DESIGN AND REPRESENTATION OF THE TIME DIMENSION IN ENTERPRISE DATA WAREHOUSES

A Business Related Practical Approach

Ahmed Hezzah, A Min Tjoa
Institute of Software Technology, Vienna University of Technology, Favoritenstr. 9-11/188, 1040 Vienna, Austria

Keywords: Data Warehouse, Time Dimension, Temporal Databases

Abstract: A data warehouse provides a consistent view of business data over time. In order to do that data is represented in logical dimensions, with time being one of the most important dimensions. Representing time, however, is not always straightforward due to the complex nature of time issues and the strong dependence of the time dimension on the type of business. This paper addresses the specific issues encountered during the design of the time dimension for multidimensional data warehouses. It introduces design and modeling techniques for representing time in the data warehouse by the use of one or multiple time dimensions or database timestamps. It also discusses generic problems linked to the design and implementation of the time dimension which have to be considered for (global) business processes, such as representing holidays and fiscal periods, increasing the granularity of business facts, considering the observation of daylight saving time and handling different time zones. These problems seem to have wide application, and yet, more in-depth investigations need to be conducted in this field for real-world time-based analysis in enterprise-wide data warehouses.

1 INTRODUCTION

As business activities tend to change over time, the business data must be able to represent that change (Devlin, 1997). Therefore, data modeling and application design approaches have not to focus only on a static view of the section of the real world to be modeled.

This might be adequate in operational applications, which manage only real-time data and take a view mainly of the current state of the business. A data warehouse, however, must explicitly consider the temporal aspects of the data it contains, because it must, by definition, provide a historical view of the business.

But the majority of today's modeling tools and databases still focus on the representations of "snapshots" of the current business information. One of the first approaches which incorporates the notion of time for modeling enterprise data is described in (Eder, 1987), where timestamps and states of entities and relationships are introduced. This approach leads to the specification of business rules with situation/activation diagrams as described in (Lang, 1997). A detailed description of most current research performed for this purpose can be found in (Wijsen, 1999) and (Wijsen, 2003). This paper will focus on some practical aspects of this research.

While operational systems are among other things designed to meet well-specified (short) response time requirements, the focus of data warehouses is the strategic analysis of data integrated from heterogeneous systems (Inmon, 1996). This business requirement has consequences for the data model design and for the architecture of the data warehouse. The absence of support for this temporal issue led data warehouse designers to take several approaches to reflect history in the database design (Snodgrass, 2000).

One important approach that is widely used is adding timestamps to the data. A timestamp is a specially defined field, in date-and-time format that tracks when a data record has been created, deleted, or changed in any way. Changes can be tracked on field-, record/row, or file/table level, depending on the required level of detail and the available storage. However, in this paper we focus on tracking changes on record/row level since this is the more common approach in the data warehouse.

But as we move toward the multidimensional approach data is represented in logical dimensions in

Hezzah A. and Min Tjoa A. (2004).
DESIGN AND REPRESENTATION OF THE TIME DIMENSION IN ENTERPRISE DATA WAREHOUSES - A Business Related Practical Approach.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 416-424
DOI: 10.5220/0002642604160424

order to provide a consistent view of the data over time, a view that can be used by decision support systems. One of the major dimensions in every multidimensional data warehouse is the time dimension. The time dimension contains descriptive temporal information, and its attributes are used as the source of most of the temporal constraints in data warehouse queries (Kimball, 1996). However, the design of the time dimension is not always straightforward as it strongly depends on the type of business and the requirements of the enterprise.

The aim of this paper is to introduce a specification of the time dimension in enterprise data warehouse systems, which is consistently applicable for handling the analysis of global enterprise data. The problems arising in multinational corporate groups when combining data with a temporal dimension are enormously cost-intensive. Even the minor problem of DST for one of the world-wide leading energy companies could cause data warehouse costs of millions of dollars, as it has been investigated by one of the authors.

This paper deals with the subject matter related to representing time in the data warehouse. It discusses the design of the time dimension and introduces design techniques for its implementation. It presents a practical approach, which also models relevant real world business issues such as holidays, seasons, daylight saving time and fiscal periods by extending the time dimension with new attributes and flags. It uses the time dimension together with timestamps to resolve major granularity issues. Finally, it addresses issues related to having different time zones and demonstrates how the use of local and universal time can resolve these issues.

