MySQL als Grundlage der DMS-Software Archivista Version 5

Referat zu MySQL 4.1.10 anlässlich lots.ch, Bern, 18. Februar 2005

Inhaltsverzeichnis

1 Einleitung **2**
- 1.1 Copyright-Notiz 2

2 Happy landing bei MySQL **3**
- 2.1 Einige Vor- und Nachteile . 4

3 Businessmodell **6**
- 3.1 Quelloffenheit 6
- 3.2 OpenSource 6

4 Tuningszenarien **8**
- 4.1 LIMIT 8
- 4.2 MEMORY/HEAP-Tabelle(n) 8
- 4.3 Cache verwenden 8
- 4.4 Grosse Tabellen 9
- 4.5 BLOB-Felder von den übrigen Tabellen separieren . . 9
- 4.6 Volltext-Abfragen 9

5 Synchronisation von Datenbanken **10**

6 Langzeitarchivierung mit MySQL? **11**

7 Leistungsfähigkeit und Zugriff **12**
- 7.1 Passwortverwaltung von MySQL 12

8 Schlussfolgerung **14**

1 Einleitung

Dieser Vortrag richtet sich in erster Linie an Personen, die von MySQL gehört haben und vor der Entscheidung stehen, ob MySQL für die eigenen Bedürfnisse sinnvoll sein könnte oder nicht. Der Zeithorizont des Vortragenden beträgt gegen die fünf Jahre Erfahrung mit MySQL.

1.1 Copyright-Notiz

2 Happy landing bei MySQL

Vor etwa vier Jahren hatten wir bei einem Kunden eine äusserst seltsame Situation. Ok, Access mag nicht unbedingt eine Referenz darstellen; es darf aber auch gesagt werden, dass Archivierungslösungen ursprünglich nicht unbedingt Datenbank-Boliden voraussetzen. Zurück zum Problem: Der Kunde hatte in etwa 600'000 Belege, die Datenbank war nicht übermässig gross, die ganze Applikation lief zur vollsten Zufriedenheit aller Beteiligten, bis eines Tages unter einigen NT4-Maschinen die gleiche Datenbank urplötzlich um ein Vielfaches langsamer arbeitete. Konkret ging es um die BLOB-Felder von Access. Und da wir einen Supportvertrag mit Microsoft hatten, haben wir uns mit dem Problem an die entsprechende Support-Abteilung gewandt.

Ich möchte hier nicht der Versuchung erliegen, eine ganze Seite über den Microsoft-Support zu füllen. Zusammengefasst sei nur soviel gesagt, dass es beim letzten Telefonat hiess, ok, es könnte genau in dieser Konstellation tatsächlich ein Problem geben, allerdings könne Microsoft das nicht weiter verfolgen; wir sollten ganz einfach auf die neuste Version migrieren, dann wäre das Problem sicherlich behoben.

Letztlich konnten wir das Problem soweit einkreisen, dass der Kunde mit dem Performance-Verlust leben konnte/musste. Für uns war aber klar, dass eine neue Datenbank her musste. Nur welche? Sicher waren wir uns, dass die neue Datenbank in irgendeiner Form mit SQL ansprechbar sein sollte. Für die engere Auswahl blieben die Kandidaten Interbase (heute Firebird), IBM DB2, Oracle und MySQL übrig. Nicht näher betrachtet haben wir MS SQL, weil diese DB nur unter Windows lief bzw. noch immer läuft und auch PostgreSQL konnten wir damals nicht in die engere Wahl aufnehmen, weil es nur unter Unix/Linux lief (was heute nicht mehr der Fall ist).

Oracle erschien uns zu mächtig, abgesehen davon ist es uns nicht gelungen, die Endverkaufspreise schlüssig in Erfahrung zu bringen. Letztlich fiel die Wahl auf MySQL, weil MySQL sich in den Tests als stabil, schlank und schnell erwies. Richtig erinnert erschien MySQL auf unserer Watchlist mit der Version 3.22, wir waren uns bewusst, dass die Limite pro Tabelle bei 2 GByte lag; eine der ersten Erfahrungen damals mit Linux war schon auch, dass Ext2 bzw. die erhältlichen Distros bei 2 GByte fast allesamt Probleme hatten.

