Firebird ODBC/JDBC-Treiber-Handbuch

Release 1.0

Alexander Potapchenko
Vladimir Tsvigun
Pavel Cisar
Jim Starkey
others
Collator und Herausgeber: Helen Borrie
Übersetzung ins Deutsche: Martin Köditz

27. November 2017, Dokumentversion 1.02

Firebird ODBC/JDBC-Treiber-Handbuch

Release 1.0

27. November 2017, Dokumentversion 1.02
von Alexander Potapchenko, Vladimir Tsvigun, Pavel Cisar, Jim Starkey und others
Collator und Herausgeber: Helen Borrie
Übersetzung ins Deutsche: Martin Köditz

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

ODBC/JDBC-Treiber für Firebird-Client-Applikationen

Dieses Handbuch dokumentiert den offiziellen Treiber für die Verbindung von ODBC-fähigen Client-Anwendungen mit einer Firebird-Datenbank und implementiert die kombinierten Funktionen von dedizierten Wrappern für die Firebird C/C++-API-Funktionen mit einer ODBC-zu-JDBC-Brücke, um plattformübergreifende Verbindungen in einer Java-VM-Umgebung zu ermöglichen.

Über den Firebird ODBC-Treiber

Der Firebird ODBC-Treiber unterstützt Client-Anwendungen, die eine Verbindung zu Firebird-Datenbanken von Windows, FreeBSD, Solaris und Linux herstellen. Für Windows- und POSIX-Plattformen sind separate Kits für die Verwendung mit 32-Bit- oder 64-Bit-Clients verfügbar. Unter Windows sind die entsprechenden dynamischen `OdbcFb.dll`- und die statischen `OdbcFb.lib`-Bibliotheken sowohl in ZIP-Archiven als auch in ausführbaren Installationsprogrammen enthalten. Die POSIX-Pakete kommen entweder als Binärdateien für x86 und amd64, beide mit dem Namen `libOdbcFb.so`, oder als Quellcode-Tarball. Diese Hilfedatei ist ebenfalls in den Installationskits enthalten.

Unterstützte Features

- Kompilieren für 32-Bit- und 64-Bit-Windows-Clients auf der Microsoft SDK-Basis
- Unicode
- Thread-sichere Abfrage und andere Verarbeitung
- Erstellen von Datenbanken über die Funktionen `SQLConfigDataSource`, `SQLDriverConnect`, `SQLExecDirect`.
- Mehrere gleichzeitige Transaktionen pro Verbindung, ggf. mit unterschiedlichen Transaktionsattributen. Zum Beispiel eine schreibgeschützte Transaktion, eine oder mehrere gleichzeitige Lese-/Lesetransaktionen.
- Transparentes Verbindungs-Pooling über Transaktionseinstellungen

- Firebird-Datenbankereignisse, die von Triggern und gespeicherten Prozeduren zurückgegeben werden
- Verwendung von Microsoft ODBC-Cursorn (`odbccr32.dll, odbccu32.dll`)
- Firebird Services API (backup & restore, statistics, repair) im Zuge der Funktion `SQLConfigDataSource`
- Die Schemas SCHEMA oder OWNER für Fälle in denen ein Schema für Cross-DBMS-Kompatibilitäten in SQL-Abfragen erforderlich ist
- Vollständig funktionierende SQL-Syntax für Service-Transaktionen über Firebirds *gpre* Pre-Compiler-Sprache („EmbedSQL“)
- Verwendung der COM-Schnittstelle für Microsoft Distributed Transaction Coordinator (DTC)

Kapitel 2

Installation des Treibers

Das Kit, das Sie installieren, hängt davon ab, wofür Sie es verwenden möchten. Unabhängig davon, ob Sie eine Verbindung zu einem 64-Bit- oder einem 32-Bit-Firebird-Server herstellen möchten, müssen Sie den Treiber und den Firebird-Client (`fbclient.dll` auf Windows, `libfbclient.so` auf Linux) installieren, die der „Bittigkeit“ Ihrer Client-Anwendung entspricht.

Die Installation ist für beide Optionen ähnlich. Sie können sowohl den 32-Bit- als auch den 64-Bit-Treiber auf demselben Computer installieren, wenn der Benutzer auf Firebird von mehreren Anwendungen mit gemischter Bitanzahl aus zugreifen soll. Es muss sorgfältig darauf geachtet werden, dass jede Anwendung eine Verbindung mit dem richtigen DSN für den erforderlichen Treiber herstellt.

> **Hinweis für das weniger technisch Versierte**
>
> ... weil wir gefragt wurden: Wenn Sie Ihre Windows-Anwendung — Excel oder LibreCalc, zum Beispiel — mit Ihrer Datenbank verbinden wollen, die auf einem Linux- oder anderen POSIX-Server läuft, müssen Sie den Windows-Treiber, nicht den POSIX-Treiber verwenden. Siehe auch den folgenden Hinweis zur Firebird-Client-Bibliothek.

Den Treiber herunterladen

Der Download-Bereich unter https://www.firebirdsql.org/en/odbc-driver/ stellt verschiedenen Kits zu jeder Plattform bereit, mit der neuesten Version oben auf der Seite. Das 32-Bit-Installationsprogramm für Windows zum Zeitpunkt der Erstellung dieses Dokuments hatte beispielsweise den Namen `Firebird_ODBC_2.0.5.156_Win32.exe`. Dies weist darauf hin, dass es sich um das ausführbare Installationsprogramm für die 32-Bit-Version handelte . Die folgende Tabelle sollte Ihnen dabei helfen anzugeben, was Sie benötigen. Das hier verwendete „N.n.n.xxx“ Infix gibt „Major1.Major2.Minor.Subrelease“ an. Der „Subrelease“-Teil ändert sich am häufigsten.

Tabelle 2.1. Firebird ODBC/JDBC-Treiber-Kits

Kit-Name	Zweck
OdbcJdbc-src-N.n.n.xxx.tar.gz	Quellcode, der bititätsunabhängig ist. Empfohlen für POSIX-Installationen mit ungewöhnlichen Regeln zum Speicherort von Bibliotheken.
Firebird_ODBC_N.n.n.xxx_Win32.exe	Ausführbares Installationsprogramm für die Verwendung mit 32-Bit-Client-Anwendungen. Verwenden Sie dies für eine Erstinstallation.
Firebird_ODBC_N.n.n.xxx_x64.exe	

Kit-Name	Zweck
	Ausführbares Installationsprogramm für die Verwendung mit 64-Bit-Client-Anwendungen. Verwenden Sie dies für eine Erstinstallation.
OdbcFb_DLL_N.n.n.xxx_Win32.zip	Zip-Kit enthält nur die dynamischen und statischen 32-Bit-Bibliotheken und Dokumentation. Dies kann verwendet werden, um die Bibliothek einer vorhandenen Installation zu aktualisieren, wenn der Treiber nicht aktiv ist. Auf einem 64-Bit-Computer befindet sich die ältere Version im Ordner `c:\Windows\SySWOW64`. Zum Überschreiben sind Administratorrechte erforderlich.
OdbcFb_DLL_N.n.n.xxx_x64.zip	Zip-Kit, das nur die dynamischen und statischen 64-Bit-Bibliotheken und die Dokumentation enthält. Dies kann verwendet werden, um die Bibliothek einer vorhandenen Installation zu aktualisieren, wenn der Treiber nicht aktiv ist. Auf einem 64-Bit-Rechner befindet sich die ältere Version im Ordner `c:\Windows\system32` und zum Überschreiben sind Administratorrechte erforderlich. Es funktioniert nicht auf einem 32-Bit-Rechner.
OdbcFb-LIB-N.n.n.xxx.i686.gz	32-Bit-Binärdatei für einen POSIX-Client, gezippt
OdbcFb-LIB-N.n.n.xxx.amd64.gz	64-Bit-Binärdatei für einen POSIX-Client, gezippt

Die richtige Firebird Client-Bibliothek erhalten

Alle Firebird RDBMS-Kits enthalten mindestens eine Version der Firebird-Client-Bibliothek. Wenn es nur einen gibt, hat er die gleiche „Bittigkeit“ wie das Server-Installationskit selbst.

> **Wichtig**
>
> Stellen Sie sicher, dass Sie die Bibliothek fbclient mit der gleichen Hauptversionsnummer wie den Server erhalten, mit dem die Verbindung hergestellt werden soll.

- Bei einer 32-Bit-Windows-Installation befindet sich `fbclient.dll` in Firebirds Ordner `bin` in Firebird-Versionen niedriger als V.3.0. Für v.3.0 und höher befindet es sich im Stammordner von Firebird, z. B. C:\Programme (x86)\Firebird\Firebird\Firebird_3_0 oder überall dort, wo Firebird installiert wurde.

- Bei einer 64-Bit-Windows-Installation ist die Version der `fbclient.dll` in Firebirds `bin`-Ordner (oder Firebirds Root-Ordner für V.3.0 und höher) die 64-Bit-Version. In einigen Builds befindet sich der 32-Bit-Client in einem Ordner namens WOW64 oder system32, der sich unter dem Stammverzeichnis von Firebird befindet.

  Wenn Ihr ODBC-DSN-Setup den 32-Bit-Dateinamen der `fbclient.dll` benötigt und nicht vorhanden ist, müssen Sie das 32-Bit-Windows-Kit `.zip` herunterladen. Ziehen Sie den 32-Bit-Client aus der Haupt-

Firebird-Download-Seite und legen Sie ihn in den gleichen Ordner wie Ihre Anwendung. Alternativ können Sie stattdessen das 32-Bit-Installationsprogramm herunterladen und eine reine Client-Installation ausführen, indem Sie das Installationsprogramm so konfigurieren, dass es an der gewünschten Position platziert wird.

- Die POSIX-Server-Kits enthalten immer nur die passende `libfbclient.so`. Sie müssen diese einem .i686-Kit extrahieren, wenn Ihre POSIX-Client-Anwendung 32-Bit ist.

Die Client-Bibliothek sollte an der richtigen Stelle sein, **bevor** Sie den Treiber installieren und den DSN konfigurieren.

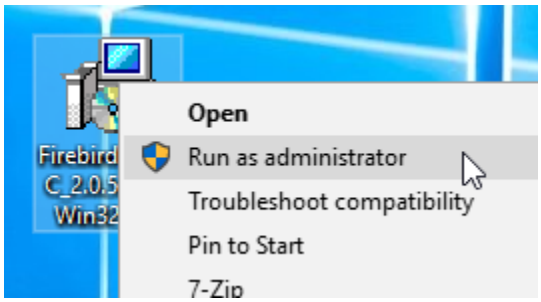
> **Kompatibilität des Treibers mit Firebird Versionen**
>
> Es wird erwartet, dass die aktuellste Version des ODBC / JDBC-Treibers mit jeder unterstützten Firebird-Version kompatibel ist.

Installieren des Treibers unter Windows

Wenn Sie den Treiber zum ersten Mal installieren oder eine ältere Version deinstalliert haben, wird empfohlen, das ausführbare Installationsprogramm zu verwenden. Diese Anweisungen gehen davon aus, dass Sie den 32-Bit-Treiber installieren, aber das Verfahren ist dasselbe für die Installation des 64-Bit-Treibers. Unter der Haube wird die 32-Bit-Treiberbibliothek in `\windows\sysWOW64` auf einem 64-Bit-Windows installiert. Bei jeder anderen Installation wird der Treiber in `windows\system32` gespeichert.

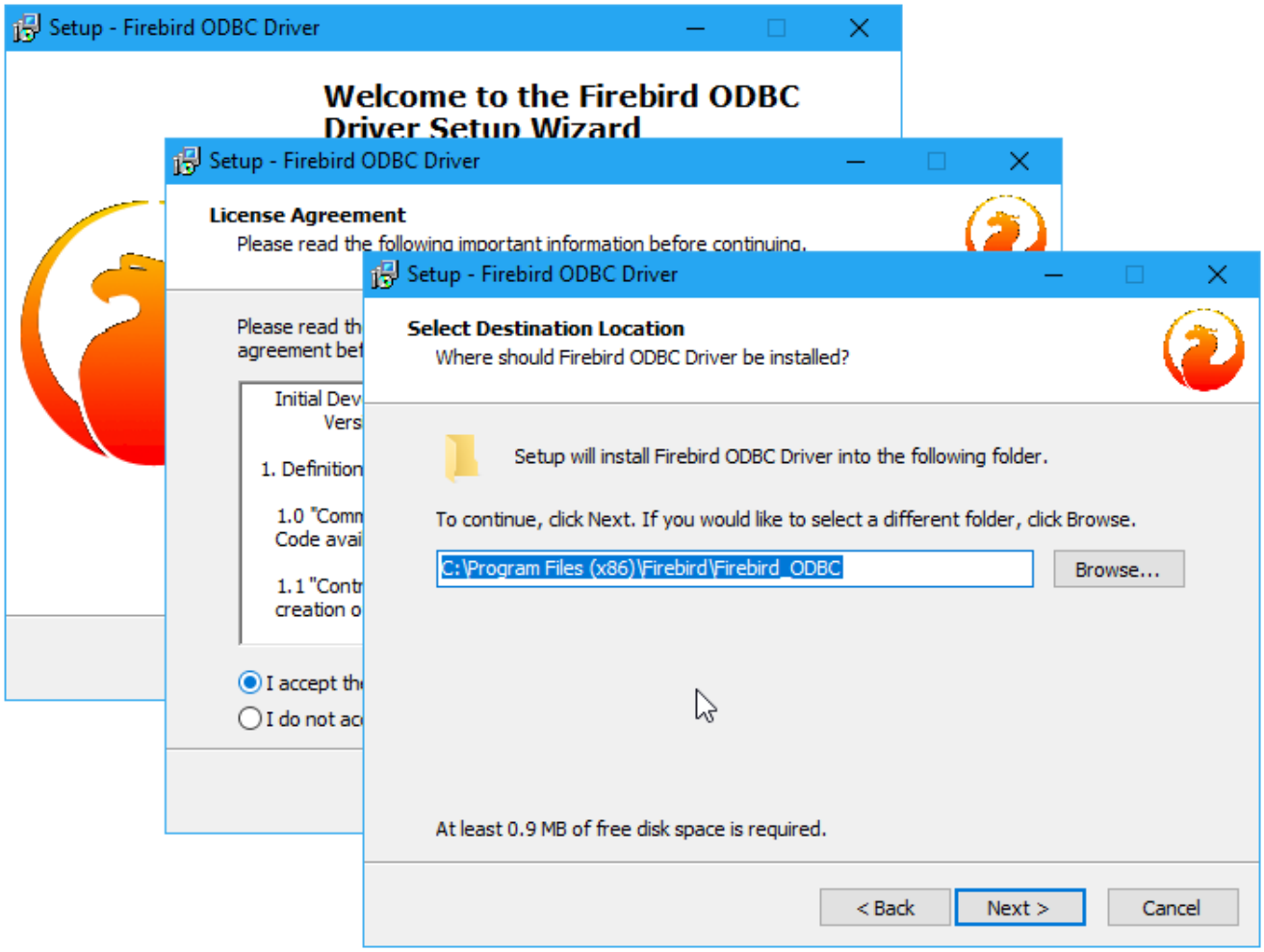
Laden Sie das ausführbare Installationsprogramm herunter oder verschieben Sie es auf den Desktop. Klicken Sie mit der rechten Maustaste darauf und wählen Sie `Als Administrator ausführen`.

