ON CONTEXT AWARE PREDICATE SEQUENCE QUERIES

Hagen Höpfner
International University in Germany
Campus 3; 76646 Bruchsal; Germany

Keywords: Mobile Information Systems, Context Aware Systems, Databases, Conjunctive Queries.

Abstract: Due to the limited input capabilities of small mobile information system clients like mobile phones, it is not a must to support a descriptive query language like SQL. Furthermore, information systems with mobile clients have to address characteristics resulting from clients mobility as well as from wireless communications. These additional functions can be supported by a reasonable, well-defined notation of queries. Moreover, such systems should be context aware. In this paper we present a query notation named "context aware predicate sequence queries" which respects these issues.

1 INTRODUCTION

Most mobile information systems (MIS) are client/server systems with mobile clients that request data from a database supported server. Clients post queries and receive the answers. In wired environments querying is done by using the structured query language SQL. Mobile devices suffer from strict limited input capabilities. It is quite uncomfortable to write SQL statements with the keyboard of a mobile or the Grafitti-System of PalmOS based PDAs. So, it is not a must to support such a descriptive query language. As a replacement applications dynamically generate queries based on information inputed via a GUI. On the other hand, information systems with mobile clients have to consider the wireless communication. In order to reduce communication costs and to improve response times mobile clients cache data. Therefore, semantic caches were suggested in the literature. Obviously, this kind of caching is not possible if we allow the full variety of SQL. But, with a predicate based query representation it is possible (Höpfner and Sattler, 2003c). Another problem resulting from caching is the consistency. We have shown in (Höpfner, 2005a; Höpfner, 2006b) that, in general, only the server is able to decide about the relevance of updates. Therefore, we have to store the posted queries on the server site (Höpfner et al., 2004). This can be done more efficient (Höpfner, 2006a) with a well-defined query notation. Moreover, MIS should be context aware. If a tourist in Munich looks for restaurants, then she is not interested in restaurants in London. So, the query representation has to support context predicates in addition to the normal query predicates.

This paper presents a query notation named "context aware predicate sequence queries" that supports conjunctive SPJ-queries with inequalities, and self-joins in combination with context predicates. The paper is structured as follows: In Section 2 we discuss the predicate sequence query notation. Our approach for handling context predicates is presented in Section 3. In Section 4 both are coalesced in our context aware predicate sequence query notation. Notes on related work are included in Section 5. The paper ends in Section 6 with a summary and an outlook.

2 PREDICATE SEQUENCES

The predicate sequence query notation (PSQ) is based on the relational algebra but also includes aspects of calculi (Codd, 1972). Since PSQ consists of join-, selection-, and projection predicates and allows to rename tables, it supports the well known conjunctive SPJ-queries with inequalities but also allows self-joins. First ideas on the way to PSQ were published in (Höpfner and Sattler, 2003c). In (Höpfner et al., 2004) these ideas were enhanced by equal-joins. Now, we support θ-joins. The predicates are defined

Höpfner H. (2006).
ON CONTEXT AWARE PREDICATE SEQUENCE QUERIES.
In *Proceedings of the First International Conference on Software and Data Technologies*, pages 108-114
DOI: 10.5220/0001312001080114
Copyright © SciTePress

as follows:

Definition 1 (Selection Predicates) *Selection predicates correspond to the σ-operator of the rel. algebra. Let $\sigma_{\mathsf{scond}}(r(R_{\mathsf{name}}))$ be such a selection with the* conjunctive *select condition* scond*. Each conjunct of the select condition has to have either the form $s = A\ \gamma\ k$ or might be a attribute selection $s = A\ \gamma\ B$ with $\gamma \in \{\leq, <, =, \neq, >, \geq\}$, $A, B \in R_{\mathsf{name}}$, and $k \in \mathsf{dom}(A)$. This selection $\sigma_{\mathsf{scond}}(r(R_{\mathsf{name}}))$ corresponds to a set of selection predicates of the form $\{[\mathsf{name}.s_1], \ldots, [\mathsf{name}.s_n]\}$ with $n \in \mathbb{N}$ and $n = |\mathsf{scond}|$. Here, for all s_i with $1 \leq i \leq n$ also $s_i \in \mathsf{scond}$ holds.*

