Cloudera Impala

John Russell

O'REILLY®

O'REILLY®
Strata
Making Data Work

ASK BIGGER QUESTIONS

cloudera®
IMPALA

Cloudera Impala

John Russell

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo O'REILLY®

Cloudera Impala

by John Russell

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Mike Loukides

October 2013: First Edition

Revision History for the First Edition:

2013-10-07: First release

ISBN: 978-1-491-94535-3

[LSI]

Table of Contents

Introduction

Cloudera Impala is an open source project that is opening up the Apache Hadoop software stack to a wide audience of database analysts, users, and developers. The Impala massively parallel processing (MPP) engine makes SQL queries of Hadoop data simple enough to be accessible to analysts familiar with SQL and to users of business intelligence tools, and it's fast enough to be used for interactive exploration and experimentation.

The Impala software is written from the ground up for high performance for SQL queries distributed across clusters of connected machines.

This Document

This article is intended for a broad audience of users from a variety of database, data warehousing, or Big Data backgrounds. SQL and Linux experience is a plus. Experience with the Apache Hadoop software stack is useful but not required.

This article points out wherever some aspect of Impala architecture or usage might be new to people who are experienced with databases but not the Apache Hadoop software stack, or vice versa.

The SQL examples in this article are geared toward new users trying out Impala for the first time, showing the simplest way to do things rather than the best practices for performance and scalability.

Impala's Place in the Big Data Ecosystem

The Cloudera Impala project arrives in the Big Data world at just the right moment. Data volume is growing fast, outstripping what can be realistically stored or processed on a single server. Some of the original practices for Big Data are evolving to open that field up to a larger audience of users and developers.

Impala brings a high degree of flexibility to the familiar database ETL process. You can query data that you already have in various standard Apache Hadoop file formats. You can access the same data with a combination of Impala, Apache Hive, and other Hadoop components such as Apache Pig or Cloudera search, without needing to duplicate or convert the data. When query speed is critical, the new Parquet columnar file format makes it simple to reorganize data for maximum performance of data warehouse-style queries.

Traditionally, Big Data processing has been like batch jobs from mainframe days, where unexpected or tough questions required running jobs overnight or all weekend. The goal of Impala is to express even complicated queries directly with familiar SQL syntax, running fast enough that you can get an answer to an unexpected question while a meeting or phone call is in progress. (We refer to this degree of responsiveness as "interactive.")

For users and business intelligence tools that speak SQL, Impala brings a more effective development model than writing a new Java program to handle each new kind of analysis. Although the SQL language has a long history in the computer industry, with the combination of Big Data and Impala, it is once again cool. Now you can write sophisticated

analysis queries using natural expressive notation, the same way Perl mongers do with text-processing scripts. You can traverse large data sets and data structures interactively like a Pythonista inside the Python shell. You can avoid memorizing verbose specialized APIs; SQL is like a RISC instruction set that focuses on a standard set of powerful commands. When you do need access to API libraries for capabilities such as visualization and graphing, you can access Impala data from programs written in languages such as Java and C++ through the standard JDBC and ODBC protocols.

How Impala Fits Into Your Big Data Workflow

Impala streamlines your Big Data workflow through a combination of flexibility and performance.

Flexibility

Impala integrates with existing Hadoop components, security, metadata, storage management, and file formats. You keep the flexibility you already have with these Hadoop strong points and add capabilities that make SQL queries much easier and faster than before.

With SQL, you can turn complicated analysis programs into simple, straightforward queries. To help answer questions and solve problems, you can enlist a wide audience of analysts who already know SQL or the standard business intelligence tools built on top of SQL. They know how to use SQL or BI tools to analyze large data sets and how to quickly get accurate answers for many kinds of business questions and "what if" scenarios. They know how to design data structures and abstractions that let you perform this kind of analysis both for common use cases and unique, unplanned scenarios.

The filtering, calculating, sorting, and formatting capabilities of SQL let you delegate those operations to the Impala query engine, rather than generating a large volume of raw results and coding client-side logic to organize the final results for presentation.

