

Convert your Firebird applications to Unicode

Source: Firebird Conference 2009 in Munich; Holger Klemt.

If you are new to the topic the use of character sets in Firebird/InterBase® we recommend you first read the Database Technology article, [Character sets and Unicode in Firebird](#).

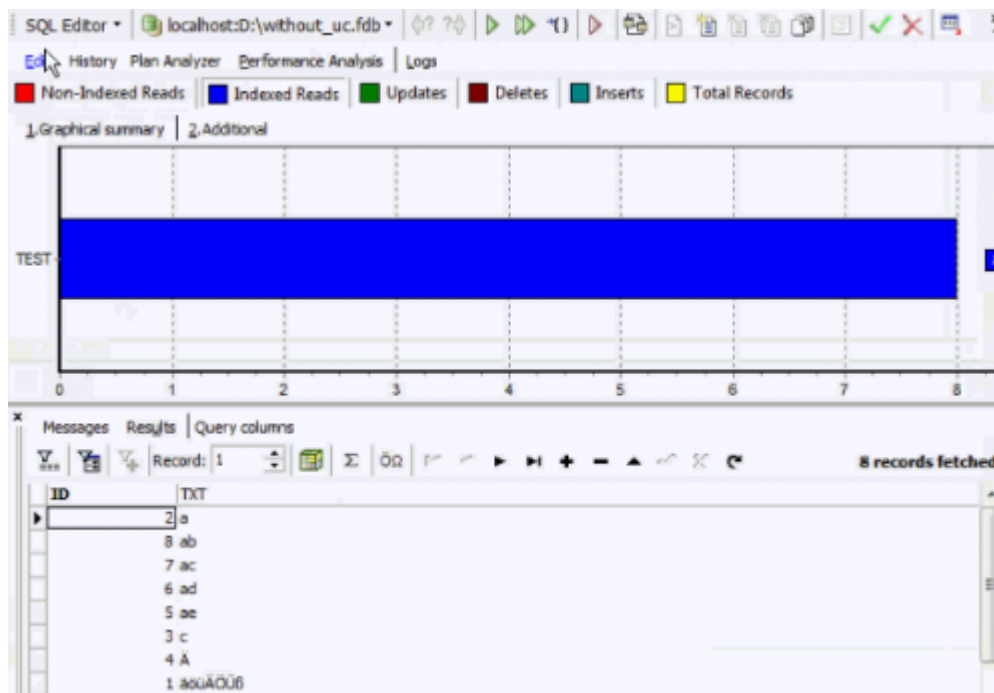
This article discusses and illustrates the use of Unicode in Firebird and Delphi. Delphi offers full Unicode support since version 2009. So if you have international applications you do not have to think about which character sets have been used to store which languages; Unicode is fully transparent. A lot of Delphi applications can now be simply recompiled and they also use Unicode in all edit controls. However using Unicode does require some background knowledge.

Using character sets in Firebird

First we will show you how to create a database with the [character set](#) most often used in Western Europe, ISO8859_1. This character set does not just offer, in addition to the usual ASCII Latin characters, special characters unique to the main Western European languages, it also holds important information regarding the sorting order of [column](#) data according to the specific rules of each of these languages. Not only understanding the order but also the byte encoding is important in order to understand how character sets really work. For example, when we create a table and add some data to it, including some special characters, all this data is stored inside the ISO character set within the first 255 characters. The ISO character set is a table that defines 255 characters which have a special meaning based on the byte code. The problem is, when we want to use the same character set, for example, for the Chinese language, this language brings up more issues regarding different characters. The simplified Chinese language has about 6,500 signs, so requires a lot more byte storage to encompass all characters.

By adding more data the impact of character sets can be more clearly seen. Think about the following: when you create a [database](#), if you do not specify the character set you may get some strange results, as the Firebird [default character set](#) is NONE. See what happens when you do an [ORDER BY](#). All special characters will be listed at the end of your result set. This does not happen, for example, when [ISO8859_1](#) is specified as the default character set.

