**CREATE TABLE Statement**

<https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_create_table.html>

Creates a new table and specifies its characteristics. While creating a table, you optionally specify aspects such as:

* Whether the table is internal or external.
* The columns and associated data types.
* The columns used for physically partitioning the data.
* The file format for data files.
* The HDFS directory where the data files are located.

**Syntax:**

The general syntax for creating a table and specifying its columns is as follows:

**Explicit column definitions:**

CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [*db\_name*.]*table\_name*

(*col\_name* *data\_type* [COMMENT '*col\_comment*'], ...)

[PARTITIONED BY (*col\_name* *data\_type* [COMMENT '*col\_comment*'], ...)]

[COMMENT '*table\_comment*']

[WITH SERDEPROPERTIES ('*key1*'='*value1*', '*key2*'='*value2*', ...)]

[

[ROW FORMAT *row\_format*] [STORED AS *file\_format*]

]

[LOCATION '*hdfs\_path*']

[TBLPROPERTIES ('*key1*'='*value1*', '*key2*'='*value2*', ...)]

[CACHED IN '*pool\_name*' [WITH REPLICATION = *integer*] | UNCACHED]

**Column definitions inferred from data file:**

CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [*db\_name*.]*table\_name*

LIKE PARQUET '*hdfs\_path\_of\_parquet\_file*'

[COMMENT '*table\_comment*']

[PARTITIONED BY (*col\_name* *data\_type* [COMMENT '*col\_comment*'], ...)]

[WITH SERDEPROPERTIES ('*key1*'='*value1*', '*key2*'='*value2*', ...)]

[

[ROW FORMAT *row\_format*] [STORED AS *file\_format*]

]

[LOCATION '*hdfs\_path*']

[TBLPROPERTIES ('*key1*'='*value1*', '*key2*'='*value2*', ...)]

[CACHED IN '*pool\_name*' [WITH REPLICATION = *integer*] | UNCACHED]

data\_type:

*primitive\_type*

| array\_type

| map\_type

| struct\_type

**CREATE TABLE AS SELECT:**

CREATE [EXTERNAL] TABLE [IF NOT EXISTS] *db\_name*.]*table\_name*

[PARTITIONED BY (*col\_name*[, ...])]

[COMMENT '*table\_comment*']

[WITH SERDEPROPERTIES ('*key1*'='*value1*', '*key2*'='*value2*', ...)]

[

[ROW FORMAT *row\_format*] [STORED AS *ctas\_file\_format*]

]

[LOCATION '*hdfs\_path*']

[TBLPROPERTIES ('*key1*'='*value1*', '*key2*'='*value2*', ...)]

[CACHED IN '*pool\_name*' [WITH REPLICATION = *integer*] | UNCACHED]

AS

*select\_statement*

primitive\_type:

TINYINT

| SMALLINT

| INT

| BIGINT

| BOOLEAN

| FLOAT

| DOUBLE

| DECIMAL

| STRING

| CHAR

| VARCHAR

| TIMESTAMP

complex\_type:

struct\_type

| array\_type

| map\_type

struct\_type: STRUCT < *name* : *primitive\_or\_complex\_type* [COMMENT '*comment\_string*'], ... >

array\_type: ARRAY < *primitive\_or\_complex\_type* >

map\_type: MAP < *primitive\_type*, *primitive\_or\_complex\_type* >

row\_format:

DELIMITED [FIELDS TERMINATED BY '*char*' [ESCAPED BY '*char*']]

[LINES TERMINATED BY '*char*']

file\_format:

PARQUET

| TEXTFILE

| AVRO

| SEQUENCEFILE

| RCFILE

ctas\_file\_format:

PARQUET

| TEXTFILE

**Statement type:** DDL

**Column definitions:**

Depending on the form of the CREATE TABLE statement, the column definitions are required or not allowed.

With the CREATE TABLE AS SELECT and CREATE TABLE LIKE syntax, you do not specify the columns at all; the column names and types are derived from the source table, query, or data file.

