Localizing databases

Sisulizer

29 May 2011

Introduction

This document describes the steps that you should perform when localizing your databases. Localization in general means translating your application into another language. In the other words it means translation of user interface elements, resource strings, images and help files. For example if you have written your original application and help files in English and you want to start selling your application in Germany you have to translate your application and help files to German. If your application uses databases that contain instructions or user interface strings you also have to localize the database. For example you might have a product catalog table that contains names, prices and descriptions of the products you sell. If you have localized your user interface that shows the product data it makes sense to localize the data as well. This makes it possible to show German product description when data is viewed through German user interface.

The first half of the document deals with internationalization and common concept of database localization. The second half is Sisulizer specific and it shows the localization process.

Process

When you localize an application you do not have several different options about how to localize. This is because each platform has its standard way to localize. In Windows applications it is to put user interface into Windows resource data (.rc) and to localize it. In Java application it is to use Java resource data (.properties) and translate them. No database has a standard localization method. Some database vendors recommend using certain localization method but you are not bound to it. The reason for this is that there is no single superior localization method for databases. Instead there are several different ways to localize data in database.

The best way to look about database localization is to go through an example. Our sample database has a Country table that contains information about countries. Each record contains data of a single country. The table structure is:

```
CREATE TABLE Country
(
  Id INTEGER NOT NULL,
  Name VARCHAR(50) NOT NULL,
  Population INTEGER NOT NULL,
  Capital VARCHAR(50) NOT NULL,
  Description VARCHAR(200) NOT NULL,
  PRIMARY KEY(Id)
);
```

If we look a sample data it looks like this

Id	Name	Populati	Capital	Description
0	United States	297	Washington, D.C.	The United States of America is a federal republic of 50 states, located primary on central North America.
1	Germany	82	Berlin	Germany or the Federal Republic of Germany is one of the world's leading industrialized countries, located in the heart
2	Japan	127	Tokyo	Japan is a country on the western edge of the Pacific Ocean.

Id field is the key value. Name field contains the name of the country. Population contains the population in millions. Capital tells the name of capital city of the country. Description is the short description of the country. Id and Population fields are language independent. They are both numbers and you should not translate numbers. However Name, Capital and Description fields contains data in English and they should be localized. Those fields are marked with bold typeface in the above SQL script. There are several ways to localize tables. The following paragraphs describe the most common ones.

Field localization

In this method the table contains localized fields. The localized fields are similar to the original field except they contain data in different language. For example if the original language is English and you want to localize the database to German and Japanese you add German and Japanese fields for those fields that contain strings to be localized. The following SQL script contains modified Country table to have English (default), German, Finnish and Japanese fields.

```
CREATE TABLE Country
(
  Id INTEGER NOT NULL,
  Name VARCHAR(50) NOT NULL,
  Name_de VARCHAR(50),
  Name_fi VARCHAR(50),
  Name_ja VARCHAR(50),
  Population INTEGER NOT NULL,
  Capital VARCHAR(50) NOT NULL,
  Capital_de VARCHAR(50),
  Capital_fi VARCHAR(50),
  Capital_ja VARCHAR(50),
  Description VARCHAR(200) NOT NULL,
  Description_de VARCHAR(200),
  Description_fi VARCHAR(200),
  Description_ja VARCHAR(200),
  PRIMARY KEY(Id)
);
```

The advantage of field localization is that you do not have to change your primary key and the amount of records in the table does not change. All localized data is in these extra fields. The disadvantage is that you have to add new fields into your table every time you add a new language. In some database you just cannot add fields after you have creates your database. This method works best when you know at the time of database creation what language you will support.

When you localize the above table you have to set localized fields to contain translations of the original field. Whenever you edit or change the table you first edit the original fields and then perform the same steps for the localized filed. If you use Sisulizer you only have to edit the original English fields. Sisulizer will automatically update the localized fields during build process.

You can find a sample database in `<sisulizer>\Database\Country\<db>\Field` directory. <db> is the name of you database.

Table localization

Table localization is similar to field localization except it does not put the localized fields into the main table but uses language specific tables that equal to the main table but contains only key field and the localized fields.

```
CREATE TABLE Country
(
  Id INTEGER NOT NULL,
  Name VARCHAR(50) NOT NULL,
  Population INTEGER NOT NULL,
  Capital VARCHAR(50) NOT NULL,
  Description VARCHAR(200) NOT NULL,
  PRIMARY KEY(Id)
);

CREATE TABLE CountryDe
(
  Id INTEGER NOT NULL,
  Name VARCHAR(50) NOT NULL,
  Capital VARCHAR(50) NOT NULL,
  Description VARCHAR(200) NOT NULL,
  PRIMARY KEY(Id)
);

CREATE TABLE CountryFi
(
  Id INTEGER NOT NULL,
  Name VARCHAR(50) NOT NULL,
  Capital VARCHAR(50) NOT NULL,
  Description VARCHAR(200) NOT NULL,
  PRIMARY KEY(Id)
);

CREATE TABLE CountryJa
(
  Id INTEGER NOT NULL,
  Name VARCHAR(50) NOT NULL,
  Capital VARCHAR(50) NOT NULL,
  Description VARCHAR(200) NOT NULL,
  PRIMARY KEY(Id)
);
```

There is one localized table for each language. The localized table contains the same key (Id) but it only contains the fields that need to be localized. In our sample the localized tables do not contain Population field.

