The data warehouse has been an ongoing battle among organizations for years. How do you build it? What data can you integrate? Should you use Kimball or Inmon, corporate information factory (CIF), or data marts? The list could go on for days -- decades, even. With big data, the questions become far more complicated, such as is a data warehouse enough? The answer lies in the enterprise. People claim that Hive is the data warehouse of Hadoop. Although true on one level, it's also something of a false claim. Sometimes, however, you have to use the tools available to you, and for that, Hive can be a data warehouse.

Three objects approach an enterprise. The first — the data warehouse — is massive: It brings history and experience, and it talks a good game. And most of it is true. But it's also bloated in many ways, expensive in a lot of other ways, and people are tired of the cost for varied results. Apache Hadoop walks into the same building and throws out claims of taking over the market. It preaches big data, velocity, volume, variety, and a bunch of other words that don't mean much outside of a marketing plan. It throws out analytics, predictions, and much more. And it's cheap. People stop and listen.

Apache Hive steps outside of the box, but does not attempt to beat the other objects. It wants to work with Hadoop, but unlike Hadoop, it doesn't want to throw the data warehouse to the curb. Hive has data warehouse capabilities, but with business intelligence (BI) and analytic limitations. It has database possibilities, but relational database management system (RDBMS) and Structured Query Language (SQL) limitations. It is more open and honest. It relates to the data warehouse. It relates to the RDBMS. But it never comes out and claims that it's more than meets the eye. Hadoop interrupts and proclaims it is the data warehouse for the Hadoop world. Hadoop seems to have sent its best marketing public relations rep, and what went from a simple conversation turned into Hive and Hadoop saving the world. It's intriguing. It's interesting. But is it really true? Sort of.

Data warehouses

Building a true data warehouse can be a massive project. There are different appliances, methodologies, and theories. What is the lowest common value? What are the facts, and what subjects relate back to those facts? And how do you mix, match, merge, and integrate systems
that might have been around for decades with systems that only came to fruition a few months ago? This was before big data and Hadoop. Add unstructured, data, NoSQL, and Hadoop to the mix, and suddenly you have a massive data-integration project on your hands.

The simplest way to describe a data warehouse is to realize that it comes down to star schemas, facts, and dimensions. How you go about creating those elements is really up to you — whether it's through staging databases; on-the-fly extract, transform, load processes; or integrating secondary indices. Certainly, you can build a data warehouse with star schemas, facts, and dimensions, using Hive as the core technology, but it won't be easy. Outside the Hadoop world, it becomes an even bigger challenge. Hive is far more an integration, transformation, quick lookup tool compared to a legitimate data warehouse. The schema might say data warehouse, but the usefulness doesn't even say RDBMS. So, why use it?

**What is a star schema**

Imagine a star — a center and a several "hands" that point in different directions. The center is the heartbeat, or fact table. The hands all point to different dimensions. Many data warehouses have one fact table and multiple dimensions.

A fact table contains any data you can weight or calculate on. In this example, you have baseball statistics like runs, home runs, batting average, etc. You can calculate, add to, subtract from, or multiply against these columns.

Dimensions are more subject-based. In this example, you have the player information dimension, time and date dimension, etc. You typically do not have calculated or weighted columns in dimensions.

In this example, the key that joins a dimensional table to a fact table is the playerID.

Simply put, sometimes, you have to use the tools put before you.

Anyone who has been in IT for any amount of time can tell you that the right tool for a job isn't always available. Or, the right tool is available, but cost-cutting factors are in play. Sometimes corporate politics play a huge role. Whatever the reason, most of us have been in situations where we are forced to build, design, and develop using a tool that might not be the best fit for the job.

I've been on numerous projects where we had to use Hive as a database, as a data warehouse, and as a slowly changing system. It was challenging; it was occasionally annoying. Sometimes, you had to just shake your head and wonder why. But at the end of the day, you still needed to make it work. And if a data warehouse had to be built and used in Hive, and you needed slowly changing dimensions and updates as well as reconciliation of old data, that's what needed to be done. It is not always about the best tool but about making the tool you have work the best.

**Hive**

Hive opens the big data Hadoop ecosystem to nonprogrammers because of its SQL-like capabilities and database-like functionality. It is often described as a data warehouse infrastructure built on top of Hadoop. This is a partially true statement — since you can transform source data into a star schema — but it's more about design than technology when you create a fact table and dimension tables.
Still, Hive is not really a data warehouse. It's not really even a database. You can build and design a data warehouse with Hive, and you can build and design database tables with Hive, but certain limitations exist that require many workarounds and will pose challenges.

