

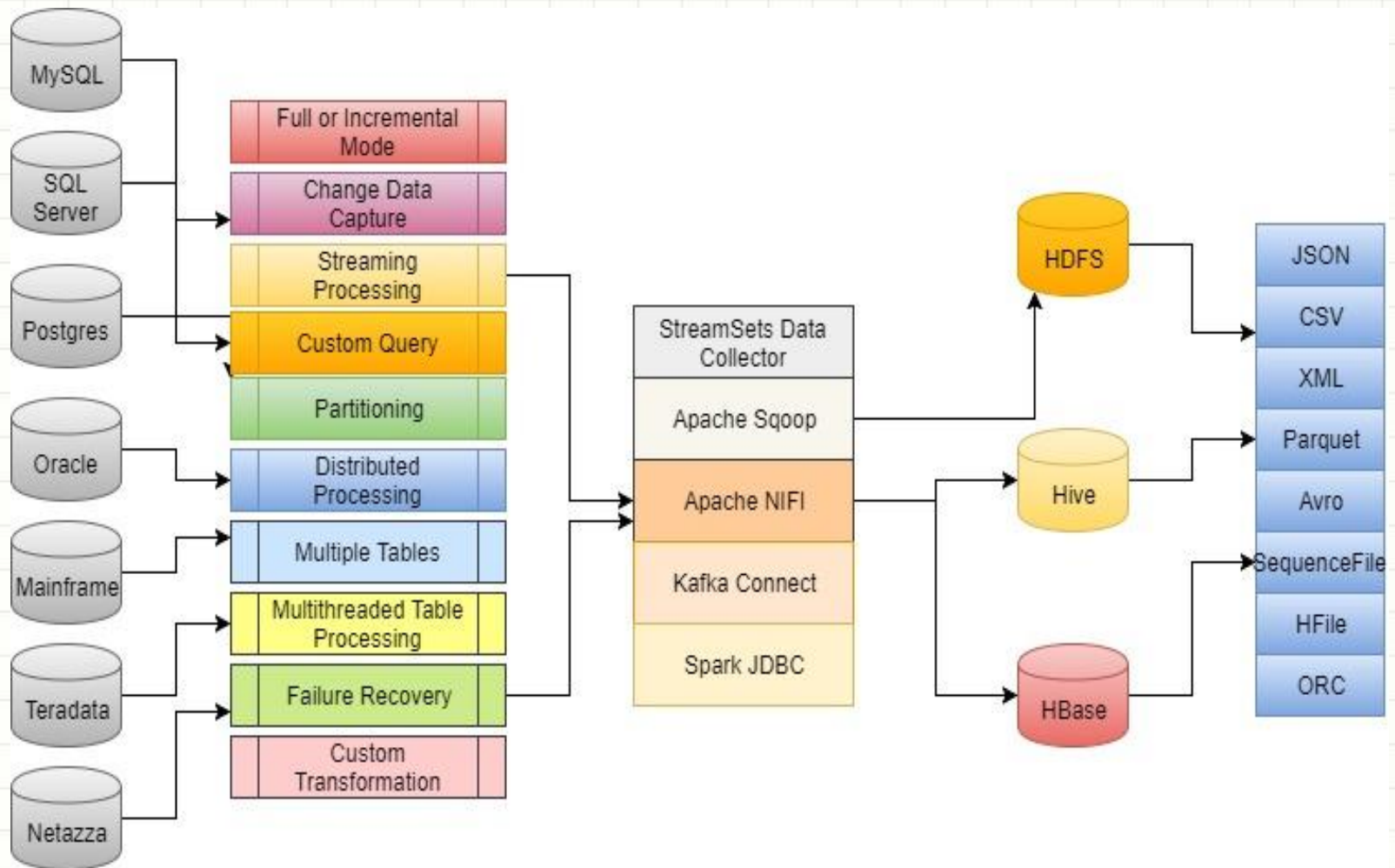


DATA TRANSFER FROM RDBMS TO DATA LAKE

Bin Jiang

12/17/2017

Data Movement Diagram



Data Transfer Frameworks

- Apache Sqoop
- Apache Nifi
- StreamSets
- Kafka Connect
- Spark JDBC
- Extracted as CSV and Imported into Hadoop

Types of Data Sources

- MySQL
- Postgres
- Oracle
- SQL Server
- Netezza
- MapR DB
- Teradata
- MainFrame
- Hive
- HBase
- HDFS

Types of Data Format in Data Lake

- CSV
- Text
- Parquet
- Avro
- Sequence File
- HFile
- JSON
- XML
- ORC

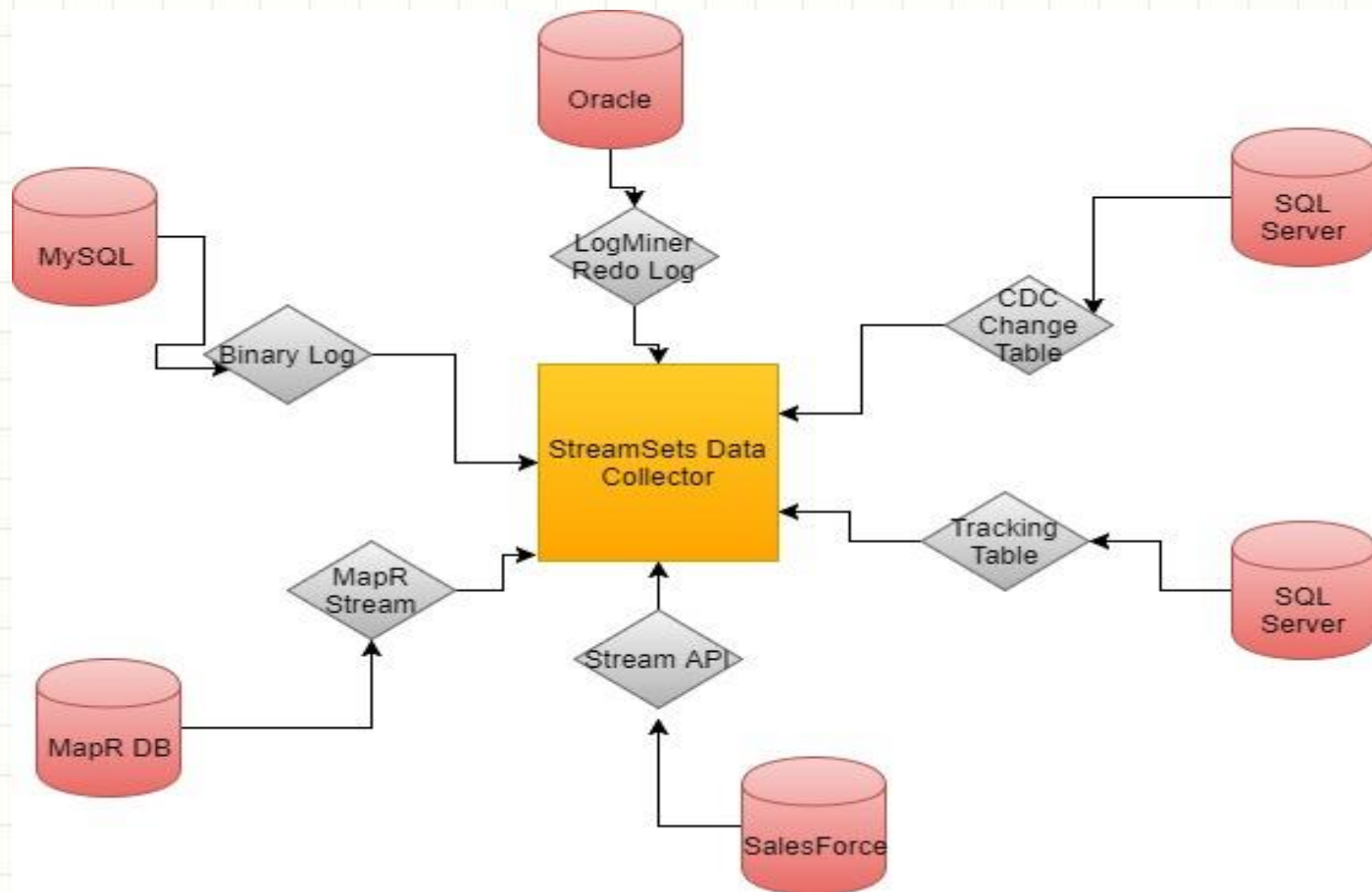
Supported Features of Connectors

- Multiple Databases Ingestion
- Multiple Tables Ingestion
- Custom Query
- Change Data Capture (Insert, Delete, Update, Upsert, Undelete, Replace)
- Multithreaded Table Processing
- Multithreaded Partition Processing
- Batch Strategies
- JDBC Header Attributes
- Event Generation
- Full and Incremental Mode
- Recovery
- Distributed Processing
- Dynamic Table
- Streaming Processing
- Back Pressure
- Replication of database

