Handle CDC (Change Data capture) using python and Spark. work on it, come up with Architecture –

CDC is a process to identify changes to records in a source system, interpret the change(s) accurately, and then replicate the change to a target system with an intent to either replicate the source system or to record the changes for historical analysis. Or in other words, it can be said that change Data capture is an innovative mechanism for data integration. It is a technology for efficiently reading the changes made to a source database and applying those to a target database. It records the modifications that happen for one or more tables in a database. CDC records write, delete, and update events. It copies a selection of tables in their entirety from a source database into the target database.

There are two types of data changes :

1. Query-based - his approach regularly checks the production database for changes. This method can also slow production performance by consuming source CPU cycles. So many organisation don't track changes directly alternatively they use different CDC methods.
2. Log-based - The CDC process is a more non-intrusive approach and does not involve the execution of SQL statements at the source. this method involves reading log files of the source database to identify the data that is being created, modified, or deleted from the source into the target Data Warehouse.

Steps to Perform CDC

STEP 1: Extract: Raw data is extracted from an array of sources and sometimes placed in a Data Lake. This data could be formatted in JSON – Social media (Facebook, etc.), XML – Third-party sources, RDBMS

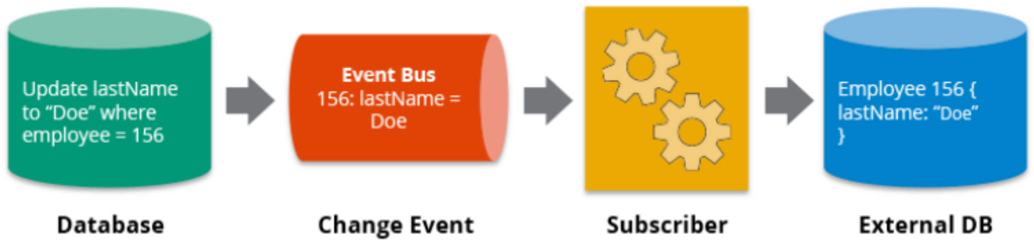
STEP 2: Transformation:

The transformation stage is where we apply any business rules and regulations to achieve.

Standardization, Deduplication, Verification, Sorting

STEP 3: Load:

To load this extracted transformed data into a new home by executing a task (job)



> The change data capture process via the publisher/subscriber method. Multiple databases and applications can subscribe to the change data.

How CDC concept came into existence ?

In the era of data as we know that data is increasing with the exponential speed. Now it is no longer enough to just store data and process it once or twice a day with a botch job. Now it require an alost immediate decision to annalize the newly come data and to do the modification accordingly in the database. In a fast-moving world, the amount of new information is so big that it has to be processed on the spot. Otherwise, the backlog will grow to an unmaintainable size. So to keeping the problem in mind this new feature is invented called change Data Capture which is also known as CDC.

Problem:

The CUSTOMER Hive Dimension table needs to capture changes from the source MySQL database. It's currently at 5 million records. 1 Million customer records in the source has changes and 4 new customer records were added to the source.

Step 1:

Run Sqoop with the incremental option to get new changes from the source MySQL database and import this into HDFS as a Parquet file

The source MySQL database has the column modified_date. For each run we capture the maximum modified_date so that on the next run we get all records greater (>) than this date. Sqoop will do this automatically with the incremental option. You just specify the column name and give a value.

```
sqoop import --connect jdbc:mysql://ip-172-31-2-69.us-west-
2.compute.internal:3306/mysql --username root --password password -
table customer -m4 --target-dir
/landing/staging_customer_update_spark --incremental lastmodified --
```

```
check-column modified_date --last-value "2016-05-23 00:00:00" --as-parquetfile
```

Step 2:

Merge the data from the Sqoop extract with the existing Hive CUSTOMER Dimension table. Read the Parquet file extract into a Spark DataFrame and lookup against the Hive table to create a new table. Go to end of article to view the PySpark code with enough comments to explain what the code is doing.

This is basic code to demonstrate how easily Spark integrates with Parquet to easily infer a schema and then perform SQL operations on the data.

Sample 5 records from the new table and compare with old table. Notice the changes for first_name, last_name, and modified_date:

New data:

a.cust_no	a.birth_date	a.first_name	a.last_name	a.gender	a.join_date	a.created_date	a.modified_date
99,868	2016-05-20	[redacted]SPARK	[redacted]SPARK	M	1943-02-22	2016-05-20 12:12:12.0	2016-05-24 21:36:48.0
166,448	2016-05-20	[redacted]SPARK	[redacted]SPARK	F	1976-05-20	2016-05-20 12:12:12.0	2016-05-24 21:36:48.0
1,630,027	2016-05-20	[redacted]SPARK	[redacted]SPARK	M	2002-05-24	2016-05-20 12:12:12.0	2016-05-24 21:36:48.0
2,358,546	2016-05-20	[redacted]SPARK	[redacted]SPARK	M	2029-07-21	2016-05-20 12:12:12.0	2016-05-24 21:36:48.0
3,322,263	2016-05-20	[redacted]SPARK	[redacted]SPARK	F	2035-06-20	2016-05-20 12:12:12.0	2016-05-24 21:36:48.0

