Preface

INTRODUCTION

A data warehouse stores a massive amount of data integrated from data sources, which can reflect the reality of the real world for reporting and analysis purposes, and its tools can be used to discover from the data, the trend or potential direction of developments. Therefore, in application areas such as commerce, health care and monitoring of global changes in the environment and biodiversity, data warehouses are used extensively for the purposes of inquiries, decision making and data mining.

The purpose of this book is to present and disseminate the latest developments in data warehousing. The focus is on the most recent research and discoveries, in particular several interesting issues and trends that have emerged in the last few years. This chapter provides introductory background information leading into the topic of Data Warehouse. It consists of five sections. Figure 1 is a flowchart showing the connection among these sections.

Traditionally, a data warehouse is mainly used as a repository for a massive amount of data. It focuses on the data storage and reflects only the historical static data. However, in the dynamic business environment typical of many organizations, a data warehouse plays an important role in both strategy and tactical decision-making, therefore, users require that the requested data be the most recent, or the ‘freshest’. This process includes extracting the new transaction data from the database, transforming, aggregating and then updating it into a data warehouse. Moreover, users usually expect prompt responses to their queries. These fundamental requirements present a challenge to the data warehouse due to the high loads and update complexity of data warehouse refresh mechanisms. Therefore, in the following section, we will present an overview of a data warehouse which includes its components, the

Figure 1. Flowchart
categories of its refresh mechanism and the slowly changing dimension which will lead us to the temporal support in data warehouses. A geospatial domain has been widely developed by the availability of wireless networks and portable devices, so the complexity of the decision making with the spatial and temporal dimension involved will greatly increase. In recent years, it has become increasingly important to extend the data warehouse with spatial and temporal features. Finally, Materialized Views are discussed, since the data warehouse must be increasingly queried effectively for decision making purposes, and the use of materialized views is presumably the most effective and popular means of speeding up query processing.

**DATA WAREHOUSING: AN OVERVIEW**

W.H. Inmon characterized a Data Warehouse (DW) as a subject-oriented, integrated, non-volatile, time-variant collection of data in support of management’s decisions [Inmon, 1996]. A DW is an environment, not a product, in which the user can gain the current or historical data. These data being used for making decision or analyzing are usually difficult or impossible to process from the operational database. Therefore, data warehousing is a technology that aggregates data into the DW for complex analysis, and quick, efficient query response and decision-making. When the data are deposited into the data warehouse, the administrator or other users can easily gain access in order to manipulate the data or obtain useful data information about business performance by using a DW application tool such as online analytical processing (OLAP).

There are several major components of the architecture of a DW (see Figure 2): the source system, the data mart, transformation mechanism, the metadata repository, central data warehouse and end-user decision support tools for data analysis.

The most common data model is multidimensional modeling, in which star schema and snow flake schema are introduced in [Inmon, 1996]. The star schema is a technique used in modeling to map multidimensional decision support data into a relational database. It consists of “fact table” and a number of “dimensional tables”. A fact table (e.g., Sales facts) represents a specific business aspect or activity by numeric measurements (values). (e.g. analysis of sales, quantity or Amount representing analysis needs in numeric form.) Dimensions (e.g., Product, Time, Customer and Location) are qualifying characteristics that are used as measures from different analysis perspectives. Further, a dimension may include descriptive attributes (e.g., customer name or sex in the Customer dimension).

The dynamic business environment of many organizations requires high freshness of requested data. Therefore, with a built-up data warehouse it is very challenging to maintain the data warehouse. In [Kimball & Ross, 2002], authors analyze different designs of data warehouses based on different real-life issues. Multidimensional modeling allows designers to deal with complex problems in real life, in [Mazón et al., 2009], a survey provides us with an overall understanding and summary of issues regarding multidimensional modeling while dealing with the complex conceptual multidimensional structure. Most particularly in recent years, time and spatial dimensions have been widely investigated in the advanced development of data warehouses. Therefore, in the following section, we will discuss the data warehouse refresh mechanism, and the slowly changing dimension issues resulting from the process of refreshing DWs.
In respect to the refreshing of a data warehouse, as changes are made in the operational database, the DW must reflect these changes, which includes the changes made in fact and the slow changes made in dimensions. Since a DW is non-volatile, the refresh of a DW here means that data is uploaded into the DW, while the data that is already in the DW will never be deleted from the DW. This section of the chapter summarizes the categories of existing data warehouse refresh mechanisms and the issue of the slow changes made in dimensions.

