Uhura, stellen Sie eine Verbindung her!
Massendaten zwischen Oracle und MySQL hin und her beamen

Martin Friemel
Tönisvorst

Schlüsselworte

MySQL, Oracle, Heterogenous Services, SQL*Net, ODBC, Database Link, PL/SQL, SQL, DML, Tuning

Einleitung

In dieser Session geht es um den Datenaustausch zwischen Oracle- und MySQL-Datenbanken. Die beiden unterschiedlichen Datenbanksysteme werden von Oracle aus mit der „Heterogeneous Connectivity“-Option verbunden.

Projekt

Anlass für den Betrieb der beiden unterschiedlichen Datenbanksysteme Oracle und MySQL ist ein Projekt der Alexander von Humboldt-Stiftung (www.humboldt-foundation.de).

Das Webangebot der Humboldt-Stiftung soll zu einem wissenschaftlichen sozialen Netzwerk ausgebaut werden mit professionellen Werkzeugen zur Vernetzung der teilnehmenden Wissenschaftler untereinander und zum Austausch fachlicher Themen.

Als Softwareplattform soll **elgg** eingesetzt werden, eine Open-Source-Social Networking Engine. Infos zu diesem System finden Sie unter http://elgg.org. Das elgg-System läuft am stabilsten auf einem Linux-Server. Es wird mit PHP entwickelt und erweitert. Als Datenbank wird MySQL verwendet. Der Linux-Server für den Betrieb des neuen Netzwerks soll bei einem externen Hosting-Anbieter stehen und somit außerhalb des internen Netzwerks betrieben werden.

Die internen Datenhaltungssysteme sind Oracle-Datenbanken auf Windows-Servern. Mit diesen Systemen muss das elgg-System Daten austauschen. Im Minutentakt sollen Benutzerdaten abgeglichen werden. Der Datenaustausch wird mit einem ETL-System realisiert, das auf einem eigenen Oracle-Server innerhalb des internen Netzwerks läuft. Das ETL-System wird in PL/SQL entwickelt. Es soll als Datendrehscheibe zwischen den internen Oracle-Datenbanken und der MySQL-Datenbank des elgg-Systems fungieren.

An dieser Stelle beginnt das Projekt interessant zu werden für die DOAG-Konferenz:

Wie koppelt man die internen Oracle-Datenbanken
mit der externen MySQL-Datenbank?

Wir haben uns entschieden, die PL/SQL-Sofware des ETL-Systems transaktionsgesichert über die Oracle Heterogeneous Services auf die MySQL-Datenbank zugreifen zu lassen. Die Datenübermittlung soll über eine getunnelte, verschlüsselte TCP/IP-Verbindung laufen.

Aus Sicht der PL/SQL-Software wird mit Hilfe der Heterogeneous Services in der Oracle-Datenbank

ein Database-Link zur MySQL-Datenbank eingerichtet. Nachfolgend werden die einzelnen Konfigurationsschritte beschrieben:

Kopplung mit Oracle Heterogeneous Services

Zuerst richten wir eine ODBC-Datenquelle ein, mit der wir vom Oracle-Windows-Server auf die MySQL-DB auf Linux zugreifen können.

Den benötigten ODBC-Treiber für MySQL laden wir auf der Seite http://dev.mysql.com/downloads/connector/odbc/ herunter:

Download: MySQL Connector / ODBC für Windows 32 bit.

Danach wechseln wir in die Systemsteuerung und richten dort eine im Karteireiter “System-DSN” Datenquelle ein:

- Name: NeOn
- TCPIP-Server: neon_dev (oder IP-Adresse)
- Port: 3306 (siehe ggfs. /etc/mysql/my.cnf auf dem Ubuntu-Server)
- User: elgg
- Password: elgg
- Database: neon_elgg