Example 1 (Selection Predicates) *Let's assume the relation* `Movies(MID, name, FSK, genre, actors, info)`*. A query could ask for all action movies with an FSK*[1] *greater than 16 years: (1) rel. algebra: $\sigma_{\mathsf{FSK}>16 \wedge \mathsf{genre}='\mathsf{action}'}(r(R_{\mathsf{movies}}))$; (2) calculus: $\{a, b, c, d, e, f | \mathsf{movies}(a, b, c, d, e, f) \wedge c > 16 \wedge d =' \mathsf{action}'\}$; (3) selection predicates: $\{[\mathsf{movies}.\mathsf{FSK} > 16], [\mathsf{movies}.\mathsf{genre} =' \mathsf{action}']\}$*

Definition 2 (Join Predicates) *Join predicates correspond to the $\bowtie_\theta$-operator of the rel. algebra. Let $r(R_{\mathsf{name}_1}) \bowtie_{\mathsf{jcond}} r(R_{\mathsf{name}_2})$ be such a θ-join with the* conjunctive *join condition* jcond*. Each conjunct of the join condition has to have the form $j = A\ \theta\ B$ with $\theta \in \{\leq, <, =, \neq, >, \geq\}$, $A \in R_{\mathsf{name}_1}$, and $B \in R_{\mathsf{name}_2}$. This join corresponds to a join predicate of the form $[\mathsf{name}_1, \mathsf{name}_2, (j'_1, \ldots, j'_n)]$ with $n \in \mathbb{N}$, and $1 \leq n \leq |\mathsf{jcond}|$. We claim a lexicographical order of the relation names within a join predicate. Furthermore, we claim that in each conjunct j'_i with $i \in \mathbb{N}$, and $1 \leq i \leq n$ the relation name is prefixed to each attribute. A conjunct in rel. algebra $j_i = A\ \theta\ B$ with $\theta \in \{\leq, <, =, \neq, >, \geq\}$, $A \in R_{\mathsf{name}_1}$, and $B \in R_{\mathsf{name}_2}$ is represented as $j'_i = \mathsf{name}_1.A\ \theta\ \mathsf{name}_2.B$.*

Example 2 (Join Predicates) *Let us assume the relation from Example 1 and the relation* `shown_in(date, time, SID, MID)`*. An example query that joins both relations could use the movie identifier* `MID`*: (1) rel. algebra: $r(R_{\mathsf{movies}}) \bowtie_{\mathsf{MID}=\mathsf{MID}} r(R_{\mathsf{shown\_in}})$ (2) calculus*[2]*: $\{a, b, c, d, e, f, g, h, i, j | \mathsf{movies}(a, b, c, d, e, f) \wedge \mathsf{shown\_in}(g, h, i, j) \wedge a = j\}$ (3) join predicate: $[\mathsf{movies}, \mathsf{shown\_in}, (\mathsf{movies}.\mathsf{MID} = \mathsf{shown\_in}.\mathsf{MID})]$*

In order to support *self joins* we allow the renaming of tables similar to the TABLE_NAME-AS-ALIAS-construct of SQL. Such an alias is written as `name@alias` and replaces the `name` in all affected predicates.

Definition 3 (Projection Predicates) *Projection predicates correspond to the π-operator of the rel. algebra. A projection $\pi_X(r(R_{\mathsf{name}}))$ with $X \subseteq R_{\mathsf{name}}$ is written as a projection predicate $[\mathsf{name}(x_1, \ldots, x_n)]$ with $n \in \mathbb{N}$, $1 \leq n \leq |X|$, and $\{x_1, \ldots, x_n\} = X$. Projections that base on join results like $\pi_{X_1, X_2, \ldots, X_i}(r(R_{\mathsf{name}_1}) \bowtie_{\theta_1} r(R_{\mathsf{name}_2}) \bowtie_{\theta_2} \cdots \bowtie_{\theta_{i-1}} r(R_{\mathsf{name}_i}))$ with $i, j \in \mathbb{N}, 1 \leq j \leq i$, and $X_j \subseteq R_{\mathsf{name}_j}$, $1 \leq n_j \leq |X_j|$, $\{x^j_1, \ldots, x^j_{n_j}\} = X_j$ are written as projection predicate $[\mathsf{name}_1(x^1_1, \ldots, x^1_{n_1}), \mathsf{name}_2(x^2_1, \ldots, x^2_{n_2}), \ldots, \mathsf{name}_i(x^i_1, \ldots, x^i_{n_i})]$.*

In calculus queries all exist quantified attributes that do not have any selecting affect are marked by "_". We handle attributes that are not specified in a projection predicate in a similar way.

