STORING DIMENSIONS AND UNITS OF PHYSICAL QUANTITIES IN A RELATIONAL DATABASE

Matúš CHOCHLÍK
Department of Informatics, Faculty of Management Science and Informatics,
University of Žilina, Univerzitná 8215/1, 010 26 Žilina, tel. 041/513 4061, E-mail: matus.chochlik@fri.uniza.sk

ABSTRACT
Many relational databases store records with columns containing various physical properties of the stored items. However when using these data, the dimension and unit checking is often left to the application logic. In physical calculations the individual terms represent various quantities both those having dimensions and dimensionless. Furthermore the values can be expressed in various units. The dimensions and units are important as a type system that is ensuring correctness and provide information about the meaning of the terms and results of the calculations. This paper presents the design of a working implementation of a subsystem representing physical quantities and units in the PostgreSQL database system.

Keywords: *physical quantities, units, calculations, conversions, relational database system, SQL, postgresql.*

1. INTRODUCTION

When storing large amounts of values of various physical quantities or when using these numbers in calculations, it is important to know their dimensions and the units in which they are expressed. When the calculations become complex this knowledge is what can prevent us from comparing or adding values representing volume or mass to values representing for example length or voltage or misinterpreting the meaning of the result of a complicated formula.

1.1. The SI unit system

The international standard *Système International d'Unites* (SI) – International Unit System, currently recognizes seven base and two supplementary (now called derived) dimensions and provides definitions of the base units for these dimensions. SI is the world's most widely used system of units, both in everyday commerce and in science [1], [2]. According to [2], [3] as of 2008, work is proceeding on a new standard ISO/IEC 80000, to be referred to as *International System of Quantities* (ISQ).

The base dimensions and their respective base units are [1], [2]; *length* [metre], *mass* [kilogram], *time* [second], *electrical current* [ampere], *thermodynamic temperature* [degree Kelvin], *amount of substance* [mole] and *luminous intensity* [candela]. The two additional quantities and units *angle* [radian] and *solid angle* [steradian] were until 1995 called supplementary, and now are being considered derived. However, in many practical calculations it is better to treat them like base units.

Nearly any physical quantity can be expressed as a combination of powers of these nine dimensions. Table 1 shows some of the possible combinations and units. Note that this is only a very small fraction of all derived quantities. The derivation is performed by the means of dimensional analysis [7], [10].

In addition to the base units the SI standard also specifies several prefixes to express amounts that are several orders of magnitude larger or smaller than the base units. These prefixes are based on the powers of 10 and 1000 and are listed for example in table [1], [2], [4]. The prefixes are applied to the names of the base SI units. Multiple prefixes are deprecated. The only base unit that has a prefix is the kilogram, thus the basic name to which prefixes are applied is one gram that equals to 10^{-3} kg.

Table 1 Examples of derived physical quantities

Derived quantity	Compound expression	Units
volume	length.lenght.length	m^3
frequency	1/time	s^{-1}
velocity	length/time	$m.s^{-1}$
electric capacitance	electric charge/voltage	$m^{-2}\cdot kg^{-1}\cdot s^4\cdot A^2$
force	mass.acceleration	$kg.m.s^{-2}$

1.2. Derived standard units

Many derived quantities have standard derived units, used for convenience instead of the lengthy expressions in terms of the SI base units. Table 2. shows few examples.

Table 2 Standard derived SI units

Quantity	Symbol	Name	In terms of base SI units
energy, work, heat	J	joule	$m^2\cdot kg\cdot s^{-2}$
electrical conductance	S	siemens	$m^{-2}\cdot kg^{-1}\cdot s^3\cdot A^2$
inductance	H	henry	$m^2\cdot kg\cdot s^{-2}\cdot A^{-2}$
illuminance	lx	lux	$cd\cdot m^{-2}$

1.3. Traditional units

In everyday life multiple non-SI units, derived from the standard units are used. For example 1 litre, a unit of volume of fluids, that is equal to 1 dm^3, which is in turn equal to 10^{-3} m^3, 1 km/hour a unit of velocity, 1 ton, unit of weight equal to 10^3 kg or 1Mg, 1 °C or 1 °F, units of temperature, etc. These units are used for convenience because they are roughly equal to the common magnitudes or multitudes in everyday practice.

