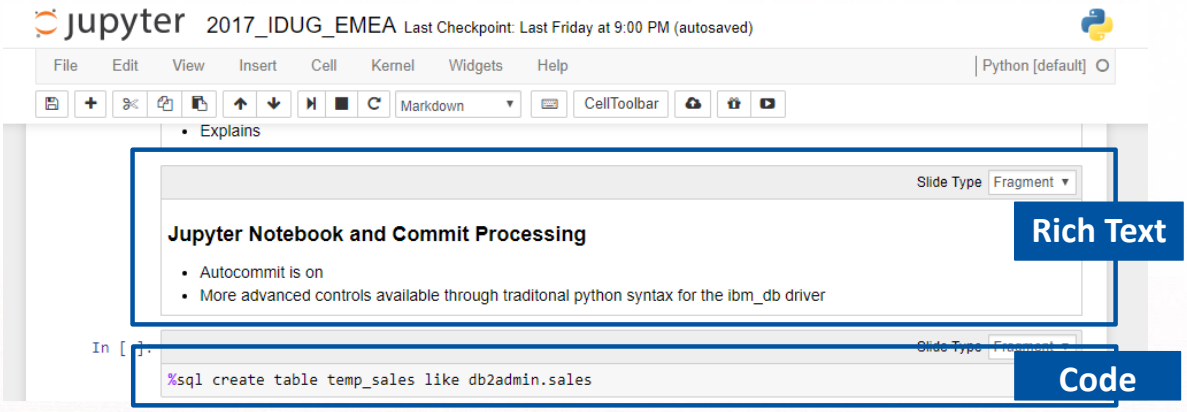
Jupyter Notebook Dashboard

- Notebook Dashboard
  - Shown in browser when you launch Jupyter Notebook
  - Features similar to a file manager
  - Used to open notebooks and manage the running kernels

These slides are available for reference.

The Notebook

- Contains both computer code and rich text elements



Jupyter Notebook Terms(2)

- Cell
  - Portion of a notebook that is either markdown-formatted text or code
  - Each cell can be independently executed in any order, but convention dictates they be executed in order
- Magic
  - Called with a command line style syntax
  - Prefixed with '%'
  - Magic Functions work at the cell or line level
  - Use %lsmagic to list the available magic functions

These slides are available for reference.

Installing and Configuring Jupyter Notebook

Installing Jupyter Notebook on Windows

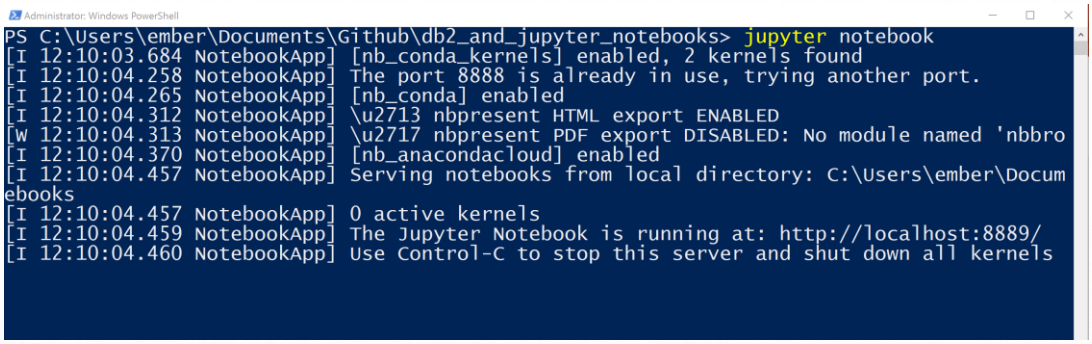
- Download Anaconda
- Install Anaconda, accepting the defaults
- Install at least a Db2 Client
- Download and install: http://landinghub.visualstudio.com/visual-cpp-build-tools
- Add Anaconda to your PATH (search control panel for “path”)
- Jupyter Notebook is likely to be installed on a DB2 client such as your laptop or a jump server

13

Instructions provided are for windows. Jupyter Notebook can also be installed on Linux or MacOS.

Launching Jupyter Notebook on Windows

- Db2 Command Window or PowerShell window with the Db2 environment variables set
- cd to directory where you store Jupyter Notebooks