Abbildung 2.1. ODBC-Treiberinstallationsprogramm auf dem Desktop



Klicken Sie sich durch die Bildschirme, bis Sie zu dem Fenster gelangen, in dem Sie Ihre Einstellungen für die Installation vornehmen:

Abbildung 2.2. Masken des ODBC-Treiberinstallationsprogramms

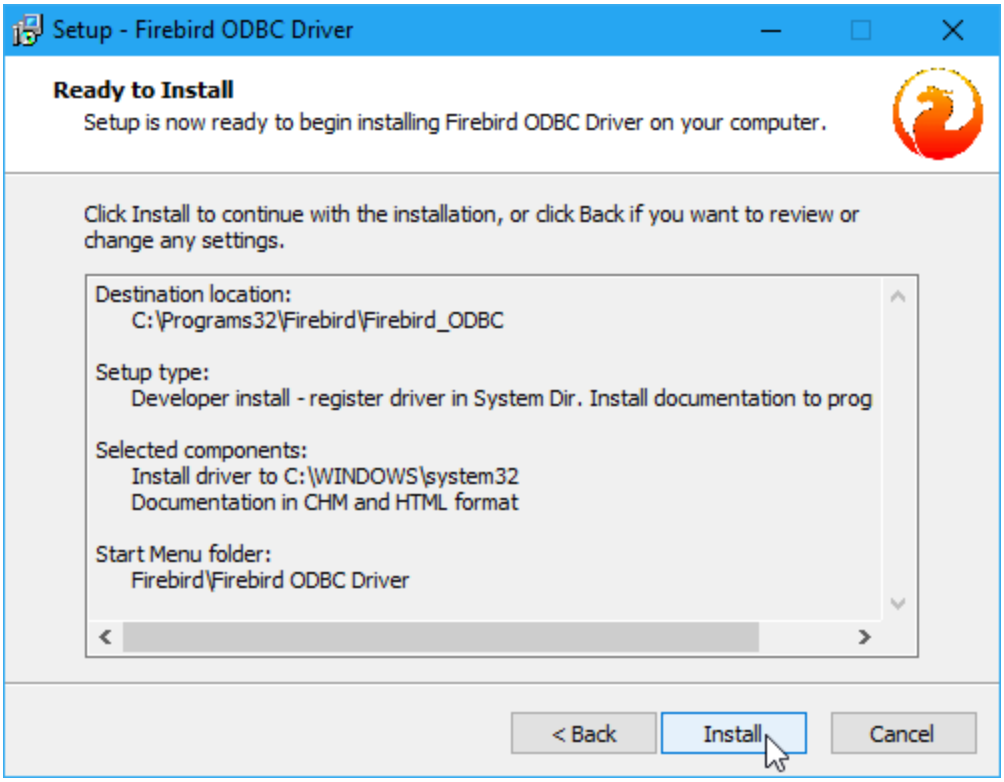


Wenn Sie möchten oder müssen, können Sie den Treiber an einem anderen als dem standardmäßig vom Installationsprogramm angebotenen Ort installieren lassen. Verwenden Sie die Schaltfläche Durchsuchen, um den Speicherort für den Treiber zu finden.

> **Anmerkung**
>
> Der Installer erstellt das Unterverzeichnis `\Firebird_ODBC`, falls dieses noch nicht existiert.

Zuletzt zeigt das Installationsprogramm die von Ihnen gewählte Konfiguration an. Wenn Sie damit zufrieden sind, klicken Sie einfach auf `Installieren` und es ist abgeschlossen.

Abbildung 2.3. Schlussmaske des ODBC-Treiberinstallationsprogramms



Anmerkung

Sie können hier feststellen, dass wir auf unserem System unter C:\Windows unsere eigenen dedizierten „Programs64“ und „Programs32“-Verzeichnisse haben. Das ist einfach die Präferenz, wie wir unseren Server organisieren und die Menge der von Windows-Updates installierten Dateien in seinen eigenen Programmordnern überwachen.

Die auf diesem Bildschirm notierten .chm- und .html-Dokumente sind ältere Stände der Dokumente, die zum Zeitpunkt des Schreibens noch mit den Kits eingebaut wurden.

Installieren des Treibers unter Linux

Pavel Cisar

Es gibt zwei Voraussetzungen für die Installation des ODBC / JDBC-Treibers unter Linux:

- Das Paket **unixODBC** muss installiert sein
- Firebird muss, zumindest zu Beginn, installiert werden, um die Installation zu testen

Entpacken der Dateien

Die ODBC/JDBC-Treiberpakete für Linux sind gezippte `tar`-Dateien. Nach `gunzip` sollten sie mit tar bearbeitet werden, oder Sie können sie in `.tar.gz` umbenennen und sie mit einem Tool wie Midnight Commander entpacken.

Aus den Quellen erzeugen

Das Erstellen aus dem Quellcode (empfohlen) erfordert das Entwicklungspaket für unixODBC. Fahren Sie mit den folgenden Schritten fort:

1. Laden Sie die Firebird-Treiberquellen herunter und entpacken Sie sie
2. Benennen Sie "makefile.linux" in `.source/Builds/Gcc.lin` zu "makefile" um
3. Legen Sie die Umgebungsvariablen `FBINCDIR` (Firebird include-Verzeichnis) und `FBLIBDIR` (Firebird lib-Verzeichnis) fest, falls notwendig.
4. Führen Sie **make** aus, welches die Bibliothek `libOdbcFb.so` in einem Unterverzeichnis erstellt.
5. Es ist möglich die Bibliothek nach `/usr/local/lib64` oder einem anderen präferierten Ort zu kopieren; oder führen Sie **make install** aus, um einen Symlink auf die Bibliothek aus dem `unixODBC`-Verzeichnis zu erstellen

Installieren des binären Pakets

Für die Installation des Binärpakets, kopieren Sie `libOdbcFb.so` nach `/usr/local/lib64`, `/usr/local/lib32` oder einen anderen angemessenen Ort.

Kapitel 3

Firebird ODBC-Konfiguration

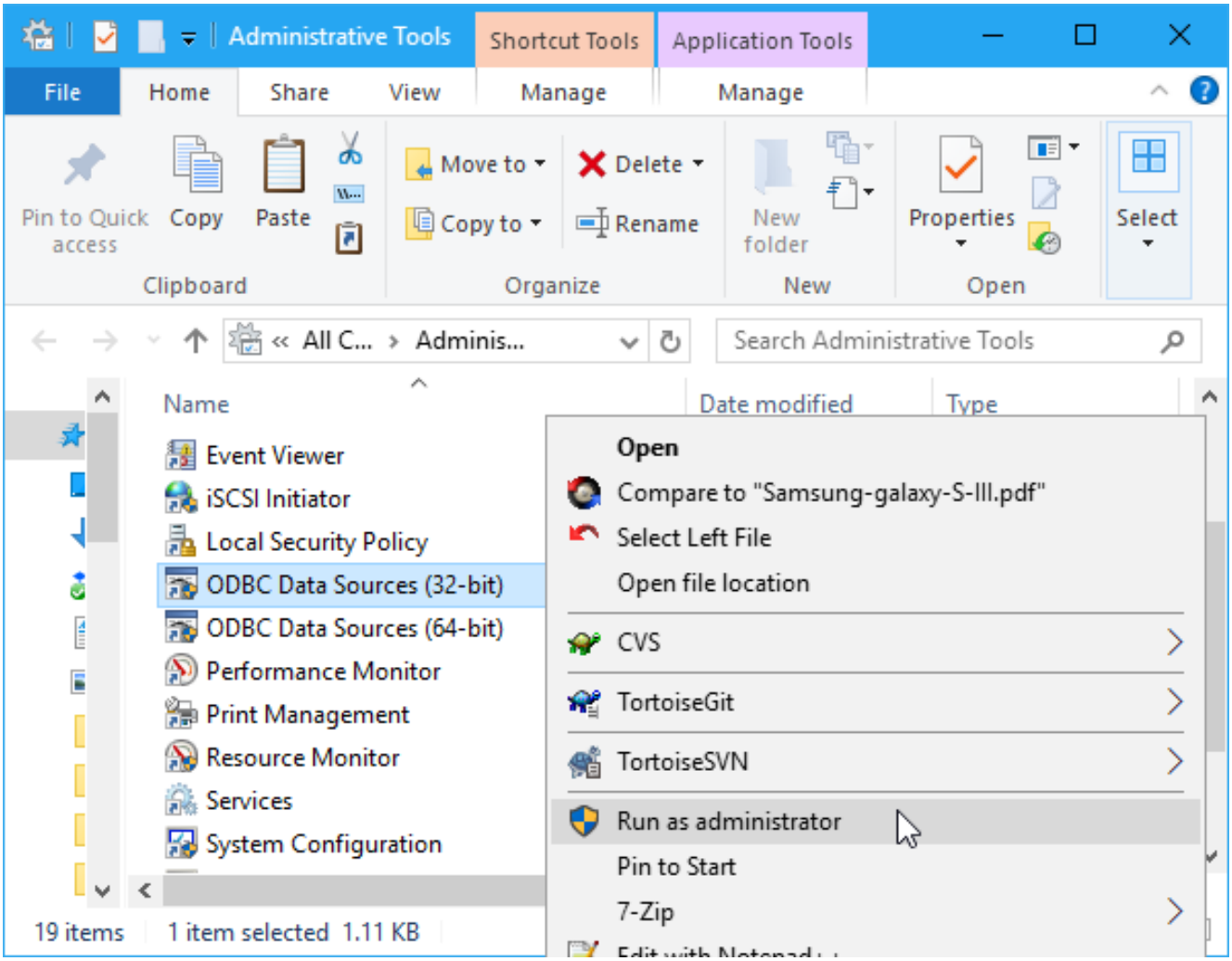
Die Konfigurationseinstellungen, die Sie in einer ODBC-Datenquellenbeschreibung („DSN“) vornehmen, definieren die Attribute für die Verbindung mit einer bestimmten Datenbank. Unter Windows erfasst ein Dialogfeld Parameter, die den Verbindungsattributen entsprechen. Unter Linux werden die Parameter manuell in Textdateien (.ini) konfiguriert.

DSN unter Windows konfigurieren

Suchen Sie zuerst die Applets im Abschnitt Verwaltung des Computers, auf dem Sie einen „Kanal“ einrichten möchten, über den sich Ihr Anwendungsprogramm mit einer Firebird-Datenbank verbindet, entweder auf demselben Computer oder an einem anderen Ort im lokalen oder Wide-Area-Netzwerk.

Auf einem 64-Bit-Computer finden Sie zwei solche Applets:

Abbildung 3.1. Auswählen eines DSN-Setup-Applets unter Windows

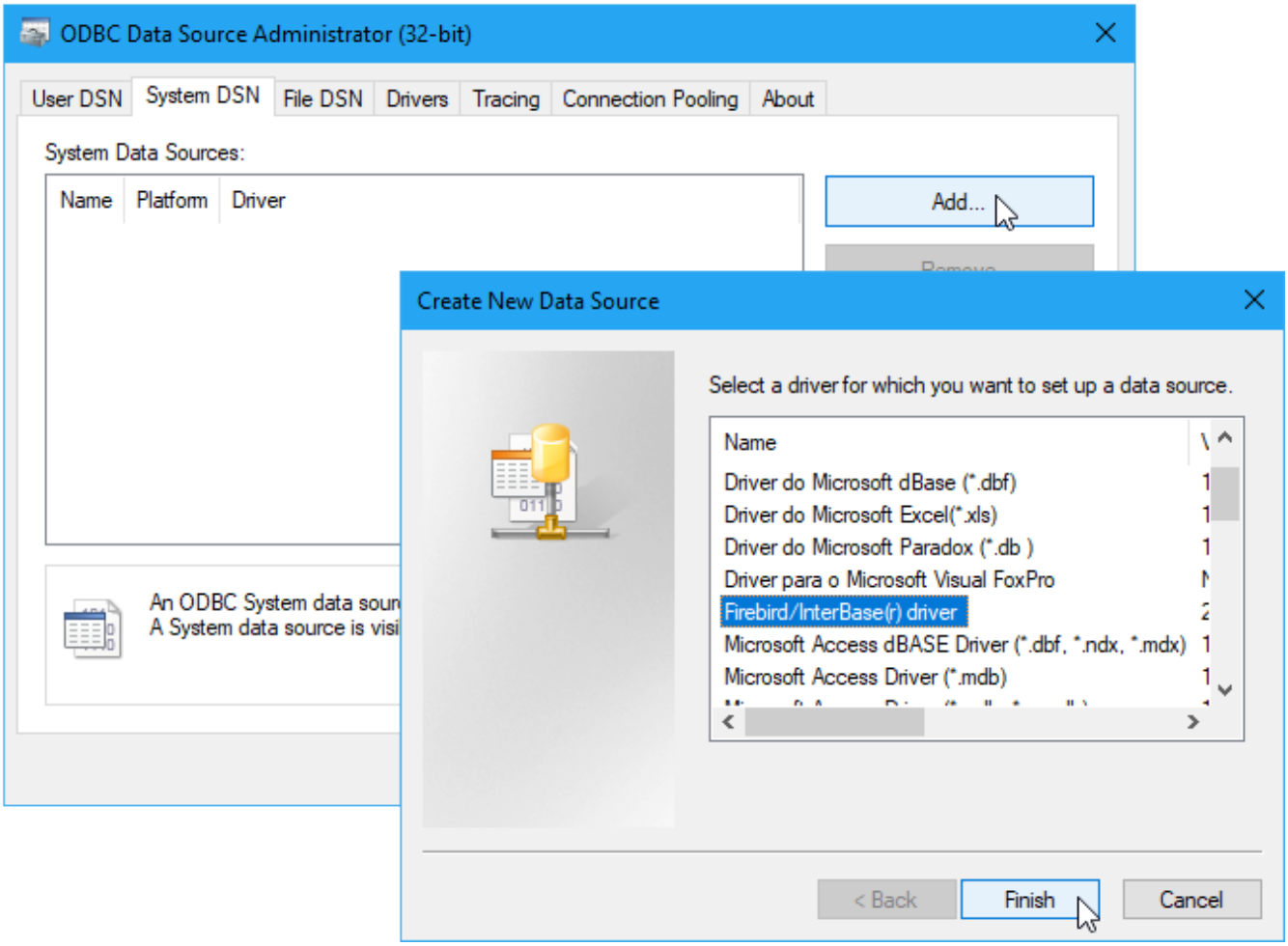


Für unser Beispiel wollen wir den Punkt *ODBC-Datenquellen (32-Bit)* auswählen. Wenn wir den 64-Bit-Treiber mit der Absicht installiert haben, ihn für eine 64-Bit-Anwendung zu verwenden, würden wir stattdessen das 64-Bit-Element aus diesem Menü auswählen.

> **Als Administrator ausführen!**
>
> Klicken Sie nicht mit der linken Maustaste auf das Objekt: Klicken Sie mit der rechten Maustaste und wählen Sie im Kontextmenü `Als Administrator ausführen`. Dies ist erforderlich, da Sie gerade einen **System-DSN** einrichten.

Klicken Sie auf den Tab „System DSN“, wo Sie mit der Einrichtung Ihres DSN beginnen.

Abbildung 3.2. Auswählen des Firebird-Treibers für den DSN



Klicken Sie im ersten Bildschirm auf `Hinzufügen...`, um die Liste der Treiber auf der nächsten anzuzeigen. Wählen Sie den **Firebird / InterBase (r)**-Treiber und klicken Sie auf Finish.

Die DSN-Einstellungen

Nachdem Sie auf dem vorherigen Bildschirm auf `Fertigstellen` geklickt haben, wird ein Formular angezeigt, in das Sie die Parameter für eine Verbindung eingeben und testen können, ob alle Parameter funktionieren.

