The Complete Reference: Grouping dataset using JPA and Criteria Builder

Sachin Shridhar Padhye

Katy, Texas, U.S
sysachinp@gmail.com

Abstract

This paper presents high level design of grouping data on its and its children's attribute using Java Persistence API (JPA) and criteria builder future. Grouping of the data set is defined as view of the data set based on unique value of one or more attributes. This paper describes the design and approach to construct grouping criteria dynamically. Grouping criteria can be defined in JSON format and then JSON will be parsed to create grouping query run time. JPA is implemented by many leading OR mapping tools, which represents data Object Oriented Programming Models. Criteria builder API provides a way to construct SQL queries dynamically against data objects

Keywords: Software Architecture, Microservices, Database, Object Oriented Programing, OR tools, JPA, Criteria Builder, SQL

Introduction

In any software application data represents information which drives business operations. Data is created and modified as per business rules which implements the business operation. There are different database systems available to persist data based on the nature of the data. In software applications data sets, which are related to each other and to save this data-set relational database is typically used. Data consists of keys and values. Keys in data provide the context for values. In relational database system, data is stored in table. Each row represents one dataset, keys are the columns, values are saved in corresponding cell. One of the common requirements of the business is to group the data set based on keys and its specific values and then consolidate the data set which is qualified for the group. E.g. Assume relational database stores Person's details. So, there will be Person table, attributes which identifies the person e.g. name, age, gender are columns and each cell stores value for that attribute, like John Doe,35, Male. Below the sample Person data in relational database.

id	first name	last-named	age	gender	City	state
1	John	Doe	28	Male	New York	NY
2	Jane	Smith	34	Female	Los Angeles	CA
3	Michael	Johnson	45	Male	Chicago	IL
4	Emily	Davis	22	Female	Austin	TX
5	David	Wilson	30	Male	Seattle	WA
6	Sarah	Martinez	29	Female	Miami	FL

id	first name	last-named	age	gender	City	state
7	James	Anderson	52	Male	Denver	CO
8	Linda	Thomas	39	Female	Phoenix	AZ
9	Robert	Taylor	41	Male	Houston	TX
10	Olivia	Moore	26	Female	San Francisco	CA

Table 1.1 Sample Person Data

There could be a business requirement to group peopleonGender, state and Age between ranges (0-5 ,6 -12, 13-18, 19-25,26-40, 40-60, 60+). There could be following groups can be created Male -> TX -> 40-60, Female -> TX -> 19-22, Male -> CA ->26-40 and so on.

Structured Query Language (SQL) supports group by cause and specify multiple columns. Grouping criteria can be defined at the run time so number of grouping attributes changes at run time. Client calling grouping needs to know the list of data key on which grouping is possible, also different filter may need to be applied before create grouping results. To implement it full dynamically, caller needs to know list of keys available to define grouping criteria and values available for corresponding keys.

This complexity increases when grouping criteria is implemented on keys which are at the children of the primary data set. E.g. Data Set A has one to many relationships with Data set B and Data set B has one to many relationships with Data set C, to defined grouping criteria on data set B or Data set C first Data set B and C needs to be left joined with Data set A. Also, because of joining the table having one to many relationships, primary key of the Data set A may be repeated only distinct value of primary key should be returned.

Another challenge is that, grouping value may be needs to be derived can be defined in the range. E.g. In above example people needs to be grouped in one of the 4 ranges of the age. 1) 0 to 25 2) 26 to 50 3) 51 to 75 4) 76 and above. Another example could be based on state, person needs to be grouped in Northeast, Midwest, South and West region. This can be implemented by using Case / When expression in SQL.

