Windowing Componentization

Harish Butani

March 5, 2013

1 Definitions

Partitioning as defined before. Implies a Partition and Order definition.

Windowing Component is a subset of Window Functions that have the same Partitioning. A *Component* may depend on 1 or more other Components; the dependency implies the order in which Component Functions are evaluated.

2 Language support (and restrictions)

- Partition and Order expressions can only refer to input columns
- You cannot have 2 Components with the same Partitioning but different dependencies. For e.g the following will not be supported:

```
select a,b,
   rank() as r over (partition by x, order by y),
   sum(r) as s over (partition by x, order by y)
from tab1
```

rank and *sum* will be part of the same component and so sum's argument cannot be resolved.

3 Inferring Partition and Order specification, concept of a 'Default Partitioning'

- Currently we require all UDAFs operate on the same Partition & Order specification.

- So as a convenience we support the concept of a 'Default' specification. A UDAF that doesn't have a Partition-Order specified infers it from the 'Default'
- The Default is:
  - If there is a Query level Partition and Order, this is taken as the Default.
  - The first Window Function or Window with a Partition specified is taken as the Default.
- With the support multiple Partitions, only the Query level Partition-Order will be used as Default.

4 Componentization

Pass 1 go over Window Functions that have an explicit Partitioning and create Windowing Components. Try to translate the Functions based on the Input RR.

- If there is a explicit Query level default partitioning then setup a Windowing Component for it. Add any Window Functions to this Windowing Component that are remaining (the ones with no explicit partitioning specified) and also resolve using Input RR.

At the end of Pass 1, at least one Windowing Component must have all functions translated. If not this is an error state. Error is cannot find Partitioning that only depends on input columns.

Associate the translated components with the input RR. Built a current RR based on the union of fields across these components.

Pass 2 Now repeatedly go over remaining Window Components. Try to translate all its functions based on current RR. If you succeed, associate this component with the current RR; and then add functions from this Component to the current RR.

Repeat pass 2 until all Components are translated or no new Window Components are translatable. If there are any remaining Components that are not fully translated, this is an Error state.

Maintain a sequence of RRs that are build over the above translation steps.

Pass 3, traverse the sequence of RRs find all functions that translate using the first RR. If there are any, add them to any component associated with this RR. Change the next RR in the sequence to include all the functions translated in this step.

Repeat using the next RR in the sequence. Stop when all functions are translated or we reach the end of the RR sequence. If we reach the end of the sequence and there are still untranslated functions this is an error state.

5 Overall flow

- Do Windowing Componentization
- Walk Windowing Component graph based on dependencies and build a PTFDesc for each Component.
- Return a Graph of PTFDescs