Abbildung 3.3. Festlegen der DSN-Parameter

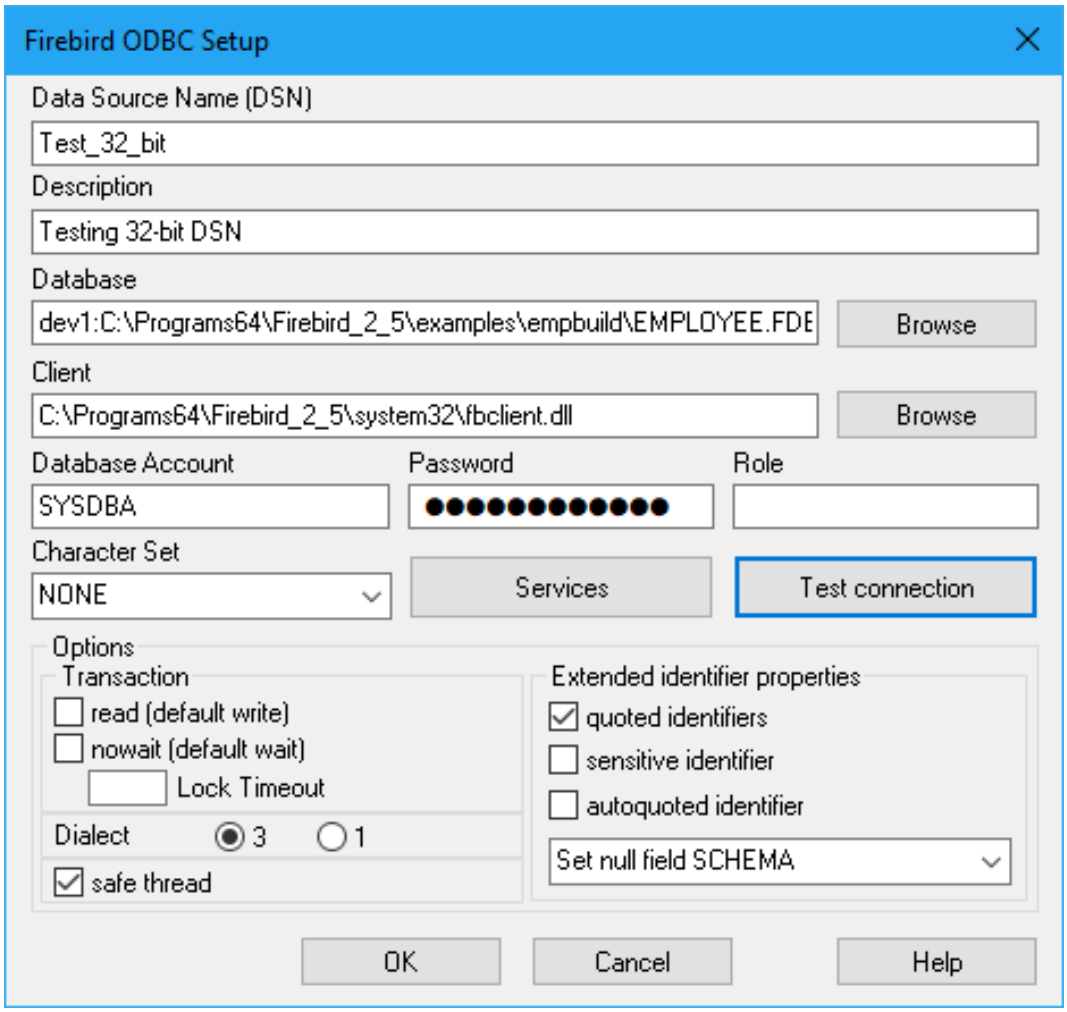


Tabelle 3.1. Parameter der DSN-Konfiguration

Parameter	Eintrag
Data Source Name (DSN)	ERFORDERLICH. Ein eindeutiger, aussagekräftiger Name, der den Verbindungstyp oder seine Verwendung angibt. Machen Sie es kurz, da Sie die Beschreibung an anderer Stelle erweitern können. Beispielsweise "Verbindung von FbEmbed" oder "ConnectFbServer"
Description	Optional. Kann verwendet werden, um weitere Details zur Datenquelle anzugeben.
Database	ERFORDERLICH. Vollständige Adresse der Datenbank, die für eine eingebettete oder Netzwerkverbindung erforderlich ist. Wenn die Verbindung entfernt ist, kann sie im TCP / IP- oder WNET-Format sein. TCP / IP wird empfohlen. Firebird-Datenbank-Aliase werden unterstützt. Siehe auch Verbindungsbeispiele.
Client	Möglicherweise benötigt. Lokaler Pfad zur Firebird Client-Bibliothek. Bei eingebetteten Verbindungen zu einem Windows-Server unter der Version V.3 kann er auf die Kopie von `fbembed.dll` im Anwendungsverzeichnis verweisen. Andernfalls verweisen Sie auf den Pfad

Parameter	Eintrag
	zur Bit-kompatiblen Firebird Remote-Client- Bibliothek, es sei denn, Sie sind sicher, dass die richtige Bibliothek automatisch an einem Systemstandort gefunden wird.
Database Account	Optional, da Anmeldeinformationen während der Verbindung mit einer Firebird-Datenbank erfasst werden können. Wenn es nicht konfiguriert ist, fragt die ODBC-Schnittstelle zur Verbindungszeit nach einer Benutzer-ID (UID oder USER).
Password	Optional, da Anmeldeinformationen während der Verbindung mit einer Firebird-Datenbank erfasst werden können. Wenn es konfiguriert ist, sollte es das Passwort für die angegebene Benutzer-ID sein. Andernfalls fordert die ODBC-Schnittstelle zur Verbindungszeit ein Kennwort (PWD oder PASSWORD) an. Jedes konfigurierte Passwort wird automatisch verschlüsselt und in `odbc.ini` gespeichert. Das Speichern des Passworts sollte somit kein Sicherheitsrisiko darstellen.
Role	Optional. Wenn die Rolle definiert ist und die Anmeldung von SYSDBA erfolgt, wird die Rolle ignoriert. Andernfalls müssen die Anmeldeinformationen, unabhängig davon, ob sie bei der Verbindung gespeichert oder erfasst wurden, vor dem Anmeldeversuch diese Rolle erhalten haben.
Character Set	Kann leer sein. Legt den Standardzeichensatz des Clients fest.
Optionen (hier in DSN festlegen oder dynamisch angeben)	
Transaktionsparameter	
Read (default write)	Transaktionen werden standardmäßig gelesen / geschrieben. Aktivieren Sie diese Option, um Transaktionen schreibgeschützt zu machen.
Nowait (default wait)	Die Transaktion wartet, wenn ein Sperrkonflikt auftritt. Überprüfen Sie, ob die Transaktion sofort einen Fehler zurückgibt, wenn ein Sperrkonflikt auftritt.
Lock timeout	Wenn eine Transaktion für die WAIT-Konfliktlösung festgelegt ist, geben Sie die Länge der Zeit in Sekunden an, bis die Sperre abgelaufen ist und ein Sperrkonfliktfehler zurückgegeben wird (`isc_lock_timeout`).
Andere optionale Parameter	
Dialect	SQL-Dialekt für den Client zum Zugriff auf die Datenbank. Die einzigen gültigen Optionen für Firebird sind 1 oder 3. Beachten Sie, dass Dialekt 1 nicht mit Zitatzeichen kompatibel ist. Dialekt 3 akzeptiert keine durch doppelte Anführungszeichen begrenzten Zeichenketten.
Quoted Identifier	Bewirkt, dass Paare von doppelten Anführungszeichen nur als Trennzeichen für die Bezeichner von Groß- und Kleinschreibung verwendet werden. Versuche, doppelte Anführungszeichen als Trennzeichen für Zeichenketten zu übergeben, werden in beiden Dialekten als Fehler behandelt. Beachten Sie, dass Strings in doppelten Anführungszeichen in Dialekt 3 immer unzulässig waren.

Parameter	Eintrag
Sensitive Identifier	Diese Option beeinflusst die Art und Weise, wie der Client die Eigenschaft `SQL_IDENTIFIER_CASE` behandelt. `SQL_IC_UPPER` (Wert = 1) ist der Standardwert und behandelt alle Bezeichner so, dass sie in Großbuchstaben gespeichert werden. Aktivieren Sie diese Option, um `SQL_IC_SENSITIVE` (Wert = 3) auszuwählen, damit die Schnittstelle alle Bezeichner behandelt, die nicht in Großbuchstaben stehen, als ob sie die Groß- / Kleinschreibung beachten würden. Dies wird nicht empfohlen! Für eine Erklärung, siehe Anmerkung (1) unten.
Autoquoted Identifier	Standard ist NEIN. Wenn Sie dies überprüfen, ändern Sie die Einstellung auf JA. In diesem Fall wird jeder Bezeichner in jeder Anweisung automatisch doppelt zitiert. Die Notwendigkeit, dies zu tun, wäre höchst ungewöhnlich und müsste gut verstanden werden, um ständige Fehler zu vermeiden.
SCHEMA options	Dropdown-Liste mit drei Optionen zur Behandlung von SQL-Schemas, die Firebird nicht unterstützt. Normalerweise belassen Sie dies bei der Standardeinstellung `Set null field SCHEMA`. Für einige Details siehe Anmerkung (2) unten.

Anmerkung (1) zu „Sensitive identifier“

Wenn diese Einstellung aktiviert ist, würde dieses Statement

```
SELECT A.Test_Field FROM Mixed_Caps_Table A
ORDER BY A.Test_Field
```

zu dieser Anweisung konvertiert:

```
SELECT A."Test_Field" FROM "Mixed_Caps_Table" A
ORDER BY A."Test_Field"
```

Das folgende Statement würd in einer falschen Konvertierung münden:

```
Select A.Test_Field From Mixed_Caps_Table A
Order By A.Test_Field
```

wird zu folgender Anweisung konvertiert:

```
"Select" A."Test_Field" "From" "Mixed_Caps_Table" A
"Order" "By" A."Test_Field"
```

Anmerkung (2) bezüglich SCHEMA settings

Einige Anwendungen generieren SQL-Anweisungen basierend auf Benutzeranfragen automatisch unter der Annahme, dass die Zieldatenbank Namespaces und SQL-Schemas unterstützt. Zum Beispiel

```
select SYSDBA.COUNTRY,SYSDBA.CURRENCY from SYSDBA.COUNTRY
```

oder

```
select * from SYSDBA.COUNTRY
```

Diese Auswahl von Schemaeinstellungen versucht, Konflikte mit Anwendungen zu verhindern, die diese Art von Vorgang ausführen. Die Dropdown-Liste bietet die drei folgenden Varianten:

1. Set null field SCHEMA
2. Remove SCHEMA from SQL query
3. Use full SCHEMA

`Set null field SCHEMA` ist der Standard. Dadurch wird das SCHEMA-Element immer dann auf NULL gesetzt, wenn es als Teil einer Abfrage angegeben wird. Das Ergebnis ist eine Abfrage, die Firebird verarbeiten kann.

`Remove SCHEMA from SQL query` filtert die Namespaceverweise aus der Anweisung, wenn der Befehl SQLExecDirect eine Anforderung wie

```
select SYSDBA.COUNTRY,SYSDBA.CURRENCY from SYSDBA.COUNTRY
```

transformiert, bevor diese an die API in Form von

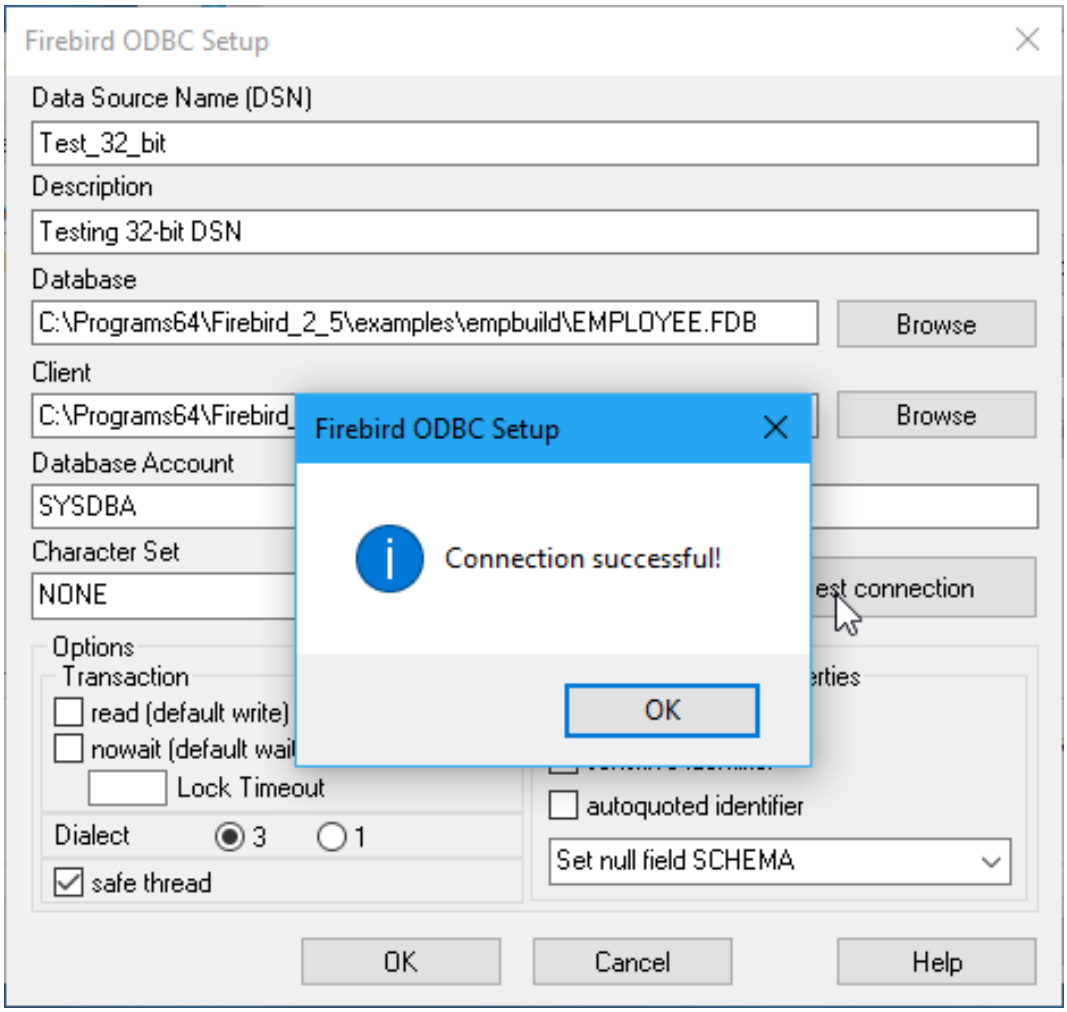
```
select COUNTRY,CURRENCY from COUNTRY
```

weitergegeben wird.

`Use full SCHEMA` ist für die Zukunft reserviert, in der Firebird die Möglichkeit hat, diese Konzepte selbst zu verarbeiten — vielleicht in Firebird 4. In diesem Fall muss der Treiber diese Konstruktionen nicht aussortieren.

Klicken Sie auf die Schaltfläche „Test connection“, um sicherzustellen, dass Ihre Konfiguration korrekt ist:

Abbildung 3.4. Testen der Konfiguration



Wenn alles in Ordnung ist, klicken Sie auf OK, kehren Sie zum Hauptformular zurück und speichern Sie die Konfiguration, indem Sie dort auch auf OK klicken.

Die Services-Schltfläche

Die Schaltfläche Services startet eine Reihe von Dienstprogrammen zur Serververwaltung über eine GUI-Verwaltungskonsole. Es wird später in beschrieben Die Dienstschnittstelle.

Konfigurieren eines DSN unter Linux

Pavel Cisar

Die Konfiguration hängt von der Linux-Distribution ab, sollte jedoch irgendwo in `/etc` oder `/etc/unixODBC` zu finden sein. Dort liegen zwei Dateien namens `odbc.ini` und `odbcinst.ini`.

