

Operational Database Hands-on Lab Guide

2 Hours to Data Innovation with
Cloudera Operational Database

December 01, 2023



Accessing Your COD Lab Environment	4
COD Instance for Lab	5
Storage Location and Data Hub Name	6
Accessing the Edge node	7
Workshop	9
Section One - COD Basics	9
Exercise 1: Accessing COD	11
Access the database & tables in COD	11
Using the Phoenix CLI	12
Using the HBase CLI	12
Using Hue	13
Exercise 2: Namespace and Table	15
Creating Namespace and table	15
Create Table	15
Identifying Table Structure and Primary Keys	18
Create Table Namespace	20
Table Data Handling - UPSERT/READ/DELETE	23
Exercise 3: Dynamic Columns and Table Views	26
Understanding Dynamic Columns	26
Adding a Dynamic Column	27
Reading Dynamic Columns from a Table	27
Table Views	31
Section 2 - COD Operations	34
Exercise 4: Monitoring COD	34
htop	34
pe	35
Region Hotspotting	35
Resource Manager	37
Grafana Charts	41
Exercise 5 : Bulkload Data	44
PSQL Client Utility	44
CSVBulkloadTool	45
Exercise 6: COD Administration	48
Snapshots	48
Understanding Your Data	53
Indexing	58
Resiliency	61
Replication Manager	64
Create a Large Table	68

Exercise 7: Auto-Scaling	76
How Does COD Auto-scale?	76
Scaling in action	77

Accessing Your COD Lab Environment

Because you'll be running a bunch of commands, let's gather some information about our COD environment and populate some variables in your session so you don't have to worry about entering details each time you need to connect. To accomplish this, proceed with the following steps:

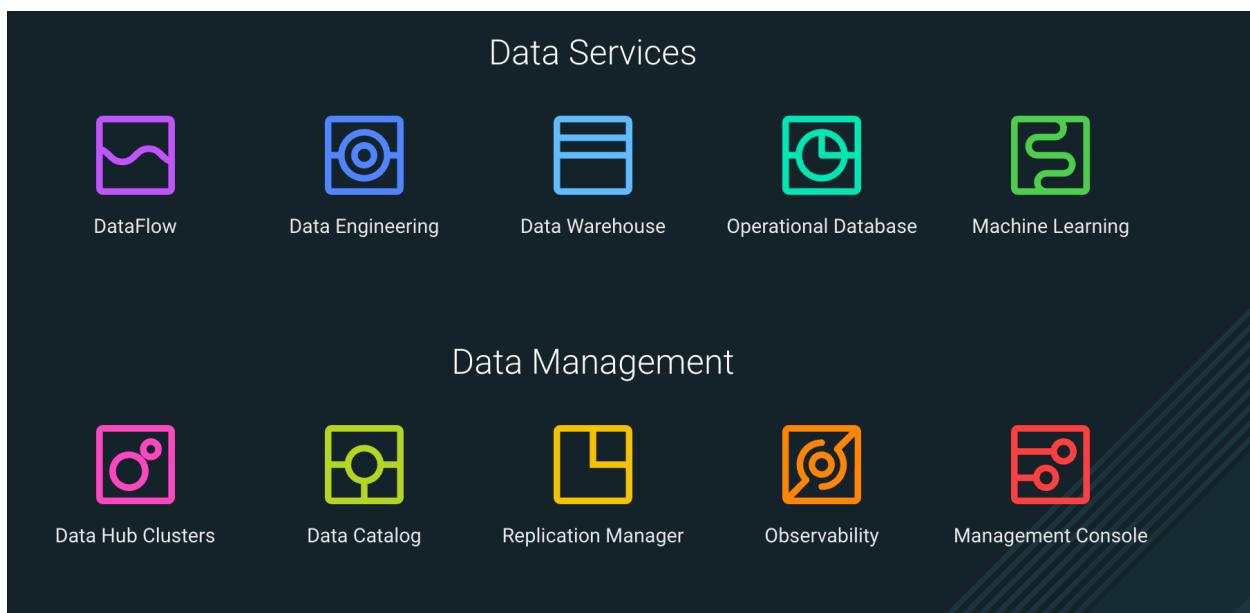
- If not already completed, *open your COD UI by using the workshop URL:* <https://bit.ly/HOL1130>

NOTE: You must NOT be on a VPN

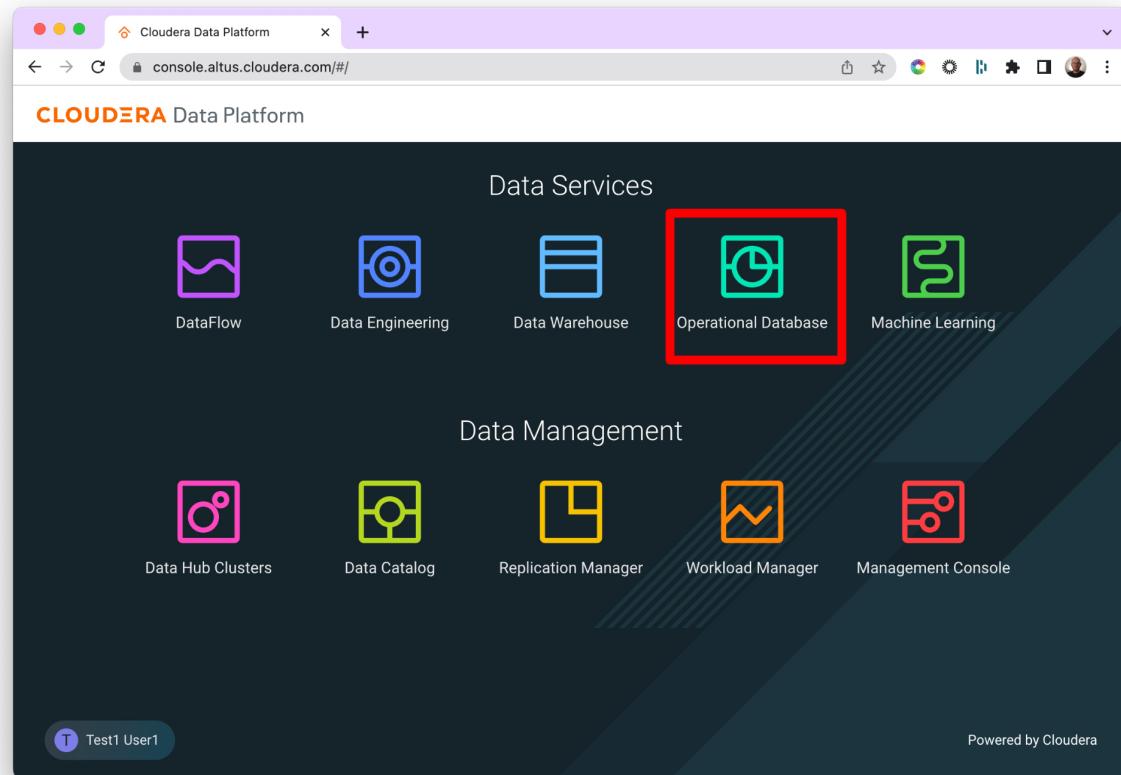
- Enter your username and password:

first name	last name	user	password
Paul	Williamson	user046	G0yvxvdms5srhyKF
Santosh	Patel	user044	G0yvxvdms5srhyKF

You should see the following screen:



Once connected, *click on the Operational Database tile*. This will get you into the COD management console.



COD Instance for Lab

- You'll see a single COD instance. The instance **cloud-hbase** will be the one we will use for the lab. *Click on cloud-hbase*.

Databases

Databases					Create Database
Status	Name	Environment	Created By	Date Created	
Available	cloud-hbase	tor-hol-cdp-env	Carlos Hernandez	11/30/2023 4:07 PM AST	⋮

Storage Location and Data Hub Name

DEFINITION: A Data Hub is a collection of servers much like a traditional cluster.

Let's begin by gathering information on where our data resides and the nodes associated with our COD cluster. Keep in mind that COD is cloud enabled and leverages object storage in either AWS, Azure, or Google. This provides for significant cost savings without negatively impacting performance.

- First, copy your **cloud storage location** from your COD UI. Save this information for later.

In our case, this is **s3a://tor-hol-buk-fdeea6f5/cod-1ihojougenp25/hbase**

Associated S3 bucket name : **tor-hol-buk-fdeea6f5**
Associated Data Hub name : **cod-1ihojougenp25**

Databases / opdb-hol-cod

Available - Updated just now
opdb-hol-cod
cm:cdp:opdb:us-west-1:f8e16d2d-57e6-43bc-8e50-75a32f7782ad:opDb:5e0699bc-f6f9-49c9-a9e6-e694a32ed83c ⓘ
VERSION CREATED BY
1.34.0 Christopher Perro

ENVIRONMENT REGION ⓘ DATA LAKE SQL EDITOR ⓘ GRAFANA DASHBOARD ⓘ CLOUD STORAGE LOCATION
opdb-hol us-east-2 opdb-hol-dl Hue N/A s3a://opdb-hol/cod-bpvr11vhwd/hbase ⓘ

Connect Charts Events

HBase HBase REST HBase Client Tarball Phoenix (Thick) Phoenix (Thin) Phoenix (ODBC) Phoenix Python

Usage ⓘ
You can use the Apache HBase Maven URL, Apache HBase Client Version, and the Apache HBase Client Configuration URL to download a file containing the Apache HBase Client Configuration.
HBase Maven URL ⓘ
https://archive.cloudera.com/p/cdp-public/7.2.17.0/maven-repository ⓘ
HBase Client Version ⓘ
2.4.6.7.2.17.0-334 ⓘ
HBase Client Configuration URL ⓘ
curl -f -o "hbase-config.zip" -u "sshaw" "https://cod-bpvr11vhwd-gateway0.opdb-hol.z30z-14kp.cloudera.site/clouderamanager/api/v41/clusters/cod-bpvr11vhwd/services/hbase/clientConfig" ⓘ
> Kerberos Configuration

Accessing the Edge node

- To identify the FQDN of the edge node to leverage SSH, we'll need to locate the edge node in the COD Data Hub. To do this, click on the **Data Hub** tile in the main CDP Control Plane



- This will open the main Data Hub page. Click on the Data Hub associated with the COD instance.

The screenshot shows the Cloudera Management Console Data Hubs page. On the left, there's a sidebar with links: Dashboard, Environments, Data Lakes, User Management, Data Hub Clusters (selected), and Data Warehouses. The main area is titled "Data Hubs" and shows a table with one item. The table columns are: Status, Name, Cloud Provider, Environment, Data Hub Type, Version, Node Count, and Created. The single row shows: Running, cod-bpvr11fvhhwd, aws, opdb-hol, 7.2.17 - Operational Database: Apache HBase, Phoenix-COD-16741f81-sft, CDH 7.2.17, 18, and 09/05/23, 05:39 PM CDT. A red arrow points to the "cod-bpvr11fvhhwd" name in the table.

- Edge nodes are listed under Nodes tab
- Copy your edge node FQDN using the icon next to the FQDN name:

user	edge node fqdn
user046	cod-1ihojougenp25-edge0.tor-hol.z30z-14kp.cloudera.site
user044	cod-1ihojougenp25-edge1.tor-hol.z30z-14kp.cloudera.site

Data Hubs / cod--bpvr11fvhhwd / Nodes

The screenshot shows the Cloudera Data Hub interface for the hub 'cod--bpvr11fvhhwd'. The left sidebar has a 'Nodes' tab selected. The main area is divided into 'Master' and 'Edge' sections. Each section contains a table with columns: Instance ID, Status, FQDN, Private IP, and Public IP. The 'Master' section has two rows: one for 'cod--bpvr11fvhhwd-master0.opdb-hol.z30z-14kp.cloudera.site' (IPs 10.10.1.148, 3.140.210.222) and another for 'cod--bpvr11fvhhwd-master1.opdb-hol.z30z-14kp.cloudera.site' (IPs 10.10.1.199, 18.221.120.155). The 'Edge' section also has two rows: one for 'cod--bpvr11fvhhwd-edge6.opdb-hol.z30z-14kp.cloudera.site' (IPs 10.10.1.181, 3.138.246.68) and another for 'cod--bpvr11fvhhwd-edge1.opdb-hol.z30z-14kp.cloudera.site' (IPs 10.10.1.159, 3.145.132.193).

	Instance ID	Status	FQDN	Private IP	Public IP
>	i-01062887388d8d6bf	Running	cod--bpvr11fvhhwd-master0.opdb-hol.z30z-14kp.cloudera.site	10.10.1.148	3.140.210.222
>	i-0544f46ed96ffff1f	Running	cod--bpvr11fvhhwd-master1.opdb-hol.z30z-14kp.cloudera.site	10.10.1.199	18.221.120.155

	Instance ID	Status	FQDN	Private IP	Public IP
>	i-072122b83e4a4cf07	Running	cod--bpvr11fvhhwd-edge6.opdb-hol.z30z-14kp.cloudera.site	10.10.1.181	3.138.246.68
>	i-03536ed608255ccee	Running	cod--bpvr11fvhhwd-edge1.opdb-hol.z30z-14kp.cloudera.site	10.10.1.159	3.145.132.193

- Now, let's set up your edge node in your shell so it will be easier to connect. On your laptop, add an *EDGENODE* variable to your bash_profile, to have them populated automatically without having to enter them all throughout the workshop

```
Unset
:
#
#use user assigned in lab
#
# EdgeNode
echo "EDGENODE=cod-1ihojougenp25-edge2.tor-hol.z30z-14kp.cloudera.site" >>
~/bash_profile

echo "COD_USER=<user01-099>" >> ~/bash_profile
source ~/bash_profile
```

- Now, SSH to your assigned edge node

```
Unset
ssh $COD_USER@$EDGENODE
```

- Finally, let's set environment variables to avoid needing to remember cryptic COD names. Value for BUCKET and COD variable is captured in section above [Storage location and underline datahub name](#)
- On the edge node, set the bucket name and COD edgenode name.

```
Unset
```

```
echo "export BUCKET=tor-hol-buk-fdeea6f5" >> ~/.bash_profile
echo "export COD=cod-1ihojougenp25" >> ~/.bash_profile
source ~/.bash_profile
```

- Now, open a second terminal (to be used later), SSH to your assigned edge node

```
Unset
```

```
ssh $COD_USER@$EDGENODE
```

You should now have 2 open terminal sessions.

Workshop

This workshop will demonstrate features and capabilities of COD with a focus on using a JDBC interface provided through Apache Phoenix

You will be executing each exercise against a COD cluster instance. All commands are run on your assigned edge node.

Section One - COD Basics

- Run the cd command to the `cod-workshop` directory on your edge node.

```
Unset
```

```
cd cod-workshop/
```

A script has already been run to generate 1M CDR (call data records) in the `cdr` subdirectory.

- Now let's take a look at one line of the CDR data generated:

```
Unset
```

```
zcat cdr/cdr-20231201_00.csv.gz | head -1
```

```
0033733147213,c1dc8fec-076f-4289-883e-d4b882a0a81d,0033670188928,2023-02-06  
00:16:00,2023-02-06  
00:23:33,0,SMS,SUCCESS,1,GSM,21889,1.350476,49.690095,9,GSM,2516,5.901936,47.244771
```

This is a compressed CSV file. Here are the significant parameters:

```
Calling_msisdn : 0033648163427  
Session_id : 2eff6824-6315-4f09-b96c-add56d7d3015  
Called_msisdn : 0033648143665  
date_beginning of the call : 2023-02-06 00:25:00  
Date_end : 2023-02-06 00:38:25  
duration (seconds) : 805
```

```
call_type (could be Voice/SMS): Voice
Call_result : SUCCESS
calling_net_type (Mobile Network Code) : 2
Radio_calling : GSM
cell_calling (ID of the cell) : 45343
Longitude of calling cell : 5.147953
Latitude of calling cell : 47.285376
Called_net_type : 2
Radio_called : GSM
Cell_called : 14168
Lon_called : -1.649971
Lat_called : 48.592906
```

Exercise 1: Accessing COD

There are a number of tools and options available to access the HBase cluster managed by COD and the same is applicable for HBase clusters provisioned in CDP Base or CDP Private Cloud clusters.

1. **Phoenix-sqlline CLI** - enables a client to run SQL on HBase. It runs from the command line of your Edge node.
2. **HBase Shell** - enables a client to deploy applications and manage HBase. It also runs from the command line of your Edge node.
3. **External SQL editor tools** - can also connect to databases in a COD cluster, their connection is managed via a gateway node. We will use HUE as SQL Editor, later in the Lab

Access the Database and Tables in COD

[Apache Phoenix](#) provides a JDBC interface to connect to a COD cluster. Apache Phoenix allows users and applications to interface with HBase through standard SQL and provides full ACID capabilities and the benefits of schema-on-read.

Apache Phoenix provides for either a **thick client** which resides on the edge node or a **thin client** that allows for connection pooling on a **Phoenix Query Server (PQS)**.

Phoenix-sqlline is typically useful to developers and engineers to prepare and test queries. The Phoenix-sqlline uses the Phoenix thick client.

For this lab, CLI tools are already deployed on the edge node with the correct default parameters; you don't need to specify the arguments (e.g. zookeeper quorum url etc).

NOTE: As mentioned before you should have two separate SSH terminal sessions connecting to Edge node to help facilitate lab work.

Let's first do a quick test on each utility to make sure they are working correctly for the lab.