2 RELATED WORKS

The functions needed to implement a data warehouse architecture including different types of data are described in (Devlin, 1997) and (Inmon, 1996). It addresses the use of timestamps to store periodic and historical business data, but doesn't consider the multidimensional approach widely used today. This is more discussed in (Kimball, 1996) with case studies of data warehouses for different types of businesses, almost all using a daily-based time dimension, unfortunately with no focus on the issues related to its implementation.

Adding history to the temporal database application is investigated in (Snodgrass, 2000) with focus on issues related to valid and transaction time, intervals and periods and state tables for valid and transaction time. It also presents some implementation considerations for the temporal database logical and physical design and demonstrates application development issues using SQL.

A conceptual multidimensional data model, which facilitates even sophisticated constructs based on multidimensional data units or dimension members, is introduced in (Nguyen, 2000). This model is able to represent and capture natural hierarchical relationships among dimension members within a dimension as well as the relationships between dimension members and measure values and is modeled using UML. Dimension updates are formally discussed in (Vaisman, 2001).

(Bruckner, 2001) presents an approach for modeling conceptual time consistency problems and introduces a model that deals with timely delays. However, this model doesn't address issues related to the time dimension as much as data consistency and updating issues. Changes of dimension data are discussed in (Eder, 2002), which presents an approach to represent temporal behavior of master data within existing, non-temporal data warehouses.

In (Ravat, 2000) a data warehouse class concept is introduced, which is based on the concepts of temporal filter and archive filter. It defines mapping functions to specify derived, calculated, and specific properties, and organize the inheritance hierarchy of the warehouse classes, allowing extracting only relevant data. In (Yang, 2000) Yang and Widom study incremental maintenance of temporal views using a temporal query language equivalent to TSQL2. Although (Ravat, 2000) does not organize data multidimensionally, it provides a more flexible temporal model than (Yang, 2000) because the purging values are not deleted, but they are archived.

The use of multiple time dimensions is mentioned in (Kimball, 1999), which introduces the concept of a data webhouse. It uses a clickstream data mart to store all web activities for later analysis of user behavior. This is also discussed briefly in (Pedersen, 2001) with focus on the influence of the web on data warehousing, but also the design of clickstream fact tables and dimension tables.

Other temporal issues like fiscal periods and granularity are briefly discussed in (Kimball, 2002) and (Kimball, 1997) with more focus on using the time dimension to resolve this issues, but design issues are not investigated in detail.

The aim of this paper is to give a framework for modeling the time dimension in data warehouses for enterprise wide (global) information systems with focus on its applicability for practical issues, such as daylight saving time (DST) or problems related to time zones, holidays, and fiscal periods. This paper

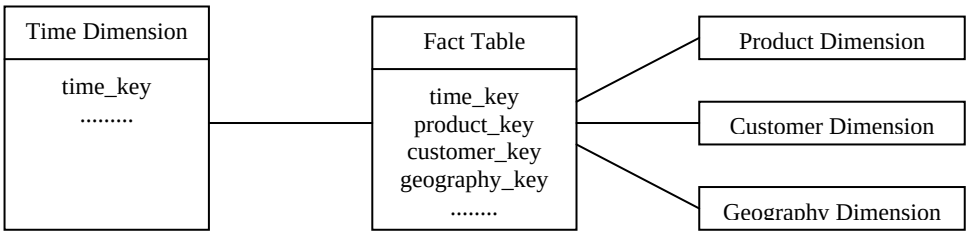

Figure 1: A multidimensional model

presents design techniques for representing time in multidimensional data warehouses and temporal databases. It provides an approach to model practical problems of different time representations, which will be demonstrated considering relevant real-world examples, such as the DST-problem, the modeling of holidays, fiscal periods, etc., by using one or multiple time dimensions and timestamps.

3 TEMPORAL ISSUES

3.1 The Time Dimension

The TIMESTAMP type available in SQL provides a representation of time in a very fine precision. It stores the year, month, day, together with the hour, minute, second, and a number of fractional digits of the second. It can be used in a database table to record the occurrence of certain events (e.g. deposit to and withdrawal from a bank account), as well as the start and end of a certain state (e.g. a certain employee belongs to a certain department).

