# Hive 0.13 and Hive 2 Code and Syntax Compatibility

Created by user-93867, last modified by user-dbe23 on May 17, 2021

Most queries from Hive 0.13 should work with minimal or no modification in Hive 2. However, Hive 2 is stricter than Hive 0.13.

## Code Compatibility

| **Element** | **Hive 0.13** | **Hive 2** |
|---|---|---|
| SQL syntax | HiveQL<br>(A MySQL-like dialect) | ANSI SQL syntax |
| Reserved words | | More reserved words than HiveQL. Quote reserve words with backticks. |
| Numeric data types | | Some values must be CAST explicitly. |
| Treasure Data specific syntax | | Obsolete. Remove syntax from queries. |

## Syntax Updates

The following Hive 0.13 constructs must be modified to be ready for Hive 2.

## Stricter Syntax

### Stricter Syntax: UNION and Matching or Unmatching Types

### Stricter Syntax: Additional Reserved Words and Function Names Require Escapes

Reserved words and UDF names used as column names must be escaped with backticks.

The most common case of this is `date`.

**Example**

```
date -> `date`
```

Reserved words manifest as parse errors in Hive 2. Parse error issues indicate that you need to find the keyword in the script and put backticks around it.

**Example**

Parse errors received from Hive 2.

```
Cannot recognize input near 'date' '>=' '1548403200' in expression specification
Cannot recognize input near 'timestamp' '=' '1548835200' in expression specification
Cannot recognize input near 'AS' 'date' ',' in selection target
```

To fix this, given the current query syntax of the following query.

```
SELECT queue_id
, cs_id
, SUM(refunds) AS refunds_qtd
, SUM(x_refunds) AS x_refunds_qtd

FROM daily_cs_stats
WHERE date >= 1546329600ANDdate < 1547884800
```

Rewrite the query. Notice that the date is surrounded by left-single quotation marks.

```
SELECT queue_id
, cs_id
, SUM(refunds) AS refunds_qtd
, SUM(x_refunds) AS x_refunds_qtd

FROM daily_cs_stats
WHERE `date` >= 1546329600AND`date` < 1547884800
```

For more information, review the TD Hive UDF documentation. Additionally, you might want to review the additional UDFs in HiveMall UDF documentation.

## Stricter Syntax: In SELECT, Use “AS” with Aliases

English
In testing some SELECT statements you might encounter a semantic error such as:
Translated by machine

FAILED: SemanticException Line 0:-1 Invalid table alias or column reference 'sq_1': (possible column names are...

This can happen when Hive 2 is unable to unambiguously determine whether an identifier is a column alias or the name of a real column. For example, the following query produced an error.

```
SELECTtime,
replace(replace(k_uuid,'%2B','+'),'/','/') k_uuid, category_id,ao_category_id FROM X
WHEREEXISTS(SELECT * FROM A WHERE A.mtime = X.time AND A.x_uuid = X.x_uuid)
GROUPBY1,2,3,4
```

The following error was thrown:

FAILED: SemanticException Line 0:-1 Invalid table alias or column reference 'sq_1': (possible column names are: _table_or_col a) x_uuid) sq_corr_1)), (. (tok_table_or_col sq_1) sq_corr_1))

The problem is that *x_uuid* is the name of a source column in the source table X, as well as the desired column alias in the SELECT.

The fix is to use an AS to make it unambiguous that the identifier is an alias.

```
SELECTtime,
 replace(replace(x_uuid,'%2B','+'),'/','/') AS x_uuid, category_id,ao_category_id FRO
WHEREEXISTS(SELECT * FROM A WHERE A.mtime = X.time AND A.x_uuid = X.x_uuid)
GROUPBY1,2,3,4
```

Alternatively, you can avoid using the name of an existing column as an alias.

```
SELECTtime,
replace(replace(x_uuid,'%2B','+'),'/','/') x_uuid_out, category_id,ao_category_id FRO
WHEREEXISTS(SELECT * FROM A WHERE A.mtime = X.time AND A.x_uuid = X.x_uuid)
GROUPBY1,2,3,4
```

## Stricter Syntax: Group by Columns and Partition by Columns are Different

A query that uses a Group by and Partition in a single query does not work on Hive 2.0.

To run this query on Hive 2.0, you would use the following code where all deleted columns are in Group by.