What these records looked like before the changes:

customer.cust_no	customer.birth_date	customer.first_name	customer.last_name	customer.gender	customer.join_date	customer.created_date	customer.modified_date
99,868	1960-09-17	[redacted]	[redacted]	M	1994-07-15	2016-05-20 12:12:12.0	2016-05-20 12:12:12.0
166,448	1955-05-13	[redacted]	[redacted]	F	1997-11-10	2016-05-20 12:12:12.0	2016-05-20 12:12:12.0
1,630,027	1952-04-13	[redacted]	[redacted]	M	1986-11-07	2016-05-20 12:12:12.0	2016-05-20 12:12:12.0
2,358,546	1957-07-25	[redacted]	[redacted]	M	1989-07-26	2016-05-20 12:12:12.0	2016-05-20 12:12:12.0
3,322,263	1955-04-20	[redacted]	[redacted]	F	1990-02-27	2016-05-20 12:12:12.0	2016-05-20 12:12:12.0

PySpark code:

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

# -*- coding: utf-8 -*-


import sys

import os

from pyspark.sql import *

from pyspark import SparkConf, SparkContext, SQLContext

from pyspark.sql import HiveContext

from pyspark.sql.types import *
```

```
## write to snappy compressed output

conf = (SparkConf()

         .setAppName("spark_cdc")

         .set("spark.dynamicAllocation.enabled", "true")

         .set("spark.shuffle.service.enabled", "true")

         .set("spark.rdd.compress", "true"))

sc = SparkContext(conf = conf)


sqlContext = HiveContext(sc)


## read parquet file generated by the Sqoop incremental extract from
MySQL and imported into HDFS

path = "hdfs://ip-172-31-2-63.us-west-
2.compute.internal:8020/landing/staging_customer_update_spark/*parquet*
"

parquetFile = sqlContext.read.parquet(path)

parquetFile.registerTempTable("customer_extract");




sql = "DROP TABLE IF EXISTS customer_update_spark"

sqlContext.sql(sql)


sql = """

CREATE TABLE customer_update_spark

(

 cust_no       int

 ,birth_date    date

 ,first_name    string

 ,last_name     string

 ,gender        string

 ,join_date     date
```

```
 ,created_date  timestamp
 ,modified_date timestamp
)
STORED AS PARQUET

"""
sqlContext.sql(sql)



## get those records that did not change
## these are those records from the existing Dimension table that are
not in the Sqoop extract

sql = """
INSERT INTO TABLE customer_update_spark
SELECT
a.cust_no,
a.birth_date,
a.first_name,
a.last_name,
a.gender,
a.join_date,
a.created_date,
a.modified_date
FROM customer a LEFT OUTER JOIN customer_extract b ON a.cust_no =
b.cust_no
WHERE b.cust_no IS NULL
"""

sqlContext.sql(sql)
```

```
## get the changed records from the Parquet extract generated from
Sqoop

## the dates in the Parquet file will be in epoch time with
milliseconds

## this will be a 13 digit number

## we don't need milliseconds so only get first 10 digits and not all
13


## for birth_date and join date convert to date in format YYYY-MM-DD

## for created_date and modified date convert to format YYYY-MM-DD
HH:MI:SS

sql = """

INSERT INTO customer_update_spark

SELECT

a.cust_no,

TO_DATE(FROM_UNIXTIME(CAST(SUBSTR(a.created_date, 1,10) AS INT))) AS
birth_date,

a.first_name,

a.last_name,

a.gender,

TO_DATE(FROM_UNIXTIME(CAST(SUBSTR(a.join_date, 1,10) AS INT))) AS
join_date,

FROM_UNIXTIME(CAST(SUBSTR(a.created_date, 1,10) AS INT)) AS
created_date,

FROM_UNIXTIME(CAST(SUBSTR(a.modified_date, 1,10) AS INT)) AS
modified_date

FROM customer_extract a

"""

sqlContext.sql(sql)
```

```
-- the customers dimension table in Hive before capturing new updates from the source MySQL database
-- the hive table is in Parquet format

SELECT * FROM customer LIMIT 5
```

FINISHED

customer.cust_no	customer.birth_date	customer.first_name	customer.last_name	customer.gender	customer.join_date	customer.created_date	customer.modified_date
1	1953-09-02	[redacted]	[redacted]	M	1986-06-26	2016-05-20 12:12:12.0	2016-05-20 12:12:12.0
2	1964-06-02	[redacted]	[redacted]	F	1985-11-21	2016-05-20 12:12:12.0	2016-05-20 12:12:12.0
3	1959-12-03	[redacted]	[redacted]	M	1986-08-28	2016-05-20 12:12:12.0	2016-05-20 12:12:12.0
4	1954-05-01	[redacted]	[redacted]	M	1986-12-01	2016-05-20 12:12:12.0	2016-05-20 12:12:12.0
5	1955-01-21	[redacted]	[redacted]	M	1989-09-12	2016-05-20 12:12:12.0	2016-05-20 12:12:12.0

customer.cust_no	customer.birth_date	customer.first_name	customer.last_name	customer.gender	customer.join_date	customer.created_date	customer.modified_date
99,868	1960-09-17	[redacted]	[redacted]	M	1994-07-15	2016-05-20 12:12:12.0	2016-05-20 12:12:12.0
166,448	1955-05-13	[redacted]	[redacted]	F	1997-11-10	2016-05-20 12:12:12.0	2016-05-20 12:12:12.0
1,630,027	1952-04-13	[redacted]	[redacted]	M	1986-11-07	2016-05-20 12:12:12.0	2016-05-20 12:12:12.0
2,358,546	1957-07-25	[redacted]	[redacted]	M	1989-07-26	2016-05-20 12:12:12.0	2016-05-20 12:12:12.0
3,322,263	1955-04-20	[redacted]	[redacted]	F	1990-02-27	2016-05-20 12:12:12.0	2016-05-20 12:12:12.0

```
-- count of records in the original customers dimension table

SELECT COUNT(1) AS total_record_count FROM customer
```

total_record_count
5,000,000