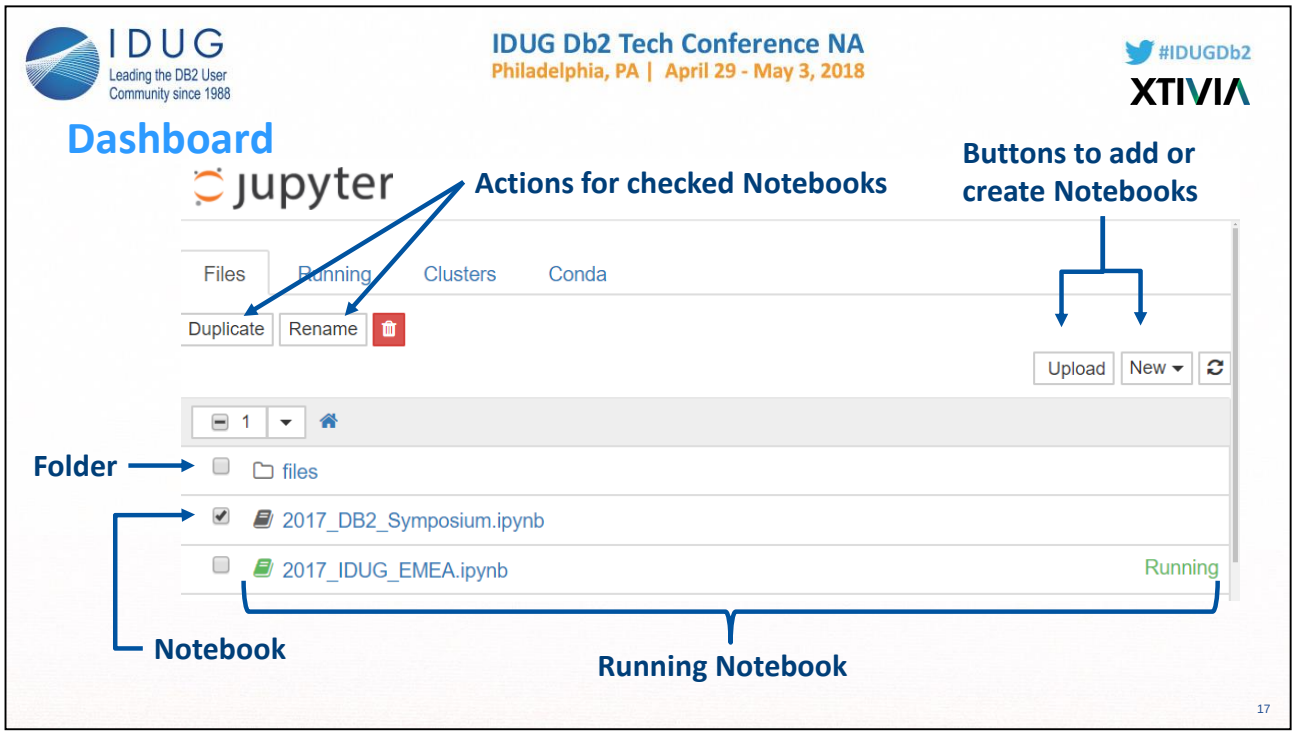


On Windows, it is easiest to launch Jupyter Notebook from a PowerShell or Db2 Command Window that has all of the Db2 Environment variables set. This means you have access to db2 commands as well as SQL.

Working with Jupyter Notebook

Presentation Source

- Much of the rest of this presentation will be done looking at Jupyter Notebooks live. The slides are presented as a reference for the material covered.
- All notebooks used are available at https://github.com/ecrooks/db2_and_jupyter_notebooks

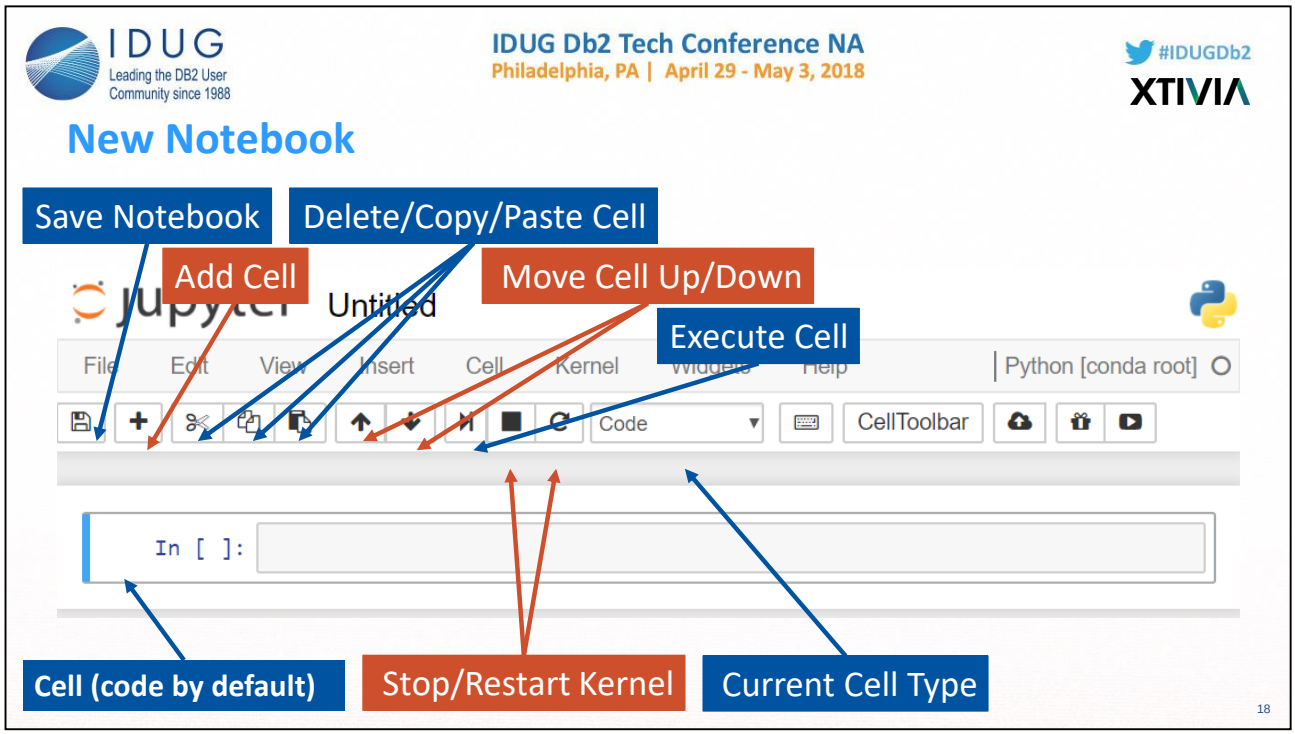


The Jupyter Notebook dashboard bears many similarities to a simple file manager. Each file or notebook has it's own line. Running notebooks are denoted by a green color and the "Running" keyword at the right.

