Loading Data with Dynamic Partition

One way to load data into a partitioned table is to use *dynamic partitioning,* which automatically defines partitions when you load the data, using the values in the partition column.(The other way is to manually define the partitions. See “Loading Data with Static Partitioning” for this method.)

To use dynamic partitioning, you must load data using an INSERT statement. In the INSERT statement, you must use the PARTITION clause to list the partition columns. The data you are inserting must include values for the partition columns. The partition columns *must* be the rightmost columns in the data you are inserting, and they must be in the same order as they appear in the PARTITION clause.

```
INSERT OVERWRITE TABLE customers_by_country

    PARTITION(country)

    SELECT cust_id, name, country FROM customers;
```

The example shown above uses an INSERT … SELECT statement to load data into the customers_by_country table with dynamic partitioning. Notice that the partition column, country, is included in the PARTITION clause and is specified last in the SELECT list.

When Hive or Impala executes this statement, it automatically creates partitions for the country column and loads the data into these partitions based on the values in the country column. The resulting data files in the partition subdirectories do not include values for the country column. Since the country is known based on which subdirectory a data file is in, it would be redundant to include country values in the data files as well.

Note: Hive includes a safety feature that prevents users from accidentally creating or overwriting a large number of partitions. (See “Risks of Using Partitioning” for more about this.) By default, Hive sets the property hive.exec.dynamic.partition.mode to strict. This prevents you from using dynamic partitioning, though you can still use static partitions.

You can disable this safety feature in Hive by setting the property hive.exec.dynamic.partition.mode to nonstrict:

```
SET hive.exec.dynamic.partition.mode=nonstrict;
```

Then you can use the INSERT statement to load the data dynamically.

Hive properties set in Beeline are for the current session only, so the next time you start a Hive session this property will be set back to strict. Your system administrator can configure properties permanently, if necessary.

Try It!

If you did not create the customers_by_country table in the “Creating Partitioned Tables” reading, do so before continuing. If you did the exercises in the “Loading Data with Static Partitioning” reading before doing this one, drop the table and recreate it (without loading the data).

1. First, look at the contents of the customers_by_country table directory in HDFS. (Where this directory exists depends on which database holds the table.) You can use Hue or an hdfs dfs -ls command to list the contents of the directory. Since you haven't loaded any data, it should be empty.

2. If you want to use Hive, disable the partition safety feature by running

```
SET hive.exec.dynamic.partition.mode=nonstrict;
```

If you want to use Impala for the rest of these exercises, you don't need to run that command.

3. The customers table has only four rows, and each has a different code in the country column. Use the following command to insert the data into the partitioned customers_by_country table:

```
INSERT OVERWRITE TABLE customers_by_country
```

PARTITION(country)

SELECT cust_id, name, country FROM default.customers;

4. Look at the contents of the customers_by_country directory. It should now have one subdirectory for each value in the country column.

5. Look at the file in one (or more, if you like) of those directories, using Hue or an hdfs dfs -cat command. Notice that the file contains the row for the customer from that country, and no others; notice also that the country value is not included.

6. Run some SELECT queries on the partitioned table. Try one that does no filtering (like SELECT * FROM customers_by_country;) and one that filters on country. It's a small table so there won't be a significant difference in the time it takes to run; the point is to notice that you will not query the table any differently than you would query the customers table.