University of Zurich[UZH]

**Department of Informatics**

University of Zürich
Department of Informatics
Binzmühlestr. 14
CH-8050 Zürich
Phone. +41 44 635 43 11
Fax +41 44 635 68 09
www.ifi.uzh.ch/dbtg

UZH, Dept. of Informatics, Binzmühlestr. 14, CH-8050 Zürich

Erik Hasselberg

**Prof. Dr. Michael Böhlen**
Professor
Phone +41 44 635 43 33
Fax +41 44 635 68 09
boehlen@ifi.uzh.ch

Zürich, May 6, 2014

**Facharbeit in Informatik**
**Datenbanktechnologie**

**Topic: The Multigranular Robust Nearest Neighbor Join Operator**

Overview:

Given an outer and an inner relation, the RNNJoin is an operator computing the equijoin according to a common identifier, plus the nearest neighbor join according to the time for all outer tuples not having an equijoin match. Since the time is not an identifier (i.e. many tuples could be present in the inner relation with the same timestamp), all nearest neighbors of a given outer tuples are aggregated.

The goal of this Facharbeit is to design, define and implement an extension of the RNNJoin operator that deals with relations having a multigranular timestamp (year granularity, season granularity, day granularity, etc.).

Detailed description: For dealing with different granualirites we introduce the *med*$(r, s)$, i.e. Multigranular Eucledian Distance, that returns the distance from $r$ to $s$. *Med* is not commutative because $r$ specify the granularity to be considered in computing the distance; here is an example of med applied to two timestamps.

- $med$(2013-05-21,2013-05-20) = 1 (day)
- $med$(2013-05-21, Spring 2013) = 62 (days), because Spring 2013 corresponds to a timestamp going from 20 Mar 2013 to 20 Jun 2013; therefore we safely consider the worst case, i.e. the most far extreme
- $med$(Summer 2013, 2013-05-21) = 1 (Season), because 2013-05-21 is in Spring 2013
- $med$(Summer 2013, Autumn 2013)= 1 (Season)

- $med$(Summer 2013, 2013) = 2 (Seasons), we consider the most far season in 2013
- $med$(2013, Summer 2013) = 0 (years)

Given an outer relation $\mathbf{r}$ of schema $\mathbf{R}$ = [$G$,$E$,$T$ ] and in inner relation $\mathbf{s}$ of schema $\mathbf{S}$ = [$G$, $E$, $T$,$F$,$M$ ], the RNNJoin operator is defined as following:

$$\mathbf{r} \overset{\mathrm{NN(T,}\,\boldsymbol{\Phi}\mathrm{)}}{\underset{\mathrm{EQ(E)}}{\bowtie_{\mathrm{G}}}} \mathbf{s} = {}_{\mathbf{R}}\vartheta_{\boldsymbol{\Phi}(\mathbf{S}-\mathbf{R})}\left(\mathbf{r} \bowtie_{E,G} \mathbf{s} \cup \left((\mathbf{r} \triangleright_{E,G} \mathbf{s}) \overset{\mathrm{NN(T)}}{\bowtie_{G}} \mathbf{s}\right)\right)$$

where $\bowtie_{\mathbf{E,G}}$ is the equijoin operator on $E$, $\triangleright_{\mathbf{E,G}}$ the antijoin on $E$, $\overset{\mathrm{NN(T)}}{\bowtie_{\mathbf{G}}}$ the nearest neighbor join with distance *med* on $T$ and $\vartheta$ the aggregation operator. Attribute $G$ is a grouping attribute used for specifying that the join matches of a given $\mathbf{r}$ tuple, must be found among the $\mathbf{s}$ tuples sharing the same $G$ value; T is a temporal attribute.

In the following example, we refer to a tuple $r \in \mathbf{r}$ as a *query point.*
We use the RNNJ for retrieving the Crude Protein $cp$ and the Organic Matter $om$ values of a set of query points $\mathbf{r}$: the $cp$ values are available in $\sigma_{F=cp}(\mathbf{s})$ while the $om$ values in $\sigma_{F=om}(\mathbf{s})$.
For a given query point $r \in \mathbf{r}$, if a tuple $s \in \sigma_{F=cp}(\mathbf{s})$ (or $\sigma_{F=cp}(\mathbf{s})$) does not exists with $s.E = r.E$ and same $G$, we consider as its $cp$ (or $om$) value the temporal closest tuple with same $G$.

