This fairly trivial query was picked because it demonstrates something close to a worst-case here, where instrumenting the query causes the result to slow dramatically, to almost 10X as long. Using `EXPLAIN ANALYZE` is great for getting real times, but you shouldn't assume the exact proportions or time to be the same when running the query normally.

Hot and cold cache behavior

Returning to the regular version of the query seen previously, that it executed in 7.994 milliseconds. This represents "hot" cache behavior, meaning that the data needed for the query was already in either the database or operating system caches. It was left behind in cache from when the data was loaded in the first place. Whether your cache is hot or cold (not in the cache) is another thing to be very careful of. If you run a query twice with two different approaches, the second will likely be much faster simply because of caching, regardless of whether the plan was better or worse.

You can look at how long a query against the entire table takes as a way to measure the effective transfer rate for that table. In the hot cache case, it gives you an idea how fast data moves between two sections of memory:

```
SELECT pg_size_pretty(CAST(pg_relation_size('customers') / 7.994 * 1000
as int8)) AS bytes_per_second;
 bytes_per_second
------------------
 465 MB
```

As this was run on a simple laptop, getting 456 MB/s of rows processed by a query is respectable. In this case, repeatedly running the query takes around the same amount of time each run, which means that the amount cached is staying constant and not impacting results. In this case, it's 100% cached.

Clearing the cache

The exact way you clear all these caches out, to get cold cache performance again, varies based on operating system. Just stopping the database server isn't enough, because the operating system cache can be expected to still have plenty of information cached. On Linux, you can use the `drop_caches` feature to discard everything it has in its page cache. Here's a complete example of cleaning the data out of memory for this database on Linux:

```
$ pg_ctl stop
$ sudo su -
# sync
```

```
# echo 3 > /proc/sys/vm/drop_caches
# logout
$ pg_ctl start -l $PGLOG
```

The `sync` here is to try and flush all data to disk before we just blow away the caches. This `drop_caches` feature on Linux is not intended for regular production server use, it is more of a debugging feature that's potentially dangerous.

Re-running the same benchmark shows quite different performance:

```
$ psql -d dellstore2
dellstore2=# \timing
Timing is on.
dellstore2=# SELECT count(*) FROM customers;
 count
-------
 20000
Time: 204.738 ms
dellstore2=# SELECT pg_size_pretty(CAST(pg_relation_size('customers') /
204.738 * 1000 as int8)) AS bytes_per_second;
 bytes_per_second
------------------
 18 MB
```

Now you're seeing hard drive sequential read speeds—18 MB/s from a laptop hard drive, and this data isn't necessarily even contiguous. It's hard to achieve full drive speed on something so small though.

Repeating the query now returns to the original speed we know to expect from a hot cache run:

```
dellstore2=# SELECT count(*) FROM customers;
Time: 8.067 ms
```

Tests against real data sets need to be very careful to recognize whether their data is already in the cache or not. The usual technique is to run each query three times. If the first is much slower than the second and third, it probably started with a cold cache. If the times are all the same, the cache was likely hot before starting. And if all three vary in some other way, there's probably another variable involved besides just whether the data is in cache. It may take a larger number of queries to extract the pattern for why speeds vary.