Below is generic SQL statement that can be used generate the groups.

```
Select
[Group_Attribute_1,Group_Attribute_2,….Group_Attribute_N],
Case
WhenGroup_Attribute_N1_Condition1thenResult_N1,
WhenGroup_Attribute_N1_Condition2thenResult_N2,
WhenGroup_Attribute_N1_Condition3thenResult_N3,
OtherwiseResult_N4
END AS  Group_Attribute_N1,
Count( distinct PK_DATA_SET_A ),
Group_concat( distinct PK_DATA_SET_A)

From DATA_SET_A
```

```
LEFT Join DATA_SET_B on A_B_FK=A_PK
LEFT Join DATA_SET_C on B_C_FK=B_PK
 …
where[Group_AttributesConditions…]

groupByGroup_Attribute_1,Group_Attribute_2,…..Group_Attribute_N,Group_Attribute_N1

order byGroup_Attribute_1,Group_Attribute_2,…..Group_Attribute_N,Group_Attribute_N1

limit PAGE_SIZE

offset OFFSET_RESULTS
```

Algorithm

Entire algorithm can be divided into four parts 1) Defining Grouping Criteria. 2) Construct and Execute Query 3) Construct Response 4) Support pagination and available filters. This algorithm uses Object Relational Mapping Tool which is commonly used to convert relational data in Object and Criteria builder API, which allows programmatically construct complex SQL queries.

Defining Grouping Criteria

In this step user of an applications can define grouping criteria to construct the grouping results. In OR tool each data set is referred as Entity, Entity contains attributes and relation to other entities. So an API needs to be exposed which will give list of keys on which grouping criteria can be constructed.

```
[
{
"keyName":"attributeA1",
"dataType":"String"
},
{
"keyName":"attributeA2",
"dataType":"Date"
},
{
"keyName":"attributeA3",
"dataType":"Integer"
},
{
"keyName":"datasetB.attributeB1",
"dataType":"String"
},
{
"keyName":"datasetB.dataSetC.attributeC1",
"dataType":"String"
}
]
```

Based on the available keys and imposed systems restrictions, user can create the grouping rules. These grouping rule will be saved in user's profile, whenever user logs in and get group, this grouping rule will be applied to present groups as per the configurations. Also, use can edit and update these rules as needed. System for Cross-domain Identity Management (SCIM) protocol can be used to define filter rules. There are open-source libraries to parse SCIM based filter string and construct the Criteria builder API objects. Below is the example of Grouping Rules represented in JSON format.

```
{
"groupingRule":[
{
"displayName":"Attribute_1 Display Name",
"filterName":"attribute1"
},
{
"displayName":"Attribute_2 Display Name",
"filterName":"attribute2"
},
{
"displayName":"Attribute_3 Display Name",
"filterName":"attribute3",
"rule":[
{
"when":"attribute3 lt 18",
"then":"less than 18"
},
{
"when":"attribute3 ge 18 and attribute3 le35",
"then":"Between 18 and 35"
},
{
"when":"attribute3 ge 36 and attribute3 le 55",
"then":"Between 36 and 55"
},
{
"when":"attribute3 ge 56",
"then":"56 +"
},
{
"when":"default",
"then":"N/A"
}
]
},
{
"displayName":"Dataset B Att 1 Name",
"filterName":"datasetb.attributeB1"
```

```
},
{
"displayName":"Dataset C Att 1 Name",
"filterName":"datasetc.attributeC1"
}
]
}
```

Construct And Execute Query

Next whenever user make grouping request, system needs to apply Grouping rules to return grouping results. Important component of the requests are as follow.

a) **groupingFilter** – User may want to filter groups on one of the grouping key and values. In above grouping rules user may need specific groups whose "datasetc.attributeC1" is "VALUE_C1". There could be zero or more group filters can be applied. If no groupFilter is provided then default grouping rule set by the user will be applied.

b) **itemFilter** – These filters are to filter groups on not Grouping attributes. In other words. User may need to filter groups based on attributes which are not used for creating the group. In above example, user may need to get groups which contains at least one item whose "attribute4" is "ATTRIBUTE_4_VALUE". Key difference between "groupingFilters" and "itemFilters" are grouping filters regenerate the groups where as itemFilters just filters out group which does contain any items matching to itemFilters.

c) **pageNumber** – This is used for pagination where user can request specific page when many groups are available.

d) **pageSize** – Number of results needs to be displayed on the page.

e) **includeMetaData** – This parameter drives inclusion of meta data information along with grouping results. E.g. all possible values for grouping filters. By default, this parameter will be false.

f) **sortList** – This parameter tells to sort results based on grouping attributes.

g) **includeTotalResults** – This parameter drives inclusion of total results in response.

