NYC DATA SCIENCE ACADEMY

Apache Hive

NYC Data Science Academy

Outline

Why Databases?

- All files are stored in file systems. In Hadoop, they are stored in HDFS. So what does a database add?
- Databases impose *structure* on data.
  - For example, in relational databases, data is in rectangular tables, where each column has a *name* and a *data type*.
- Using this structure, databases provide convenient and efficient methods of accessing the data. For relational databases, this is SQL.

Why Databases?

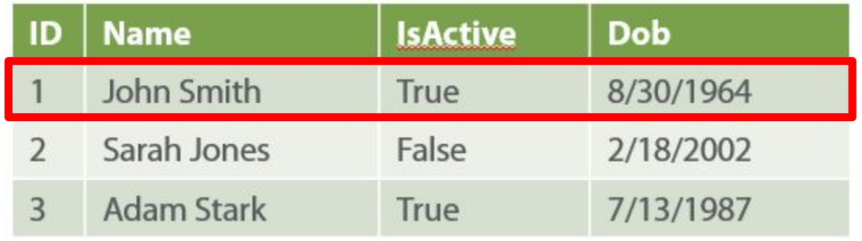
- When updating, databases ensure that the structure is maintained.
  - Data must have the correct type.
  - Transactions - groups of updates - are supported, so that consistency can be maintained when updating multiple rows.
- Databases often provide other services, like encryption and versioning.

Relational Databases (RDBMS's)

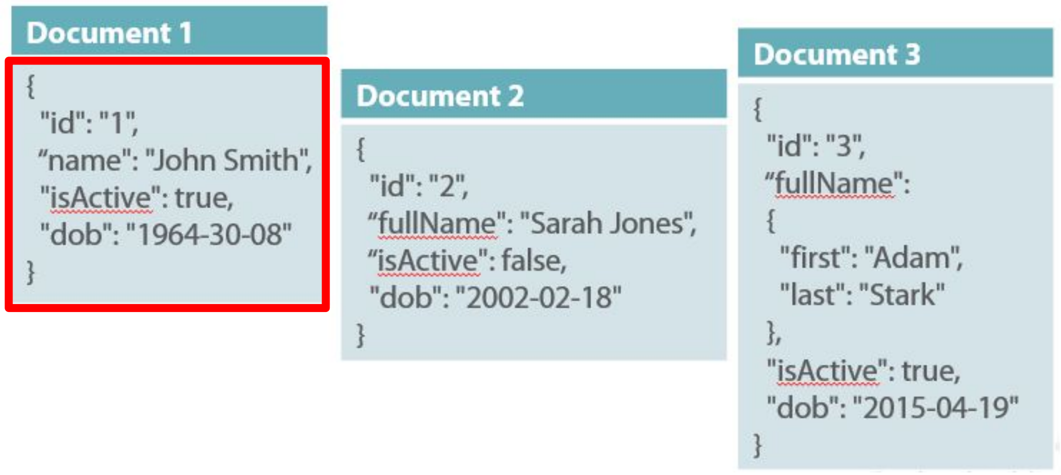
- Data is structured in tables, with records (rows) and fields (columns).
    - Each column of each table has a *name* and a *data type.*
    - Each *cell* - a particular column in a particular row - contains a simple value, like an integer or character string.
- SQL is the usual query language; mostly standardized in the early 90's.
    - Based on three operations: projection, selection, join.
- RDBMS's: MySQL, PostgreSQL, Oracle DB, MS SQL Server, etc.
- Popular SQL databases for Hadoop: **Impala** (from Cloudera), **Hive and Presto** (from Facebook).