Selection boxes allow the following actions on one or more files:

- Duplicate (make a copy)
- Rename
- Delete

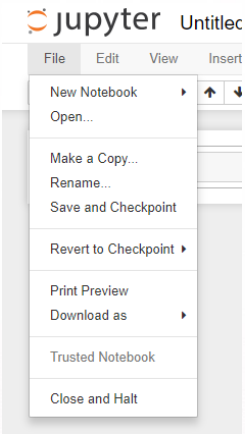
This is also where a new notebook is added or created.

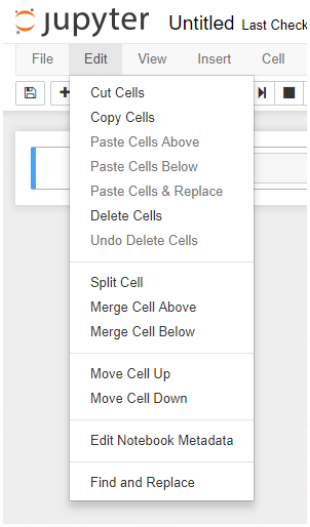


There is a lot going on in the toolbars within a notebook. At the top, the title ("Untitled" in this image) can be changed simply by clicking on the text and changing it. Graphic Icons allow for the most common actions to be in easy reach:

- **Save Notebook** to save changes to this notebook and create a checkpoint. Note that changes are automatically saved every few minutes, so if you're making changes you don't want to save, make sure you're working on a copy of a notebook or have a good checkpoint to go back to.
- **Add cell** – adds a cell in the notebook below the currently selected cell. You can also choose to add a cell above by choosing an option from the Cell menu
- **Delete cell** – deletes the currently selected cell, even if it has content
- **Copy cell** – makes a copy of the currently selected cell in memory
- **Paste cell** – pastes a copy of a cell from memory
- **Move cell up/down** – moves a cell's position within the notebook
- **Execute cell** – executes the code in a cell. For markdown cells, renders the cell using specified formatting
- **Stop/Restart Kernel** – stops or restarts the Kernel – useful if something is running long or you made a mistake
- **Cell type selector** – displays and allows you to change the type of the currently selected cell

Notebook Menus (File/Edit)



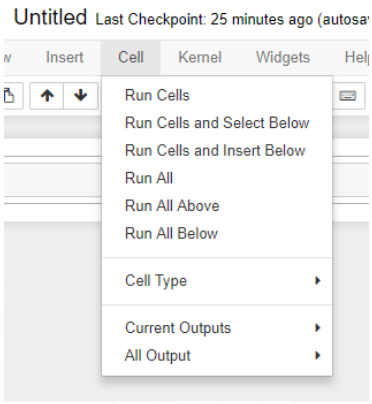


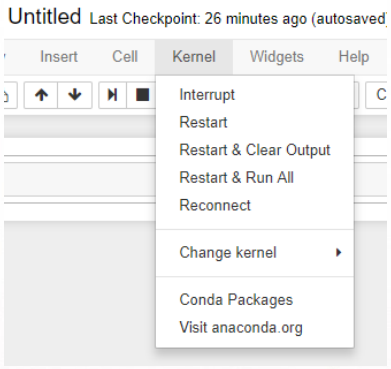
The file menu has options that look very familiar and are similar to those in other applications.

Note the option to “Revert to Checkpoint”. This allows you to revert a notebook to a saved checkpoint. Once reverted, there is no undoing the reversion

The Edit menu offers additional options for working with cells beyond what is available via the icons on the tool bar. Note that cells can be merged and split.

Notebook Menus (Cell/Kernel)

