

Author : Chris Drawater  
Date : April 2008  
Version : 2.0

## Database Design – the Time Dimension

### Abstract

Design a time dimension table and algorithm and use it across all your data warehouses and marts.

### Document Status

This document is Copyright © 2008 by Chris Drawater.

This document is freely distributable under the license terms of the [GNU Free Documentation License](http://www.gnu.org/copyleft/fdl.html) (<http://www.gnu.org/copyleft/fdl.html>). It is provided for educational purposes only and is NOT supported – use at your own risk !

### Introduction

This paper provides an overview into the use of the time or calendar dimension and makes available a simple algorithm and code examples for re-use.

### Abbreviation & Definitions

DDL → (SQL) Data Definition Language  
DML → (SQL) Data Manipulation Language  
DW → Data Warehouse  
3NF → 3<sup>rd</sup> normal form (stage used in database normalization)  
DBMS – Database Management System

## Dimensional based models

Within a star or snowflake schema, the dimension table contains a (domain specific) combination of attributes - often de-normalized, often with textual descriptions instead of foreign keys to other tables. The dimension table can sometimes be viewed as a number of related reference tables squashed into a single table.

For example, if so required, a single dimension table could cover all the 'sales' aspects (salesman, sales location, sales area...) whilst another might cover all the 'product' aspects, both without reference to any other tables. In a strict 3NF model, the sales related data would be modeled as series of related (and/or cascading hierarchy of) tables.

Each different dimension code referenced within the row of a fact table thus characterizes a different dimensional aspect of that row.

Frequently the main fact table of a star schema contains just

- dimension codes (effectively foreign keys or pointers to the dimension tables), plus
- additive numeric values (such as money, unit quantities etc).

For example, a simplified 'fact' table hosting priced mobile phone calls may contain →

```
dimensional values =>
    time dimension code
    network dimension code
    phone dimension code
    customer dimension code
    product dimension code
numeric values =>
    call duration
    pre-discount price
    discount
others =>
    call date/time
```

Often a data warehouse is based upon a dimensional based model for a number of reasons →

- data compression : detailed transaction data is transformed into physically smaller (fact table) rows so it becomes possible to hold months or years worth of data.
- an easy to understand/navigate data model : this facilitates complex multidimensional queries and simplifies the SQL needed for data retrieval.
- provides data model extensibility : it is easy to enhance the overall model by adding new, or modifying existing, dimension tables - so you can start simple !

The downside is the transformation needed (and sheer processing capability) required to get large numbers of records into the dimensional model format.

## Time or Calendar Dimension

The time dimensional table can be considered to be representing a **time hierarchy** →  
For example,

year → week no → day of week  
→ month → day of month → hour → qtr hour  
→ day of year  
→ year quarter

which might be represented in a DBMS agnostic table (ref 1) as →

*create table calendar*

```
(  
    tcode          varchar(15),          /* generated */  
    year           numeric(4),  
    month_n        numeric(2),          /* 06 */  
    month_c        varchar(3),         /* June */  
    daymonth       numeric(2),  
    dayweek_c     varchar(3),         /* eg Mon, Tues */  
    hourband       numeric(2),  
    qtrhrband     numeric(1),  
    qtr           numeric(1),         /* 1,2,3 or 4 */  
    week           numeric(2),         /* 24 */  
    day           numeric(3),         /* 112 */  
    tz             varchar(3),         /* timezone */  
);
```

This simple time dimension might further be enhanced by including columns such as →

- Fiscal period
  - Significant days ( such as religious festivals)
  - Public holiday
- amongst others.

## Algorithm

Rather than using the traditional sequential unique number for the time dimension code (*tcode* in the above table) , it might be more efficient to generate it.

The algorithm for generating the time dimension code can be relatively simple and purely date/time dependent.

For example,

```
Timecode = to_number {concatenated strings of  
                    (4 digit year  
                    + 2 digit month  
                    + 2 digit daymonth  
                    + 2 digit 24hourband  
                    + 1 digit quarterband  
                    )  
                    }
```

where *quarterband* is defined (on the basis of number of minutes) as follows →

```
if (mins < 15) then
    qtrh := '1';
elsif (mins < 30) then
    qtrh := '2';
elsif (mins < 45) then
    qtrh := '3';
else
    qtrh := '4';
end if;
```

This algorithm maps nicely onto the more common date representations.

So the time code for

*31/04/2008 20:10:33*

becomes

*'2008' || '04' || '31' || '20' || '1' => 20080431201*

Because the timecode is a numeric representation of a 'point in time' 15 minute interval, it is also possible to undertake date comparisons and even simple arithmetic.

eg. *select..... where fact.timecode > 2008043120*

However, this algorithm does not take into consideration, timezones.

The introduction of timezones transforms the data warehouse into a tool that can support and cross analyze international data from different countries with differing timezones.

The example algorithm can be simply modified to incorporate timezone (TZ) →

```
Timecode = {concatenated strings of
    (4 digit year
    + 2 digit month
    + 2 digit daymonth
    + 2 digit 24hourband
    + 1 digit quarterband
    + 3 digit TZ
    )
}
```

Note however that *timecode* above is no longer a number but is alphanumeric.

