Impala使用

[简述 3](#_Toc13562)

[Impala是什么 3](#_Toc14073)

[Impala架构 3](#_Toc10794)

[支持的字段类型 4](#_Toc2107)

[SQL操作符 5](#_Toc7566)

[SQL语句 5](#_Toc2991)

[ALTER TABLE(DDL) 5](#_Toc16623)

[ALTER VIEW(DDL) 7](#_Toc17528)

[COMPUTE STATS(DDL) 7](#_Toc10614)

[CREATE DATABASE(DDL) 7](#_Toc24415)

[CREATE TABLE(DDL) 8](#_Toc13287)

[CREATE VIEW(DDL) 11](#_Toc9398)

[DESCRIBE 11](#_Toc21376)

[DROP DATABASE(DDL) 17](#_Toc7727)

[DROP TABLE(DDL) 17](#_Toc26864)

[DROP VIEW(DDL) 18](#_Toc14449)

[EXPLAIN 18](#_Toc27152)

[INSERT(DDL) 21](#_Toc9855)

[INVALIDATE METADATA 24](#_Toc10296)

[LOAD DATA(DML) 26](#_Toc15660)

[REFRESH 30](#_Toc12904)

[SELECT 30](#_Toc28448)

[SHOW 43](#_Toc31138)

[USE 53](#_Toc5709)

[内嵌函数 53](#_Toc20774)

[数学函数 53](#_Toc24000)

[类型转换函数 59](#_Toc29036)

[时间和日期函数 59](#_Toc30509)

[条件函数 67](#_Toc7182)

[字符串函数 70](#_Toc25001)

[特殊函数 74](#_Toc12302)

[聚合函数 75](#_Toc29552)

[appx\_median() 75](#_Toc14602)

[avg() 75](#_Toc9900)

[count() 76](#_Toc6501)

[max() 76](#_Toc27026)

[min() 76](#_Toc11887)

[ndv() 76](#_Toc29783)

[stddev(),stddev\_samp(),stddev\_pop() 77](#_Toc1808)

[sum() 77](#_Toc31683)

[variance(),variance\_samp(),variance\_pop(),var\_samp(),var\_pop() 77](#_Toc18562)

[Impala Shell 77](#_Toc3439)

[命令行选项 77](#_Toc31259)

[连接到impalad 79](#_Toc2805)

[运行命令 80](#_Toc30779)

[命令参考 80](#_Toc9114)

[查询参数设置 81](#_Toc4058)

[Impala 存储 95](#_Toc26675)

[文件格式选择 95](#_Toc3913)

[Text 96](#_Toc13147)

[Parquet 100](#_Toc15896)

# 简述

大家看到这个东西的时候，应该已经可以获取到一键安装impala集群的方法(impala集成包，这个安装包外面可没有的哦)，可以省去复杂的安装过程，使得掌握基本linux操作的同学也可以在不懂得Hadoop的情况下部署impala集群。作为笔者来说，我想分享一下这些日子使用impala对大数据进行分析的经验，目的是让大家可以了解impala是什么? 如果遇到impala适合的大数据场景，如何使用impala将它的性能最优的发挥出来!

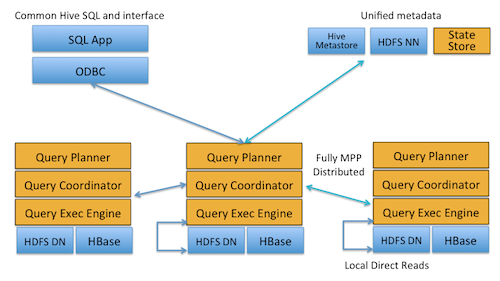
当然这个文档是本人通过项目积累得到的，基本包含了impala全部性能优化的方法，可以省去大家去看官方的原版文档，其实那些文档很难看懂。再来可能提供的一些sample的语法有点问题，我没把命令粘贴到impala中测试过，因为本文档的目的是提供给大家优化impala性能的方法思路，具体使用的话可以与我联系！

# Impala是什么

Impala是cloudera公司针对大数据提供的一个实时查询分析的利器，它运行在分布式文件系统HDFS之上，其借鉴了MPP（Massively Parallel Processing，大规模并行处理）并行数据库的思想，抛弃了MapReduce这个不太适合做SQL查询的范式，从而让Hadoop支持处理交互式的工作负载。不同于apache spark，它更接近传统关系型数据库的使用方式，在不同的场景情况下性能和spark各有千秋。

# Impala架构

这里简单的说下impala的大致情况，每个进程组件的作用,先看图:



这是官方的架构图啦，图中黄色部分就是impala掌握的区域，其中含三个进程组件:

**Impala Daemon:**它是存在于每个数据节点的进程，是最关键的进程，进程名为impalad，主要负责读写数据文件，接受来自外部的查询(impala-shell或者是odbc，jdbc)，并与集群中的其他impalad节点并行工作，并将查询结果返回给中心协调者节点。

我们可以向任何一个节点的impalad进程提交一个查询，提交查询的这个节点将作为这个查询的”协调者节点”为这个查询提供服务。其他节点的运算结果会被传输到协调者节点，协调者节点将最终运算结果返回。我们可以使用impala-shell加上IP地址和端口号的方式连接到任何一个impalad节点，对它发送查询命令，但在生产环境我们一般不这么做，因为当所有查询都发送到同一impalad节点，那么会导致负载不均衡，建议使用odbc/jdbc提供的round-robin方式将每个查询提交到不同的impalad节点上。

**Impala Statstore:**一个impala集群只需要一个statestored进程节点，它主要用来收集集群中所有impalad进程的健康情况，并不断的将每个节点的impalad健康状况发送给其他的impalad节点，当集群中有impalad节点发生类似网络通信故障，硬件故障，软件故障而导致该节点不可用时，statestore会将这一信息及时传达到所有其它的impalad节点，当有新的查询请求时，impalad进程将不会把查询请求发送到不可用的节点上。

**Impala catalog:**当impala集群中执行的sql语句会引起元数据变化时，catalog服务负责将这些变化推送到所有其它的impalad节点上。Catalog的进程名为catalogd，一个impala集群只需要一个catalogd进程，因为所有的请求都是通过statestored进程发送过来，所以建议将statestored进程和catalogd进程部署在同一节点。

Impala的所有组件就是以上三个啦，在一个impala集群中，statestored和catalog进程各部署一个，而在每个计算节点都需要部署impalad节点。当然这些东西包含但不限于hadoop，hive，mysql等都会在一键部署程序中自动安装，只需要通过修改配置文件来告诉部署程序需要安装多少台规模的impala集群。

# 支持的字段类型

* BIGINT
* BOOLEAN
* DOUBLE
* FLOAT
* [DECIMAL](http://www.cloudera.com/content/cloudera/en/documentation/cloudera-impala/latest/topics/impala_decimal.html)
* CHAR
* VARCHAR
* INT
* REAL
* SMALLINT
* STRING
* TIMESTAMP
* TINYINT

# SQL操作符

* BETWEEN操作符

示例: select \* from test where a between “XX” and “XXX”;

* 比较操作符，支持=,!=,<>,<=,<,>,>=

示例: select \* from test where a =”xx”;

* IN操作符

示例: select \* from test where a in (“xx”,”XXX”);

* IS NULL或IS NOT NULL

示例: select \* from test where a is not null;

* LIKE操作符

示例: select \* from test where a like ‘xx\_’;

select \* from test where a like ‘xx%’;

**注意哦:\_是匹配单个字符，而%表示匹配多个字符**

* REGEXP操作符(正则匹配)

示例: select \* from test where a regexp ‘a.\*’;

select \* from test where a regexp ‘^a.\*d$’;

# SQL语句

## ALTER TABLE(DDL)

语法:

ALTER TABLE [*old\_db\_name*.]*old\_table\_name* RENAME TO [*new\_db\_name*.]*new\_table\_name*

ALTER TABLE *name* ADD COLUMNS (*col\_spec*[, *col\_spec* ...])

ALTER TABLE *name* DROP [COLUMN] *column\_name*

ALTER TABLE *name* CHANGE *column\_namenew\_namenew\_type*

ALTER TABLE *name* REPLACE COLUMNS (*col\_spec*[, *col\_spec* ...])

ALTER TABLE *name* { ADD | DROP } PARTITION (*partition\_spec*)

ALTER TABLE *name* [PARTITION (*partition\_spec*)]

SET { FILEFORMAT *file\_format*

| LOCATION '*hdfs\_path\_of\_directory*'

| TBLPROPERTIES (*table\_properties*)

| SERDEPROPERTIES (*serde\_properties*) }

ALTER TABLE *name* [PARTITION (*partition\_spec*)] SET { CACHED IN '*pool\_name*' | UNCACHED }

*new\_name* ::= [*new\_database*.]*new\_table\_name*

*col\_spec* ::= *col\_nametype\_name*

*partition\_spec* ::= *partition\_col*=*constant\_value*

*table\_properties* ::= '*name*'='*value*'[, '*name*'='*value*' ...]

*serde\_properties* ::= '*name*'='*value*'[, '*name*'='*value*' ...]

*file\_format* ::= { PARQUET | TEXTFILE | RCFILE | SEQUENCEFILE | AVRO }

示例: alter table old\_ name rename to new\_name; --重命名表名

alter table table\_name set location ‘hdfs\_path\_of\_directory’; --修改表在hdfs上物理存储的位置

alter table table\_name add columns(column\_defs); --添加表字段

alter table table\_name replace columns(column\_defs); --覆盖表字段

alter table table\_name change columns(column\_defs); --修改表字段

alter table table\_name drop columns(column\_defs); --删除表字段

alter table part\_t add partition(year=2011);--增加分区子表，关于表分区技术后面详细说明

alter table part\_t drop partition(year=2011);--删除分区子表，关于表分区技术后面详细说明

## ALTER VIEW(DDL)

语法:

ALTER VIEW [database\_name.]view\_name AS select\_statement

ALTER VIEW [database\_name.]view\_name RENAME TO [database\_name.]view\_name

示例: create table t1 (x int, y int, s string);

create table t2 like t1;

create view v1 as select \* from t1;

alter view v1 as select \* from t2;

alter view v1 as select x, upper(s) s from t2;

## COMPUTE STATS(DDL)

语法:

COMPUTE STATS [db\_name.]table\_name

COMPUTE INCREMENTAL STATS [db\_name.]table\_name [PARTITION (partition\_spec)]

partition\_spec ::= partition\_col=constant\_value

示例: COMPUTE STATS; --收集所有表信息

COMPUTE STATS table\_name; --收集个别表信息

说明:这一语句完成对表,分区,列的数据量和数据分布信息收集。收集的相关信息会被存储在元数据库中。Impala在对查询进行优化时将使用到这些信息。如果Impala知道是大表还是小表，每列值的区分度大小，将能更好的判断对一个JOIN或者INSERT操作是否进行并行化操作，以提升性能。

## CREATE DATABASE(DDL)

语法:

CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database\_name[COMMENT 'database\_comment']

[LOCATION hdfs\_path];

示例: create database d1; --创建一个数据库

drop database d1; --删除一个数据库

## CREATE TABLE(DDL)

语法:

创建具有详细列定义的表(内部表)

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

[(col\_namedata\_type [COMMENT 'col\_comment'], ...)]

[COMMENT 'table\_comment']

[PARTITIONED BY (col\_namedata\_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']

列的定义来自数据文件(内部表)

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

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

[COMMENT 'table\_comment']

[PARTITIONED BY (col\_namedata\_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']

data\_type

: primitive\_type

克隆一张表 (包含数据，内部表)

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

[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']

AS

select\_statement

支持的数据类型，分割语法，存储格式

primitive\_type

: TINYINT

| SMALLINT

| INT

| BIGINT

| BOOLEAN

| FLOAT

| DOUBLE

| DECIMAL

| STRING

| CHAR

| VARCHAR

| TIMESTAMP

row\_format

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

[LINES TERMINATED BY 'char']

file\_format:

PARQUET

| TEXTFILE

| AVRO

| SEQUENCEFILE

| RCFILE

创建一张外部表(数据文件不属于impala自身)

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

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

克隆一张外部表(数据文件不属于impala自身)

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']

示例: --克隆t1表的结构和数据

create table clone\_of\_t1 as select \* from t1;

--克隆表结构不包含数据

create table empty\_of\_t1 as select \* from t1 where 1=0;

--克隆部分数据

create table subset\_of\_t1 AS SELECT \* FROM t1 WHERE x > 100 AND y LIKE 'A%';

create table summary\_of\_t1 AS SELECT c1, sum(c2) AS total, avg(c2) AS average FROM t1 group by c2;

--存储为列存模式

create table parquet\_version\_of\_t1 STORED AS PARQUET AS SELECT \* FROM t1;

--创建一张略不同于原始表的新表

create table some\_columns\_from\_t1 AS SELECT c1, c3, c5 FROM t1;

create table reordered\_columns\_from\_t1 AS SELECT c4, c3, c1, c2 FROM t1;

create table synthesized\_columns AS SELECT upper(c1) AS all\_caps, c2+c3 AS total, "California" AS state FROM t1;

## CREATE VIEW(DDL)

语法

CREATE VIEW [IF NOT EXISTS] view\_name [(column\_list)]

AS select\_statement

示例:--创建视图的各种方式

create view v1 as select \* from t1;

create view v2 as select c1, c3, c7 from t1;

create view v3 as select distinct c1, c3, c7 from t1 where c1 is not null and c5 > 0;

create view v4 as select c4 as last\_name, c6 as address, c2 as birth\_date from t1;

create view v5 as select c1, cast(c3 as string) c3, concat(c4,c5) c5, trim(c6) c6, "Constant" c8 from t1;

create view v6 as select t1.c1, t2.c2 from t1 join t2 on t1.id = t2.id;

## DESCRIBE

语法:

DESCRIBE [FORMATTED] [db\_name.]table\_name

示例:

[localhost:21000] > create table t1 (x int, y int, s string);

Query: create table t1 (x int, y int, s string)

[localhost:21000] > describe t1;

Query: describe t1

Query finished, fetching results ...

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

| name | type | comment |

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

| x | int | |

| y | int | |

| s | string | |

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

Returned 3 row(s) in 0.13s

[localhost:21000] > describe formatted t1;

Query: describe formatted t1

Query finished, fetching results ...

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

| name | type | comment |

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

| # col\_name | data\_type | comment |

| | NULL | NULL |

| x | int | None |

| y | int | None |

| s | string | None |

| | NULL | NULL |

| # Detailed Table Information | NULL | NULL |

| Database: | describe\_formatted | NULL |

| Owner: | cloudera | NULL |

| CreateTime: | Mon Jul 22 17:03:16 EDT 2013 | NULL |

| LastAccessTime: | UNKNOWN | NULL |

| Protect Mode: | None | NULL |

| Retention: | 0 | NULL |

| Location: | hdfs://127.0.0.1:8020/user/hive/warehouse/describe\_formatted.db/t1 | NULL |

| Table Type: | MANAGED\_TABLE | NULL |

| Table Parameters: | NULL | NULL |

| | transient\_lastDdlTime | 1374526996 |

| | NULL | NULL |

| # Storage Information | NULL | NULL |

| SerDe Library: | org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe | NULL |

| InputFormat: | org.apache.hadoop.mapred.TextInputFormat | NULL |

| OutputFormat: | org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat | NULL |

| Compressed: | No | NULL |

| Num Buckets: | 0 | NULL |

| Bucket Columns: | [] | NULL |

| Sort Columns: | [] | NULL |

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

Returned 26 row(s) in 0.03s

[localhost:21000] > create view v1 as select x, upper(s) from t1;

Query: create view v1 as select x, upper(s) from t1

[localhost:21000] > describe v1;

Query: describe v1

Query finished, fetching results ...

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

| name | type | comment |

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

| x | int | |

| \_c1 | string | |

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

Returned 2 row(s) in 0.10s

[localhost:21000] > describe formatted v1;

Query: describe formatted v1

Query finished, fetching results ...

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

| name | type | comment |

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

| # col\_name | data\_type | comment |

| | NULL | NULL |

| x | int | None |

| \_c1 | string | None |

| | NULL | NULL |

| # Detailed Table Information | NULL | NULL |

| Database: | describe\_formatted | NULL |

| Owner: | cloudera | NULL |

| CreateTime: | Mon Jul 22 16:56:38 EDT 2013 | NULL |

| LastAccessTime: | UNKNOWN | NULL |

| Protect Mode: | None | NULL |

| Retention: | 0 | NULL |

| Table Type: | VIRTUAL\_VIEW | NULL |

| Table Parameters: | NULL | NULL |

| | transient\_lastDdlTime | 1374526598 |

| | NULL | NULL |

| # Storage Information | NULL | NULL |

| SerDe Library: | null | NULL |

| InputFormat: | null | NULL |

| OutputFormat: | null | NULL |

| Compressed: | No | NULL |

| Num Buckets: | 0 | NULL |

| Bucket Columns: | [] | NULL |

| Sort Columns: | [] | NULL |

| | NULL | NULL |

| # View Information | NULL | NULL |

| View Original Text: | SELECT x, upper(s) FROM t1 | NULL |

| View Expanded Text: | SELECT x, upper(s) FROM t1 | NULL |

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

Returned 28 row(s) in 0.03s

[localhost:21000] > create external table t2 (x int, y int, s string) stored as parquet location '/user/cloudera/sample\_data';

[localhost:21000] > describe formatted t2;

Query: describe formatted t2

Query finished, fetching results ...

