Modeling and Managing Data with Impala and Hive

Prof. Hyuk-Yoon Kwon

Modeling and Managing Data in Impala and Hive

In this chapter you will learn

- How Impala and Hive use the Metastore
- How to use Impala SQL and HiveQL DDL to create tables
- How to create and manage tables using Hue or HCatalog
- How to load data into tables using Impala, Hive, or Sqoop

How Hive and Impala Load and Store Data (1)

- **Queries operate on tables, just like in an RDBMS**
  - A table is simply an HDFS directory containing one or more files
  - Default path: **`/user/hive/warehouse/<table_name>`**
  - Supports many formats for data storage and retrieval
- **What is the structure and location of tables?**
  - These are specified when tables are created
  - This metadata is stored in the *Metastore*
    - Contained in an RDBMS such as MySQL
- **Hive and Impala work with the same data**
  - Tables in HDFS, metadata in the Metastore

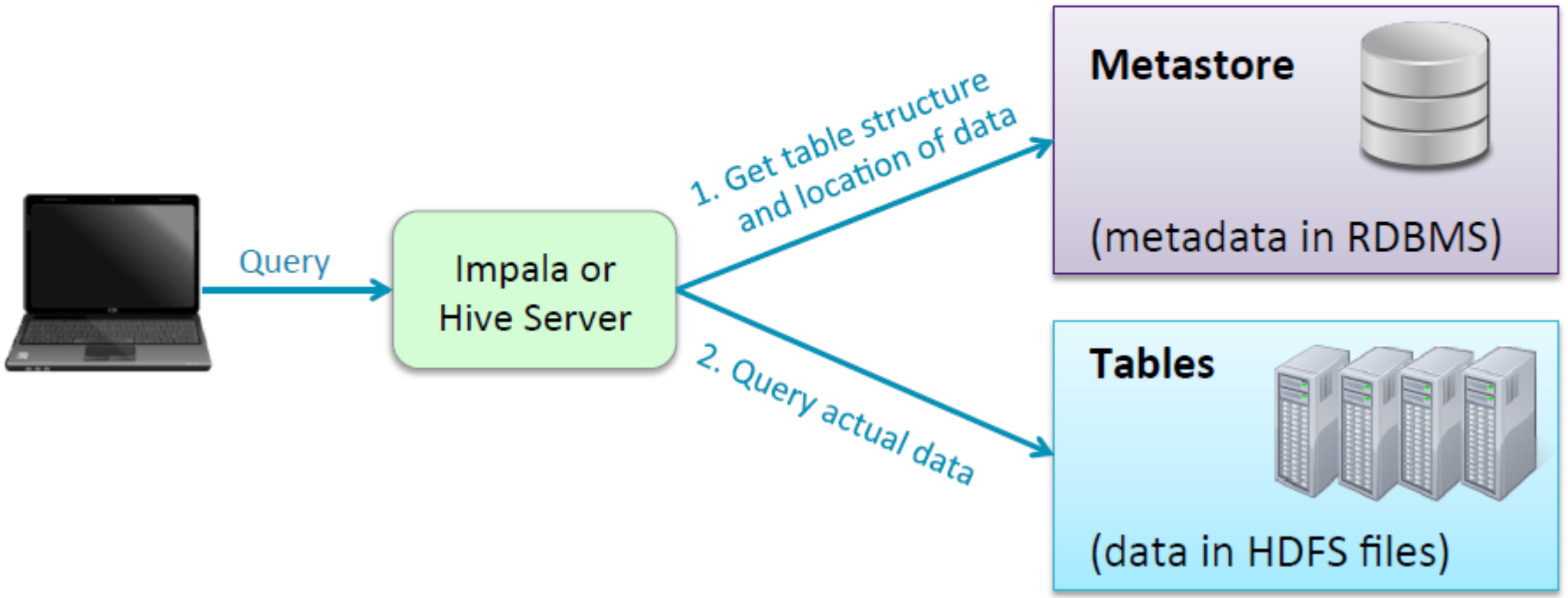
hive ⇒ load all data in directory, not only individual file

not store metadata in HDFS.

How Hive and Impala Load and Store Data (2)

- **Hive and Impala use the Metastore to determine data format and location**
  - The query itself operates on data stored in HDFS

which schema is applied?



must define schema of table when creating it.
create table w/ rigid structure
data should satisfy specified conditions / restrictions before added in table

able to store data first, w/o knowing its format
import data w/o defining schema

Hadoop & Hive ⇒ require to know format of data only when analyzing them, rather than storing them

conflicts between expected & actual data will not be detected when records are added in RDBMS
⇒ RDBMS: pre-defined strict structure of the schema. invalid data ⇒ not allowed to store
Hadoop: possible to store all data

Detection is occured at query executing

Data and Metadata

- ***Data* refers to the information you store and process**
  - Billing records, sensor readings, and server logs are examples of data
- ***Metadata* describes important aspects of that data** schema info
  - Field name and order are examples of metadata

