Ungleiche Zwillinge

InterBase 7 und Firebird 1.5 Alpha: Neuigkeiten von Borlands Datenbankserver und seinem Open Source-Pendant

von Karsten Strobel

Es ist in der Softwarebranche inzwischen zwar ein vertrauter Vorgang, wenn sich kommerzielle Produkte und lizenzkostenfreie Open Source-Programme Konkurrenz machen, aber die klonartige Ähnlichkeit von InterBase 6.0 und Firebird 1.0 dürfte bei einem solchen Wettbewerb einmalig sein. Um hier bestehen zu können, musste Borlands InterBase dringend technischen Vorsprung gewinnen. Das Ergebnis liegt nun vor. Doch auch der Vogel-aus-der-Asche legt nach, auch wenn dessen nächste Version noch nicht flügge ist und präsentiert seine Vision einer kostenlosen Datenbankzukunft.

Seit der Aufregung über den von Borland zunächst angekündigten Open Source-Gang der hauseigenen Relationalen Datenbank InterBase, der erst ein halber Rückzieher (Open Source: Ja, Engagement von Borland: Nein) und schließlich die Rückkehr zur kommerziellen Vermarktung des Produktes folgte, sind inzwischen zwei Jahre vergangen. Der Open Source-Version 6.0 folgte zunächst eine quasi identische, so genannte *zertifizierte Variante*, für die man auch wieder Geld an den Hersteller aus Scotts Valley zahlen durfte, vor einem Jahr dann eine schon wieder rein kommerzielle Version 6.5 und nun mit der 7.0 ein vollwertiges neues Release. Aber auch die „Rebellen“, die unter dem Namen Firebird aus den offen gelegten Quellcodes ein freies Konkurrenzprodukt schmieden, legen nach. Hier ist kürzlich eine frühe Alpha-Version des geplanten 1.5er-Releases erschienen, und auch ein zweites Wartungsupdate der derzeit noch aktuellen Version 1.0 ist kurz vor Weihnachten noch freigegeben worden.

Es wird kein Vergleichstest zwischen InterBase 7.0 und Firebird 1.5 Alpha vorgenommen, denn schließlich ist ersteres einen fertiges Produkt, letzteres befindet sich noch in der Entwicklungsphase. Daher halte ich es für besser, beide Kandidaten unabhängig voneinander vorzustellen und in einem abschließenden Resümee ein paar Verbindungen herzustellen.

Original gegen Bares

Für InterBase 7 hat Borland eine grundlegende Modernisierung der internen Softwarearchitektur vorgenommen und das Multithreading des Datenbankservers erheblich verbessert. Zwar verwendete InterBase schon in früheren Versionen Threads, um die Anfragen mehrere Sitzungen parallel ausführen zu können, aber die Sperr-Mechanismen waren zu grobkörnig, sodass sich die Threads allzu häufig gegenseitig ausbremsten, wodurch das Arbeiten ziemlich stockend werden konnte, wenn Abfragen viel Rechenzeit benötigten. Die Kontrolle der internen Ressourcen wie globale Speicherbereiche, wurde in InterBase 7 daher überarbeitet, mit dem Ziel, dass jeder Thread eine benötigte Ressource nur so lange wie unbedingt notwendig und nur im erforderlichen Umfang sperrt, wobei gemeinsames Lesen der selben Ressource durch mehrere Threads aber ohne Sperrung möglich sein soll. Auch der Einsatz von Symmetrischen Multiprozessor-Systemen (SMP), also Servern mit zwei oder mehr CPUs, soll nach diesen Änderungen einen erheblichen Leistungsschub bringen. In bisherigen Versionen profitierte InterBase überhaupt nicht vom Zukauf weiterer CPUs.