Example 3 (Projection Predicates) *Let us assume the relation from Example 1. An example query might ask for* `FSK`*,* `genre`*, and* `name`*: (1) rel. algebra: $\pi_{\mathsf{name},\mathsf{genre},\mathsf{FSK}}(r(R_{\mathsf{movies}}))$ (2) calculus: $\{x, y, z | \mathsf{movies}(\_, x, z, \_, \_, y)\}$ (3) projection predicate: $[\mathsf{movies}(\mathsf{FSK}, \mathsf{genre}, \mathsf{name})]$*

Another example query might be based on the join result of Example 2 and ask for `name`*,* `date`*, and* `time`*: (1) rel. algebra: $\pi_{\mathsf{name},\mathsf{date},\mathsf{time}}(r(R_{\mathsf{movies}}) \bowtie_{\mathsf{MID}=\mathsf{MID}} r(R_{\mathsf{shown\_in}}))$ (2) calculus: $\{b, g, h | \mathsf{movies}(a, b, c, d, e, f) \wedge \mathsf{shown\_in}(g, h, i, j) \wedge a = j\}$ (3) join predicate: $[\mathsf{movies}, \mathsf{shown\_in}, (\mathsf{movies}.\mathsf{MID} = \mathsf{shown\_in}.\mathsf{MID})]$ (4) projection predicate: $[\mathsf{movies}(\mathsf{name}), \mathsf{shown\_in}(\mathsf{date}, \mathsf{time})]$*

With these definitions a predicate sequence query can be defined. Therefore, PP is the set of all projection predicates, VP is the set of all join predicates[3], and SP is the union of all selection predicate sets.

Definition 4 (Predicate Sequence Query) *With $V \subseteq VP$, $pp \in PP \cup \{\varepsilon\}$, $S \subseteq SP$, and $V \cup \{pp\} \cup S \neq \emptyset$ a conjunctive query $Q = \bigwedge_{vp \in V} vp \wedge \bigwedge_{sp \in S} sp \wedge pp$ is represented as a predicate sequence query $Q' = \langle vp_1 \ \ldots \ vp_n \ sp_1 \ \ldots \ sp_o \ pp \rangle$ with*
$\forall i, k \in \mathbb{N}, 1 \leq i < k \leq n; vp_i, vp_k \in V \cup \{\varepsilon\} \Rightarrow vp_i \lhd vp_k$, and $\forall i, k \in \mathbb{N}, 1 \leq i < k \leq o; sp_i, sp_k \in S \cup \{\varepsilon\} \Rightarrow sp_i \lhd sp_k$. At this, $\lhd$ means lexicographically smaller.

Predicates appear with regard to the following rules: **(1)** All join predicates have to be linked using relation names and it is not allowed to

[1]The FSK in Germany specifies the minimal age required for watching a movie.

[2]$\{a, b, c, d, e, f, g, h, i, a | \mathsf{movies}(a, b, c, d, e, f) \wedge \mathsf{shown\_in}(g, h, i, a)\}$ would also be correct. We here used a "θ-join compatible notation"

[3]In order to be consistent with our previous works we use VP instead of JP here.

have two join predicates joining the same two relations. Formally this means: Let be $vp_1 = [\texttt{name}_1^1, \texttt{name}_2^1, (X_1)]$ a correct PS-query. For correct predicate sequence queries with n join predicates $vp_1 = [\texttt{name}_1^1, \texttt{name}_2^1, (X_1)], \ldots, vp_n = [\texttt{name}_1^n, \texttt{name}_2^n, (X_n)]$ we claim that for each join predicate vp_i of the form $vp_i = [\texttt{name}_1^i, \texttt{name}_2^i, (X_i)]$ with $i \in \mathbb{N}$, and $1 < i \leq n$ there has to be at least one join predicate $vp_j = [\texttt{name}_1^j, \texttt{name}_2^j, (X_j)]$ with $j \in \mathbb{N}$ and $1 \leq j < i$ and that $\texttt{name}_1^i = \texttt{name}_1^j \vee \texttt{name}_1^i = \texttt{name}_2^j \vee \texttt{name}_2^i = \texttt{name}_1^j \vee \texttt{name}_2^i = \texttt{name}_2^j$ holds. Furthermore, it not allowed to have a join predicate $vp_j = [\texttt{name}_1^j, \texttt{name}_2^j, (X_j)]$ for which $\texttt{name}_1^i = \texttt{name}_1^j \wedge \texttt{name}_2^i = \texttt{name}_2^j$ holds.
(2) If a PS-query contains join predicates, selection predicates and projection predicates must not use relation names that are not used in a join predicate.
(3) In join free PS-queries all predicates must use the same relation name.
(4) PS-queries without joins and selections must contain exactly one projection predicate that uses exactly one relation name.