CDC-Enabled Data Sources

- MapR DB (**MapR Streams**)
- MySQL (**Binary Log**)
- Oracle CDC Client (**LogMiner Redo Log**)
- Salesforce (**Stream API**)
- SQL Server (**CDC Table**)
- SQL Server (**Change Tracking Table**)

CDC-Enabled Data Sources



Supported Features of Transformation

- Record Deduplication
- Data Masking
- Data Hasher
- Content Merge
- Field Order
- Field Flattener
- Data Pivot
- Field Splitter



APACHE SQOOP

Bin Jiang

12/17/2017

Objective

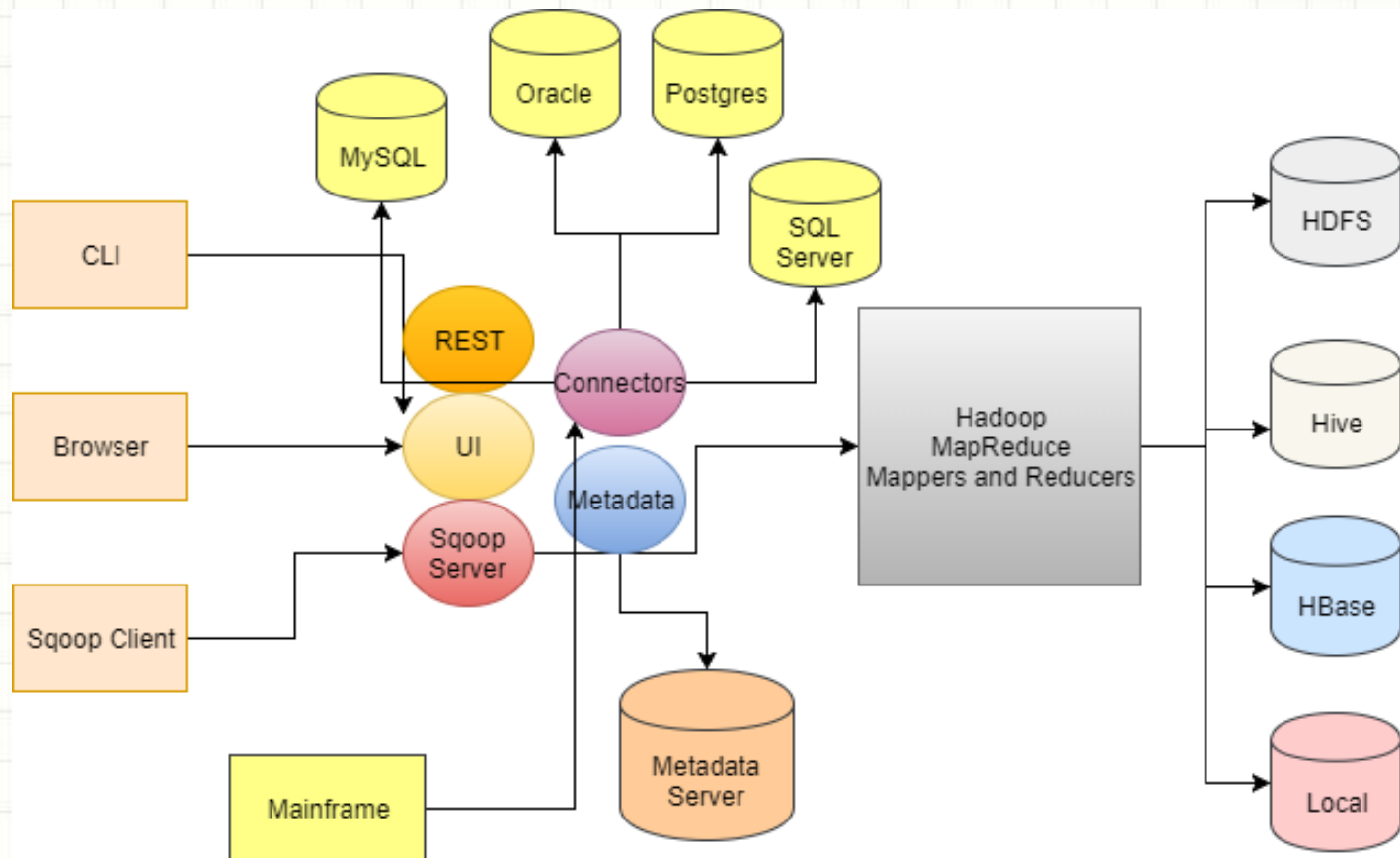
Participants will learn about

- Apache Sqoop, a tool for data transfer between relational database and hdfs, hive, hbase
- How to control the performance of import and export processes
- Sqoop's supported file formats

What's Sqoop

- Sqoop is a command-line tool for efficient two-way data transfer between relational database and the Hadoop Distributed File System (HDFS)
- Hadoop-based systems, such as Hive and HBase can leverage Sqoop's data load facilities
- Sqoop is written in Java
- Sqoop comes bundled with a number of connectors to popular databases and data warehousing systems
- If needed, developer can use Sqoop's API for development of new system connectors

Architecture of Sqoop



Features

- Data transfer from mainframe to data lake
- Command line tool and easy to use
- Distributed processing such as map reduce
- Partitioning
- Full and incremental mode
- Support both tables and custom query import
- Stored procedure Export
- Import data validation
- Import data as avro, parquet, sequence file and text file
- Import data into HDFS, Hive and HBase
- Boundary query to use for creating data split
- Support direct database connector
- Only support one split column

Features

- Support data compression
- Null string support
- No reducers
- Number of mappers can be controlled
- Transaction isolation
- Type mapping
- Support large object such as blob and clob
- Support unambiguous parsing (--enclosed-by)
- Saved import and export jobs
- Integration with Hcatalog
- Direct supported databases: MySQL, Oracle, Postgres, SQL Server, Netezza

Sqoop Import/Export

- In Sqoop's terminology, Import refers to extracting data from a relational database and loading it into HDFS. export is the reverse operation, where data is extracted from HDFS and inserted into a relational database
- From the relational database's point of view, the above operations are the opposite: Sqoop's import operation is seen as the database export operation, The export operation (extraction of from HDFS into the target relational database) is seen as the import operation
- Sqoop uses MapReduce to import and export the data that offers parallel task execution and fault tolerance (failed tasks are transparently restarted)

Sqoop Help

To list all available commands in Sqoop, run the `sqoop help` command which produces the following (shortened version) output:

```
$sqoop help
```

- `eval` Evaluate a SQL statement and display the results
- `export` Export an HDFS directory to a database table
- `Import` Import a table from a database to HDFS
- `import-all-tables` Import tables from a database to HDFS`
- `list-databases` List available databases on a server
- `list-tables` List available tables in a database
- `version` Display version information

Preparing MySQL

- Using your SSH client, connect to your virtual box. login as root
- Check if business.csv exist under the folder /root/TrainingOnHDP/dataset on the HDP sandbox

