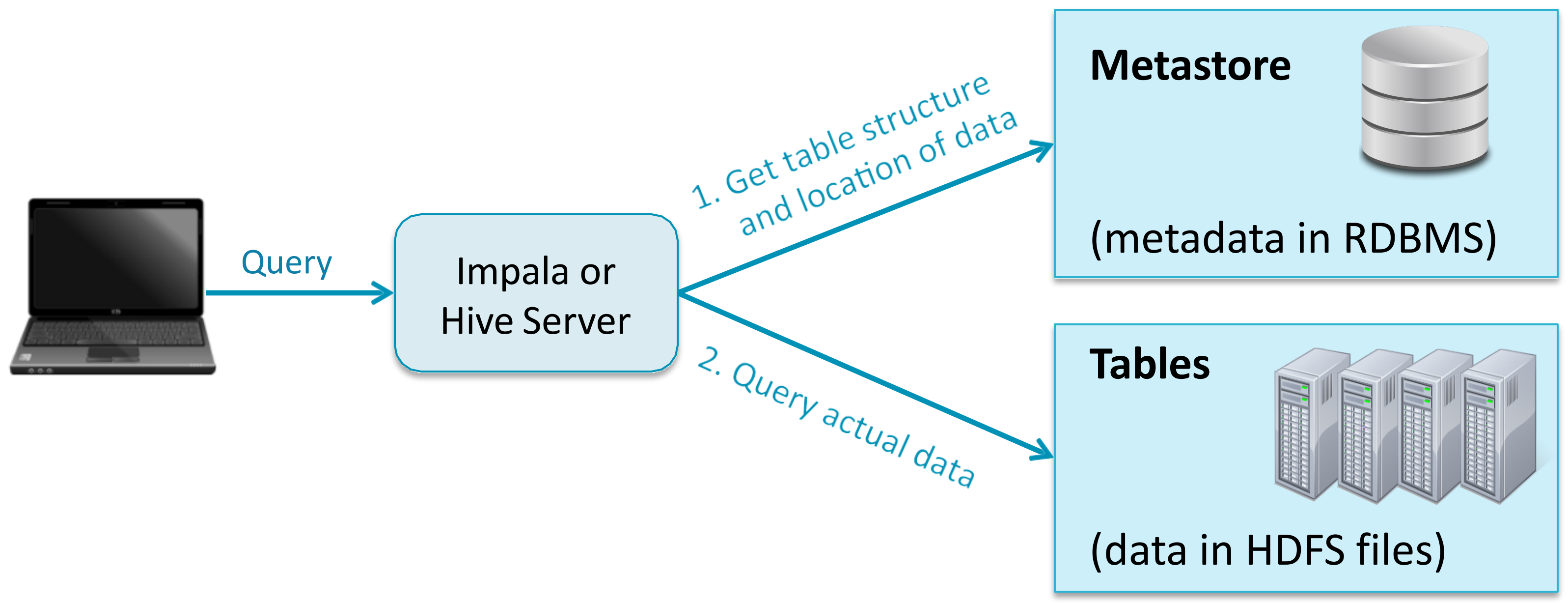
®

Apache Hive Data Management

Recap: How Hive and Impala Load and Store Data

- **Hive and Impala use the metastore to determine data format and location**
  - The query itself operates on data stored in a filesystem (typically HDFS)