Below is the sample request.

```
{
"startIndex":1,
"itemsPerPage":10,
"sortList":["attribute1 asc","attribute2 desc"],
"itemFilter":"attribute4 co VALUE_1",
"groupFilter":"attribute3 eq VALUE_3_1 or VALUE_3_2 eq VALUE_4",
"includeMetaData":false,
"includeTotalResults":true
}
```

Once user request is received by the backend, backend will merge it with group config rules created by the user and add additional default system level filtering if required. Then grouping configuration will be sent to Criteria creation component which will create criteria query, execute the query and send response back.

1) Create Join Map if any grouping attributes are present on child tables i.e. in our example it will be Join with Data_Set_B and Data_Set_C. Key to the Map is data set name. e.g. For datasetB.attributeB1 , datasetB.datasetC.attributeC1 grouping keys below map will be created.

```
datasetB-> dataSetA.join("datasetB");
datasetC-> dataSetA.join("datasetB").join("datasetC");
```

2) Create List of javax.persistence.criteria.Selection<X>Object based on grouping properties.
3) If Database supports "group_concat" function, then add additional "javax.persistence.criteria.Selection<X>" object to get concatenated id.
4) Add "javax.persistence.criteria.Selection<X>" object to get number of records groups for given grouping criteria.
5) Create complex Select objects for derived column using Case statement. Below pseudo code shows how Select clause can be constructed using criteria building API.

```
List<Selection<?>> selections =newArrayList<>();
for(String column : columns){
      selections.add(root.get(column));
}
Expression<String> groupConcatExpr = cb.function(
"group_concat", cb.literal("DISTINCT "),String.class, root.get("id")
);
Expression<Long> count = cb.countDistinct(root.get("id");
Expression<String> ageGroup = cb.selectCase()
.when(cb.lessThan(personRoot.get("age"),18),"Under 18")
.when(cb.between(personRoot.get("age"),18,35),"18-35")
.when(cb.between(personRoot.get("age"),36,50),"36-50")
.otherwise("51+");
selections.add(count);
selections.add(groupConcatExpr);
selections.add(ageGroup);
```

6) Create javax.persistence.criteria.Predicates to construct filters for grouping criteria.

```
Predicate[] predicates =newPredicate[3];
predicate [0]= cb.in(root.get(filterName1)).value(inputValue1);
predicate [1]= cb.in(root.get(filterName2)).value(inputValue2);
predicate [2]= cb.in(join.get(filterName3)).value(inputValue3);
```

7) Create groupBy and sort cause for given grouping criteria. Also apply limit and offset for pagination and execute Query.

```
Path<String> prop1Path = root.get("prop1");
Path<String> prop2Path = root.get("prop2");
// Apply GROUP BY
cq.groupBy(prop1Path, prop2Path);
```

```
// Optional: Order results
cq.orderBy(cb.asc(prop1Path), cb.asc(prop2Path));

TypedQuery<Person> query = em.createQuery(cq);
// Apply offset and limit here
query.setFirstResult(offset); // OFFSET
query.setMaxResults(limit); // LIMIT
```

There are opensource Java based SCIM parser libraries available. Couple of them are a) Apache Syncope SCIM Filter Parser b) Captain-P-Goldfish SCIM SDK. parse the SCIM based rules and construct abstract syntax tree (AST) or similar structure. Criteria builder Expression object can be constructed by parsing each AST node.

