Die Neuerungen in Firebird 2.1 im Überblick

Neu entflammt

Mit Firebird 2.0 veröffentlichte das Firebird-Projekt Ende letzten Jahres ein stabiles DBMS mit einer Vielzahl an Neuerungen. Die Interessantesten davon habe ich Ihnen in den beiden vorangegangen Artikeln [1], [2] im Entwickler Magazin bereits vorgestellt. Durch die kontinuierliche Weiterentwicklung, die durch die Firebird Foundation [3] finanziell unterstützt wird, steht nun bereits eine erste öffentliche Beta-Version von Firebird 2.1 zur Verfügung. Dieser Artikel gibt Ihnen einen Überblick, welche Neuerungen Sie in Firebird 2.1 erwarten können.

von Thomas Steinmaurer

Firebird 2.1 steht zum Redaktionsschluss dieses Artikels (7/2007) als erste öffentliche Beta-Version zur Verfügung. Obwohl es sich noch um kein Release handelt, das für Produktionsumgebungen bestimmt ist, kann man sich bereits zu diesem Zeitpunkt einen sehr guten Überblick darüber verschaffen, welche Neuerungen man erwarten kann. Diese frühen Binaries, die der Öffentlichkeit zugänglich gemacht werden, sind erstens aus technologischer Sicht interessant, um zu sehen, in welche Richtung sich Firebird als RDBMS entwickelt und zweitens erleichtert es Entscheidungsträger enorm, eine frühzeitige Planung für den Einsatz einer neuen Firebird-Version durchzuführen. In Firebird 2.1 ist für unterschiedliche Anwendergruppen wieder etwas Neues dabei. Datenbankentwickler werden die Erweiterungen im Bereich von SQL und PSQL zu schätzen wissen, wohingegen für Administratoren einer Firebird-Datenbank die vielfach geforderten Monitoring-Tabellen, ähnlich wie sie in InterBase seit Version 7 vorhanden sind, in Firebird 2.1 verfügbar sein werden. Sehen wir uns die wichtigsten Neuerungen nun etwas genauer an.

Monitoring

Eine Schwäche von Firebird gegenüber anderen Produkten war, dass es bis zur Version 2.1 keine Möglichkeit gab, detailliert abzufragen, was in einer Daten-

Tabelle	Zweck
MON$DATABASE	Informationen zur Datenbank wie Dateiname, Page Size, ODS, Sweep-Intervall, OIT, OAT usw.
MON$ATTACHMENTS	Verbindungen, die zur Datenbank bestehen inklusive der IP-Adresse, dem Prozessnamen der Client-Anwendung usw.
MON$TRANSACTIONS	Transaktionen, die aktuell ausgeführt werden
MON$STATEMENTS	Anweisungen, die aktuell ausgeführt werden
MON$CALL_STACK	Call-Stack von aktiven PSQL-Requests

Tabelle 1: Monitoring-Tabellen

bank zum aktuellen Zeitpunkt vorging. Dazu gehörte die Ermittlung, welche Datenbankverbindungen existierten bzw. welche Transaktionen und SQL-Anweisungen gerade ausgeführt wurden. Vor allem langlaufende Transaktionen konnten durch die Multi-Generationen-Architektur (MGA) von Firebird erhebliche Performanceprobleme verursachen, da Datensatzversionen einer älteren Transaktion auch für neuere Transaktionen als interessant angesehen werden mussten und somit der Verwaltungsaufwand für den Firebird-Server stieg. Typischerweise handelt es sich um so ein Szenario, wenn die Performance einer Firebird-Datenbank nach einem Backup/Restore-Zyklus sehr gut ist und im Laufe des Betriebs immer schlechter wird. Sollte dies der Fall sein, dann führen Sie ein *gstat –h* für diese Datenbank aus und beobachten Sie die Werte für *Oldest Transaction*, *Oldest Active Transaction* und *Next Transaction*. Sollten sich bei den hier ermittelten Werten große Lücken auftun, dann ist das ein Indiz für schlechtes Transaktionsmanagement in der Anwendung. Ist man für die Entwicklung der Anwendung selbst zuständig, dann kann man das Transaktionsmanagement einem Review unterziehen und hierbei Schwachstellen identifizieren und beseitigen. Schwieriger wird es, wenn man keinen Zugriff auf den Quellcode der Anwendung hat. In diesem Fall wüsste man vielleicht, dass man ein Problem mit einer langlaufenden Transaktion hat, aber man hat keine Möglichkeit, diese Transaktion zu identifizieren. Mit den neuen Monitoring-Tabellen gibt es nun einen Weg, mit normalen SELECT-Anweisungen abzufragen, welche Transaktionen aktiv sind und welche Anweisungen aktuell ausgeführt werden. So kann durch die folgende Abfrage ermittelt werden, welche Transaktionen in einer Datenbank derzeit gestartet sind:

```
SELECT * FROM MON$TRANSACTIONS
```

Tabelle 1 gibt einen Überblick über die vorhandenen Monitoring-Tabellen und ihren Zweck.