Zunächst haben wir versucht, via ODBC mit MySQL zu kommunizieren. Dies hätte für uns bedeutet, dass die Applikation praktisch nicht neu geschrieben hätte werden müssen. Allerdings hatten wir bereits bei ca. 1 Mio. Seiten (entspricht Anzahl Datensätze) um die 20 Sekunden Wartezeit beim Aufstarten der Applikation.

Daher war für uns klar, dass wir einen nativen Zugriff auf die Datenbank benötigten, auch wenn damit mehr Aufwand verbunden war und auch wenn wir uns damit weit über den Microsoft-Programmier-Gartenzaun hinauswagten. Die Programmierung selber war relativ aufwändig (ca. 12 Personen-Monate). Dies nicht in erster Linie, weil die Datenbank nicht stabil gewesen wäre, sondern, weil wir für die Abstraktion der nativen Schnittstelle zu unserem Produkt einige Hürden überwinden mussten.

Am Ende des Prozesses ist die Version 5.0 von Archivista entstanden, die zunächst einmal genau das bot, was wir mit der Verison 4.x als Produkt ebenfalls hatten. Verkaufstechnisch hatte dies vorallem den Nachteil, dass beim bisher einzigen kostenpflichtigen Update (in sieben Jahren!) die Kunden nicht unbedingt verstanden haben, was nun genau so neu an der Version 5.0 war.

2.1 Einige Vor- und Nachteile

Nachfolgend wollen wir einige Vor- und Nachteile auflisten. Vorteile wären da in etwa:

- Nur 30 KByte Datenfluss beim MySQL-Login, bei Access waren es immerhin ca. 300 KByte.
- Wesentlich kleinerer Performance-Bedarf, noch heute kann eine MySQL 4.x-Datenbank auf einem älteren Pentium-Rechner laufen.
- Echte Plattformunabhängigkeit, Access war unter Novell bzw. Samba nicht immer problemfrei.
- Wir können genau die MySQL-Version verwenden, die wir für sinnvoll halten.
- Der Kunde kann praktisch jede MySQL-Version verwenden, die er haben möchte (derzeit von 3.23 bis 4.1).

- Keine Datenverluste auf Tabellen; unter Linux hatten wir auch nie ein Problem mit den Index-Dateien, unter Windows 200x ca. einen Supportfall pro Jahr.

Wo es Vorteile gibt, gibt es wohl auch Nachteile:

- Ein gutes MySQL-Frontend war lange Zeit nicht erhältlich. Der Umweg über ODBC (meist mit Access) war eher eine Notlösung.
- Eine komplexe Abfrage kann die gesamte MySQL-Datenbank kurzfristig lahmlegen.
- Support, wenn er denn in Anspruch genommen wird, ist teurer, wobei die Feedbacks zumindest klar sind, auch wenn MySQL nicht brillieren sollte.
- Das Produkt MySQL stand vor ca. 2 Jahren an einem kritischen Punkt, als Nusphere MySQL relativ 'feindlich' zusetzen wollte.
- Gewisse SQL-Features (z.B. Views) vermissen wir bis heute, mit der Zeit kriegt jede/r Lust auf mehr.

3 Businessmodell

3.1 Quelloffenheit

An dieser Stelle möchte ich kurz auf das Businessmodell von Archivista zu sprechen kommen. Die Software Archivista ist entstanden, weil es vor ca. 10 Jahren keine Archivierungssoftware mit quelloffenen Datenstrukturen gab. Es gab mehere Erlebnisse, bei denen ich beruflich mit Situationen konfrontriert war, wo ich quellgeschlossene Systeme zu warten hatte. Es gab das Erlebnis, dass in einem Bundesbetrieb sämtliche elektronisch erfassten Text-Dokumente von ca. 5 Jahren nicht konvertiert werden konnten, weil kein Konvertierungsfilter vorhanden war. Die Fragestellung für mich lautete, können wir es uns leisten, all diese Dokumente fast schon mutwillig zu zerstören?

Die elektronische Archivierung setzt gerade hier ein, Dokumente werden so gesichert, dass der Zugriff später elektronisch sichergestellt ist. Die Information wird so gesichert, dass keine sichtbaren Verluste auftreten. In einem Word-Dokument fehlen bald einmal Querverweise, Bilder oder andere Elemente. Ok, was hat das alles mit einem Businessmodell zu tun? Ganz einfach, wäre es nicht toll, eine Archivlösung aufzubauen, die bei der Datenhaltung ganz auf offene Standards setzt? Was bringen neue geschlossene Dateiformate, ausfüllbare PDF-Formulare, die pro Klick in Rechnung gestellt werden? Quelloffene Formate verlangt nach einer Datenbank und Datenstrukturen, die vollkommen dokumentiert sind, OpenSource dürfte hier sicherlich das Optimum darstellen.