When you localize the above table you just have to add equal amount of row to each table and make sure that fields in the localized tables contain translated values. If you use Sisulizer you only have to edit the original table. Sisulizer will automatically populate tables and update fields during build process.

When Sisulizer performs the build process it adds the rows of the original table into the localized table. The primary key values will be the same but the field values will contain translated values. Because the primary key values must match the original table Sisulizer must be able to edit them freely. This is why you cannot use auto increment fields in the primary key of the localized table. The original table can use auto increment fields but localized table cannot use. You have to replace auto increment fields with standard integer fields.

You can find a sample database in `<sisulizer>\Database\Country\<db>\Table` directory. <db> is the name of you database.

Row localization

Both field and table localization method required your to change your database structure every time you add a new language. It is possible to make a language independent table structure. However this requires that you have to change your primary key. In the above samples we always had the same primary key that was an integer value starting from 0. If we combine this value with the language code we can use the same integer value with several rows as long as the language codes are different.

```
CREATE TABLE Country
(
  Id INTEGER NOT NULL,
  Language VARCHAR(10) NOT NULL,
  Name VARCHAR(50) NOT NULL,
  Population INTEGER NOT NULL,
  Capital VARCHAR(50) NOT NULL,
  Description VARCHAR(200) NOT NULL,
  PRIMARY KEY(Id, Language)
);
```

The primary key is combination of Id and Language fields. This makes it possible to have same id in several languages. The picture below contains the country table localized into English, German, Finnish and Japanese.

Id	Language	Name	Populati	Capital	Description
0	de	Vereinigte Sta	297	Washington, D.C.	Die Vereinigten Staaten von Amerika sind ein Zusammenschluss von 50 Staaten hauptsächlich auf dem Nordamerikanis
0	en	United States	297	Washington, D.C.	The United States of America is a federal republic of 50 states, located primary on central North America.
0	fi	Yhdysvallat	297	Washington DC	Amerikan yhdysvallat on 50 osavaltion liittovaltio, joka sijaitsee suurelta osin Pohjois-Amerikassa
0	ja	アメリカ合衆	297	ワシントンDC	アメリカ合衆国は50の州からなる連邦共和国であり、主として北アメリカの中央部に位置しています。
1	de	Deutschland	82	Berlin	Die Bundesrepublik Deutschland ist eine der führenden Industrienationen im Herz von Europa.
1	en	Germany	82	Berlin	Germany or the Federal Republic of Germany is one of the world's leading industrialized countries, located in the heart
1	fi	Saksa	82	Berliini	Saksa tai Saksan liittotasavalta on yksi maailman johtavista teollisuusmaista Euroopan sydämessä.
1	ja	ドイツ	82	ベルリン	ドイツ、またはドイツ連邦共和国は世界の主要な工業国の一つであり、ヨーロッパの中心に位置してい
2	de	Japan	127	Tokio	Japan ist ein Land am westlichen Rand des Pazifischen Ozean.
2	en	Japan	127	Tokyo	Japan is a country on the western edge of the Pacific Ocean.
2	fi	Japani	127	Tokio	Japani on maa tyynen valtameren länsipuolella.
2	ja	日本	127	東京	日本は太平洋の西端に位置する国です。

Row localization is possible to implement without combined primary key. In this approach we have a single filed primary key (RowId) that is an auto increment field. In addition of the primary key we have ResourceId key that specified the item and Language keys that contains the language code. ResourceId and Language code together make a unique row id. ResourceId contains the context and Language contains the language.

```
CREATE TABLE Country
(
  RowId INTEGER NOT NULL,
  ResourceId INTEGER NOT NULL,
  Language VARCHAR(10) NOT NULL,
  Name VARCHAR(50) NOT NULL,
  Population INTEGER NOT NULL,
  Capital VARCHAR(50) NOT NULL,
  Description VARCHAR(200) NOT NULL,
  PRIMARY KEY(RowId)
);
```

Here is a sample data.

