|  |
| --- |
| Close-up image showing the leaf-sides of two oversized books side-by-side on a bookshelf, with additional books in soft focus background |
| HIVE IN HAND  HIVE WITH TALEND TUTORIAL NOTES |
| |  |  |  | | --- | --- | --- | | Sourav Nayak | 9/14/17 | Hive With TALEND | |

**Hive In Hand**

Contents

[Introduction to Hive 3](#_Toc495108741)

[How can we achieve ACID in HIVE? 3](#_Toc495108742)

[What is ACID? 3](#_Toc495108743)

[Transactions in Hive 3](#_Toc495108744)

[Row-level Transactions Available in Hive 0.14 4](#_Toc495108745)

[Creating a Table That Supports Hive Transactions 4](#_Toc495108746)

[Inserting Data into a Hive Table 5](#_Toc495108747)

[Updating the Data in Hive Table 7](#_Toc495108748)

[Deleting a Row from Hive Table 9](#_Toc495108749)

[Top 15 Hive Interview Questions 10](#_Toc495108750)

[1.What is Hive? 10](#_Toc495108751)

[2.What is Hive a metastore? 10](#_Toc495108752)

[3.What is the limitation of Derby database for Hive metastore? 10](#_Toc495108753)

[4.What are managed and external tables? 10](#_Toc495108754)

[5.What are the complex data types in Hive? 10](#_Toc495108755)

[MAP 10](#_Toc495108756)

[STRUCT 10](#_Toc495108757)

[ARRAY 11](#_Toc495108758)

[UNIONTYPE 11](#_Toc495108759)

[6.How does partitioning help in the faster execution of queries? 11](#_Toc495108760)

[7.How to enable dynamic partitioning in Hive? 11](#_Toc495108761)

[8.How does bucketing help in the faster execution of queries? 11](#_Toc495108762)

[9.How to enable bucketing in Hive? 11](#_Toc495108763)

[10.Which method has to be overridden when we use custom UDF in Hive? 12](#_Toc495108764)

[11.What are the different file formats in Hive? 12](#_Toc495108765)

[12.How is SerDe different from File format in Hive? 12](#_Toc495108766)

[13.What is RegexSerDe? 12](#_Toc495108767)

[14.How is ORC file format optimized for data storage and analysis? 12](#_Toc495108768)

[15.How to access HBase tables from Hive? 13](#_Toc495108769)

[Hortonworks ACID Implementation 13](#_Toc495108770)

[INTRODUCTION 13](#_Toc495108771)

[PREREQUISITES 13](#_Toc495108772)

[CONCEPTS 13](#_Toc495108773)

[1. ENABLING ACID TRANSACTIONS USING AMBARI 13](#_Toc495108774)

[2. HELLO ACID: CREATE A PARTITIONED ACID TABLE AND INSERT SOME DATA 13](#_Toc495108775)

[3. STREAMING DATA INGESTION 13](#_Toc495108776)

[4. PRACTICAL USES OF SQL ACID 13](#_Toc495108777)

[4.1 SEARCHED UPDATES 13](#_Toc495108778)

[4.2 SEARCHED DELETES 13](#_Toc495108779)

[5. OVERWRITING EXISTING RECORDS WITH NEW RECORDS 13](#_Toc495108780)

[6. OPERATIONAL TOOLS FOR ACID 13](#_Toc495108781)

[6.1 SEEING TRANSACTIONS: 13](#_Toc495108782)

[6.2 SEEING LOCKS: 13](#_Toc495108783)

[6.3 ABORTING TRANSACTIONS: 13](#_Toc495108784)

[7. REAL-WORLD PERFORMANCE CONSIDERATIONS 13](#_Toc495108785)

[8. FOR EXPERTS ONLY 13](#_Toc495108786)

# **Introduction to Hive**

# **How can we achieve ACID in HIVE?**

In this blog post, we have explained about the row-level transactions available in Hive. This post will provide you a good idea of how to implement the row-level transactions on the Hive table.

Before beginning with the transactions in Hive, let’s look at the ACID properties, which are vital for any transaction.

set hive.support.concurrency = true;

set hive.enforce.bucketing = true;

set hive.exec.dynamic.partition.mode = nonstrict;

set hive.txn.manager = org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;

set hive.compactor.initiator.on = true;

set hive.compactor.worker.threads = a positive number on at least one instance of the Thrift metastore service;

## **What is ACID?**

ACID stands for Atomicity, Consistency, Isolation, and Durability.

Atomicity means, a transaction should complete successfully or else it should fail completely i.e. it should not be left partially. Consistency ensures that any transaction will bring the database from one valid state to another state. Isolation states that every transaction should be independent of each other i.e. one transaction should not affect another. And Durability states that if a transaction is completed, it should be preserved in the database even if the machine state is lost or a system failure might occur.