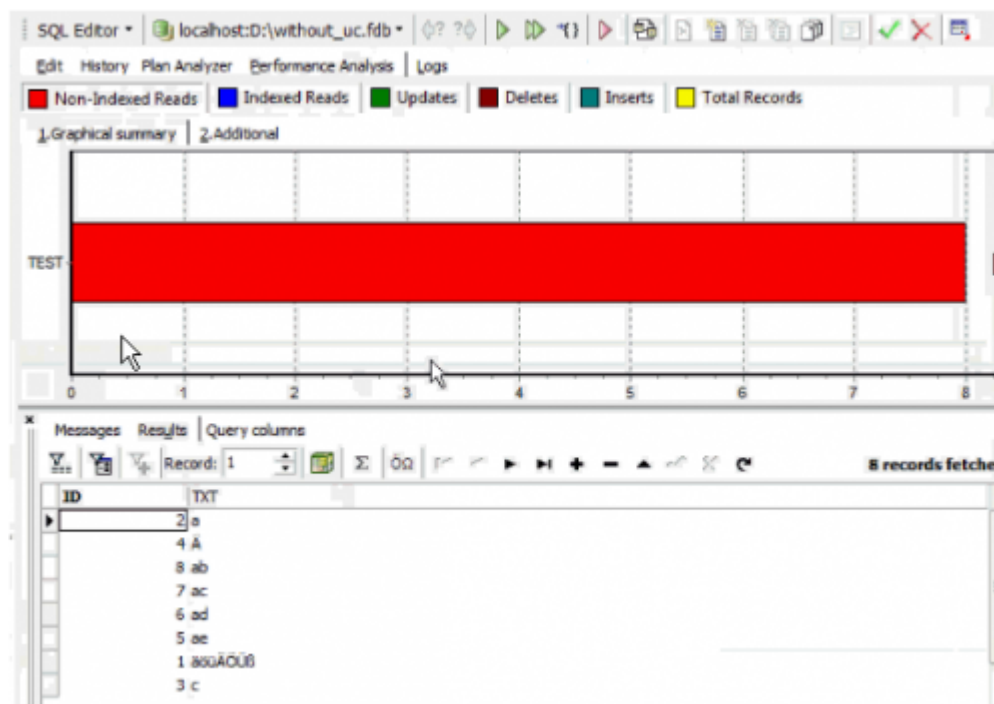
If you then proceed to create an [index](#) based on this column, you will see that by doing a simple ORDER BY, the result set is indexed but the characters are probably not ordered in the way you need them:



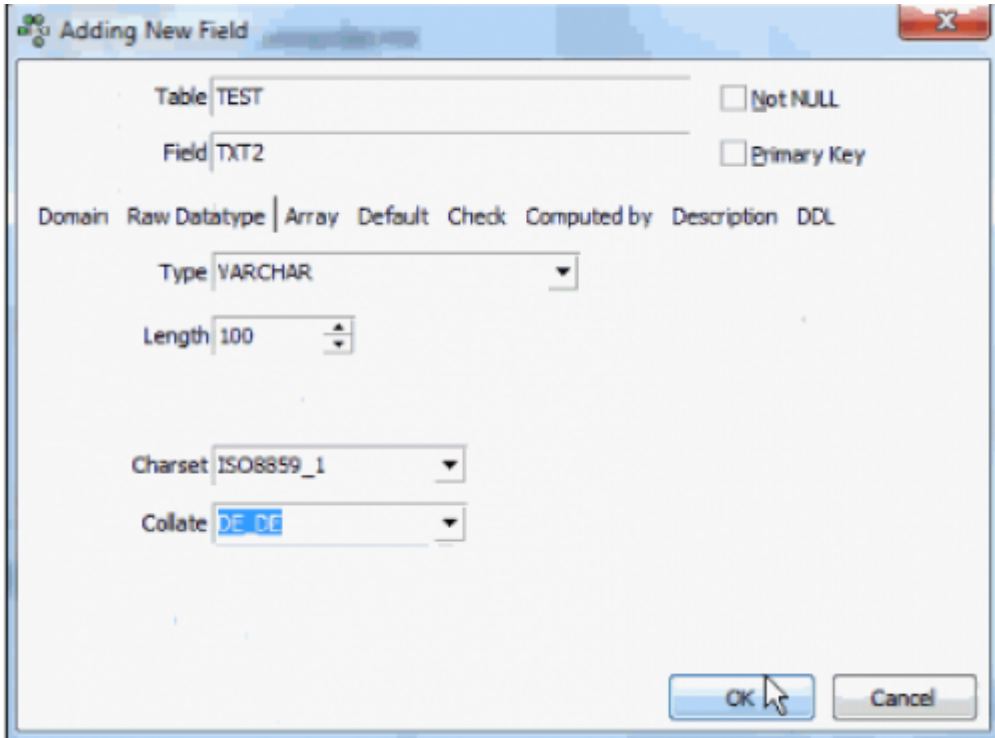
The ISO character set is implemented so that special characters of different European countries can be stored and, by defining a [collation](#), can be ordered in a specific way. To order the data according to, for example, the German norm (Ä = AE, Ö = OE, Ü = UE) you will also need to specify the collation order DE_DE.

```
SELECT * FROM TEST
ORDER BY TXT COLLATE DE_DE
```