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

| name | type | comment |

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

| # col\_name | data\_type | comment |

| | NULL | NULL |

| x | int | None |

| y | int | None |

| s | string | None |

| | NULL | NULL |

| # Detailed Table Information | NULL | NULL |

| Database: | describe\_formatted | NULL |

| Owner: | cloudera | NULL |

| CreateTime: | Mon Jul 22 17:01:47 EDT 2013 | NULL |

| LastAccessTime: | UNKNOWN | NULL |

| Protect Mode: | None | NULL |

| Retention: | 0 | NULL |

| Location: | hdfs://127.0.0.1:8020/user/cloudera/sample\_data | NULL |

| Table Type: | EXTERNAL\_TABLE | NULL |

| Table Parameters: | NULL | NULL |

| | EXTERNAL | TRUE |

| | transient\_lastDdlTime | 1374526907 |

| | NULL | NULL |

| # Storage Information | NULL | NULL |

| SerDe Library: | org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe | NULL |

| InputFormat: | com.cloudera.impala.hive.serde.ParquetInputFormat | NULL |

| OutputFormat: | com.cloudera.impala.hive.serde.ParquetOutputFormat | NULL |

| Compressed: | No | NULL |

| Num Buckets: | 0 | NULL |

| Bucket Columns: | [] | NULL |

| Sort Columns: | [] | NULL |

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

Returned 27 row(s) in 0.17s

这个语句用于显示表中的列名，列数据类型等元数据信息。加上FORMATED参数显示更详细的信息。

## DROP DATABASE(DDL)

语法:

DROP (DATABASE|SCHEMA) [IF EXISTS] database\_name;

示例: --删除数据库

drop database db;

## DROP TABLE(DDL)

语法:

DROP TABLE [IF EXISTS] [db\_name.]table\_name

示例: --删除表

drop table unimportant;

## DROP VIEW(DDL)

语法:

DROP VIEW [IF EXISTS] [db\_name.]view\_name

示例: --删除视图

drop view few\_rows\_from\_t1;

## EXPLAIN

语法:

EXPLAIN { select\_query | ctas\_stmt | insert\_stmt }

该语句返回一个语句执行计划。它从底层显示impala如何读取数据，如何在各个节点之间协调工作，传输中间结果，并获得最终结果的全过程。EXPLAIN后面接一个完整的SELECT语句。

select\_query是一个select语句; insert\_stmt是一个向表中插入或者重写数据的INSERT语句，具体可以使用INSERT…SELECT或者INSERT…VALUES语法。ctas\_stmt语句是一个使用了AS SELECT子句的CREATE TABLE语句。

**使用注意点：**

根据执行计划，我们可以判断这个语句的执行效率，如果发现语句效率过低，可以调整语句本身，或者调整表结构。例如，我们通过改变WHERE的查询条件，对JOIN查询使用hints引入子查询，改变表的连接顺序，改变表的分区，搜集表及列统计信息或者其他方式来对语句进行性能优化。

自低向上读取执行计划:

1. 执行计划的最后一部分内容是一些底层的详细信息。这些信息可以显示的是像需要被读取的数据量，根据这些信息我们可以判读分区的策略是否有效，还可以基于表的总大小及每个节点要读取的大小来判读这个语句需要花费多长时间。
2. 接下来我们可以看到哪些操作被分布到每个Impala节点上并行执行。
3. 从宏观上，我们还可以看到中间结果是如何组合起来在节点之间传输的。

我们一定要时刻记住，Impala是用来优化数据仓库中超大表的全表扫描的。数据的结构和分布完全与OLTP系统不同，对一个大表的全表扫描是常态操作。当然，我们也可以通过减少需要全表扫描的数据的数量级来优化查询。比如，我们只扫描查询中需要的数据所在的分区，而不是扫描整张表来将查询的时间从分钟级提高到秒级。

**扩展的EXPLAIN输出:**

对于复杂查询的性能调整和容量规划，我们可以让EXPLAIN语句输出更详细的信息。在impala-shell中，执行SET EXPLAIN\_LEVEL=level，这里的level可以是0到3的整数，分别代表minimal，standard，extended，和verbose。

示例:

--标准的EXPLAIN输出从物理级别到逻辑级别做了完整的显示。查询语句开始于扫描特定数量的数据；每个节点执行一个聚集操作(在本地节点执行count(\*)操作生成中间结果；中间结果被传输回协调节点)；最后，将中间结果汇总为最终结果。

[impalad-host:21000] > explain select count(\*) from customer\_address;

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

| Explain String |

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

| Estimated Per-Host Requirements: Memory=42.00MB VCores=1 |

| |

| 03:AGGREGATE [MERGE FINALIZE] |

| | output: sum(count(\*)) |

| | |

| 02:EXCHANGE [PARTITION=UNPARTITIONED] |

| | |

| 01:AGGREGATE |

| | output: count(\*) |

| | |

| 00:SCAN HDFS [default.customer\_address] |

| partitions=1/1 size=5.25MB |

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

--扩展EXPLAIN输出依据COMPUTE STATS搜集的统计信息显示了更为准确，详细的输出。默认的扩展输出中，像数据大小，分布情况等信息都显示为”unavailable”。Impala可以确定裸数据文件的大小，但是没有进行额外的分析之前无法确定记录数，每列非重复值的个数等信息。COMPUTE STATS语句执行分析后，EXPLAIN能够显示出为优化查询提供帮助的更为详细的信息。

[localhost:21000] > set explain\_level=extended;

EXPLAIN\_LEVEL set to extended

[localhost:21000] > explain select x from t1;

[localhost:21000] > explain select x from t1;

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

| Explain String |

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

| Estimated Per-Host Requirements: Memory=32.00MB VCores=1 |

| |

| 01:EXCHANGE [PARTITION=UNPARTITIONED] |

| | hosts=1 per-host-mem=unavailable |

**| | tuple-ids=0 row-size=4B cardinality=unavailable |**

| | |

| 00:SCAN HDFS [default.t2, PARTITION=RANDOM] |

| partitions=1/1 size=36B |

**| table stats: unavailable |**

**| column stats: unavailable |**

| hosts=1 per-host-mem=32.00MB |

**| tuple-ids=0 row-size=4B cardinality=unavailable |**

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

[localhost:21000] > compute stats t1;

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

| summary |

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

| Updated 1 partition(s) and 1 column(s). |

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

[localhost:21000] > explain select x from t1;

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

| Explain String |

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

| Estimated Per-Host Requirements: Memory=64.00MB VCores=1 |

| |

| 01:EXCHANGE [PARTITION=UNPARTITIONED] |

| | hosts=1 per-host-mem=unavailable |

| | tuple-ids=0 row-size=4B cardinality=0 |

| | |

| 00:SCAN HDFS [default.t1, PARTITION=RANDOM] |

| partitions=1/1 size=36B |

**| table stats: 0 rows total |**

**| column stats: all |**

| hosts=1 per-host-mem=64.00MB |

**| tuple-ids=0 row-size=4B cardinality=0 |**

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

## INSERT(DDL)

语法:

[with\_clause]

INSERT { INTO | OVERWRITE } [TABLE] table\_name

[(column\_list)]

[ PARTITION (partition\_clause)]

{

[hint\_clause] select\_statement

| VALUES (value [, value ...]) [, (value [, value ...]) ...]

}

partition\_clause ::= col\_name [= constant] [, col\_name [= constant] ...]

hint\_clause ::= [SHUFFLE] | [NOSHUFFLE] (Note: the square brackets are part of the syntax.)

* 通过INSERT INTO向表中插入数据
* 通过INSERT OVERWRITE替换表中数据
* 从其他表中使用SELECT拷贝数据
* 在INSERT\_SELECT向一个Parquet分区表插入数据的语句中，SELECT关键字之前的hint可以很好的优化插入操作。这个hint包括[SHUFFLE][NOSHUFFLE]。由于向Parquet分区表插入数据会引起同时向HDFS写多个文件，所有INSERT操作是一个资源敏感的操作。

示例:

创建测试数据库及不同存储格式的测试表

CREATE DATABASE IF NOT EXISTS file\_formats;

USE file\_formats;

DROP TABLE IF EXISTS text\_table;

CREATE TABLE text\_table

( id INT, col\_1 BOOLEAN, col\_2 DOUBLE, col\_3 TIMESTAMP )

STORED AS TEXTFILE;

DROP TABLE IF EXISTS parquet\_table;

CREATE TABLE parquet\_table

( id INT, col\_1 BOOLEAN, col\_2 DOUBLE, col\_3 TIMESTAMP )

STORED AS PARQUET;

插入行存表数据，每条新数据被插入到现有数据之后，运行2次insert之后，共插入10行数据

[localhost:21000] > insert into table text\_table select \* from default.tab1;

Inserted 5 rows in 0.41s

[localhost:21000] > insert into table text\_table select \* from default.tab1;

Inserted 5 rows in 0.46s

[localhost:21000] > select count(\*) from text\_table;

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

| count(\*) |

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

| 10 |

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

Returned 1 row(s) in 0.26s

覆盖现有表中的数据

[localhost:21000] > insert into table parquet\_table select \* from default.tab1;

Inserted 5 rows in 0.35s

[localhost:21000] > insert overwrite table parquet\_table select \* from default.tab1 limit 3;

Inserted 3 rows in 0.43s

[localhost:21000] > select count(\*) from parquet\_table;

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

| count(\*) |

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

| 3 |

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

Returned 1 row(s) in 0.43s

VALUES子句，可以通过为所有列指定常量值插入一个或多个行。表达式的数量，类型和顺序必须与表定义相匹配

create table val\_test\_1 (c1 int, c2 float, c3 string, c4 boolean, c5 timestamp);

insert into val\_test\_1 values (100, 99.9/10, 'abc', true, now());

create table val\_test\_2 (id int, token string);

insert overwrite val\_test\_2 values (1, 'a'), (2, 'b'), (-1,'xyzzy');

向分区表插入数据

create table t1 (i int) **partitioned by (x int, y string)**;

insert into t1 **partition(x=10, y='a')** select c1 from some\_other\_table;

以下是各类插入方法的示例

create table t1 (c1 int, c2 int);

create table t2 like t1;

insert into t2 select \* from t1;

insert into t2 select c1, c2 from t1;

insert into t2 (c1) select c1 from t1;

insert into t2 (c2, c1) select c1, c2 from t1;

alter table t2 replace columns (x int, y int);

insert into t2 (y) select c1 from t1;

create table pt1 (x int, y int) partitioned by (z int);

insert into pt1 (x) partition (z=5) select c1 from t1;

insert into pt1 (x,z) select x, z from t2;

一些向分区表中插入数据的示例

create table source (x int, y int, year int, month int, day int);

create table destination (x int, y int) partitioned by (year int, month int, day int);

---向source表中填充点数据--

insert overwrite destination partition (year, month, day) select \* from source where year=2014;

insert overwrite destination partition (year, month, day) select \* from source;

create table source\_2014 (x int, y int, month, day);

---向source\_2014表中填充点数据--

insert overwrite destination partition (year=2014, month, day) select \* from source\_2014;

## INVALIDATE METADATA

语法:

INVALIDATE METADATA [[db\_name.]table\_name]

该语句用于将表的元数据标记为过期。在一张表通过HIVE命令创建后，我们执行该命令后，才能让Impala对该表正常访问。Impala在对一个元数据标记为过期的表进行查询之前，会自动重新加载最新的相关元数据信息。同时，与REFRESH相比，这个操作是一个成本极高的操作，因为REFRESH是对元数据进行增量更新。在一般情况下，比如向已存在的表添加数据文件之类的操作，建议优先使用REFRESH。

为了精确的响应查询，Impala客户端必须能够直接访问数据库和表最新的元数据信息。因此，如果Impala更新某些Impala和Hive共享的对象的定义信息，由Impala缓存过的原来的元数据信息必须被更新为最新的。当然，这也并不是说要更新所有的元数据信息，Impala只会更新需要的元数据信息。

**INVALIDATE METADATA和REFRESH对比:**

INVALIDATE METADATA在执行查询需要访问元数据信息时加载相关表的所有元数据信息。这个操作尤其是对那些拥有很多分区的大表来说，成本相当高。REFRESH加载元数据速度更快，因为只加载新增加的数据文件的块位置数据，所以成本也相对比较低。如果数据被像HDFS balancer这样的外部方式更改，使用INVALIDATE METADATA可以避免降低本地读取的性能损失。

使用INVALIDATE METADATA命令，默认情况下，缓存的所有表的元数据信息都会被清除。如果语法中指定了表名，那么将会清除指定表的元数据信息。即使是对单张，INVALIDATE METADATA也比REFRESH成本更高。在向表中添加数据文件后，优先使用REFRESH操作。

在以下情况下，impalad实例会请求元数据更新操作:

* 元数据发生变化
* 其他impalad实例或者Hive修改了元数据
* Impala-shell或者用过ODBC方式修改了数据库

如果在某个节点上执行了ALTER TABLE,INSER或者其他的修改语句之后，在同一节点上执行查询时，不需要进行元数据的更新操作。

数据库或者表的元数据修改包括:

* 通过Hive执行ALTER,CREATE,DROP或INSERT操作
* 通过impalad执行CREATE table，ALTER TABLE或INSERT操作

INVALIDATE METADATA语句会强制表的元数据信息过期，这样，在下次表被引用的时候元数据信息就会重新加载。对于一个超大表进行INVALIDATE METADATA将会消耗大量的时间。如果使用REFRESH，可能可以避免不可预知的延迟。

示例:

示例演示了在Hive中创建一个新的数据库和一张新的表，在Impala中使用表全名进行INVALIDATE METADATA操作，之后可以发现Impala就可以识别新的数据库和新的表了。

$ hive

hive> create database new\_db\_from\_hive;

OK

Time taken: 4.118 seconds

hive> create table new\_db\_from\_hive.new\_table\_from\_hive (x int);

OK

Time taken: 0.618 seconds

hive> quit;

$ impala-shell

[localhost:21000] > show databases like 'new\*';

[localhost:21000] > refresh new\_db\_from\_hive.new\_table\_from\_hive;

ERROR: AnalysisException: Database does not exist: new\_db\_from\_hive

[localhost:21000] > invalidate metadata new\_db\_from\_hive.new\_table\_from\_hive;

[localhost:21000] > show databases like 'new\*';

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

| name |

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

| new\_db\_from\_hive |

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

[localhost:21000] > show tables in new\_db\_from\_hive;

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

| name |

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

| new\_table\_from\_hive |

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

## LOAD DATA(DML)

语法:

LOAD DATA INPATH 'hdfs\_file\_or\_directory\_path' [OVERWRITE] INTO TABLE tablename [PARTITION (partcol1=val1, partcol2=val2 ...)]

* 加载的数据文件是被移动而不是被拷贝到Impala的数据目录
* 我们可以指定移动HDFS路径中的单个文件或者所有文件。我们无法使用通配符只移动某些需要的文件。在加载一个目录下所有的数据文件时，需要保证数据文件位于该目录之下，而不是嵌套的子目录下。
* 目前，Impala的LOAD DATA只能从HDFS加载数据文件，而不是从本地文件系统加载数据文件。目前不支持Hive中的LOCAL关键字。我们必须制定一个路径，而不是形如hdfs://的URL。
* 为了保证加载的速度，加载过程对错误的检测很有限。如果加载的数据文件的文件格式错误，源文件中的列数与目标表中的列数不一致，或者其他类型的不匹配问题，Impala在LOAD DATA时不会报任何错误。目前，Impala唯一做的检查就是避免未压缩的文本文件和LZO方式压缩的文本文件出现在同一张表中。
* 当我们在LOAD DATA中指定一个HDFS目录时，目录中的任何隐藏文件(文件的名字以.开头的文件)将会被忽略。
* 除非有命名冲突，否则加载到目标位置的数据文件与源数据文件名称保持一致。
* 通过LOAD DATA这种方式将HDFS中的文件加载到Impala中，简化了加载过程，同时我们也不必关注表对应的数据文件究竟在哪个HDFS目录中，如何分布的，分别对应哪个数据库，哪张表。我们可以通过DESCRIBE FORMATTED table\_name语句，快速的了解数据文件和表的对应关系。
* PARTITION子句对于增加新的数据分区特别方便。当我们生成了按时间段，或者地理位置，或者其他特定含义的新的数据时，可以将数据直接加载到Impala数据目录，将其作为Impala的一个或者多个分区。对于分区表，加载数据时，必须使用常量值指定要将数据加载到哪个分区。

示例：

首先我们使用python脚本生成数字串（每行一个），并存储到HDFS上

$ random\_strings.py 1000 | hdfs dfs -put - /user/cloudera/thousand\_strings.txt

$ random\_strings.py 100 | hdfs dfs -put - /user/cloudera/hundred\_strings.txt

$ random\_strings.py 10 | hdfs dfs -put - /user/cloudera/ten\_strings.txt

然后，我们创建表，并将数据加载进去。大家需要留意，除非指定了STORE AS子句，Impala默认使用TEXTFILE做默认的数据文件格式，使用Ctrl-A(16进制的01)作为分隔符。本例中使用的是只有一个列的表，所以分隔符不太重要。对于大型的ETL作业，我们需要使用像Parquet或者Avro这样的数据文件格式，并将数据加载到相应文件格式的Impala表中。

[localhost:21000] > create table t1 (s string);

[localhost:21000] > load data inpath '/user/cloudera/thousand\_strings.txt' into table t1;

Query finished, fetching results ...