With the basic CREATE TABLE syntax, you must list one or more columns, its name, type, and optionally a comment, in addition to any columns used as partitioning keys. There is one exception where the column list is not required: when creating an Avro table with the STORED AS AVRO clause, you can omit the list of columns and specify the same metadata as part of the TBLPROPERTIES clause.

**Complex type considerations:**

The Impala complex types (STRUCT, ARRAY, or MAP) are available in CDH 5.5 / Impala 2.3 and higher. Because you can nest these types (for example, to make an array of maps or a struct with an array field), these types are also sometimes referred to as nested types. See [Complex Types (CDH 5.5 or higher only)](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_complex_types.html#complex_types) for usage details.

Impala can create tables containing complex type columns, with any supported file format. Because currently Impala can only query complex type columns in Parquet tables, creating tables with complex type columns and other file formats such as text is of limited use. For example, you might create a text table including some columns with complex types with Impala, and use Hive as part of your to ingest the nested type data and copy it to an identical Parquet table. Or you might create a partitioned table containing complex type columns using one file format, and use ALTER TABLE to change the file format of individual partitions to Parquet; Impala can then query only the Parquet-format partitions in that table.

Partitioned tables can contain complex type columns. All the partition key columns must be scalar types.

**Internal and external tables (EXTERNAL and LOCATION clauses):**

By default, Impala creates an "internal" table, where Impala manages the underlying data files for the table, and physically deletes the data files when you drop the table. If you specify the EXTERNAL clause, Impala treats the table as an "external" table, where the data files are typically produced outside Impala and queried from their original locations in HDFS, and Impala leaves the data files in place when you drop the table. For details about internal and external tables, see [Overview of Impala Tables](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_tables.html#tables).

Typically, for an external table you include a LOCATION clause to specify the path to the HDFS directory where Impala reads and writes files for the table. For example, if your data pipeline produces Parquet files in the HDFS directory /user/etl/destination, you might create an external table as follows:

CREATE EXTERNAL TABLE external\_parquet (c1 INT, c2 STRING, c3 TIMESTAMP)

STORED AS PARQUET LOCATION '/user/etl/destination';

Although the EXTERNAL and LOCATION clauses are often specified together, LOCATION is optional for external tables, and you can also specify LOCATION for internal tables. The difference is all about whether Impala "takes control" of the underlying data files and moves them when you rename the table, or deletes them when you drop the table. For more about internal and external tables and how they interact with the LOCATION attribute, see [Overview of Impala Tables](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_tables.html#tables).

**Partitioned tables (PARTITIONED BY clause):**

The PARTITIONED BY clause divides the data files based on the values from one or more specified columns. Impala queries can use the partition metadata to minimize the amount of data that is read from disk or transmitted across the network, particularly during join queries. For details about partitioning, see [Partitioning for Impala Tables](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_partitioning.html#partitioning).

Prior to CDH 5.7 / Impala 2.5, you could use a partitioned table as the source and copy data from it, but could not specify any partitioning clauses for the new table. In CDH 5.7 / Impala 2.5 and higher, you can now use the PARTITIONED BY clause with a CREATE TABLE AS SELECT statement. See the examples under the following discussion of the CREATE TABLE AS SELECT syntax variation.

**Specifying file format (STORED AS and ROW FORMAT clauses):**

The STORED AS clause identifies the format of the underlying data files. Currently, Impala can query more types of file formats than it can create or insert into. Use Hive to perform any create or data load operations that are not currently available in Impala. For example, Impala can create an Avro, SequenceFile, or RCFile table but cannot insert data into it. There are also Impala-specific procedures for using compression with each kind of file format. For details about working with data files of various formats, see [How Impala Works with Hadoop File Formats](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_file_formats.html#file_formats).

**Note:** In Impala 1.4.0 and higher, Impala can create Avro tables, which formerly required doing the CREATE TABLE statement in Hive. See [Using the Avro File Format with Impala Tables](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_avro.html#avro) for details and examples.

