Location-aware Privacy and More: A Systems Approach using Context-aware Database Management Systems*

Walid G. Aref
Department of Computer Science
Purdue University
West Lafayette, Indiana 47907
aref@cs.purdue.edu

Hicham G. Elmongui
Department of Computer Science
Purdue University
West Lafayette, Indiana 47907
elmongui@cs.purdue.edu

Mourad Ouzzani
Cyber Center
Purdue University
West Lafayette, Indiana 47907
mourad@cs.purdue.edu

ABSTRACT

When a user issues a query, database engines will usually return results based solely on the query and the content of the database. However, query issuers have a "context" which if taken into account will certainly change the outcome of the query. Thus, when responding to the query, the database system can consider the query issuer's context and return only the objects/tuples in the database that not only satisfy the query predicates but also are relevant to the query issuer's context. In this paper, we give an overview of Chameleon; a context-aware database management system. Chameleon introduces SQL-level constructs that describe the "context" in which the query is issued as well as the reciprocal contexts of the objects in the database. By tying the query issuer's contexts with the corresponding contexts of the objects in the database, Chameleon can retrieve the objects of relevance to the query context. Using Chameleon's general interfaces for context definition and awareness activation, we show how database systems that offer not only location-sensitive privacy but also other forms of privacy, e.g., both location-sensitive and time-sensitive privacy profiles for their users can be realized easily. Several modeling and performance challenges for realizing context-aware database management systems are presented.

Categories and Subject Descriptors

H.2 [**Database Management**]. H.2.1 [**Logical Design**]: Data models.

General Terms

Algorithms, Management, Performance, Design, Languages.

Keywords

Context awareness, privacy, preferences, personalization, database systems

1. INTRODUCTION

As applications and application requirements grow in complexity, the underlying data management system has to increase in sophistication to cope with this complexity. In the early days, when applications, e.g., Geographic Information Systems (GIS), demanded efficient handling of large amounts of spatial data, DBMSs had to increase in sophistication to handle location data efficiently. This included the costly development of spatial indexing techniques that support concurrency and recovery, spatial query processing algorithms, e.g., spatial join algorithms with and without spatial indexes, and locality preserving strategies for disk placement of spatial data.

Similarly, Hippocratic databases have been proposed to address the privacy policy requirements that users' data are being used only by the intended recipients and only for the purposes approved by the data owners [1]. Systems, e.g., Hippocratic PostgreSQL [2] have been prototyped to provide controlled disclosure of the users' data according to the users' approved privacy policies.

In order for database systems to provide users with location-aware privacy, tremendous effort has to take place to develop combined Hippocratic and spatial database engines, which is very costly. In this paper, we present a systems approach to address this issue. Chameleon, a context-aware DBMS, is an extensible database server that uses contexts to eliminate the need for tailoring specialized engines [3], e.g., a spatial database engine, a Hippocratic database engine, a location-sensitive Hippocratic database engine, a time-sensitive, location-sensitive Hippocratic database engine (refer to Figure 1). Instead, using Chameleon, one can realize these systems by defining appropriate contexts using Chameleon's context definition and manipulation languages.

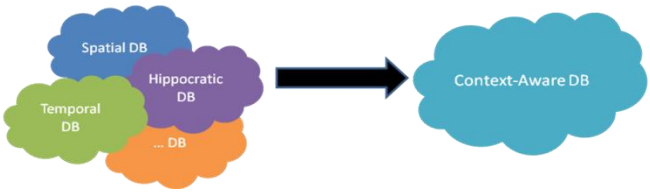

Figure 1: The vision behind Chameleon.

Chameleon supports two notions of context: the context surrounding the query issuer and the reciprocal contexts of the objects stored in the database. The query context reflects the situation of the query issuer, e.g., the query issuer's location, the time the query is issued, the identity of the query issuer, or even the temperature or the weather conditions surrounding the query issuer. Chameleon takes these situations into consideration when answering a query. For example, in Figure 2, when querying the database asking for a close-by restaurant, the user wants the database system to return restaurant responses that match the user's current context, i.e., her location, the time the query is issued, and her personal diet and dietary restrictions.

*The authors acknowledge the support of the National Science Foundation under Grant IIS-0811954.

ACM SPRINGL '09 November 3, 2009. Seattle, WA, USA

Database objects have contexts that are reciprocal to the query issuer's contexts, e.g., (refer to Figure 3), the location of the database objects, the time duration of an object (or when the object can be available for querying), and the identity of the object (or the ids of the query issuers or classes of query issuers that are allowed to access the object).

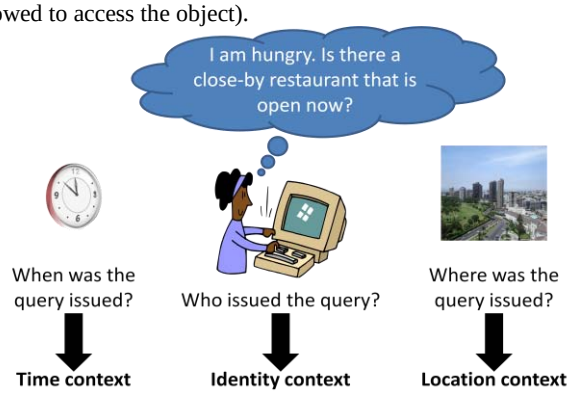

Figure 2: Chameleon considers the user's active contexts (user's time, location, and identity) when responding to the user's query.

In Chameleon, we can combine multiple contexts into more complex ones using the proposed context composition, e.g., a Hippocratic DBMS that also is location- and time-sensitive by combing the location-, temporal-, and identity-sensitive contextual services.

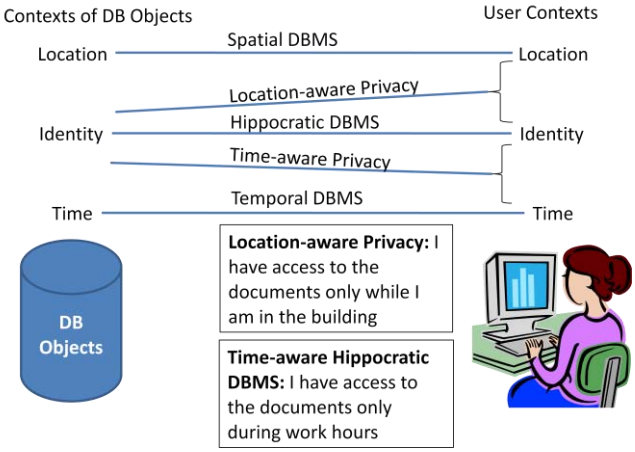

Figure 3: Illustration of what user and DB object contexts are combined in Chameleon to realize various specialized DB engines using the same context interfaces in Chameleon.

