Ein Streifzug durch die aktuellen Entwicklungen im Firebird-Projekt

Firebird 2.0

Im *Entwickler* 2.2005 [1] haben wir uns bereits näher mit den Neuerungen in Firebird 1.5 beschäftigt. Das 1.5er-Release ist für unterschiedliche Plattformen verfügbar und hat sich mit den unterschiedlichen Firebird-Architekturen (Super/Classic/Embedded Server) als sehr stabil erwiesen. Dies hat Firebird einen ausgezeichneten Ruf eingebracht, selbst für unternehmenskritische Anwendungen. Wir haben es hier nun mit einem ernsten Mitstreiter in der Szene der Open-Source-DBMS zu tun. Grund genug, uns die aktuellen Entwicklungen rund um Firebird 2.0 anzusehen.

von Thomas Steinmaurer

Wenn Sie diesen Artikel lesen, wird vermutlich bereits eine Beta-Version von Firebird 2.0 verfügbar sein. Dieser Artikel basiert noch auf Firebird 2.0 Alpha 3. Bereits diese öffentlich verfügbare Alpha-3-Version, die stabiler ist als manch andere Produkte im Beta/RC-Stadium, zeigt die zu erwartenden Neuerungen sehr eindrucksvoll. Bevor Sie jedoch weiter lesen und eigene Experimente mit Firebird 2.0 Alpha/Beta durchführen, ein ernst gemeinter Rat des Firebird-Projekts und auch von mir: Diese frühen 2.0er-Versionen sind nicht für Produktionsumgebungen geeignet!

Installation

Aktuelle Distributionen von Firebird 2.0 Alpha/Beta stehen auf der Firebird-Projektseite unter [2] zur Verfügung. Verwenden Sie am besten einen eigenen Rechner oder Virtualisierungsprodukte wie VMware oder Microsoft Virtual PC, um Firebird 2.0 zu testen. Für Windows steht bereits mit Alpha 3 eine Setup-Routine zur Verfügung, die die Installation zum Kinderspiel macht. Meine bevorzugte Variante ist allerdings die Installation über das ZIP-Archiv, das einem alle Freiheiten bei der Auswahl der zu testenden Architektur (Super/Classic Server) einräumt. Entpacken Sie hierfür das ZIP-Archiv in ein Verzeichnis Ihrer Wahl und führen Sie *install_super.bat* (Super Server) bzw. *install_classic.bat* (Classic Server) aus. Diese Dateien befinden sich im *\bin*-Unterverzeichnis. Hiermit wird Firebird in der Super- oder Classic-Server-Architektur als Dienst registriert und auch automatisch gestartet. Im Anschluss müssen Sie noch sicherstellen, dass Anwendungen die Firebird-2.0-Version der Clientbibliothek *fbclient.dll* bzw. *gds32.dll* verwenden. Mit dieser Variante der Installation können Sie auch Firebird 1.5 und 2.0 Seite an Seite auf einer Maschine einrichten. Es kann jedoch immer nur ein Firebird-Server einer Version gestartet sein. Sie sollten hierbei auch sicherstellen, dass im Windows-Systemverzeichnis immer die Version der Clientbibliothek vorgefunden wird, die der Version des laufenden Servers entspricht.

Hier auf jede Neuerung im Detail einzugehen würde bedeuten, die bereits sehr umfangreichen, jedoch noch unvollständigen Release Notes ins Deutsche zu übersetzen. Ich werde mich im weiteren Verlauf des Artikels auf die aus meiner Sicht interessantesten Neuerungen konzentrieren.