English

Translated by machine

**Example**

```
select sex,name
-- ,count(DISTINCT id)
,count(DISTINCT id) over (PARTITION BY sex,name)
from ani
GROUP BY id,sex,name
```

## Stricter Types

### Stricter Types: CAST Non-Integer Numeric Literals to DOUBLE Before Storage in TD

Numeric literals that are not integers, default to DECIMAL in Hive 2. However, TD storage only supports DOUBLE. Therefore you must CAST any numeric value to DOUBLE before storing it in a table.

```
51.5211 -> CAST(51.5211 AS DOUBLE)
```

Without the casting, you may see an error similar to this:

```
FAILED: SemanticException java.lang.IllegalArgumentException: Failed to cast hive typ
```

Here is an example that works with Hive 0.13, but fails with Hive 2 if not type cast.

```
WITH sourceCTE as (
select 51.5211 as col1decimal
)
insert into table new_table_with_no_columns
select col1decimal from sourceCTE
```

Assume that new_table_with_no_columns does not exist when this statement is run. TD assigns new_table_with_no_columns a schema based on the schema of sourceCTE. The constant 51.5211 has type DECIMAL(6,4), so sourceCTE.col1decimal is of type DECIMAL(6,4).

English

Translated by machine

The INSERT tries to insert the DECIMAL(6,4) into the output table. In Hive 0.13, the value is automatically CAST as needed to DOUBLE and written. In Hive 2, this CAST is not automatic, so you get the error.

This requirement only applies to literal numeric values; if you attempt to store an INTEGER column, return value from a function, etc. into a TD table, TD Hive casts the value appropriately.

## Stricter Types: All Numeric Literals in an Array Must be CAST to the Same Data Type

All numeric literals in an array must be CAST to a consistent data type.

Here is an example that works with Hive 0.13 and gives an error with Hive 2.

```
SELECT

CAST(tuple[0] AS BIGINT) as offset,

tuple[1] as increase_weight,

tuple[2] as decrease_weight

FROM

(

SELECT

Array(

Array(0, 0.0, 1.0),

Array(1, 0.019675926, 0.9803323),

Array(2, 0.074074074, 0.925925926),

Array(3, 0.15625, 0.84375),

Array(4, 0.259259259, 0.740740741),

Array(5, 0.376157407, 0.623842593),

Array(6, 0.5, 0.5)
```

English

Translated by machine

```
) as tuples

) t

LATERAL VIEW explode(tuples) exploded AS tuple
```

Can produce an error similar to this:

```
FAILED: SemanticException [Error 10016]: Line 10:0 Argument type mismatch '0.9022: Ar
```

To fix the error, CAST all values to decimal(19,9) (19 digits with 9 digits after the decimal).

```
SELECT

CAST(tuple[0] AS BIGINT) as offset,

tuple[1] as increase_weight,

tuple[2] as decrease_weight

FROM

(

SELECT

Array(

Array(0, cast(0.0 as decimal(19,9)), cast(1.0 as decimal(19,9))),

Array(1, cast(0.019675926 as decimal(19,9)),

cast(0.9803323 as decimal(19,9))),

Array(2, cast(0.074074074 as decimal(19,9)),

cast(0.925925926 as decimal(19,9))),

Array(3, cast(0.15625 as decimal(19,9)),
```

English

Translated by machine

```
cast(0.84375 as decimal(19,9))),

Array(4, cast(0.259259259 as decimal(19,9)),

cast(0.740740741 as decimal(19,9))),

Array(5, cast(0.376157407 as decimal(19,9)),

cast(0.623842593 as decimal(19,9))),

Array(6, cast(0.5 as decimal(19,9)),

cast(0.5 as decimal(19,9))) )

as tuples

) t

LATERAL VIEW explode(tuples) exploded AS tuple
```

## Stricter Types: Mismatched data types on either side of a UNION

Hive 0.13 allows mismatched data types ( in some cases) for a given column name on either side of a UNION.

Hive 2020.1 produces the following error due to stricter data type checking:

```
FAILED: SemanticException <line no.>:<character number> Schema of both sides of union
```

There are two solutions that enable you to run a UNION in both Hive 0.13 and Hive 2020.1.

**Example**

If you have two tables with the following data types and are trying to run a UNION that used to work in Hive 0.13 but doesn't in Hive 2020.1.

```
CREATE
table mytest.normaltypes (timetwo bigint, first varchar, prod_id bigint, counter bigi
```

English

Translated by machine

```
CREATE
table mytest.Allstrings (timetwo varchar, first varchar, prod_id varchar, counter var
```

A UNION will work in Hive 0.13 but fail in Hive2020.1.

```
select * from mytest.normaltypes
UNION
Select * from mytest.allstrings
```

## Solution 1

In this solution, we do not change the underlying types of tables. but Instead, CAST columns in the SQL. The following code describes two options.

### Solution 1: Option A

```
SELECT
CAST (timetwo as varchar), first, last, cast(prod_id as varchar),
cast(counter as varchar) from normaltypes
UNION
SELECT
timetwo, first, last, prod_id, counter from allstrings
```

### Solution 1: Option B

```
SELECT
timetwo, first, last, prod_id, counter from normaltypes,
UNION
SELECT
CAST (timetwo as integer), first, last, cast(prod_id as bigint), cast(counter as bigi
```

## Solution 2

In this solution, change the underlying types of one of the tables to match.