NoSQL Databases

- Most often refers to “Non-relational databases”.
- Designed for distributed environments.
- NoSQL saves data in documents (the format is similar to JSON)



ID	Name	IsActive	Dob
1	John Smith	True	8/30/1964
2	Sarah Jones	False	2/18/2002
3	Adam Stark	True	7/13/1987



Document 1
```
{
 "id": "1",
 “name": "John Smith",
 "isActive": true,
 "dob": "1964-30-08"
}
```

Document 2
```
{
 "id": "2",
 “fullName": "Sarah Jones",
 “isActive": false,
 "dob": "2002-02-18"
}
```

Document 3
```
{
 "id": "3",
 “fullName":
 {
  "first": "Adam",
  "last": "Stark"
 },
 "isActive": true,
 "dob": "2015-04-19"
}
```

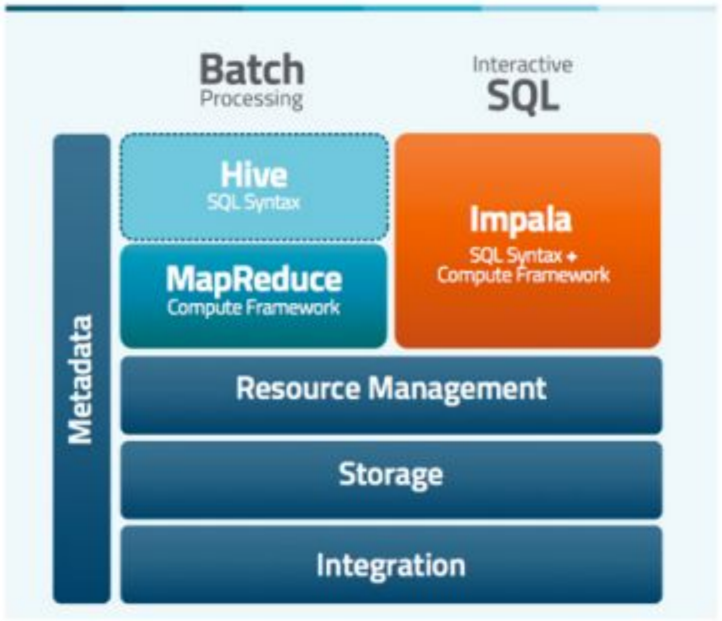
link

Types of NoSQL Databases

- Key-value stores
  - Use keys to access stored values
  - Efficient and scalable
  - Example: HBase, Redis
- Document stores
  - JSON, XML, etc
  - Document can have different structure, no schema is needed
  - Example: MongoDB, DynamoDB(AWS), ElasticSearch DB