Using Phoenix-sqlline

- Using one of your 2 SSH sessions, launch Phoenix-sqlline CLI

```
Unset
phoenix-sqlline
0: jdbc:phoenix:>
```

- list your tables with the !tables command:

```
Unset
!tables

+-----+-----+-----+-----+-----+-----+-----+-----+
| TABLE_CAT | TABLE_SCHEMA | TABLE_NAME | TABLE_TYPE | REMARKS | TYPE_NAME | SELF_REFERENCING_COL_NAME | REF_GENERATION | INDEX_STATE | IMMUTABLE_ROWS | SALT_BUCKETS |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          | SYSTEM      | CATALOG     | SYSTEM TABLE |          |          |          |          |          |          |          |          |
|          | SYSTEM      | CHILD_LINK   | SYSTEM TABLE |          |          |          |          |          |          |          |          |
|          | SYSTEM      | FUNCTION     | SYSTEM TABLE |          |          |          |          |          |          |          |          |
|          | SYSTEM      | LOG          | SYSTEM TABLE |          |          |          |          |          |          |          |
|          | SYSTEM      | MUTEX        | SYSTEM TABLE |          |          |          |          |          |          |          |
|          | SYSTEM      | SEQUENCE     | SYSTEM TABLE |          |          |          |          |          |          |          |
|          | SYSTEM      | STATS        | SYSTEM TABLE |          |          |          |          |          |          |          |
|          | SYSTEM      | TASK          | SYSTEM TABLE |          |          |          |          |          |          |          |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

- Exit phoenix-sqlline

```
Unset
```

```
!exit
```

Using the HBase CLI

- Open another SSH session, on your edge node, launch the HBase CLI:

```
Unset
```

```
hbase shell
hbase:001:0>
```

- You can list HBase tables with the `list` command:

```
Unset
```

```
hbase:002:0> list

TABLE
SYSTEM:CATALOG
SYSTEM:CHILD_LINK
SYSTEM:FUNCTION
SYSTEM:LOG
SYSTEM:MUTEX
SYSTEM:SEQUENCE
SYSTEM:STATS
SYSTEM:TASK
OMID_COMMIT_TABLE
OMID_TIMESTAMP_TABLE
10 row(s)
Took 1.9995 seconds
=> ["SYSTEM:CATALOG", "SYSTEM:CHILD_LINK", "SYSTEM:FUNCTION", "SYSTEM:LOG",
"SYSTEM:MUTEX", "SYSTEM:SEQUENCE", "SYSTEM:STATS", "SYSTEM:TASK",
"OMID_COMMIT_TABLE", "OMID_TIMESTAMP_TABLE"]
hbase:003:0>
```

NOTE: You can use any SQL editors (DBeaver, SQLWorkbench, whatever) to connect to the COD cluster using JDBC URL. We will be using phoenix-sqlline for the labwork.

Using Hue

CDP provides a HUE web-based SQL editor UI, it is integrated with COD.

- From the COD database instance page, click on the HUE link to start HUE editor in a new tab.

Databases / opdb-hol-cod

Available - Updated just now
opdb-hol-cod
cm:cdp:opdb:us-west-1:f8e16d2d-57e6-43bc-8e50-75a32f7782ad:opDb:5e0699bc-f6f9-49c9-a9e6-e694a32ed83c ⓘ
VERSION CREATED BY
1.34.0 Christopher Perro

ENVIRONMENT REGION DATA LAKE SQL EDITOR GRAFANA DASHBOARD CLOUD STORAGE LOCATION
opdb-hol us-east-2 opdb-hol-dl Hue N/A s3a://opdb-hol/cod-bpvr11vhwd/hbase ⓘ

Actions ▾

- Type “Show tables” to show the user’s tables.

phoenix ⌂ Add a name... Add a description...

1| show tables

▶ ⟲

Exercise 2: Namespace and Table

This lab demonstrates how to create a namespace and table, and access table schemas.

Creating Namespace and table

Apache Phoenix provides a SQL layer over Apache Hbase, allowing users/ applications to connect to the Hbase cluster using JDBC connections and interact with the cluster leveraging the widely used SQL language. Apache Phoenix client compiles a user provided SQL query into very efficient HBase native calls thus shielding developers from the complexity of HBase native interface while helping improve developer productivity and reduce application code complexity.

Apache community performed a number of benchmarks, and identified that Apache Phoenix delivers equal or better performance than using Hbase native interface.

Create Table

Following is a list of key points concerning creating an HBase table through Phoenix.

1. A Phoenix table can be created using `CREATE TABLE...` SQL syntax.
2. When a table is created using Apache Phoenix it is mapped directly to the HBase table. So while the user/application is interacting with the Phoenix table, data is stored in the HBase table and HBase controls the management of the data.
3. Any table's primary key(s) defined are mapped to an HBase table's rowkey.

To load the test data created earlier, you'll create a table with a *PRIMARY KEY* set on `[calling_msisdn, session_id]`

WHY?: If the primary key would be set only on `calling_msisdn` then msisdn input records would be automatically deduplicated through the inserts.

- Use Phoenix-sqlline cli to create your first table and replace `${USER}` with your username.

Unset

```
CREATE TABLE IF NOT EXISTS test_${USER}
(calling_msisdn BIGINT NOT NULL,
session_id VARCHAR NOT NULL,
called_msisdn BIGINT,
date_beg VARCHAR,
date_end VARCHAR,
duration INTEGER,
call_type VARCHAR,
call_result CHAR(9),
calling_net_type TINYINT,
radio_calling VARCHAR,
cell_calling INTEGER,
lon_calling DECIMAL(10,8),
lat_calling DECIMAL(10,8),
called_net_type TINYINT,
radio_called VARCHAR,
cell_called INTEGER,
lon_called DECIMAL(10,8),
lat_called DECIMAL(10,8)
CONSTRAINT pk PRIMARY KEY (calling_msisdn,session_id));
```

Now list the tables and notice the table you just created shows in the list

Unset

```
0: jdbc:phoenix:> !tables
```

TABLE_CAT	TABLE_SCHEM	TABLE_NAME	TABLE_TYPE	REMARKS	TYPE_NAME	SELF_REFERENCING_COL_NAME	REF_GENERATION	INDEX_STATE	IMMUTABLE_ROW
/	SYSTEM	CATALOG	SYSTEM TABLE						
/		false							
/	SYSTEM	CHILD_LINK	SYSTEM TABLE						
/		false							

```

|          | SYSTEM      | FUNCTION     | SYSTEM TABLE |          |
|          |             | false        |              |          |
|          | SYSTEM      | LOG          | SYSTEM TABLE |          |
|          |             | true         |              |          |
|          | SYSTEM      | MUTEX        | SYSTEM TABLE |          |
|          |             | true         |              |          |
|          | SYSTEM      | SEQUENCE    | SYSTEM TABLE |          |
|          |             | false        |              |          |
|          | SYSTEM      | STATS        | SYSTEM TABLE |          |
|          |             | false        |              |          |
|          | SYSTEM      | TASK         | SYSTEM TABLE |          |
|          |             | false        |              |          |
|          |             | TEST_NLADDHA | TABLE       |          |
|          |             | false        |              |          |
+-----+-----+-----+-----+-----+
-----+-----+-----+-----+
0: jdbc:phoenix:>

```

This has created a simple phoenix table without any salting applied. So the underlying table is created with only one region. We will cover SALTING and SALT_BUCKETS later.

- **Using the HBase CLI,** let's check in Hbase using `hbase-shell`, `list` command provides a list of Hbase tables and `list_regions` command provides a list of regions associated with the specified table. Make sure you type the table name in **ALL CAPS**.

Unset

```

hbase:003:0> list

hbase:004:0> list_regions "TEST_${USER}"
                                         SERVER_NAME |
REGION_NAME | START_KEY | END_KEY | SIZE | REQ | LOCALITY |
-----+-----+-----+-----+-----+-----+-----+
----- | ----- | ----- | ----- | ----- | ----- |
cod-bhyfehf766wh-worker4.ps-sandb.a465-9q4k.cloudera.site,16020,1689078075786 |
TEST_NLADDHA,,1689333890021.a4f5c6a922bb88ff79d6cd20bec9d32e. |
|     0 |     1 |      0.0 |
1 rows

```

```
Took 0.5524 seconds  
hbase:005:0>
```

Notice also that the table is created under the default namespace (database schema).

Now look at the table, notice the table columns all appear in uppercase. Notice also the Primary Key appears in another color.

- **Use the Phoenix-sqlline cli** to perform a `SELECT * from` newly created table.

```
Unset
```

```
SELECT * FROM test_${USER};  
  
+-----+-----+-----+  
| CALLING_MSISDN | SESSION_ID | CALLED_MSISDN | DATE_BEG |  
+-----+-----+-----+  
+-----+-----+-----+  
No rows selected (0.033 seconds)
```

NOTE: phoenix-sqlline requires a semicolon at the end of all SQL queries

Identifying Table Structure and Primary Keys

Let's look at the table schema. This is useful when the table structure/schema is unknown. You can do with `!describe` command

- Use Phoenix-sqlline cli to execute the `!describe` command

Unset

```
0: jdbc:phoenix:> !describe TEST_${USER}
```

TABLE_CAT	TABLE_SCHEM	TABLE_NAME	COLUMN_NAME	DATA_TYPE	TYPE_NAME	COLUMN_SIZE	BUFFER_LENGTH	DECIMAL_DIGITS	NUM_PREC_RAD
		TEST_NLADDHA	CALLING_MSISDN	-5	BIGINT	null	null		
		TEST_NLADDHA	SESSION_ID	12	VARCHAR	null	null		
		TEST_NLADDHA	CALLED_MSISDN	-5	BIGINT	null	null		
		TEST_NLADDHA	DATE_BEG	12	VARCHAR	null	null		
		TEST_NLADDHA	DATE_END	12	VARCHAR	null	null		
		TEST_NLADDHA	DURATION	4	INTEGER	null	null		
		TEST_NLADDHA	CALL_TYPE	12	VARCHAR	null	null		
		TEST_NLADDHA	CALL_RESULT	1	CHAR	9	null		
		TEST_NLADDHA	CALLING_NET_TYPE	-6	TINYINT	null	null		
		TEST_NLADDHA	RADIO_CALLING	12	VARCHAR	null	null		
		TEST_NLADDHA	CELL_CALLING	4	INTEGER	null	null		
		TEST_NLADDHA	LON_CALLING	3	DECIMAL	10	null		
8		TEST_NLADDHA	LAT_CALLING	3	DECIMAL	10	null		
8		TEST_NLADDHA	CALLED_NET_TYPE	-6	TINYINT	null	null		
		TEST_NLADDHA	RADIO_CALLED	12	VARCHAR	null	null		
		TEST_NLADDHA	CELL_CALLED	4	INTEGER	null	null		
		TEST_NLADDHA	LON_CALLED	3	DECIMAL	10	null		
8		TEST_NLADDHA	LAT_CALLED	3	DECIMAL	10	null		

When you want to check table primary keys, it can be displayed using `!primarykeys` command

- Use Phoenix-sqlline cli to execute the `!primarykeys` command

Unset

```
0: jdbc:phoenix:> !primarykeys TEST_${USER}
```

TABLE_CAT	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	KEY_SEQ	PK_NAME	ASC_OR_DESC	DATA_TYPE	TYPE_NAME	COLUMN_SIZE	TYPE_ID	VIE
-5	BIGINT	null	-5	1	PK	A	TEST_NLADDHA	CALLING_MSISDN			
12	VARCHAR	null	12	2	PK	A	TEST_NLADDHA	SESSION_ID			

Note that KEY_SEQ column provides the order of primary keys, it is important information while defining a performant `SELECT` query to read record(s).

Create Table Namespace

A table namespace (database schema) allows you to group a table under a defined namespace.

- First create a `WORKSHOP_<USERNAME>` schema by using the `CREATE SCHEMA` syntax. We are adding the username value since multiple schemas with the same name cannot exist.

Unset

```
0: jdbc:phoenix:> CREATE SCHEMA WORKSHOP_${USER};
```

```
No rows affected (0.343 seconds)
```

- Create the same table as above but under the `WORKSHOP_<USERNAME>` schema. Note that the table name uses the format of `<namespace>. <tablename>`

Unset

```
0: jdbc:phoenix:> CREATE TABLE IF NOT EXISTS workshop_${USER}.test_${USER}
  (calling_msisdn BIGINT NOT NULL,
  session_id VARCHAR NOT NULL,
  called_msisdn BIGINT,
  date_beg VARCHAR,
  date_end VARCHAR,
  duration INTEGER,
  call_type VARCHAR,
  call_result CHAR(9),
  calling_net_type TINYINT,
  radio_calling VARCHAR,
  cell_calling INTEGER,
  lon_calling DECIMAL(10,8),
  lat_calling DECIMAL(10,8),
  called_net_type TINYINT,
  radio_called VARCHAR,
  cell_called INTEGER,
  lon_called DECIMAL(10,8),
  lat_called DECIMAL(10,8)
  CONSTRAINT pk PRIMARY KEY (calling_msisdn,session_id));
```

- Now check the list of tables using the !tables command. Notice that the results now include both tables: one with schema name WORKSHOP and another without the schema name (created earlier). Note that both are two separate tables.

Unset

```
0: jdbc:phoenix:> !tables
```

TABLE_CAT	TABLE_SCHEM	TABLE_NAME	TABLE_TYPE	REMARKS	TYPE_NAME	SELF_REFERENCING_COL_NAME
REF_GENERATION	INDEX_STATE	IMMUTA				
	SYSTEM	CATALOG	SYSTEM TABLE			
	false					
	SYSTEM	CHILD_LINK	SYSTEM TABLE			
	false					
	SYSTEM	FUNCTION	SYSTEM TABLE			
	false					
	SYSTEM	LOG	SYSTEM TABLE			
	true					

	SYSTEM	MUTEX	SYSTEM TABLE			
	true					
	SYSTEM	SEQUENCE	SYSTEM TABLE			
	false					
	SYSTEM	STATS	SYSTEM TABLE			
	false					
	SYSTEM	TASK	SYSTEM TABLE			
	false					
		TEST_NLADDHA	TABLE			
	false					
	WORKSHOP	TEST_NLADDHA	TABLE			
	false					

Lets now check the same tables in HBase using `hbase-shell list` command. This is a simple command to provide a list of all HBase tables.

- Using hbase-shell cli, type the `list` command

```
Unset
hbase:005:0> list

TABLE
SYSTEM:CATALOG
SYSTEM:CHILD_LINK
SYSTEM:FUNCTION
SYSTEM:LOG
SYSTEM:MUTEX
SYSTEM:SEQUENCE
SYSTEM:STATS
SYSTEM:TASK
WORKSHOP:TEST_${USER}
OMID_COMMIT_TABLE
OMID_TIMESTAMP_TABLE
TEST_${USER}
12 row(s)
Took 0.0138 seconds
=> [ "SYSTEM:CATALOG", "SYSTEM:CHILD_LINK", "SYSTEM:FUNCTION", "SYSTEM:LOG",
"SYSTEM:MUTEX", "SYSTEM:SEQUENCE", "SYSTEM:STATS", "SYSTEM:TASK",
"WORKSHOP:TEST_NLADDHA", "OMID_COMMIT_TABLE", "OMID_TIMESTAMP_TABLE",
"TEST_NLADDHA" ]
```

```
hbase :006 :0>
```

NOTE: All phoenix table names and their columns names are CAPITALIZED by default in HBase, so remember to type phoenix table names in CAPITAL letters when using the hbase-shell CLI

TIP: In a Production deployment, it is best practice to create a database schema (aka table namespace), and then create tables under the appropriate schema. This allows you to group tables using namespaces.

Table Data Handling - UPSERT/READ/DELETE

Data in HBase is never static. CRUD (create, read, update, delete) is fundamental to any database operation, whether relational or a nosql database like HBase. Let's see how the phoenix-sqlline handles each scenario.

- **Using your phoenix-sqlline CLI**, let's do the following:

1. UPSERT to ingest and/update record(s) to table
2. SELECT to read record(s)
3. DELETE to delete record(s)

```
Unset
```

```
UPSERT INTO test_${USER} (CALLING_MSISDN,SESSION_ID)
VALUES(123456789,'session');
```

```
1 row affected (0.064 seconds)
```

```
select CALLING_MSISDN,SESSION_ID,CALLED_MSISDN from test_{USER} where  
CALLING_MSISDN=123456789;
```

```
+-----+-----+-----+  
| CALLING_MSISDN | SESSION_ID | CALLED_MSISDN |  
+-----+-----+-----+  
| 123456789     | session    | null         |  
+-----+-----+-----+  
1 row selected (0.024 seconds)
```

```
DELETE FROM test_{USER} WHERE CALLING_MSISDN=123456789;
```

```
1 row affected (0.029 seconds)
```

```
!exit
```

Now let's insert some lines by importing a file. To exit from your Phoenix CLI, type `!exit` or you can open a new terminal window.

- On your edge node, create a file named data.csv with 10 lines of data:

```
Unset  
echo  
"0033733147213,c1dc8fec-076f-4289-883e-d4b882a0a81d,0033670188928,2023-02-06  
00:16:00,2023-02-06  
00:23:33,0,SMS,SUCCESS,1,GSM,21889,1.350476,49.690095,9,GSM,2516,5.901936,47.24  
4771  
0033648163427,2eff6824-6315-4f09-b96c-add56d7d3015,0033648143665,2023-02-06  
00:25:00,2023-02-06  
00:38:25,805,Voice,SUCCESS,2,GSM,45343,5.147953,47.285376,2,GSM,14168,-1.649971  
,48.592906  
0033690127891,c2d0fe02-4096-4cd1-aea0-7b46dc41aace,0033653204161,2023-02-06  
00:20:00,2023-02-06  
00:20:45,45,Voice,SUCCESS,9,GSM,11510,5.317312,47.944747,10,GSM,22656,4.871122,  
45.031399  
0033751119091,ce8bfe49-ab05-42ab-b808-94f6dda09c6c,0033795178310,2023-02-06  
00:11:00,2023-02-06  
00:27:02,962,Voice,SUCCESS,9,GSM,15438,5.894631,48.575936,2,GSM,7574,0.186621,4  
6.057284
```

