MySQL-Umgebungen absichern mit Standby Datenbank

Franz Diegruber
Libelle AG
Stuttgart

Schlüsselworte
MySQL, Standby Datenbank, Desaster Recovery, RTO, RPO, RCO

Einleitung

Die Datenbank bildet die Hauptschlagader, häufig sogar das Herz des Geschäftsbetriebes. Wie leistet man aber erste Hilfe, wenn es zum Infarkt kommt? Welche Maßnahmen der Reanimation wirken?
Dieser Vortrag zeigt anhand BestPractices und Betriebserfahrung mittelständischer und großer Unternehmen, wie MySQL-Systeme unterschiedlichster Komplexität systemübergreifend, vollständig und ganzheitlich abgesichert werden können. Zielsetzungen der hier vorgestellten Ansätze und Projekte waren die Definition und Umsetzung unterschiedlicher Verfügbarkeits- und Desaster-Recovery-Szenarien hinsichtlich RTO, RPO und RCO.
Was ist zu tun, damit alles bereit ist für den Ernstfall?
Eine weitere Komplexitätsstufe tritt auf, wenn aufgrund von Verknüpfungen Datenbanken voneinander abhängen.
Bedacht werden sollte auch die Kommunikationsebene: Virtuelle IP-Adressen und Hostnames sorgen dafür, dass nach dem Umschalten auf das Ausfallsystem die bestehenden Zugriffskanäle weiterhin sauber funktionieren.

Gründe für Standby Datenbank

Denkt man an die Beweggründe ein Standby-System an einem entfernten Standort aufzubauen, kommen häufig zuerst die Desastervorsorge gegen Brand, Wasserschäden oder auch Terror in den Sinn. Daneben schreiben oft auch betriebliche Bestimmungen oder gesetzliche Vorgaben eine Spiegelung an einen räumlich getrennten Standort vor. Zu nennen sind hier beispielsweise **Basel II** und **SOX** (Sarbanes-Oxley Act).
Zusätzlicher Nutzen kann aus dem Standby-System für die Zentralisierung operativer RZ-Aufgaben wie z.B. **Reporting und Backup** gezogen werden. So gibt es den Ansatz von den Zweigstellen in ein zentrales Rechenzentrum zu spiegeln und dort das Reporting oder das Backup durchzuführen. Mit dem Backup vom Standby-System bietet sich die Möglichkeit ein offline-backup zu fahren und dennoch für die Produktion einen 7x24-Stunden-Betrieb zu gewährleisten. Ein Standby-System kann beispielsweise auch für die Erzeugung von **Systemkopien für Entwicklungs- oder Testzwecke** ohne Beeinflussung des Produktivbetriebes herangezogen werden.
Auch im temporären Einsatz, z.B. bei **Umzugs- und Migrationsprojekten**, macht es häufig Sinn mit WAN-Spiegelungen auf Basis von Standby-Systemen zu arbeiten, da hierbei durch eine definierte Umschaltung lediglich minimale Betriebsunterbrechungen auftreten.

Neben der Datenbankebene gilt es jedoch auch die Applikationsebene zu betrachten. Zahlreiche Applikationen schreiben im laufenden Betrieb umfangreiche **Daten außerhalb der Datenbank**. Diese Applikationen sind nur dann mit einem aktuellen Stand auf dem Standby-System lauffähig, wenn diese Daten in das Ausfallrechenzentrum übertragen werden. Hier sind z.B. Profilefiles, Transport- und Schnittstellenverzeichnisse etc., sowie die eigentlichen Programm-Binaries zu nennen. Aus diesem Grund ist es in vielen Fällen nicht ausreichend auf eine reine Datenbankspiegelung zurückzugreifen. Es bedarf eines integrierten Ansatzes zur ganzheitlichen Applikationsspiegelung.

Hierfür bietet sich neben der individuellen Entwicklung von Skripten auch die Möglichkeit auf den Einsatz entsprechender Produkte von Drittanbietern, wie beispielsweise der Firma Libelle.

