Feuervogel

Firebird 1.5 – Das neue Release

Die Entwickler der Open-Source-Datenbank Firebird haben nach dem Erscheinen der Version 1.0 nun mit der Version 1.5 eine stark überarbeitete Version ihres Datenbank-Management-Systems veröffentlicht. In diesem Artikel werden Ihnen die neuen Features vorgestellt.

von Thomas Steinmaurer

Da Firebird auf den Quellcodes von InterBase 6 basiert, der im Jahre 2000 von Borland freigegeben wurde. Diese Tatsache ist gerade für den unternehmenskritischen Einsatz wichtig, da man kein vollkommen neues Produkt in den Händen hält, sondern ein Stück bewährter Datenbanktechnologie. Das Firebird-Projekt [1] wurde gerade mal eine Woche nach der Veröffentlichung der InterBase 6 Quellcodes von einer „Rebellengruppe“ bei SourceForge .net [2] ins Leben gerufen. Im März 2002 wurde Firebird 1.0 der Öffentlichkeit vorgestellt. Dieses Release hatte eine Unmenge von Fehlern in den InterBase 6 Quellcodes beseitigt und konnte auch mit ein paar kleineren Erweiterungen aufwarten.

Quellcode

Den Quellcode finden Sie auf der aktuellen Profi-CD sowie unter www.derentwickler.de

Gleichzeitig mit dem Release von Version 1.0 wurde auch bereits mit der Arbeit an 1.5 begonnen, das damals als einziges Ziel die Migration des Quellcodes von C nach C++ zum Ziel hatte. Man versprach sich davon, den Quellcode über ein entsprechendes objektorientiertes Design leichter wartbar, verständlicher und somit auch reizvoller für neue Entwickler zu machen. Dieser fromme Wunsch hielt nicht lange. Schon bald zeichnete sich ab, dass die Version 1.5 viel mehr als eine reine Codemigration sein werden würde. Einerseits gab es die ständige Nachfrage nach neuen Funktionalitäten in den Newsgroups, andererseits war die Konkurrenz und vielleicht auch der Wettbewerbsgedanke zu *Yaffil*, einem russischen Entwicklungszweig basierend auf Firebird 1.0, dafür ausschlaggebend. Durch die aktive Weiterentwicklung ist das Firebird-Projekt im Februar 2004 an dem wichtigen Meilenstein – Release 1.5 – angelangt, das eine Fülle an Verbesserungen und Erweiterungen bietet. Aber nicht nur an der Datenbank-Engine selbst, sondern auch an unterschiedlichen Zugriffsschnittstellen wurde fleißig gearbeitet. So gibt es mittlerweile einen stabilen JDBC-Type 4 (und 2) und ODBC-Treiber sowie einen .NET Provider, die alle kostenlos unter einer Open-Source-Lizenz verfügbar sind und direkt vom Firebird-Projekt stammen. Was nun Firebird 1.5 an Neuerungen bietet, werde ich im weiteren Verlauf des Artikels vorstellen.

Verbesserter Optimizer

Ein Bereich, der nie gut genug für jeden funktionieren wird, ist der Optimizer, der dafür verantwortlich ist, den schnellstmöglichen Zugriffsweg für eine Anfrage zu ermitteln. Oftmals verhielt sich dieser in InterBase 6 und auch Firebird 1.0 etwas störrisch. Da kam es schon mal vor, dass ein Index für eine Abfrage nicht verwendet wurde, wo dies offensichtlich Sinn gemacht hätte. Wenn man den Firebird 1.5 Release Notes – die auch auf Deutsch erhältlich sind [3] – glauben darf, dann kann man sich in bestimmten Situationen eine Performancesteigerung von 30 bis 60 Prozent erwarten, wenn es sich um eine komplexe Abfrage handelt. Bei der Interpretation solcher Zahlen soltle man allerdings etwas vorsichtig sein, aber unterschiedli-

chen Meldungen in den Newsgroups zufolge dürfte darin schon ein kleines Stückchen Wahrheit stecken. Allein der Umstieg von Version 1.0 auf 1.5, ohne dabei etwas an der Anwendung zu ändern, reichte aus, um bei komplexen Abfragen eine spürbar bessere Performance zu bekommen. Der Optimizer ist nun cleverer und macht einen viel besseren Job als in früheren Versionen.