In this paper, we give several proof-of-concept instantiations of Chameleon, e.g., one to realize a privacy-aware (Hippocratic) database server, and another to realize a spatial database server using the same proposed constructs and interfaces of Chameleon. Further, we show how contexts can be combined within Chameleon to realize more complex systems, e.g., a server that supports location- and time-aware privacy database, i.e., one where the privacy profiles of database objects depend not only on the identity and purpose of the query issuers but also on the query issuer's location and time when they issue the query (location-aware and time-aware privacy).

Chameleon is built using extensions to PostgreSQL that include:

(1) New syntax and query rewrite components to define contexts and to issue queries that use contexts,
(2) New query operators that process contexts, e.g., the Skyline join and FilterMark operators that are vital when processing queries that involve contexts, and
(3) Extensions to the query optimizer to invoke these new operators when appropriate.

The rest of this paper proceeds as follows. In Section 2, we present the context classes in Chameleon and the dimensions that we use to define a context. Section 3 presents the syntax and semantics of the extended SQL constructs within Chameleon that defines contexts. Section 4 addresses conceptual evaluation of context-aware SQL commands and implementation issues. Section 5 gives several example instantiations to realize a spatial database server, a Hippocratic database server, and a location-aware and time-aware Hippocratic database server. The latter illustrates how contexts can be combined in Chameleon to realize more complex servers. Section 6 discusses related work. Section 7 includes some research challenges and concluding remarks.

2. CONTEXT CLASSES

For illustration, we use a simple bookstore example, where users express their preferences when accessing the bookstore database. Later in the paper, we will show more sophisticated cases, mainly for system realizations of location-aware privacy as well as location- and time-aware privacy.

Table 1 gives a projection on the table books that contains information about books in a certain bookstore. Among other pieces of information included in this table, we can find the name of a book, the years of its publishing, the category under which this book falls, the type of the cover (HC for hardcover or PB for paperback), as well as whether or not the book is in stock. Only the books in stock that are relevant to the user's context are retrieved.

In contrast to existing work on context-aware systems that are built on top of a database, we propose to incorporate context awareness inside the DBMS. We adopt a broad definition of what a context is. For example, the physical location in space of the query issuer when he/she issues the query can be part of the query context. The time the query is issued and the identity of who issued the query may also both be part of the query context. We support the following two classes of contexts:

(1) The User Context, i.e., the context of the query issuer.

(2) The Object Context, i.e., the contexts of the queried data.

Figure 4 illustrates a high-level view of Chameleon's context model. We classify user contexts according to three dimensions. These dimensions will be used when the application developer defines a context in Chameleon. These dimensions will reflect in the access method selection of any query on the tables that are affected by that context.

Dimension 1 - Context Sign: The sign of a user context is either "positive" (S) or "negative" (G). A positive context defines what the context is. For instance, if the context is location, an instance of a positive context is the preferred locations by the user, e.g.,

specified as a range. On the other hand, a negative context defines what the context is not. An instance of a negative location context

row	title	year	category	cover	instock
1	book01	2004	Science Fiction	HC	√
2	book02	2002	Travel	PB	√
3	book03	2001	Medicine	HC	X
4	book04	2000	Cooking	PB	√
5	book05	1997	Science Fiction	PB	√
6	book06	2001	Medicine	HC	√
7	book07	1995	Cooking	PB	X
8	book08	1996	Travel	PB	√
9	book09	2000	Science Fiction	PB	√
10	book10	2003	Medicine	PB	√
11	book11	2005	Travel	HC	√
12	book12	2006	Cooking	HC	X
13	book13	2004	Medicine	PB	√
14	book14	2006	Science Fiction	HC	√
15	book15	2005	Travel	HC	√
16	book16	2006	Cooking	HC	√
17	book17	1976	Medicine	PB	√
18	book18	2001	Travel	PB	√
19	book19	2007	Science Fiction	HC	√
20	book20	1988	Cooking	PB	X
21	book21	1993	Science Fiction	PB	√
22	book22	2006	Medicine	HC	X
23	book23	1999	Cooking	PB	X
24	book24	2006	Medicine	HC	√
25	book25	2006	Travel	PB	√

Table 1: The example bookstore database.

is the locations or regions not desired by or prohibited to the user. In the running example, an instance of a positive context is the willingness to buy hardcover books only. However, trying to avoid science fiction books is a case for a negative context.

Dimension 2 - Contextual Relation: The contextual relation is the relation among the contextual data. This relation mainly shows the order of relevance of the contextual data. The contextual relation can be an equivalence relation (Q). In this case, data that comply with all contextual values are reported with no special ordering. Besides, the contextual relation can also be a total ordering relation (T). This relation would reflect on the data being reported to the user. The data will be sorted on the rank of the contextual values with which the data conform. Moreover, the contextual relation can be a partial ordering relation (P). In contrast to the previous relation, the rank of the contextual values here will follow a partial order rather than a linear order. Referring to the books table, an example of an equivalence relation is the equal willingness to buy a science fiction book or a travel book. However, if the user is interested in new books, a total ordering relation would be more appropriate to retrieve the latest books first. If the user prefers cooking books over science fiction books, and travel books over medicine books, with no specific preference among the other combinations, would need to specify her context to contain partially ordered contextual values. Partially ordered values may be transformed into linear ordered values using an appropriate (possibly online) topological sort algorithm. This is out of the scope of this work, but we add the modeling part here for completeness.

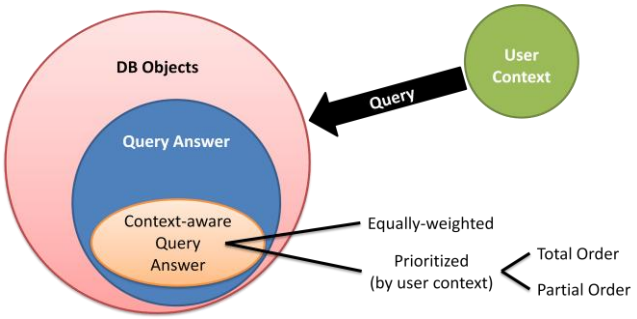

Figure 4: Abstraction of Contexts in Chameleon.

Dimension 3 - Listing of Data: By listing of data we refer to how the data should be listed. Specifically, should the data that does not conform to the user context be excluded from the listed data? Or, should those data be included but come last? The former case is termed "unlisted excluded" (X), whereas the latter is termed "unlisted included" (N).

