uhm, localization at the database level? Isn't that something that is supposed to take place at the client side, i.e. the side that typically has the different locales to deal with?

Yes, it is typically the end-user application that contains the code and data necessary to support whatever locale. This localization is, however, only dealing with the content of the database, i.e. the actual rows of the various tables. The meta data, which is the data that provides access to the content data, is typically provided in one language only. Maybe its time for a small example:

create table colors (red integer, green clob, blue varchar (100000));

A table, called colors, is created with three columns: red which is of type INTEGER, green which is of type CLOB (Character Large OBject), and finally blue which is of type CHARACTER VARYING with a maximum length of 100000 characters.

The above example and the following examples are all given using what is known as SQL (Structured Query Language), but lets leave any further SQL details out for now.

To fetch all rows, i.e. the content data, one would issue a so called select statement like:

select red, green, blue from colors;

The content data for all rows in the table is returned to the client side application, which can do whatever with the data (incl. any localization) before they are presented to the user.

In many cases the end-user doesn't see anything but the content data, i.e. there is no need to deal with the meta data directly. There are situations, however, where end-users may want to deal with the meta data. Take for example an application that pulls the meta data from a database and lets
the end-user build e.g. select queries via a graphical user interface. This at least offer the end-user the comfort of not having to remember the spelling of the SQL keywords like select and from. The mentioned applications will typically allow the end-user to build a socalled model. Such models often provide name mapping, i.e. mapping the meta data names into localized names.

So far so good, but lets assume that a given danish end-user doesn't know a word of english, then even the words colors, red, green, blue are gibberish to him/her. Returning to the title of this story and the rescue is close. The latest ANSI/ISO standard for SQL is called SQL 92 and it contains two concepts - SCHEMA and VIEW - that allow us to offer "Localization at the Database Level". The idea being to offer a schema for each lanaguage, with the actual tables etc. residing in one schema and then referenced via views from the other schemas. Lets illustrate with an example:

```
create schema English


create table colors(red integer, green clob, blue varchar(100000));


create schema Danish


create view farver(rød,    grøn, blå) as select    red, green, blue from
English.colors;


create schema Deutch


create view farben(rot, grön, blau) as select red, green, blue from
English.colors;
```

By picking e.g. Danish as schema name, the end-user will only see the localized meta data - farver, rød, grøn, blå - and not the original, so to speak, English words. The actual end-user application can easily localize words like e.g. integer, but it simply can't localize whatever names the database schema designer has concocted.

If the implementation of schemas and views is correct, no performance overhead worth mentioning is introduced. The typical operations like INSERT, UPDATE and DELETE can be applied to the views, simply because the views only represent front-ends to the real schema and its tables, i.e. no copies floating around.

The author:

Geert B. Clemmensen is the President of the Frontline Software Group, with offices in Copenhagen, Denmark and Austin, Texas, USA (and soon in Germany). Frontline Software, the zero-downtime experts, are planning to release a high-performance SQL 92 compliant database server for Rhapsody 1.0 when it is released. You can reach Geert at

gclem@frontline-software.dk.