These ACID properties are essential for a transaction and every transaction should ensure that these properties are met.

## **Transactions in Hive**

Transactions in Hive are introduced in Hive 0.13, but they only partially fulfill the ACID properties like atomicity, consistency, durability, at the partition level. Here, Isolation can be provided by turning on one of the locking mechanisms available with zookeeper or in memory.

But in Hive 0.14, new API’s have been added to completely fulfill the ACID properties while performing any transaction.

Transactions are provided at the row-level in Hive 0.14. The different row-level transactions available in Hive 0.14 are as follows:

1. Insert
2. Delete
3. Update

There are numerous limitations with the present transactions available in Hive 0.14. ORC is the file format supported by Hive transaction. It is now essential to have ORC file format for performing transactions in Hive. The table needs to be bucketed in order to support transactions.

## **Row-level Transactions Available in Hive 0.14**

Let’s perform some row-level transactions available in Hive 0.14. Before creating a Hive table that supports transactions, the transaction features present in Hive needs to be turned on, as by default they are turned off.

The below properties needs to be set appropriately in ***hive shell***, order-wise to work with transactions in Hive:

|  |  |  |
| --- | --- | --- |
|  |  | ***hive>set hive.support.concurrency = true;***  ***hive>set hive.enforce.bucketing = true;***  ***hive>set hive.exec.dynamic.partition.mode = nonstrict;***  ***hive>set hive.txn.manager = org.apache.hadoop.hive.ql.lockmgr.DbTxnManager;***  ***hive>set hive.compactor.initiator.on = true;***  ***hive>set hive.compactor.worker.threads = a positive number on at least one instance of the Thrift metastore service;*** |

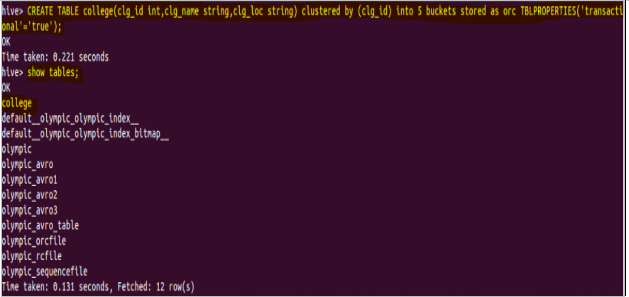
If the above properties are not set properly, the ‘Insert’ operation will work but ‘Update’ and ‘Delete’ will not work and you will receive the following error:

|  |  |  |
| --- | --- | --- |
|  |  | ***FAILED: SemanticException [Error 10294]: Attempt to do update or delete usingtransaction manager thatdoes not support these operations.*** |

## **Creating a Table That Supports Hive Transactions**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | ***CREATE TABLE college(clg\_id int,clg\_name string,clg\_loc string) clustered by (clg\_id) into 5 buckets stored as orc TBLPROPERTIES('transactional'='true');*** |  |

The above syntax will create a table with name ‘*college’*and the columns present in the table are ‘*clg\_id, clg\_name, clg\_loc’. W*e are *bucketing* the table by ‘*clg\_id’*and the table format is ‘*orc’,*also we are enabling the transactions in the table by specifying it inside the *TBLPROPERTIES* as *‘transactional’=’true’*



We have successfully created a table with name ‘*college’*which supports row-level transactions of Hive.

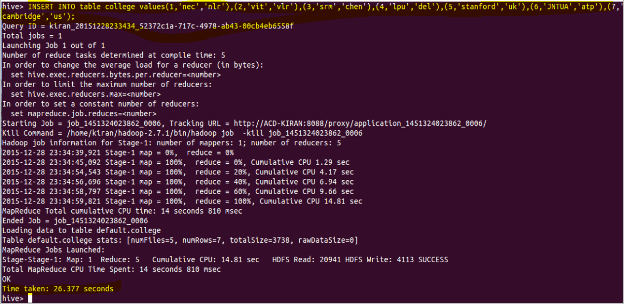
The create table can be checked using the command

***show tables.***

## **Inserting Data into a Hive Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | ***INSERT INTO table college values(1,'nec','nlr'),(2,'vit','vlr'),(3,'srm','chen'),(4,'lpu','del'),(5,'stanford','uk'),(6,'JNTUA','atp'),(7,'cambridge','us');*** |  |

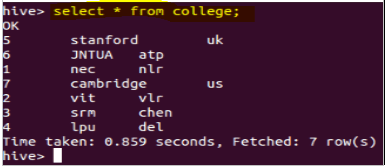
The above command is used to insert row wise data into the Hive table. Here, each row is seperated by ‘*( )’ brackets.*