Preparing MySQL

- Execute the following command:
 1. `mysql -u root -p`
 2. `USE test;`
 3. `create table business (location_id varchar(100),business_account_number INT,qwnership_name varchar(100),dba_name varchar(100),street_address varchar(100),city varchar(100),state varchar(100),source_zipcode varchar(100),business_start_date varchar(100),business_end_date varchar(100), location_start_date varchar(100),location_end_date varchar(100),mail_address varchar(100),mail_city varchar(100),mail_zipcode varchar(100), mail_state varchar(100),naics_code varchar(100),naics_code_description varchar(100),parking_tax varchar(100),transient_occupancy_tax varchar(100), lic_code varchar(100),lic_code_description varchar(100),supervisor_district varchar(100),neighborhoods_analysis_boundaries varchar(100),business_corridor varchar(100),business_location varchar(100));`

Preparing MySQL

- Execute the following command:
- load data local infile '/root/TrainingOnHDP/dataset/business.csv' into table business fields terminated by ',' enclosed by '"' lines terminated by '\n' ignore 1 lines
(location_id,business_account_number,ownership_name,dba_name,street_address,city,state,source_zipcode,business_start_date,business_end_date,location_start_date,location_end_date,mail_address,mail_city,mail_zipcode,mail_state,naics_code,naics_code_description,parking_tax,transient_occupancy_tax,lic_code,lic_code_description,supervisor_district,neighborhoods_analysis_boundaries,business_corridor,business_location);
- ALTER TABLE business ADD id INT AUTO_INCREMENT PRIMARY KEY;

Examples of Using Sqoop Commands

- In examples below, the jdbc:mysql://localhost/[DB_name] string is a sample JDBC (Java Database Connectivity) connect string to MySQL relational database
- sqoop list-databases lists the available database schemas, e.g.

```
$sqoop list-databases --connect jdbc:mysql://localhost --username  
root -P
```

- sqoop list-tables lists the tables in the specified database schema, e.g.

```
$sqoop list-tables --connect jdbc:mysql://localhost/hive --  
username root -P
```

Examples of Using Sqoop Commands

- sqoop import imports a table from a database to HDFS, e.g.
`$sqoop import -connect jdbc:mysql://localhost/test --table business --fetch-size 10 --username root -P`
- run the following command to check the files
`hadoop fs -ls /user/root/business`
- run the following command to remove the files
`hadoop fs -rm -R -skipTrash /user/root/business`
- sqoop eval is Sqoop's primitive SQL execution shell which you can use as follows:
`$sqoop eval --connect jdbc:mysql://localhost/test -e 'SELECT count(*) FROM business;' --username root -P`

Data Import Example

- The following import command will create the target file in the default text format (the `--as-textfile` argument is omitted) with comma-separated fields-terminated-by

```
$sqoop import --connect jdbc:mysql://localhost/test --table  
business --target-dir /user/root/mybusiness --fields-terminated-by  
';' --fetch-size 10 --username root -P
```

- run the following command to check the files

```
hadoop fs -ls /user/root/mybusiness
```

- run the following command to remove the files

```
hadoop fs -rm -R -skipTrash /user/root/mybusiness
```

Fine-tuning Data Import

- By default, Sqoop uses the `SELECT * FROM myTable` command when importing data with the `sqoop import --table myTable` command
- You can limit the number of rows to be imported by providing the `WHERE` clause with the `--where` import control argument:

```
$sqoop import --connect jdbc:mysql://localhost/test --table  
business --where "location_id='0000071-02-001'" --fields-  
terminated-by ';' --fetch-size 10 --username root -P
```

Fine-tuning Data Import

- Similarly, you can control the choice of columns that you may want to include in the data import; for that use the `--columns` import control argument: `--columns <col1 [,col2, col...]>`

```
$sqoop import --connect jdbc:mysql://localhost/test --table  
business --columns "id,street_address,city" --where  
"location_id='0000071-02-001'" --fields-terminated-by ';' --fetch-  
size 10 --username root -P
```

Controlling the Number of Import Processes

- Sqoop imports data in parallel from most relational databases
- By default, it uses four concurrent import processes (Hadoop map tasks)
- You can explicitly specify the number of the import parallel processes by specifying the value to the -m argument of the import command, e.g. -m 8
- Do not set this number too high (definitely, it should not be greater than the number of map tasks available within your hadoop cluster), a value between 8 and 16 may be optimal

Data Splitting

- When running parallel import processes, Sqoop tries to horizontally split source table data into chunks of the same size in order to evenly distribute workload across import processes
- By default, Sqoop tries to use the source table's primary key column's primary key column's values to identify split points
- If the values in the primary key column are not uniformly distributed, Sqoop may create uneven workload across the processes; to avoid this, you should explicitly choose a more suitable "splitting" column and using it as a value to the --split-by argument

Data Splitting

Notes:

Sqoop uses a simple data splitting algorithm to create the workload for import processes. By default, Sqoop will try to use the primary key column in the source table as the splitting column. Sqoop performs data sampling to find the minimum and maximum values in the splitting column. Then, it divides the value range by the number of map tasks (the default value is four) and arrives at the number of rows in each split. Each process will receive its own SELECT statement with a customized WHERE clause (which is represented by the \$CONDITIONS clause - more on this later). For example, if the splitting column in the T table contains auto-incremented values from 1 to 1,000,000 and the number of map tasks is four, then each of the four import processes will get 250,000 rows to fetch and import into HDFS. The first process will have the following import SELECT statement: SELECT * FROM T WHERE id >=1 AND id <= 250000; The next process will have it as: SELECT * FROM T WHERE id >= 250001 AND id <= 500000; And so on.

```
$sqoop import -connect jdbc:mysql://localhost/test --table business --fetch-size 10 --username root -P --split-by business_account_number
```

Helping Sqoop Out

- You can help Sqoop perform better by letting it leverage Hadoop's built-in processing parallelism (on top of the -m configuration parameter)
- You can do so by specifying the following attributes of the import statement:
 1. The \$CONDITIONS token (which is just a Sqoop's placeholder in the WHERE clause; Sqoop will replace it with a unique condition expression per process to help partition the data to fetch)
 2. --target-dir followed by HDFS destination directory (normally, it has the same name as the table which is being imported)
 3. --split-by followed by the column name that will be used to split the data for parallel load

Example of Executing Sqoop Load in Parallel

- The data import (load) optimized command syntax looks as follows:

```
$sqoop import -connect jdbc:mysql://localhost/test --query 'SELECT * FROM  
business WHERE $CONDITIONS' --fetch-size 10 --username root -P --split-by  
business_account_number --target-dir /user/root/business -m 8
```

- The above free-form query is enclosed in single quotes, if you want to use double quotes instead (""), then you will need to escape the Sqoop's optimization tokens: \ \$CONDITIONS, e.g.

```
--query "SELECT * FROM business WHERE \ $CONDITIONS"
```

A Word of Caution: Avoid Complex Free-Form Queries

- You can combine your WHERE clause with the Sqoop's \$CONDITIONS placeholder, e.g.:

```
'SELECT * FROM business WHERE business_account_number = 71 AND $CONDITIONS'
```

- Avoid using complex queries such as queries that have sub-queries and using the OR condition as it may lead to unexpected results; the AND condition is fine

Using Direct Export from Databases

- By default, Sqoop uses cross-platform JDBC statements to import data
- Many databases also support native import/export tools which perform better than JDBC, e.g. MySQL provides the **mysqldump** tool for exporting (dumping) data from MySQL
- You can override the default JDBC-based import by supplying the `--direct` import command argument (Sqoop will use the appropriate) database data export tool transparently)

Example of Using Direct Export from MySQL

- The following command will use MySQL's default export utility `mysqldump` (you don't need to specify its name as Sqoop is intelligent enough to know how to use it)

```
$sqoop import -connect jdbc:mysql://localhost/test --table business --fetch-size 10 --username root -P --direct
```

- The source database's native export/import utilities must be visible (present on the path) to the shell from which you execute the sqoop utility.