### Solution 2: Option A

```
CREATE
table mytest.normaltypes2Allstrings (timetwo varchar, first varchar, prod_id varchar,

SELECT * from Allstrings
UNION
select * from normaltypes2allstrings
```

English

Translated by machine

### Solution 2: Option B

```
create table mytest.Allstrings2normaltypes (timetwo bigint, first varchar, prod_id bi

SELECT * from normaltypes
UNION
select * from Allstrings2normaltypes
```

## Miscellaneous

### CROSS JOIN requires Query Hint

As a precaution, TD Hive 2 restricts the use of CROSS JOINs because they are often coding errors and are expensive to execute. If you intend to perform a CROSS JOIN, apply the following query hint to bypass this protection:

```
- TD enable_cartesian_product: true
```

This is similar to setting hive.mapred.mode to nonstrict in a generic Hive environment.

Explicit CROSS JOIN example:

```
SELECT * FROM employee CROSSJOIN department
```

Implicit CROSS JOIN example (a JOIN without a WHERE clause):

```
SELECT * FROM employee, department
```

If you run a query with an explicit or implicit CROSS JOIN without adding the query hint, you'll see an error like this:

**Information**

FAILED: SemanticException Cartesian products are disabled for safety reasons. If you know what you are doing, please set hive.strict.checks.cartesian.product to false and that hive.mapred.mode is not set to 'strict' to proceed. Note that if you may get errors or incorrect results if you make a mistake while using some of the unsafe features.

## TD Hive 0.13 Obsolete Feature: v-column Syntax

v-column syntax is an obsolete TD-specific method of referring to columns in TD tables. It has been deprecated for years, but still occasionally appears in some old queries. It is not supported in Hive 2.

v-column syntax is still useful in certain TD debugging scenarios, examining ingested data before schema is assigned to it. In such scenarios, we recommend the use of Hive 0.13 during debugging.

The following changes are required for any query using v column[] syntax:

- v column values are always returned as STRING type
- Columns selected through v column where names start with underscore need to be escaped with back quotes (`) when rewritten without v column

### v column values are always returned as STRING type

In Hive 2, rather than use the v column syntax, reference the column, but explicitly CAST it to STRING, or whatever other type is needed. For example:

```
v['country_code'] --> cast(country_code as STRING)

cast(v['age'] as int) + 1 --> cast(age as int) + 1

- if the table already has an explicit schema with column 'age' as numeric, you may n
```

If you use v column in Hive 2, you'll see the following error message:

```
Invalid table alias or column reference 'v'
```

English

### Columns selected through v column where names start with underscore need to be escaped with back quotes (`) when rewritten without v column

For example:

```
v['_country_code'] -> CAST(`_country_code` AS STRING)

- note the use of CAST
```

Make sure that the queries don't use v[] syntax because the Hive 2 compiler will give an error when using v[] syntax. For example:

'Invalid table alias or column reference 'v'

If the original query looks like this:

V column version:

```
SELECT v['user_id'], v['a_type'], v['time_to_view'], v['action'],
time
FROM m_announcements
WHEREtime > 1546156800
```

Rewrite this as:

```
SELECTcast(`user_id`asSTRING), cast(`a_type`asBIGINT),

cast(`time_to_view`asDOUBLE),cast(`action`asBIGINT),
time
FROM m_announcements
WHEREtime > 1546156800
```

Example: If a current query is:

```
SELECT email, fx_id, pl_type FROM (
SELECT v['user_id'] as uid FROM apps WHERE v['action'] = 'x_ping'AND v['user_id'] ISN
JOIN (
SELECTv['_id'] as uid, email, fx_id, pl_type FROM user_tbl WHEREtime > 1548748800AND
ON r.uid = u.uid
```

English

Translated by machine

Rewrite this as:

```
SELECT email, fx_id, pl_type FROM (
SELECTcast('user_id'as STING) as uid FROM apps WHEREcast('action'asSTRING) = 'x_ping'
JOIN (
SELECTcast(`_id`asSTRING) as uid, email, fx_id, pl_type FROM user_tbl WHEREtime > 154
ON r.uid = u.uid
```

## Semantic Error: GROUP BY and PARTITION BY Columns are Different

The syntax correctly fails in Hive 2 when GROUP BY and PARTITION BY columns are different. The same query runs successfully in Hive 0.13, but the query results are incorrect.

Example of a successful query because there is no mismatch in aggregation and selected columns.

```
select sex,name
,count(DISTINCT id)
from ani
GROUP BY sex,name
```

Example of a correct fail in Hive 2 because id column must be included in GROUP BY if using a window function such as PARTITION BY.

```
select sex,name
,count(DISTINCT id) over (PARTITION BY sex,name)
from ani
GROUP BY sex,name
```

To return a successful result, include all selected columns in GROUP BY and rewrite as:

```
select sex,name
,count(DISTINCT id) over (PARTITION BY sex,name)
from ani
GROUP BY id,sex,name
```

No labels

English

Translated by machine