Aber nicht nur der lesende Zugriff auf diese Tabellen ist möglich, Firebird 2.1 wird zum Beispiel auch das Beenden von laufenden Anweisungen über eine *DELETE*-Anweisung auf die Tabelle *MON$STATEMENTS* erlauben. Um dieses neue Monitoring-Feature verwenden zu können, muss die Datenbank mit einer ODS (On-Disc Structure) in der Version 11.1 erstellt worden sein. Dieser folgt automatisch, wenn die Datenbank mit Firebird 2.1 erstellt bzw. wenn ein Firebird-Backup einer Datenbank mit *gbak* oder dem Services-API unter Firebird 2.1 zurückgesichert wurde.

Datenbank-Trigger

Ein weiteres interessantes Feature sind Datenbank-Trigger, die nicht an Tabellen gebunden sind und bei Datenänderungen feuern, sondern für eines der folgenden Ereignisse:

- Verbindungsaufbau (ON CONNECT)
- Verbindungsabbau (ON DISCONNECT)
- Start einer Transaktion (ON TRANSACTION START)
- Commit einer Transaktion (ON TRANSACTION COMMIT)
- Rollback einer Transaktion (ON TRANSACTION ROLLBACK)

Die vollständige Syntax zur Erstellung eines Datenbank-Triggers ist in Listing 1 angeführt.

Die Einsatzzwecke für Datenbank-Trigger sind vielseitig. So kann zum Beispiel der Start oder Commit einer Transaktion oder auch der Verbindungsaufbau bzw. -abbau durch einen Eintrag in einer Tabelle mitprotokolliert werden. Des Weiteren ist auch ein Unterbinden des Verbindungsaufbaus durch das Auslösen einer Exception in einem *ON CONNECT*-Trigger möglich. Als Beispiel dient der *ON CONNECT*-Trigger in Listing 2, der sicherstellt, dass ein Verbindungsaufbau nur mit einem Firebird-Benutzer *TOURISM* möglich ist.

Globale temporäre Tabellen

Neu in Firebird 2.1 sind auch Globale temporäre Tabellen (GTT). Sie wurden ursprünglich für Fyracle [4] entwickelt und finden nun mit Version 2.1 den Weg in die offizielle Codebasis von Firebird. Bei Fyracle handelt es sich um einen kommerziellen Ableger von Firebird, der einen Betrieb von Firebird in einem eingeschränkten Oracle-Kompatibilitätsmodus erlaubt. Ein vollständiger Artikel im Entwickler Magazin hat sich bereits mit Fyracle auseinandergesetzt. Dieser Artikel kann online unter [5] nachgelesen werden. Bei GTT handelt es sich um Tabellen, deren Metadaten permanent in den Systemtabellen gespeichert werden und für alle Datenbankverbindungen sichtbar sind. Die Daten sind dabei nur temporär für die Lebensdauer der Transaktion (*ON COMMIT DELETE*) bzw. der Datenbankverbindung (*ON COMMIT PRESERVE*) beschränkt. Jede Datenbanksession kann die Definition einer GTT verwenden, um, für andere Benutzer isoliert, Daten in einer GTT einzufügen, zu löschen, zu ändern und abzufragen. Die Syntax für die Definition einer GTT stellt sich folgendermaßen dar:

```
CREATE GLOBAL TEMPORARY TABLE <table_name>
                                <table_elements>
[ON COMMIT {PRESERVE | DELETE} ROWS]
```

Ein Einsatzszenario für GTT liegt zum Beispiel im Aufbereiten von Daten von komplexen Abfragen für das Berichtswesen oder der Datenanalyse. So können Zwischenergebnisse in GTT gespeichert und auf diese Daten in späterer Folge zugegriffen werden. Für die Speicherung von temporären Daten bieten GTT gegenüber normalen Tabellen den Vorteil, dass das Entfernen von Datensätzen viel schneller erfolgen kann, da bei GTT unter anderem keine Garbage Collection notwendig ist. Für die Optimierung des Zugriffs auf GTT ist es möglich, Indizes auf Feldern in einer GTT anzulegen. Ein Beispiel für eine einfache GTT, dessen Daten nach dem Beenden der Transaktion automatisch gelöscht werden, könnte so aussehen:

```
create global temporary table t1_on_delete (
 id integer
)
on commit delete rows;

commit;
```