By default (when no STORED AS clause is specified), data files in Impala tables are created as text files with Ctrl-A (hex 01) characters as the delimiter. Specify the ROW FORMAT DELIMITED clause to produce or ingest data files that use a different delimiter character such as tab or |, or a different line end character such as carriage return or newline. When specifying delimiter and line end characters with the FIELDS TERMINATED BYand LINES TERMINATED BY clauses, use '\t' for tab, '\n' for newline or linefeed, '\r' for carriage return, and \0for ASCII nul (hex 00). For more examples of text tables, see [Using Text Data Files with Impala Tables](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_txtfile.html#txtfile).

The ESCAPED BY clause applies both to text files that you create through an INSERT statement to an Impala TEXTFILE table, and to existing data files that you put into an Impala table directory. (You can ingest existing data files either by creating the table with CREATE EXTERNAL TABLE ... LOCATION, the LOAD DATA statement, or through an HDFS operation such as hdfs dfs -put *file* *hdfs\_path*.) Choose an escape character that is not used anywhere else in the file, and put it in front of each instance of the delimiter character that occurs within a field value. Surrounding field values with quotation marks does not help Impala to parse fields with embedded delimiter characters; the quotation marks are considered to be part of the column value. If you want to use \ as the escape character, specify the clause in impala-shell as ESCAPED BY '\\'.

**Note:** The CREATE TABLE clauses FIELDS TERMINATED BY, ESCAPED BY, and LINES TERMINATED BY have special rules for the string literal used for their argument, because they all require a single character. You can use a regular character surrounded by single or double quotation marks, an octal sequence such as '\054' (representing a comma), or an integer in the range '-127'..'128' (with quotation marks but no backslash), which is interpreted as a single-byte ASCII character. Negative values are subtracted from 256; for example, FIELDS TERMINATED BY '-2' sets the field delimiter to ASCII code 254, the "Icelandic Thorn" character used as a delimiter by some data formats.

**Cloning tables (LIKE clause):**

To create an empty table with the same columns, comments, and other attributes as another table, use the following variation. The CREATE TABLE ... LIKE form allows a restricted set of clauses, currently only the LOCATION, COMMENT, and STORED AS clauses.

CREATE [EXTERNAL] TABLE [IF NOT EXISTS] [*db\_name*.]*table\_name*

LIKE { [*db\_name*.]*table\_name* | PARQUET '*hdfs\_path\_of\_parquet\_file*' }

[COMMENT '*table\_comment*']

[STORED AS *file\_format*]

[LOCATION '*hdfs\_path*']

**Note:**

To clone the structure of a table and transfer data into it in a single operation, use the CREATE TABLE AS SELECT syntax described in the next subsection.

When you clone the structure of an existing table using the CREATE TABLE ... LIKE syntax, the new table keeps the same file format as the original one, so you only need to specify the STORED AS clause if you want to use a different file format, or when specifying a view as the original table. (Creating a table "like" a view produces a text table by default.)

Although normally Impala cannot create an HBase table directly, Impala can clone the structure of an existing HBase table with the CREATE TABLE ... LIKE syntax, preserving the file format and metadata from the original table.

There are some exceptions to the ability to use CREATE TABLE ... LIKE with an Avro table. For example, you cannot use this technique for an Avro table that is specified with an Avro schema but no columns. When in doubt, check if a CREATE TABLE ... LIKE operation works in Hive; if not, it typically will not work in Impala either.

If the original table is partitioned, the new table inherits the same partition key columns. Because the new table is initially empty, it does not inherit the actual partitions that exist in the original one. To create partitions in the new table, insert data or issue ALTER TABLE ... ADD PARTITION statements.

Prior to Impala 1.4.0, it was not possible to use the CREATE TABLE LIKE *view\_name* syntax. In Impala 1.4.0 and higher, you can create a table with the same column definitions as a view using the CREATE TABLE LIKEtechnique. Although CREATE TABLE LIKE normally inherits the file format of the original table, a view has no underlying file format, so CREATE TABLE LIKE *view\_name* produces a text table by default. To specify a different file format, include a STORED AS *file\_format* clause at the end of the CREATE TABLE LIKE statement.

Because CREATE TABLE ... LIKE only manipulates table metadata, not the physical data of the table, issue INSERT INTO TABLE statements afterward to copy any data from the original table into the new one, optionally converting the data to a new file format. (For some file formats, Impala can do a CREATE TABLE ... LIKE to create the table, but Impala cannot insert data in that file format; in these cases, you must load the data in Hive. See [How Impala Works with Hadoop File Formats](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_file_formats.html#file_formats) for details.)