Neue Architekturen unter Windows

Mit Firebird 1.5 stehen nun unter Windows eine wiederbelebte und eine neue Architektur zur Verfügung. Diese sind:

- Classic Server
- Embedded Server

Den alten InterBase-Hasen oder den Leuten, die InterBase/Firebird unter Linux einsetzen, wird der Classic Server sicher ein Begriff sein. Es handelt sich hierbei um die Vorgängerversion der „Ein-Prozess-Multi-Threaded" SuperServer-Architektur. Unter der Classic-Architektur wird für jede Anfrage ein neuer Prozess gestartet, wobei jede Datenbankverbindung einen eigenen Datenbank-Cache verwaltet. Der Nachteil dieser Architektur besteht darin, dass wesentlich mehr Ressourcen (Hauptspeicher) im Vergleich zu SuperServer benötigt werden. Der große Vorteil hingegen ist, dass Mehrprozessorsysteme automatisch unterstützt werden, da es dem Betriebssystem obliegt, wie die Prozesse auf die unterschiedlichen Prozessoren aufgeteilt werden.

Für eine genauere Betrachtung der Unterschiede dieser beiden Architekturen möchte ich auf [4] verweisen. Da SuperServer auch in 1.5 nicht SMP-fähig ist, bleibt die Classic-Architektur im Moment die einzige Möglichkeit, wenn man die Leistung mehrerer Prozessoren voll ausschöpfen möchte.

Eine äußerst interessante Sache ist der Embedded Server. Hierbei handelt es sich um einen Client inklusive vollfunktionalem Server in der SuperServer-Architektur, gekapselt in einer DLL. Der Embedded Server implementiert das vollständige Firebird API und besitzt, wie der reguläre Server, eine vollständige Transaktionsunterstützung, Stored Procedures, Trigger und vieles mehr. Dies eröffnet neue Wege bei der Distribution einer Einzelplatzanwendung, die auf Firebird basiert, da der Embedded Server nicht über eine Setup-Routine installiert werden muss, sondern es ausreicht, bestimmte Firebird-spezifische Dateien mit der eigenen Anwendung weiterzugeben. Dies eignet sich auch ausgezeichnet, um datenbankgestützte Anwendungen auf CD-ROM oder einem Wechselmedium wie USB-Festplatte oder USB-Stick zu betreiben. Beim Verbindungsaufbau über den Embedded Server wird die Datenbankdatei mit einer exklusiven Sperre versehen. Somit kann eine geöffnete Datenbankdatei über einen regulären Firebird-Server oder einem weiteren Embedded Server nicht angesprochen werden. Dies ist notwendig, damit sich unterschiedliche Firebird-Serverprozesse nicht in die Quere kommen und die Datenbank beschädigen. Der Embedded Server erlaubt nur einen lokalen Datenbankzugriff. Eine Datenbankverbindung über „localhost" (TCP/IP Loopback) ist nicht möglich. Die DLL kann jedoch als Client für einen Datenbankzugriff auf einem entfernten Rechner dienen. Eine weitere Einschränkung ist, dass die Benutzerdatenbank für den Verbindungsaufbau keine Rolle spielt. Eine beliebige Benutzer-/Passwort-Kombination reicht aus, um sich zur Datenbank zu verbinden. Es werden aber sehr wohl SQL-Privilegien, die mit der GRANT-Anweisung definiert wurden, beim Datenzugriff überprüft.

