

A methodology for preference-based personalization of contextual data*

Antonio Miele
Politecnico di Milano
Italy
miele@elet.polimi.it

Elisa Quintarelli
Politecnico di Milano
Italy
quintare@elet.polimi.it

Letizia Tanca
Politecnico di Milano
Italy
tanca@elet.polimi.it

ABSTRACT

The widespread use of mobile appliances, with limitations in terms of storage, power, and connectivity capability, requires to minimize the amount of data to be loaded on user's devices, in order to quickly select only the information that is really relevant for the users in their current contexts: in such a scenario, specific methodologies and techniques focused on data reduction must be applied. We propose an extension to the data tailoring approach of Context-ADDICT, whose aim is to dynamically hook and integrate heterogeneous data to be stored on small, possibly mobile devices. The main goal of our extension is to *personalize* the context-dependent data obtained by means of the Context-ADDICT methodology, by allowing the user to express preferences that specify which data s/he is more interested in (and which not) in each specific context. This step allows us to impose a partial order among the data, and to load only the top (most preferred) portion of the data chunks. A running example is used to better illustrate the approach.

1. INTRODUCTION

Today's portable devices, with limited resources such as computational power, battery life and memory, require applications able to manage the most interesting data, keeping on board only the small portion that – in that moment – the user prefers. Information access needs thus to be appropriately personalized, in order for the user not to be taken aback by the huge amount of available data.

Due to this quest for data personalization, the criteria for performing either off-line or dynamic *data tailoring* [4] play a relevant role, as they play a central role in determining which parts of a database (in term of tuples and attributes) should be kept and which should be discarded.

When addressing the problem of mobile data personaliza-

*This research is partially supported by the Italian MIUR project ARTDECO

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the ACM. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permissions from the publisher, ACM. EDBT 2009, March 24–26, 2009, Saint Petersburg, Russia. Copyright 2009 ACM 978-1-60558-422-5/09/0003 ...\$5.00

tion, one of the most adopted criteria for data tailoring is based on the notion of *context*. The idea is to exploit the knowledge about the situation the user is placed in, the adopted channel, and/or the environment, to reduce the amount of information stored on mobile devices.

One of the systems tackling this problem, is Context-ADDICT (Context-Aware Data Design, Integration, Customization and Tailoring) [3], providing a framework for selecting and integrating the relevant information to be delivered on user's devices on the basis of his/her current context. Even if the proposed methodology [4, 3] is simple and effective in filtering data portions to reduce information noise, it presents two main limitations: 1) in the methodology, each possible context is associated with the portions of the database relevant for that context. This association is performed at design-time in a coarse-grain, crisp way, basically defining sets of views in terms of selection, projection, and semi-join operations on a global relational database. 2) No memory occupation model is considered, neither a quantification of the amount of data to be stored on the user's device. 3) The user provides partial information on his/her current context, by choosing a role and one (or more) interests defined at design-time for the target application. Thus, the approach is addressed to “classes” of users sharing the same contexts, but it is not customized for to accommodate data ranking, done on the basis of single users' tastes.

The highlighted limitations indicate the need for more powerful, fine grain personalization mechanisms for the scenario of mobile devices. A simple, yet effective approach proposed in the literature for performing a fine-grain personalization of query answers, customized for each user, exploits *user preferences*.

There are many approaches in the literature exploiting preferences, presented in Section 2. However, at the best of our knowledge, all the proposed methodologies are targeted for query-answer personalization, and none has addressed the problem in a scenario where the contextual views, specifying *sets of relations*, are defined at design time, and the preferences must be specified on them. Even though query personalization is more demanding since it must be performed dynamically for any user query, personalizing contextual views requires to consider more than one relation and, therefore, the related integrity constraints (e.g. foreign key constraints) among the relations forming the contextual view, constraints that should be preserved in the portion of

data stored on the user device.

Goal and contributions: extension of Context-ADDICT system by adopting an additional perspective for data tailoring based on *contextual preferences*, and specifically: 1) a preference model, performing a more advanced personalization w.r.t. the literature, to express interests not only on tuples but also on attributes; 2) a methodology that, starting from context-dependent views (defining sets of relations), performs a fine-grained preference-based ranking and a successive filtering on both attributes and tuples; 3) the resulting, preference-based contextual views, fit into the user device memory, due also to the application of the *top-K* operator, and satisfy foreign key constraints.

The paper is organized as follows. In the next section the related work on preferences are discussed by analyzing benefits and limitations. Section 3 present the running example used throughout our work. Section 4 briefly describes the context model used in the Context-ADDICT framework. Section 5 presents the adopted preference model, while Section 6 describes the methodology and its integration in the Context-ADDICT framework. Finally, conclusions and future work are drawn in the last section.

2. RELATED WORK

Literature on preferences applied in the database field is extensive; however, the research can be grouped into two main approaches, mainly aimed at query answer personalization: a quantitative and a qualitative one [8]. With the *quantitative approach* preferences are expressed by using a scoring function that associates a numeric score with every tuple of the query answer, implying a total order among the tuples of a result set; e.g., a tuple t_1 is preferred to another one t_2 if the score of t_1 is higher than the score of t_2 . A general framework is presented in [2] where numerical scores between 0 and 1 are assigned to tuples on the basis of the matching of specified attribute values. Another framework [14]; allows to express interests in terms of scores on atomic query elements (such as simple selections and join conditions), as opposed to specific attribute values. Since the quantitative approach imposes a total order among tuples, the query answer personalization is easily performed by ordering tuples according to scores and by applying the *top-K* operator [6]. However, the personalization is limited, due to the fact that not every preference relation can be intuitively expressed by scoring functions, and it does not allow to impose orders weaker than the total one.

As an alternative, the *qualitative approach* consists of preferences specified directly by using binary preference relations among tuples. This approach is strictly more general than the quantitative one, and offers a higher expressiveness, since scoring functions can be defined as an explicit preference relation, not all preference relations can be expressed in terms of scoring functions, especially when the order relation is not a total order. Several frameworks have been proposed in literature, by using first order logic formulas [7] or algebraic formalisms using basic preference operators and composition rules [13]; they consider not only the total order relations, but also other types of relations such as the strict partial order or the weak one. As a consequence, the personalization based on ordering and *top-K* operators is not supported;

thus, novel relational algebra operators have been proposed for embedding the preference application into relational algebra, either through relational operators or by means of special preference constructors which select from their input the set of the most preferred tuples (e.g., Winnow [7], Best [8], Preference BMO [13], and Skyline [5]).

The adoption of preferences for the personalization of data in context-aware frameworks has been also investigated in the literature; in such a scenario, preferences are not general, but depend on the user's current context. In [16], a general framework supporting query answer personalization by means of contextual preferences is presented. Contextual preferences are modeled by associating with each preference rule, modeled as in [2], a description of the context where the rule holds; a hierarchical model composed of a set of multidimensional attributes is adopted for representing the notion of context. The framework provides also strategies for automatically identifying from the user preference repository, called profile, preferences relevant for his/her current context and for ranking the query result.

Contextual preferences, called situated preferences, are also discussed in [12], where the ER model is extended for modeling the context, called situation, and the preferences are expressed with the qualitative approach proposed in [13]. Situations are uniquely linked through an N:M relationship with preferences, stored in an XML repository, implying a more rigid structure with respect to the hierarchy proposed in [16]. Finally, a strategy for automatic extraction of preferences from the user history is retrieved from [11].

In [17], description logic is used for defining a knowledge-based, context-aware query preference model. The context is modeled by using values of the domains of the relational attributes, and preference rules are expressed with the quantitative approach. Scores are computed based on a probabilistic model for information retrieval, on the basis of the user history [18]. In the implementation, a mapping is defined between description logic concepts and relational ones and reasoning is used for establishing the preferences holding in the current context. Finally, we mention the framework proposed in [1] that adopts a qualitative approach, without a description of the context model since the context is intended as values taken by each considered tuple.

More recently, two proposals have focused also on the personalization of the schema of query results. In [9] the authors propose an algorithm for the automatic selection of the most "useful" attributes for a query result. This approach could be used as a default case in our methodology, when the user wants to specify his/her preferences only on tuples, by allowing the system to personalize attributes in an automatic way. In [10] the authors propose efficient algorithms to evaluate positive preferences over discrete attribute domains of a single relational table. The proposal lack of generality because it does not consider the possibility of specifying join conditions on multiple tables and the presence of constraints between relations.