Metadata
Schema

Data
instance

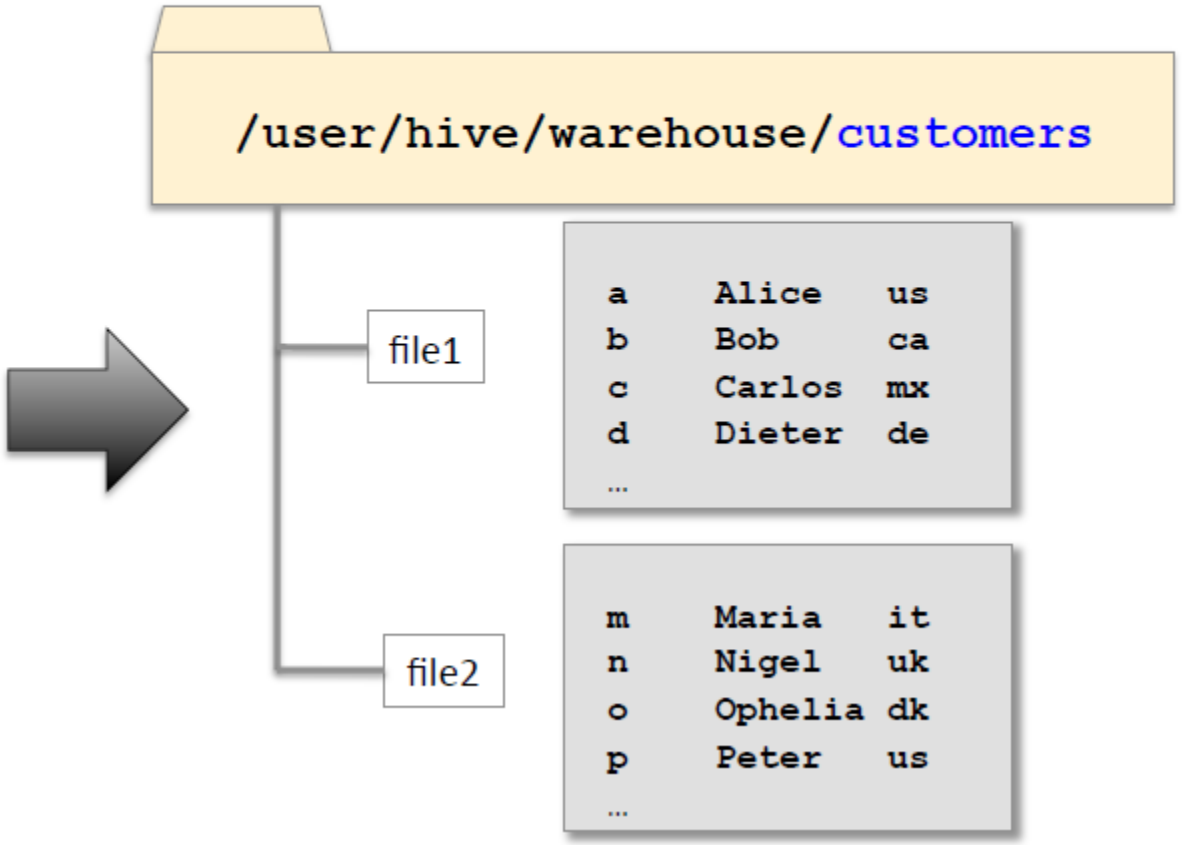
cust_id	name	country
001	Alice	us
002	Bob	ca
003	Carlos	mx
…	…	…
392	Maria	it
393	Nigel	uk
394	Ophelia	dk
…	…	…

The Data Warehouse Directory

- By default, data is stored in the HDFS directory `/user/hive/warehouse`
- Each table is a subdirectory containing any number of files

customers table

cust_id	name	country
001	Alice	us
002	Bob	ca
003	Carlos	mx
…	…	…
392	Maria	it
393	Nigel	uk
394	Ophelia	dk
…	…	…



Defining Databases and Tables

- **Databases and tables are created and managed using the DDL (Data Definition Language) of HiveQL or Impala SQL**
  - Very similar to standard SQL DDL
  - Some minor differences between Hive and Impala DDL will be noted

Creating a Database

- **Hive and Impala databases are simply namespaces**
  - Helps to organize your tables
- **To create a new database** mapping structure

```
CREATE DATABASE loudacre;
```

1. Adds the database definition to the metastore
2. Creates a storage directory in HDFS

   e.g. /user/hive/warehouse/loudacre.db

- **To conditionally create a new database**
  - Avoids error in case database already exists (useful for scripting)

```
CREATE DATABASE IF NOT EXISTS loudacre;
```

Removing a Database

- **Removing a database is similar to creating it**
  - Just replace **CREATE** with **DROP**

```
DROP DATABASE loudacre;
```

```
DROP DATABASE IF EXISTS loudacre;
```

- **These commands will fail if the database contains tables**
  - In Hive: Add the **CASCADE** keyword to force removal
  - Caution: this command might remove data in HDFS!


```
DROP DATABASE loudacre CASCADE;
```

Practice

1. **Create a database named “practice”**
2. **Remove the database named “practice”**
3. **Try to use conditional commands to create and remove the database and compare the results with the case the conditional commands are not used**
4. **Try to use CASCADE to remove the database**
    1. Create a new database
    2. In the created database, make a new table
    3. Remove the database without CASCADE
    4. Remove the database using CASCADE

Data Types

파일 load 시 schema 검증안씀. 쿼리 처리 시 문제 발생.

- **Each column is assigned a specific data type**
  - These are specified when the table is created
  - NULL values are returned for non-conforming data in HDFS
- **Here are some common data types**

Name	Description	Example Value
STRING	Character data (of any length)	Alice
BOOLEAN	TRUE or FALSE	TRUE
TIMESTAMP	Instant in time	2014-03-14 17:01:29
INT	Range: same as Java int	84127213
BIGINT	Range: same as Java long	7613292936514215317
FLOAT	Range: same as Java float	3.14159
DOUBLE	Range: same as Java double	3.14159265358979323 85

Hive (not Impala) also supports a few complex types such as maps and arrays

Creating a Table (1)

DDL

- **Basic syntax for creating a table:**

```
CREATE TABLE tablename (colname DATATYPE, ...)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY char
  STORED AS {TEXTFILE|SEQUENCEFILE|…}
```

optional

- **Creates a subdirectory in the database's warehouse directory in HDFS**
  - Default database:
    /user/hive/warehouse/*tablename*
  - Named database:
    /user/hive/warehouse/*dbname*.db/*tablename*

Creating a Table (2)

```
CREATE TABLE tablename (colname DATATYPE, ...)
  ROW FORMAT DELIMITED
  FIEL
  STOR
```

Specify a name for the table, and list the column names and datatypes (see later)

Creating a Table (3)

```
CREATE TABLE tablename (colname DATATYPE, ...)
   ROW FORMAT DELIMITED
   FIELDS TERMINATED BY char
   STOR
```

row-wise

This line states that fields in each file in the table's directory are delimited by some character. The default delimiter is Control-A, but you may specify an alternate delimiter...

Creating a Table (4)

```
CREATE TABLE tablename (colname DATATYPE, ...)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY char
  STORED AS {TEXTFILE|SEQUENCEFILE|...}
```

Column-wise

…for example, tab-delimited data would require that you specify **`FIELDS TERMINATED BY '\t'`**

Creating a Table (5)

```
CREATE TABLE tablename (colname DATATYPE, ...)
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY char
  STORED AS {TEXTFILE|SEQUENCEFILE|...}
```

Finally, you may declare the file format. **STORED AS TEXTFILE** is the default and does not need to be specified.
Other formats will be discussed later in the course.