3.2 OpenSource

Und weshalb nicht gleich ganz OpenSource? Wohl keine ketzerische Frage an einem Anlass wie lots.ch. Historisch bedingt ist Archivista auf der Windows-Plattform entstanden. Im Bereich des Dokumentenscannings ist Windows im Moment Linux leider noch überlgen, die Axis-Scanboxen mal ausser acht gelassen[1]. Ein weiteres Feld betrifft die OCR-Texterkennung; es gibt bis heute leider keine echte Alternative auf Linux-Basis. Als Anbieter einer Archivierungssoftware können wir immerhin eine OCR-Erkennung (FineReader) unter Linux anbieten, die keine Seitenbegrenzungen enthält. Ok, was hat das mit MySQL zu tun? Vielleicht, dass OpenSource sich bislang

[1] www.axis.com

dort durchgesetzt hat, wo eine gewisse Breite der Anwendung gegeben ist. Alle Welt braucht ein Betriebssystem, eine Datenbank und ganz viele auch eine Archivierungslösung. Und jeder Schritt in Richtung quelloffene Software benötigt auch Zeit.

Wir haben vor ca. drei Jahren begonnen, konsequent für Linux und Windows zu programmieren, ob wir das ohne MySQL auch getan hätten, wer weiss. Wir haben heute die gesamte Software auf Linux lauffähig, d.h. nun nicht, dass wir je 100 Prozent OpenSource haben werden. Immerhin haben wir aber einen zentralen Teil unter OpenSource, eine komplette Lösung für selbstragende Archive. MySQL bildet zudem das Rückgrat mit dem Seitenblick, dass wir mittlerweile auch andere OpenSource-Datenbanken bedienen könnten. MySQL hat so gesehen unser Denken aus einer monolithischen in eine pluralistische Datenbank- und EDV-Welt geöffnet.

4 Tuningszenarien

Dieses Referat erhebt nicht unbedingt den Anspruch, technische Hilfestellung zu geben. Der Vortragende wird dies vielleicht zu anderer Zeit nachholen, nicht aber aus dem Stegreif nach einer anstrengenden Arbeitswoche.

Nachfolgend möchte ich einige Punkte anführen, die der Performanz i.d.R. dienlich sind:

4.1 LIMIT

Im Prinzip bin ich versucht zu sagen, eine Abfrage ohne LIMIT ergibt keinen Sinn. Ganz sicher aber führen z.B. Volltextabfragen ohne Antowrtbegrenzung dazu, dass jeder Server relativ schnell lahm wird. Ok, übergrosse Abfragen können mit Parametern abgefangen werden, ich würde aber trotzdem LIMIT empfehlen wollen. Mit dem kleinen Zusatz vielleicht, dass es so implementiert werden sollte, dass später im Code nicht Tausende 'LIMIT'-Fragmente hängen, eine einzelne Variable würde auch genügen. Nicht nur, weil weniger Fehler entstehen, sondern auch damit später mühelos auf eine andere Datenbank geswitcht werden kann.

4.2 MEMORY/HEAP-Tabelle(n)

Daten, die nur temporär benötigt werden, z.B. Session-Informationen können ohne Probleme in einer MEMORY/HEAP-Tabelle abgelegt werden. Die Tabelle wird allerdings zerstört, sobald der Server downgefahren wird.

4.3 Cache verwenden

Mit den cache-Variablen **`set global query_cache_size = xxx`** kann ein Speicherblock reserviert werden, der als Zwischenspeicher für Abfragen verwendet werden kann. Die gleiche Abfrage benötigt zwar nicht weniger Zeit beim ersten Aufruf, aber keine weitere Zeit bei der zweiten Anfrage; selbstverständlich natürlich nur, solange der Cache nicht durch spätere Abfragen überschrieben wird.

4.4 Grosse Tabellen

Mit **max_length** und **max_rows** können die Anzahl der Länge eines Datensatzes und die maximale Anzahl der Datensätze festgelegt werden. Wozu das alles? Wir benötigen diese Werte, um bei sehr grossen MyISAM-Tabellen die Grenze von 4 GBytes zu überschreiben. Sofern dies notwendig ist, geben wir ein: **alter table xxx type myisam max_rows = 10000000**, um z.B. 1 Million Datensätze zu ermöglichen.