*Now, we have successfully inserted the data into the Hive table.*

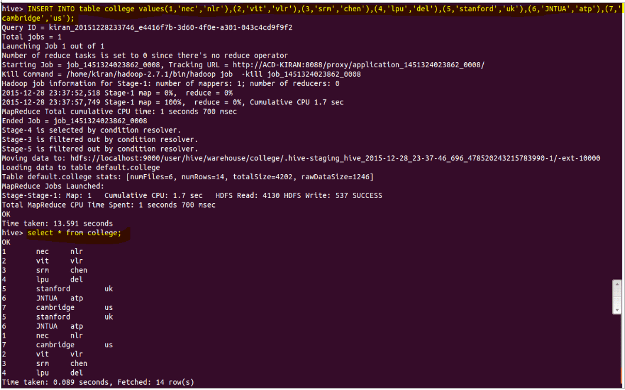
*The contents of the table can be viewed using the command*

***select \* from college***



From the above image, we can see that the data has been inserted successfully into the table.

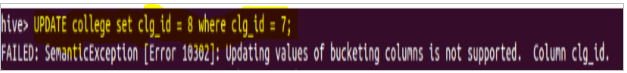
Now if we try to re-insert the same data again, it will be appended to the previous data as shown below:



## **Updating the Data in Hive Table**

|  |  |
| --- | --- |
|  | ***UPDATE college set clg\_id = 8 where clg\_id = 7;*** |

The above command is used to update a row in Hive table.



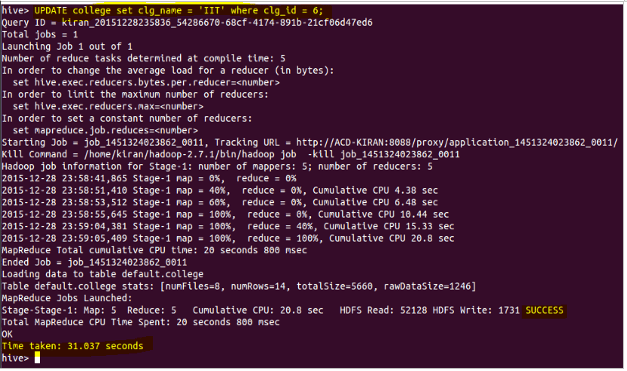
From the above image, we can see that we have received an error message. This means that the Update command is not supported on the columns that are bucketed.

In this table, we have bucketed the ***‘clg\_id’*** column and performing the Update operation on the same column, so we have go the error

***FAILED: SemanticException[Error 10302]: Updating values of bucketing columns is not supported. Column clg\_id***

**Now let’s perform the update operation on Non bucketed column**

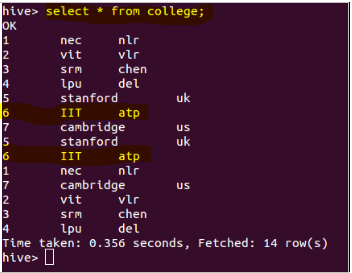
|  |  |
| --- | --- |
|  | ***UPDATE college set clg\_name = 'IIT' where clg\_id = 6;*** |



We have successfully updated the data.

The updated data can be checked using the command

**select \* from college**.



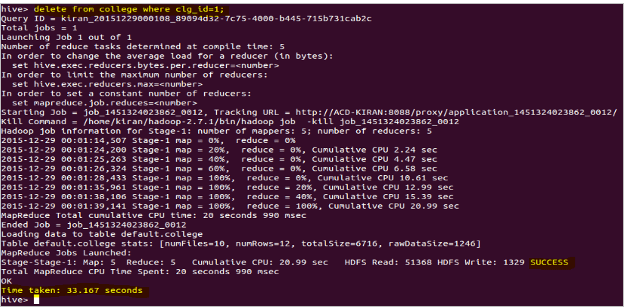
We can see that the data has been updated successfully.

Now let’s perform the Delete operation on the same table.

## **Deleting a Row from Hive Table**

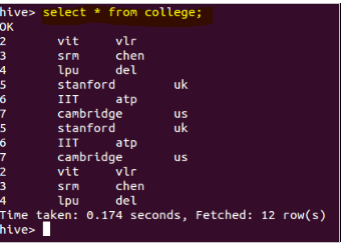
|  |  |
| --- | --- |
|  | ***delete from college where clg\_id=5;*** |

The above command will delete a single row in the Hive table.



We have now successfully deleted a row from the Hive table. This can be checked using the command

**select \* from college.**



We can see that there is no row with***clg\_id =1***. This means that we have successfully deleted the row from the Hive table.

This is how the transactions or row-wise operations are performed in Hive.

# **Top 15 Hive Interview Questions**

## **1.What is Hive?**

Hive is a data warehousing tool. It is an abstraction and it gives SQL queries to perform an analysis. It gives you logical abstraction over the databases and the tables but it is not a database.