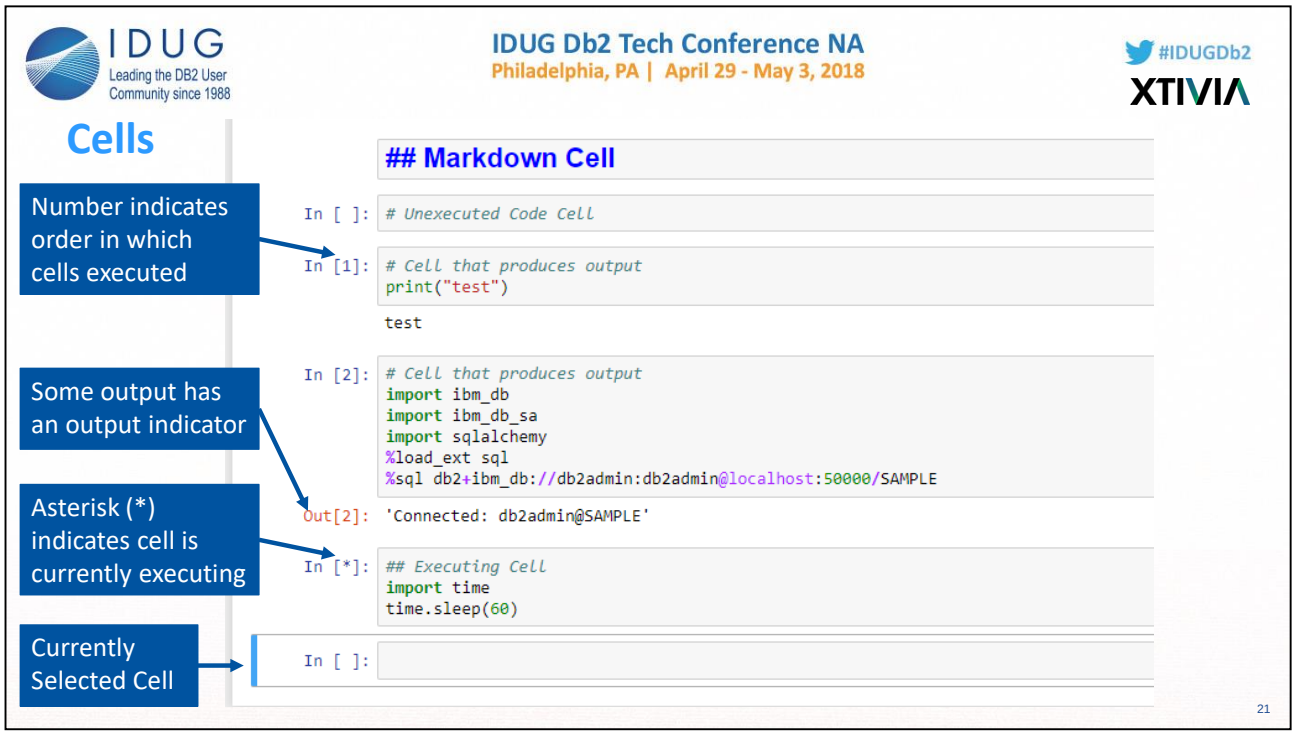




20

The Cell menu offers options for changing cell types, and running groups of cells without selecting them.

One of the more useful options on the Kernel menu is to restart & clear output. This provides a fresh starting point, particularly if you are looking to share notebooks.



BasicsOfJupyterNotebook.ipynb

Cells within a workbook are primarily either code cells or Markdown cells.

- Unexecuted Markdown cells look like the first one in this example. When a markdown cell is executed, the text appears formmated as specified. To edit an executed markdown cell, double click on the text.
- Unexecuted code cells are prefixed with “In [ ]:”.
- A number within the brackets for a code cell indicates that the cell has been executed. The number indicates the order in which the cell was executed.
- An asterisk (*) within the brackets for a code cell indicates that the cell is currently executing.
- The currently selected cell has a border around it with a blue or green bar at the beginning to indicate selection. Many actions from the toolbar will apply only to the selected cell(s).

Markdown

- Easily readable and forgiving syntax for formatting text in html-consistent ways
- References:
  - https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet
  - https://daringfireball.net/projects/markdown/syntax

```
# H1
## H2
### H3
#### H4
*italic* **bold** ~~strikethrough~~
1. First ordered list item
2. Another item
   * Unordered sub-list.
1. Actual numbers don't matter, just that it's a number

[link text](https://www.db2commerce.com)
```


Markdown is a standardized and intuitive syntax for formatting text. Many guides exist for it online.

Python Basics

- Indentation matters
  - Loops and functions are defined not by starting and terminating characters, but by proper use of indentation
- # starts a comment line
- No line terminators -> line continuation characters (\ )
- No prefix to indicate a variable

SQL Magic

Setup for Using Db2 and the SQL Magic in a Notebook

- Use the following within the Jupyter Notebook to set things up

```
import sys,os,os.path
os.environ['IBM_DB_HOME']='C:\Program Files\IBM\SQLLIB'
!pip install ipython-sql
!pip install ibm_db
!pip install ibm_db_sa
import ibm_db
import ibm_db_sa
import sqlalchemy
%load_ext sql
```

25

If you're using SQL Magic for the first time, there are certain packages that will have to be installed. Some of these will not install properly unless you've set the environment variable (on Windows) IBM_DB_HOME to the SQLLIB of an instance before running the install commands. After installing these packages, the Kernel will have to be restarted before they can be used.

Using SQL in Jupyter Notebook

- SQL magic makes SQL quick and easy
- Db2 commands can be executed, when the notebook was launched from a command window, when prefixed with !
- Limited options available in sql magic - full ibm_db offers more options using Python or the core language of your choice
- Basic setup to use sql magic with Db2 is in the first few cells of the 2017 Db2 Symposium Notebook: https://github.com/ecrooks/db2_and_jupyter_notebooks

Database Connections from Jupyter Notebook

- Store credentials in a separate file for easier notebook sharing
- Connection string should be fairly easy to understand:
  - %sql db2+ibm_db://user:password@host:port/db
- Explicitly closing connections not currently possible – connections closed when notebook is halted.

Jupyter Notebook and SQL Magic Topics

- How does commit processing work by default and how can you change it?
- Using SQL magic for whole cell vs. line by line
- Displaying data in interesting ways
- Using host variables/parameter markers
- Explains

Jupyter Notebook and Commit Processing

```
In [9]: %sql insert into temp_sales select * from db2admin.sales
        41 rows affected.
Out[9]: []

In [10]: %sql select count(*) from temp_sales
         Done.
Out[10]:
| 1 |
|---|
| 123 |

In [11]: %sql insert into temp_sales select * from db2admin.sales
         %sql rollback
         %sql select count(*) from temp_sales
         41 rows affected.
         Done.
         Done.
Out[11]:
| 1 |
|---|
| 164 |
```

- Autocommit is on
- More advanced controls available through traditional python syntax for the ibm_db driver

When using SQL Magic, Autocommit is on. There is no current way to turn it off, but there may be in the future.