On-Disk-Structure (ODS) Version 11

Gleich zu Beginn eine der wichtigsten Neuerungen in Firebird 2.0, nämlich die

Page Size [Bytes]	Maximale Indexlänge [Bytes]
1.024	252
2.048	508
4.096	1.020
8.192	2.044
16.384	4.092

Tabelle 1: Maximale Indexlänge in Abhängigkeit der Page Size

geänderte ODS, die nun als Version 11 bezeichnet wird. Die Highlights der Änderungen in ODS 11 können wie folgt aufgezählt werden:

- Neue Indexstruktur
- Erhöhung der maximalen Länge des Textes einer benutzerdefinierten Exception von 78 auf 1.021 Bytes
- Je ein *RDB$DESCRIPTION*-Feld in den Systemtabellen für Generatoren (*RDB$GENERATORS*) und Rollen (*RDB$ROLES*)
- 40-Bit Datensatzzeiger, um ein ~30-GB-Limit der maximalen Tabellengröße aufzuheben

Vor allem die neue Indexstruktur hat es mir persönlich sehr angetan, da hiermit vorhandene Limits und Performanceprobleme aus Firebird 1.x beseitigt/verbessert wurden, die im direkten Zusammenhang mit Indizes stehen. In Firebird 2.0 beträgt nun das Limit der maximalen Indexlänge nicht mehr 252 Bytes, sondern dies ist nun von der verwendeten Page Size der Datenbank abhängig. Das exakte Limit (in Bytes) kann mit der Formel *(Page Size / 4) – 4* berechnet werden. Ein Beispiel: In einer Datenbank mit einer Page Size von 4.096 Bytes kann nun ein VARCHAR(1020) indiziert werden, sofern ein(e) 1-Byte-Zeichensatz/Sortierreihenfolge verwendet wurde. Wird die, für den Zeichensatz ISO8859_1-gängige, 3-Byte-Sortierreihenfolge DE_DE verwendet, so kann noch immer ein VARCHAR(340) indiziert werden. Tabelle 1 gibt einen Überblick über die maximale Indexlänge in Abhängigkeit der Page Size.

Ein gewichtiger Vorteil der neuen Indexstruktur, neben der größeren maximalen Indexlänge, ist die Beseitigung eines Performanceproblems im Zusammenhang mit der Garbage Collection (= Entfernen alter Datensatzversionen), wenn Indexeinträge mit sehr vielen Duplikaten entfernt werden müssen. In Firebird 1.x hatte das eine hohe CPU-Auslastung zur Folge, die über einen längeren Zeitraum auftreten konnte, je nach Anzahl der zu entfernenden Duplikate. Die neue Indexstruktur bietet des Weiteren eine effektivere Komprimierung der Indexeinträge und die Wartung der Selektivität pro Feld in einem von mehreren Feldern zusammengesetzten Index.

Um die Vorteile der ODS 11 nutzen zu können, ist ein Backup/Restore-Zyklus notwendig. Hierbei erstellen Sie ein Backup Ihrer Datenbank mit Firebird 1.x und führen ein Restore mit Firebird 2.0 durch. Mit diesem Vorgehen wird die Datenbank im ODS-11-Format neu erstellt.

Performancesteigerung

Der Optimizer wurde im Vergleich zu Firebird 1.0 bereits in Firebird 1.5 erheblich verbessert. In 2.0 setzte man die kontinuierliche Verbesserung dieser Kernkomponente fort. Der Optimizer ist in bestimmten Situationen nochmals cleverer geworden, was die Ermittlung des schnellsten Zugriffspfades für die angeforderten Daten betrifft. Welche Situationen dies sind, entnehmen Sie am besten den Release Notes, die in jeder Firebird-2.0-Distribution enthalten sind. Weitere Merkmale der Performancesteigerung sind Verbesserungen im Lock- und Thread-Pool-Manager in der Super-Server-Architektur. Dies bedeutet allerdings nicht, dass Super Server nun eine SMP-Unterstützung anbietet. Die Vorteile einer Mehrprozessorumgebung sind nach wie vor der Classic-Server-Architektur vorbehalten. Des Weiteren wurde die maximale Cachegröße auf 128K Pages erhöht. Somit ist abhängig von der Page Size der Datenbank mehr Hauptspeicher für das Caching verwendbar. Außerdem wurde der Mechanismus für die Garbage Collection in der Super-Server-Architektur überarbeitet. Vormals war dies ein Hintergrund-Thread, der für die Garbage Collection verantwortlich war. In Firebird 2.0 Super Server kann dies nun eine Mischung aus kooperativem Garbage Collection und Hintergrund-Garbage-Collection sein. Unter kooperativem Garbage Collection versteht man, dass eine Abfrage, die Datensätze besucht, alte Datensatzversionen sofort entfernt, und die Entsorgung nicht einem Hintergrund-Thread überlässt. Welcher Mechanismus für die Garbage Collection in der Super-Server-Architektur verwendet werden soll, kann mit dem neuen Konfigurationsparameter *GCPolicy* in *firebird.conf* definiert werden. Die Garbage Collection in der Classic-Architektur erfolgt immer kooperativ.