Consider the bookstore example, if the user context is the willingness to buy travel books only, the user context gets the "unlisted (other book categories) excluded". Nevertheless, an "unlisted included" context can be illustrated by the preference to buy hardcover books but still get the paperback books down in the list -- after retrieving all hardcover books). In a location context example, if the user context is the willingness to buy houses that lie within a certain geometric region, say R, then "unlisted excluded" means that houses outside R are not reported to the user, whereas "unlisted included" lists the houses outside R after listing the houses inside R.

2.1 User Context as a 3D Point

Based on these three dimensions, each user context is viewed as a point in the 3D space defined above. For instance, in the bookstore example, one might be willing to buy only science fiction or travel books with no particular preference between these two types. This is an example of a positive user context having an equivalence contextual relation with the unlisted contextual values excluded. Whenever a user with the aforementioned context selects all tuples from the table books, only rows 1, 2, 5, 8, 9, 11, 14, 15, 18, 19, 21, and 25 are retrieved. If the user defines the same context to have a total ordering relation instead of an

equivalence relation such that science fiction books have higher rank than travel books, the retrieved rows will be: 1, 5, 9, 14, 19, 21, 2, 8, 11, 15, 18, and 25.

All points in this 3D space are valid when the user context is positive. However, when the user context is negative, only contextual values with the unlisted included are valid. This restriction is due to the definition of a negative user context; the user is specifying what context values are not current, and hence all the others should be current (or nothing will be ever returned). Moreover, for a negative user context, since the user only describes the complement of her positive context, no rank is explicitly specified for that actual positive context. Therefore, the equivalence relation would be implicitly understood for the contextual values. Figure 5 summarizes the overall model for contexts in Chameleon along with the three dimensions. The next section illustrates how these context dimensions can be used to specify contexts using newly proposed SQL constructs.

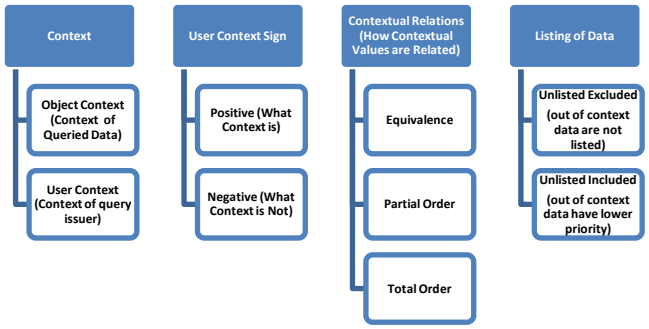

Figure 5: Conceptual Model of Contexts in Chameleon.

3. SQL EXTENSIONS FOR CONTEXT AWARENESS

In this section, we cover the various constructs that Chameleon uses to enable context awareness inside a DBMS. A brief overview of these constructs is also presented in [3].

Creating Object Contexts: Chameleon uses the CREATE OBJECT CONTEXT statement to define an object context. When the object context is part of the object relation, it does not need to be defined explicitly.

```
CREATE OBJECT CONTEXT context_name (
  {col_spec | table_constraint} [, . . . ]
  , table_binding );
```

Contextual values will be stored inside relations to be easily incorporated within the query processor. The CREATE OBJECT CONTEXT statement has similar constructs to those in the CREATE TABLE statement and it also creates an object context relation. For instance, *col_spec* refers to the specification of a column such as name, data type, default values, and so on. On a similar vein, *table_constraint* refers to any constraints on the whole context table such as check constraints.

The construct *table_binding* is the main construct that connects the object with its context. Specifically, *table_binding* has the format below.

```
BINDING KEY ([col_name [, . . . ]])
  REFERENCES ref_table [( ref_col [, . . . ] )]
  WITH bool_expr
```

The first part of the BINDING KEY is similar to the FOREIGN KEY. There are three main differences between these two types of keys. The first difference is that a foreign key in a table has to refer to a primary key in another table. This constraint does not exist for the binding key. A binding key binds the contextual value to possibly more than one object, since more than one object may exist in the same context. The second difference is that the decision to bind a contextual value with an object does not have to be equality with a column value in the referenced table. The WITH construct defines a Boolean expression that serves as the binder in case the expression evaluates to true. The third difference is that the binding key might not contain any context attribute referencing an attribute in the base table, but rather only the Boolean expression that might also contain attributes from any object context to the referenced table. Examples will be shown in the case studies section to illustrate these differences further.

Creating User Contexts: Similar to object contexts, each user context will materialize to a relation. Chameleon uses the following syntax to define a user context.

```
CREATE  [context_sign] CONTEXT context_name (
            {col_spec | table_constraint} [, . . . ]
            , table_binding [, . . . ]
            [, substitution_ key [, . . . ] ]
) [AS contextual_relation]
[WITH UNLISTED unlisted_status];

context_sign: positive | negative

contextual_relation: equivalence
                   | total order [USING ordering_func]
                   | partial order

unlisted_status: excluded | included
```

For each table affected by a user context, a binding key is used to show how the context reflects on the table. Therefore, there might be more than one binding key in a user context. Upon the creation of a user context, an implicit column is created to hold the user name of the current user. Therefore, each contextual value is associated with a certain user. Also, if an ordering relation is used for the contextual relation, then another implicit column is created to hold the rank of that contextual value. This rank can either be input by the application while acquiring contextual data, or can be computed using an ordering function ordering_func. In the latter case, the rank column does not need to exist.

Chameleon builds default indexes for context relations. Object contexts get non-clustered indexes on the context keys. User contexts are clustered in a B-tree index using the clustering key (*user_name, context_key*) if the contextual relation is equivalence. If the contextual relation is a total ordering relation, then the user context is clustered on (user name, context key) if the unlisted are to be included and on (*user_name, rank*) if the unlisted are to be excluded.

The substituting key will be discussed in detail in the next section. Populating the contextual relations will be made using standard SQL INSERT statements. Also, other data manipulation statements will still work on the contextual relations.

Global Substitution Construct: Some attributes need to be modified for presentation purposes if we want to enable context awareness. For instance, if the context is the location of a user, and the user is currently in France, then we might want all prices, in all tables, to be converted to Euro. This conversion is called global substitution, since the substitution occurs for all tables according to the current context. The substituting key defines such conversion, and is specified while defining the user context as follows.

SUBSTITUTE table_name (col_name)
BY *expression*;

The expression that substitutes the attribute can be any expression in which attributes from table_name, its object contexts, as well as the user context may appear. The substitute clause is useful in limiting the disclosure of an attribute value if the query issuer is not allowed to view that value. The substitute expression would be to display a null value instead of the original attribute value.