Hive

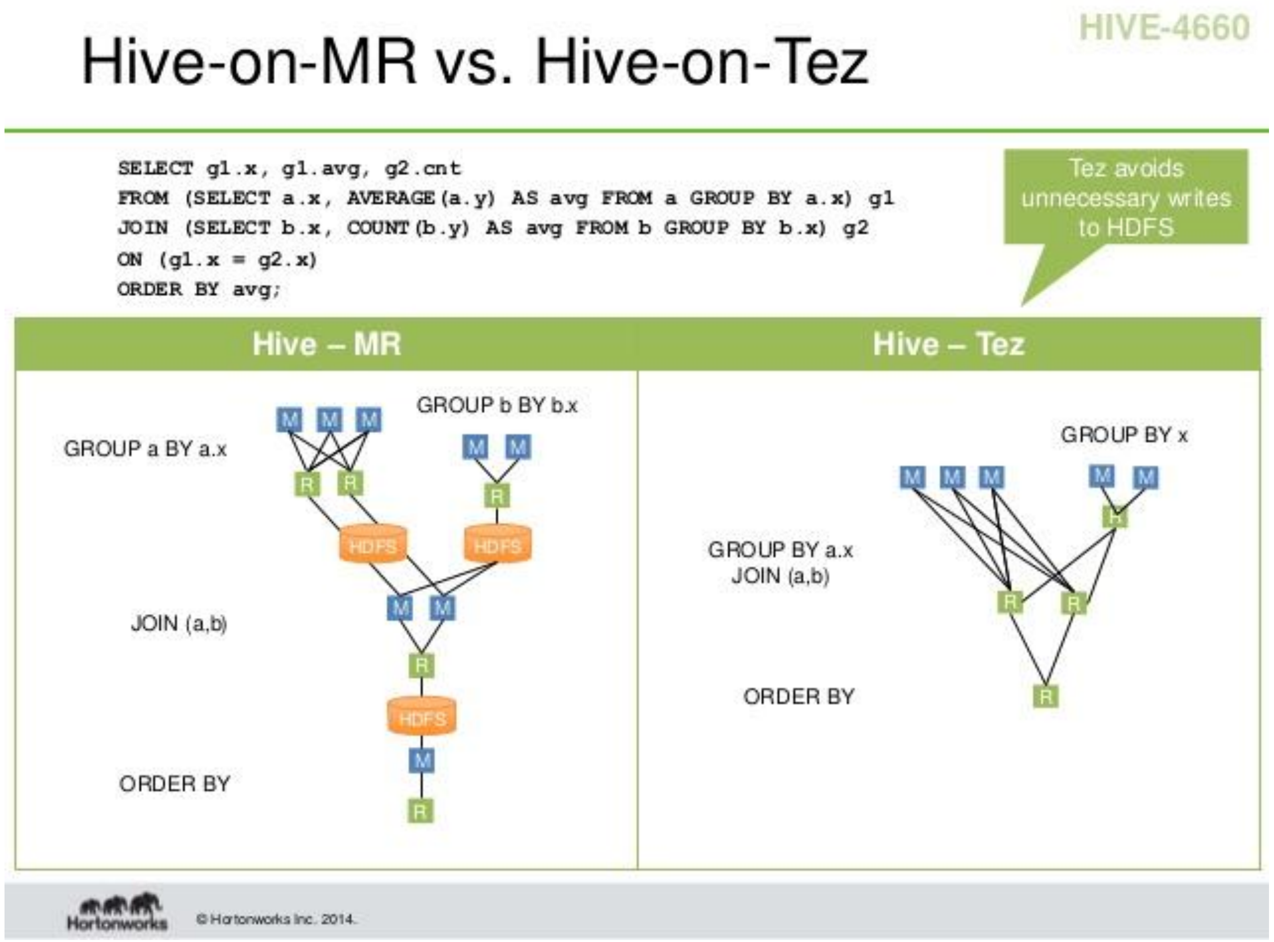
- Hive is a Hadoop-based data warehousing framework developed at Facebook.
- Facebook donated this project to Apache in 2008.



Hive processes data via MapReduce, Impala is a stand-alone MPP framework

link

Hive + Tez


Hive-on-MR vs. Hive-on-Tez

HIVE-4660

```
SELECT g1.x, g1.avg, g2.cnt
FROM (SELECT a.x, AVERAGE(a.y) AS avg FROM a GROUP BY a.x) g1
JOIN (SELECT b.x, COUNT(b.y) AS avg FROM b GROUP BY b.x) g2
ON (g1.x = g2.x)
ORDER BY avg;
```

link

Features Of Hive

- Hive allows users to submit queries in an SQL-like query language called HiveQL.
- HiveQL queries are implemented using MapReduce, so it can handle huge data sets.
- Hive offers:
  - Linear scalability.
  - The ability of real-time query, easy to integrate with business intelligence and visualization tools.
  - Allows SQL programmers with no MapReduce experience to do MapReduce.

Distinctive Features of Hive

- No native format; can handle files with many different formats (e.g. CSV)
  - Populating a new database involves no data movement; Hive just takes control of the file.
  - “schema-on-read”: Data format is not checked until a query is performed.
- Several methods to make queries (esp. joins) more efficient:
  - Partitions
  - Buckets
  - Indexes

Comparison With Traditional Databases

- Traditional DB: *Schema on Write*
  - The data is “ingested” when table is created: checked against the schema and converted to internal format.
- Hive: *Schema on Read*
  - The data is not initially ingested; checked against schema only when a query is processed.
    - Fast initial load: just a file move (i.e. rename)
    - More flexible: may have more than one schemas for the same underlying data

Metastore And Datastore

- Like HDFS, Hive divides databases into metadata and the data itself:
  - Metadata is in the *metastore*, which is normally a MySQL database (i.e. not in HDFS)
  - Data is stored in HDFS files.
    - Different formats are supported, e.g. csv
    - Data files can be *managed* - owned by Hive - or *external* - owned by you.
- Hive-managed files are stored in directory `/user/hive/warehouse/<database>.db/<tablename>/…`

Outline

HiveQL

Hive Data Manipulation Language is a mixture of SQL-92, MySQL, and Oracle’s SQL dialect, usually called HiveQL.