But if we add the COLLATE DE_DE, our ORDER BY orders correctly, but no index is used:



This problem can be simply solved by adding a column, TXT2, and defining this as VARCHAR (100) with the character set ISO8859_1 with the collation DE_DE:



TXT2 contains the same data and index as TXT. When the query is now updated and the initial select repeated, it collates correctly and uses the index without the extra `COLLATE` command. If you want to see which character sets are available, you can simply open a [system table](#) inside your database, `RDB$CHARACTER_SETS`. If you look at `ISO8859_1` you will see it has the ID 21, and by referring to a second system table, `RDB$COLLATIONS`, you can see that for the `RDB$CHARACTER_SETS` ID 21, there is a wide range of collation options:

RDB\$COLLATION_NAME	RDB\$COLLATION_ID	RDB\$CHARACTER_S...	RDB\$COLLATION_ATTRIBUTES	RDB\$SYSTEM_FLAG
NXT_FRA	3	19	1	1
EN_US	14	21	1	1
SV_SV	11	21	1	1
ES_ES	10	21	1	1
DA_DA	1	21	1	1
ES_ES_CI_AI	17	21	7	1
IS_IS	7	21	1	1
DE_DE	6	21	1	1
FR_CA	5	21	1	1
FR_FR_CI_AI	18	21	7	1
ISO8859_1	0	21	1	1
EN_UK	12	21	1	1
NO_NO	9	21	1	1
PT_BR	16	21	7	1
PT_PT	15	21	1	1
IT_IT	8	21	1	1
FI_FI	3	21	1	1
DU_NL	2	21	1	1
FR_FR	4	21	1	1
CS_CZ	1	22	1	1
ISO_HUN	2	22	1	1
ISO_PLK	3	22	1	1
ISO8859_2	0	22	1	1
ISO8859_3	0	23	1	1
ISO8859_4	0	34	1	1
ISO8859_5	0	35	1	1
ISO8859_6	0	36	1	1

[back to top of page](#)

Using Unicode as the default character set in Firebird

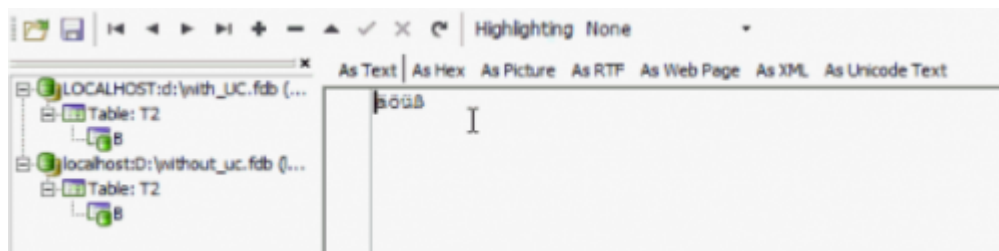
So, what do you need to consider when creating a new database? You can only have one default character set (and from Firebird 2.5 onwards you can also define one default collation). All columns that have no individual character set and collation definitions, use this default specification. But when you want to have, for example, an application based on an ISO character set, you have to decide which character set is best suited when creating the database. If you do not want to consider this when initially creating your database, you should think about setting the default character set to UTF8. Whether you want to create a new application or convert an existing application we recommend the UTF8 character set as opposed to Unicode_FSS, because of the issues already explained in [Character sets and Unicode in Firebird](#).