Setting Active Contexts: The application user may have many contexts, not all of them need to be current all the time. Therefore, we introduce the construct SET ACTIVE CONTEXT to define the current contexts to be taken into account for that user. The user_name has the CURRENT USER as a default.

SET ACTIVE CONTEXT [FOR USER user_name]
AS context_name [, . . .];
{ [WITH RANKING ORDER context_name [, . . .]]
| [WITH RANKING EXPRESSION *expression*
| [WITH SKYLINE OF *expression* {MAX | MIN} [, . . .]]};

Before querying, the command SET ACTIVE CONTEXT is issued. This command specifies the contexts to be considered when evaluating the query. It also specifies how to prioritize and combine multiple contexts.

The SET ACTIVE CONTEXT statement allows composing complex contexts from basic ones. If all the basic contexts that are used to compose a complex context have equivalence contextual relations only, then the order of executing the contexts is given by the order the contexts are listed in the AS clause.

In Chameleon, we support three different ways for combining contexts (refer to Figure 6). Consider the following two contexts C1 and C2 that are set to be active upon issuing a query. We use two contexts for simplicity in the presentation.

- Ordering:
  WITH RANKING ORDER Context1, Context2
- Multi-feature Ranking:
  WITH RANKING EXPRESSION
  2*Context1.rank+Context2.rank
- Skyline:
  SKYLINE Context1 MIN, Context2 MAX

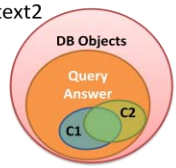

Figure 6: Examples of Three Ways for Combining Contexts in Chameleon.

Option 1 - Ordering: Using this option, Chameleon can combine the contexts by simply listing the contexts in some order, e.g., C1, C2. In this case, objects in the database are ordered according to Context C1 with ties broken according to the order within C2.

Option 2 - Multi-feature Ranking: We can combine the contexts C1 and C2 according to a weighted ranking function of C1's and C2's individual ranks.

Option 3 - Skyline Ordering: Skyline ordering is needed when the multiple contexts are independent and their ranks cannot be aggregated together. We can combine the individual contexts by returning the tuples that are not dominated by any other tuples (see Figure 7). The skyline operator [18] is used for that purpose. The WITH SKYLINE clause is used to specify to the skyline operation which expressions to use as the input ranks in the computation.

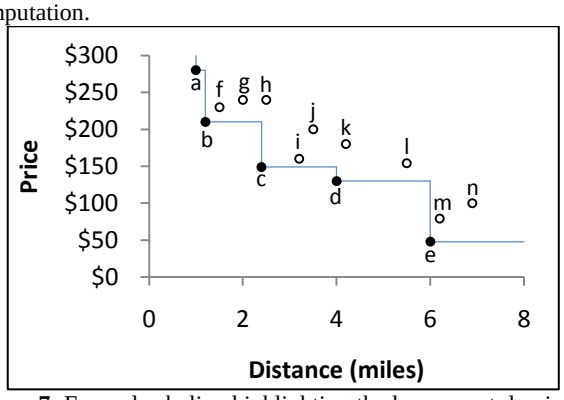

Figure 7: Example skyline highlighting the houses not dominated by other houses with respect to price and closeness to the beach.

Querying given the Active Contexts: When issuing a query, the query issuer invokes the active contexts in the following way:

WITHIN_MY_CONTEXT <Select Statement>

This command invokes the active contexts set by the SET ACTIVE CONTEXT command when evaluating the query.

4. CONCEPTUAL EVALUATION

In this section, we show why the above constructs enable context-aware query processing. We continue with our running example where someone is accessing the database of a bookstore. Only the books in stock that are relevant to the user's context are retrieved. Examples of contexts are given, their definitions using the above constructs are provided, and then we show how they are evaluated to give the desired results. First, we start by simple contexts, and later we show how these contexts are combined together to compose more complex contexts. In all the scenarios below, the user is executing the following query, and the results are the relevant tuples.

```
SELECT *
FROM books
WHERE books.instock;
```

Context 1: *The user has a preference for only books of a certain category (e.g., Science fiction).*

This context may be defined as:

```
CREATE POSITIVE CONTEXT ctxt_category_SQX (
    category varchar(20),
    BINDING KEY (category)
        REFERENCES books(category)
) AS EQUIVALENCE WITH UNLISTED EXCLUDED;

SET ACTIVE CONTEXT AS ctxt_category_SQX;
```

We give the suffix SQX to the context name above to emphasize that it is a positive [S] context with an equivalence [Q] contextual relation and that the unlisted categories in the context are to be excluded [X]. For the above example, when the user issues Q_u above, the actual query that is executed is given below. Typically, the binding key is used to join the books table with the context table, and only the books whose category exists in the context are to be returned. The following query reflects this semantic.

```
SELECT T.*
FROM books T
   INNER JOIN ctx_category_SQX C1
      ON(T.category = C1.category
      AND C1.user_name = CURRENT_USER)
WHERE T.instock;
```

Context 2: *The user's preference is for books published in 2005, and then those published in 2006 before all other books.*

This context may be defined as:

```
CREATE POSITIVE CONTEXT ctxt_year_STI (
    year integer,
   BINDING KEY (year) REFERENCES books(year)
) AS TOTAL ORDER WITH UNLISTED INCLUDED;

SET ACTIVE CONTEXT AS ctxt year STI;
```

Again, the suffix STI of the current context emphasizes that it is a positive [S] context with a total order [T] contextual relation and that the unlisted years in the context are to be included [I]. For the above example, in response to Q_u, the actual query that is executed is given below. Typically, the binding key is used to join the books table with the context table. In this case, the type of join is a left outer join, and therefore, all books will be returned at the end. The output rows are to be sorted based on the year rank, which is specified implicitly in the context as it is an ordering context. Rows with NULL context rank appear later in the list. The following query reflects this semantics.

```
SELECT T.*
FROM books T
   LEFT OUTER JOIN ctx_year_STI C1
      ON(T.year = C1.year
      AND C1.user_name = CURRENT_USER)
WHERE T.instock
ORDER BY C1.rank;
```

Context 3: *The user prefers hardcover over paperback books.*

This context may be defined as:

```
CREATE POSITIVE CONTEXT ctxt_cover_STX (
    cover integer,
   BINDING KEY (cover) REFERENCES books(cover)
) AS TOTAL ORDER WITH UNLISTED EXCLUDED;

SET ACTIVE CONTEXT AS ctxt_cover_STX;
```

