Partitioned Table Functions with Hive

Why

- Support basic Dimensional Analytics
  - Ordering: ranking, accumulation, lead, lag
  - Hierarchical Navigation: parent, children, sibling
  - Time Series Analysis: Npath
  - Assumption Dimension not 'flat & very large'. Time, Geography, Product classification, Customer, Supplier fit into this mold.
- Expose functionality in SQL as windowing clauses & table functions.
- Open to other Use cases expressible as Parallelizable Functions
  - For eg Market Basket Analysis: Parallelizable Iterative Algorithm

What

- Query abstraction is a basic select statement:
  - Specify how data is partitioned and ordered
  - Windowing clauses
  - Table function invocation
  - Input to processing is a Hive Table or Query.
- Provided as a CLI & API
  - windowingCli: hive service: interperse hive queries with windowing queries.
  - Simple API: create WindowingShell, execute Query.

What: Examples

- **Basic Query**
  - *Rank Parts within Manufacturer by weight*

```
from part_rc
partition by p_mfgr
order by p_mfgr, p_size
with
  rank() as r
select p_mfgr,p_name, p_size, r
```

- **TopN**
  - *Calculate the Top 3 Tracts(based on land area) by County.*

```
from <select county, tract, arealand
           from geo_header_sf1
           where sumlev = 140>
partition by county
order by county, arealand desc
with
   rank() as r, sum(arealand) over rows between unbounded preceding and current row as cum_area
select county, tract, arealand, r, cum_area
where <r < 3>
```

What: Examples

- **Time Series Analysis using NPath table function** (Details here)
  - *List incidents where a Flight(to NY) has been more than 15 minutes late 5 or more times in a row.*

```
from npath(<select origin_city_name, year, month, day_of_month, arr_delay, fl_num
                    from flightsdata
                    where dest_city_name = 'New York' and dep_time != ''>
                partition by fl_num
                order by year, month, day_of_month,
                'LATE.LATE.LATE.LATE.LATE+',
                <[LATE : "arr_delay \\> 15"]>,
                <["origin_city_name", "fl_num", "year", "month", "day_of_month",
                            ["(path.sum() { it.arr_delay})/((double)count)", "double", "avgDelay"],
                            ["count", "int", "numOfDelays"]
                ]>)
select origin_city_name, fl_num, year, month, day_of_month, avgDelay, numOfDelays
```

What: Examples

- **Hierarchical Evaluation** (Details here)
  - *For a Country, State, City Geo hierarchy compute: % of Country Sales, Top City Sales, Avg. City Sales*

```
from hierarchyEvaluate(
  < select Country, State, City, sum(Sales) from Sales group by Country, State, City>
    partition by Country
    order by Country, State, City,
<['Country', 'State', 'City']>,
<[
   ["Sales / Ancestor('Country', 'Sales') * 100.0", "% Country"],
   ["TopN(Descendants('City'), 'Sales', ,1)", "Top City"],
   ["Avg(Descendants('City'), Sales)", "Avg. City"]
]>)
select Country, State, City, '% Country', 'Top City', 'Avg. City'
```

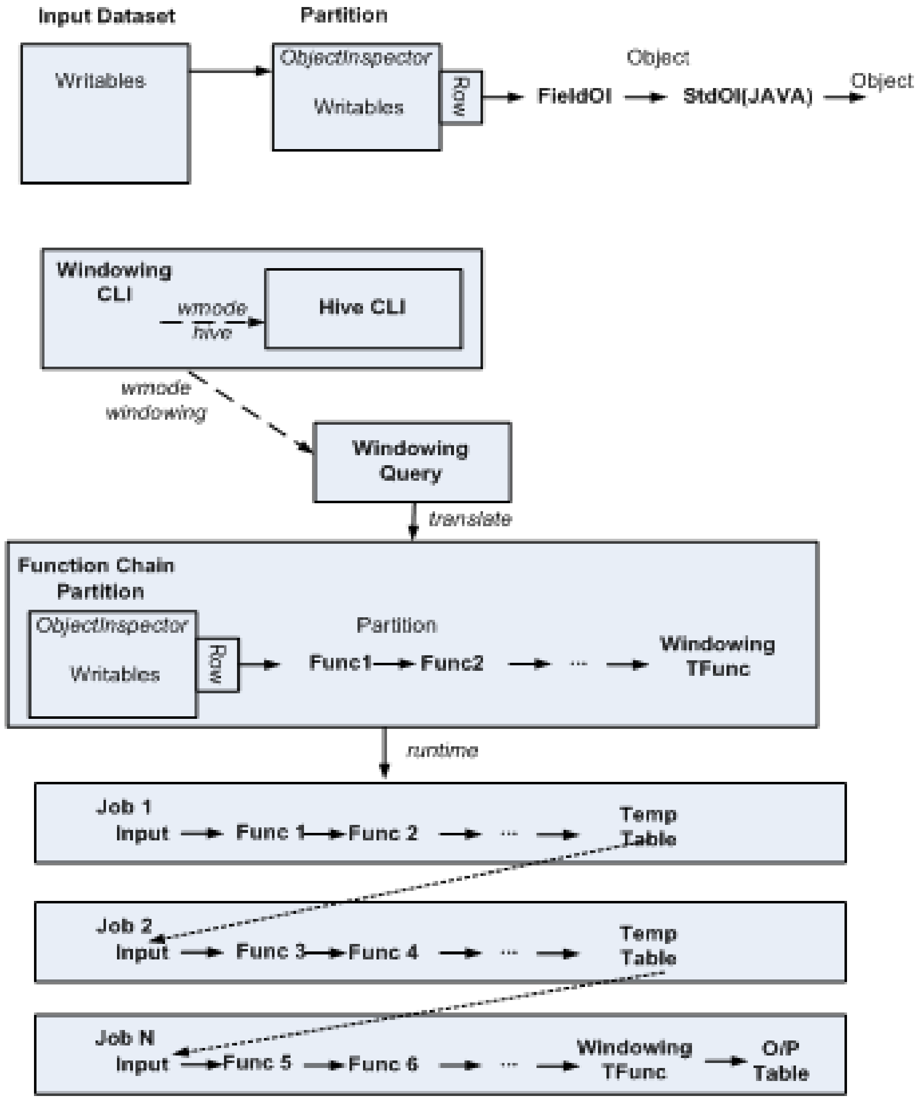
- **Market Basket Analysis** (Details here)
  - *Calculate the Frequent Itemsets from a Basket Dataset.*

```
from frequentItemsets(
   <select l_orderKey, p_name
    from lineitem join part
          on lineitem.p_partkey = part.p_partkey
    order by l_orderkey
   >
partition by l_orderkey)
select itemSet
```

