Evolution statt Revolution – Ein Streifzug durch die Neuerungen

Firebird 2.5

Das Firebird-Projekt setzt die kontinuierliche Weiterentwicklung des gleichnamigen Open Source RDBMS fort und kann in Release 2.5 wieder mit interessanten Features aufwarten. Dieser Artikel gibt Ihnen einen kompakten Überblick über die wichtigsten Neuerungen.

von Thomas Steinmaurer

Heutzutage ist eine Revolution im Umfeld der über die Jahrzehnte gereiften Datenbanktechnologie kaum mehr möglich. Interessant ist es aber auch zu betrachten, in welche Richtungen sich (Open-Source-) RDBMS-Produkte weiterentwickeln, um als Kernkomponente für die Datenhaltung von strukturierten Daten noch bessere Services bieten zu können. Das gilt auch für das Open Source RDBMS Firebird [1]. Über die Änderungen in den einzelnen Versionen habe ich bereits in [2], [3], [4] und [5] berichtet. Nun möchte ich die Tradition mit diesem Artikel fortsetzen, der ausgewählte Neuerungen in Release 2.5 vorstellt. Zum Zeitpunkt des Verfassens des Artikels stand Firebird 2.5 Release Candidate 1 kurz vor der Veröffentlichung. Ein guter Zeitpunkt also, um sich vor dem Release der finalen Version einen Überblick über die neuen Features zu verschaffen. Denn so kann man eine Entscheidungsgrundlage für die Durchführung eines Upgrades finden.

Serverarchitekturen

Da heutzutage jeder handelsübliche Rechner mit zwei oder mehr CPU-Kernen ausgestattet ist und im Serverbereich teilweise acht oder mehr Kerne/CPUs ihre Arbeit verrichten, hat das Firebird-Projekt die bestmögliche Unterstützung von Mehrprozessorsystemen (SMP) zu seinem Ziel erklärt. Eine große Bedeutung kommt dabei der effizienten Verwendung der vorhandenen CPU-Ressourcen in Form der Verteilung von mehreren Aufgaben auf die verfügbare CPU-Landschaft zu.

Durch das „ein Prozess pro Verbindung"-Modell der Firebird-*ClassicServer*-Architektur kann das Betriebssystem die gestarteten Prozesse auf die verfügbaren CPUs aufteilen. Das bedeutet, dass diese Architektur SMP-fähig ist. Allerdings hat *ClassicServer* den Nachteil, dass jede Datenbankverbindung (jeder *fb_inet_server*-Prozess) einen privaten Page- und Metadaten-Cache besitzt. Im Gegensatz dazu steht die *SuperServer*-Architektur, die einen Shared Cache für alle Verbindungen zu einer Datenbank verwaltet. Bei *ClassicServer* muss somit der verfügbare physikalische Hauptspeicher beachtet werden, da die Berechnung der maximalen Cache-Größe je Datenbank die Anzahl der Verbindungen inkludiert: *Page Size der Datenbank * Page Buffers * Anzahl der Verbindungen*. Wenn man jetzt ein einfaches Rechenbeispiel bei einer Datenbank mit einer Page Size von 8 KB, 5000 Page Buffers und im Schnitt 50 Datenbankverbindungen durchführt, dann kommt man hier auf ca. 1,95 GB maximal benötigten Hauptspeicher für das Caching – allerdings nur für diese eine Datenbank. Befinden sich mehrere Datenbanken auf einem Server, multipliziert sich das entsprechend. *SuperServer* ist hier genügsamer, da ein dezidierter Prozess *fbserver* alle Verbindungen zu den Datenbanken bedient, sodass in der angeführten Formel die Anzahl der Verbindungen wegfällt. Ein weiteres Argument für den Einsatz von *SuperServer* ist, dass der bei *ClassicServer* notwendige Kontext-Switch zwischen Prozessen kostspieliger als zwischen Threads innerhalb eines Prozesses ist. Im Gegensatz dazu ist für *SuperServer* auf einem 32-Bit-Betriebssystem maximal 2 GB (durch einen Switch in *boot.ini* auf 3 GB ausdehnbar) Hauptspeicher nutzbar. Das ist unter anderem ein Grund, warum das Firebird-Projekt den *ClassicServer* vor Jahren wieder zum Leben erweckt hat. Leider ist *SuperServer* auch in Firebird 2.5 nicht voll SMP-fähig.