Impala embodies the Big Data philosophy that large data sets should be just as easy and economical to work with as small ones. Large volumes of data can be imported instantaneously, without any changes

to the underlying data files. You have the flexibility to query data in its raw original form, or convert frequently queried data to a more compact, optimized form. Either way, you do not need to guess which data is worth saving; you preserve the original values, rather than condensing the data and keeping only the summarized form. There is no required step to reorganize the data and impose structure and rules, such as you might find in a traditional data warehouse environment.

Performance

The Impala architecture provides such a speed boost to SQL queries on Hadoop data that it will change the way you work. Whether you currently use MapReduce jobs or even other SQL-on-Hadoop technologies such as Hive, the fast turnaround for Impala queries opens up whole new categories of problems that you can solve. Instead of treating Hadoop data analysis as a batch process that requires extensive planning and scheduling, you can get results any time you want them. Instead of doing a mental context switch as you kick off a batch query and later discover that it has finished, you can run a query, evaluate the results immediately, and fine-tune the query if necessary. This fast iteration helps you zero in on the best solution without disrupting your workflow. Instead of trying to shrink your data down to the most important or representative subset, you can analyze everything you have, producing the most accurate answers and discovering new trends.

Perhaps you have had the experience of using software or a slow computer where after every command or operation, you waited so long that you had to take a coffee break or switch to another task. Then when you switched to faster software or upgraded to a faster computer, the system became so responsive that it lifted your mood, reengaged your intellect, and sparked creative new ideas. That is the type of reaction Impala aims to inspire in Hadoop users.

Coming to Impala from an RDBMS Background

When you come to Impala from a background with a traditional relational database product, you find the same familiar SQL query language and DDL statements. Data warehouse experts will already be familiar with the notion of partitioning. If you have only dealt with smaller OLTP-style databases, the emphasis on large data volumes will expand your horizons.

Standard SQL

The great thing about coming to Impala with relational database experience is that the query language is completely familiar: it's just SQL! The **`SELECT`** syntax works like you are used to, with joins, views, relational operators, aggregate functions, **`ORDER BY`** and **`GROUP BY`**, casts, column aliases, built-in functions, and so on.

Because Impala is focused on analytic workloads, it currently doesn't have OLTP-style operations such as **`DELETE`**, **`UPDATE`**, or **`COMMIT`** / **`ROLLBACK`**. It also does not have indexes, constraints, or foreign keys; data warehousing experts traditionally minimize their reliance on these relational features because they involve performance overhead that can be too much when dealing with large amounts of data.

The initial Impala release supports a set of core column data types: **`STRING`** instead of **`VARCHAR`** or **`VARCHAR2`**; **`INT`** and **`FLOAT`** instead of **`NUMBER`**; and no **`BLOB`** type.

The **`CREATE TABLE`** and **`INSERT`** statements incorporate some of the format clauses that you might expect to be part of a separate data-

loading utility, because Impala is all about the shortest path to ingest and analyze data.

The **`EXPLAIN`** statement provides a logical overview of statement execution. Instead of showing how a query uses indexes, the Impala **`EXPLAIN`** output illustrates how parts of the query are distributed among the nodes in a cluster, and how intermediate results are combined at the end to produce the final result set.

Impala implements SQL-92 standard features with some enhancements from later SQL standards. It does not yet have does not yet have the SQL-99 and SQL-2003 analytic functions, although those items are on the product roadmap.

Storage, Storage, Storage

Several aspects of the Apache Hadoop workflow, with Impala in particular, are very freeing to a longtime database user:

- The data volumes are so big that you start out with a large pool of storage to work with. This reality tends to reduce the bureaucracy and other headaches associated with a large and fast-growing database.
- The flexibility of Impala schemas means there is less chance of going back and reorganizing old data based on recent changes to table structures.
- The HDFS storage layer means that replication and backup are handled at the level of an entire cluster rather than for each individual database or table.

The key is to store the data in some form as quickly, conveniently, and scalably as possible through the flexible Hadoop software stack and file formats. You can come back later and define an Impala schema for existing data files. The data loading process for Impala is very lightweight; you can even leave the data files in their original locations and query them there.