PSQ implies the order of predicates within a query. At first join predicates have to appear, followed by selection predicates and, at the end, the projection predicate. This corresponds to how such a query is answered. Selections base on join results and might use attributes that are not included in the final projection.

Example 4 (Predicate Sequence Query) *Let us reuse the relations from Example 3 and Example 2. An example query that uses all three kinds of predicates might ask for the starting time of action movies for adults (FSK>16). (1) rel. algebra:* $\pi_{\texttt{time},\texttt{name}}(\sigma_{\texttt{FSK}>16 \wedge \texttt{genre}='\texttt{action}'}(r(R_{\texttt{movies}}) \bowtie_{\texttt{MID}=\texttt{MID}} r(R_{\texttt{shown\_in}})))$ *(2) calculus:* $\{e, b | \exists a \exists b \exists c \exists d \exists e \exists f \texttt{movies}(a, b, c, d, \_, \_) \wedge \texttt{shown\_in}(\_, e, \_, f) \wedge f = a \wedge d =' \texttt{action}' \wedge c > 16\}$ *(3) predicate sequence query:* $\langle[\texttt{movies}, \texttt{shown\_in}, (\texttt{movies.MID} = \texttt{shown\_in.MID})]\ [\texttt{movies.FSK} > 16]\ [\texttt{movies.genre} =' \texttt{action}']\ [\texttt{movies}(\texttt{name}), \texttt{shown\_in}(\texttt{time})]\rangle$

Two important issues of calculus based query languages or representations are the relational completeness and the safety. We know that a calculus is complete if for each term τ of the rel. algebra there is a equivalent (safe) term η in the calculus. PSQ is not relational complete. We do not support (yet) set operations. However, because it is possible to map PSQ to SQL for the execution, PSQ has to be safe.

2.1 Converting PSQ to SQL

Mobile clients post PS-queries to a server where these queries have to be answered. Since we do not have a native PSQ support for query answering, we have to convert PS-queries into SQL statements.The conversion algorithm works as follows:
(1) Join predicates are reflected by the `WHERE`-part of an SQL statement. At this, the affected relation names are added to the `FROM`-list.
(2) Selection predicates are also reflected by the `WHERE`-part of an SQL statement and the relation names have to be added to the `FROM`-list
(3) Projection predicates are added to the `select_list` and the affected relation names flow into the `FROM`-list. For predicate sequence queries without projection predicates we add "`*`" to the `select_list`.
(4) A renaming of tables like `name@alias` is added to the `FROM`-part and there converted into `name as alias`. In the `WHERE`-part and in the `select_list` we finally replace this relation name by the `alias`.

Relation names or relation-name-alias-combinations that appear more than once, are included only once into the `FROM`-list. The following example illustrates the conversion from PSQ to SQL.

Example 5 (Conversion PSQ to SQL)
Conventing a query that do not use table renaming requires only the first three conversion steps. Let us assume self-join-free query $\langle[\textit{movies},\textit{shown\_in},(\textit{movies.MID}=\textit{shown\_in.MID})]\ [\textit{movies.FSK}>16]\ [\textit{movies.genre}=\textit{'action'}]\ [\textit{movies}(\textit{name})][\textit{shown\_in}(\textit{time})]\rangle$*. The result is:*

```
SELECT movies.name, shown_in.time
FROM movies, shown_in WHERE
movies.MID=shown_in.MID
AND movies.FSK>16 AND
movies.genre='action'
```

On the other hand, the query $\langle[\texttt{movies@M1}, \texttt{movies@M2}, (\texttt{movies@M1.MID} = \texttt{movies@M2.MID})]\ [\texttt{movies@M1.name!='a@b'}]\ [\texttt{movies@M1}(\texttt{name}), \texttt{movies@M2}(\texttt{FSK})]\rangle$ *is converted using all four steps:*