Firebird 2.5 diente für die Firebird-Entwickler als entwicklungstechnischer Migrationspfad von 2.1 auf 3.0 in Bezug auf die Vereinheitlichung der Threading-Architektur, mit dem Ziel, SMP-Systeme

bestmöglich zu unterstützen. Die in 2.5 neu eingeführte *SuperClassic*-Architektur mit verbesserter SMP-Unterstützung ist dafür ein sehr wichtiger Schritt. Jetzt könnte der Leser zu Recht kritisch anmerken: „Noch eine Architektur? Und was soll man jetzt verwenden?“ Im nächsten Abschnitt werde ich versuchen, Ihnen die neue Architektur schmackhaft zu machen – bei mir hat sie sich auf 64-Bit-Systemen bereits bewährt – und Ihnen die Auswahl mit einer Gegenüberstellung zu erleichtern.

SuperClassic – Verbesserte SMP-Unterstützung

Ich habe *SuperServer* als nicht voll SMP-fähig bezeichnet. Konkret bedeutet das, dass Abfragen in simultanen Verbindungen zu einer Datenbank zu einem Zeitpunkt immer nur auf einer CPU laufen können, da die Requests je Datenbank intern von der Engine unter *SuperServer* serialisiert werden. Allerdings können in *SuperServer* simultane Verbindungen zu mehreren Datenbanken sehr wohl auf mehrere CPUs verteilt werden. Das ist ein wichtiger Unterschied. Vereinfacht ausgedrückt, bedeutet das nämlich, dass in *SuperServer* je Datenbank zu einem Zeitpunkt nur eine CPU nutzbar ist. Hier schafft die neue *SuperClassic*-Architektur Abhilfe, die, wie der Name schon sagt, eine Kombination aus *Super-* und *ClassicServer* darstellt. *SuperClassic* kann auch bei mehreren simultanen Verbindungen zu einer Datenbank mehrere CPUs nutzen. Die neue Architektur kann wie folgt charakterisiert werden:

- Ein dezidierter Serverprozess (*fb_inet_server –m* unter Windows bzw. *fb_smp_server* unter POSIX) für alle Verbindungen (identisch zu *SuperServer*)
- Privater Page- und Metadaten-Cache. Das impliziert die identische Berechnung der maximalen Hauptspeichernutzung für das Caching wie bei *ClassicServer*.
- SMP-Fähigkeit für simultane Verbindungen zu einer und mehreren Datenbanken
- SMP-Fähigkeit bei Sweeps und Services-API-Requests
- Kein exklusiver Lock auf die Datenbankdatei (identisch zu *ClassicServer*).
- Gecachte Verbindung zur Benutzerdatenbank für einen schnelleren Verbindungsaufbau (identisch zu *SuperServer*)
- Sicherer Shutdown aller Datenbankverbindungen durch Beenden des dezidierten Serverprozesses (identisch zu *SuperServer*)
- Automatisches Beenden aller Datenbankverbindungen beim Crash des Serverprozesses (identisch zu *SuperServer* – ein Nachteil)
- Zielplattform: Natürlich unter 32-Bit lauffähig, allerdings erscheint der Einsatz unter 64-Bit sinnvoller, da hier keine Limitierung in Bezug auf die maximale Hauptspeicheradressierung pro Prozess existiert (identisch zu *SuperServer*).