```
0033780183331,3aefe1a2-5be1-47bf-b9b3-ca941a3ba85a,0033635151591,2023-02-06  
00:05:00,2023-02-06  
00:16:35,695,Voice,SUCCESS,21,GSM,449,3.6701202392578,50.440292358398,10,GSM,45  
201,5.064914,45.672283  
0033678176048,79ae6532-a4b8-4695-98f4-a76cea2c2fad,0033616116529,2023-02-06  
00:21:00,2023-02-06  
00:31:06,606,Voice,SUCCESS,1,GSM,1857,4.853037,45.311832,15,GSM,29308,7.204447,  
43.762039  
0033693155656,10223aeb-d35f-453d-a378-c9e4b4b0fefafa,003342753209969,2023-02-06  
00:39:00,2023-02-06  
00:43:00,240,Voice,SUCCESS,21,GSM,24614,0.519582,48.096866,1,GSM,14410,7.271668  
,43.699879  
0033772106353,56531bc3-e929-4c4a-ae65-4d80f0da9686,0033758186973,2023-02-06  
00:39:00,2023-02-06  
00:40:50,110,Voice,SUCCESS,2,GSM,12969,2.470707,49.099391,20,GSM,57388,4.771405  
,44.154108  
0033725195471,dfde6cc2-c75f-47ca-9a0d-7f573f56fca0,0033716114791,2023-02-06  
00:13:00,2023-02-06  
00:24:04,664,Voice,SUCCESS,15,GSM,29308,7.204447,43.762039,8,GSM,5724,6.706298,  
49.127397  
0033716123232,b73e3b81-a90a-4661-800c-3e6dd75d8a53,0033617105755,2023-02-06  
00:38:00,2023-02-06  
00:52:23,0,Voice,FAILURE,1,GSM,52377,5.975871,47.218095,8,GSM,10061,6.114757,49  
.071954" > ./data.csv
```

Now let's import that data to the table we created above.

- From the edge node, execute the following command to import records from a file. This will use the *phoenix-psql* utility. You will perform the following steps:
 - Run the *phoenix-psql* command. This will list all available command line options
 - Run the *phoenix-psql -t* command. This will load the data into the table
 - Run *phoenix-sql* and execute a *SELECT ** statement to retrieve the records. Be sure to replace <USERNAME> with your username IN ALL CAPS.

Unset

phoenix-psql

```
phoenix-psql -t TEST_${USER} data.csv
```

```
CSV Upsert complete. 10 rows upserted
Time: 0.063 sec(s)
```

```
phoenix-sqlline
```

```
0: jdbc:phoenix:> select * from test_${USER}; #(output has been cut)
```

CALLING_MSISDN	SESSION_ID	CALLED_MSISDN	DATE_BEG
33648163427 00:25:00	2eff6824-6315-4f09-b96c-add56d7d3015	33648143665	2023-02-06
33678176048 00:21:00	79ae6532-a4b8-4695-98f4-a76cea2c2fad	33616116529	2023-02-06
33690127891 00:20:00	c2d0fe02-4096-4cd1-aea0-7b46dc41aace	33653204161	2023-02-06
33693155656 00:39:00	10223aeb-d35f-453d-a378-c9e4b4b0fefea	3342753209969	2023-02-06
33716123232 00:38:00	b73e3b81-a90a-4661-800c-3e6dd75d8a53	33617105755	2023-02-06
33725195471 00:13:00	dfde6cc2-c75f-47ca-9a0d-7f573f56fcfa0	33716114791	2023-02-06
33733147213 00:16:00	c1dc8fec-076f-4289-883e-d4b882a0a81d	33670188928	2023-02-06
33751119091 00:11:00	ce8bfe49-ab05-42ab-b808-94f6dda09c6c	33795178310	2023-02-06
33772106353 00:39:00	56531bc3-e929-4c4a-ae65-4d80f0da9686	33758186973	2023-02-06
33780183331 00:05:00	3aefe1a2-5be1-47bf-b9b3-ca941a3ba85a	33635151591	2023-02-06

```
10 rows selected (0.052 seconds)
```

Exercise 3: Dynamic Columns and Table Views

Understanding Dynamic Columns

Phoenix supports dynamic schemas. At runtime, a column can be added without redefining the table. It is called a *dynamic column*.

To do so, we will perform an `upsert` statement and add the new column name along with its type. In this case, we will add a `boolean` flag field to the call record for demonstration purposes.

NOTE: The `UPSERT` command is documented here:
https://phoenix.apache.org/atomic_upsert.html

We will use the HUE editor in this lab. To access HUE see section [Using Hue](#). Alternatively you can do the following using phoenix-sqlline

Adding a Dynamic Column

- Let's add a new dynamic column called "FLAG" of Boolean type to your previously created table. Be sure to specify the dynamic column name and its data type in the `UPSERT` query. Replace `${USER}` with your user

Unset

```
UPSERT INTO TEST_${USER}
(CALLING_MSISDN,
SESSION_ID,
CALLED_MSISDN,
DATE_BEG,DATE_END,
DURATION,CALL_TYPE,
CALL_RESULT,
CALLING_NET_TYPE,
RADIO_CALLING,CELL_CALLING,
LON_CALLING,
LAT_CALLING,
CALLED_NET_TYPE,
RADIO_CALLED,
CELL_CALLED,
```

```

LON_CALLED,
LAT_CALLED,
FLAG_BOOLEAN)
VALUES
(0033716123234, 'b73e3b81-a90a-4661-800c-3e6dd75d8a54', 0033617105755, '2023-02-06
00:38:00', '2023-02-06
00:52:23', 0, 'Voice', 'FAILURE', 1, 'GSM', 52377, 5.975871, 47.218095, 8, 'GSM', 10061, 6.
114757, 49.071954, TRUE);

SELECT * FROM TEST_${USER};

```

Reading Dynamic Columns from a Table

- Without referencing the dynamic column in your `SELECT` query you may notice that the new field and value is not returned.

Unset

```
SELECT * FROM test_${USER} WHERE calling_msisdn = 33716123234;
```

The screenshot shows the Apache Phoenix interface. At the top, there's a toolbar with icons for refresh, add name, add description, and help. Below the toolbar, the query input field contains:

```
1|select * from test_BHAGAN where calling_msisdn = 33716123234|
```

Below the query input, there are navigation buttons for back, forward, and search. The main area is divided into three tabs: Query History, Saved Queries, and Results (1). The Results tab is selected and displays the following table:

	CALLING_MSISDN	SESSION_ID	CALLED_MSISDN	DATE_BEG	DATE_END	DURATION	CAI
1	33716123234	b73e3b81-a90a-4661-800c-3e6dd75d8a54	33617105755	2023-02-06 00:38:00	2023-02-06 00:52:23	0	Voi

The screenshot shows the Apache Phoenix interface. At the top, there are buttons for 'phoenix' (with a logo), 'Add a name...', 'Add a description...', and icons for graph, copy, and more. Below the header, a query is being typed into the editor: '1| select * from test_BHAGAN where calling_msisdn = 33716123234|'. The status bar indicates '0.31s'. The results section shows a single row with the following data:

	_CALLING	LON_CALLING	LAT_CALLING	CALLED_NET_TYPE	RADIO_CALLED	CELL_CALLED	LON_CALLED	LAT_CALLED
1	7	5.975871	47.218095	8	GSM	10061	6.114757	49.071954

- We need to specify the dynamic field and its data type in our `SELECT` statement to let it know that we want to read data from the dynamic column as well.

Unset

```
SELECT * FROM test_${USER} (FLAG BOOLEAN) WHERE calling_msisdn = 33716123234;
```

The screenshot shows the Apache Phoenix interface. The query is identical to the one above: '1| select * from test_BHAGAN (FLAG BOOLEAN) where calling_msisdn = 33716123234|'. The status bar indicates '0.37s'. The results section now includes an additional column 'FLAG' in the table header and displays the value 'true' in the last column of the single row.

	_CALLING	LON_CALLING	LAT_CALLING	CALLED_NET_TYPE	RADIO_CALLED	CELL_CALLED	LON_CALLED	LAT_CALLED	FLAG
1	5.975871	47.218095	8	GSM	10061	6.114757	49.071954	true	

If you execute the same query from `phoenix-sqlline`, you'll notice that only the columns that fit in the terminal are displayed

```
0: jdbc:phoenix:> SELECT * FROM TEST_USER001 WHERE CALLING_MSISDN=1000;
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| CALLING_MSISDN | SESSION_ID | CALLED_MSISDN | DATE_BEG | DATE_END | DURATION | CALL_TYPE | CALL_RESULT | CALLING_NET_TYPE | RADIO_CALLING | CELL_CALLING |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1000 | 1000a | 2000 | 2023-01-01 | 2023-01-01 | 10 | LOCAL | GOOD | 5 | N | 1 |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row selected (0.027 seconds)
```

One workaround is to only select the columns you want to see

Unset

```
SELECT CALLING_MSISDN,SESSION_ID,FLAG FROM TEST_${USER} (FLAG BOOLEAN) WHERE
CALLING_MSISDN=33716123234;
```

```
+-----+-----+-----+
| CALLING_MSISDN | SESSION_ID | FLAG |
+-----+-----+-----+
| 33716123234 | b73e3b81-a90a-4661-800c-3e6dd75d8a54 | true |
+-----+-----+-----+
```

Another option is to leverage the `!outputformat vertical` option.

- Try running the full `SELECT *` query again, but this time include the `!outputformat vertical` option

Unset

```
!outputformat vertical
SELECT * FROM TEST_${USER} (FLAG BOOLEAN) WHERE CALLING_MSISDN=33716123234;
```

```
CALLING_MSISDN      33716123234
SESSION_ID          b73e3b81-a90a-4661-800c-3e6dd75d8a54
CALLED_MSISDN       0033617105755
DATE_BEG            2023-02-06 00:38:00
DATE_END            2023-02-06 00:52:23
DURATION            0
CALL_TYPE            Voice
CALL_RESULT          Failure
CALLING_NET_TYPE    1
RADIO_CALLING       GSM
CELL_CALLING        52377
LON_CALLING         5.975871
LAT_CALLING         47.218095
```

```
CALLED_NET_TYPE    8
RADIO_CALLED      GSM
CELL_CALLED       10061
LON_CALLED        6.114757
LAT_CALLED        49.071954
FLAG              true
```

```
1 row selected (0.012 seconds)
```

NOTE: There are a number of other formats you can use. Feel free to experiment with others. In order to return to the default, execute the !outputformat table option.

NOTE: Dynamic columns are not defined in table schema and hence are not displayed through !describe command

USE CASE: Our customers are extensively using dynamic column features to simplify their solution design.

For example, an Insurance client, selling various insurance products (e.g. home insurance, car insurance) are managed by various systems. For downstream data processing purposes they require all insurance policy records from various systems to be accessible in a centralized location.

While there are few common attributes (fields) across product families, each one also has a set of product specific fields which are not common across other products.

The customer defined a policy table using Apache Phoenix in HBase where common fields across product families are defined as static fields (i.e. part of table definition) and product specific fields are recorded as dynamic fields.

Through the dynamic column (schema evolution) feature, Apache Phoenix (HBase) provided them a capability where they can bring all policy records to a single location..

This solution also supported the need for a downstream data processing pipeline where it processed and updated each record to add new computed fields (as dynamic columns) for each record (based on applicable business rule).

Section 2 - COD Operations

Exercise 4: Monitoring COD

In this exercise we will look at various tools to monitor your COD cluster. These tools include HBase native tools, as well as a built-in Grafana dashboard. First off, we will look at how to monitor for the common issue of region hotspotting.

Region hotspotting can be a familiar performance issue on an HBase cluster. In the majority cases, region hotspotting is due to a poorly designed HBase table. HBase clusters can scale to petabytes of data by providing horizontal scalability. Tables are divided into regions and regions are spread over the nodes in the cluster. The regions are hosted and managed by region servers. If data is being loaded into and accessed from only one or a few number of regions and not utilizing the rest of the cluster, this is termed as region hotspotting. It is always important to avoid region hotspotting.

To have more understanding about what's going on in HBase and how we can detect region hotspotting, we're going to use two things: `hbtop` and `pe`.

`hbtop`

HBase `hbtop` is a real-time monitoring tool for HBase, inspired by the 'top' command you might know from Unix. `hbtop` provides two key functionalities:

1. A convenient way to see and understand what's happening in a COD cluster instance, like whether a region is over-solicited compared to the others, data locality, etc.
 2. Stats can be displayed at different levels (namespace, table, region, regionserver) and you can drill down on a metric. For example if you spot a hot region in a namespace, you can then go down to the table then the regions.
- Firstly, on your edge node, launch `hbtop`

Unset

`hbase hbtop`

- Now we will examine the various parameters. Execute each one and notice the results.

1. Enter `m` to change mode from region to table and then to namespace
2. Enter `d` and then "1" to change computing delay to 1s
3. Type `ESC` to exit

pe

`pe` is a versatile tool for testing HBase performance. It is capable of testing three key access patterns:

1. Random reads
2. Random writes
3. Sequential scan

In this section, we will quickly run through how to set up `pe` and how to use it to best understand how our HBase cluster will perform under various access conditions.

Though `pe` provides a lot of option, in this section we will only do the following:

1. Launch a performance evaluation
 2. Presplit on 10 regions
 3. Load data using random writes with 3 threads
 4. Random Writes
 5. Sequential Writes
- Type `CTRL+Z` to exit

Region Hotspotting

Now using two separate SSH sessions to the edge node, we will start the `pe` tool in one window to generate data load and use the other window to monitor the table regions where data is being loaded.

- In one of your terminal windows, ssh to your edge node. On your edge node, launch `hbttop`.

Unset

hbase hbtop

- Increase refresh rate from 3 sec to 1 sec, then: go on to the region level and filter on table <USERNAME>_TestTable, which is the name of the table that will be created using the `pe` tool. You'll also sort by RegionName

Unset

```
d 1
m [region]
o TABLE=${USER}_TestTable
f [REGION] s q
```

```
HBase hbtop - 19:40:06
Version: 2.4.6.7.2.17.0-334
Cluster ID: 90c2befc-b6a1-4cf2-b546-d190d2fac053
RegionServer(s): 5 total, 5 live, 0 dead
RegionCount: 212 total, 0 rit
Average Cluster Load: 42.40
Aggregate Request/s: 193665
add filter #1 (ignoring case) as: [!]FLD?VAL TABLE=USER089_TestTable
NAMESPACE      TABLE          REGION                         RS
USER099        CDR            f8571c0a8a88882b06b58436d8cef74e cod--bpvr1
USER099        CDR            9b6971de17462f4a682ff43196142c4f cod--bpvr1
USER099        CDR            45d1c46a758e9d67bd3ba58df1325df9 cod--bpvr1
```

- Using second terminal window, on your edge node, launch `pe`, load data to table with presplitting on 10 regions, and load data using random writes with 3 threads

Unset

hbase pe --nomapred --table=\${USER}_TestTable --presplit=10 randomWrite 3

Take a look at your HBase `hbtop` now, you can see requests well distributed, 150k requests/s total, each thread writing a default number of 1 million lines.

NAMESPACE	TABLE	REGION	RS	#REQ/S	#READ/S	#FREAD/S	#WRITE/S
default	user089_TestTable	ee95accebe6a40e281cb60dcc9bf825d	cod--bpvr11fvhhwd-worker0:16020	358	0	0	358
default	user089_TestTable	e49cd971dcf76cdab5b793c5890529bf6	cod--bpvr11fvhhwd-worker2:16020	0	0	0	0
default	user089_TestTable	d34853e25d52beb072a8797394b19417	cod--bpvr11fvhhwd-worker1:16020	1890	0	0	1890
default	user089_TestTable	c6cb203734f6c5ac56edef4077f6184	cod--bpvr11fvhhwd-worker2:16020	0	0	0	0
default	user089_TestTable	a9d3e2434f3a6a61659e8864f0a0c57d	cod--bpvr11fvhhwd-worker0:16020	382	0	0	382
default	user089_TestTable	a3f894cd4a0c6958935c841cd1ae2a9	cod--bpvr11fvhhwd-worker3:16020	84	0	0	84
default	user089_TestTable	7d94ebc100bc9237129399161cc77c4d	cod--bpvr11fvhhwd-worker1:16020	1892	0	0	1892
default	user089_TestTable	76489fa950a5d997c50537461f87b390	cod--bpvr11fvhhwd-worker4:16020	0	0	0	0
default	user089_TestTable	4aa2c3b50ef4fb61900f77570a3c3e	cod--bpvr11fvhhwd-worker4:16020	0	0	0	0
default	user089_TestTable	2bd0dce83534ffa7719c51b345e4813a	cod--bpvr11fvhhwd-worker3:16020	36	0	0	36

- Now launch the `pe` command again but with a sequential writing process, like writing rowkey 0, 1... to 1M

Unset

```
hbase pe --nomapred --table=${USER}_TestTable --presplit=10 sequentialWrite 3
```

Take a look at the `hbttop` terminal, what are you observing?

It's writing on 3 regions (because 3 threads here), sometimes 4 because of region splitting, but what you're observing here is **REGION HOTSPOTTING**

This is what you want to avoid at all costs! It is very important to have optimal row key design, as well as using table salting for Phoenix tables.

NOTE: If you want to read the standard on HBase row design, here is an excellent article to get you started.

<https://ajaygupta-spark.medium.com/design-principles-for-hbase-key-and-rowkey-3016a77fc52d>

```

HBase htop - 19:47:13
Version: 2.4.6.7.2.17.0-334
Cluster ID: 90c2befbe-b6a1-4cf2-b546-d190d2fac053
RegionServer(s): 5 total, 5 live, 0 dead
RegionCount: 212 total, 0 rit
Average Cluster Load: 42.40
Aggregate Request/s: 190074