Example Table Definition

- **The following example creates a new table named `jobs`**
  - Data stored as text with four comma-separated fields per line

```
CREATE TABLE jobs (
    id INT,
    title STRING,
    salary INT,
    posted TIMESTAMP
  )
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ',';
```

example.jobs
prefix for specifing DB, 「example.db」

  - Example of corresponding record for the table above

```
1,Data Analyst,100000,2013-06-21 15:52:03
```

Creating Tables Based On Existing Schema

- Use `LIKE` to create a new table based on an existing table definition

```
CREATE TABLE jobs_archived LIKE jobs;
```

- Column definitions and names are derived from the existing table
  - New table will contain no data

⇒ create new metadata & directory for new table.

Creating Tables Based On Existing Data

- **Create a table based on a SELECT statement**
  - Often know as 'Create Table As Select' (CTAS)

```
CREATE TABLE ny_customers AS
  SELECT cust_id, fname, lname
  FROM customers
  WHERE state = 'NY';
```

⇒ kind of creating actual view

- **Column definitions are derived from the existing table**
- **Column names are inherited from the existing names**
  - Use aliases in the SELECT statement to specify new names
- **New table will contain the selected data**

Controlling Table Data Location

- By default, table data is stored in the warehouse directory
- This is not always ideal
  - Data might be shared by several users
- Use LOCATION to specify the directory where table data resides

ALTER TABLE ...
⇒ change location of existing table

```
CREATE TABLE jobs (
    id INT,
    title STRING,
    salary INT,
    posted TIMESTAMP
  )
  ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
  LOCATION '/loudacre/jobs';
```

Externally Managed Tables

- **CAUTION: Dropping a table removes its data in HDFS**
  - Tables are "managed" or "internal" by default ⇒ when dropping table, all data in table deleted together.
- **Using EXTERNAL when creating the table avoids this behavior** ... using ALTER TABLE for existing table also.
  - Dropping an *external* table removes only its *metadata* ⇒ remaining table data

generally used together...

```
CREATE EXTERNAL TABLE adclicks
  ( campaign_id STRING,
    click_time TIMESTAMP,
    keyword STRING,
    site STRING,
    placement STRING,
    was_clicked BOOLEAN,
    cost SMALLINT)
  LOCATION '/loudacre/ad_data';
```

Practice

1. **Create jobs table in the previous slide**
2. **Create a table based on existing schema**
   - Existing schema: device
   - A new table: device_copy
3. **Create a table based on existing data**
   - Existing table: device
   - Condition for inserting the data: device_num = 5
   - A new table: device_5
4. **Make a new table jobs_2 by controlling the location**
   - Location: /user/temp/

5. **Make a new table jobs_3 for externally managed tables**

6. **Compare two tables: jobs_2(internal) and jobs_3(external)**

- Insert data into each table
- Drop table
- Check the difference

Exploring Tables (1)

- The `SHOW TABLES` command lists all tables in the current database

```
SHOW TABLES;
+---------------+
|   tab_name    |
+---------------+
| accounts      |
| employees     |
| job           |
| vendors       |
+---------------+
```

- The `DESCRIBE` command lists the fields in the specified table

```
DESCRIBE jobs;
+--------+-----------+---------+
| name   | type      | comment |
+--------+-----------+---------+
| id     | int       |         |
| title  | string    |         |
| salary | int       |         |
| posted | timestamp |         |
+--------+-----------+---------+
```

Exploring Tables (2)

- **DESCRIBE FORMATTED** also shows table properties

```
DESCRIBE FORMATTED jobs;
+------------------+----------------------------------------+---------+
| name             | type                                   | comment |
+------------------+----------------------------------------+---------+
| # col_name       | data_type                              | comment |
| id               | int                                    | NULL    |
| title            | string                                 | NULL    |
| salary           | int                                    | NULL    |
| posted           | timestamp                              | NULL    |
|                  | NULL                                   | NULL    |
| # Detailed Table | NULL                                   | NULL    |
| Information
| Database:        | default                                | NULL    |
| Owner:           | training                               | NULL    |
| CreateTime:      | Wed Jun 17 09:41:23 PDT 2015           | NULL    |
| LastAccessTime:  | UNKNOWN                                | NULL    |
| Protect Mode:    | None                                   | NULL    |
| Retention:       | 0                                      | NULL    |
| Location:        | hdfs://localhost:8020/loudacre/jobs    | NULL    |
| Table Type:      | MANAGED_TABLE                          | NULL    |
...
```

Exploring Tables (3)

- **SHOW CREATE TABLE** displays the SQL command to create the table

```
SHOW CREATE TABLE jobs;
+-----------------------------------------------------+
| CREATE TABLE default.jobs                           |
|   id INT,                                           |
|   title STRING,                                     |
|   salary INT,                                       |
|   posted TIMESTAMP                                  |
| )                                                   |
| ROW FORMAT DELIMITED FIELDS TERMINATED BY ','       |
…
```

Practice

1. Make a table in Impala-Shell

- Check the result for SHOW TABLES
- Check the result for DESCRIBE and DESCRIBE FORMATTED
- Check the result for SHOW CREATE TABLE
- Check the result in Beeline