In general, DW refresh mechanisms have gradually shifted from traditional DW to real-time DW mechanisms, as outlined in the following four categories:

- The first category is a traditional DW refresh mechanism which consists of traditional batch-based updates. The whole DW is rebuilt from scratch by running the schema, repopulating all the data from the current operational database with a timestamp. Based on the schedule of the business at that time, DW can be refreshed monthly, quarterly, even yearly [Mumick et al., 1997]. The benefit of this is that all the data in the DW remains consistent [Bin et al., 2002], because while batching the data into the DW, the users do not have ready access to the DW, and will not get the data analyzing that is not the current data in the DW [Thiele et al., 2009; Truban et al., 2008]. However, the full refresh mechanism cannot possibly meet users’ requirements when the size of the DW is greatly increasing; the data need to be refreshed more frequently and the batch windows for data acquisition need to be increasingly smaller.
- The second category is incremental batch refresh based on a timestamp, which means that instead of redoing the entire old and new data, it recalculates only the data updated since the last refresh,
and it will place only these new data into the DW. Compared with the naive way of rebuilding a DW, incrementally updating the DW is both time-saving and cost-effective. However, when a DW deals not only with strategic decision support but also with tactical decision support, timestamp refresh may potentially cause a considerable delay between the information in the DW and the reality in data sources.

- The third is the continuous feed refresh mechanism of a real-time DW. This is the most ideal refresh mechanism for an advanced DW. As the real-time DW is always refreshed [Truban et al., 2008], up-to-date information can always be obtained. This mechanism seems to be very tempting for the business world; however, a significant burden is imposed on the operational database side. One transaction may take a long time to convert and load data into the DW, which will seriously affect the daily transaction performance. Second, it may be unnecessary to refresh immediately after each change to the operational database.

Another one is the Operational Data Store (ODS), that is a type of database which is updated continuously throughout the course of business operations. It is similar to a small scale real-time DW, but it stores only very recent update information which is kept separate from the main DW [Truban et al., 2008]. This ODS technique is useful only when the user queries target the most recent information separately from the DW and therefore the queries can be directed toward the ODS only. However, when the query coverage includes both recent and historical data, then a merging technique has to be applied and this can be very costly.

- The fourth mechanism, an incremental continuous refresh mechanism, is based on investigating a combination of the second and third approaches. This mechanism aggregates only the new insertions and updates from the operational database, when the system accurately identifies that the DW needs to be continuously refreshed or just-in-time [Maurer et al., 2009].

In [Italiano et al., 2006], the authors argue that not all transactional data need to be immediately dealt with despite even a real-time decision making requirement. In the same DW, we should allow different time priorities, such as urgent, which should be in real-time, or just be in time based on the business characteristics. The DW data freshness time interval should always be driven by the business requirements, not the technology itself. It is a challenging and critical goal to enable a DW to provide multiple levels of freshness time intervals. In [Chen et al., 2010], a near real-time refresh DW can reflect the real world requirements by making the interval of the timestamp 'dynamic’ or flexible depending on factors such as impact from one update, the number of records affected, and the frequency of requests. It proves that a near real-time DW refresh can be a beneficial extension to the existing DW environment and shows that a near real-time DW could save significant operational costs.

**Slowly Changing Dimension (SCD)**

When the issue of updating a DW arises, usually the focus is on the changes relating to the attributes in the fact table, such as customer orders or product quantity; there is little concern about the changes needed to support dimension attributes, such as customer name or product description. Compared with changes to facts, the data changes in dimension are comparatively slow and static, so we call this *Slowly Changing Dimensions* (SCD). [Kimball, 1996] suggests three types of SCD [Ross & Kimball, 2005].
• **Type 1** is the most common one; when the data of a dimensional attribute changes, the system updates it directly (overwritten), without recording the old value. Generally, this seems to work. For instance, if the price of a product labeled P0002 is 40 dollars (e.g., in Figure 3), and then the next day the provider changes the price of P0002 to 60 dollars, then the price of P0002 is overwritten. And if a customer bought 10 Products P0002 several days earlier, and now wants to buy the same quantity of Product P0002 the customer will be faced with a different total amount due. Any user will find the pricing of this product confusing since we cannot track back to the old price of the product, because the DW stores only the latest value.

• **Type 2** is when the dimensional attribute changes. We insert a new row with a new surrogate primary key to record the changes, and both rows (old one and new one) will be included. At the same time, there are the most-recent-row flags, the row effective and expiration dates recorded for them. This type can retain all the changing information, but it will not be very appropriate for application to a large dimensional table whose attributes are semi-rapidly changing. In Figure 4, it is obvious that except for the price column changes, the data of the same product contains the same, and if the price of products is frequently changing in a large dimension table containing thousands of products, this method takes up a lot of space and is expensive. Furthermore, it is difficult to implement and query due to the changing nature of the surrogate key. It might be better to have another table to record the old values.