To sum up, the approaches we have described focus on the personalization of a single query result set; our work aims at personalizing data views composed by several relations, de-

fined by views over a global database, where foreign key constraints are specified. Therefore, this work partially draws on the presented approaches and it focuses on a wider scenario; in particular, it extends the proposal [16], which is in our opinion the most similar to the Context-ADDICT philosophy.

3. RUNNING EXAMPLE

The example considered in this work is the scenario of a group of independent restaurants joining forces to promote themselves by offering their services through the “Pick-up Your Lunch” (PYL) corporation. These restaurants offer on-line ordering for either pick-up from several pick-up sites or delivery by the joined taxi company, allowing clients to put together their favorite meal also by taking dishes from different restaurants. A web site is used to promote the business, where prospective clients can browse and search restaurants and menus; on the other hand, registered users can also download on their mobile smartphone a small application to perform orders.

All applications composing the information system of the PYL corporation rely on a central database storing all kinds of managed information. The subset of the relational schema of the PYL database considered in this work is presented in Figure 1.

```

CUISINES(cuisine_id, description)
DISHES(dish_id, description, isVegetarian, isSpicy,
       isMildSpicy, wasFrozen, category_id)
RESERVATIONS(reservation_id, customer_id, restaurant_id,
             date, time)
RESTAURANT_CUISINE(restaurant_id, cuisine_id)
RESTAURANTS(restaurant_id, name, address, zipcode,
            city, state, zone_id, rnumber, phone, fax, email,
            website, openinghourslunch, openinghoursdinner,
            closingday, capacity, parking, minimumorder, rating)
RESTAURANT_SERVICE(restaurant_id, service_id)
SERVICES(service_id, name, description)

```

Figure 1: Database schema of the running example

4. THE CDT CONTEXT MODEL

The context model defined in the Context-ADDICT framework is called *Context Dimension Tree* (CDT) [3] and models the notion of context as a tree-shaped structure; the CDT represents as children of the root node the *context dimensions*, each capturing a different perspective of the context: Figure 2 that shows the CDT for the PYL running example. A dimension value can be further analyzed with respect to different viewpoints (called sub-dimensions although referred in general as dimensions), generating a subtree in its turn. In a CDT, black nodes represent dimensions (e.g., `interest_topic`) and sub-dimensions (e.g., `cuisine`, `services`, etc.); white nodes represent the values the dimensions can assume (e.g., for the `interest_topic` dimension, `orders`, `clients`, and `food`). When the number of possible values of a dimension is large (e.g., when they are constituted by a range of numerical values) *attribute nodes* (represented by two concentric circles) are used, and their instances are the admissible values for that dimension (e.g., `cost`). Similarly, attributes are also used to select specific instances in the set of values represented by a white node. In this case, an attribute node related to a white node expresses a *restriction*

parameter which can be used to single-out data pertaining to the required element [3]. The parameter can be a constant value (e.g., “Chinese” for the `$ethid` attribute node), a variable name whose value is acquired from the application (e.g., `$data_range`) or the result of a function (e.g., `getMile()` for the `$mid` attribute node). In all cases the leaves of the CDT can only be either white nodes or attribute nodes.

Note that, since the context representation is strictly related to the application scenario, it cannot be a-priori defined and only the dimensions which are meaningful for the target application are included in the CDT.

A context instance, called *context configuration*, is described by means of *context elements*. A context element may have two different specifications: *dim_name* : *value* or *dim_name* : *value(param_value)*, where *dim_name* is the name of a dimension, and *value* is a value (possibly restricted by a parameter) for that dimension (e.g., `interest_topic : food`). The context configuration is represented as a conjunction of context elements; indeed, it can be written as:

```

(role : client(“Smith”) ^ location : zone(“CentralSt.”) ^
class : lunch ^ cuisine : vegetarian)

```

that represents a client whose name is Smith, who is at the Central Station and is interested in a vegetarian lunch.

Considering the hierarchical organization of the CDT, it is possible to define a descendant relationship on context elements, stating that a context element ce_i is a descendant of another context element ce_j , if ce_j is an instantiation of a dimension in the subtree rooted in ce_i ; in the same way, the ascendant relationship can be defined. Moreover, to be coherent with the hierarchy, if a context element ce_i has as ascendant context element ce_j with an attribute, the context element ce_i inherits the attribute of ce_j (and the descendant and ascendant relationships are extended to nodes with parameters); for instance, the context element (`type : delivery`) inherits the `$data_range` from the ancestor orders and becomes

```

(type : delivery(“20/07/2008” – “23/07/2008”).)

```

The set of all the context elements that are descendants of a context element ce_i is called *desc*(ce_i).

At design time, once the CDT has been defined, the list of its context configurations is combinatorially generated. However, given an application scenario and the corresponding CDT, not necessarily all the possible combinations of context elements make sense. The model allows the expression of *constraints* among the values of a CDT to avoid the generation of meaningless ones. In our example, a constraint imposes to exclude contexts including both values `guest` and `orders`, since the guests of the Web site do not access the list of current orders. Here we do not delve into the use of constraints, which are thoroughly dealt with in [3]. Once the meaningful context configurations are determined, the designer associates each of them with a view corresponding to the relevant portion of the information domain schema. This process is done by directly writing a query in the language supported by the underlying database or by using a graphical interface. In [3] the view associated with each context is formalized as a set of relational algebra expressions.

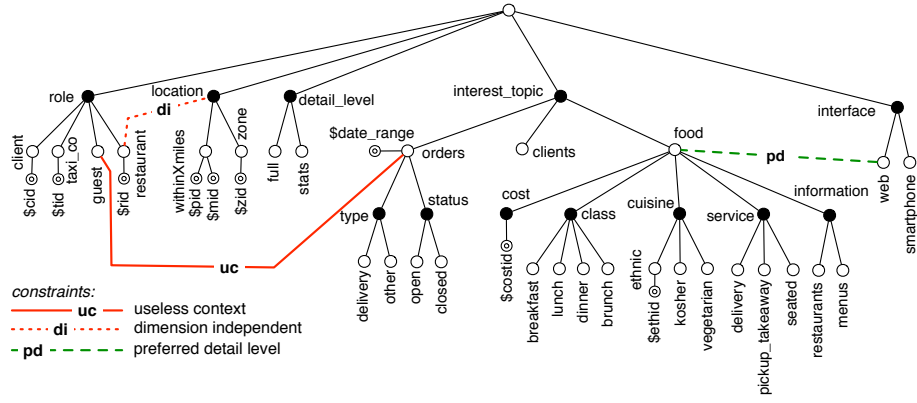


Figure 2: The CDT of our application scenario

5. THE PREFERENCE MODEL

The preference model defined in this work is more general with respect to those proposed in literature (see Section 2), in that, as shown in Section 2, previous work aims at personalizing the answer of a single specific query posed by the user, to produce a reduced, but ordered, result set; therefore, a single set of tuples is the only item on which preferences are applied. Instead, this work aims at selecting a contextual view of a global database to be loaded on the user device, by filtering *both rows and columns of several relations*, possibly related by foreign key constraints.

Our purpose is to adopt preferences as a mechanism for performing a further reduction (i.e. by filtering the preferred tuples and attributes with the *top-K* operator) of the relations composing the contextual view; thus, preferences are not applied to the single table which is the query result set, but to each set of relations composing the view associated to each context. It seems appealing to express preferences not only on table rows, but also on columns. It is worth noting that, even if not completely exploited in past research on preferences, the need for a more powerful personalization mechanism acting on both tuples and attributes is highlighted by several of today’s common data-oriented applications; some examples are e-mail clients or multimedia file manager tools, that allow user-defined data filtering and customization of the attributes to be displayed.

Though the methodology proposed in this work can be easily adapted to qualitative preferences, here we adopt quantitative preferences, our choice is motivated by today’s trend, shown by common data-oriented applications and web sites, that exploits very intuitive ranking mechanisms based on scores for expressing interest. More precisely, a preference is expressed by assigning a degree of interest to tuples or attributes, by means of scores belonging to a predefined numerical domain; for simplicity, in this work the range of real values between $[0, 1]$ is adopted as score domain. Value 1 represents extreme interest, while value 0 indicates absolutely no interest; in the middle, value 0.5 states indifference. Nevertheless, any other integer or real range can be adopted as score domain; in fact, the only prerequisite of the scoring domain is to be a totally ordered set to compare

different score values.