Row	ResourceId	Language	Name	Populati	Capital	Description
1	0	en	United States	297	Washington, D.C.	The United States of America is a federal republic of 50 states, located primary on central North America.
2	1	en	Germany	82	Berlin	Germany or the Federal Republic of Germany is one of the world's leading industrialized countries, located in the heart
3	2	en	Japan	127	Tokyo	Japan is a country on the western edge of the Pacific Ocean.
31	1	fi	Saksa	82	Berliini	Saksa tai Saksan liittotasavalta on yksi maailman johtavista teollisuusmaista Euroopan sydämessä.
32	2	fi	Japani	127	Tokio	Japani on maa tyynen valtameren länsipuolella.
33	0	fi	Yhdysvallat	297	Washington DC	Amerikan yhdysvallat on 50 osavaltion liittovaltio, joka sijaitsee suurelta osin Pohjois-Amerikassa
34	1	de	Deutschland	82	Berlin	Die Bundesrepublik Deutschland ist eine der führenden Industrienationen im Herz von Europa.
35	2	de	Japan	127	Tokio	Japan ist ein Land am westlichen Rand des Pazifischen Ozean.
36	0	de	Vereinigte Sta	297	Washington, D.C.	Die Vereinigten Staaten von Amerika sind ein Zusammenschluss von 50 Staaten hauptsächlich auf dem Nordamerikanis
37	1	ja	ドイツ	82	ベルリン	ドイツ、またはドイツ連邦共和国は世界の主要な工業国の一つであり、ヨーロッパの中心に位置してい
38	2	ja	日本	127	東京	日本は太平洋の西端に位置する国です。
39	0	ja	アメリカ合衆	297	ワシントンDC	アメリカ合衆国は50の州からなる連邦共和国であり、主として北アメリカの中央部に位置しています。

The ResourceId field can have a foreign key to the table that contains the valid resource ids.

```
CREATE TABLE Resource
(
  Id INTEGER NOT NULL,
  Comment VARCHAR(50),
  PRIMARY KEY(Id)
);

CREATE TABLE Country
(
  RowId INTEGER NOT NULL,
  ResourceId INTEGER NOT NULL,
  Language VARCHAR(10) NOT NULL,
  Name VARCHAR(50) NOT NULL,
  Population INTEGER NOT NULL,
  Capital VARCHAR(50) NOT NULL,
  Description VARCHAR(200) NOT NULL,
  PRIMARY KEY(RowId),
  FOREIGN KEY(ResourceId) REFERENCES Resource(Id)
);
```

The advantage of this row id method is that our primary key is a single field. Some database designers require this. However this is not as elegant as the original row localization. We recommend using this method only if you cannot include language if into your primary key.

You can find a sample database in `<sisulizer>\Database\Country\<db>\Row` and `<sisulizer>\Database\Country\<db>\RowId` directories. <db> is the name of you database.

Database cloning

The final localization method is the easiest one. Just make a copy of the database. The localization database is identical to the original but it contains translated field value. Unfortunately this can be used only when dealing local databases that store data in a file or files. Remember that Sisulizer cannot create new database into database server. Neither can it add new tables. Sisulizer can only update fields and add rows into existing table. This is why this method can be only used with local databases such as Access, SQLite and SQL Server Compact.

You can find a sample database in `<sisulizer>\Database\Country\Access\Clone` directory.

Character sets

You should use Unicode with database just as with applications and data in general. Unfortunately some databases do not support Unicode fields and some databases have supported them in a very short time. This is why most databases at the moment do not use Unicode but Ansi strings that use a language specific code page. If you database uses Ansi strings it is no obstacle for localization. Sisulizer can read and write Ansi strings. However you have to make sure that your application can handle localized data that is encoded with a different code pages.

If you use Unicode you do not have to worry about code pages. Whenever you read and write data to the database you use the same encoding: Unicode. If your database does not support for Unicode you can store data as UTF-8. UTF-8 is a Unicode encoding that is used for example in XML files and in web. The good thing about UTF-8 is that it does not require 2 byte characters in string fields but can be store into 1 byte per character Ansi fields. So any database string field can store UTF-8 data.

Using Sisulizer

We have now internationalized our database. It is time to localize it. The rest of the document is used to describe how to use Sisulizer to localize a database.

Sisulizer supports all database localization methods describe in the previous chapters. Sisulizer lets you localize all string, memo and blob fields. You can have plain string, combined strings, XML data, HTML data, or image data in the database. Sisulizer supports all major image formats such as JPG, PNG, BMP, GIF and icons. Sisulizer supports both Unicode and Ansi encoded string fields.

Things to remember

When Sisulizer scan the database that is localizing it reads all strings into memory. This sets some limits to the size of the database you can localize with Sisulizer. If you have a multi-gigabyte database containing hundreds of thousands records or more you cannot localize it with Sisulizer. Fortunately such databases very seldom require localization. In most cases what you want to localize is a moderately sized table that contains instructions, strings, product data, etc.

Database localization is very new concept and very few localization tools support it. There are several localization methods, dozens of different database engines, and millions of different database structures. We cannot test them all. We have tested localization of Access, MySQL, SQL Server, Interbase, Paradox, dBase, DBISAM, Firebird, PostgreSQL and Oracle tables (most heavily tested databases first). You might face some glitches when you try to localize your database especially if you use database that we have tested only little. In that case please contact us through our support forum. We will help you in your localization process.