NAMESPACE TABLE REGION RS #REQ/S #READ/S #FREAD/S #WRITE/S
default user089_TestTable e35e707e70c8ae2816bc2c2408308edc cod--bpvr1lfvhwd-worker4:16020 22234 0 0 22234
default user089_TestTable bccc00b71fc4fa8cb5e5daf9c6f4363 cod--bpvr1lfvhwd-worker2:16020 0 0 0 0
default user089_TestTable aa6ed8a6a0dea83275f82b0ea981c616 cod--bpvr1lfvhwd-worker0:16020 0 0 0 0
default user089_TestTable 8f261e82efeced972776df9f5427c008 cod--bpvr1lfvhwd-worker3:16020 0 0 0 0
default user089_TestTable 887f14a5bca7427862009dd56f6567e4e cod--bpvr1lfvhwd-worker4:16020 16758 0 0 16758
default user089_TestTable 84b7114a1ceaf0d3a681accf486045 cod--bpvr1lfvhwd-worker1:16020 0 0 0 0
default user089_TestTable 818766cd53360703649a0b01980e5b cod--bpvr1lfvhwd-worker3:16020 0 0 0 0
default user089_TestTable 5a10fbf9cd33388fe97c8420b6bebe42 cod--bpvr1lfvhwd-worker2:16020 24256 0 0 24256
default user089_TestTable 2274c40e169dadff16e740a5ca844a0d9 cod--bpvr1lfvhwd-worker1:16020 0 0 0 0
default user089_TestTable 0c174f204ca6a7d442fc988d31eb7bf6 cod--bpvr1lfvhwd-worker0:16020 0 0 0 0

```

Resource Manager

COD and the underlying HBase environment is still a fully part of the CDP ecosystem. You still have access to many of the open source tools you may be using today on your existing cluster. One of these tools is the YARN Resource Manager. We'll use Resource Manager in more detail later in the lab. For now, let's go to the Resource Manager page

- For that, open ResourceManager from your COD underlying Datahub page:

The screenshot shows the Cloudera Management Console interface for a Data Hub cluster named 'cod-de37z9agj7dr'. The left sidebar includes options like Dashboard, Environments, Data Lakes, User Management, Data Hub Clusters (which is selected), Data Warehouses, ML Workspaces, Classic Clusters, Audit, Shared Resources, and Global Settings. The main panel displays cluster details: STATUS (Running), NODES (9 nodes, 0 pending, 0 failed), CREATED AT (02/08/23, 01:22 AM GMT+1), and CLUSTER TEMPLATE (7.2.16 - Operational Database: Apache HBase, Phoenix-COD-c3c22fc-sft). Below this, there's an Environment Details section for 'aws' showing NAME (tkood-aw-env), DATA LAKE (tkood-aw-dl), CREDENTIAL (tkood-aw-xaccount-cred), REGION (us-east-2), and AVAILABILITY ZONE (N/A). The Services section lists CM-UI, HBase UI, HUE, Name Node, Queue Manager, Job History Server, and Token Integration, with 'Resource Manager' highlighted by a red arrow. At the bottom, there's a Cloudera Manager Info section with CM URL (https://cod-de37z9agj7dr-gateway.tkood-aw.yovj7g7d.cloudera.site/cod-de37z9agj7dr/cdp-proxy/cm/home/), CM VERSION (7.9.0), RUNTIME VERSION (7.2.16-1.cdh7.2.16.p0.36344516), and LOGS (Command logs, Service logs). Navigation links include Event History, Endpoints (6), Tags (18), Nodes, Network, Load Balancers, Telemetry, Repository Details, Image Details, Recipes (10), Cloud Storage, Database, and Upgrade.

- Click on the Applications tab. You won't see any jobs here yet for your userid.

APACHE Hadoop

Cluster Overview Queues Applications Services Flow Activity Nodes Tools

Logged in as: sshaw

Home / Applications

Reg Search... Search

User (2)

<input checked="" type="checkbox"/> user099	4
<input checked="" type="checkbox"/> user090	1
More	
Apply	Clear

Application ID	Application Type	Application Tag	Application Name	User	State	Queue	Progress
application_1694708973356_0005	MAPREDUCE	N/A	Phoenix MapRe...	user090	● Fini...	default	<div style="width: 10%;">1</div>
application_1694708973356_0004	MAPREDUCE	N/A	rowcounter_US...	user099	● Fini...	default	<div style="width: 10%;">1</div>
application_1694708973356_0003	MAPREDUCE	N/A	ExportSnapshot...	user099	● Fini...	default	<div style="width: 10%;">1</div>
application_1694708973356_0002	MAPREDUCE	N/A	ExportSnapshot...	user099	● Fini...	default	<div style="width: 10%;">1</div>
application_1694708973356_0001	MAPREDUCE	N/A	ExportSnapshot...	user099	● Fini...	default	<div style="width: 10%;">1</div>

- Leave the tab open and we can see them later.

Grafana Charts

Another useful monitoring feature of COD is the use of Grafana Charts. These charts are built into your COD cluster by default. Let's take a quick look at the details they provide.

- Navigate to the COD database page and click on the `opdb-hol-cod` database.

Databases

Status	Name	Environment	Created By	Date Created	⋮
● Available	opdb-cod-repl	opdb-hol	Scott Shaw	09/14/2023 7:49 AM CDT	⋮
● Available	opdb-hol-cod	opdb-hol	Christopher Perro	09/05/2023 5:39 PM CDT	⋮

Displaying 1 - 2 of 2 < 1 > 25 / page

- On the next screen, select the Charts tab to view the graphs.

Available - Updated just now
opdb-hol-cod
cm:cdp:opdb:us-west-1:f8e16d2d-57e6-43bc-8e50-75a32f7782ad:opDb:5e0699bc-f6f9-49c9-a9e6-e694a32ed83c ⓘ
VERSION CREATED BY
1.34.0 Christopher Perro

ENVIRONMENT REGION DATA LAKE SQL EDITOR GRAFANA DASHBOARD CLOUD STORAGE LOCATION
opdb-hol us-east-2 opdb-hol-dl Hue N/A s3a://opdb-hol/cod-bpvr11fvhhwd/hbase ⓘ

Connect Charts Events

HBase HBase REST HBase Client Tarball Phoenix (Thick) Phoenix (Thin) Phoenix (ODBC) Phoenix Python

Usage ⓘ You can use the Apache HBase Maven URL, Apache HBase Client Version, and the Apache HBase Client Configuration URL to download a file containing the Apache HBase Client Configuration.

HBase Maven URL ⓘ <https://archive.cloudera.com/p/cdp-public/7.2.17.0/maven-repository>

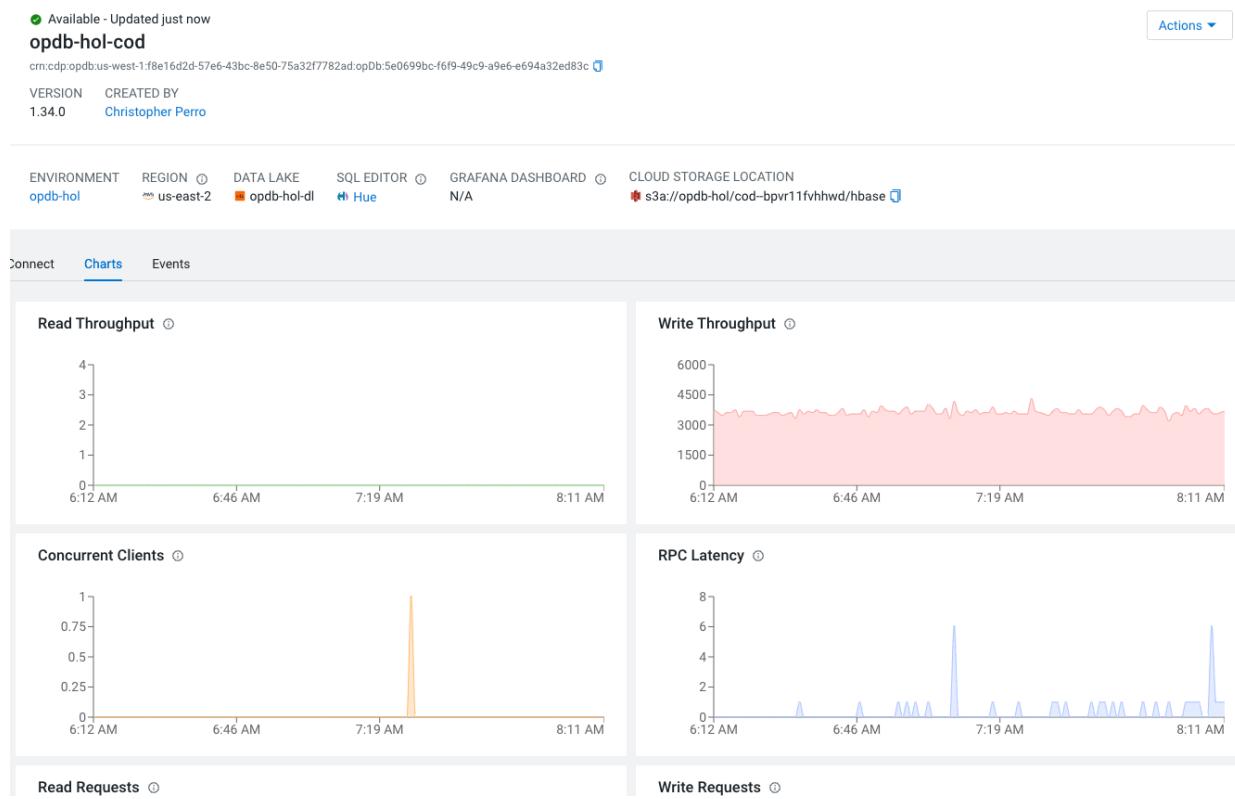
HBase Client Version ⓘ 2.4.6.7.2.17.0-334

HBase Client Configuration URL ⓘ curl -f -o "hbase-config.zip" -u "sshaw" "https://cod-bpvr11fvhhwd-gateway0.opdb-hol.z30z-14kp.cloudera.site/clouderamanager/api/v41/clusters/cod-bpvr11fvhhwd/services/hbase/clientConfig"

> Kerberos Configuration

From this page you are able to quickly view a number of key metrics for your COD cluster. These metrics include:

1. Read Throughput
2. Write Throughput
3. Concurrent Clients
4. RPC Latency
5. Read Requests
6. Write Requests
7. P99 Get Latency
8. Block Cache Hit Ratio
9. Compaction Queue Size



These metrics can help an administrator to quickly identify potential issues that may be negatively impacting cluster performance. From here an administrator can identify a potential problem and then dig further and address the problem through other methods.

Exercise 5 : Bulkload Data

There are multiple use cases requiring the loading of a large volume of records from a CSV or JSON file to a Phoenix (Hbase) table. For example, a source system providing daily/weekly/monthly data extracts.

Bulk loading utilities are provided to load data to Phoenix tables and/or Hbase tables. Here for the lab we will focus on bulk load utility to load data to a Phoenix table

In this exercise, we will use the test data CSV that you generated in section [Prepare test data](#),

We have two ways of doing this:

1. PSQL client utility
2. CSV Bulkload

PSQL Client Utility

The `phoenix-psql` command line utility is accessible on the edge node. It is a single-threaded client loading tool, provided out of the box.

- On your edge node, you will first copy the file as a .gz using `cp`. You will then unzip the file using `gunzip`. Finally, you will load the data using `phoenix-psql` with a `time` parameter. Feel free to move on with the lab while it loads. Normally it takes 4-8 minutes.
- Make sure the <USER> is UPPER CASE

Unset

```
cp cdr/cdr-20231201_00.csv.gz ./cdr.csv.gz
gunzip cdr.csv.gz

time phoenix-psql -t TEST_<USER> cdr.csv

CSV Upsert complete. 1000000 rows upserted
```

```
Time: 495.803 sec(s)
```

```
real    8m20.352s
user    0m50.751s
sys     0m3.292s
```

As you will notice, the imports are taking a long time. The `phoenix-psql` utility is useful to load small amounts of data. It does not require initiating a MR reduce job as well as there is no need to move data to HDFS or a S3 bucket (as explain in next section)

CSVBulkloadTool

In the case you have a large volume of data available as CSV files, it is more efficient to use a MapReduce job in order to take advantage of parallelism for bulk loading operations. Following are some of the key advantages of the bulkload tool.

1. The `CsvBulkLoadTool` Map-Reduce job utility provided out of the box
 2. The utility natively supports compression, source systems can compress files before copying to HDFS or S3 buckets.
 3. The job can be initiated from edge-node.
-
- Firstly, let's verify that you have an available S3 bucket. If you do not, then run `hdfs dfs -mkdir` to create the directory.

Unset

```
// Create a home directory for your username
hdfs dfs -mkdir s3a://$BUCKET/user/${USER}
```

- Once you have verified the directory, you will put the file on S3 and verify it is there

```
Unset  
# put your CSV in HDFS  
hdfs dfs -put ./cdr/cdr*.csv.gz s3a://$BUCKET/user/${USER}/  
  
# check the file is here  
hdfs dfs -ls -C s3a://$BUCKET/user/${USER}/ 2>/dev/null
```

- Finally, run the `CsvBulkLoadTool` to load the data

```
Unset  
# Import with MapReduce  
time hbase org.apache.phoenix.mapreduce.CsvBulkLoadTool --table test_${USER}  
--input s3a://$BUCKET/users/${USER}/*  
  
23/01/12 18:35:05 INFO mapreduce.Job: The url to track the job:  
https://...cloudera.site:8090/proxy/application_1673261990934_0009/  
23/01/12 18:35:05 INFO mapreduce.Job: Running job: job_1673261990934_0009  
23/01/12 18:35:17 INFO mapreduce.Job: map 0% reduce 0%  
23/01/12 18:35:40 INFO mapreduce.Job: map 13% reduce 0%  
23/01/12 18:35:46 INFO mapreduce.Job: map 33% reduce 0%  
23/01/12 18:35:52 INFO mapreduce.Job: map 68% reduce 0%  
23/01/12 18:35:58 INFO mapreduce.Job: map 74% reduce 0%  
23/01/12 18:36:08 INFO mapreduce.Job: map 100% reduce 0%  
23/01/12 18:36:25 INFO mapreduce.Job: map 100% reduce 40%  
23/01/12 18:36:37 INFO mapreduce.Job: map 100% reduce 100%  
23/01/12 18:36:37 INFO mapreduce.Job: Job job_1673261990934_0009 completed successfully  
[...]  
    Phoenix MapReduce Import  
        Upserts Done=1000000  
[...]  
  
real 1m47.828s  
user 0m19.076s  
sys 0m1.065s
```

- Much faster! Now let's scale it out and generate a whole day of data, being 10 files of 1M records each (it should only take a couple of minutes).

Unset

```
for i in {0..9}; do python3 cdr-fr.py 20230206 1000000 $i & done; wait;
echo "finished"
```

- Copy the data to your S3 bucket

Unset

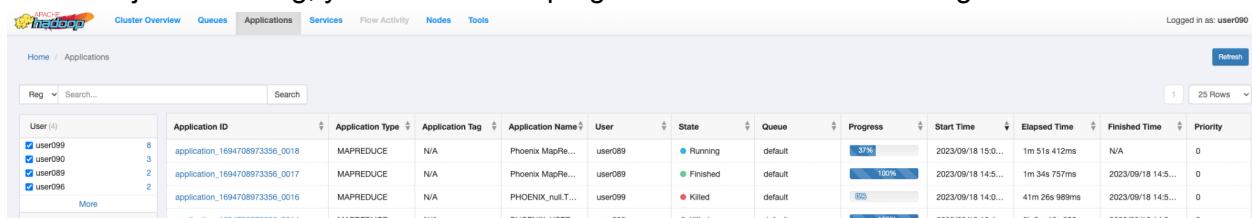
```
hdfs dfs -put ./cdr/* s3a://${BUCKET}/user/${USER}/
```

- Now import them all as done previously. A new parameter is introduced to skip system-table-checks. This helps to further improve bulk data load operations. Note the total time required for ingesting the data

Unset

```
time hbase org.apache.phoenix.mapreduce.CsvBulkLoadTool
-DPhoenix.skip.system.tables.existence.check=true --table test_${USER} --input
"s3a://${BUCKET}/user/${USER}/cdr*"
```

- While the job is running, you can observe progress in the Resource Manager



User (i)	Application ID	Application Type	Application Tag	Application Name	User	State	Queue	Progress	Start Time	Elapsed Time	Finished Time	Priority
<input checked="" type="checkbox"/> user099	8	MAPREDUCE	N/A	Phoenix MapRe...	user089	Running	default	<div style="width: 37%;">37%</div>	2023/09/18 15:0...	1m 51s 412ms	N/A	0
<input checked="" type="checkbox"/> user090	3	MAPREDUCE	N/A	Phoenix MapRe...	user089	Finished	default	<div style="width: 100%;">100%</div>	2023/09/18 14:5...	1m 34s 757ms	2023/09/18 14:5...	0
<input checked="" type="checkbox"/> user089	2	MAPREDUCE	N/A	Phoenix null.T...	user099	Killed	default	<div style="width: 0%;">0%</div>	2023/09/18 14:0...	41m 26s 989ms	2023/09/18 14:0...	0
<input checked="" type="checkbox"/> user096	2	MAPREDUCE	N/A	PHOENIX_null.T...	user099	Killed	default	<div style="width: 100%;">100%</div>	2023/09/18 19:4...	9h 8m 18s 059ms	2023/09/18 14:4...	n
More		annovation_1694708973356_0016	MAPREDUCE	N/A	phoenixnullmapred	user099	Killed	<div style="width: 0%;">0%</div>	2023/09/18 19:4...	9h 8m 18s 059ms	2023/09/18 14:4...	n

Exercise 6: COD Administration

In this exercise we will focus on the process of replicating data using HBase snapshots. This is the process native to HBase and is the core method of HBase migration in CDP Replication Manager. This also allows you to take hot backups without incurring costly writes.

Snapshots

- Let's start from a snapshot on another bucket: First look at all existing snapshots located on S3

Unset

```
hdfs dfs -ls -C s3a://${BUCKET}/snapshots/.hbase-snapshot 2>/dev/null

s3a://opdb-hol/snapshots/.hbase-snapshot/.tmp
s3a://opdb-hol/snapshots/.hbase-snapshot/CDR_Index_snapshot
s3a://opdb-hol/snapshots/.hbase-snapshot/CDR_snapshot
s3a://opdb-hol/snapshots/.hbase-snapshot/snapshot_10M
```

You should see a snapshot named snapshot_10M snapshot. This particular snapshot contains 10M records

- Copy the snapshot to your instance and rename it by appending your username

Unset

```
time hbase snapshot export --snapshot snapshot_10M \
--copy-from s3a://${BUCKET}/snapshots \
--copy-to s3a://opdb-hol/cod--bpvr11fvhhwd/hbase \
--target snapshot_10M_${USER}
```

This launches a MapReduce job called ExportSnapshot-snapshot_10M, belonging to your user id.

- Check for the snapshot by executing the list_snapshots from within the HBase shell

```
Unset
```

```
hbase shell
```

```
hbase:001:0> list_snapshots
```

```
SNAPSHOT                                     TABLE + CREATION TIME
snapshot_10M_lede1                          TEST (2023-02-08 12:59:06 UTC)
6 row(s)
Took 0.8244 seconds
=> ["snapshot_10M_lede1"]
```

Now we'll work to get the snapshot restored back to a Phoenix table.

- Using the Phoenix CLI, let's create a new table in Phoenix. Replace \${USER} with your username.