Billions and Billions of Rows

Although Impala can work with data of any volume, its performance and scalability shine when the data is large enough to be impractical to produce, manipulate, and analyze on a single server. Therefore, after

you do your initial experiments to learn how all the pieces fit together, you very quickly scale up to working with tables containing billions of rows and gigabytes, terabytes, or larger of total volume. The toy problems you tinker with might involve data sets bigger than you ever used before. You might have to rethink your benchmarking techniques if you are used to using smaller volumes—meaning millions of rows or a few tens of gigabytes. You will start relying on the results of analytic queries because the scale will be bigger than you can grasp through your intuition.

For problems that do not tax the capabilities of a single machine, many alternative techniques offer about the same performance. After all, if all you want to do is sort or search through a few files, you can do that plenty fast with Perl scripts or Unix commands such as grep. The Big Data issues come into play when the files are too large to fit on a single machine, or when you want to run hundreds of such operations concurrently, or when an operation that takes only a few seconds for megabytes of data takes hours when the data volume is scaled up to gigabytes or petabytes.

You can learn the basics of Impala SQL and confirm that all the prerequisite software is configured correctly using tiny data sets, as in the examples throughout this article. That's what we call a "canary test," to make sure all the pieces of the system are hooked up properly.

To start exploring scenarios involving performance testing, scalability, and multi-node cluster configurations, you typically use much, much larger data sets. Try generating a billion rows of representative data, then once the raw data is in Impala, experiment with different combinations of file formats, compression codecs, and partitioning schemes.

Don't put too much faith in performance results involving only a few gigabytes of data. Only when you blow past the data volume that a single server could reasonably handle or saturate the I/O channels of your storage array can you fully appreciate the performance speedup of Impala over competing solutions and the effects of the various tuning techniques. To really be sure, do trials using volumes of data similar to your real-world system.

If today your data volume is not at this level, next year it might be. You should not wait until your storage is almost full (or even half full) to set up a big pool of HDFS storage on cheap commodity hardware. Whether or not your organization has already adopted the Apache

Hadoop software stack, experimenting with Cloudera Impala is a valuable exercise to future-proof your enterprise.

How Impala Is Like a Data Warehouse

With Impala, you can unlearn some notions from the RDBMS world. Long-time data warehousing users might already be in the right mindset, because some of the traditional database best practices naturally fall by the wayside as data volumes grow and raw query speed becomes the main consideration. With Impala, you will do less planning for normalization, skip the time and effort that goes into designing and creating indexes, and stop worrying when queries cause full-table scans.

Impala, as with many other parts of the Hadoop software stack, is optimized for fast bulk read and data load operations. Many data warehouse-style queries involve either reading all the data ("what is the highest number of different visitors our website ever had in one day?") or reading some large set of values organized by criteria such as time ("what were the total sales for the company in the fourth quarter of last year?"). Impala divides up the work of reading large data files across the nodes of a cluster. Impala also does away with the performance overhead of creating and maintaining indexes, instead taking advantage of the multimegabyte HDFS block size to read and process high volumes of data in parallel across multiple networked servers. As soon as you load the data, it is ready to be queried. Impala can run efficient ad hoc queries against any columns, not just preplanned queries using a small set of indexed columns.

In a traditional database, normalizing the data and setting up primary key / foreign key relationships can be time consuming for large data volumes. That is why data warehouses (and also Impala) are more tolerant of denormalized data, with values that are duplicated and possibly stored in raw string form rather than condensed to numeric IDs. The Impala query engine works very well for data warehouse-style input data by doing bulk reads and distributing the work among nodes in a cluster. Impala can even condense bulky, raw data into a data warehouse-friendly layout automatically as part of a conversion to the Parquet file format.

When executing a query involves sending requests to several servers in a cluster, the way to minimize total resource consumption (disk I/O, network traffic, and so on) is to make each server do as much local

processing as possible before sending back the results. Impala queries typically work on data files in the multimegabyte or gigabyte range, where a server can read through large blocks of data very quickly. Impala does as much filtering and computation as possible on the server that reads the data to reduce overall network traffic and resource usage on the other nodes in the cluster. Thus, Impala can very efficiently perform "full table scans" of large tables, the kinds of queries that are common in analytical workloads.