For example, indexing is limited in Hive. How do you overcome this issue? You can create an index in Hive by using the `org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler` function. Hive and slowly changing dimensions aren't exactly possible, either. But if you build staging tables and use a certain amount of joins (and you plan to add a new table, dumping the old one and keeping only the most recent, updated table for comparison), it is a possibility.

External reporting or analytic systems connecting to Hive have been a huge problem. Even with JDBC connections, you are limited to connecting to the default database only. There has been a push for more and improved metadata, and tools like Apache HCatalog are helping to connect various services to the Hive metastore. In the future, this could be a great addition if utilized properly.

So, although Hive is not a hardcore data warehouse or database, ways exist in which you can use Hive to be that data warehouse or database. It just takes effort and several workarounds to make Hive that system. Why would you go through all that again? Because you have to use the tool you have available to you and make it work.

**Example: Build a data warehouse for baseball information**

InfoSphere BigInsights Quick Start Edition

InfoSphere BigInsights Quick Start Edition is a complimentary, downloadable version of InfoSphere BigInsights, IBM's Hadoop-based offering. Using Quick Start Edition, you can try out the features that IBM has built to extend the value of open source Hadoop, like Big SQL, text analytics, and BigSheets. Guided learning is available to make your experience as smooth as possible including step-by-step self-paced tutorials and videos to help you start putting Hadoop to work for you. With no time or data limit, you can experiment on your own time with large amounts of data. Watch the videos, and download BigInsights Quick Start Edition.

The following baseball data example shows how to design and build a data warehouse in Hive using baseball data from Sean Lahman's website. I liked the challenge of denormalizing and building a data warehouse from that data. In "Build a data library with Hive," I created an IBM 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®.

To get started with this example, download the IBM InfoSphere BigInsights Basic Edition (see Related topics). You will need to have an IBM Universal ID or register to get an ID before you can download InfoSphere BigInsights Basic Edition.

**Import the data**

Begin by downloading the CSV file that contains statistics about baseball and baseball players (see Download). From within Linux, create a directory, then run:
$ Sudo mkdir /user/baseball.


The example contains four main tables, each with a unique column — Master table, Batting, Pitching, and Fielding — and several secondary tables.

Design the data warehouse

This data is structured for a data library, but for a data warehouse, you have to figure out the facts and dimensions. The data warehouse design is simple: You denormalize this database and create a fact table based on player statistics. Then you create dimensions based off of certain subject areas related to those statistics. Hive is not terribly good when it comes to joins, and MapReduce isn’t much better, so having a denormalized star schema will help with certain queries.

The design consists of a fact table called fact_Player_Stats, which includes every statistical column found in the various CSV files and tables. You need data from the core tables (Batting, Pitching, and Fielding), as well as from some of the supplemental tables, which also contain statistics. Therefore, you must add the statistical columns from the following tables:

- AllStarFull
- hall of Fame
- BattingPost
- PitchingPost
- FieldingOF
- Salaries
- AwardsPlayers
- AwardsSharePlayers
- Appearances
- SchoolsPlayers

Some of the tables have only a few statistical columns. For example, from the FieldingOF table, you need add only the columns stint, Glf, Gcf, and Grf to the fact_Player_Stats fact table. For the SchoolsPlayers table, take only the yearMin and yearMax columns. Take similar steps with the other tables. Only statistical columns are needed in the fact table.

Note: You will not be using any data from the tables Managers, Teams, TeamsHalf, SeriesPost, etc.

The fact_Player_Stats fact table consists only of the keys playerID, FranchID, yearID, and SchoolID. For the dimension tables, you must take out the statistics (if any exist) and keep only the subject-related columns:

- The dim_Players dimensional table takes the data (player names, date of birth, biographical information, etc.) from the Master table. The primary key is playerID.
- The dim_TeamFranchise dimensional table takes all the data from the TeamFranchise table. The primary key is FranchID.
• The dim_Schools dimensional table takes all data from the Schools table.
• The dim_Year is a time dimensional table based on months and years (1871-2012).

Use the data library for the data warehouse

If you haven't already created the baseball data library, I recommend doing so now, then deriving the data warehouse from those base tables. You could write complicated scripts to grab certain columns from a flat file, and reuse that same flat file for another table, but for this article, I choose to use the data library created previously in "Build a data library with Hive."

Build the data warehouse with Hive

With the data analysis and design complete, it's time to build the data warehouse based off of your star schema design. In the Hive shell, create the baseball_stats database, create the tables, load the tables, and verify that the tables are correct. (This process is provided in "Build a data library with Hive."). Next, create the data warehouse fact table. Listing 1 shows the code.