Comparing Unicode to ISO8859_1

So we've created a new database with a new default character set, UTF8. And within this UTF8 character set, I will now try to convert the data that is already inside my local database, for example, WITHOUT_UC.FDB is without a character set and WITH_UC.FDB is with Unicode. There are some differences, some limits: for example when I create a [blob](#):

```
CREATE TABLE T2  
(BLOB SUB_TYPE TEXT)
```

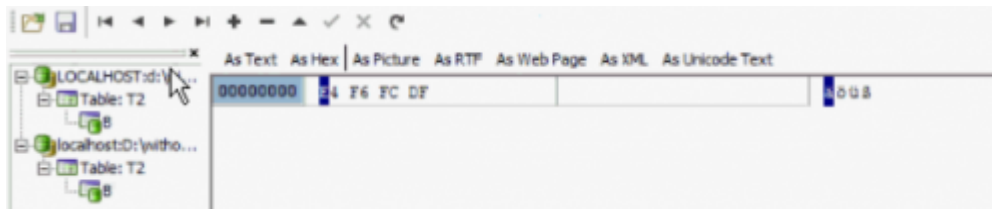
the blob `sub_type` text also depends on a character set. I create this [table](#) in both databases, and we will look at how the data is stored from a technical aspect. I add the German special characters, **ÄÖÜß**, as blob data to the blob table T2 in both databases. When I view **ÄÖÜß** on the As Text page in the IBExpert [Blob Viewer/Editor](#):



the result appears very similar in both databases. There are however technical differences, which can be seen by looking at the physical storage on the Hex page for the Unicode database:



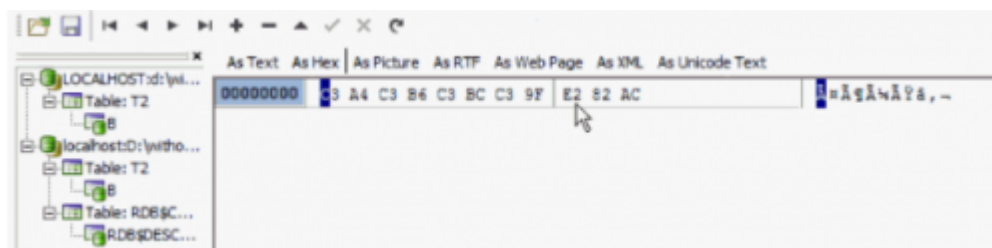
You will see that we have 9 bytes. So the **ß** entered in the database covers 3 bytes. If we look at the same data in the table in the `WITHOUT_UC.FDB` with the ISO character set:



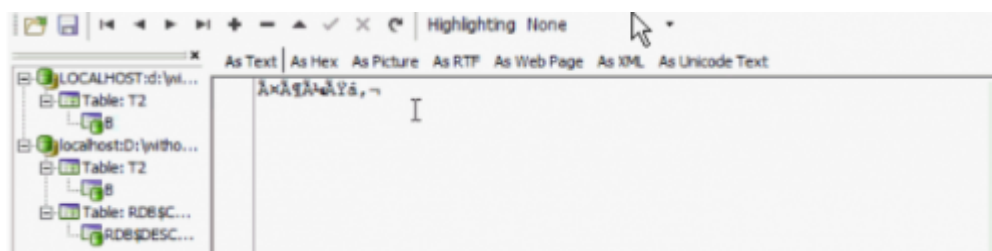
we see that the binary representation is completely different. In the system table, RDB\$CHARACTER_SETS, we can see that the ISO character set requires only 1 byte per character:

RDB\$CHARACTER_SETS				
RDB\$CHARACTER_SET_ID	RDB\$SYSTEM_FLAG	RDB\$DESCRIPTION	RDB\$FUNCTION_NAME	RDB\$BYTES_PER_CHARACTER
0	1	<null>	<null>	1
1	1	<null>	<null>	1
2	1	<null>	<null>	1
3	1	<null>	<null>	3
4	1	<null>	<null>	4
5	1	<null>	<null>	2
6	1	<null>	<null>	2
10	1	<null>	<null>	1
11	1	<null>	<null>	1
12	1	<null>	<null>	1
21	1	<null>	<null>	1
22	1	<null>	<null>	1

The UTF8 character set stores the data in *up* to 4 bytes per character. Up to is the key here. The UTF8 character set internally has all the characters below ASCII sign number 127 and these are exactly the same as they are in the ASCII character set. But on the higher level, you get some points where you have a combination of signs. For example, the German special characters are stored in 2 bytes. So there is a step in the first implementation that we see starting with C3. A4 is a sign that is Ä. The other three characters start again with C3. When I add the € sign, you will see it has 3 bytes:



So, finding binary data that was created by a Unicode-based application, inside a non-Unicode-based database will produce some strange results:



[back to top of page](#)

Converting your data to Unicode

Before you can think about converting a database from, for example, an ISO character set to a non-

ISO character set a few things need to first be considered.