Using SQL Magic at the Cell Level vs. Line Level

- Line Level
  - Each line is prefixed with %
  - If a command fails, subsequent lines are still executed
- Cell Level
  - Using SQL Magic at the cell level involves starting a cell with %%sql
  - Nothing should be in the cell before %%sql
  - All lines in the cell are then interpreted as SQL
  - If a command fails, subsequent lines are not executed

Using SQL Magic at the Cell Level vs. Line Level

In [14]:
```
%%sql
--sql magic at the cell level
select * from syscat.tables;
select * from dual;
select * from syscat.bufferpools;
```

```
Done.
(ibm_db_dbi.ProgrammingError) ibm_db_dbi::ProgrammingError: SQLNumResultCols failed: [IBM][CLI Driver][DB2/NT64] SQL0204N  "DB2A
DMIN.DUAL" is an undefined name.  SQLSTATE=42704\r SQLCODE=-204 [SQL: 'select * from dual;']
```

In [15]:
```
#sql magic at the line level
%sql select * from syscat.tables;
%sql select * from dual;
%sql select * from syscat.bufferpools;
```

```
Done.
(ibm_db_dbi.ProgrammingError) ibm_db_dbi::ProgrammingError: SQLNumResultCols failed: [IBM][CLI Driver][DB2/NT64] SQL0204N  "DB2A
DMIN.DUAL" is an undefined name.  SQLSTATE=42704\r SQLCODE=-204 [SQL: 'select * from dual;']
Done.
```

Out[15]:

bpname	bufferpoolid	dbpgname	npages	PAGESIZE	estore	numblockpages	blocksize	ngname
IBMDEFAULTBP	1	None	250	8192	N	0	0	None
BUFF32K	2	None	-2	32768	N	0	0	None

31

One failing statement is intentionally included here for demonstration purposes.

Each cell returns the output from the last statement executed. This can be changed by using print statements and other methods if desired

The first cell shows the use of an sql magic at the cell level. Notice:

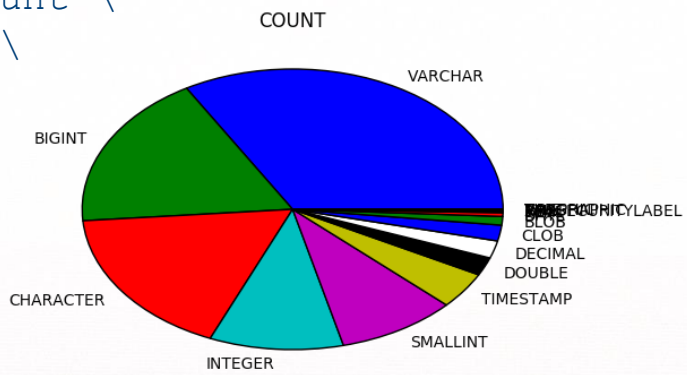
- The first line of the cell is prefixed with two percentage symbols (%%)
- The comment character then becomes not the “#” that is standard for Python, but the”--” which is used for comments in SQL
- When a line fails in the middle of the cell, no further statements are executed after the failure

The second cell here shows the same commands run with SQL Magic at the line level. Notice:

- The failed statement does not prevent the following statements from executing
- The output is that from the last statement only

Displaying Data in Interesting Ways – Pie Chart

```
result= %sql select typename \
    , count(*) as count \
from syscat.columns \
group by typename \
order by count desc

result.pie()
```



After importing some modules we need, creating a pie chart can be as easy as a single line after the SQL statement

Displaying Data in Interesting Ways – Data Frame

```
result= %sql WITH SUM_TAB (SUM_RR, SUM_CPU, SUM_EXEC, SUM_SORT, SUM_NUM_EXEC) AS ( \
      SELECT  nullif(FLOAT(SUM(ROWS_READ)),0), \
              nullif(FLOAT(SUM(TOTAL_CPU_TIME)),0), \
              nullif(FLOAT(SUM(STMT_EXEC_TIME)),0), \
              nullif(FLOAT(SUM(TOTAL_SECTION_SORT_TIME)),0), \
              nullif(FLOAT(SUM(NUM_EXECUTIONS)),0) \
          FROM TABLE(MON_GET_PKG_CACHE_STMT ( 'D', NULL, NULL, -2)) AS T \
      ) \
SELECT substr(stmt_text,1,25) as STATEMENT, \
      ROWS_READ, \
      coalesce(DECIMAL(100*(FLOAT(ROWS_READ)/SUM_TAB.SUM_RR),5,2),0) AS PCT_TOT_RR, \
      TOTAL_CPU_TIME, \
      coalesce(DECIMAL(100*(FLOAT(TOTAL_CPU_TIME)/SUM_TAB.SUM_CPU),5,2),0) AS PCT_TOT_CPU, \
      STMT_EXEC_TIME, \
      coalesce(DECIMAL(100*(FLOAT(STMT_EXEC_TIME)/SUM_TAB.SUM_EXEC),5,2),0) AS PCT_TOT_EXEC, \
      TOTAL_SECTION_SORT_TIME, \
      coalesce(DECIMAL(100*(FLOAT(TOTAL_SECTION_SORT_TIME)/SUM_TAB.SUM_SORT),5,2),0) AS PCT_TOT_SRT, \
      NUM_EXECUTIONS, \
      coalesce(DECIMAL(100*(FLOAT(NUM_EXECUTIONS)/SUM_TAB.SUM_NUM_EXEC),5,2),0) AS PCT_TOT_EXECS, \
      DECIMAL(FLOAT(STMT_EXEC_TIME)/FLOAT(NUM_EXECUTIONS),10,2) AS AVG_EXEC_TIME, \
      RTRIM(STMT_TEXT) as FULL_STATEMENT \
  FROM TABLE(MON_GET_PKG_CACHE_STMT ( 'D', NULL, NULL, -2)) AS T, SUM_TAB \
  WHERE DECIMAL(100*(FLOAT(ROWS_READ)/SUM_TAB.SUM_RR),5,2) > 10 \
      OR DECIMAL(100*(FLOAT(TOTAL_CPU_TIME)/SUM_TAB.SUM_CPU),5,2) >10 \
      OR DECIMAL(100*(FLOAT(STMT_EXEC_TIME)/SUM_TAB.SUM_EXEC),5,2) >10 \
      OR DECIMAL(100*(FLOAT(TOTAL_SECTION_SORT_TIME)/SUM_TAB.SUM_SORT),5,2) >10 \
      OR DECIMAL(100*(FLOAT(NUM_EXECUTIONS)/SUM_TAB.SUM_NUM_EXEC),5,2) >10 \
  ORDER BY ROWS_READ DESC FETCH FIRST 20 ROWS ONLY WITH UR
```

```
%matplotlib inline
df=result.DataFrame()
```

