Build a data library with Hive

Peter J. Jamack

May 21, 2013

Storing massive amounts of data is great until you need to do something with it. No incredible discoveries or futuristic predictions come from unused data, no matter how much of it you store. Big data can be a complicated beast. Writing complex MapReduce programs in the Java™ programming language takes time, good resources, and know-how that most organizations don't have available. This is where building a data library using a tool like Hive on top of Hadoop becomes a powerful solution.

If a company does not have the resources to build a complex big data analytics platform, what can it do? And what happens when business intelligence (BI), data warehousing, and analytics tools cannot connect to the Apache Hadoop system or become more complicated than need be? Most organizations have employees who have experience with relational database management systems (RDBMSes) and Structured Query Language (SQL). Apache Hive allows these database developers or data analysts to use Hadoop without knowing the Java programming language or MapReduce. Now, instead of challenging MapReduce code, you can design a star schema data warehouse or a normalized database. Suddenly, BI and analytic tools like IBM Cognos® or SPSS® Statistics can connect to the Hadoop ecosystem.

Data libraries

Building a data library and being able to use that data are not Hadoop problems or database problems. People have been organizing data into libraries for years. These are age-old problems: How do you organize and classify your data into categories? How do you connect all this data into an integrated platform or cabinet or library? Different philosophies have come about over the years.

People invented methods like the Dewey Decimal System. They alphabetized phone books by classifying a person or business by name. There have been metal filing cabinets, warehouses with shelves, address card file systems, and more. Employers tried to keep track of employees with time cards, punch cards, and time sheets. People have a need to structure and organize things. And they have a need to reflect and check that data. Is there any real purpose to storing vast amounts of data if you have no way of accessing, structuring, or understanding it?

RDBMSes used set theory and third normal form. Data warehousing has Kimball, Inmon, star schemas, Corporate Information Factory, and data marts. They have master data management,
enterprise resource planning, customer relationship management, electronic medical records, and the thousands of other systems people have come up with to organize things into a certain structure and theme. Now, we have massive amounts of unstructured or semi-structured data from sources like social media, email, phone calls, machines, telematics, and more. And this new data has to be integrated with the already complicated and massive systems that hold structured new and old legacy data. How do you categorize it all so your sales manager can pull up a report? How do you build a library so that an executive can access charts and graphs?

You need to find a way to structure that data into a data library. Otherwise, you have a bunch of data that only a few data scientists can access. Sometimes, people just need simple reports. Sometimes, they just want to drag and drop or write an SQL query.

**Big data, Hadoop, and InfoSphere BigInsights**

This section introduces you to InfoSphere® BigInsights™ and how it relates to Hadoop, big data, Hive, data libraries, and more. InfoSphere BigInsights is the IBM distribution of Hadoop. You know about Apache and Cloudera, but many in the industry are putting their own marks on Hadoop. It starts with open source Hadoop with MapReduce and the Hadoop Distributed File System (HDFS) and usually includes other pieces, like ZooKeeper, Oozie, Sqoop, Hive, Pig, and HBase. What differentiates these distributions from vanilla Hadoop is what they add on top of Hadoop. InfoSphere BigInsights falls into this category.

You can use InfoSphere BigInsights on top of Cloudera's distribution of Hadoop. In addition, InfoSphere BigInsights provides a fast, unstructured analytics engine you can combine with InfoSphere Streams. InfoSphere Streams is a real-time engine for analytics that opens the possibilities of combining real-time and batch-oriented analytics.

InfoSphere BigInsights also comes with a built-in browser-based spreadsheet called BigSheets. This spreadsheet allows everyday analysts to use big data and Hadoop in a spreadsheet-style fashion. Other features include LDAP integration for role-based security and administration; integration with InfoSphere DataStage® for extract, transform, load (ETL); accelerators for common use cases like log and machine data analytics; an application catalog with common catalog and reusable jobs; Eclipse plug-ins; and BigIndex, which is essentially a Lucene-based index tool built on top of Hadoop.

You can gain performance improvements by using Adaptive MapReduce, compressed text files, and adaptive scheduling enhancements. Plus, you can integrate with other applications like content analytics and Cognos Consumer Insights.

**Hive**

Hive is a powerful tool. It uses HDFS, a metastore (by default, an Apache Derby database), shell commands, drivers, a compiler, and an execution engine. It also supports Java database connectivity (JDBC) connections. Hive opens the big data Hadoop ecosystem to nonprogrammers because of its SQL-like capabilities and database-like functionalities. It enables external BI software like Cognos to connect to it because of the JDBC driver and web clients.
Instead of searching long and far for Java MapReduce programmers, you can rely on existing database developers. The beauty is that instead of a nonprogrammer or a programmer forced to write a complicated MapReduce program consisting of 200 or more lines of code, you have a database developer writing 10-15 lines of SQL code optimized and translated into MapReduce code.