The Firebird engine has a nice little feature that can, for example, connect to the Unicode database with a different database character set. For example, I create a Unicode database connection, with ISO8859_1, open the `WITHOUT_UC.FDB` and in the IBExpert [SQL Editor](#), I do an INSERT (this is an IBExpert feature, not a Firebird feature):


```
INSERT INTO [LOCALHOST:D:\WITH_UC.FDB ISO].TESTFROMISO
SELECT * FROM TEST
```

The IBExpert engine now connects to the Unicode database but with the definition of an ISO character set, and when I send data to a Unicode database using an ISO character set for my connection options, I can directly convert the data without any additional operations. So even if the source of the table does not exist in the target it is now directly transferred into the second database, and for people who not always like to manually insert the CREATE TABLE statement, you can directly copy a table from one database to another database using this IBExpert syntax, and set the alias name directly in front of the new table name where it should be created.

What we see now here in the second database is that we have a new table, TESTFROMISO, complete with all data exactly as it was in the ISO character set, physically stored in exactly the same way:

[illegible]

So you always have a UTF8 character set, a different level of byte code storing, but your existing application can open a UTF8 Unicode database with an ISO character set, and write the data as it is in the original database, for example, for conversion.



```
Script | Statements
/*****
  Generated by IBExpert 2009.11.20 20.11.2009 16:11
  *****/

SET SQL DIALECT 3;

SET NAMES ISO8859_1;

CREATE DATABASE 'localhost:D:\with_uc.fdb'
USER 'SYSDBA' PASSWORD 'masterke'
PAGE_SIZE 16384
DEFAULT CHARACTER SET UTF8;

SET NAMES ISO8859_1;

SET SQL DIALECT 3;

SET CLIENTLIB 'gds32.dll';

CONNECT 'LOCALHOST:d:\with_UC.fdb'
USER 'SYSDBA' PASSWORD 'masterke';
```

In this case we have a script that creates an empty database based on UTF8 that connects to a database based on ISO character set, and inserts the data. The data is now byte coded for the ISO character set, but within the physical storage process inside the target database it is converted from ISO to Unicode.

[back to top of page](#)

The importance of field size

The extra storage required by different character sets needs to be taken into consideration when specifying [field](#) size. For example, when you are working in a non-Unicode database:

```
CREATE TABLE TEST5 (TXT VARCHAR(32000))
```

is not a problem, even though it's not necessarily a good idea to define your [VARCHAR\(32000\)](#) that big. But you may have a problem in the Unicode database, because Firebird has to consider that the information that is stored here is could be, for example, a complete list of Chinese characters. As every single character needs up to 4 bytes, it must have the room to store up to 32000×4 . So the actual maximum VARCHAR length with UTF8 is 8191 and not 32000.

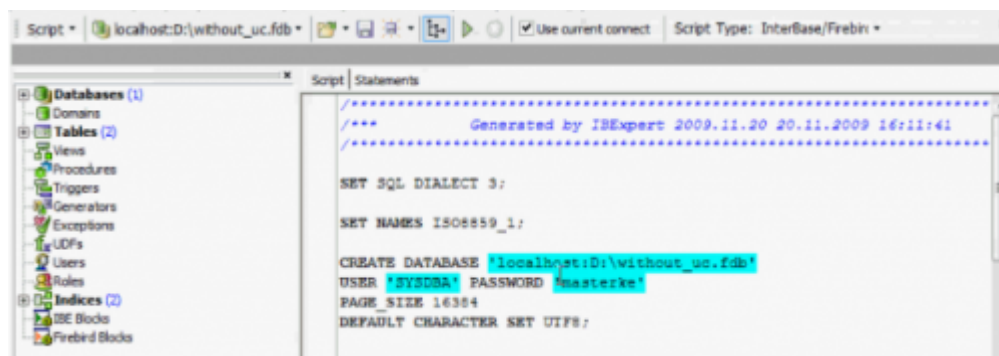
This also needs to be taken into consideration when you are using [variables](#); you have to think about the same issue. You should always take into consideration that variables might be subject to internal limitation.