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

| summary |

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

| Loaded 1 file(s). Total files in destination location: 1 |

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

Returned 1 row(s) in 0.61s

[kilo2-202-961.cs1cloud.internal:21000] > select count(\*) from t1;

Query finished, fetching results ...

+------+

| \_c0 |

+------+

| 1000 |

+------+

Returned 1 row(s) in 0.67s

[localhost:21000] > load data inpath '/user/cloudera/thousand\_strings.txt' into table t1;

ERROR: AnalysisException: INPATH location '/user/cloudera/thousand\_strings.txt' does not exist.

在数据加载进表之后，通过上例中的最后一条语句可以看到，数据文件已经被移动而不是复制到了Impala数据目录中。

下面的示例中，我们可以看到数据文件被移动到Impala数据目录中后，保持了原来的数据文件名称。

$ hdfs dfs -ls /user/hive/warehouse/load\_data\_testing.db/t1

Found 1 items

-rw-r--r-- 1 cloudera cloudera 13926 2013-06-26 15:40 /user/hive/warehouse/load\_data\_testing.db/t1/thousand\_strings.txt

下面的示例将演示在INTO TABLE和OVERWRITE TABLE直间的不同之处。表原始数据有1000行。在执行了带有INTO TABLE子句的LOAD DATA语句后，表中的数据多了100行，总共1100行。在执行了带有OVERWRITE TABLE子句的LOAD DATA语句加载了10条数据后，原有的数据丢失了，表中总共包含10条数据。

[localhost:21000] > load data inpath '/user/cloudera/hundred\_strings.txt' into table t1;

Query finished, fetching results ...

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

| summary |

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

| Loaded 1 file(s). Total files in destination location: 2 |

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

Returned 1 row(s) in 0.24s

[localhost:21000] > select count(\*) from t1;

Query finished, fetching results ...

+------+

| \_c0 |

+------+

| 1100 |

+------+

Returned 1 row(s) in 0.55s

[localhost:21000] > load data inpath '/user/cloudera/ten\_strings.txt' overwrite into table t1;

Query finished, fetching results ...

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

| summary |

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

| Loaded 1 file(s). Total files in destination location: 1 |

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

Returned 1 row(s) in 0.26s

[localhost:21000] > select count(\*) from t1;

Query finished, fetching results ...

+-----+

| \_c0 |

+-----+

| 10 |

+-----+

Returned 1 row(s) in 0.62s

## REFRESH

语法:

REFRESH [*db\_name*.]*table\_name*

为了能够准确的对数据进行查询，我们通过impala-shell，JDBC或者ODBC所连接到的节点将充当协调者节点，它负责获取将要查询的数据库和表的最新元数据信息。

使用REFRESH语句加载最新的元数据信息，特定表的块位置信息等的操作适用于如下场景:

* 在加载了新的数据文件到表对应的HDFS目录之后。
* 在Hive中执行了ALTER TABLE，INSERT，LOAD DATA或者其他对表的修改的SQL语句之后。

我们只需要在协调者节点上执行REFRESH语句即可。协调者节点负责依据本节点上的元数据信息将任务分发到集群所有的Impala节点上，而不依赖于其他节点的元数据信息。

REFRESH负责从元数据库中重新加载表的元数据信息，而且是增量的加载新的数据文件的块位置信息。这是一个可以对单表进行的低负载操作，通常在向表添加了新的数据文件之后，将使用这个命令更新元数据信息。

这个命令只更新指定表的元数据，这张表不管是Impala使用CREATE TABLE方式创建的表，还是之前使用INVALIDATE METADATA语句加载过全量的元数据信息，都必须保证这张表可以被Impala识别。

REFRESH与INVALIDATE METADATA的区别已在介绍INVALIDATE METADATA部分提及，在此不在说明。

示例:

这里演示的是在手动向Impala数据目录添加了数据文件之后，使用REFRESH刷新元数据信息的例子。

[impalad-host:21000] > refresh t1;

[impalad-host:21000] > refresh t2;

[impalad-host:21000] > select \* from t1;

...

[impalad-host:21000] > select \* from t2;

...

## SELECT

SELECT语句执行查询，从一张或者多张表获取数据，并返回由行列组成的结果集。在Impala的INSERT语句中，通常会包含一个SELECT语句用于定义要从源表中拷贝哪些数据。

### 1.Impala的SELECT查询支持

* SQL数据类型：BOOLEAN，TINYINT，SMALLINT，INT，BIGINT，FLOAT，DOUBLE，TIMESTAMP，STRING，VARCHAR，CHAR。
* 在SELECT关键字前指定WITH子句中可以定义一个子查询，在SELECT中可以引用这个子查询。
* DISTINCT子句。
* 在FROM子句中使用子查询。
* WHERE，GROUP BY，HAVING子句。
* ORDER BY在排序的同时也可以使用LIMIT关键字。
* 包括LEFT，RIGHT，SEMI，FULL，OUTER等各种JOIN连接查询。在Impala1.2.2及以上版本支持CROSS JOIN连接。在性能调整期间，我们可以通过在SELECT关键字后加STRAIGHT\_JOIN覆盖原有的连接顺序。
* UNION ALL。
* LIMIT。
* 外部表。
* 像>,<,=的关系操作符。
* 像+,-的数学运算符。
* 逻辑/布尔运算符AND,OR,NOT等，但不支持&&,||,!等。
* 像COUNT,SUM,CAST,LIKE,IN,BETWEEN,COALESCE等通过的SQL内嵌函数。

### 2.JOIN

连接查询对两个或者多个表的数据进行某种形式的合并，并返回包含特定要求的结果集。

语法:

SELECT *select\_list* FROM

*table\_or\_subquery1* [INNER] JOIN *table\_or\_subquery2* |

*table\_or\_subquery1* {LEFT [OUTER] | RIGHT [OUTER] | FULL [OUTER]} JOIN *table\_or\_subquery2* |

*table\_or\_subquery1* {LEFT | RIGHT} SEMI JOIN *table\_or\_subquery2* |

*table\_or\_subquery1* {LEFT | RIGHT} ANTI JOIN *table\_or\_subquery2* |

[ ON *col1* = *col2* [AND *col3* = *col4* ...] |

USING (*col1* [, *col2* ...]) ]

[*other\_join\_clause* ...]

[ WHERE *where\_clauses* ]

SELECT *select\_list* FROM

*table\_or\_subquery1*, *table\_or\_subquery2* [, *table\_or\_subquery3* ...]

[*other\_join\_clause* ...]

WHERE

*col1* = *col2* [AND *col3* = *col4* ...]

SELECT *select\_list* FROM

*table\_or\_subquery1* CROSS JOIN *table\_or\_subquery2*

[*other\_join\_clause* ...]

[ WHERE *where\_clauses* ]

### 3.SQL-92和SQL-89连接

查询中显式指定JOIN关键字的是SQL-92风格的JOIN。使用ON或者USING关键字指定哪些列作为连接键。

SELECT t1.c1, t2.c2 FROM **t1 JOIN t2**

**ON t1.id = t2.id and t1.type\_flag = t2.type\_flag**

WHERE t1.c1 > 100;

SELECT t1.c1, t2.c2 FROM **t1 JOIN t2**

**USING (id, type\_flag)**

WHERE t1.c1 > 100;

即使两个表关联列的列名不同，也可以使用ON关键字进行连接。而USING是ON的缩写形式，专门用在各表关联列具有相同名称的情况下。当然，我们也可以不使用USING,ON关键字，直接使用WHERE条件进行关联列的等值比较，但是这种书写方式会将连接比较和过滤混淆，降低了可读性，不易维护。

使用逗号分隔参与连接的各表或者子查询的查询是SQL-89风格的JOIN。在这种查询中，使用的就是WHERE条件对关联列进行等值比较。这种语法很容易使用，但是也很容易由于删除某些WHERE子句导致连接无法正常工作。

SELECT t1.c1, t2.c2 FROM **t1, t2**

WHERE

**t1.id = t2.id AND t1.type\_flag = t2.type\_flag**

AND t1.c1 > 100;

### 4.自连接

Impala支持自连接，允许对某张表的不同列进行关联查询以展示数据之间的父子关系或者树形结构。对于自连接查询无须显式指定自连接关键字。只要对同一张表指定不同的别名，将其作为两张表对待进行连接即可。

SELECT lhs.id, rhs.parent, lhs.c1, rhs.c2 FROM tree\_data lhs, tree\_data rhs WHERE lhs.id = rhs.parent;

### 5.笛卡尔连接

为了避免由于书写错误产生海量结果集，Impala不支持诸如下述例子中的笛卡尔连接。

SELECT ... FROM t1 JOIN t2;

SELECT ... FROM t1, t2;

如果我们想实现笛卡尔乘积，可以使用CROSS JOIN作为连接操作符。但是CROSS JOIN连接形式不支持ON子句，因为该连接将对连接的表的所有行进行组合。通过这种方式产生的结果集仍然可以使用WHERE子句进行过滤操作。

SELECT ... FROM t1 CROSS JOIN t2;

SELECT ... FROM t1 CROSS JOIN t2 WHERE *tests\_on\_non\_join\_columns*;

### 6.内连接和外连接

内连接是最常用最熟悉的连接类型: 内连接的结果集包含所有参与连接的表中匹配的列，这些列具有满足在不同表之间关联列的等值匹配。如果参与连接的表具有相同的列名，则需要使用指定完全限定名或者列别名对该列进行引用。Impala的内连接支持SQL-89 SQL-92语法。

SELECT t1.id, c1, c2 FROM t1, t2 WHERE t1.id = t2.id;

SELECT t1.id, c1, c2 FROM t1 JOIN t2 ON t1.id = t2.id;

SELECT t1.id, c1, c2 FROM t1 INNER JOIN t2 ON t1.id = t2.id;

外连接从左手型表，或右手型表，或者全外连接表获取所有的行数据。如果外连接的表中没有与其他表关联匹配的数据，在结果集中相关列会被置为NULL。在一个外连接查询中需要包含OUTER关键字作为连接操作符，OUTER可与LEFT,RIGHT,FULL配合使用。

SELECT \* FROM t1 LEFT OUTER JOIN t2 ON t1.id = t2.id;

SELECT \* FROM t1 RIGHT OUTER JOIN t2 ON t1.id = t2.id;

SELECT \* FROM t1 FULL OUTER JOIN t2 ON t1.id = t2.id;

对于外连接查询，Impala使用的是SQL-92语法，也就是说使用JOIN关键字而不是使用逗号对参与关联的表进行连接。Impala不支持(+)或者\*=等兼容SQL-89的连接语法。

### 7.等值连接和非等值连接

默认情况下，Impala对表进行等值连接查询。无论是Insert，outer，full，semi连接，都执行的是等值连接

可使用比较运算符实现非等值连接，这种类型的连接可以避免产生超出资源限制的超大结果集。若执行的非等值连接产生的结果集大小可以接受，则可使用cross join操作符，并在where子句中进行额外的比较操作

SELECT ... FROM t1 CROSS JOIN t2 WHERE t1.total > t2.maximum\_price;

### 半连接

半连接相对较少使用。如果使用的是left semi join，就意味着只有左手型表中满足on或where子句与指定的与右手型表关联条件的数据会被返回

SELECT t1.c1, t1.c2, t1.c2 FROM t1 LEFT SEMI JOIN t2 ON t1.id = t2.id;

impala不支持自然连接和反连接

当on或where子句中关联列具有相同名称时，需要指定像db\_name.table\_name这样的完全限定名

select t1.c1 as first\_id, t2.c2 as second\_id from t1 join t2 on first\_id = second\_id; select fact.custno, dimension.custno from customer\_data as fact join customer\_address as dimension using (custno)

为了控制连接查询的结果集，需要在on或using子句中指定表的列名，或者在where条件中对列进行等值比较

[localhost:21000] > select c\_last\_name, ca\_city from customer join customer\_address where c\_customer\_sk = ca\_address\_sk; +-------------+-----------------+ | c\_last\_name | ca\_city | +-------------+-----------------+ | Lewis | Fairfield | | Moses | Fairview | | Hamilton | Pleasant Valley | | White | Oak Ridge | | Moran | Glendale | ... | Richards | Lakewood | | Day | Lebanon | | Painter | Oak Hill | | Bentley | Greenfield | | Jones | Stringtown | +-------------+------------------+ Returned 50000 row(s) in 9.82s

### ORDER BY

语法：

ORDER BY col\_ref [, col\_ref ...] [ASC | DESC] [NULLS FIRST | NULLS LAST] col\_ref ::= column\_name | integer\_literal

对select语句产生的结果集中的单列或多列进行排序。对于分布式查询，在排序之前，整个结果需要传输到一个节点上进行。

默认的排序顺序为升序，指定desc可改为降序

NULLS FIRST/NULLS LAST 子句可以控制null值在结果集的开头还是结尾

### GROUP BY

使用时需要使用count()、sum()、avg()、min()、max()聚集函数。在group by子句中指定的所有列都不会参与聚集运算

示例：

select

**ss\_item\_sk** as Item,

**count**(ss\_item\_sk) as Times\_Purchased,

**sum**(ss\_quantity) as Total\_Quantity\_Purchased

from store\_sales

**group by ss\_item\_sk**

order by sum(ss\_quantity) desc

limit 5;

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

| item | times\_purchased | total\_quantity\_purchased |

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

| 9325 | 372 | 19072 |

| 4279 | 357 | 18501 |

| 7507 | 371 | 18475 |

| 5953 | 369 | 18451 |

| 16753 | 375 | 18446 |

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

### HAVING

对带有group by的select查询执行过滤操作，它过滤的是聚集函数运算结果，而非原始数据

### LIMIT

limit子句限制select产生的结果集输出的最大行数，在以下情况下使用：

·TOP-N查询

·从一个或查询中采样数据。对于没有order by子句限制的查询，使用limit限制获取的数据时随机的

·保持从海量结果集中返回的数据规模。若一个查询由于where子句匹配的行数比预期的结果多，或其它不可预知的情况下产生了一个很大的结果集，使用limit可以有效减少结果集的规模

### OFFSET

offset子句可以查询字逻辑第一行后的某行开始返回结果。返回的结果集从0开始编号。常结合order by和limit一起使用

示例：

[localhost:21000] > create database limits;

[localhost:21000] > use limits;

[localhost:21000] > create table numbers (x int);

[localhost:21000] > insert into numbers values (1), (3), (4), (5), (2);

Inserted 5 rows in 1.34s

[localhost:21000] > select x from numbers limit 100;

+---+

| x |

+---+

| 1 |

| 3 |

| 4 |

| 5 |

| 2 |

+---+

Returned 5 row(s) in 0.26s

[localhost:21000] > select x from numbers limit 3;

+---+

| x |

+---+

| 1 |

| 3 |

| 4 |

+---+

Returned 3 row(s) in 0.27s

[localhost:21000] > select x from numbers where x > 2 limit 2;

+---+

| x |

+---+

| 3 |

| 4 |

+---+

Returned 2 row(s) in 0.27s

[localhost:21000] > select x as "Top 3" from numbers order by x desc limit 3;

+-------+

| top 3 |

+-------+

| 5 |

| 4 |

| 3 |

+-------+

[localhost:21000] > select x as "Bottom 3" from numbers order by x limit 3;

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

| bottom 3 |

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

| 1 |

| 2 |

| 3 |

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

### UNION

语法：

query\_1 UNION [DISTINCT | ALL] query\_2

合并多个查询的结果集，默认情况下，合并的结果好像是使用了distinct操作符一样

示例:

[localhost:21000] > create table few\_ints (x int);

[localhost:21000] > insert into few\_ints values (1), (1), (2), (3);

[localhost:21000] > set default\_order\_by\_limit=1000;

[localhost:21000] > select x from few\_ints order by x;

+---+

| x |

+---+

| 1 |

| 1 |

| 2 |

| 3 |

+---+

Returned 4 row(s) in 0.41s

[localhost:21000] > select x from few\_ints union all select x from few\_ints;

+---+

| x |

+---+

| 1 |

| 1 |

| 2 |

| 3 |

| 1 |

| 1 |

| 2 |

| 3 |

+---+

Returned 8 row(s) in 0.42s

[localhost:21000] > select \* from (select x from few\_ints union all select x from few\_ints) as t1 order by x;