Data Frame - Describe

- df.describe()

	rows_read	total_cpu_time	stmt_exec_time	total_section_sort_time	num_executions
count	6.000000	6.000000	6.000000	6.0	6.000000
mean	28.666667	2604.166667	7.500000	0.0	17.500000
std	26.553091	6378.879538	9.354143	0.0	15.514509
min	0.000000	0.000000	0.000000	0.0	5.000000
25%	5.500000	0.000000	1.250000	0.0	10.000000
50%	28.000000	0.000000	3.500000	0.0	13.000000
75%	47.500000	0.000000	11.000000	0.0	16.000000
max	64.000000	15625.000000	24.000000	0.0	48.000000

Data Frame – Shape and Columns

- Show the number of rows and the number of columns of the output

```
In [71]: df.shape
Out[71]: (5, 13)
```

- Show the names of all columns, in a comma separated list

```
In [72]: df.columns
Out[72]: Index(['STATEMENT', 'rows_read', 'pct_tot_rr', 'total_cpu_time', 'pct_tot_cpu',
                'stmt_exec_time', 'pct_tot_exec', 'total_section_sort_time',
                'pct_tot_srt', 'num_executions', 'pct_tot_execs', 'avg_exec_time',
                'full_statement'],
               dtype='object')
```

Data Frame – Sort Output

In [86]:
```
# sort output by a different column
df.sort_values(by=['stmt_exec_time'], ascending=False)
```

Out[86]:

	STATEMENT	rows_read	pct_tot_rr	total_cpu_time	pct_tot_cpu	stmt_exec_time	pct_tot_exec	total_section_sort_time	pct_tot_s
4	CALL SYSPROC.SYSINSTALLOB	2	1.16	15625	100.00	24	48.00	0	0.00
3	SELECT POLICY FROM SYSTOO	16	9.30	0	0.00	13	26.00	0	0.00
0	SELECT TRIGNAME FROM SYS	64	37.20	0	0.00	5	10.00	0	0.00
2	select tabschema , ta	40	23.25	0	0.00	2	4.00	0	0.00
1	SELECT COLNAME, TYPENAME	50	29.06	0	0.00	1	2.00	0	0.00
5	SET CURRENT LOCK TIMEOUT	0	0.00	0	0.00	0	0.00	0	0.00

Data Frame – Convert Data Types

```
In [74]: df.dtypes

Out[74]: STATEMENT                object
         rows_read                 int64
         pct_tot_rr               object
         total_cpu_time            int64
         pct_tot_cpu              object
         stmt_exec_time            int64
         pct_tot_exec             object
         total_section_sort_time   int64
         pct_tot_srt              object
         num_executions            int64
         pct_tot_execs            object
         avg_exec_time            object
         full_statement           object
         dtype: object
```

- Some data types may not be what is expected

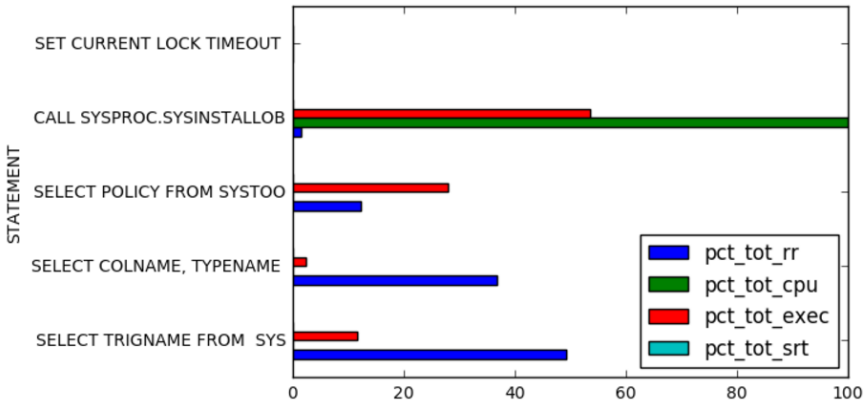
```
In [75]: df[['pct_tot_rr']]=df[['pct_tot_rr']].astype(float)
         df[['pct_tot_cpu']]=df[['pct_tot_cpu']].astype(float)
         df[['pct_tot_exec']]=df[['pct_tot_exec']].astype(float)
         df[['pct_tot_srt']]=df[['pct_tot_srt']].astype(float)
         df[['pct_tot_execs']]=df[['pct_tot_execs']].astype(float)
         df[['avg_exec_time']]=df[['avg_exec_time']].astype(float)
```

```
In [76]: df.dtypes

Out[76]: STATEMENT                object
         rows_read                 int64
         pct_tot_rr              float64
         total_cpu_time            int64
         pct_tot_cpu             float64
         stmt_exec_time            int64
         pct_tot_exec            float64
         total_section_sort_time   int64
         pct_tot_srt             float64
         num_executions            int64
         pct_tot_execs           float64
         avg_exec_time           float64
         full_statement           object
         dtype: object
```

Data Frame – Horizontal Bar Graph

```
In [77]: df.plot(x='STATEMENT', y=['pct_tot_rr','pct_tot_cpu','pct_tot_exec','pct_tot_srt'], kind='barh')
         plt.show
```

```
Out[77]: <function matplotlib.pyplot.show>
```



