FFII Database Design Principles

Hartmut PILCH
http://a2e.de/i2p/ffiidb/meta

April 17, 2009

During the years 2002-5 the FFII codified much of the information regarding internal structures in terms of relational data, accessible by Standard Query Language (SQL), using PostgreSQL, so as to allow the most general possible use of the data. We accumulated an internal database for member and finance data. To allow for the greatest possible freedom of programming (and customisation of existing programs), we imposed a certain discipline on ourselves.

Contents

1 Design Principles

- A table is like a hash-type variable: it has a name, a set of keys and a set of values.
- Each record is identified by a unique set of keys, each table is indexed on this key-set
- Apart from these key columns, the table should also have at least one value column. One common practise is to have one extremely human-oriented value column, i.e. for remarks (text type), and one extremely machine-oriented one where programs can enter minimal information without overwriting anything valuable, i.e. of timestamp type.
- The keys are designed for human use (not automatically generated integers)

- Columns have unique names throughout the database. If a column name occurs in two tables, then this means that the tables should be naturally joined via this column.
  - We never use column names like `remark`, but rather `mailrem`, `adrrem` etc
- Table names and column names are all not more than 8 characters long, inline documentation on them is available in the database, invocable with `\d+` and `\dd` in the psql client
  - Table and column names are to be understood as symbols rather than as descriptions. For descriptions, we use the PostgreSQL documentation strings.
- In key columns, usually only a limited set of values can be used. This set is typically defined by a separate table of the same name in which a column of this name is defined as primary key, and any further info that programs might use, e.g. rules for description and validation, is stored.
- The scheme is moreover multilingually described in detail in a separate set of tables (meta-tables, e.g. `tabflds`)
- An simpler text-file based description of the scheme is also possible[1]
- Using such a machine-parsable scheme description, an application language can use very concise commands for editing the database[2].

2 Application Examples

Some examples can be found in

- Monthly Debiting Procedures
- a2e PHP Users – this user managment system can be easily adapted, by configuration files, to any database schema that adheres to the design principles
- Data input via Deplate – write a thesis or manual and silently embed database insert/update operations into it: this works only with the kind of database structure that we are advocating here.

3 Resources on Database Design

- Perl Modules: apart from A2E::Daba these seem not to match the above design principles but rather to assume an anonymous serial primary key or, in the case of Tie::DBI, a single key field

[1] see e.g. http://old.ffii.org/assoc/verbdefs.txt

[2] as practised e.g. in a Deplate input syntax (implemented in Ruby) and some Perl modules

- Perlmonks discussion: Tie::DBI vs Class::DBI
- DBIx::Abstract
- Alzabo
- Class::DBI
-