Wie man sieht, handelt es sich dabei um eine bunte Mischung aus *Super-* und *ClassicServer*-Eigenschaften. Der *SuperClassic*-Server ist im Download von *ClassicServer* enthalten. Um mit dem *SuperClassic*-Server arbeiten zu können, muss unter Windows *fb_inet_server.exe* mit dem Switch *–m* (steht für Multi-Threading) gestartet werden. Unter POSIX steht eine eigene ausführbare Datei *fb_smp_server* zur Verfügung, allerdings ist nach der Installation, z. B. über das *rpm* Package, *ClassicServer* und nicht *SuperClassic* betriebsbereit. Ein mitgeliefertes Shellt-Skript *changeMultiConnectMode.sh* im Firebird-*bin*-Verzeichnis erlaubt die einfache Umschaltung zwischen *ClassicServer* und *SuperClassic*. Dieses Skript registriert *fb_smp_server* sogar als System-Daemon in Runlevel 2, 3 und 5. Der XINET-Service wird für *SuperClassic* nicht benötigt. Somit hat man es in Firebird 2.5 nun mit vier Architekturen zu tun: *SuperServer*, *ClassicServer*, *SuperClassic* und *Embedded Server*. Um Ihnen die Auswahl einer Architektur zu erleichtern, habe ich in Tabelle 1 die Eigenschaften der Architekturen gegenübergestellt. Fett dargestellte Einträge sind Neuerungen in der jeweiligen Architektur in Firebird 2.5.

Obwohl das primäre Ziel von 2.5 die Verbesserung der SMP-Unterstützung hin zu einer vereinheitlichten Threading-Architektur für 3.0 war, findet man auch neue Features vor, die den Einsatz von Firebird noch einfacher und produktiver gestalten. Diese werden auszugsweise in den nachfolgenden Abschnitten erläutert.

Embedded basiert nun auf SuperClassic

Über Firebird Embedded habe ich in [6] berichtet. In 2.5 weist der Embedded Server die folgenden Eigenschaften auf:

- Basiert nun auf *SuperClassic* (vormals *SuperServer*) inklusive aller SMP-Vorteile
- Kein exklusiver Lock mehr auf die Datenbankdatei
- Thread Safe

Da kein exklusiver Lock mehr auf die Datenbankdatei benötigt wird, verfügt der Embedded Server über neue Einsatzmöglichkeiten. Nun ist nämlich eine Mischung aus mehreren Embedded-Prozessen und regulären *Classic-* und *SuperClassic*-Servern möglich, mit der man auf ein und dieselbe Datenbankdatei lesend und schreibend zugreifen kann. Die benötigte Synchronisation erfolgt dabei über eine globale Lock-Tabelle, die z. B. unter Windows (englische Ausgabe) in *C:\Documents and Settings\All Users\Application Data\firebird* verwaltet wird. *SuperServer* kann in diesem Multi-Connect-Kontext nicht verwendet werden, weil diese Architektur einen exklusiven Lock auf die Datenbankdatei benötigt.

Thread Safe bedeutet, dass nicht mehr eine eigenständige Datenbankverbindung pro Thread notwendig ist, sondern *ein* Verbindungs-Handle von mehreren Threads sicher verwendet werden kann. Die dafür auf Verbindungsebene benötigten Synchronisationroutinen befinden sich nun im Embedded Server bzw. in den Clientbibliotheken, das heißt, auch die Clientbibliotheken (*gds32.dll*, *fbclient.dll* etc.) sind nun Thread Safe. Es ist allerdings in der Clientanwendung eine Thread-Synchronisation notwendig, um Seiteneffekte zu vermeiden, z. B. wenn auch ein Transaktions-Handle von mehreren Threads gemeinsam benutzt wird. Hierzu mein Tipp: Behalten Sie die alte Denkweise bei: eigenständige Verbindung und Transaktion pro Thread, um Seiteneffekte zu vermeiden. Des Weiteren kann die *SuperClassic*-SMP-Fähigkeit im Embedded Server nur mit einer separaten Verbindung pro Thread voll ausgeschöpft werden.