Data Frame – Displaying Full Text

```
In [90]: pd.set_option('display.max_colwidth', -1)
         df[['full_statement']]
```

Out[90]:

	full_statement
0	SELECT TRIGNAME FROM SYSCAT.TRIGGERS WHERE TABNAME='POLICY' AND TABSCHEMA='SYSTOOLS'
1	SELECT COLNAME, TYPENAME FROM SYSCAT.COLUMNS WHERE TABNAME='POLICY' AND TABSCHEMA='SYSTOOLS'
2	select tabschema , tabname , controlauth , deleteauth , insertauth , selectauth , updateauth from syscat.tabauth where grantee = ?
3	SELECT POLICY FROM SYSTOOLS.POLICY WHERE MED='DB2CommonMED' AND DECISION='NOP' AND NAME='CommonPolicy'
4	CALL SYSPROC.SYSINSTALLOBJECTS('POLICY','V','','')
5	SET CURRENT LOCK TIMEOUT 5

Using Python Variable in SQL – Literal value

In [5]:
```
check_id = 'DB2ADMIN'
%sql select tabschema \
    , tabname \
    , controlauth \
    , deleteauth \
    , insertauth \
    , selectauth \
    , updateauth \
from syscat.tabauth \
where grantee = '{check_id}'
```

Done.

Out[5]:

tabschema	tabname	controlauth	deleteauth	insertauth	selectauth	updateauth
DB2ADMIN	TEMP_SALES	Y	G	G	G	G
SYSTOOLS	ADVISE_INDEX	Y	G	G	G	G
SYSTOOLS	ADVISE_INSTANCE	Y	G	G	G	G
SYSTOOLS	ADVISE_MQT	Y	G	G	G	G
SYSTOOLS	ADVISE_PARTITION	Y	G	G	G	G

Using Python Variable in SQL – Parameter Marker

In [6]:
```
check_id = 'DB2ADMIN'
%sql select tabschema \
    , tabname \
    , controlauth \
    , deleteauth \
    , insertauth \
    , selectauth \
    , updateauth \
from syscat.tabauth \
where grantee = :check_id
```

Done.

Out[6]:

tabschema	tabname	controlauth	deleteauth	insertauth	selectauth	updateauth
SYSTOOLS	POLICY	Y	G	G	G	G
SYSTOOLS	HMON_ATM_INFO	Y	G	G	G	G
SYSTOOLS	HMON_COLLECTION	Y	G	G	G	G
DB2ADMIN	TEMP_SALES	Y	G	G	G	G
SYSTOOLS	EXPLAIN_INSTANCE	Y	G	G	G	G

Generating Explain Plan in Jupyter Notebook

In [ ]:
```
# This cell only needs to be executed if the explain tables do not exist
%sql call sysproc.sysinstallobjects('EXPLAIN','C',NULL,NULL)
```

In [91]:
```
%sql set current explain mode explain
# Below will return CLI0115E, but that is expected and it works fine.
%sql select bpname from syscat.bufferpools
```

```
    916         except AttributeError:
    917             return self._non_result([])

C:\Program Files\Anaconda3\lib\site-packages\ibm_db_dbi.py in fetchall(self)
   1459         after executing an SQL statement which produces a result set.
   1460         """
-> 1461         return self._fetch_helper()
   1462
   1463     def nextset(self):

C:\Program Files\Anaconda3\lib\site-packages\ibm_db_dbi.py in _fetch_helper(self, fetch_size)
   1416                 self.messages.append(_get_exception(inst))
   1417             if len(row_list) == 0:
-> 1418                 raise self.messages[len(self.messages) - 1]
   1419             else:
   1420                 return row_list

InternalError: (ibm_db_dbi.InternalError) ibm_db_dbi::InternalError: Fetch Failure: [IBM][CLI Driver] CLI0115E  Invalid curso
r state. SQLSTATE=24000 SQLCODE=-99999
```

Generating Explain Plan in Jupyter Notebook

```
In [92]: %sql set current explain mode no
         Done.
Out[92]: []

In [93]: !db2exfmt -d SAMPLE -1 -o query_exfmt.txt
         Connecting to the Database.
         DB2 Universal Database Version 11.1, 5622-044 (c) Copyright IBM Corp. 1991, 2015
         Licensed Material - Program Property of IBM
         IBM DATABASE 2 Explain Table Format Tool

         Connect to Database Successful.
         Using SYSTOOLS schema for Explain tables.
         Output is in query_exfmt.txt.
         Executing Connect Reset -- Connect Reset was Successful.
```

Pulling Explain Plan Into Jupyter Notebook

In [94]:
```
with open("query_exfmt.txt") as f:
    for line in f:
        if line.rstrip() == "Access Plan:":
            print("")
            for line in f:
                if line.rstrip() == "Extended Diagnostic Information:":
                    break
                print(line.rstrip())
```

```
----------
        Total Cost:             0.00922488
        Query Degree:           1


       Rows
      RETURN
      (   1)
       Cost
        I/O
        |
         2
      IXSCAN
      (   2)
   0.00922488
         0
        |
         2
  INDEX: SYSIBM
 INDBUFFERPOOLS01
        Q1
```

Practical Uses of Jupyter Notebook

Ideas for Possible Uses

- Partially automated processes
- Team documentation
- Teaching the details of a process
- Viewing memory allocations
- Health checks
- Storing frequently used SQL
- Emulating monitor reset in newer monitoring interfaces

