Das XCOPY-Deployment-Paradigma außerhalb der .NET-Welt

Feuervogel

Die Verteilung (Deployment) einer datenbankbasierten Anwendung bedingt in der Regel eine Installation des Datenbankmanagementsystems (DBMS) und der eigenen Anwendungsdateien. Man kann sich glücklich schätzen, wenn man dazu eine in sich abgeschlossene Installationsroutine besitzt, die beide Komponenten automatisch installiert und einrichtet. Dass dies nicht immer so ist, kennen Sie sicherlich aus der Praxis. Sollten Sie Firebird einsetzen, dann gibt es eine interessante Möglichkeit für die Distribution der eigenen Anwendung, nämlich den Firebird Embedded Server.

von Thomas Steinmaurer

In der Ausgabe 1.2006 des *Entwickler Magazins* [1] setzten wir uns mit den wichtigsten Neuerungen in Firebird 2.0 auseinander. Der Firebird Embedded Server als eigenständige Architektur des Firebird DBMS ist keine Neuerung in 2.0, sondern existiert bereits seit dem ersten 1.5-Release (Februar 2004). Diese Architektur wurde bereits kurz in meinem Artikel in [2] erwähnt, ohne dabei jedoch ins Detail zu gehen. Ein vollständiges RDBMS mit einfachem Deployment muss kein Widerspruch sein, darum steht dieser Artikel auch unter dem Motto: „Warum kompliziert, wenn es auch einfach geht?“ Bleiben Sie dran!

Architekturen braucht das Land

Seit Firebird 1.5 gibt es den Firebird Server in unterschiedlichen „internen“ Architekturen. Mehrere Architekturen bedeuten natürlich auch, dass man die Qual der Wahl hat. Tabelle 1 gibt Ihnen einen Überblick über die vorhandenen Architekturen, deren Eigenschaften und den unterstützten Plattformen.

Quellcode

Der Quellcode zum Artikel befindet sich auf der Profi-CD und auf www.entwickler-magazin.de.

Auf die Super-Server- und Classic-Server-Architektur möchte ich hier nicht näher eingehen. Man sollte jedoch ein Grundverständnis dafür haben, welche Vor- und Nachteile eine bestimmte Architektur aufweist, da dies den Einsatz von Firebird, abhängig von den Gegebenheiten bezüglich Hardware und Ihren Anforderungen, beeinflussen wird.

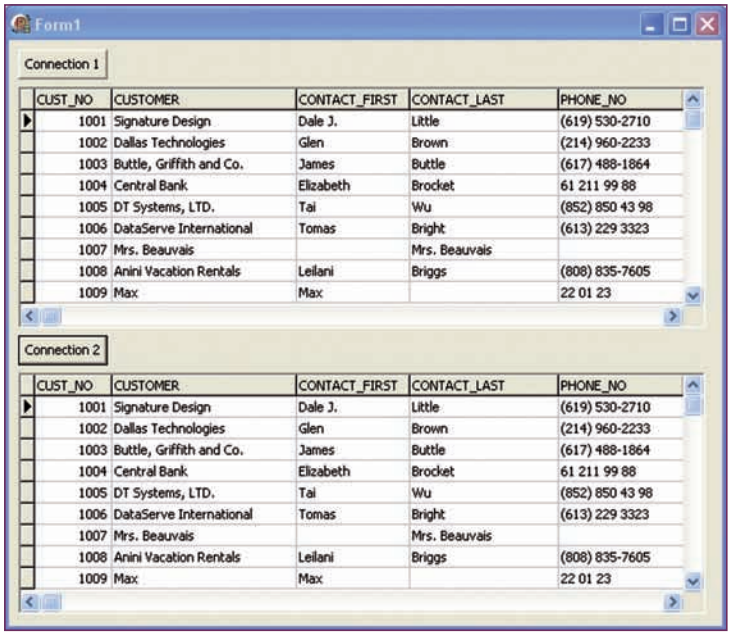
Eigenschaften des Embedded Server

Embedded ist nicht gleich Embedded. Beim Embedded Server von Firebird handelt es sich nicht um Komponenten für Delphi oder C++Builder, die es einem erlauben, die Datenbank-Engine direkt in die eigene ausführbare Datei mit hineinzukompilieren. Möchten Sie diesen Weg gehen, dann sollten Sie einen Blick auf andere Produkte, wie zum Beispiel NexusDB [3] werfen. Der Firebird Embedded Server ist ein vollfunktionaler Firebird Server in Form einer Windows-DLL mit Unterstützung für Transaktionen, Stored Procedures, Trigger, Views, Unterabfragen, Events, User-Defined-Funktionen etc. in der Super-Server-Architektur. Somit gelten für den Embedded Server die gleichen Eigenschaften bzw. Spielregeln (Tabelle 1) wie für den Super Server, jedoch mit den folgenden Einschränkungen:

- Nur unter Windows verfügbar
- Einträge in der Registry für Firebird werden ignoriert

Architektur	Eigenschaften	Plattformen
Super Server	Ein Prozess mit mehreren Threads zur Bearbeitung von Clientabfragen Ein gemeinsamer Cache für alle Verbindungen Ressourcenschonender Keine SMP-Unterstützung Skaliert bei hoher Last schlechter Services-API vollständig unterstützt	Windows, Linux, Solaris (x86)
Classic Server	Ein eigenständiger Prozess pro Client-Verbindung Jede Verbindung besitzt einen eigenen Cache Ressourcenintensiver SMP-Unterstützung Skaliert bei hoher Last besser Services-API vollständig unterstützt	Windows, Linux, Solaris (x86, Sparc), HPUX 11, Mac OS X, FreeBSD
Embedded Server	Entspricht Super Server	Windows

Tabelle 1: Firebird-Server-Architekturen



CUST_NO	CUSTOMER	CONTACT_FIRST	CONTACT_LAST	PHONE_NO
1001	Signature Design	Dale J.	Little	(619) 530-2710
1002	Dallas Technologies	Glen	Brown	(214) 960-2233
1003	Buttle, Griffith and Co.	James	Buttle	(617) 488-1864
1004	Central Bank	Elizabeth	Brocket	61 211 99 88
1005	DT Systems, LTD.	Tai	Wu	(852) 850 43 98
1006	DataServe International	Tomas	Bright	(613) 229 3323
1007	Mrs. Beauvais		Mrs. Beauvais	
1008	Anini Vacation Rentals	Leilani	Briggs	(808) 835-7605
1009	Max	Max		22 01 23

Abb. 1: Embedded Server: einfache Beispielanwendung

- Nur ein rein lokaler Zugriff ist möglich, auch nicht über „localhost“, jedoch als Clientbibliothek für Remote-Zugriffe verwendbar
- Exklusive Sperre der Datenbankdatei
- Firebird-Benutzerdatenbank *security.fdb* wird ignoriert, aber
- SQL-Privilegien werden geprüft

Die exklusive Sperre der Datenbankdatei ist notwendig, da sonst eine Beschädigung der Datenbankdatei möglich ist, wenn ein anderer Firebird-Server-Prozess ebenfalls auf diese Datenbankdatei zugreift. Sehen wir uns dazu ein einfaches Beispiel an. Abbildung 1 zeigt eine gestartete erste Instanz einer einfachen Delphi-2006-Anwendung mit zwei Datengrids, die über eine separate Verbindungskomponente den Inhalt der Tabelle „Customer“ der Beispieldatenbank *employee.fdb* anzeigen. Diese Anwendungsinstanz hat somit zwei Datenbankverbindungen offen, was auch einwandfrei funktioniert, da der Embedded Server im Kontext der eigenen Anwendung in Form einer DLL geladen wurde. Starten Sie nun eine zweite Instanz unserer Beispielanwendung. Der Verbindungsversuch zur selben Datenbank ist nun mit einer Fehlermeldung fehlgeschlagen, da unsere zweite Anwendungsinstanz als zweiter Prozess mit dem Embedded Server angesehen wird, und dieser Prozess durch die exklusive Sperre der Datenbankdatei die Datenbank nicht öffnen kann (Abb. 2).

