Query 1: **Having+Join**
Default:

```
-> Sort: spl.split_id  (actual time=2.867..2.885 rows=270 loops=1)
    -> Filter: (count(spl.split_id) > 1)  (actual time=2.602..2.771 rows=270 loops=1)
        -> Table scan on <temporary>  (actual time=0.000..0.041 rows=622 loops=1)
            -> Aggregate using temporary table  (actual time=2.600..2.707 rows=622 loops=1)
```

CREATE Index Idx on Splits(split_id):

```
-> Sort: spl.split_id  (actual time=6.708..6.727 rows=270 loops=1)
    -> Filter: (count(spl.split_id) > 1)  (actual time=6.407..6.579 rows=270 loops=1)
        -> Table scan on <temporary>  (actual time=0.001..0.043 rows=622 loops=1)
            -> Aggregate using temporary table  (actual time=6.405..6.515 rows=622 loops=1)
```

CREATE Index Idx on Customer(age):

```
-> Sort: spl.split_id  (actual time=2.776..2.794 rows=270 loops=1)
    -> Filter: (count(spl.split_id) > 1)  (actual time=2.504..2.677 rows=270 loops=1)
        -> Table scan on <temporary>  (actual time=0.000..0.043 rows=622 loops=1)
            -> Aggregate using temporary table  (actual time=2.503..2.612 rows=622 loops=1)
```

CREATE Index Idx on Customer(name):

```
-> Sort: spl.split_id  (actual time=2.941..2.962 rows=270 loops=1)
    -> Filter: (count(spl.split_id) > 1)  (actual time=2.669..2.837 rows=270 loops=1)
        -> Table scan on <temporary>  (actual time=0.000..0.041 rows=622 loops=1)
            -> Aggregate using temporary table  (actual time=2.667..2.775 rows=622 loops=1)
```

Analysis:
Based on our collected data, we conclude that Age index is the fastest.
After analysis, we think it is because Age is an attribute that is not related to anything.
We perform no joins from Age, so different columns won't affect the operation costs.
Furthermore, since age is an int, the hashing times are faster averaged over many queries.