*Figure 3. Type 1 SCD Implementation*

<table>
<thead>
<tr>
<th>Surrogate Key</th>
<th>Product ID</th>
<th>Description</th>
<th>Color</th>
<th>Size</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001</td>
<td>P0001</td>
<td>Bag</td>
<td>Black</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>10002</td>
<td>P0002</td>
<td>Shoes</td>
<td>White</td>
<td>7</td>
<td>40–60</td>
</tr>
<tr>
<td>10003</td>
<td>P0003</td>
<td>Coat</td>
<td>Red</td>
<td>9</td>
<td>60</td>
</tr>
</tbody>
</table>

*Figure 4. Type 2 SCD Implementation*

<table>
<thead>
<tr>
<th>Surrogate Key</th>
<th>Product ID</th>
<th>Description</th>
<th>Color</th>
<th>Size</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001</td>
<td>P0001</td>
<td>Bag</td>
<td>Black</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>10002</td>
<td>P0002</td>
<td>Shoes</td>
<td>White</td>
<td>7</td>
<td>40</td>
</tr>
<tr>
<td>10003</td>
<td>P0003</td>
<td>Coat</td>
<td>Red</td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>10004</td>
<td>P0001</td>
<td>Bag</td>
<td>Black</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>10005</td>
<td>P0002</td>
<td>Shoes</td>
<td>White</td>
<td>7</td>
<td>30</td>
</tr>
</tbody>
</table>
• **Type 3** requires putting another attribute into the existing dimension row as either the previous value recorded or new current value. This is the least commonly required technique. Up to a certain point, people can refer only to the latest changes in the dimensional table, but are unable to obtain the historical records. In Figure 5, we can see that we can record only the current value and the previous one. So only partial historical data is recorded using this approach.

SCD is complicated, each type having its own problems, but it is still necessary to consider it when updating a DW. Based on SDC solution, [Nguyen et al., 2007] classify the incoming information into state-oriented data and event-oriented data, and they provide a *Comprehensive enhanced SCD solution* (CSCD) solution to retain the entire historical data of dimensions by combining type2 and type3. [Eder et al., 2004] provide two different approaches called ‘Grouping’ and ‘Fixing’ to detect the structure changes in dimensions; the first one performs faster and the second one can provide better results. [Malinowski & Zimányi, 2006; Malinowski & Zimányi, 2008] use object-relational nested table collection types to store the changes of attributes (e.g. Figure 6); Box 1 is a snapshot of object-relational nested table implementation. Note that we can also use array. However, since the number of changes can be huge, the nested table would be a more suitable implementation.

*Figure 5. Type 3 SCD Implementation*

![Type 3 SCD Implementation](image)

*Figure 6. Nested table collection type*

![Nested table collection type](image)
The solutions provided by SCD either cannot retain the whole history data, or the process is quite complicated. Therefore, based on the development of temporal databases, a temporal data warehouse can address this shortcoming.

TEMPORAL SUPPORT IN DATA WAREHOUSE

Due to dynamic changes of business activities, data consistency in the application of analysis is very important. Clients of the warehouse may be interested not only in fact measurement data, but also in the history showing how the dimension data has evolved. However, currently, the time dimension in data warehouses can track only the changes to measurement values in the fact table; it cannot represent the changes that occur in the dimension tables. These changes have been mentioned previously, and are termed ‘slowly changing dimensions’. The solutions of SCD either cannot retain the whole history data or the process is quite complicated.

In an attempt to address this problem, a Temporal Data Warehouse (TDW) is a new field of research that adopts the concept of temporal database and is combined with a data warehouse since a temporal database can manage and represent the data that vary over time. A survey on temporal data warehousing is presented in [Golfarelli et al., 2009].

Temporal Databases

A conventional database stores only the current data, and the historical data (updated, deleted) are overwritten by the current data. So the temporal meaning of data is buried. Realizing the importance of modeling time and handling historical data, temporal databases have been extensively investigated dur-

---

**Box 1. example code to create object-relational nested table collection types for price**

```sql
CREATE OR REPLACE TYPE IntervalType as OBJECT
(FromDate          DATE,
ToDate          DATE)/
CREATE OR REPLACE TYPE IntervalType_Table AS TABLE OF IntervalType/
CREATE OR REPLACE TYPE PriceType AS OBJECT
(Value          Number,
ValueInterval     IntervalType_Table)/
CREATE OR REPLACE TYPE PriceType_Table AS TABLE OF PriceType/
CREATE OR REPLACE TYPE ProductType AS OBJECT
(Product_ID     VARCHAR2(10),
Description     VARCHAR2(20),
Color               VARCHAR2(10),
Size               NUMBER,
Price               PriceType_Table)/
CREATE TABLE Product OF Price_Tab
(NESTED TABLE ValueInterval STORE AS Value_Tab);
```
ing the past two decades. Researching topics covered temporal models, query languages (e.g., [Chau et al., 2008]), and implementation techniques. A Temporal Database System is a system that represents and manages the data with timestamps being integral components (e.g., [Clifford & Tansel, 1993]). There are several ways to represent time in a temporal database [Dyreson et al., 1994; Clifford et al., 1997]:

- **Valid time (VT)** represents the time interval during which the fact is true in the model reality; this can be past, current or even future. It relies on the behavior of the reality, so that if an error is made in the reality, valid time can be corrected by altering the database. Valid time has a wide range of applications such as sales, renting etc. For example, if an employee quits his job on 15th Jan 2010, and his information is deleted from the company’s database on 31st Jan 2010, the employee’s valid time will be the time from when he began working for the company until 15th Jan 2010; not until 31st Jan 2010. If the finance department calculates his payment, it should be based on the valid time, even though his information was not deleted from the database until later.

- **Transaction time (TT)** records the time when the transaction occurred in the database. Transaction time cannot be altered since it is the timestamp captured by the system. TT does not exist without the database. It plays a very important role in the applications that place particular emphasis on traceability and accountability, such as auction, auditing, billing etc. So if we apply this to the example above, 31st Jan 2010 will be regarded as the transaction time when the employee’s information is deleted to show that the employee no longer works at the company.

- **Bitemporal time (BT)** is when data is associated with both valid time and transaction time. Bitemporal time can be used to more accurately review the reality and allow the retroactive and prospective changes. Therefore, in the example of the employee, the system will record both time 15th Jan 2010 and 31st Jan 2010 as BT.

Based on the time dimension(s) within the database, databases can be categorized into four types, as shown in Table 1.

**Time Temporality Type and Relationship**

The time dimension is ubiquitous and very important in the temporal data warehousing design. Much research work on the modeling of a temporal data warehouse has provided several temporal types to facilitate time manipulations (eg., [Dyreson et al., 1994; Ravat & Teste.,2000; Rizzi & Golfarelli, 2006;

<table>
<thead>
<tr>
<th>Type of Database</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snapshot Database</td>
<td>The state or an instance represents only the current contents and information in the database. If operations (insert/update/delete) are done to the database, the previous state or instance is thrown completely.</td>
</tr>
<tr>
<td>Rollback Database</td>
<td>The database supports the query of selecting a snapshot state by moving along the transaction time axis. The operation of selecting the snapshot is called ‘Rollback’. It records the history of database activities rather than the history of the real world.</td>
</tr>
<tr>
<td>History Database</td>
<td>The database time axis is changed to valid time, the data corresponds more closely to reality rather than the history of database updating operations.</td>
</tr>
<tr>
<td>Bitemporal Database</td>
<td>Both transaction time and valid time are recorded, so that the database is able to store not only the current data, but also the historical data and even future data expected to occur.</td>
</tr>
</tbody>
</table>
Some of temporality types are directly derived from the temporal database, such as valid time, transaction time. And other temporality types are widely accepted such as chronons, instant, interval, lifespan, and loading time.

**Chronon** is defined as a ‘non-decomposable time interval of some fixed, minimal duration’ [Dyreson et al., 1994]. This means a chronon is the finest unit in the dimension time. In reality, the time dimension is linear, but in data warehousing, we treat time as a discrete entity, so the time axis is defined as a series of chronons.

**Instant** is like an instance of chronons. It is used to describe when an event happened or will happen. For example, instant is used to record a patient’s time of death.

**Interval** implies a period duration which includes a set of chronons between two instant limits. For example, during 10/01/2010 and 31/01/2010, the patient stayed in the hospital. It includes a start time and an end time. And **duration** is just the length of an interval. There are thirteen possible ways in which an ordered pair of intervals can be related [Allen, 1983]. Figure 7 shows the 13 distinct relationships between two intervals X and Y with X being described in relation to Y. The right column can be viewed as the inverse relationship of the left column. For example, X before Y is the same as Y after X. It depends only on which time interval is regarded as the comparison object.

