LanguageManual DDL BucketedTables

This is a brief example on creating and populating bucketed tables. (For another example, see Bucketed Sorted Tables.)

Bucketed tables are fantastic in that they allow much more efficient sampling than do non-bucketed tables, and they may later allow for time saving operations such as mapside joins. However, the bucketing specified at table creation is not enforced when the table is written to, and so it is possible for the table's metadata to advertise properties which are not upheld by the table's actual layout. This should obviously be avoided. Here's how to do it right.

First, table creation:

```
CREATE TABLE user_info_bucketed(user_id BIGINT, firstname STRING, lastname STRING)
COMMENT 'A bucketed copy of user_info'
PARTITIONED BY(ds STRING)
CLUSTERED BY(user_id) INTO 256 BUCKETS;
```

Note that we specify a column (user_id) to base the bucketing.

Then we populate the table

```
set hive.enforce.bucketing = true;  -- (Note: Not needed in Hive 2.x onward)
FROM user_id
INSERT OVERWRITE TABLE user_info_bucketed
PARTITION (ds='2009-02-25')
SELECT userid, firstname, lastname WHERE ds='2009-02-25';
```

Version 0.x and 1.x only

The command `set hive.enforce.bucketing = true;` allows the correct number of reducers and the cluster by column to be automatically selected based on the table. Otherwise, you would need to set the number of reducers to be the same as the number of buckets as in `set mapred.reduce.tasks = 256;` and have a `CLUSTER BY ...` clause in the select.

How does Hive distribute the rows across the buckets? In general, the bucket number is determined by the expression `hash_function (bucketing_column) mod num_buckets`. (There's a '0x7FFFFFFF in there too, but that's not that important). The hash_function depends on the type of the bucketing column. For an int, it's easy, `hash_int(i) == i`. For example, if user_id were an int, and there were 10 buckets, we would expect all user_id's that end in 0 to be in bucket 1, all user_id's that end in a 1 to be in bucket 2, etc. For other datatypes, it's a little tricky. In particular, the hash of a BIGINT is not the same as the BIGINT. And the hash of a string or a complex datatype will be some number that's derived from the value, but not anything humanly-recognizable. For example, if user_id were a STRING, then the user_id's in bucket 1 would probably not end in 0. In general, distributing rows based on the hash will give you a even distribution in the buckets.

So, what can go wrong? As long as you use the syntax above and `set hive.enforce.bucketing = true` (for Hive 0.x and 1.x), the tables should be populated properly. Things can go wrong if the bucketing column type is different during the insert and on read, or if you manually cluster by a value that's different from the table definition.