# fn get_margin

Michael Shoemate

This proof resides in **"contrib"** because it has not completed the vetting process.

Proves soundness of `get_margin` in `mod.rs` at commit f5bb719 (outdated[1]).

`get_margin` returns a `Margin` for a given set of grouping columns (`by`) whose descriptors are no more restrictive than what is known in `FrameDomain`.

## 1 Hoare Triple

### Precondition

#### Compiler-verified

Types matching pseudocode.

#### Human-verified

None

### Pseudocode

```
def get_margin(domain: FrameDomain, by: set[Expr]) -> Margin:
    margin = next(  #
        (m for m in domain.margins if m.by == by),
        Margin.by(by)
    )

    subset_margins = [  #
        margin
        for margin in domain.margins
        if margin.by.issubset(by)
    ]

    margin.max_length = min(  #
        m.max_length
        for m in subset_margins
        if m.max_length is not None
    )

    from math import prod

    all_mngs = [  #
        (m.by, m.max_groups)
        for m in domain.margins
        if m.max_groups is not None
    ]
```

[1] See new changes with `git diff f5bb719..6d7f19a5 rust/src/domains/polars/frame/mod.rs`

```
26      mngs_covering = find_min_covering(grouping_columns, all_mngs)
27      if mngs_covering is not None:
28          margin.max_num_partitions = prod(v for _, v in mngs_covering)
29
30      all_infos = (  #
31          m.invariant
32          for m in domain.margins
33          if by.issubset(m.by)
34      )
35      margin.invariant = max(all_infos, key={None: 0, "Keys": 1, "Lengths": 2}.get)
36
37      if not by:
38          if margin.invariant is None:
39              margin.invariant = Invariant.Keys
40          margin.max_groups = 1
41
42      return margin
```

### Postcondition

Returns a `Margin` that describes properties of members of `domain` when grouped by `by`.

## 2 Proofs

*Proof.* On line 2, `margin` is either a valid margin descriptor for `by`, by the definition of `domain`, or it is the default margin, which is a valid margin descriptor for all potential datasets.

We now update descriptors based on other information available in `domain`.

On line 7, `subset_margins` is the subset of margins spanned by `by`. Then 13 assigns the smallest known descriptors over any margin spanning a subset of the grouping columns.

If `max_partition_length` is known about a coarser data grouping (when grouped by fewer columns), then these descriptors still apply to a finer data grouping, as partition length or per-partition contributions can only decrease when more finely splitting data.

Therefore `max_partition_length` remain valid after mutation.

Line 21 retrieves all known `max_groups` descriptors. There are no manual preconditions to `find_min_covering`, therefore we claim the postcondition, that the output is a covering for `by`.

The number of partitions can be no greater than the cardinality of the cartesian product of the partitions for each of the grouping keys. Therefore the code finds a set of `max_groups` descriptors that covers the grouping columns, and then updates `margin` to the product of the covering.

On an aside, for utility, while the covering found may not be the smallest, the greedy algorithm will always choose a singleton cover if it is available, therefore this update to the descriptor cannot increase the descriptor.

Finally, on 30, `all_infos` contains descriptors for grouping key invariants for any margin that includes `by`. If partition keys and/or lengths are known for a finer partitioning, then they are also known for a coarser partitioning. Therefore `invariants` is updated to the most permissive descriptor for a partitining as fine or finer than `by`.

Since the initial margin (2) was valid, and all updates have also been shown to be valid, `get_margin` returns a `Margin` that describes properties of members of `domain` when grouped by `by`. □