4.5 BLOB-Felder von den übrigen Tabellen separieren

BLOB-Felder sollten nach Möglichkeit in eine eigene Tabelle mit einem Zeiger auf die Haupttabelle gelegt werden. Nach dem Inhalt von BLOB-Feldern wird wohl eher nicht gesucht werden (Fulltext-Felder sind TEXT-Felder, siehe unten).

4.6 Volltext-Abfragen

MySQL ist seit der Version 4.x in der Lage einen akzeptablen Volltext zu erstellen. Akzeptabel bis gut sind dabei Erstellungszeit und Abfragegeschwindigkeit grösserer Indexdateien (z.B. mehrere GBytes). Bei Volltextabfragen sollte auf 'IN BOOLEAN MODE' zurückgegriffen werden, weil die 'normale' Volltextengine bei grösseren Abfragen langsamer ist und ohnehin nur Treffer zurückgibt, sofern nicht zuviele Treffer (mehr als die Hälfte) vorhanden sind. Das ist unschön und auch etwas unpraktisch, wenn ich immer zunächst noch nachschauen muss, ob jetzt zuviele, also keine, Treffer vorhanden sind.

Dazu ein Beispiel (im Volltextfeld 'Text' wird nach dem Suchbegriff 'Seite' gesucht):

```
select * from seiten where match Text against('Seite' in boolean mode);
```

5 Synchronisation von Datenbanken

Das Synchronisieren von Datenbanken steht interessanterweise gerade bei vielen von unseren Kunden auf dem Programm, die an sich gar nicht so grosse Datenbestände (z.B. 50 MByte) haben. Meist geht es dabei darum, das Notebook mit dem Tischgerät abzugleichen. Das Abgleichen von Datenbanken, d.h. die klassische Master-Slave-Konstellation (Hauptrechner=Master, Nebenrechner=Slave) ist an sicht nicht so schwierig.

Wer gar einen Blick in die Datei 'my.cnf' macht, wird die Lösung fast schon pfannenfertig in den Händen halten. Zunächst muss beim Master die ID=1 gesetzt werden.

```
server-id = 1
```

Danach können wir auf dem Slave ebenfalls my.cnf anpassen:

```
server-id = 2
master-host = <hostname>
master-user = <username>
master-password = <password>
master-port = <port>
log-bin
```

Anschliessend starten wir die beiden Server neu, das Gespann wäre nun zusammen.

6 Langzeitarchivierung mit MySQL?

Die Frage müsste vielleicht so lauten, eignet sich überhaupt eine Datenbank für die Archivierung? Wie können wir grosse Datenbestände in Datenbanken langfristig speichern? Pragmatisch betrachtet würde ich nicht allzu hohe Erwartungen an die Langzeitarchivierung (10 bis 30, 50 oder noch mehr Jahre haben). Mag sein, dass wir einige Datenbanken im Museum noch antreffen werden[1], damit ist aber noch nicht gesagt, dass es einfach sein wird, auf diese Daten zuzugreifen. Und wer kann heute eine verlässliche Aussage über das langfristige Verfügbarhalten elektronischer Daten abgeben?

Was für die Lanfristigkeit von MySQL spricht ist, dass die Sourcen vorhanden sind, dass die Lauffähigkeit unter vielen Betriebssystemen gegeben ist und dass MySQL mit den Ressourcen sparsam umgeht. Ebenfalls dafür spricht, dass jederzeit ein SQL-Dump einer Datenbank (mysqldump) gemacht werden, wobei die Ausgabe wahlweise in ASCII, XML oder HTMl erfolgen kann.

Dagegen spricht wohl, dass MySQL mittlerweile eine stattliche Anzahl von Tabellentypen unterstützt, dass verschiedenste Zeichensätze (charsets) vorhanden sind und das mit MySQL 4.1 zwar Unicode Einzug hält, allerdings (aus Performance-Gründen zwar nachvollziehbar) auf Stufe Spalte, womit das Zeichensatz-Chaos wohl früher oder später entstehen dürfte.