Die gesamte Kommunikation zwischen Clients, Applikationsserver und Datenbankserver findet über eine spezifizierte IP-Adresse, Hostname oder NetBIOS-Name statt. Wird eine Datenbank oder ein Filesystem von einem Server auf einen anderen umgeschaltet, so ist die Anwendung normalerweise nicht in der Lage diesen anderen Server automatisch zu identifizieren. Generell setzt eine Umschaltung auf einen anderen Datenbankserver mehrere manuelle Eingriffe in der Anwendung (z.B. SAP R/3-Profiles) voraus. Dies führt zu längeren Downtimes für die Applikation und Anwender.
Ist die Möglichkeit gegeben, dass im Notbetrieb über eine Virutelle IP-Adresse (**VIP**) die bisherige Konnektivität wieder verfügbar gemacht werden kann, sind keine Anpassungen für den connect notwendig. Dennoch kann es für die Umschaltung erforderlich sein, Prozesse auf Applikationsservern im Netzwerk durchzustarten. Dies kann über Agents oder Skripte auf den Applikationsservern umgesetzt werden, welche bei der Umschaltung angetriggert werden.

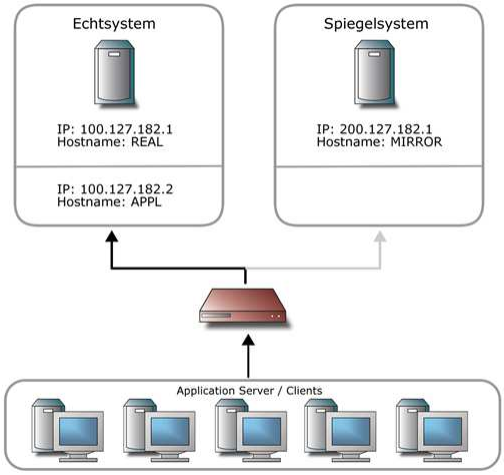


Abb. 1: Applikationszugriff über VIP

Laut Studien lassen sich etwa 70% aller Ausfälle auf **logische Fehler** zurückführen. Unter logischen Fehlern sind **Datenkorruptionen oder -löschungen** aufgrund von Benutzer- oder Softwarefehlern zu verstehen. Nur durch ein zeitversetztes Recovery kann auf diese reagiert werden. Deshalb sollte es das Ziel sein die Transaktionen mit einem möglichst geringen Zeitversatz auf das Standbysystem zu übertragen, dort aber mit einem Zeitversatz gegen die Standby-DB zu applizieren.

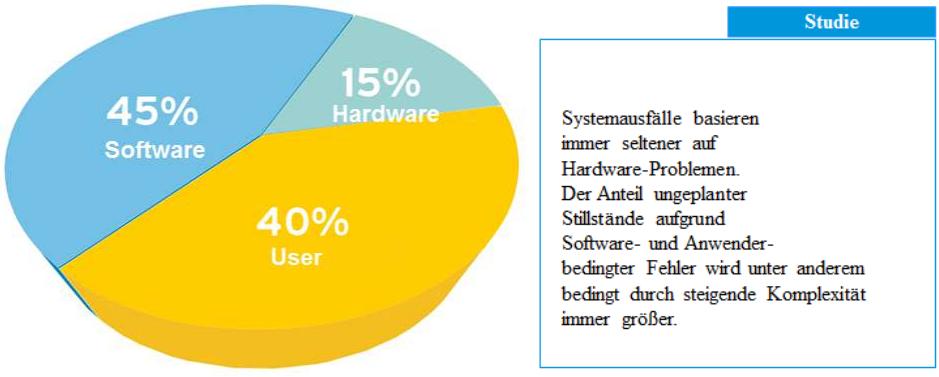


Abb. 2: Ursachen für Systemausfälle

MySQL - technische Übersicht
Eine der wichtigsten Unterschiede von MySQL zu kommerziellen Datenbank-Systemen ist die Möglichkeit, **verschiedene Storage Engines** zu benutzen. Diese Storage Engines beeinflussen nicht nur die rein physische Speicherung der Tabellen, sondern können auch eine **große Wirkung** auf Eigenschaften wie **Backup, Locking und Transaktions-Handling** haben. Je nach Typ der Applikation kann man bestimmen, wie die Daten für eine bestimmte Tabelle physisch abgelegt werden. Hier können Fragen berücksichtigt werden wie:

- Braucht die Tabelle einen Transaktions-Support?
- Handelt es sich im Wesentlichen um eine Read-Only-Tabelle?
- Werden die Daten einer Tabelle parallel von vielen Benutzern aktualisiert?
- Werden Daten periodisch mit Batch-Jobs geladen?

Je nach Antwort kann eine adäquate Storage Engine ausgewählt und so der Ressourcenverbrauch möglichst klein gehalten werden. Wird eine Tabelle zum Beispiel kaum modifiziert oder werden deren Daten nur periodisch von einer Ladeprozedur aktualisiert, dann ist die MyISAM Storage Engine die perfekte Wahl. Sind parallele Updates zu erwarten oder ist Transaktions-Support notwendig, dann sollte in der Regel InnoDB genutzt werden.
Weitere Engines sind "Memory" (für volatile Daten, die einfach wiederhergestellt werden können), "Merge" (für partitionierte Tabellen, aber deutlich weniger leistungsfähig als zum Beispiel "Oracle Partitionen"), "BDB" (Berkley Database), "NDB Cluster" (für MySQL-Cluster) und "Federated" (für den Remote-Database-Access, ähnlich einem "Oracle Database Link", aber für nur eine Tabelle).

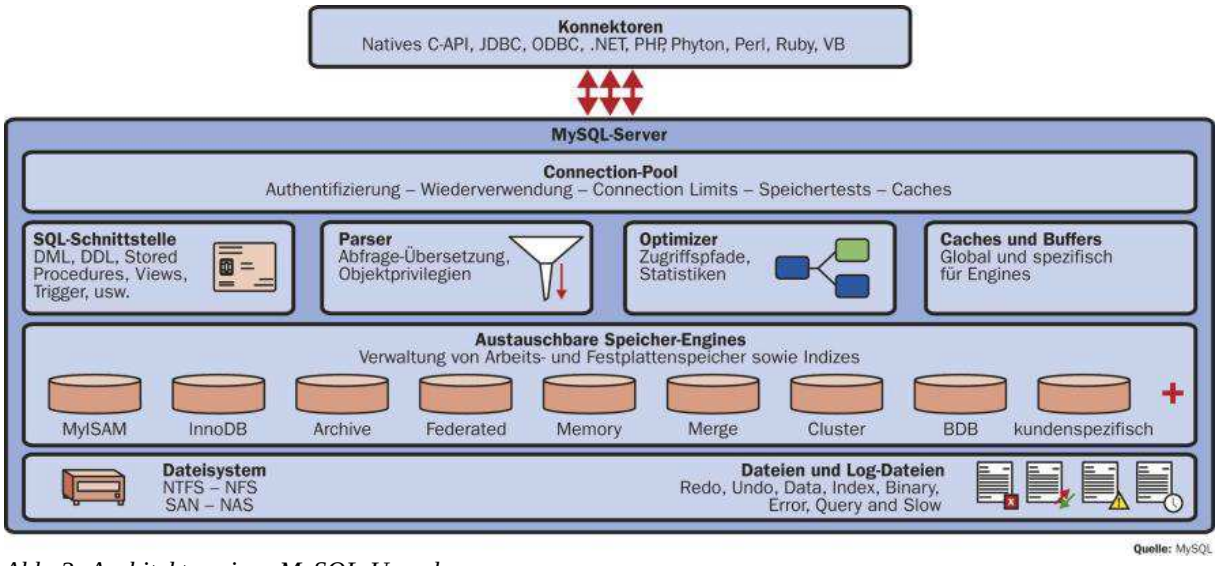


Abb. 3: Architektur einer MySQL-Umgebung

MySQL Community Server (Standard Edition)
ist die frei verfügbare Version der weltweit populärsten open source Datenbank.