Construct Response

Next part is to construct the generic response from the results of the criteria query execution. Result in Json response can be represented as follow. Tuple can be used to get the result from the query then use column name (alias name) used in select clause or index to fetch results and construct the response Object. Later Response object can be serialized in JSON to send it back as the response. Below is the JSON response from the grouping results.

```
{
"results":[
{
"count":4,
"ids":["123","345","678","910"],
"groupingFilters":[
{
"displayName":"Attribute_1 Display Name",
"filterName":"attribute1",
"Value":"VALU_1"
},
{
"displayName":"Attribute_2 Display Name",
"filterName":"attribute2",
"value":"VALUE2"
},
{
"displayName":"Attribute_3 Display Name",
"filterName":"attribute3",
"Value":"VALU_3"
},
{
"displayName":"Dataset B Att 1 Name",
"filterName":"datasetb.attributeB1",
"Value":"VALU_4"
},
```

```
{
"displayName":"Dataset C Att 1 Name",
"filterName":"datasetc.attributeC1",
"Value":"VALU_5"
}
]
}
]
}
```


Support pagination and available filters

To support the pagination, caller need to know total count of the groups available. This can be supported by adding “totalResults” in response. This can be calculated by executing the same query just for count and without limit and offset.

Another requirement could be to include meta data. This can be helpful to create grouping filters. This will be achieved by executing query without limit and offset. The response for all possible values of each grouping criteria will be constructed by going through the query results. Below is the sample response for all possible values for grouping criteria.


```
{
"allGroupingFilters":[
{
"displayName":"Attribute_1 Display Name",
"filterName":"attribute1",
"value":[
{
"value":"VALUE1",
"displayName":"VALUE1",
"count":14
},
{
"value":"VALUE2",
"displayName":"VALUE2",
"count":49
}
]
},
{
"displayName":"Attribute_3 Display Name",
"filterName":"attribute3",
"value":[
{
"value":"attribute3 le 18",
"displayName":"less than 18",
"count":5,
```

```
},
{
"value":"attribute3 gt 55",
"displayName":"55 +",
"count":8,
},
      {
"value":"attribute3 gt 18 and attribute3 le 35",
"displayName":"Between 18 and 35",
"count":8,
}
]
}
]
}
```


Challenges and Solutions

First challenge is that group_concat() function is not part of the SQL standard, it's a vendor specific extension. All databases do not support this function. Challenge is to get list of ids of primary dataset keeping code database agnostic. There are two possible solutions.

A) Avoid including list of ids in grouping results, rather have separate API to return list of primary datasets based on the given grouping criteria.

B) Second option will be to get database specific function at runtime. From environment variable read the database name and based on the database name use appropriate SQL function. Below is the list of popular relational databases and equivalent group concatenation functions.

Supports GROUP_CONCAT ()	**Does Not Support GROUP_CONCAT()**
MySQL	PostgreSQL (use STRING_AGG)
MariaDB	Oracle (use LISTAGG)
SQLite	SQL Server (use STRING_AGG)
	DB2 (use LISTAGG)

Table 2- Group Concat method support

Second challenge is, conditionally skip grouping on one of the grouping attributes. E.g. Data_Set_A has attribute A, B, C then if attribute A has value A1 or A2 then skip grouping on B, also If A has value A3 and B has value B1 then Skip grouping on C. This conditional exclusion needs to be incorporate in Grouping rule JSON. Grouping JSON will follow SCIM protocol to define the rule. Below is the sample grouping JSON will look like.


```
{
"groupingFilters":[
{
"displayName":"Attribute A",
```

```
"filterName":"a"
},
{
"displayName":"Attribute B",
"filterName":"b",
"rule":[
{
"when":"a eq A1 or a eq A2",
"then":"null"
},
{
"when":"default",
"then":"b"
}
]
},
{
"displayName":"Attribute C",
"filterName":"c",
"rule":[
{
"when":"a eq A3 and ( b eq 'B1' or b eq null )",
"then":"null"
},{
"when":"default",
"then":"c"
}
]}]}
```

Conditionally skipping grouping can be achieved by using Case-When cause using SQL and Using JPA criteria builder as follow.