Fügen Sie der odbcinst.ini folgende Daten hinzu:

```
[Firebird]
Description     = InterBase/Firebird ODBC Driver
Driver          = /usr/local/lib64/libOdbcFb.so
Setup           = /usr/local/lib64/libOdbcFb.so
Threading       = 1
FileUsage       = 1
CPTimeout       =
CPReuse         =
```

Fügen Sie der odbc.ini folgende Daten hinzu:

```
[employee]
Description     = Firebird
Driver          = Firebird
Dbname          = localhost:/opt/firebird/examples/empbuild/employee.fdb
User            = SYSDBA
Password        = masterkey
Role            =
CharacterSet    =
ReadOnly        = No
NoWait          = No
```

Testen der Konfiguration

UnixODBC hat ein Werkzeug namens `ISQL` (nicht zu verwechseln mit dem gleichnamigen Tool von Firebird!), mit dem Sie die Verbindung wie folgt testen können:

```
isql -v employee
```

Stellen Sie bei Verbindungsproblemen sicher, dass sich das Verzeichnis, in dem Sie die gemeinsam genutzte Firebird ODBC-Bibliothek gespeichert haben, z. B. `/usr/local/lib64/libOdbcFb.so`, im ladbaren Bibliothekspfad des Systems befindet. Wenn nicht, können Sie dies festlegen:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/odbc
```

oder einfacher:

```
export LD_LIBRARY_PATH=/usr/lib/odbc
```

Wenn Sie noch immer Probleme haben, können Sie als nächstes versuchen diese mittels `strace` zu identifizieren:

```
strace -o output.txt isql -v employee
```

Kapitel 4

Verbindung zu Firebird über Anwendungen herstellen

Der ODBC/JDBC-Treiber versucht, einen Client mit dem Firebird-Server gemäß einer Reihe von Attributen zu verbinden, die standardmäßig denen entsprechen, die von der DSN-Definition bereitgestellt werden. Diese gespeicherten Attribute können und werden normalerweise von Parametern überschrieben, die von der Anwendung übergeben oder aus einer Datei (FILEDSN) gelesen werden, wenn sie die Verbindung vorbereitet.

Verbindungsparameter

Die Verbindungsparameter für den Treiber bestehen aus einer Liste von Strings in der Form `KEYWORD=value`, die durch Semikolons getrennt sind. In der folgenden Tabelle sind die Schlüsselwörter mit ihren ausführlichen Bedeutungen und, wenn sie nicht offensichtlich sind, mit ihren möglichen Werten aufgelistet.

Tabelle 4.1. Schlüsselwörter für Verbindungsattribute

Schlüsselwort	Beschreibung	Weitere Informationen
UID oder USER	Datenbankaccount, z.B. Benutzername	...
PWD oder PASSWORD	Kennwort	...
ROLE	Role	...
DSN	Name der Datenquelle	...
DRIVER	Treibername	z.B. die Zeichenkette `Firebird/InterBase(r) driver`. Der Standardwert ist der im DSN definierte Treiber.
DBNAME oder DATABASE	Datenbank	Vollständiger Pfad zur Datenbank *vom Server* aus gesehen, einschließlich der IP-Adresse | Servername [/ Port] für eine Remoteverbindung. Der Standardwert ist die im DSN definierte Datenbank.
CLIENT	Lokaler Pfad zur erforderlichen `fbclient`-Bibliothek	Wird möglicherweise benötigt, wenn die Verbindung über eine eingebettete Serverbibliothek in einem Anwendungsordner hergestellt werden soll.
CHARSET oder CHARACTERSET	Clientseitiger Standardzeichensatz	Sollte nach Möglichkeit der Standardzeichensatz der Datenbank sein; oder eine, die bekanntermaßen Codepage-kompatibel ist.

Schlüsselwort	Beschreibung	Weitere Informationen
READONLY	Nur-Lesen	Setzen Sie Transaktionen in dieser Verbindung auf schreibgeschützt. Der Standard ist Lesen / Schreiben.
NOWAIT	Nicht warten	Setzen Sie Transaktionen in dieser Verbindung auf NO WAIT-Sperrmechanismus. Der Standardwert ist WAIT.
LOCKTIMEOUT	Setzen Sie das Sperrzeitlimit für die WAIT-Transaktion	Übergeben Sie die Anzahl der Sekunden, nach denen ein Sperrkonflikt auftritt, bis eine Transaktion einen Fehler zurückgibt. Nicht gültig, wenn die Transaktion auf NO WAIT-Sperrmechanismus gesetzt ist.
DIALECT	Legt den SQL-Dialekt fest	Nur 1 oder 3 ist gültig. Normalerweise wäre dies in der DSN festgelegt worden. Es muss dem Dialekt der Datenbank entsprechen.
QUOTED	In Anführungszeichen gesetzte Bezeichner festlegen	Wenn im DSN festgelegt, sollte diese Einstellung korrekt sein, d.h. bereits EIN oder AUS.
SENSITIVE	Legt die Bezeichner für Groß-und Kleinschreibung fest.	Wenn im DSN festgelegt, sollte die Einstellung korrekt sein, d.h. bereits EIN oder AUS.
AUTOQUOTED	Automatisch-Zitierende Bezeichner festlegen	Wenn im DSN festgelegt, sollte die Einstellung korrekt sein, d.h. bereits EIN oder AUS.
USESCHEMA	Setzt „use schema“ auf an	Wenn im DSN festgelegt, sollte die Einstellung korrekt sein.
SAFETHREAD	Safe threading	...
FILEDSN	Datei-DSN	Pfad zu einer Datei, in der die Attributstrings einer vorherigen Verbindung gespeichert sind. Wenn diese Zeichenfolge vorhanden ist, hat der Inhalt der Datei Vorrang vor dem Haupt-DSN.
SAVEDSN	Save-DSN	Pfad zu einer Datei, in der die Attributstrings dieser Verbindung bei Erfolg gespeichert werden sollen. Das Passwort wird verschlüsselt gespeichert.

Leserichtung der Schlüssel

Die ODBC-Funktion `SQLDriverConnect` weist den in der Verbindungszeichenfolge definierten Attributen ihre Priorität zu, wobei nur die in der DSN oder in einer angegebenen Datei-DSN gespeicherten Attribute abgerufen werden, um Lücken zu füllen.

Verbindungsbeispiele

Einige Beispiele für Verbindungszeichenfolgen für Anwendungen, die die ODBC-Funktion `SQLDriver-Connect` verwenden:

```
Open("DSN=myDb;")
```

Hier wird erwartet, dass die Funktion alles, was sie benötigt, aus dem DSN liest. Benutzername und Passwort werden nicht in der Zeichenfolge angegeben. Wenn sie auch nicht im DSN vorhanden werden

1. die Umgebungsvariablen ISC_PASSWORD und ISC_USER verwendet, wenn sie gesetzt sind. Andernfalls
2. wird es nach Benutzername und Kennwort fragen

```
Open("DSN=myDb; UID=MCSSITE; PWD=mcssite;")
```

Die Funktion sollte über die erforderlichen Voraussetzungen für die Herstellung dieser Verbindung verfügen, vorausgesetzt, der Benutzername und das Kennwort werden vom Server authentifiziert.

```
Open("DSN=myDb; UID=MCSSITE; PWD=mcssite; DBNAME=172.17.2.10:/usr/local/db/myDb.fdb;")

Open("DSN=myDb; UID=MCSSITE; PWD=mcssite; DBNAME=myserver:/usr/local/db/myDb.fdb;")
```

Der Schlüssel DBNAME zeigt im ersten Beispiel auf die IP-Adresse des Servers mit dem Pfad zur Datenbankdatei im POSIX-Format. Im zweiten Beispiel wird dieselbe Verbindung hergestellt, wobei der Host-Name des Servers anstelle der IP-Adresse verwendet wird.

Drei Beispiele, die das DRIVER-Attribut in der Zeichenfolge enthalten:

```
Open("DRIVER=Firebird/InterBase(r) driver; DBNAME=172.17.2.10:/usr/local/db/myDb.fdb;")

Open("DRIVER=Firebird/InterBase(r) driver; UID=MCSSITE; PWD=mcssite; DBNAME=172.17.2.10:/us

Open("DRIVER=Firebird/InterBase(r) driver; UID=MCSSITE; PWD=mcssite; DBNAME=dummy;")
```

Im letzten Beispiel eine lokale Verbindung mit einem Datenbankalias anstelle des Datenbankdateipfads. Natürlich muss der Alias in `aliases.conf` im Root-Verzeichnis des Firebird-Servers vorhanden sein(oder, für Firebird 3 und höher, in `databases.conf`).

Verwenden der Server-IP-Adresse und Angabe eines alternativen Ports mit der Zieldatenbank auf einem POSIX-Server; und dasselbe mit dem Hostnamen des Servers:

```
172.17.2.10/3051:/usr/local/db/myDb.fdb

myserver/3051:/usr/local/db/myDb.fdb
```

Verwenden der Server-IP-Adresse mit der Zieldatenbank auf einem Windows-Server; und dasselbe mit dem Hostnamen des Servers:

```
172.17.2.10:c:\db\myDb.fdb

myserver:c:\db\myDb.fdb
```

Verwenden der Server-IP-Adresse und Angeben eines alternativen Ports mit der Zieldatenbank auf einem Windows-Server; und dasselbe mit dem Hostnamen des Servers:

```
172.17.2.10/3051:c:\db\myDb.fdb

myserver/3051:c:\db\myDb.fdb
```

Verwenden von lokalen TCP/IP-Loopbacks unter Verwendung der L/L-IP-Adresse auf einem POSIX-Server; und dasselbe mit dem L/L Host-Namen `localhost`:

```
127.0.0.1:/usr/local/db/myDb.fdb

localhost:/usr/local/db/myDb.fdb
```

Die gleichen Dinge auf einem Windows-Server:

```
127.0.0.1:c:\db\myDb.fdb

localhost:c:\db\myDb.fdb
```

DBNAME für Embedded-Verbindungen

Der DBNAME-Wert für eingebettete Verbindungen und für den Verbindungsstil „Windows Local“ (XNET) verwendet nur den Dateipfad oder Alias ohne Hostnamen, IP-Adresse oder Portnummer.

> **Anmerkung**
>
> Ab Windows 3 ist die Art und Weise, wie wir Nicht-Netzwerkverbindungen auf allen Plattformen konzeptualisieren, einheitlicher als für die früheren Versionen. Aus der Sicht des ODBC/JDBC-Treibers hat sich jedoch der Ausdruck des DBNAME-Werts nicht geändert, unabhängig von der Plattform, auf der wir unsere eingebettete Verbindung herstellen.

Lokale Verbindung auf einem Windows-Server, wobei zuerst der Dateipfad und anschließend ein Alias verwendet wird:

```
DBNAME=C:\db\myDb.fdb

DBNAME=C:dummy
```

Auf einem POSIX-Server:

```
DBNAME=/usr/local/db/myDb.fdb

DBNAME=dummy
```

DBNAME unter Verwendung eines Alias

Es wird dringend empfohlen, Aliase zu definieren und zu verwenden, um das Leben für Sie und Ihre Benutzer zu vereinfachen. Es macht Ihre DBNAME-Werte für das Dateisystem vollkommen neutral und um vieles weniger umständlich. In unseren letzten Beispielpaaren wurde derselbe Alias sowohl für Windows als auch für POSIX verwendet. Der auf dem Windows-Server wäre so definiert:

```
dummy = C:\db\myDb.fdb
```

während dieser auf dem Linux-Server definiert werden würde:

```
dummy = /usr/local/db/myDb.fdb
```

Kapitel 5

Entwickeln mit dem Firebird ODBC/JDBC-Treiber

Der Firebird ODBC-Treiber unterstützt mehrere gleichzeitige Verbindungen zu verschiedenen Datenbanken und verschiedenen Servern, wobei jede Verbindung unabhängig von anderen funktioniert.

Multithreading

Thread-Protection kann auf zwei Ebenen festgelegt werden:

1. teilen eines Umgebungshandles
2. teilen eines Verbindungshandles

Standardmäßig wird der Treiber mit der folgenden Definition erstellt:

```
#define DRIVER_LOCKED_LEVEL      DRIVER_LOCKED_LEVEL_CONNECT
```

Dadurch kann eine einzelne Verbindung mehrere lokale Threads gemeinsam nutzen.

Die Standardeinstellung spiegelt sich in der anfänglichen Einrichtung des DSN unter Windows wider: `SAFETHREAD=Y`.

Wenn der Treiber mit der folgenden Definition erstellt wird:

```
#define DRIVER_LOCKED_LEVEL      DRIVER_LOCKED_LEVEL_NONE
```

Dann wird der Treiber ohne Multi-Threading-Unterstützung erstellt und die Verantwortung für die Threading-Steuerung wird in die Firebird-Client-Bibliothek übertragen. Dies sorgt für die schnellste Leistung.

Wenn Sie einen Build erstellt haben, der mit dieser Definition erstellt wurde, sollten Sie ihn zum Standard-Thread-Verhalten für den DSN machen, indem Sie `SAFETHREAD = N` in dessen Schnittstelle konfigurieren.

Wenn der Treiber mit der folgenden Definition erstellt wird:

```
#define DRIVER_LOCKED_LEVEL      DRIVER_LOCKED_LEVEL_ENV
```

dann kann ein einzelner Umgebungs-Handle von mehreren lokalen Threads gemeinsam genutzt werden.

Anmerkung

Sie können eine bestimmte Gruppe von Verbindungsbedingungen oder Außerkraftsetzungen in einem Datei-DSN speichern.

Transaktionen

Firebird unterstützt drei Transaktionsisolationsstufen:

- READ COMMITTED
- SNAPSHOT („concurrency“ oder „repeatable read“)
- SNAPSHOT TABLE STABILITY „consistency“)

Die Standardisolationsstufe des ODBC/JDBC-Treibers ist READ COMMITTED, die in anderen Datenbanksystemen dem Lesevorgang festgeschrieben ist. Firebirds andere Isolationsstufen lassen sich nicht so einfach abbilden. Im ODBC/JDBC-Treiber wird SNAPSHOT mit einigen Optimierungen auf REPREATABLE READ und SNAPSHOT TABLE STABILITY auf SERIALIZABLE abgebildet.

Seit Version 2.0 konnte der Treiber alle Transaktionskonfigurationen unterstützen, die Firebird unterstützen kann, einschließlich der Tabellenreservierung („table blocking“). Dies wurde erreicht, indem die so genannte „EmbeddedSQL“ -Syntax in den alten Pre-Compiler *gpre* integriert wurde, um Aufrufe an die ODBC-API mit der Funktion `SQLExecDirect` vorzubereiten.

Locking

Firebird implementiert optimistisches Locking auf Zeilenebene unter allen Bedingungen. Eine Transaktion versucht nicht, einen Datensatz zu sperren, bis er eine Aktualisierungsoperation bereitstellen kann, die sich auf diesen Datensatz auswirkt. Es kann vorkommen, dass ein Update fehlschlägt, weil ein anderer Client eine Sperre für den Datensatz hat, selbst wenn die fehlgeschlagene Transaktion vor der Transaktion gestartet wurde, die die Sperre gesichert hat.

Die Recording-Engine von Firebird ist in der Lage, eine Granularität zu erzielen, die feiner ist als bei der herkömmlichen Sperrung auf Zeilenebene. Bei der Versionsverwaltung kann eine beliebige Anzahl von Transaktionen eine konsistente Kopie eines bestimmten Datensatzes lesen, auch wenn andere Transaktionen dieselbe Zeile gleichzeitig aktualisieren. Leser und Schreiber blockieren sich nie gegenseitig und Firebirds Wartung von Datensatzversionen ist für den Benutzer völlig transparent.

Transaktionsanforderungssyntax

Die Syntax für eine ODBC-freundliche Transaktionsanforderung folgt.