For the above example, in response to Q_u, the actual query that is executed is given below. Typically, the binding key is used to join the books table with the context table. The output rows are to be sorted based on the cover rank, which is specified implicitly in the context as it is an ordering context. The following query reflects this semantics.

```
SELECT T.*
FROM books T
   INNER JOIN ctx_cover_STX C1
      ON(T.cover = C1.cover
      AND C1.user_name = CURRENT_USER)
WHERE T.instock
ORDER BY C1.rank;
```

Context 4: *The user does not prefer (wants to avoid) any science fiction books.*

This context may be defined as:

```
CREATE NEGATIVE CONTEXT ctxt_category_GQI (
   category integer,
   BINDING KEY (category)
      REFERENCES books(category)
) AS EQUIVALENCE WITH UNLISTED INCLUDED;

SET ACTIVE CONTEXT AS ctxt_category_GQI;
```

In response to Q_u, the actual query that is executed is given below. Rows in books, whose category exists as any of the contextual values of this context, are eliminated from the answer set.

```
SELECT T.*
FROM books T
WHERE T.category NOT IN (
          SELECT C1.category
          FROM ctxt_category_GQI C1)
          WHERE T.instock;
```

The basic contexts, which are not composed from other contexts, reflect in the actual executed query according to Table 2. This table shows whether an ORDER BY clause is necessary, and which type of join we need according to the context properties. We use the same symbols of the context classification as in Section 2 ([G] for negative context, [S] for positive context, etc.).

Next, we compose complex contexts from the above basic contexts. We start with the following context.

Context 5: *The user prefers books published in 2005, and then those published in 2006 before all other books. For the books that are similarly ranked, the user prefers hardcover books over books with paperback cover.*

This context may be viewed as the composition of ctxt_year_STI and ctxt_cover_STX. Therefore, we do not need to define a new

context. Conversely, we just need to set the active context appropriately to reflect to the desired context.

```
SET ACTIVE CONTEXT FOR user1
AS ctxt_year_STI, ctxt_cover_STX
WITH RANKING ORDER ctxt_year_STI, ctxt_cover_STX;
```

As a result of this combined context, queries to select tuples from books will work as if the query below was executed. First, the books in stock will be sorted based on the rank of the years, and then in case of ties, the cover type will be considered. This semantics is given by the following query rewrite.

```
SELECT T.*
FROM books T
   LEFT OUTER JOIN ctx_year_STI C1
      ON (T.year = C1.year
            AND C1.user_name = CURRENT_USER)
   INNER JOIN ctx_cover_STX C2
      ON (T.cover = C2.cover
            AND C2.user_name = CURRENT_USER)
WHERE T.stock
ORDER BY C1.rank, C2.rank;
```

Context Class	ORDER BY	Join Operation
GQN	X	NOT IN
SQN	X	LEFT OUTER JOIN
SQX	X	INNER JOIN
STN	√	LEFT OUTER JOIN
STX	√	INNER JOIN
SPN	√	LEFT OUTER JOIN
SPX	√	INNER JOIN

Table 2: The type of join used for each context class combination.

5. CHAMELEON PROOF-OF-CONCEPT INSTANTIATIONS

In this section, we illustrate how one can instantiate and realize specialized database servers using Chameleon. We begin with the first case study: privacy-aware databases. Then, we present spatial databases as our second case study. Finally, we conclude with two case studies that illustrate the ideas of context composition.

5.1 Realizing a Privacy Database Server

In this section, we show how we can limit disclosure, as what happens in Hippocratic Databases, using context awareness in Chameleon. In Table 3, we use the same patient table used in [33]. This table contains patient personal information.

Consider a healthcare facility that owns this data. Whenever a patient is admitted to the facility, he/she has to sign a privacy policy. The privacy policy specifies which information is to be released to which recipient. Moreover, the policy also specifies for which purposes the information is to be released. On an opt-in basis, the healthcare facility also allows patients to choose if they want any of their personal information to be released to other recipients. For instance, a nurse who is treating a patient is allowed to see the patient's name, age, and phone, but is not allowed to see his/her address for any reason. The patient may opt-in and choose that only his/her age is to be released to charity for solicitation.

pid	name	age	address	phone
1	Alice Adams	10	1 April Ave.	111-1111
2	Bob Blaney	20	2 Brooks Blvd.	222-2222
3	Carl Carson	30	3 Cricket Ct.	333-3333
4	David Daniels	40	4 Dogwood Dr.	444-4444

Table 3: The Patients Table.

Beside limited disclosure, limited retention is also modeled using context awareness. For simplicity, and without loss of generality, we assume that patient data is to be retained for 90 days only. By the end of this period, the patient data should have fulfilled the purposes for which the data has been collected. After this period, different recipients cannot retrieve the data.

It is important to make it clear that the patients in this context are the objects. Object contexts are the contexts of the patients. Moreover, users are those that use an application at the healthcare facility to retrieve patients' data. To model the above example of limiting the disclosure and retention of patients' data in Chameleon, we define the object contexts patient_privacy_pref and policy_signature as follows.

```
CREATE OBJECT CONTEXT patient_privacy_pref (
      recipient varchar(30),     purpose varchar(30),
      pid integer,               pid_pref boolean,
      name_pref boolean,         age_pref boolean,
      address_pref boolean,      phone_pref boolean,
   BINDING KEY(pid) REFERENCES patient(pid));

CREATE OBJECT CONTEXT policy_signature (
      pid integer,               sign_date date,
   BINDING KEY(pid) REFERENCES patient(pid));
```

Let the object context patient_privacy_pref contain the contextual data in Table 4. The following user context enforces the limited disclosure and limited retention of patients' data. Table 5 gives the context of three users. If the three users execute the query "SELECT * FROM patient;", they retrieve the data in Table 6.

```
CREATE POSITIVE CONTEXT identity_activity (
   job varchar(30),           activity varchar(30),
BINDING KEY(job, activity) REFERENCES
   patient_privacy_pref(recipient, purpose)
SUBSTITUTE patient(pid)
   WITH (CASE WHEN patient_privacy_pref.pid_pref
         AND today() <= policy_signature.sign_date + 90
         THEN patient.pid ELSE NULL)
SUBSTITUTE patient(name)
   WITH (CASE WHEN patient_privacy_pref.pid_pref
         AND today() <= policy_signature.sign_date + 90
         THEN patient.name ELSE NULL)
...
) AS EQUIVALENCE WITH UNLISTED EXCLUDED;
```

recipient	purpose	pid	pid_pref	name_pref	age_pref	address_pref	phone_pref
charity	solicitation	1	√	√	√	√	√
nurse	treatment	1	√	√	√	X	√
account clerk	billing	1	√	√	X	√	√
charity	solicitation	2	X	X	X	X	X
nurse	treatment	2	√	√	√	X	√
account clerk	billing	2	√	√	X	√	√
charity	solicitation	3	√	X	X	√	√
nurse	treatment	3	√	√	√	X	√
account clerk	billing	3	√	√	X	√	√
charity	solicitation	4	√	√	X	X	X
nurse	treatment	4	√	√	√	X	√
account clerk	billing	4	√	√	X	√	√