Aber nicht nur für den erlauchten Kreis der SMP-Server, sondern auch für ganz normale Ein-Prozessor-Rechner, zahlt sich die Überarbeitung des Threadings aus. Der Server behandelt gleichzeitige Anfragen jetzt mit gleichmäßiger Aufmerksamkeit. Zwar reagiert ein stark ausgelasteter Server natürlich nach wie vor langsamer als ein unbeschäftigter, aber wenn über eine Sitzung eine sehr aufwendige Abfrage gestartet wird, werden andere Sitzungen jetzt weiterhin mit akzeptablen Reaktionszeiten bedient, und auch der Aufbau einer neuen Datenbankverbindung geschieht jetzt wesentlich schneller. Um die neuen Fähigkeiten der Software auf einem echten Mehrprozessor-Computer ausschöpfen zu können, muss man neuerdings je weiterer CPU eine besondere Lizenz erwerben. Für Einplatz-Anwendungen, bei denen nur eine einzige Datenbankverbindung unterhalten wird, ergibt sich allerdings praktisch kein Vorteil, da InterBase einzelne Abfragen nicht

in mehrere Teilaufgaben auf Threads zu verteilen vermag.

Die zweite große Innovation bei InterBase 7 sind die Systemtabellen, mit denen der Datenbank-Administrator Zugriff auf eine enorme Menge an Informationen über die angemeldeten Benutzer, Sitzungen, Transaktionen, Statements, die Speicherausnutzung, das Caching und eine Menge mehr erhält. In früheren Versionen standen solche Administrations- und Statistikdaten nur sehr spärlich zur Verfügung. Interessant ist nun vor allem der Ansatz, wie InterBase diese Daten dem Anwender zur Verfügung stellt: Eigens für diesen Zweck wurde nämlich das Konzept der Temporären Tabellen eingeführt. Temporäre Tabellen sind das Privateigentum der Verbindung, in deren Kontext sie erzeugt werden. Die temporären Daten sind nur für Transaktionen dieser Verbindung sichtbar und mit Beendigung der Verbindung werden sie automatisch verworfen. Temporäre Tabellen verschiedener Sitzungen kommen einander nicht einmal dann in die Quere, wenn sie unter dem selben Namen erzeugt werden.

Laut SQL-Standard lautet die Syntax zum Anlegen einer solchen Tabelle *CREATE TEMPORARY TABLE*. Leider geht die Implementierung dieser Technik bei InterBase 7 noch nicht so weit, dass der Anwender eigene Tabellen dieser Art selbst erzeugen könnte, d.h. die zuvor genannte Syntax wird von InterBase 7 nicht akzeptiert. Aber für jede Verbindung entsteht automatisch eine Kollektion von Temporären Tabellen, die vom Datenbankkern automatisch erzeugt und mit Daten belegt werden. Der Anwendung kann auf diese Informationstabellen mit SELECT zugreifen, sie sogar in Stored Procedures oder Views verwenden und in einigen Fällen auch mit UPDATE steuernd Eingreifen, zum Beispiel um exzessiv lange laufende Statements abzubrechen oder das Trennen einer Sitzung zu erzwingen. Damit erhält der Datenbankadministrator endlich ein einfach anwendbares Mittel, um Zugriffe, die aus dem Ruder gelaufen, gezielt zu beenden, ohne dafür die Datenbank herunterfahren zu müssen.

Analog zu den mit *RDB$...* beginnenden Systemtabellen, in denen InterBase die Datenbankstruktur speichert, fangen die Namen der Temporären Systemtabellen mit *TMP$...* an. Ihre Struktur ist in den Release Notes und in der Language Reference dokumentiert. Es gibt insgesamt acht solche Tabellen, von denen ich nur die wichtigsten aufzählen möchte. Die hier nicht aufgeführten Tabellen betreffen vor allem die interne Speicherverwaltung des Servers und werden in der Praxis nur selten von Bedeutung sein.

- *TMP$DATABASE* enthält genau einen Datensatz, der die aktuelle Datenbank einschließlich dem aktuellen Ressourcenbedarf beschreibt.
- *TMP$ATTACHMENTS* enthält einen Datensatz für jede bestehende Verbindung.
- *TMP$TRANSACTIONS* beschreibt alle aktuellen Transaktionen.
- *TMP$STATEMENTS* liefert eine Liste aller gegenwärtig geladener Statements.
- *TMP$RELATIONS* enthält Informationen über die von der aktuellen Verbindung angesprochenen Tabellen.