SQL-Privilegien werden überprüft

Spracherweiterungen

Einige wichtige und zugleich mächtige Erweiterungen, die Datenbank- und Anwendungsentwicklern das Leben erleichtern, sind Spracherweiterungen in den Bereichen Datentypen, Metadaten, DSQL und PSQL. In Firebird 1.0 und InterBase 6 (und höher) konnte man in einer Dialekt-3-Datenbank mit NUMERIC (18,0) einen 64-bit Integer definieren. In Firebird 1.5 ist dies nun über den neuen Datentyp BIGINT möglich. Im Bereich der Metadaten können nun auch Indizes, die für die Realisierung benutzerdefinierter Constraints automatisch angelegt werden, mit einem sprechenden Namen versehen werden. Die Syntax sieht wie folgt aus:

```
...
[ADD] CONSTRAINT [<constraint-identifier>]
<constraint-type> <constraint-definition>
[USING [ASC[ENDING] | DESC[ENDING]] INDEX
                                    <index_name>]
```

Ein konkretes Beispiel, wie man den zugrundeliegenden Index für einen PRIMARY KEY Constraint *PK_KUNDE* benennen (*IDX_PK_KUNDE*) und automatisch anlegen lassen kann, ist in Listing 1 abgebildet.

Eine weitere interessante Neuerung sind die so genannten Multi-Aktionen-Trigger. Hiermit ist es zum ersten Mal möglich, einen Trigger für multiple Operationstypen zu erstellen, der dann zum Beispiel sowohl bei einer Einfüge- als auch Änderungsoperation feuert. Die erweiterte Syntax für die *CREATE-/ALTER TRIGGER*-Anweisung sieht dann wie folgt aus:

```
CREATE TRIGGER name FOR table
 [ACTIVE | INACTIVE]
 {BEFORE | AFTER} <multiple_action>
 [POSITION number]
AS trigger_body

<multiple_action> ::= <single_action> [OR <single_action>
                                    [OR <single_action>]]
<single_action> ::= {INSERT | UPDATE | DELETE}
```

Angenommen, ein Feld VOLLNAME, das sich aus NACHNAME und VORNAME zusammensetzt, soll serverseitig gewartet werden, so musste man mit Firebird 1.0 zwei Trigger implementieren, damit die entsprechende Logik sowohl bei einer Einfüge- als auch einer Änderungsoperation ausgeführt wurde. In Firebird 1.5 kann dies nun mit einem Trigger realisiert werden (Listing 2).

Zusätzliche DDL-Anweisungen, wie *RECREATE VIEW, CREATE OR ALTER (TRIGGER | PRCOEDURE)* und das Erlauben von NULL-Zuständen in UNIQUE-

Constraints und Indizes, runden das Angebot der Erweiterungen im Bereich der Metadaten positiv ab.

Auch im DSQL-Bereich gibt es einige nennenswerte Neuerungen. Wo früher noch eine Selectable Stored Procedure notwendig war, um abhängig von Feldwerten bzw. -zuständen die Ergebnismenge einer Abfrage zu beeinflussen, kann dies nun mit den eingebauten Funktionen *CASE, COALESCE* und *NULLIF* direkt in einer *SELECT*-Anweisung durchgeführt werden. Des Weiteren bekam Firebird 1.5 eine SQL-99-konforme Implementierung so genannter Savepoints spendiert. Hierbei handelt es sich um benannte „Lesezeichen", die innerhalb einer laufenden Transaktion mit *SAVEPOINT <bezeichner>* gesetzt werden können, um im Kontext dieser Transaktion mit *ROLLBACK TO SAVEPOINT <bezeichner>* zu diesem Lesezeichen zurückzukehren, und alle Änderungen ab dieser Marke zu verwerfen, ohne jedoch die komplette Transaktion beenden zu müssen. Sehen wir uns diese Neuerung anhand eines einfachen Beispiels (Listing 3) etwas näher an.