Audit und Trace Services

Den neuen Audit und Trace Services in Firebird 2.5 werde ich einen eigenen Artikel in einer der nächsten Ausgaben widmen, da eine vollständige Abhandlung dieses sehr interessanten, neuen Features in diesem Artikel keinen Platz findet, zumal die Dokumentation (Release Notes) etwas spärlich ist und manche Fragen zur Verwendung unbeantwortet bleiben. Nur soviel vorweg: Bei den Audit und Trace Services handelt es sich um einen konfigurierbaren Mechanismus, der die kontinuierliche Protokollierung von Operationen auf Datenbankebene und von Services-API-Aufrufen in eine Textdatei ermöglicht.

Grundsätzlich wird zwischen *System Audit* und *User Trace* unterschieden. Ein System Audit wird von der Firebird Engine beim Start des Serverprozesses automatisch gestartet – die Konfiguration liegt in einer Konfigurationsdatei. Als Beispielkonfiguration wird hier eine Datei *fbtrace.conf* im Firebird-Installationsverzeichnis mitgeliefert, die für die eigenen Bedürfnisse angepasst werden kann. Ein System Audit wird durch das Setzen des neuen Konfigurationsparameters *AuditTraceConfigFile* in *firebird.conf* aktiviert. Im Vergleich zu einem System Audit muss ein User Trace explizit von einem Benutzer über einen speziellen Services-API-Aufruf mit einer übergebenen Trace-Konfiguration gestartet werden. Ein User Trace überlebt einen Server-Restart nicht. Wie gesagt, mehr dazu in einer der nächsten Ausgaben.

Erweiterungen in den MON$-Tabellen

Version 2.5 weist auch Erweiterungen in den Monitoring- (*MON$*-)Tabellen auf, die allerdings nur in Datenbanken mit einer ODS (On-Disk Structure) 11.2 verfügbar sind. Hier handelt es sich um Datenbanken, die mit 2.5 erstellt bzw. unter dieser Version mittels *gbak* oder des Services API zurückgesichert wurden. Generell sollte ein Backup-/Restore-Zyklus bei einem Upgrade gewählt werden, um alle neuen ODS-bezogenen Features „freizuschalten". Firebird 2.5 kann allerdings auch mit Datenbankdateien arbeiten, die mit früheren Versionen erstellt wurden. In ODS 11.2 sind zwei neue *MON$*-Tabellen hinzugekommen:

- *MON$MEMORY_USAGE*: Aktuelle Speichernutzung auf Datenbank-, Session-, Transaktions- bzw. Anweisungsebene
- *MON$CONTEXT_VARIABLES*: Überblick über alle mit *RDB$SET_CONTEXT* gesetzten benutzerdefinierten Kontextvariablen

Seit der Einführung der Monitoring-Tabellen in 2.1 konnten einzelne Anweisungen mit einem *DELETE* auf *MON$STATEMENTS* beendet werden. In Firebird 2.5 ist es nun möglich, mit einem *DELETE* auf die *MON$ATTACHMENTS*-Tabelle komplette Datenbankverbindungen zu beenden. Alle zum Abbruchzeitpunkt offenen Transaktionen werden implizit mit einem Rollback zurückgesetzt. Folgendes Beispiel beendet alle Datenbankverbindungen außer der eigenen: *DELETE FROM MON$ATTACHMENTS WHERE MON$ATTACHMENT_ID <> CURRENT_CONNECTION.*

Eine weitere Neuerung ist, dass reguläre Benutzer (nicht *SYSDBA* bzw. der Datenbank-Owner) nicht mehr nur die Monitoring-Daten der eigenen Verbindung sehen, sondern die Einschränkung auf Basis des angemeldeten Benutzernamens geschieht. Diese Erweiterung wurde auch in den 2.1 Branch zurückportiert und steht seit 2.1.2 auch für Firebird-2.1-Benutzer zur Verfügung.

