


[Search](#)

- My Coursera
- Vietnam National University - Hanoi
- Coursera for Students
- [Browse](#)
- [Top Courses](#)
- [Profile](#)
- [My Purchases](#)
- [Settings](#)
- [Updates](#)
- [Accomplishments](#)
- [Help Center](#)
- [Log Out](#)
-  Trung Nghia Hoang ▾

Unmanaged (external) table section below). It's up to you whether you would rather make the changes to a table, or drop it and recreate the table.

## Renaming a Table

To rename a table, use

```
ALTER TABLE old_tablename RENAME TO new_tablename;
```

For example, this renames the table **customers** to **clients**:

```
ALTER TABLE customers RENAME TO clients;
```

When you rename a table, Hive or Impala changes the table's name in the metastore, and if the table is internally managed, it also renames the table's directory in HDFS.

## Moving a Table to a Different Database

To move a table to a different database, you also use **RENAME TO**, and you specify the fully qualified names of the old (existing) and new tables, including the database names:

```
ALTER TABLE old_database.tablename RENAME TO new_database.tablename;
```

For example, this moves the existing table named **clients** from the **default** database to the **dig** database:

```
ALTER TABLE default.clients RENAME TO dig.clients;
```

When you move a table to a different database, Hive or Impala changes the associated metadata in the metastore, and if the table is managed, it also moves the table's directory in HDFS into the subdirectory for the different database.

## Changing Column Name or Data Type

To change the name or data type of a column, use

```
ALTER TABLE tablename CHANGE old_colname new_colname type;
```

If you are not changing the data type, you still need to supply the type. If you are not changing the column name—only the data type—then repeat the column name.

For example, the following changes the **first\_name** column (of type **STRING**) in the **employees** table to **given\_name** (but keeps it a **STRING** column).

```
ALTER TABLE employees CHANGE first_name given_name STRING;
```

The following example changes **salary** from **INT** to **BIGINT** without changing the column name:

```
ALTER TABLE employees CHANGE salary salary BIGINT;
```

## Changing Column Order (*Hive only*)

When you create a table for existing data, your columns need to be provided in the same order that they appear in the data files. If you have made a mistake and put them in a different order in the **CREATE TABLE** statement, you can fix it with the **ALTER TABLE** statement.

To change where a column goes using Hive, use the **CHANGE** keyword just as if you were changing the column name and add either **AFTER column** or **FIRST** at the end.

For example, the **employees** table in the VM has columns in this order: **empl\_id, first\_name, last\_name, salary, office\_id**. Suppose you discover that the file data actually lists the office ID before the employee's salary. The following moves the **salary** after the **office\_id** column:

```
ALTER TABLE employees CHANGE salary salary INT AFTER office_id;
```

If the column to move needs to be the first (leftmost) column, then the statement would be

```
ALTER TABLE tablename CHANGE col_name col_name col_type FIRST;
```

## Notes

This feature is available in Hive but not Impala. For Impala, see “Replacing All Columns” for an alternative method.

You always need to give the “old” and “new” names of the column you're moving, along with its data type, *even if those details are not changing*.

This does not change the data files. If you change the order of columns to something *different* from the order in the files, you'll need to recreate the data files using the new order.

## Adding or Removing Columns

You can add one or more columns to the end of the column list using **ADD COLUMNS**, or (with Impala only) you can delete columns using **DROP COLUMN**. The general syntax is

```
ALTER TABLE tablename ADD COLUMNS (col1 TYPE1,col2 TYPE2,... );
```

```
ALTER TABLE tablename DROP COLUMN colname;
```

For example, you can add a **bonus** integer column to the **employees** table:

```
ALTER TABLE employees ADD COLUMNS (bonus INT);
```

Or you can drop the **office\_id** column from the **employees** table:

```
ALTER TABLE employees DROP COLUMN office_id;
```

## Notes

**DROP COLUMN** is not available in Hive, only in Impala. However, see “Replacing All Columns” below.

You can only drop one column at a time. To drop multiple columns, use multiple statements or use the method to replace columns (see below).

You cannot add a column in the middle of the list rather than at the end. You can, however, add the column then change the order (see above) or use the method to replace columns (see below).

As with changing the column order, these do not change the data files.

- If the table definition agrees with the data files before you drop any column other than the last one, you will need to recreate the data files without the dropped column's values.
- If you drop the last column, the data will still exist but it will be ignored when a query is issued.
- If you add columns for which no data exists, those columns will be **NULL** in each row.

## Replacing All Columns

You can also completely replace all the columns with a new column list. This is helpful for dropping multiple columns, or if you need to add columns in the middle of the list. The general syntax is

```
ALTER TABLE tablename REPLACE COLUMNS (col1 TYPE1,col2 TYPE2,... );
```

This completely removes the existing list of columns and replaces it with the new list. Only the columns you specify in the **ALTER TABLE** statement will exist, and they will be in the order you provide.

## Note

Again, this does not change the data files, only the metadata for the table, so you'll either want the new list to match the data files or need to recreate the data files to match the new list.

## Changing to an Unmanaged (External) Table

If you have created a table as a managed table (without the **EXTERNAL** keyword) and later realize you want it unmanaged, you can use **ALTER TABLE** with **TBLPROPERTIES** to make it unmanaged. This is particularly helpful if you want to drop a table without losing the data.

