Kudu 集成 Apache Impala

- 使用Apache Kudu与Apache Impala（孵化）
- 要求
- 配置
- 使用 Impala Shell
  - 内部和外部 Impala 表
    - Internal（内部）
    - External（外部）
  - 查询 Impala 中现有的 Kudu 表
  - 从 Impala 创建一个新的 Kudu 表
    - 指定 Tablet Partitioning（Tablet 分区）
    - Impala 数据库和 Kudu
    - 不支持 Kudu 表的 Impala 关键字
  - 优化评估 SQL 谓词的性能
  - 分区表
    - 基本分区
    - 高级分区
    - Partitioning Rules of Thumb（拇指分区规则）
  - 将数据插入 Kudu 表
    - 插入单个值
    - 批量插入
    - INSERT and Primary Key Uniqueness Violations（插入和主键唯一性违规）
  - 更新行
    - 批量更新
  - 删除行
    - 批量删除
  - INSERT，UPDATE 和 DELETE 操作期间的故障
  - 更改表属性
  - 使用 Impala 删除 Kudu 表
- 下一步做什么？
  - 已知问题和限制

原文链接：http://kudu.apache.org/docs/kudu_impala_integration.html

译文链接：http://cwiki.apachecn.org/pages/viewpage.action?pageId=10813620

贡献者：小瑶 ApacheCN Apache中文网

使用Apache Kudu与Apache Impala（孵化）

Kudu 与 Apache Impala（孵化）紧密集成，允许您使用 Impala 使用 Impala 的 SQL 语法从 Kudu tablets 插入，查询，更新和删除数据，作为使用 Kudu API 构建自定义 Kudu 应用程序的替代方法。此外，您可以使用 JDBC 或 ODBC 将使用任何语言，框架或商业智能工具编写的现有或新应用程序与使用 Impala 作为代理的 Kudu 数据进行连接。

要求

- 此文档特定于 Impala 的某些版本。描述的语法将仅在以下版本中运行：
  * CDH 5.10 附带的 Impala 2.7.0 版本。 SELECT VERSION() 将报告 impalad version 2.7.0-cdh5.10.0 。
  * 从源代码编译的 Apache Impala 2.8.0 版本。 SELECT VERSION() 将报告 impalad version 2.8.0 。
  较早版本的 Impala 2.7（包括以前提供的特殊 IMPALA_KUDU 版本）具有不兼容的语法。未来版本可能与此语法兼容，但我们建议您检查这是与您安装的相应版本相对应的最新可用文档。
- 本文档不描述 Impala 安装过程。请参阅 Impala 文档，并确保您可以在继续之前对 HDFS 上的 Impala 表运行简单查询。

配置

Kudu 内不需要进行配置更改，从而可以访问 Impala 。

虽然不是绝对必要的，但建议您配置 Impala 与 Kudu Master servers 的位置：

- 在 Impala 服务配置中设置 --kudu_master_hosts = <master1> [：port] ， <master2> [：port] ， <master3> [：port] 标志。如果您正在使用 Cloudera Manager ，请参阅相应的 Cloudera Manager 文档。

如果在 Impala 服务中未设置此标志，则每次创建表格时都必须手动提供此配置，方法是在 TBLPROPERTIES 子句中指定 kudu_master_addresses 属性。

本指南的其余部分假设配置已设置。

使用 Impala Shell

> 注意
> 这只是 Impala Shell 功能的一小部分。有关详细信息，请参阅 Impala Shell 文档。

- 使用 impala-shell 命令启动 Impala Shell 。默认情况下， impala-shell 尝试连接到端口 21000 上的 localhost 上的 Impala 守护程序。要连接到其他主机，请使用 -i <host:port> 选项。要自动连接到特定的 Impala 数据库，请使用 -d <database> 选项。例如，如果您的所有 Kudu 表都位于 Impala_kudu 数据库中的 Impala 中，请使用 -d impala_kudu 来使用此数据库。
- 要退出 Impala Shell，请使用以下命令：quit;

内部和外部 Impala 表

使用 Impala 创建新的 Kudu 表时，可以将表创建为内部表或外部表。

Internal (内部)

内部表由 Impala 管理，当您从 Impala 中删除时，数据和表确实被删除。当您使用 Impala 创建新表时，通常是内部表。

External (外部)