As we move toward a multidimensional approach the simple timestamp is replaced with a time dimension. The time dimension is then filled with a lot of helpful calendar attributes and is connected to the fact table by a foreign key (Figure 1). But do we really need a dimension for time? Wouldn't it be better to use an SQL timestamp in the fact table instead of the foreign key and avoid this expensive join? To answer this question let's take a look at these simple queries:

- Show all the transactions that occurred within a given period of time.
- Determine whether a certain transaction occurred within a given period of time.
- Show transactions using complex calendar navigation capabilities including seasons, fiscal periods, day numbers, week numbers, weekdays and holidays.

While the first two queries are pretty simple to define using a single timestamp that stores the occurrence time of each transaction, this is not the case for the third query. Since SQL timestamp know nothing about an organization's calendar, fiscal periods or holidays, these attributes are modeled using a time dimension (Figure 2). This way the application designer doesn't have to embed these calendar constrains in the application design, which would require a set of complex queries to determine these attributes. Besides that this would be very slow, the end-user application can't easily produce the needed SQL.

A time dimension can easily be built using a simple spreadsheet. A 20-year time dimension on daily basis contains about 7300 rows, which is not much. It can also be filled with a single SQL INSERT statement, as we will see later. However, problems will start to arise when the fact table requires granularity smaller than a day, let it be an hour, a minute, or a second. It's not possible to create a single time dimension with all the minutes or seconds over a long period of time. There are more than 500,000 minutes and 31 million seconds in a year. So, for these cases the only way seems to be to use SQL timestamps despite the limitations we mentioned and to give up the ability to navigate through seasons and fiscal periods to the nearest second. We will come to this later when we talk about granularity to see how to overcome this issue.

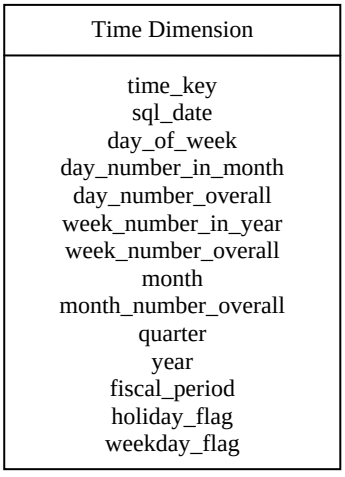

Figure 2: The time dimension

3.2 Holidays, Seasons and Fiscal Periods

The time dimension as defined above gives us the ability to track business facts very well on a daily basis. But as the business requirements become more complex the time dimension must be extended by new attributes to meet those requirements. For example, the business might require looking at sales on holidays versus non-holidays. With an OLTP model holidays are usually stored in an own table, which is filled with all the holidays and can be joined with the sales table by date key. Thus a query is then used to join this table with the sales table:

```
select sum(sales.quantity_sold)
from sales, holidays
where trunc(sales.date_time_of_sale) =
  trunc(holidays.holiday_date);
```

As this looks pretty simple, since the holidays table eliminates sales on days that are not holidays, the case looks slightly different when we want to eliminate sales on days that are holidays. The query will look like that:

```
select sum(sales.quantity_sold)
from sales
where trunc(sales.date_time_of_sale)
  not in (select holiday_date from
  holidays);
```

Of course this query will take longer time to execute. The sales table may contain millions of rows, and the holidays table will contain about 50 to 100 rows. Here, the sub-query will be performed for each row examined by the main query. So the time to execute this query might be much longer than the holidays query. Besides, it will become more complex when we want to look at sales, not just on holidays, but also on different seasons, fiscal periods or weekdays. For each of these attributes a separate table is necessary, which then must be joined with the sales table.

