Windowing Specifications in HQL

Butani

March 27, 2013

1 Definitions

- **Window Specification:** comprises of a *Partition Spec.*, *Order Spec.*, *Window Frame*, and a *source name for a Window Def.* All 4 components are optional.
- **Partition Specification:** comprises of 1 or more *expressions.*
  - Expressions cannot be aggregations and cannot contain Window specifications.
  - **Standard Compliance:** we defer from the SQL standard here, which only allows *expressions* to be Column References.
- **Order Specification:** comprises of 1 or more *(expression, ordering)* combinations. An *ordering* is **ASC** or **DESC**, with the default being **ASC**.
  - Same rules apply as Partition Spec expressions.
  - **Standard Compliance:** we defer from SQL standard in the same way.
  - **Null Handling:** Hive doesn't support Nulls First/Last specification. In Hive NULLs are returned first.
  - **Collating Clause:** is not supported in HQL.
- **Window Frame:** has a *Frame Type*, a *start Boundary* and optionally an *end Boundary.*
- **Window Frame Type:** can be **ROW** or **RANGE**
- **Window Frame Boundary:** has a *Direction* and an *amount.*

- **Boundary Direction:** can be **PRECEDING** or **FOLLOWING**
- **Boundary Amount:** can be an Unsigned Int constant or the keyword **UNBOUNDED**
  - the Sql Standard allows amount to be any expression of type *Unsigned Int*. We are only allowing this to be a constant.
- **Window Definition:** A Query may contain one or more *Window Definitions*. A Window Definition is a named *Window Specification*.
  - A Query may not contain 2 Window definitions with the same name.
  - Window Definitions cannot have a cyclic dependency.
- **Window Frame Exclusion:**
  - **Standard Compliance:** we don't plan to support this feature in release 1.

2 Effective Window Specification

A UDAF invocation that has an **over** clause will have an *Effective Window Specification* applied to it.

2.1 Inheritance of Window Specifications

- A Window Spec. that has no Partition Spec will inherit it from its Source Window Spec, if there is a source Window Spec.
- A Window Spec. that has no Order Spec will inherit it from its Source Window Spec, if there is a source Window Spec.
- A Window Spec. that has no Window Frame Spec inherit it from its Source Window Spec, if there is a source Window Spec.
- **Standard Compliance:** inheritance rules are not very clear in the standard. Please provide feedback/guidance on this point.

2.2 Effective Window Frame

- A Window Frame that has only the /start/boundary, then it is interpreted as:

```
BETWEEN <start boundary> AND CURRENT ROW
```

- A Window Specification with an Order Specification and no Window Frame is interpreted as:

```
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

This is a *deviation* from the SQL Standard. Since we treat this as a Range Window Frame, the Order By clause can contain 1 Expression. The SQL Standard doesn't impose this restriction; even though the set of rows imposed by this case is what you would get if you had specified a Range Window Frame from Unbounded Preceding and Current Row.

- A Window Specification with no Order and no Window Frame is interpreted as:

```
ROW BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
```

3 Semantics of a Window Specification

- Given a row **R** in the input table, it belongs to the Partition based on its Partition Specification expressions. If there is no Partition Specification all rows belong to the same Partition.

- Given a row **R** its order in its Partition is based on its Order Specification expressions. If there is no Order Specification, the order is implementation dependent, but must be stable. *How do we achieve this stable order? Any suggestions.*

3.1 Window Frame of a row

- goal is to find the **Range** [start, end) of rows that span this Window for row **R**.

3.1.1 When there is no Window Frame

- this is covered by the Effective Window Frame Rules. An Effective Window Frame is associated with the Window Spec.

3.1.2 A Row based Window Frame

Range Start Computation:

Boundary1.type	Boundary1.amt	Behavior
PRECEDING	UNBOUNDED	start = 0
PRECEDING	unsigned int	start = R.idx - Boundary1.amt
CURRENT ROW		start = R.idx
FOLLOWING	UNBOUNDED	Error
FOLLOWING	unsigned int	start = R.idx + b1.amt

Range End Computation:

Boundary2.type	Boundary2.amt	Behavior
PRECEDING	UNBOUNDED	Error
PRECEDING	unsigned int	end = R.idx - Boundary2.amt
		b2.amt == 0 => end = R.idx + 1
CURRENT ROW		end = R.idx + 1
FOLLOWING	UNBOUNDED	end = Part Spec.size
FOLLOWING	unsigned int	end = R.idx + b2.amt + 1

3.1.3 A Range based Window Frame

Range Start computation:

Use Case	Boundary1 type	Boundary1 amt	Sort Key	Order	Behavior
1.	PRECEDING	UNB	ANY	ANY	start = 0
2.	PRECEDING	unsigned int	NULL	ASC	start = 0
3.				DESC	scan backwards to row R2 such that R2.sk is not null start = R2.idx + 1
4.	PRECEDING	unsigned int	not NULL	DESC	scan backwards until row R2 such that R2.sk - R.sk > amt start = R2.idx + 1
5.	PRECEDING	unsigned int	not NULL	ASC	scan backward until row R2 such that R.sk - R2.sk > bnd1.amt start = R2.idx + 1
6.	CURRENT ROW		NULL	ANY	scan backwards until row R2 such that R2.sk is not null start = R2.idx + 1
7.	CURRENT ROW		not NULL	ANY	scan backwards until row R2 such R2.sk != R.sk start = R2.idx + 1
8.	FOLLOWING	UNB	ANY	ANY	Error
9.	FOLLOWING	unsigned int	NULL	DESC	start = partition.size
10.				ASC	scan forward until R2 such that R2.sk is not null start = R2.idx
11.	FOLLOWING	unsigned int	not NULL	DESC	scan forward until row R2 such that R.sk - R2.sk > amt start = R2.idx
12.				ASC	scan forward until row R2 such that R2.sk - R.sk > amt

Range End computation:

Use Case	Boundary2 type	Boundary2 amt	Sort Key	Order	Behavior
1.	PRECEDING	UNB	ANY	ANY	Error
2.	PRECEDING	unsigned int	NULL	DESC	end = partition.size()
3.				ASC	end = 0
4.	PRECEDING	unsigned int	not null	DESC	scan backward until row R2 such that R2.sk - R.sk > bnd.amt end = R2.idx + 1
5.	PRECEDING	unsigned int	not null	ASC	scan backward until row R2 such that R.sk - R2.sk > bnd.amt end = R2.idx + 1
6.	CURRENT ROW		NULL	ANY	scan forward until row R2 such that R2.sk is not null end = R2.idx
7.	CURRENT ROW		not null	ANY	scan forward until row R2 such that R2.sk != R.sk end = R2.idx
8.	FOLLOWING	UNB	ANY	ANY	end = partition.size()
9.	FOLLOWING	unsigned int	NULL	DESC	end = partition.size()
10.				ASC	scan forward until row R2 such that R2.sk is not null end = R2.idx
11.	FOLLOWING	unsigned int	not NULL	DESC	scan forward until row R2 such R.sk - R2.sk > bnd.amt end = R2.idx
12.				ASC	scan forward until row R2 such R2.sk - R2.sk > bnd.amt end = R2.idx

4 Other differences from SQL Standard

- **Notion of inheriting from Query level Distribute/Sort:** this behavior has been removed.
- **Notion of supporting a filter on output Partitions:** using the *Having* clause. Will be removing this behavior.
- **Specification of a Range based Window:** is not based on the Sort expression; the user can specify another expression to compute the

Window. Though this maybe more flexible than typing the Window to the Sort Key, it is very confusing; will remove this behavior.

- **Notion of UDAFs that shouldn't have a Window Frame:** Rank, Dense_Rank, Percent_Rank, Cume_Dist, Lead, Lag are configured this way. But (as far as I can tell) nothing about this in the Standard. Should we keep this or remove it?
- **Notion of Lead/Lag UDFs usable as arguments in UDAFs:** Allows certain patterns to be expressed more easily like delta sum: 'sum(price - lag(price,1)'. Should we keep this?