Listing 1. Create the data warehouse fact table

```hql
$ Hive
Create Database baseball_stats;

Create table baseball_stats.fact_player_stats as

( SELECT a.playerID, FranchID, yearID, SchoolID, stint int, g int,
g_batting int, ab int, r int, h int, 2b int, 3b int, hr int, rbi int, sb int,
cs int, bb int, so int, ibb int, hbp int, sh int, sf int, gdp int, w int,
l int, g int, gs int, cg int, sho int, sv int, ipouts int, ph int, er int,
hr int, pbb int, ps0 int, baoppt int, era int, pibb int, wp int, phbp int,
bk int, bfp int, gf int, pr int, p sh int, psf int, p gdp int, fg int,
fhs int, innouts int, po int, a int, e int, dp int, pb int, wp int, fsb int,
fcf int, zr int, gamenum int, allstargp int, ballots int, needed int, votes int,
playoff_g int, playoff_ab int, playoff_r int, playoff_h int,
playoff_2b int, playoff_3b int, playoff_hr int, playoff_rbi int, playoff_sb int,
playoff_cs int, playoff_bb int, playoff_so int, playoff_ibt int, playoff_hbp int,
playoff_sh int, playoff_sf int, playoff_gdp int, playoff_w int,
playoff_p int, playoff_g int, playoff_pp int, playoff_gp int, playoff_w int,
playoff_gf int, playoff_rg int, playoffpch int, playoff_SF int,
playoff_gdp int, glf int, grf int, gcf int, salary double, award int,
fieldplays_g int, fieldplays_hp int, fieldplays_innouts int,
fieldplays_p int, fieldplays_a int, fieldplays_e int, fieldplays_dp int,
fieldplays_ds int, fieldplays_sp int, fieldplays_pb int, fieldplays_sb int,
fieldplays_c int, appearances_g_all int, appearances_g_all int,
appearances_gu batting int, appearances_defense int,
appearances_g_p int, appearances_g_c int, appearances_g_1b int,
appearances_g_2b int, appearances_g_3b int, appearances_g_ss int, appearances_g_ss int,
appearances_g_1f int, appearances_g_cf int, appearances_g rf int, appearances_gcff int,
appearances_g_ph int, appearances_pr int, yearMin double, yearMax double
from baseball.Batting B JOIN Pitching P ON B.playerID = P.playerID
JOIN fielding F ON B.playerID = F.playerID
JOIN Team T ON b.teamid = t.teamid JOIN TeamFranchises TF ON
T.FranchID = t.franchid ...);
```

Now, create the data warehouse dimension tables. Listing 2 shows the code.
Listing 2. Create the data warehouse dimension tables

```$ Hive
Create table baseball_stats.dim_Players AS
    ( SELECT 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 
      FROM baseball.master .... );
```

Run a query

Let's run a few queries to make sure the data looks right. First, select all the data from the fact table (limit it to the first 10 rows). You can run a couple of other queries to ensure that the dimensional tables look right, that they connect to the fact table, and so on. You also can run a count on the fact table to make sure the total rows line up. Of course, you would have to correlate that back to the original base tables and add them up. Listing 3 shows the code to test whether data exists and is correct and whether dimensions connect to the fact table.

Listing 3. Test to see whether data exists and is correct and whether dimensions connect to the fact table

```$ HIVE
Use baseball_stats;
Select * from fact_player_stats limit 10;
Select A.PlayerID, A.name, B.teamID, B.AB, B.R, B.H, B.2B, B.3B, B.HR, B.RBI 
    FROM dim_players A JOIN fact_player_stats B ON a.playerid = b.playerid;
Select count(*) from fact_player_stats;
Select count(*) from dim_players.
Select max(r) from fact_player_stats where playerid=1234;
```

If you want to do a more thorough validation of the data in the Hive data warehouse, you can use a minimum, maximum, or average on certain columns or all of the columns and compare the results to the original base tables. You would be looking for exact matches.

Conclusion

Obviously, a lot of design work goes into creating a simple star schema. You can go back and create a fact table based on teams, for example. The benefit of this data warehouse schema is that you won't have to join a lot of tables. And for this example, there are few updates besides yearly stats, and, in that case, overwriting the data warehouse tables or adding another year and recalculating shouldn't be a problem.

Hive certainly has its limitations, but if you're working on a budget or the tools have been mandated from on high, with a bit of work, Hive can give you the data warehouse you need.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample CSV file</td>
<td>lahan2012-csv.zip</td>
<td>11MB</td>
</tr>
</tbody>
</table>
Related topics

- Check out the The Data Warehouse Institute (TDWI), a valuable resource for BI and data warehousing.
- Visit Hadoop.org for information about all things Hadoop.
- Check out the Hive and Sqoop project sites to learn more.
- Download InfoSphere BigInsights Quick Start Edition, available as a native software installation or as a VMware image.