Now we define two kinds of preferences acting respectively on tuples and attributes of a relation. Since they mimic the selection and the projection operators, they are called σ -preferences and π -preferences.

σ -preference is a generalization of the preference model proposed in [16]. It allows the user to express a quantitative score on tuples by specifying a *selection rule*, i.e. a condition selecting the tuples of a specified relation which are interested by the preference, and a numerical value expressing the score used for ranking such tuples. The formal definition of the σ -preference is the following:

DEFINITION 5.1 (σ -PREFERENCE). A σ -preference $P_{\sigma_i}(R)$ on the relation $R(X)$ is a pair $\langle SQ_{\sigma_i}, S_i \rangle$, where

- the selection rule SQ_{σ_i} is a query expressed as

$$\sigma_{cond_i} r [\times \sigma_{cond_{i1}} t_1 \dots \times \sigma_{cond_{in}} t_n]$$

i.e., by applying the selection operator on the relation $r(X)$, called origin table, which is optionally semi-joined with subsets of other relations $t_1(Y_1), \dots, t_n(Y_n)$ only on foreign key attributes, and

- the score S_i is a real number in $[0, 1]$.

Each selection condition is a propositional formula obtained as conjunction (\wedge) of, possibly negated (\neg), atomic conditions of the form $A\theta B$ or $A\theta c$, where:

- A and B are attributes of R ;
- θ is a comparison operator ($=, \neq, >, <, \geq, \leq$) applicable to the domains of A and B ;
- c is a constant belonging to the domain of A .

It is worth noting that the rule is defined as a simple selection, since it just aims at identifying the tuples of the relation where the score has to be applied; projection and other elaborations are not meaningful because they modify the result set schema or produce tuples not in the original

relation. By joining the origin relation with other relations on the foreign key attributes, it is possible to extend the domain on which the selection is performed; in such a way an advanced ranking is performed on the origin table by considering attributes of other connected relations. At the same time we do not introduce too complex expressions that may make the algorithms proposed in the next sections more complex. For the same reason a reduced grammar is proposed for the selection condition.

EXAMPLE 5.2. *Let us consider a customer, named Mr. Smith, who is browsing restaurants' dishes; he likes spicy food very much, and is not enthusiastic of vegetarian dishes. His interests are expressed as:*

$$P_{\sigma 1} = \langle \sigma_{\text{isSpicy}=1}(\text{dishes}), 1 \rangle$$

$$P_{\sigma 2} = \langle \sigma_{\text{isVegetarian}=1}(\text{dishes}), 0.3 \rangle$$

After consulting the menu, the customer decides to make a reservation in a restaurant; as suggested by past experiences, he would like to rank restaurants on the basis of the cuisine types:

$$P_{\sigma 3} = \langle \text{restaurant} \bowtie \text{restaurant_cuisine} \bowtie \sigma_{\text{cuisine.description}=\text{"Mexican"}} \text{cuisine}, 0.7 \rangle$$

$$P_{\sigma 4} = \langle \text{restaurant} \bowtie \text{restaurant_cuisine} \bowtie \sigma_{\text{cuisine.description}=\text{"Indian"}} \text{cuisine}, 0.3 \rangle$$

It is worth noting that in the first situation both selection rules specify a simple selection, while in the two latter selection rules use semi-joins with other tables.

While the σ -preference acts on tuples of the selected relation, the second type of preference, called π -preference, aims at expressing an interest score attributes. The formal definition is the following:

DEFINITION 5.3 (π -PREFERENCE). *A π -preference $P_{\pi_i}(R)$ on the relation $R(X)$ is a tuple $\langle A_{\pi_i}, S_i \rangle$ where $A_{\pi_i} \in X$ is an attribute of a relation schema $R(X)$ and S_i is a constant real number in $[0, 1]$.*

Even if the idea of assigning a constant score to a specified attribute is very simple and intuitive, at the best of our knowledge, preferences on schema attributes taking into account foreign key constraints are introduced for the first time. In order to obtain a more compact formula, even if its expressiveness is not increased, a compound π -preference can be defined by specifying it on a set of attributes, instead of indicating a single attribute in the A_{π} field of the preference.

EXAMPLE 5.4. *Let us consider our customer Smith looking for a restaurant for a phone reservation; he is not interested in the full address of the restaurant, but only in the phone number and the zipcode necessary for identifying approximately the zone where the restaurant is. Therefore, he expresses the following preferences:*

$$P_{\pi 1} = \langle \{\text{name, zipcode, phone}\}, 1 \rangle$$

$$P_{\pi 2} = \langle \{\text{address, city, state, rnumber, fax, email, website}\}, 0.2 \rangle$$

A final consideration has to be drawn for both preference types: in our opinion, often it is not meaningful to express

preferences on surrogate attributes such alphanumeric IDs used as primary keys or foreign keys, since they do not carry any semantics, but are used only for data management purposes (e.g., the `restaurant_id` attribute of table RESTAURANT). In this way, it happens that no preference is expressed on bridge tables; as it will be presented in the next section, the customization of such relations will be performed as a consequence of the personalization of the relations they refer to.

It is now necessary to relate the defined preference model with the Context-ADDICT context model. To this purpose, the two types of preferences are extended by adding the context configuration representing the situation where the preference rule holds:

DEFINITION 5.5 (CONTEXTUAL PREFERENCE). *A contextual preference CP is a tuple $\langle C, P \rangle$, where C is a context configuration and P is either a π -preference or a σ -preference.*

EXAMPLE 5.6. *Preferences expressed in Examples 5.2 and 5.4 are now contextualized. For instance, preferences $P_{\sigma 1}, \dots, P_{\sigma 4}$ can be associated with a very general context specifying only the role and the customer name, i.e., $C_1 = \langle \text{role} : \text{client}(\text{"Smith"}), \dots \rangle$, because they are related to his general tastes. Instead, preferences $P_{\pi 1}, P_{\pi 2}$ might hold when the customer, living near Central Station, is at home. Thus, the contextual preferences $\langle C_2, P_{\pi 1} \rangle$ and $\langle C_2, P_{\pi 2} \rangle$ associates these preferences with the context configuration $C_2 = \langle \text{role} : \text{client}(\text{"Smith"}) \wedge \text{location} : \text{zone}(\text{"CentralSt."}) \rangle$.*

6. PERSONALIZATION FRAMEWORK

The approach for contextual-preference-based personalization is implemented as an extension of the Context-ADDICT architecture: when a context-aware view is materialized by the system, before loading it on the user device, the personalization is performed by means of the preferences expressed by that user that are relevant to his/her current context. The methodological flow (shown in Figure 3) is composed of four main tasks: 1) active preference selection, 2) attribute ranking, 3) tuple ranking, and 4) view personalization.

The personalization is performed in two distinct steps: first on tuples and then on attributes. However, other possibilities can be introduced in our framework: for instance the selectivity of contextual views could be used to guide attribute personalization; moreover, automatic attribute personalization, similar to the approach described in [9], could be considered when the user does not specify any attribute ranking.

The Context-ADDICT mediator is provided with a repository containing, for each user, the list of his/her contextual preferences; this list is called *preference profile*. When the user's device connects to the application server and requires a synchronization of the data view according to the current context, it sends the current *context configuration*, i.e., the descriptor of the context. In step 1, the preference profile is analyzed to select those instances that are relevant for the current context (called *active preferences*). In particular, a preference is active if its context configuration is equal to, or "more general" (see below) than, the current context descriptor. This choice is motivated by the fact that a more general context is related to a wider portion of data, w.r.t.