Zu Beginn wird eine Tabelle *SAVEPOINTTEST* erstellt. Danach wird ein Datensatz eingefügt und über *COMMIT* bestätigt. Anschließend wird ein zweiter Datensatz eingefügt, aber die Transaktion wird noch nicht beendet, d.h., die Transaktion ist weiterhin aktiv. Die darauf folgende *SELECT*-Anweisung gibt zwei Datensätze zurück. Über die nächste Anweisung wird ein Savepoint mit dem Namen X gesetzt. Jetzt werden alle Datensätze aus der Tabelle *SAVEPOINTTEST* gelöscht. Die Transaktion ist noch immer aktiv. Die Abfrage im Anschluss gibt erwartungsgemäß keine Datensätze zurück. Nun werden über die *ROLLBACK TO X*-Anweisung alle Änderungen, die ab dem Setzen der Marke X stattgefunden haben, zurückgesetzt. Die nächste Abfrage gibt nun wieder zwei Datensätze zurück. Mit dem *ROLLBACK* wird nun die komplette Transaktion zurückgesetzt, und die darauf folgende Abfrage gibt einen Datensatz zurück, da ja die erste Einfügeoperation mit *COMMIT* bestätigt wurde. Leider ist es nicht möglich, so wie in InterBase 7.x, Savepoints auch in PSQL zu verwenden.

Eine weitere interessante Neuerung sind pessimistische Sperren. Firebird verfolgt im Mehrbenutzerbetrieb, für die Sicherstellung einer konsistenten Sicht auf Daten, den Ansatz der optimistischen Datensatzsperren. Es wird davon ausgegangen, dass die Wahrscheinlichkeit, dass ein und

Listing 1

```
CREATE TABLE KUNDE (
 ID BIGINT NOT NULL,
 VORNAME VARCHAR(20),
 NACHNAME VARCHAR(40),
 VOLLNAME VARCHAR(62)
);
COMMIT;
ALTER TABLE KUNDE ADD CONSTRAINT PK_KUNDE
   PRIMARY KEY (ID) USING ASC INDEX IDX_PK_KUNDE;
COMMIT;
```

derselbe Datensatz von mehreren Benutzern gleichzeitig verändert wird, sehr niedrig ist. Erst beim Zurückschreiben einer Änderung wird überprüft, ob es sich um einen konkurrierenden Schreibzugriff handelt oder nicht. In manchen Situationen macht es allerdings Sinn, dass bereits beim Lesen eines Datensatzes dieser exklusiv gesperrt wird. Ein Anwendungsfall ist die Vergabe einer fortlaufenden Rechnungsnummer, ohne dass hierbei Lücken entstehen. Ein Generator ist hierfür nicht geeignet, da ein Rollback nach dem Holen des nächsten Generatorwertes diesen nicht wieder zurücksetzt. Mit dem pessimistischen Sperrverfahren könnte man sich die nächste Rechnungsnummer aus einer Nummerntabelle holen und den Datensatz mit der nächsten Nummer bereits beim Lesen mit einer Sperre versehen. Die Rechnung wird erzeugt und der Wert in der Nummerntabelle wird um eins erhöht. Anschließend wird die Transaktion beendet, und somit auch die Datensatzsperre in der Nummerntabelle entfernt. Die Syntax ist relativ einfach, man muss bloß die *WITH LOCK*-Klausel in einer *SELECT*-Anweisung verwenden:

```
SELECT ... FROM <sometable>
[WHERE ...]
[FOR UPDATE [OF ...]]
[WITH LOCK]
...;
```

Aber Vorsicht! Man sollte das pessimistische Sperrverfahren nur auf sehr kleinen Ergebnismengen anwenden (im Optimalfall nur auf einen Datensatz), da man sich hiermit den Nutzen der MGA (Multi-Generationen-Architektur), einer Stärke von Firebird, einfach zunichte machen kann. Des Weiteren wurde die Behandlung von Aggregaten verbessert und die Einordnung von NULL-Zuständen in einer Ergebnismenge kann nun über die *NULLS* (*FIRST*|*LAST*)-Klausel im *ORDER BY*-Abschnitt einer Abfrage benutzerdefiniert erfolgen.