**CREATE TABLE AS SELECT:**

The CREATE TABLE AS SELECT syntax is a shorthand notation to create a table based on column definitions from another table, and copy data from the source table to the destination table without issuing any separate INSERT statement. This idiom is so popular that it has its own acronym, "CTAS".

The following examples show how to copy data from a source table T1 to a variety of destinations tables, applying various transformations to the table properties, table layout, or the data itself as part of the operation:

-- Sample table to be the source of CTAS operations.

CREATE TABLE t1 (x INT, y STRING);

INSERT INTO t1 VALUES (1, 'one'), (2, 'two'), (3, 'three');

-- Clone all the columns and data from one table to another.

CREATE TABLE clone\_of\_t1 AS SELECT \* FROM t1;

+-------------------+

| summary |

+-------------------+

| Inserted 3 row(s) |

+-------------------+

-- Clone the columns and data, and convert the data to a different file format.

CREATE TABLE parquet\_version\_of\_t1 STORED AS PARQUET AS SELECT \* FROM t1;

+-------------------+

| summary |

+-------------------+

| Inserted 3 row(s) |

+-------------------+

-- Copy only some rows to the new table.

CREATE TABLE subset\_of\_t1 AS SELECT \* FROM t1 WHERE x >= 2;

+-------------------+

| summary |

+-------------------+

| Inserted 2 row(s) |

+-------------------+

-- Same idea as CREATE TABLE LIKE: clone table layout but do not copy any data.

CREATE TABLE empty\_clone\_of\_t1 AS SELECT \* FROM t1 WHERE 1=0;

+-------------------+

| summary |

+-------------------+

| Inserted 0 row(s) |

+-------------------+

-- Reorder and rename columns and transform the data.

CREATE TABLE t5 AS SELECT upper(y) AS s, x+1 AS a, 'Entirely new column' AS n FROM t1;

+-------------------+

| summary |

+-------------------+

| Inserted 3 row(s) |

+-------------------+

SELECT \* FROM t5;

+-------+---+---------------------+

| s | a | n |

+-------+---+---------------------+

| ONE | 2 | Entirely new column |

| TWO | 3 | Entirely new column |

| THREE | 4 | Entirely new column |

+-------+---+---------------------+

See [SELECT Statement](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_select.html#select) for details about query syntax for the SELECT portion of a CREATE TABLE AS SELECTstatement.

The newly created table inherits the column names that you select from the original table, which you can override by specifying column aliases in the query. Any column or table comments from the original table are not carried over to the new table.

**Note:** When using the STORED AS clause with a CREATE TABLE AS SELECT statement, the destination table must be a file format that Impala can write to: currently, text or Parquet. You cannot specify an Avro, SequenceFile, or RCFile table as the destination table for a CTAS operation.

Prior to CDH 5.7 / Impala 2.5, you could use a partitioned table as the source and copy data from it, but could not specify any partitioning clauses for the new table. In CDH 5.7 / Impala 2.5 and higher, you can now use the PARTITIONED BY clause with a CREATE TABLE AS SELECT statement. The following example demonstrates how you can copy data from an unpartitioned table in a CREATE TABLE AS SELECT operation, creating a new partitioned table in the process. The main syntax consideration is the column order in the PARTITIONED BYclause and the select list: the partition key columns must be listed last in the select list, in the same order as in the PARTITIONED BY clause. Therefore, in this case, the column order in the destination table is different from the source table. You also only specify the column names in the PARTITIONED BY clause, not the data types or column comments.

create table partitions\_no (year smallint, month tinyint, s string);

insert into partitions\_no values (2016, 1, 'January 2016'),

(2016, 2, 'February 2016'), (2016, 3, 'March 2016');

-- Prove that the source table is not partitioned.

show partitions partitions\_no;

ERROR: AnalysisException: Table is not partitioned: ctas\_partition\_by.partitions\_no

-- Create new table with partitions based on column values from source table.