Spracherweiterungen

Firebird 2.0 wird so genannte „Derived Tables" unterstützen. Hierbei handelt es sich um Unterabfragen in der FROM-Klausel einer SQL-Anweisung. Das fol-

SQL-Anweisung	Case-Insensitive Suche	Zugriffsplan	Verwendeter Index
*select * from customer where customer = 'Signature Design';*	Nein	PLAN (CUSTOMER INDEX (CUSTNAMEX))	CUSTNAMEX
*select * from customer where **upper**(customer) = 'SIGNATURE DESIGN';*	Ja	PLAN (CUSTOMER **NATURAL**)	Keiner
*select * from customer where **upper**(customer) = **upper**('signature design');*	Ja	PLAN (CUSTOMER **NATURAL**)	Keiner

Tabelle 2: Case-Insensitive Suche ohne Expression-Index

SQL-Anweisung	Case-Insensitive Suche	Zugriffsplan	Verwendeter Index
*select * from customer where customer = 'Signature Design';*	Nein	PLAN (CUSTOMER INDEX (CUSTNAMEX))	CUSTNAMEX
*select * from customer where **upper**(customer) = 'SIGNATURE DESIGN';*	Ja	PLAN (CUSTOMER **INDEX (CUSTNAME_CASEINSENS)**)	CUSTNAME_ CASEINSENS
*select * from customer where **upper**(customer) = **upper**('signature design');*	Ja	PLAN (CUSTOMER **INDEX (CUSTNAME_CASEINSENS)**)	CUSTNAME_ CASEINSENS

Tabelle 3: Case-Insensitive Suche mit Expression-Index

gende Listing zeigt ein einfaches Beispiel für eine Derived Table:

```
select min(sum_budget) from
(
select
 department
 , sum(budget) as sum_budget
from
 department
group by
 department
);
```

Neu in Firebird 2.0 sind auch Expression-Indizes. Hiermit wird es möglich sein, Indizes für den indizierten Zugriffspfad, basierend auf Ausdrücken, zu erstellen. Ein gutes Beispiel hierfür ist die indizierte Case-insensitive Suche in einem *VARCHAR*-Feld, die bisher nur über den Umweg eines Trigger-gewarteten Schattenfeldes, mit der großgeschriebenen Entsprechung, umsetzbar war. Dies kann in 2.0 über einen Expression-Index sehr einfach realisiert werden. Tabelle 2 zeigt die ausgeführte Anweisung sowie den vom Optimizer verwendeten Zugriffsplan, ohne dass ein Expression-Index auf dem Feld CUSTOMER existiert.

Erstellen wir nun einen Expression-Index mit UPPER auf dem Feld CUSTOMER:

```
create index custname_caseinsenson customer
                        computed by (upper(customer));
commit;
```

Der Optimizer verwendet nun einen Index auch für die case-insensitive Suche, da wir mit *UPPER(CUSTOMER)* die großgeschriebene Entsprechung des Inhalts des Feldes CUSTOMER indizieren. Tabelle 3 zeigt die verwendeten Zugriffspläne. Eine weitere interessante Spracherweiterung ist eine dem SQL99-Standard konforme Implementierung von Sequenzen (Sequences), die bereits von Oracle vertraut sein dürften. Mit der *COMMENT*-Anweisung können nun Beschreibungen von Datenbankobjekten über diese DDL-Anweisung gewartet werden. Bisher war dies nur mit direkten Änderungen des *RDB$DESCRIPTION*-Feldes in den Systemtabellen möglich.