Impala makes use of partitioning, another familiar notion from the data warehouse world. Partitioning is one of the major optimization techniques you will employ to reduce disk I/O and maximize the scalability of Impala queries. Partitioned tables physically divide the data based on one or more criteria, typically by date or geographic region, so that queries can filter out irrelevant data and skip the corresponding data files entirely. Although Impala can quite happily read and process huge volumes of data, your query will be that much faster and more scalable if a query for a single month only reads one-twelfth of the data for that year, or if a query for a single US state only reads one-fiftieth of the data for the entire country. Partitioning typically does not impose much overhead on the data loading phase; the partitioning scheme usually matches the way data files are already divided, such as when you load a group of new data files each day.

Your First Impala Queries

To get your feet wet with the basic elements of Impala query syntax such as the underlying data types and expressions, you can run queries without any table or **`WHERE`** clause at all:

```
SELECT 2+2;
SELECT SUBSTR('Hello world',1,5);
SELECT CAST(99.5 AS INT);
SELECT CONCAT('aaa',"bbb",'ccc');
SELECT 2 > 1;
SELECT NOW();
```

Because Impala does not have any built-in tables, running queries against real data requires a little more preparation. We'll use the **`INSERT ... VALUES`** statement to create a couple of "toy" tables, although for scalability reasons we would quickly leave the **`VALUES`** clause behind when working with data of any significant volume.

```
-- Set up a table to look up names based on abbreviations.
CREATE TABLE canada_regions (name STRING, abbr STRING);
```

```
INSERT INTO canada_regions VALUES
  ("Newfoundland and Labrador" ,"NL"),
  ("Prince Edward Island","PE"),
  ("New Brunswick","NB"), ("Nova Scotia","NS"),
  ("Quebec","PQ"), ("Ontario","ON"),
  ("Manitoba","MB"), ("Saskatchewan","SK"), ("Alberta","AB"),
  ("British Columbia","BC"), ("YT","Yukon"),
  ("Northwest Territories","NT"), ("Nunavut","NU");

-- Set up a potentially large table
-- with data values we will use to answer questions.
CREATE TABLE canada_facts
  (id STRING, sq_mi INT, population INT);

-- The INSERT statement either appends to existing data in
-- a table via INSERT INTO, or replaces the data entirely
-- via INSERT OVERWRITE.

-- Here we start by inserting partial data...
INSERT INTO canada_facts VALUES ("NL",156453,514536),
  ("PE",2190,140204);
-- ... then we replace the entire contents of the table
-- with the complete data.
INSERT OVERWRITE canada_facts VALUES ("NL",156453,514536),
  ("PE",2190,140204), ("NB",28150,751171), ("NS",21345,921727),
  ("PQ",595391,8054756), ("ON",415598,13505900),
  ("MB",250950,1208268), ("SK",251700,1033381),
  ("AB",255541,3645257), ("BC",364764,4400057),
  ("YT",186272,33897), ("NT",519734,41462), ("NU",78715,31906);

-- A view is an alias for a longer query, and takes no time or
-- storage to set up.
-- Querying a view avoids repeating clauses over and over.
CREATE VIEW atlantic_provinces AS SELECT * FROM canada_facts
  WHERE id IN ('NL','PE','NB','NS');
CREATE VIEW maritime_provinces AS SELECT * FROM canada_facts
  WHERE id IN ('PE','NB','NS');
CREATE VIEW prairie_provinces AS SELECT * FROM canada_facts
  WHERE id IN ('MB','SK','AB');

-- We can query a single table, multiple tables via joins, or
build new queries on top of views.
SELECT name AS "Region Name" FROM canada_regions
  WHERE abbr LIKE 'N%';
+---------------------------+
| region name               |
+---------------------------+
| Newfoundland and Labrador |
| New Brunswick             |
| Nova Scotia               |
| Northwest Territories     |
```

```
| Nunavut                   |
+---------------------------+
-- Here we get the population figure from one table and the
-- full name from another.
SELECT canada_regions.name, canada_facts.population
  FROM canada_facts JOIN canada_regions
  ON (canada_regions.abbr = canada_facts.id);
+---------------------------+------------+
| name                      | population |
+---------------------------+------------+
| Newfoundland and Labrador | 514536     |
| Prince Edward Island      | 140204     |
| New Brunswick             | 751171     |
| Nova Scotia               | 921727     |
| Quebec                    | 8054756    |
| Ontario                   | 13505900   |
| Manitoba                  | 1208268    |
| Saskatchewan              | 1033381    |
| Alberta                   | 3645257    |
| British Columbia          | 4400057    |
| Northwest Territories     | 41462      |
| Nunavut                   | 31906      |
+---------------------------+------------+
-- Selecting from a view lets us compose a series of
-- filters and functions.
SELECT SUM(population) AS "Total Population"
  FROM atlantic_provinces;
+------------------+
| total population |
+------------------+
| 2327638          |
+------------------+
SELECT AVG(sq_mi) AS "Area (Square Miles)"
  FROM prairie_provinces;
+---------------------+
| area (square miles) |
+---------------------+
| 252730.3333333333   |
+---------------------+
```