More on Direct Mode Import

- By default, data from the source table will go to a new HDFS directory, if the destination directory already exists, Sqoop will abort the import operation
- Use the `--append` command arguments to append the data

```
$sqoop import -connect jdbc:mysql://localhost/test --table business --fetch-size 10 --username root -P --direct --append
```

- In direct mode, you can specify arguments that are passed to the export utility directly; use the `--argument` for that:

```
$sqoop import -connect jdbc:mysql://localhost/test --table business --fetch-size 10 --username root -P --direct -- --default-character-set=utf8
```

Changing Data Types

- By default, Sqoop transparently maps most SQL types to appropriate Java or Hive counterparts
- You have an ability to override the default mapping using the following parameters:

--map-column-java <mappings>

--map-column-hive <mappings>

- The <mappings> is a comma separated list of mappings in the form of `name_of_column=target_type [,name_of_column2 = target_type2]`

```
$sqoop import -connect jdbc:mysql://localhost/test --table business --fetch-size 10 --username root -P --direct --map-column-java location_start_date=String
```

Examples of Default Types Overriding

- Changing the type of col1 (e.g. it may be varchar) to Java's Integer type:

```
$sqoop import ... --map-column-java col1=Integer
```

- Sqoop will throw an exception in case of failed mapping operation

File Formats

- Sqoop supports three imported file formats of the file created in HDFS after the import operation:
 1. CSV (Character-Separated Values) text file format (default format, using comma as field separator)
 2. SequenceFile binary format
 3. Avro binary format
 4. Parquet binary format
- You specify the import file format by passing the appropriate arguments:
 1. --as-textfile argument (default)
 2. --as-sequencefile
 3. --as-avrodatafile
 4. --as-parquetfile

File Formats

```
$sqoop import -connect jdbc:mysql://localhost/test --table business --  
fetch-size 10 --username root -P --as-sequencefile
```

```
$sqoop import -connect jdbc:mysql://localhost/test --table business --  
fetch-size 10 --username root -P --as-avrodatafile
```

```
$sqoop import -connect jdbc:mysql://localhost/test --table business --  
fetch-size 10 --username root -P --as-parquetfile
```


The Apache Avro Serialization System

- Apache Avro is a self-described compact data serialization system
- The avro system can store a variety of data structures which are described in the JSON-encoded data structure schema (meta information record) The schema is stored within the binary file reducing the number of external artifact dependencies This self-sufficiency makes it different from the SequenceFile binary format which requires a serialization class
- To work with the Avro binary file format, your custom application will need to have access to Avro libraries

Binary vs Text

- The CSV file format is used in cases where further data manipulation with other tools, such as Hive, is expected
- Processing files in the SequenceFile or Avro binary formats is more efficient than processing text files but requires custom MapReduce programs

The binary data is stored in the source databases in columns of the binary type, e.g. VARBINARY

- The SequenceFile is a special Hadoop file format that implements the `org.apache.hadoop.io.Writable` Java interface

More on the SequenceFile Binary Formats

- When you perform an import operation, in addition to writing the contents of the table to HDFS, Sqoop also generate a Java source file named after the source table
- This file is used, for example, to deserialized records from the SequenceFile storage or to interact with the table records. This is the dependency, Avro does not have

Importing Data In Compressed Formats

Sometimes we would need to keep imported data on HDFS in compressed state in order to reduce the overall disk utilization. As sqoop is based on Map Reduce execution, it inherits Hadoop's all compression features. It allows us to save imported data into various compressed formats e.g. GZIP or BZ2 etc. We can execute following query to use by default gzip compression,

```
$sqoop import -connect jdbc:mysql://localhost/test --table business --fetch-size 10 --username root -P --compress
```

This will store all imported files with .gz extension. To use other compression codecs, we have mentioned the same at the time of execution.

```
$sqoop import -connect jdbc:mysql://localhost/test --table business --fetch-size 10 --username root -P --compress --compression-codec org.apache.hadoop.io.compress.BZip2Codec
```

This will store all files with .bz2.

Incremental imports

- You can also perform incremental imports. Sqoop supports two types: **"append,"** which works for numerical data that is incrementing over time, such as auto-increment keys, and **"lastmodified,"** which works on time-stamped data. In both cases, you need to specify the column using **-check-column**, the mode via the **--incremental** argument (the value must be either "append" or "lastmodified"), and, finally, the actual value to use to determine the incremental changes, **--last-value**.

Example of Incremental imports

- **Create options file named import_options.txt, the content is as follows:**

```
import
--connect
jdbc:mysql://localhost/test
--username
root
-P
```

- **Upload import_options.txt to VM under /root**

- **Run the follow commands:**

```
$sqoop --options-file ~/import_options.txt --check-column "id" --incremental "append" --
last-value "198565" --table business --fetch-size 10
```

```
$sqoop import --check-column "id" --incremental "append" --last-value "198565" --
connect jdbc:mysql://localhost/test --table business --fetch-size 10 --username root -P
```

- **Note:**

If use "lastmodified", check-column type has to be either date or timestamp

Sqoop jobs and the metastore

- You can see in the command output the last value that was encountered for the increment column. How can we best automate a process that can reuse that value? Sqoop has the notion of a job, which can save this information and reuse it in subsequent executions:

```
$sqoop job --create business_increment -- import --append --check-column "id"  
--incremental "append" --last-value "198560" --fetch-size 10 --connect  
jdbc:mysql://localhost/test --username root -P --table business
```

- This merely saves the notion of this command as a job in something called the Sqoop metastore. A Sqoop metastore keeps track of all jobs. By default, the metastore is contained in your home directory under .sqoop and is only used for your own jobs. If you want to share jobs, you would need to install a JDBC-compliant database and use the --meta-connect argument to specify its location when issuing job commands.

Sqoop jobs and the metastore

- The job create command executed in the previous example didn't do anything other than add the job to the metastore. To run the job, you need to explicitly execute it as shown in this listing:

1. `$sqoop job --list`
2. `$sqoop job --exec business_increment`
3. `$sqoop job --show business_increment`

#1 — List all jobs in the metastore

#2 — Executes your job

#3 — Shows metadata information about your job

Sqoop jobs and the metastore

Note:

- The metadata includes the last value of your incremental column ("lastmodified"). This is actually the time that the command was executed, and not the last value seen in the table. If you are using this feature, make sure that the database server and any clients interacting with the server (including the Sqoop client) have their clocks synced with the Network Time Protocol (NTP).
- For security reasons, by default your database password will not be stored in the Sqoop metastore. When executing a saved job, you will need to reenter the database password. Uncomment this setting to enable saved password storage. (INSECURE!). Another method is by using the parameter `--password-file` to pass in the file containing the password.

Controlling Distributed Cache

Sqoop will copy the jars in \$SQOOP_HOME/lib folder to job cache every time when start a Sqoop job. When launched by Oozie this is unnecessary since Oozie use its own Sqoop share lib which keeps Sqoop dependencies in the distributed cache. Oozie will do the localization on each worker node for the Sqoop dependencies only once during the first Sqoop job and reuse the jars on worker node for subsequencial jobs. Using option **--skip-dist-cache** in Sqoop command when launched by Oozie will skip the step which Sqoop copies its dependencies to job cache and save massive I/O.