Zusätzlich zu dem *FIRST/SKIP*-Konstrukt, welches die Anzahl der Datensätze aus einer Ergebnismenge limitiert, unterstützt Firebird 2.0 nun auch die *ROWS*-Syntax, die ebenso in *UNIONs*, Unterabfragen, *DELETE/UPDATE*-Anweisungen und auch in View-Definitionen verwendet werden kann. Die Unterstützung von *CROSS JOINs*, eine verbesserte Typermittlung bei *UNION*-Abfragen, die Möglichkeit der Verwendung von Feldaliases in einem *ORDER/GROUP BY* sowie die Sortierung nach dem Feldindex bei einem *SELECT * FROM TABELLE* sind willkommene Erweiterungen, die einem Entwickler das Leben erleichtern werden. Interessant ist auch die *RETURNING*-Klausel in einer *INSERT INTO*-Anweisung, die es zum Beispiel ermöglicht, einen durch einen Generator erzeugten Primärschlüsselwert im Kontext dieser Einfügeoperation zum Client zu transferieren. Ein einfaches Beispiel dafür zeigt das folgende Listing:

Neu in Firebird 2.0 sind auch Expression-Indizes

```
SQL> insert into customer (customer) values
                    ('Thomas Steinmaurer') returning cust_no;

   CUST_NO
========
     1016
```

Im Bereich von PSQL, der prozeduralen Sprache für Stored Procedures und Trigger, kann Firebird 2.0 ebenfalls mit Neuerungen aufwarten. Explizite Cursordeklarationen (vergleichbar zu denen in Oracle), Default-Werte für Übergabeparameter von Stored Procedures und die Unterstützung von *ROW_COUNT* für *SELECT*-Anweisungen sind interessante Aspekte für Entwickler, die viel im Bereich der serverseitigen Programmierung zu tun haben. Interessant ist auch die *EXECUTE BLOCK*-Anweisung, die quasi ein Ausführen von „dynamischem" PSQL erlaubt. Dahinter verbirgt sich die Möglichkeit, PSQL-Code als eine reguläre SQL-Anweisung auszuführen, als wäre es eine Stored Procedure, ohne dabei vorher jedoch eine Stored Procedure in der Datenbank ablegen zu müssen.

Benutzerdefinierte Kontextvariablen

Der Begriff Kontextvariablen wird einem zum Beispiel in Verbindung mit *CURRENT_USER* (= aktuell verbundener Firebird-Benutzer) vertraut sein. Welche Kontextvariablen verfügbar sind, war bis Firebird 1.x fix vorgegeben. Mit Firebird 2.0 geht man einen Schritt weiter und erlaubt benutzerdefinierte Kontextvariable im Kontext einer Datenbanksession und/oder einer Transaktion. Das Setzen und das Holen eines Wertes einer benutzerdefinierten Kontextvariable geschieht über die internen Systemfunktionen *RDB$SET_CONTEXT* und *RDB$GET_CONTEXT*. Es gibt hier das Konzept der Namensräume (Namespaces), die den Gültigkeitsbereich einer Kontextvariablen definieren. Es existieren drei fix vordefinierte Namensräume:

- *USER_SESSION:* Eine Kontextvariable ist für die aktuelle Datenbank-Verbindung/Session gültig.
- *USER_TRANSACTION:* Eine Kontextvariable ist für die aktuelle Transaktion gültig.
- *SYSTEM:* Vordefinierte Kontextvariablen wie zum Beispiel *NETWORK_PROTOCOL, CLIENT_ADDRESS.*

Damit das Ganze nicht zu abstrakt ist, wird in folgendem Listing der Namensraum *SYSTEM* verwendet, um Informationen über die aktuelle Client-Verbindung abzufragen.

```
SQL> select
CON> rdb$get_context('SYSTEM','NETWORK_PROTOCOL')
                                        as protocol
CON> , rdb$get_context('SYSTEM','CLIENT_ADDRESS')
                                    as client_address
CON> from
CON> rdb$database;

PROTOCOL    CLIENT_ADDRESS
========== ===============
TCPv4       127.0.0.1
```