Getting Data into an Impala Table

Because Impala is typically near the end of the extract-transform-load (ETL) pipeline, its focus is on working with existing data rather than creating data from scratch. Thus, you typically start with data files, then get them into Impala using one of these techniques:

- Issue a **`LOAD DATA`** statement to move data files into the Impala data directory for a table.

- Physically copy or move data files into the Impala data directory for a table. (Not needed as much now, since the **`LOAD DATA`** statement debuted in Impala 1.1.)
- Issue a **`CREATE EXTERNAL TABLE`** statement with a **`LOCATION`** clause, to point the Impala table at data files stored in HDFS outside the Impala data directories.
- Issue an **`INSERT ... SELECT`** statement to copy data from one table to another. You can convert the data to a different file format in the destination table, filter the data using **`WHERE`** clauses, and transform values using operators and built-in functions.
- Use any of the Hive data loading techniques, especially for tables using the Avro, SequenceFile, or RCFile formats. Because Impala and Hive tables are interchangeable, once data is loaded through Hive, you can query it through Impala.

Pro Tip

If you are already using batch-oriented SQL-on-Hadoop technology through the Apache Hive component, you can reuse Hive tables and their data directly in Impala without any time-consuming loading or conversion step. This cross-compatibility applies to Hive tables that use Impala-compatible types for all columns.

For Beginners Only

Issue one or more **`INSERT ... VALUES`** statements to create new data from literals and function return values. We list this technique last because it really only applies to very small volumes of data, or to data managed by HBase. Each **`INSERT`** statement produces a new tiny data file, which is a very inefficient layout for Impala queries against HDFS data. On the other hand, if you are entirely new to Hadoop, this is a simple way to get started and experiment with SQL syntax and various table layouts, data types, and file formats, but you should expect to outgrow the **`INSERT ... VALUES`** syntax relatively quickly. You might graduate from tables with a few dozen rows straight to billions of rows when you start working with real data. Make sure to clean up any unneeded small files after finishing with **`INSERT ... VALUES`** experiments.

Coming to Impala from a Unix or Linux Background

If you are a Unix-oriented tools hacker, Impala fits in nicely at the tail end of your workflow. You create data files with a wide choice of formats for convenience, compactness, or interoperability with different Apache Hadoop components. You tell Impala where those data files are and what fields to expect inside them. That's it! Then, let the SQL queries commence. You can see the results of queries in a terminal window through the **`impala-shell`** command, save them in a file to process with other scripts or applications, or pull them straight into a visualizer or report application through the standard ODBC or JDBC interfaces. It's transparent to you that behind the scenes, the data is spread across multiple storage devices and processed by multiple servers.

Administration

When you administer Impala, it is a straightforward matter of some daemons communicating with each other through a predefined set of ports. There is an `impalad` daemon that runs on each data node in the cluster and does most of the work, a `statestored` daemon that runs on one node and performs periodic health checks on the **`impalad`** daemons, and the roadmap includes one more planned service. Log files show the Impala activity occurring on each node.

Administration for Impala is typically folded into administration for the overall cluster through the Cloudera Manager product. You monitor all nodes for out-of-space problems, CPU spikes, network failures,

and so on, rather than on a node-by-node or component-by-component basis.

Files and Directories

When you design a Impala schema, the physical implementation maps very intuitively to a set of predictably named directories. The data for a table is made up of the contents of all the files within a specified directory. Partitioned tables have extra levels of directory structure to allow queries to limit their processing to smaller subsets of data files. For files loaded directly into Impala, the files even keep their original names.