SQL

```
Select
   A,
   CASE
      WHEN A=’A1’or A=’A2’ THEN null
      OTHERWISE B
   ENDas B,
   CASE
      WHEN A=’A3’and (B=’B1’ or B is null) THEN null
      OTHERWISE C
   ENDas C
From
Data_Set_A
GroupBy
```

```
A,
CASE
   WHEN A='A1'or A='A2' THEN null
   OTHERWISE B
END,
CASE
   WHEN A='A3'and B='B1' THEN null
   OTHERWISE C
END
```

Criteria Builder

```
Predicate p1 = criteriaBuilder.equal(shipmentRoot.get("a"),"A1");
Predicate p2 = criteriaBuilder.equal(shipmentRoot.get("a"),"A2");


Expression<Object> b = criteriaBuilder.selectCase()
.when(criteriaBuilder.or(p1,p2), criteriaBuilder.nullLiteral(String.class))
.otherwise(shipmentRoot.get("b"));

Predicate p3 = criteriaBuilder.equal(shipmentRoot.get("a"),"A3");
Predicate p4 =criteriaBuilder.equal(shipmentRoot.get("b"),"B1");
Predicate p5 = criteriaBuilder.isNull(shipmentRoot.get("b"));


Expression<Object> c = criteriaBuilder.selectCase()
.when(criteriaBuilder.and(p3,
criteriaBuilder.or(p4,p5)), criteriaBuilder.nullLiteral(String.class))

.otherwise(shipmentRoot.get("c"));

criteriaQuery.multiselect(shipmentRoot.get("a"), b, c);

criteriaQuery.groupBy(shipmentRoot.get("a"), b, c);
```

Use Case

One of the leading logistic software service provider company want to give an ability to group their shipment arriving in united states for custom clearance purpose. User of the software are freight forwarders or individual importer. One of the largest customers of the software is expecting 5K-10K shipments arriving to united states per month at different ports. Each customer have their unique business rules for consolidation, these rules are based on various attributes of the shipments e.g. date of arrival, port of unlading, importer etc. Also, they want to search for groups for specific shipments or group of shipments.
Above design enabled customers to manage their own grouping rule, view shipments as per the grouping rules and filter the groups either on grouping attributes or shipment attributes.

Conclusion

Below diagram describes the design of an application. As shown in below diagram there are three main API with which use communicate with the application. A) Get API for filter options. B) Create, Read, Update, Delete Api for managing Grouping Rules C) Get API for Grouping results. Each component of an application is loosely coupled, at high level Grouping Api will receive request in Json format, if there are any user defined grouping rules configured then those will be applied to construct the final grouping rules. These grouping rules are defined in JSON format, then this JSON is parsed to construct criteria builder API to get the grouping results.

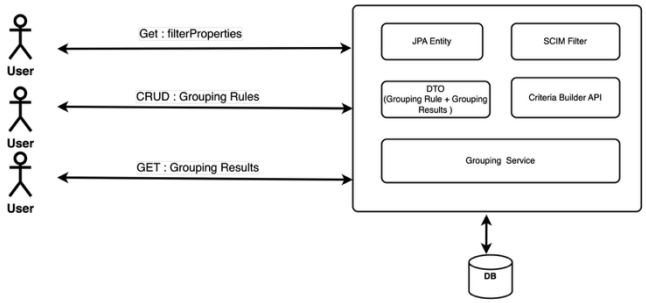

Fig. 1 High level Design

Reference

[1] P. Hunt, K. Grizzle, M. Ansari, E. Wahlstroem, and C. Mortimore, “RFC 7644: System for Cross-domain Identity Management: Protocol,” *IETF Datatracker*, 2015. https://datatracker.ietf.org/doc/html/rfc7644

[2] M. Keith and Merrick Schincariol, *Pro JPA 2 : mastering the Java Persistence API. [Java EE 6 compliant. create robust, data-driven applications with this definitive guide to the new JPA 2]*. Berkeley, Calif.: Apress, 2013.

[3] T. Nield, *Getting started with SQL : a hands-on approach for beginners*. Sebastopol, Ca: O’reilly, 2016.

[4] Smith, Ben. *Beginning JSON*. Apress, 2015.

[5] Pezoa, Felipe, et al. "Foundations of JSON schema." *Proceedings of the 25th international conference on World Wide Web*. 2016.