Protecting Your Password

You have four options besides specifying the password on the command line with the `--password` parameter:

- Specifying the password on the command line with the `--P` parameter, that will instruct Sqoop to read the password from standard input
- You can save your password in a file and specify the path to this file with the parameter **`--password-file`**
- **Encrypt the password in the password, then use hadoop Credentials Loader to decrypt the password**
- Use **password alias**

Protecting Your Password

- You can save your password in a file and specify the path to this file with the parameter `--password-file`

Here's an example of reading the password from a file:

```
$sqoop import --connect jdbc:mysql://localhost/test --username root --table  
business --fetch-size 10 --password-file sqoop-password.txt
```


Protecting Your Password

- `mysql -u root -p`
- `DROP USER 'sqoop_user'@'localhost';`
- `CREATE USER 'sqoop_user'@'localhost' IDENTIFIED BY 'password';`
- `GRANT ALL PRIVILEGES ON *.* TO 'sqoop_user'@'localhost' WITH GRANT OPTION;`
- `GRANT FILE ON *.* TO 'sqoop_user'@'localhost' WITH GRANT OPTION;`
- `FLUSH PRIVILEGES;`
- `quit;`
- `echo -n "password" > sqoop.password`
- `hadoop fs -put sqoop.password /user/root/sqoop.password`
- `hadoop fs -chmod 400 /user/root/sqoop.password`
- `rm sqoop.password`
- `sqoop import --connect jdbc:mysql://localhost/test --username sqoop_user --table business --fetch-size 10 --password-file sqoop.password (HDFS)`
- `sqoop import --connect jdbc:mysql://localhost/test --username sqoop_user --table business --fetch-size 10 --password-file file:///root/sqoop.password (Local File System)`

Protecting Your Password

- Encrypt the password in the password, then use hadoop Credentials Loader to decrypt the password

```
$sqoop import -  
Dorg.apache.sqoop.credentials.loader.class=org.apache.sqoop.util.password.CryptoFileLoader -  
Dorg.apache.sqoop.credentials.loader.crypto.passphrase=sqoop2 --connect jdbc:mysql://localhost/test --  
username sqoop_user --password-file file:///root/sqoop.password --table business --fetch-size 10
```

- Use password alias

```
$hadoop credential create mypassword_alias -provider jceks://hdfs/user/root/test.jceks  
$sqoop import -Dhadoop.security.credential.provider.path=jceks://hdfs/user/root/test.jceks --connect  
jdbc:mysql://localhost/test --username sqoop_user --table business --fetch-size 10 --password-alias  
mypassword_alias
```

Generating the Java Table Record Source Code

- The codegen tools generate Java classes which help work with imported table records
- This tool allows for generation of the files without the need to import data
- Use the following command:

```
$sqoop codegen --connect jdbc:mysql://localhost/test --username  
sqoop_user --password password --table business
```

- You can optionally specify the class name of your preference by using the `--class-name` argument option

Data Export from HDFS

- Sqoop supports data export from HDFS into relational databases
- It uses the export tool to export a set of files
- The target table must already exist in the database
- The definition of the table must match the values in the source files

Export Tool Common Arguments

- The export command has the following syntax:

`$sqoop export arguments`

- The most common export arguments are:
 1. `--connect <jdbc-uri>` Specify JDBC connect string
 2. `--driver <class-name>` Manually specify JDBC driver class to use
 3. `--help` Print usage instructions
 4. `--passwd-file` Path to file containing the password`
 5. `--P` Read password from console
 6. `--password <password>` Set authentication password
 7. `--username <username>` Set authentication username

Data Export Control Arguments

- Many data export control arguments match those used in the import operation
- The most common export control arguments are:

<code>--direct</code>	Use direct export fast path
<code>--export-dir <dir></code>	HDFS source path for the export
<code>-m</code>	Set the number of parallel export map tasks
<code>--table <table-name></code>	Table to populate
<code>--call <stored-proc-name></code>	Stored Procedure to call
<code>--batch</code>	Use batch mode for underlying statement execution

Data Export Control Arguments

- Relational database provide native high-performance data import tools that Sqoop can use to offer better performance for its export operation than the default JDBC-based export mode. For example, MySQL provides the mysqlimport tool for direct data export mode. When using direct export mode (via the --direct argument), the target database data import utility must be on the path visible to the shell.

Data Export Example

- The following sqoop export command will export data from the /user/me/export/HDFS folder and insert it into a table named foo created in a MySQL database named DBNAME:
- The target table must already exist in the database and its structure should match the data to be inserted:

```
$sqoop export --connect jdbc:mysql://localhost/test --table business_import --  
export-dir /user/root/business/ --username sqoop_user --password password
```

- Sqoop performs a set of INSERT INTO JDBC operations to populate the target table
- If any INSERT transaction fails (e.g. table constraints are violated), then the export will fail

Using a Staging Table

- During the data export process some of the parallel jobs may fail
- As a result, the target table may end up to be only partially populated
- In some scenarios, you may also end up with duplicate records
- Sqoop helps users avoid data export problems by offering the **--staging-table** option with designates a temp table from where data is inserted in the target table in a single database transaction, this temp table must have the structure identical to that of the target table and exist (and be empty) prior to running the export job

Using a Staging Table

- Each export process uses a **separate JDBC connection** to the database each of which uses its own (isolated from other connections) transactions. Each process commits results individually. There is no two-phase commit arrangement in place and rolling-back one transaction has no effect on other transaction. If an individual export task fails, its current transaction will be rolled back. Any previously-committed data will stay durable in the table, leading to a partial export operation

INSERT and UPDATE Statements

- Sqoop uses a set of JDVC-based INSERT or UPDATE statement to make changes to the content of the target table
- In default mode, Sqoop generates the necessary INSERT statements to inject new rows into the target table
- in update mode, Sqoop generates UPDATE statements that update existing records in the target table

INSERT Operations

- By default, Sqoop export operation injects new rows into the target table
- Sqoop will generate a JDBC-based INSERT statement for each row from the source files
- In most cases, a new and empty table is used to receive the exported rows
- The Export operation will failed if an exception is thrown in the underlying JDBC system (e.g. primary key constraint is violated, or there is type mismatch etc.)

UPDATE Operations

- By specifying the **--update-key** argument to the export command, you can instruct Sqoop to modify existing rows in the target table
- In this case, row attributes are made part of an UPDATE statement that attempts to modify an existing row (if any found)
- In an UPDATE statement modifies no rows (there are no rows matching the where clause), the export process continues without an error

Examples of the Update Operations

- If you have an existing table with data having the following definition:

```
CREATE TABLE T (id INT NOT NULL PRIMARY KEY, tweet VARCHAR(144));
```

and the source data file in HDFS with the following content:

```
1000,whatever...
```

```
1001,whenever...
```

then the following export command

```
$sqoop export --table T --update-key id --export-dir /path/to/datafile --connect
```

will have SQoop generate the following UPDATE statements

```
UPDATE T SET tweet='whatever...' WHERE id = 1000;
```

```
UPDATE T SET tweet='whenever...' WHERE id = 1001;
```

Failed Exports

- Export may fail for any of these reasons:
- Lost connectivity to the target database
- Capacity issues (insufficient size quotas for file system partitions etc)
- Violation of a database constraints (e.g. inserting a duplicate value in the column with a unique index constraint)
- Corrupt data in the source data file in HDFS

Sqoop2

- Currently, in order to run Sqoop, all the sqoop drivers and connectors must be installed on the client machine
- The next iteration of Sqoop, Sqoop2, is re-architected using a service based model where the Sqoop server is set-up remotely from the clients
- The new design helps with security and concurrent multi-user support
- Sqoop2 supports the Sqoop's CLI that is connected to the server via a RESTful end-point
- The project is still under development and not recommended for production use

Summary

- Sqoop is a command-line tool for efficient two-way data transfer between relational databases and HDFS
- You can control the performance of import and export processes by allocating the number of parallel processes
- Data splitting by specifying the column to split the data by is a useful technique to help with Sqoop's performance
- Sqoop supports three imported file formats: two binary and one text

Best Practices

- It pays to use formatting arguments, Sometimes the problem doesn't show up until much later

ID	LABEL	STATUS
1	Critical, test.	ACTIVE
3	By "agent-nd01"	DISABLED

\$ sqoop import --fields-terminated-by , --escaped-by --enclosed-by ""

"1","Critical, test", "ACTIVE"

1,"Critical, test", ACTIVE

"3","By "agent-nd01""", "DISABLED"

3,"By "agent-nd01""",DISABLED

1,Critical,test,ACTIVE

3,By "agent-nd01",DISABLED

Best Practices

- With the power of parallelism comes great responsibility!
- Use direct connectors for fast prototyping and performance

Two types of connectors:

1. common (JDBC)
2. direct (vendor specific batch tools).