## **2.What is Hive a metastore?**

Hive contains two things: data and the metadata. The metadata contains the (column names, partitions information, bucketing information, SerDe etc.) i.e., the data about the actual table this is by default stored in the Derby database, we can also configure it to Oracle or MySQL database.

## **3.What is the limitation of Derby database for Hive metastore?**

With derby database, you cannot have multiple connections or multiple sessions instantiated at the same time. Derby database runs in the local mode and it creates a log file so that multiple users cannot access Hive simultaneously.

## **4.What are managed and external tables?**

We have got two things, one of which is data present in the HDFS and the other is the metadata, present in some database.

There are two categories of Hive tables i.e., Managed and External Tables.

In the Managed tables, both the data and the metadata are managed by Hive and if you drop the managed table, both data and metadata are deleted.

There are some situations where your data will be controlled by some other application and you want to read that data but you must allow Hive to delete that data. In such case, you can create an external table in Hive. In the external table, metadata is controlled by Hive but the actual data will be controlled by some other application. So, when you delete a table accidentally, only the metadata will be lost and the actual data will reside wherever it is.

## **5.What are the complex data types in Hive?**

### **MAP**

The Map contains a key-value pair where you can search for a value using the key.

### **STRUCT**

A Struct is a collection of elements of different data types. For example, if you take the address, it can have different data types. For example, pin code will be in Integer format.

### **ARRAY**

An Array will have a collection of homogeneous elements. For example, if you take your skillset, you can have N number of skills

### **UNIONTYPE**

It represents a column which can have a value that can belong to any of the data types of your choice.

## **6.How does partitioning help in the faster execution of queries?**

With the help of partitioning, a subdirectory will be created with the name of the partitioned column and when you perform a query using the WHERE clause, only the particular sub-directory will be scanned instead of scanning the whole table. This gives you faster execution of queries.

## **7.How to enable dynamic partitioning in Hive?**

Related to partitioning there are two types of partitioning Static and Dynamic.

In the static partitioning, you will specify the partition column while loading the data.

Whereas in dynamic partitioning, you push the data into Hive and then Hive decides which value should go into which partition. To enable dynamic partitioning, you have set the below property

**set hive.exec.dynamic.parition.mode = nonstrict;**

Example

insert overwrite table emp\_details\_partitioned

partition(location)

select \* from emp\_details;

## **8.How does bucketing help in the faster execution of queries?**

If you have to join two large tables, you can go for reduce side join. But if both the tables have the same number of buckets or same multiples of buckets and also sorted on the same column there is a possibility of SMBMJ in which all the joins take place in the map phase itself by matching the corresponding buckets. Buckets are basically files that are created inside the HDFS directory.

There are different properties which you need to set for bucket map joins and they are as follows:

**set hive.enforce.sortmergebucketmapjoin = false;**

**set hive.auto.convert.sortmerge.join =  false;**

**set hive.optimize.bucketmapjoin =  ture;**

**set hive.optimize.bucketmapjoin.sortedmerge = true;**

## **9.How to enable bucketing in Hive?**

By default bucketing is disabled in Hive, you can enforce to enable it by setting the below property

**set hive.enforce.bucketing  = true;**

## **10.Which method has to be overridden when we use custom UDF in Hive?**

Whenever you write a custom UDF in Hive, you have to extend the UDF class and you have to override the **evaluate()** function.

## **11.What are the different file formats in Hive?**

There are different file formats supported by Hive

* Text File format
* Sequence File format
* RC file format
* Parquet
* Avro
* ORC

Every file format has its own characteristics and Hive allows you to choose easily the file format which you wanted to use.

## **12.How is SerDe different from File format in Hive?**

SerDe stands for Serializer and Deserializer. It determines how to encode and decode the field values or the column values from a record that is: how you serialize and deserialize the values of a column

But file format determines how records are stored in key value format or how do you retrieve the records from the table.

## **13.What is RegexSerDe?**

Regex stands for a regular expression. Whenever you want to have a kind of pattern matching, based on the pattern matching, you have to store the fields. RegexSerDe is present in **org.apache.hadoop.hive.contrib.serde2.RegexSerDe.**

In the SerDeproperties, you have to define your input pattern and output fields. For example, you have to get the column values from line **[xyz/pq@def](about:blank)** if you want to take xyz, pq and def separately.

To extract the pattern, you can use:

‘input.regex’ = ‘(.\*)/(.\*)@(.\*)’

To specify how to store them, you can use

‘output.format.string’ = ‘%1$s%2$s%3$s’;

## **14.How is ORC file format optimized for data storage and analysis?**

ORC stores collections of rows in one file and within the collection the row data will be stored in a columnar format. With columnar format, it is very easy to compress, thus reducing a lot of storage cost.

While querying also, it queries the particular column instead of querying the whole row as the records are stored in columnar format.

ORC has got indexing on every block based on the statistics min, max, sum, count on columns so when you query, it will skip the blocks based on the indexing.