1.4. Other unit systems

Due to historical, cultural and other reasons there also are several other unit systems that are still in use today, most notably the *Imperial Unit System, U.S. Customary Units* or *Avoirdupois* [1], [2], [5]. However, virtually any physical quantity expressed in non-SI units can be converted to SI units.

1.5. Derived standard units

Having meta-information about the quantities, that are stored as plain numbers in tables of a relational database can be useful in many situations. They allow doing conversions to other units and calculations with quantity type checking with a much greater flexibility compared to conversions and calculation hard-coded into the applications. It is therefore useful to have an extensible subsystem that allows storing and using the dimensions and units and allows exploring the relationships between them. This article presents a model and an implementation of such subsystem in the PostgreSQL [6] DBS, which provides several useful extensions to the SQL language.

2. DESIGN AND MODEL

In order to achieve maximal usability, our model needs to capture the essence of the concepts from the SI standard, like quantity, unit, prefix, composition and conversion mentioned in the introduction.

2.1. Modelling the base concepts

2.1.1. Dimensions

One of the most important properties of the physical quantities is, that they can be combined into other quantities. For example *force = mass . acceleration, area = length . length* *pressure = force / area, electric charge = time . electric current*, etc. [7].

In order to support this kind of calculations and to allow exploring the relationships between quantities and units, the concept of unit dimensions is modeled by the type `si_base_unit_space`. The model has been inspired by an example from the Boost MPL library [10], which implements a similar functionality in C++. This composite type is defined as:

```
si_base_unit_space(
      metre,
      kilogram,
      second,
      ampere,
      kelvin,
      mole,
      candela,
      radian,
      steradian,
      qty_type
),
```

where the attributes `metre, kilogram, ..., steradian` are of `smallint` type and express the power of the base unit dimension in the unit or quantity. The `qty_type` attribute is of `char(1)` type and is used to distinguish between various types of quantities, mainly those that are dimensionless, like the count of periodic events vs. the count of aperiodic events per one second (i.e. 1 Hz vs. 1 Bq) both defined as s^{-1}.

This type also provides the means to relate units to quantities and defines the usual comparison operators (=, <>, <, <=, >, >=) and arithmetic operators (+, -, *, /,^), that allow the addition and subtraction of equal quantities and the multiplication and division of quantities resulting in a new quantity.

2.1.2. Quantities

Quantities are stored in the table `physical_quantities`, defined as:

```
physical_quantities(
      #oid,
      quantity_name,
      quantity_space_dims
).
```

The `oid` and `quantity_name` are self-explaining, and the `quantity_space_dims` attribute is of type `si_base_unit_space` and represents the powers of the base unit dimensions.

2.1.3. SI unit prefixes

The SI prefixes are stored in the `si_prefixes` table defined as:

```
si_prefixes(
      #exponent,
      short_prefix,
      full_prefix
),
```

where `exponent` is the 10^n exponent for the particular prefix, `short_prefix` and `full_prefix` are the short and the full prefixes pre-pended to the SI unit symbols and names respectively. Two functions, `apply_short_ prefix` and `apply_full_prefix` are defined to apply the prefix symbol and the prefix name to unit symbol and unit name respectively, when given the unit and the exponent of the prefix.