```
Unset
```

```
CREATE TABLE IF NOT EXISTS test3_${USER} (
  calling_msisdn BIGINT NOT NULL,
  session_id VARCHAR NOT NULL,
  called_msisdn BIGINT,
  date_beg VARCHAR,
  date_end VARCHAR,
  duration INTEGER,
  call_type VARCHAR,
  call_result CHAR(9),
  calling_net_type TINYINT,
  radio_calling VARCHAR,
  cell_calling INTEGER,
  lon_calling DECIMAL(10,8),
  lat_calling DECIMAL(10,8),
  called_net_type TINYINT,
  radio_called VARCHAR,
  cell_called INTEGER,
  lon_called DECIMAL(10,8),
  lat_called DECIMAL(10,8)
  CONSTRAINT pk PRIMARY KEY (calling_msisdn,session_id));
```

- Open the HBase Master UI by going to the Data Hub page and clicking on HBase UI

cod--bpvr11fvhhwd  

crn:cdp:datahub:us-west-1:f8e16d2d-57e6-43bc-8e50-75a32f7782ad:cluster:766b4610-49bd-4fa9-a03e-173ccb2fbb58 

STATUS	NODES	CREATED AT	CLUSTER TEMPLATE	STATUS REASON
 Running	 18  0  0	09/05/23, 05:39 PM CDT	7.2.17 - Operational Database: Apache HBase, Phoenix-COD-16741f81-sft	N/A

aws Environment Details

NAME opdb-hol	DATA LAKE opdb-hol-dl	CREDENTIAL wkshp-2hours-cdw	REGION us-east-2	AVAILABILITY ZONE us-east-2c
------------------	--------------------------	--------------------------------	---------------------	---------------------------------

Services

 CM UI	 HBase UI	 HBase UI	 HUE	 Job History Server
 Name Node	 Name Node	 Queue Manager	 Resource Manager	 Token Integration

Cloudera Manager Info

CM URL https://cod--bpvr11fvhhwd-gateway0.opdb-hol.z30z-14kp.cloudera.site/cod--bpvr11fvhhwd/cdp-proxy/cmft/home/	CM VERSION 7.11.0	RUNTIME VERSION 7.2.17-1.cdh7.2.17.p0.42350016	LOGS Command logs , Service logs
---	----------------------	---	---



- If it is the backup/standby master, go to the active one clicking on the link

Backup Master cod--bpvr11fvhhwd-master0.opdb-hol.z30z-14kp.cloudera.site

Current Active Master: cod--bpvr11fvhhwd-master1.opdb-hol.z30z-14kp.cloudera.site

Tasks

Show All Monitored Tasks	Show non-RPC Tasks	Show All RPC Handler Tasks	Show Active RPC Call	Show Client Operations
View as JSON				
Start Time	Description	State	Status	
Wed Sep 06 15:01:17 UTC 2023	Master startup	RUNNING (since 151hrs, 53mins, 20sec ago)	Blocking until becoming active master (since 151hrs, 53mins, 20sec ago)	



Find your table in the HBase tables list, there's only one region, as expected.

default	TEST_USER099	ENABLED	1

When you create a table in Phoenix, all the underlying table properties are created in HBase. So, you should be able to view the table in HBase as well as in Phoenix. Because of this, we first have to delete it from HBase but keep the schema defined in Phoenix.

- While in the HBase shell, drop the underlying HBase table by using the `disable` and `drop` commands. Remember to use all caps for the table name.

```
Unset  
hbase:001:0> disable 'TEST3_${USER}'  
  
Took 1.8049 seconds  
  
hbase:002:0> drop 'TEST3_${USER}'  
  
Took 1.2282 seconds
```

- Now we will use the `clone_snapshot` command to restore the snapshot to the newly created table. Be sure to use uppercase for the table name. Replace <USERNAME> with your username.

```
Unset  
hbase:003:0> clone_snapshot 'snapshot_10M_${USER}', 'TEST3_${USER}'  
  
Took 8.2659 seconds
```

- Validate in the HBase UI that your table took the number of regions that belong to the snapshot

default	TEST3_USER099	ENABLED	3
---------	---------------	---------	---

- Finally, check the results in Phoenix by executing a `SELECT *`

Unset

```
0: jdbc:phoenix:> select count(*) from TEST3_${USER};

+-----+
| COUNT(1) |
+-----+
| 10000000 |
+-----+
1 row selected (57.166 seconds)
```

NOTE: If you do that on your existing (populated) table, you'll have exceptions due to outdated cache boundaries which are stored in the Phoenix meta table.

Unset

```
Caused by: org.apache.hadoop.hbase.ipc.RemoteWithExtrasException:
org.apache.hadoop.hbase.DoNotRetryIOException: ERROR 1108 (XCL08): Cache of region
boundaries are out of date. tableName=TEST2_LJE
    at
o.a.p.c.BaseScannerRegionObserver.throwIfScanOutOfRegion(BaseScannerRegionObserver.j
ava:211)
    at
o.a.p.c.BaseScannerRegionObserver.preScannerOpen(BaseScannerRegionObserver.java:239)
    at
o.a.p.c.UngroupedAggregateRegionObserver.preScannerOpen(UngroupedAggregateRegionObse
rver.java:334)
    [...]
```

For that, delete the region boundaries in Phoenix:

Unset

```
0: jdbc:phoenix:> DELETE FROM SYSTEM.STATS WHERE PHYSICAL_NAME='TEST3_${USER}' ;
```

Understanding Your Data

Let's do a bit of analytics to understand the data. In the Phoenix shell, execute the following queries (replace \${USER} with your username):

1. `SELECT COUNT(DISTINCT(calling_msisdn)) FROM test3_${USER};`
2. `SELECT calling_msisdn,COUNT(*) AS nb FROM test3_${USER} GROUP BY calling_msisdn ORDER BY nb DESC LIMIT 5;`
3. `SELECT AVG(compteur) FROM (SELECT calling_msisdn, COUNT(*) AS compteur FROM test3_${USER} GROUP BY calling_msisdn);`

Unset

```
# how many different phone numbers has an activity
0: jdbc:phoenix:> select count(distinct(calling_msisdn)) from test3_${USER};

+-----+
| DISTINCT_COUNT(CALLING_MSISDN) |
+-----+
| 7555696                         |
+-----+
1 row selected (93.106 seconds)

# nb of records by phone number
0: jdbc:phoenix:> select calling_msisdn,count(*) as nb from test3_${USER}
group by calling_msisdn order by nb desc LIMIT 5;

+-----+----+
| CALLING_MSISDN | NB  |
+-----+----+
| 33718139793   | 8   |
| 33760153496   | 8   |
| 33725158497   | 8   |
| 33632141625   | 7   |
| 33797117678   | 7   |
+-----+----+
```

```

10 rows selected (350.031 seconds)

# how many calls by phone number in average
0: jdbc:phoenix:> SELECT AVG(compteur) FROM (select calling_msisdn, count(*) as
compteur from test3_${USER} group by calling_msisdn);

+-----+
| AVG(COMPTEUR) |
+-----+
| 1.3235       |
+-----+
1 row selected (166.842 seconds)

```

You might realize by now that Phoenix is not designed for analytics. Phoenix is designed for OLTP (Online Transactional Processing). The primary focus of Phoenix is to support very high speed and high volume data records handling, i.e. insert/update/read/delete records etc. with ability to support billions of records with a horizontal scaling architecture. Phoenix also supports a high number of concurrent connections and queries serving multiple clients.

When you combine Phoenix with COD, you get a full OLTP solution that is easy to deploy, easy to administer, and easy to access.

Our customers are using it to successfully

1. deliver high traffic web/ mobile applications
2. deliver CDC pipeline, where transactional records from operational systems are streamed to Phoenix/ HBase
3. implementing real-time streaming use-cases, where data from multiple sources are ingested/updated to Phoenix.
4. Implementing streaming transformation use-cases that enrich billions of records and updates to Phoenix.
5. Utilized in streaming use cases to enrich data using fast lookup on phoenix table
6. Batch jobs are loading data to phoenix/hbase using bulk load utility to phoenix/hbase and it is accessed by web application. Thus allows customers to meet requirements to provide access to billions of transactions.

There are several reasons why Phoenix and COD make a good fit for these use cases:

1. HBase cells contain binary data, it doesn't have any meaning per se
2. Simple count(*) is doing a full scan of the table, going row by row to increment a counter...

From an architecture perspective, you might understand the following:

1. The **row key design** in HBase (or Primary Key in Phoenix) is the most important thing.
 2. The row key can't be changed once defined. You can add columns but you cannot update the row keys so it is important to give special attention to them when you build your table.
 3. Any query in HBase/Phoenix has to take the row key as the performance factor:
 - a. row key being MSISDN only is the best choice
 - b. row key being MSISDN and DATE would limit drastically the number of rows being scanned, if you're doing queries over period of times and not the whole history
- Let's do some counting now. In the Phoenix shell execute a `SELECT COUNT(*)`.

```
Unset
0: jdbc:phoenix:> select count(*) from test3_${USER};

+-----+
| COUNT(1) |
+-----+
| 100000000 |
+-----+
1 row selected (47.053 seconds)
```

It's taking a lot of time since it's a full table scan. Counting a table with billions of rows would probably take hours!

There are several MapReduce utilities in HBase to help you do basic stuff, and counting rows is one of them.

NOTE: The HBase table row number is the same as the Phoenix table!

- Let's test the HBase utility. Execute the `hbase rowcount` command from the HBase CLI.

Unset

```
hbase rowcounter TEST3_${USER}
```

- (Optional) You can check the MapReduce job in the YARN UI

The screenshot shows the Hadoop MapReduce Job Overview UI. The top navigation bar includes the Hadoop logo and the job ID: **MapReduce Job job_1675867773588_0002**. A message "Logged in as: ledel" is displayed. On the left, a sidebar menu lists: Cluster, Application, Job (selected), Overview, Counters, Configuration, Map tasks, Reduce tasks, AM Logs, and Tools. The main content area displays the "Job Overview" table with the following data:

Job Overview						
Job Name:	rowcounter_TEST3_LJE					
User Name:	ledel					
Queue Name:	default					
State:	RUNNING					
Uberized:	false					
Started:	Wed Feb 08 20:14:35 UTC 2023					
Elapsed:	58sec					

Below this is the "ApplicationMaster" table:

Attempt Number	Start Time	Node	Logs
1	Wed Feb 08 20:14:31 UTC 2023	cod-hc6adqjch28c-worker1.tko-cdp.a465-9g4k.cloudera.site:8044	https://cod-hc6adqjch28c-gateway0.tko-cdp.a465-9g4k.cloudera.site:443/cod-hc6adqjch28c/cdp-proxy/yarnui/v2/logs

Finally, there is a "Task Type" table:

Task Type	Progress	Total	Pending	Running	Complete
Map	<div style="width: 33%;"></div>	3	0	2	1
Reduce	<div style="width: 0%;"></div>	0	0	0	0

Sub-tables for "Maps" and "Reduces" show the following counts:

Attempt Type	New	Running	Failed	Killed	Successful
Maps	0	2	0	0	1
Reduces	0	0	0	0	0

Notice that the job is taking as many mappers as my table has regions, in this case 3.

At the very end, you have your result, 10 million rows

```

Map-Reduce Framework
    Map input records=10000000
    Map output records=0
    Input split bytes=638
    Spilled Records=0
    Failed Shuffles=0
    Merged Map outputs=0
    GC time elapsed (ms)=516
    CPU time spent (ms)=65980
    Physical memory (bytes) snapshot=2510913536
    Virtual memory (bytes) snapshot=19547738112
    Total committed heap usage (bytes)=4019191808
    Peak Map Physical memory (bytes)=965447680
    Peak Map Virtual memory (bytes)=6763069440
HBaseCounters
    BYTES_IN_REMOTE_RESULTS=191518734
    BYTES_IN_RESULTS=740000000
    MILLIS_BETWEEN_NEXTS=175727
    NOT_SERVING_REGION_EXCEPTION=0
    REGIONS_SCANNED=3
    REMOTE_RPC_CALLS=25881
    REMOTE_RPC_RETRIES=0
    ROWS_FILTERED=1000001
    ROWS_SCANNED=11000001
    RPC_CALLS=100002
    RPC_RETRIES=0
org.apache.hadoop.hbase.mapreduce.RowCounter$RowCounterMapper$Counters
    ROWS=10000000
File Input Format Counters
    Bytes Read=0
File Output Format Counters
    Bytes Written=0
23/02/08 20:16:06 INFO impl.MetricsSystemImpl: Stopping s3a-file-system metrics system

```

The bigger the table, the more regions you have, the more mappers you have, the more you're saving time over a SELECT COUNT(*) result in Phoenix or HBase.

NOTE: Use the following query to identify the approximate number of rows (which is usually less than the actual number) from the Phoenix table rather than the exact number of rows to get an understanding of table size.

```

SELECT PHYSICAL_NAME, COLUMN_FAMILY, SUM(GUIDE_POSTS_ROW_COUNT)
FROM SYSTEM STATS WHERE PHYSICAL_NAME='<table-name>' GROUP BY
PHYSICAL_NAME, COLUMN_FAMILY;

```

Indexing

Selecting data on a DATE (or any field that is not part of the primary key) will result in a full scan, thus very long query time. Remember that Phoenix is built on HBase, which is a Key->Value store! In this exercise we will build indexes on our test table.

- Execute the following command in the Phoenix shell
CREATE INDEX index_date_\${USER} ON test (date_beg) ASYNC;

```
Unset
[edge0 ~]$ phoenix-sqlline

0: jdbc:phoenix:> CREATE INDEX index_date_${USER} ON TEST3_${USER}(date_beg)
ASYNC;

No rows affected (4.387 seconds)

0: jdbc:phoenix:> !tables

+-----+-----+-----+-----+-----+
| TABLE_SCHEM | TABLE_NAME | TABLE_TYPE | INDEX_STATE | SALT_BUCKETS |
| INDEX_TYPE |           |
+-----+-----+-----+-----+-----+
| CDR          | INDEX_DATE_xxx | INDEX          | BUILDING    | null         |
| GLOBAL       |           |               |             |             |
| SYSTEM       | CATALOG      | SYSTEM TABLE |           | null         |
|             |           |               |             |             |
| SYSTEM       | CHILD_LINK   | SYSTEM TABLE |           | null         |
|             |           |               |             |             |
| SYSTEM       | FUNCTION     | SYSTEM TABLE |           | null         |
|             |           |               |             |             |
| SYSTEM       | LOG          | SYSTEM TABLE |           | 32           |
|             |           |               |             |             |
| SYSTEM       | MUTEX        | SYSTEM TABLE |           | null         |
|             |           |               |             |             |
| SYSTEM       | SEQUENCE     | SYSTEM TABLE |           | null         |
|             |           |               |             |             |
| SYSTEM       | STATS        | SYSTEM TABLE |           | null         |
|             |           |               |             |             |
| SYSTEM       | TASK          | SYSTEM TABLE |           | null         |
|             |           |               |             |             |
```

```
|           | test          | TABLE          |           | null          |
|           +-----+          +-----+          +-----+
+-----+          +-----+          +-----+
-----+
```

As our index builds, note that we built it as asynchronous (not dependent on the writes), so we now need to trigger the initial build.

- On your edge node, create the directory where the index files will be located

Unset

```
hdfs dfs -mkdir s3a://opdb-hol/cod--bpvr11fvhhwd/hbase/index_cdr_${USER}
```

- Now launch the job using the IndexTool

Unset

```
hbase org.apache.phoenix.mapreduce.index.IndexTool \
--data-table TEST3_${USER} --index-table INDEX_DATE_${USER} \
--output-path s3a://opdb-hol/cod--bpvr11fvhhwd/hbase/index_cdr_${USER}
```

While mappers are finishing (this could take a minute), understand what the index is

Unset