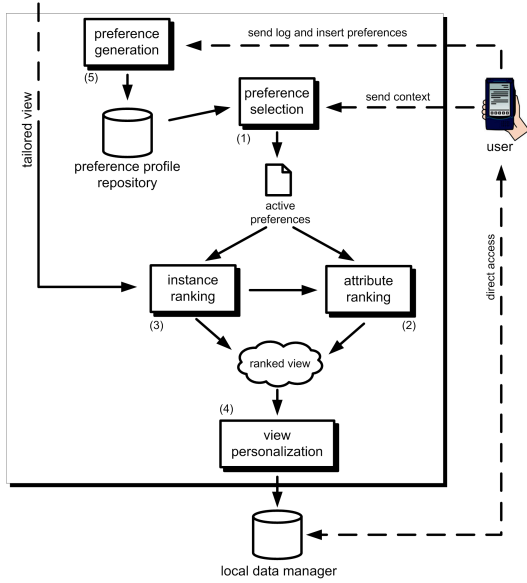


Figure 3: The view personalization framework

the view associated to the current context. Thus, the preferences specified on a wider view can be applied, with an appropriate weight, also to the more refined portion of data associated with the current context.

Then, the set of active σ -preferences and π -preferences are applied on the attributes (step 2) and tuples (step 3) of the view associated with the current context. The result of these two steps is a view with both tuples and attributes decorated with scores. Finally, in step 4, the view is definitely personalized and reduced by discarding the less interesting tuples and attributes. It is worth noting that another relevant issue is related to the mechanisms provided to the user for generating preferences (step 5). These steps are accurately described in the following subsections.

6.1 Relevant preference selection

The first step of the methodology for view personalization (step 1 of Figure 3) consists in identifying, in the user preference profile, the set of *active* contextual preferences for the current context.

In order to formalize the concept of active preference, we need to define an index of relevance for stating how much a context configuration is close to the current one. Intuitively, it is possible to state that a context configuration A is more abstract than (or dominates) another context configuration B (represented by means of the \succ operator), if for each CDT dimension, the context elements specified in A are equal or more general than the ones specified in B ; this happens when context elements of each dimension instantiated in B belong to the descendant set of the corresponding context elements of A . The formal definition is the following:

DEFINITION 6.1 (\succ DOMINANCE RELATION). *Let us consider two context configurations $C_1 = d_{11} : v_{11} \wedge \dots \wedge d_{1n} : v_{1n}$ and $C_2 = d_{21} : v_{21} \wedge \dots \wedge d_{2m} : v_{2m}$, C_1 is more abstract*

than C_2 , written as $C_1 \succ C_2$, if and only if for each conjunct $d_{1i} : v_{1i}$ in C_1 there is a conjunct $d_{2j} : v_{2j}$ in C_2 such that either $d_{2j} : v_{2j} \in \text{desc}(d_{1i} : v_{1i}) \cup \{d_{1i} : v_{1i}\}$.

It is easy to show that the \succ dominance relation defines a partial order on the context configuration domain. Finally, $C_1 \sim C_2$ states that C_1 and C_2 cannot be compared. Note that only few configurations turn out to be comparable with each other; however, it seems to us the most reasonable definition, since two configurations are not comparable w.r.t. the dominance relation, when they contain concepts of the CDT that are mutually exclusive.

EXAMPLE 6.2. *Let us consider the following context configurations:*

$$\begin{aligned} C_1 &= \langle \text{role} : \text{client}(\text{"Smith"}) \wedge \text{location} : \text{zone}(\text{"CentralSt."}) \rangle \\ C_2 &= \langle \text{role} : \text{client}(\text{"Smith"}) \wedge \text{location} : \text{zone}(\text{"CentralSt."}) \\ &\quad \wedge \text{cuisine} : \text{vegetarian} \wedge \text{information} : \text{menus} \rangle \\ C_3 &= \langle \text{role} : \text{client}(\text{"Smith"}) \wedge \text{location} : \text{zone}(\text{"CentralSt."}) \\ &\quad \wedge \text{interface} : \text{smartphone} \rangle \end{aligned}$$

From definition 6.1, $C_1 \succ C_2$, $C_1 \succ C_3$ and $C_2 \sim C_3$.

Furthermore, it is possible to define a function expressing the distance between two comparable configurations, measuring "how different" the configurations are.

DEFINITION 6.3 (CONTEXT CONFIGURATION DISTANCE).

Let us consider two context configurations $C_1 = d_{11} : v_{11} \wedge \dots \wedge d_{1n} : v_{1n}$ and $C_2 = d_{21} : v_{21} \wedge \dots \wedge d_{2m} : v_{2m}$, with either $C_1 \succ C_2$ or $C_2 \succ C_1$. For $i \in \{1, 2\}$, consider the sets

$$AD_{C_i} = \bigcup_{j \in \{1, \dots, n\}} \{d | d = d_{ij} \text{ or } d \text{ is a dimension ancestor of } d_{ij}\}$$

Then $\text{dist}(C_1, C_2) = \text{abs}(\|AD_{C_1}\| - \|AD_{C_2}\|)$ ¹.

EXAMPLE 6.4. *Let us consider the context configurations defined in Example 6.2. $\text{dist}(C_1, C_2) = 3$ and $\text{dist}(C_1, C_3) = 1$, while the distance $\text{dist}(C_2, C_3)$ is not defined.*

Having defined the dominance relation and the distance function, it is possible to define Algorithm 1, the algorithm for selecting the active contextual preferences. The function takes as input the current context C_{curr} as defined in Section 4, and the user profile \mathbf{CP}_{user} , organized as a list of contextual preferences. The output is the list of active preferences, i.e., the list of all preferences whose context configuration dominates the current context, together with their relevance index. The algorithm scans the whole user profile (Line 3): if the context configuration of a contextual preference cp dominates the current context C_{curr} , it is considered active (Line 4). Its relevance index is computed, in percentage, as:

$$\text{relevance}(cp) = \frac{\text{dist}(C_{curr}, C_{root}) - \text{dist}(cp.C, C_{curr})}{\text{dist}(C_{curr}, C_{root})},$$

where $\text{dist}(C_{curr}, C_{root})$ represents the highest possible distance of the current context w.r.t. context configurations

¹The function abs returns the absolute value of a number, whereas the function $\|A\|$, applied on a set A , returns the cardinality of the set.

Algorithm 1 Active preference selection.

```
1: function PREFERENCESELECTION( $C_{curr}$ ,  $\mathbf{CP}_{user}$ )
2:    $\mathbf{P}_{active} \leftarrow \emptyset$ 
3:   for all  $cp \in \mathbf{CP}_{user}$  do
4:     if  $cp.C \succ C_{curr}$  then
5:        $R \leftarrow \text{relevance}(cp)$ 
6:        $\mathbf{P}_{active} \leftarrow \mathbf{P}_{active} \cup \{(cp.P, R)\}$ 
7:     end if
8:   end for
9:   return  $\mathbf{P}_{active}$ 
10: end function
```

of any other active preference; indeed, the most abstract context configuration is the one corresponding to the root of the CDT, C_{root} (Line 5). In this way, preferences having the context descriptor equal to the current context have the maximum relevance index, that is 1, while preferences having the context descriptor equal to the root have the minimum relevance, that is 0. Then, the pair composed by the preference rule contained in the contextual preference and the relevance index is added to the set \mathbf{P}_{active} (Line 6). At the end of the algorithm the \mathbf{P}_{active} set contains the list of active preferences, each provided with the relevance index w.r.t. the current context; this set will be split into two subsets separately elaborated in the subsequent two phases.

EXAMPLE 6.5. *Let us consider a preference profile (for the sake of space, preference rules are omitted):*

```
 $CP_1 = \langle C_1 = \text{role} : \text{client}(\text{"Smith"}) \wedge \text{location} : \text{zone}(\text{"CentralSt."}) \wedge \text{information} : \text{restaurant},$   
 $SQ_{\sigma_1} = \dots, S_{\sigma_1} = 0.8 \rangle$   
 $CP_2 = \langle C_2 = \text{role} : \text{client}(\text{"Smith"}) \wedge \text{information} : \text{restaurant},$   
 $SQ_{\sigma_2} = \dots, S_{\sigma_2} = 0.5 \rangle$   
 $CP_3 = \langle C_3 = \text{role} : \text{client}(\text{"Smith"}) \wedge \text{location} : \text{zone}(\text{"CentralSt."}) \wedge \text{interface} = \text{smartphone},$   
 $A_{\pi_3} = \dots, S_{\pi_3} = 0.8 \rangle$ 
```

if the current context is:

```
 $C_{curr} = \langle \text{role} : \text{client}(\text{"Smith"}) \wedge \text{location} : \text{zone}(\text{"CentralSt."}) \wedge \text{information} : \text{restaurants} \rangle$ 
```

the resulting list of active preferences and their relevance index is $\langle P_{\sigma_1}, 1 \rangle$ and $\langle P_{\sigma_2}, 0.75 \rangle$.