The general syntax is

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

## Notes

Both **EXTERNAL** and **TRUE** are in quotes, and they must be uppercase, here.

You can also use **SET TBLPROPERTIES** with other properties that were not set at creation.

## Try It!

Test these out with the **investors** table.

The first thing you'll do is verify that the table is unmanaged (external) so you can drop the table without losing the data, in case you make a mistake. You can then recreate the table using the exercise from “The ROW FORMAT Clause” reading in “The CREATE TABLE Statement” lesson, where you first created the **investors** table. (You can leave this alteration, no need to change it back.) If it's *not* unmanaged, then you should change it.

1. In Hive or Impala, execute the following on the **customers** table so you can see what a *managed* table looks like. Be sure the **default** database is your active database.

```
DESCRIBE FORMATTED customers;
```

Look down the results for **Table Type**; the value should be **MANAGED\_TABLE**. This means it was created without the **EXTERNAL** keyword.

2. Now run the same command on the **investors** table, and note what the value is for **Table Type**. If it's also **MANAGED\_TABLE**, execute the following to change it to an unmanaged table, then run the **DESCRIBE FORMATTED** statement again and check the value for **Table Type**. (If the value is not **MANAGED\_TABLE**, you don't need to run this command—though it does no harm if you do.)

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

Next, try changing the name.

3. Execute the following statement:

```
ALTER TABLE investors RENAME TO companies;
```

4. Verify that the table name changed in the data source panel (refresh the display if necessary), or by running **SELECT \* FROM companies**;

5. Change the name back to **investors** and verify the change.

Now move it to the **dig** database.

6. Execute the following statement:

```
ALTER TABLE default.investors RENAME TO dig.investors;
```

7. Verify that the table is no longer in the **default** database, and that it *is* in the **dig** database.

8. Check the Hive warehouse directory. The **investors** subdirectory has not moved to the **dig.db** directory—it is still in the **default** database. Do you know why? (See the “Postscript” section below for the answer!)

9. Change the directory back to **default** and verify the change.

Change the column **amount** to **holdings**.

10. Execute the following statement:

```
ALTER TABLE investors CHANGE amount holdings INT;
```

11. Verify that the column's name has changed, using the data source panel or by running **DESCRIBE investors**;

12. Change the column name back to **amount** and verify the change.

Change the column **amount** from **INT** to **BIGINT**.

13. Execute the following statement:

```
ALTER TABLE investors CHANGE amount amount BIGINT;
```

14. Verify that the column's type has changed, using the data source panel or by running **DESCRIBE investors**;

15. Change the column type back to **INT** and verify the change.

Use Hive (*not Impala*) to put the **amount** column at the end instead of the middle, and see the effect.

16. First take a look at the data by running **SELECT \* FROM investors**;

17. Execute the following statement:

```
ALTER TABLE investors CHANGE name name STRING AFTER amount;
```

18. Verify that the columns have been reordered in the metadata *but not in the data itself* by running

```
SELECT * FROM investors;
```

Notice that **amount** is **NULL** in each row—this is because the data being loaded is the non-numeric character data that used to go into **name**. Since **amount** is an **INT** column, the data isn't valid. The integer data meant for **amount** is valid for the **STRING** column **name**, so those values are not **NULL**. (The point here is that the data itself did not change.)

19. Change the column order back by executing

```
ALTER TABLE investors CHANGE name name STRING FIRST;
```

20. Verify that the columns are back to normal using the **SELECT \*** query.

Use Impala to drop a column and then add it back. (You can add with Hive or Impala, but Hive doesn't recognize the **DROP COLUMN** keyword, so to make things easier, use Impala for both.)

21. First drop the **share** column:

```
ALTER TABLE investors DROP COLUMN share;
```

22. Run the **SELECT \*** query to verify the column has been dropped. The values in the other columns has not changed.

23. Add the **share** column back:

```
ALTER TABLE investors ADD COLUMNS (share DECIMAL(4,3));
```

24. Verify that the column has been restored.

Finally, change the columns completely!

25. Execute the following statement:

```
ALTER TABLE investors REPLACE COLUMNS (company STRING, holdings BIGINT, share DOUBLE);
```

26. Verify the table columns have changed using **DESCRIBE investors**;

27. Restore the original columns:

```
ALTER TABLE investors REPLACE COLUMNS  
(name STRING, amount INT, share DECIMAL(4,3));
```

28. Verify that the table has changed back.

## Postscript

The **investors** table directory didn't move when moved to a different database because it's an unmanaged table. Just as dropping the table won't delete the data, changing the database will not move the data.

Try moving **customers**, which is managed, to the **dig** database, and confirm that the table directory moved in that case. (That is, **/user/hive/warehouse/** should no longer have the **customers/** subdirectory, but **/user/hive/warehouse/dig.db/** should have it instead.) You might need to refresh the display. Be sure to move the table back to the **default** database when you're done.

Mark as completed

 Like



Dislike



Report an  
issue

### Confirm Navigation

Are you sure you want to leave this page?

Stay on this  
Page

Leave this  
Page