r

| | $E$ | $G$ | $T$ |
|---|---|---|---|
| $r_0$ | #111 | Hay | 2013-06-20 |
| $r_1$ | #222 | Hay | Summer 2013 |
| $r_2$ | #333 | Hay | 2013 |
| $r_3$ | #444 | Pea | 2011-07-21 |

s

| | $E$ | $G$ | $T$ | $A$ | $R$ | $F$ | $M$ |
|---|---|---|---|---|---|---|---|
| $\sigma_{F=cp}$ | | | | | | | |
| $s_1$ | #100 | Hay | 2013-06-19 | 1010 | 0.8 | CP | 1.20 |
| $s_3$ | #129 | Hay | 2013-06-21 | 1030 | 1.0 | CP | 1.40 |
| $s_4$ | #323 | Hay | Summer 2013 | 1050 | 0.6 | CP | 1.30 |
| $s_5$ | #456 | Pea | 2013-07-16 | 1100 | 0.8 | CP | 4.20 |
| $\sigma_{F=om}$ | | | | | | | |
| $s_6$ | #111 | Hay | 2013-06-20 | 1030 | 0.9 | OM | 8.85 |
| $s_7$ | #339 | Hay | Autumn 2013 | 1200 | 0.9 | OM | 9.10 |
| $s_8$ | #419 | Hay | 2013 | 1020 | 0.7 | OM | 9.15 |
| $s_9$ | #456 | Pea | 2011-07-16 | 1100 | 0.8 | OM | 7.12 |

$$\mathbf{z}_1 = \rho_{G,E,T,A_{cp},R_{cp},F_{cp},CP}\left(\mathbf{r} \overset{\mathrm{NN(T,\,Avg,Avg,Min,Avg)}}{\underset{\mathrm{EQ(E)}}{\bowtie_{\mathrm{G}}}} \sigma_{F=cp}(\mathbf{s})\right)$$

| | | $E$ | $G$ | $T$ | $A_{cp}$ | $R_{cp}$ | $F_{cp}$ | $CP$ | |
|---|---|---|---|---|---|---|---|---|---|
| $\mathbf{r}_{hay}$ | $r_0$ | #111 | Hay | 2013-06-20 | 1020 | 0.9 | CP | 1.30 | $s_1, s_3$ |
| | $r_1$ | #222 | Hay | Summer 2013 | 1040 | 0.8 | CP | 1.35 | $s_3, s_4$ |
| | $r_2$ | #333 | Hay | 2013 | 1030 | 0.8 | CP | 1.30 | $s_1, s_3, s_4$ |
| $\mathbf{r}_{pea}$ | $r_3$ | #444 | Pea | 2011-07-21 | 1000 | 0.8 | CP | 4.20 | $s_5$ |

Since possible multiple join matches are aggregated, each query point $r$ can result at most into one result tuple. For example, for query point $r_1$ no equijoin match on $E$ exists but two nearest neighbors on $T$ exist in $\sigma_{F=cp}(\mathbf{s})$: tuples $s_3$ and $s_4$ are both at 0 seasons distance from $r_1$. The operator joins therefore $r_1$ with the average value among $s_3.M$ and $s_4.M$

$$\mathbf{Q} = \Pi_{E,G,T,CP,OM}\left(\rho_{\mathbf{Z_1},A_{om},R_{om},F_{om},OM}\left(\mathbf{z}_1 \overset{\mathrm{NN(T,\,Avg,Avg,Min,Avg)}}{\underset{\mathrm{EQ(E)}}{\bowtie_{\mathrm{G}}}} \sigma_{F=om}(\mathbf{s})\right)\right)$$

| | | $E$ | $G$ | $T$ | $CP$ | $OM$ | |
|---|---|---|---|---|---|---|---|
| $\mathbf{r}_{hay}$ | $r_0$ | #111 | Hay | 2011-06-20 | 1.40 | 8.85 | $s_6$ |
| | $r_1$ | #222 | Hay | Summer 2013 | 1.08 | 8.85 | $s_6, s_7$ |
| | $r_2$ | #333 | Hay | 2013 | 0.94 | 9.03 | $s_6, s_7, s_8$ |
| $\mathbf{r}_{pea}$ | $r_3$ | #444 | Pea | 2011-07-21 | 4.20 | 7.12 | $s_9$ |

Table $\mathbf{Q}$ is obtained applying the RNNJoin operator between the query points $\mathbf{r}$ and $\sigma_{F=cp}(\mathbf{s})$ and $\sigma_{F=om}(\mathbf{s})$.

The Algorithm implementing the RNNJoin will be programmed in C++ extending the engine of the DBMS of PostgreSQL and, referring to the example above, should be called at query level as follows:

```
SELECT E, G, T, CP, M AS OM
FROM (SELECT E, G, T, M AS CP
      FROM  r RNNJ s_cp EQUAL ON E NN BY G USING T [AGGR M] ) r
       RNNJ s_om EQUAL ON E NN BY G USING T [AGGR M]
```

where the optionality of the AGG clause, establish if multiple join matches have to be aggregated or not.

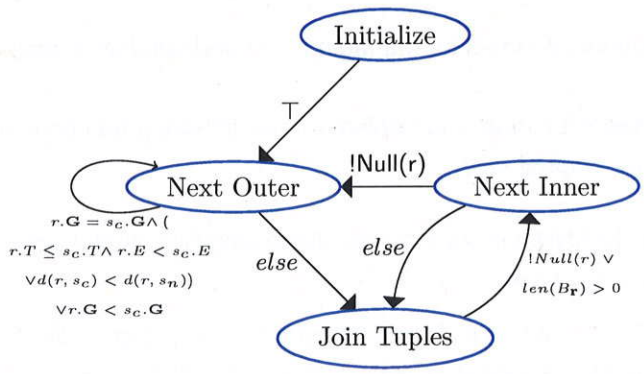
Implementation:

The RNNJoin has been integrated in the kernel of PostgreSQL as a sort merge procedure.