Here comes the big advantage of the time dimension. All these attributes can be integrated into the time dimension in a way that effectively reduces query execution time and provides more functionality than using conventional RDBMS tables. The tables for holidays, seasons, fiscal periods and weekdays are replaced with attributes and flags in the time dimension, which will look like this:

```
create table time_dimension (
  time_key       integer primary key,
  -- this is midnight (TRUNC) of the
  -- date in question
  sql_date       date not null,
  day_of_week    varchar(9) not null,
  -- 'Sunday', 'Monday'...
  day_number_in_month integer not null,
  -- 1 to 31
  day_number_overall integer not null,
  -- first day is 1
  week_number_in_year integer not null,
  -- 1 to 52
  week_number_overall integer not null,
  -- weeks start on Sunday
  month  integer not null, -- 1 to 12
  month_number_overall   integer not
  null, -- first month is 1
  quarter integer not null, -- 1 to 4
  year   integer not null,
  fiscal_period  varchar(10),
  holiday_flag   char(1) default 'f'
  check (holiday_flag in ('t', 'f')),
  weekday_flag   char(1) default 'f'
  check (weekday_flag in ('t', 'f')),
  season varchar(50),
);
```

Of course these attributes must be defined and filled according to the organization's fiscal calendar. Now if we want to report sales by season, the query will be straightforward:

```
select td.season, sum(f.dollar_sales)
from sales_fact f, time_dimension td
where f.time_key = td.time_key
group by td.season;
```

Using the group by command we can report by other attributes like holidays and fiscal periods in an identical way, which makes the queries much faster than using separate tables. Using other dimensions like products, customers, manufacturers, etc. makes us able to report by different criteria.

As mentioned above, the time dimension can be populated using a spreadsheet or even easier with a simple SQL INSERT statement. This is done using SQL date formatting functions and a help table `integers`, which supplies a series of numbers to be added to a selected starting date. For this example let January 1st 1998 be the first date:

```
-- Uses the integers table to drive the
-- insertion, which just contains
-- a set of integers, from 0 to n.
```

```
-- d below is the SQL date of the day -
-- we're inserting.

insert into time_dimension
  (time_key, sql_date, day_of_week,
  day_number_in_month,
  day_number_overall,
  week_number_in_year,
  week_number_overall,
  month, month_number_overall, quarter,
  year, weekday_flag)
select n, d, rtrim(to_char(d, 'Day')),
to_char(d, 'DD'), n +
1,to_char(d,'WW'),trunc((n + 4) / 7),
-- Jan 1, 1998 was a Thursday, so +4 to
-- get the week numbers to line up with
-- the week
to_char(d, 'MM'),trunc(months_between
(d, '1998-01-01') + 1),to_char(d, 'Q'),
to_char(d, 'YYYY'),
decode(to_char(d,'D'), '1', 'f', '7',
'f', 't')
from (select n, to_date('1998-01- 01',
'YYYY-MM-DD') + n as d
from integers);
```

The remaining fields (`season`, `fiscal_period`, `holiday_flag`) cannot be filled using SQL date functions and have to be populated afterwards. Fiscal period and season depend on the organization's choice of fiscal year. To update the `holiday_flag` field, which is '`f`' by default, we need two help tables: one for the fixed holidays and one for the floating holidays.

```
create table fixed_holidays (
  month  integer not null
  check (month >= 1 and month <= 12),
  day    integer not null
  check (day >= 1 and day <= 31),
  name   varchar(100) not null,
  primary key (month, day)
);

-- Specifies holidays that fall on the
-- n-th DAY_OF_WEEK in MONTH.
-- Negative means count backwards from
-- the end.

create table floating_holidays (
  month  integer not null
  check (month >= 1 and month <= 12),
  day_of_week    varchar(9) not null,
  nth    integer not null,
  name   varchar(100) not null,
  primary key (month, day_of_week, nth)
);
```

Some example holidays:

```
insert into fixed_holidays (name,
month, day)
   values ('New Year''s Day', 1, 1);

insert into floating_holidays (month,
day_of_week, nth, name)
  values (11, 'Thursday', 4,
  'Thanksgiving');
```

After that, it is easy to update the `holiday_flag` in the time dimension using these two help tables and any procedural language like PL/SQL to set the `holiday_flag` to '`t`' for the days just inserted into the two tables.

In order to consider holidays in different countries or in different time zones we could use multiple holiday flags (`holiday_flag_1`… `holiday_flag_n`), one for each country we need to consider. For instance, October 3rd is a national holiday in Germany, so we set the `holiday_flag` for Germany to '`t`', while for all other countries we leave it '`f`'. This way, we can run queries like "how did this German holiday affect sales in neighboring countries like Austria and Switzerland?".