Because of the potentially huge volume of low level data that a data warehouse can potentially hold, a 15 minute granularity is probably about right to enable fine resolution date/time based SQL.

And as the actual date/time is also likely to be held against the 'raw' data, timecode and actual date/time can always be referenced together for further accuracy.

## Generate not lookup

During data warehouse loading, avoid performing an SQL lookup to determine the time dimension code for a given date – instead generate the code using the simple algorithm above.

Why? Performance – each table lookup will require at least 1 SQL select and SQL execution is an expensive operation, even when the data is cached in the DBMS memory. Generating, instead of using SQL to determine, the correct time code during data load will save at the least 1 ( maybe more) SQL statements per row which could translate to millions within a data warehouse environment.

Perhaps the generated ids can be held in memory within the loader code. In fact, the dimension table can be made self-populating during verification - an important factor when no-one has the time or inclination to provide pre-loaded dimension data.

In fact, the database table representing the time code might only really exist for the purpose of participating in dimensional based SQL queries against the “fact” table.

For example, using the above calendar table, to help identify the Top 20 products by usage for a specified day →

```
select * from
(
select
p.product,
p.subproduct,
count(e.pcode) total_usage
from event e, product p, calendar c
where c.year = 2007 and c.month_n = 10 and c.daymonth = 16
and c.tcode = e.tcode
and e.pcode = p.pcode
group by p.product, p.subproduct
order by total_usage desc
)
where rownum <= 20;
```

## Standardization

Once a data warehouse or mart is built and loaded using a particular time dimension, that dimension cannot easily be changed without updating all the tables making use of it.

It is thus a good idea for all corporate data warehouses and marts to use the same time dimension encoding. This will ensure that events or activities or sales (whatever is in the fact tables) can be related across physically separate databases and viewed as a single virtual database.

For example, a network traffic database and revenue mart, using a common time dimension model, might be able to provide intelligence such as →

- Hourly Revenue on Monday mornings relative to cell traffic or network faults
- Revenue by week during 3rd quarter of the year from mobiles abroad relative to mobiles hired or phones purchased at airports during the same period.

Furthermore, the databases no longer need to be of the same type : mix and match time based information from Oracle, PostgreSQL, MS SQL Server or even Database Appliances etc.

Probably more so than the application systems, a company’s data, dimensions and models are a valuable long term corporate asset.

## Concluding Remarks

It is worth spending a little time designing the time dimensional model so it

- covers all potential time based scenarios
- can facilitate data analysis across timezones
- can be implemented as the one common model across all your data warehouses, data marts and database appliances etc.

Hopefully this paper has provided a few ideas that can be re-used to help achieve that goal.

*Chris Drawater has been working with RDBMSs since 1987 and can be contacted at [drawater@btinternet.com](mailto:drawater@btinternet.com).*

## References

1. Drawater(2008), Going about Physical Database Schema Design , <http://www.holindis.co.uk>

## Appendix 1. Java code components to generate timecode

```
private java.util.Date d;

private int month_n;
private int qtrhr;
private String year_s;
private String month_ns;           // eg 03
private String dayOfMonth_s;
private String hour24_s;
private String qtrhr_s;

private int qtr;
private int week;
private int day;

private String tz;
private String timecode;

year_s = String.format("%tY",d);
month_ns = String.format("%tm",d);
dayOfMonth_s = String.format("%td",d);
hour24_s = String.format("%tH",d);
qtrhr = 1 + (int)(Integer.parseInt(String.format("%tM",d))/15);
qtrhr_s = String.valueOf(qtrhr);
tz = String.format("%tZ",d);

timecode = year_s + month_ns + dayOfMonth_s + hour24_s + qtrhr_s + tz;
```

## Appendix 2. Java code components to insert timecode into database

```
private PreparedStatement ds;
private Connection con = null;

month_n = Integer.parseInt(month_ns);
day = Integer.parseInt(String.format("%tj",d));
week = 1 + (int)day/7;

if (month_n < 4)
{
    qtr = 1;
}
else if ( month_n < 7)
{
    qtr = 2;
}
else if ( month_n < 10)
{
    qtr = 3;
}
else
{
    qtr = 4;
}
ds = con.prepareStatement("insert into calendar
(tc,year,month_n,month_c,daymonth,dayweek_c,hourband,qtrhrband,qtr,week,day,tz) values
(?,?,?,?,?,?,?,?,?,?,?,?,?)");

ds.setString(1,timecode); // timecode
ds.setInt(2,Integer.parseInt(year_s)); // numeric year
ds.setInt(3,month_n); // numeric month 1 - 12
ds.setString(4,String.format("%tb",d)); // local short name month
ds.setInt(5,Integer.parseInt(dayOfMonth_s)); // numeric day of month
ds.setString(6,String.format("%ta",d)); // local day of week
ds.setInt(7,Integer.parseInt(hour24_s)); // numeric 24 hr band
ds.setInt(8,qtrhr); // numeric qtr hr band
ds.setInt(9,qtr); // numeric qtr
ds.setInt(10,week); // numeric week
ds.setInt(11,day); // numeric day
ds.setString(12,tz); // char TZ

ds.executeUpdate();
con.commit();
```