Partitioned tables

Why partitions?

- Partitions can help us boost the performance of our queries. Create a partitioned version of dummy, by initial letter of name*/

First create a view that will include the partition key (in this case initial) to feed the table

```
CREATE VIEW dummy_view AS
SELECT id, val, name, location_id,
substr(name,1,1) as initial
FROM dummy_normalized;
```

Now we create partitioned table with the same structure as `dummy_normalized`, partitioned by `initial`

```
CREATE TABLE dummy_partitioned (
  id bigint,
  val int,
  name string,
  location_id smallint
  )
PARTITIONED BY (initial string) STORED AS PARQUET;
```

And insert into the table from the view:

```
INSERT INTO dummy_partitioned PARTITION(initial)
SELECT * FROM dummy_view;
```

Exercise

- Use the `big_dummy.csv` file in the training environment.
- Assume that the granularity of your queries is `state`
- Design a storage strategy that could help you boost the speed of your queries. This can be:
  Normalization, *Storage as Parquet*, *Create Partitions*. Or even a combination of them.
- Compare your strategy against the 'pure text' benchmark (simply uploading the table and writing the query).
- Consider queries of type
  - `SELECT state, AVG(val) FROM table GROUP BY state;`
  - `SELECT state, AVG(val) FROM table WHERE state = 'California';`