SQL Statements Versus Unix Commands

The default data format of human-readable text fits easily into a typical Unix toolchain. You can even think of some of the SQL statements as analogous to familiar Unix commands:

- **`CREATE DATABASE`** = `mkdir`
- **`CREATE TABLE`** = `mkdir`
- **`CREATE TABLE ... PARTITIONED BY`** = `mkdir -p`
- **`CREATE EXTERNAL TABLE`** = `ln -s`
- **`ALTER TABLE ... ADD PARTITION`** = `mkdir`
- **`USE`** = `cd`
- **`SELECT`** = `grep`, `find`, `sed`, `awk`, `cut`, `perl`, `python`; here is where you spend most of your time and creativity
- **`INSERT`** = `cp`, `tee`, `dd`, `grep`, `find`, `sed`, `awk`, `cut`, `perl`, `python`; with Impala, most **`INSERT`** statements include a **`SELECT`** portion, because the typical use case is copying data from one table to another
- **`DROP DATABASE`** = `rmdir`
- **`DROP TABLE`** = `rm -r`
- ***`SELECT COUNT(*)`*** = `wc -l`, `grep -c`

A Quick Unix Example

Here is what your first Unix command line session might look like when you are using Impala:

```
$ cat >csv.txt
1,red,apple,4
2,orange,orange,4
3,yellow,banana,3
4,green,apple,4
^D
$ cat >more_csv.txt
5,blue,bubblegum,0.5
6,indigo,blackberry,0.2
7,violet,edible flower,0.01
8,white,scoop of vanilla ice cream,3
9,black,licorice stick,0.2
^D
$ hadoop fs -mkdir /user/hive/staging
$ hadoop fs -put csv.txt /user/hive/staging
$ hadoop fs -put more_csv.txt /user/hive/staging
```

Now that the data files are in the HDFS filesystem, let's go into the Impala shell and start working with them. (Some of the prompts and output are abbreviated here for easier reading by first-time users.)

```
$ impala-shell
> create database food_colors;
> use food_colors;
> create table food_data
    (id int, color string, food string, weight float)
    row format delimited fields terminated by ',';
> -- Here's where we move the data files from an arbitrary
  -- HDFS location to under Impala control.
> load data inpath '/user/hive/staging' into table food_data;
Query finished, fetching results ...
+-----------------------------------------------------------+
| summary                                                   |
+-----------------------------------------------------------+
| Loaded 2 file(s). Total files in destination location: 2 |
+-----------------------------------------------------------+
> select food, color as "Possible Color" from food_data where
    food = 'apple';
Query finished, fetching results ...
+-------+----------------+
| food  | possible color |
+-------+----------------+
| apple | red            |
| apple | green          |
+-------+----------------+
Returned 2 row(s) in 0.66s
```

```
> select food as "Top 5 Heaviest Foods", weight
    from food_data
    order by weight desc limit 5;
Query finished, fetching results ...
+---------------------------+---------------------+
| top 5 heaviest foods      | weight              |
+---------------------------+---------------------+
| orange                    | 4                   |
| apple                     | 4                   |
| apple                     | 4                   |
| scoop of vanilla ice cream | 3                   |
| banana                    | 3                   |
+---------------------------+---------------------+
Returned 5 row(s) in 0.49s
> quit;
```

Back in the Unix shell, see how the **`CREATE DATABASE`** and **`CREATE TABLE`** statements created some new directories and how the **`LOAD DATA`** statement moved the original data files into an Impala-managed directory:

```
$ hadoop fs -ls -R /user/hive/warehouse/food_colors.db
drwxrwxrwt   - impala hive          0 2013-08-29 16:14 /user/h
ive/warehouse/food_colors.db/food_data
-rw-rw-rw-   3 hdfs   hive         66 2013-08-29 16:12 /user/h
ive/warehouse/food_colors.db/food_data/csv.txt
-rw-rw-rw-   3 hdfs   hive        139 2013-08-29 16:12 /user/h
ive/warehouse/food_colors.db/food_data/more_csv.txt
```

In one easy step, you have gone from a collection of human-readable text files to a SQL table that you can query using standard, widely known syntax. The data is automatically replicated and distributed across a cluster of networked machines by virtue of being put into an HDFS directory.

