**Issue :**

Support SQL aggregated functions with Hibernate Search Queries.

**Use Case :**

We have an Entity in Database with about 1200K records. Our application hits the database via Hibernate to display all such results on UI. Since more such entries are regularly getting added to the DB the application is getting really slow. Hence, we thought of using Hibernate search instead of Apache Solr since our application was already using Hibernate.

So we started migrating all our Hibernate queries to Full text search queries.

However it seems that not all queries can be easily migrated. For example : queries which include complex SQL operations like aggregated functions.

Below is an example of such Query –

@Query("select b from RuleOutput b where b.ruleInfo.tagName like %?1% and (b.ruleId, b.updateTime) in (select ruleId, max(updateTime) from RuleOutput where cast(createTime as date) >= ?2 and cast(createTime as date) <= ?3 group by ruleId,originalDate)")

List<RuleOutput> findAllByGroupTagByDate(String groupTagName, Timestamp createTime1, Timestamp createTime2);

Now since aggregate functions like ‘GROUP BY’ or ‘IN’ are not supported we had to build the query using both full text search query and plain java implementation of aggregate functions.

The corresponding java implementation had to use Java collectors and stream for grouping which again increased our computation. Moreover, for each such query we had write separate functions in java using full text search and java functions.

**RuleOutput Entity:**

@Entity

@Indexed

@Table(name = "TDQC\_RUL\_OUTP\_TYP\_CD")

@Getter

@Setter

@NoArgsConstructor

**public** **class** RuleOutput **extends** DomainObject{

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

@Id

@Column(name = "RUL\_OUTP\_TYP\_ID")

**private** Long ruleOutId;

@Field(index = Index.***YES***, analyze=Analyze.***NO***, store = Store.***YES***)

@Column(name = "RUL\_TYP\_ID")

@Facet

**private** Long ruleId;

@Field

@Column(name = "RSLT\_CD")

**private** String result;

@Field

@Column(name = "RUL\_STA\_MSG")

**private** String ruleStatus;

@Field

@Column(name = "REC\_INSRT\_TS")

**private** Timestamp createTime;

@Field(index = Index.***YES***, analyze=Analyze.***NO***, store = Store.***YES***)

@Column(name = "REC\_UPD\_TS")

@Facet

**private** Timestamp updateTime;

@Field

@Column(name = "STA\_DESC")

**private** String statusMessage;

@Field

@Column(name = "COMNT\_TXT")

**private** String comment;

@Field(index = Index.***YES***, analyze=Analyze.***NO***, store = Store.***YES***)

@Column(name = "ODATE")

**private** String originalDate;

@Transient

**private** Long alertCode;

@JsonIgnore

@ManyToOne

@IndexedEmbedded

@NotFound(action=NotFoundAction.***IGNORE***)

@JoinColumn(name = "RUL\_TYP\_ID",referencedColumnName = "RUL\_TYP\_ID",insertable=**false**, updatable=**false**)

**private** Rule ruleInfo;

**Outcome**:

The performance was boosted however, it increased our lines of code to such a level that is seems almost impossible to maintain now. Each time there is a small change in the query we have to completely rewrite our code. We have to constantly check for methods which are no longer supported by java and hibernate.

**How can it help?**

We won’t have to integrate it with java for achieving results. It will make it easier to understand code. It will reduce thousands of lines of code.

Support for such functions will help to achieve code maintenance and reusability.

It will highly reduce risk of breaking any functionality. I am sure it will help many to use hibernate search for their use cases.

**What can be done?**

At least we can introduce ways to use Elastic search's aggregation support from within Hibernate Search. Few alternatives as suggested in the below links section.

**Links:**

* [https://stackoverflow.com/questions/48743482/why-are-aggregate-functions-like-group-by-not-supported-in-hibernate-search/48744246#48744246](https://stackoverflow.com/questions/48743482/why-are-aggregate-functions-like-group-by-not-supported-in-hibernate-search/48744246%2348744246)