B. Mitschang et al. (Hrsg.): BTW 2017 – Workshopband,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 181

Personalized Stream Analysis with PreferenceSQL

Lena Rudenko[1] and Markus Endres[2]

Abstract: In this paper we present our demo application which allows preference-based search for interesting information in a data stream. In contrast to existing stream analysis services, the application uses the attributes of the stream records in combination with soft conditions to achieve the best possible result for a query.

Keywords: Stream analysis, preferences, personalization

1 Introduction

Stream query processing is becoming increasingly important as more time-oriented data is produced and analyzed nowadays. Existing approaches for stream analysis have to consider the special characteristics of data streams which do not take the form of persistent database relations, but rather arrive in continuous, rapid and time-varying data objects. Hence, analyzing streams is a difficult and complex task which is in the focus of many researchers all over the world, cp. for example [ScZ05].

Due to the increasing amount of data – often produced by humans in social networks via mobile devices – such an endless flow of stream objects contains important and interesting records as well as spam and information trash. In order to distinguish between important and unimportant information one has to observe the stream objects which have attributes with additional information. These attributes can be used to analyze a stream with the goal to find the most relevant and personalized records.

In this demo paper we present an application which exploits *user preferences* to evaluate queries on various data streams. These user preferences are like soft constraints, requiring a match-making process: *If my favorite choice is available in the dataset, I will take it. Otherwise, instead of getting nothing, I am open to alternatives, but show me only the best ones available*.

We want to show that our preference-based approach is an alternative to the common used attribute-based stream analysis which follows a conditional filtering based on hard constraints. Our application allows to build a query in an intuitive and flexible way to search for the best matches in a stream. This prevents information flooding and unimportant data as well as the empty result effect.

[1] University of Augsburg, Institute for Computer Science, Universitätsstr. 6a, 86159 Augsburg, Germany, lena.rudenko@informatik.uni-augsburg.de
[2] University of Augsburg, Institute for Computer Science, Universitätsstr. 6a, 86159 Augsburg, Germany, markus.endres@informatik.uni-augsburg.de

2 Preference SQL

In our demo application we use Preference SQL which is a declarative extension of SQL by preferences, behaving like soft constraints under the Best-Matches-Only query model, cp. [KEW11]. Syntactically, Preference SQL extends the `SELECT` statement of SQL by an optional `PREFERRING` clause, cp. Figure 1a. The keywords `SELECT`, `FROM` and `WHERE` are treated as the standard SQL keywords. The `PREFERRING` clause specifies a preference which is evaluated after the `WHERE` condition.

```
SELECT <projection>
FROM   <table_reference>
WHERE  <hard_conditions>
  PREFERRING <soft_conditions>
```

(a) PreferenceSQL syntax.

```
SELECT STREAM *
FROM TwitterStream
  PREFERRING tweet_language
  IN ('de') ELSE ('en')
  PARETO followers_count HIGHEST;
```

(b) PreferenceSQL example.

Fig. 1: PreferenceSQL query block.

[KEW11] proposes several preference constructors to specify user preferences on *numerical*, *categorical*, *temporal*, and *spatial* domains. In Figure 1b for example, the first part after the `PREFERRING` keyword is a *POS/POS* preference, which means, that a desired value of some attribute (`tweet_ language`) should be amongst a finite first set (`IN ('de')`). Otherwise it should be from a second disjoint set of attributes (`ELSE ('en')`). If this is also not feasible, better than getting nothing any other value is acceptable. The second part of the user preference is a *numerical* `HIGHEST` preference. It expresses the wish to consider authors having the *most followers* in the social network Twitter if `followers_count` is an attribute of a stream object. These preferences can be combined into more *complex preferences*. For example, the *Pareto preference* (combined by the `PARETO` keyword in the `PREFERRING` clause, cp. Figure 1b) treats two or more preferences as equally important, whereas in a Prioritization (`PRIOR TO`) one preference is more important than another one. For more details we refer to [KEW11].

3 Showcase Application

In our demo application we show that preference-based stream analysis has several advantages over current methods. For example, the evaluation conditions are more flexible, because it is possible to determine the relative importance of the user wishes. Moreover, preference-based approaches always provide the best result w.r.t. the user preferences.

Figure 2 shows the main view of our application, which allows users to construct queries and to evaluate them on data streams. Thereby one can build preference-based queries as well as "standard" queries involving hard constraints. This allows us to present the advantage of personalized query evaluation on data streams in contrast to a conditional filtering as in the `WHERE` clause of stream-based languages like CQL (Continuous Query Language) or StreamSQL.