Auch sämtliche Namen der Felder in diesen Tabellen beginnen mit *TMP$...* Der Umfang ist viel zu umfangreich, um ihn hier komplett vorzustellen. Ich greife nur beispielhaft eine einzige Fragestellung heraus, die in der Praxis besonders oft vorkommt und für die frühere InterBase Versionen keine brauchbare Lösung anzubieten hatten: Wie stelle ich fest, welcher Benutzer mit welcher Anweisung meinen Datenbankserver zur Zeit so stark belastet und wie schaffe ich es, diese unerwünschte Aktion abzubrechen?

Wenn ein Benutzer eine extrem langwierige Abfrage laufen lässt, wie zum Beispiel das folgende, bewusst ineffizient formulierte SELECT-Statement, das auf die Employee-Demodatenbank angewendet werden kann [1]

```
select count(*)
from sales
full join customer on (1=1)
full join employee on (1=1)
full join job on (1=1)
full join salary_history on (1=1)
```

dann wird vermutlich nach einiger Zeit der Administrator anfangen, unangenehme Fragen zu stellen, weil der Server plötzlich dauernd ausgelastet ist und sich die Antwortzeiten für andere Anwender verschlechtern. Während zur Ermittlung des Verursachers früher Einzelverhöre notwendig waren, kann der Admin den Fall jetzt bequem vom Schreibtisch aus lösen:

```
select s.tmp$statement_id, s.tmp$sql, s.tmp$timestamp,
a.tmp$attachment_id, a.tmp$user, a.tmp$user_ip_addr,
a.tmp$user_host
from tmp$statements s
inner join tmp$attachments a on (a.tmp$attachment_
                                 id = s.tmp$attachment_id)
where s.tmp$state = 'ACTIVE' and
(current_timestamp - s.tmp$timestamp >= (1/24/60*5))
```

Diese Anweisung liest aus der Tabelle *TMP$STATEMENTS* alle zur Zeit aktiven (d.h. zur Zeit laufenden) Abfragen, die seit mindestens fünf Minuten laufen. Über die in *TMP$STATEMENTS* gespeicherte *tmp$attachment_id* werden außerdem aus der Tabelle *TMP$ATTACHMENTS* die Daten der Client-Verbindung gelesen, zu der die fragliche Anweisung gehört. Als Ergebnis erhält man also eine Liste der schon verdächtig lange laufenden Abfragen einschließlich deren eindeutige Identnummern (*s.tmp$statement_id*) und sogar den Klartext der laufenden SQL-Anweisungen (*s.tmp$sql*). Auch der Verursacher, also der Datenbank-Client, der die Abfrage gestartet hat, lässt sich anhand des InterBase-Benutzernamens (*a.tmp$user*) und der IP-Adresse sowie des Host-Namens (*a.tmp$user_ip_addr* und *a.tmp$user_host*) leicht feststellen.

Wenn sich herausstellt, dass eine schon lange laufende Anweisung wohl nicht in akzeptabler Zeit fertig werden wird, kann man sie durch Änderung der Spalte *tmp$state* auf den Strings *CANCEL* gezielt abbrechen:

```
update tmp$statements
set tmp$state = 'CANCEL'
where tmp$statement_id = 4711
```

Auf ähnliche Weise lassen sich auch Transaktionen oder Client-Verbindungen (Attachments) zwangsweise beenden.

Wahr oder unwahr?

Ebenfalls neu ist der schon lange vermisste Datentyp *BOOLEAN*, der bisher

durch Verwendung eines alternativen Typs und Vereinbarung eines Wertes für die beiden Zustände emuliert werden musste. Zusammen mit dem *BOOLEAN*-Typ wurden die Schlüsselworte *TRUE*, *FALSE* und *UNKNOWN* eingeführt. *UNKNOWN* bedeutet nichts anderes als *NULL* und ein *BOOLEAN*-Feld mit diesem Status verhält sich auch in jeder Hinsicht so wie ein *NULL*-Feld. Diese Neuerung erlaubt nun endlich auch die verkürzte Schreibweise boolscher Verknüpfungen wie zum Beispiel *BOOL_FIELD_1 OR BOOL_FIELD_2* anstatt *(BOOL_FIELD_1 = 1) OR (BOOL_FIELD_2 = 1)*.