外部表（由 CREATE EXTERNAL TABLE 创建）不受 Impala 管理，并且删除此表不会将表从其源位置（此处为 Kudu ）丢弃。相反，它只会去除 Impala 和 Kudu 之间的映射。这是 Kudu 提供的用于将现有表映射到 Impala 的语法。

有关内部和外部表的更多信息，请参阅 Impala文档 。

查询 Impala 中现有的 Kudu 表

通过 Kudu API 或其他集成（如 Apache Spark ）创建的表不会在 Impala 中自动显示。要查询它们，您必须先在 Impala 中创建外部表以将 Kudu 表映射到 Impala 数据库中：

```
CREATE EXTERNAL TABLE my_mapping_table
STORED AS KUDU
TBLPROPERTIES (
  'kudu.table_name' = 'my_kudu_table'
);
```

从 Impala 创建一个新的 Kudu 表

从 Impala 在 Kudu 中创建一个新表类似于将现有的 Kudu 表映射到 Impala 表，但您需要自己指定模式和分区信息。

使用以下示例作为指导。 Impala 首先创建表，然后创建映射。

```
CREATE TABLE my_first_table
(
  id BIGINT,
  name STRING,
  PRIMARY KEY(id)
)
PARTITION BY HASH PARTITIONS 16
STORED AS KUDU;
```

在 CREATE TABLE 语句中，必须首先列出构成主键的列。此外，主键列隐式标记为 NOT NULL 。

创建新的 Kudu 表时，需要指定一个分配方案。请参阅 分区表。为了简单起见，上面的表创建示例通过散列 id 列分成 16 个分区。有关分区的指导，请参阅 Thumb 的分区规则。

```
CREATE TABLE AS SELECT
```

您可以使用 CREATE TABLE ... AS SELECT 语句查询 Impala 中的任何其他表或表来创建表。以下示例将现有表 old_table 中的所有行导入到 Kudu 表 new_table 中。 new_table 中的列的名称和类型将根据 SELECT 语句的结果集中的列确定。请注意，您必须另外指定主键和分区。

```
CREATE TABLE new_table
PRIMARY KEY (ts, name)
PARTITION BY HASH(name) PARTITIONS 8
STORED AS KUDU
AS SELECT ts, name, value FROM old_table;
```

指定 Tablet Partitioning（Tablet 分区）

表分为每个由一个或多个 tablet servers 提供的 tablets 。理想情况下，tablets 应该相对平等地拆分表的数据。 Kudu 目前没有自动（或手动）拆分预先存在的 tablets 的机制。在实现此功能之前，您必须在创建表时指定分区。在设计表格架构时，请考虑使用主键，您可以将表拆分成以类似速度增长的分区。使用 Impala 创建表时，可以使用 PARTITION BY 子句指定分区：

> 注意
> Impala 关键字（如 group ）在关键字意义上不被使用时，由背面的字符包围。

```
CREATE TABLE cust_behavior (
  _id BIGINT PRIMARY KEY,
  salary STRING,
  edu_level INT,
  usergender STRING,
  `group` STRING,
  city STRING,
  postcode STRING,
  last_purchase_price FLOAT,
  last_purchase_date BIGINT,
  category STRING,
  sku STRING,
  rating INT,
  fulfilled_date BIGINT
)
PARTITION BY RANGE (_id)
(
    PARTITION VALUES < 1439560049342,
    PARTITION 1439560049342 <= VALUES < 1439566253755,
    PARTITION 1439566253755 <= VALUES < 1439572458168,
    PARTITION 1439572458168 <= VALUES < 1439578662581,
    PARTITION 1439578662581 <= VALUES < 1439584866994,
    PARTITION 1439584866994 <= VALUES < 1439591071407,
    PARTITION 1439591071407 <= VALUES
)
STORED AS KUDU;
```

如果您有多个主键列，则可以使用元组语法指定分区边界：（'va'，1），（'ab'，2）。该表达式必须是有效的 JSON 。

Impala 数据库和 Kudu

每个 Impala 表都包含在称为数据库的命名空间中。默认数据库称为默认数据库，用户可根据需要创建和删除其他数据库。

当从 Impala 中创建一个受管 Kudu 表时，相应的 Kudu 表将被命名为 my_database :: table_name 。

不支持 Kudu 表的 Impala 关键字

创建 Kudu 表时不支持以下 Impala 关键字： - PARTITIONED - LOCATION - ROWFORMAT

优化评估 SQL 谓词的性能

如果您的查询的 WHERE 子句包含与 operators = ， <= ， '\' ， '\' ， > = ， BETWEEN 或 IN 的比较，则 Kudu 直接评估该条件，只返回相关结果。这提供了最佳性能，因为 Kudu 只将相关结果返回给 Impala 。对于谓词 != ， LIKE 或 Impala 支持的任何其他谓词类型， Kudu 不会直接评估谓词，而是将所有结果返回给 Impala ，并依赖于 Impala 来评估剩余的谓词并相应地过滤结果。这可能会导致性能差异，这取决于评估 WHERE 子句之前和之后的结果集的增量。