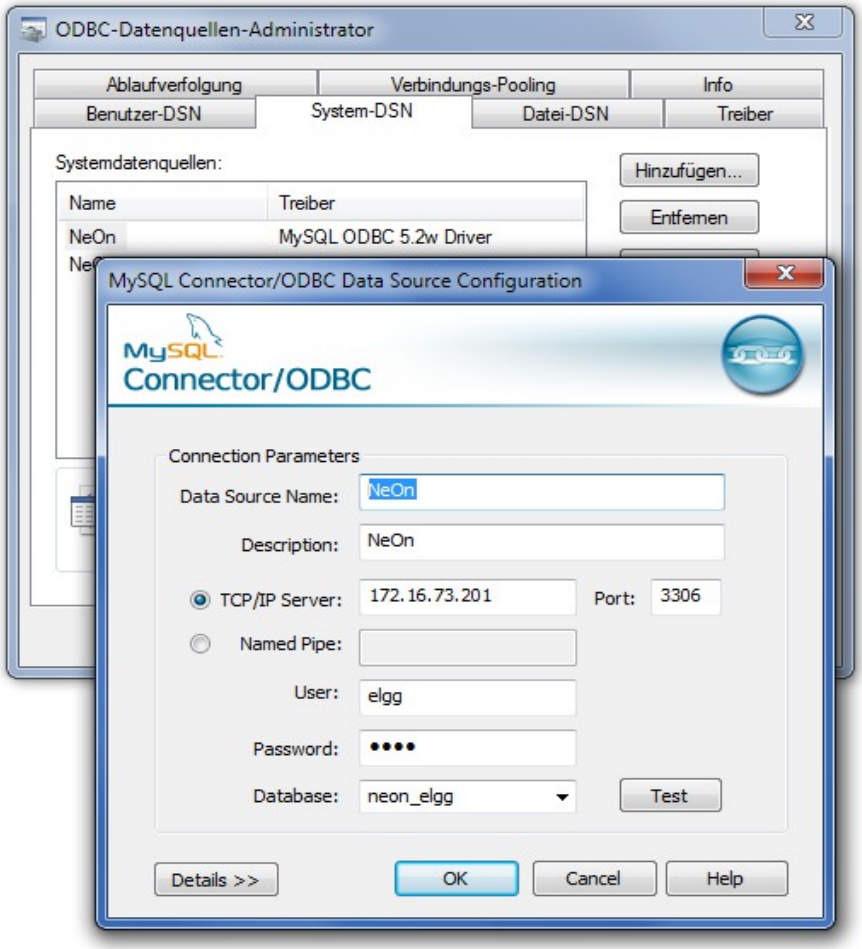

Abb. 1: ODBC Datenquelle einrichten

Oracle Heterogeneous Services

Die Verbindung zur ODBC-Datenquelle wird im Verzeichnis $ORACLE_HOME\hs\admin konfiguriert. Die erforderlichen Konfigurationsschritte sind:

Wir erzeugen eine neue Datei mit dem Dateinamen init**neon**.ora als Kopie der bereits vorhandenen Beispieldatei init**dg4odbc**.ora.

Inhalt der neuen Datei init**neon**.ora (abgesehen von Kommentaren):

```
HS_FDS_CONNECT_INFO = NeOnDatenquelle
HS_FDS_TRACE_LEVEL = 16
HS_LANGUAGE = German_Germany.AL32UTF8
```

Erklärung: Der Name **NeOnDatenquelle** im Parameter **HS_FDS_CONNECT_INFO** ist der Name der zuvor angelegten ODBC-Datenquelle.

Nun müssen noch einige Einträge in die SQL*Net-Konfigurationsdateien geschrieben werden:

Datei: $ORACLE_HOME\network\admin**listener.ora**

Einfügen im Abschnitt “SID_LIST_LISTENER =”

```
(SID_DESC =
  (SID_NAME = neon)
  (ORACLE_HOME=C:\app\oracle\product\11.2.0\dbhome_1)
  (PROGRAM=dg4odbc)
)
```

Erklärung: Der angegebene SID-name „**neon**“ entspricht dem Namensteil der zuvor angelegten Init-Datei im Verzeichnis hs/admin init**neon**.ora zwischen den Namensbestandteilen „init“ und „.ora“.

Datei: **tnsnames.ora**

```
NeOn =
  (DESCRIPTION=
    (ADDRESS=
      (PROTOCOL=TCP)
      (HOST=localhost)
      (PORT=1521)
    )
    (CONNECT_DATA=
      (SID=neon)
    )
    (HS=OK)
  )
```

Das war's. Jetzt wird der TNS-Listener neu gestartet und die Verbindung kann genutzt werden, um einen Database Link von Oracle aus zur MySQL-Datenbank anzulegen:

```
create database link neon
  connect to "my_username"
  identified by "my_password"
  using 'NeOn';
```

Username und Password sind in MySQL case-sensitiv, daher muss man sie im create-database-Statement in Gänsefüßchen im einfassen.