Im Bereich der prozeduralen Sprache (PSQL) – für Stored Procedures und Trigger – hat sich ebenfalls einiges getan. Eine der mächtigsten Erweiterungen ist die *EXECUTE STATEMENT*-Anweisung, die es erlaubt, eine Zeichenkette dynamisch auszuführen. Die auszuführende Anweisung muss somit nicht mehr bereits zur Übersetzungszeit der Stored Procedure oder des Triggers feststehen. *EXECUTE STATEMENT* kann wie folgt verwendet werden:

- *EXECUTE STATEMENT <anweisung>;*
- *EXECUTE STATEMENT <anweisung> INTO <variablenliste>;*
- *FOR EXECUTE STATEMENT <anweisung> INTO <variablenliste> DO ...;*

Listing 4 zeigt eine einfache Stored Procedure, der ein Generatorname mit dem Eingabeparameter *AGENERATORNAME* übergeben wird, und den nächsten Wert des Generators über den Rückgabeparameter *RID* zurückgibt.

Mit *FOR EXECUTE STATEMENT* kann man sogar die Ergebnismenge einer dynamisch zusammengesetzten Abfrage durchlaufen. Sogar DDL-Anweisungen können über *EXECUTE STATEMENT* ausgeführt werden. Wenn möglich, sollte dies allerdings vermieden werden, da eine DDL-Anweisung in der Regel über ein COMMIT bestätigt werden muss, bevor man zum Beispiel eine neu erstellte Tabelle mit Daten befüllt. Da in einer Stored Procedure und einem Trigger jedoch keine Transaktionskontrolle zur Verfügung steht, sollte man DDL-Anweisungen über *EXECUTE STATEMENT* nur in Ausnahmefällen verwenden. In manchen Situationen ist es allerdings ganz praktisch, so zum Beispiel, wenn man über eine Stored Procedure die Selektivität aller Indizes aktualisieren möchte.

In PSQL stehen nun auch neue Kontextvariablen zur Verfügung: *CURRENT_CONNECTION*, *CURRENT_TRANSACTION* und *ROW_COUNT*. Mit *ROW_COUNT* kann man die Anzahl der Datensätze, die nach einer datenmanipulierenden Operation betroffen waren, abfragen. Für Multi-Aktionen-Trigger ganz hilfreich sind die booleschen Kontextvariablen *DELETING*, *INSERTING* und *UPDATING*. Hiermit kann in einem Multi-Aktionen-Trigger abgefragt werden, welcher Operationstyp den Trigger angestoßen hatte. Ein verbessertes Exception-Handling, eine erweiterte Variablendeklaration (z.B.: Setzen eines Initialwertes bereits bei der Deklaration) und die *LEAVE/BREAK*-Anweisung zum Verlassen einer Schleife, ergänzen das Angebot der PSQL-Erweiterungen.

Listing 2

```
SET TERM ^^ ;
CREATE TRIGGER TRI_KUNDE_VOLLNAME FOR KUNDE
        ACTIVE BEFORE INSERT OR UPDATE POSITION 0 AS
BEGIN
 IF (NEW.VORNAME <> '') THEN
  NEW.VOLLNAME = NEW.NACHNAME || ', '
                                  || NEW.VORNAME;
 ELSE
  NEW.VOLLNAME = NEW.NACHNAME;
END
^^
SET TERM ; ^^
```

Listing 3

```
CREATE TABLE SAVEPOINTTEST (ID INTEGER NOT NULL);
COMMIT;
INSERT INTO SAVEPOINTTEST VALUES (1);
COMMIT;
INSERT INTO SAVEPOINTTEST VALUES (2);
SELECT * FROM SAVEPOINTTEST;
                        // Gibt zwei Datensätze zurück
SAVEPOINT X;
DELETE FROM SAVEPOINTTEST;
SELECT * FROM SAVEPOINTTEST;
                        // Gibt keine Datensätze zurück
ROLLBACK TO X;
SELECT * FROM SAVEPOINTTEST;
                        // Gibt zwei Datensätze zurück
ROLLBACK;
SELECT * FROM SAVEPOINTTEST;
                        // Gibt einen Datensatz zurück
```