Security

Firebird 2.5 stellt, wiederum nur in ODS-11.2-Datenbanken, eine neue Rolle *RDB$ADMIN* zur Verfügung. Mit dieser Rolle können regulären Benutzern SYSDBA-ähnliche Rechte auf Datenbankebene eingeräumt werden. Ein Beispiel: *GRANT RDB$ADMIN TO USER U1.*

Der Benutzer *U1* ist hiermit Mitglied der Rolle *RDB$ADMIN*. Nun muss zur Verbindungszeit noch die Rolle *RDB$ADMIN* angegeben werden, um die SYSDBA-ähnlichen Rechte auch tatsächlich nutzen zu können. Mit einem *REVOKE*: *REVOKE RDB$ADMIN FROM USER U1* können dem Benutzer *U1* diese Rechte wieder entzogen werden.

Eine Änderung in Firebird 2.5 in Bezug auf die seit 2.1 unterstützte Win-

Architektur	-	Neu in 2.5	-	SC
Ausführbare Datei	*fbserver*	*fb_inet_server –m* (Windows) *fb_smp_server* (POSIX)	*fb_inet_server*	*fbembed.dll* (Windows) *libfbembed.so* (POSIX)
Verbindungsmodell	- Dezidierter Prozess pro Instanz - Mehrere (gepoolte) Worker Threads	wie in SS	- Dezidierter Prozess pro Verbindung - Separate Worker Threads für Sweep und Services-API-Requests pro Verbindungsprozess	Serverprozess läuft im Adressraum der Clientanwendung
Cache-Modell	Shared Page und Metadaten-Cache pro Datenbank	Non-Shared Page und Metadaten-Cache pro Datenbankverbindung	wie in SC	**wie in SC**
Max. Cache-Größe pro Datenbank	PageSize * PageBuffers	PageSize * PageBuffers * Anzahl der Verbindungen	Berechnung wie in SC	**Berechnung wie in SC**
Max. Cache-Größe pro Instanz	Cache-Größe pro Datenbank * Anzahl Datenbank pro Instanz	Berechnung wie in SS [1]	Berechnung wie in SS [1]	**Berechnung wie in SS** [1]
Max. Cache-Größe pro Server	Cache-Größe pro Instanz * Anzahl der Instanzen	Berechnung wie in SS [2]	Berechnung wie in SS [2]	**Berechnung wie in SS** [2]
SMP-Support bei simultanen Verbindungen je Datenbank	Nein [3]	Ja	Ja	**Ja** [4]
SMP-Support bei simultanen Verbindungen bei unterschiedlichen Datenbanken	Ja	Ja	Ja	**Ja**
SMP-Support bei Sweep	Nein	Ja	Ja	**Ja**
SMP-Support bei Services-API-Anfragen	Teilweise [5]	Ja	Ja	**Ja**
Exklusiver Lock auf Datenbankdatei	Ja	Nein	Nein	**Nein**
Thread-sichere Clientbibliothek	**Ja**	**Ja**	**Ja**	**Ja**
Liste der verbunden Datenbanken/Benutzer via API	Ja	Ja	Nein	Ja
Gecachte Verbindung zur Benutzerdatenbank	Ja	Ja	Nein	Ja
Instanz kann sicher heruntergefahren werden	Ja	Ja	Nein	Ja
Beendigung aller Verbindungen bei Crash	Ja	Ja	Nein	Ja
Verbindungsprotokoll für Multi-Threaded-Clientanwendungen	**Beliebig** [7]	**Beliebig** [7]	**Beliebig** [7]	-
Zielplattform	64-Bit [8]	64-Bit [8]	32/64-Bit	32/64-Bit [8]
Konfigurierbarer TCP-Port für Eventnotifikation	Ja [9]	Ja [9]	**Ja** [9]	Ja [9]
Unmittelbare Detektierung von abgebrochenen Clientverbindungen	Ja	Ja	**Ja**	Ja