Hive is often described as a data warehouse infrastructure built on top of Hadoop. The truth is that Hive is nowhere near a data warehouse. If you want to build a real data warehouse, use something like IBM Netezza. But if you want to build a data library using Hadoop, but have no Java or MapReduce knowledge, Hive can be a great alternative (if you know SQL). It gives you the ability to write SQL-like queries using HiveQL for Hadoop and HBase. It also allows you to build star schemas on top of HDFS.

### Hive vs. RDBMSes

Hive is a schema on Read system, whereas RDBMSes are typically schema on Write systems. Traditional RDBMSes verify the schema when writing data. If the data does not conform to the structure, it is rejected. Hive does not really care about the structure of the database — at least not at first — and it does not verify the schema when you are loading data into it. Rather, it cares about the schema only after you run a query on it.

### Limitations of Hive

A few challenges arise when it comes to using Hive. First, it is not SQL-92 compliant. Certain standard SQL functions, such as NOT IN, NOT LIKE, and NOT EQUAL, do not exist or require certain workarounds. Similarly, some math functions are severely limited or do not exist. Timestamp, or date, is a recent addition more Java than SQL date-compliant. Something simple like date difference does not work.

In addition, Hive is not made for low-latency, real-time, or near-real-time querying. SQL queries translate to MapReduce, which means slower performance for certain queries compared to traditional RDBMS.

Another limitation is that the metastore defaults to a Derby database and is not enterprise- or production-ready. Some users of Hadoop wind up using an external database for the metastore, but those external metadata metastores have their own challenges and configuration issues. It also means someone has to maintain and administer an RDBMS system outside of Hadoop.

### Installing InfoSphere BigInsights

This baseball data example shows you how to build a common data library from flat files in Hive. Although the sample is rather small, it shows how easy it is to use Hive to build a data library, and with this data, you can run statistics to make sure it matches up with what it's supposed to look like. In the future, when you're trying to structure unstructured data, you won't have that information to check against.

After the database has been built, you can build a web or GUI front end in any language as long as you can connect to the Hive JDBC. (Configuring and setting up a thrift server and Hive JDBC
is a conversation for another day.) I created an InfoSphere BigInsights virtual machine (VM) using VMware Fusion on my Apple Macbook. This was a simple test, so my VM had 1 GB of RAM and 20 GB of solid-state disk storage space. The operating system was a CentOS 6.4 64-bit distro of Linux®. You could use something like Oracle VM VirtualBox or, if you’re a Windows® user, you can use VMware Player to create the InfoSphere BigInsights VM. (Setting up the VM on Fusion, VMware Player, or VirtualBox is beyond the scope of this article.)

Begin by downloading IBM InfoSphere BigInsights Basic Edition (see Related topics). You will need to have an IBM ID or register for one before you can download InfoSphere BigInsights Basic.

Import and analyze data

You can get data anywhere these days. Millions of sites offer data in comma-separated values (CSV) format — weather, energy, sports, finances, blogs. For this example, I used structured data from Sean Lahman’s website. Unstructured data takes a little more work.

Begin by downloading the CSV file (see Figure 1).

Figure 1. Download the sample database

If you would rather do it from Linux® in a more manual fashion, create a directory, then run `wget`:

```
$ Sudo mkdir /user/baseball.
```

The data uses a Creative Commons Attribution-ShareAlike 3.0 Unported license.

The zipped file contains statistics about baseball and baseball players in CSV files. The example contains four main tables, each with a unique column (Player_ID):

- **Master table.csv** — Player names, dates of birth, biographical information
- **Batting.csv** — Batting statistics
- **Pitching.csv** — Pitching statistics
- **Fielding.csv** — Fielding statistics
The secondary tables are:

- **AllStarFull.csv** — All-star appearances
- **Hall of Fame.csv** — Hall of Fame voting data
- **Managers.csv** — Managerial statistics
- **Teams.csv** — Yearly statistics and standings
- **BattingPost.csv** — Post-season batting statistics
- **PitchingPost.csv** — Post-season pitching statistics
- **TeamFranchises.csv** — Franchise information
- **FieldingOF.csv** — Outfield position data
- **FieldingPost.csv** — Post-season infield data
- **ManagersHalf.csv** — Split-season data for managers
- **TeamsHalf.csv** — Split-season data for teams
- **Salaries.csv** — Player salary data
- **SeriesPost.csv** — Post-season series information
- **AwardsManagers.csv** — Awards won by managers
- **AwardsPlayers.csv** — Awards won by players
- **AwardsShareManagers.csv** — Award voting for manager awards
- **AwardsSharePlayers.csv** — Award voting for player awards
- **Appearances.csv**
- **Schools.csv**
- **SchoolsPlayers.csv**

**Design the data library**

Much of the work of designing the data library has already been done. Player_ID is the primary key for the four main tables — Master, Batting, Pitching, and Fielding. (For a better understanding of the table structure and dependencies, read Readme2012.txt.)