Two versions of the operator have been implemented so far:

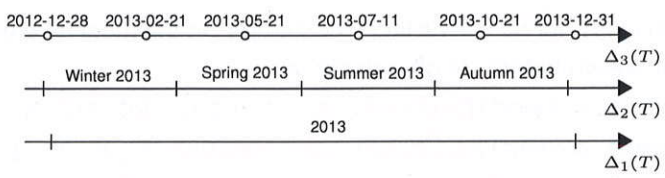
- one version aggregating, for a given query point, multitple join matches (nodeRNNJ.c)
- one version returning, for a given query point, all join matches (nodeRNNJwoBUFF.c)

The approach is modeled by a finite state machine composed by 4 main states: it is available in the above files and it is described in [1],[2].




Time Management

A graphical representation of timestamps with different granularities is shown below.




As shown in the picture above, the main problem when dealing with mutigranular timestamps is that sorting cannot be directly used (days have date format, season strings, year integers,

etc.) We need a unique format that allows to express timestamps with different granularities. We propose that each timestamp shoud be converted into an interval [a,b] using the encoding shown below:

1. 2013-07-11 = [20130711-1 , 20130711-1]
2. Spring 2013 = [20130320-2 , 20130620-2]
3. 2013 = [20130101-3 , 20131231-3]

The interval is composed by two element of the finest granularity; the interval says from when to when a timestamp spans (for example, Spring 2013 starts on 20130320 and ends on 20130620). The last number maps the interval to the granularity of the original timestamp (1 for day-granularity, 2 for season-granularity, 3 for year-granularity) . The strength of such a format is that it can be expressed on a DB level using two integers: SORTING ON ONE OF THEM CAN BE USED!

Tasks:

**April**: Reflect if the defintion of *med* is the most appropriate for a RNNJoin.
Example (1): $med$(31-12-2013, 'Winter 2013') = 79 considering the most far winter day
Example (2): $med$(31-12-2013, 'Winter 2013') = 0 considering the closest winter day
Example (3): $med$(31-12-2013, 'Winter 2013') = 34 considering the middle winter day, i.e. 03-02-2014
Example (4): $med(31-12-2013,'Winter2013',p) = d$ where a value $0 \leq p \leq 1$ specifies which of the above (or an intermediate) case should be considered and $0 \leq d \leq 79$

**April**: Adapt the pseudocode in [2] for managing $med$ and build an example for testing it.

**May**: Formally defined your *med* and implement the following functions in nodeRNNJ.c (the examples refers to the original $med$):

1. char *granularity(RNNJoinState *node, TupleTableSlot *tuple) returning the granularity of the timestamp of *tuple*
2. char *reduceT(int $t_s$, int $t_r$) reducing timestamp $t_s$ to the granulity of timestamp $t_r$.
   high-level example: reduceT(20130521, Summer 2013)= Spring 2013
   low-level example: reduceT(20130521-1, 20130521-1, 20130621-2)= 20130521-2

   high-level example (2): reduceT(Winter 2014, 2013)= 2014
   low-level example (2): reduceT(20131221-2, 20140319-2, 20130101-3)= 20140319-3
3. char *convertT(int $t_s$, int $t_r$) converting timestamp $t_s$ to the most far timestamp (the worst case we have shown before) of granularity of $to$
   high-level example: convertT(Spring 2013, 20130721) = 2013.03.20
   low-level example: convertT(20130320-2, 20130620-2, 20130721-1) = 20130320-1

   high-level example (2): converT(2014, Summer 2013)= Winter 2015
   low-level example (2): convertT(20140101-3, 20141231-3, 20130621-1)= 20141231-2
4. med(int t1, int t2) calculate the Multigranular Eucledian Distance between timestamps

$t1$ and $t2$.
$med$(Summer 2013, 2013.05.21) = 1 (Season); *reduce* will help you
$med$(2013.05.21, Summer 2013) = 123 days ; *convert* will help you

**Jun-Jul**: Extend the provided implementations of the RNNJoin operator for managing different granularities.

**Jul**: In your report define the problem and your solution precisely and design a representative running example to illustrate your approach. Describe your algorithm using pseudo code.

Present progress and plans once every week to your supervisor.

Literature:

[1] Derived Facts in Vertical Data Warehouses. F. Cafagna, M. Boehlen

[2] The aggregate and buffer effect in the RNNJoin operator. Urs Voegeli

Supervisor: Francesco Cafagna

Starting date: 6th May 2014

Ending date: 31st Jul 2014

Department of Informatics, University of Zurich

Prof. Dr. Michael Böhlen