```
[edge0 ~]$ phoenix-sqlline
```

```
0: jdbc:phoenix:> select * from INDEX_DATE_${USER} limit 5;
```

```
+-----+-----+-----+
+-----+-----+-----+
|      0:DATE_BEG      | :CALLING_MSISDN | :SESSION_ID
|
```

```

+-----+-----+
+
| 2023-01-01 00:01:00 | 003325566187884 | 42deb97-a312-466d-b724-12b68287315c
|
| 2023-01-01 00:01:00 | 003332841174595 | 3b4c703e-3ce4-4f1e-a032-841032e160d2
|
| 2023-01-01 00:01:00 | 003337013187940 | 028a0fef-38b4-401f-9605-fad5ccdb39e9
|
| 2023-01-01 00:01:00 | 0033610121797 | 2bc2d5d5-e1ca-4041-9d0b-3398f7d87449
|
| 2023-01-01 00:01:00 | 0033610127836 | 0a7eb95b-a773-411f-8894-8328ad0c90fe
|
+-----+-----+
+

```

Our INDEX is a table with indexed fields as primary key, and primary key of our source table as value.

Check how many rows do we have in the tables

```

Unset
[edge0 ~]$ phoenix-sqlline

0: jdbc:phoenix:> select count(*) from TEST3_${USER};

+-----+
| COUNT(1) |
+-----+
| 1000000 |
+-----+
1 row selected (7.86 seconds)

0: jdbc:phoenix:> select count(*) from index_date_${USER};

+-----+
| COUNT(1) |
+-----+
| 1000000 |
+-----+

```

```
1 row selected (6.933 seconds)
```

The indexing process can take time: it should be around 5 minutes, but for a 1.4B rows table, it took 6 hours on a 5 workers COD instance.

Resiliency

So, what happens when a RegionServer goes down?

- In the Phoenix CLI, let's create a thin table with only 2 records

```
Unset
```

```
CREATE TABLE TEST_${USER} ( id BIGINT not null primary key, name VARCHAR);

UPSERT INTO TEST_${USER} VALUES(1, 'bar');

UPSERT INTO TEST_${USER} VALUES(2, 'foo');

SELECT * FROM TEST_${USER};

+----+-----+
| ID | NAME |
+----+-----+
| 1  | bar  |
| 2  | foo  |
+----+-----+
2 rows selected (0.014 seconds)
```

Since the table is very thin, it only has a single region. Let see where this region is located and kill the RegionServer hosting that region to see the resiliency.

- You can check the region in HBase UI, or you can use HBase shell:

```
Unset
```

```
list_regions 'TEST_${USER}'
```

REGION_NAME	START_KEY	END_KEY	SIZE	REQ	LOCALITY	SERVER_NAME
cod--bpvr11fvhhwd-worker1.opdb-hol.z30z-14kp.cloudera.site,16020,1694853593864 563163ab7d.			0	3	0.0	
1 rows						Took 0.5798 seconds

Here, the SERVER_NAME hosting the region is

cod--bpvr11fvhhwd-worker1.opdb-hol.z30z-14kp.cloudera.site,16020,1694853593864

The ENCODED_REGIONNAME is

TEST_099,,1694875378606.6b6554f7c85e9f90626bb7563163ab7d.

Save this ENCODED_REGIONNAME for future reference. You can also notice start and end keys if that makes sense (ie when you got several regions for the table) and the number of requests for example.

For everybody to be on the same RegionServer (you don't want to kill all the RegionServers!), let's force assign that region to RegionServer on worker0.

During the life of the cluster, HBase master is making decisions on how to place regions, and allow for every RegionServer to serve globally the same number of regions. We do not want all regions of the same table being located on the same RegionServer, this would cause *hotspotting*.

- Now let's identify our RegionServers

```
Unset
```

```
hbase:004:0> list_liveservers
```

```
SERVERNAME
cod--bpvr11fvhhwd-worker0.opdb-hol.z30z-14kp.cloudera.site,16020,1694853596218
cod--bpvr11fvhhwd-worker1.opdb-hol.z30z-14kp.cloudera.site,16020,1694853593864
cod--bpvr11fvhhwd-worker2.opdb-hol.z30z-14kp.cloudera.site,16020,1694853594878
cod--bpvr11fvhhwd-worker3.opdb-hol.z30z-14kp.cloudera.site,16020,1694853594721
cod--bpvr11fvhhwd-worker4.opdb-hol.z30z-14kp.cloudera.site,16020,1694853594488
5 row(s)
Took 0.0216 seconds
```

- Copy worker0 line (<EDGENODE FQDN>), it is our destination SERVER_NAME
- Use the "move" command, the syntax being move 'ENCODED_REGIONNAME', 'SERVER_NAME'. Replace the parameters accordingly and launch, still in HBase shell:

```
Unset
move '6b6554f7c85e9f90626bb7563163ab7d',
'bpvr11fvhhwd-worker0.opdb-hol.z30z-14kp.cloudera.site,16020,1675853620633'
Took 3.5633 seconds
```

- Let's check if the move worked, the SERVER_NAME must be worker0

```
Unset
list_regions 'TEST_{USER}'
```

Now ask for RegionServer to be killed by the presenter when everyone's ready!

- Do a SELECT again in Phoenix

```
Unset
0: jdbc:phoenix:> SELECT * FROM TEST_${USER};

+----+----+
| ID | NAME |
+----+----+
| 1  | bar  |
| 2  | foo  |
+----+----+
2 rows selected (6.078 seconds)
```

It took 6s because of reassigning the region on another server! The region is now hosted by another worker. All regions that were hosted on worker0 have been reassigned to other RegionServers until worker() joins again and a new regions repartition can be triggered by the Master.

Create a Large Table

Now that you understand the data and some commands, let's go with sample production data that represents a much larger dataset. That means here we'll load 1.4 billion lines in a single table!

- First, let's take a look at HDFS.

```
Unset
[edge0 ~]$ hdfs dfs -ls /

Found 4 items
drwxr-xr-x  - hbase hbase          0 2023-01-10 17:29 /hbase-wals
drwxrwxrwt  - hdfs supergroup      0 2023-01-10 17:32 /tmp
drwxr-xr-x  - hdfs supergroup      0 2023-01-11 16:07 /user
drwxr-xr-x  - hdfs supergroup      0 2023-01-10 17:29 /yarn
```

Why do we have only those directories? No /hbase? The reason, HDFS is only for WALs (write-ahead logs). The data files are on S3!

As you might recall, your COD instance data is located in `opdb-hol`, with `cod--bpvr11fvhhwd` instance, in directory `hbase`. The COD instance name is the same as the underlying data hub being used, that you can find in your environment (storage).

Unset

```
hdfs dfs -ls -C s3a://opdb-hol/cod--bpvr11fvhhwd/hbase 2>/dev/null
```

```
s3a://opdb-hol/cod--bpvr11fvhhwd/hbase/.hbase-snapshot
s3a://opdb-hol/cod--bpvr11fvhhwd/hbase/.hbck
s3a://opdb-hol/cod--bpvr11fvhhwd/hbase/.tmp
s3a://opdb-hol/cod--bpvr11fvhhwd/hbase/MasterData
s3a://opdb-hol/cod--bpvr11fvhhwd/hbase/archive
s3a://opdb-hol/cod--bpvr11fvhhwd/hbase/data
s3a://opdb-hol/cod--bpvr11fvhhwd/hbase/hbase.id
s3a://opdb-hol/cod--bpvr11fvhhwd/hbase/hbase.version
s3a://opdb-hol/cod--bpvr11fvhhwd/hbase/index_cdr
s3a://opdb-hol/cod--bpvr11fvhhwd/hbase/mobdir
s3a://opdb-hol/cod--bpvr11fvhhwd/hbase/staging
```

We have a cold backup of our production table that has been stored in
`s3a://opdb-hol/snapshots`

Let's bring that snapshot to our instance

NOTE: This could take some time, around 20mn because of the amount of data. You can start over with an existing snapshot.

- Using your terminal session, execute the `ExportSnapshot` command to restore the snapshot. Replace `$ [USER]` with your username.

Unset

```
hbase org.apache.hadoop.hbase.snapshot.ExportSnapshot \
--snapshot CDR_snapshot \
--target CDR_snapshot_${USER} \
--copy-from s3a://opdb-hol/snapshots \
--copy-to s3a://${EDGENODE FQDN}
```

This is again launching a MapReduce job, you can follow it in the console and/or check that in Resource Manager. Click Enter every 10 minutes or so if you don't want to be disconnected from your session!

You can recall cloning snapshots in on HBase level, so the process will be:

1. Create a table through Phoenix
 2. Disable it (so we can't read/write to it anymore)
 3. Drop the table
 4. Restore the snapshot on the table
- In the HBase shell, check to see if the snapshot is there

Unset

```
hbase:014:0> list_snapshots

SNAPSHOT                                TABLE + CREATION TIME
CDR_snapshot                             CDR:CDR (2023-02-07 11:59:05 UTC)
CDR_snapshot_lede1                        CDR:CDR (2023-02-07 11:59:05 UTC)
snapshot_10M_lede1                        TEST (2023-02-08 12:59:06 UTC)
test_snapshot_01                           TEST (2023-02-07 12:52:50 UTC)

7 row(s)
Took 0.5299 seconds
=> ["CDR_snapshot", "CDR_snapshot_lede1", "snapshot_10M_lede1", "test_snapshot_01"]
```

NOTE: The "production" way is to use snapshots "as is" but here we're cloning tables from a snapshot. You can also *restore* a snapshot.

When you do a simple `restore_snapshot <SNAPSHOT>`, it just restores the table as is. Here you're restoring to a new table, thus you have to drop the table in HBase. The usual process is simpler as you're already having your existing table in Phoenix. Here are the commands you would execute in the HBase shell

```
Unset  
disable <TABLE>  
restore_snapshot <SNAPSHOT>  
enable <TABLE>
```

As we're all working on the same instance, we can't just `clone_snapshot` because it would result in the same CDR:CDR table.

We need to define our Phoenix schema first. Storing data like we're doing also has some constraints. We're using HBase, so the row key (or the primary key in Phoenix) design is having a lot of impact.

We can see a problem here: the distribution of the MSISDN is obviously not homogeneous! the quantity of calling MSISDNs between 336000000 and 336100000 is (way probably) not the same as in the range 337500000 and 337600000 for example. The latest attributed MSISDNs are starting with 337, 336 are "old" numbers.

- Let's first create the table in Phoenix. You can run this from the Phoenix CLI or from HUE

```
Unset  
CREATE SCHEMA ${USER};  
  
CREATE TABLE IF NOT EXISTS ${USER}.CDR (  
calling_msisdn VARCHAR NOT NULL,  
session_id VARCHAR NOT NULL,  
called_msisdn VARCHAR,  
date_beg VARCHAR,  
date_end VARCHAR,  
duration INTEGER,  
call_type VARCHAR,  
call_result CHAR(7),  
calling_net_type TINYINT,
```

```
radio_calling VARCHAR,  
cell_calling INTEGER,  
lon_calling DECIMAL(10,8),  
lat_calling DECIMAL(10,8),  
called_net_type TINYINT,  
radio_called VARCHAR,  
cell_called INTEGER,  
lon_called DECIMAL(10,8),  
lat_called DECIMAL(10,8)  
CONSTRAINT pk PRIMARY KEY (calling_msisdn,session_id))  
SALT_BUCKETS = 40;
```

phoenix Add a name... Add a description...

```
1 CREATE SCHEMA ${USER};  
2  
3 CREATE TABLE IF NOT EXISTS ${USER}.CDR (calling_msisdn VARCHAR NOT NULL, session_id VARCHAR NOT NULL, called_msisdn VARCHAR,  
4 date_beg VARCHAR, date_end VARCHAR, duration INTEGER, call_type VARCHAR, call_result CHAR(7), calling_net_type TINYINT,  
5 radio_calling VARCHAR, cell_calling INTEGER, lon_calling DECIMAL(10,8), lat_calling DECIMAL(10,8), called_net_type TINYINT,  
6 radio_called VARCHAR, cell_called INTEGER, lon_called DECIMAL(10,8), lat_called DECIMAL(10,8)  
7 CONSTRAINT pk PRIMARY KEY (calling_msisdn,session_id)) SALT_BUCKETS = 40;
```

C 2/2

▶ USER user001

Query History Saved Queries

- Notice the SALT_BUCKETS column has been populated

TABLE_CAT	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE	REMARKS	TYPE_NAME	SELF_REFERENCING_COL_NAME	REF_GENERATION	INDEX_STATE	IMMUTABLE_ROWS	SALT_BUCKETS	MULTI_TENANT	VIEW_STATEMENT
	SYSTEM	CATALOG	SYSTEM TABLE						false	null	false	
	SYSTEM	CHILD_LINK	SYSTEM TABLE						false	null	false	
	SYSTEM	FUNCTION	SYSTEM TABLE						false	null	false	
	SYSTEM	LOG	SYSTEM TABLE						true	32	false	
	SYSTEM	MUTEX	SYSTEM TABLE						true	null	false	
	SYSTEM	SEQUENCE	SYSTEM TABLE						false	null	false	
	SYSTEM	STATS	SYSTEM TABLE						false	null	false	
	SYSTEM	TASK	SYSTEM TABLE						false	null	false	
	CDR	CDR	TABLE						false	20	false	

NOTE: You can't do that in Hue, but you can see the statistics like SALT_BUCKETS:

Unset

```
select * from SYSTEM.CATALOG where TABLE_SCHEMA=UPPER( '${USER}' );
```

- Now go see what the table looks like from an HBase point of view. Again, go into your Data Hub instance and open the HBase Master UI.

Data Hubs / cod-bpvr11fhhhwd / Event History

The screenshot shows the Cloudera Manager interface for a specific Data Hub cluster. At the top, it displays the cluster name 'cod--bpvr11fhhhwd' along with status indicators (Running, 18 nodes, 0 failed, 0 pending) and creation details (09/05/23, 05:39 PM CDT). Below this, the 'Environment Details' section shows the AWS environment configuration, including the name 'opdb-hol', data lake 'opdb-hol-dl', credential 'wkshp-2hours-cdw', region 'us-east-2', and availability zone 'us-east-2c'. The 'Services' section lists various components: CM UI, Name Node, HBase UI (with a red arrow pointing to it), Name Node, Queue Manager, HUE, Resource Manager, Job History Server, and Token Integration. At the bottom, 'Cloudera Manager Info' provides the CM URL (<https://cod-bpvr11fhhhwd-gateway0.opdb-hol.z30z-14kp.cloudera.site/cod-bpvr11fhhhwd/cdp-proxy/cmf/home/>), CM version '7.11.0', runtime version '7.2.17-1.cdh7.2.17.p0.42350016', and logs links for 'Command logs' and 'Service logs'.

Master cod-hc6adqjch28c-master1.tko-cdp.a465-9q4k.cloudera.site

Region Servers

ServerName	Base Stats	Memory	Requests	Storefiles	Compactions	Replications	# Start time	# Last contact	# Version	# Requests Per Second	# Num. Regions
cod-hc6adqjch28c-worker0.tko-cdp.a465-9q4k.cloudera.site,16020,167562984689							Sun Feb 05 20:34:24 UTC 2023	2 s	2.4.6.7.2.16.0-287	0	29
cod-hc6adqjch28c-worker1.tko-cdp.a465-9q4k.cloudera.site,16020,167562984754							Sun Feb 05 20:34:24 UTC 2023	0 s	2.4.6.7.2.16.0-287	0	29
cod-hc6adqjch28c-worker2.tko-cdp.a465-9q4k.cloudera.site,16020,1675629850877							Sun Feb 05 20:34:26 UTC 2023	0 s	2.4.6.7.2.16.0-287	0	28
cod-hc6adqjch28c-worker3.tko-cdp.a465-9q4k.cloudera.site,16020,167562985061							Sun Feb 05 20:34:25 UTC 2023	0 s	2.4.6.7.2.16.0-287	0	29
cod-hc6adqjch28c-worker4.tko-cdp.a465-9q4k.cloudera.site,16020,167562987107							Sun Feb 05 20:34:27 UTC 2023	1 s	2.4.6.7.2.16.0-287	0	29
Total:										0	144

Backup Masters

ServerName	Port	Start Time
cod-hc6adqjch28c-master0.tko-cdp.a465-9q4k.cloudera.site	16000	Sun Feb 05 20:34:00 UTC 2023
Total:1		

Tables

Tables		User Tables	System Tables	Snapshots
14 table(s) in set. [Details]. Click count below to see list of regions currently in 'state' designated by the column title. For 'Other' Region state, browse to hbase:meta and adjust filter on 'Meta Entries' to query on states other than those listed here. Queries may take a while if the hbase:meta table is large.				
Namespace	Name	# State	# Regions	Description
CDR	CDR	ENABLED	40	OPEN OPENING CLOSED CLOSING OFFLINE SPLIT Other

Notice the \${USER}.CDR table with 40 regions is here. Also notice the default parameters that has been set by Phoenix: the coprocessors, etc

- On Table details, look at the regions in your **CDR** table

Table Regions

Base Stats	Localities	Compactions	Name(40)	Region Server	ReadRequests (40)	WriteRequests (0)	StorefileSize (0 MB)	Num.Storefiles (0)	MemSize (0 MB)	Start Key	End Key	Region State
USER001:CDR,,1675894371686.431f07ca89f6ae632b95c9112ac7497a.ite:16030	cod-hc6adqjch28c-worker11.tko-cdp.a465-9q4k.cloudera.sit	1	0	0 MB	0	0 MB	\x01\x00 \x00\x00	OPEN				
USER001:CDR,\x01\x00\x00,1675894371686.48943d5ba40f75fbe67ad214ee651d24b.ite:16030	cod-hc6adqjch28c-worker2.tko-cdp.a465-9q4k.cloudera.sit	1	0	0 MB	0	0 MB	\x01\x00 \x00\x00	OPEN				
USER001:CDR,\x02\x00\x00,1675894371686.e9f7d1e8306117097134452f1ab5e5c9.ite:16030	cod-hc6adqjch28c-worker0.tko-cdp.a465-9q4k.cloudera.sit	1	0	0 MB	0	0 MB	\x02\x00 \x00\x00	OPEN				
USER001:CDR,\x03\x00\x00,1675894371686.8f4a74a8f8236a25dd7e6b8fa.c6c92ed.ite:16030	cod-hc6adqjch28c-worker1.tko-cdp.a465-9q4k.cloudera.sit	1	0	0 MB	0	0 MB	\x03\x00 \x00\x00	OPEN				
USER001:CDR,\x04\x00\x00,1675894371686.6e5913062ec7637d1b801a8c20fbed6.ite:16030	cod-hc6adqjch28c-worker11.tko-cdp.a465-9q4k.cloudera.sit	1	0	0 MB	0	0 MB	\x04\x00 \x00\x00	OPEN				
USER001:CDR,\x05\x00\x00,1675894371686.9a7c2e7c034bf61b174727a.d2918b17.ite:16030	cod-hc6adqjch28c-worker10.tko-cdp.a465-9q4k.cloudera.sit	1	0	0 MB	0	0 MB	\x05\x00 \x00\x00	OPEN				

- Now proceed with the restore of the snapshot. In HBase shell, execute the following commands

Unset