Hier noch einige der kleineren Neuerungen in InterBase 7:

- Die maximale Länge von Objektnamen (Tabellennamen, Feldnamen usw.) hat sich von 31 auf 63 Zeichen (jeweils plus ein terminierendes Null-Zeichen) verdoppelt.
- Die neue empfohlene Namenserweiterung für InterBase-Datenbanken lautet jetzt *.IB* (früher *.GDB*). Diese rein organisatorische Änderung ist erforderlich, weil Windows ME und XP beim Öffnen von Dateien mit der *.GDB*-Endung automatisch eine Kopie der Datei erstellt, was für Datenbankdateien i.d.R. unerwünscht ist. Die frühere Benutzerdatenbank *ISC4.GDB* heißt nun *ADMIN.IB*, ändert sich inhaltlich aber nicht.
- Externe Dateien können nun nicht mehr in einem beliebigen Verzeichnis des Servers, sondern nur noch im InterBase-Unterverzeichnis *\EXT* oder an einer per Parameter freizugebenden Stelle angelegt werden. Damit wird eine erhebliche Sicherheitslücke geschlossen, die einem Datenbankanwender früher unter Umständen den Diebstahl oder das Überschreiben fremder Dateien ermöglichte.
- Beim Ausführen von SQL-Skripten kann vor dem Deklarieren von Stored Procedures und Triggers nun auf das Umdefinieren des Terminatorzeichens (mit *SET TERM*) verzichtet werden. Offenbar ist man zu der Einsicht gelangt, dass dieser Konstrukt schlicht unnötig ist und eine Prozedur auch ohne diese etwas unelegante Maßnahme in einem Script isoliert werden kann. Diese Änderung betrifft eigentlich nur das Konsolenprogramm *isql.exe*. *IBConsole.exe* bietet diesen Komfort leider nicht.
- Für Java-Entwickler steht jetzt mit einer neuen Version von InterClient ein JDBC-Treiber der Typklasse 4 zur Verfügung. Dieser neue Typ-4-Treiber trägt verwirrender Weise die Versionsnummer 3.0. Der Vorteil der neuen Architektur ist, dass der Java-Client direkt mit dem InterBase-Server kommuniziert, womit die bisher erforderliche Zwischenschicht InterServer (die serverseitige JDBC-Implementierung nach Typ 3) nicht mehr länger benötigt wird.
- Die Client-Bibliothek *gds32.dll* ist jetzt multithread-tauglich. In früheren Versionen musste man eine Reihe von Beschränkungen beachten, wenn ein InterBase-Clientprogramm mit mehreren Threads auf die Datenbank zugreifen sollte, um das Risiko von Schutzverletzungen und andere fatale Ereignisse zu vermeiden. Diese neue Version dieser DLL serialisiert konkurrierende Zugriffe auf Objekte, die von unsynchronisierten Threads gemacht werden.

Ein Blick in die Release Notes zur Version 7.0 verrät auch etwas über die weiteren Pläne von Borland: Unter der Überschrift „Die folgenden Worte werden wahrscheinlich künftig als Schlüsselworte definiert werden“ sind die Begriffe *PRESERVE*, *GLOBAL* und *TEMPORARY* aufgeführt. Dies deutet darauf hin, dass man in einem künftigen Release die Funktionalität der Temporären Tabellen auch für den Anwendet zugänglich machen möchte, da diese drei Worte im SQL-Standard in diesem Zusammengang verwendet werden.