And simply replacing your large VARCHARs with [blobs](#) is not necessarily the solution. Every time you change it, the versions are stored in the memory which can have a huge negative impact on server performance. So if you do work with variables that are blobs, do not manipulate them too often. Or alternatively work on VARCHAR variables and combine them in the last step to a single blob.

[back to top of page](#)

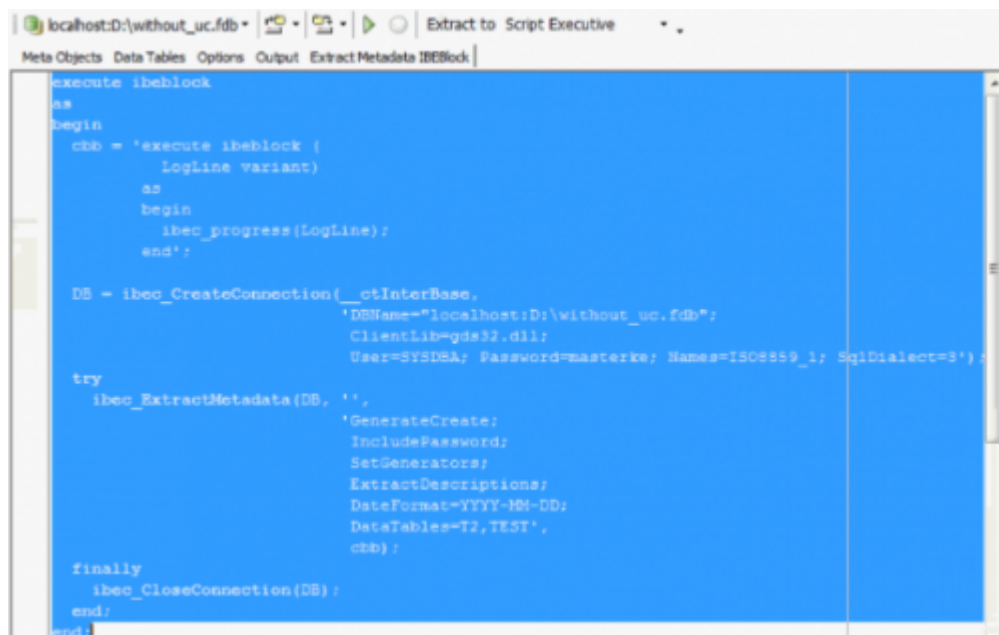
Converting your database to Unicode using Extract Metadata

To convert all your database data use IBExpert's [Extract Metadata](#) function: extract everything to a script file that is based on the original character set, manually create a UTF8 character set by changing the default character set in the create database script:



and run the script to create a fully copy of your original database with the new default character set. Should you have character set specifications at field level, these can also be altered in the script manually.

If you want to automate the conversion of the database, you can for example execute the Extract Metadata automatically using this [IBEBlock](#):



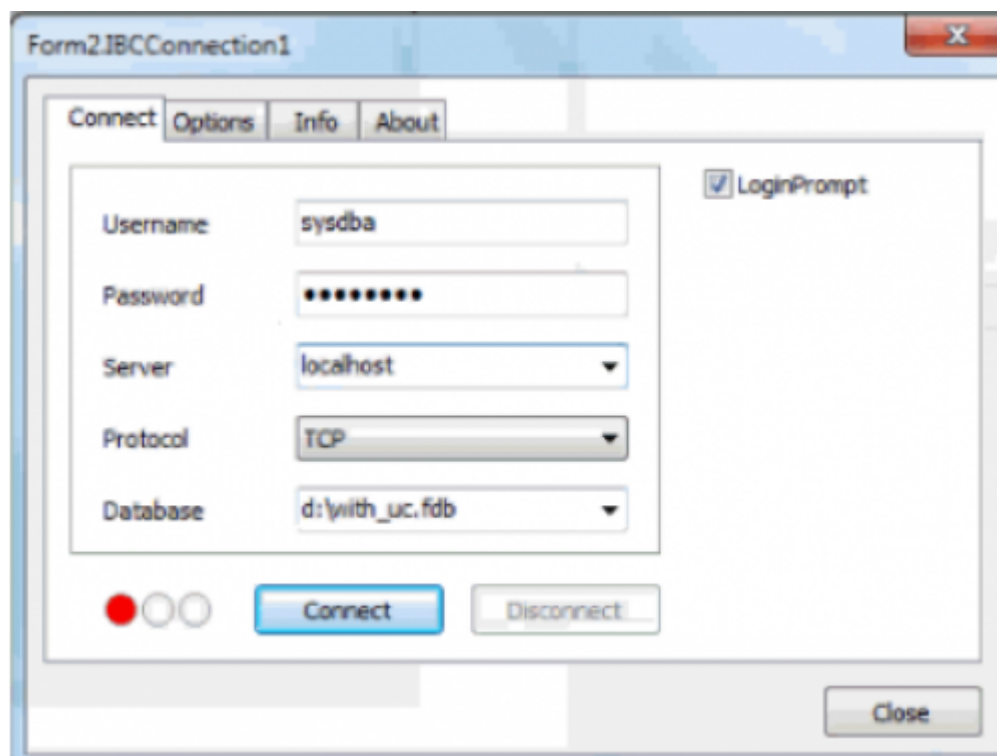
You can directly search and replace in the resulting SQL file to create a UTF8 database, but connect with your old character set, and simply add two new lines after the [CREATE DATABASE](#) statement. There are also some functionalities inside the [IBEScript](#) language, which can be used for search and replacing data in the scripts automatically. So you can see that it is really not difficult to convert a database from ISO or any other character set to UTF8 without any active interaction of an administrator or programmer.