Note:

1. Performance
2. Escape characters, type mapping, column and row delimiters may not be supported. •
Binary formats don't work

Best Practices

- Use a boundary query for better performance

Resolve the data skew issue

- Do not use the same table for import and export
- Use an options file for reusability
- Use the proper file format for your needs
- Prefer batch mode when exporting

Best Practices

- Use a staging table
 1. All data is written to staging table first.
 2. Data is copied to the final destination if all tasks succeed: all-or-nothing semantics.
 3. Structure must match exactly: columns and types.
 4. Staging table must exist before and must be empty. (`--clear-staging-table` parameter)

Best Practices

- Aggregate data in Hive

Some Tips

- Use the split-by Option

Split-by option can not be used with import-all-tables Sqoop option (Primary key is must)

All tables need primary key if Import-all-tables Sqoop option is used
Rule to choose Split-by option

1. Choose a column which contains values that are uniformly distributed
2. **primary key -> date column -> integer column - > first column**
3. Sqoop split-by option works in this manner:
4. `SELECT MIN(your_split_by_column) FROM your_table`
5. `SELECT MAX(your_split_by_column) FROM your_table`
6. Finds out boundaries and this number divides by the number of mappers

Some Tips

- Use the split-by Option

Your program fails right in the moment when run into your_split_by_column with null values. Actually, it won't fail. It will just skip those rows. And your solution is not correct.

Changed column with this option, which required in tables with no index columns or multi-column keys

Some Tips

- Manage Database Fetch Size

Problem:

MySQL GC Overhead limit exceeded error message

Solution:

MySQL JDBS Driver

?dontTrackOpenResources=true&defaultFetchSize=1000&useCursorFetch=true

Some Tips

- Convert Dates to Strings

Problem:

Sqoop converting dates to seconds since epoch

Solution:

```
--map-column-java my_date=String
```

Some Tips

- Using implicit and explicit number of mappers matters. Normally Sqoop uses 4 mappers if you do not specify your own requirements
- Using boundary query
 1. Data skew issue
 2. By default, Sqoop will use the entire query as a subquery to calculate max/min:
INEFFECTIVE
 3. **Store boundary values in a separate table - Good for incremental imports. (--last-value)**
 4. Run query prior to Sqoop and save its output in a temporary table

Some Tips

- Import into S3

```
sqoop import --connect jdbc:mysql://localhost/test --table business --fetch-size 10 --username sqoop_user -password password --target-dir s3n://xyz@somehwere/a/b/c --fields-terminated-by='\001' -m 1 --direct
```

Trouble shooting

- Increasing number of mappers in sqoop command gives java heap space error

By default, each map and reduce task runs in its own JVM. Hence, each mapper will consume certain amount of physical memory. As you keep increasing number of mappers, memory requirement will also keep growing. If java process cannot allocate enough memory it throws `java.lang.OutOfMemoryError`. Add property on `$HADOOP_HOME/conf/mapred-site.xml` as below

```
<property>
    <name>mapred.child.java.opts</name>
    <value>-Xmx512m</value>
</property>
```

or add `-Dmapred.child.java.opts=-Xmx4000M` to command line

Real World Considerations

- The --options-file argument, which referred to your local file with your username and password, doesn't work with jobs in Sqoop, The password also can't be specified when creating the job, Sqoop will instead prompt for the password when running the job, how to make this work in an automated script?

You need to use Expect, a Linux automation tool, to supply the password from a local file when it detects Sqoop prompting for a password.

Real World Considerations

```
#!/usr/bin/expect -f

set sqoop_job [lrange $argv 0 0]

# check if argument was supplied
if { $sqoop_job == "" } {
    puts "Usage: <sqoop_job>\n"
    exit 1
}

set fp [open "~/sqoop_pwd" r]
set password [read $fp]

set timeout -1

# launch Sqoop
spawn sqoop job --exec $sqoop_job
match_max 100000

# look for passwd prompt
expect "*?assword:*"

# send password aka $password
send -- "$password\r"

expect eof
```

Real World Considerations

- Mainframe Considerations

1. Data sources unsupported by Sqoop

- DB2
- VSAM, QSAM, or binary log files, which are not addressed by Sqoop, which need to write software to interpret the files and manage proprietary mainframe technologies including COBOL Copybooks, EBCDIC character encoding, packed decimal encoding and others

2. Resource impacts on the mainframe

- SQL pulls can be resource intensive, and the resource impact on the mainframe's real-time transaction processing

Real World Considerations

- Mainframe Considerations

1. Computing costs

2. Storage costs

Storing data on the mainframe for an intermediate transformation step will increase storage requirements and costs

3. Security

Mainframe transaction data can be extremely sensitive and special care should be taken to keep it secure

Real World Considerations

- Mainframe Considerations

4. Connect to the mainframe host z390 via ftp

```
$ sqoop import-mainframe --connect z390 --username david --password-file  
${user.home}/.password
```

5. Selecting the Files to Import

You can use the `--dataset` argument to specify a partitioned dataset name. All sequential datasets in the partitioned dataset will be imported

Real World Considerations

- HP Vertica

```
$sqoop import -m 1 --driver com.vertica.jdbc.Driver --connect  
"jdbc:vertica://*****:5433/dbName" --password "*****" --username  
"*****" --target-dir "/tmp/cdsdj" --verbose --query 'SELECT t.col1 FROM  
schema.tableName t where $CONDITIONS'
```

Real World Considerations

- Teradata

```
$sqoop import -libjars $LIB_JARS Dteradata.db.input.job.type=hive -  
Dteradata.db.input.target.table=hive_table -Dteradata.db.input.target.table.schema="emp_no  
int, birth_date string, first_name string, last_name string, gender string, hire_date string" --  
connect jdbc:teradata://td-host/Database=dbname --connection-manager  
org.apache.sqoop.teradata.TeradataConnManager --username tduser --password tduserpass --  
table tablename
```


Real World Considerations

- Teradata

Note:

Make sure you have the connection manager set in `/etc/sqoop/conf/managers.d`

1. `cd /etc/sqoop/conf`
2. `mkdir managers.d && cd managers.d`
3. `echo '<manager factory class> = <manager factory jar>' > td_connector.txt`
4. Manager factory jar should be put in `/var/lib/sqoop`

Real World Considerations

- Scheduled Sqoop Job using Crontab

1. List the scheduled job

```
crontab -l
```

2. `echo -n "sqoop import --connect jdbc:mysql://localhost/test --username sqoop_user --table business --fetch-size 10 --password-file file:///root/sqoop.password --delete-target-dir" > scheduledjob.sh`

3. `chmod 777 scheduledjob.sh`

4. `crontab -e`

5. `add */2 * * * * /root/scheduledjob.sh >> /root/script_output.log 2>&1`

6. `crontab -l`



SPARK JDBC

Bin Jiang

12/17/2017

What's Spark JDBC

- Spark SQL includes a data source that can read data from other databases using JDBC, the results are returned as a DataFrame and they can easily be processed in Spark SQL or joined with other data sources

Features

- Distributed processing over Spark
- Partitioning
- Only one partitioned column is support
- Transaction isolation
- Truncate table is supported
- Create table column types
- Need to write the code to support incremental import
- Table or custom query import
- Supported database: MySQL, Postgres, Oracle, DB2, SQL Server
- Can provide complex transformation logic
- Save as any format as long as Spark support, e.g. parquet, orc, avro, sequence file, text, csv, JSON, xml
- Destination data source can be HBase, Hive, HDFS, flat file, NoSQL, Azure data lake, S3 and so on