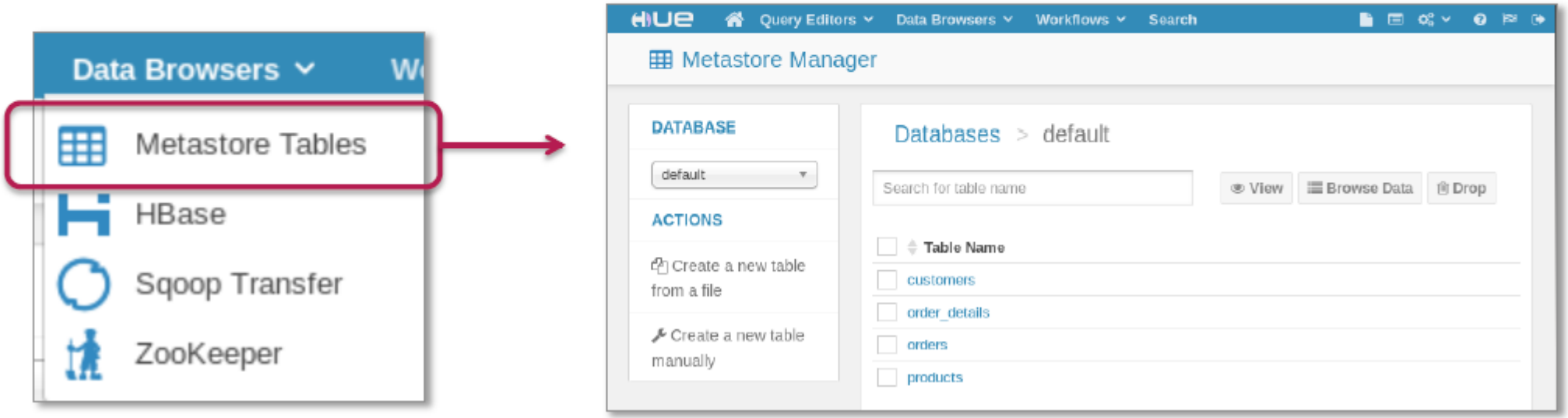
2. Check the difference between Impala-Shell and Beeline

- Make another table in Beeline
- Check the result using the commands above
- Check the result in Impala-Shell using the commands above
- Execute “invalidate metadata”, then check the result in Impala-Shell

beeline에서 생성한 table은 impala에서 바로 확인할 수 없음!
⇒ invalidate metadata; 실행 이후에 확인 가능.
(∵ beeline과 impala shell의 caching 정책이 다르기 때문!)

Using the Hue Metastore Manager

- The Hue Metastore Manager
  - An alternative to using SQL commands to manage metadata
  - Allows you to create, load, preview, and delete databases and tables
    - Not all features are supported yet



Data Validation

- **Impala and Hive are 'schema on read'**
  - Unlike an RDBMS, they do not validate data on insert
    - Files are simply moved into place
  - Loading data into tables is therefore very fast
  - Errors in file format will be discovered when queries are performed
- **Missing or invalid data will be represented as `NULL`** automatically

Loading Data From HDFS Files

copy files into HDFS!

- **To load data, simply add files to the table's directory in HDFS**
  - Can be done directly using the `hdfs dfs` commands
  - This example loads data from HDFS into the `sales` table

```
$ hdfs dfs -mv \
  /tmp/sales.txt /user/hive/warehouse/sales/
```

local HDFS

- **Alternatively, use the `LOAD DATA INPATH` command**
  - Done from within Hive or Impala
  - This *moves* (copies) data within HDFS, just like the command above
  - Source can be either a file or directory

```
LOAD DATA INPATH '/tmp/sales.txt'
   INTO TABLE sales;
```

local HDFS

Overwriting Data From Files

- **Add the `OVERWRITE` keyword to delete all records before import**
  - Removes all files within the table's directory
  - Then moves the new files into that directory

```
LOAD DATA INPATH '/tmp/sales.txt'
  OVERWRITE INTO TABLE sales;
```

HW Assignment #1 – Inserting Data from a File

1. **Make a simple python script to generate a given numbers of strings (one per line)**
   - python random_strings.py 100
2. **Make three files using the python script to generate random strings with 10, 100, and 1000 numbers, respectively (called, ten_strings.txt, hundred_strings.txt, and thousand_strings.txt)**
3. **Upload the generated files into HDFS**
4. **Create a table in Impala to store the generated data**
5. **Load a file, called thousand_strings.txt, into the table and check the total inserted number of data**
6. **Append a file, called hundred_strings.txt, into the table and check the total inserted number of data**
7. **Overwrite a file, called ten_strings.txt into the table and check the total inserted number of data**

Appending Selected Records to a Table

- **Another way to populate a table is through a query**
  - Use **INSERT INTO** to add results to an *existing* Hive table

```
INSERT INTO TABLE accounts_copy
  SELECT * FROM accounts;
```

  - Specify a **WHERE** clause to control which records are appended

```
INSERT INTO TABLE loyal_customers
  SELECT * FROM accounts
  WHERE YEAR(acct_create_dt) = 2008
    AND acct_close_dt IS NULL;
```

Practice – Inserting Selected Records to a Table

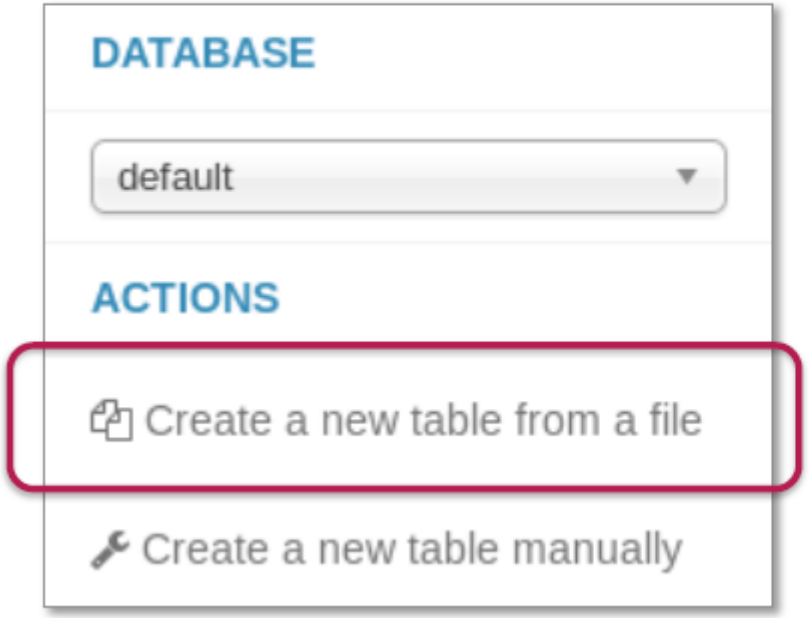
1. **Insert all data in device table into device_copy table**

2. **Append the data whose device_num is greater than 5 in device table into device_copy table**