+---+

| x |

+---+

| 1 |

| 1 |

| 1 |

| 1 |

| 2 |

| 2 |

| 3 |

| 3 |

+---+

Returned 8 row(s) in 0.53s

[localhost:21000] > select x from few\_ints union all select 10;

+----+

| x |

+----+

| 10 |

| 1 |

| 1 |

| 2 |

| 3 |

+----+

Returned 5 row(s) in 0.38s

[localhost:21000] > select x from few\_ints union select x+1 from few\_ints;

+---+

| x |

+---+

| 3 |

| 4 |

| 1 |

| 2 |

+---+

Returned 4 row(s) in 0.51s

[localhost:21000] > select x from few\_ints union select 10;

+----+

| x |

+----+

| 2 |

| 10 |

| 1 |

| 3 |

+----+

Returned 4 row(s) in 0.49s

[localhost:21000] > select \* from (select x from few\_ints union select x from few\_ints) as t1 order by x;

+---+

| x |

+---+

| 1 |

| 2 |

| 3 |

+---+

Returned 3 row(s) in 0.53s

### WITH

放在select子句之前。用于为复杂表达式定义别名，该别名定义好后可以在后续的select语句中多次引用。但无法持久化，其定义的别名也不可同表名或视图名相同

示例：

-- Define 2 subqueries that can be referenced from the body of a longer query.

with t1 as (select 1), t2 as (select 2) insert into tab select \* from t1 union all select \* from t2;

-- Define one subquery at the outer level, and another at the inner level as part of the

-- initial stage of the UNION ALL query.

with t1 as (select 1) (with t2 as (select 2) select \* from t2) union all select \* from t1;

### Hints

语法:

SELECT *select\_list* FROM

*join\_left\_hand\_table*

JOIN [{BROADCAST|SHUFFLE}]

*join\_right\_hand\_table*

*remainder\_of\_query*;

INSERT *insert\_clauses*

[{SHUFFLE|NOSHUFFLE}]

SELECT *remainder\_of\_query*;

使用hints可以更好地从底层调整sql查询的工作方式。对于丢失了统计信息或其它因数导致查询成本异常昂贵时，使用hints可以作为临时的解决方案。hints的使用方式为使用[]将特定的hints括起来放在sql语句中

[broadcast]和[shuffle]控制join查询的连接方式，可在语句中的join关键字后加上这个hints

[broadcast]：join操作使用广播技术，将右手型表的所有内容发送到与之关联的所有节点上。当表或索引的统计信息不可用时，这种连接方式是默认的连接方式。适用于大表和小表之间的连接

[shuffle]：join操作使用分区技术，将所有表的相关联的行使用hash算法分片，具有相同hash值的数据被发送到同一个节点上，适用于大表之间的连接

示例：

SELECT *select\_list* FROM

*join\_left\_hand\_table*

JOIN /\* +BROADCAST|SHUFFLE \*/

*join\_right\_hand\_table*

*remainder\_of\_query*;

SELECT *select\_list* FROM

*join\_left\_hand\_table*

JOIN -- +BROADCAST|SHUFFLE

*join\_right\_hand\_table*

*remainder\_of\_query*;

INSERT *insert\_clauses*

/\* +SHUFFLE|NOSHUFFLE \*/

SELECT *remainder\_of\_query*;

INSERT *insert\_clauses*

-- +SHUFFLE|NOSHUFFLE

SELECT *remainder\_of\_query*;

当向分区插入数据时，尤其向Parquet格式的表插入数据时，可对insert语句使用hints来降低资源的使用，提高性能

使用注意：

·可在使用insert向分区插入数据因为容量限制导致失败，或插入性能低下时使用hints

·使用[shuffle]或[noshuffle]的hints时，需将其放在partition关键字后，select关键字前

·[shuffle]关键字将导致选择并发向hdfs写文件数最小的执行计划，使用1G缓存不同的分区。这将大大减少insert操作的资源消耗，由于Impala总将同一分区的数据放在同一节点上，这一操作可能引起节点间的数据传输

·[noshuffle]关键字将导致选择一个插入速度较快，但是会生成很多小文件的执行计划，可能引起insert操作由于超过容量限制而失败

·若insert.......select查询的源表中包含分区列，即使没有列统计信息，Impala也会自动选择[shffule]方式插入

·若insert......select查询源表中的分区列的统计信息可用，Impala将根据列的distinct值的个数和insert操作的节点数来确定使用[shuffle]还是[noshuffle]

### DISTINCT

用于过滤并删除结果集中的重复值

示例：

select distinct c\_birth\_country from customer;

select distinct c\_salutation, c\_last\_name from customer;

select count(distinct c\_birth\_country) from customer;

但impala不支持在同一个查询中使用多次聚集函数

## SHOW

语法：

SHOW DATABASES [[LIKE] '*pattern*']

SHOW SCHEMAS [[LIKE] '*pattern*'] - an alias for SHOW DATABASES

SHOW TABLES [IN *database\_name*] [[LIKE] '*pattern*']

SHOW [AGGREGATE | ANALYTIC] FUNCTIONS [IN *database\_name*] [[LIKE] '*pattern*']

SHOW CREATE TABLE [*database\_name*].*table\_name*

SHOW TABLE STATS [*database\_name*.]*table\_name*

SHOW COLUMN STATS [*database\_name*.]*table\_name*

SHOW PARTITIONS [*database\_name*.]*table\_name*

SHOW ROLES

SHOW CURRENT ROLES

SHOW ROLE GRANT GROUP *group\_name*

SHOW GRANT ROLE *role\_name*

示例：

可以使用通配符\*来代表任意字符

show databases 'a\*';

show databases like 'a\*';

show tables in some\_db like '\*fact\*';

use some\_db;

show tables '\*dim\*|\*fact\*';

### SHOW ROLES

该语句显示角色，可显示用create role语句所创建的角色

show roles;

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

| role\_name |

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

| analyst |

| role1 |

| sales |

| superuser |

| test\_role |

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

### SHOW CURRENT ROLE

该语句显示分配给当前用户的角色

show current roles;

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

| role\_name |

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

| role1 |

| superuser |

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

### SHOW DATABASE

该语句显示所有数据库名称

[localhost:21000] > show databases;

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

| name |

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

| \_impala\_builtins |

| analyze\_testing |

| avro |

| ctas |

| d1 |

| d2 |

| d3 |

| default |

| file\_formats |

| hbase |

| load\_data |

| partitioning |

| regexp\_testing |

| reports |

| temporary |

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

Returned 14 row(s) in 0.02s

[localhost:21000] > show tables in file\_formats;

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

| name |

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

| parquet\_table |

| rcfile\_table |

| sequencefile\_table |

| textfile\_table |

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

Returned 4 row(s) in 0.01s

[localhost:21000] > use file\_formats;

[localhost:21000] > show tables like '\*parq\*';

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

| name |

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

| parquet\_table |

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

Returned 1 row(s) in 0.01s

### SHOW CREATE TABLE

该语句显示某个表的表结构

create table show\_create\_table\_demo (id int comment "Unique ID", y double, s string)

partitioned by (year smallint)

stored as parquet;

show create table show\_create\_table\_demo;

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

| result |

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

| CREATE TABLE scratch.show\_create\_table\_demo ( |

| id INT COMMENT 'Unique ID', |

| y DOUBLE, |

| s STRING |

| ) |

| PARTITIONED BY ( |

| year SMALLINT |

| ) |

| STORED AS PARQUET |

| LOCATION 'hdfs://127.0.0.1:8020/user/hive/warehouse/scratch.db/show\_create\_table\_demo' |

| TBLPROPERTIES ('transient\_lastDdlTime'='1418152582') |

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

alter table show\_create\_table\_demo drop column s;

alter table show\_create\_table\_demo set fileformat textfile;

show create table show\_create\_table\_demo;

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

| result |

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

| CREATE TABLE scratch.show\_create\_table\_demo ( |

| id INT COMMENT 'Unique ID', |

| y DOUBLE |

| ) |

| PARTITIONED BY ( |

| year SMALLINT |

| ) |

| STORED AS TEXTFILE |

| LOCATION 'hdfs://127.0.0.1:8020/user/hive/warehouse/demo.db/show\_create\_table\_demo' |

| TBLPROPERTIES ('transient\_lastDdlTime'='1418152638') |

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

### SHOW TABLE STATS

该语句对于调优性能和诊断性能问题十分重要,尤其是在最大的表和最复杂的连接查询

它显示表数据提供的一些关于表的一般信息,如文件的数量,总体规模的数据,是否部分或全部HDFS数据的缓存,有用的文件格式,是否已经运行计算统计数据声明等

show table stats store\_sales;

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

| #Rows | #Files | Size | Bytes Cached | Format | Incremental stats |

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

| -1 | 1 | 370.45MB | NOT CACHED | TEXT | false |

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

show table stats customer;

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

| #Rows | #Files | Size | Bytes Cached | Format | Incremental stats |

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

| -1 | 1 | 12.60MB | NOT CACHED | TEXT | false |

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

compute stats customer;

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

| summary |

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

| Updated 1 partition(s) and 18 column(s). |

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

show table stats customer;

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

| #Rows | #Files | Size | Bytes Cached | Format | Incremental stats |

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

| 100000 | 1 | 12.60MB | NOT CACHED | TEXT | false |

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

compute incremental stats store\_sales;

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

| summary |

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

| Updated 1 partition(s) and 23 column(s). |

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

show table stats store\_sales;

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

| #Rows | #Files | Size | Bytes Cached | Format | Incremental stats |

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

| 2880404 | 1 | 370.45MB | NOT CACHED | TEXT | false |

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

### SHOW COLUMN STATS

该语句对于调优性能和诊断性能问题十分重要,尤其是在最大的表和最复杂的连接查询。可用于显示的输出显示列数据声明

show column stats customer;

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

| Column | Type | #Distinct Values | #Nulls | Max Size | Avg Size |

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

| c\_customer\_sk | INT | -1 | -1 | 4 | 4 |

| c\_customer\_id | STRING | -1 | -1 | -1 | -1 |

| c\_current\_cdemo\_sk | INT | -1 | -1 | 4 | 4 |

| c\_current\_hdemo\_sk | INT | -1 | -1 | 4 | 4 |

| c\_current\_addr\_sk | INT | -1 | -1 | 4 | 4 |

| c\_first\_shipto\_date\_sk | INT | -1 | -1 | 4 | 4 |

| c\_first\_sales\_date\_sk | INT | -1 | -1 | 4 | 4 |

| c\_salutation | STRING | -1 | -1 | -1 | -1 |

| c\_first\_name | STRING | -1 | -1 | -1 | -1 |

| c\_last\_name | STRING | -1 | -1 | -1 | -1 |

| c\_preferred\_cust\_flag | STRING | -1 | -1 | -1 | -1 |

| c\_birth\_day | INT | -1 | -1 | 4 | 4 |

| c\_birth\_month | INT | -1 | -1 | 4 | 4 |

| c\_birth\_year | INT | -1 | -1 | 4 | 4 |

| c\_birth\_country | STRING | -1 | -1 | -1 | -1 |

| c\_login | STRING | -1 | -1 | -1 | -1 |

| c\_email\_address | STRING | -1 | -1 | -1 | -1 |

| c\_last\_review\_date | STRING | -1 | -1 | -1 | -1 |

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

### SHOW PARTITIONS

显示每个分区的分区表信息

[localhost:21000] > show partitions census;

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

| year | #Rows | #Files | Size | Format |

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

| 2000 | -1 | 0 | 0B | TEXT |

| 2004 | -1 | 0 | 0B | TEXT |

| 2008 | -1 | 0 | 0B | TEXT |

| 2010 | -1 | 0 | 0B | TEXT |

| 2011 | 4 | 1 | 22B | TEXT |

| 2012 | 4 | 1 | 22B | TEXT |

| 2013 | 1 | 1 | 231B | PARQUET |

| Total | 9 | 3 | 275B | |

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

### SHOW FUNCTIONS

默认情况下，显示的是UDFs，而show aggregate functions显示的是UDAFs，show functions的输出包含了每个函数的参数签名

若数据库名为\_impala\_builtins，可显示impala的内嵌函数

show functions in \_impala\_builtins;

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

| return type | signature |

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

| BOOLEAN | ifnull(BOOLEAN, BOOLEAN) |

| TINYINT | ifnull(TINYINT, TINYINT) |

| SMALLINT | ifnull(SMALLINT, SMALLINT) |

| INT | ifnull(INT, INT) |

...

show functions in \_impala\_builtins like '\*week\*';

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

| return type | signature |

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

| INT | weekofyear(TIMESTAMP) |

| TIMESTAMP | weeks\_add(TIMESTAMP, INT) |

| TIMESTAMP | weeks\_add(TIMESTAMP, BIGINT) |

| TIMESTAMP | weeks\_sub(TIMESTAMP, INT) |

| TIMESTAMP | weeks\_sub(TIMESTAMP, BIGINT) |

| INT | dayofweek(TIMESTAMP) |

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

## USE

语法:

USE *db\_name*

默认情况下连接的数据库为default。在impala-shell会话中执行use db\_name可以切换到指定的数据库

# 内嵌函数

## 数学函数

### abs()

语法：abs(numeric\_type a)

功能：返回参数的绝对值。

返回类型：与输入值相同

### acos()

语法：acos(doublea)

功能：返回参数的反余弦值。

返回类型：double

### asin()

语法：asin(double a)

功能：返回参数的反正余弦值

返回类型：double

### atan()

语法：atan(double a)

功能：返回参数的反正切值

返回类型：double

### bin()

语法：bin(bigint a)

功能：返回参数的二进制代码表示

返回类型：string

### ceil(),ceiling()

语法：ceil(double a), ceiling(double a), ceil(decimal(p,s) a), ceiling(decimal(p,s) a)

功能：返回大于或等于参数值的最小整数

返回类型：整型

### conv()

语法：conv(bigint num, int from\_base, int to\_base), conv(string num, int from\_base, int to\_base)

功能：将参数值转换为指定进制的数值字符串

返回类型：string

### cos()

语法：cos(double a)

功能：返回参数的余弦值

返回类型：double

### degrees()

语法：degrees(double a)

功能：将弧度值转换为度数

返回类型：double

### e()

语法：e()

功能：返回自然对数e

返回类型：double

### exp()

语法：exp(double a)

功能：返回自然对数e的a次方

返回类型：double

### floor()

语法：floor(double a)

功能：返回小于或等于参数值的最大整数值

返回类型：int

### fmod()

语法：fmod(double a, double b), fmod(float a, float b)

功能：返回参数a除以参数b的余数

返回类型：double/float，取决于参数类型

使用注意：

该函数自Impala1.1.1起可以使用

因该函数是操作double或float值，存在潜在的四舍五入的错误可能不能精确地表示结果。更适合使用整数或能精确表示的double/float的值

### fnv\_hash()

语法：fnv\_hash(type v)

功能：从输入参数返回一个64位的值，实现在程序中的hash运算

返回类型：bigint

使用注意：

在很多情况下，需要在应用程序中基于该函数的返回值来进行将某些不同特征的值聚集在一起处理来实现负载均衡或其它的分布式技术

因为返回值是一个64位的值，可以使abs()和%来将结果限制在某个范围内：abs(fnv\_hash(x)%10)

在不支持CRC32算法的系统上，该函数与Impala内部使用hash算法相同

该函数是FNV-la的变种，某些不同值组合作为输入参数，有可能产生相同的返回值，所以它不适合用于加密

示例：

[localhost:21000] > create table h (x int, s string); [localhost:21000] > insert into h values (0, 'hello'), (1,'world'), (1234567890,'antidisestablishmentarianism'); [localhost:21000] > select x, fnv\_hash(x) from h; +------------+----------------------+ | x | fnv\_hash(x) | +------------+----------------------+ | 0 | -2611523532599129963 | | 1 | 4307505193096137732 | | 1234567890 | 3614724209955230832 | +------------+----------------------+

[localhost:21000] > select s, fnv\_hash(s) from h; +------------------------------+---------------------+ | s | fnv\_hash(s) | +------------------------------+---------------------+ | hello | 6414202926103426347 | | world | 6535280128821139475 | | antidisestablishmentarianism | -209330013948433970 | +------------------------------+---------------------+ [localhost:21000] > select s, abs(fnv\_hash(s)) % 10 from h; +------------------------------+-------------------------+ | s | abs(fnv\_hash(s)) % 10.0 | +------------------------------+-------------------------+ | hello | 8 | | world | 6 | | antidisestablishmentarianism | 4 | +------------------------------+-------------------------+

对于较短的参数值，其结果的高阶位有比较低的熵：

[localhost:21000] > create table b (x boolean); [localhost:21000] > insert into b values (true), (true), (false), (false); [localhost:21000] > select x, fnv\_hash(x) from b; +-------+---------------------+ | x | fnv\_hash(x) | +-------+---------------------+ | true | 2062020650953872396 | | true | 2062020650953872396 | | false | 2062021750465500607 | | false | 2062021750465500607 | +-------+---------------------+

### greatest()

语法：greatest(bigint a[, bigint b ...]),greatest(double a[, double b ...]),greatest(decimal(p,s) a[, decimal(p,s) b ...]),greatest(string a[, string b ...]),greatest(timestamp a[, timestamp b ...])