In [Rizzi & Golfarelli, 2006], the authors maintain that transaction time and valid time are not sufficient to ensure the correctness of meaningful historical analysis. Delayed registration should be considered as well. Here the delayed registration means the delay occurring in real life. For example, when students enroll in their university subjects, the validation date will be the day they pay their tuition fee, but there might be a delay in the bank transactions (sometimes even a week for international transactions). In [Malinowski & Zimányi, 2008], the authors defined lifespan (LS) or existence time and loading time. Lifespan implies the duration of an object’s existence. It can be the duration of the valid time of the related fact, or relationships between two objects, or a transaction time indicating the time when the object or relationship is currently in the database. Once data is inserted into a data warehouse, it will

---

**Figure 7. Time interval relations**

[Diagram showing the 13 distinct relationships between two intervals X and Y with X being described in relation to Y. The right column can be viewed as the inverse relationship of the left column. For example, X before Y is the same as Y after X. It depends only on which time interval is regarded as the comparison object.]
neither be modified nor deleted, so loading time is used to record when the data is loaded into the data warehouse since there is likely to be a delay between the data source and the data warehouse. These features are very helpful in the application of data analysis and decision-making.

[Larsen et al., 2006] presented an approach to deal with ill-known temporal data by supporting fuzzy time intervals. In [Winarko & Roddick, 2007], the authors propose an algorithm for discovering richer relative temporal association rules from interval-based data. [Moreno et al., 2010] specifically tackle season queries on a temporal multidimensional model for OLAP, and they propose a new operator to make the query for season simple and concise.

For many observations regarding time, temporal information interacts with it. An efficient exploitation of the time dimension may provide users with a more compact visualization of the temporal data and the temporal relationships that exist between the data and the time. We may be able to further explore the temporal data warehouse and gain more insights. Here, we present two different methods for analyzing time.

**Based on the Temporality of Time Type**

According to the temporality of the time, we can define the time as either Temporal or Permanent. Here the temporality of time is different from the temporality types we mentioned above.

**The temporal time point/interval** means the data associated with the time is temporal. When we insert/update/delete it with the temporal time, we know that when the time point/interval has expired, the data will no longer be valid. For example, in the supermarket, when we update one item’s price from 100 dollars to 50 dollars in a promotion for one week, we know that after one week, the cheaper price will expire.

**The permanent time point/interval** means the data associated with the time is permanent. When we insert/update/delete it with the permanent time, at that time we know the data is potentially valid forever. So, when one data associated with a permanent time interval is replaced by a data with temporal time, after the temporal time expires, the data with permanent time will continue. For example, after one week’s promotion, the item’s price will still be 100 dollars. Obviously, one data with permanent time can be replaced by another data with permanent time; when this happens, the previous one will no longer be permanent.

**Based on Regularity of Time**

We can divide the regularity of time into four categories:

1. **Regular Time Interval** means the different events carry on for the same length of duration, such as every week, every month, and the relevant data may change significantly based on this. For example, a 5% discount on one product for one week, compared with 50% discount on one product for one week, will affect the sales figure very differently.

2. **Irregular Time Interval** means the same data repeats for the different lengths of duration. Then, the relevant data may change significantly based on this. For example, a 20% discount on one product may last one day or one week, so the sales figure will be affected very differently.

3. **Regular Time Instant** means a different event happened at the same time instant; the relevant data may change significantly based on this. For example, if a car accident occurs at 8:00 a.m. on
a particular morning, the traffic would be totally different compared with 8.00 a.m. traffic on a morning when no accident occurs.

4. **Irregular Time Instant** means the same data is repeats for a different instant; the relevant data may change significantly based on this. For example, if a car accident occurs at midnight, the traffic situation will be significantly different from traffic conditions caused by an accident occurring at 8:00 a.m.

Based on Temporality Level

No matter what time temporal types are applied, they are all used to timestamp the temporal changes to the database. The temporality level of changes simply relates to two types of changes. One is a type of change that occurs as a whole member; we term this temporal instance level (for example, inserting or deleting an employer from a list of employers of a company). The other one is a type of change that occurs within the whole member; we term this temporal attribute level (for example, updating an employee’s address within the list of company employees).

Being able to record both types of changes is very important for analytical purposes. For example, the allocation of employees to different departments or the addition of more labor may influence the company’s budget.

Design for Temporal Data Warehouses

The conventional data warehouse systems can effectively manage the changes in the fact table, but not the data in dimension members. Many temporal data warehouse design models (eg., [Chamoni & Stock, 1999; Eder & Koncilia, 2001; Malinowski & Zimányi, 2008]) have been proposed to overcome these limitations. These designs have made a considerable contribution to the development of advanced data warehouse systems.

An object-oriented paradigm has been proven applicable to complex data modeling [Bukhres, 1995]. Furthermore, [Ravat & Teste, 2000] present an object-oriented data warehouse model to manage and present the temporal data. Their model is not based on a multidimensional model concept. They introduced the two concepts of temporal filter and archive filter to define the past states and the archive states. Some mapping functions are specified. But a query language based on temporal extension of OQL needs to be specified to handle their DW elements. However, the multidimensional model is accepted widely and applied in various areas.