```
disable '${USER}:CDR'
```

```
drop '${USER}:CDR'
```

```
clone_snapshot 'CDR_snapshot_${USER}', '${USER}:CDR'
```

Took 18.6040 seconds

There it is! 18 seconds for 1.4 billion lines...

- Refresh your HBase UI, see the data is here

Table Regions

Name(40)	Region Server	ReadRequests (0)	WriteRequests (0)	StorefileSize (311.65 GB)	Num.Storefiles (68)	MemSize (0 MB)	Star Key
USER001:CDR,,1675000543245.6d93af9e7088e418db5568c8f16f65c8.	cod-hc6adqjch28c-worker1.tko-cdp.a465-9q4k.cloudera.sit e:16030	0	0	7.80 GB	3	0 MB	
USER001:CDR,\x01\x00\x00,1675000543245.4c1ed9d94b322034a0c50db4d3ba3e11.	cod-hc6adqjch28c-worker0.tko-cdp.a465-9q4k.cloudera.sit e:16030	0	0	7.80 GB	3	0 MB	\x01\x00
USER001:CDR,\x02\x00\x00,1675000543245.e0f4929e177e9905d59a1cf9a c1a13c4.	cod-hc6adqjch28c-worker1.tko-cdp.a465-9q4k.cloudera.sit e:16030	0	0	7.80 GB	3	0 MB	\x02\x00
USER001:CDR,\x03\x00\x00,1675000543245.fa7181d758ac60550fb2730aa8e80f01.	cod-hc6adqjch28c-worker0.tko-cdp.a465-9q4k.cloudera.sit e:16030	0	0	7.79 GB	1	0 MB	\x03\x00
IISFRN01:CDR,\vn01\vn00\vn0_1675000543245_af553r9r4Rra1r41Ra047Rh1	cod-hc6adqjch28c-worker2.tko-cdp.a465-9q4k.cloudera.sit e:16030	0	0	7.78 GB	1	0 MB	\vn01

- Let's count rows! In your terminal window, execute the `hbase rowcounter` command

Unset

```
hbase rowcounter ${USER}:CDR
```

If you look at the application, it again makes sense. Since our table has 40 salt buckets, then we're having 40 regions, and 40 mappers!

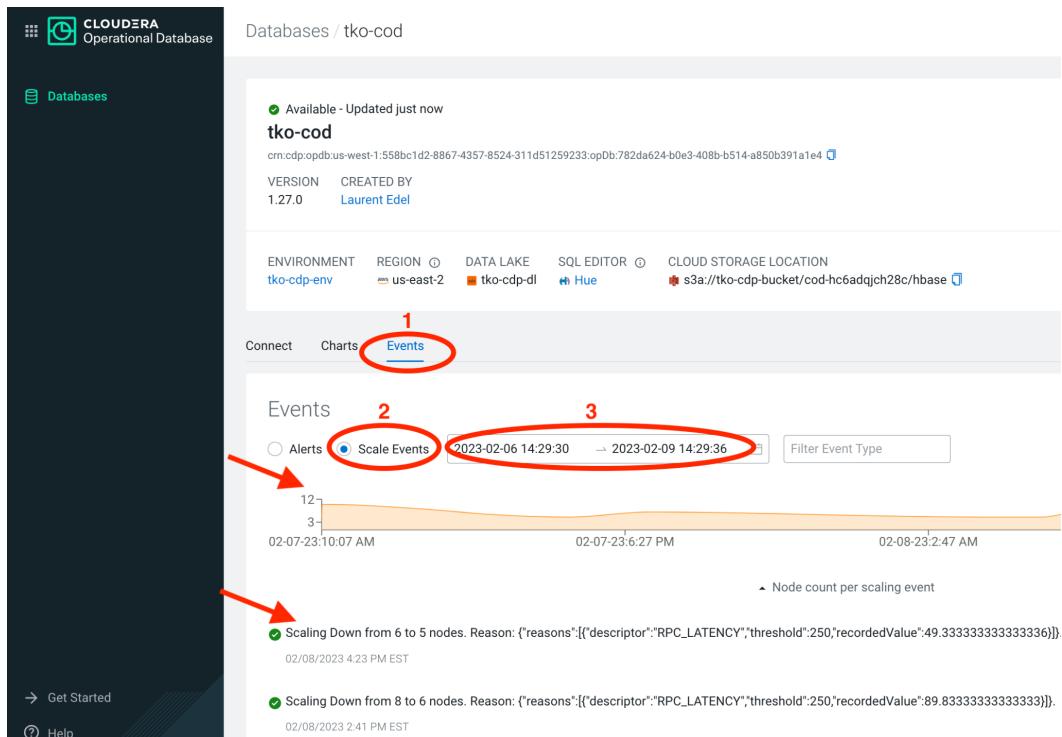
Exercise 7: Auto-Scaling

How Does COD Auto-scale?

You can monitor the events related to scaling in COD. We first need to create an event by doing the following in the Events window in our COD database.

1. Select Events
2. Select Scaled Events
3. Enter Start and End dates

Below that you'll see the Scaling Up or or Scaling down events and a graph. Typical scaling events you'll see with COD auto-scaling are RPC_LATENCY and HDFS_USAGE.



You can also filter on the event types by selecting a type of events - either Scaling Up, Scaling Down, or both.

The screenshot shows the 'Events' tab in the CloudBees DevCenter interface. At the top, there are tabs for 'Connect', 'Charts', and 'Events'. Below the tabs, there are two radio buttons: 'Alerts' (unchecked) and 'Scale Events' (checked). A date range selector shows '2023-02-06 14:29:30' to '2023-02-09 14:29:36'. To the right of the date range is a search bar with the placeholder 'Scaling Up x |' which is circled in red. A red arrow points from the left towards this search bar. Below the search bar, there are two log entries:

- Scaling Up from 10 to 12 nodes. Reason: {"reasons": [{"descriptor": "RPC_LATENCY", "threshold": 250, "recordedValue": 310.26666666666665}]}
02/08/2023 8:54 AM EST
- Scaling Up from 5 to 10 nodes. Reason: {"reasons": [{"descriptor": "RPC_LATENCY", "threshold": 250, "recordedValue": 666.1833333333333}]}
02/08/2023 7:32 AM EST

Auto-scaling in Action

Apache JMeter is a free open-source java-based application which is used for designing and running comprehensive performance tests of web applications, databases, SOAP and REST Web Services, FTP, LDAP and mail servers, TCP and SMTP protocols. JMeter can emulate a heavy workload on the server and help analyze performance under different test scenarios. JMeter comes with a GUI which is used for designing and running tests while measuring a variety of performance metrics such as average, min and max response time, standard deviation, throughput, and etc. Adding test plan elements (thread groups, listeners, samplers etc.) to the test plan can be easily done by right-clicking on an element in the tree and choosing a new element from the “add” list.

One of the more useful features provided by JMeter is the ability to easily change the number of concurrent threads running a particular workload.

Once a test scenario has been designed in JMeter, it can be saved into the file with the .jmx extension. For load testing and getting optimal results, it is recommended to run an already created test scenario in a Non-GUI CLI mode by supplying a .jmx file to the JMeter executable.

NOTE: You can learn a lot more about JMeter capabilities by downloading it from <https://jmeter.apache.org/> and reading tutorials/documentation

We're going to deploy JMeter on the Edge node and run a stress test scenario on COD.

- Log on to the edge node that you deployed earlier.

Unset

```
ssh $USER@$EDGENODE
```

- Download JMeter binaries

Unset

```
wget https://dlcdn.apache.org/jmeter/binaries/apache-jmeter-5.5.tgz
tar xvf apache-jmeter-5.5.tgz
```

- Now you should have a Jmeter installed at your home directory.

Unset

```
ls -lrt
```

```
drwxr-x--- 8 pnovokshonov pnovokshonov      138 Jan  2 1970 apache-jmeter-5.5
-rw-r----- 1 pnovokshonov pnovokshonov 85476161 Jun 14 2022
apache-jmeter-5.5.tgz
```

- We need to let JMeter know which driver to use to connect to COD. You'll use Phoenix JDBC Thick driver. You can check versions, get all the required drivers and JDBC URLs by going to the COD Connect Tab and choosing Phoenix (Thick) option.

ENVIRONMENT opdb-hol REGION us-east-2 DATA LAKE opdb-hol-dl SQL EDITOR Hue GRAFANA DASHBOARD N/A CLOUD STORAGE LOCATION s3a://opdb-hol/cod-bpvr11fhhwd/hbase

Connect Charts Events

HBase HBase REST HBase Client Tarball **Phoenix (Thick)** Phoenix (Thin) Phoenix (ODBC) Phoenix Python

Usage
The Apache Phoenix Thick driver communicates directly with Apache ZooKeeper and Apache HBase. You can connect your application to the COD instance using the provided connection and configuration URLs.

Phoenix (Thick) Client Jar [Download Phoenix Client Jar](#)

Phoenix Maven URL <https://archive.cloudera.com/p/cdp-public/7.2.17.0/maven-repository>

- To get Phoenix Thick Driver switch to the Jmeter lib/ext directory and download the jar file.

```
Unset
cd ./apache-jmeter-5.5/lib/ext

wget
https://repository.cloudera.com/artifactory/cloudera-repos/org/apache/phoenix/phoenix-client-hbase-2.4/5.1.1.7.2.16.0-287/phoenix-client-hbase-2.4-5.1.1.7.2.16.0-287.jar
```

- Now you need to add 2 COD config files: **hbase-site.xml** and **core-site.xml** to the Thick Driver jar file. You can get those config files by going to the Hbase configuration tab in the COD Connect page and downloading the corresponding hbase-confi.zip file. It contains a set of HBase configuration files.

Connect Charts Events

HBase HBase REST HBase Client Tarball Phoenix (Thick) Phoenix (Thin) Phoenix (ODBC) Phoenix Python

Usage ⓘ
You can use the Apache HBase Maven URL, Apache HBase Client Version, and the Apache HBase Client Configuration URL to download a file containing the Apache HBase Client Configuration.

HBase Maven URL ⓘ
`https://archive.cloudera.com/p/cdp-public/7.2.17.0/maven-repository`

HBase Client Version ⓘ
`2.4.6.7.2.17.0-334`

HBase Client Configuration URL ⓘ
`curl -f -o "hbase-config.zip" -u "sshaw" "https://cod-bpvr11fvhhwd-gateway0.opdb-hol.z30z-14kp.cloudera.site/clouderamanager/api/v41/clusters/cod--bpvr11fvhhwd/services/hbase/clientConfig"`

> Kerberos Configuration

Unset

```
curl -f -o "hbase-config.zip" -u ${USER}  
"https://cod--bpvr11fvhhwd-gateway0.opdb-hol.z30z-14kp.cloudera.site/clouderamanager/api/v41/clusters/cod--bpvr11fvhhwd/services/hbase/clientConfig"  
  
unzip hbase-config.zip  
  
cp ./hbase-conf/hbase-site.xml .  
cp ./hbase-conf/core-site.xml .
```

Unset

```
cp hbase-site.xml /home/${USER}/apache-jmeter-5.5/lib/ext  
  
cp core-site.xml /home/${USER}/apache-jmeter-5.5/lib/ext
```

- Let's add 2 config files to the Phoenix Thick Driver jar file

Unset

```
jar uf phoenix-client-hbase-2.4-5.1.1.7.2.16.0-287.jar hbase-site.xml  
jar uf phoenix-client-hbase-2.4-5.1.1.7.2.16.0-287.jar core-site.xml
```

- Let's check that those 2 config files were successfully added to the jar file. You should see those at the end.

Unset

```
jar -tf phoenix-client-hbase-2.4-5.1.1.7.2.16.0-287.jar
```

...

```
META-INF/services/org.apache.hadoop.hbase.master.MetricsAssignmentManagerSource  
hbase-site.xml  
core-site.xml
```

Let's get the JMeter test scenario (.jmx) file and a parameter file, then upload those files to the Edge node. The **CDRLoadTest.jmx** file contains the test scenario and the **msisdn.csv** file contains a list of 10,000 MSISDN ids to be used in the test query WHERE clause.

- You can get these files from the following location and download them to your local host. Then just copy the files to your Edge node as shown below.

https://drive.google.com/drive/u/1/folders/1qUc5MWBy8gyT0hFvrYaA1_WnOBmX_a20

Unset

```
scp CDRLoadTest.jmx ${USER}@${EDGENODE}:/home/${USER}/apache-jmeter-5.5/bin  
  
scp msisdn.csv ${USER}@${EDGENODE}:/home/${USER}/apache-jmeter-5.5/bin
```

- Logon back to the Edge node and switch to the JMeter bin directory. You should find the files there.

```
Unset
```

```
cd $HOME/apache-jmeter-5.5/bin

ls -lrt
...
-rw-r----- 1 pnovokshonov pnovokshonov 160000 Feb  9 01:52 msisdn.csv
-rw-r----- 1 pnovokshonov pnovokshonov 17088 Feb  9 01:53
CDRLoadTest.jmx
```

Let's run the JMeter test and put some load on the COD.

Here is an example of running a test with 10 concurrent threads, a ramp time for those threads to become active in 5 seconds and a test duration of 120 seconds.

```
Unset
```

```
./jmeter.sh -n -t CDRLoadTest.jmx -Jthreads=10 -Jramp=5 -Jduration=120
```

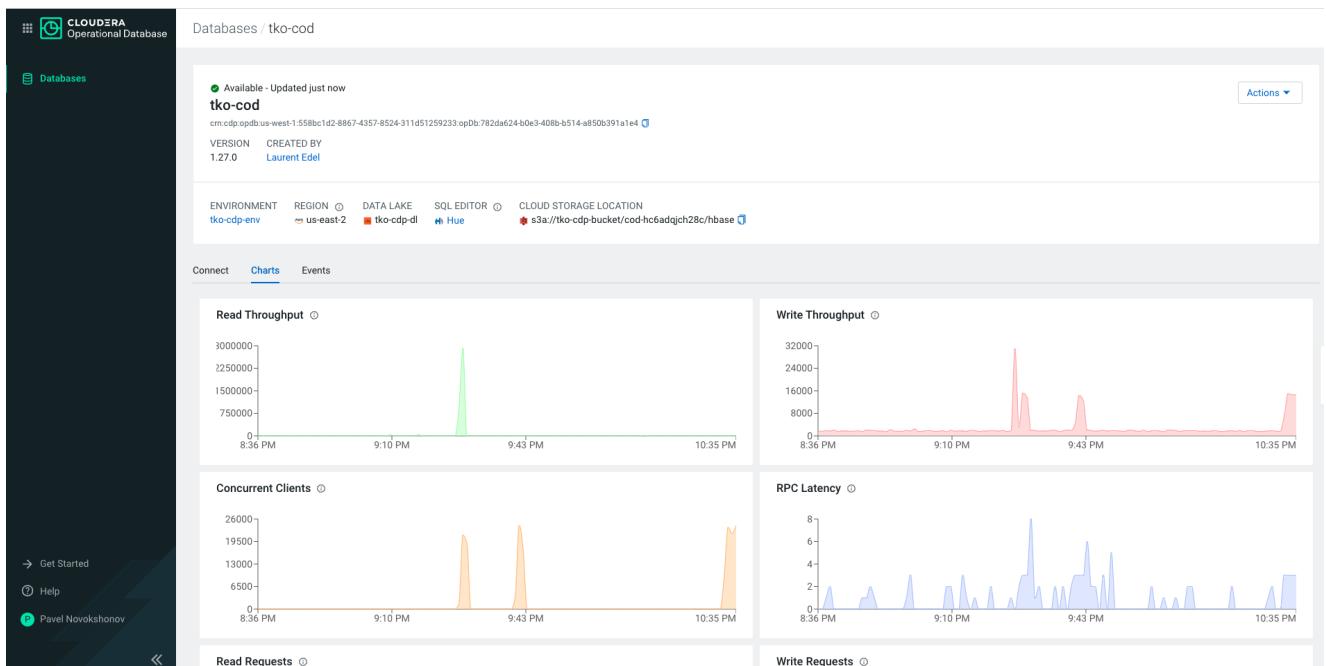
The output should look like this. In 120 seconds it ran 61,121 queries with max latency of 3128 msec and average latency of 19 msec. Feel free to run the test with more concurrent threads or change test duration. Err: 0 means that there were no errors during the run.