分区表

根据主键列上的分区模式将表格划分为 tablets 。每个 tablet 由至少一台 tablet server 提供。理想情况下，一张表应该分成多个 tablets 中分布的 tablet servers ，以最大化并行操作。您使用的分区模式的详细信息将完全取决于您存储的数据类型和访问方式。关于 Kudu 模式设计的全面讨论，请参阅 Schema Design。

Kudu 目前没有在创建表之后拆分或合并 tablets 的机制。创建表时，必须为表提供分区模式。在设计表格时，请考虑使用主键，这样您就可以将表格分为以相同速率增长的 tablets 。

您可以使用 Impala 的 PARTITION BY 关键字对表进行分区，该关键字支持 RANGE 或 HASH 分发。分区方案可以包含零个或多个 HASH 定义，后面是可选的 RANGE 定义。 RANGE 定义可以引用一个或多个主键列。基本 和 高级分区 的示例如下所示。

基本分区

PARTITION BY RANGE (按范围划分)

您可以为一个或多个主键列指定范围分区。 Kudu 中的范围分区允许根据所选分区键的特定值或值的范围拆分表。这样可以平衡并行写入与扫描效率。

假设您有一个具有列 state ，name 和 purchase_count 的表。以下示例创建 50 个 tablets ，每个 US state 。

> 注意
> 单调增加 Values
>
> 如果您在其值单调递增的列上按范围进行分区，则最后一个 tablet 的增长将远大于其他平台。此外，插入的所有数据将一次写入单个 tablet ，限制了数据摄取的可扩展性。在这种情况下，请考虑通过 HASH 分配，而不是或除 RANGE 之外。

```
CREATE TABLE customers (
  state STRING,
  name STRING,
  purchase_count int,
  PRIMARY KEY (state, name)
)
PARTITION BY RANGE (state)
(
  PARTITION VALUE = 'al',
  PARTITION VALUE = 'ak',
  PARTITION VALUE = 'ar',
  -- ... etc ...
  PARTITION VALUE = 'wv',
  PARTITION VALUE = 'wy'
)
STORED AS KUDU;
```

PARTITION BY HASH (哈希分区)

您可以通过散列分发到特定数量的 “buckets”

，而不是通过显式范围进行分发，或与范围分布组合。您指定要分区的主键列，以及要使用的存储桶数。通过散列指定的键列来分配行。假设散列的值本身不会表现出显着的偏差，这将有助于将数据均匀地分布在数据桶之间。

您可以指定多个定义，您可以指定使用复合主键的定义。但是，在多个散列定义中不能提及一列。考虑两列a和b：√ * HASH(a)，HASH(b)* √ HASH(a，b)* × HASH(a)，HASH(a，b)

> 注意
> 没有指定列的 HASH 的 PARTITION BY HASH 是通过散列所有主键列来创建所需数量的桶的快捷方式。

如果主键值均匀分布在其域中，并且数据偏移不明显，例如 timestamps（时间戳）或 IDs（序列号），则哈希分区是合理的方法。

以下示例通过散列 id 和 sku 列创建 16 个 tablets 。这传播了所有 16 个 tablets 的写作。在这个例子中，对一系列 sku 值的查询可能需要读取所有 16 个 tablets ，因此这可能不是此表的最佳模式。有关扩展示例，请参阅 高级分区。

```
CREATE TABLE cust_behavior (
  id BIGINT,
  sku STRING,
  salary STRING,
  edu_level INT,
  usergender STRING,
  `group` STRING,
  city STRING,
  postcode STRING,
  last_purchase_price FLOAT,
  last_purchase_date BIGINT,
  category STRING,
  rating INT,
  fulfilled_date BIGINT,
  PRIMARY KEY (id, sku)
)
PARTITION BY HASH PARTITIONS 16
STORED AS KUDU;
```