In this section we will discuss some basic data types in Hive and how to use HiveQL to manipulate data in Hive with a focus on SELECT syntax.

Data Types In Hive

Hive supports both primitive and complex data types. A subset of primitive data types supported by Hive is list below:

- **INT/INTEGER** (4-byte signed integer, from -2,147,483,648 to 2,147,483,647)
- **FLOAT** (4-byte single precision floating point number)
- **DOUBLE** (8-byte double precision floating point number)
- **STRING**
- **VARCHAR**
- **BOOLEAN**

For a complete list of supported data types in Hive, please visit: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types

Outline

Creating Tables

When a table is being created, Hive stores table metadata in its metastore, but not in HDFS. Here we first show the CREATE TABLE statement, examples about table creation will be shown later.

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]table_name
[(col_name data_type [COMMENT col_comment], ...)]
  [PARTITIONED BY (col_name data_type, ...)]
  [CLUSTERED BY (col_name, ...) INTO num_buckets BUCKETS]
  [
   [ROW FORMAT row_format]
   [STORED AS file_format]
  ]
  [LOCATION hdfs_path]
  [TBLPROPERTIES (property_name=property_value, ...)]
  [AS select_statement];
```

Loading Data Into Tables

Hive does not do any transformation while loading data into tables. Load operations are currently pure copy/move operations that move datafiles into locations corresponding to Hive tables.

```
CREATE TABLE IF NOT EXISTS my_table (col1 INT, col2 STRING);
LOAD DATA [LOCAL] INPATH 'filepath' INTO TABLE my_table;
```

Inserting Data Into Tables

Query Results can be inserted into tables by using the insert clause.

❖ INSERT OVERWRITE will overwrite any existing data in the table

```
INSERT OVERWRITE TABLE my_table
Select_statement
FROM from_statement;
```

❖ INSERT INTO will append to the table, keeping the existing data intact

```
INSERT INTO TABLE my_table
Select_statement
FROM from_statement;
```

```
INSERT INTO TABLE my_table
VALUES (1,'barney rubble'), (2, 'fred firestone);
```

Update And Delete

UPDATE and DELETE are available starting in Hive 0.14 and can only be performed on tables that support ACID.

- UPDATE

```
UPDATE my_table
SET col2='smith' WHERE col1=1;
```

- DELETE

```
DELETE FROM my_table
WHERE col1=1;
```

Creating Table Using AS SELECT

It’s often very convenient to store the output of a Hive query in a new table with the new table’s columns definitions derived by the SELECT clause. For example:

```
CREATE TABLE
AS
SELECT DISTINCT col2
FROM my_table;
```

Outline

Querying Data

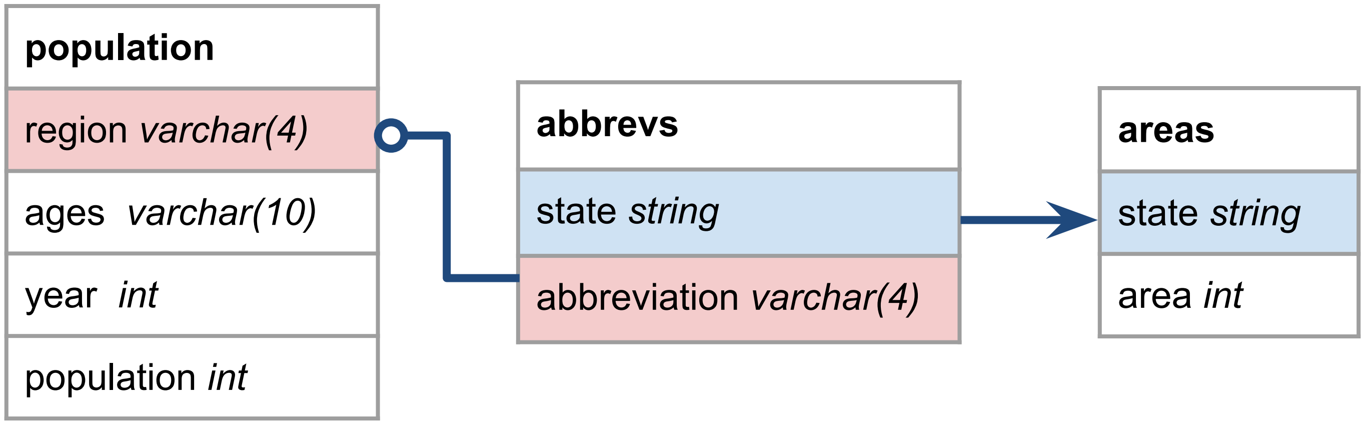
HiveQL **SELECT** syntax is very similar to MySQL **SELECT** syntax. The basic syntax is:

```
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
  FROM table_reference
  [WHERE where_condition]
  [GROUP BY col_list]
  [ORDER BY col_list]
  [CLUSTER BY col_list
    | [DISTRIBUTE BY col_list] [SORT BY col_list]
  ]
 [LIMIT [offset,] rows]
```

For a complete reference, please see:
https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select

Tables In Hive

In the following exercise we will use three tables under database `state` to show how to retrieve data from Hive using SELECT statement. The table schemas are shown below:



Launching Hive Cluster

- Make sure you have pulled the latest version of this repository to your local: https://github.com/casunlight/hadoop-tutorial
- Follow this link to start the hadoop cluster. The docker image that has Hive installed is `hadoop-stack`:
- Move your current working directory into the `examples` directory under `hadoop-tutorial`, and start the cluster with the following command:

```
$ docker run -it \
 -p 8088:8088 \
 -p 19888:19888 \
 -p 50070:50070 \
 --name hadoop-stack \
 --mount type=bind,source="$(pwd)",target=/home/hadoop/examples \
  syan83/hadoop-stack
```

NYC DATA SCIENCE ACADEMY

Creating Hive Tables

- When your hadoop cluster is up, in the terminal window, run the following commands to create database and tables in Hive:

```
$ cd examples/hive/
$ ./run-hive-jobs.sh
```

- When finish, start Hive session by typing `hive` and verify with:

```
$ hive
hive> show databases;
hive> use state;
hive> show tables;
```

- Then run the following commands to see how data is being stored:

```
hive> dfs -ls;
hive> dfs -ls /user/hive/warehouse;
```

Querying Data

In Hive CLI, Make sure you have selected `state` database, then answer the following questions with Hive queries:

1. What is the abbreviation of state New York?
2. What is the area of state New York?
3. What is the population under 18 of USA? Sort the result by year.
4. What is the average population per state for age under 18 in 2013?
5. Which state has the largest area? What about the smallest state? Return state full name and area size.
6. What is the area of state CA?
7. (Optional) What's the (total) population density per sq mile for each state in 2013? Round your result to 2 decimal places.

Querying Data Solutions

```
USE state;

-- 1.
SELECT *
FROM abbrevs
WHERE state="New York";

-- 2.
SELECT *
FROM areas
WHERE state="New York";

-- 3.
SELECT *
FROM population
WHERE region="USA" AND ages="under18"
ORDER BY year;

-- 4.
SELECT ROUND(AVG(population), 2)
FROM population
WHERE region<>"USA" AND year=2013 AND ages="under18";
```

Querying Data Solutions

```
-- 5.
SELECT state, area
FROM areas
WHERE area IN (
    SELECT MIN(area) FROM areas
    UNION
    SELECT MAX(area) FROM areas
);

-- 6.
SELECT *
FROM areas
JOIN abbrevs
ON areas.state=abbrevs.state
WHERE abbrevs.abbreviation="CA";
```

Querying Data Solutions