6.2 Attribute ranking

The second step of the tailored view personalization methodology (Step 2 of Figure 3) consists in the ranking of the attributes included in the view tailored by the designer. The algorithm consists in decorating each attribute of the tailored view by using scores of the active π -preferences; if more than one rule refers to the same attribute, scores are combined by applying a specific function, such as the average value of the scores with the highest relevance, while attributes not mentioned by any preference are decorated with an indifference score, that is the value 0.5.

Two particular cases have to be considered when managing π -preferences: primary key attributes and attributes where a foreign key constraint is defined must be labeled with the maximum score assigned to attributes of the relation they belong to. Moreover, attributes that are referenced by other attributes must be labeled with the maximum value among

Algorithm 2 Attribute ranking algorithm.

```
1: function ATTRIBUTESHEMA( $\mathbf{R}_T$ ,  $\mathbf{P}_{\pi_{active}}$ )
2:   for all  $R_i(X_i) \in \mathbf{R}_T$  do
3:     for all  $A_j \in X_i$  do
4:       if  $\mathbf{P}_{\pi_{active}}[A_j.name] \neq \emptyset$  then
5:          $A_j.score \leftarrow \text{comb\_score}_{\pi}(\mathbf{P}_{\pi}[A_j.name])$ 
6:       else
7:          $A_j.score \leftarrow 0.5$ 
8:       end if
9:       if  $A_j \in R_i[\text{ref\_attribute}]$  then
10:         $A_j.score \leftarrow \max(\{A_j.score\} \cup \{A_k.score : \forall A_k \in \text{get\_related\_fk}(A_j)\})$ 
11:      end if
12:    end for
13:     $\text{max\_score} \leftarrow \max(\{A_j.score : \forall A_j \in X_i\})$ 
14:     $R_i[key].score \leftarrow \text{max\_score}$ 
15:    for all  $A_j \in R_i[\text{foreign\_key}]$  do
16:       $A_j.score \leftarrow \text{max\_score}$ 
17:    end for
18:  end for
19:  return  $\mathbf{R}_T$ 
20: end function
```

the scores of all foreign keys. In fact, primary keys and foreign keys are required to join different relations and, therefore, should have the least probability to be eliminated; in the same way, the referenced attributes must have a higher relevance than those of the foreign keys they refer to.

The pseudo-code of a possible attribute ranking function is presented in Algorithm 2. The function takes as input the list \mathbf{R}_T of the relations belonging to the tailored view, and the list \mathbf{P}_{π} of active π -preferences, selected during the previous preference selection step. \mathbf{R}_T is organized as a list of relation entries, each composed by the relation name and the scored list of attributes, i.e. pairs $(name, score)$, belonging to the relation. Each relation has a reference to its key attribute called *key*, the reference to the list of foreign keys called *foreign_keys* and the list of attributes referenced by foreign keys *referenced_attributes*. Finally the algorithm requires the list to be ordered according to the dependency graph of the foreign keys in such a way that each relation having one or more foreign keys precedes all the referenced relations; in case foreign keys generate a loop of dependencies among relations, the designer decides the least relevant foreign key, and that is not considered, in order to break the loop. It is worth noting that this simplification does not affect the effectiveness of the overall methodology; moreover, the attribute ranking algorithm can be easily modified to take loops into consideration.

The list of active π -preferences \mathbf{P}_{π} is reorganized as a multi-map in the form $(key : A_{\pi}, value : (S_{\pi}, R))$, i.e., by using the attribute name as map key, and the numerical score and the relevance as entry value. It is worth noting that the multi-map structure is chosen in order to refer in a easy way to a preference, given the attribute of interest, and to avoid key collisions that may occur in the simple map; in fact, several active preferences may refer to the same attribute. The output of the algorithm is the tailored view schema ranked with preference scores.

The algorithm scans all attributes for each relation schema of the tailored view (Lines 2–3); for each attribute \mathbf{a} , if there is a set of preferences referring to \mathbf{a} (Line 4), the scores in the set are combined and the result is assigned to the *score* label of the attribute (Line 5); otherwise an indifference score (0.5) is assigned (Line 7). More precisely, when a key is specified (in the pseudo-code identified by using square brackets), the multi-map returns a list of all (S_π, R) tuples having the specified key; the $comb_score_\pi$ function takes in input a (not empty) list of (S_π, R) tuples and returns an overall numerical score. Several $comb_score_\pi$ functions may be adopted for combining multiple scores referred to the same attribute; the most intuitive one is defined as:

$$comb_score_\pi(list_{(S_\pi, R)}) = \frac{\sum_{(s_i, r_i) \in list: \exists (s_j, r_j) r_j > r_i} s_i}{|(s_i, r_i) \in list: \exists (s_j, r_j) r_j > r_i|},$$

that is the computation of the average value of all the scores of the preferences at a minimum distance, i.e., with the highest relevance index, from the current context. The other preferences, that are more distant, are not considered.

According to the referential integrity constraints, if the current attribute is referenced by one or more foreign keys of other relations, it must have a score equal or greater than the maximum one of all the referencing foreign keys, that are returned by the $get_related_fk$ function (Lines 9–11). It is worth noting that the particular ordering on the relation list guarantees that foreign keys are scored before the referenced attribute. For the same reason, after all the attributes of a relation have been analyzed, the scores of its primary key and foreign keys are updated with the maximum score contained in the relation (Lines 14–17).

It may happen that some preferences are related to attributes not contained in the examined view; in this case, the algorithm automatically discards those preferences.

EXAMPLE 6.6. *Let us consider a view containing a projection of the RESTAURANT table, and RESTAURANT_CUISINE and CUISINE tables of the PYL database schema proposed in Figure 1, with the following list of active π -preferences:*

$P_{\pi 1} = \{\{\text{name, cuisine.description, phone, closingday}\}, 1\}, R = 1$
 $P_{\pi 2} = \{\{\text{address, city, state, phone}\}, 0.1\}, R = 0.2$
 $P_{\pi 3} = \{\{\text{fax, email, website}\}, 0.1\}, R = 0.2$

The resulting ranked schema is:

RESTAURANTS(restaurant_id:1, name:1, address:0.1, zipcode:0.5,
city:0.1, phone:1, fax:0.1, email:0.1, website:0.1,
openinghourslunch:0.5, openinghoursdinner:0.5,
closingday:1, capacity:0.5, parking:0.5)
RESTAURANT_CUISINE(restaurant_id:0.5, cuisine_id:0.5)
CUISINES(cuisine_id:1, description:1)

As said above, these values will be used to automatically decide which columns will be loaded on the user device, based on the available memory.

6.3 Tuple ranking

The third step of the methodology (step 3 of Figure 3), that is performed in parallel with the previous one, consists in ranking the tuples contained in the view. The basic idea is the same as for attribute ranking: for each tailored relation,

Algorithm 3 Tuple ranking algorithm

```

1: function INSTANCEDATA( $\mathbf{r}_{db}, \mathbf{Q}_T, \mathbf{P}_\sigma$ )
2:   view  $\leftarrow \emptyset$ 
3:   for all  $q \in \mathbf{Q}_T$  do
4:     score_map  $\leftarrow \emptyset$ 
5:     for all  $p \in \mathbf{P}_\sigma$  do
6:       if  $q.get\_from\_table() = p.get\_origin\_table()$ 
7:         then
8:            $r_{dummy\_view} \leftarrow q.selection(\mathbf{r}_{db}) \cap$ 
9:              $p.SQ_\sigma(\mathbf{r}_{db})$ 
10:          for all  $t \in r_{dummy\_view}$  do
11:            score_map[t.key] =  $p$ 
12:          end for
13:        end if
14:      end for
15:       $r_{curr\_view} \leftarrow q(\mathbf{r}_{db})$ 
16:      for all  $t \in r_{curr\_view}$  do
17:        if score_map[t.key]  $\neq \emptyset$  then
18:          t.score  $\leftarrow comb\_score_\sigma(score\_map[t.key])$ 
19:        else
20:          t.score  $\leftarrow 0.5$ 
21:        end if
22:      end for
23:    end for
24:  return view

```

it applies all the scores of the active σ -preferences; if more than one rule refers to the same tuple, scores are combined by means of a function similar to those discussed in the previous section, while tuples not referred by any preference are decorated with an indifference score.