Anwendungsszenarien

Die Anwendungsszenarien für den Einsatz des Embedded Server sind breit gefächert:

Datei	Anmerkung
Projekt1.exe	Ausführbare Datei der Anwendung
employee.fdb	Datenbankdatei
firebird.conf	Firebird-Konfigurationsdatei
firebird.msg	Datei mit den Firebird-Meldungstexten
gds32.dll	Embedded Server DLL. Nach *gds32.dll* umbenannte *fbembed.dll*, da die verwendeten Zugriffskomponenten *gds32.dll* als Clientbibliothek erwarten
ib_util.dll	Wird in der Regel von UDFs benötigt
\intl	Wird benötigt, wenn die Datenbank Zeichensätze (ISO8859_1, ...) verwendet
\udf	Standard-UDF-Bibliotheken

Tabelle 2: Dateiliste für Minimalinstallation

- Firebird-basierte Anwendung auf einem USB-Stick
- Demo-CDs
- Nachschlagewerke (zum Beispiel Produktkataloge, Namensverzeichnisse, ...) auf einem Nur-lesen-Datenträger
- Einzelplatzanwendungen jeglicher Art

Ein Problem, mit dem Anwendungsentwickler immer mehr konfrontiert werden, sind Benutzer mit eingeschränkten Rechten, wie zum Beispiel Außendienstmitarbeiter, die auf Ihrem Laptop keine Administratorenrechte besitzen, was die Installation einer Firebird-gestützten Anwendung erschweren würde. Mit dem Embedded Server müssen nur ein paar Dateien kopiert werden und fertig ist die Installation. So habe ich ein selbstgeschriebenes, einfaches CRM-System basierend auf einer Firebird-Datenbank auf meinem USB-Stick immer bei mir, um jederzeit Kundendaten abzurufen und Produktlizenzen erstellen zu können. Wie das Deployment einer Anwendung mit dem Firebird Embedded Server aussieht, werden wir uns im nächsten Abschnitt näher ansehen.

Deployment einer Anwendung

Bei der Embedded-Server-Distribution handelt es sich um eine ZIP-Datei, die von der Firebird-Projekt-Webseite [4] heruntergeladen werden kann. Das Firebird Embedded Server Release 1.5.3 finden Sie auf der Heft-CD, sodass Sie sofort loslegen können. Sofern Ihre Anwendung keine Einträge in der Registry oder andere Aktionen benötigt, die während einer Setup-Routine durchgeführt werden müssen, reicht ein einfaches Kopieren all Ihrer Anwendungsdateien (Datenbankdatei, ausführbare Datei(en), Laufzeitbibliotheken, usw. ...) mit den Dateien aus der Embedded-Server-Distribution. Gehen Sie dazu wie folgt vor:

- Extrahieren Sie das ZIP-Archiv der Embedded-Server-Distribution in ein Verzeichnis Ihrer Wahl
- Kopieren Sie alle Ihre Anwendungsdateien inklusive Datenbankdatei in dieses Verzeichnis
- Benennen Sie die Datei *fbembed.dll* so um, dass diese Datei den Namen der

Firebird-Clientbibliothek aufweist, die Ihre Anwendung erwartet (zum Beispiel *gds32.dll* oder *fbclient.dll*, je nachdem, welche Zugriffskomponenten verwendet werden)

Das ist alles! Sie haben nun eine lauffähige Firebird-basierte Anwendung, die den Embedded Server als Datenbank-Engine verwendet. Ich denke, Sie können mir zustimmen, dass dieses Vorgehen sehr stark mit dem XCOPY-Deployment-Paradigma aus der .NET-Welt vergleichbar ist. Einfach die benötigten Dateien kopieren und fertig. Zum Thema „benötigte Dateien" ist noch zu sagen, dass Sie keinerlei Dokumentationsdateien aus der Embedded-Server-Distribution benötigen, das heißt diese Dateien können Sie getrost außen vor lassen, um Speicherplatz zu sparen. Dies ist vor allem dann interessant, wenn es sich beim Zielmedium zum Beispiel um einen USB-Stick handelt, auf dem nur wenig Speicherplatz vorhanden ist. Tabelle 2 gibt einen Überblick über die benötigten Dateien für eine von mir empfohlene Minimalinstallation unserer einfachen Beispielanwendung.

Die Gesamtgröße der Minimalinstallation beträgt ca. 4,2 MB, wobei die Datenbankdatei alleine eine Größe von etwa einem MB aufweist. Es ginge auch noch kleiner, da man das *\udf*-Unterverzeichnis und die Datei *ib_util.dll* weglassen kann, wenn keine UDFs eingesetzt werden. Meine Empfehlung ist allerdings: Nehmen Sie ruhig die Datei *ib_util.dll* sowie die Verzeichnisse *\intl* und *\udf* in Ihren Deployment-Prozess auf, da man so sichergehen kann, dass diese Bestandteile für eine spätere Verwendung vorhanden sind, auch wenn sie im Moment nicht benötigt werden.