**create table partitions\_yes partitioned by (year, month)**

**as select s, year, month from partitions\_no;**

+-------------------+

| summary |

+-------------------+

| Inserted 3 row(s) |

+-------------------+

-- Prove that the destination table is partitioned.

show partitions partitions\_yes;

+-------+-------+-------+--------+------+...

| year | month | #Rows | #Files | Size |...

+-------+-------+-------+--------+------+...

| 2016 | 1 | -1 | 1 | 13B |...

| 2016 | 2 | -1 | 1 | 14B |...

| 2016 | 3 | -1 | 1 | 11B |...

| Total | | -1 | 3 | 38B |...

+-------+-------+-------+--------+------+...

The most convenient layout for partitioned tables is with all the partition key columns at the end. The CTAS PARTITIONED BY syntax requires that column order in the select list, resulting in that same column order in the destination table.

describe partitions\_no;

+-------+----------+---------+

| name | type | comment |

+-------+----------+---------+

| year | smallint | |

| month | tinyint | |

| s | string | |

+-------+----------+---------+

-- The CTAS operation forced us to put the partition key columns last.

-- Having those columns last works better with idioms such as SELECT \*

-- for partitioned tables.

describe partitions\_yes;

+-------+----------+---------+

| name | type | comment |

+-------+----------+---------+

| s | string | |

| year | smallint | |

| month | tinyint | |

+-------+----------+---------+

Attempting to use a select list with the partition key columns not at the end results in an error due to a column name mismatch:

-- We expect this CTAS to fail because non-key column S

-- comes after key columns YEAR and MONTH in the select list.

create table partitions\_maybe partitioned by (year, month)

as select year, month, s from partitions\_no;

ERROR: AnalysisException: Partition column name mismatch: year != month

For example, the following statements show how you can clone all the data in a table, or a subset of the columns and/or rows, or reorder columns, rename them, or construct them out of expressions:

As part of a CTAS operation, you can convert the data to any file format that Impala can write (currently, TEXTFILE and PARQUET). You cannot specify the lower-level properties of a text table, such as the delimiter.

**Sorting considerations:** Although you can specify an ORDER BY clause in an INSERT ... SELECT statement, any ORDER BY clause is ignored and the results are not necessarily sorted. An INSERT ... SELECT operation potentially creates many different data files, prepared on different data nodes, and therefore the notion of the data being stored in sorted order is impractical.

**CREATE TABLE LIKE PARQUET:**

The variation CREATE TABLE ... LIKE PARQUET '*hdfs\_path\_of\_parquet\_file*' lets you skip the column definitions of the CREATE TABLE statement. The column names and data types are automatically configured based on the organization of the specified Parquet data file, which must already reside in HDFS. You can use a data file located outside the Impala database directories, or a file from an existing Impala Parquet table; either way, Impala only uses the column definitions from the file and does not use the HDFS location for the LOCATIONattribute of the new table. (Although you can also specify the enclosing directory with the LOCATIONattribute, to both use the same schema as the data file and point the Impala table at the associated directory for querying.)

The following considerations apply when you use the CREATE TABLE LIKE PARQUET technique:

* Any column comments from the original table are not preserved in the new table. Each column in the new table has a comment stating the low-level Parquet field type used to deduce the appropriate SQL column type.
* If you use a data file from a partitioned Impala table, any partition key columns from the original table are left out of the new table, because they are represented in HDFS directory names rather than stored in the data file. To preserve the partition information, repeat the same PARTITION clause as in the original CREATE TABLE statement.
* The file format of the new table defaults to text, as with other kinds of CREATE TABLE statements. To make the new table also use Parquet format, include the clause STORED AS PARQUET in the CREATE TABLE LIKE PARQUET statement.
* If the Parquet data file comes from an existing Impala table, currently, any TINYINT or SMALLINT columns are turned into INT columns in the new table. Internally, Parquet stores such values as 32-bit integers.
* When the destination table uses the Parquet file format, the CREATE TABLE AS SELECT and INSERT ... SELECTstatements always create at least one data file, even if the SELECT part of the statement does not match any rows. You can use such an empty Parquet data file as a template for subsequent CREATE TABLE LIKE PARQUET statements.