3.3 Granularity

Granularity is the level of detail of the facts stored in a data warehouse. As mentioned above, if we are only modeling calendar days the time dimension provides a very good approach to track business on a daily basis. But what if we need to add some more precision to the fact table in order to store more temporal details? Can we just increase the granularity of the time dimension to the nearest hour, minute or even second or do we have to give up the ability to navigate by time and to specify seasons, fiscal periods and holidays?

To answer this question let's first take a look at a time dimension that stores all the days in a defined period of time. This dimension will contain a row for each day, which means that a 10-year dimension will contain 3650 rows. Now if we want to track changes to the nearest hour, minute, or second, this could be done in one of the following ways:

Increase the granularity of the time dimension

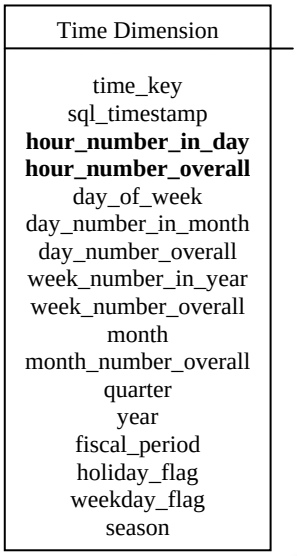

Figure 3: A time dimension on hourly basis

With this approach the time dimension is changed to store all the hours, minutes, or seconds of the specified time period. For a 10-year time dimension this will mean that it will contain approx. 87600 rows (3650 x 24) to store each hour, 500,000 rows for each minute, and over 31 million rows for each second.

While this might be an acceptable size for storing hours this is definitely not the case for minutes and seconds. Moreover, in order to keep the size of the time dimension small and predictable, the duration must be kept constant by deleting old entries when new ones are inserted. This makes the business data stored only semi-periodic.

This approach is useful if the granularity is limited to the nearest hour for a not too long period of time (Figure 3), which also makes it possible to navigate by hour, for e.g. to see what day times were the best for sales. However, if we need to store the time to the nearest minute or second, we prefer using one of both other approaches.

Add timestamps to the fact table

Adding SQL TIMESTAMPS directly to the fact table provides a very high precision as the granularity of these timestamps can go up to some fractions of the second. Occurring events can thus be captured on the second of occurrence, and the start and end of a status are also stored second exact.

This is good if a very high precision is needed over the navigation features of the multidimensional model. If we choose this approach we have to live with the limitations of SQL TIMESTAMPS and give up the ability to specify seasons, holidays, or fiscal periods to the nearest second.

Use twin timestamps

This approach combines the advantages of both previous approaches by using two timestamps on each transaction record in the fact table. The first would be an SQL TIMESTAMP as described in the previous paragraph, and the second would be a day id, a foreign key connecting to a calendar day dimension (Figure 4).

The time of day could also be stored in a separate numeric field instead of using a timestamp, or even a separate dimension can be used for the time of day, as we will see when we talk about time zones. But anyway it should not be combined into one key with the calendar day as this would make the time dimension simply too large.

This way we can search for very precise time periods, but also navigate to see all transactions that occurred on a holiday.

3.4 Daylight Saving Time

Daylight Saving Time (DST) is the practice of turning the clock ahead as warmer weather approaches and back as it becomes colder again. DST varies from country to country.

Here we are more concerned about the representation of time on those days when the time

time	day_key	product_key	customer_key	quantity	price
25-MAR-2003 15:37:13	1910	12265	7657654	5	200
25-MAR-2003 16:15:45	1910	34324	2423555	8	240
03-APR-2003 11:47:02	1919	25254	3545466	3	150

Figure 4: A fact table using twin timestamps

is shifted. This happens on two days every year. In the European Union, DST starts the last Sunday in March at 1 am UTC and ends the last Sunday in October at the same time.

These two days must be treated differently in the time dimension because they are different than other days. While all other days of the year have 24 hours and can thus be modeled as shown above the day when the time is set to DST has only 23 hours since the time is set from 0 am directly to 2 am UTC. On the other hand, the day when the time is set back has 25 hours because the hour from 1 am to 2 am is repeated twice.

Let's take as an example March 30th 2003 and October 26th 2003. On March 30th there is actually no point in time when it is 1 am. The clock jumps from 00:59:59 directly to 02:00:00. Therefore there is no need to include the 1 am hour in an hourly-based time dimension.