## **15.How to access HBase tables from Hive?**

Using Hive-HBase storage handler, you can access the HBase tables from Hive and once you are connected, you can query HBase using the SQL queries from Hive. You can also join multiple tables in HBase from Hive and retrieve the result.

# Hortonworks ACID Implementation

## INTRODUCTION

Hadoop is gradually playing a larger role as a system of record for many workloads. Systems of record need robust and varied options for data updates that may range from single records to complex multi-step transactions.

Some reasons to perform updates may include:

* Data restatements from upstream data providers.
* Data pipeline reprocessing.
* Slowly-changing dimensions (e.g. SCD Type 1)
* Dimension history / evolution (e.g. SCD Type 2)

Standard SQL provides ACID operations through INSERT, UPDATE, DELETE, transactions, and the more recent MERGE operations. These have proven to be robust and flexible enough for most workloads. Hive offers INSERT, UPDATE and DELETE, with more of capabilities on the roadmap.

## PREREQUISITES

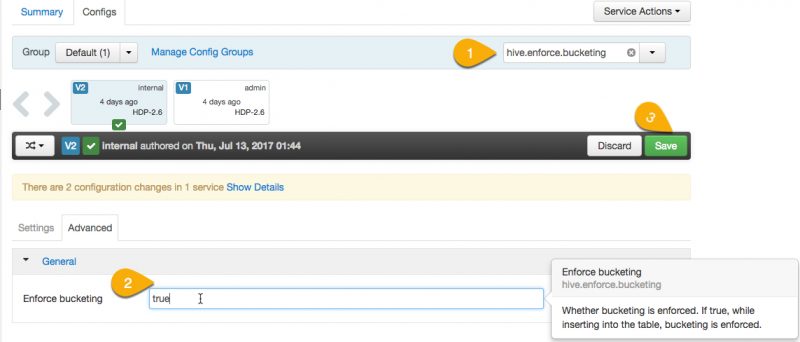
* [Download Hortonworks Sandbox](https://hortonworks.com/downloads/#sandbox)
* Complete the [Learning the Ropes of the Hortonworks Sandbox tutorial,](https://hortonworks.com/tutorial/learning-the-ropes-of-the-hortonworks-sandbox/) you will need it for logging into Ambari.

## CONCEPTS

* **Transactional Tables**: Hive supports single-table transactions. Tables must be marked as transactional in order to support UPDATE and DELETE operations.
* **Partitioned Tables**: Hive supports table partitioning as a means of separating data for faster writes and queries. Partitions are independent of ACID. Large tables in Hive are almost always partitioned. Large ACID tables should be partitioned for optimal performance.
* **ACID Operations (INSERT / UPDATE / DELETE)**: Standard SQL commands that allow data inserts, updates and deletes.
* **Primary Key**: Databases use primary keys to make records easy to locate, which facilitates updates or deletes. Hive does not enforce the notion of primary keys, but if you plan to do large-scale updates and deletes you should establish a primary key convention within your application.
* **Streaming Ingest**: Data can be streamed into transactional Hive tables in real-time using Storm, Flume or a lower-level direct API.
* **Optimistic Concurrency**: ACID updates and deletes to Hive tables are resolved by letting the first committer win. This happens at the partition level, or at the table level for unpartitioned tables.
* **Compactions**: Data must be periodically compacted to save space and optimize data access. It is best to let the system handle these automatically, but these can also be scheduled in an external scheduler.

## 1. ENABLING ACID TRANSACTIONS USING AMBARI

* Log in to Ambari using user credentials **raj\_ops/raj\_ops**. Enabling Hive ACID is simple: **Ambari** -> **Hive** -> **Configs**. Set **ACID Transactions** to **On**.
* Set **hive.enforce.bucketing** to **true** and Save changes.

[](https://2xbbhjxc6wk3v21p62t8n4d4-wpengine.netdna-ssl.com/wp-content/uploads/2017/08/enforce-bucketing.jpg)

* After saving / accepting configuration changes, Restart all affected services.

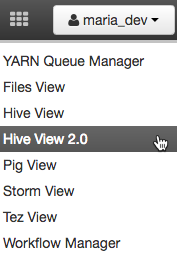
Let’s explore these options:

* **ACID Transactions** – Turn this on to enable the creation of transactional tables globally.
* **Run Compactor** – The “Run Compactor” setting should always be set to true.
* **Number of threads used by Compactor** – This controls the maximum number of background MapReduce jobs that may run at any given time to compact tables. It is best to have this be a ratio of the number of transactional tables that are actively updated. In any event the value should always be greater than 0. Typically 5 to 10 would be appropriate in production settings. 1 thread is only applicable for sandbox environments.

If you’re building a production application you can use Ambari Blueprints to provision ACID-enabled HDP clusters in a consistent and repeatable way.