Algorithm 3 presents the tuple-ranking function. This time, instead of taking as input the schema of the relations in the tailored view, the function takes as input the global database \mathbf{r}_{db} organized as a set of relations $\{r_1, \dots, r_n\}$, the set of queries \mathbf{Q}_T provided by the designer for tailoring the view, and the set of active σ -preferences \mathbf{P}_σ provided by the preference selection algorithm and organized as a list of elements (SQ_σ, S_σ, R) . It is assumed that all the queries contained in \mathbf{Q}_T are composed by selection and projection operations on a relation, or at most they contain semi-join operators, similarly to the definition of the σ -preference selection query; in fact, the tailoring methodology [3] aims at selecting the part of the database to be presented to the user and, therefore, it does not perform any advanced elaboration that might modify either the relation schema or the instance values. The output of the algorithm is the data view tailored by the designer with tuples decorated with the computed preference scores.

The algorithm is composed by a main loop for scanning all the queries contained in \mathbf{Q}_T (Line 3). For each query, aiming at generating a relation belonging to the view, the \mathbf{P}_σ set is visited in order to select preferences expressed on the current relation (Line 5); the test is performed by comparing the name of the origin table of the preference and the name of the relation on which the query is performed ($p.get_origin_table$ and $q.get_from_table$ are the functions for retrieving the names of the two relations; Line 6).

The subset of tuples interested by the current preference is computed by intersecting the result of SQ_σ and the one of

the selection expressed by the current tailoring query both performed on the database r_{db} (Line 7); the projections expressed in the tailoring query are not performed in order to obtain a result set with a schema equal to the origin table. For each tuple of the selected subset, the preference is stored in a (initially empty) multi-map $score_map$ organized in the form $(key : tuple_key, value : (SQ_\sigma, S_\sigma, R))$. Several preferences may refer to the same tuple, therefore there may be several scores expressed for the same tuple (Line 8).

Once all the active preferences on the current relation have been processed, they are combined and applied to the table of the view which is obtained by performing the current tailoring query on the global database (Lines 13). Thus, for each tuple of the current relation, scores are retrieved from the map by specifying the tuple key, and combined by means of the $comb_score_\sigma$ function (Lines 14–16); if no score is specified, an indifference score is assigned (Line 18). As for the attribute ranking in Section 6.2, several $comb_score_\sigma$ functions can be defined. The most intuitive one computes the average value of all active σ -preferences that are not overwritten by ($ovwr_by$) any other preference:

$$comb_score_\sigma(list(SQ_\sigma, S_\sigma, R)) = \frac{\sum_{(q_i, s_i, r_i) \in list: \exists (q_j, s_j, r_j): (q_i, s_i, r_i) ovwr_by(q_j, s_j, r_j)} S_i}{|(s_i, r_i) \in list: \exists (q_j, s_j, r_j): (q_i, s_i, r_i) ovwr_by(q_j, s_j, r_j)|}$$

It is possible to state that a σ -preference P_{σ_1} is overwritten by another preference P_{σ_2} if and only if:

- the relevance of P_{σ_1} is smaller than the relevance of P_{σ_2} ,
- the SQ_{σ_1} of P_{σ_1} and the SQ_{σ_2} of P_{σ_2} are such that:
 - for each selection s_{cond_i} expressed in P_{σ_1} there is a selection s_{cond_j} in P_{σ_2} expressed on the same relation, and
 - for each atomic condition ac_k contained in s_{cond_i} there is an atomic condition ac_m contained in s_{cond_j} expressed with the same form ($A\theta B$ or $A\theta C$) on the same attribute (or two attributes).

Nevertheless other formulas can be defined for combining scores.

Finally, the obtained ranked relations are added to a **view** set (Line 20) that, at the end of the algorithm, represents the final tailored view decorated with instance scores. Preferences that have been considered active for the current context, but refer to relations discarded by the designer during the tailoring process, are automatically discarded.

EXAMPLE 6.7. *Let us consider the views on the RESTAURANT, RESTAURANT_CUISINE and CUISINE tables of the PYL database shown in Figure 4; the following list of preferences represents some common preferences of the user, tagged with relevance 0.2, and some preferences specific for the current context, tagged with relevance 0.8 or 1:*

$$\begin{aligned} P_{\sigma_1} &= \langle \text{restaurant} \times \text{restaurant_cuisine} \times \\ &\quad \sigma_{\text{cuisine.description}=\text{"Chinese"}} \text{cuisine}, 0.8 \rangle, R = 1 \\ P_{\sigma_2} &= \langle \text{restaurant} \times \text{restaurant_cuisine} \times \\ &\quad \sigma_{\text{cuisine.description}=\text{"Pizza"}} \text{cuisine}, 0.6 \rangle, R = 0.8 \\ P_{\sigma_3} &= \langle \text{restaurant} \times \text{restaurant_cuisine} \times \\ &\quad \sigma_{\text{cuisine.description}=\text{"Steakhouse"}} \text{cuisine}, 1 \rangle, R = 1 \\ P_{\sigma_4} &= \langle \text{restaurant} \times \text{restaurant_cuisine} \times \\ &\quad \sigma_{\text{cuisine.description}=\text{"Kebab"}} \text{cuisine}, 0.2 \rangle, R = 0.2 \\ P_{\sigma_5} &= \langle \sigma_{\text{openinghourslunch}=13:00} \text{restaurant}, 0.8 \rangle, R = 0.2 \\ P_{\sigma_6} &= \langle \sigma_{\text{openinghourslunch}=15:00} \text{restaurant}, 0.2 \rangle, R = 0.2 \\ P_{\sigma_7} &= \langle \sigma_{\text{openinghourslunch} \geq 11:00 \wedge \text{openinghourslunch} \leq 12:00} \\ &\quad \text{restaurant}, 1 \rangle, R = 1 \\ P_{\sigma_8} &= \langle \sigma_{\text{openinghourslunch}=13:00} \text{restaurant}, 0.5 \rangle, R = 1 \\ P_{\sigma_9} &= \langle \sigma_{\text{openinghourslunch} > 13:00} \text{restaurant}, 0.2 \rangle, R = 1 \end{aligned}$$

The listed preferences can be divided into two groups according to the attributes on which they are expressed: one expressing preferences on the restaurant cuisine and the other one on the opening hours. Then, the scores are assigned to each restaurant as shown in Figure 5.

Restaurant	opening hour	cuisine
Pizzeria Rita	(1, 1)	(0.6, 0.2)
Cing Restaurant	(1, 1)	(0.6, 0.2), (0.8, 1)
Cantina Mariachi	(0.5, 1), (0.8, 0.2)	–
Turkish Kebab	(1, 1)	(0.6, 0.2), (0.2, 0.2)
Texas Steakhouse	(1, 1)	(1, 1)
Cong Restaurant	(0.2, 1)	(0.8, 0.2)

Figure 5: Example of assignment of scores to tuples

In the obtained list there are two overwritten scores: (0.6, 0.2) for “Cing Restaurant” and (0.6, 0.2) for “Cantina Mariachi”. Finally, the final score of each tuple is computed as the average of all the atomic scores. Figure 6 shows the RESTAURANT ranked table; all tuples of other tables are ranked with 0.5 score since no preference is expressed on them.

rest_id	name	openinghours	...	score
1	Pizzeria Rita	12 : 00	...	0.8
2	Cing Restaurant	11 : 00	...	0.9
3	Cantina Mariachi	13 : 00	...	0.5
4	Turkish Kebab	12 : 00	...	0.6
5	Texas Steakhouse	12 : 00	...	1
6	Cong Restaurant	15 : 00	...	0.5

Figure 6: Example of scored RESTAURANT table

6.4 View personalization

The last step of the methodology (step 4 of Figure 3) aims at performing the personalization of the tailored view proposed by the designer. This personalization is performed by filtering out the less relevant parts of the tailored view, which are identified on the basis of the scores computed in the previous steps. It is worth noting that the view can only be reduced and cannot be extended by including tuples and attributes discarded by the designer, because this would be in contrast with the previous tailoring steps; therefore, all the possible personalized views are contained in the original tailored view. Moreover, data filtering has to be performed without violating referential constraints specified by the database schema; therefore, relations related by foreign key constraint must contain coherent data.