For more details about creating Parquet tables, and examples of the CREATE TABLE LIKE PARQUET syntax, see [Using the Parquet File Format with Impala Tables](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_parquet.html#parquet).

**Visibility and Metadata (TBLPROPERTIES and WITH SERDEPROPERTIES clauses):**

You can associate arbitrary items of metadata with a table by specifying the TBLPROPERTIES clause. This clause takes a comma-separated list of key-value pairs and stores those items in the metastore database. You can also change the table properties later with an ALTER TABLE statement. You can observe the table properties for different delimiter and escape characters using the DESCRIBE FORMATTED command, and change those settings for an existing table with ALTER TABLE ... SET TBLPROPERTIES.

You can also associate SerDes properties with the table by specifying key-value pairs through the WITH SERDEPROPERTIES clause. This metadata is not used by Impala, which has its own built-in serializer and deserializer for the file formats it supports. Particular property values might be needed for Hive compatibility with certain variations of file formats, particularly Avro.

Some DDL operations that interact with other Hadoop components require specifying particular values in the SERDEPROPERTIES or TBLPROPERTIES fields, such as creating an Avro table or an HBase table. (You typically create HBase tables in Hive, because they require additional clauses not currently available in Impala.)

To see the column definitions and column comments for an existing table, for example before issuing a CREATE TABLE ... LIKE or a CREATE TABLE ... AS SELECT statement, issue the statement DESCRIBE *table\_name*. To see even more detail, such as the location of data files and the values for clauses such as ROW FORMAT and STORED AS, issue the statement DESCRIBE FORMATTED *table\_name*. DESCRIBE FORMATTED is also needed to see any overall table comment (as opposed to individual column comments).

After creating a table, your impala-shell session or another impala-shell connected to the same node can immediately query that table. There might be a brief interval (one statestore heartbeat) before the table can be queried through a different Impala node. To make the CREATE TABLE statement return only when the table is recognized by all Impala nodes in the cluster, enable the SYNC\_DDL query option.

**HDFS caching (CACHED IN clause):**

If you specify the CACHED IN clause, any existing or future data files in the table directory or the partition subdirectories are designated to be loaded into memory with the HDFS caching mechanism. See [Using HDFS Caching with Impala (CDH 5.1 or higher only)](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_perf_hdfs_caching.html#hdfs_caching) for details about using the HDFS caching feature.

In Impala 2.2 / CDH 5.4 and higher, the optional WITH REPLICATION clause for CREATE TABLE and ALTER TABLE lets you specify a **replication factor**, the number of hosts on which to cache the same data blocks. When Impala processes a cached data block, where the cache replication factor is greater than 1, Impala randomly selects a host that has a cached copy of that data block. This optimization avoids excessive CPU usage on a single host when the same cached data block is processed multiple times. Cloudera recommends specifying a value greater than or equal to the HDFS block replication factor.

**Column order**:

If you intend to use the table to hold data files produced by some external source, specify the columns in the same order as they appear in the data files.

If you intend to insert or copy data into the table through Impala, or if you have control over the way externally produced data files are arranged, use your judgment to specify columns in the most convenient order:

* If certain columns are often NULL, specify those columns last. You might produce data files that omit these trailing columns entirely. Impala automatically fills in the NULL values if so.
* If an unpartitioned table will be used as the source for an INSERT ... SELECT operation into a partitioned table, specify last in the unpartitioned table any columns that correspond to partition key columns in the partitioned table, and in the same order as the partition key columns are declared in the partitioned table. This technique lets you use INSERT ... SELECT \* when copying data to the partitioned table, rather than specifying each column name individually.
* If you specify columns in an order that you later discover is suboptimal, you can sometimes work around the problem without recreating the table. You can create a view that selects columns from the original table in a permuted order, then do a SELECT \* from the view. When inserting data into a table, you can specify a permuted order for the inserted columns to match the order in the destination table.

