fn match_num_groups_predicate

Michael Shoemate

September 14, 2025

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

Proves soundness of `match_num_groups_predicate` in `mod.rs` at commit `f5bb719` (outdated[1]).

1 Hoare Triple

Precondition

Compiler Verified

Types matching pseudocode.

Precondition

None

Function

```
def match_num_groups_predicate(
    ranks: Expr,
    partition_by: Vec[Expr],
    identifier: Expr,
    threshold: u32,
) -> Optional[Bound]:
    # check if is a rank function
    if not isinstance(ranks, Expr.Function) or not isinstance(
        ranks.function, FunctionExpr.Rank
    ):  #
        return None

    input = ranks.input
    options = ranks.function.options

    if partition_by != [identifier]:  #
        raise "num_groups truncation must use the identifier in the over clause"

    if not isinstance(options.method, RankMethod.Dense):  #
        raise "num_groups truncation's rank must be dense"

    if len(input) != 1:  #
        raise "rank function must be applied to a single input"

```

[1] See new changes with `git diff f5bb719..1478d47 rust/src/transformations/make_stable_lazyframe/truncate/matching/mod.rs`

```
25      input_item = input[0]
26
27      # Treat as_struct as a special case that represents multiple columns.
28      if isinstance(input_item, Expr.Function) and isinstance(
29          input_item.function, FunctionExpr.AsStruct
30      ):  #
31
32          # If the first field is a hash of the second field,
33          # then interpret the grouping columns as the hash input.
34          # The second field disambiguates hash collisions when ranking.
35          if isinstance(input_item, Expr.Function) and isinstance(
36              input_item.function, FunctionExpr.Hash
37          ):  #
38              hash_input = input_item.input
39              if hash_input.get(0) == input.get(1):
40                  if not isinstance(hash_input.get(0), Expr.Function) or not isinstance(
41                      hash_input.get(0).function, FunctionExpr.AsStruct
42                  ):
43                      raise f"expected hash input to be a struct, found {hash_input}"
44                  input = hash_input.get(0)
45
46          by = set(input_item.input)
47      else:
48          by = {input_item}
49
50      return Bound(by=by, per_group=None, num_groups=threshold)  #
```

Postcondition

Theorem 1.1 (Postcondition)**.** If `ranks` is a dense ranking of grouping columns, and `partition_by` is a singleton of `identifier`, then returns the bound on per-identifier contributions, or an error if the truncation is mis-specified.

Proof. Due to the ambiguity between matching predicates that bound `num_groups` or `per_group`, an error is only raised if the predicate is unambiguously a `per_group` truncation predicate.

The `per_group` predicate is only unambiguously identified. This check happens on line 10.

Line 16 checks that `partition_by` is a singleton of the `identifier`, meeting the conditions of the postcondition.

We now check whether the truncation predicate is well-defined, on lines 19 and 22.

Finally, line 30 extracts the grouping columns from the predicate. Special consideration is made for structs, which are considered multiple grouping columns.

Further consideration is made if the first field is a hash of the second field, on line 37. This effectively randomizes the ranking of the grouping columns, as the ranking is based on a lexicographic ordering. and gives in effect a random sample of grouping columns. The second field of grouping columns is kept in the ranker for the rare case that there are hash collisions, which prevents different combinations of grouping keys from being assigned the same rank, preventing more distinct groups being kept than is permitted.

This predicate corresponds to a `num_groups` truncation predicate, because the over expression groups by the `identifier` column, and within each group, a dense ranking is applied to unique combinations of the grouping columns. If the only rows kept are those corresponding to grouping keys assigned dense ranks less than `threshold`, then each user identifier will have at most `threshold` unique combinations of the grouping columns after filtering by the predicate.

Therefore the bound on user contributions constructed on line 50 is valid. □