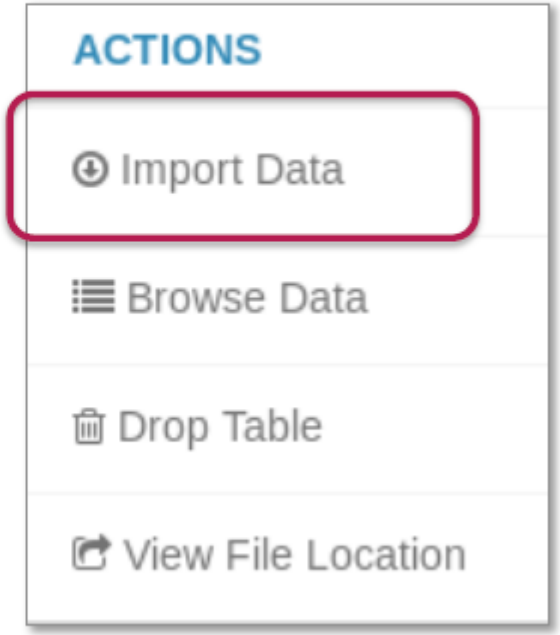
Loading Data Using the Metastore Manager

- **The Metastore Manager provides two ways to load data into a table**

Table creation wizard



Import data wizard



Practice – Hue Metastore Manager

1. **Create a new table from a file using Hue Metastore Manager**
   - A new table: device_meta
   - A file: /user/training/device/part-m-00000

2. **Execute a SQL to the device_meta table for finding the results**
   - Condition: device_num = 5

Loading Data From a Relational Database

- Sqoop has built-in support for importing data into Hive and Impala
- Add the `--hive-import` option to your Sqoop command
  - Creates the table in the Hive metastore
  - Imports data from the RDBMS to the table's directory in HDFS

```
$ sqoop import \
  --connect jdbc:mysql://localhost/loudacre \
  --username training \
  --password training \
  --fields-terminated-by '\t' \
  --table employees \
  --hive-import
```

  - Note that `--hive-import` creates a table accessible in both Hive and Impala

The Metastore and HCatalog

- **HCatalog is a Hive sub-project that provides access to the Metastore**
  - Accessible via command line and REST API
  - Allows you to define tables using HiveQL DDL syntax
  - Access those tables from Hive, Impala, MapReduce, Pig, and other tools
  - Included with CDH 4.2 and later

Creating Tables in HCatalog

- **HCatalog uses Hive's DDL (data definition language) syntax**
  - You can specify a single command using the `-e` option

```
$ hcat -e "CREATE TABLE vendors \
      (id INT, company STRING, email STRING) \
      ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' \
      LOCATION '/dualcore/vendors'"
```

- **Tip: save longer commands to a text file and use the `-f` option**
  - If the file has more than one command, separate each with a semicolon

```
$ hcat -f createtable.txt
```

Displaying Metadata in HCatalog

- The **SHOW TABLES** command also shows tables created directly in Hive

```
$ hcat -e 'SHOW TABLES'
employees
vendors
```

- The **DESCRIBE** command lists the fields in a specified table
  - Use DESCRIBE FORMATTED instead to see detailed information

```
$ hcat -e 'DESCRIBE vendors'
id          int
company     string
email       string
```

Removing a Table in HCatalog

- **The `DROP TABLE` command has the same behavior as it does in Hive and Impala**
  - Caution: this will remove the data as well as the metadata (unless table is **`EXTERNAL`**)!

```
$ hcat -e 'DROP TABLE vendors'
```

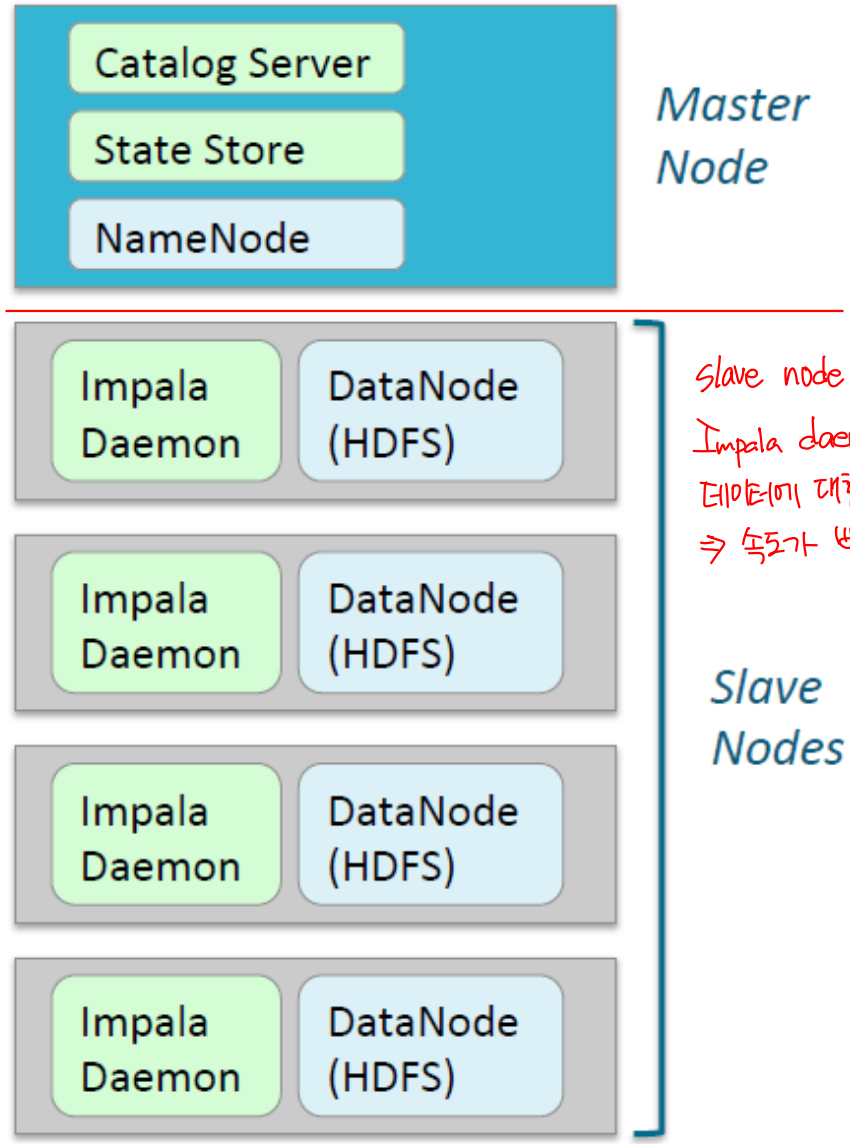
Practice - HCatalog ⇒ handling table itself, not data

1. Create a table vender writing a SQL in hcat command
2. Remove a table vender writing a SQL in hcat command
3. Create a table vender using the SQL file
4. Execute SHOW TABLES and DESCRIBE to check the created table

Impala in the Cluster

metadata caching in impala ⇒ executing command w/ invalid metadata
why invalidate metadata; is required?