Malinowski and Zimanyi, leading temporary data warehouse experts, have proposed a temporal extension for the multidimensional model. [Malinowski & Zimányi, 2008] refer to different temporality types supported by the model, describe different temporal level, hierarchies and measures needing to be considered, and finally, they provide the mapping of the constructs of the MultiDim model to the ER and OR models. They provide a formal basis which can lead to better understanding of issues, alternatives and techniques.

[Eder et al., 2002] adopt the COMET Metamodel for temporal warehouse. This model is able to represent and manage all temporary changes including data and structure changes for both fact and dimensions. Based on this model, [Eder et al, 2004; Eder & Wiggisser, 2008] focused on the data transformation, analyze the transformation functions’ impact and their efficient representation. They present
six transformation operations and elaborate on two different representations (matrix-based and graph-based) for them. Further optimization techniques are needed.

**SPATIAL SUPPORT IN DATA WAREHOUSE**

Worldwide globalization has significantly increased the complexity of the problems. The geo-spatial domain has been widely developed due to the availability of wireless networks and portable devices, so the complexity of the decision-making process, with the spatial and temporal dimension involved, will greatly increase. Therefore, our community requires an enhanced and more knowledgeable data management system to solve the complex problems in many areas including economic and social environments. DW plays a significant role in Decision Support Systems (DSS). If DSS can deal with both spatial and non-spatial dimensions and measures, the functionality of DSS will be widely enhanced. Within a spatial data warehouse model, it is natural to extend existing spatial data models with time. Since all these spatial data are constantly changing, a great deal of spatial data will exist between the spatial database and DW. Therefore, in the last years, extending data warehouse with spatial and temporal features has attracted the attention of the GIS and database communities (e.g. [Lopez et al., 2005; Rivest et al., 2005; Viqueira & Lorentzos, 2007]).

There are two features that the data warehouse house system intends to cover: one is the capability to manage a more accurate and complete analysis to the underlying data, process and events. The data does not represent only the current status, but also the past and future status; second, thanks to the availability of sensor networks and mobile devices (e.g. [Deligiannakis & Kotidis, 2006; Medeiros et al., 2010]), many business processes contain geo-spatial information, so there is increasing demand for a system which is able to include the spatial information [Bertino & Damiani, 2005]. A spatial database system (SDBS) is a fully-fledge database system with extra capabilities for handling spatial data [Güting, 1994]. Geographic Information System (GIS) is able to capture, manipulate, analyze and display all forms of spatial data. Moreover, it is extensively used for geographical applications such as choosing sites, targeting market segments, planning distribution networks, responding to emergencies, or re-drawing country boundaries. Therefore, another research area based on both solid techniques has appeared: Spatial Data Warehousing (SDW) [Bimonte et al., 2005]. Spatial temporal data warehouse (STDW) combines both temporal and spatial features, both being included in the data warehouse.

**Recent Work**

In recent years, much work has been conducted in the area of spatial temporary data warehousing. Several works (e.g. [Šaltenis & Jensen, 2002; Choi et al., 2006]) focus on indexing structures and search structures. Other works (e.g. [Orlando et al., 2007]) concentrate on the Trajectory Data Warehouse since moving objects are typical examples of spatial-temporal objects. Others focus on the conceptual models of a spatial temporal data warehouse. [Spaccapietra et al., 2008] apply a concept model called MADS model to show how time and spatial data are represented and provide the data to be analyzed from different perspectives. It is worth noting that [Malinowski & Zimányi, 2008] the dimension level as spatial based on the multi-dimension design. In [Vaisman & Zimányi, 2009], the authors maintain that there is still no clearly defined meaning of spatial data warehouses among these efforts, and existing works do not clearly specify the kinds of queries that are made. So it is hard to compare the different
proposals and approaches. In this paper, the authors reviewed the conceptual models proposed in order to improve our understanding about the spatial and temporal characteristics of real-world phenomena and the recent implementations. They also defined a conceptual framework supported by a spatial-temporal data warehouse.

**Spatial Dimension Types in Spatial Temporal Data Warehouse**

When we talk about time issues in reference to STDW, most works focus on the different time granularities. [Camossi et al., 2009] emphasized that it is crucial to base the selection of the appropriate on what is required from the system; factors such as efficient performance vs. data reduction must be considered. The data stored at the finest granularity, might cause waste of space and additional increase in query cost. However, storing data at coarser granularity will improve the efficiency of the system, but the detailed information might be ignored. Therefore, authors provide ST²_ODMGe, a spatio-temporal data model to be able to define multi-granular spatial temporal data. For example, according to whether the information is current or old, we might store the current data at a finer granularity, and the older data at a coarser granularity.