Nun können wir die neue Verbindung zur MySQL-Datenbank von Oracle aus testen:

```
select * from "NHT_NEON_INTERCHANGE"@neon where rownum < 10;
```

Performance

NHT_NEON_INTERCHANGE ist die Tabelle auf MySQL-Seite, in die unser ETL-System Benutzerdaten einfügen soll. Die ETL-Prozesse wurden in PL/SQL programmiert. Der erste naheliegende Versuch, Daten in die MySQL-Tabelle einzufügen, ist daher ein einfaches INSERT über den Database-Link:

```
INSERT INTO  "NHT_NEON_INTERCHANGE"@neon (
   (Spaltenliste)
```

```
)
VALUES (
   (Werteliste)
);
```

Die ETL-Logik entscheidet in einer Schleife, welche Sätze hinüber zur MySQL-Datenbank gesendet werden sollen. Mit dem o.a. INSERT war das quälend langsam – nur ca. 10 INSERTs pro Sekunde wurden ausgeführt.

Eine deutliche Steigerung der Geschwindigkeit wurde erreicht, indem das INSERT nicht mehr in der Oracle-Datenbank ausgeführt wurde, sondern als dynamisches SQL-Statement an die MySQL-Datenbank gesendet wurde, um erst dort ausgeführt zu werden.

Das Stichwort SQL-PASS-THROUGH kennt jeder ODBC-Programmierer. Wir können weiter in PL/SQL programmieren, denn Oracle unterstützt den Passthrough-Aufruf über unsere ODBC-Strecke – also über unseren Heterogenous-Services Database-Link - mit dem SYS-Package **DBMS_HS_PASSTHROUGH**.

Ein Blick in die Oracle-Doku verrät den Zweck dieses Packages:

„The DBMS_HS_PASSTHROUGH PL/SQL package allows you to send a statement directly to a non-Oracle system without being interpreted by the Oracle server. This can be useful if the non-Oracle system allows operations in statements for which there is no equivalent in Oracle.“

http://docs.oracle.com/cd/E11882_01/appdev.112/e25788/d_hspass.htm

Wir bauen unser INSERT also um: Statt es direkt in der PL/SQL-Engine auszuführen, generieren wir einen VARCHAR2-String, der das komplette INSERT mit allen Werten enthält. Dieser String wird mit dem DBMS_HS_PASSTHROUGH-Package an die MySQL-Datenbank gesendet:

```
l_hs_sqlrowcount :=
   DBMS_HS_PASSTHROUGH.execute_immediate@neon(s => l_hs_sql);
```

Trickreich: Man ruft die Function **execute_immediate** mit Angabe des MySQL Database-Link-Namens auf, gerade so als würde das Package DBMS_HS_PASSTHROUGH drüben in der MySQL-Datenbank liegen. Diese spezielle Syntax wird von Heterogeneous Services verarbeitet.

Performancegewinn: Mindestens die 10-fache Anzahl INSERTs gingen über die Leitung. Nicht schlecht, aber so richtig schnell sind 100 INSERTs pro Sekunde auch nicht. Es ist offenbar die Menge an einzelnen INSERTs, die so viel Zeit kosten, wenn sie über ODBC verschickt werden.

Der PL/SQL-Entwickler guckt über den Tellerrand

Was kann man noch tun, um die vielen INSERTs zu beschleunigen? Der Satz „*This can be useful if the non-Oracle system allows operations in **statements for which there is no equivalent in Oracle**.“* aus Oracle's DBMS_HS_PASSTHROUGH-Beschreibung fordert praktisch dazu auf, einmal einen Blick in die MySQL-Dokumentation zu schauen, oder?