- **Each slave node in the cluster runs an Impala daemon**
  - Co-located with the HDFS slave daemon (DataNode)
- **Two other daemons running on master nodes support query execution**
  - The **State Store** daemon
    - Provides lookup service for Impala daemons
    - Periodically checks status of Impala daemons
  - The **Catalog** daemon
    - Relays metadata changes to all the Impala daemons in a cluster



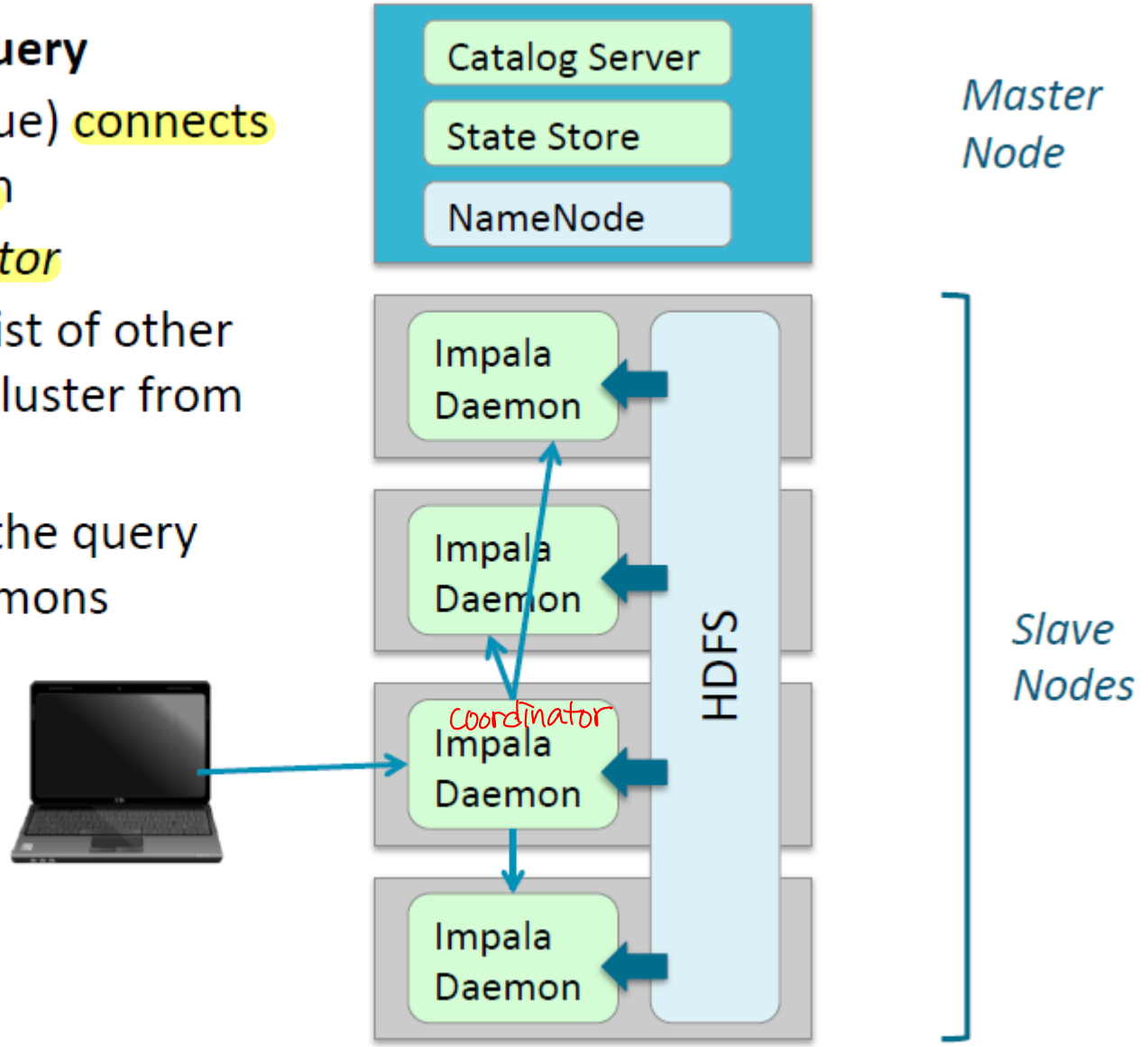
Slave node 별
Impala daemon이 datanode의 저장/관리하는 데이터에 대한 쿼리 요청을 로컬에서 처리.
⇒ 속도가 빠름.

periodically update catalog data(metadata) of slaves' impala daemons ⇒ user manages only metadata of master node

How Impala Executes a Query

- **Impala daemon plans the query**
  - Client (impala-shell or Hue) connects to a local impala daemon
    - This is the *coordinator*
  - Coordinator requests a list of other Impala daemons in the cluster from the State Store
  - Coordinator distributes the query across other Impala daemons
  - Streams results to client