[1] Maximale Cache-Größe je Datenbank kann höher sein, daher kann auch die maximale Cache-Größe je Instanz höher als in SS sein.
[2] Maximale Cache-Größe je Instanz kann höher sein, daher kann auch die maximale Cache-Größe je Server höher als in SS sein.
[3] Anfragen zur selben Datenbank werden von der Engine intern serialisiert.
[4] Falls eine separate Verbindung je Thread verwendet wird
[5] Abhängig vom Services-API-Call kann dieser wieder zurück in die Engine gehen und wird daher serialisiert.
[6] Neu in Firebird 2.5: Verwendung einer Datenbankdatei über mehrere Embedded bzw. reguläre SC- und CS-Prozesse ist möglich.
[7] TCP/IP, Netbeui oder lokal (XNET). Vormals nur TCP/IP.
[8] Läuft auch unter 32-Bit, allerdings wird 64-Bit wegen der erweiterten Speicheradressierung je Prozess bevorzugt.
[9] Via *RemoteAuxPort*-Konfigurationsparameter in *firebird.conf*

Tabelle 1: Vergleich der Firebird-2.5-Architekturen

dows-Authentifizierung ist, dass ein Windows-Benutzer mit Administratorrechten nicht mehr automatisch auf SYSDBA gemappt wird. Ein Beispiel: Mit Firebird 2.1 unter Verwendung der Windows-Authentifizierung mit meinem Windows-Benutzer, ausgestattet mit Administratorrechten:

```
SQL> select current_user from rdb$database;

USER
========================================
SYSDBA
```

Unter Firebird 2.5:

```
SQL> select current_user from rdb$database;
USER
========================================
DOMAIN\STEINMAURER
```

Das hat natürlich direkte Auswirkungen auf die Aktionen, die nun in der Datenbank angestoßen werden dürfen. In Firebird 2.1 war man im „SYSDBA-Modus“, mit der Möglichkeit, Datensätze zu manipulieren, bestehende Tabellen zu ändern bzw. zu droppen, usw. In Firebird 2.5 müssen dem Benutzer *STEINMAURER* diese Berechtigungen erst durch SYSDBA bzw. dem Datenbankeigentümer eingeräumt werden. Da das allerdings eine zeitaufwendige Sache darstellen kann, hat man hier in der Implementierung noch weiter gedacht und eine Möglichkeit zum „Auto Admin Mapping“ geschaffen. Je Datenbank kann definiert werden, ob ein angemeldeter Windows-Benutzer mit Administratorrechten unter Verwendung der Windows-Authentifizierung automatisch SYSDBA-/Admin-Rechte für die Datenbank erhält. Das erfolgt mit: *ALTER ROLE RDB$ADMIN SET AUTO ADMIN MAPPING;*. Und kann wie folgt auch wieder entfernt werden: *ALTER ROLE RDB$ADMIN DROP AUTO ADMIN MAPPING;*.

Ein weiterer Punkt in Hinblick auf die Kompatibilität zu 2.1 unter Windows ist, dass in 2.5 als Default-Authentifizierungsmechanismus der Modus *native*, also über die Firebird-Benutzerdatenbank, aktiviert ist. In 2.1 war *mixed* voreingestellt, also Windows-Authentifizierung, sofern beim Verbindungsaufbau kein (Firebird-)Benutzername angegeben wird. Mit dem Parameter *Authentication* in *firebird.conf* kann der Authentifizierungsmechanismus gewählt werden.

Spracherweiterungen

Firebird 2.5 kann auch in Bezug auf Spracherweiterungen in den unterschiedlichen Bereichen – DDL, DML und PSQL – wieder mit Neuerungen aufwarten. Eine komplette Übersicht kann den Release Notes entnommen werden. Die aus meiner Sicht interessantesten Entwicklungen möchte ich hier anführen:

- *CREATE/ALTER/DROP USER* für das Benutzermanagement. Vormals war das nur mit dem Tool *gsec* bzw. über die Services API möglich
- *ALTER VIEW* für das Ändern von View-Definitionen. Vormals war ein *DROP* und *CREATE* notwendig, das sich allerdings bei Abhängigkeiten zu anderen Objekten als schwierig erwies