Step 1: `SELECT`
```
  FROM movies@M1, movies@M2
  WHERE movies@M1.MID=movies@M2.MID
```
Step 2: `SELECT`
```
  FROM movies@M1, movies@M2
  WHERE movies@M1.MID=movies@M2.MID
  AND movies@M1.name!='a@b'
```
Step 3: `SELECT movies@M1.name,`
```
  movies@M2.FSK
  FROM movies@M1, movies@M2
  WHERE movies@M1.MID=movies@M2.MID
  AND movies@M1.name!='a@b'
```

Step 4:
```
SELECT M1.name, M2.FSK
  FROM movies AS M1, movies AS M2
  WHERE M1.MID=M2.MID AND
  M1.name!='a@b'
```

3 CONTEXT PREDICATES

As mentioned in Section 1 MIS should consider the context under which a query is posted. This includes location, time as well as other contextual information. In fact, there exist many different definitions of the term context (Schilit, 1995; Dey, 2000; Chen and Kotz, 2000; Pascoe, 2001; Mitchell, 2002; Schmidt, 2002). We here use the definition by Dey:

> "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

Of course, formulating context predicates depends on the underlying context model. We here use our own generic context model published in (Höpfner and Sattler, 2003a). The basic ideas are as follows: We enhance parts of the database (db) on the server by additional information (e.g. position, time) and call these extended parts fragments. The additional information are called extensions v. Now, a mobile client submits (in addition to a predicate sequence query) information about its context (e.g. "I am in London."). On the server, this information is used for selecting fragments, that relate to the context of the mobile client. We distinguish between server-extensions $v^{se}_{name} \in \mathcal{V}_{se}$ and client-extensions $v^{ce}_{name} \in \mathcal{V}_{ce}$. Both are vectors (e.g. two-dimensional geographic coordinates $v_{gc}(x, y); x, y \in \mathbb{R}$). The difference between client- and server-extensions is the expressiveness of the parameters (in our last example x and y). On the server side parameters might be parameter-functions that compute the parameter values by querying the database. So, redundant storage of data, that is already in the database, is avoided.

A fragment $\mathcal{F}$ is an almost arbitrary segment of one or more relation(s) and can be explained as tuple $\mathcal{F} = (f, V)$ with a fragmentation-function f and a set of server-extensions $V = \{v_0, \cdots, v_n\}$; $n \in \mathbb{N}$; $v_0, \cdots, v_n \in \mathcal{V}_{se}$. A fragmentation-function $f(r(R)) = r'(R')$ with $R' \subseteq R$ and $r'(R') \subseteq \pi_{R'}(r(R))$ is a query in the query-language used by the database system on the host holding the database (e.g. SQL). In other words: A fragment is a view augmented by V.

In the original paper we called the context information, that are submitted to the server, replication criteria. But, we recognized that this construct can be used as *context predicate*, too. It consists of a client-extension, and of the minimal postulated, and the maximal allowed tolerance of this criterion. For example, $rk_{gc} = ((100, 200), 0, 20)$, means that the current position of the client is (100, 200) and the requested data must refer to a position in the circumference with the radius 20.

The decision, whether a fragment is of interest for the client or not, is done on the server using a so called ξ-function. If the result of this function is greater than or equal to the minimal postulated threshold and lower than or equal to the maximal allowed tolerance the respective fragment is taken into account. In our example such a ξ-function might be the Euclidean distance $\xi^{Euc}_{gc}(v_{gc_1}, v_{gc_2}) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ or the Manhattan distance $\xi^{Man}_{gc}(v_{gc_1}, v_{gc_2}) = |x_1 - x_2| + |y_1 - y_2|$. However, a ξ-function does not necessarily have to have the properties of a distance function, e.g., has not to be symmetric in the arguments.