Table 4: The patient_privacy_pref object context.

user_name	job	activity
user1	charity	solicitation
user2	nurse	treatment
user3	account clerk	billing

Table 5: identity_activity contextual values

	pid	name	age	address	phone
u1	1	Alice Adams	10	1 April Ave.	111-1111
	3			3 Cricket Ct.	333-3333
	4	David Daniels			
u2	1	Alice Adams	10		111-1111
	2	Bob Blaney	20		222-2222
	3	Carl Carson	30		333-3333
	4	David Daniels	40		444-4444
u3	1	Alice Adams		1 April Ave.	111-1111
	2	Bob Blaney		2 Brooks Blvd.	222-2222
	3	Carl Carson		3 Cricket Ct.	333-3333
	4	David Daniels		4 Dogwood Dr.	444-4444

Table 6: Result of "SELECT * FROM patient;" for all users u1, u2, and u3.

5.2 Realizing a Spatial Database Server

Spatial databases are optimized to store and query data related to objects in space. This type of databases has more complex geometrical data types, e.g., points, lines, and rectangles.

Consider a real-estate database containing information about houses. The houses table has the following schema: houses (id, bedrooms, price, city). An application developer is interested in providing some spatial queries to this database, but has no privileges to add the location of the house to this table. An object context is created to add the location of houses.

5.2.1 Range Queries

Let the user context be the willingness to buy a house in certain regions. Hence, a user context is created in Chameleon to declare that only houses contained in relevant regions are to be returned.

The definitions of the object and user contexts, house_loc and houses_in_region, respectively, are given below. The function "contained" retrieves any house with location (x, y) that exist with the rectangular region (x1, y1, x2, y2). The binding between object and user contexts is through the scalar function "contained" that retrieves only the database objects within the query issuer's range context. Notice that there is no prioritization for the objects within the range, and hence the EQUIVALENCE keyword specifies the lack of any ordering.

```
CREATE OBJECT CONTEXT house_loc (
        id integer,
        x integer,          y integer,
  PRIMARY KEY(id),
  BINDING KEY id REFERENCES  houses(id));

CREATE POSITIVE CONTEXT houses_in_region (
        x1 integer,         y1 integer,
        x2 integer,         y2 integer,
  BINDING KEY() REFERENCES house_loc
  WITH contained (house_loc.x, house_loc.y, x1, y1, x2, y2)
) AS EQUIVALENCE WITH UNLISTED EXCLUDED;
```

5.2.2 Nearest Neighbor Queries

Another class of queries in spatial databases is the nearest neighbor query. In this class, the user wants to retrieve the object that is nearest to a pivot location. An extension to this class of queries is the k nearest-neighbors query. The answer of this query is the k objects that are nearest to the pivot location. In the real estate database, a user willing to retrieve the houses listed by proximity to a point may declare her context as follows:

```
CREATE POSITIVE CONTEXT nearby_houses (
        x integer,          y integer,
  BINDING KEY() REFERENCES house_loc
  WITH true
 ) AS TOTAL ORDER USING
        dist(x, y, house_loc.x, house_loc.y)
WITH UNLISTED EXCLUDED;
```

Notice that the clause WITH UNLISTED EXCLUDED can be omitted since all the houses are totally ordered based on distance.

The equivalent SQL query with the awareness of this context would be:

```
SELECT T.*
FROM houses T
  INNER JOIN house_loc OC1
        ON(T.id = OC1.id),
  nearby_houses C2
ORDER BY dist(C2.x, C2.y, OC1.x, OC1.y)
```

Notice that for finding nearest-neighbors to a user's location, the binding between the database objects and the user's focal point is via a total order based on a scalar function "distance".

5.3 Combining Contexts

5.3.1 Skylines

Skyline queries emerge in spatial databases. Assume that user2 wants to buy a house that is close to his work in downtown and that is also cheap (or at least reasonable) in price. Since it is not easy to combine such preferences in a ranking expression, user2 decides to select from the skyline houses.

Such context is defined as the composition of several contexts, namely houses_in_region, nearby_houses, and the context price already in the houses table. The first context will include a bounding box representing downtown. The second and third contexts will be used to compute the skyline. This composition is instantiated by setting the active context as follows:

```
SET ACTIVE CONTEXT FOR user2
   AS houses_in_region, nearby_houses
WITH SKYLINE OF nearby_houses.rank MIN,
                houses.price MIN;
```

Notice that after defining the houses in certain areas, and then defining the closeness to pivot points, the SET ACTIVE CONTEXTS combines both of these contexts (the range context and the nearest-neighbors context) together along with an object context (price) that is part of the house relation to get the SKYLINE of distance and price. This illustrates a more complex usage of contexts to answer conjunctions of spatial predicates.

5.3.2 Location-aware Privacy

Consider an application scenario where a query issuer, e.g., a doctor, may be allowed to access a database object's record, e.g., patient's record, only when the doctor is in the hospital premises. Otherwise, the doctor is not allowed to access the records.

In order to realize a location-aware privacy database server, we make use of the two contexts patient_privacy_pref and identity_activity that we define in Section 5.1 to realize the privacy context. For the location context, we make use of the two simple object and user contexts (valid_location and current_location, respectively). In this example, the location is modeled by a string value that gives a high-level description of the user's or the object's location (in contrast to physical coordinate locations). The mapping from the physical location to the named location is skipped here for simplicity.

```
CREATE OBJECT CONTEXT valid_location (
   pid integer,
   location varchar(30),
   BINDING KEY (pid) REFERENCES patient(pid) );

CREATE POSITIVE CONTEXT current_location (
    location varchar(30),
    BINDING KEY (location)
        REFERENCES valid_location (location)
) AS EQUIVALENCE WITH UNLISTED EXCLUDED;

SET ACTIVE CONTEXT identity_activity, current_location;
```

Notice that only the user contexts are listed since the binding activates the corresponding object contexts.

5.3.3 Location-aware and Time-aware Privacy

The example below gives a more complex context composition of the identity, location, and time contexts for both the database objects and the query issuers to realize a database server that provides both location-aware and time-aware privacy. This server would be useful in guaranteeing that, for example, a doctor may be allowed to access a patient's record only when the doctor is in the hospital but not after the hospital's regular hours.