```
-- 7.
SELECT a.state, a.area, p.population, ROUND(p.population/a.area, 2)
FROM (
    SELECT region state, population
    FROM population
    WHERE year=2013 AND ages="total"
) p
JOIN (
    SELECT abbreviation state, area
    FROM areas
    JOIN abbrevs
    ON areas.state=abbrevs.state
) a
ON p.state=a.state;
```

Outline

Tables in Hive

- A Hive table is made up of the **data** being stored and the associated **metadata** describing the layout of the data in the table.
- The data is typically stored in HDFS, Hive also supports loading data from other filesystems, such as local filesystem and AWS S3.
- The metadata is stored in a relational database, which is usually a Mysql, PostgreSQL or Oracle server.

Managed Tables and External Tables

- **Managed Tables**:
  - when you create a table in Hive, by default Hive will manage the data by moving the data into its warehouse directory.
  - If the table is later dropped, the table, including both data and metadata, are deleted

```
-- to create a managed table
CREATE TABLE areas_managed (
  state STRING,
  area INT
);
LOAD DATA INPATH 'path-to-data' OVERWRITE INTO TABLE
areas_managed;

-- to drop the table
DROP TABLE areas_managed;
```

Managed Tables and External Tables

- **External Tables**:
  - Hive refer to the data at an existing location outside its warehouse directory. The table location is specified with LOCATION keyword.
  - The user control the creation and deletion of the data
  - When the external data is dropped, Hive will leave the data untouched and only delete the metadata.

```
-- to create a external table
CREATE EXTERNAL TABLE areas_external (
  state STRING,
  area INT
)
LOCATION '/user/hadoop/areas_external';

LOAD DATA INPATH 'path-to-data' OVERWRITE INTO TABLE
areas_external;
```

Managed Tables and External Tables

1. Make two copies of data `state-areas.csv` from `examples/data` directory into HDFS home directory, name them `sareas-m.csv` and `areas-e.csv`, respectively.

```
hive> dfs -put /home/hadoop/examples/data/state-areas.csv areas-m.csv;
hive> dfs -put /home/hadoop/examples/data/state-areas.csv areas-e.csv;
hive> dfs -ls;
```

2. Create a managed table with:

```
USE default;
CREATE TABLE areas_managed (
  state STRING,
  area INT
)ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
TBLPROPERTIES ("skip.header.line.count"="1");

LOAD DATA INPATH 'areas-m.csv' OVERWRITE INTO TABLE
areas_managed;
```

Managed Tables and External Tables

3. Check the HDFS directories with:

```
hive> dfs -ls;
hive> dfs -ls /user/hive/warehouse;
hive> dfs -ls /user/hive/warehouse/areas_managed;
```

4. Drop table `areas_managed` and then check HDFS.

```
DROP TABLE areas_managed;
```

```
hive> dfs -ls;
hive> dfs -ls /user/hive/warehouse;
```

Managed Tables and External Tables

6. Create an external table with:

```
CREATE EXTERNAL TABLE areas_external (
  state STRING,
  area INT
)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ','
LOCATION '/user/hadoop/areas_external'
TBLPROPERTIES ("skip.header.line.count"="1");

LOAD DATA INPATH 'areas-e.csv' OVERWRITE INTO TABLE
areas_external;
```

7. Check the HDFS directories:

```
hive> dfs -ls;
hive> dfs -ls areas_external;
hive> dfs -ls /user/hive/warehouse;
```

8. Drop table `areas_external`in Hive and check HDFS again.

Managed Tables and External Tables

- How to choose?
  - In most cases, there is not much difference between the tow.
  - Use **managed** table if you are doing all your processing with Hive.
  - Use **external** table if you also use other tools on the same dataset or if you need to associate multiple schemas with the same dataset.

Outline

Partitions And Buckets

- Hive organized tables into **partitions** by dividing a table into parts based on the value of a partition column, such as date.
- Using partitions can make queries running faster on slices of the data.
- Tables or partitions may be divided further into **buckets** to give extra structure to the data.

Partitions

- Advantage:
  - Imagine we have a data stream being appended to one of our Hive tables on a daily basis.
  - If we partition the table by date, then records for the same date will be stored in the same partition so queries that are restricted to a particular date or a set of dates can run more efficiently, because only the files in the partitions of those dates will be scanned.
- Partitions are defined at table creation time using PARTITIONED BY clause.
- When we load data into a partitioned table, the partition values are specified explicitly.

Partitions

- At the filesystem level, partitions are stored as nexted subdirectories of the table directory.
- Assuming you partitioned a table on column `partition_col`, then the HDFS directory structure might look like:

```
/user/hive/warehouse/partitioned_table
  |-- partition_col=val1/
  |      |-- file0
  |      `-- file1
  |-- partition_col=val2/
  |      `-- file2
  |-- partition_col=val3/
  |      |-- file3
  |      |-- file4
  |      `-- file5
  `-- partition_col=val2/
         |-- file6
         `-- file7
```

Partitions

1. Copy data directory `examples/data/population_partitions` into HDFS home directory.

```
hive> dfs -put /home/hadoop/examples/data/population_partitions
population_partitions;
```

2. Create a partitioned table `population_partitions`:

```
CREATE TABLE population_partitions (
  region VARCHAR(4),
  ages VARCHAR(10),
  population INT)
PARTITIONED BY (year INT)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY ',';
```

Partitions

3. Run the following command to insert each file into the corresponding partition of the table (change year between 2004 and 2013and run it again to add different partitions):

```
LOAD DATA INPATH 'population_partitions/2013'
INTO TABLE population_partitions
PARTITION (year=2013);
```

4. Run some queries again the table filtered by year.

5. Check the HDFS directories:

```
hive> dfs -ls /user/hive/warehouse/population_partitions;
```

6. Note: In case you don't want to do it manually, change your working directory to `examples/hive` and execute `./run-hive-partitions.sh`, which will automatically create the table and add data to it.

Buckets

- Bucketing imposes extra structure on the table, which enables more efficient queries.
- When we create a bucketed table, we specify the column to bucket on and the number of buckets. Hive then use the bucketed column to determine the bucket by hashing the value and reducing the module the number of buckets
- When working with large dataset, it's very convenient to try out queries on a fraction of dataset, we will see how to do efficient sampling using buckets.

Buckets

- At the filesystem level, each bucket is just a file in the table directory
- A bucketed table with number of buckets set to 6 might look like:

```
/user/hive/warehouse/bucketed_table
    |-- 000000_0
    |-- 000000_1
    |-- 000000_2
    |-- 000000_3
    |-- 000000_4
    `-- 000000_5
```

Buckets

1. In your Hive session, create a bucketed table with:

```
CREATE TABLE bucketed_population (
  region VARCHAR(4),
  ages VARCHAR(10),
  year INT,
  population INT)
CLUSTERED BY (region) INTO 8 BUCKETS;
```

2. To populate the bucketed table, we need to set the Hive property to true:

```
hive> SET hive.enforce.bucketing=true;
```

3. Populate data into table using `INSERT ... AS SELECT`

```
INSERT OVERWRITE TABLE bucketed_population
SELECT * FROM state.population
WHERE region<>"USA";
```

Buckets

4. We can see how data in the bucketed table is being stored in HDFS with command:

```
hive> dfs -ls /user/hive/warehouse/bucketed_population;
```

5. Now lets run some queries against the bucketed table:

```
-- unique regions in bucket 1
SELECT DISTINCT(region)
FROM bucketed_population
TABLESAMPLE(BUCKET 1 OUT OF 8 ON region);

-- average population using 2 buckets
SELECT ROUND(AVG(population), 2)
FROM bucketed_population
TABLESAMPLE(BUCKET 1 OUT OF 4 ON region)
WHERE year=2013 AND ages="total";

-- average population with all buckets
SELECT ROUND(AVG(population), 2)
FROM bucketed_population
WHERE year=2013 AND ages="total";
```