Have you ever been frustrated by your query language not grouping numeric or similar data?

<table>
<thead>
<tr>
<th>Region</th>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region Metropolitan</td>
<td>16.753</td>
</tr>
<tr>
<td>Región Metropolitan</td>
<td>11.005</td>
</tr>
<tr>
<td>region metropolitan</td>
<td>76.029</td>
</tr>
<tr>
<td>región metropolitan</td>
<td>5.443</td>
</tr>
<tr>
<td>R.METROPOLITANA</td>
<td>32.639</td>
</tr>
<tr>
<td>Metropolitan Region</td>
<td>11.356</td>
</tr>
</tbody>
</table>
Are you tired of…

...having to export your data into a CSV to perform clustering?
Now Introducing
Clustering for SPARQL

Sebastián Ferrada, Benjamin Bustos and Aidan Hogan
Linköpings Universitet, Sweden
IMFD, Universidad de Chile, Chile

AMW 2023
We extended SPARQL to include clustering

• Semantics extension: new solution modifier
  • How should this modifier work?
  • Relationships with existing solution modifiers

• Syntax extension: new keywords

• Implementation + Performance Evaluation
CLUSTER BY – Semantics

New SPARQL Operator:

\[ \chi_{V,C,c}(M) = \{ C(\pi_V(X), \pi_V(\mu), c) | \mu \in X \} \]

- A Fresh variable to bind the cluster identifier
- The clustering function w/Parameters
- The variables w.r.t the clustering is computed
CLUSTER BY – Semantics

\( C \) extends each solution mapping \( \mu \) in \( X \) with an identifier bound to variable \( c \) that represents the cluster in which \( \pi_V(\mu) \) would be assigned into when considering the space \( \pi_V(X) \) and the clustering algorithm of \( C \)
CLUSTER BY – Semantics

Consider $\mu = \{? x \leftarrow 2, ? y \leftarrow 4\}$ and $V = \{? x, ? y\}$.

$C(\pi_V(X), \pi_V(\mu), c) = \{? x \leftarrow 2, ? y \leftarrow 4, ? c \leftarrow 1\}$
We chose to extend the solution mappings, since creating groups (like `GROUP BY` does) would force us to make clustering and grouping mutually exclusive.
CLUSTER BY – Syntax

We extend the SPARQL Grammar:

\[
\text{SolutionModifier} ::= \text{ClusterByClause? GroupClause? HavingClause? OrderClause? LimitOffsetClauses?}
\]

\[
\text{ClusterByClause} ::= 'CLUSTER BY' \text{ Var+ WithClause ClusterAlias}
\]

\[
\text{WithClause} ::= 'WITH' \text{ iriOrFunction}
\]

\[
\text{ClusterAlias} ::= 'AS' \text{ Var}
\]
CLUSTER BY – Syntax

SELECT ?lifex ?growth ?code ?c WHERE {
    ?c1 wdt:P31 wd:Q6256 ;
    wdt:P2250 ?lifex ;
    wdt:P2219 ?growth ;
    wdt:P297 ?code .
}

CLUSTER BY ?lifex ?growth

WITH sim:kmeans(4) AS ?c
CLUSTER BY – GROUP BY

• The Solution Modifier Order is then:
  • CLUSTER BY
  • GROUP BY
  • HAVING
  • ORDER BY
  • SELECT
  • DISTINCT
  • REDUCED
  • OFFSET and LIMIT
CLUSTER BY – k-means

SELECT ?lifex ?growth ?code ?c WHERE {
  ?c1 wdt:P31 wd:Q6256 ;
  wdt:P2250 ?lifex ;
  wdt:P2219 ?growth ;
  wdt:P297 ?code .
}

CLUSTER BY ?lifex ?growth
WITH sim:kmeans(4) AS ?c
CLUSTER BY and GROUP BY

SELECT AVG(?lifex) MIN(?growth) ?c WHERE {
    ?c1 wdt:P31 wd:Q6256 ;
    wdt:P2250 ?lifex ;
    wdt:P2219 ?growth .
}

CLUSTER BY ?lifex ?growth WITH sim:kmeans(4) AS ?c
GROUP BY ?c

<table>
<thead>
<tr>
<th>AVG(\lifex)</th>
<th>MIN(\growth)</th>
<th>?c</th>
</tr>
</thead>
<tbody>
<tr>
<td>83.3</td>
<td>-5.4</td>
<td>1</td>
</tr>
<tr>
<td>78.6</td>
<td>-3.2</td>
<td>2</td>
</tr>
<tr>
<td>73.1</td>
<td>-18.0</td>
<td>3</td>
</tr>
<tr>
<td>61.9</td>
<td>-13.8</td>
<td>4</td>
</tr>
</tbody>
</table>
CLUSTER BY – k-medoids

  ?iris a ?type;
  tdb:petalLength ?plength ;
  tdb:petalWidth ?pwidth .
}

CLUSTER BY ?plength ?pwidth
WITH sim:kmedoids(3) AS ?c
CLUSTER BY – DBSCAN

SELECT ?temp ?lum ?c WHERE {
  ?star stp:color ?temp ;
  stp:absMagnitude ?lum ;
  FILTER (?error < 75)
  BIND((?lum+1.4444)/(14.7121 + 1.4444) AS ?lum
  BIND((?temp+0.17)/(1.88+0.17) AS ?temp2)
}

CLUSTER BY ?temp2 ?lum2 

WITH sim:dbscan(0.05, 10) AS ?c
Implementation

JavaCC
The most popular parser generator for use with Java applications.

View on GitHub  Download 7.0.10.zip  Download 7.0.10.tar.gz
Performance Evaluation

We run `CLUSTER BY` queries over synthetic Wikidata Queries that retrieve numerical properties.

We compare the `CLUSTER BY` version of each query against the execution time of evaluating the underlying BGP.

We use several parameter combinations for each algorithm.
Performance Evaluation – k-Means

$O(knm)$ Time

The Fastest by far and the most “predictable”
Performance Evaluation – k-Medoids

Naive k-Medoids takes $O(kn^2)$ and a better algorithm takes $O(n^2)$ anyways.

Queries with $n > 5700$ took more than 12 hours to complete.
Performance Evaluation – DBSCAN

DBSCAN’s execution time depends on the distance distribution of the underlying vector space.

The parameters need to be carefully selected beforehand.
Conclusions

• We defined a clustering algebra operator for SPARQL + Syntax
  • Previously introduced by Ławrynowicz (no implementation, no group by)
  • k-Means clustering for SPARQL proposed by Qi et al. but uses INSERT/UPDATE

• CLUSTER BY is readily combinable with all other SPARQL 1.1 operators

• k-Means implementation takes marginally longer to compute than the underlying BGP