```
SET | DECLARE TRANSACTION [LOCAL] [NAME <transaction-name> [USING <namedUniqueWorkspace>]]
[READ WRITE | READ ONLY]
[WAIT | NO WAIT]
[AUTOCOMMIT]
```

```
[NO_AUTO_UNDO]
[[ISOLATION LEVEL] {SNAPSHOT [TABLE STABILITY] or REPEATABLE READ
| SERIALIZABLE
| READ COMMITTED [[NO] RECORD_VERSION]}]
[RESERVING <table-name-1> [, <table-name-2>[, ...<table-name-n>] ]
[FOR [SHARED | PROTECTED] {READ | WRITE}] [, ]
```

Was bedeuten die Optionen?

DECLARE TRANSACTION ... deklariert die beschriebene Transaktion, ohne sie zu aktivieren. SET TRANSACTION ... hingegen aktiviert die Transaktion und schaltet das globale Attribut SQL_ATTR_AUTOCOMMIT der ODBC-API vorübergehend auf SQL_AUTOCOMMIT_OFF um. Die Transaktion muss explizit abgeschlossen werden. Wenn sie endet, wird die Regel der API fortgesetzt.

LOCAL beschränkt eine Transaktion auf die Ausführung nur im Kontext der aktuellen Verbindung.

<transaction-name> ist eine Transaktion mit eindeutigem Namen, die für die Verwendung durch beliebige Verbindungen in der globalen Umgebung vorbereitet ist.

USING <namedUniqueWorkspace> ist ein eindeutig benannter Transaktionsarbeitsbereich NAME <transaction-name>, der so eingestellt werden kann, dass sie von beliebigen Verbindungen in der globalen Umgebung ausgeführt wird. Identisch benannte Transaktionen mit unterschiedlichen Parametern können in demselben benannten Arbeitsbereich ausgeführt werden.

Benannte Transaktionen und Transaktionsarbeitsbereiche

Das Konstrukt DECLARE TRANSACTION ... NAME <transaction-name> [USING <namedUniqueWorkspace>] ermöglicht es, explizite Transaktionen zu konfigurieren und in der globalen Umgebung zu speichern, um sie für eine wiederholte Verbindungsanfrage oder eine aktive Verbindung vorzubereiten. Eine Instanz der gespeicherten Transaktion kann durch ein bestimmtes Formular des Befehls SET TRANSACTION aufgerufen werden:

Für eine Verbindungsanfrage:

```
SET TRANSACTION NAME MyReadTransaction
```

oder

```
SET TRANSACTION NAME MyReadTransaction USING MyDsnDb1
```

für separate Anfragen innerhalb einer einzigen aktiven Verbindung:

```
SET TRANSACTION LOCAL NAME MyReadTransaction
```

oder

```
SET TRANSACTION LOCAL NAME MyReadTransaction USING MyDsnDb1
```

und in diesem Zusammenhang für eine andere Anfrage:

```
SET TRANSACTION LOCAL NAME MyWriteTransaction
```

oder

```
SET TRANSACTION LOCAL NAME MyWriteTransaction USING MyDsnDb1
```

Die Form SET TRANSACTION ... NAME <transaction-name> [USING <namedUniqueWorkspace>] unterscheided sich von früheren Implementierungen, wobei die Konfiguration durch den Befehl SET für die nächste Transaktion erneut verwendet werden kann. Die Einbeziehung der NAME- und/oder USING-Klauseln macht die Konfiguration bei Bedarf durch Verwendung des Namens wiederholbar.

> **Wichtig**
>
> Eine Rückkehr zu dem normalen Betriebsmodus erfordert einen Trenn-/Verbindungszyklus.

Explizite Transaktionen beenden

In SQL wird eine Transaktion mittels COMMIT oder ROLLBACK oder beendet. ODBC hat Methoden, die das eine oder andere tun, wie z.B. `SQLEndTran`. Einige Programme können `SQLExecDirect` aufrufen, jedoch nicht `SQLEndTran`. Für diese Programme muss man explizit

```
SQLExecDirect( hStmt, "COMMIT" )
```

aufrufen, um sicherzustellen, dass die Schnittstelle

```
SQLEndTran( SQL_HANDLE_DBC, hConnection, SQL_COMMIT );
```

abschließend aufruft.

> **Anmerkung**
>
> Wenn eine Transaktion lokal initiiert wird, wird der Treiber `SQLEndTran` für das lokale `hStmt` aufrufen.

Zwei-Phasen-Commit-Transaktionen

Der ODBC/JDBC-Treiber unterstützt zweiphasige Festschreibungstransaktionen, d.h. eine einzelne Transaktion in verschiedenen Firebird-Datenbanken. Auf bis zu 16 Datenbanken kann gleichzeitig in einer solchen Transaktion zugegriffen werden, was ein absolutes Limit darstellt.

Der Aufruf zum Starten einer zweiphasigen Commit-Transaktion lautet:

```
SQLSetConnectAttr (connection, 4000, (void*) TRUE, 0);
```

So brechen Sie die gemeinsame Verbindung ab:

```
SQLSetConnectAttr (connection, 4000, (void*) FALSE, 0);
```

Mehr Transaktionen

Firebird ODBC verwendet standardmäßig eine Transaktion pro Verbindung. Programmatisch können Sie eine flexiblere Transaktionsstruktur verwenden. Beispielsweise können Sie mehrere Transaktionen innerhalb einer Verbindung verwenden, wobei eine einzelne Verbindung mehrere Lese-/Schreibtransaktionen gleichzeitig verwenden kann.

Ein Beispiel

```
    HSTMT stmtRd;
    HSTMT stmtWr;
    SQLAllocHandle( SQL_HANDLE_STMT, connection, &stmtRd );
    SQLAllocHandle( SQL_HANDLE_STMT, connection, &stmtWr );
    SQLExecDirect( stmtRd, (UCHAR*)
     "SET TRANSACTION LOCAL\n"
     "READ ONLY\n"
     "ISOLATION LEVEL\n"
     "READ COMMITTED NO RECORD_VERSION WAIT\n",
     SQL_NTS );
    SQLExecDirect( stmtWr, (UCHAR*)
     "SET TRANSACTION LOCAL\n"
     "READ WRITE\n"
     "ISOLATION LEVEL\n"
     "READ COMMITTED NO RECORD_VERSION WAIT\n",
     SQL_NTS );
    SQLExecDirect( stmtRd,(UCHAR*)
     "SELECT CURRENCY FROM COUNTRY"
     "   WHERE country = 'Canada'"
     "   FOR UPDATE OF CURRENCY",
     SQL_NTS );
    SQLFetch( stmtRd );
    SQLPrepare( stmtWr, (UCHAR*)
  "update COUNTRY\n"
  "set    CURRENCY = 'CndDlr'\n"
  "where  COUNTRY = 'Canada'\n",
  SQL_NTS );
    SQLExecute( stmtWr );
    SQLExecDirect( stmtWr, (UCHAR*)"COMMIT", SQL_NTS );
```

MS DTC-Transaktionen

Der Microsoft Distributed Transaction Coordinator-Dienst (MS DTC) ist eine Windows-Komponente, die für die Koordination von Transaktionen verantwortlich ist, die mehrere Ressourcenmanager umfassen, z. B. Datenbanksysteme, Nachrichtenwarteschlangen und Dateisysteme. Es kann globale, einphasige oder zweiphasige Commit-Transaktionen mit MSSQL Server, Sybase und anderen Servern durchführen, die damit arbeiten können. Unser ODBC/JDBC-Treiber bietet diese Möglichkeit für Firebird-Server.

Ein Beispiel mit MS DTC

```
// Include MS DTC specific header files.
//--------------------------------------------------------------------------
#define INITGUID
#include "txdtc.h"
#include "xolehlp.h"
    ITransactionDispenser *pTransactionDispenser;
    ITransaction *pTransaction;
    // Obtain the ITransactionDispenser Interface pointer
    // by calling DtcGetTransactionManager()
    DtcGetTransactionManager( NULL,// [in] LPTSTR pszHost,
         NULL,// [in] LPTSTR pszTmName,
         IID_ITransactionDispenser,// [in] REFIID rid,
         0,// [in] DWORDdwReserved1,
         0, // [in] WORDwcbReserved2,
         NULL,// [in] void FAR * pvReserved2,
         (void **)&pTransactionDispenser // [out] void** ppvObject
         );
    // Establish connection to database on server#1
    LogonToDB( &gSrv1 );
    // Establish connection to database on server#2
    LogonToDB( &gSrv2 );
    // Initiate an MS DTC transaction
    pTransactionDispenser->BeginTransaction(
         NULL,// [in] IUnknown __RPC_FAR *punkOuter,
         ISOLATIONLEVEL_ISOLATED,// [in] ISOLEVEL isoLevel,
         ISOFLAG_RETAIN_DONTCARE,// [in] ULONG isoFlags,
         NULL,// [in] ITransactionOptions *pOptions
         &pTransaction// [out] ITransaction **ppTransaction
         );
    // Enlist each of the data sources in the transaction
    SQLSetConnectOption( gSrv1->hdbc, SQL_COPT_SS_ENLIST_IN_DTC, (UDWORD)pTransaction );
    SQLSetConnectOption( gSrv2->hdbc, SQL_COPT_SS_ENLIST_IN_DTC, (UDWORD)pTransaction );
    // Generate the SQL statement to execute on each of the databases
    sprintf( SqlStatement,
      "update authors set address = '%s_%d' where au_id = '%s'",
       gNewAddress, i, gAuthorID );
    // Perform updates on both of the DBs participating in the transaction
    ExecuteStatement( &gSrv1, SqlStatement );
    ExecuteStatement( &gSrv2, SqlStatement );
    // Commit the transaction
    hr = pTransaction->Commit( 0, 0, 0 );
    // or roll back the transaction
    //hr = pTransaction->Abort( 0, 0, 0 );
```

Kennwort-Sicherheit

Wenn ein DSN mit dem Benutzernamen und dem Kennwort erstellt wird, ist das Datenbankkennwort verschlüsselt und wird in `odbc.ini` gespeichert. Alternativ können die Anmeldedaten während der Datenbankverbindungsphase eingegeben oder mit der Verbindungszeichenfolge übergeben werden.

Cursor

Im aktuellen Firebird ODBC/JDBC-Treiber werden die Dynamic- und Keyset-Cursor so modifiziert, dass sie den statischen Cursor verwenden, über den keine Sets aktualisiert werden können.

Für die beste Leistung, nutzen Sie den Cursor `ForwardOnly`.

Die Leseoperatoren `SQLFetch`, `SQLExtendedFetch` und `SQLScrollFetch` nutzen `SQL_ROWSET_SIZE` und `SQL_ATTR_ROW_ARRAY_SIZE`.

Verwenden Sie den Operator `SQLBindParameter`, unabhängig von der Größe des BLOB-Felds, um die beste Leistung mit BLOB-Feldern zu erzielen, da dies viel schneller als mit `SQLPutData/SQLGetData` funktioniert.

Rufen Sie die folgenden Anweisungen auf, um die Cursor des Firebird-Treibers zu verwenden:

```
// Specify that the Firebird ODBC Cursor is always used, then connect.
SQLSetConnectAttr( hdbc, SQL_ATTR_ODBC_CURSORS, (SQLPOINTER)SQL_CUR_USE_DRIVER, 0 );
SQLConnect( hdbc, (UCHAR*)connectString, SQL_NTS, NULL, 0, NULL, 0 );
```

ODBC Cursor-Bibliothek

Dieses Thema ist in MSDN ausführlich dokumentiert. Wir müssen jedoch die absolute Anforderung betonen, diese Anweisungen vor dem Verbinden zu verwenden:

```
// Geben Sie an, dass die ODBC-Cursor-Bibliothek immer verwendet wird, und verbinden Si
SQLSetConnectAttr( hdbc, SQL_ATTR_ODBC_CURSORS, (SQLPOINTER)SQL_CUR_USE_ODBC, 0 );
SQLConnect( hdbc, (UCHAR*)connectString, SQL_NTS, NULL, 0, NULL, 0 );
```

Diese Daten setzen Schlüssel (?) In den Rowset-Puffern. Das Aktualisieren der Puffer erfordert diese Anweisung:

```
SQLFetchScroll( hstmtSel, SQL_FETCH_RELATIVE, 0 );
```

Gespeicherte Prozeduren

In Firebird können wir zwei Arten von gespeicherten Prozeduren haben, bekannt als *ausführbare* und *auswählbar* (abfragbar). Beide Typen können Eingabeparameter und Rückgabewerte verwenden, unterscheiden sich jedoch sowohl in der Schreibweise als auch im Aufrufmechanismus.

- Die Ausgabe einer ausführbaren Prozedur ist optional und jede zurückgegebene Ausgabe ist eine Menge von nicht mehr als einer „Zeile“ von Werten. Wenn die Ausgabe definiert ist und keine Ausgabe erfolgt, ist die Ausgabe null.

Das Zurückgeben von Daten ist nicht das primäre Ziel einer ausführbaren Prozedur. Sein Zweck ist es, Datenoperationen auszuführen, die für den Benutzer unsichtbar sind.

Der Mechanismus zum Aufrufen einer ausführbaren Prozedur ist die SQL-Anweisung EXECUTE PROCEDURE. Zum Beispiel

```
execute procedure MyProc(?,?)
```

- Eine abfragbare Prozedur wird mit dem Ziel geschrieben, einen Datensatz von null, einer oder mehreren Datenzeilen zurückzugeben. Es kann verwendet werden, um Daten zu ändern, aber es sollte nicht dafür geschrieben werden. Die PSQL-Anweisung SUSPEND wird in dieser Prozedur verwendet, um eine Ausgabezeile zu übergeben, die innerhalb einer Iteration einer FOR SELECT ...-Schleife in einem Puffer gesammelt wurde.

  Der Mechanismus zum Aufrufen einer auswählbaren Prozedur ist die SQL-Anweisung SELECT.

In diesem Beispiel haben wir eine auswählbare Prozedur, von der wir erwarten, basierend auf den Eingabeparametern eine Menge von null oder mehr Zeilen zu erhalten:

```
select * from MyProc(?,?)
```

Microsoft Excel und einige andere Anwendungen verwenden diese Anweisung, um eine gespeicherte Prozedur aufzurufen:

```
{[? =] Call MyProc (?,?)}.
```

Der Firebird ODBC/JDBC-Treiber bestimmt aus den Metadaten der Firebird-Engine, welcher Aufruf beim Ausführen einer gespeicherten Prozedur verwendet werden soll. Firebird markiert eine Prozedur als 'ausführbar' oder 'auswählbar' entsprechend der Anzahl der SUSPEND-Anweisungen im zusammengesetzten (BLR) Code ihrer Definition. Für ein triviales Beispiel:

```
create procedure TEST
  as
    begin
    end
```

Da die Prozedur über keine SUSPEND-Anweisungen verfügt, kann der ODBC-Treiber den Aufruf als execute procedure TEST weitergeben.

Für dieses Verfahren:

```
create procedure "ALL_LANGS"
   returns ("CODE" varchar(5),
         "GRADE" varchar(5),
         "COUNTRY" varchar(15),
         "LANG" varchar(15))
   as
   BEGIN
```