高级分区

您可以组合 HASH 和 RANGE 分区来创建更复杂的分区模式。您可以指定零个或多个 HASH 定义，后跟零个或一个 RANGE 定义。每个定义可以包含一个或多个列。虽然枚举每个可能的分发模式都超出了本文档的范围，但是几个例子说明了一些可能性。

PARTITION BY HASH and RANGE

考虑上面的 简单哈希 示例，如果您经常查询一系列 sku 值，可以通过将哈希分区与范围分区相结合来优化示例。

以下示例仍然创建了16个 tablets ，首先将 id 列分为 4 个存储区，然后根据 sku 字符串的值应用范围划分将每个存储区分为四个数据块。至少四片（最多可达16张）。当您查询相邻范围的 sku 值时，您很有可能只需从四分之一的 tablets 中读取即可完成查询。

> 注意
> 默认情况下，使用 PARTITION BY HASH 时，整个主键是散列的。要只对主键进行散列，可以使用像 PARTITION BY HASH(id, sku) 这样的语法来指定它。

```
CREATE TABLE cust_behavior (
  id BIGINT,
  sku STRING,
  salary STRING,
  edu_level INT,
  usergender STRING,
  `group` STRING,
  city STRING,
  postcode STRING,
  last_purchase_price FLOAT,
  last_purchase_date BIGINT,
  category STRING,
  rating INT,
  fulfilled_date BIGINT,
  PRIMARY KEY (id, sku)
)
PARTITION BY HASH (id) PARTITIONS 4,
RANGE (sku)
(
  PARTITION VALUES < 'g',
  PARTITION 'g' <= VALUES < 'o',
  PARTITION 'o' <= VALUES < 'u',
  PARTITION 'u' <= VALUES
)
STORED AS KUDU;
```

Multiple **`PARTITION BY HASH`** Definitions (多重划分由 HASH 定义)

再次扩展上述示例，假设查询模式将是不可预测的，但您希望确保写入分布在大量 tablets 上您可以通过在主键列上进行散列来实现整个主键的最大分配。

```
CREATE TABLE cust_behavior (
  id BIGINT,
  sku STRING,
  salary STRING,
  edu_level INT,
  usergender STRING,
  `group` STRING,
  city STRING,
  postcode STRING,
  last_purchase_price FLOAT,
  last_purchase_date BIGINT,
  category STRING,
  rating INT,
  fulfilled_date BIGINT,
  PRIMARY KEY (id, sku)
)
PARTITION BY HASH (id) PARTITIONS 4,
             HASH (sku) PARTITIONS 4
STORED AS KUDU;
```

该示例创建16个分区。您也可以使用 HASH(id,sku) PARTITIONS 16.但是，对于 sku 值的扫描几乎总是会影响所有16个分区，而不是可能限制为 4 。

Non-Covering Range Partitions (不覆盖分区)

Kudu 1.0 及更高版本支持使用非覆盖范围分区，其解决方案如下：

- 没有未覆盖的范围分区，在需要考虑不断增加的主键的时间序列数据或其他模式的情况下，服务于旧数据的 tablet 的大小相对固定，而接收新数据的 tablets 将不受限制地增长。
- 在您希望根据其类别（如销售区域或产品类型）对数据进行分区的情况下，无需覆盖范围分区，则必须提前了解所有分区，或者如果需要添加或删除分区，请手动重新创建表。 例如引入或消除产品类型。

非覆盖范围分区有一些注意事项。请务必阅读链接：/docs/schema_design.html [Schema Design guide]。

此示例每年创建一个 tablet （ 共5个 tablets ），用于存储日志数据。该表仅接受 2012 年至 2016年 的数据。这些范围之外的键将被拒绝。

```
CREATE TABLE sales_by_year (
  year INT, sale_id INT, amount INT,
  PRIMARY KEY (sale_id, year)
)
PARTITION BY RANGE (year) (
  PARTITION VALUE = 2012,
  PARTITION VALUE = 2013,
  PARTITION VALUE = 2014,
  PARTITION VALUE = 2015,
  PARTITION VALUE = 2016
)
STORED AS KUDU;
```

当记录开始进入 2017 年时，他们将被拒绝。在这一点上， 2017 年的范围应该如下：

```
ALTER TABLE sales_by_year ADD RANGE PARTITION VALUE = 2017;
```