MySQL Enterprise Edition
ist die kommerzielle Editon des MySQL welche folgende Kompoenten beinhaltet:

- MySQL Database
- MySQL Enterprise Backup
- MySQL Enterprise Monitor
- MySQL Workbench Standard Edition

Kleine Historie
Im Januar 2010 wurde Sun Microsystems von Oracle gekauft.
Ende 2010 wurde MySQL 5.5 veröffentlicht. InnoDB wurde zur Standard-Speicherengine.
Im Jahr 2012 wurde die MySQL 5.6 veröffentlicht.

Mit **MySQL 5.6** werden einige praktische Funktionen eingeführt:
- High performance
- High availability
- Data integrity
- Replication Performance
- Multi-Threaded Slaves
- Binary Log Group Commit
- Optimized Row Based Replication
- Failover and Recovery
- Mysqlfailover
- Mysqlrpladmin
- Crash Safe Slaves and Binlog
- Data Integrity - Replication Event Checksums
- MySQL Replication Utilities
- Time Delayed Replication
- Remote Binlog Backup
- Server UUIDs
- Informational Log Events

Administration
Unter Linux installiert sich MySQL in das Verzeichnis /var/lib/mysql/.
Unter Windows legt der Nutzer den Ablageort der Datendateien fest - Standard ist der Ordner %ProgramFiles%\MySQL.
Grundeinstellungen werden durch den Administrator in der Datei **my.cnf** (Linux) bzw. **my.ini** (Windows) vorgenommen.
Zur Verwaltung von MySQL-Datenbanken dient der mitgelieferte **Kommandozeilen-Client** (Kommandos mysql und mysqladmin). Zum Funktionsumfang gehören außerdem die folgenden Kommandozeilenwerkzeuge (Binaries (tool etc.) unter <Grundverzeichnis>/bin):
- mysqlimport
- mysqldump
- perror: zeigt zu Fehlercodes erweiterte Informationen an. Als Parameter wird beim Programmstart der Errorcode benötigt.
- mysqlshow: gibt Metadaten zu Datenbanken, Tabellen oder einzelnen Tabellenspalten aus.

DB-Version ermitteln
```
mysqld -V
```

Liste der DBs ermitteln
```
mysql  -uroot -pgeheim -e "SHOW DATABASES"
```
=>Database\nDatabase1\nDatabase2...

DB-Dateien ermitteln
Grundverzeichnis mittels mysql –e “SHOW VARIABLES LIKE ’basedir’ “
Datenverzeichnis mittels mysql –e “SHOW VARIABLES LIKE ’datadir’ “

Parameter anzeigen mit denen die DB läuft
```
mysqladmin variables -uroot –pgeheim
```

DRBD und Cluster
Neben der MySQL Replikation gibt es mit DRBD und Cluster noch weitere Möglichkeiten eine Ausfallsicherheit aufzubauen. Diese haben aber die Zielsetzung der **High Availability** (HA). Deshalb wollen wir uns über diese nur kurz informieren.

Distributed Replicated Block Device (DRBD)
- Synchrone Block-Replication zwischen Aktivem und Passivem DRBD Server auf Storageebene.
- Automatische Resynchronizierung nachdem Server wieder verfügbar
- Kein zeitversetzes Applizieren der Transaktionen
- Die Distributed Replicated Block Device (DRBD) ist ein Linux Kernel module das auf einem verteilten storagesystem basiert.

MySQL Cluster
- ermöglicht die Installation von MySQL auf einem Computercluster in einer **Shared Nothing Architecture**.
- MySQL Cluster ist eine Speicher-Engine des freien Datenbanksystems MySQL in der aktuell verfügbaren Version 7.2.
- MySQL Cluster ist für den schnellen, immer verfügbaren Zugriff mit hohem Durchsatz designed worden.