```
    "LANG" = null;
    FOR SELECT job_code, job_grade, job_country FROM job
    INTO :code, :grade, :country
    DO
      BEGIN
        FOR SELECT languages FROM show_langs(:code, :grade, :country)
        INTO :lang
          DO
            SUSPEND;
            /* Put nice separators between rows */
            code = '=====';
            grade = '=====';
            country = '===============';
            lang = '==============';
            SUSPEND;
      END
    END
```

Der BLR-Code für die gespeicherte Prozedur enthält mehr als null SUSPEND-Anweisungen, sodass der ODBC-Treiber select * from "ALL_LANGS" verwendet.

ARRAY-Datentypen

Um eindimensionale Array-Datentypfelder zu ändern, müssen Sie die folgenden Regeln beachten:

- Geben Sie einfache Typen an (INTEGER, etc.) als {1, 2, 3}
- Geben Sie einfache Typen an (CHAR, etc.) als {'1', '2', '3'}

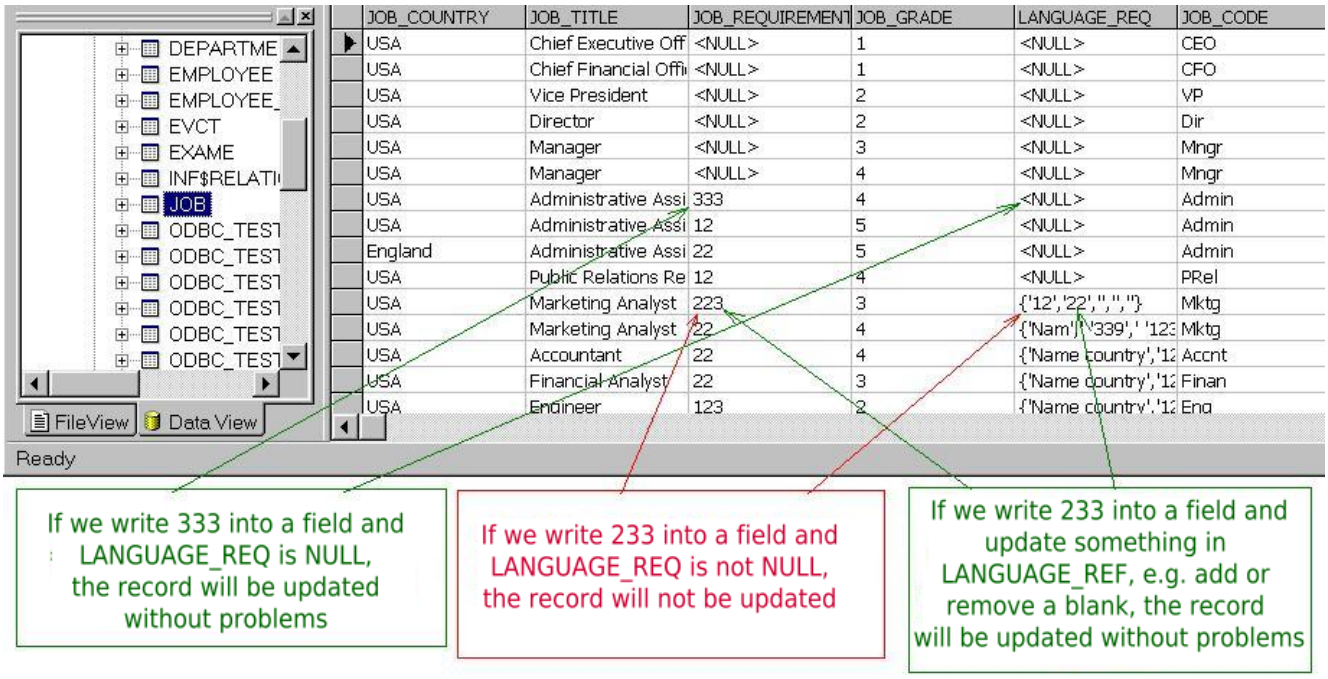
FALLEN!

Wenn Sie z.B. ein Element des Arrays 1, 2 und 5 spezifizieren, jedoch nicht die anderen Elemente des Arrays, z.B. 3 und 4, dann werden die anderen Elemente des Arrays auf Null (Integer) oder leer (String) gesetzt.

Bei einigen Programmen, deren Spalten von Array-Daten abhängig sind, können Array-Daten in eine aktuelle Array-Spalte NULL eingegeben werden, ohne dass eine Gültigkeitsprüfung der verschiedenen Array-Elemente durchgeführt wird. Unter diesen Umständen müssen die Array-Elemente vor dem Eingeben der Spaltendaten angegeben werden.

Abbildung 5.1. Datenverlust beim Aktualisieren eines ARRAY-Feldes (1)

Abbildung 5.2. Datenverlust beim Aktualisieren eines ARRAY-Feldes (2)



Verwendung mit Clarion

Jorge Andres Brugger
Vernon Godwin
Vladimir Tsvigun

Clarion-Benutzer können in Firebird mit Namen für gemischte Groß- und Kleinbuchstaben arbeiten.

1. Erstellen Sie Ihre Datenbank mit Firebird. Sie können Tabellennamen wie "Pending_Invoices" und Felder wie "Order_Number" angeben.

2. Erstellen Sie den DSN für die Datenbank, und stellen Sie sicher, dass alle Optionen in "Extended Identifier Properties" aktiviert sind

3. Öffnen Sie Ihr Wörterbuch und importieren Sie mehrere Tabellen wie gewohnt aus der ODBC-Quelle. Es wird funktionieren, aber versuchen Sie nicht, die Dateien in einer Anwendung noch zu durchsuchen oder zu verwenden.

4. Geben Sie für jedes Feld den Namen "External Name" des Felds ein, das von Anführungszeichen umgeben ist (geben Sie beispielsweise "Order_Number" im externen Namen ein).

Das ist es! Verwenden Sie jetzt Ihr Wörterbuch mit den Groß- und Kleinbuchstaben, ohne Probleme. Denken Sie jedoch daran—Sie müssen in allen SQL-Anweisungen innerhalb von Clarion doppelte Anführungszeichen um Objektnamen verwenden.

Kapitel 6

Firebird-Ereignisse

Um die Verwendung von Firebird-Ereignissen mit dem ODBC/JDBC-Treiber zu veranschaulichen, verwenden wir die Beispieldatenbank `employee.fdb` und arbeiten mit der Tabelle SALES. Diese Tabelle enthält einen AFTER INSERT-Trigger POST_NEW_ORDER, der die Anweisung POST_EVENT 'new_order'; enthält. Seine Wirkung besteht darin, einem Listener auf der Clientseite zu signalisieren, wenn ein neuer Datensatz in SALES übergeben wird.

Nehmen wir an, dass die Tabelle auch einen BEFORE UPDATE-Trigger hat, der ein Ereignis "change_order" in nachfolgenden Operationen veröffentlicht, wenn das Feld ORDER_STATUS geändert wird.

Tipp

Der Trigger BEFORE UPDATE existiert nicht, dieses Szenario dient nur zur Veranschaulichung, aber Sie könnten es erstellen, wenn Sie möchten:

```
CREATE OR ALTER TRIGGER BI_SALES FOR SALES
ACTIVE BEFORE UPDATE
AS BEGIN
  IF (NEW.ORDER_STATUS = 'new') THEN
  BEGIN
     NEW.ORDER_STATUS = 'open';
     POST_EVENT 'change_order';
  END
END
```

Für unsere Demo müssen wir einen neuen Datensatz in SALES einfügen. Das Feld ORDER_STATUS auf dem neu eingefügten Datensatz enthält den Standardwert 'new'. Nach dem Festschreiben, indem das Ereignis 'new_order' gepostet wird, möchten wir zurückgehen und etwas im neuen Datensatz ändern. Wenn Sie dies tun, überprüft unser BEFORE UPDATE-Trigger BI_SALES, ob der Wert von ORDER_STATUS immer noch 'new' ist und wenn ja, dann wird es in "open" ändern und das Ereignis "change_order" veröffentlichen.

Anmerkung

Es interessiert uns nicht wirklich, wie sich das Einfügen und Ändern des Datensatzes auf den Datenbankzustand auswirkt. Die Idee hier ist, zu zeigen, wie der Treiber für die Verwaltung von mehreren Ereignissen eingerichtet wird.

Den Treiber dazu veranlassen, auf Ereignisse zu warten

Der erste Schritt zum Einrichten des Treibers für das Abhören von Ereignissen besteht darin, eine Verbindung zu einer ODBC-Schnittstellendatei herzustellen, die die Verarbeitung von Firebird-Ereignissen beschreibt:

```
#include "OdbcUserEvents.h"
```

Als nächstes spezifizieren wir in der Tabelle `eventInfo` die Ereignisse, an denen wir interessiert sind. Für unser Beispiel ist das Ereignis 'new_order' das einzige, an dem wir uns zu diesem Zeitpunkt interessieren. Das Ereignis 'change_order' ist nur auf dem Bild zu sehen, um die Fähigkeit des Treibers zu demonstrieren, mehrere Ereignisse zu verwalten.

```
ODBC_EVENT_INFO eventInfo[] =
{
 INIT_ODBC_EVENT("new_order"),
 INIT_ODBC_EVENT("change_order")
};
```

Nun müssen wir eine Struktur erstellen — die wir `MyUiqueData` nennen werden— um die Datenaufgaben zu speichern, die an unserer Operation beteiligt sind. In unserem Beispiel wird ein Feld `event_flag` ein Ereignis signalisieren, das vom Server geliefert wird. Unsere Arbeit beginnt an dieser Stelle.

```
struct MyUniqueData
{
 int event_flag;
 //... andere Definitionen für die Verwendung in astRoutine
};
```

Wir müssen eine Callback-Funktion erstellen, `astRoutine`, die aktiviert wird, wenn in der `eventInfo`-Tabelle definierte Ereignisse markiert sind:

```
void astRoutine( void *userEventsInterfase, short length, char * updated )
{
    PODBC_USER_EVENTS_INTERFASE userInterface = (PODBC_USER_EVENTS_INTERFASE)userEventsInte
    SQLSetConnectAttr( userInterfase->hdbc, SQL_FB_UPDATECOUNT_EVENTS, (SQLPOINTER)updated,
    MyUniqueData &myData = *(MyUniqueData*)userInterfase->userData;
    myData.event_flag++;
    printf( "ast routine was called\n" );
}
```

Die Funktion muss einen Anruf haben:

```
  SQLSetConnectAttr( userInterfase->hdbc,
                       SQL_FB_UPDATECOUNT_EVENTS,
                       (SQLPOINTER)updated,
                       SQL_LEN_BINARY_ATTR( length ) );
```

Dieser Aufruf wird benötigt, um den Status von Ereignissen in unserer Struktur `eventInfo` zu aktualisieren. Diese Struktur hat ein Feld `countEvents`, das eine Gesamtzahl von Ereignisoperationen verwaltet, und ein Boolesches Feld `changed`, das auf Wahr gesetzt wird, wenn die 'vor'- und 'nach'-Werte von `countEvents` unterschiedlich sind.

Wenn wir ein Ereignis, an dem wir interessiert sind, kennzeichnen möchten, geben wir folgenden Befehl aus:

```
myData.event_flag++;
```

Es bietet einen ziemlich primitiven Mechanismus zur Synchronisierung von Arbeitsabläufen, aber es ist ausreichend für unsere Bedürfnisse. Seine Einrichtung ist wie folgt:

- Zur Verbindungszeit oder beim Erstellen des DSN muss die Option NOWAIT auf OFF gesetzt werden
- Die folgenden Anweisungen müssen ausgegeben werden:

```
// Geben Sie an, dass der Firebird ODBC-Cursor immer verwendet wird, und verbinden Si
SQLSetConnectAttr( hdbc, SQL_ATTR_ODBC_CURSORS, (SQLPOINTER)SQL_CUR_USE_DRIVER, 0 );
SQLConnect( hdbc, (UCHAR*)connectString, SQL_NTS, NULL, 0, NULL, 0 );
```

- Zum Zweck unserer Demonstration müssen wir eine SQL-Cursor-Anfrage vorbereiten. Ihr eigenes, realistisches Szenario wäre natürlich weniger trivial.

```
  SQLPrepare( stmtSel, (UCHAR*)
"SELECT po_number"
" FROM sales"
             " WHERE order_status = 'new'"
" FOR UPDATE",
SQL_NTS );
```

- Wir konstruieren die Cursor-Abfrage für unsere Demo und nennen sie 'C':

```
  char *cursor = "C";
  SQLSetCursorName( stmtSel, (UCHAR*)cursor, sizeof( cursor ) );

  SQLPrepare( stmtUpd, (UCHAR*)
"UPDATE sales"
             " SET order_status = 'open'"
" WHERE CURRENT OF C",
       SQL_NTS );
```

- Initialisieren Sie die Struktur `ODBC_EVENTS_BLOCK_INFO` als die Ereignisschnittstelle, die an den Treiber übergeben wird:

```
myData.event_flag = 0;
ODBC_EVENTS_BLOCK_INFO eventsBlockInfo = INIT_EVENTS_BLOCK_INFO( hdbc, eventInfo, ast
SQLSetConnectAttr( hdbc, SQL_FB_INIT_EVENTS, (SQLPOINTER)&eventsBlockInfo, SQL_LEN_BI
- to inform connection, that we are ready to accept events.
SQLSetConnectAttr( hdbc, SQL_FB_REQUEUE_EVENTS, (SQLPOINTER)NULL, 0 );
```

- Ereignisse beginnen ...