We make use of the contexts patient_privacy_pref and identity_activity (defined in Section 5.1) to realize the privacy context, the contexts valid_location and current_location (defined in Section 5.3.2) to realize the location context, and the temporal contexts valid_time, current_time_not_expired, and current_valid_time, defined below.

```
CREATE OBJECT CONTEXT policy_signature (
   pid integer,
   sign_date date,        expire_date date,
   BINDING KEY(pid) REFERENCES patient(pid) );

CREATE OBJECT CONTEXT valid_time (
   pid integer,
   from_time date,        to_time date,
   BINDING KEY(pid) REFERENCES patient(pid) );

CREATE POSITIVE CONTEXT current_time_not_expired (
  BINDING KEY() REFERENCES patient
     WITH today() >= policy_signature.sign_date
     AND today() <= policy_signature.expire_date)

CREATE POSITIVE CONTEXT current_time_valid (
  BINDING KEY() REFERENCES patient
     WITH now() >= valid_time.from_time
     AND now() <= valid_time.to_time)

SET ACTIVE CONTEXT identity_activity, current_location,
current_time_not_expired, current_time_valid;
```

Notice that the user context current_time_not_expired provides limited retention, i.e., that data is made available only for the duration agreed upon by the data owner.

6. RELATED WORK

There have been several definitions of context and context-awareness (e.g., see [4, 6, 7, 17, 25, 40, 43, 44]). Most of these definitions define the context in terms of examples with special emphasis on the location context. Similarly, there have been several definitions of context-aware applications that include various synonyms, e.g., adaptive applications [44], reactive applications [16], responsive applications [19], situated applications [25], contented-sensitive applications [42], and environment directed applications [21]. In this paper, we adhere with the most formal definitions given in [17]. Recently, there has been interest in adding the context-awareness to relational database systems and query processors (e.g., see [30, 46]). However, the main focus is either on the modeling of the context

information and how to integrate it into the query definition, or on very specific examples that consider only one type of context. None of the previous work have discussed or proposed a full-fledge realization of context-awareness inside a DBMS.

There has been several work for presenting preferences in terms of relational calculus, first order logic, and query languages (e.g., see [14, 28, 32, 50]). In terms of query processing, there are two extremes for preference-aware queries, namely, top-k and skyline queries. Top-k queries have been well studied in various fields (e.g., [9, 12, 20, 38]). Also, there have been numerous algorithms for embedding top-k queries into database operators (e.g., see [8, 13, 22, 26, 34]). On the other hand, the term skyline queries has been coined in the database literature [5] to refer to the secondary storage version of the maximal vector set problem [31, 36]. Due to its practicality, various versions of skyline queries have been studied in the literature, e.g., sorted data [15], partially-ordered domains [10], high-dimensional data (e.g., [11, 41, 49, 52, 53]), progressive and online computations (e.g., [29, 39, 47]), sliding window [35, 48], continuous skyline computations [24, 37, 51], mobile ad-hoc networks [23], spatial skylines [45], and data mining [27]. Unlike the case for top-k queries, there is no previous work in integrating skyline queries at the core of query operators or database systems.

7. RESEARCH CHALLENGES AND CONCLUDING REMARKS

A working demonstration of the Chameleon context-aware database management system is currently available based on extensions to PostgreSQL. In the resulting context-aware DBMS, Chameleon, we also implement several operators to combine multiple contexts, mainly, the SkylineJoin, the RankJoin, and the FilterMark operators. Figure 8 illustrates the components we modified in PostgreSQL to realize Chameleon.

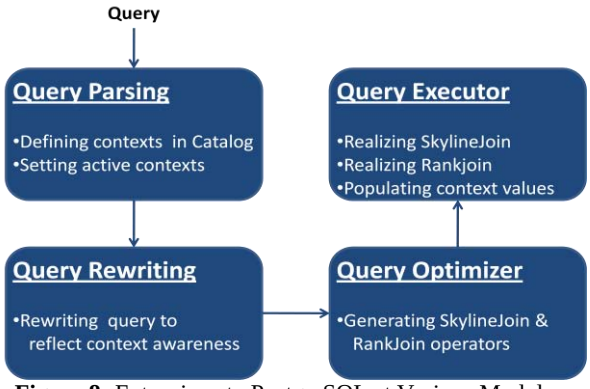

Figure 8: Extensions to PostgreSQL at Various Modules.

Based on this prototype of Chameleon, the authors have identified the following research challenges:

Performance: In the authors' opinion, the introduction of context-aware database management systems as outlined in this paper (and as depicted in Figure 1) is a profound and an important step. The declarative approach in defining the queries in relational databases (in contrast to the procedural approach in network databases) was one of the main factors that made the relational model prevail. Analogously, the declarative approach in defining contexts in context-aware database management systems can have a strong impact. Over thirty five years of efficient implementation and tuning has made the relational model overcome the efficiency hurdle. Similarly, efficient realization is the main challenge in context-aware database management systems. Efficient realization and execution are the authors' main focus for future work.

Dynamic Contexts: Another important challenge is that of dynamic contexts. So far, what Chameleon offers is static contexts. In many application scenarios, changes take place in the contexts, e.g., some active contexts may become inactive, inactive ones may become active, or new contexts get introduced. Another form of change is that the contextual values themselves within a context may change, e.g., the surrounding temperature may change or the location of a moving object may change, etc. These changes may affect the query being executed. This is similar in spirit to mid-query reoptimization [54]. However, the difference is that when the contexts change, the system may need to augment the query being executed by additional predicates that reflect that change in contexts.

Expressiveness and Completeness: Finally, issues related to the expressiveness and completeness of the context-aware model presented in this paper need to be studied.

8. REFERENCES

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *VLDB*, Hong Kong, China, August 2002.

[2] J. Padma, Y.N. Silva, M.U. Arshad, W.G. Aref: Hippocratic PostgreSQL. In ICDE, Shanghai, (Mar. 2009) 1555-1558.

[3] H.G. Elmongui, W.G. Aref, and M. Mokbel. Chameleon: Context-Awareness inside DBMS. In ICDE, Shanghai, (Mar. 2009) 1335-1338.

[4] Merriam-Webster Dictionary. http://www.m-w.com/.

[5] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In ICDE, 2001.

[6] P. Brown. The Stick-e document: a framework for creating context-aware applications. Electronic Publishing, 8(2&3), 1996.

[7] P. J. Brown, J. D. Bovey, and X. Chen. Context-aware Applications: From the laboratory to the marketplace. IEEE Personal Communications, 4(5), 1997.