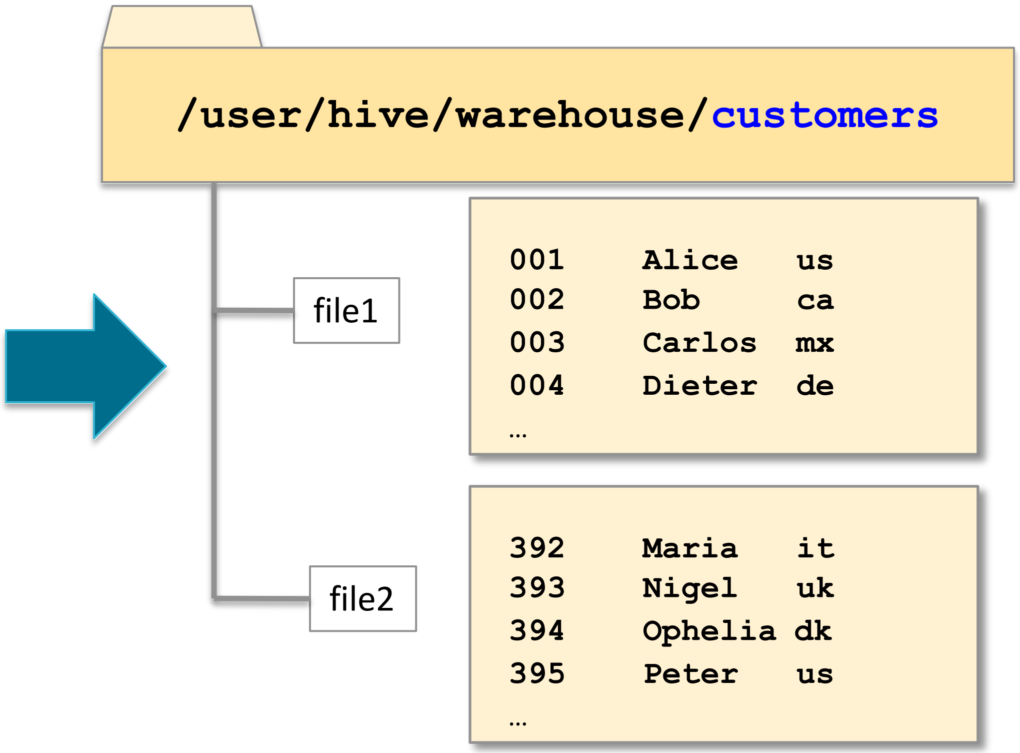


The Warehouse Directory

- **By default, Hive and Impala store data 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
…	…	…



```
001     Alice    us
002     Bob      ca
003     Carlos   mx
004     Dieter   de
…
```

```
392     Maria    it
393     Nigel    uk
394     Ophelia dk
395     Peter    us
…
```

Creating a Database

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

```
CREATE DATABASE dualcore;
```

1. Adds the database definition to the metastore
2. Creates a storage directory in HDFS
   For example, **`/user/hive/warehouse/dualcore.db`**

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

```
CREATE DATABASE IF NOT EXISTS dualcore;
```

Creating a Table (1)

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

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

- **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`

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 of corresponding record for the table above

```
1,Data Analyst,135000,2016-12-21 15:52:03
```

Controlling Table Data Location

- **By default, table data is stored in the warehouse directory**
- **This is not always ideal**
  - Data might be part of a bigger workflow
- **Use `LOCATION` to specify the directory where table data resides**

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

Externally Managed Tables

- **CAUTION: Dropping a table removes its data in HDFS**
- **Using `EXTERNAL` when creating the table avoids this behavior**
  - Dropping an *external* (*unmanaged*) table removes only its *metadata*

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

Data Validation

- **Hive and Impala 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`**

Loading Data from HDFS Files

- **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 sales.txt /user/hive/warehouse/sales/
```

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

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

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 '/incoming/etl/sales.txt'
  OVERWRITE INTO TABLE sales;
```

.

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 metastore
  - Imports data from the RDBMS to the table's directory in HDFS

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

.

Removing a Database

- **Removing a database is similar to creating it**

```
DROP DATABASE dualcore;
```

```
DROP DATABASE IF EXISTS dualcore;
```

- **These commands will fail if the database contains tables**
  - Add the **CASCADE** keyword to force removal
    - Supported in all production versions of Hive
    - Supported in Impala 2.3 (CDH 5.5.0) and higher

```
DROP DATABASE dualcore CASCADE;
```

CAUTION:
This command might remove data in HDFS!

Removing a Table

- **Table removal syntax is similar to database removal**

```
DROP TABLE customers;
```

```
DROP TABLE IF EXISTS customers;
```

- **Managed (internal) tables**
  - Metadata is removed
  - Data in HDFS is removed
  - *Caution: No rollback or undo feature!*
- **Unmanaged (external) tables**
  - Metadata is removed
  - Data in HDFS is *not* removed

Creating Views

- **Views are conceptually like a table, but backed by a query**
  - You cannot directly add data to a view

```
CREATE VIEW order_info AS
  SELECT o.order_id, order_date, p.prod_id, brand, name
  FROM orders o
  JOIN order_details d
  ON (o.order_id = d.order_id)
  JOIN products p
  ON (d.prod_id = p.prod_id);
```

- **The query is now greatly simplified**

```
SELECT * FROM order_info WHERE order_id=6584288;
```

Exploring Views

- **`SHOW TABLES` lists the tables *and views* in a database**
  - There is no separate command to list only views

```
SHOW TABLES;
```

- **Use `DESCRIBE FORMATTED` to see a view's underlying query**

```
DESCRIBE FORMATTED order_info;
```

- **Use `SHOW CREATE TABLE` to display a statement to create the view**

```
SHOW CREATE TABLE order_info;
```

Modifying and Removing Views

- Use `ALTER VIEW` to change the underlying query

```
ALTER VIEW order_info AS
   SELECT order_id, order_date FROM orders;
```

- Or to rename a view

```
ALTER VIEW order_info
   RENAME TO order_information;
```

- Use `DROP VIEW` to remove a view

```
DROP VIEW order_info;
```

.

Saving Query Output to a Table

- **`SELECT` statements display their results on screen**
- **To save results to a table, use `INSERT OVERWRITE TABLE`**
  - Destination table must already exist
  - Existing contents will be deleted

```
INSERT OVERWRITE TABLE nyc_customers
  SELECT * FROM customers
  WHERE state = 'NY' AND city = 'New York';
```

- **`INSERT INTO TABLE` adds records without first deleting existing data**

```
INSERT INTO TABLE nyc_customers
  SELECT * FROM customers
  WHERE state = 'NY' AND city = 'Brooklyn';
```

Writing Output to HDFS in Hive

- **Hive also lets you save output to a directory in HDFS**
  - *Caution: Hive does not delete existing contents of the directory!*

```
INSERT OVERWRITE DIRECTORY '/dualcore/ny/'
  ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
    SELECT * FROM customers
    WHERE state = 'NY';
```