Snapshot in Jupyter Notebook

- Purposes of a “snapshot”
  - Quick look at KPIs with potential for identifying problem areas
  - Mini health-check, at the database level
  - Tells me where to dig deeper
- Problems with traditional snapshots
  - Use older monitoring interfaces – higher impact
  - Much more information than I need for a quick look – I scroll for the few metrics I actually use
- Define your own requirements for a “snapshot”

What Ember Wants to See in a first Snapshot

Availability

Database up/down

Time activated

Recoverability/HADR

Time of last successful backup

HADR

- Role
- State
- Status
- Log Gap

Capacity

Low on space for any non-autoresize tablespaces

Performance

Index Read Efficiency

Buffer Pool Hit Ratios

Package Cache

- Hit Ratio
- Overflows

Catalog Cache

- Hit Ratio
- Overflows

Loging

- Log reads vs. log writes
- Number of log files archived per hour

Locking

- Deadlocks
- Lock Escalations
- Lock Timeouts

Sort Overflow Percentage

Ember's Jupyter Notebook Snapshot (1)

Instance Uptime

```
Database Member 0 -- Active -- Up 7 days 13:24:30 -- Date 2017-07-31-20.45.08.545000
```

Database active since

```
Done.
```

Out[10]:

db_conn_time
2017-07-27 18:39:16

This series of slides is in place in case the Jupyter Notebook does not work at presentation time

Ember's Jupyter Notebook Snapshot (2)

Last Succesful Backup

Done.

Out[12]:

TYPE	backup_time
ONLINE	2017-04-24 13:01:13

HADR details

Done.

Out[13]:

hadr_role	hadr_state	hadr_connect_status	hadr_log_gap

This series of slides is in place in case the Jupyter Notebook does not work at presentation time

Ember's Jupyter Notebook Snapshot (3)

Tablespaces

Done.

Out[37]:

num_tbsps_not_autoresize
0

Done.

Out[38]:

tbsp_name	pct_full

This series of slides is in place in case the Jupyter Notebook does not work at presentation time

Ember's Jupyter Notebook Snapshot (4)

Index Read Efficiency

Done.

Out[39]:

read_eff
10.17

Buffer Pool Hit Ratios

Overall

Done.

Out[40]:

total_l_reads	data_hit_ratio_percent	index_hit_ratio_percent	xda_hit_ratio_percent	col_hit_ratio_percent
236133	99.77	99.12	-1.00	-1.00

By Bufferpool

Done.

Out[41]:

	BP Name	Logical Reads	Data Hit Ratio	Index Hit Ratio	XML Hit Ratio	Column Hit Ratio
0	IBMDEFAULTBP	236144	99.77	99.12	-1.00	-1.00

This series of slides is in place in case the Jupyter Notebook does not work at presentation time

Ember's Jupyter Notebook Snapshot (4)

Package Cache

Done.

Out[42]:

pkg_cache_hitratio	pkg_cache_num_overflows
96.17	25

Catalog Cache

Done.

Out[43]:

cat_cache_hitratio	cat_cache_overflows
99.86	0

This series of slides is in place in case the Jupyter Notebook does not work at presentation time

Ember's Jupyter Notebook Snapshot (5)

Transaction Logs

Done.

Out[44]:

log_reads	log_writes
0	10887

Done.

Out[45]:

tot_log_use_pct	sec_logs_allocated	sec_log_used_top
0.78	0	0

Done.

Out[46]:

avg_arch_per_hour	max_arch_in_hour	min_arch_in_hour
1	2	1

This series of slides is in place in case the Jupyter Notebook does not work at presentation time

Ember's Jupyter Notebook Snapshot (6)

EmberSnapshot-Reset

Locking

Done.

Out[47]:

deadlocks	lock_timeouts	lock_escals
0	0	0

Sorts

Done.

Out[48]:

sort_overflow_pct
0.00

This series of slides is in place in case the Jupyter Notebook does not work at presentation time

Notebooks from this Presentation

https://github.com/ecrooks/db2_and_jupyter_notebooks

- Basic Db2 Connection.ipynb
  - Only the basics needed to get to a connection
- DB2MemoryAnalysis.ipynb
  - Analyzes and plots memory utilization for an instance/database
- EmberSnapshot.ipynb
  - Ember's take on a snapshot done in Jupyter Notebook
- EmberSnapshot-Reset.ipynb
  - Ember's snapshot with reset functionality for multiple mon_get table functions

Notebooks from this Presentation

https://github.com/ecrooks/db2_and_jupyter_notebooks

- Basic Db2 Connection DB2 Extension.ipynb
  - Only the basics needed to get to a connection
- DB2MemoryAnalysis.ipynb
  - Analyzes and plots memory utilization for an instance/database
- EmberSnapshot_Db2_Extension.ipynb
  - Ember's take on a snapshot done in Jupyter Notebook
- EmberSnapshot-Reset_Db2_Extension.ipynb
  - Ember's snapshot with reset functionality for multiple mon_get table functions

Other Resources

- Github for DB2 Extensions
  - https://github.com/DB2-Samples/db2jupyter
- Docker Resources:
  - Docker jupyter notebook image: search docker hub on jupyter_base
  - Github of docker image including db2 client: https://github.com/ibmdbanalytics/dashdb_analytic_tools
    - Note: references dashDB, but should work with Db2 on-prem

Ember is a DB2 Lead DBA and Delivery Manager at XTIVIA. She is responsible for the design, build, and management of a wide range of DB2 databases for multiple clients. Ember has 16+ years of experience with DB2 on Linux, Unix, and Windows platforms. She is the founder and principal author of the popular db2commerce.com technical blog where she educates herself and others through example and case study. Ember is an IBM Gold Consultant and IBM Champion in Information Management.