[1] MySQL als OpenSource-Datenbank-Leader dürfte den Sprung bestimmt schaffen, betr. Archivista würde ich erhoffen, dass wir vor der Museumstüre im Gebrauch blieben

7 Leistungsfähigkeit und Zugriff

Es ist schwierig, eine gute Aussage zur Leistungsfähigkeit von MySQL zu machen. Sicher ist, dass die MyISAM-Tabellen wesentlich schneller sind als InnoDB-Tabellen, doch das hängt mehr damit zusammen, dass MyISAM nicht transaktionsfähig ist, während dies bei InnoDB gegeben ist. Aus eigenen Erfahrungen kann gesagt werden, dass Datenbankgrössen von mehreren Dutzend GBytes keine nennenswerte Schwierigkeiten verursachten. Das mag nach wenig erscheinen, verglichen mit den 2 GByte bei der Version 3.22 ist es aber doch eine ganze Menge mehr. Eine Volltextabfrage über ca. 6 GByte auf einem alten Duron 800 MHz Rechner benötigt i.d.R. ca. 1 bis 3 Sekunden, wer es nicht glauben möge, darf unter Books4free, unserem Testrechner für MySQL-Datenbanken gerne selber nachsehen. Die Webpage lautet: www.books4free.org.

Der Zugriff auf MySQL-Datenbanken kann über praktisch sämtliche Sprachen erfolgen. Ein nativer Zugriff via libmysql.so oder libmysql.dll bringt mehr Performance, aber auch mehr Arbeit.

7.1 Passwortverwaltung von MySQL

Die Passwortverwaltung von MySQL ist kein einfacher Brocken. Es wird zunächst unterschieden zwischen einem globalen Zugriff (user-Tabelle), dem Zugriff nach Datenbanken, Tabellen und Spalten sowie einem dritten über den Host.

Derzeit nicht möglich ist eine Zugriffsbeschränkung auf Stufe Datensatz (Views), diese Funktion ist erst für die Version 5.x vorgesehen; es darf davon ausgegangen werden, dass es noch ein zwei Jahre dauern wird. Bis dahin können Views nur simuliert werden, das geht z.B. mit einem Eigentümer-Feld und verschlüsseltem Login; nicht gerade schön, aber immer noch besser als gar kein Zugriffsschutz auf Stufe Datensatz.

Ebenso (und fast schon ärgerlich) ist die Tatsache, dass MySQL die Benutzer global je Server verwaltet. Soll also 'Otto' nur in 'friesen' surfen dürfen, nicht aber in der ch-orte-Datenbank, so bedarf es einer Menge Arbeit, das entsprechend hinzukriegen. Oder, um es anders zu sagen, wir hatten doch einige Supportfälle, weil einige Anbieter einfach global Rechte setzen bzw. auch wieder entfernen. Eine eigene (optionale) Benutzerverwaltung je Datenbank(en) wäre

sicher ein Vorteil, ein müdes Lächeln eines jeden PostgreSQL-Menschen wird uns hier sicher sein.

8 Schlussfolgerung

MySQL hatte zu der Zeit, als wir auf die Datenbank aufmerksam wurden, nur wenig Beachtung gefunden. Wenn wir von langfristiger Archivierung mit MySQL gesprochen haben, so wurde uns dann und wann mangelnde Seriosität unterstellt. Dies ist heute definitiv nicht mehr der Fall.

MySQL hat sich etabliert. Der Entscheid, sich für MySQL zu engagieren, hat sich für uns gelohnt. Während andere Anbieter in vier bis fünf Jahren bereits zwei bis drei Migrationen durchführen mussten, können wir selbst heute die erste von Archivista Version 5.0 unterstützte MySQL-Version, die auf der Datenbank 3.23 beruhte, problemlos unterstützen. Unsere Kunden sind frei, welche Version sie verwenden möchten.

MySQL gibt uns und unseren Kunden sehr viel Flexiblität. Und während diese am Anfang z.T. auf der Konsole erarbeitet werden musste, stehen heute grafische Frontents[1] zur Verfügung. Dabei können z.B. Bilder in BLOB-Felder direkt betrachtet werden, womit eine weitere Möglichkeit verraten wäre, wie Archivista-Daten abgefragt werden können.

[1] siehe www.mysql.com und www.knoda.org

Kontakt: Archivista GmbH, Postfach, CH-8042 Zürich

Tel: +41 (0)1 254 54 00, Fax: +41 (0)1 254 54 02, Web: www.archivista.ch