Here, we can also distinguish three types of spatial dimensions based on time lines:

- **Static spatial type**: It can be used to describe a fixed location, such as the location of a building. A building does not move and its location is therefore static.

- **Semi-static spatial type**: It is used to describe the activities of something that can change from static to active. For example, let us consider a volcano. When the volcano is dormant, the magma remains almost static; its state does not change too much for a period of time. But once the volcano becomes active, the level of magma, the temperature, the dimensions are changing constantly; its state then becomes active. So the state of an object referring to a semi-static spatial dimension can change from static to active.

- **Dynamic/Active spatial type**: It can describe a dynamic environment. The state of the environment is changing all the time (e.g. moving cars, bushfires). These objects’ spatial dimension is not static but changing greatly all the time.

From the above, we can see that for the first category, maintaining spatial dimension in real time is unnecessary; it can be maintained by the old batch window method. However, for the second and third categories, the maintenance of spatio-temporal DW in real time is very impractical due to the complexity of the transformation process and the large size of spatial data. The cost is extremely high. Therefore, near real-time temporary DW is most applicable in the spatial temporary data environment. It can accurately detect the right time to refresh spatial DW, offering a significant benefit in terms of refresh operation cost, while simultaneously maintaining a high freshness level of the data warehouse. The spatio-temporal data warehouse is still a young research area compared with the conventional data warehouse. In the current work, further research is needed to tackle the near-real time spatial temporal data warehouse which remains an open research problem.
MATERIALIZED VIEW

In conjunction with the data warehouse refresh, since the DW is required to be queried very effectively for decision-making, increasingly the use of materialized views is becoming the most effective and popular means of speeding up query processing efficiently [Rizzi & Saltarelli, 2003; Goldstein & Larson, 2001; Mistry et al., 2001; Mumick et al., 1997]. Hence, the concept of materialized view must also be considered. This section gives an overview and discussion of existing research regarding materialized views and how they are maintained.

In the database, a view is a virtual table that is derived from other existing tables. These other tables can be ordinary tables or another previously-defined view. It does not exist in physical form, as opposed to an ordinary base table whose records are actually stored in the database. We usually create a view for specifying a table that we need to reference frequently.

A materialized view (MV) is similar to a view but the difference is that the data in an MV is actually stored on disk (that is “materialized”) and it must be refreshed from the original base tables from time to time. Moreover, in some ways, we can treat a MV as a real table, since anything that we do to a table can be done to an MV as well. Therefore, we can build indexes on any column with the advantage that this improves the system’s performance by speeding up query time. So, a materialized view is used in order to decrease the cost of expensive joins or aggregations for an important and large class of queries.

A DW contains a large amount of information aggregated from diverse and independent data sources. One of the main reasons for designing a DW relates to queries and analysis, such as on-line analytical processing (OLAP), where hundreds of complex aggregations of queries have evolved over large volumes of data. It is not feasible to compute these queries by connecting the data source each time. For example, perhaps some data source is not always able to be accessed for it might be located at different places. So in order to speed up the queries in such an environment, many summary tables stored in the DW can be pre-computed including aggregated data such as the sum of sales. These tables represent materialized views. When users query a DW, frequent queries of the actual base tables is extremely expensive so the MV will be queried instead of the DW, which enables much more efficient access and saves cost [Samtani et al., 1999; Mumick et al., 1997] (e.g., Figure 8).

Maintenance of Materialized View Based on Data Warehousing

DW maintenance can be seen, in a somewhat simplified way, as a generalization of view maintenance used in relational databases. When a warehouse is updated, the MV must also be updated to reflect the changes in the operational database. In the dynamic environment business world, two types of changes should be considered:

Data changes. As data changes, a naive method of maintaining a MV is to re-compute the MV from scratch [Mumick et al., 1997]. However, this is impractical because it is extremely costly.

A number of algorithms have been proposed for batch incremental view. [Mumick et al., 1997; Samtani et al., 1999] use auxiliary tables (ATR) to keep some additional information in the DW. Auxiliary tables contain two kinds of attributes: primary key, which is in a select group and used for finding the corresponding actual values and actually attributes, which are the attributes we aggregate; they are usually the main things to be analyzed. Usually, there are more than one ATR in the DW, because each auxiliary table represents one individual data source.
This approach handles the problem of making MV self-maintainable, because data sources could be located anywhere in the world; therefore, accessing them in order to keep MVs up to date could be very time consuming. Moreover, it minimizes the batch time needed for maintenance by splitting the maintenance process into two parts, propagation which occurs outside of the batch window and refresh which occurs inside of the batch window. This method considers the situation where the changes in the data source are only the content changes, i.e., insert/update/delete records; however, it does not consider schema changes, i.e., add/modify/drop an attribute or a table, that are very common operations occurring in the data source. In this case, MVs cannot be self-maintained.

