OQL Aggregate functions and UDA

Background

Geode allows querying data stored in regions using query language called OQL. Currently the query engine supports COUNT aggregate function but missing other standard functions like MIN, MAX, AVG, SUM. This document is to propose implementing Aggregate functions (with GROUP BY extension) and also to support UDA(user defined aggregate) functionalities with a common a frame work.

Requirement

Enhance the OQL engine to support:

- Aggregate functions (AVG , MAX, MIN, COUNT, SUM)

- Associated GROUP BY clause

- A common framework to support both Aggregate and UDAs

Proposal

Provide a common interface that will be used to support both standard (internal) and UDAs.

OQL engine will provide a common interface "com.gemstone.gemfire.cache.query.Aggregator" that will defined the methods used to support both standard aggregates and UDA.

The interface "Aggregator" is described in details in the later section of the document (UDA).

OQL Grammar changes :

- Support non distinct order by
- Detect Group By Clause
- Detect Group By Path expressions
- Detect Aggregate functions in the projection attribute

Query correctness criteria:

If the query contains Group By Clause, then the projection attribute should only contain aggregate functions & only those columns which are present in Group By Clause.

E.g.:

select col1, avg(col2) from /portfolio group by col1 // Valid query

select col1, col3, avg(col2) from /portfolio group by col1 // invalid query (col3 missing from group by clause)

If the query does not contain Group By clause then its projection can either not contain any aggregate functions or they can only contain aggregate functions.

E.g.:

select avg(col2) from /portfolio // Valid

select col1, avg (col2) from /portfolio (col1 should not be present in projection attributes) // invalid

Query Types & implementation:

Implementation for queries containing only group by clause & projection containing only columns (no aggregate function)

Select pf.ID , pf.status as status from /portfolio pf where pf.ID > 100 group by pf.ID , status

The above query can be converted into a distinct – order by query to get the result.

Transformed query => select distinct pf.ID , pf.status as status from /portfolio pf where pf.ID > 100 order by pf.ID, status

Replicated Region query: For replicated executing the transformed query should yield the correct results.

Partitioned Region: Should work without change (assuming distinct order by is supported for PR)

Implementation for queries containing only group by clause & projection containing at least one aggregate function & zero or more columns

select pf.status, AVG(pf.ID) from /portfolio pf where pf.ID > 0 group by pf.status

Replicated Region query:

Get the projected rows (by stripping the aggregate functions)

Sort the projected rows based on the Group By Columns

For each row of sorted data , feed the row to the aggregate functions in the projection & maintain the aggregate.

Prepare the resultset at the end of complete row scan.

Partitioned Region query:

For each Bucket:

Get the projected rows (by stripping the aggregate functions)

Sort the projected rows based on the Group By Columns

For each sorted row , feed the row to the aggregate functions in the projection & maintain the aggregate.

Prepare the resultset at the end of complete row scan.

For the aggregate function AVG , it should also keep track of the number of rows used to produce the result of AVG.

For all buckets present locally

The result of each bucket needs to be merged locally, to get the resultant resultset. The group by columns fetched in the projection attribute would be used to merge & calculate the resultant resultset

For the query node

Resultset from each node needs to be merged in the same manner as for all buckets merged locally.

Final resultset computed

Modification in current Querying Engine and things to consider:

The current OQL engine needs to support ***non distinct order by clause***. The order by comparator attributes which map to run time iterators, need to map to projection attributes, so that order by clause, can be applied on the query node without firing another order by query. The results from individuals bucket nodes need to be N - way merged, as the results from individual nodes comes sorted.

In case of PR, for certain types of aggregate functions, the implementation of the aggregate function on bucket nodes, would defer from the implementation of the aggregate functions for the query node.

E.g.: select col1, avg(col2) from /portfolios group by col1.

The computed average value for each bucket , cannot be just fed in the avg() function on the query node. The avg() function on each bucket node needs to pass , the sum as well as the number of elements , to the query node. The query node avg function would take these two as input & compute a final value of avg.

For the queries which involve distinct keyword, also requires difference in implementation on the bucket node & query nodes.

E.g.: select col1, sum(distinct col2) from /portfolios group by col1.

Here it will not be possible to pass the individual sum calculated from each of the bucket nodes, to be passed to the query node to compute the final sum. The reason being that we need to sum only distinct values of col2 and it is possible two buckets have the same value & so only one such value should be added to sum. In such case, the sum() function on the bucket node needs to return a Set of the values seen, rather than sum of those values.

In built aggregate functions:

MAX: Usage : MAX(Expression)

example : MAX(pf.ID) : The expression must evaluate to a java.lang.Comparable type

MIN: Usage : MIN(Expression)

example : MIN(pf.ID) : The expression must evaluate to a java.lang.Comparable type

AVG: Usage : AVG(Expression)

example : AVG(pf.ID) : The expression must evaluate to java.lang.Number type.

AVG: Usage : AVG (distinct expression)

SUM: Usage : SUM(Expression)

example : SUM(pf.ID) : The expression must evaluate to java.lang.Number type.

SUM: Usage : SUM(distinct expression)

COUNT(*): Already supported

COUNT (expression | column)

COUNT (distinct expression | column)

User Defined Aggregate Function (UDA):

OQL Engine will provide following interface to be implemented by the user

com.gemstone.gemfire.cache.query.Aggregator

The Aggregator interface will have following methods :

```
public interface Aggregator {

  /**
   * Accumulate the next scalar value
   * @param value
   */
  public void accumulate(Object value);

  /**
   * Initialize the Aggregator
   */
  public void init();

  /**
   * @return Return the result scalar value
   */
  public Object terminate();

  /**
   * Merges the incoming aggregator from bucket nodes with the resultant aggregator
   * on the query node
   * @param otherAggregator
   */
  public void merge(Aggregator otherAggregator);
}
```

There should be a zero arg constructor present in the implementing class.

Example of query using built in aggregate

> select pf.ID , SUM(pos.mktValue) from /portfolios pf , pf.positions pos group by pf.ID
>
> The above query will give the total market value of all the positions for a portfolio identified by its ID
>
> **Note:** As a first cut , we will make it mandatory for the group by column to be part of projection attribute (Will see if we can make implement in a reasonable period the code to relax this constraint. i.e group by columns not be part of projection columns)

Proposed Syntax for using UDA:

Creation of UDA

Using API:

QueryService qs = CacheUtils.getQueryService();

qs.createUDA(“udaAlias”, "com.gemstone.gemfire.cache.query.dunit.UDACreationDUnitTest$SumUDA");

XML based creation:

```
<uda-manager>
   <uda name="uda2" class="com.gemstone.gemfire.cache.query.dunit.UDACreationDUnitTest$UDACLass2"/>
   <uda name="uda3" class="com.gemstone.gemfire.cache.query.dunit.UDACreationDUnitTest$UDACLass3"/>
</uda-manager>

cache-9.0 xsd
<xsd:element maxOccurs="1" minOccurs="0" name="uda-manager">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="uda">
        <xsd:complexType>
          <xsd:attribute name="name" type="xsd:string" use="required" />
          <xsd:attribute name="class" type="xsd:string" use="required" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Using UDA in Query:

String queryStr = "select p.status , myUDA(p.ID) from /portfolio p where p.ID > 0 group by p.status order by p.status";

This will require us to maintain a registry of the aggregate functions defined & will also have to ensure that it gets created on newly joining nodes.

UDA Schema exchange:

The UDA schema/definitions are exchanged between cluster nodes via profile exchange mechanism.