功能：返回参数列表中的最大值

返回类型：与参数类型相同（若输入参数为int会被提升为bigint；若输入参数为float会被提升为double。可以使用cast()函数转换为自己需要的类型）

### hex()

语法：hex(bigint a), hex(string a)

功能：返回参数的16进制表示

返回类型：string

### least()

语法：least(bigint a[, bigint b ...]), least(double a[, double b ...]), least(decimal(p,s) a[, decimal(p,s) b ...]),least(string a[, string b ...]), least(timestamp a[, timestamp b ...])

功能：返回参数列表中的最小值

返回类型：与参数类型相同（若输入参数为int会被提升为bigint；若输入参数为float会被提升为double。可以使用cast()函数转换为自己需要的类型）

### ln()

语法：ln(double a)

功能：返回参数的自然对数

返回类型：double

### log()

语法：log(double base, double a)

功能：返回以base为底的a的对数

返回类型：double

### log10()

语法：log10(double a)

功能：返回以10为底的a的对数

返回类型：double

### log2()

语法：log2(double a)

功能：返回以2为底的a的对数

返回类型：double

### mod()

语法：mod(numeric\_type a, same\_type b)

功能：返回一个数的模，相当于%运算符。适合任何大小的int，float及十进制类型

返回类型：与参数相同

使用注意：

自Impala 2.2.0起可以使用

### negative()

语法：negative(numeric\_type a)

功能：返回输入参数的相反数

返回类型：与参数相同

### pi()

语法：pi()

功能：返回圆周率常数π

返回类型：double

### pmod()

语法：pmod(bigint a, bigint b), pmod(double a, double b)

功能：返回参数a除以参数b的正的余数

返回类型：与参数相同

### positive()

语法：positive(numeric\_type a)

功能：反会参数原始值

返回类型：与输入参数相同

### pow(),power()

语法：pow(double a, double p),power(double a, double p)

功能：返回参数a的p次幂

返回类型：double

### precision()

语法：precision(numeric\_expression)

功能：返回一个数字的精度值

返回类型：int

示例：

[localhost:21000] > select precision(5), scale(5);

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

| precision(5) | scale(5) |

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

| 3 | 0 |

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

[localhost:21000] > select precision(1.333), scale(1.333); +------------------+--------------+

| precision(1.333) | scale(1.333) |

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

| 4 | 3 |

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

[localhost:21000] > with t1 as ( select cast(12.34 as decimal(20,2)) x union select cast(1 as decimal(8,6)) x ) select precision(x), scale(x) from t1 limit 1; +--------------+----------+

| precision(x) | scale(x) |

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

| 24 | 6 |

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

### quotient()

语法：quotient(int a, int b)

功能：返回a除以b的整数部分

返回类型：int

### radians()

语法：radians(double a)

功能：返回度数对应的弧度数

返回类型：double

### rand()

语法：rand(), rand(int seed)

功能：返回一个介于0和1之间的随机数。若以某个特定的数值作为参数，它将返回一个确定的随机数

返回类型：double

使用注意：目前，随机序列每次查询后重置，多次调用rand()函数后返回相同的值。可通过一个独特的seed进行调用，如：select rand(unix\_timestamp())

### sign()

语法：sign(double a)

功能：符号函数，若参数为正，返回1；若参数为负，返回-1；若参数为0，返回0

返回类型：int

### sin()

语法：sin(double a)

功能：返回参数的正弦值

返回类型：double

### sqrt()

语法：sqrt(double a)

功能：返回参数的平方根

返回类型：double

### tan()

语法：tan(double a)

功能：返回参数的正切值

返回类型：double

### unhex()

语法：unhex(string a)

功能：返回16机制字符串对应的ASCII值的字符

返回类型：string

## 类型转换函数

### cast()

语法：cast(exprastype)

功能：强制转换，将其它函数返回的结果转换为指定的数据类型

返回类型：指定的数据类型

示例：

select concat('Here are the first ',10,' results.'); -- Fails select concat('Here are the first ',cast(10 as string),' results.'); -- Succeeds

## 时间和日期函数

### adddate()

语法：adddate(timestamp startdate, int days),adddate(timestamp startdate, bigint days)

功能：返回对startdate的时间戳增加指定days后的天数

返回类型：timestamp

### current\_timestamp()

语法：current\_timestamp()

功能：返回当前时间，是now()函数的别名

返回类型：timestamp

### date\_add()

语法：date\_add(timestamp startdate, int days), date\_add(timestamp startdate, interval\_expression)

功能：返回对startdate的时间戳增加指定days后的天数

返回类型：timestamp

使用注意：

与函数adddate()不同的是，第一个参数的类型可以是字符串，Impala会自动将其转换为timestamp类型。第二个参数除了可以以整型的方式指定天数外，还可以使用interval表达式来指定周、年、小时、秒等其它的时间单元

### date\_sub()

语法：date\_sub(timestamp startdate, int days), date\_sub(timestamp startdate, interval\_expression)

功能：返回从startdate的时间戳减去days后的天数

返回类型：timestamp

使用注意：

第一个参数的类型可以是字符串，Impala会自动将其转换为timestamp类型。第二个参数除了可以以整型的方式指定天数外，还可以使用interval表达式来指定周、年、小时、秒等其它的时间单元

### datediff()

语法：datediff(string enddate, string startdate)

功能：返回两个参数表示的日期的相差的天数

返回类型：int

### day(),dayofmonth()

语法：day(string date), dayofmonth(string date)

功能：从字符串参数表示的日期中返回每月的第几号

返回类型：int

### dayname()

语法：dayname(string date)

功能：从字符串参数表示的日期中返回英文表示的星期几。显示范围从“Sunday”到“saturd”

返回类型：string

### dayofweek()

语法：dayofweek(string date)

功能：从字符串参数表示的日期中返回是一周的第几天。显示范围从1（Sunday）到7（saturd）

返回类型：int

### dayofyear()

语法：dayofyear(timestamp date)

功能：返回参数时间戳表示的是一年的第几天。返回值范围为1(1月1号)到366(12月31号)

返回类型：int

### days\_add()

语法：days\_add(timestamp startdate, int days), days\_add(timestamp startdate, bigint days)

功能：返回对参数时间戳startdate加上指定的days的天数

返回类型：timestamp

使用注意：

该函数类似date\_add(),区别在于days\_add()函数的第一个输入参数为timestamp

### days\_sub()

语法：date\_sub(timestamp startdate, int days), date\_sub(timestamp startdate, interval\_expression)

功能：返回对参数时间戳startdate减去指定的days的天数

返回类型：timestamp

使用注意：

该函数类似date\_sub(),区别在于days\_sub()函数的第一个输入参数为timestamp

### from\_unixtime()

语法：from\_unixtime(bigint unixtime[, string format])

功能：从UNIX纪元时间到指定时间经历的秒数作为输入参数，按指定格式转换成字符串

返回类型：string

使用注意：

字符串的格式接受所有可以对timestamp指定的时间格式：日期时间，只有日期，只有时间，可以指定的秒的精度

可以使用不同数量的占位符对日期时间做不同的表示。使用多个y，d，H，可以在输出中表示为零前导符。对于M表示的月份，非零前导符表示为3，则使用MM表示的包含零前导符的形式表示为03，而使用MMM则表示为Mar，Impala不允许使用更多的M表示月份。一个日期字符串的格式可以表示为“yyyy-MM-dd HH:mm:ss:SSSSSS”，“dd/MM/yyyy HH:mm:ss:SSSSSS”,“MMMdd,yyy HH.mm.ss(SSSSSS)”等

示例：

[localhost:21000] > select from\_unixtime(1392394861,"yyyy-MM-dd HH:mm:ss.SSSS"); +-------------------------------------------------------+ | from\_unixtime(1392394861, 'yyyy-mm-dd hh:mm:ss.ssss') | +-------------------------------------------------------+ | 2014-02-14 16:21:01.0000 | +-------------------------------------------------------+

[localhost:21000] > select from\_unixtime(1392394861,"yyyy-MM-dd"); +-----------------------------------------+ | from\_unixtime(1392394861, 'yyyy-mm-dd') | +-----------------------------------------+ | 2014-02-14 | +-----------------------------------------+

[localhost:21000] > select from\_unixtime(1392394861,"HH:mm:ss.SSSS"); +--------------------------------------------+ | from\_unixtime(1392394861, 'hh:mm:ss.ssss') | +--------------------------------------------+ | 16:21:01.0000 | +--------------------------------------------+

[localhost:21000] > select from\_unixtime(1392394861,"HH:mm:ss"); +---------------------------------------+ | from\_unixtime(1392394861, 'hh:mm:ss') | +---------------------------------------+ | 16:21:01 | +---------------------------------------+

[localhost:21000] > select from\_unixtime(unix\_timestamp(now() + interval 3 days), 'yyyy/MM/dd HH:mm');

### from\_utc\_timestamp()

语法：from\_utc\_timestamp(timestamp, string timezone)

功能：将一个指定的UTC时间戳转换为指定时区的本地时间

返回类型：timestamp

### hour()

语法：hour(string date)

功能：从一个字符串表示的日期中返回小时部分

返回类型：int

### hours\_add()

语法：hours\_add(timestamp date, int hours), hours\_add(timestamp date, bigint hours)

功能：返回时间戳加上指定的小时数生成的新时间戳

返回类型：timestamp

### hours\_sub()

语法：hours\_sub(timestamp date, int hours), hours\_sub(timestamp date, bigint hours)

功能：返回时间戳减去指定的小时数生成的新时间戳

返回类型：timestamp

### microseconds\_add()

语法：microseconds\_add(timestamp date, int microseconds), microseconds\_add(timestamp date, bigint microseconds)

功能：返回时间戳加上指定的毫秒数生成的新的时间戳

返回类型：timestamp

### microseconds\_sub()

语法：microseconds\_sub(timestamp date, int microseconds), microseconds\_sub(timestamp date, bigint microseconds)

功能：返回时间戳减去指定的毫秒数生成的新的时间戳

返回类型：timestamp

### minute()

语法：minute(string date)

功能：返回字符串时间戳所表示的时间的分钟数

返回类型：int

### minutes\_add()

语法：minutes\_add(timestamp date, int minutes), minutes\_add(timestamp date, bigint minutes)

功能：返回时间戳加上指定的分钟数表示的新时间戳

返回类型：timestamp

### minutes\_sub()

语法：minutes\_sub(timestamp date, int minutes), minutes\_sub(timestamp date, bigint minutes)

功能：返回时间戳加上指定的分钟数表示的新时间戳

返回类型：timestamp

### month()

语法：month(string date)

功能：返回字符串时间戳表示时间的月份

返回类型：int

### months\_add()

语法：months\_add(timestamp date, int months), months\_add(timestamp date, bigint months)

功能：返回时间戳加上指定月份数表示的新时间戳

返回类型：timestamp

### months\_sub()

语法：months\_sub(timestamp date, int months), months\_sub(timestamp date, bigint months)

功能：返回时间戳减去指定月份数表示的新时间戳

返回类型：timestamp

### nanoseconds\_add()

语法：nanoseconds\_add(timestamp date, int nanoseconds), nanoseconds\_add(timestamp date, bigint nanoseconds)

功能：返回时间戳加上指定的纳秒表示的新时间戳

返回类型：timestamp

### nanoseconds\_sub()

语法：nanoseconds\_sub(timestamp date, int nanoseconds), nanoseconds\_sub(timestamp date, bigint nanoseconds)

功能：返回时间戳减去指定的纳秒数表示的新时间戳

返回类型：timestamp

### now()

语法：now()

功能：返回当前UTC标准日期和时间

返回类型：timestamp

### second()

语法：second(string date)

功能：返回字符串所表示的时间的秒数

返回类型：int

### seconds\_add()

语法：seconds\_add(timestamp date, int seconds), seconds\_add(timestamp date, bigint seconds)

功能：返回时间戳加上指定的秒数表示的新时间戳

返回类型：timestamp

### seconds\_sub()

语法：seconds\_sub(timestamp date, int seconds), seconds\_sub(timestamp date, bigint seconds)

功能：返回时间戳减去指定的秒数表示的新时间戳

返回类型：timestamp

### subdate()

语法：subdate(timestamp startdate, int days), subdate(timestamp startdate, bigint days)

功能：返回时间戳减去指定的天数生成的新时间戳

返回类型：timestamp

使用注意：

该函数与date\_sub()的区别在于本函数的输入参数为时间戳，而非字符串

### to\_date()

语法：to\_date(timestamp)

功能：返回基于某个时间戳的字符串

返回类型：string

### to\_utc\_timestamp()

语法：to\_utc\_timestamp(timestamp, string timezone)

功能：将指定时区的本地时间转换成UTC标准时间

返回类型：timestamp

### unix\_timestamp()

语法：unix\_timestamp(), unix\_timestamp(string datetime), unix\_timestamp(string datetime, string format), unix\_timestamp(timestamp datetime)

功能：返回当前时间，或指定时间与UNIX纪元时间之间经历的秒数

返回类型：bigint

### weekofyear()

语法：weekofyear(string date)

功能：返回字符串时间为一年中的第多少周。返回值范围为1-53

返回类型：int

### weeks\_add()

语法：weeks\_add(timestamp date, int weeks), weeks\_add(timestamp date, bigint weeks)

功能：返回时间戳加上指定的周数表示的新时间戳

返回类型：timestamp

### weeks\_sub()

语法：weeks\_sub(timestamp date, int weeks), weeks\_sub(timestamp date, bigint weeks)

功能：返回时间戳减去指定的周数表示的新时间戳

返回类型：timestamp

### year()

语法：year(string date)

功能：返回字符串表示的时间的年份

返回类型：int

### years\_add()

语法：years\_add(timestamp date, int years), years\_add(timestamp date, bigint years)

功能：返回时间戳加上指定的年数表示的新时间戳

返回类型：timestamp

### years\_sub()

语法：years\_sub(timestamp date, int years), years\_sub(timestamp date, bigint years)

功能：返回时间戳加上减去指定的年数表示的新时间戳

返回类型：timestamp

### extract()

语法：extract(timestamp, string unit) extract(unit FROM timestamp)

功能：从一个时间戳中返回一个数值型日期

返回类型：int

### milliseconds\_add()

语法：milliseconds\_add(timestamp date, int milliseconds), milliseconds\_add(timestamp date, bigint milliseconds)

功能：返回时间戳加上指定的毫秒数的新时间戳

返回类型：timestamp

### milliseconds\_sub()

语法：milliseconds\_sub(timestamp date, int milliseconds), milliseconds\_sub(timestamp date, bigint milliseconds)

功能：返回时间戳减去指定的毫秒数的新时间戳

返回类型：timestamp

## 条件函数

### CASE a WHEN b THEN c END

语法：CASE a WHEN b THEN c [WHEN d THEN e]... [ELSE f] END

功能：将表达式与多个可能值进行比较，如果能匹配到，则返回相应的结果

返回类型：与初始参数值相同。但若是整型，则会被提升为bigint；若是float型，则被提升为double。

使用注意：

示例：

select case x when 1 then 'one' when 2 then 'two' when 0 then 'zero' else 'out of range' end from t1;

### CASE WHEN a THEN b [WHEN c THEN d]... [ELSE e] END

语法：CASE WHEN a THEN b [WHEN c THEN d]... [ELSE e] END

功能：测试一个序列表达式是否为true，返回第一个表达式为true的结果

返回类型：与初始参数值相同。但若是整型，则会被提升为bigint；若是float型，则被提升为double。

示例：

select case when dayname(now()) in ('Saturday','Sunday') then 'result undefined on weekends' when x > y then 'x greater than y' when x = y then 'x and y are equal' when x is null or y is null then 'one of the columns is null' else null end from t1;

### coalesce()

语法：coalesce(type v1, type v2, ...)