Definition 5 (Client Extension) *A client extension is defined as $v^{se}_{name} = (\psi_0, \cdots, \psi_i)$ with the type identifier (name) and ψ_i ($i \in \mathbb{N}$) constant parameters.*

Definition 6 (Context Predicates) *A context predicate is defined similar to the* repl. criterion *in (Höpfner and Sattler, 2003a). Hence, it is a triple $rk = (v', \Delta^{min}, \Delta^{max})$ with the client extension $v' \in \mathcal{V}_{ce}$ and the threshold values Δ^{max}, $\Delta^{min} \in \mathbb{R}$ ($\Delta^{max} \geq \Delta^{min}$) of the allowed variance of this client extension. CP is the set of all context predicates.*

From (Höpfner, 2005b) we can learn the PSQ-conform notation of context predicates. In order to query the database, the identifying $name$ of each repl. criterion must be included into the repl. criterion values. For our geographical coordinates example the context predicate is `[gc,(100,200),0,20]`.

4 CONTEXT AWARE PSQ

With the definitions from Section 2 and Section 3 we can now discuss context aware predicate sequence queries (CaPSQ). Since we allow only conjunctive linked conditions in PSQ we also claim, that context predicates are added in a conjunctive manner.

Definition 7 (Context Aware PSQ) *With a predicate sequence query Q and a set of context predicates $C \subseteq CP$ a context aware predicate sequence query is defined as $Q' = Q \circ \langle\ cp_1\ \ldots\ cp_o \rangle$ with $\forall i, k \in \mathbb{N}, 1 \leq i < k \leq o; cp_i, cp_k \in C \cup \{\varepsilon\} \Rightarrow cp_i \lhd cp_k$. At this, $\lhd$ means lexicographically smaller and $\circ$ is the concatenation of the PSQ-query string and the context predicate list.*

4.1 Performing Context Aware PSQ

In contrast to native PSQ queries it is not “trivial” to perform CaPS-queries on a database system. Beside converting the PSQ-part one has to handle the context information. As discussed in Section 3 the evaluation of the context predicates uses ξ-functions that have to be implemented either in a middle-ware or directly as stored procedures in the DBS. We here assume, that an additional component, a mobility server (Höpfner and Sattler, 2003b), is used. Clients send their queries to the mobility server that evaluates the contextual information and rewrites the remaining db query.

Therefore, the mobility server maintains a table of server-extensions and fragments which are relevant for this type of context.

Table 1 illustrates this list of fragments. At the moment the fragments are defined manually. An administrator has to create a view which is then added to the list of fragments where she has do enhance it by the contextual information. If now a mobile client submits a CaPS-query, the context information are extracted and compared to the fragment list by Algorithm 1.

Algorithmus 1: *Finding relevant fragments*

```
ENSURE: INPUT: Fragment list
ENSURE: RETURN=$\emptyset$; // reset return set
01. for each $cp_i = (v', \Delta^{max}, \Delta^{min})$ do
02.     for each fragment $\mathcal{F}_j = (f, V)$ do
03.         if $\exists(v_n \in V | v_n$ is compatible to $v')$ then
04.             if $\Delta^{min} \leq \xi_v(v_n, v') \leq \Delta^{max}$ then
                RETURN=$\mathcal{F}_j \cup$ RETURN
05.         done
06.     done
07. return(RETURN);
```

For the first time we assume, that each view contains all relations and attributes that are required for answering the query. However, as mentioned above, client queries are generated by applications. So, it is known, which relations might be used. Nevertheless, beside others, the specification of the fragments is part of ongoing research. Therefore, we assume here - for simplification - that the fragments are defined over the Cartesian product of all relations in the database. Obviously, this assumption is not acceptable in real world systems but it guarantees the availability of all required information here.

The result of Algorithm 1 is a set of views that might be used for answering the clients query. But, the PSQ-part of the submitted query uses the relation names and the attribute names of the underlying database. So, we have to adapt the predicate sequence query to the selected views. Let $\mathbb{F}$ be the set of all selected views. We know that the result might contain tuples of all $\mathcal{F}_i \in \mathbb{F}$ with $i \in \mathbb{N}$, and $0 \leq i \leq |\mathbb{F}|$. If $\mathbb{F} = \emptyset$ holds, then the query result has to be empty. Otherwise, $Q(D) \cap \left(\bigcup_{i=0}^{|\mathbb{F}|} \mathcal{F}_i(D) \right)$ is the result. Here $Q(D)$ means that the query is performed directly on the database D. Because the views are defined on the same database, the intersection of the union of all selected fragments and the result of the query performed on the database form the result. Nevertheless, this is correct but not efficient. Especially if the views are materialized a better idea would be to use only the view, but this is ongoing work.