## 2. HELLO ACID: CREATE A PARTITIONED ACID TABLE AND INSERT SOME DATA

This section is written assuming you will use the Hive View within Ambari. Log out of Ambari from **raj\_ops** user and re login with credentials **maria\_dev/maria\_dev**.  
When you have logged into Ambari, go to the Hive View 2.0 by hovering in the top right and selecting Hive View.

[](https://2xbbhjxc6wk3v21p62t8n4d4-wpengine.netdna-ssl.com/wp-content/uploads/2017/08/select_hive_view.png)

Let’s start by creating a transactional table. Only transactional tables can support updates and deletes. Within the Hive View query editor insert this query text:

DROP TABLE IF EXISTS hello\_acid;

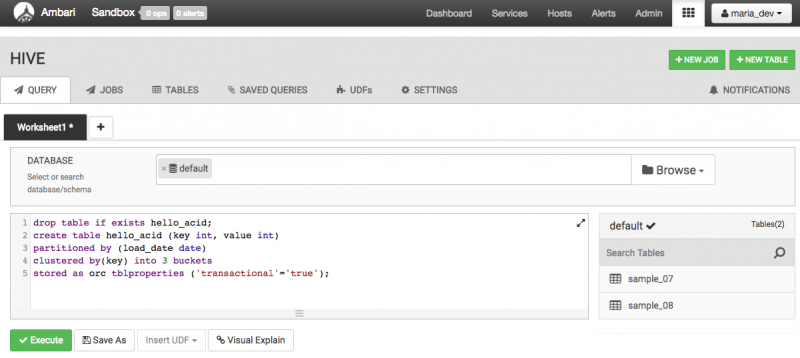
CREATE TABLE hello\_acid (key int, value int)

PARTITIONED BY (load\_date date)

CLUSTERED BY(key) INTO 3 BUCKETS

STORED AS ORC TBLPROPERTIES ('transactional'='true');

Within the editor it will look as below. Press Execute to create the table.

[](https://2xbbhjxc6wk3v21p62t8n4d4-wpengine.netdna-ssl.com/wp-content/uploads/2017/06/create_table_hello_acid.png)

From here on out, everything is familiar SQL you’ve likely used for many years. Here is an example that inserts some records, deletes one record and updates one record.

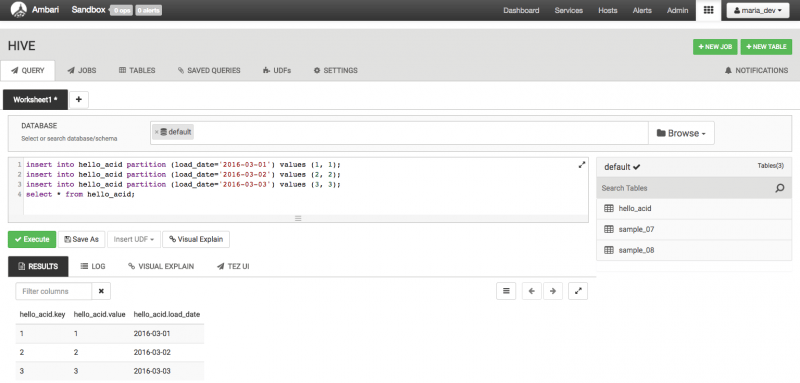
INSERT INTO hello\_acid partition (load\_date='2016-03-01') VALUES (1, 1);

INSERT INTO hello\_acid partition (load\_date='2016-03-02') VALUES (2, 2);

INSERT INTO hello\_acid partition (load\_date='2016-03-03') VALUES (3, 3);

SELECT \* FROM hello\_acid;

You should see the following results:

[](https://2xbbhjxc6wk3v21p62t8n4d4-wpengine.netdna-ssl.com/wp-content/uploads/2017/06/insert_into_hello_acid.png)

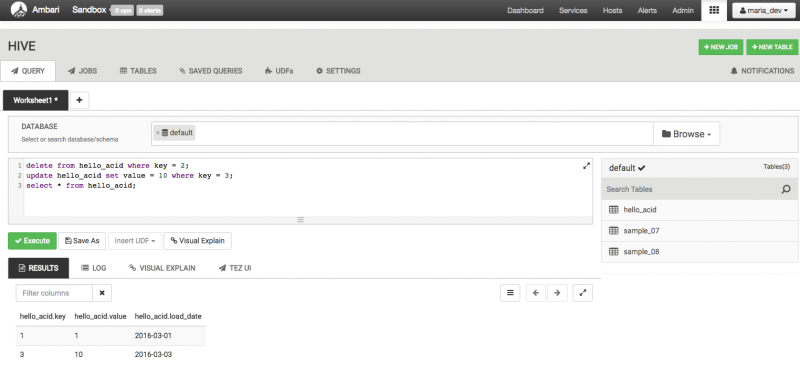
Next, let’s delete and update data in the same window execution:

DELETE FROM hello\_acid WHERE key = 2;

UPDATE hello\_acid SET value = 10 WHERE key = 3;

SELECT \* FROM hello\_acid;

You should see this:

[](https://2xbbhjxc6wk3v21p62t8n4d4-wpengine.netdna-ssl.com/wp-content/uploads/2017/06/update_hello_acid.png)

This example shows the most basic ways to add data into a Hive table using INSERT, UPDATE and DELETE commands. Later we will see some more powerful ways of adding data to an ACID table that involve loading staging tables and using INSERT, UPDATE or DELETE commands, combined with subqueries, to manage data in bulk.

These DML commands are designed to deal with large amounts of data in a microbatch manner. These DML statements should not be used for record-level data management. If you have small batches of constantly arriving, you should use Streaming Data Ingestion instead.

## 3. STREAMING DATA INGESTION

Many organizations have continuous data streams and would like easy ways of running complex SQL analytics on these streams. Hive provides streaming data ingestion that to make this easy by automatically landing streaming data into Hive tables and supporting simultaneous ingest and query.

Broadly there are 2 options for using Hive Streaming Data Ingestion:

1. Use an existing [Storm Hive Bolt](https://github.com/apache/storm/tree/master/external/storm-hive) or [Flume Hive Sink](https://flume.apache.org/FlumeUserGuide.html#hive-sink) integration. This route is relatively simple but also relatively “buttoned up”, focusing on moving data from point A to point B with minimal processing.
2. Use the low-level [Streaming Data Ingest API](https://cwiki.apache.org/confluence/display/Hive/Streaming+Data+Ingest) directly.

Before you use the streaming API you must create a partitioned transactional table and use it as your sink. From a query perspective everything else is identical. The rest of this document will focus on using the SQL DDL/DML approach.

## 4. PRACTICAL USES OF SQL ACID

Inserting a couple of records helps to get acquainted but in a real setting you need to deal with thousands or millions of records at a time. This section discusses how to get deal with data batches across a number of common scenarios.

These patterns require you to establish a primary key. Hive does not enforce primary key uniqueness, you will need to do this in your application. Although Hive 2.1 introduced the notion of non-validating foreign key relationships. There are currently no integrity checks enforced by the system.

### 4.1 SEARCHED UPDATES

Hive ACID supports searched updates, which are the most typical form of updates. It is important to realize that, based on Hive ACID’s architecture, updates must be done in bulk. Doing row-at-a-time updates will not work at any practical scale. Searched updates can work well when you’re doing a first pass to update a large number of records in a fixed way.

For example let’s consider a dimension table which includes a flag to indicate whether the record is the most current value. This allows tracking a dimension’s evolution over time, a common strategy for dealing with slowly-changing dimensions (SCDs). When a set of dimension table updates arrive you want to mark all existing values as old. This is easy to do with an in list and subquery on a common key.

DROP TABLE IF EXISTS mydim;

CREATE TABLE mydim (key int, name string, zip string, is\_current boolean)

CLUSTERED BY (key) INTO 3 BUCKETS

STORED AS ORC TBLPROPERTIES ('transactional'='true');

INSERT INTO mydim VALUES

(1, 'bob', '95136', true),

(2, 'joe', '70068', true),

(3, 'steve', '22150', true);

DROP TABLE IF EXISTS updates\_staging\_table;

CREATE TABLE updates\_staging\_table (key int, newzip string);

INSERT INTO updates\_staging\_table VALUES (1, 87102), (3, 45220);

— **Before**

SELECT \* FROM mydim;

UPDATE mydim SET is\_current = false

WHERE mydim.key IN (SELECT key FROM updates\_staging\_table);

— **After**

SELECT \* FROM mydim;

The result will be that records 1 and 3 have their is\_current flag set to false. From there we can add the new, updated values to the end of the table with their is\_current flag set to true.

### 4.2 SEARCHED DELETES

Bulk deletes can be done easily with a staging table and in list + subquery. This requires you have a common key between the tables, similar to how you would use a primary key in RDBMS.

Example:

DELETE FROM mydim

WHERE mydim.key IN (SELECT key FROM updates\_staging\_table);

SELECT \* FROM mydim;

## 5. OVERWRITING EXISTING RECORDS WITH NEW RECORDS

There are situations where you need to update a batch of records to a new set of values. For example, Type 1 SCD updates or restatements of inaccurate data. Hive now supports [SQL MERGE](https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DML#LanguageManualDML-Merge), which will make this task easy.