Die Beschreibung des INSERT-Statements in MySQL findet man hier:
http://dev.mysql.com/doc/refman/5.7/en/insert.html

Dort steht weiter unten folgende interessante Syntax-Variante für MySQL-INSERT-Statements:

*INSERT statements that use VALUES syntax can insert multiple rows. To do this, include **multiple lists of column values**, each enclosed within parentheses and separated by commas. Example:*

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3),(4,5,6),(7,8,9);
```

Mehrere Wertelisten in einem INSERT-Statement. Das klingt pfiffig! Ehrlich gesagt, habe ich erstmal verstohlen ausprobiert, ob es das auch in Oracle-SQL gibt und ich das nur noch nie mitbekommen hatte. Geht aber nicht …

Vor dem DBMS_HS_PASSTHROUGH.**execute_immediate@neon** wird nun ein dynamisches INSERT-Statment generiert, bei dem mehrere Value-Listen mit Komma getrennt hintereinander gehängt werden.

Die Laufzeit sinkt proportional zur Anzahl der Wertelisten je INSERT, oder besser gesagt propotional zur Verringerung der Anzahl einzelner INSERT-Statements, die auf die ODBC-Strecke geschickt werden.

Das kosten wir aus: Bis zu 16 KB darf das INSERT-Statement lang sein. Das reicht, um die Anzahl der übergebenen Rows je INSERT-Statment so weit zu steigern, dass der gesamt ETL-Ablauf mit wunderbar zufriedenstellender Geschwindigkeit läuft!

REPLACE statt INSERT

Natürlich hat es Auswirkungen auf die Programmlogik, wenn man in der PL/SQL-Software mit einem INSERT-Statement mehrere hundert Sätze einfügt. Das gesamte INSERT-Statement bricht ab, wenn auch nur einer der Einzel-INSERTs auf MySQL-Seite scheitert. Darauf muss man dann in seinen EXCEPTION-Handlern achten.

Beim Stöbern in der MySQL-Dokumentation ist mir in diesem Zusammenhang ein interessanter DML-Befehl aufgefallen, der beim Abfedern dieses Problems helfen kann:

http://dev.mysql.com/doc/refman/5.7/en/replace.html

*„**REPLACE** works exactly like INSERT, except that if an old row in the table has the same value as a new row for a PRIMARY KEY or a UNIQUE index, the old row is deleted before the new row is inserted.“*

Ich kann also statt eines INSERT-Statements ein REPLACE-Statement mit gleicher Syntax ausführen. REPLACE fügt den Satz genau wie INSERT ein. Bereits vorhandene Sätze würde REPLACE ändern, statt eine DUP_VAL_ON_INDEX-Exception auszulösen.

Das ist sehr komfortabel für Programmierer. Auch die Syntax mit vielen Wertelisten je Statement klappt mit REPLACE.