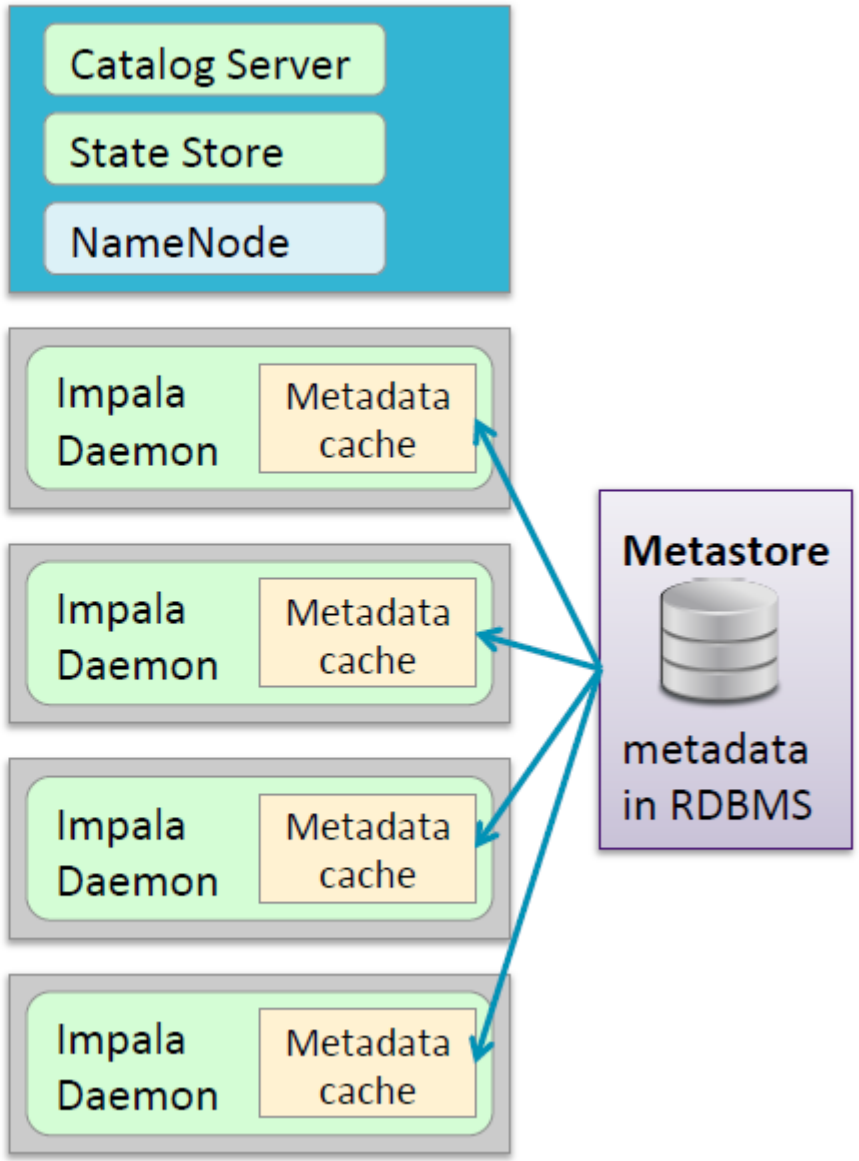
load balancer makes impala daemons to avoid overloaded status.
⇒ no bottleneck of coordinator!



Metadata Caching (1)

- **Impala daemons cache metadata**
  - ① – The tables' schema definitions
  - ② – The locations of tables' HDFS blocks
- **Metadata is cached from the Metastore at startup**

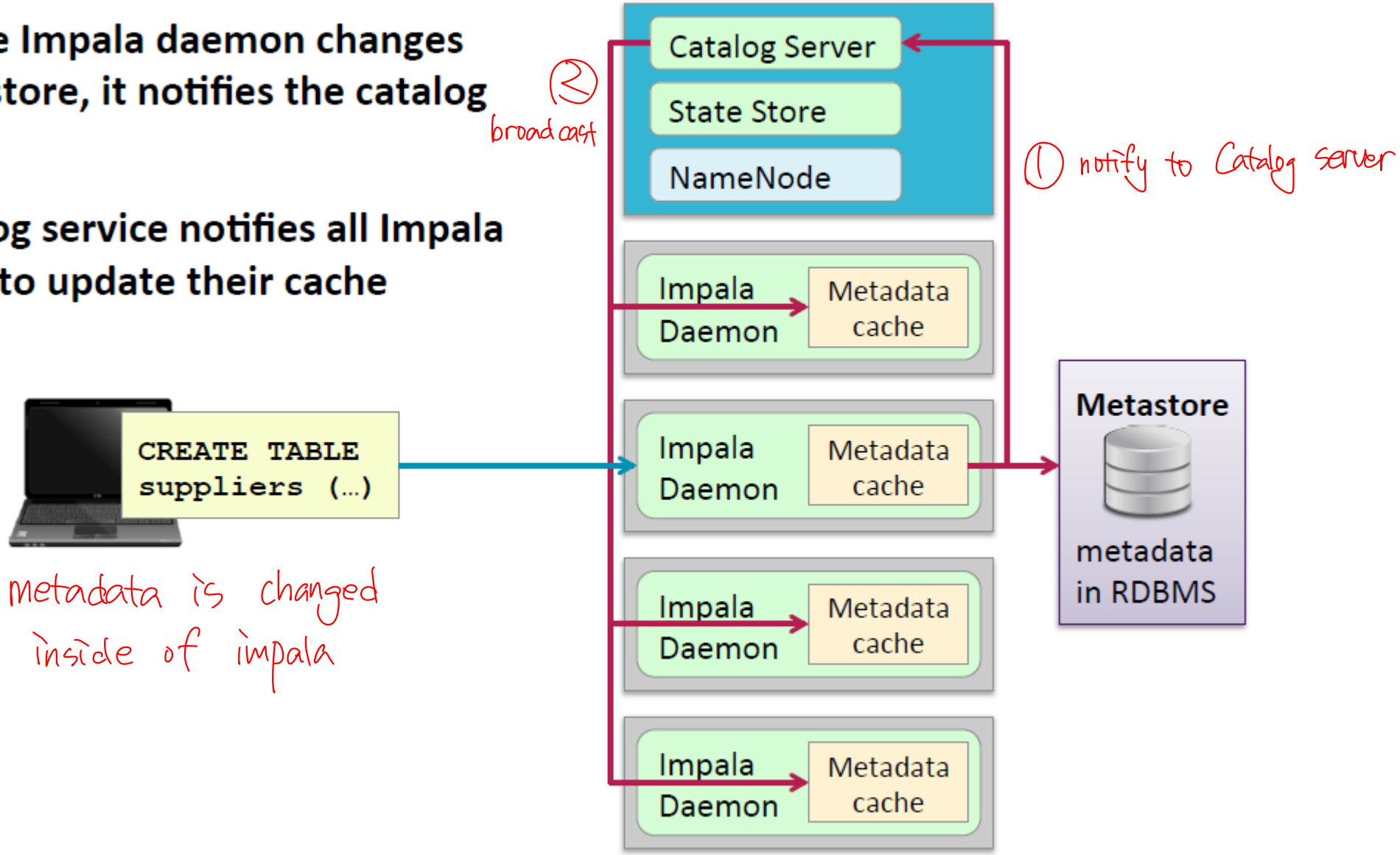
Cached data in the nodes are not changed when update is occured.