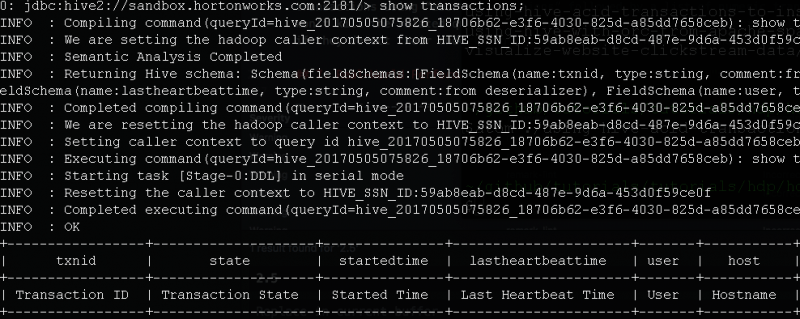
## 6. OPERATIONAL TOOLS FOR ACID

ACID transactions create a number of locks during the course of their operation. Transactions and their locks can be viewed using a number of tools within Hive.

### 6.1 SEEING TRANSACTIONS:

show transactions;

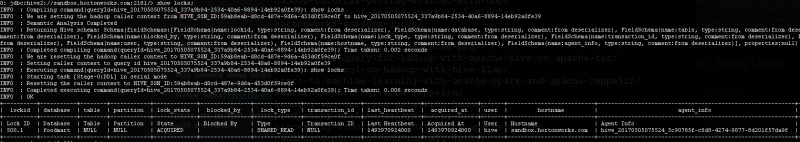
This command shows active and aborted transactions. Here is some example output:

[](https://2xbbhjxc6wk3v21p62t8n4d4-wpengine.netdna-ssl.com/wp-content/uploads/2017/06/show_transactions.png)

### 6.2 SEEING LOCKS:

show locks;

This command shows locks, along with their associated transaction IDs. Example:

[](https://2xbbhjxc6wk3v21p62t8n4d4-wpengine.netdna-ssl.com/wp-content/uploads/2017/06/show_locks.png)

Locks can be Read, Update or X locks. Update lock is compatible with Read locks but not other updates. X is not compatible with anything.

### 6.3 ABORTING TRANSACTIONS:

ABORT TRANSACTIONS transactionID;

It may be necessary to abort a transaction, for example because a transaction is running too long. You can abort a set of transactions using “abort transactions” followed by a list of numeric transaction IDs. Note that aborting a transaction won’t kill the related query immediately. Instead ACID queries periodically heartbeat, every 2.5 minutes by default, and if they detect their underlying transaction has been aborted they will exit at that time.

## 7. REAL-WORLD PERFORMANCE CONSIDERATIONS

These performance tips will help you survive in the real world:

1. Partition your data. Hive compactions are not tiered: major compactions re-write all data in modified partitions, one partition at a time. Partitioning data is essential to ensure you can manage large datasets without degradation. Partitioning by date is the most common approach.

2. Inserts are fast, updates and deletes are relatively slower. Updates and deletes perform full partition scans. Plan for this by batching data appropriately.

3. If your workload includes a large number of updates and deletes, compact regularly. Your total data size will grow until you compact, and analytical queries will slowly degrade until compaction is done.

## 8. FOR EXPERTS ONLY

In the real world things go wrong. When things go wrong you need options for creative solutions. At Hortonworks we have used the information in this section to get past some very tricky problems. Warning: Improper application of this information may cause data corruption or permanent data loss. Ensure you fully understand the system before using this information, test it out on data you can afford to lose and always backup any data you really care about.

ACID tables have a hidden column called row\_\_id. You should consider this column a system internal and assume that its name or even its presence may change at any time without warning. You should not build a long-term solution on top of this column, only use it to get you past a tough spot.

With that out of the way, this column records:

1. The transactionid that was active when the data was inserted or updated.
2. The bucketid, the bucket number where the data lives.
3. The rowid, the rowid within this transaction/bucket combo.

You can access this data as follows using beeline:

beeline -u 'jdbc:hive2://sandbox.hortonworks.com:2181/;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2'

0: jdbc:hive2://sandbox.hortonworks.com:2181/> select row\_\_id from hello\_acid;

Ouptut should look like:

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

| row\_\_id |

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

| {"transactionid":1,"bucketid":1,"rowid":0} |

| {"transactionid":3,"bucketid":0,"rowid":0} |

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

A common need is to confirm that all records were ingested. Let’s say your upstream provider insists data is missing in Hive. Your provider (e.g. Storm Bolt) can tell you the transaction ID used to insert data. You can count the actual records using the transactionid. Replace X with your transactionid:

set hive.optimize.ppd=false;

select count(\*) from hello\_acid where row\_\_id.transactionid = X;

Keep in mind that data from this transaction may have been deleted by a subsequent UPDATE or DELETE statement, so if the counts don’t match, consider if records may be altered some other way.

Use information related to this hidden field very carefully.You can exit beeline by issuing the command:

!quit