```
while ( !iret )
{
```

```
       // Wenn das Ereignis ausgelöst wurde, den Puffer zurücksetzen und neu einreihen
if ( myData.event_flag )
{
    myData.event_flag = 0;
    // Suchen Sie nach dem ersten ast_call. Mit isc_que_events
    // wird jedes Ereignis ausgelöst, um die Verarbeitung zu starten
    if ( first )
         first = 0;
    else
    {
 // Wählen Sie eine Abfrage aus, um nach ausgelösten Ereignissen zu suchen
 ret = SQLExecute( stmtSel );
 for (;;)
 {
  ret = SQLFetch( stmtSel );
  if ( ret == SQL_NO_DATA_FOUND )
   break;
  ret = SQLExecute( stmtUpd );
 }
    /* Warteschlange für das nächste Ereignis */
    SQLSetConnectAttr( hdbc, SQL_FB_REQUEUE_EVENTS, (SQLPOINTER)NULL, 0 );
    /* Das blockiert nicht, aber als Beispielprogramm gibt es für
    ** uns nichts anderes, also machen wir ein Nickerchen
    */
    Sleep(1000);
}
   }
```

Kapitel 7

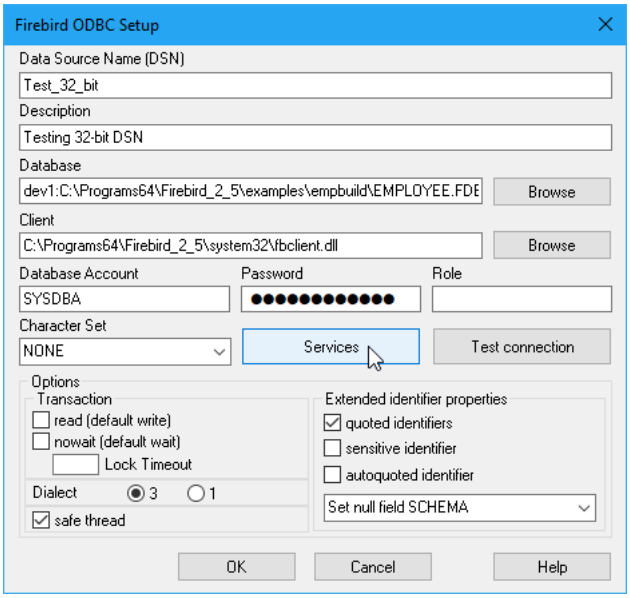
Die Service-Schnittstelle

Auf der Konfigurationsseite für Ihren Firebird-DSN unter Windows haben Sie Zugriff auf eine nützliche grafische Verwaltungskonsole, die über die ODBC API und die Firebird Services-API erstellt wird. Es gibt einem Datenbankadministrator unter Windows eine benutzerfreundliche Möglichkeit, Dienstprogramme auszuführen, die andernfalls über ein Befehlszeilentool ausgeführt würden. Wir verwenden es, um dieses Kapitel vorzustellen, da der Quellcode eine nützliche Ressource für Entwickler sein könnte, die nach Ideen suchen, Services-Funktionen in ihren Anwendungen zu integrieren.

Erkundung der ODBC-Services-Konsole

Um die Konsole zu verwenden, öffnen Sie diese Konfigurationsseite und klicken Sie auf die Schaltfläche in der Mitte mit der Beschriftung „Services“:

Abbildung 7.1. Starten der Service-Benutzeroberfläche unter Windows



Die Konsole ist ein Registerkarten-Display, das Zugriff auf viele Dienste-API-Funktionen bietet, wobei die Registerkarte "Backup" oben angezeigt wird.

Abbildung 7.2. Firebird ODBC-Services-Konsole—Backup-Register

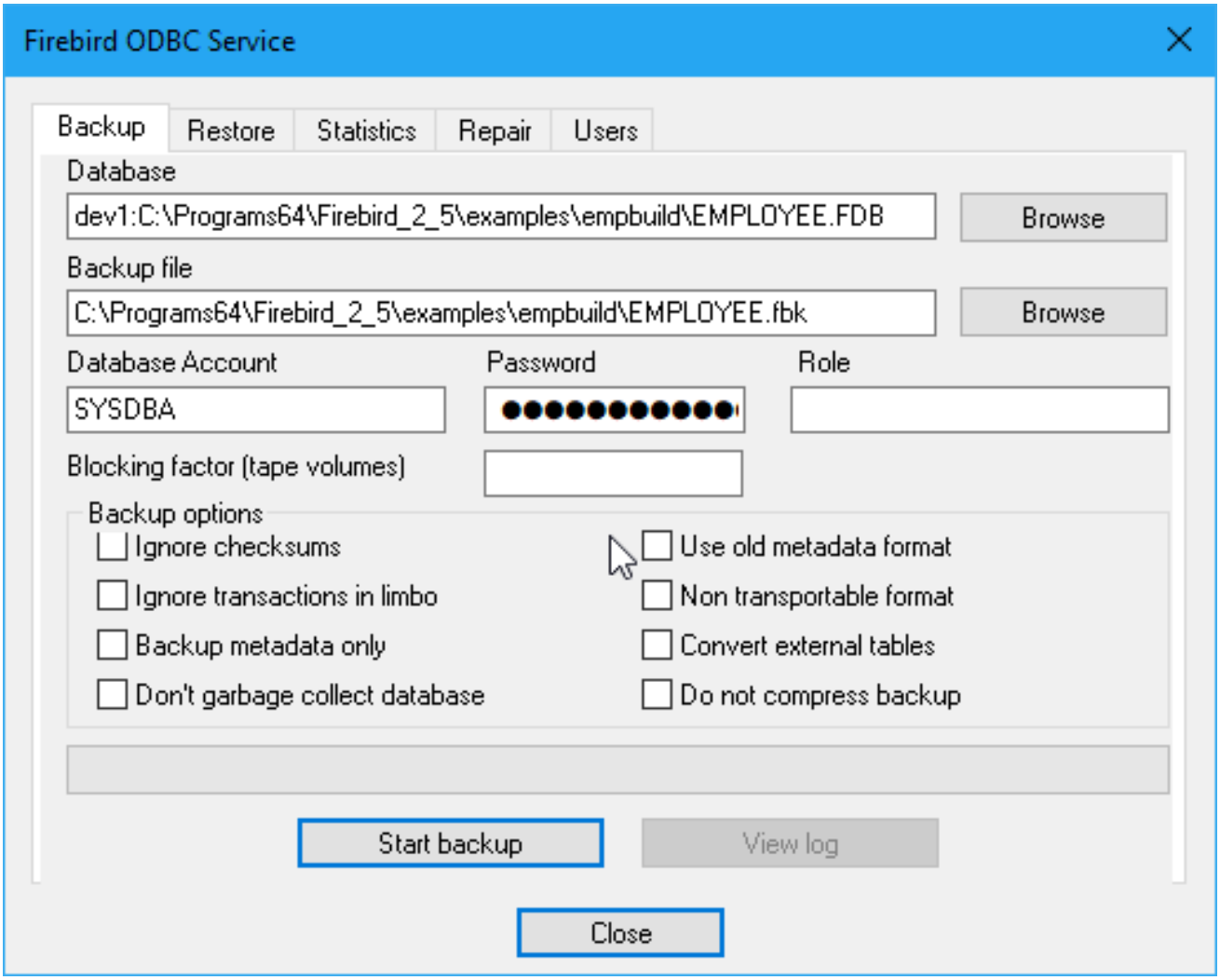


Abbildung 7.3. Restore-Register

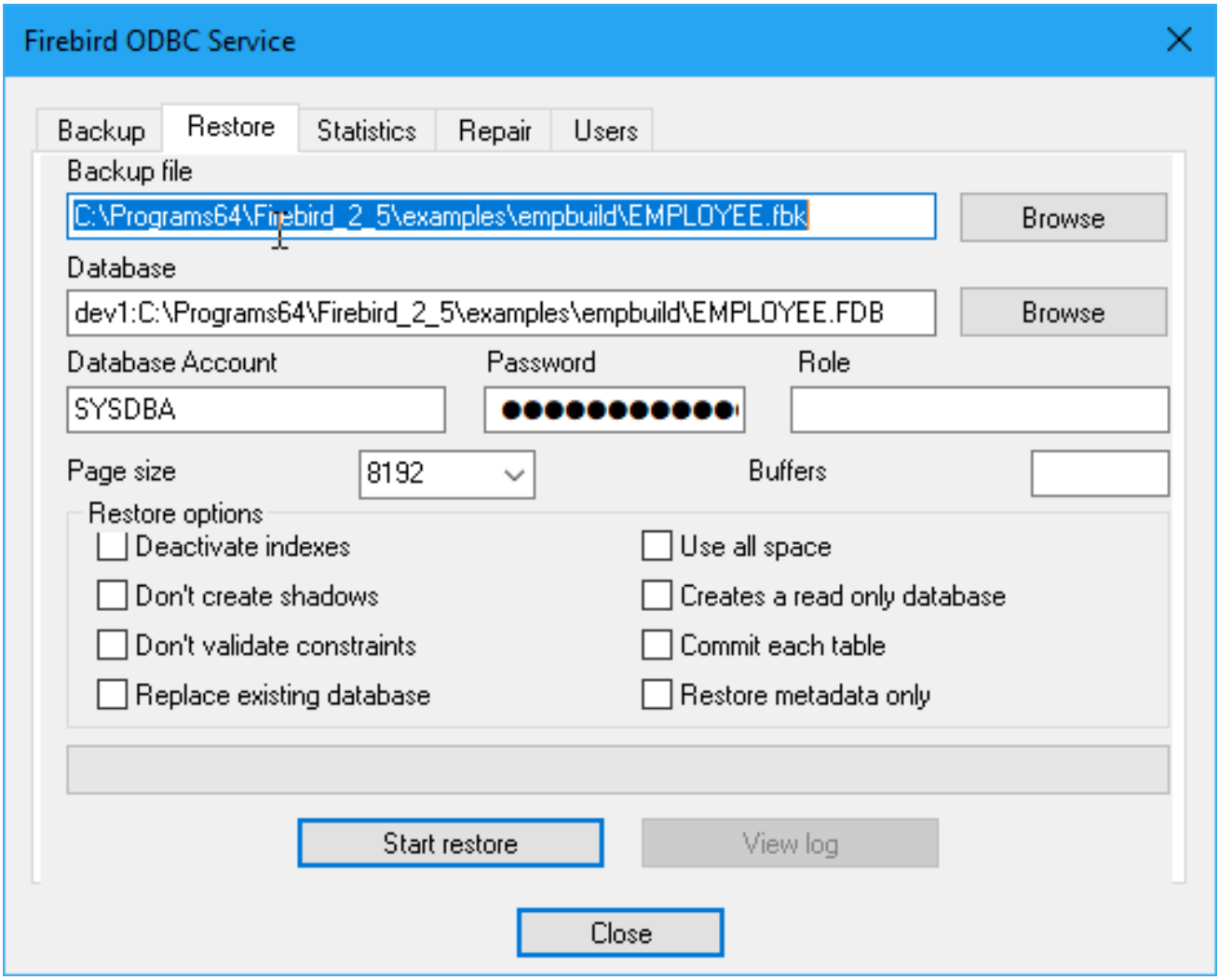
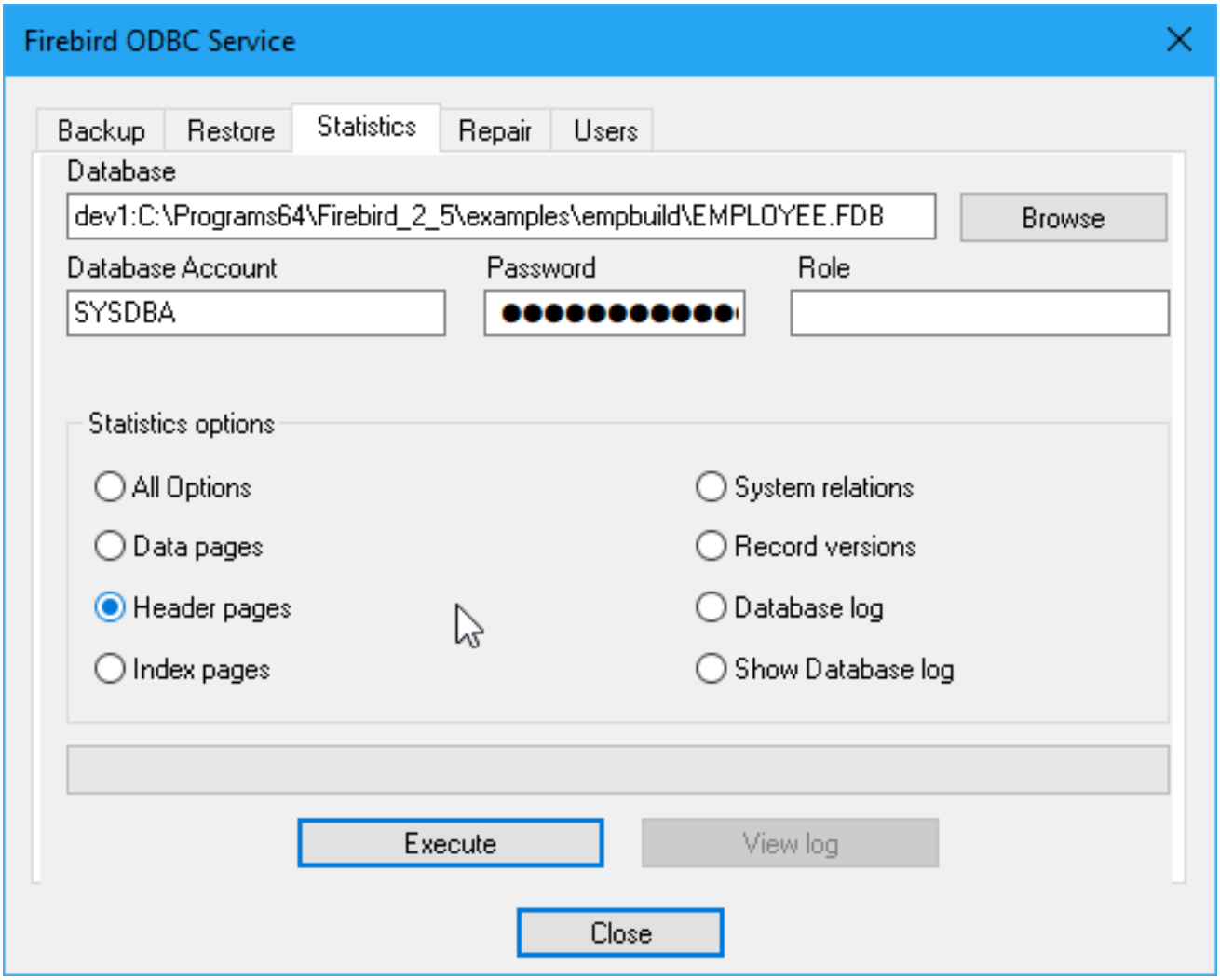
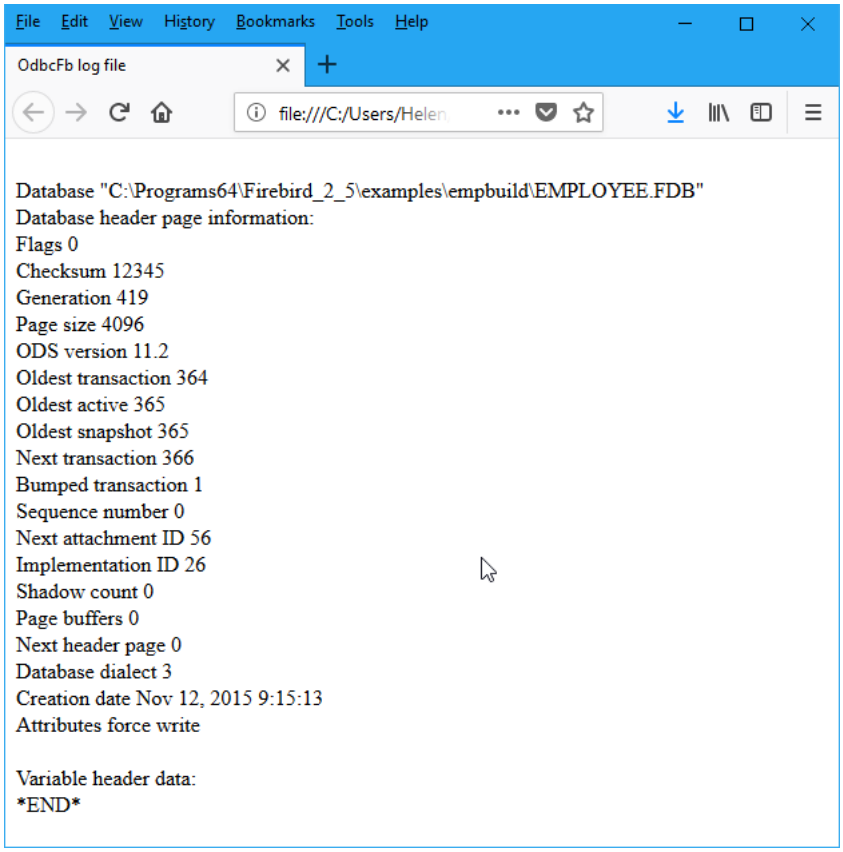


Abbildung 7.4. Statistics-Register



Wir haben „Header pages“ ausgewählt, die den `gstat -h`-Bericht für unsere Datenbank erstellt haben. Wenn Sie auf die Schaltfläche „View log“ klicken, wird die Ausgabe an den Browser gesendet:

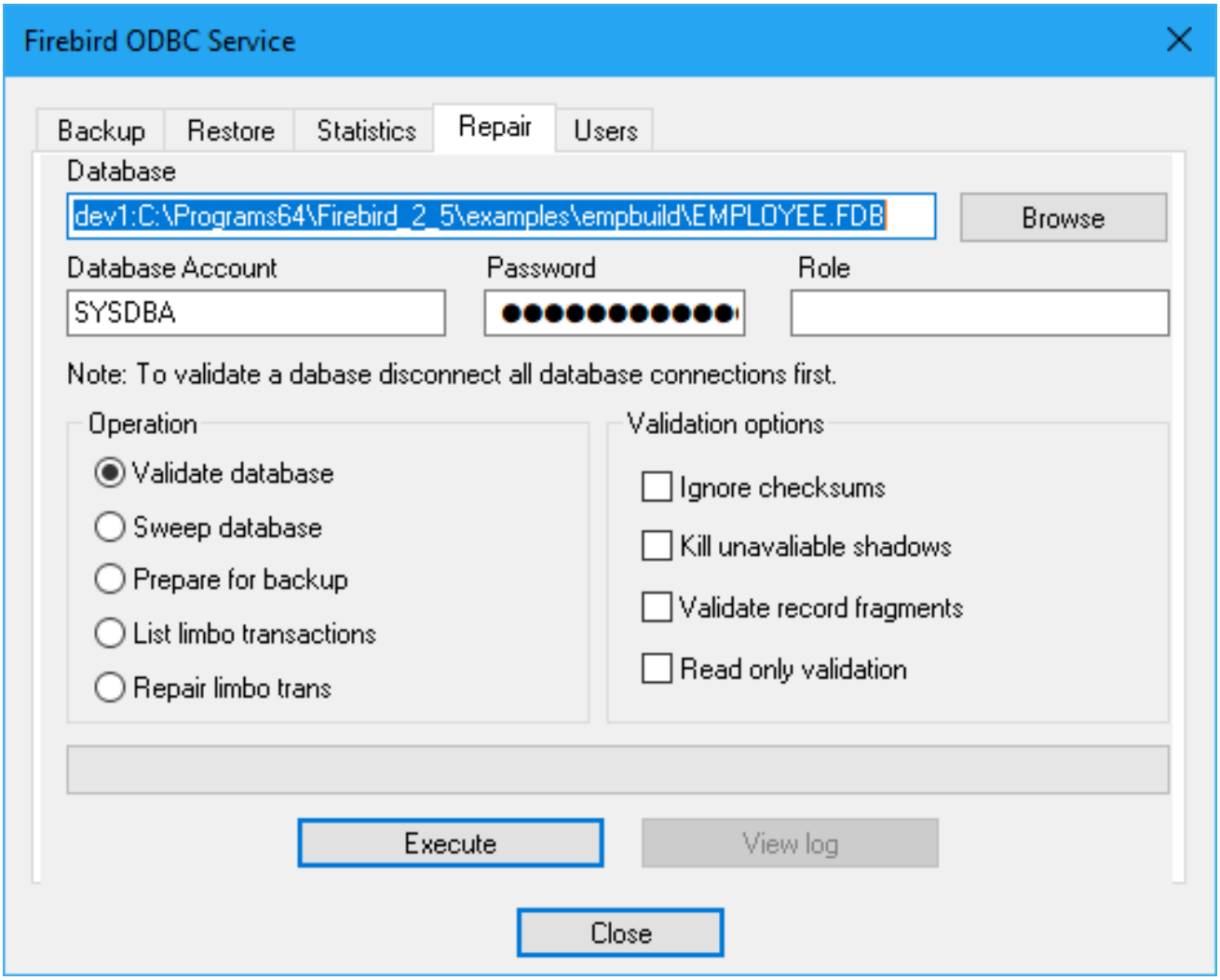
Abbildung 7.5. Statistics log



Natürlich können Sie einen Statistikbericht, das Firebird-Protokoll, Metadatenberichte und mehr haben.

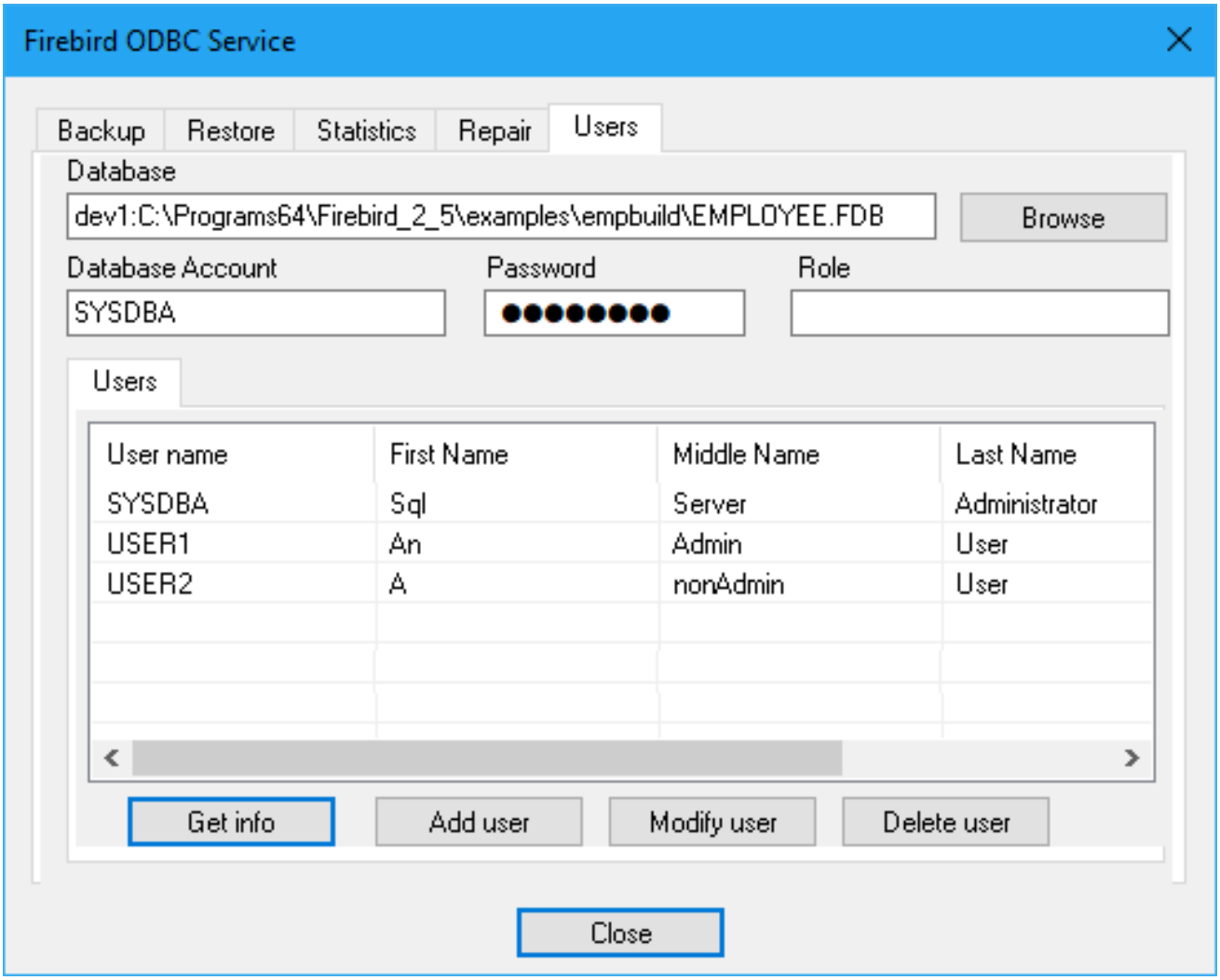
Die Registerkarte "Repair" bietet einfachen Zugriff auf die meisten der *gfix* Verwaltungsfunktionen:

Abbildung 7.6. Repair tab



Die Registerkarte Users konnte verwendet werden, um Konten in der Sicherheitsdatenbank einer Firebird-Version vor V.3.0 zu verwalten, obwohl ab V.2.5 von der Services-API-Methode abgeraten wurde. Die Services-API-Methode ist weiterhin verfügbar, um Benutzer in Firebird 3-Datenbanken zu verwalten, wenn sie mithilfe der Authentifizierungsverwaltung `Legacy_Auth` definiert wurden. Es funktioniert nicht mit Benutzern, die mit dem Standard-Authentifizierungs-Manager SRP definiert sind.

Abbildung 7.7. Users-Register



Klicken Sie auf die entsprechende Schaltfläche, um einen Benutzer hinzuzufügen, zu ändern oder zu löschen. Denken Sie daran, dass der Benutzer, der diese Aufgaben ausführt, SYSDBA oder ein Benutzer mit erhöhten Serverberechtigungen sein muss. Die Rolle RDB$ADMIN ist nicht ausreichend privilegiert.

Abbildung 7.8. Benutzer anlegen

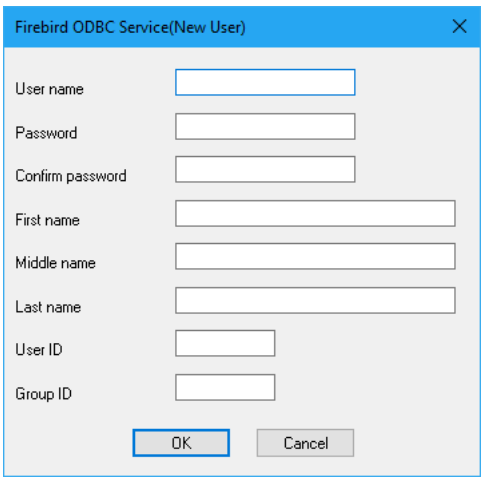


Abbildung 7.9. Benutzer ändern

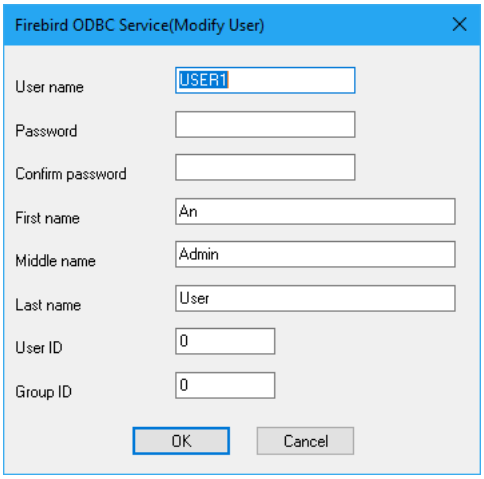
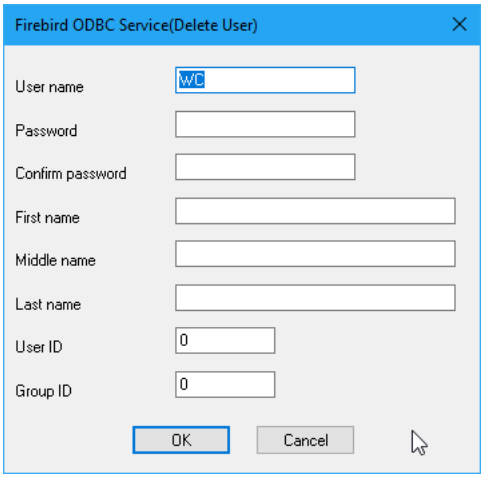


Abbildung 7.10. Benutzer löschen



Protokolle über die Benutzeroberfläche anzeigen

Wenn eine Protokolldatei bei der Ausführung einer Service-API-Funktion verfügbar ist, wird die Schaltfläche „View Log“ aktiviert. Die Benutzeroberfläche stellt sie bei Bedarf im HTML-Format bereit und öffnet sie in Ihrem Standardbrowser. Wenn Sie sich fragen, wie Sie dies in Ihre eigene ODBC-Anwendung einprogrammieren können, ist der Quellcode eine Ressource, die Ihnen kostenlos zur Verfügung steht.

Verwenden der Services-API

Der ODBC/JDBC-Treiber umschließt viele der Service-API-Funktionen. Die Managementkonsole, die in die Windows DSN-Schnittstelle integriert ist, enthält Beispiele für die meisten von ihnen. Eine Sache, die Sie nicht über die Konsole tun können, ist das Erstellen von Datenbanken. Doch keine Angst! Der Treiber hat es eingepackt!

Im Kapitel "Verbindung" finden Sie eine Tabelle der verfügbaren Schlüsselwörter, die die Werte für Attechments über die „reguläre“ API von Firebird angeben. Die folgende Tabelle enthält die Schlüsselwörter für die

`KEYWORD=value`-Parameter zum Herstellen einer Verbindung zum Server und zum Starten einer Dienstanforderung. Diese sind zusätzlich zu den relevanten Verbindungsparametern. In einigen Fällen sind die Standardeinstellungen aus dem DSN, falls verwendet, für Serviceanforderungen korrekt.

Tabelle 7.1. Schlüsselwörter für Dienstanforderungsattribute

Schlüsselwort	Beschreibung	Mehr Informationen
BACKUPFILE	Backup-Datei	Dies ist ein Dateisystempfad und ein Dateiname. Im Gegensatz zu einer Datenbank kann ein Sicherungspfad eine Netzwerkspeicheradresse sein.
LOGFILE	Pfad und Name der Protokolldatei für den Service	Optional, gültig für jeden Dienst, der eine Protokolldateioption bereitstellt. Es gelten dieselben Dateisystemregeln wie für Sicherungsdateien.
CREATE_DB	Datenbank erstellen	Beachten Sie die folgenden Beispiele zur Verwendung
BACKUP_DB	Datenbank sichern	Der Pfad und Name der Datenbanksicherungsdatei für Sicherungen und Wiederherstellungen.
RESTORE_DB	Der Netzwerkpfad und der Name der Datenbank, für die eine Sicherung wiederhergestellt werden soll. Dies kann keine Netzwerkspeicheradresse sein. Der Dateiname kann ein Alias sein, sofern der Alias existiert.	...
REPAIR_DB	Datenbank reparieren	Lokaler Pfad zu der Datenbank, die repariert oder validiert werden soll. Der Fernzugriff ist ungültig.
COMPACT_DB	Datenbank komprimieren	Derzeit nicht auf Firebird-Datenbanken anwendbar.
DROP_DB	Datenbank löschen	Derzeit nicht auf Firebird-Datenbanken anwendbar.

Beispiele für die Verwendung von Services

Die folgenden Beispiele zeigen, wie die verschiedenen Serviceanforderungen konfiguriert werden.

Eine Datenbank erstellen