Noch ist die Sache nicht besonders aufregend. So richtig interessant wird es allerdings, wenn eigene Kontextvariablen

und dessen Werte in den Namensräumen *USER_SESSION* bzw. *USER_TRANSACTION* gespeichert werden können. Eine gutes Anwendungsbeispiel für den Verwendungszweck benutzerdefinierter Kontextvariablen sind anwendungsspezifische Benutzer/Logins, die nicht notwendigerweise mit dem Firebird-Benutzer, der bei der Datenbankverbindung verwendet wurde, etwas zu tun haben. Wollen Sie nun das Login der Anwendung zum Beispiel in einem Trigger auswerten, dann können Sie sich diese Information in einer benutzerdefinierten Kontextvariable ablegen und diese später wieder abrufen. Das folgende Listing zeigt hierfür ein einfaches Beispiel:

```
SQL> select
CON> rdb$set_context('USER_SESSION','APP_LOGIN',
                                          'Thomas')
CON> from
CON> rdb$database;

RDB$SET_CONTEXT
==========
      0

SQL> select
CON> rdb$get_context('USER_SESSION','APP_LOGIN')
                                       as app_login
CON> from
CON> rdb$database;

APP_LOGIN
==========
Thomas
```

Administration

Eine wichtige Neuerung, die alle Firebird-Administratoren freuen wird, ist die Unterstützung für inkrementelle Online-Backups in Form der neuen Kommandozeilentools *NBak/NBackup*. In Firebird 1.x kann für ein Online-Backup einer Datenbank das Kommandozeilentool *gbak* oder das Services-API verwendet werden. Dies geschieht allerdings nach dem Alles-oder-Nichts-Prinzip. Gerade für Datenbanken im zwei- oder dreistelligen Gigabyte-Bereich musste eine Backup-Strategie gut geplant sein, da ein Backup/Restore-Zyklus durchaus einiges an Zeit in Anspruch nehmen konnte. *NBak/NBackup* wird *gbak* allerdings nicht ersetzen können, da die Verwendung von *NBak/NBackup* zum Beispiel kein Sweep oder ein Verkleinern der Datenbank durchführt. Des Weiteren kann mit *NBak/NBackup* auch nicht der Eigentümer einer Datenbank geändert werden. Ein transportables Backup muss auch weiterhin mit *gbak* erstellt werden. Bei *NBak/NBackup* handelt es sich vielmehr um ein inkrementelles Page-Level Online-Backup Tool. Mit *NBak/NBackup* wird es auch möglich sein, eine Datenbank mit einer Sperre zu versehen, um zum Beispiel eine konsistente Kopie der Datenbankdatei auf Dateisystemebene zu erstellen. Das Besondere daran ist, dass währenddessen aktive Verbindungen zur Datenbank bestehen dürfen, die auch Änderungen durchführen können. Solange eine Sperre auf einer Datenbank besteht, werden diese Änderungen in eine „Delta-Datei" geschrieben. Wird die Sperre von der Datenbank mit *NBak/NBackup* wieder entfernt, so wird die Datenbank mit der Delta-Datei zusammengeführt.

Sonstiges

Neben der Datenbank-Engine selbst wird auch aktiv am ODBC- bzw. JDBC-Treiber und auch dem Firebird .NET Data Provider gearbeitet. Hier stehen regelmäßig neue Versionen zur Verfügung. Auch im Bereich der Administrationswerkzeuge kommt mit „FlameRobin" [3] etwas direkt vom Firebird-Projekt. Es handelt sich hierbei um ein Cross-Plattform-Werkzeug für die Administration einer Firebird-Datenbank. FlameRobin befindet sich allerdings noch in einem frühen Entwicklungsstadium. Für die professionelle Entwicklung von Firebird-Datenbanken empfehle ich Ihnen weiterhin den Einsatz von Drittthersteller-Produkten wie Database Workbench [4] oder IBExpert [5].

Zusammenfassend kann man sagen, dass sich im Firebird-Projekt viel tut. Firebird 2.0 wird mit vielen Neuerungen aufwarten, das zeigen die Alpha/Beta-Versionen. In der nächsten Ausgabe befassen wir uns mit dem Firebird Embedded Server, einer Variante für das Deployment einer Firebird-basierten Anwendung. ■

Links & Literatur

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

[2] www.firebirdsql.org

[3] www.flamerobin.org

[4] www.upscene.com

[5] www.ibexpert.com