- *CREATE OR ALTER VIEW* für die Neuanlage bzw. Änderung einer View-Definition, abhängig davon, ob die View bereits existiert oder nicht
- *ALTER TABLE* für *COMPUTED BY*-Felder
- Möglichkeit der Verwendung von Stored Procedures in der *FROM*-Klausel einer View
- Default *COLLATION* auf Datenbankebene
- Regular-Expression-Unterstützung in Form eines neuen *SIMILAR TO*-Prädikats

und Vieles mehr. Auf zwei Spracherweiterungen im Bereich PSQL möchte ich mit je einem Beispiel etwas genauer eingehen: *AUTONOMOUS TRANSACTION* und Cross-Datenbankabfragen mit einer Erweiterung in *EXECUTE STATEMENT*. *AUTONOMOUS TRANSACTION* ist ein Konzept in PSQL, um Datenoperationen ohne Beeinflussung durch die aufrufende Transaktion durchzuführen. Ein Beispiel aus den Release Notes soll das verdeutlichen (Listing 1).

Listing 1

Beispiel für AUTONOMOUS TRANSACTION

```
create table log (
 logdate timestamp,
 msg varchar(60)
);

create exception e_conn 'Connection rejected';

set term !;
create trigger t_conn on connect
as
begin
 if (current_user = 'BAD_USER') then
 begin
  in autonomous transaction do
  begin
   insert into log (
    logdate
    , msg
   ) values (
    current_timestamp
    , 'Connection rejected'
   );
  end
  exception e_conn;
 end
end!
set term ;!
```

Ein *ON CONNECT* Trigger protokolliert Verbindungsversuche von *BAD_USER* in eine Tabelle *LOG* und wirft eine Exception *e_conn*, um den Verbindungsversuch zu unterbinden. Der Log-Datensatz wird allerdings trotzdem geschrieben, da das in einem *AUTONOMOUS TRANSACTION*-Block erfolgt. Ohne dieses Konzept wäre dieser Log-Eintrag nämlich nicht möglich, da das Werfen der Exception *e_conn* die aufrufende Transaktion zurücksetzt und somit auch die Einfügeoperation zurückgesetzt werden würde.

Das zweite Beispiel behandelt eine syntaktische Erweiterung der *EXECUTE STATEMENT*-Anweisung, die es nun ermöglicht, Abfragen in PSQL über Datenbankgrenzen hinweg auszuführen. Dafür zuständig ist die neue *ON EXTERNAL*-Klausel in Verbindung mit der *AS USER*- und *PASSWORD*-Erweiterung. Listing 2 veranschaulicht das Ausführen einer Abfrage auf eine andere Datenbank inklusive der Rückgabe der Ergebnismenge an das aufrufende Programm.

EXECUTE STATEMENT ist natürlich nicht an *EXECUTE BLOCK* gebunden, sondern kann auch in Stored Procedures und Triggern verwendet werden.

Listing 2

Beispiel für EXECTURE STATEMENT ... ON EXTERNAL

```
SET TERM !!;
EXECUTE BLOCK
 RETURNS (COMPANY_NAME VARCHAR(40))
AS
BEGIN
 FOR
  EXECUTE STATEMENT 'SELECT COMPANY_NAME FROM
CUSTOMER'
  ON EXTERNAL 'localhost/3051:mydb2.fdb'
  AS USER 'U1'
  PASSWORD 'u1'
  INTO :COMPANY_NAME
 DO
 BEGIN
  SUSPEND;
 END
END
!!
SET TERM ;!!
```

Noch was?