**Hive considerations:**

Impala queries can make use of metadata about the table and columns, such as the number of rows in a table or the number of different values in a column. Prior to Impala 1.2.2, to create this metadata, you issued the ANALYZE TABLE statement in Hive to gather this information, after creating the table and loading representative data into it. In Impala 1.2.2 and higher, the COMPUTE STATS statement produces these statistics within Impala, without needing to use Hive at all.

**HBase considerations:**

**Note:**

The Impala CREATE TABLE statement cannot create an HBase table, because it currently does not support the STORED BY clause needed for HBase tables. Create such tables in Hive, then query them through Impala. For information on using Impala with HBase tables, see [Using Impala to Query HBase Tables](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_hbase.html#impala_hbase).

**Amazon S3 considerations:**

To create a table where the data resides in the Amazon Simple Storage Service (S3), specify a s3a:// prefix LOCATION attribute pointing to the data files in S3.

In CDH 5.8 / Impala 2.6 and higher, you can use this special LOCATION syntax as part of a CREATE TABLE AS SELECT statement.

In CDH 5.8 / Impala 2.6 and higher, Impala DDL statements such as CREATE DATABASE, CREATE TABLE, DROP DATABASE CASCADE, DROP TABLE, and ALTER TABLE [ADD|DROP] PARTITION can create or remove folders as needed in the Amazon S3 system. Prior to CDH 5.8 / Impala 2.6, you had to create folders yourself and point Impala database, tables, or partitions at them, and manually remove folders when no longer needed. See [Using Impala with the Amazon S3 Filesystem](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_s3.html#s3) for details about reading and writing S3 data with Impala.

**Sorting considerations:** Although you can specify an ORDER BY clause in an INSERT ... SELECT statement, any ORDER BY clause is ignored and the results are not necessarily sorted. An INSERT ... SELECT operation potentially creates many different data files, prepared on different data nodes, and therefore the notion of the data being stored in sorted order is impractical.

**HDFS considerations:**

The CREATE TABLE statement for an internal table creates a directory in HDFS. The CREATE EXTERNAL TABLEstatement associates the table with an existing HDFS directory, and does not create any new directory in HDFS. To locate the HDFS data directory for a table, issue a DESCRIBE FORMATTED *table* statement. To examine the contents of that HDFS directory, use an OS command such as hdfs dfs -ls hdfs://*path*, either from the OS command line or through the shell or ! commands in impala-shell.

The CREATE TABLE AS SELECT syntax creates data files under the table data directory to hold any data copied by the INSERT portion of the statement. (Even if no data is copied, Impala might create one or more empty data files.)

**HDFS permissions:**

The user ID that the impalad daemon runs under, typically the impala user, must have both execute and write permission for the database directory where the table is being created.

**Security considerations:**

If these statements in your environment contain sensitive literal values such as credit card numbers or tax identifiers, Impala can redact this sensitive information when displaying the statements in log files and other administrative contexts. See [Sensitive Data Redaction](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/sg_redaction.html#log_redact) for details.

**Cancellation:** Certain multi-stage statements (CREATE TABLE AS SELECT and COMPUTE STATS) can be cancelled during some stages, when running INSERT or SELECT operations internally. To cancel this statement, use Ctrl-C from the impala-shell interpreter, the **Cancel** button from the **Watch** page in Hue, **Actions > Cancel** from the **Queries** list in Cloudera Manager, or **Cancel** from the list of in-flight queries (for a particular node) on the **Queries** tab in the Impala web UI (port 25000).

**Related information:**

[Overview of Impala Tables](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_tables.html#tables), [ALTER TABLE Statement](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_alter_table.html#alter_table), [DROP TABLE Statement](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_drop_table.html#drop_table), [Partitioning for Impala Tables](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_partitioning.html#partitioning), [Internal Tables](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_tables.html#internal_tables), [External Tables](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_tables.html#external_tables), [COMPUTE STATS Statement](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_compute_stats.html#compute_stats), [SYNC\_DDL Query Option](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_sync_ddl.html#sync_ddl), [SHOW TABLES Statement](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_show.html#show_tables), [SHOW CREATE TABLE Statement](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_show.html#show_create_table), [DESCRIBE Statement](https://www.cloudera.com/documentation/enterprise/5-8-x/topics/impala_describe.html#describe)