```

rw-r----- 1 provokshonov provokshonov 26494 Feb 9 02:34 jmeter.log
[pnovokshonov@cod-hc6adqjch28c-edge0 bin]$ ./jmeter.sh -n -t CDRLoadTest.jmx -Jthreads=10 -Jramp=5 -Jduration=120
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/pnovokshonov/apache-jmeter-5.5/lib/log4j-slf4j-impl-2.17.2.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/pnovokshonov/apache-jmeter-5.5/lib/ext/phoenix-client-hbase-2.4-5.1.1.7.2.16.0-287.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Creating summariser <summary>
Created the tree successfully using CDRLoadTest.jmx
Starting standalone test @ February 9, 2023 2:40:02 AM UTC (1675910402917)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
Generate Summary Results + 11170 in 00:00:27 = 420.5/s Avg: 21 Min: 5 Max: 3128 Err: 0 (0.00%) Active: 10 Started: 10 Finished: 0
summary + 11170 in 00:00:27 = 420.5/s Avg: 21 Min: 5 Max: 3128 Err: 0 (0.00%) Active: 10 Started: 10 Finished: 0
summary + 16145 in 00:00:30 = 538.2/s Avg: 18 Min: 5 Max: 101 Err: 0 (0.00%) Active: 10 Started: 10 Finished: 0
Generate Summary Results + 16145 in 00:00:30 = 538.2/s Avg: 18 Min: 5 Max: 101 Err: 0 (0.00%) Active: 10 Started: 10 Finished: 0
Generate Summary Results = 27315 in 00:00:57 = 482.9/s Avg: 19 Min: 5 Max: 3128 Err: 0 (0.00%)
summary = 27315 in 00:00:57 = 482.9/s Avg: 19 Min: 5 Max: 3128 Err: 0 (0.00%)
Generate Summary Results + 15958 in 00:00:30 = 532.0/s Avg: 18 Min: 5 Max: 201 Err: 0 (0.00%) Active: 10 Started: 10 Finished: 0
Generate Summary Results = 43273 in 00:01:27 = 499.9/s Avg: 19 Min: 5 Max: 3128 Err: 0 (0.00%)
summary + 15958 in 00:00:30 = 532.0/s Avg: 18 Min: 5 Max: 201 Err: 0 (0.00%) Active: 10 Started: 10 Finished: 0
summary = 43273 in 00:01:27 = 499.9/s Avg: 19 Min: 5 Max: 3128 Err: 0 (0.00%)
Generate Summary Results + 15998 in 00:00:30 = 533.1/s Avg: 18 Min: 5 Max: 120 Err: 0 (0.00%) Active: 10 Started: 10 Finished: 0
Generate Summary Results = 59271 in 00:01:57 = 508.5/s Avg: 19 Min: 5 Max: 3128 Err: 0 (0.00%)
summary + 15998 in 00:00:30 = 533.2/s Avg: 18 Min: 5 Max: 120 Err: 0 (0.00%) Active: 10 Started: 10 Finished: 0
summary = 59271 in 00:01:57 = 508.5/s Avg: 19 Min: 5 Max: 3128 Err: 0 (0.00%)
summary + 1850 in 00:00:04 = 528.1/s Avg: 18 Min: 7 Max: 113 Err: 0 (0.00%) Active: 0 Started: 10 Finished: 10
summary = 61121 in 00:02:00 = 509.0/s Avg: 19 Min: 5 Max: 3128 Err: 0 (0.00%)
Generate Summary Results + 1850 in 00:00:04 = 528.0/s Avg: 18 Min: 7 Max: 113 Err: 0 (0.00%) Active: 0 Started: 10 Finished: 10
Generate Summary Results = 61121 in 00:02:00 = 509.0/s Avg: 19 Min: 5 Max: 3128 Err: 0 (0.00%)
Tidying up ...
@ February 9, 2023 2:42:03 AM UTC (1675910523512)
... end of run
[pnovokshonov@cod-hc6adqjch28c-edge0 bin]$ 

```

You can monitor COD performance metrics by going to the Charts tab

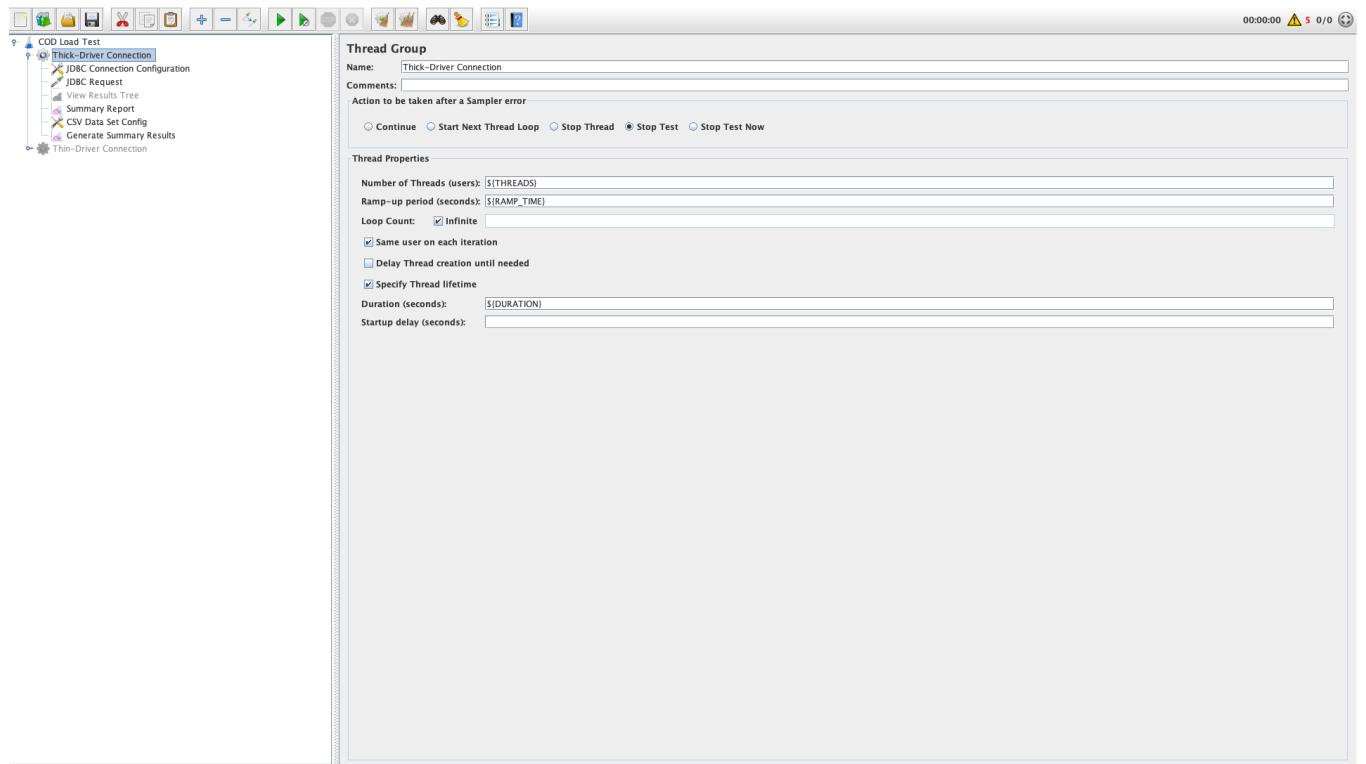


What is it we're running? You can open the **CDRLoadTest.jmx** and find the following select statement. This query is using part of the row key to pull records from Phoenix/HBase. Getting data by row key is a point lookup and a very fast operation.

Unset

```
select * from CDR.CDR where CALLING_MSISDN=${CALLING_MSISDN};
```

What if you want to look at or change the test contents, add more queries, modify database parameters, create a new test, etc? You have to run JMeter in a GUI mode and open a .jmx file which is **CDRLoadTest.jmx** in our case. One option would be to install JMeter locally on your laptop, copy and open a .jmx file. Below are some screenshots depicting our test scenario.



CDRLoadTest.jmx (C:\Users\pnovokshonov\CDRLoadTest.jmx) - Apache JMeter (5.5)

00:00:00 0/0

CD Load Test

- Thick-Driven Connection
 - JDBC Connection Configuration
 - JDBC Request
 - View Results Tree
 - Summary Report
 - CSV Data Set Config
 - Generate Summary Results
- Thin-Driven Connection

JDBC Request

Name: JDBC Request

Comments:

Variable Name Bound to Pool
Variable Name of Pool declared in JDBC Connection Configuration: pool

SQL Query

Query Type: Select Statement

```
1 select * from CDR.CDR where CALLING_MSISDN='${CALLING_MSISDN}'
```

Parameter values: \${CALLING_MSISDN}

Parameter types: VARCHAR

Variable names:

Result variable name:

Query timeout (s):

Limit ResultSet:

Handle ResultSet: Store as String

00:00:00 0/0

CD Load Test

- Thick-Driven Connection
 - JDBC Connection Configuration
 - JDBC Request
 - View Results Tree
 - Summary Report
 - CSV Data Set Config
 - Generate Summary Results
- Thin-Driven Connection

JDBC Connection Configuration

Name: JDBC Connection Configuration

Comments:

Variable Name Bound to Pool
Variable Name for created pool: pool

Connection Pool Configuration

Max Number of Connections: 10
Max Wait time: 10000
Time Between Eviction Checks: 60000
Auto Commit: True
Transaction Isolation: DEFAULT
Pool Prepared Statements: -1
Preinit Pool: False

Init SQL statements separated by new line:

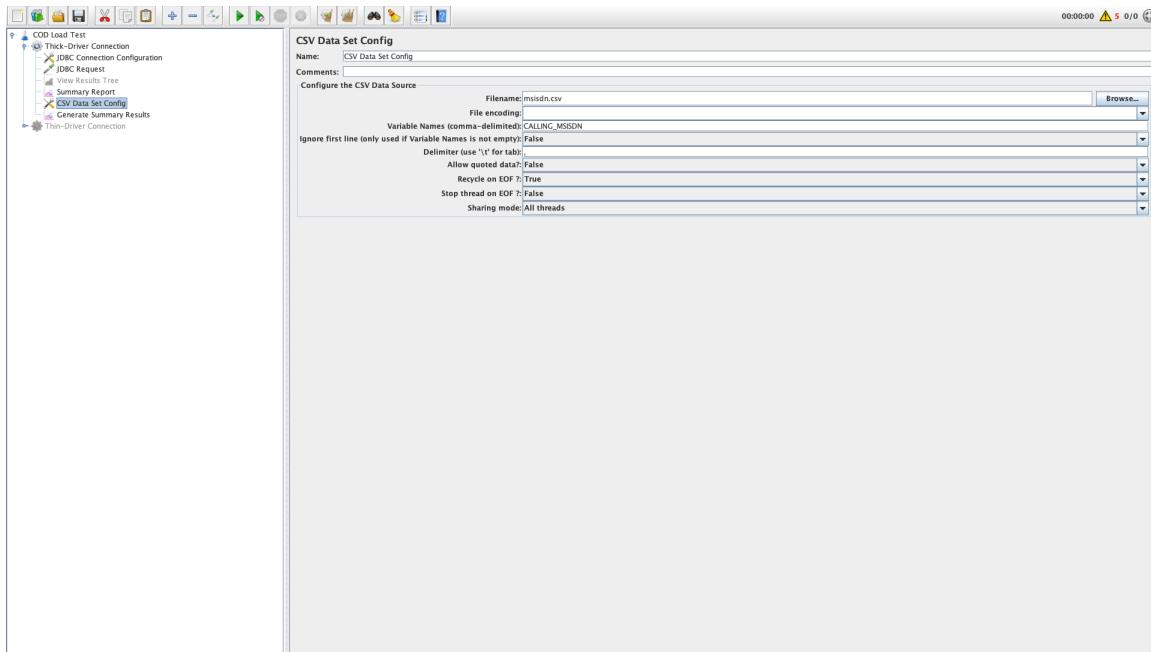
```
1
```

Connection Validation by Pool

Test While Idle: False
Soft Min Evictable Idle Time(m): 5000
Validation Query:

Database Connection Configuration

Database URL: jdbc:phoenix:cod-hc6adqjch28c-leader0.ilo-cdp.a465-9q4k.cloudera.site.cod-hc6adqjch28c-master0.ilo-cdp.a465-9q4k.cloudera.site.cod-hc6adqjch28c-master1.ilo-cdp.a465-9q4k.cloudera.site.2181:/base
JDBC Driver class: org.apache.phoenix.jdbc.PhoenixDriver
Username:
Password:
Connection Properties:



NOTE: JMeter can run in GUI mode on the Linux server (Edge node) as well, but you would have to install and configure **X11 - a remote-display protocol used by Linux/Unix machines.**

<https://www.businessnewsdaily.com/11035-how-to-use-x11-forwarding.html>

Appendix

Table Views

In Apache Phoenix, Table views can be created to read the data from table(s) as well as to write (update) tables provided conditions are met, (discussed below)

Lets create a Phoenix view using `CREATE VIEW....`

- Using phoenix-sqlline cli, execute the `CREATE VIEW` statement and replace `${USER}` with your username

```
Unset
```

```
CREATE VIEW simple_call_${USER}_vw AS SELECT * FROM test_${USER}
```

SELECT query powering the view can have complex WHERE clause e.g. joining two tables.

NOTE: Views in Phoenix are READ_ONLY

- Execute a `SELECT *` to read records using the view

```
Unset
```

```
SELECT * from simple_call_${USER}_vw
```

NOTE: When creating a Phoenix view over an existing Phoenix table, there is no need to explicitly define static columns that are already defined in the underlying table. The list of columns defined in the select query would be automatically included with the table view columns

You can also use a view to add new dynamic columns to an existing table. Phoenix allows you to write data (i.e execute DML) using a view as long as the view is defined on a single table and uses a simple equality expression in the `WHERE` clause.

Here we will add another table view on an existing table with an additional column named "FLAG2,. Any new columns added from the view would be added as dynamic columns to the underlying table.

- Run a `CREATE VIEW` statement and add the FLAG2 dynamic column. Replace `<USERNAME>` with your username

Unset

```
CREATE VIEW callView_{USER} (FLAG2 BOOLEAN) AS select * FROM test_{USER}
```

NOTE: The dynamic column “FLAG2” added in this view is different from the dynamic column FLAG that was added to the table in the earlier example.

Due to the COLUMN ENCODING feature added in Phoenix 4.10, data from any dynamic column added directly to the table would not be rendered from the view.

Any records (including existing records) that were present without the “specified” column will return a null value.

Similarly, when you add columns to a table with a view, you need to upsert data into the view.
You don't have to add the new column's type when using UPSERT into a view

Replication Manager

Another useful tool for migration HBase data is Replication Manager. RM is an easy method to move all your tables or a single table from one HBase instance to another. For example, if you need to move your on-prem Hbase cluster (either HDP or CDH) to COD on CDP Public Cloud on object storage, you can use replication to move the data and then port your application to the new COD cluster with little to no code changes. Although we won't create an active replication policy, here we will do a quick walk-through of the key features of Replication Manager.

- First, let's open up the CDP Control Plane. Select the Replication Manager tile



This will get you into the Replication Manager administration page. Since this is a new cluster, you won't see any replication policies. It's on this page you have the ability to create replication policies for HBase, Hive, and HDFS.

- Click on Create Policy to begin walking through the policy features.

The screenshot shows the "Overview" page of the Replication Manager. It features three main sections: "Classic Clusters" (with tabs for Error, Active, Warning, and Total), "Policies" (with tabs for Active, Suspended, Unhealthy, and Total), and "Jobs" (with tabs for In Progress, Failed Last, Failed in Last 10, and Total). A prominent red arrow points to the "Create Policy" button located in the bottom right corner of the "Jobs" section. Below the "Jobs" section, there is a section titled "Issues & Updates" which displays "No issues to report".

On the policy page you will see a simple step-by-step process for creating a policy. The first page is the general page where you will provide a policy name, (optional) description, and what service you want to replicate.

- On the General policy page, type a policy name and select HBase as the service you want to replicate. Click Next.

Create Replication Policy

The screenshot shows the 'Create Replication Policy' interface. On the left, a vertical navigation bar lists three steps: 1. General (selected), 2. Select Source, and 3. Select Destination. The main area is titled 'General' and contains fields for 'Policy Name' (set to 'test_policy') and 'Description' (with placeholder text 'Enter a description for the policy'). Below these are three radio buttons labeled 'Type': 'Hive' (unselected), 'HDFS' (unselected), and 'HBase' (selected, indicated by a blue outline). A large gray button at the bottom right is labeled 'Next Step'.

The process for replicating HBase is simple. You only need to select a source and a destination. Keep in mind that both source and destination need to exist.

- In this example, we will select our COD cluster as the source. Click on Source Cluster or Database and select the `opdb-hol-cod` cluster from the drop down. Type in your `test_<USERNAME>` as the source table

The screenshot shows the 'Select Source' page of the replication policy creation. The left sidebar shows the current step is 'Select Source'. The main area has a warning message: 'Some clusters may not be viewed by the current logged in user if role ClassicClusterUser is not assigned.' Below this is a dropdown menu for 'Source Cluster or Database' set to 'opdb-hol-cod (opdb-hol)'. Under 'Source Tables', there is a text input field containing 'test_User099' with a plus sign icon to its right. A checkbox 'Perform Initial Snapshot' is checked. On the right, a 'Summary' panel shows the 'Type' as 'HBase' and the 'Policy Name' as 'user099_repl_policy'. A large gray button at the bottom right is labeled 'Next Step'.

- In the Select Destination page, choose the database `opdb-cod-repl` as the destination database. You'll notice some warnings indicating that both the source and destination will go through rolling restarts and there will be a validation that the replication was successful.

- You will also need to Set the Workload user. For this enter the following

Username: cod_repl
Password: Clouderal234

Create Replication Policy

- On the Initial Snapshot screen, keep the defaults and click Create.

Create Replication Policy

After a brief pause, you should be taken back to the policy screen where you monitor the replication process.

Replication Policies

0	0	0	0	1
Error	Active	Suspended	Expired	Total
Search Replication Policy <input type="text"/> Type (All) 				
Status	Type	Name	Source	Destination
Configuring clusters	HBase	user099_repl_policy	opdb-hol-cod   opdb-hol	→ opdb-cod-repl   opdb-hol
Jobs Duration Last Success Next Run				
				  <1m n/a - 
Items per page: 10  1 - 1 of 1   				