2.1.4. Base SI units

The seven base (and the two supplemental or derived) SI units are stored in the `si_base_units` table:

```
si_base_units(
      base_symbol,
      #base_name,
      default_exp,
      unit_space_dims
).
```

The `base_symbol` and `base_name` are respectively the symbol and the name of the unit, like `'m','metre'` for metre, `'s','second'` for second, etc. The kilogram being a base unit and also having a prefix introduces a minor complication. As multiple prefixes are deprecated, we need to store the base symbol `'g'` and the base name `'gram'` for this unit in order to be able to construct proper names and symbols for multiplies of this unit, like `'mg'`, `'dg'`, `'dag'`, `'kg'`, etc.

However we need also express the fact that to the *kilogram*, not the *gram* is the base unit, in order to avoid mistakes in calculations. This is the objective of the `default_exp` attribute. It is a foreign key into the `si_prefixes` table and stores the exponent of the prefix, that is default for the base unit. Thus in the row storing information about *kilogram*, this attribute has the value of 3, that is the key of the `'kilo-'` prefix and in all other cases there is the value of 0 meaning no prefix.

The last attribute; `unit_space_dims` is of `si_base_unit_space` type and has the same role as the `quantity_space_dims` attribute in the `physical_quantities` table.

2.1.5. Derived SI units

The SI derived units, mainly those having a special name like the *hertz, newton, pascal, joule, watt, coulomb,* etc. are stored in the `si_derived_units` table;

```
si_derived_units(
      base_symbol,
      #base_name,
      unit_space_dims
).
```

More precisely only those units u, that can be expressed by the formula

$$u = \prod_{i \in 1,2,\ldots} u_i^{p_i}; u_i \in \text{si\_base\_units}, p_i \in \mathrm{Z} \qquad (1)$$

are stored here. Therefore units like °C, °F, years, square miles, etc. cannot be stored in this table, because the conversion between these units and the (base or compound) SI units requires addition and/or multiplication of constant values.

2.1.6. Viewing all SI units

There are two views that combine the base and derived units and the SI prefixes. The `si_units` view is basically defined as:

```
si_units=
      π(si_base_units)∪
      π(si_derived_units),
```

and the `si_prefixed_units` is defined as:

```
si_prefixed_units=π(
      si_units × si_prefixes
).
```

2.1.7. Non-SI units

As mentioned in the introduction, there are many units in use, which were not defined by the SI standard. Generally, when converting values representing a physical property expressed in SI units to values expressed in non-SI units and vice-versa a conversion function is necessary.

$$\mathbf{y}[v] = f(\mathbf{x}[u]); \mathbf{x},\mathbf{y} \in R, u, v \in U, \qquad (2)$$
$$U \textit{ is the set of physical units}$$

However, conversions between the SI units most of the non-SI units [1], [3], [5] can be preformed according to the following formula:

$$\mathbf{y}[v] = (((\mathbf{x}[u] + a_x)\cdot k_x + b_x) - b_y)/k_y - a_y;$$
$$\mathbf{x},\mathbf{y} \in P;\ u,v \in U \textit{ a set of physical units;} \qquad (3)$$
$$a_x, b_x, a_y, b_y, k_x, k_y \in R$$

Factors k_x, k_y can be expressed as fractions, both for convenience and for additional precision, when storing them in database where the precision of floating point numbers is limited:

$$k_x = m_x / d_x;\ d_x, k_x, m_x \in R;\ d_x \neq 0 \qquad (4)$$

The non-SI units are stored in the table `non_si_units` defined as:

```
non_si_units(
      base_symbol,
      #base_name,
      unit_space_dims,
      pre_mult_offs,
      multiplier,
      divisor,
      post_mult_offs
).
```

The columns `base_symbol`, `base_name`, `unit_space_dims` have the same meaning as in the previews tables, the `pre_mult_offs` is the term *a*, the `post_mult_offs` is the term *b*, and the `multiplier` and `divisor` are the terms *m, d* respectively.

We use the SI as reference unit system thus for any **x**, in unit u the result **z**, of the expression $\mathbf{z}[u_{si}] = (\mathbf{x}[u] + a_x)\cdot m_x + b_x$ must be the value of **x** expressed in SI units for the given physical quantity.