[back to top of page](#)

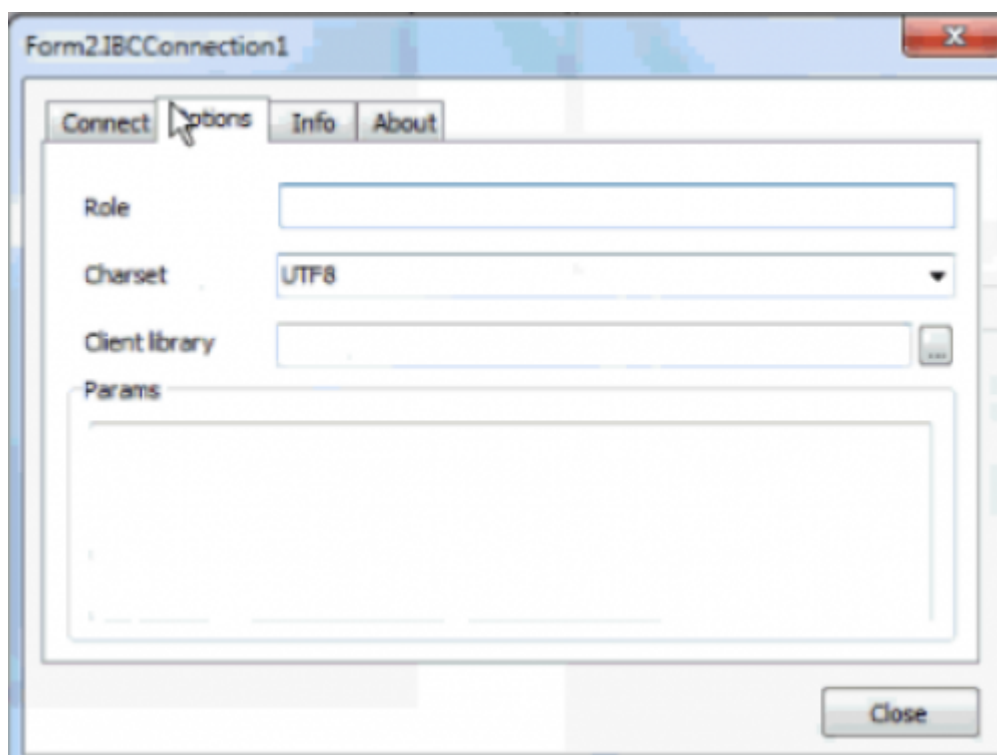
Converting the application

The next stage is to think about your application. Let us first consider how you can write applications with a database with a UTF8 character set behind it. To illustrate this we start a new project. My personal favorite for accessing data from Delphi inside a Unicode UTF8 database is using the third party component, [InterBase Access?](#).

