2.7 집계 연산 (201~, 16)

7.4 그룹화 셋

- 여러 그룹에 걸쳐 집계할 수 있도록, 여러 집계를 결합하는 저수준 기능.
- null 값을 제거하지 않는다면 부정확한 결과를 얻게 됨.

ex1) 아래와 같이 GROUPING SETS 를 지정하는 경우, 각 (id, code) 조합에 대해 sum(qty) 집계 처리.

```
SELECT id, code, sum(qty) as sum FROM dfNotNull GROUP BY id, code
GROUPING SETS ( (id, code) ) ORDER BY id DESC, codd DESC +-----+-----+-
----+ |id |code |sum | +-----+-----+-----+ |1001 |c01 |101 | |1002 |c02
|101 | | ... | ... | ... | +-----+-----+-----+
```

SQL

ex2) 아래와 같이 GROUPING SETS를 지정하는 경우, 각 (id, code)조합과 전체 sum 집계 처리.

```
SELECT id, code, sum(qty) as sum FROM dfNotNull GROUP BY id, code
GROUPING SETS ( (id, code), () ) ORDER BY id DESC, codd DESC +-----+---
--+-----+ |id |code |sum | +-----+-----+-----+ |null |null |202 | <---
추가 |1001 |c01 |101 | |1002 |c02 |101 | | ... | ... | ... | +-----+----
-+-----+
```

SQL

7.4.1 롤업

-

group-by 스타일의 다양한 연산을 수행할 수있는 다차원 집계 기능
- rollup 대상이 되는 group-by 컬럼 목록은 계층적으로 다루어진다. 즉, 순서가 중요.

ex1) 총합(null, null), 날짜별 총합(date, null), 날짜별 국가별 총합(date, country) 기준으로 집계

순서가 중요함. date, country 로 rollup을 했기 때문에 국가별 총합은 나오지 않는다.

```
dfNotNull.rollup( "Date", "Country" ) .agg( sum("Quantity") ) .selectExpr( "Date", "Country", "`sum(Quantity)` as Total" ) .orderBy ( "Date", "Country" ) .show() +----------+--------------+-------+ | Date| Country| Total| +----------+--------------+-------+ | null| null|5176450| <--- 총합 |2010-12-01| null| 26814| <--- 날짜별 총합 |2010-12-01| Australia| 107| <--- 날짜별 국가별 총합 |2010-12-01| EIRE| 243| |2010-12-01| France| 449| | ... | ... | ... | |2010-12-02| null| 21023| <--- 날짜별 총합 |2010-12-02| EIRE| 4|<--- 날짜별 국가별 총합 |2010-12-02| Germany| 146| | ... | ... | ... | +----------+--------------+-------+
```

Scala

7.4.2 큐브

- 모든 기준으로 집계를 수행한다.
- 롤업 대비 모든 차원에 대해 동일한 작업을 수행한다. 롤업처럼 요소들을 계층적으로 다루지 않는다.

ex1) 총합(null, null), 날짜별 총합(date, null), 국가별 총합(null, country),

날짜별 국가별 총합(date, country) 기준으로 집계. 순서 상관없이 모든 기준으로 집계.

```
dfNotNull.cube( "Date", "Country" ) .agg( sum("Quantity") ) .selectExpr( "Date", "Country", "`sum(Quantity)` as Total" ) .orderBy ( "Date", "Country" ) .show() +----------+----------+----------+ |Date |Country |Total | +----------+----------+----------+ |null |null |5176450 | <--- 총합 |2012-10-01|null |1234567 | <--- 날짜별 총합 |2012-10-01|germany |12345 | <--- 날짜별 국가별 총합 |2012-10-01|italy |12345 | |2012-10-01|france |12345 | | ... | ... | ... | |2012-10-02|null |1234567 | <--- 날짜별 총합 |2012-10-02|germany |12345 | <--- 날짜별 국가별 총합 |2012-10-02|italy |12345 | |2012-10-02|france |12345 | | ... | ... | ... | |null |germany |12345 | <--- 국가별 총합 |null |italy |12345 | | ...
```

```
| ... | ... | +----------+----------+----------+ +----+----------------
--+-------+ |Date| Country| Total| +----+-----------------+-------+
|null| null|5176450| <-- 총합 |null| Australia| 83653| <-- 국가별 총합
|null| Austria| 4827| |... | ... | ... | <-- 날짜별 총합 <-- 날짜별 국가별
총합 <-- 날짜별 총합 <-- 날짜별 국가별 총합
```

Scala

7.4.3 그룹화 메타데이터

- 집계 수준을 확인하기 위해 grouping_id를 제공. 개별 그룹화 ID 값을 반환
- 2^n만큼의 ID 값이 생성됨 (n은 group-by 컬럼 수)
- 0부터 시작. grouping_id가 0인 경우는 가장 낮은 레벨의 총합
- grouping_id가 2^n -1 인 경우는 가장 높은 레벨로써, 전체 총합

ex1) 2개 컬럼 Date, Country를 기준으로 grouping_id 확인.