Listing 4

```
SET TERM ^^ ;
CREATE PROCEDURE PRO_GETNEXT_GENID (
 AGENERATORNAME VarChar(31))
 RETURNS (
 RID NUMERIC(18,0))
AS
BEGIN
 EXECUTE STATEMENT 'SELECT GEN_ID
                        ('' || AGENERATORNAME || '', 1)
                    FROM RDB$DATABASE' INTO :RID;
 SUSPEND;
END
^^
SET TERM ; ^^
```

Neues Konfigurationsmanagement

Viele Dateinamen in Firebird 1.5 wurden einer Generalüberholung unterzogen. So wurde der „ib“-Teil vieler Dateinamen mit „fb“ ersetzt. Zum Beispiel: *fbguard.exe* und *fbserver.exe* anstatt von *ibguard.exe* bzw. *ibserver.exe*. Neu hinzugekommen ist auch ein erweitertes Konfigurationsmanagement in Form der Datei *firebird.conf*, die sich im Wurzelverzeichnis der Firebird-Installation befindet. Darin befindet sich eine Fülle an Konfigurationsparametern, mit denen der Firebird-Server konfiguriert werden kann. Trotzdem bleibt Firebird ein einfach zu administrierendes DBMS, da in der Regel diese Parameter nicht angerührt werden müssen. Sollte jedoch ein Parameter geändert werden müssen, dann darf man nicht vergessen, dass die Änderungen erst nach einem Neustart des Firebird-Servers in Kraft treten. Die unterschiedlichen Einstellungsmöglichkeiten sind gut dokumentiert. Ich möchte in Tabelle 1 nur die wichtigsten herausgreifen.

Man sieht bereits durch die hier angeführten Konfigurationsparameter, dass eine wichtige Anforderung bei der Implementierung des neuen Konfigurationsmanagements in der verbesserten Sicherheit einer Firebird-Installation lag. So ist zum Beispiel per Vorgabe der Zugriff auf externe Tabellen über den Parameter *ExternalFileAccess* von Haus aus unterbunden. Sollten Sie also externe Tabellen verwenden, dann müssen Sie zuerst das Verzeichnis, in dem die externen Tabellen liegen, über diesen Parameter dem Firebird-Server bekannt geben. Zwei interessante Einstellungsmöglichkeiten sind *RemoteServicePort* und *RemoteAuxPort*. Mit *RemoteServicePort* kann ein beliebiger, freier Port dem Firebird-Serverprozess zugewiesen werden. Somit ist es möglich, dass man Firebird 1.0 oder InterBase mit Firebird 1.5 parallel betreibt. Sogar mehrere Firebird 1.5 „Instanzen“ sind hiermit möglich, sofern man die manuelle Installation über das ZIP-Paket und nicht über die Setup-Routine wählt. Ebenfalls hilfreich ist der Parameter *RemoteAuxPort*, mit dem ein fixer Port für die Eventkommunikation definiert werden kann. Gerade in einer Umgebung mit einer Firewall ist das enorm hilfreich, da nur dieser eine Port freigeschaltet werden muss, damit die Eventkommunikation funktioniert. In Firebird 1.0 wurden dafür willkürliche (unvorhersagbare) Ports verwendet.