In anderer Beziehung lassen die Release Notes leider eine Lücke offen: Im Gegensatz zu früheren Versionen werden keinerlei Bugfixes aufgezählt. Ob dieses unschöne Versäumnis nun ein Versehen ist, oder ob man die vielleicht ziemlich kurze Liste einfach nicht gerne preisgeben wollte, entzieht sich meiner Kenntnis. In den einschlägigen Diskussionsforen wird in diesem Zusammenhang gerne auf Borlands „QC“ (Quality Central) [2] verwiesen. Unter diesem Titel haben die Borländer nämlich eine öffentlich zugängliche Online-Datenbank für Fehlerbeschreibungen, Wünsche, Anregungen und dergleichen für ihre ganze Produktpalette geschaffen. Und tatsächlich sind in der InterBase-Sektion dieser Liste einige Bugs als „gefixt in 7.0“ markiert. Doch leider ist diese Liste alles andere als vollständig, und es ist allgemein bekannt, dass Borland intern mit einer anderen Fehler-Datenbank arbeitet, die bis zum heutigen Tag nicht mit dem öffentlichen „QC“ abgeglichen worden ist. Qualität und Zuverlässigkeit ist – gerade im Bereich der Datenbanken – ein vorrangiges Kriterium bei der Produktauswahl. Neben einigen Leistungsmerkmalen, mit denen sich das Borland-Produkt von der selbstgezüchteten aber ungeliebten Konkurrenz abhebt, ist es vor allem die Erwartung der Kunden in eine höhere Stabilität und bessere Unterstützung, die das Klientel veranlassen könnte, nicht auf die Gratis-Variante umzusteigen. Diese Erwartung gilt es aber – besonders in Anbetracht früherer Fehlleistungen in Punkto Fehlerbereinigung – erst noch zu erfüllen. Und auch bei der Version 7.0 gibt es leider noch altbekannte Bugs, die zum Teil schon in Version-5-Zeiten beschrieben wurden, aber die offenbar nicht totzukriegen sind (auch wenn sich „QC“ darüber ausschweigt).

Leerer Werkzeugkasten

Nicht pünktlich fertig geworden ist Borland mit der neuen Management-Oberfläche, die unter dem Namen IBConsolePlus als Nachfolger von IBConsole Verbesserungen im Handling, mehr visuelle Möglichkeiten bei der Arbeit mit Datenbankobjekten und Zugang zu den neuen Server-Features bieten soll. Ausgeliefert wird derzeit weiterhin IBConsole. Für den Nachfolger mit dem Plus im Namen läuft ein öffentliches Betaprogramm, für das in den Release Notes zu InterBase 7 eine Downloadadresse angegeben ist. Die aktuelle Beta-Version (Abb. 1) macht einen noch sehr unfertigen Eindruck, und kann produktiv wegen permanenter Fehlermeldungen nicht verwendet werden. Vielleicht hat sich der Autor dieses Programms zu sehr in Skins (ladbare Layouts

für die Programmoberfläche) verliebt, und vor lauter Begeisterung für dieses Feature, das sonst eher von „Trendsoftware“ wie MP3-Playern bekannt ist, das Hauptziel aus den Augen verloren? Auch ist (derzeit) noch nichts zu sehen von der Unterstützung der neuen Überwachungsfunktionen über die Temporären Systemtabellen. Andere sind hier wieder mal schon weiter. So bietet zum Beispiel IBExpert [3] bereits speziell für IB7 ein Polling-Fenster zur zyklischen Anzeige von Daten, bereits vorbereitet (aber selbst erweiterbar) mit SELECT-Statements für die neuen Systemtabellen.

Geflügel gratis

Im Herbst 2002 überraschte die Firebird-Gemeinde mit dem Downloadangebot einer frühen Testversion des geplanten Releases 1.5 [4,5]. Auf die erste Alpha Version folgten mit hoher Schlagzahl weitere; inzwischen ist man bei Alpha-5 angekommen. Die Liste der Neuerungen ist auf den ersten Blick beeindruckend lang, was bei genauerem Hinsehen aber auch dadurch begründet ist, dass neue Bugs in frühen Alpha-Ständen in den Folgeversionen wiederum als erledigt markiert werden. Eine erhebliche Arbeitsleistung verbirgt sich hinter so schlichten Notizen wie „Code port from C to C++“ oder „New exception handling logic“. Hier wurde eine ziemlich grundlegende Überarbeitung des ursprünglichen InterBase-Quellcodes vorgenommen mit dem Ziel, modernere objektorientierte Methoden einschließlich eines zeitgemäßen Handlings von Ausnahmen (*try .. catch*) nutzen zu können, womit die Codebasis insgesamt zukunftssicherer gemacht werden soll. Im Idealfall merken Anwender ohne Programmier-Ambitionen von solchen Maßnahmen gar nichts.