Metadata Caching (2)

updating cached data => depending on how metadata change occurs.

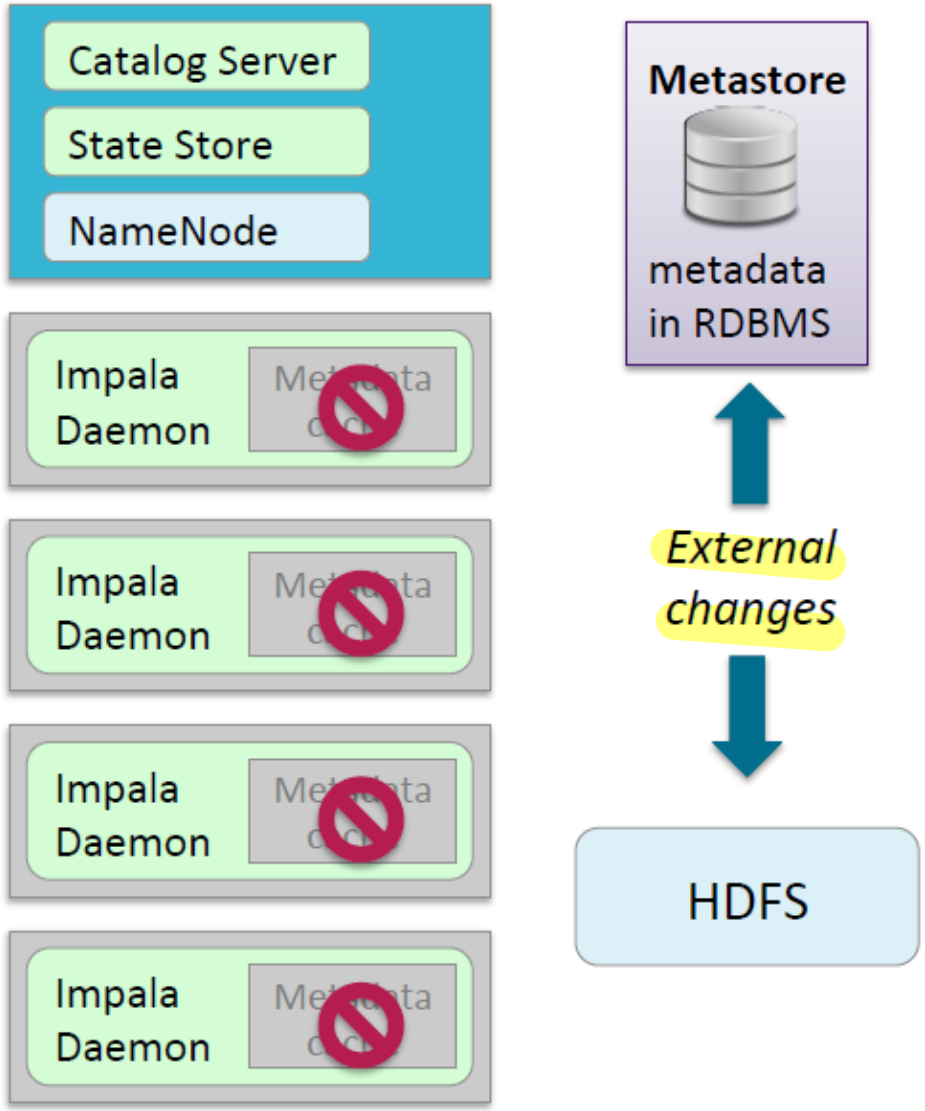
- When one Impala daemon changes the metastore, it notifies the catalog service
- The catalog service notifies all Impala daemons to update their cache



External Changes and Metadata Caching

update from outside of impala.
⇒ no auto-update

- **Metadata updates made *from outside of Impala* are not known to Impala, e.g.**
  - Changes via Hive, HCatalog or Hue Metadata Manager
  - Data added directly to directory in HDFS
- **Therefore the Impala metadata caches will be invalid**
- **You must manually refresh or invalidate Impala's metadata cache**



Updating the Impala Metadata Cache ⇒

External Metadata Change	Required Action	Effect on Local Caches
New table added	**INVALIDATE METADATA** (with no table name) <lazy reloading> reload when requested.	Marks the entire cache as stale; metadata cache is reloaded as needed.
Table schema modified *or* New data added to a table	**REFRESH <table>** ⇒ preferred. <eager reloading> force coming query will use the table daemon wants to be ready before executing	Reloads the metadata for one table *immediately*. Reloads HDFS block locations for new data files only.
Data in a table extensively altered, such as by HDFS balancing	**INVALIDATE METADATA <table>**	Marks the metadata for a single table as stale. When the metadata is needed, all HDFS block locations are retrieved.

expensive

Catalog daemon broadcast updates to all impala daemons when the actions are executed

... necessary to only run actions for single impala daemon!

Essential Points

- **Each table maps to a directory in HDFS**
  - Table data is stored as one or more files
  - Default format: plain text with delimited fields
- **The Metastore stores data *about* the data in an RDBMS**
  - E.g. Location, column names and types
- **Tables are created and managed using the Impala SQL or HiveQL Data Definition Language**
- **Impala caches metadata from the Metastore**
  - Invalidate or refresh the cache if tables are modified outside Impala
- **HCatalog provides access to the Metastore from tools outside Hive or Impala (e.g. Pig, MapReduce)**

Essential Points

- **Impala and Hive are tools for performing SQL queries on data in HDFS**
- **HiveQL and Impala SQL are very similar to SQL-92**
  - Easy to learn for those with relational database experience
  - However, does *not* replace your RDBMS
- **Hive generates jobs that run on the Hadoop cluster data processing engine**
  - Runs MapReduce jobs on Hadoop based on HiveQL statements
- **Impala execute queries directly on the Hadoop cluster**
  - Uses a very fast specialized SQL engine, not MapReduce