Converting non-Latin characters to Unicode in SQL databases

I've been working on an interesting task recently for a client. My customer has a logistics application with a SQL 2000 backend and is upgrading to a new version of the application. The new application version runs on SQL Server 2005 and is Unicode-compliant, which is the key to this tale.

The (Dutch) customer has a subsidiary in Poland that uses the logistics application with a remote connection, so the data is accessed from Poland but is physically stored on a database server in Holland. The present database is non-Unicode compliant, so all strings are stored in *char* and *varchar* data type fields. The Dutch server is configured with a "normal" Western codepage 1252 and is able to support all characters for English-speaking and Western European subsidiaries, but for Poland the system breaks down a bit.

The Polish alphabet is a superset of the Latin alphabet and has 32 characters, so if a Polish user on a Polish-configured Windows workstation enters "WARSZAWA" in an address field that's what gets saved in the database and displayed at all locations and in all circumstances.

If, however, she types in "ŁÓDŹ" or "KĄTY WROCŁAWSKIE" what gets saved in the Dutch database is "£ÓD" and "K¥TY WROC£AWSKIE" respectively. The Polish user still sees the correct Polish characters on her display, but a Dutch or French user won't be able to read the Polish names or addresses properly.

As long as the various subsidiaries don't have to deal with each other's information there's no problem, but once they need to look at each other's data it quickly becomes unworkable. These problems only become more urgent when non-Latin based alphabets and character sets like Cyrillic or Arabic come into play.

So, my job was to ensure that the entire database content was migrated from a non-Unicode-compliant format to a Unicode-compliant format, and thereafter to convert the "faulty" Polish characters to the "true" Polish characters so that the Polish character set was readable for all users in all circumstances.

The first step was relatively easy: I set up a copy of the original database with NCHAR and NVARCHAR fields replacing the CHAR and VARCHAR fields from the original, then copied all the data from the original to the Unicode database. This didn't change the saved character strings, of course, so "£ÓD" was still "£ÓD" in the new database. A new entry of "ŁÓDŹ" was saved and displayed properly, but the existing data still needed to be converted.

I first searched to see if I could identify strings in one field containing the pound character "£" as follows:

```
select unicode('£')
```

...returns **163**, so...

```
select distinct [city]
from  [DataTable]
where charindex(nchar(163), [city]) > 0
order
by    [city]
```

...returned this list:

```
City
====
£¥CK
£¥KORZ
£APANÓW
£ASZCZÓW
£ÊCZNA
£ÊCZYCA
£OCHÓW
£ÓD□
£ODYGOWICE
£OM¯A
...
```

I then looked up **163** on the Polish code page layout and saw that it mapped to the Polish character 'Ł'.

```
select unicode(N'Ł')
```

...gave me **321**, so I now knew that every occurrence of **nchar(163)** in strings needed to be replaced with **nchar(321)**.

```
Select distinct [city], replace([city], nchar(163), nchar(321)) as
      UpdatedCity
from  [DataTable]
where charindex(nchar(163), [city]) > 0
order
by    [city]
```

...returned:

```
City         UpdatedCity
====         ===========
£¥CK         Ł¥CK
£¥KORZ       Ł¥KORZ
£APANÓW      ŁAPANÓW
£ASZCZÓW     ŁASZCZÓW
£ÊCZNA       ŁÊCZNA
£ÊCZYCA      ŁÊCZYCA
£OCHÓW       ŁOCHÓW
£ÓD          ŁÓD
£ODYGOWICE   ŁODYGOWICE
£OM¯A        ŁOM¯A
...
```

...which, expressed as an update statement, becomes:

```
Update [DataTable]
set    [City] =    replace([city], nchar(163), nchar(321))
where charindex(nchar(163), [city]) > 0
```

This worked well enough, but there were 16 characters (eight lower case, eight upper case) that needed replacing in over 1600 fields spread over 300 tables. It obviously wasn't desirable to carry these updates out one by one, so I wrote a script to loop through all the *nchar* and *nvarchar* fields from all tables and for each field to loop through all 16 possible character substitutions (stored in a table variable) and run the update statement for each one.

The first test update ran fine, but there were a couple of problems with the test run. The first was that some of the "faulty" characters were in fact "good" characters for other languages than Polish. The “Ê” character was in widespread use by the French subsidiary, for instance, and the “Ñ” character was used by the Spanish branch.

This was easy to fix with a filter on the land or language code fields present in the customer's database (UPDATE only where landcode = 'PL', for instance), but the second problem was subtler: the database collation (**SQL_Latin1_General_CP1_CI_AS**) is case-insensitive, so upper-case "Ñ" characters were being replaced with lower-case Polish "ń" characters.

The fix proved easy; by changing the

```
replace([field], nchar(163), nchar(321))
```

clause to:

```
replace([field] collate SQL_Latin1_General_CP1_CS_AS, nchar(163),
nchar(321))
```

...the field collation was temporarily seen as case-sensitive and so upper-case characters were ignored during the lower-case updates and vice-versa.

The city field from the first example after conversion looks like this for all users at all locations:

```
City
====
ŁĄCK
ŁĄKORZ
ŁAPANÓW
ŁASZCZÓW
ŁĘCZNA
ŁĘCZYCA
ŁOCHÓW
ŁODYGOWICE
ŁÓDŹ
...
```

The final script can be downloaded here and ran on a 26 GB database in around 11 minutes.

Paul Clancy
360Data