On October 26th, however, there are two points in time when it is 1 am. The clock goes from 01:59:59 back to 01:00:00 again instead of 02:00:00. After another hour the time is actually 02:00:00.

For our time dimension shown in Figure 3 this means that on all last Sundays in March we only need to insert 23 hours by leaving the 1 am hour, as it does not really exist. The `hour_number_in_day` thus goes from 1 to 23. And on all last Sundays in October we insert 25 hours by repeating the 1 am hour. The `hour_number_in_day` thus goes from 1 to 25 (Figure 5). Of course the DST days have to be determined in advance before the time dimension is filled with values.

The optional attribute `second_in_day` can be useful for the application to correctly determine the timestamp and other time periods on those two special days and it must be interpreted differently on those two days than on normal days. Depending on the application and the needed queries this attribute can be used or not.

Finally, to make it easier for SQL to determine the days when time is set to DST and back we use another flag `DST_flag` which is zero on normal days, -1 on all last Sundays in March, and +1 on all last Sundays in October.

Please note that the described model is using UTC. If you design your time dimension for another time zone, you have to consider the days and times when the time is switched to DST and back.

3.5 Time Zones

Different time zones not only mean having different times, but also DST is observed differently in different regions, which makes the design of the time dimension even more complicated.

As we mentioned at the end of the previous section, the days and times when time is set to DST must be considered in the design of the time dimension. However, business data are not always entered within the same time zone. This is one of the reasons for using a Geography dimension. In a Sales data warehouse this would help to store where a product was sold. But which time should be used: local or universal time?

Now that the web has become an extremely important source of data warehouse data, a source that produces data with the speed of a click, it brings up several issues that are not yet resolved. Data enters the warehouse from thousands of users in different time zones, but must all be stored into the same database.

Generally, it's easier to store the local time than to compute it based on time zones, which is very useful for queries such as "at what time of day were the most orders placed". But as this can be done in any time zone, and as online stores begin to be a very important point of sale the use of universal time might make more sense to compare order times worldwide.

Therefore it is better to store both: local time and universal time. This can be done by adding another copy of the time dimension for universal time. And as mentioned previously, to add more flexibility and granularity the time-of-day can be separated from the day by using two dimensions: a date dimension and a time-of-day dimension for both local and universal time (Figure 6). This gives us altogether four dimensions for representing time. We split the date from the time-of-day because these two components of time have different descriptors. Date relates to calendar and weekdays and seasons, and time-of-day relates to the specific spot we are in within a day.

The time-of-date dimension might also be used if we have some specific intervals during the day that we want to assign names to, afternoon, evening, etc.

This way we can navigate through sales facts by

time_key	sql_timestamp	seconds_in_day	hour_no_in_day	hour_no_overall
50975	26-OCT-2003 00:00:00	0	1	50975
50976	26-OCT-2003 01:00:00	3600	2	50976
50976	26-OCT-2003 01:00:00	7200	3	50976
50975	26-OCT-2003 02:00:00	10800	4	50975
.	.	.	.	.
.	.	.	.	.
50999	26-OCT-2003 23:00:00	90000	25	50999

Figure 5: October, 26th 2003 modeled in the time dimension

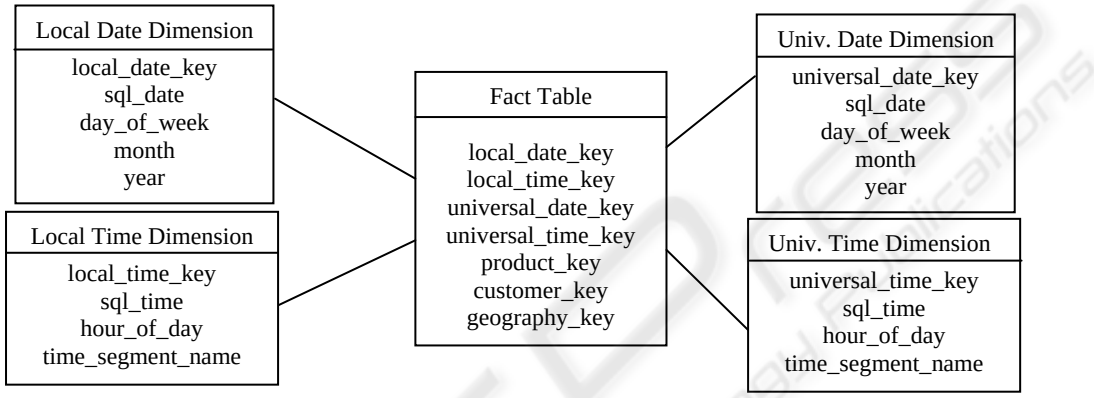

Figure 6: Splitting the time dimension into a date dimension and a time-of-day dimension