2.1.8. Named physical units

To view the physical units with a special name and prefix, like the *metre, kilonewton, milligram, terahertz, farad,* etc. (as opposed to compound units, like *metre per second squared, joule per mole, ampere per metre,* etc.) the view `named_physical_units(`

```
   symbol,
   base_symbol,
   default_exponent,
   unit_name,
   base_name,
   unit_space_dims,
   pre_mult_offs,
   multiplier,
   divisor,
   post_mult_offs
```

`)` is defined as:

```
 named_physical_units=
      π(si_units)∪
      π(non_si_units),
```

the `symbol` and `unit_name` attributes are defined as:

```
symbol=apply_short_prefix(
      base_symbol,
      default_exponent
),
unit_name=apply_full_prefix(
      base_name,
      default_exponent
).
```

Since the `si_units` view does not have the `pre_mult_offs`, `multiplier`, `divisor` and `post_mult_offs` columns they are defined as follows:

`pre_mult_offs=0`

`post_mult_offs=0`

`multiplier=`*if* `default_exponent>0` *then* $10^{\text{default\_exponent}}$ *else* `1`

`divisor=`*if* `default_exponent<0` *then* $10^{\text{default\_exponent}}$ *else* `1`

2.2. Advanced concepts

2.2.1. Compound physical quantities

In many cases it is useful to know, whether the physical quantities can be combined into other physical quantities. As mentioned before, the arithmetic operators +,-,*,/,^ are defined on the `si_base_unit_space` type. The quantities can be combined according to formula,

$$\mathbf{x} = \prod_{i=1,2,3,\ldots} \mathbf{q}_i^{p_i}; \mathbf{x},\mathbf{q}_i \in \text{physical\_quantites}; \quad (5)$$
$$p_i \in \{\ldots,-3,-2,-1,0,1,2,3,\ldots\}$$

if

$$[\mathbf{x}] = [q_1]^{p_1} * [q_2]^{p_2} * \ldots * [q_n]^{p_n}.$$

Finding all possible combinations even for a single quantity **x** by using SQL statements can require excessive amounts of time, especially if there are many records in the `physical_quantities` table, n > 2 and the set of possible values for powers p_i is large.

Therefore there is a table `quantity_compound_term_exponents` storing reasonable values for the p_i exponents and three tables, named `compound_physical_ quantity_#`, where # is 1, 2 or 3, that store the pre-calculated identifiers of quantities and the powers of the individual terms that form a valid quantity combination. The content of these tables is changed by the means of trigger functions that are executed upon insertions, deletions or modifications on the `physical_quantities` table.

2.2.2. Compound SI units

To view all compound SI units that do not have a special name but are derived from the named SI units there is a view called `si_compound_units(`

```
  symbol,
  unit_name,
  unit_space_dims,
  rank
).
```

2.2.3. All physical units

To view all physical units stored in the database and to be able to do conversions between them, the view:

```
physical_units(
   symbol,
   base_symbol,
   default_exponent,
   unit_name,
   base_name,
   unit_space_dims,
   pre_mult_offs,
   multiplier,
   divisor,
   post_mult_offs
).
```

2.2.4. Conversion between physical units

A function defined as:

```
simple_unit_conversion(
      dst_unit_name,
      src_unit_name,
      src_value
```

`)` is provided for convenience to simplify the conversions of values between various units visible through the `physical_units` view. Table `unit_conversion_functions` stores the triples (*destination units, conversion function, source units*) for conversions where the formula (3) is not applicable. The function defined as:

```
special_unit_conversion(
      dst_unit_name,
      src_unit_name,
      src_value
```

`)`, does the conversions by using the functions stored in this table.

A general conversion function `unit_conversion(`

```
  dst_unit_name,
  src_unit_name,
  src_value
)
```

uses both the `simple_unit_conversion` and the `special_unit_conversion` functions at need.