5 RELATED WORK

As aforementioned in Section 4 there are many publications in the field of context aware information systems. Beside the already issued ones, (Stefanidis et al., 2005) discusses the usage of OLAP techniques for realizing context aware systems. But, context awareness is also discussed in different research areas. Especially in the field of desktop computing. Here the time of the usage of an object is used to support a search later on (Freeman and Gelernter, 1996; Rekimoto, 1999; Ringel et al., 2003). A combination of location, time and task awareness is suggested in (Voida et al., 2002; Kaptelinin, 2003; Myka, 2005).

In combination with mobile clients, context awareness is often reduced to location awareness. For example, (Seydim et al., 2001b; Seydim et al., 2001a) discuss approaches for location based hoarding. But, location is only one part of the context of the usage of a system. (Ren and Dunham, 2000) presents an approach for handling location based queries. Here, the focus lies on the client site where the results of such queries are cached. Unfortunately, the authors do not say something about how they perform their queries. Furthermore, we do not limit ourself to location awareness but support a generic context model.

The ContextSQL approach presented in (Höding et al., 2003a) tries to integrate contextual information in the optimization of query processing. Therefore, a new SQL clause `BY CONTEXT` is introduced and a middle-ware component (Höding et al., 2003b) manages the context usage. This approach is comparable to our approach. But, on the one hand we know, that the variety of SQL statements hampers the cache management on the client as well as the cache maintenance by the server. On the other hand ContextSQL was presented as a first idea without algorithms or proper definitions and the author did not publish any new results.

As far as we know, there is no other approach that tries to combine context awareness with a query nota-

Table 1: List of fragments.

fragment	used server-extensions	frag.-function
Chelsea	(gc,`SELECT x,y FROM geocoord WHERE city='London' and district='Chelsea'`)	Chelsae_Data
Greenwich	(gc,`SELECT x,y FROM geocoord WHERE city='London' and district='Greenwich'`	Greenwich_Data

tion that considers the special capabilities of information systems with mobile clients. But, of course, there is related work.

6 SUMMARY AND OUTLOOK

We presented an approach to notate db queries in a way that allows the easy usage of the queries for indexing purposes in a context aware environment. We introduced context aware predicate sequence queries that correspond to conjunctive queries with inequalities and self-joins but also allow the integration of contextual information into the query. We discussed the formal definitions of CaPSQ as well as how to perform such queries. The usability of our approach, at least of the PSQ notation, was proven during a PhD-Project (Höpfner, 2005a) where it was used for server site client indexing. Since PSQ as a well defined syntax, it is obviously more suitable for indexing issues than descriptive query languages, logical queries or (of course) graphical approaches. However, we have shown that it is possible to translate PSQ into SQL, which is necessary for performing the queries on existing database management systems.

Although, CaPSQ is a good basis, it is a starting point for more research. Some of the open issues were already issued in the text. Fragments are defined manually, but we'll try to learn such structures. We also work on the integration of the various subsystems and on the implementation of our scalable mobility server that will support an easy realization of database based information systems with mobile clients. Furthermore, we have to handle with the consistency of context based systems. In previous works we considered consistency without the context predicates. But, our context model opens two new consistency levels which result from context modifications and from database modifications that affect the frag.-functions. The most urgent issue we will focus is the pure usage of materialized views instead of frag.-functions. From the theoretical point of view this issue should not lead to bigger problems but especially attribute name conditions in real existing database management systems raise some practical issues.

REFERENCES

Chen, G. and Kotz, D. (2000). A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College.

Codd, E. F. (1972). Relational Completeness of Data Base Sublanguages. In *Data Base Systems*, pages 65–98, Englewood Cliffs, New Jersey. Prentice-Hall.

Dey, A. K. (2000). *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, College of Computing, Georgia Inst. of Technology.

Freeman, E. and Gelernter, D. (1996). Lifestreams: a storage model for personal data. *ACM SIGMOD Record*, 25(1):80–86.

Höding, M., Leich, T., and Plack, M. (2003a). Conext-Based Querying for Mobile Applications with ContextSQL. In *Proc. of the 7th World SCI Multiconference*.

Höding, M., Plack, M., and Leich, T. (2003b). A Flexible Middleware for Mobile Computing Using ContextSQL. In *Proc. of the 7th World SCI Multiconference*.

Höpfner, H. (2005a). *Relevanz von Änderungen für Datenbestände mobiler Clients*. Dissertation, Fakult. für Informatik, Universität Magdeburg. in German.