absolute time as well as relative to the customer's time. Having separate dimensions for local and universal time we don't need to implement the time calculation based on time zones into the application logic. This makes our queries more efficient as well.

4 CONCLUSIONS

This paper addressed the most common issues related to representing time in the multidimensional data warehouse. It introduced design techniques using one or multiple time dimensions or time-stamps. The representation of holidays, seasons and fiscal periods has been demonstrated by extending the time dimension with new attributes and flags. Different approaches have been introduced to increase the granularity of business facts up to the nearest second by using a combination of the time dimension and timestamps. The paper also addressed the issue of observing daylight saving time DST and how it affects the design of the time dimension, and provided an approach to handle this issue by introducing 23-hour and 25-hour days. This approach has been then modified to handle different time zones by using multiple time dimensions and storing the universal time in addition to the user or customer's local time and separating the time-of-day in another dimension.

REFERENCES

Bruckner, R., Tjoa A.M., 2001. *Managing Time Consistency for Active Data Warehouse Environments*, In Proc. of the Third International Conf on Data Warehousing and Knowledge Discovery (DaWaK 2001)

Devlin, B., 1997. *The Data Warehouse, from Architecture to Implementation*, Addison Wesley Longman, Inc.

Eder, J., Kappel, G., Tjoa, A.M., Wagner, R., 1987, *BIER - The Behavior Integrated Entity Relationship Approach*, in: S. Spaccapietra (ed.), Proceedings of the 5th Intenational Conference on Entity-Relationship Approach, North-Holland, Amsterdam

Eder, J., Koncilia, C., 2002. *Representing Temporal Data in Non-Temporal OLAP Systems*, University of Klagenfurt

Inmon, W., 1996. *Building The Data Warehouse*, John Wiley & Sons, Inc.

Kimball, R., 1996. *The Data Warehouse Toolkit*, John Wiley & Sons, Inc.

Kimball, R., 1997. *It's Time for Time*, DBMS Online

Kimball, R., 1999. *The Clickstream Data Mart in the Data Webhouse*, Intelligent Enterprise

Kimball, R., 2002. *Tricky Time Spans*, Intelligent Enterprise

Lang, P., Obermair, W., Schrefl, M., 1997, *Modeling Business Rules with Situation/Activation Diagrams*, In: A. Gray, P. Larson (eds.): Proceedings of 13th International Conference on Data Engineering (ICDE '97), Birmingham, U.K., IEEE Computer Society Press

Nguyen, T., Tjoa, A.M., Wagner, R., 2000. *An Object Oriented Multidimensional Data Model for OLAP*, In Proc. of 1st Int. Conf. on Web-Age Information Management (WAIM 2000)

Pedersen, P., 2001. *e-Decisions Transcript*, Norwegian School of Economics and Business Administration

Ravat, F., Teste, O., 2000. *A Temporal Object-Oriented Data Warehouse Model*, DEXA'00

Snodgrass, R., 2000. *Developing Time-Oriented Database Applications in SQL*, Morgan Kaufmann Publishers

Vaisman, A., Mendelzon, A., Ruaro, W., Cymerman, S., 2002. *Supporting Dimension Updates in an OLAP Server*, CAiSE'02

Wijsen, J., Ng, R.T., 1999 *Temporal Dependencies Generalized for Spatial and Other Dimensions*, Proc. Spatio-Temporal Database Management

Wijsen, J., Bès A., 2003, *On query optimization in a temporal SPC algebra*, Data & Knowledge Engineering, Volume 44

Yang, J., Widom, J., 2000. *Temporal View Self-Maintenance in a Warehousing Environment*, EDBT'00