```
SQLConfigDataSource( NULL,
                     ODBC_ADD_DSN,
                     "Firebird/InterBase(r) driver",
```

```
                      "ODBC\0"
                      "CREATE_DB = D:\\TestService\\test.fdb\0"
                      "DESCRIPTION = My Firebird database\0"
                      "UID         = SYSDBA\0"
                      "PWD         = masterkey\0"
                      "CHARSET     = NONE\0"
                      "PAGESIZE    = 8192\0"
                      "DIALECT     = 3\0" );
```

Weitere alternative Beispiele zum Erstellen von Datenbanken finden Sie am Ende dieses Kapitels.

Eine Datenbank sichern

```
SQLConfigDataSource( NULL,
                     ODBC_ADD_DSN,
                     "Firebird/InterBase(r) driver",
                     "ODBC\0"
                     "BACKUP_DB = D:\\TestService\\test.fdb\0"
                     "BACKUPFILE = D:\\TestService\\test.fbk\0"
                     "UID         = SYSDBA\0"
                     "PWD         = masterkey\0" );
```

Eine Datenbank wiederherstellen

```
SQLConfigDataSource( NULL,
                     ODBC_ADD_DSN,
                     "Firebird/InterBase(r) driver",
                     "ODBC\0"
                     "RESTORE_DB = D:\\TestService\\testNew.fdb\0"
                     "BACKUPFILE = D:\\TestService\\test.fbk\0"
                     "LOGFILE = D:\\TestService\\test.log\0"
                     "UID         = SYSDBA\0"
                     "PWD         = masterkey\0" );
```

Eine Datenbank reparieren

```
SQLConfigDataSource( NULL,
                     ODBC_ADD_DSN,
                     "Firebird/InterBase(r) driver",
                     "ODBC\0"
                     "REPAIR_DB = D:\\TestService\\test.fdb\0"
                     "UID         = SYSDBA\0"
                     "PWD         = masterkey\0" );
```

Weitere Wege um eine Datenbank zu erstellen

Erstellen Sie eine Datenbank mit der ODBC-API-Funktion `SQLConfigDataSource`. Eine praktische Methode zum Erstellen einer Datenbank, die von jemand anderem verwaltet wird.

```
SQLConfigDataSource( NULL,
                     ODBC_ADD_DSN,
                     "Firebird/InterBase(r) driver",
                     "ODBC\0"
                     "CREATE_DB = D:\\TestService\\test.fdb\0"
                     "DESCRIPTION = My Firebird database\0"
                     "UID         = SYSDBA\0"
                     "PWD         = masterkey\0"
                     "CHARSET     = NONE\0"
                     "PAGESIZE    = 8192\0"
                     "DIALECT     = 3\0" );
```

Erstellen Sie eine Datenbank mit der ODBC-API-Funktion `SQLDriverConnect`. Praktisch, wenn der Job von einer Benutzeranwendung ausgeführt werden soll. Der Treiber wird Fehler behandeln und weiterhin versuchen, die Datenbank zu erstellen, bis dieser schließlich eine Verbindung herstellen kann. Der Zugriff wird beim Erfolg an den Client übergeben.

```
    UCHAR buffer[1024];
    SWORD bufferLength;
    SQLDriverConnect( connection, hWnd,
                      (UCHAR*)"DRIVER=Firebird/InterBase(r) driver;"
                      "UID=SYSDBA;"
                      "PWD=masterkey;"
                      "PAGESIZE=8192;"
                      "DBNAMEALWAYS=C:\\Temp\\NewDB.fdb", SQL_NTS,
                      buffer, sizeof (buffer), &bufferLength,
                      SQL_DRIVER_NOPROMPT );
```

Erstellen Sie eine Datenbank mit der ODBC-API-Funktion `SQLExecDirect`. Dieses Szenario ist insofern interessant, als die Datenbank im Kontext einer vorhandenen Clientverbindung erstellt wird. Es ist daher nicht notwendig ;"DRIVER = Firebird / InterBase (r) driver;" im Aufruf einzuschließen, da es von der aktuellen Verbindung verwendet wird.

Wie bei der ersten Methode, die `SQLConfigDataSource` verwendet hat, erhält der aktuelle Benutzer keine Verwaltungsrechte für die erstellte Datenbank. Für diese Anforderung sollte stattdessen `SQLDriverConnect` verwendet werden.

```
    SQLExecDirect( hStmt,
                   "CREATE DATABASE \'C:/TEMP/NEWDB00.FDB\'"
                   "   PAGE_SIZE 8192"
                   "   SET NAMES \'NONE\'"
                   "   USER \'SYSDBA\'"
                   "   PASSWORD \'masterkey\';",
                   SQL_NTS );
```

Kapitel 8

Verbindungsbeispiele

Diese Seite ist optimistisch (fast) leer.

```
Dieser Platz ist für Ihr beigetragenes Beispiel reserviert.
```

Wenn Sie etwas zu bieten haben, können Sie es zippen und es in den Tracker als Verbesserung eintragen, entweder in der ODBC oder der DOC-Sektion.

Wir würden eine kurze Beschreibung begrüßen, die angbit, was Ihr Beispiel zeigt, in welcher Programmier- oder Skriptsprache und auf welcher OS-Plattform Sie es getestet haben.

Unsterblichkeit kommt in so vielen Gestalten vor.

Anhang A: Lizenzhinweise

Dokumentations-Lizenz

Die originale Dokumentation wurde unter dem Titel *Firebird ODBC/JDBC Driver Manual* veröffentlicht.

Die ursprünglichen Schreiber der Originaldokumentation sind: Alexander Potapchenko, Vladimir Tsvigun, James Starkey und andere.

Software-Lizenz

Der Inhalt dieses Handbuchs bezieht sich auf den Firebird ODBC/JDBC-Treiber, der ursprünglich von James Starkey zum Firebird Project beigetragen wurde und seither von Vladimir Tsvigun, Alexander Potapchenko und anderen unter der Initial Developer's Public License V.1.0. weiterentwickelt.

Anhang B: Dokumenthistorie

Versionsgeschichte

0.2-de	05. Oktober 2018	MK	Übersetzung ins Deutsche
0.2	27. November 2017	H.E.M.B.	Fehlende Infos über Multi-Threading am Beginn von Kapitel 5 aufgenommen
0.1	25. November 2017	H.E.M.B.	Alte .chm-Hilfedatei aufgeräumt und ins Firebird Dokumentationsformat gewandelt.