How (details here)

- **Partitions** are containers of Rows.
  - Represented as a Spillable List of Writables + ObjectInspector
- Rows exposed as Groovy Binding; Partition exposed as Groovy iterable.
- All Evaluation in context of Row + Partition [+ Window]
- A **PartitionedTableFunction** (PTF) given a Partition computes an output Partition.
  - An invocation of PTF can also specify how input dataset should be partitioned and ordered.
  - A PTF defines shape of Output.
  - A PTF may operate on raw data before it is partitioned and ordered.
- A Query is a chain of PTFs.
  - Input of chain is a Hive Query or table
- Windowing clauses are syntax sugar for the special Windowing Table function.
- .Query translated to a series of Jobs. Each Job executes part of the entire Function Chain
  - In the Map side the first function in the subchain can optionally reshape the Raw data.
  - The Reduce side streams partitions through a chain of PTFs.
  - The output is written to hdfs and exposed as a Temporary Table to be used by the next Job in the Chain.

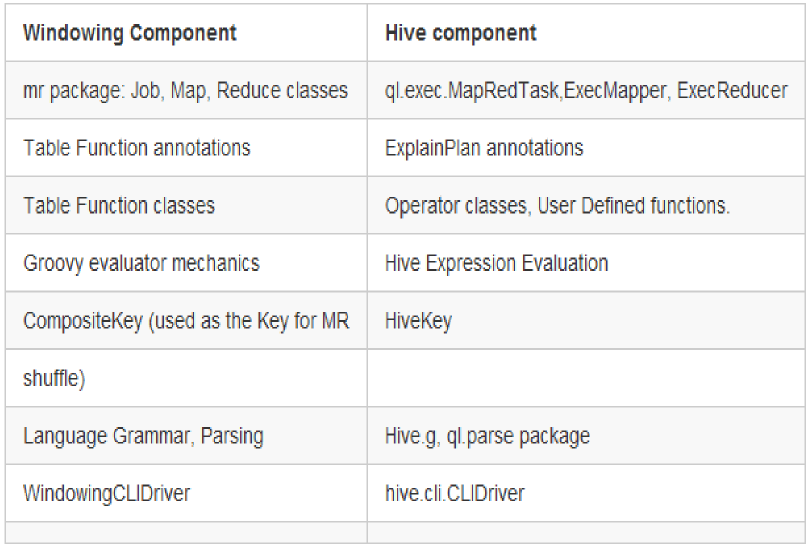
How



Input Dataset
Writables
Partition
ObjectInspector
Writables
Row
FieldOI
Object
StdOI(JAVA)
Object
Windowing CLI
wmode hive
Hive CLI
wmode windowing
Windowing Query
translate
Function Chain Partition
ObjectInspector
Writables
Row
Partition
Func1
Func2
...
Windowing TFunc
runtime
Job 1
Input
Func 1
Func 2
...
Temp Table
Job 2
Input
Func 3
Func 4
...
Temp Table
Job N
Input
Func 5
Func 6
...
Windowing TFunc
O/P Table

Fold into Hive

- Mapping concepts/components to Hive



Windowing Component	Hive component
mr package: Job, Map, Reduce classes	ql.exec.MapRedTask,ExecMapper, ExecReducer
Table Function annotations	ExplainPlan annotations
Table Function classes	Operator classes, User Defined functions.
Groovy evaluator mechanics	Hive Expression Evaluation
CompositeKey (used as the Key for MR	HiveKey
shuffle)	
Language Grammar, Parsing	Hive.g, ql.parse package
WindowingCLIDriver	hive.cli.CLIDriver

A Path to moving into Hive (Details here)

- Step 1: Move to Hive MR mechanics for Job execution
- Step 2: Move to Hive Evaluators and Expressions
- Step 3: Introduce the concept of Pure Table Functions in Hive
- Step 4: Allow Table function invocations to appear in Table Expressions
- Step 5: Extend HQL with Windowing Clauses

Summary

- Partition Table Functions for basic Dimensional Analysis in SQL; open for other use cases, possibly Iterative algorithms.
- Currently on top of Hive; possibly fold into Hive (gradually)
- Available at https://github.com/hbutani/SQLWindowing/wiki