功能：返回参数中的第一个非空值，若所有参数均为null，则返回null

返回类型：与初始参数值相同。但若是整型，则会被提升为bigint；若是float型，则被提升为double。

### decode()

语法：decode(type expression, type search1, type result1 [, type search2, type result2 ...] [, type default] )

功能：将表达式与一个或多个可能的值进行比较，当找到匹配时，返回结果

返回类型：与初始参数值相同。但若是整型，则会被提升为bigint；若是float型，则被提升为double。

示例：

SELECT event, decode(day\_of\_week, 1, "Monday", 2, "Tuesday", 3, "Wednesday", 4, "Thursday", 5, "Friday", 6, "Saturday", 7, "Sunday", "Unknown day") FROM calendar;

### if()

语法：if(boolean condition, type ifTrue, type ifFalseOrNull)

功能：根据一个表达式的结果为true，false，null。来返回相应的结果值

返回类型：与参数类型相同

### ifnull()

语法：ifnull(type a, type ifNotNull)

功能：若表达式的结果非空，则返回表达式的结果。若表达式的结果为null，则返回第二个参数值

返回类型：与参数相同

### isfalse()

语法：isfalse(boolean)

功能：测试一个布尔表达式是否为false。若为false，则返回true；若为null，返回false；

返回类型：boolean

使用注意：

该函数在Impala 2.2.0加入

### isnotfalse()

语法：isnotfalse(boolean)

功能：测试表达式是否为not false。若为not false，则返回true；若为null，则返回true

返回类型：boolean

### isnottrue()

语法：isnottrue(boolean)

功能：测试表达式是否为not true。若为not true，则返回true；若为null，则返回true

返回类型：boolean

### isnull()

语法：isnull(type a, type ifNotNull)

功能：测试一个表达式是否为null，若不为null，则返回表达式结果；若为null，则返回第二个参数

返回类型：与参数相同

### istrue()

语法：istrue(boolean)

功能：测试一个表达式是否为true。若为true，则返回true；若为null，则返回false

返回类型：boolean

### notnullvalue()

语法：notnullvalue(expression)

功能：测试一个表达式是否为null。若为null，则返回false；

返回类型：boolean

### nullif()

语法：nullif(expr1,expr2)

功能：若两个表达式相等，则返回null。若不相等，则返回第一个表达式的值

返回类型：null/与原始的返回值类型相同

使用注意：

两个表达式的数据类型必须兼容，且第一个表达式不能为null，若为null，则永远不会与第二个表达式进行匹配

此函数是case表达式的简写

CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END

该函数通常使用在除法表达式中来防止被除数为0的情况

select 1.0 / nullif(c1,0) as reciprocal from t1;

### nullifzero()

语法：nullifzero(numeric\_expr)

功能：若表达式结果为0，则返回null；若表达式非0，则返回表达式本身

返回类型：与原始返回值类型相同

### nullvalue()

语法：nullvalue(expression)

功能：测试表达式是否为null。若为null，则返回true

返回类型：boolean

### nvl()

语法：nvl(type a, type ifNotNull)

功能：是isnull()函数的别名

返回类型：与参数相同

### zeroifnull()

语法：zeroifnull(numeric\_expr)

功能：若表达式为null，则返回0；若表达式为非null，则返回表达式本身

返回类型：与原始返回值类型相同

## 字符串函数

### ascii()

语法：ascii(string str)

功能：返回首字母的ascii码值

返回类型：int

### char\_length()

语法：char\_length(string a), character\_length(string a)

功能：返回参数的长度，是length()的别名

返回类型：int

### concat()

语法：concat(string a, string b...)

功能：将输入参数拼接在一起返回一个单字符串

返回类型：string

### concat\_ws()

语法：concat\_ws(string sep, string a, string b...)

功能：将各参数以指定的分隔符连在一起

返回类型：string

### find\_in\_set()

语法：find\_in\_set(string str, string strList)

功能：返回一个字符串在一个逗号分隔符中第一次出现的位置。若参数都为null，则返回null；若没有匹配到字符串或搜索字符串中出现了逗号，则返回0。

返回类型：int

### group\_concat()

语法：group\_concat(string s [, string sep])

功能：用于拼接结果集中的每一行返回一个字符串。若指定了分隔符，则最终返回的字符串将以指定的分隔符分隔。

返回类型：string

### initcap()

语法：initcap(string str)

功能：返回首字母大写的字符串

返回类型：string

### instr()

语法：instr(string str, string substr)

功能：返回子串在字符串中第一次出现的位置

返回类型：int

### length()

语法：length(string a)

功能：返回字符串的长度

返回类型：int

### locate()

语法：locate(string substr, string str[, int pos])

功能：返回子串在字符串中第一次出现的位置，可从某个位置之后开始匹配

返回类型：int

### lower(),lcase()

语法：lower(string a), lcase(string a)

功能：将参数改为小写

返回类型：string

### lpad()

语法：lpad(string str, int len, string pad)

功能：基于第一个输入参数返回一个指定长度的字符串。若第一个输入参数的长度比指定的长度短，则以第三个输入参数的重复序列进行前导填充。若第一个参数的长度比指定长度长，则从右侧截断字符串

返回类型：string

### Ltrim()

语法：ltrim(string a)

功能：去掉字符串左侧空格

返回类型：string

### parse\_url()

语法：parse\_url(string urlString, string partToExtract [, string keyToExtract])

功能：返回URL的指定部分，各部分包括PROTOCAL，HOST，PATH，REF，AUTHORITY，FILE，USERINFO/QUERY。必须使用大写形式

返回类型：string

### regxp\_extract()

语法：regexp\_extract(string subject, string pattern, int index)

功能：返回匹配正则模式的指定分组。分组0只是提取字符串。而分组1、分组2等指的是第一、第二部分

返回类型：

使用注意：

Impala支持的是Boost库使用的扩展的POSIX正则表达式语法。不支持.\*?非贪婪匹配

impala-shell解释器使用\作为转义字符，若需要通过impala-shell提交正则需要使用\\作为转义字符。可以使用字符类名[[:digit:]]，同时也可以使用\d，转移过后是\\d

示例：

[localhost:21000] > select regexp\_extract('abcdef123ghi456jkl','.\*?(\\d+)',0); +------------------------------------------------------+ | regexp\_extract('abcdef123ghi456jkl', '.\*?(\\d+)', 0) | +------------------------------------------------------+ | abcdef123ghi456 | +------------------------------------------------------+ Returned 1 row(s) in 0.11s

[localhost:21000] > select regexp\_extract('abcdef123ghi456jkl','.\*?(\\d+)',1); +------------------------------------------------------+ | regexp\_extract('abcdef123ghi456jkl', '.\*?(\\d+)', 1) | +------------------------------------------------------+ | 456 | +------------------------------------------------------+ Returned 1 row(s) in 0.11s

[localhost:21000] > select regexp\_extract('AbcdBCdefGHI','.\*?([[:lower:]]+)',1); +--------------------------------------------------------+ | regexp\_extract('abcdbcdefghi', '.\*?([[:lower:]]+)', 1) | +--------------------------------------------------------+ | def | +--------------------------------------------------------+

[localhost:21000] > select regexp\_extract('AbcdBCdefGHI','.\*?([[:lower:]]+).\*?',1); +-----------------------------------------------------------+ | regexp\_extract('abcdbcdefghi', '.\*?([[:lower:]]+).\*?', 1) | +-----------------------------------------------------------+ | bcd | +-----------------------------------------------------------+

### regexp\_replace()

语法：regexp\_replace(string initial, string pattern, string replacement)

功能：返回满足匹配模式的字符串替换首个参数中得到子串而得到的字符串

返回类型：string

示例：

[localhost:21000] > select regexp\_replace('aaabbbaaa','b+','xyz'); +------------------------------------------+ | regexp\_replace('aaabbbaaa', 'b+', 'xyz') | +------------------------------------------+ | aaaxyzaaa | +------------------------------------------+ Returned 1 row(s) in 0.11s

[localhost:21000] > select regexp\_replace('aaabbbaaa','(b+)','<\\1>'); +----------------------------------------------+ | regexp\_replace('aaabbbaaa', '(b+)', '<\\1>') | +----------------------------------------------+ | aaa<bbb>aaa | +----------------------------------------------+ Returned 1 row(s) in 0.11s

[localhost:21000] > select regexp\_replace('123-456-789','[^[:digit:]]',''); +---------------------------------------------------+ | regexp\_replace('123-456-789', '[^[:digit:]]', '') | +---------------------------------------------------+ | 123456789 | +---------------------------------------------------+ Returned 1 row(s) in 0.12s

### repeat()

语法：repeat(string str, int n)

功能：返回重复了次数的字符串

返回类型：string

### reverse()

语法：reverse(string a)

功能：返回翻转顺序后的字符串

返回类型：string

### rpad()

语法：rpad(string str, int len, string pad)

功能：基于第一个输入参数返回一个指定长度的字符串。若第一个输入参数的长度比指定的长度短，则以第三个输入参数的重复序列进行右侧填充。若第一个输入参数的长度比指定的长度长，则从右侧截断字符串

返回类型：string

### rtrim()

语法：rtrim(string a)

功能：返回去掉字符串右侧空格后的新字符串

返回类型：string

### space()

语法：space(int n)

功能：返回一个指定数目的空格字符串

返回类型：string

### strleft()

语法：strleft(string a, int num\_chars)

功能：返回字符串左侧指定数目的字母

返回类型：string

### strright()

语法：strright(string a, int num\_chars)

功能：返回字符串右侧指定数目的字母

返回类型：string

### substr(),substring()

语法：substr(string a, int start [, int len]), substring(string a, int start [, int len])

功能：返回从指定位置开始返回指定长度的字符串的子串。字符串中字符从1开始编号

返回类型：string

### translate()

语法：translate(string input, string from, string to)

功能：返回将第一个参数中得到子字符串替换为另一个子字符串后得到的新字符串

返回类型：string

### trim()

语法：trim(string a)

功能：返回去掉字符串首尾空格后的新字符串

返回类型：string

### upper(),ucase()

语法：upper(string a), ucase(string a)

功能：将字符串的所有字母转换成大写

返回类型：string

## 特殊函数

### current\_database()

语法：current\_database()

功能：返回当前数据库名称

返回类型：string

### effective\_user()

语法：effective\_user()

功能：与user()函数相同，但若授权启用时，返回授权用户的id

返回类型：string

### pid()

语法：pid()

功能：返回当前回话链接到的impala进程的PID

返回类型：int

### user()

语法：user()

功能：返回连接到impalad进行的Linux用户的用户名

返回类型：string

### version()

语法：version()

功能：返回当前连接到的impalad进程的编译日期及精确的版本号

返回类型：string

# 聚合函数

## appx\_median()

语法： APPX\_MEDIAN([DISTINCT | ALL] expression)

功能：返回一组数字结果集的中位数

返回类型：与参数类型相同

## avg()

语法：AVG([DISTINCT | ALL] expression) [OVER (analytic\_clause)]

功能：返回一组数字结果集的平均值

返回类型：double

使用注意：

在计算时将忽略null值

## count()

语法：COUNT([DISTINCT | ALL] expression) [OVER (analytic\_clause)]

功能：返回记录的条数

返回类型：bigint

使用注意：

count(\*)将null值的行记录在内

count(col\_name)只会计算col\_name为非空的行的记录

## max()

语法：MAX([DISTINCT | ALL] expression) [OVER (analytic\_clause)]

功能：返回一组数字结果集中的最大值

返回类型：与参数类型相同

使用注意：

包含null值将被忽略

## min()

语法：MIN([DISTINCT | ALL] expression) [OVER (analytic\_clause)]

功能：返回一组数字结果集中的最小值

返回类型：与参数类型相同

使用注意：包含的null值将被忽略

## ndv()

语法：NDV([DISTINCT | ALL] expression)

功能：返回列记录非重复的个数

返回类型：bigint

使用注意：

它运算速度要快于count(distinct col)，因为使用固定大小的内存进行计算

## stddev(),stddev\_samp(),stddev\_pop()

语法：{ STDDEV | STDDEV\_SAMP | STDDEV\_POP } ([DISTINCT | ALL] expression)

功能：返回一组数字的标准偏差

返回类型：double

## sum()

语法：SUM([DISTINCT | ALL] expression) [OVER (analytic\_clause)]

功能：返回一组数字结果集的和

返回类型：bigint/double

## variance(),variance\_samp(),variance\_pop(),var\_samp(),var\_pop()

语法：{ VARIANCE | VAR[IANCE]\_SAMP | VAR[IANCE]\_POP } ([DISTINCT | ALL] expression)

功能：返回一组数字的方差

返回类型：double

# Impala Shell

## 命令行选项

可以通过制定以下命令行选项来启动impala-shell来改变shell命令的执行方式

### -B或--delimited

对查询结果去格式化，并将其按照指定的分隔方式进行分隔。分隔符使用--output\_delimiter选项指定。-B可以将查询结果输出到一个文件中。在其它hadoop组件需要使用Impala产生的结果时，这个选项可以避免Impala对查询结果进行格式化的性能负载。

示例：

# impala-shell -B

### -o或--output\_file

该选项后跟文件名称，可以将输出结果存储为指定的文件。

示例：

# impala-shell -o tmp.txt

### --output\_delimiter=

该选项对使用-B选项去格式化输出的查询结果指定各字段间的分隔符。默认的分隔符为制表键(\t)

示例：

# impala-shell -B -output\_delimiter=’,’ -o tmp.txt

### -p或--show\_profiles

显示查询的执行计划(explain语句输出相同)和每个查询语句底层的执行步骤的详细信息

示例：

# impala -p

### -h或--help

显示帮助信息

### -i或--impalad=

后接主机名用来指定连接到的impalad节点。默认的连接接口为21000。若Impala启动时使用了备用端口，则需使用--fe\_port选项标标识

示例：

# impala-shell -i node1

### -q或--query=

该选项用于执行一个查询语句或shell命令。impala不必等待查询或者shell执行结束即可返回。这里的查询仅限于select，create table，show tables等单条语句

示例：

# impala-shell -q ‘select \* from tab1’

### -f或--query\_file=

后接sql查询脚本文件，脚本文件中的sql必须使用“；”来分隔

示例：

# impala-shell -f query.sql

### -k或--kerberos

该选项用来指定当shell连接到impalad节点时使用kerberos身份验证

### -s或--kerberos\_service\_name

后接kerberos服务名称

### -V或--verbose

启用详细信息输出

### --quiet

禁用详细信息输出

示例：

# imapal-shell --quiet

### -v或--version

显示版本信息

### -e

若查询失败继续执行

### -r或--refresh\_after\_connect

连接后刷新Impala元数据信息。等同于使用refresh语句

### -d或--database=

后接数据库名称用于连接到指定的数据库

示例：

# impala-shell -d d1

### -l

该选项用于启用ldap身份验证

### -u

若使用了-l选项，则用-u来指定身份验证的用户名称

## 连接到impalad

### 启动impala-shell

$ impala-shell

Welcome to the Impala shell. Press TAB twice to see a list of available commands.

Copyright (c) 2012 Cloudera, Inc. All rights reserved.

(Shell build version: Impala Shell v2.1.x (*hash*) built on *date*)

[Not connected] >

### 使用connect连接到impala实例

[Not connected] > connect impalad-host

[impalad-host:21000] >

## 运行命令

可按两次tab键获得所有命令列表

[impalad-host:21000] >

connect describe explain help history insert quit refresh select set shell show use version

[impalad-host:21000] >

也可在这个shell中直接执行sql语句

[localhost:21000] > select \*

> from t1

> limit 5;

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

| s1 | s2 |

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

| hello | world |

| goodbye | cleveland |

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

## 命令参考

alter：改变一个impala表，hive表的表结构或则配置信息。具体命令有alter table和alter view语句

compute stats：搜集表性能相关的统计信息

connect：连接到Impalad节点。默认端口21000

desc：显示指定表的列，列数据类型，列注释信息等。describe formatted命令还会显示hdfs数据目录，分区，表内部属性等更详细的信息

drop：删除一个对象

explain：显示一个查询的执行计划及执行的各个步骤信息

help：显示所有可用的命令和选项的列表

history：命令历史记录，它记录在~/.impalahistory文件中

insert：向指定的表中插入数据

invalidate metadata：更新元数据

profile：显示最近的查询到底层信息用于问题诊断或者性能优化

quit：退出shell

refresh：更新hdfs数据文件位置相关的元数据信息

select：查询满足特定条件的数据集

set：管理impala-shell回话的查询选项，使用set option=value语法

shell：不退出shell执行操作系统的命令，也可使用！作为shell命令的缩写

show：显示对象的元数据信息

unset：取消通过set设置的查询选项

use：切换到指定的数据库

version：返回impala的版本信息

## 查询参数设置

### [abort\_on\_error](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_abort_on_error.html)

若未启用该选项，当某个Impala进程节点故障时，已经发出的查询会在其它节点继续执行，并返回一个依赖部分Impalad进程节点运算的结果，该结果可能不完全正确，至少是不完整的。当启用该选项时，若任何节点发生错误，Impala将会立刻中断查询。该选项可以搜集错误发生时的诊断信息

类型：布尔

默认值：false

### [appx\_count\_distinct](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_appx_count_distinct.html)

允许在一个查询中使用多个count(distinct)操作。通过NDV()函数进行内部重写count(distinct)，得到一个近似的结果

类型：布尔

默认值：false

示例：

[localhost:21000] > select count(distinct x) from int\_t;

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

| count(distinct x) |

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

| 10 |

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

[localhost:21000] > select count(distinct property) from int\_t;

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

| count(distinct property) |

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

| 7 |

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

[localhost:21000] > select count(distinct x), count(distinct property) from int\_t;

ERROR: AnalysisException: all DISTINCT aggregate functions need to have the same set of parameters

as count(DISTINCT x); deviating function: count(DISTINCT property)