Einsatzgebiete des Clusters
MySQL Cluster wird oft als DBMS im Web-Umfeld eingesetzt, wo es darauf ankommt, dass **viele Lesezugriffe schnell ausgeführt** werden und dass eine **hohe Ausfallsicherheit (HA)** gewährleistet wird. Für solche Anforderungen hat MySQL Cluster bei Tests schon bessere Zugriffszeiten bewiesen als Oracle, DB2 und MS SQL.
Die Cluster-Technik von MySQL zeichnet sich durch das **Shared-Nothing-Konzept** aus, das rasche Antwortzeiten bei Lesezugriffen garantiert, weil alle Daten im Hauptspeicher gehalten werden (können). Dieses Konzept hat allerdings einen Overhead bei Schreibzugriffen zur Folge, insbesondere wenn viele Knotenrechner in einer Server-Farm koordiniert werden müssen.

Übersicht über die Replikation einer MySQL-DB
Bei einer Oracle Datenbank können „einfach“ die generierten Transaktionsprotokolle an das Standby-System gesendet werden. Bei MySQL gibt es auch Storage Engines ohne Transaktionsverhalten. Deswegen muss hier das Vorgehen etwas anders sein.
Bei MySQL handelt es sich um generische Transaktionsprotokolle (das **Binary Log**). In diesem binären Logfile werden alle Statements protokolliert, die Änderungen an den Daten vornehmen (DML, DDL, DCL).
Der Slave kopiert beständig das Logfile (das **Relay Log**) zu sich lokal und extrahiert sich aus dem Logfile wieder SQLStatements und spielt diese gegen den eigenen Datenbestand ein (**SQL-Thread**).

Die Log-Dateien sind in einem platzsparenden Binärformat, können aber mittels **mysqlbinlog** dekodiert werden.
Im Wesentlichen besteht ein Logfiles aus MySQL-Kommandos, die sich direkt in einen Client einspeisen lassen. Zudem enthalten sie für jedes Kommando weitreichende Meta-Informationen wie z.B. den Zeitpunkt des Befehls und die Durchführungszeit. Beispiel:

```
#120823   9:35:36 server id 1   end_log_pos 108     Query     thread_id=6
exec_time=0 error_code=0
```

```
SET TIMESTAMP=1193384136/*!*/;
insert into cluster values ("Max"),("Moritz")/*!*/;
```

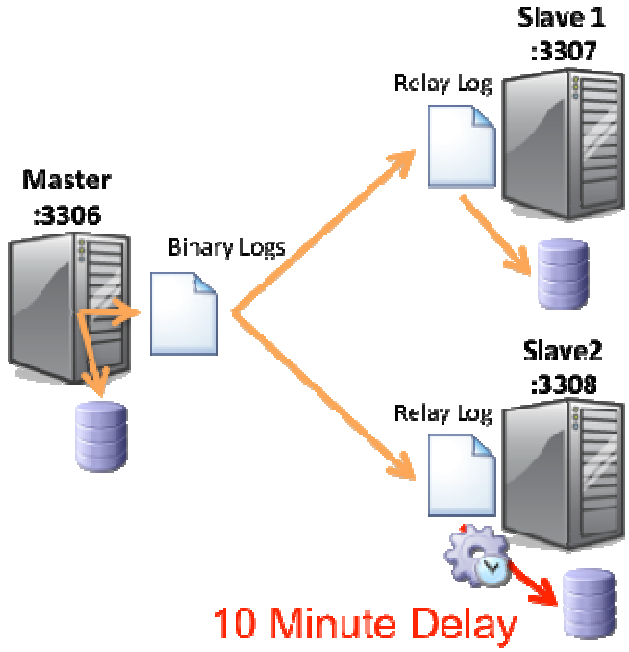
In der **ersten Zeile** befinden sich Datum, Uhrzeit, Server-ID, etc.
In der **zweiten Zeile** wird der Zustand des Servers so angepasst wie er zur ursprünglichen Ausführung des Befehls war.
In der **dritten Zeile** steht schließlich der Befehl.
Die **Index Datei** enthält eine **Liste** der bisher angelegten Log-Files (**kompletter Pfad**).