> Alias-Namen-/Wert-Paare werden definiert

Datenbankdatei-Aliasing

Aus Sicht des Clients ist es nun in Firebird 1.5 nicht mehr notwendig, dass dieser den absoluten Pfad einer Datenbankdatei am Server kennen muss. In der Textdatei *aliases.conf*, die sich ebenfalls im Wurzelverzeichnis der Firebird-Installation befindet, können nun Alias-Namen-/Wert-Paare definiert werden. Zum Beispiel:

```
tourism=c:\daten\tourism.fdb
```

Der Aliasname *tourism* „zeigt“ auf die Datenbankdatei *tourism.fdb*, die sich im Verzeichnis *c:\daten* auf dem Server befindet. Der Client braucht nun bloß den Ser-

Parameter	Bedeutung
DatabaseAccess	Einschränkung des Datenbankzugriffs auf Dateisystemebene
ExternalFileAccess	Einschränkung des Zugriffs auf externe Tabellen
UdfAccess	Einschränkung des Zugriffs auf UDF-Bibliotheken
RemoteServicePort	Portnummer, die der Firebird Server für eingehende Anfragen verwendet
RemoteAuxPort	Portnummer, die für die Eventkommunikation verwendet wird
TempDirectories	Angabe des Speicherortes von temporären Sortierdateien

Tabelle 1: Konfigurationsparameter in firebird.conf (Auszug)

ver- und den Alias-Namen kennen, um sich zur Datenbank zu verbinden. Dies geschieht mit der hier folgenden Notation:

```
server_name:alias_name
```

In unserem Beispiel reicht der folgende Remote-Verbindungsstring aus, wenn der Server den Rechnernamen *myserver* besitzt:

```
myserver:tourism
```

Sollte sich nun der Pfad zur Datenbankdatei ändern, dann kann dies transparent für den Client mit dem Alias-Eintrag am Server geändert werden.

Sonst noch etwas?

Gibt es sonst noch etwas Neues? Ja: Firebird 1.5 arbeitet problemlos mit Windows 2003 Server zusammen. Dies kann für Firebird 1.0 und InterBase kleiner 7.1 nicht gesagt werden, da hier lange Connect-/Disconnect-Zeiten unter Windows 2003 Server berichtet wurden. Dies hat übrigens nichts mit dem System-Restore-Problem aus Windows XP zu tun, da Windows 2003 Server dieses Feature gar nicht anbietet. Auch die ODS (On-Disc-Structure) wurde in Firebird 1.5 um eine Nebenversion auf 10.1 angehoben.

Dies war notwendig, da drei neue Indizes auf Systemtabellen hinzugekommen sind, und auch Multi-Aktionen-Trigger entsprechend in der Systemtabelle *RDB$TRIGGER* unterstützt werden mussten. Auch das alte Problem, dass VARCHAR-Felder in deren voller Länge über das Netz gehen, wurde in Firebird 1.5 beseitigt. Angenommen, Sie haben ein Feld vom Typ VARCHAR(255), und darin ist nur ein Zeichen abgespeichert.

In Firebird 1.5 wird nun nur mehr der tatsächlich belegte Platz, also ein Zeichen, vom Server zum Client gesendet. Für abfrageintensive Tabellen ist es interessant zu wissen, dass die maximale Index-anzahl pro Tabelle von 64 auf 256 erhöht wurde. Datenbankverbindungen werden nun generell viel schneller als in Firebird 1.0 aufgebaut und neue Zeichensätze/Sortierreihenfolgen fanden ebenfalls den Weg in das Release 1.5.

Kompatibilität zu den InterBase-Express-(IBX-)Komponenten

In Firebird 1.5 wurde die Client-Bibliothek von *gds32.dll* auf *fbclient.dll* umbenannt. Die gute Nachricht gleich vorweg: Das Firebird-Projektteam hat sich große Mühe gegeben, dass IBX-Anwendungen auch mit Firebird 1.5 lauffähig bleiben. Man muss sich nur darum kümmern, dass sich eine aktuelle Version der Client-Bibliothek *gds32.dll* im Anwendungsverzeichnis oder im Suchpfad befindet. Hier ist es wichtig, dass die DLL einen Versionsstring beginnend mit „6.3“ aufweist, da ansonsten die IBX Admin-Komponenten nicht lauffähig sind. Es reicht allerdings nicht aus, *fbclient.dll* auf *gds32.dll* umzubenennen, da *fbclient.dll* einen Versionsstring beginnend mit „1.5“ besitzt. Man kann dazu die entsprechende „Legacy“-Option in der Setup-Routine während der Firebird-Installation verwenden. Da hierbei allerdings eine existierende *gds32.dll* im Windows-Systemverzeichnis nicht überschrieben wird, kann man die Installation einer „korrekten“ *gds32.dll* Version auch im Nachhinein mit dem Kommandozeilentool *instclient.exe* erledigen. In den Release Notes finden Sie Beispiele, wie dieses Tool zu verwenden ist. Längerfristig sollte man allerdings auf Zugriffskomponenten setzen, die Firebird explizit auch in der Zukunft unterstützen werden.