Höpfner, H. (2005b). Towards Update Relevance Checks in a Context Aware Mobile Information System. In *INFORMATIK 2005 - Informatik LIVE!, Band 2*, volume P-68 of *LNI*, pages 553–557, Bonn, Germany. GI.

Höpfner, H. (2006a). Anfragebasierte Client-Indexierung in Client-Server-Informationssystemen. *Informatik - Forschung & Entwicklung*, 20(4):209–221.

Höpfner, H. (2006b). Update Relevance under the Multiset Semantics of RDBMS. In *Mobile Informationssysteme*, volume P-76 of *LNI*, pages 33–44, Bonn, Germany. GI.

Höpfner, H. and Sattler, K.-U. (2003a). Semantic Replication in Mobile Federated Information Systems. In *Proc. of the 5th EFIS*, pages 36–41, Amsterdam. Ios Press Inc.

Höpfner, H. and Sattler, K.-U. (2003b). SMoS: A Scalable Mobility Server. In *Posters of 20th BNCOD*, pages 49–52. School of Math. & Inform. Sciences; Coventry Univ.

Höpfner, H. and Sattler, K.-U. (2003c). Towards Trie-Based Query Caching in Mobile DBS. In *Post-Proc.*

of the Workshop Scalability, Persistence, Transactions - Database Mechanisms for Mobile Applications, volume P-43 of *LNI*, pages 106–121, Bonn. GI.

Höpfner, H., Schosser, S., and Sattler, K.-U. (2004). An Indexing Scheme for Update Notification in Large Mobile Information Systems. In *EDBT 2004 Workshops*, volume 3268 of *LNCS*, pages 345–354, Berlin. Springer.

Kaptelinin, V. (2003). UMEA: translating interaction histories into project contexts. In *Proc. of the SIGCHI conference on Human factors in computing systems*, pages 353–360, New York, NY, USA. ACM Press.

Mitchell, K. (2002). *Supporting the Development of Mobile Context-Aware Computing*. PhD thesis, Departement of Computing, Lancaster University.

Myka, A. (2005). Nokia Lifeblog — towards a truly personal multimedia information system. In *Proc. of the 8th Workshops of the GI-working group mDBIS*.

Pascoe, J. (2001). *Context-Aware Software*. PhD thesis, Computing Lab., University of Kent at Canterbury.

Rekimoto, J. (1999). Time-machine computing: a time-centric approach for the information environment. In *Proc. of the 12th annual ACM symposium on User interface software and technology*, pages 45–54, New York, NY, USA. ACM Press.

Ren, Q. and Dunham, M. H. (2000). Using semantic caching to manage location dependent data in mobile computing. In *Proc. of the MobiComm2000*, pages 210–221.

Ringel, M., Cutrell, E., Dumais, S., and Horvitz, E. (2003). Milestones in Time: The Value of Landmarks in Retrieving Information from Personal Stores. In *Proc. of the Ninth IFIP TC13 International Conference on Human-Computer Interaction*, pages 184–191, Amsterdam. IOS Press.

Schilit, W. N. (1995). *A System for Context-Aware Mobile Computing*. PhD thesis, Columbia University, New York, USA.

Schmidt, A. (2002). *Ubiquitous Computing – Computing in Context*. PhD thesis, Comp. Dep., Lancaster Univ.

Seydim, A., Dunham, M., and Kumar, V. (2001a). An architecture for location dependent query processing. In *DEXA '01:Proceedings of the 12th International Workshop on Database and Expert System Applications*, pages 549–550, Washington. IEEE.

Seydim, A. Y., Dunham, M. H., and Kumar, V. (2001b). Location-Dependent Query Processing. In *Proc. of the 2nd ACM International Workshop on Data Engineering for Wireless and Mobile Access*, pages 47–53, New York, NY, USA. ACM Press.

Stefanidis, K., Vassiliadis, P., and Pitoura, E. (2005). On Supporting Context-Aware Preferences in Relational Database Systems. In *Proc. of the First International Workshop on Managing Context Information in Mobile and Pervasive Environments*.

Voida, S., Mynatt, E. D., MacIntyre, B., and Corso, G. M. (2002). Integrating virtual and physical context to support knowledge workers. *IEEE Journal on Pervasive Computing*, 1(3):73–79.