How To Export

```
def main(args: Array[String]): Unit = {  
  
  val spark = SparkSession  
    .builder()  
    .appName("Move data from Hive to MySQL using Spark JDBC")  
    .enableHiveSupport()  
    .getOrCreate()  
  
  val options = Map(  
    "driver" -> "com.mysql.jdbc.Driver",  
    "url" -> "jdbc:mysql://localhost:3306/foodmart_spark",  
    "user" -> "root",  
    "password" -> "hadoop",  
    "dbtable" -> "foodmart_spark.customer"  
  )  
  
  var df = spark.sql("select * from foodmart.customer")  
  df.write.mode("overwrite").format("jdbc").options(options)  
    .option("dbtable", "foodmart_spark.customer").save()  
  
  df = spark.sql("select * from foodmart.product")  
  df.write.mode("overwrite").format("jdbc").options(options)  
    .option("dbtable", "foodmart_spark.product").save()  
  
  df = spark.sql("select * from foodmart.store")  
  df = df.withColumn("last_remodel_date", col("last_remodel_date").cast("date"))  
  df = df.withColumn("first_opened_date", col("first_opened_date").cast("date"))  
  df.write.mode("overwrite").format("jdbc").options(options)  
    .option("dbtable", "foodmart_spark.store").save()  
}
```


How To Import

```
val spark = SparkSession
  .builder()
  .appName("Move data from MySQL to Hadoop using Spark JDBC")
  .enableHiveSupport()
  .getOrCreate()

val options = Map(
  "driver" -> "com.mysql.jdbc.Driver",
  "url" -> "jdbc:mysql://localhost:3306/foodmart_spark",
  "user" -> "root",
  "password" -> "hadoop",
  "dbtable" -> "foodmart_spark.customer"
)

var df = spark.read.format("jdbc").options(options).load()
df.cache()
df.write.mode("overwrite").format("parquet").save("/root/data/foodmart_spark/customer")
df.write.mode("overwrite").format("orc").saveAsTable("foodmart_spark.customer")

df = spark.read.format("jdbc").options(options).option("dbtable", "foodmart_spark.product").load()
df.cache()
df.write.mode("overwrite").format("parquet").save("/root/data/foodmart_spark/product")
df.write.mode("overwrite").format("orc").saveAsTable("foodmart_spark.product")

df = spark.read.format("jdbc").options(options).option("dbtable", "foodmart_spark.store").load()
```

JDBC Options

```
val JDBC_URL = newOption("url")
val JDBC_TABLE_NAME = newOption("dbtable")
val JDBC_DRIVER_CLASS = newOption("driver")
val JDBC_PARTITION_COLUMN = newOption("partitionColumn")
val JDBC_LOWER_BOUND = newOption("lowerBound")
val JDBC_UPPER_BOUND = newOption("upperBound")
val JDBC_NUM_PARTITIONS = newOption("numPartitions")
val JDBC_BATCH_FETCH_SIZE = newOption("fetchsize")
val JDBC_TRUNCATE = newOption("truncate")
val JDBC_CREATE_TABLE_OPTIONS = newOption("createTableOptions")
val JDBC_BATCH_INSERT_SIZE = newOption("batchsize")
val JDBC_TXN_ISOLATION_LEVEL = newOption("isolationLevel")
}
```



APACHE NIFI

Bin Jiang

12/17/2017

What's NIFI SQL Processor

Apache NiFi has four processors for extracting rows from relational databases:

- **ExecuteSQL** - executes an arbitrary SQL statement and returns the results as one FlowFile, in Avro format, containing all of the result records. works with a broad set of statements including stored procedure calls. Designed for general-purpose use, does not have specific features for incremental extraction. ExecuteSQL can accept incoming FlowFiles, and FlowFile attributes may be used in expression language statements to make the SQL.

What's NIFI SQL Processor

- **QueryDatabaseTable** - designed specifically for incremental extraction. Computes SQL queries based on a given table name and incrementing column. Maintains NiFi state data tracking the last incremental value retrieved. Results are formatted as Avro files.

What's NIFI SQL Processor

- **GenerateTableFetch** - used to generate a sequence of paged query statements for use with ExecuteSQL, making it practical to query very large data sets in manageable chunks.

What's NIFI SQL Processor

- **CaptureChangeMySQL** - Retrieves Change Data Capture (CDC) events from a MySQL database. CDC Events include INSERT, UPDATE, DELETE operations. Events are output as individual flow files ordered by the time at which the operation occurred

Features

- Web-based tool and easy to use
- Distributed processing is not naturally supported
- Partitioning
- Full and incremental mode
- Support both tables and custom query import
- Stored procedure import
- Import data as avro, and can be converted into JSON or other format
- Import data into HDFS, Hive and HBase
- Direct database connector is not supported
- Multiple split columns are supported

Features

- Support data compression
- Complex transformation is supported
- Change Data Capture is supported only for MySQL
- Near real time streaming process
- Recovery
- Duplication detection
- Back pressure

Lab Examples

← → ↻ ⓘ localhost:9090/nifi/ ☆ 🌐

PORTONWORKS DATAFLOW **HDF**

3 5,365 / 856.72 KB 0 0 9 11 0 0 18:05:36 UTC 🔍

Navigate

Operate

Change Data Capture for MySQL
Process Group
01601004-9fe2-18f3-74e4-363f659c6578

DELETE

Large Data Transfer In Parallel 3

0 0 0 9 0 0 0 0 0

Queued	5,365 (856.72 KB)	
In	0 (0 bytes) → 0	5 min
Read/Write	770.61 KB / 722.23 KB	5 min
Out	0 → 0 (0 bytes)	5 min

No comments specified

Bulk Data Transfer from MySQL to HDFS

0 0 0 0 4 0 0 0 0

Queued	0 (0 bytes)	
In	0 (0 bytes) → 0	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 → 0 (0 bytes)	5 min

No comments specified

Change Data Capture for MySQL

0 0 0 0 0 0 0 0 0

Queued	0 (0 bytes)	
In	0 (0 bytes) → 0	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 → 0 (0 bytes)	5 min

No comments specified

Incremental Data Transfer from MySQL to Local S...

0 0 0 0 7 0 0 0 0

Queued	0 (0 bytes)	
In	0 (0 bytes) → 0	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 → 0 (0 bytes)	5 min






No comments specified

Controller Services

Process Group Configuration

GENERAL

CONTROLLER SERVICES

Name ^					
		Type	State	Process Group	
	MySQLConnectionPool	DBCPCConnectionPool	 Disabled	7c84501d-d10c-407c-b9f3-1d80e38fe36a	  

Controller Services

Configure Controller Service

SETTINGS

PROPERTIES

COMMENTS

Required field

+

Property	Value
Database Connection URL	<div>?</div> jdbc:mysql://localhost:3306/foodmart_spark
Database Driver Class Name	<div>?</div> com.mysql.jdbc.Driver
Database Driver Location(s)	<div>?</div> /root/TrainingOnHDP/RDBMS2Hadoop/nifi/lib/...
Database User	<div>?</div> root
Password	<div>?</div> Sensitive value set
Max Wait Time	<div>?</div> 500 millis
Max Total Connections	<div>?</div> 8
Validation query	<div>?</div> No value set

CANCEL

APPLY

QueryDatabaseTable

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field



Property		Value	
Database Connection Pooling Service	?	MySQLConnectionPool	→
Database Type	?	Generic	
Table Name	?	demo_products	
Columns to Return	?	productID,productCode,name,quantity,price	
Maximum-value Columns	?	productID	
Max Wait Time	?	0 seconds	
Fetch Size	?	100	
Max Rows Per Flow File	?	0	
Normalize Table/Column Names	?	true	

ExecuteSQL

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field



Property		Value	
Database Connection Pooling Service	?	MySQLConnectionPool	→
SQL select query	?	No value set	
Max Wait Time	?	0 seconds	
Normalize Table/Column Names	?	false	