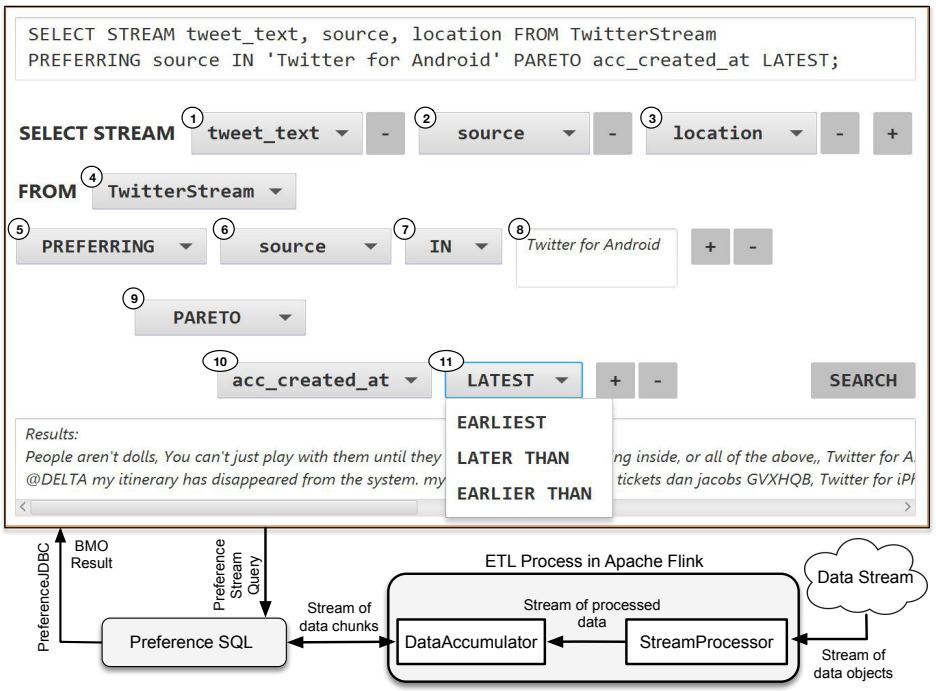

Fig. 2: Architecture of our application.

A user can build queries using *hard* or *soft constraints* (preferences). For example, in Figure 2 one wants to analyze data provided by Twitter. First, a desired stream source has to be selected from a drop-down menu list in ④. In our application a new stream connector can easily be implemented using a pluggable interface. The next step is to choose the projection attributes the user wants to see in the result. One can do it using drop-down menus where the list items are the attributes available in the stream objects (①, ② and ③ in Figure 2). The kind of attributes is different for each data source and is dynamically generated by the corresponding pre-implemented stream interface. In ⑤ the query selection mode can be specified, i.e., preference-based (`PREFERRING`) or based on hard constraints (a SQL `WHERE` clause). In ⑥ it is possible to select the attribute on which a preference or the hard condition ⑦ should be evaluated. The preference choice is determined by the data type of the selected attributes. For some preferences it is necessary to specify attribute values, such as in the POS/POS preference in Section 2. For this we provide a text field as depicted in ⑧. Such simple preferences can be combined to `PARETO` or `PRIORITIZATION` in ⑨ with additional preference conditions added by "+". In our example the preference in ⑥, ⑦ and ⑧ is equally important to a `LATEST` preference, cp. ⑩ and ⑪, which expresses that we prefer tweets posted in accounts created most recently. For hard conditions (`WHERE` in ⑤) the mode in ⑨ refers to `AND` or `OR`. Finally, the constructed query can be seen at the top of our application. Afterwards, the generated query is sent to our *preference-based stream processing framework* for evaluation.

The current University prototype of the Preference SQL system adopts a (Java 1.7-based) middleware approach as depicted at the bottom of Figure 2. For a seamless application integration we have extended standard JDBC technology towards a Preference SQL / JDBC package. This enables the client application to submit Preference SQL queries through familiar SQL clients. The Preference SQL middleware parses the query and performs preference query optimization. The final result will be computed in the execution engine by preference evaluation algorithms. Preference SQL works on well structured finite data but streams are endless and have diverse formats. Therefore, we have to preprocess the stream objects to get an attribute-based format like *attributeName = attributeValue* and to split the stream into finite chunks. This ETL process happens in Apache Flink[3] (see Figure 2). The transformed data can now be evaluated with Preference SQL, and the result is depicted at the bottom of the application. For more technical details we refer to [Ru16].

To illustrate the advantages of our approach we consider the following example: *the user wants to read tweets from Washington ("Washington (US)") about Donald Trump ("Trump") or US presidential election ("ElectionDay")*, cp. Figure 3.

```
SELECT STREAM *
FROM TwitterStream
 WHERE hashtags
 IN ('Trump', 'ElectionDay')
 AND place IN ('Washington(US)');
```

(a) SQL query.

```
SELECT STREAM *
FROM TwitterStream
 PREFERRING hashtags
 IN ('Trump', 'ElectionDay')
 PARETO place IN ('Washington(US)');
```

(b) PreferenceSQL query.

Fig. 3: Example query.

The evaluation of the query can provide different results depending on the evaluation mode. To satisfy the `WHERE` condition in the query both constraints must be fulfilled. Otherwise the *hard condition* mode provides an empty result set. In contrast, the *preference*-based query looks for best possible matches w.r.t. the query conditions. Therefore, results are also achieved when not all provided tweets correspond with the user's wish, but contain relevant and interesting information. Our demo application shows the advantage of preference-based stream analysis and that one gets personalized and interesting information instead of spam and information trash.

References

[KEW11] Kießling, W.; Endres, M.; Wenzel, F.: The Preference SQL System - An Overview. Bulletin of the Technical Commitee on Data Engineering, IEEE CS, 34(2):11–18, 2011.

[Ru16] Rudenko, L.; Endres, M.; Roocks, P.; Kießling, W.: A Preference-based Stream Analyzer. In: Workshop on Large-scale Learning from Data Streams in Evolving Environments, STREAMVOLV. 2016.

[ScZ05] Stonebraker, M.; Çetintemel, U.; Zdonik, S.: The 8 Requirements of Real-time Stream Processing. SIGMOD Rec., 34(4):42–47, December 2005.

[3] `https://flink.apache.org/`