The design is simple: The main tables are connected through the Player_ID. Hive doesn't really employ the concept of primary keys or referential integrity. *Schema on Read* means that Hive dumps whatever you give it into the tables. If the files were a bit discombobulated and out of sorts, you might have to figure out the best way to connect them. Or you would do some sort of transformation before loading the data into HDFS or Hive. Bad data becomes really bad data in Hive because of the Schema on Read philosophy. That is why data analysis — whether at the source level or at the HDFS level — is an important step. Without it, you wind up with raw data no one can use. Luckily, this baseball example has data that has been cleaned and organized before you dump it into Hadoop.

**Load data into HDFS or Hive**

Different theories and practices are used to load data into Hadoop. Sometimes, you ingest raw files directly into HDFS. You might create certain directories and subdirectories to organize the files, but it's a simple process of copying or moving files from one location to another.

For this example, issue a `put` command, then create a directory called **baseball**:
Hdfs dfs -mkdir /user/hadoop/baseball
hdfs dfs -put /LOCALFILE /user/hadoop/baseball

Build the data library with Hive

With the data analysis and design complete, it's time to build the database.

I don't have space here to show all the examples, but if you follow how the first one is done, you can figure out how the rest should be done. I usually build some SQL text scripts, then import or paste them into Hive. Others might use Hue or another tool to build out the databases and tables.

To keep it simple, let's use the Hive shell. The high-level steps are:

1. Create the baseball database
2. Create the tables
3. Load the tables
4. Verify that the tables are correct

You will see options like creating external and internal databases and tables, but for this example, stick to the default setting of Internal. In essence, internal means that Hive handles where the database is stored and now. Listing 1 shows the process in the Hive shell.

**Listing 1. Create the database**

```sql
$ Hive
Create Database baseball;
Create table baseball.Master
    ( lahmanID int, playerId int, managerID int, hofID int, birthyear INT,
    birthMonth INT, birthDay INT, birthCountry STRING, birthState STRING,
    birthCity STRING, deathYear INT, deathMonth INT, deathDay INT,
    deathCountry STRING, deathState STRING, deathCity STRING,
    nameFirst STRING, nameLast STRING, nameNote STRING, nameGive STRING,
    nameNick STRING, weight decimal, height decimal, bats STRING,
    throws STRING, debut INT, finalGame INT,
    college STRING, lahman40ID INT, lahman45ID INT, retroID INT,
    holtzID INT, hbrefID INT )
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' ;
```

Follow this procedure for all the other tables. To load data into the Hive table, open the Hive shell again, then run the following code:

```sql
$hive
LOAD DATA LOCAL INPATH Master.csv OVERWRITE INTO TABLE baseball.Master;
```

Build a normalized database with Hive

The baseball database is more or less normalized: You have the four main tables and several secondary tables. Again, Hive is a Schema on Read, so you have to do most of the work in the data analysis and ETL stages because there is no indexing or referential integrity such as in traditional RDBMSes. If you want the ability to index, using something like HBase is the next step. See the code in Listing 2.
Listing 2. Run a query

```
$ HIVE
Use baseball;
Select * from Master;
Select PlayerID from Master;
Select A.PlayerID, B.teamID, B.AB, B.R, B.H, B.2B, B.3B, B.HR, B.RBI
   FROM Master A JOIN BATTING B ON A.playerID = B.playerID;
```

Conclusion

That is the beauty of Hive and the benefit of building data libraries: They create structure in a chaotic world. As much as we love to talk about unstructured or semi-structured data, it all comes down to who can analyze it, who can run reports based on it, and how fast you can get it to work. Most users look at it as some sort of black box: They don't care where the data came from, and they really don't care what you had to do to get it to them in the correct format. Nor do they care how difficult it was to integrate or verify as long as it is accurate. This usually means that you must have organization and structure. Otherwise, your library becomes a dead zone of unlimited data kept for eternity that nobody can or wants to use.

Data warehouses with complicated structures have become ghost towns. Things have gotten better over the years, but the concepts remain the same: It's business, and business users want results, not programming logic. That's why building data libraries in Hive can be a start in the right direction.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample CSV file</td>
<td>lahman2012-csv.zip</td>
<td>11MB</td>
</tr>
</tbody>
</table>
Related topics

- Read *Big data: The next frontier for innovation, competition, and productivity*, from McKinsey & Company.
- Learn more about The Data Warehouse Institute (TDWI), a valuable resource for BI and data warehousing.
- Explore free courses from Big Data University on topics ranging from Hadoop fundamentals and text analytics essentials to SQL access for Hadoop and real-time stream computing.
- Visit Hadoop.org for information about all things Hadoop.
- Visit the Hive project site for more information about Apache Hive.
- Visit the Sqoop project site for more information about Apache's bulk data transfer tool.
- Visit the HBase project site for more information about Apache's distributed data store.
- Learn more about IBM Netezza.
- Learn more about InfoSphere BigInsights.
- Download InfoSphere BigInsights Basic Edition for Linux 2.0.0.0.
- Learn more about SPSS Statistics.
- Learn more about Cognos.
- Get a trial version of IBM InfoSphere Streams.

© Copyright IBM Corporation 2013
Trademark
(www.ibm.com/developerworks/ibm/trademarks/)