3. IMPLEMENTATION AND EXAMPLES

A proof-of-concept implementation in the PostgreSQL database system can be found and downloaded at http://kifri.fri.uniza.sk/~chochlik/dbs/psql_unit_system.tar . This archive contains a database dump that can be used to create a set of types, tables, functions, operators and triggers, described by this paper. It has been developed and tested under the version 8.2.6. Note that it currently does contain only a subset of common quantities and physical units.

Here follow some examples of the most important use-cases.

3.1. Selecting all units of energy

```
SELECT unit_name, symbol
FROM physical_units
JOIN physical_quantities
ON(unit_space_dims =
quantity_space_dims)
WHERE quantity_name = 'Energy';
```

3.2. Selecting compound units of power

```
SELECT unit_name, symbol
FROM si_compound_units
JOIN physical_quantities
ON(unit_space_dims =
quantity_space_dims)
WHERE quantity_name = 'Power';
```

3.3. Selecting non SI units of time

```
SELECT base_name, base_symbol
FROM non_si_units
JOIN physical_quantities
ON(unit_space_dims =
quantity_space_dims)
WHERE quantity_name = 'Time';
```

3.4. Selecting names of units to which a value in metres can be converted

```
SELECT DISTINCT uy.unit_name
FROM physical_units ux
JOIN physical_units uy
USING(unit_space_dims)
WHERE ux.unit_name = 'metre';
```

3.5. Converting a value in leagues to nautical miles

```
SELECT simple_unit_conversion(
      'nautical mile',
      'league',
      1270
);
```

3.6. Converting a value in minutes to years

```
SELECT simple_unit_conversion(
      'year,
      'minute',
      100000
);
```

Several other example SQL queries demonstrating the usage of the tables, views, operators and functions, can be found in the archive at http://kifri.fri.uniza.sk/~chochlik /dbs/psql_unit_system_samples.tar .

4. FUTURE WORK

Further specialized tables for storing other types of quantities, units and prefixes, could be devised and could be united with the current `physical_units` view into a general `units` view and the `unit_conversion` function could be extended to do conversions between these functions as well.

An explicit relationship between the columns of database tables or the results and parameters of functions and the units in which they are expressed, can provide very useful semantic information. Such information would allow creating very flexible database applications that could derive and convey much more data from many existing databases.

REFERENCES

[1] The NIST Reference on Constants, Units and Uncertainity, International System of units: http://www.physics.nist.gov/cuu/Units/units.html, Retr. on 2008-02-29

[2] http://en.wikipedia.org

[3] http://www.bipm.org/en/si/si_brochure/, Retr. on 2008-02-29

[4] http://www.iso.org/iso/iso_catalogue.htm, Retr. on 2008-03-03

[5] General Tables of Units of Measurement, NIST, United States Government, http://ts.nist.gov/WeightsAndMeasures/Publications/upload/h4402_appenc.pdf Retr. on 2008-03-03

[6] http://www.postgresql.org/

[7] Szirtes, T.: Applied Dimensional Analysis and Modeling, McGraw-Hill Professional, 1998

[8] Gruber, M.: Mistrovství v SQL, Soft Press, 2004

[9] Hernandez, M. J. Viescas. J. R.: Myslíme v Jazyku SQL, Grada, 2005

[10] http://www.boost.org/libs/mpl/doc/tutorial/dimensional-analysis.html

[11] http://www.gnu.org/software/units/

[12] Matiaško, K., Zábovská, K., Zábovský, M,: Building the Unified Data Access Framework, Journal of Information, Management and Control Systems, Vol. 2, No. 2, 2004

Received March 10, 2008, accepted April 28, 2008

BIOGRAPHY

Matúš Chochlík was born in 1981. In 2005 he graduated in the study field of Information And Management Systems at the Faculty of Management and Informatics of the University of Žilina. Since 2005 he is a PhD. Student at this faculty. His research and employment activities include object-oriented design and programming, reflection and meta-programming, object-oriented and distributed databases and visualization.