在需要数据保留的滚动窗口的情况下，范围分区也可能会丢弃。例如，如果不再保留 2012 年的数据，则可能会批量删除数据：

```
ALTER TABLE sales_by_year DROP RANGE PARTITION VALUE = 2012;
```

请注意，就像删除表一样，这不可逆转地删除存储在丢弃的分区中的所有数据。

Partitioning Rules of Thumb (拇指分区规则)

- 对于大型表格，如事实表，目标是在集群中拥有核心数量的 tablets 。
- 对于小型表格（如维度表），目标是足够数量的 tablets ，每个 tablets 的大小至少为 1 GB 。

一般来说，请注意，在当前的实现中， tablets 的数量限制了读取的并行性。增加 tablets 数量超过核心数量可能会有减少的回报。

将数据插入 Kudu 表

Impala 允许您使用标准 SQL 语句将数据插入 Kudu 。

插入单个值

此示例插入单个行。

```
INSERT INTO my_first_table VALUES (99, "sarah");
```

此示例使用单个语句插入三行。

```
INSERT INTO my_first_table VALUES (1, "john"), (2, "jane"), (3, "jim");
```

批量插入

批量插入时，至少有三种常用选择。每个可能有优点和缺点，具体取决于您的数据和情况。

Multiple single **`INSERT`** statements (多个单独的 INSERT 语句)

这种方法具有易于理解和实现的优点。这种方法可能是低效的，因为 Impala 与 Kudu 的插入性能相比具有很高的查询启动成本。这将导致相对较高的延迟和较差的吞吐量。

Single INSERT statement with multiple VALUES (具有多个 VALUES 的单个 INSERT 语句)

如果包含超过 1024 个 VALUES 语句，则在将请求发送到 Kudu 之前，Impala 将其分组为 1024 （或 batch_size 的值）。通过在 Impala 方面摊销查询启动处罚，此方法可能会执行比多个顺序 INSERT 语句略好。要设置当前 Impala Shell 会话的批量大小，请使用以下语法：set batch_size = 10000;

> 注意
> 增加 Impala 批量大小会导致 Impala 使用更多的内存。您应该验证对群集的影响并进行相应调整。

Batch Insert (批量插入)

从 Impala 和 Kudu 的角度来看，通常表现最好的方法通常是使用 Impala 中的 SELECT FROM 语句导入数据。

1. 如果您的数据尚未在 Impala 中，则一种策略是 从文本文件 (如 TSV 或 CSV 文件) 导入。
2. 创建 Kudu 表，注意指定为主键的列不能为空值。
3. 通过查询包含原始数据的表将值插入到 Kudu 表中，如以下示例所示：

```
INSERT INTO my_kudu_table
  SELECT * FROM legacy_data_import_table;
```

Ingest using the C++ or Java API (采用C ++ 或 Java API)

在许多情况下，适当的摄取路径是使用 C ++ 或 Java API 直接插入到 Kudu 表中。与其他 Impala 表不同，通过 API 插入到 Kudu 表中的数据可用于 Impala 中的查询，而不需要任何 INVALIDATE METADATA 语句或其他 Impala 存储类型所需的其他语句。

`INSERT` and Primary Key Uniqueness Violations (插入和主键唯一性违规)

在大多数关系数据库中，如果您尝试插入已插入的行，则插入将失败，因为主键将被重复。请参阅 INSERT，UPDATE 和 DELETE 操作中的故障。然而，Impala 不会失败查询。相反，它会生成一个警告，但是继续执行 insert 语句的其余部分。

如果插入的行意图替换现有行，则可以使用 UPSERT 而不是 INSERT 。

```
INSERT INTO my_first_table VALUES (99, "sarah");
UPSERT INTO my_first_table VALUES (99, "zoe");
-- the current value of the row is 'zoe'
```

更新行

```
UPDATE my_first_table SET name="bob" where id = 3;
```

> 注意
> 当目标表在 Kudu 时，UPDATE 语句仅适用于 Impala 。

批量更新

您可以使用批量插入中相同的方法 批量更新 。

```
UPDATE my_first_table SET name="bob" where age > 10;
```

删除行

```
DELETE FROM my_first_table WHERE id < 3;
```

您也可以使用更复杂的语法进行删除。 FROM 子句中的逗号是 Impala 指定连接查询的一种方法。有关 Impala 连接的更多信息，请参阅http://www.cloudera.com/content/cloudera/en/documentation/core/latest/topics/impala_joins.html。

```
DELETE c FROM my_second_table c, stock_symbols s WHERE c.name =
s.symbol;
```