Entwicklungs-Werkzeuge

Welche Tools nimmt man als PL/SQL-Entwickler, wenn man auch in MySQL-Datenbanken wühlt?

Für die PL/SQL-Entwicklung nehmen die meisten Entwickler TOAD oder den Oracle SQL Developer.

Ich habe für die Arbeit mit dem Database-Link zur MySQL-Datenbank mit TOAD gearbeitet, weil der Oracle SQL Developer einige male beim Zugriff über den Database Link abgestürzt war. Allerdings konnte ich solche Fehlersituation beim Schreiben dieses Vortrag nicht nochmal nachstellen, daher vermute ich, dass beide Tools für die Arbeit im Oracle/MySQL-Projekt geeignet sind.

Man braucht aber auf jeden Fall ein passendes Tool zur Arbeit direkt mit MySQL. Dabei fiel mir ein, dass es von DELL auch eine TOAD-Variante für MySQL gibt. Download der Freeware unter http://www.quest.com/toad-for-mysql/. Damit bin ich gut klar gekommen, allerdings hängt sich das Tool häufig beim Beenden des Porgramms auf und man muss es mit dem Taskmanager killen.

Stabiler funktioniert das offizielle Tool von MySQL, die MySQL Workbench. Die Bedienung ist für jeden Oracle-Entwickler leicht zu verstehen, die Unterschiede zu den bekannten Oracle-Entwickler-Tools sind gering.

Download: http://www.mysql.de/products/workbench/
Aktuelle Version: 6.0

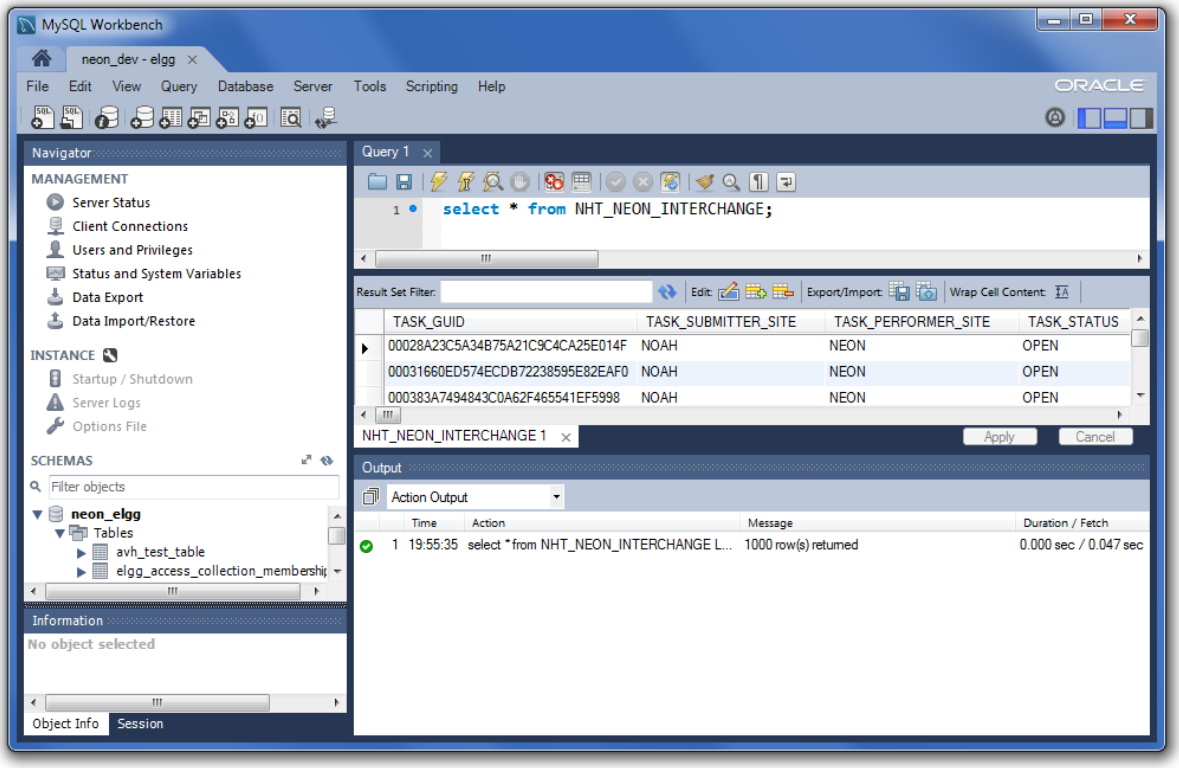

Abb. 2: Screenshot MySQL Workbench

Fazit aus Sicht des Oracle-Entwicklers

Die transaktionssichere Kopplung zwischen Oracle und MySQL läuft sehr stabil. Besonders schnell ist die ODBC-Strecke aber nicht, wenn man nicht alle Kniffe und Tricks ausschöpft.

Als Oracle-Entwickler sollte man sich ein wenig mit den MySQL-Besonderheiten beschäftigen. So ist das Locking-Modell der verschiedenen Storage-Engines für MySQL-Tabellen zu beachten. Nur die

relativ neue Storage-Engine InnoDB sperrt in Transaktionen auf Satzebene, wie wir es von Oracle her kennen.

Mehr Informationen zu den MySQL Storage-Engines:
http://dev.mysql.com/doc/refman/5.7/en/storage-engines.html

Wir haben in unserem Projekt daher darauf geachtet, dass wir von Oracle aus über den Database Link nur in eigene Tabellen schreiben und nicht in die Tabellen des elgg-Systems. Wir wollten uns auf diese Weise nicht in die Gefahr begeben, unkontrollierten Dead-Lock-Situationen zu begeben.

Kontaktadresse:
Martin Friemel
Rosenstraße 44 E
D-47918 Tönisvorst

Telefon:	+49 (0) 2151-935621
Fax:	+49 (0) 2151-935623
E-Mail	mfriemel@webag.com
Internet:	www.webag.com