当使用appx\_count\_distinct后

[localhost:21000] > set APPX\_COUNT\_DISTINCT=true;

[localhost:21000] > select count(distinct x), count(distinct property) from int\_t;

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

| count(distinct x) | count(distinct property) |

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

| 10 | 7 |

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

### [batch\_size](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_batch_size.html)

sql操作符一次可以进行计算的行数。若未指定该值或指定为0，则使用默认值1024

默认值：0(1024)

### [compression\_codec](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_compression_codec.html)

当使用insert写入数据文件时，底层的压缩由compression\_codec控制

类型：string

值：SNAPPY(默认)、GZIP和NONE

示例：

set compression\_codec=gzip;

insert into parquet\_table\_highly\_compressed select \* from t1;

set compression\_codec=snappy;

insert into parquet\_table\_compression\_plus\_fast\_queries select \* from t1;

set compression\_codec=none;

insert into parquet\_table\_no\_compression select \* from t1;

set compression\_codec=foo;

select \* from t1 limit 5;

ERROR: Invalid compression codec: foo

### [debug\_action](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_debug_action.html)

用于cloudera内部故障诊断和测试

类型：string

默认值：empty string

### [default\_order\_by\_limit](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_default_order_by_limit.html)

为了避免对海量的结果进行排序，Impala在查询时若使用了order by则必须使用limit子句限定返回的数据条数。可以通过default\_order\_by\_limit参数设定允许排序之后返回的最大记录行数。若返回的行数小于设定的值，则无影响

默认值：-1(不设定上限)

### [disable\_codegen](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_disable_codegen.html)

该参数用于诊断系统崩溃的调试参数。若查询由于”illegal instruction”或其它特定的硬件错误导致查询失败，可设置disable\_codegen=true，再次运行有问题的查询。若查询只有在true时才能运行成功，则须向厂商提供详细的问题报告。在其它任何情况下，不得将该参数设为true，会增加一定的额外开销，导致性能问题

类型：布尔

默认值：false

### [explain\_level](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_explain_level.html)

该参数控制explain语句输出的信息量

类型：string或int

默认值：1

参数：范围0-3

0：即minimal，是执行计划的最小输出，包含执行计划最简要的信息。每个操作占用-行，使用该参数主要用来检查一个长sql中连接的顺序是否符合预期

1：即standard，是执行计划的标准输出，包含分布式查询执行的逻辑路径

2：即extednded，是执行计划扩展输出，包括查询优化器如何使用统计信息

3：即verbose，是执行计划最详细的输出，包括一个查询时如何被分配到各个节点，并通过管道进行连接等底层信息

示例：

显示不同级别下的explain输出的信息

[localhost:21000] > create table t1 (x int, s string);

[localhost:21000] > set explain\_level=1;

[localhost:21000] > explain select count(\*) from t1;

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

| Explain String |

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

| Estimated Per-Host Requirements: Memory=10.00MB VCores=1 |

| WARNING: The following tables are missing relevant table and/or column statistics. |

| explain\_plan.t1 |

| |

| 03:AGGREGATE [MERGE FINALIZE] |

| | output: sum(count(\*)) |

| | |

| 02:EXCHANGE [PARTITION=UNPARTITIONED] |

| | |

| 01:AGGREGATE |

| | output: count(\*) |

| | |

| 00:SCAN HDFS [explain\_plan.t1] |

| partitions=1/1 size=0B |

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

[localhost:21000] > explain select \* from t1;

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

| Explain String |

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

| Estimated Per-Host Requirements: Memory=-9223372036854775808B VCores=0 |

| WARNING: The following tables are missing relevant table and/or column statistics. |

| explain\_plan.t1 |

| |

| 01:EXCHANGE [PARTITION=UNPARTITIONED] |

| | |

| 00:SCAN HDFS [explain\_plan.t1] |

| partitions=1/1 size=0B |

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

[localhost:21000] > set explain\_level=2;

[localhost:21000] > explain select \* from t1;

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

| Explain String |

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

| Estimated Per-Host Requirements: Memory=-9223372036854775808B VCores=0 |

| WARNING: The following tables are missing relevant table and/or column statistics. |

| explain\_plan.t1 |

| |

| 01:EXCHANGE [PARTITION=UNPARTITIONED] |

| | hosts=0 per-host-mem=unavailable |

| | tuple-ids=0 row-size=19B cardinality=unavailable |

| | |

| 00:SCAN HDFS [explain\_plan.t1, PARTITION=RANDOM] |

| partitions=1/1 size=0B |

| table stats: unavailable |

| column stats: unavailable |

| hosts=0 per-host-mem=0B |

| tuple-ids=0 row-size=19B cardinality=unavailable |

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

[localhost:21000] > set explain\_level=3;

[localhost:21000] > explain select \* from t1;

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

| Explain String |

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

| Estimated Per-Host Requirements: Memory=-9223372036854775808B VCores=0 |

**| WARNING: The following tables are missing relevant table and/or column statistics. |**

**| explain\_plan.t1 |**

| |

| F01:PLAN FRAGMENT [PARTITION=UNPARTITIONED] |

| 01:EXCHANGE [PARTITION=UNPARTITIONED] |

| hosts=0 per-host-mem=unavailable |

| tuple-ids=0 row-size=19B cardinality=unavailable |

| |

| F00:PLAN FRAGMENT [PARTITION=RANDOM] |

| DATASTREAM SINK [FRAGMENT=F01, EXCHANGE=01, PARTITION=UNPARTITIONED] |

| 00:SCAN HDFS [explain\_plan.t1, PARTITION=RANDOM] |

| partitions=1/1 size=0B |

**| table stats: unavailable |**

**| column stats: unavailable |**

| hosts=0 per-host-mem=0B |

| tuple-ids=0 row-size=19B cardinality=unavailable |

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

收集表的统计信息

[localhost:21000] > compute stats t1;

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

| summary |

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

| Updated 1 partition(s) and 2 column(s). |

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

[localhost:21000] > explain select \* from t1;

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

| Explain String |

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

| Estimated Per-Host Requirements: Memory=-9223372036854775808B VCores=0 |

| |

| F01:PLAN FRAGMENT [PARTITION=UNPARTITIONED] |

| 01:EXCHANGE [PARTITION=UNPARTITIONED] |

| hosts=0 per-host-mem=unavailable |

| tuple-ids=0 row-size=20B cardinality=0 |

| |

| F00:PLAN FRAGMENT [PARTITION=RANDOM] |

| DATASTREAM SINK [FRAGMENT=F01, EXCHANGE=01, PARTITION=UNPARTITIONED] |

| 00:SCAN HDFS [explain\_plan.t1, PARTITION=RANDOM] |

| partitions=1/1 size=0B |

**| table stats: 0 rows total |**

**| column stats: all |**

| hosts=0 per-host-mem=0B |

| tuple-ids=0 row-size=20B cardinality=0 |

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

对于连接查询，或者其他的复杂查询，需要根据具体的情况确定执行计划的输出级别，并检查输出的执行计划信息。如下示例显示了一个三表连接的sql输出的执行计划，然后使用[shuffle]的hint将前两个表的连接方式从broadcast改为shuffle

[localhost:21000] > set explain\_level=1;

[localhost:21000] > explain select one.\*, two.\*, three.\* from t1 one, t1 two, t1 three where one.x = two.x and two.x = three.x;

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

| Explain String |

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

| Estimated Per-Host Requirements: Memory=4.00GB VCores=3 |

| |

| 07:EXCHANGE [PARTITION=UNPARTITIONED] |

| | |

**| 04:HASH JOIN [INNER JOIN, BROADCAST] |**

| | hash predicates: two.x = three.x |

| | |

**| |--06:EXCHANGE [BROADCAST] |**

| | | |

| | 02:SCAN HDFS [explain\_plan.t1 three] |

| | partitions=1/1 size=0B |

| | |

**| 03:HASH JOIN [INNER JOIN, BROADCAST] |**

| | hash predicates: one.x = two.x |

| | |

**| |--05:EXCHANGE [BROADCAST] |**

| | | |

| | 01:SCAN HDFS [explain\_plan.t1 two] |

| | partitions=1/1 size=0B |

| | |

| 00:SCAN HDFS [explain\_plan.t1 one] |

| partitions=1/1 size=0B |

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

[localhost:21000] > explain select one.\*, two.\*, three.\* from t1 one join [shuffle] t1 two join t1 three where one.x = two.x and two.x = three.x;

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

| Explain String |

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

| Estimated Per-Host Requirements: Memory=4.00GB VCores=3 |

| |

| 08:EXCHANGE [PARTITION=UNPARTITIONED] |

| | |

**| 04:HASH JOIN [INNER JOIN, BROADCAST] |**

| | hash predicates: two.x = three.x |

| | |

**| |--07:EXCHANGE [BROADCAST] |**

| | | |

| | 02:SCAN HDFS [explain\_plan.t1 three] |

| | partitions=1/1 size=0B |

| | |

**| 03:HASH JOIN [INNER JOIN, PARTITIONED] |**

| | hash predicates: one.x = two.x |

| | |

**| |--06:EXCHANGE [PARTITION=HASH(two.x)] |**

| | | |

| | 01:SCAN HDFS [explain\_plan.t1 two] |

| | partitions=1/1 size=0B |

| | |

**| 05:EXCHANGE [PARTITION=HASH(one.x)] |**

| | |

| 00:SCAN HDFS [explain\_plan.t1 one] |

| partitions=1/1 size=0B |

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

如下示例显示了前两个表如何被hash分布到各节点上，又将第三个表broadcast到所有节点上进行处理的过程

[localhost:21000] > set explain\_level=0;

[localhost:21000] > explain select one.\*, two.\*, three.\* from t1 one join [shuffle] t1 two join t1 three where one.x = two.x and two.x = three.x;

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

| Explain String |

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

| Estimated Per-Host Requirements: Memory=4.00GB VCores=3 |

| |

| 08:EXCHANGE [PARTITION=UNPARTITIONED] |

**| 04:HASH JOIN [INNER JOIN, BROADCAST] |**

**| |--07:EXCHANGE [BROADCAST] |**

| | 02:SCAN HDFS [explain\_plan.t1 three] |

**| 03:HASH JOIN [INNER JOIN, PARTITIONED] |**

**| |--06:EXCHANGE [PARTITION=HASH(two.x)] |**

| | 01:SCAN HDFS [explain\_plan.t1 two] |

**| 05:EXCHANGE [PARTITION=HASH(one.x)] |**

| 00:SCAN HDFS [explain\_plan.t1 one] |

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

### [hbase\_cache\_blocks](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_hbase_cache_blocks.html)

该参数相当于在Hbase Java应用程序中调用org.apache.hadoop.hbase.client.Scan的setCacheBlocks方法，可用于控制HBase region服务器的内存压力

类型：boolean

默认值：false

### [hbase\_caching](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_hbase_caching.html)

该参数相当于在Hbase Java应用程序中调用了org.apache.hadoop.hbase.client.Scan的setCaching方法，可用于控制HBase region服务器的内存压力

类型：boolean

默认值：0

### [max\_errors](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_max_errors.html)

在Impala日志文件中可记录的某个查询的非关键错误的最大个数。若指定为0或不指定，则使用默认值为1000.

默认值：0(1000)

### [max\_scan\_range\_length](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_max_scan_range_length.html)

扫描范围的最大长度。该参数和表的hdfs块数共同决定集群使用多少个CPU核来处理查询请求

该参数越小，就越增加并行度，要使用更多的CPU核进行运算，但若过小，也可能带来性能问题

该参数只对hdfs表起作用，对Parquet表无影响。

默认值：0

### [mem\_limit](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_mem_limit.html)

若没有启用资源管理功能，该参数用来定义每个节点的最大内存数量。若在查询过程中任何一节点超过了限制，impala则会自动终止查询。在执行查询过程中，Impala会定期检查各节点的内存是否超过限制。即使真正使用的内存超过了该参数的限制，只要在impala检查时没有超过，这个查询就不会被终止

若启用了资源管理其，且设置该参数值，它的值将覆盖Impala自动计算的内存值。Impala为每个从yarn请求该参数指定的内存数。只有足够的内存可用，查询才会进行。若没有设置该参数值，impala会为每一个查询计算每个节点需要的内存数量，然后从yarn请求相应大小的内存且初始化该参数。

默认值：0

### [num\_nodes](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_num_nodes.html)

仅在调试过程中限制节点数时使用。

参数：0、1

0：所有节点均参与运算

1：仅在协调节点上运行

### [num\_scanner\_threads](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_num_scanner_threads.html)

一个查询在每个节点上扫描线程的最大数量。默认情况下，节点由多少个CPU核就会启动多少个扫描线程。

默认值：0

### [parquet\_compression\_codec](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_parquet_compression_codec.html)

当Impala向parquet表插入数据时，底层的压缩算法由该参数控制。

参数：snappy(默认)、gzip、none

### [parquet\_file\_size](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_parquet_file_size.html)

在向impala的parquet表插入数据时，该参数指定parquet数据文件的最大大小。对于小表或者分区表，默认的parquet块为1GB。若要增大并行度，让数据尽可能分布在多个节点上，可以减少这个参数的值。

默认值：0(1GB)

### [query\_timeout\_s](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_query_timeout_s.html)

设置空闲会话查询超时值

默认值：0

### [yarn\_pool](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_request_pool.html)

查询需要提交的池或者队列的名称。该参数只有在启用了Impala准入控制特性或基于yarn的资源管理时有效

默认值：空

### [reservation\_request\_timeout (cdh 5 only)](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_reservation_request_timeout.html)

Impala为一个完全授权或者拒绝授权预留的最大毫秒数

默认值：300000(5分钟)

### [support\_start\_over](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_support_start_over.html)

保持该值为false，这是一个只读的设置,由色调等客户端应用程序进行测试

默认值：false

### [sync\_ddl](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_sync_ddl.html)

启用该参数会导致任何DDL语句必须通过Catalog服务将更新的元数据推送到所有节点之上。故后续若连接到某个节点上时，不用担心无法识别新创建的对象

默认值：false

### [v\_cpu\_cores (cdh 5 only)](http://www.cloudera.com/content/www/en-us/documentation/enterprise/latest/topics/impala_v_cpu_cores.html)

从yarn中申请的每台主机的虚拟CPU核数。若设置了该参数，Impala将使用该值覆盖Impala自动计算的值。

默认值：0(使用自动计算的值)

# Impala 存储

## 文件格式选择

Impala支持的文件格式如下

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |
| 文件类型 | 格式 | 压缩编码 | Impala是否可以直接创建 | Impala是否可以直接插入 |
| Parquet | 结构化 | Snappy,Gzip  当前默认Snappy | 是 | 是，create table、insert、load data或者查询均可 |
| Text | 非结构化 | LZO | 是，使用create table创建表是默认的文件格式，各字段通过ASCII码0x01分隔 | 是，create table，insert，load data或者查询均可。若使用了lzo方式压缩，必须在Hive完成创建表和加载数据的过程 |
| Avro | 结构化 | Snappy，Czip，deflate，bzip2 | 是 | 否，通过load data加载正确格式的文件，或者通过Hive使用insert方式加载 |
| RCfile | 结构化 | snappy，Gzip，deflate，Bzip2 | 是 | 否，通过load data加载正确格式的文件，或者通过Hive使用insert方式加载 |
| SequenceFile | 结构化 | Snappy，Gzip，deflate，Bzip2 | 是 | 否，通过load data加载正确格式的文件，或者通过Hive使用insert方式加载 |

不同的文件格式和压缩方式与具有不同特定的数据集能够更好地协同工作，提高性能。对于一个指定的表，应使用哪种文件格式，哪种压缩方式可以依据以下几点：

1. 若已存在的数据文件格式是被Impala支持的，则Impala表通常使用同样的文件格式，除非原始的文件格式的性能、资源使用率不能满足应用的要求，则须使用不同的文件格式和压缩方式来重建这张表。重建时，可以使用insert语句将所有的数据拷贝至新表中。
2. 通常情况下，很多工具可以直接生成文本文件，且文本文件因其易读性而方便校验和调试。通常是通过拷贝CSV或者TSV文件将数据加载到Impala中，然后使用insert....select的方式讲数据复制到基于其它数据格式的impala表中
3. 若自身的架构将被查询的数据存放在内存中，则不使用压缩。因为在内存中查询没有和磁盘交互的I/O消耗，同时又增加了对数据解压时CPU的负载

## Text

### 查询性能

以文本格式作为存储文件格式占用的体积较大，且不像Parquet这样的二进制文件格式效率高。若接受到的数据为文本格式，不想做进一步处理且使用文本格式就可以满足应用的需求的情况下，可选择改格式存储。若对性能要求很高，须尝试将文本格式替换为其它二进制文件格式

对于频繁查询的数据，可将原始的文本数据文件加载到Impala表中，然后使用insert语句将存储的文件格式转换为其它的文件格式。若目标表定义为特定的二进制文件格式，在数据插入的过程中转换会自动进行

对于更紧凑的数据，可对文本文件进行LZO压缩。由于LZO数据文件天然的可分割特性，可将同一个LZO压缩文件的不同部分发送到不同节点上并行运算

### 创建文本表

若不知道字段分隔符之类的文本数据的准确格式，我们可以使用不带后续子句的create table语句建表

create table my\_table(id int, s string, n int, t timestamp, b boolean);

默认情况下创建的文本格式的数据文件可以接收ctrl+A（OxO1）作为分隔符插入的列值

一种更通用的情况是将已存在的文本数据文件导入到Impala表。需要指定fields terminated by子句，且前面必须有对应的row format delimited子句，该语句以stored as textfile作为结束，也可不指定这个选项，因文本格式是Impala默认的文件格式

例如：

create table csv(id int, s string, n int, t timestamp, b boolean)

row format delimited

fields terminated by ',';

create table tsv(id int, s string, n int, t timestamp, b boolean)

row format delimited

fields terminated by '\t';

create table pipe\_separated(id int, s string, n int, t timestamp, b boolean)

row format delimited

fields terminated by '|'

stored as textfile;

也可以对具有特定分隔符的文本文件使用insert........select 语法将数据文件拷贝至Impala数据目录中

### 数据文件

当Impala对文本表进行查询时，会遍历这个表对应的数据目录中的所有数据文件，但会忽略以”.”开头的那些文件

通过Impala的insert语句插入数据时，底层会自动为数据文件生成一个唯一名称以避免和其它文件冲突

一个insert......select语句会为在该节点处理的select的数据产生一个数据文件。而每条insert....values语句将会产生一个单独的数据文件。Impala在对大数据文件查询的效率更高，所以强烈不建议使用insert....values的方式加载批量数据。若已经使用insert...values语句产生的大量小文件而导致的效率问题，必须通过insert...select的方式迁移到另一个表中

文本数据文件特殊值说明如下：

1. Impala对float和double数据列使用字符串inf表示无穷，使用nan表示不是一个数字
2. Impala将字符串”\N”表示为null，当使用sqoop时，需要指定--null-non-string和--null-string确保所有的null值可以被正确处理。默认情况下，sqoop使用字符串”null”表示一个null值，这可能导致Impala行转换错误。对于已经存在的数据文件的表，可以通过设置表的属性解决这个问题：

ALTER TABLEname SET TBLPROPERTIES("serialization.null.format"="null").)