FirebirdSQL-Stiftung unterstützt Projekte

Durch die Glaskugel geschaut

Firebird ist ein äußerst aktives Open-Source-Projekt. Nicht nur die Entwicklung an der Datenbank-Engine selbst, sondern auch an dem wichtigen „Rundherum“ eines DBMS wie JDBC/ODBC/.NET-Treiber, dem neuen Administrationswerkzeug FlameRobin (das sich noch im Alpha-Stadium befindet) und einer Dokumentation, wird aktiv gearbeitet. Über die im Jahre 2002 gegründete FirebirdSQL-Stiftung [5] – einer Non-Profit-Organisation – ist es möglich, das Firebird-Projekt sowohl als Einzelperson, als auch als Unternehmen in Form von bezahlten Mitgliedschaften oder Sponsoring finanziell zu unterstützen. Über diese Stiftung werden bereits einige Entwickler für deren Aufwand finanziell entschädigt.

Derzeit ist das zweite Wartungsrelease (1.5.2) von Firebird verfügbar. Die Roadmap für Firebird 2.0 und dem aktuell laufenden Projekt mit Codenamen *Vulcan* des InterBase-Erfinders Jim Starkey (he is back!) ist auf der Firebird-Website [6] veröffentlicht. Vulcan basiert auf einem sehr frühen Stand des Firebird 2.0 Quellcodes, mit dem Ziel, die Architekturen SuperServer, Classic und Embedded in einer Architektur zu vereinen, inklusive feingranularem Multi-Threading für Mehrprozessorumgebungen und Unterstützung von 64-bit Betriebssystemen. Firebird 3.0 ist als Kombination von Firebird 2.0 + Vulcan + Yaffil (einem bereits erwähnten russischen Entwicklungszweig basierend auf Firebird 1.0) geplant. Um nun die Verwirrung zu vervollständigen: Haben Sie von *Fyracle* [7] schon einmal etwas gehört? Nein? Es handelt sich hier nicht um eine Krankheit, sondern um einen Entwicklungszweig des Unternehmens *Janus Software* basierend auf Firebird, das Firebird quasi in einem Oracle-Modus laufen lassen kann. Hört sich verrückt an, aber bei dieser Entwicklung ging es darum, dass man das Open-Source-ERP/CRM-System *Compiere* [8] mit Firebird betreiben kann, ohne all zu viele DB-spezifische Code-Änderungen in *Compiere* vornehmen zu müssen. So unterstützt Fyracle u. a. das Oracle CONNECT BY-Konstrukt, oder auch PL/SQL-kompatible Stored Procedures. Hoffentlich hat Ihnen der Artikel einen guten Überblick über die neuesten Features von Firebird 1.5 verschafft.

Links & Literatur

[1] www.firebirdsql.org
[2] www.sourceforge.net
[3] www.ibphoenix.com/downloads/Firebird_v15.108_ReleaseNotesGerman.pdf
[4] www.ibphoenix.com/main.nfs?a=ibphoenix&s=1102862285:464178&page=ibp_ss_vs_classic
[5] www.firebirdsql.org/index.php?op=ffoundation
[6] firebird.sourceforge.net/index.php?op=devel&sub=engine&id=roadmap
[7] www.janus-software.com/fb_fyracle.html
[8] www.compiere.org