**Schema changes.** In order to maintain MVs and enable them to handle both schema and relationships, certain approaches have been proposed to solve the problem. They fall into two categories:

- **Schema evolution** (e.g., [Rundensteiner et al., 1999; Lee et al., 2002; Bébel et al., 2004]). A naive approach to solving the problem of schema changes is to isolate the changes from a data warehouse, which means that once schema changes occur, the attributes relating to the changes in the DW are restricted to be queried. This approach might be used for a limited period of time only, but is not an ultimate appropriate solution for it is not practical in business decision-making and analysis. Once the schema changes in the sources, it is impossible to query that part; hence, the DW function becomes incomplete.

- **Another better approach** [Rundensteiner et al., 1999; Lee et al., 2002, Bébel et al., 2004] is to transfer the schema changes related to the old view schema in MV and rewrite the schema. This approach considers only the rename attributes or relations at sources, drop attributes or relations.
**Versioning** (e.g., [Bébel et al., 2004; Morzy & Wrembel, 2004; Golfarelli et al., 2006; Rizzi et al., 2006]). In [Bébel et al., 2004; Morzy & Wrembel, 2004; Rizzi et al., 2006], both content and schema changes are tracked in the multi-version data warehouse (MVDW). In such a DW, versioning of schema, where histories of the schema are retained, changes to a schema or content may be applied to a new version of a DW. In [Bébel et al., 2004], there are two kinds of versions are proposed in the DW, one is called ‘real version’ which reflects the real world, it can be treated as a linear sequence of different versions to record the historical records for a certain period of time [Morzy & Wrembel, 2004]. Good decision making should be able to forecast future business behavior based on both current and historical data from which decision makers make assumptions. This kind of processing is termed ‘What-if analysis’ [Papastefanatos et al., 2007]. Another version of this approach [Morzy & Wrembel, 2004] is the ‘alternative version’, which is used for the what-if analysis or for simulating changes in the structure of a DW schema. According to the hypothetical scenario from the decision maker, a DW can create alternative data warehouse versions. Each version, whether real or alternative, has a valid time which is valid for a certain period of time, which can be represented by two timestamps, i.e. beginning valid time (BVT) and ending valid time (EVT). And this multi-version approach can be similarly adopted by the materialized view, and create a different version for recording the changes in MV. Compared with the schema evolution, the multi-version data warehouse may not be able to offer adequate functionality [Rizzi et al., 2006]. Researchers [Morzy & Wrembel, 2004] focus on new analytical tools and extended query language to extract partial results from versions, and integrate them into one consistent and meaningful result for users.

Materialized views are widely used in DW for query efficiency. A conventional database usually contains non-temporal data; i.e., only the current state of the data is available in the database. Thus, temporal materialized views can significantly benefit the query over the history of the source data. For example, a company’s database might contain only current employees’ information, and the previous employees’ information might be removed once they quit their job or are fired. Nevertheless, the company analysts might want an overview of all the employees who have ever worked in the company [Yang & Widom, 2000]. The problem of maintaining materialized views is highly challenging. On the one hand, a temporal materialized view does not only relate to the content and schema changes of operational database, but also to changes as time advances. When the data is removed or updated in the operational database, the replaced information might not be lost forever; it might be just temporally removed or invalid for a certain period. And the interaction between the data and time two dimension of change is evolved as well. [Yang and Widom, 2000] introduce a framework for maintaining temporal views over the non-temporal data sources. They present a temporal model that can be applied directly to the temporal support. The maintenance of temporal materialized views becomes more challenging than the non-temporal materialized views.

Little research has been conducted in temporal materialized views due to the problem of complex maintenance and view adaptation. And due to the large size of SDWs, the performance of spatial multidimensional queries and the efficient creation of a geographic materialized view without causing the problem termed ‘explosion of aggregates’ also present challenges.
CONCLUSION

The purpose of this preface was to highlight some of the trends and advancements made by the research work in data warehousing. We showed that attention to data warehouses has shifted from textual DWs to temporal and spatial DWs due to the requirements from the business world and diverse application contexts. Within this chapter, we focused on the time issues including DW refresh mechanisms, temporal time dimensions and the spatial feature in data warehousing. We categorized different types of time dimension and spatial dimension to broaden our perspective and gain a better insight into the field of data warehousing development.

Certainly, research has not been limited to the issues mentioned in this preface. The chapters of this book will give more details and techniques pertaining to various modern areas of data warehousing and data mining.

David Taniar
Monash University, Australia

Li Chen
LaTrobe University, Australia

REFERENCES