> 注意
> 当目标表在 Kudu 时，DELETE 语句仅适用于 Impala 。

批量删除

您可以使用 “插入批量” 中概述的相同方法 批量删除 。

```
DELETE FROM my_first_table WHERE id < 3;
```

INSERT，UPDATE 和 DELETE 操作期间的故障

INSERT ，UPDATE 和 DELETE 语句不能被视为整体事务。如果这些操作中的一个无法部分通过，则可能已经创建了密钥（在 INSERT 的情况下），或者记录可能已被其他进程修改或删除（在 UPDATE 或 DELETE 的情况下）。您应该设计您的应用程序。

更改表属性

您可以通过更改表的属性来更改 Impala 与给定 Kudu 表相关的元数据。这些属性包括表名， Kudu 主地址列表，以及表是否由 Impala （内部）或外部管理。

Rename an Impala Mapping Table（重命名 Impala 映射表）

```
ALTER TABLE my_table RENAME TO my_new_table;
```

注意
使用 ALTER TABLE ... RENAME 语句重命名表仅重命名 Impala 映射表，无论该表是内部还是外部表。这样可以避免可能访问基础的 Kudu 表的其他应用程序的中断。

Rename the underlying Kudu table for an internal table (重新命名内部表的基础 Kudu 表)

如果表是内部表，则可以通过更改 kudu.table_name 属性重命名底层的 Kudu 表：

```
ALTER TABLE my_internal_table
SET TBLPROPERTIES('kudu.table_name' = 'new_name')
```

Remapping an external table to a different Kudu table (将外部表重新映射到不同的 Kudu 表)

如果另一个应用程序在 Impala 下重命名了 Kudu 表，则可以重新映射外部表以指向不同的 Kudu 表名称。

```
ALTER TABLE my_external_table_
SET TBLPROPERTIES('kudu.table_name' = 'some_other_kudu_table')
```

Change the Kudu Master Address (更改 Kudu Master 地址)

```
ALTER TABLE my_table
SET TBLPROPERTIES('kudu.master_addresses' =
'kudu-new-master.example.com:7051');
```

Change an Internally-Managed Table to External (将内部管理的表更改为外部)

```
ALTER TABLE my_table SET TBLPROPERTIES('EXTERNAL' = 'TRUE');
```

使用 Impala 删除 Kudu 表

如果表是使用 Impala 中的内部表创建的，则使用 CREATE TABLE ，标准 DROP TABLE 语法会删除底层的 Kudu 表及其所有数据。如果表被创建为一个外部表，使用 CREATE EXTERNAL TABLE，Impala 和 Kudu 之间的映射被删除，但 Kudu 表保持原样，并包含其所有数据。

```
DROP TABLE my_first_table;
```

下一步做什么？

上面的例子只是探索了 Impala Shell 所能做的一小部分。

- 了解 Impala 项目
- 阅读 Impala 文档
- 查看 Impala SQL 参考
- 阅读 Impala 内部部分或了解如何在 Impala Wiki 上为 Impala 做出贡献。
- 阅读原生的 Kudu API 。

已知问题和限制

- 在使用 Impala 中的外部表格时，必须为具有大写字母或非 ASCII 字符的名称的 Kudu 表分配备用名称。
- 具有包含大写字母或非 ASCII 字符的列名称的 Kudu 表可能不会用作 Impala 中的外部表。专栏可能在 Kudu 更名为解决这个问题。
- 创建 Kudu 表时，CREATE TABLE 语句必须在主键顺序中包含其他列之间的主键列。
- 包含 UNIXTIME_MICROS 类型列的库杜表可能不会用作 Impala 中的外部表。
- Impala 不能使用 TIMESTAMP ， DECIMAL ， VARCHAR 或嵌套类型的列创建 Kudu 表。
- Impala 无法更新主键列中的值。
- NULL，NOT NULL，!= 和 LIKE 谓词不会被推送到 Kudu ，而是会被 Impala 扫描节点评估。这可能会降低相对于其他类型谓词的性能。
- 通过 Impala 的更新，插入和删除是非事务性的。如果查询失败的一部分途径，其部分效果将不会回滚。
- 单个查询的最大并行度仅限于表中的 tablets 数量。为了获得良好的分析性能，针对每个主机10个或更多个 tablets 或使用大型表。