Im Gegensatz zu Borlands InterBase liegt bei Firebird 1.5 der (bisherige) Schwerpunkt weniger auf der internen Architektur oder den Managementfunktionen. Stattdessen stehen SQL-Spracherweiterungen im Mittelpunkt. Neben so nützlichen Neuerungen wie der Möglichkeit, in einer *GROUP BY*-Klausel einen UDF-Aufruf zu verwenden oder in einer *ORDER BY*-Klausel mit *NULLS {FIRST | LAST}* festzulegen, ob Null-Spalten vor

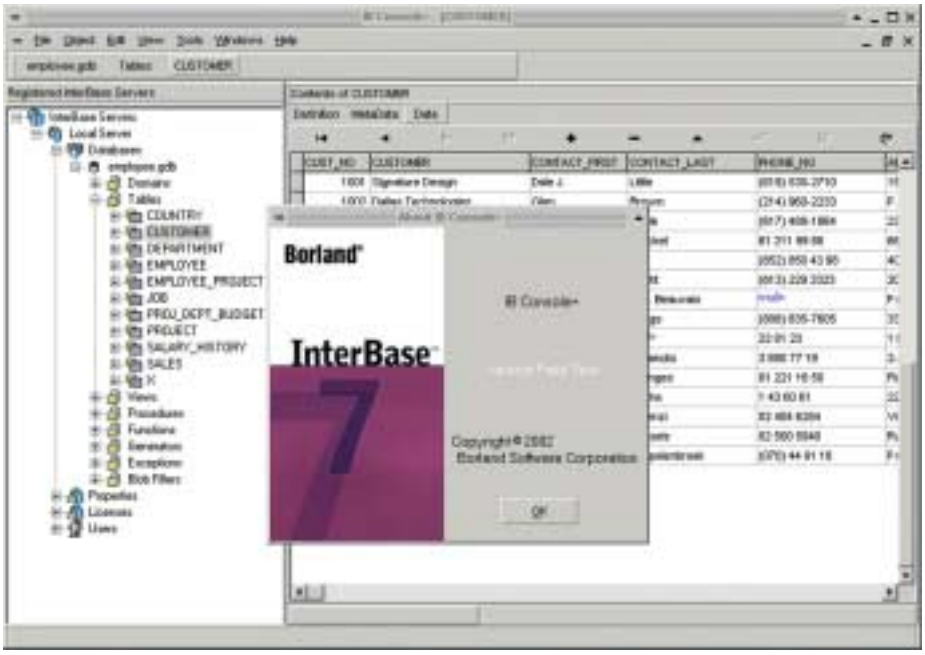

Abb. 1: Frühe Testversion von IBConsole+

oder hinter den Nicht-Null-Spalten angeordnet werden sollen, verdienen vor allem die neu eingeführten Ausdrücke *CASE*, *NULLIF* und *COALESCE* Erwähnung.

Die *CASE*-Funktion erlaubt es, innerhalb einer *SELECT*-Anweisung den Wert einer Spalte anhand eines Ausdrucks zu ermittelt. Nützlich ist das zum Beispiel allem dann, wenn man den Wert eines Feldes in Abhängigkeit einer Bedeutungsliste umformen möchte:

```
select
 case t.Feld1
  when 1 then 'eins'
  when 2 then 'zwei'
  else 'weiss nicht'
 end
from Meine_Tabelle t
```

Die *NULLIF*-Funktion besitzt zwei Parameter, die auf Gleichheit untersucht werden. Das Ergebnis der Funktion ist *NULL*, wenn beide Parameter gleich sind. Anderenfalls wird der Wert des ersten Parameters zurückgegeben. Die Funktion verwandelt also einen Parameter in *NULL*, wenn die Bedingung erfüllt ist. *COALESCE* ist zwar nicht die exakte Gegenfunktion zu *NULLIF*, man kann aber doch ungefähr den umgekehrten Ansatz erkennen, denn diese Funktion wird im Allgemeinen genutzt, um *NULL*-Spalten zu vermeiden. Aus einer variabel langen Parameterliste liefert diese Funktion den Wert des ersten Parameters (von links nach rechts) zurück, der nicht *NULL* ist. Sowohl *NULLIF* als auch *COALESCE* lassen sich übrigens – etwas komplizierter – auch mit *CASE* ausdrücken.

