

## Planning for Apache Impala

Date published: 2019-11-01

Date modified: 2021-06-08



# Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

# Contents

<b>Guidelines for Schema Design.....</b>	<b>4</b>
<b>User Account Requirements.....</b>	<b>5</b>

# Guidelines for Schema Design

Use the guidelines in this topic to construct an optimized and scalable schema that integrates well with your existing data management processes.

## Prefer binary file formats over text-based formats

To save space and improve memory usage and query performance, use binary file formats for any large or intensively queried tables. Parquet file format is the most efficient for data warehouse-style analytic queries. Avro is the other binary file format that Impala supports, that you might already have as part of a Hadoop ETL pipeline.

Although Impala can create and query tables with the RCFile and SequenceFile file formats, such tables are relatively bulky due to the text-based nature of those formats, and are not optimized for data warehouse-style queries due to their row-oriented layout. Impala does not support INSERT operations for tables with these file formats.

Guidelines:

- For an efficient and scalable format for large, performance-critical tables, use the Parquet file format.
- To deliver intermediate data during the ETL process, in a format that can also be used by other Hadoop components, Avro is a reasonable choice.
- For convenient import of raw data, use a text table instead of RCFile or SequenceFile, and convert to Parquet in a later stage of the ETL process.

## Use Snappy compression where practical

Snappy compression involves low CPU overhead to decompress, while still providing substantial space savings. In cases where you have a choice of compression codecs, such as with the Parquet and Avro file formats, use Snappy compression unless you find a compelling reason to use a different codec.

## Prefer numeric types over strings

If you have numeric values that you could treat as either strings or numbers (such as YEAR, MONTH, and DAY for partition key columns), define them as the smallest applicable integer types. For example, YEAR can be SMALLINT, MONTH and DAY can be TINYINT. Although you might not see any difference in the way partitioned tables or text files are laid out on disk, using numeric types will save space in binary formats such as Parquet, and in memory when doing queries, particularly resource-intensive queries such as joins.

## Partition, but do not over-partition

Partitioning is an important aspect of performance tuning for Impala. Set up partitioning for your biggest, most intensively queried tables.

If you are moving to Impala from a traditional database system, or just getting started in the Big Data field, you might not have enough data volume to take advantage of Impala parallel queries with your existing partitioning scheme. For example, if you have only a few tens of megabytes of data per day, partitioning by YEAR, MONTH, and DAY columns might be too granular. Most of your cluster might be sitting idle during queries that target a single day, or each node might have very little work to do. Consider reducing the number of partition key columns so that each partition directory contains several gigabytes worth of data.

For example, consider a Parquet table where each data file is 1 HDFS block, with a maximum block size of 1 GB. (In Impala 2.0 and later, the default Parquet block size is reduced to 256 MB. For this exercise, let's assume you have bumped the size back up to 1 GB by setting the query option `PARQUET_FILE_SIZE=1g`.) If you have a 10-node cluster, you need 10 data files (up to 10 GB) to give each node some work to do for a query. But each core on each machine can process a separate data block in parallel. With 16-core machines on a 10-node cluster, a query could process up to 160 GB fully in parallel. If there are only a few data files per partition, not only are most cluster nodes sitting idle during queries, so are most cores on those machines.

You can reduce the Parquet block size to as low as 128 MB or 64 MB to increase the number of files per partition and improve parallelism. But also consider reducing the level of partitioning so that analytic queries have enough data to work with.

### Run **COMPUTE STATS** after loading data

Impala makes extensive use of statistics about data in the overall table and in each column, to help plan resource-intensive operations such as join queries and inserting into partitioned Parquet tables. Because this information is only available after data is loaded, run the **COMPUTE STATS** statement on a table after loading or replacing data in a table or partition.

Having accurate statistics can make the difference between a successful operation, or one that fails due to an out-of-memory error or a timeout. When you encounter performance or capacity issues, always use the **SHOW STATS** statement to check if the statistics are present and up-to-date for all tables in the query.

When doing a join query, Impala consults the statistics for each joined table to determine their relative sizes and to estimate the number of rows produced in each join stage. When doing an **INSERT** into a Parquet table, Impala consults the statistics for the source table to determine how to distribute the work of constructing the data files for each partition.

### Verify sensible execution plans with **EXPLAIN** and **SUMMARY**

Before executing a resource-intensive query, use the **EXPLAIN** statement to get an overview of how Impala intends to parallelize the query and distribute the work. If you see that the query plan is inefficient, you can take tuning steps such as changing file formats, using partitioned tables, running the **COMPUTE STATS** statement, or adding query hints.

After you run a query, you can see performance-related information about how it actually ran by issuing the **SUMMARY** command in `impala-shell`. Prior to Impala 1.4, you would use the **PROFILE** command, but its highly technical output was only useful for the most experienced users. **SUMMARY**, new in Impala 1.4, summarizes the most useful information for all stages of execution, for all nodes rather than splitting out figures for each node.

## User Account Requirements

Impala creates and uses a user and group named `impala`.

Do not delete this account or group and do not modify the account's or group's permissions and rights. Ensure no existing systems obstruct the functioning of these accounts and groups. For example, if you have scripts that delete user accounts not in a white-list, add these accounts to the list of permitted accounts.

For correct file deletion during **DROP TABLE** operations, Impala must be able to move files to the HDFS trashcan. You might need to create an HDFS directory `/user/impala`, writeable by the `impala` user, so that the trashcan can be created. Otherwise, data files might remain behind after a **DROP TABLE** statement.

Impala should not run as root. Best Impala performance is achieved using direct reads, but root is not permitted to use direct reads. Therefore, running Impala as root negatively affects performance.

By default, any user can connect to Impala and access all the associated databases and tables. You can enable authorization and authentication based on the Linux OS user who connects to the Impala server, and the associated groups for that user. These security features do not change the underlying file permission requirements. The `impala` user still needs to be able to access the data files.