The Development of Ordered SQL Packages to Support Data Warehousing

WILFRED NG
Hong Kong University of Science and Technology, Hong Kong

MARK LEVENE
University of London, UK

Data warehousing is a corporate strategy that needs to integrate information from several sources of separately developed Database Management Systems (DBMSs). A future DBMS of a data warehouse should provide adequate facilities to manage a wide range of information arising from such integration. We propose that the capabilities of database languages should be enhanced to manipulate user-defined data orderings, since business queries in an enterprise usually involve order. We extend the relational model to incorporate partial orderings into data domains and describe the ordered relational model. We have already defined and implemented a minimal extension of SQL, called OSQL, which allows querying over ordered relational databases. One of the important facilities provided by OSQL is that it allows users to capture the underlying semantics of the ordering of the data for a given application. Herein we demonstrate that OSQL aided with a package discipline can be an effective means to manage the interrelated operations and the underlying data domains of a wide range of advanced applications that are vital in data warehousing, such as temporal, incomplete and fuzzy information. We present the details of the generic operations arising from these applications in the form of three OSQL packages called: OSQL_TIME, OSQL_INCOMP and OSQL_FUZZY.

Data warehousing is a corporate strategy that addresses a broad range of decision support requirements such as querying information over its underlying databases and managing ordered data for the purpose of analysis. One of the main characteristics of data warehousing is that in order to build its foundation, it should consist of integrated data from several sources of separately developed information systems. The transmission of data relies on the network system which connects all these information systems. As a result, the integrated database has the following important features:

- **It involves huge amounts of historical data.**

  Data warehouse is described as a “subject-oriented, integrated, non-volatile, time variant” collection of data which is intended to support management decisions (Inmon, 1996). It is widely recognised that the underlying database in a data warehouse should capture transactions and snapshots in time in an efficient manner in order to carry out the activities of market forecast and strategic planning (McCabe & Grossman, 1996).

- **It is usually incomplete.**

  This is due to two main reasons. First, some sources of the databases may be incomplete in order to protect sensitive data or to improve the speed of the process of data downloading via a network. Second, it has been observed in Libkin (1995) that even if each source of the database is complete, the integrated database may still not be complete. Hence, incompleteness may show up in the integrated database or in the answer to users’ queries.

- **It is mainly used for decision support in an enterprise.**

  However, many management professionals may not necessarily have good knowledge about the technical aspects of a data warehouse. As a result, their queries over the database are sometimes fuzzy in nature due to the ambiguity.
of natural languages. For example, they may ask to find the “best performed” shares in the Hong Kong stock market this month in order to carry out some share trading activities.

Many database researchers have recently recognised that ordering is inherent to the underlying structure of data in many database applications (Maier & Vance, 1993; Libkin, 1995; Buneman et al., 1997) including temporal information (Tansel et al., 1993), incomplete information (Codd, 1986) and fuzzy information (Buckles & Petry, 1982). However, current relational Database Management Systems (DBMSs) still confine the ordering of elements in data domains to only a few kinds of built-in orderings. SQL2 (or simply SQL) (Date, 1997), for instance, supports three kinds of orderings considered to be essential in practical utilisation: the alphabetical ordering over the domain of strings, the numerical ordering over the domain of numbers and the chronological ordering over the domain of dates (Date, 1990). Let us call these ordered domains system domains or alternatively, domains with system ordering.

With the advent of the Internet technology, there is strong evidence that the limited support for ordering provided by current relational DBMSs is inadequate for future commercial applications. For example, a large proportion of the useful business information available in global Web sites is available only in hypermedia format. Hypermedia information normally consists of a very large amount of image data and thus resolution is an effective means to manage the size of data domain element. We illustrate this concept with the following simplified multi-resolution domain: {‘Null’ < ’Black and white icon’ < ’Black and white raster’ < ’8-bit Colour raster’ < ’24-bit Colour raster’ }. This domain consists of five distinct levels of resolution and thus the users can select the appropriate level to save the transmission time for downloading a hypermedia document. However, the semantics of RESOLUTION_LEVEL cannot be captured by any one of the system orderings.

In order to alleviate the above-mentioned problems, we have extended SQL to Ordered SQL (OSQL) by providing the facility of user-defined orderings over data domains (Ng & Levene, 1997), which we refer to as semantic orderings. Queries in OSQL are formulated in essentially the same way as using standard SQL. We demonstrate this mode of querying with the following example, which shows how OSQL simplifies the specification of certain queries which might be useful in business decisions. We note that the following queries are not easy to formulate in SQL due to the fact that they must involve non-trivial use of aggregate functions and nesting (see Sections 25.1 and 26 in Celko (1995) and Section 9 in Pascal (2000)).

**Example 1** In this example we assume that the attributes in their respective relation schemas are linearly ordered.

1. Get the third and sixth lowest share prices from a stock market.

\[
(Q_1) \text{SELECT} (\text{SHARE\_PRICE}) (3,6) \text{FROM} \text{STOCK\_MARKET}.
\]

2. Get the names of exactly five participating banks from a syndicated loan record.

\[
(Q_2) \text{SELECT} (\text{BANK\_NAME}) (1..5) \text{FROM} \text{SYNDICATED\_LOAN}.
\]

3. Get the names of all bosses of John.

\[
(Q_3) \text{SELECT} (\text{EMPLOYEE\_NAME}) (_,) \text{FROM} \text{EMPLOYEE\_TABLE} \text{WHERE} \text{EMPLOYEE\_NAME} > 'John' \text{WITHIN} \text{EMP\_RANK}.
\]

Although we have not yet formally introduced OSQL, the meaning of the above statements is quite easy to understand, assuming that the reader has some knowledge of standard SQL. For instance, the clause (3,6) in the query (Q1) means that the third and sixth tuples, according to the order of SHARE_PRICE, are output and the clause (1..5) in the query (Q2) means that the first to fifth tuples, according to the order of BANK_NAME, are output. The keyword WITHIN in the query (Q3) specifies that the comparison EMPLOYEE\_NAME > 'John' is interpreted according to semantic ordering of the domain EMP\_RANK.

The usual way to tackle the above problems is to use a programming approach such as embedded SQL. However, as most data warehouses are built upon a client-server architecture, the programming approach has to pay the performance penalty in the data extraction process; if there are too many calls from the programming level to the relational level. In this respect, OSQL offers the advantage that it can help to relieve the burden of the bandwidth of a network system and the loads of client processes, if such kinds of queries can be performed in the database server instead of the client platform.

Herein we investigate the introduction of a package discipline into OSQL, which allows us to modularise a collection of generic operations on an ordered data domain. These operations can then be called from within OSQL whenever the package they belong to is loaded into the system. For example, the OSQL statement (Q4) uses the function SNAPSHOT provided by the OSQL package OSQL\_TIME, which returns the prices of shares of the temporal relation STOCK\_MARKET in 1990.

\[
(Q_4) \text{SELECT} (\text{SHARE\_PRICE}) (_,) \text{FROM} \text{SNAPSHOT(\text{STOCK\_MARKET}, 1990)}.
\]

The package discipline makes it easier to formulate queries relating to the underlying ordered domains of the package and allows us to extend OSQL with powerful operations, which enhance its applicability and expressiveness. We demonstrate that OSQL aided with a package discipline is extremely powerful and has a very wide range of applicability. In particular, we demonstrate that OSQL is very useful in managing the three advanced database applications described in Part 2 of this paper.