Auch im Bereich der Stored Procedures und Triggers gibt es einige Innovationen. Zum einen können lokale Variablen in dem Stil

```
DECLARE my_var INTEGER = 123;
```

mit einem Initialisierungswert deklariert werden. Das Erzeugen von Prozeduren oder Trigger kann mit der Einleitung „CREATE OR ALTER ...“ vorgenommen werden, womit es egal ist, ob das betreffende Objekt neu ist oder redefiniert wird. Schließlich können Triggers nun mehreren Ereignissen (z.B. *BEFORE INSERT OR UPDATE*) zugeordnet werden.

Sehr interessant erscheint mir auch der neue Ansatz, der es innerhalb einer Stored Procedure (oder in einem Trigger) erlauben soll, SQL-Statements indirekt auszuführen. Mit dem neuen Befehl *EXECUTE VARCHAR <variable>* soll es möglich werden, ein SQL-Statement wie z.B. ein Insert oder Update-Befehl, der in einem Varchar-Feld gespeichert ist, zur Ausführung zu bringen. Einige komplizierte Probleme könnten auf diese Weise eine einfa-

che Lösung erhalten. Das Konzept hat aber leider die Schwäche, dass kein Ergebniswert ausgewertet werden kann. Daher ist die indirekte Ausführung von *SELECTs* praktisch nicht möglich.

Sinnvoll ist auch eine Neuerung, mit der sich Indices, die implizit für Primary- oder Foreign Keys erzeugt werden, benennen und in ihrer Orientierung (aufsteigend oder absteigend) definieren lassen. Besonders das Lesen von Plan-Ausgaben (die Beschreibung der Zugriffsstrategie beim Ausführen eines Befehls) könnte den Charakter einer Folter verlieren, wenn man statt der bisher üblichen, automatisch erzeugten Indexnamen wie *RDB$FOREIGN7* endlich sinnvolle, benutzerdefinierte Begriffe einsetzen könnte. Die Syntax dieser segensreichen Neuerung sieht so aus:

```
CREATE TABLE Tabelle_A (Feld1 INTEGER NOT NULL PRIMARY
    KEY USING DESCENDING INDEX Idx_TabA_Feld1)
```

Nur eine Randnotiz ist mir die Einführung des Datentyps *BIGINT* wert, der mit *NUMERIC(18)* äquivalent ist. Ich meine, dass man auf neue native Datentypen besser verzichten sollte, sofern sie keinen realen Vorteil bringen. Wem es nur um die Benennung zu tun ist, kann auch jetzt schon einen Domain-Typ definieren.

Mehr Gefallen finde ich dagegen an der Einführung serverseitiger Datenbank-Aliase. Die bisherige Situation, dass ein InterBase/Firebird-Client den Pfad zur Datenbankdatei auf der Festplatte des Servers kennen muss, ist ebenso unkomfortabel wie unsinnig und hat sich gelegentlich nicht nur als Ursache für Missverständnisse erwiesen, sondern sogar die Brisanz gewisser Sicherheitslücken erheblich vergrößert. Abhilfe war hier schon lange überfällig. Bei Firebird 1.5 kommt sie in schlichter Form, nämlich als eine einfache Textdatei namens *aliases.conf*, in der auf dem Server in dem Stil *MeinAlias=Pfad-und-Dateiname* ein Ersatzname für den vollständigen Pfad definiert werden kann. Ein Remote-Client kann dann den Verbindungsstring wahlweise in *MeinHost:MeinAlias* verkürzen. Als weiterer Verbesserungsvorschlag hierzu wird auch eine Konfigurations-Option diskutiert, mit der die Auswertung konkreter Pfadnamen in Verbindungsstrings ganz abgeschaltet werden könnte. Ich finde die Schlichtheit dieser Lösung bestechend, besonders angesichts der manchmal viel zur komplizierten Ansätze anderer Datenbankprodukte, bei denen zur Lösung dieses simplen Problems mitunter Verzeichnisdienste bemüht werden, womit die Erstkonfiguration keineswegs einfacher wird.