Ja. Die Services API wurde dahingehend erweitert, dass nun die Nbackup-Funktionalität für inkrementelle Backups, das neue Auditing/Tracing und die erweiterten Online-/Shutdown-Modi aus Firebird 2.1 über Services-API-Aufrufe in eigene Anwendungen integriert werden können. Der Konfigurationsparameter *RemoteAuxPort* für die Definition des zu verwendenden TCP-Ports bei der Firebird-Event-Notifikation wird nun auch für *ClassicServer* unterstützt. Eine sehr willkommene Erweiterung, wenn man *ClassicServer* hinter einer Firewall betreibt und man auf Firebird Events nicht verzichten möchte. Wird das Auditing/Tracing via Services API von Ihren Zugriffskomponenten (IBObjects, FIBPlus, IBDAC, Jaybird etc.) noch nicht unterstützt, dann können Sie auf das neue Kommandozeilentool *fbtracemgr* zurückgreifen, das Ihnen die Tür zu diesem neuen Feature öffnet.

Ausblick

Firebird 2.5 hat Einiges zu bieten. Vor allem die Möglichkeit eines kontinuierlichen Trace und die verbesserte SMP-Unterstützung machen den Einsatz sehr reizvoll. In Bezug auf die verbesserte SMP-Fähigkeit ist zu sagen, dass in einer Präsentation [7] auf der Firebird-Konferenz 2008 von Dmitry Yemanov, Entwicklungsleiter im Firebird-Projekt, die Sprache von einer 25 %igen Performancesteigerung in einer SMP-Umgebung bei *SuperClassic* gegenüber dem *ClassicServer* in einem intern durchgeführten TPC-C Benchmark ist. Version 2.5 ist allerdings nicht das Ende der Fahnenstange. Im Oktober 2009 wurde der HEAD Branch für die Entwicklung von 3.0 freigegeben. In einem ersten Schritt werden die lokalen Entwicklungen der Entwickler in diesen HEAD Branch gemerged. Mögliche Neuerungen sind:

- Shared Page und Metadaten-Cache bei feingranularem Multi-Threading-Support
- Support für Packages zur logischen Gruppierung von Stored Procedures und Functions

- DDL-Trigger, die bei *CREATE/RECREATE-/ALTER/DROP*-Anweisungen feuern
- Berechtigungskonzept (*GRANT/REVOKE* für *CREATE/ALTER* ...) für das Ausführen von DDL-Anweisungen
- Histogramminformationen für Indizes
- Stored Functions
- Java-basierte Stored Procedures und Functions

Vor der ersten freigegebenen Alpha-Version werden periodische Snapshot Builds verfügbar sein [8], um mit bereits umgesetzten Neuerungen experimentieren und Feedback an das Firebird-Projekt zurückgeben zu können. Die erste offizielle Alpha-Version wird wohl noch etwas dauern.

Zum Abschluss: Gratulation an das Firebird-Projekt, das nach 2007 auch in 2009 wieder den SourceForge Community Choice Award in der Kategorie „Best Project for the Enterprise“ [9] gewonnen hat.

Thomas Steinmaurer *ist wissenschaftlicher Mitarbeiter und Projektleiter am Software Competence Center Hagenberg (SCCH) [10] mit Schwerpunkt Datenmanagement und Data Warehousing im industriellen Umfeld. Des Weiteren ist er für die LogManager-Produktreihe bei Upscene Productions [11] verantwortlich.*

Links & Literatur

[1] http://www.firebirdsql.org

[2] Steinmaurer, Thomas: „Firebird 1.5“, in Entwickler Magazin 2.05

[3] Steinmaurer, Thomas: „Firebird 2.0“, in Entwickler Magazin 1.06

[4] Steinmaurer, Thomas: „Firebird 2.0b“, in Entwickler Magazin 3.07

[5] Steinmaurer, Thomas: „Firebird 2.1“, in Entwickler Magazin 5.07

[6] Steinmaurer, Thomas: „Firebird Embedded Server“, in Entwickler Magazin 3.06

[7] http://www.slideshare.net/ibsurgeon/firebird-25-architecture-by-dmitry-yemanov-in-english

[8] http://www.firebirdsql.org/index.php?op=files&id=snapshots

[9] http://sourceforge.net/community/cca09/winners/

[10] http://www.scch.at

[11] http://www.upscene.com