Terminology

GROUPING SETS
The GROUPING SETS clause in GROUP BY allows us to specify more than one GROUP BY option in the same record set. All GROUPING SET clauses can be logically expressed in terms of several GROUP BY queries connected by UNION.

GROUPING__ID
Is a virtual column used with GROUP BY GROUPINGS SETS, ROLLUP and CUBE.
It composes a bit vector with the "0" and "1" values for every column which is GROUP BY section. "1" is or a row in the result set if that column has been aggregated in that row. Otherwise the value is "0".
e.g. In GroupBy expression (a, b, c),
(a, b, c), bit 0 represents the expression “a”, bit 1 represents the expression “b”, etc.

0, 1, 1 => 110(b) = 6, then the grouping set is (b, c)
1, 0, 0 => 001(b) = 1, then the grouping set is (a)
1, 0, 1 => 101(b) = 5, then the grouping set is (a, c)

Examples:

Aggregate Query with GROUPING SETS	**Equivalent Aggregate Query with GROUP BY**
SELECT a, b, SUM(c) FROM tab1 GROUP BY a, b GROUPING SETS ((a,b))	SELECT a, b, SUM(c) FROM tab1 GROUP BY a, b /*GROUPING__ID=3, grouping set(a, b) */
SELECT a, b, SUM(c) FROM tab1 GROUP BY a, b GROUPING SETS ((a,b), a)	SELECT a, b, SUM(c) FROM tab1 GROUP BY a, b, 3 /*GROUPING__ID=3 grouping set(a,b)*/ UNION SELECT a, null, SUM(c) FROM tab1 GROUP BY a, null, 1 /*GROUPING__ID=1, grouping set(a), “b” is not in this grouping set, then it will be considered as constant null in the projection*/
SELECT a,b, SUM(c) FROM tab1 GROUP BY a, b GROUPING SETS (a,b)	SELECT a, null, SUM(c) FROM tab1 GROUP BY a, null,1 /*GROUPING__ID=1, grouping set(a) b is considered as constant null*/ UNION SELECT null, b, SUM(c) FROM tab1 GROUP BY null, b, 2 /*GROUPING__ID=2, grouping set(b), a is considered as constant null*/
SELECT a, b, SUM(c) FROM tab1 GROUP BY a, b GROUPING SETS ((a, b), a, b, ( ))	SELECT a, b, SUM(c) FROM tab1 GROUP BY a, b, 3 /*GROUPING__ID=3*/ UNION SELECT a, null, SUM(c) FROM tab1 GROUP BY a, null, 1 /*GROUPING__ID=1*/ UNION SELECT null, b, SUM(c) FROM tab1 GROUP BY null, b, 2 /*GROUPING__ID=2*/ UNION SELECT null, null, SUM(c) FROM tab1 GROUP BY null, null, 0/*GROUPING__ID=0*/
SELECT MAX((a+b)*c), b FROM tab1 GROUP BY a+b, b GROUPING SETS (a+b, b)	SELECT MAX(null*c), b FROM tab1 GROUP BY null, b, 2 /*GROUPING__ID=2*/ UNION

Aggregate Query with GROUPING SETS	Equivalent Aggregate Query with GROUP BY
	SELECT MAX((a+b)*c), null FROM tab1 GROUP BY a+b, null, 1 /*GROUPING__ID=1*/ • Hive currently (0.12) doesn't support the grouping set expression appear within the aggregation expression, but we DO.

Cubes

CUBE CLAUSE creates a subtotal of all possible combinations of the set of column in its argument. Once we compute a CUBE on a set of dimension, we can get answer to all possible aggregation questions on those dimensions.

e.g.

GROUP BY a, b, c WITH CUBE is equivalent to

GROUP BY a, b, c GROUPING SETS ((a, b, c), (a, b), (b, c), (a, c), (a), (b), (c), ( )).

Rollups

ROLLUP clause is used with GROUP BY to compute the aggregate at the hierarchy levels of a dimension. GROUP BY a, b, c with ROLLUP assumes that the hierarchy is "a" drilling down to "b" drilling down to "c".

GROUP BY a, b, c, WITH ROLLUP is equivalent to GROUP BY a, b, c GROUPING SETS ((a, b, c), (a, b), (a), ( )).

General Idea of Grouping Set Computation

Thus all GROUPING SET clauses can be logically expressed in terms of several GROUP BY queries connected by UNION, in practice, we prefer a better way, which **expands the input rows and then send to a single Aggregate execution**.

How to expand the source input rows?

For example:

```
SELECT a, b FROM src GROUP BY a, b GROUPING SETS ((a, b), a, b, ())
```

Assume rows in table "src" are:

1, 2

3, 4

According to the semantic of the GROUPING SETS, we can expand the input row into 8 rows (4 grouping sets multiple 2 rows):

1, 2	(bitmask 3: a, b)
1, null	(bitmask 1: a)
null, 2	(bitmask 2: b)
null, null	(bitmask 0:)
3, 4	(bitmask 3: a, b)
3, null	(bitmask 1: a)

null, 4 (bitmask 2: b)

null, null (bitmask 0:)

And then the above 8 rows will be feed into a single Aggregate Operator, but the GROUP BY keys are no longer be "**a, b**", but "**a, b, GROUPING__ID**". (GROUPING__ID is the grouping set bitmask)

We have a physical operator "GroupingSetExpansion", it has a pre-computed variable in the type of Array[Projection], the Projection is indexed by the GROUPING__ID. During the pre-computation, the non-selected (according to the GROUPING__ID) grouping sets appear in the raw projection expression trees will be replaced with Literal (null).

What if non-attributes in the GROUPING SETS?

For example: SELECT MAX (a+b+c), d FROM src GROUP BY a+b, d GROUPING SETS ((a+b), d)

The "a+b" is null when it's not in the grouping set (when the GROUPING__ID is 0 or 2). In order to simplify the computation, we will insert a temporal Projection node, which kind of like:

SELECT (MAX (e+c), d FROM **(SELECT a+b as e, d FROM src) t** GROUP BY e, d GROUPING SETS (e, d)

Cube & Rollup

Semantically, we can convert both Cube & Rollup into GroupingSet, the only difference is the bitmasks; we have the following algorithm (In Scala) to calculate the bitmasks for them:

CUBE**:** (0 to 2^expression.length – 1)

ROLLUP**:** (0 to expressions.length).map(idx => { (1 << idx) – 1 }

References

https://cwiki.apache.org/confluence/display/Hive/Enhanced+Aggregation,+Cube,+Grouping+and+Rollup