When considering a tailored view composed by several relations, and scoring is adopted for tuples and for attributes,

rest_id	name	openinghourslunch	...
1	Pizzeria Rita	12 : 00	...
2	Cing Restaurant	11 : 00	...
3	Cantina Mariachi	13 : 00	...
4	Turkish Kebab	12 : 00	...
5	Texas Steakhouse	12 : 00	...
6	Cong Restaurant	15 : 00	...

(a) RESTAURANT

cuisine_id	description
1	Pizza
2	Mexican
3	Kebab
4	Chinese
5	Steakhouse

(b) CUISINE

restaurant_id	cuisine_id
1	1
2	4
2	1
3	2
4	1
4	3
5	5
5	2
6	4

(c) RESTAURANT_CUISINE

Figure 4: Example of tables of PYL database

there are several degrees of freedom for the personalization, leading to a large set of possible approaches. Due to this large set of approaches, a greedy algorithm is adopted here for performing view personalization in an easy and fast way; the aim is to perform a balanced personalization among the several relations, by considering the available memory space.

6.4.1 Physical considerations

The view personalization step requires a model of the data occupation in memory; more precisely, it is necessary to estimate 1) which is the size of a given relation and 2) which is the maximum number of tuples allowed for relation for a given space amount in memory. The memory model strongly depends on the storage format of the view; in particular, two different formats can be considered for storing the view on the user device: the textual format (such as the XML-based one), and the DBMS-based one. In case of textual format, the size of a table, and in general of the global database, can be estimated as the dimension of the text file containing the data, that is equal to the number of ASCII characters contained into the file multiplied by the cost of a single character (usually 1 byte for ASCII code). Otherwise, in case a DBMS is used for storing the view on the user device, several DBMSs provide models for estimating the occupation of a single table and of the overall database. For instance, the occupation model for Microsoft SQL Server is proposed in [15]. Given a database schema, formulas provided by both models can be inverted in order to compute the maximum number of tuples that fit into a memory area with a specified size. In case the occupation model is not specified for a particular DBMS, it is necessary to adopt an iterative greedy approach for identifying the number of tuples that can be stored in memory; this topic will be delved into at the end of this section.

In this work, two functions are used for the estimation of memory occupation independently of the storage format: $size(\#tuples, relation_schema)$, which computes the amount of memory occupied by a table containing a specified number of tuples, and $get-K(memory_dimension, relation_schema)$, which computes the maximum number of tuples to be stored in a table fitting a specific amount of memory.

Considering the adopted model, we specify the constraint on the memory dimension as the simple formula:

$$\sum_{r_i \in \text{view}} size(|r_i|, R_i) \leq dim_{memory},$$

that means that the sum of the occupation of each table of the view, computed by using the function provided by the

occupation model, must be less or equal than the available memory space.

6.4.2 The Personalization Algorithm

The pseudo-code of the view personalization function is presented in Algorithm 4. The function takes as input the scored **view** with its scored schema \mathbf{R}_T produced by the previous steps, and two numerical parameters representing the memory dimension dim_{memory} and the *threshold* specified by the user in the domain $[0; 1]$. The output is the **personalized.view** with its updated schema \mathbf{R}_T .

The algorithm is composed by two main parts, the first one performing a medium-grain tailoring of the relations' attributes and the second one a fine grain filtering on tuples. The first part is composed by a loop scanning the whole set of view relations. Each relation is tailored according to the *threshold* received as parameter: all attributes having a score smaller than the *threshold* are discarded (Lines 3–7). It is worth noting that the greater this parameter is, the larger the resulting relation schema: if *threshold* is set to 1, no attribute is removed from the schema proposed by the designer; on the contrary, if it is 0, the entire schema is dropped and the view is empty. Moreover, when considering scoring rules adopted in Section 6.2, this tailoring process produces always a view where referential integrity constraints are satisfied; in fact, it is not possible that a relation has no primary key or a foreign key is a dangling reference. Then, for each relation the average schema score is computed on the remaining attributes (Line 8) and, finally, the inner loop of a bubble-sort algorithm is performed to sort the already analyzed part of the relation list according to the average schema score from the highest to the lowest. In case of equal average schema score, relations are ordered so that relation R_i with foreign keys come after relations that R_i refers to. At the end of the computation, the result of this first part is the list of relations personalized on the attributes and ordered on average schema scores, which represent the degree of interest of each relation, and, secondary, on foreign key dependencies.

In the second part of the algorithm, the list of tables' schemas previously customized is scanned in the computed order, based on the average schema score, and each relation is personalized according to scores on tuples and satisfying relations dependencies. For each relation, the tuples are updated by projecting only the selected columns (Line 17); then, they are filtered according to dependencies with other relations already analyzed that are stored in the **personalized.view** set (Lines 18–23): the current relation is semi-

Algorithm 4 View personalization algorithm

```

1: function PERSONALIZEVIEW(view,  $\mathbf{R}_T$ ,  $dim\_memory$ ,
    $threshold$ )
2:   for all  $R_i(X_i) \in \mathbf{R}_T$  do
3:     for all  $A_j \in X_i$  do
4:       if  $A_j.score < threshold$  then
5:          $R_i(X_i) \leftarrow R_i(X_i) - \{A_j\}$ 
6:       end if
7:     end for
8:      $R_i(X_i).score \leftarrow average\_score(R_i(X_i))$ 
9:     for all  $j = 0$  to  $i - 1$  do
10:      if  $R_j(X_j).score < R_i(X_i).score \vee$ 
          $(R_j(X_j).score = R_i(X_i).score$ 
          $\wedge R_j(X_j)$  references to  $R_i(X_i))$  then
11:         $swap(R_i(X_i), R_j(X_j))$ 
12:      end if
13:    end for
14:  end for
15:  personalized\_view  $\leftarrow \emptyset$ 
16:  for all  $R_i(X_i) \in \mathbf{R}_T$  do
17:     $r_i \leftarrow \pi_{\forall a \in R_i(X_i)}(\mathbf{view})$ 
18:    for all  $j = 0$  to  $i - 1$  do
19:      if  $R_j(X_j)$  references to  $R_i(X_i) \vee$ 
          $R_i(X_i)$  references to  $R_j(X_j)$  then
20:         $r_j \leftarrow \pi_{\forall a \in R_j(X_j)}(\mathbf{personalized\_view})$ 
21:         $r_i \leftarrow r_i \bowtie r_j$ 
22:      end if
23:    end for
24:     $quota_{R(X)} \leftarrow base\_quota +$ 
        $\left[ \frac{R_i(X_i).score}{\sum_{R_j(X_j) \in \mathbf{R}_T} R_j(X_j).score} \right] \cdot (1 - base\_quota)$ 
25:     $K \leftarrow get\_K(dim\_memory \cdot quota_{R(X)}, R(X))$ 
26:     $r_i \leftarrow top\_K(order\_by\_tuple\_score(r_i))$ 
27:    personalized\_view  $\leftarrow \mathbf{personalized\_view} \cup \{r_i\}$ 
28:  end for
29:  return personalized\_view,  $\mathbf{R}_T$ 
30: end function

```

joined with each already analyzed relation with which it is related by means of a foreign key. In this way, each relation contains only tuples satisfying referential integrity constraints. It is worth noting that each relation is constrained by the previous ones; therefore, according to the list ordering criterion, relations with higher schema score have a higher freedom with respect to the ones with lower schema scores. Moreover, it may happen that, during this filtering, tuples with a high score are removed in order to satisfy referential integrity constraints; however, this situation does not represent a problem since the preference strategy is intended as a “soft” approach aiming at suggesting a possible personalization while referential integrity represents a hard constraint to be satisfied.