Datenbank auf Nur-lesen-Datenträger

Dass für eine Einzelplatzanwendung keine Installation notwendig ist, macht den Embedded Server äußerst attraktiv für Nachschlagedatenbanken auf einem Nur-lesen-Datenträger wie einer CD-ROM, oder wenn es etwas größer sein darf auch auf einer Daten-DVD. Aus Sicht der Firebird-Engine werden Nur-lesen-Datenbanken unterstützt, sofern natürlich die Clientanwendung keine Schreiboperationen ausführt. Es reicht aber nicht aus, die Datenbank mit einem Schreibschutz zu versehen und danach auf eine CD-ROM/DVD zu brennen, sondern es muss auch ein spezielles Read-only-Flag auf Datenbankebene gesetzt sein. Wird nur der Schreibschutz auf Dateiebene im Windows Explorer verwendet, dann erhalten Sie beim Ausführen unserer Beispielanwendung die Fehlermeldung aus Abbildung 3, obwohl wir nur eine Datenbankverbindung öffnen und eine Tabelle abfragen, ohne dabei datenmanipulierende Operationen auszuführen. Der Grund hierfür ist, dass beim Ausführen der Abfrage implizit eine Transaktion gestartet wird und somit intern die Transaction Inventory Page (TIP) aktualisiert wird. Dies schlägt fehl, da die Datenbankdatei mit einem Schreibschutz versehen ist. Wenn man aber das Read-only-Attribut für die Datenbank, zum Beispiel mit dem Kommandozeilentool *gfix* setzt, dann klappt auch der Nur-lesen-Zugriff auf unsere Firebird-Datenbank. Die Verwendung von *gfix* sieht in diesem Fall wie folgt aus:

```
gfix employee.fdb -user sysdba -password masterkey -mode
read_only
```

Ein zusätzlicher Trick bei Nur-lesen-Datenbanken besteht darin, keinen Speicherplatz auf den Datenseiten für alte Datensatzversionen zu reservieren, da ja keine Datenänderungen möglich sind. Dies ist vor allem dann interessant, wenn der verfügbare Speicherplatz auf dem Zielmedium einen kritischen Faktor darstellt. Man kann hier bis zu 20 Prozent der Datenbankdateigröße einsparen. Diese Eigenschaft kann ebenfalls mit *gfix* gesetzt werden. Wichtig: Der folgende Aufruf von *gfix* muss natürlich vor dem Setzen des Read-only-Flags stattfinden!

```
gfix employee.fdb -user sysdba -password masterkey -use full
```

Jetzt muss nur noch ein Backup/Restore (zum Beispiel mit *gbak*) durchgeführt werden, um die Datenbank ohne reserviertem Speicherplatz für alte Datensatzversionen neu zu erstellen.

Abb. 2: Exlusive Sperre der Datenbankdatei: Fehlermeldung

Abb. 3: Nur-lesen-Datenbank: Fehlermeldung

Wollen Sie die Nur-lesen-Datenbank zu einem späteren Zeitpunkt nicht nur von der CD-ROM/DVD betreiben, sondern auch auf Ihrem Server, um Datenänderungen zu ermöglichen, dann gehen Sie wie folgt vor:

- Datenbankdatei auf die Server-Festplatte kopieren
- Schreibschutz der Datenbankdatei mit dem Windows Explorer entfernen
- *gfix* mit dem Kommandozeilenschalter *–mode read_write* ausführen

Für den Fall, dass Sie den Trick mit der vollen Befüllung der Datenseiten angewendet haben, sollten Sie mit *–use reserve* wieder Speicherplatz für alte Datensatzversionen reservieren. In diesem Fall ist wiederum ein Backup/Restore-Zyklus notwendig, damit die Datenbank neu aufgebaut wird.

Ich hoffe, dass ich Ihr Interesse an dieser attraktiven Form des Deployments einer Firebird-basierten Anwendung geweckt habe. Vielleicht erweist sich diese Architektur des Firebird Servers ja bereits in einem Ihrer nächsten Projekte als nützlich. Treu unserem Motto: „Warum kompliziert, wenn es auch einfach geht?".

Links & Literatur

[1] Thomas Steinmaurer: Firebird 2.0, in *Entwickler Magazin* 1.2006

[2] Thomas Steinmaurer: Firebird 1.5 – Das neue Release, in *Der Entwickler* 2.2005

[3] www.nexusdb.com

[4] www.firebirdsql.org