These same basic techniques scale up to enormous tables with billions of rows. By that point, you would likely be using a more compact and efficient data format than plain text files, and you might include a partitioning clause in the **`CREATE TABLE`** statement to split up the data files by date or category. Don't worry, you can easily upgrade your Impala tables and rearrange the data as you learn the more advanced Impala features.

Coming to Impala from an Apache Hadoop Background

If you are already experienced with the Apache Hadoop software stack and are adding Impala as another arrow in your quiver, you will find it interoperable on several levels.

Apache Hive

Apache Hive is the first generation of SQL-on-Hadoop technology, focused on batch processing with long-running jobs. Impala tables and Hive tables are highly interoperable, allowing you to switch into Hive to do a batch operation such as a data import, then switch back to Impala and do an interactive query on the same table. You might see HDFS paths such as **`/user/hive/warehouse`** in Impala examples, because for simplicity we sometimes use this historical default path for both Impala and Hive databases.

For users who already use Hive to run SQL batch jobs on Hadoop, the Impala SQL dialect is highly compatible with HiveQL. The main limitations involve nested data types, UDFs, and custom file formats. These are not permanent limitations—they're being worked through in priority sequence based on the Impala roadmap.

If you are an experienced Hive user, one thing to unlearn is the notion of a SQL query as a long-running, heavyweight job. With Impala, you typically issue the query and see the results in the same interactive session of the Impala shell or a business intelligence tool. For example, when you ask for even a simple query such as **`*SELECT COUNT(\*)*`** in the Hive shell, it prints many lines of status output showing mapper

and reducer processes, and even gives you the **`kill`** command to run if the query takes too long or goes out of control. Impala requires a lot less startup time, administrative overhead, and data transfer between nodes. When you issue a **`*SELECT COUNT(*)*`** query in the Impala shell, Impala just tells you how many rows are in the table. The same applies when you run more complicated queries, possibly involving joins between multiple tables and various types of aggregation and filtering operations; Impala takes care of the behind-the-scenes setup, lets you focus on interpreting the results, and processes the query so fast that it encourages exploration and experimentation.

Apache HBase

Apache HBase is a key-value store that provides some familiar database features but does not include a SQL interface. If you store data in HBase already, Impala can run SQL queries against that data. Querying HBase data through Impala is a good combination for looking up single rows or ranges of rows.

MapReduce and Apache Pig

If you use MapReduce, Apache Pig, or other Hadoop components to produce data in standard file formats such as text-based with optional LZO compression, Avro, SequenceFile, or RCFile, you can bring those data files under Impala control by simply moving them to the appropriate directory, or even have Impala query them from their original locations. The new Parquet file format is natively supported on Hadoop, with access from MapReduce, Pig, Impala, and Hive. You can also produce data files in these various formats through libraries available for Python, Java, and so on. (You could create the simple delimited text format from any programming language, even a simple shell script.) You can choose whichever format is most convenient based on your current workflow, and either leave the data in that original format, or do a final conversion step if a different format offers much better compression or query performance for frequently consulted data.

Schema on Read

One of the tenets of Hadoop is "schema on read," meaning that you are not required to do extensive planning up front about how your data is laid out, and you are not penalized if you later need to change

or fine-tune your original decisions. Historically, this principle has clashed with the traditional SQL model where a **CREATE TABLE** statement defines a precise layout for a table, and data is reorganized to match this layout during the load phase. Impala bridges these philosophies in clever ways:

- Impala lets you define a schema for data files that you already have and immediately begin querying that data with no change to the underlying raw files.
- Impala does not require any length constraints for strings. No more trying to predict how much room to allow for the longest possible name, address, phone number, product ID, and so on.
- In the simplest kind of data file (using text format), fields can be flexibly interpreted as strings, numbers, timestamps, or other kinds of values.
- Impala allows data files to have more or fewer columns than the corresponding table. It ignores extra fields in the data file, and returns **NULL** if fields are missing from the data file. You can rewrite the table definition to have more or fewer columns and mix and match data files with the old and new column definitions.
- You can redefine a table to have more columns, fewer columns, or different data types at any time. The data files are not changed in any way.
- In a partitioned table, if newer data arrives in a different file format, you can change the definition of the table only for certain partitions, rather than going back and reformatting or converting all the old data.
- Impala can query data files stored outside its standard data repository. You could even point multiple tables (with different column definitions) at the same set of data files—for example, to treat a certain value as a string for some queries and a number for other queries.