[8] N. Bruno, S. Chaudhuri, and L. Gravano. Top-k Selection Queries over Relational Databases: Mapping Strategies and Performance Evaluation. TODS, 27(2), 2002.

[9] N. Bruno, L. Gravano, and A. Marian. Evaluating Top-k Queries over Web-Accessible Databases. In ICDE, 2002.

[10] C.Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified Computation of Skylines with Partially-Ordered Domains. In SIGMOD, 2005.

[11] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. Finding k-Dominant Skylines in High Dimensional Space. In SIGMOD, 2006.

[12] K. C.-C. Chang and S. won Hwang. Minimal Probing: Supporting Expensive Predicates for Top-k Queries. In SIGMOD, 2002.

[13] Y.-C. Chang, L. D. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith. The Onion Technique: Indexing for Linear Optimization Queries. In SIGMOD, 2000.

[14] J. Chomicki. Preference Formulas in Relational Queries. TODS, 28(4), 2003.

[15] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with Presorting. In ICDE, 2003.

[16] J. R. Cooperstock, K. Tanikoshi, G. Beirne, T. Narine, and W. Buxton. Evolution of a Reactive Environment. In Proceeding of the International Conference on Human Factors in Computing Systems, CHI, 1995.

[17] A.K. Dey and G.D. Abowd. Towards a better understanding of context and context-awareness. In Workshop on the What, Who, Where, When, and How of Context-Awareness, CHI, 2000.

[18] H.G. Elmongui and W.G. Aref. Skyline-Aware Join Operator. Tech. Rep. CSD TR 08-007, Purdue Univ., 2008.

[19] S. Elrod, G. Hall, R. Costanza, M. Dixon, and J. des Rivières. Responsive Office Environments. Communications of ACM, 36(7), 1993.

[20] R. Fagin, R. Kumar, and D. Sivakumar. Comparing Top k Lists. SIAM Journal on Discrete Mathematics, 17(1), 2003.

[21] S. Fickas, G. Kortuem, and Z. Segall. Software Organization for Dynamic and Adaptable Wearable Systems. In International Symposium on Wearable Computers, 1997.

[22] V. Hristidis, N. Koudas, and Y. Papakonstantinou. PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries. In SIGMOD, 2001.

[23] Z. Huang, C. S. Jensen, H. Lu, and B. C. Ooi. Skyline Queries Against Mobile Lightweight Devices in MANETs. In ICDE, 2006.

[24] Z. Huang, H. Lu, B.C. Ooi, and A. K. Tung. Continuous Skyline Queries for Moving Objects. TKDE, 18(12), 2006.

[25] R. Hull, P. Neaves, and J. Bedford-Roberts. Towards Situated Computing. In International Symposium on Wearable Computers, 1997.

[26] I.F. Ilyas, W.G. Aref, A.K. Elmagarmid, H.G. Elmongui, R. Shah, and J.S. Vitter. Adaptive Rank-Aware Query Optimization in Relational Databases. TODS, 31(4), 2006.

[27] W. Jin, J. Han, and M. Ester. Mining Thick Skylines over Large Databases. In PKDD, 2004.

[28] W. Kießling. Foundations of Preferences in Database Systems. In VLDB, 2002.

[29] D. Kossmann, F. Ramsak, and S. Rost. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In VLDB, 2002.

[30] G. Koutrika and Y. E. Ioannidis. Personalized Queries under a Generalized Preference Model. In ICDE, 2005.

[31] H. T. Kung, F. Luccio, and F. P. Preparata. On Finding the Maxima of a Set of Vectors. Journal of ACM, 22(4), 1975.

[32] M. Lacroix and P. Lavency. Preferences: Putting More Knowledge into Queries. In VLDB, 1987.

[33] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xuy, and D. DeWitt. Limiting disclosure in Hippocratic databases. In VLDB, 2004.

[34] C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song. RankSQL: Query Algebra and Optimization for Relational Top-k Queries. In SIGMOD, 2005.

[35] X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the Sky: Efficient Skyline Computation over Sliding Windows. In ICDE, 2005.

[36] J. Matousek. Computing Dominances in E^n. Information Processing Letters, 38(5), 1991.

[37] M. D. Morse, J. M. Patel, and W. I. Grosky. Efficient Continuous Skyline Computation. In ICDE, 2006.

[38] A. Natsev, Y.-C. Chang, J. R. Smith, C.-S. Li, and J. S. Vitter. Supporting Incremental Join Queries on Ranked Inputs. In VLDB, 2001.

[39] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. TODS, 30(1), 2005.

[40] J. Pascoe. Adding Generic Contextual Capabilities to Wearable Computers. In International Symposium on Wearable Computers, 1998.

[41] J. Pei, W. Jin, M. Ester, and Y. Tao. Catching the Best Views of Skyline: A Semantic Approach Based on Decisive Subspaces. In VLDB, 2005.

[42] J. Rekimoto, Y. Ayatsuka, and K. Hayashi. Augment-able Reality: Situated Communication Through Physical and Digital Spaces. In Intl. Symp. on Wearable Computers, 1998.

[43] T. Rodden, K. Chervest, N. Davies, and A. Dix. Exploiting Context in HCI design for Mobile Systems. In HCI, 1998.

[44] B.N. Schilit and M.M. Theimer. Disseminating Active Map Information to Mobile Hosts. IEEE Network, 8(5), 1994.

[45] M. Sharifzadeh and C. Shahabi. The Spatial Skyline Queries. In VLDB, 2006.

[46] K. Stefanidis, E. Pitoura, and P. Vassiliadis. Adding Context to Preferences. In ICDE, 2007.

[47] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient Progressive Skyline Computation. In VLDB, 2001.

[48] Y. Tao and D. Papadias. Maintaining Sliding Window Skylines on Data Streams. TKDE, 18(2), 2006.

[49] Y. Tao, X. Xiao, and J. Pei. SUBSKY: Efficient Computation of Skylines in Subspaces. In ICDE, 2006.

[50] G.K. Werner Kießling. Preference SQL - Design, Implementation, Experiences. In VLDB, 2002.

[51] T. Xia and D. Zhang. Refreshing the Sky: The Compressed Skycube with Efficient Support for Frequent Updates. In SIGMOD, 2006.

[52] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. X. Yu, and Q. Zhang. Efficient Computation of the Skyline Cube. In VLDB, 2005.

[53] Z. Zhang, X. Guo, H. Lu, A. K. H. Tung, and N. Wang. Discovering Strong Skyline Points in High Dimensional Spaces. In CIKM, 2005.

[54] N. Kabra, D.J. DeWitt: Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans. SIGMOD Conference 1998: 106-117.