Common Table Expressions

Ebenfalls aus der Entwicklung von Fyracle stammt eine dem SQL-Standard konforme Implementierung der so genannten Common Table Expressions (CTE). War vormals in Firebird eine rekursive Selectable Stored Procedure notwendig, um hierarchische Baumstrukturen abzufragen, so kann dies nun in Firebird 2.1 über eine CTE-Anweisung viel eleganter und vor allem für die Engine effizienter durchgeführt werden. Angenommen, man will in einer gegebenen hierarchischen Struktur für eine Abteilung mit der Abteilungsnummer *010* alle Unterabteilungen ermitteln, so kann dafür in Firebird 2.1 eine CTE wie in Listing 3 dargestellt, formuliert werden.

Domains in PSQL

Eine tolle Neuigkeit gibt es für diejenigen, die regelmäßig mit der Entwicklung von PSQL-Code für Trigger und Stored Proce-

Listing 1

Datenbank-Trigger-Syntax

```
<database-trigger> ::=
{CREATE | RECREATE | CREATE OR ALTER}
TRIGGER <name>
[ACTIVE | INACTIVE]
ON <event>
[POSITION <n>]
AS
BEGIN
...
END

<event> ::=
CONNECT
| DISCONNECT
| TRANSACTION START
| TRANSACTION COMMIT
| TRANSACTION ROLLBACK
```

Listing 2

ON CONNECT Trigger

```
create exception exc_connect 'Nur Benutzer TOURISM
                              darf sich verbinden!';

set term !!;
create trigger tri_on_connect on connect
as
begin
 if (current_user <> 'TOURISM') then
 begin
  exception exc_connect;
 end
end
!!
set term ;!!

commit;
```

Listing 3

Hierarchische Abfrage über CTE-Anweisung

```
with recursive d(dept_no, department) as
 (select dept_no, department
  from department
  where dept_no = '010'
 union all
 select d2.dept_no, d2.department
  from department d2, d
  where d.dept_no = d2.head_dept)
select dept_no, department from d;
```

Listing 4

Domains in PSQL

```
CREATE DOMAIN DOM AS INTEGER;

CREATE PROCEDURE SP (
 I1 TYPE OF DOM,
 I2 DOM)
RETURNS (
 O1 TYPE OF DOM,
 O2 DOM)
AS
 DECLARE VARIABLE V1 TYPE OF DOM;
 DECLARE VARIABLE V2 DOM;
BEGIN
...
END
```

dures zu tun haben. In Firebird 2.1 wird es endlich möglich sein, definierte Domains in PSQL für Parameter- und lokale Variablendeklarationen zu verwenden. Die Verwendung von Domains in PSQL kann auf zwei Arten erfolgen:

- Nur der Domainname; dann werden alle Attribute der Domain, inklusive einem vorhandenen *CHECK*-Constraint und einer *DEFAULT*-Definition verwendet.
- Dem Domainnamen wird die *TYPE OF*-Klausel vorangestellt; dann wird nur der zugrunde liegende Datentyp der Domain, ohne dem CHECK-Constraint und der DEFAULT-Definition, für die Deklaration herangezogen.

Ein einfaches Beispiel in Listing 4 soll die zu verwendende Syntax verdeutlichen Der Eingabeparameter I1, der Rückgabeparameter O1 und die lokale Variable V1 wurden mit *TYPE OF DOM* deklariert, somit wird nur der Datentyp der Domain DOM berücksichtigt. Im Gegensatz dazu wird für I2, O2 und V2 nicht die *TYPE OF*-Klausel verwendet. In diesem Fall wird der Datentyp, der *CHECK*-Constraint und die *DEFAULT*-Definition der Domain DOM für die Deklaration verwendet.

Weitere SQL-Spracherweiterungen

Neben den bereits erwähnten Neuerungen gibt es noch weitere interessante SQL-Spracherweiterungen, die in Firebird 2.1 verfügbar sein werden. Zum Beispiel:

- *UPDATE OR INSERT*-Anweisung für die Aktualisierung bzw. für die Neuanlage eines Datensatzes, abhängig davon, ob ein Datensatz, basierend auf einem benutzerdefinierten Matching-Kriterium (z.B. Primärschlüsselwert), existiert oder nicht. In früheren Firebird-Versionen musste man sich dazu eine Stored Procedure mit entsprechendem Exception Handling implementieren.
- Die *LIST()*-Funktion in einer *SELECT*-Anweisung ermöglicht die Konkatenation von Feldwerten mit einem benutzerdefinierten Trennzeichen. Auch diese Funktionalität war in früheren Versionen serverseitig nur über eine Stored Procedure realisierbar.
- Eine *RETURNING*-Klausel erlaubt die automatische Rückgabe von Feldinhalten zum Client nach dem Ausführen einer datenändernden Operation. Dies ist zum Beispiel für serverseitig generierte Primärschlüsselwerte interessant, um damit in der Client-Anwendung weiterarbeiten zu können.
- Eine Fülle an skalaren Funktionen sind nun in die Engine fix integriert und brauchen somit nicht mehr über User-Defined-Functions (UDFs) eingebunden werden. Neben Standardfunktionen wie ABS, SIN, COS usw. sind auch Funktionen wie *DECODE* (bekannt aus Oracle) oder auch *HASH* zur Berechnung eines Hashwertes verfügbar.

Multi-Instanzen

Für den Parallelbetrieb von mehreren Firebird-Serverinstanzen bietet Version 2.1 erstmals eine vollständige Unterstützung an. In früheren Firebird-Versionen war es möglich, zwei Serverinstanzen parallel auf unterschiedlichen TCP-Ports laufen zu lassen, wobei eine Instanz als Dienst und die andere als Anwendung gestartet werden musste. Firebird 2.1 bietet nun die Möglichkeit, über die *-name* Option des Kommandozeilentools *instsvc* einen benutzerdefinierten Instanznamen für die Installation des Dienstes anzugeben. Somit können mehrere Serverinstanzen als Dienst betrieben werden. Es ist allerdings darauf zu achten, dass der TCP-Port über den Parameter *RemoteServicePort* in der Konfigurationsdatei *firebird.conf* auf einen Wert eingestellt werden muss, der noch von keinem anderen Prozess verwendet wird. Alternativ dazu kann der Serverprozess auch mit dem *-p <port>*-Kommandozeilenschalter gestartet werden. Aus Benutzersicht wäre es wünschenswert, wenn die entsprechenden Einstellungen für die Installation von Multi-Instanzen einfach über die Setup-Routine erledigt werden könnten. In der Installationsroutine der Beta 1 ist hierfür noch nichts vorgesehen.

64-Bit-Unterstützung für Windows

Seit Firebird 2.0 gibt es bereits eine 64-Bit-Version für Linux (AMD64, EMT64). Neben den vielen Neuerungen im Bereich der Datenbankentwicklung und der Administration soll auch nicht unerwähnt bleiben, dass es mit Firebird 2.1 erstmals eine offizielle 64-Bit-Version für Windows geben wird. Firebird ist somit auch unter Windows in der Lage, Hauptspeicher > 2 GB entsprechend zu nutzen. Eine willkommene Neuerung, wenn es um den Betrieb von sehr großen Firebird-Datenbanken in einem abfrageintensiven Umfeld unter 64-Bit-Windows geht.

Fazit

Firebird 2.1 bietet wieder eine Fülle an interessanten Neuerungen. Leider wird auch Version 2.1 keine entsprechende SMP-Unterstützung in der Superserver-Architektur anbieten. Dies ist erst für Firebird 3.0 vorgesehen [6]. Gespannt darf man auch sein, wie sehr sich die Testphase von Firebird 2.1 hinzieht, da es verkündetes Projektziel auf der Firebird-Konferenz 2006 war, in kürzeren Zeitabständen Builds für Produktionsumgebungen mit Neuerungen und vor allem Bugfixe zur Verfügung zu stellen. Zum Abschluss möchte ich noch darauf hinweisen, dass ich auf der diesjährigen EKON 11 [7] mehr über die aktuellen Entwicklungen im Firebird-Projekt berichten werde. Ich würde mich freuen, Sie dort begrüßen zu dürfen.

Thomas Steinmaurer ist wissenschaftlicher Mitarbeiter am Software Competence Center Hagenberg (SCCH), Österreich. Des Weiteren ist er verantwortlich für die LogManager Series bei Upscene Productions. Er ist Mitbegründer der Firebird Foundation, regelmäßiger Autor im Entwickler Magazin und Sprecher auf Konferenzen, unter anderem auch auf der diesjährigen EKON11.

Links & Literatur

[1] entwickler-magazin.de/zonen/magazine/psecom,id,8,online,783,p,0.html

[2] Steinmaurer, Thomas: Firebird 2.0 Final, in: Entwickler Magazin 3.2007, S. 115

[3] www.firebirdsql.org/index.php?op=ffoundation

[4] www.janus-software.com/fb_fyracle.html

[5] entwickler-magazin.de/zonen/magazine/psecom,id,8,online,1145,p,0.html

[6] www.firebirdsql.org/index.php?op=devel&sub=engine&id=roadmap_2007&nosb=1

[7] entwickler-konferenz.de