Once the constraints are fulfilled, tuples are personalized by ordering their scores and by applying a *top-K* selection (Lines 24–27). The K parameter states the maximum number of tuples that are stored in each table; for this reason, it is computed according to the relevance of the relation. A percentage quota of the memory space is assigned by means of a formula as the ratio between the average schema scores of the current relation and the sum of schema score of all the relations belonging to the view. It is worth noting that, by definition, the sum of all the percentage quotas is 1. Moreover, a *base_quota*, by default equal to 0, can be set by the designer or by the user in order to assign a minimum space

to tables; the higher the *base_quota*, the lower is the variance on relation dimensions. Given the percentage quota, the K parameter is computed by means of the *get-K* function defined by the memory occupation model. Note that this function is used for computing the maximum number of tuples of each relation that can be stored on the device. However, if an original relation belonging to the view tailored by the designer contains a smaller number of tuples or the customization process performs a very hard filter, part of the reserved space may remain empty. Thus, an improved version of Algorithm 4 may be defined for redistributing the spare space among the other tables. Finally, the overall customized view, called **personalized_view**, is updated with the computed relation.

Since the algorithm computes the K value for the *top-K* function by means of a memory occupation model, in case this model is missing the algorithm cannot be applied as it is. Instead, it is possible to tackle the problem in an iterative way by incrementally adding tuples to tables by fulfilling the balancing established by the table quotas.

EXAMPLE 6.8. *Let us consider the ranked tables’ schema from Example 6.6:*

```

CUISINES(cuisine_id:1, description:1)
RESTAURANT_CUISINE(restaurant_id:0.5, cuisine_id:0.5)
RESTAURANTS(restaurant_id:1, name:1, address:0.1,
            zipcode:0.5, city:0.1, phone:1, fax:0.1, email:0.1,
            website:0.1, closingday:1, openinghourslunch:0.5,
            openinghoursdinner:0.5, capacity:0.5, parking:0.5)

```

When using a threshold equal to 0.5, the resulting reduced schema is:

```

CUISINES(cuisine_id:1, description:1)
RESTAURANT_CUISINE(restaurant_id:0.5, cuisine_id:0.5)
RESTAURANTS(restaurant_id:1, name:1, zipcode:0.5,
            phone:1, closingday:1, openinghourslunch:0.5,
            openinghoursdinner:0.5, capacity:0.5, parking:0.5)

```

Then, it is possible to compute the average schema score: the second column of the table in Figure 7 shows the average scores of previous tables together with ones of other tables omitted in the previous part of the example. Finally, when considering 2Mb of available memory, it is possible to compute the space reserved for each table as shows the third column of the table in Figure 7.

Table	Average Score	Memory (Mb)
CUISINES	1	0.50
RESTAURANTS	0.72	0.35
RESERVATION	0.72	0.35
SERVICE	0.6	0.30
RESTAURANT_CUISINE	0.5	0.25
RESTAURANT_SERVICE	0.5	0.25

Figure 7: Example of computation of table disc space

6.5 Preference generation

Besides the described personalization flow, a further issue to be taken into account is the generation of preferences (step 5 of Figure 3). Two main approaches can be used for

generating contextual preferences: they can be specified by the user or automatically generated by means of a mining process. These approaches are briefly addressed in this subsection; however, as discussed in Section 7, an appropriate investigation is left as future work.

The user can express preferences while browsing data on his/her device, according to the model presented in Section 5; in particular, a user interface can be provided for easily specifying preferences. The specified preferences are stored on the mobile device in a local user profile and at the first connection they are synchronized with the preference profile stored in the repository on the server. The user can use the provided interface also for browsing and updating the preference profile by changing or deleting stored preferences. It is worth noting that, when accessing the local database, the user can only browse the view provided for the current context and, therefore, only express preferences on it; in this way, it is avoided that the user express preferences on data that do not belong to the current view leading to useless preferences.

A mechanism of preference mining can be adopted for automatic generation. In particular, preferences can be generated by means of a frequency analysis on the data access log recorded by the device; when the device is connected, the log is transmitted to the server where it is elaborated. In this way, the user implicitly suggest its interests on the basis of his/her data access frequencies. Furthermore, other advanced mechanisms of preferences generation may be based on the analysis of preferences in similar contexts and preferences of other users in similar contexts.

It is worth noting that the generation of the preferences is a critical task since in order to be effective, preferences have to act orthogonally w.r.t. the designer tailoring. In fact, if preferences express interests similar or opposite to the ones expressed by the tailoring, they may result inapplicable, if they are expressed on tailored data, and useless, if they are expressed on a whole relation of the tailored view. Moreover, the more details in the CDT, the more restrictive the query to derive the final view associated with a context is and the higher the risk of incurring in inapplicable preferences; it is the designer who shall choose the best trade-off. Partially this problem is overcome by allowing the user to express preferences only on the current view and by the fact that the mined log is related to the current view.

7. CONCLUSIONS

In this work we have proposed a novel approach to include preferences both on tuples and on attributes of contextual views. The approach is an extension of Context-ADDICT, a framework for tailoring data of integrated and possibly heterogeneous sources by using the notion of context. We are now defining a strategy to mine the user preferences on the basis of his/her previous interaction with the system. Moreover, we are developing a Java prototype to add the possibility to specify preferences and apply them before storing contextual portion of data on a user's small device. The prototype will perform a fine-grained tailoring of data by considering also the available device memory, as described in our proposal.

8. REFERENCES

- [1] R. Agrawal, R. Rantzaou, and E. Terzi. Context-sensitive ranking. In *Proc. of the 2006 ACM SIGMOD Intl. Conf. on Management of data*, pages 383–394, 2006.
- [2] R. Agrawal and E. L. Wimmers. A framework for expressing and combining preferences. In *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, pages 297–306, 2000.
- [3] C. Bolchini, E. Quintarelli, and R. Rossato. Relational data tailoring through view composition. In *ER*, pages 149–164, 2007.
- [4] C. Bolchini, F. A. Schreiber, and L. Tanca. A methodology for a very small data base design. *Information Systems*, 32(1):61–82, 2007.
- [5] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. of Intl. Conf. on Data Engineering*, pages 421–430, 2001.
- [6] M. J. Carey and D. Kossmann. On saying “enough already!” in sql. *SIGMOD Rec.*, 26(2):219–230, 1997.
- [7] J. Chomicki. Preference formulas in relational queries. *ACM Trans. on Database Syst.*, pages 427–466, 2003.
- [8] P. Ciaccia. Processing preference queries in standard database systems. In *Proc. of Intl. Conf. on Advances in Information Systems*, pages 1–12, 2006.
- [9] G. Das, V. Hristidis, N. Kapoor, and S. Sudarshan. Ordering the attributes of query results. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD Intl. Conf. on Management of data*. ACM, 2006.
- [10] P. Georgiadis, I. Kapantaidakis, V. Christophides, E. M. Nguer, and N. Spyrtos. Efficient rewriting algorithms for preference queries. In *ICDE 2008: Proceedings of the IEEE 24th International Conference on Data Engineering*, pages 1101–1110, 2008.
- [11] S. Holland, M. Ester, and W. Kießling. Preference mining: A novel approach on mining user preferences for personalized applications. In *Proc. of European Conf. on PKDD*, pages 204–216, 2003.
- [12] S. Holland and W. Kießling. Situated preferences and preference repositories for personalized database applications. In *ER*, pages 511–523, 2004.
- [13] W. Kießling. Foundations of preferences in database systems. In *Proc. of Intl. Conf. on VLDB*, pages 311–322, 2002.
- [14] G. Koutrika and Y. E. Ioannidis. Personalization of queries in database systems. In *Proc. of Intl. Conf. on Data Engineering*, pages 597–608, 2004.
- [15] MSDN Library SQL Server Developer Center - Page: Estimating the Size of a Database. <http://msdn.microsoft.com/en-us/library/ms187445.aspx>.
- [16] K. Stefanidis, E. Pitoura, and P. Vassiliadis. Adding context to preferences. In *Proc. of IEEE Intl. Conf. on Data Engineering*, pages 846–855, 2007.
- [17] A. H. van Bunningen, L. Feng, and P. M. G. Apers. A context-aware preference model for database querying in an ambient intelligent environment. In *DEXA*, pages 33–43, 2006.
- [18] A. H. van Bunningen, M. M. Fokkinga, P. M. G. Apers, and L. Feng. Ranking query results using context-aware preferences. In *ICDE*, pages 269–276, 2007.