GenerateTableFetch

Configure Processor

SETTINGS

SCHEDULING

PROPERTIES

COMMENTS

Required field



Property		Value	
Database Connection Pooling Service	?	MySQLConnectionPool	→
Database Type	?	Generic	
Table Name	?	sales_fact_dec_1998	
Columns to Return	?	product_id, time_id, customer_id, promotion_id,...	
Maximum-value Columns	?	time_id	
Max Wait Time	?	0 seconds	
Partition Size	?	10	



KAFKA CONNECT

Bin Jiang

12/17/2017

What's Kafka Connect

- Kafka Connect is a framework included in Apache Kafka that integrates Kafka with other systems. Its purpose is to make it easy to add new systems to your scalable and secure stream data pipelines.
- To copy data between Kafka and another system, users instantiate Kafka Connectors for the systems they want to pull data from or push data to. Source Connectors import data from another system (e.g. a relational database into Kafka) and Sink Connectors export data (e.g. the contents of a Kafka topic to an HDFS file).
- JDBC Source Connector

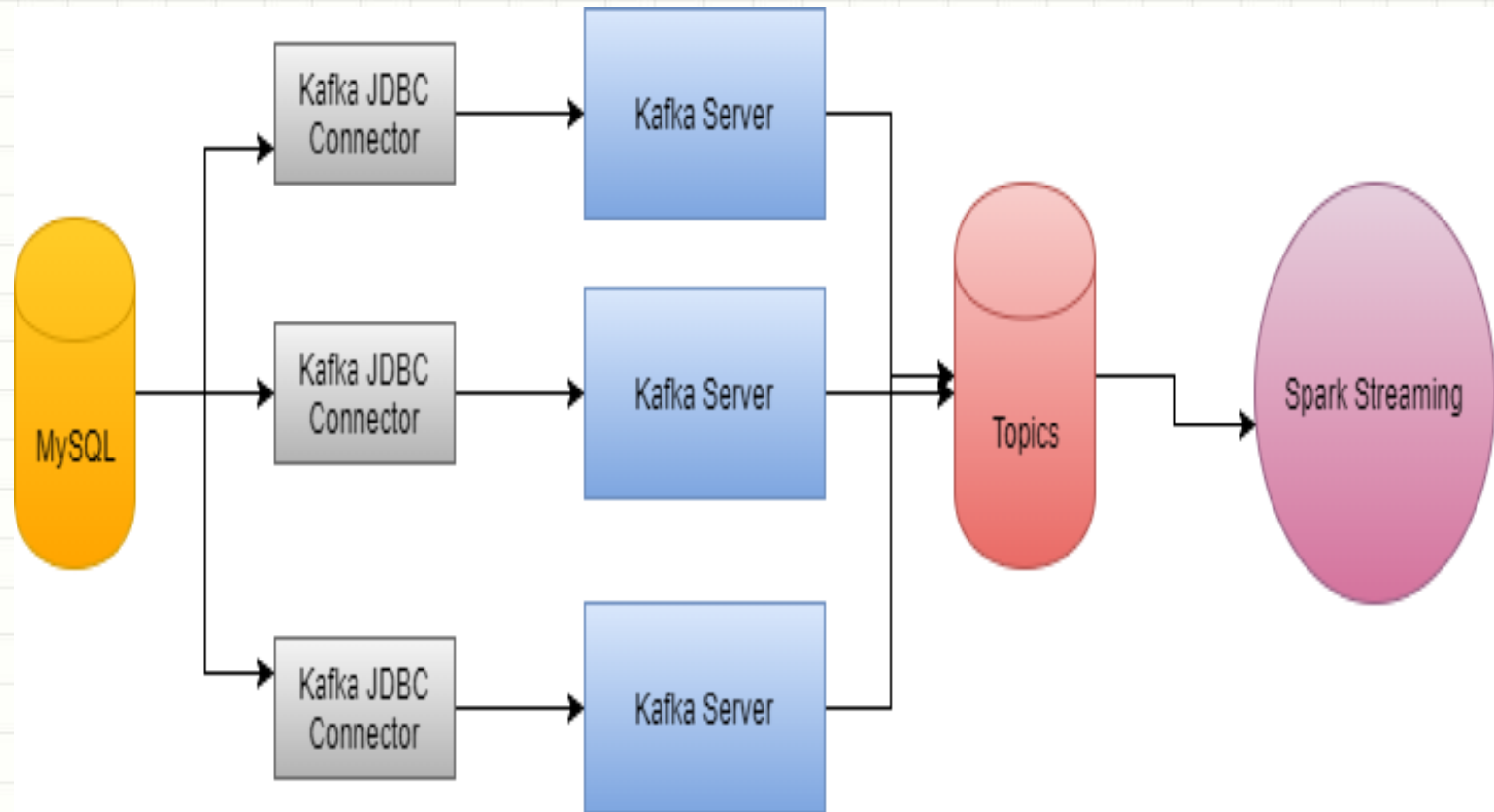
Features

- Supports copying tables with a variety of JDBC data types
- Adding and removing tables from the database dynamically, whitelists and blacklists,
- Controlling how data is incrementally copied from the database.
- Incrementing Column
- Timestamp Column
- Custom Query
- Bulk Load
- Multiple tasks running over multiple brokers
- Limited Transformation
- Streaming Process
- Recovery

Configuration

```
name=mysql-jdbc-autoincrement
connector.class=io.confluent.connect.jdbc.JdbcSourceConnector
tasks.max=1
connection.url=jdbc:mysql://localhost:3306/foodmart_spark?user=root&password=hadoop
table.whitelist=demo_products
mode=incrementing
incrementing.column.name=productID
topic.prefix=mysql-jdbc-
```

Architect of Kafka Connect





STREAMSETS DATA COLLECTOR

Bin Jiang

12/17/2017

What's StreamSets Data Collector

- StreamSets Data Collector is a lightweight, powerful engine that streams data in real time. Use Data Collector to route and process data in your data streams.
- To define the flow of data for Data Collector, you configure a pipeline. A pipeline consists of stages that represent the origin and destination of the pipeline, and any additional processing that you want to perform. After you configure the pipeline, you click Start and Data Collector goes to work.
- Data Collector processes data when it arrives at the origin and waits quietly when not needed. You can view real-time statistics about your data, inspect data as it passes through the pipeline, or take a close look at a snapshot of data.

RDBMS Origins

- JDBC Multitable Consumer
- JDBC Query Consumer
- MapR DB CDC
- MySQL Binary Log
- Oracle CDC Client
- SQL Server CDC Client
- SQL Server Change Tracking


Features

- Multiple Tables Ingestion
- Custom Query
- Change Data Capture (Insert, Delete, Update, Upsert, Undelete, Replace)
- Multithreaded Table Processing
- Multithreaded Partition Processing
- Batch Strategies
- JDBC Header Attributes
- Event Generation
- Full and Incremental Mode
- Recovery
- Distributed Processing
- Dynamic Table
- Back Pressure
- Replication of database

Supported Features of Transformation



- Record Deduplication
- Data Masking
- Data Hasher
- Content Merge
- Field Order
- Field Flattener
- Data Pivot
- Field Splitter

Data Transfer from MySQL

 StreamSets

Pipelines / Data Transfer from MySQL to Data Lake

☑ All Changes Saved

 Capture MySQL Changes →  Save to Elasticsearch

Destinations

Type to search

- Amazon S3
- Azure Data Lake Store
- Azure Event Hub
- Azure IoT Hub
- CoAP
- Cassandra
- CoAP Client
- Einstein Analytics
- Elasticsearch
- Flume
- Google BigQuery

Data Transfer from MySQL to Data Lake ▾

Retry Attempts ⓘ -1

Max Pipeline Memory (MB) ⓘ $\${jvm:maxMemoryMB()} * 0.85\}$

On Memory Exceeded ⓘ Log ▾

Rate Limit (records / sec) ⓘ 0

Max runners ⓘ 0