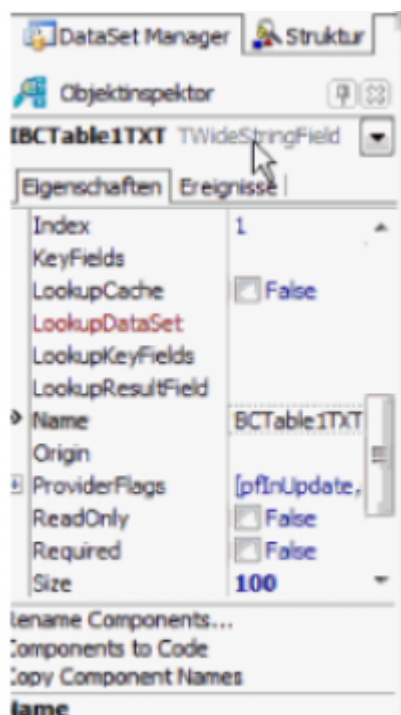
I have, for example, an *IBConnection*, I take a *TIBCTable*, and I connect to the database:



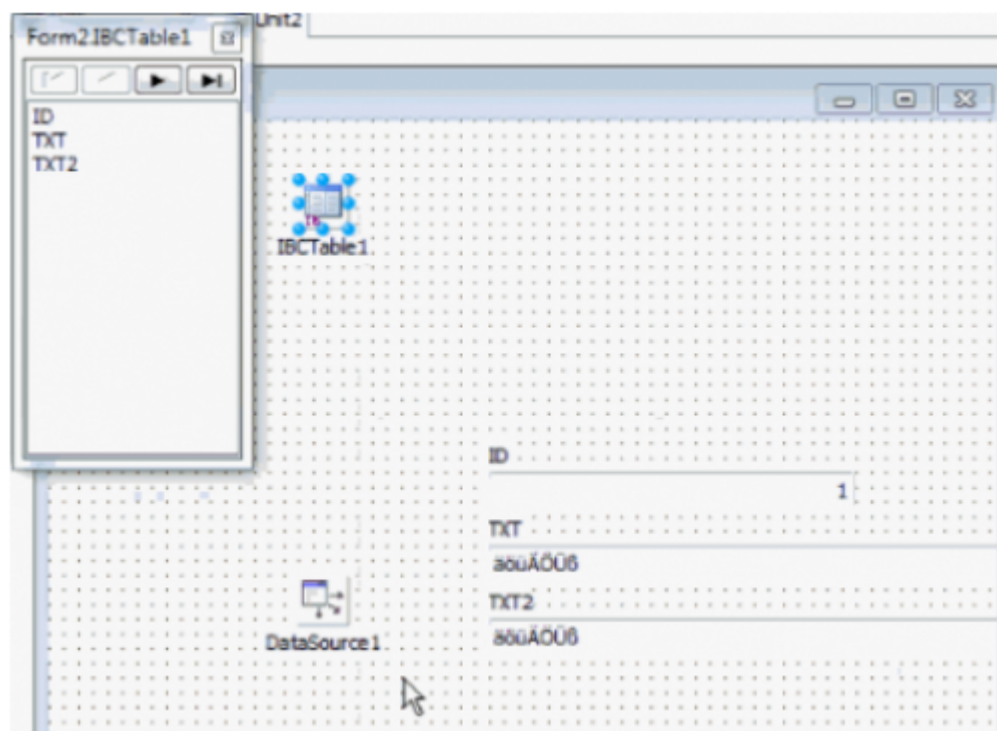
I specify the UTF8 character set:



and I connect. In order to view the internal representation as Unicode, go back to the connection and specify that this is Unicode in the options. If we then go back to the table and look at the *Object Inspector*, this was previously a *StringField* and now it's a *WideStringField*:



When I now add the data, the Unicode representation can be seen:



And there's one more setting you should be aware of: when you are working with blob columns, it is also necessary to add *EnableMemos*. So, if you set *EnableMemos* and *UseUnicode* to *true*, you no longer have to think about any conversion internally or externally to the database. Everything will be converted automatically.

So it's not such a difficult task to convert your database to a UTF8 database, it's also not such a daunting task to convert an application that can be compiled in Delphi 2009 or 2010, in order to use

the UTF8 character set. And, if you are able to do this, you will never have to think again about which character set to use when.

From:
<http://ibexpert.com/docu/> - IBExpert

Permanent link:
<http://ibexpert.com/docu/doku.php?id=01-documentation:01-05-database-technology:database-technology-articles:firebird-interbase-character-sets:convert-your-firebird-applications-to-unicode>

Last update: 2023/06/19 07:02