The benefits of this approach include more flexibility, less time and effort spent converting data into a rigid format, and less resistance to the notion of fine-tuning the schema as needs change and you gain more experience. For example, if a counter exceeds the maximum value for an **`INT`**, you can promote it to a **`BIGINT`** with minimal hassle. If you originally stored postal codes or credit card numbers as integers and later received data values containing dashes or spaces, you could switch those columns to strings without reformatting the original data.

Getting Started with Impala

Depending on your background and existing Apache Hadoop infrastructure, you can approach the Cloudera Impala product from different angles:

- If you are from a database background and a Hadoop novice, the Cloudera QuickStart VM lets you try out the basic Impala features straight out of the box. This single-node VM configuration is suitable to get your feet wet with Impala. (For performance or scalability testing, you would use real hardware in a cluster configuration.) You run the VM in VMWare, KVM, or VirtualBox, start the Impala service through the Cloudera Manager web interface, and then interact with Impala through the **`impala-shell`** interpreter or the ODBC and JDBC interfaces.
- For more serious testing or large-scale deployment, you can download and install the Cloudera Impala software in a real cluster environment. You can freely install the software either through standalone packages or by using the Cloudera Manager "parcel" feature, which enables easier upgrades. You install the Impala server on each data node and designate one node (typically the same as the Hadoop namenode) to also run the Impala StateStore daemon. The simplest way to get up and running is through the Cloudera Manager application, where you can bootstrap the whole process of setting up a Hadoop cluster with Impala just by specifying a list of hostnames for the cluster.
- If you want to understand how Impala works at a deep level, you can get the Impala source code from GitHub and build it yourself.

You can join the open source project discussion through the original mailing list or the new discussion forum.

Further Reading and Downloads

- Download Standard (free) version of Cloudera Manager + CDH4 + Impala
- Download Enterprise trial of Cloudera Manager + CDH4 + Impala
- Download Cloudera QuickStart VM, including Impala
- Installation and User Guide for Impala
- Impala SQL Language Reference
- GitHub repository for Impala

Conclusion

In this article, you have learned how Impala fits into the Hadoop software stack:

- Querying data files stored in HDFS.
- Enabling interactive queries for data originally managed by Hive. Using Hive where convenient for some ETL tasks, then querying the data in Impala.
- Providing a SQL frontend for data managed by HBase.
- Using data files produced by MapReduce, Pig, and other Hadoop components.
- Utilizing data formats from simple (text), to compact and efficient (Avro, RCFile, SequenceFile), to optimized for data warehouse queries (Parquet).

You have seen the interesting benefits Impala brings to users coming from different backgrounds:

- For Hadoop users, how Impala brings the familiarity and flexibility of fast, interactive SQL to the Hadoop world.
- For database users, how the combination of Hadoop and Impala makes it simple to set up a distributed database for data warehouse-style queries.

You have gotten a taste of what is involved in setting up Impala, loading data, and running queries.

The rest is in your hands!

Further Reading and Downloads

- Impala documentation
- Cloudera software downloads
- Google Groups mailing list for Impala
- Impala community forum
- Impala support site
- Impala product roadmap
- Github repository for Impala

About the Author

John Russell is a software developer and technical writer, and he's currently the documentation lead for the Cloudera Impala project. He has a broad range of database and SQL experience from previous roles on industry-leading teams. For DB2, he designed and coded the very first Information Center. For Oracle Database, he documented application development subjects and designed and coded the Project Tahiti doc search engine. For MySQL, he documented the InnoDB storage engine. Originally from Newfoundland, Canada, John now resides in Berkeley, California.

Change the world with data. We'll show you how.

strataconf.com

Strata CONFERENCE + HADOOP WORLD

Oct 28 – 30, 2013
New York, NY

O'REILLY

Strata CONFERENCE
Making Data Work

Nov 11 – 13, 2013
London, England

O'REILLY

Strata CONFERENCE
Making Data Work

Feb 11 – 13, 2014
Santa Clara, CA

O'REILLY

Strata RX CONFERENCE
Data Makes a Difference

April 23–25, 2014
Boston, MA

O'REILLY®

Spreading the knowledge of innovators.