Ganz selbstbewusst hat sich Firebird 1.5 nun nicht nur mehr äußerlich einen neuen Namen gegeben, sondern tritt auch bei der Benennung der Verzeichnisse, der Programmdateien und der Dateiendungen, der Gestaltung von Icons und der Formulierung von Meldungen aus dem Schatten des Urvaters InterBase. Was bisher *IB*.. hieß, nennt sich nunmehr *FB*.. . Aber nicht nur das Erscheinungsbild des Produktes wandelt sich. Die Firebirdler haben sich auch zu der Erkenntnis durchgerungen, dass Software zum Nulltarif auf die Dauer nur schwer zu entwickeln ist. Die Lösung dieses Dilemmas heißt nicht etwa Lizenzgebühr (das wäre schon aus rechtlichen Gründen unmöglich), sondern besteht in der Gründung der Stiftung „FirebirdSQL Foundation“, die in Australien errichtet wurde, aber weltweit für zahlende Mitglieder offen ist [6]. Diese Stiftung soll aus freiwilligen Mitgliedsbeiträgen und Sponsorengeldern von Personen und Unternehmen, die durch die Nutzung der Open Source-Software erhebliche Lizenzkosten einsparen können, einigen wichtigen Produktivkräften einen – wenn auch bescheidenen – finanziellen Ausgleich für ihren Einsatz zahlen. Sie Satzung definiert für Mitglieder ein Mitspracherecht in der Steuerung der Stiftungsaktivitäten und der Verwendung der eingenommenen Gelder.

Die schlechte Nachricht: Ein Erscheinungstermin für ein stabiles Release von Firebird 1.5 wird derzeit nicht genannt. Zum Trost kann man immerhin auf das frisch erschienene Wartungs-Release 1.0.2 verweisen, mit dem wieder ein paar Bugs aus Firebird 1.0 beseitigt worden sind [4,5].

Fazit

Während Borland im neuesten InterBase-Release vor allem die Multitasking-Architektur der Software modernisiert, hochwertige Managementfunktionen hinzugefügt und sich damit am Kern der Datenbank zu schaffen gemacht hat, konzentrieren sich die derzeit bei Firebird ruchbaren Absichten mehr auf eine weiter außen liegenden Funktionsschicht, nämlich auf eine Aufwertung der SQL-Sprache. Je nach Anwendungsart könnte man die eine oder die andere Strategie als richtig ansehen. Borlands Innovationen werden vor allem leistungshungrigen Client-Server-Anwendungen mit etlichen gleichzeitigen Verbindungen und gestressten Admins zugute kommen. Firebird verteilt Bonbons an Entwickler, die noch ein bisschen mehr aus ihren *SELECT*-Befehlen und Stored Procedures herausholen wollen.

Noch nicht ganz klar ist, von welchem Anbieter man sich die bessere Qualität und Zuverlässigkeit erwarten kann. Borland verfügt über eine sehr umfangreiche Testfallsammlung, das Personal und die Organisation für eine wirklich gute Qualitätssicherung. Wenn sich aber trotzdem Fehler in das Produkt einschleichen, hat man in der Vergangenheit leider nicht mit schnell verfügbaren Patches rechnen dürfen, sondern war meist gezwungen, lange damit zu leben. Die Firebird-Community reagiert da bedeutend flexibler. Hier könnte das Problem vielleicht in der steigenden Zahl der an der Entwicklung einer komplexen Software beteiligten Hobbyisten bestehen. Die lose strukturierte Firebird-Organisation muss erst noch klare Strukturen herausarbeiten, mit denen die Qualität neuer Versionen sichergestellt werden kann.

Für manche mögen – all die zunehmenden technischen Unterschiedlichkeiten der beiden Kontrahenten beiseite gestellt – das schlagende Argument die Lizenzkosten sein. Doch die meisten Datenbanker sind kühle Rechner und wissen, dass die Ausgaben für Lizenzen nicht der einzige Posten im *Total Cost of Ownership* sind. Je nach den gesetzten Prioritäten kann dabei die eine oder andere Variante den Vorzug bekommen.

Links & Literatur

[1] *Der Entwickler*, Panikschalter, 4/2002,
[2] qc.borland.com
[3] www.ibexpert.com
[4] firebird.sourceforge.net
[5] www.ibphoenix.com
[6] www.firebirdsql.org/ff/foundation