- gid 3 : 총합
- gid 2 : 나라별 총합
- gid 1 : 날짜별 총합
- gid 0 : 날짜별 나라별 총합

```
dfNotNull.rollup( "Date", "Country" ) .agg( sum("Quantity"),
grouping_id() ) .selectExpr( "Date", "Country", "`sum(Quantity)` as
Total", "`grouping_id()` as gid" ) .orderBy ( "Date" ) .show() +-------
---+--------------+-------+---+ | Date| Country| Total|gid| +----------
+--------------+-------+---+ | null| null|5176450| 3| |2010-12-01|
null| 26814| 1| |2010-12-01| Australia| 107| 0| |2010-12-01|United
Kingdom| 23949| 0| |2010-12-01| EIRE| 243| 0| | ... | ... | ... |...|
|2010-12-02| null| 21023| 1| |2010-12-02|United Kingdom| 20873| 0|
|2010-12-03| Portugal| 65| 0| | ... | ... | ... |...| +----------+-----
---------+-------+---+
```

Scala

7.4.4 피벗

- 로우를 컬럼으로 변환
- groupBy 컬럼, 집계 컬럼(?)만 남는다.
-

집계 컬럼(?)의 네이밍은 "피벗컬럼_집계함수(다른컬럼)" 형태로 된다.

ex1) date로 groupBy하고 Country로 pivot 후 sum() 집계함수를 실행 시킨 결과

USA_sum(Quantity)만 조회하고 있으나, "USA", "Quantity"외에도 다양한 컬럼이 존재한다.

```
val pivoted = dfWithDate.groupBy("date") .pivot("Country") .sum()
pivoted.where("date > '2011-12-05'") .select("date",
"`USA_sum(Quantity)`") .show() +----------+-----------------+ |
date|USA_sum(Quantity)| , USA_sum(UnitPrice) , Brazil_sum(Quantity) ,
... +----------+-----------------+ |2011-12-06| null| |2011-12-09|
null| |2011-12-08| -196| |2011-12-07| null| +----------+---------------
--+ pivoted.printSchema root |-- date: date (nullable = true) |--
Australia_sum(Quantity): long (nullable = true) |--
Australia_sum(UnitPrice): double (nullable = true) |--
Australia_sum(CustomerID): long (nullable = true) |--
Austria_sum(Quantity): long (nullable = true) |--
Austria_sum(UnitPrice): double (nullable = true) |--
Austria_sum(CustomerID): long (nullable = true) |--
Bahrain_sum(Quantity): long (nullable = true) |--
Bahrain_sum(UnitPrice): double (nullable = true) |--
Bahrain_sum(CustomerID): long (nullable = true) |--
Belgium_sum(Quantity): long (nullable = true) |--
Belgium_sum(UnitPrice): double (nullable = true) |--
Belgium_sum(CustomerID): long (nullable = true) |--
Brazil_sum(Quantity): long (nullable = true) |-- Brazil_sum(UnitPrice):
double (nullable = true) |-- Brazil_sum(CustomerID): long (nullable =
true) |-- Canada_sum(Quantity): long (nullable = true) |--
Canada_sum(UnitPrice): double (nullable = true) |--
Canada_sum(CustomerID): long (nullable = true) ...
```

Scala

7.5 사용자 정의 집계 함수 (user-defined aggregation function)

- UDAF는 직접 제작하는 함수나 비즈니스 규칙에 기반을 둔 자체 집계 함수를 정의하는 방법
- 스칼라, 자바에서만 가능
- UserDefinedAggregateFunction을 상속받아 구현

inputSchema, bufferSchema, dataType, deterministic, initialize, update, merge, evaluate

-

- 스파크의 입력 데이터의 모든 그룹의 중간 결과는 단일 AggregationBuffer에 저장해 관리

ex1)

```scala
import org.apache.spark.sql.expressions.MutableAggregationBuffer import
org.apache.spark.sql.expressions.UserDefinedAggregateFunction import
org.apache.spark.sql.Row import org.apache.spark.sql.types._ class
BoolAnd extends UserDefinedAggregateFunction { def inputSchema:
org.apache.spark.sql.types.StructType = StructType(StructField("value",
BooleanType) :: Nil) def bufferSchema: StructType = StructType(
StructField("result", BooleanType) :: Nil ) def dataType: DataType =
BooleanType def deterministic: Boolean = true def initialize(buffer:
MutableAggregationBuffer): Unit = { buffer(0) = true } def
update(buffer: MutableAggregationBuffer, input: Row): Unit = {
buffer(0) = buffer.getAs[Boolean](0) && input.getAs[Boolean](0) } def
merge(buffer1: MutableAggregationBuffer, buffer2: Row): Unit = {
buffer1(0) = buffer1.getAs[Boolean](0) && buffer2.getAs[Boolean](0) }
def evaluate(buffer: Row): Any = { buffer(0) } }
```

Scala

```scala
val ba = new BoolAnd spark.udf.register("booland", ba) import
org.apache.spark.sql.functions._ spark.range(1)
.selectExpr("explode(array(TRUE, TRUE, TRUE)) as t")
.selectExpr("explode(array(TRUE, FALSE, TRUE)) as f", "t")
.select(ba(col("t")), expr("booland(f)")) .show() +----------+---------
-+ |booland(t)|booland(f)| +----------+----------+ | true| false| +----
------+----------+
```

Scala