### 加载数据

为了将已存在的文件加载到Impala文本中，可使用load data语句指定数据文件在hdfs上的路径，也可直接将文件拷贝到Impala数据目录中

为了将已存在的多个数据文件加载到Impala文本表中，可通过load data语句指定这些文件所在的路径，或直接将其直接拷贝到Impala数据目录中

若要将其他二进制格式的数据文件转换成文本文件，可使用如下sql语句，创建一张与二进制格式类似表结构的文本表：

CREATE TABLE csv LIKE other\_file\_format\_table;

指定文本表的字段分隔符

ALTER TABLE csv SET SERDEPROPERTIES ('serialization.format'=',', 'field.delim'=',');

将二进制格式表的数据以insert的方式插入文本表

INSERT INTO csv SELECT \* FROM other\_file\_format\_table;

### LZO压缩

Cloudera推荐在生产环境中对文本文件进行LZO压缩，尤其是对IO密集型的查询，使用LZO可大大减少从磁盘读取的数据量，但同时由于要在内存中解压数据，对CPU有额外的消耗

LZO压缩的文件可以透明地解压到不同的的节点进行分布式存储。目前Impala只能对LZO压缩的文本文件进行查询操作，若要进行写操作，必须通过Hive使用create table并加载数据，然后在使用Impala进行查询。一旦我们创建了以基于LZO压缩的文本表，就可手动地将通过LZO或其他类似命令产生的压缩文件直接放到Impala数据目录中

1. 使用LZO准备工作

在Impala中使用LZO文本表之前，必须在集群的每个节点上安装必要的安装包

1. 安装前的准备工作

对于使用包管理的Cloudera Manager或不被Cloudera Manager管理的系统，

需要为自己的操作系统下载匹配的GPL extras的repo文件

rhel5下载到/etc/profild.d/，地址：

<http://archive.cloudera.com/gplextras/redhat/5/x86_64/gplextras/cloudera-gplextras4.repo>

rhel6下载到/etc/profild.d/，地址：

<http://archive.cloudera.com/gplextras/redhat/6/x86_64/gplextras/cloudera-gplextras4.repo>

Suse下载到/etc/zypp/repos.d/，地址：

<http://archive.cloudera.com/gplextras/sles/11/x86_64/gplextras/cloudera-gplextras4.repo>

Ubuntu 10.04下载到/etc/apt/sources.list.d/，地址：

<http://archive.cloudera.com/gplextras/ubuntu/lucid/amd64/gplextras/cloudera.list>

Ubuntu 21.04下载到/etc/apt/sources.list.d/，地址：

<http://archive.cloudera.com/gplextras/ubuntu/precise/amd64/gplextras/cloudera.list>

Debian下载到/etc/apt/sources.list.d/，地址：

<http://archive.cloudera.com/gplextras/debian/squeeze/amd64/gplextras/cloudera.list>

1. 为Impala配置LZO

使用如下的安装命令，让Hadoop支持LZO，让Impala支持LZO

****For RHEL/CentOS systems:****

$ sudo yum update

$ sudo yum install hadoop-lzo-cdh5 # For clusters running CDH 5.

$ sudo yum install hadoop-lzo # For clusters running CDH 5 or higher.

$ sudo yum install impala-lzo

****For SUSE systems:****

$ sudo apt-get update

$ sudo zypper install hadoop-lzo-cdh5 # For clusters running CDH 5.

$ sudo zypper install hadoop-lzo # For clusters running CDH 5 or higher.

$ sudo zypper install impala-lzo

****For Debian/Ubuntu systems:****

$ sudo zypper update

$ sudo apt-get install hadoop-lzo-cdh5 # For clusters running CDH 5.

$ sudo apt-get install hadoop-lzo # For clusters running CDH 5 or higher.

$ sudo apt-get install impala-lzo

1. 修改配置文件

对于客户端和服务器端的core-site.xml配置文件中，添加属性

com.hadoop.compression.lzo.LzopCodec，例如：

<property>

<name>io.compression.codecs</name> <value>org.apache.hadoop.io.compress.DefaultCodec,org.apache.hadoop.io.compress.GzipCodec,

org.apache.hadoop.io.compress.BZip2Codec,org.apache.hadoop.io.compress.DeflateCodec,

org.apache.hadoop.io.compress.SnappyCodec,com.hadoop.compression.lzo.LzopCodec</value>

</property>

1. 重启MapReduce服务和Impala服务
2. 创建LZO文本表

一张LZO文本表必须使用如下的选项通过Hive创建：

STORED AS

INPUTFORMAT 'com.hadoop.mapred.DeprecatedLzoTextInputFormat'

OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'

而且某些Hive的本地设置也需修改：

hive> SET mapreduce.output.fileoutputformat.compress=true;

hive> SET hive.exec.compress.output=true;

hive> SET mapreduce.output.fileoutputformat.compress.codec=com.hadoop.compression.lzo.LzopCodec;

hive> CREATE TABLE lzo\_t (s string) STORED AS

> INPUTFORMAT 'com.hadoop.mapred.DeprecatedLzoTextInputFormat'

> OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat';

hive> INSERT INTO TABLE lzo\_t SELECT col1, col2 FROM uncompressed\_text\_table;

一旦我们创建了LZO文本表，我们可以在Hive中使用insert...select语句将其转换为其它格式的表

在通过Hive的insert插入数据后，我们需要检查表对应的HDFS上的数据目录，确保数据文件具有.lzo的扩展名。若之前的某些配置不正确，可能导致存储的文件没有被正确的压缩，使用Impala访问这张表将报错，因为Impala认为这是一张压缩过的LZO文本表

在将数据加载到LZO压缩表后，必须为文件建立索引。在建立索引的方式是使用Linux命令运行一个java类com.hadoop.compression.lzo.DistributedLzoIndexer。这个java类包含在hadoop-lzo包中

示例：

$ hadoop jar /usr/lib/hadoop/lib/hadoop-lzo-cdh4-0.4.15-gplextras.jar

com.hadoop.compression.lzo.DistributedLzoIndexer /hdfs\_location\_of\_table/

索引文件与数据文件的文件名相同，但扩展名为.index。若数据文件没有被索引，Impala查询仍然可以运行，但从远程DataNode读取数据的效率将非常低。

一旦创建了LZO压缩的文本表，将数据加进来之后，对数据文件做索引，可以通过Impala对该表进行查询。对通过Hive创建的表第一次查询时，需要使用invalidate metadata 命令更新元数据信息，让impala能够识别这张新的表

## Parquet

### 创建Parquet

若要创建一张Parquet表，需要在impala-shell命令行中指定表名，列名，数据类型以及存储的文件格式信息，示例：

[impala-host:21000] > create table parquet\_table\_name (x INT, y STRING) STORED AS PARQUET;

也可以克隆另一张parquet表的列名和数据类型信息

[impala-host:21000] > create table parquet\_table\_name LIKE other\_table\_name STORED AS PARQUET;

若有了表的定义信息，可以依据这个表定义创建一张外部表指向parquet数据文件，即可实现parquet数据文件的访问：

CREATE EXTERNAL TABLE ingest\_existing\_files LIKE PARQUET '/user/etl/destination/datafile1.dat'

STORED AS PARQUET

LOCATION '/user/etl/destination';

也可使用parquet数据文件的表定义信息创建一张空表，然后再使用insert或load data的方式向这张表中加载数据

CREATE TABLE columns\_from\_data\_file LIKE PARQUET '/user/etl/destination/datafile1.dat'

STORED AS PARQUET;

也可创建基于parquet的分区表

CREATE TABLE columns\_from\_data\_file LIKE PARQUET '/user/etl/destination/datafile1.dat'

PARTITION (year INT, month TINYINT, day TINYINT)

STORED AS PARQUET;

然后可以通过如下类似的命令插入数据

[impala-host:21000] > insert overwrite table parquet\_table\_name select \* from other\_table\_name;

### 加载数据

若源数据存在Impala表中，只是表的文件格式不是parquet的，只需要使用insert...select语句将其转换成parquet格式的表。在转换的过程中，可以进行过滤，重新分区等

当向分区表插入数据时，在insert语句中使用合适的hint能够大大的降低操作对资源的使用，提升性能

1. 使用insert向parquet分区表中插入数据由于资源限制导致失败或插入效率低下时可以使用这些hint
2. hint的关键字[shuffle][noshuffle]可以使用在partition子句之后，select关键字之前
3. [shuffle]会尽可能地降低并发写入hdfs的文件的数目，并且为每个分区分配1GB的内存作为缓存使用。这降低大大减少了insert操作的资源使用。在插入数据的过程中，由于同一分区的数据可能在不同的节点上进行插入操作，可能会引发节点间的数据传输

任何针对parquet表的insert操作都要确保hdfs上有足够的空间可用。parquet默认的块大小为1GB。若空间不足，可能导致insert操作失败

应尽量避免使用insert...values的方式向parquet表插入数据。因该语句会产生很多单独的小文件

### 查询性能

parquet表的查询性能依赖于我们需要select的列的数目，where条件的区分度等。parquet表适于那些表本身包含很多列，但仅对很少的一些列进行的查询。当对parquet进行分区后，性能会得到进一步提升。impala会忽略where条件中没有的分区，仅对指定的分区进行数据文件读取数据

### Snaapy/Gzip压缩

当使用insert向parquet表插入数据时，可以通过参数parquet\_compression\_codec控制压缩的方式。该参数允许的值有snappy，gzip和none。该值的参数可以使用大写或小写，若该参数使用了一个impala无法识别的值，所有的Impala查询都会报错

Snappy压缩示例

[localhost:21000] > create database parquet\_compression;

[localhost:21000] > use parquet\_compression;

[localhost:21000] > create table parquet\_snappy like raw\_text\_data;

[localhost:21000] > set COMPRESSION\_CODEC=snappy;

[localhost:21000] > insert into parquet\_snappy select \* from raw\_text\_data;

Inserted 1000000000 rows in 181.98s

gzip压缩示例：

[localhost:21000] > create table parquet\_gzip like raw\_text\_data;

[localhost:21000] > set COMPRESSION\_CODEC=gzip;

[localhost:21000] > insert into parquet\_gzip select \* from raw\_text\_data;

Inserted 1000000000 rows in 1418.24s

不压缩示例：

[localhost:21000] > create table parquet\_none like raw\_text\_data;

[localhost:21000] > set COMPRESSION\_CODEC=none;

[localhost:21000] > insert into parquet\_none select \* from raw\_text\_data;

Inserted 1000000000 rows in 146.90s

parquet压缩比例测试

$ hdfs dfs -du -h /user/hive/warehouse/parquet\_compression.db

23.1 G /user/hive/warehouse/parquet\_compression.db/parquet\_snappy

13.5 G /user/hive/warehouse/parquet\_compression.db/parquet\_gzip

32.8 G /user/hive/warehouse/parquet\_compression.db/parquet\_none

[localhost:21000] > desc parquet\_snappy;

Query finished, fetching results ...

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

| name | type | comment |

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

| id | int | |

| val | int | |

| zfill | string | |

| name | string | |

| assertion | boolean | |

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

Returned 5 row(s) in 0.14s

[localhost:21000] > select avg(val) from parquet\_snappy;

Query finished, fetching results ...

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

| \_c0 |

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

| 250000.93577915 |

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

Returned 1 row(s) in 4.29s

[localhost:21000] > select avg(val) from parquet\_gzip;

Query finished, fetching results ...

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

| \_c0 |

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

| 250000.93577915 |

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

Returned 1 row(s) in 6.97s

[localhost:21000] > select avg(val) from parquet\_none;

Query finished, fetching results ...

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

| \_c0 |

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

| 250000.93577915 |

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

Returned 1 row(s) in 3.67s

### 与其他组件交换Parquet数据文件

可以通过Hive，Pig或者MapReduce读写parquet数据文件。对于Parquet表，Impala支持标量数据类型，但不支持Map，Array这样的组合类型好内嵌类型。若Parquet表的列使用了支持的数据类型，Impala就无法正确访问。

若需要拷贝Parquet数据文件，一定要使用hadoop distcp -pb命令保证拷贝过程中保持最原始数据块的大小。也可使用命令hdfs fsck -blocks HDFS\_path\_of\_impala\_table\_dir检查目标文件是否保持了原始文件的大小

### Parquet数据文件的组织方式

虽然parquet数据文件是面向列存储的，但也不是一个列一个数据文件，parquet将一行所有字段都存在同一个数据文件中。一个parquet数据文件的最大大小是1GB，要确保IO和网络的参数与这个大小匹配

在这个1GB的数据文件中，所有的行被重新组织，所有行的第一列的值相邻存放，然后是第二列，第三列，以此推类。把同一列的值集中存放，因数据类型都是一样的，所有也使针对单列的压缩更有效率

当Impala需要读取某列数据时，只需打开数据文件只读取连续被才存放在一起的该列相关的数据就可以了。若要查询或where条件中还出现了其它列，也可在同一个数据文件中找到。

若insert插入的数据没有1GB，那么最终的数据文件可能小一点。若使用多个etl作业来进行insert操作，需要尽量保证数据量是1Gb左右

1. RLE和数据字典编码

基于生产环境的真实数据值，parquet自动采用想游程编码(RLE)，或者数据字典编码方式对数据进行自动压缩。在数据以上述的技术对值进行相对紧凑的编码之后，还可以进一步使用压缩算法对其进行压缩。parquet支持三种方式：Snappy，Gzip和无压缩。

Impala对parquet文件中的数据值使用RLE或者数据字典进行压缩编码，对整个parquet数据文件本身使用Snappy或者Gzip进行压缩。

1. 压缩数据文件

若使用已有的表结构或者针对parquet表的ETL过程，可能会遇到很多小数据文件的情况，将会降低查询效率。以下两种情况可能产生此种情况：

·源数据数据量过小

·单个分区包含的数据量过小

1. 模式进化

这里的模式进化指的是使用alter table...replace columns来改变parquet表的数据类型、列数等，可以执行的变更包括以下几种：

·Impala中的alter table语句从来不会改变对应的数据文件本身，从Impala的视角来看，模式进化就是使用新的表的对应来解释同一个数据文件而已。

·insert语句总是依据最新的表定义进行插入操作。若在insert的过程中改变了列的定义可能会导致底层数据的存储依据了不同的表定义

·若通过alter table ....replace columns为表定义了数据文件中没有的列，则在查询中会自动忽略数据文件中多余的列的值

·parquet使用32位的整型值来存储tinyint，smallint，int类型的值

这意味着可很容易进行tinyint，smallint，int各类型之间的转换，因在底层数据的存储是相同的。

若将一个范围大的类型强制转换为一个范围小的类型，仍然可以产生溢出错误。不能将tinyint，smallint，int转换为bigint。即使执行alter table时可以成功，但在真正查询时仍会报错

同样，其它类型的转换都可能引起查询时的转换错误。