Einsatzgebiete Replikation

- Es können auch 1 (Master) : N (Slaves) Konstrukte aufgebaut werden. Die Slaves stehen für lesende Zugriffe zur Verfügung. So kann auch eine Lastverteilung mittels Slaves verfolgt werden.
- Somit kann der Slave auch für Reporting genutzt werden.
- Desweiteren können die Slaves für das Backup genutzt werden. Somit kann die Last der Backups auf den Slave ausgelagert werden.
- Natürlich steht der Slave auf für ein HA oder DR Konzept zur Verfügung.

MySQL Absicherung mit Replikation

Grundsätzlich läuft die Replikation asynchron. Mit der Version 5.5 wurde die **semisynchrone Replikation** eingeführt. Hierbei muss die Transaktion innerhalb eines Timeouts auf mindestens einen Slave übertragen worden sein. Kann dies nicht durchgeführt werden (z.B. Netzwerkfehler), wird auf asynchrone Replikation gewechselt.
Mit der Version 5.6 wurde die Möglichkeit geschaffen die SQLs auf dem Slave zeitversetzt einzuspielen. Dies wird über ein commando „change master“ auf dem Slave durchgeführt. Bei mehreren Slaves können so **unterschiedliche „recover delays“** definiert werden.



Quelle: MySQL

Abb. 4: SQL-Apply mit Zeitversatz

MySQL-Replikation einrichten
Binary Logs werden angelegt, sobald der Server mit dem Parameter „**--log-bin**“ gestartet wird.
Ein neues Logfile wird von mysqld stets dann angefangen, wenn das alte eine bestimmte Größe erreicht hat. Man kann den Server allerdings auch dazu zwingen, das momentane Logfile abzuschließen und ein neues zu beginnen, indem man den Befehl „FLUSH LOGS“ ausführt.

```
mysqld --e “FLUSH LOGS“
```

Die Dateien des Binary Logs befinden sich im Datenverzeichnis des MySQL-Server (unter Linux standardmäßig /var/opt/mysql/) und werden nach folgender Konvention benannt:
<hostname>-bin.<fortlaufende Nummer>
Der Index befindet sich ebenfalls in diesem Verzeichnis und heißt:
<hostname>-bin.index
Achtung: Es kann auch manuell ein anderer Präfix angegeben werden, indem man dem --log-bin Parameter einen String übergibt, also --log-bin=<Präfix>
Auch die Index-Datei kann mittels des Parameters –log-bin-index=<Dateiname> frei gewählt werden.

Zudem muss jeder Server (Master und Slaves) eine **eindeutige ID** bekommen über die Variable „**server-id**“.

Diese beiden Parameter müssen im Parameterfile my.cnf bzw. my.ini auf jedem beteiligten Server definiert werden. Nach dem Definieren ist ein restart der DB notwendig.

```
[mysqld]
datadir=/var/lib/mysql
log-bin=/var/log/mysql/mysql-bin.log
#binlog-do-db=test
server-id=1
```

In die Konfiguartion des Slave muss zusätzlich der Parameter „replicate-do-db=<DB die repliziert werden soll>“ definiert werden.

Anschliessend muss auf dem Master und Slave der entsprechende User angelegt werden, mit welchem der oder die Slaves auf den Master zugreifen sollen.

```
CREATE USER 'replicationusr'@'192.168.1.%'
IDENTIFIED BY 'geheim';
GRANT REPLICATION SLAVE ON *.*
TO 'replication'@'192.168.1.%';
```

Die Log-Position auf dem Master wird mittels SHOW MASTER STATUS ermittelt (file:mysql-bin.000003, Position:1312).

Um einen Slave aufsetzen zu können ist es zudem notwendig einen initialen konsistenten Dump des Masters zu ziehen, welcher zusätzlich die dazugehörenden Information des Binary Log Files sowie der aktuellen Position, welche dem Dump entspricht, beinhaltet. Diese geschieht mit dem Befehl:

```
mysqldump --all-databases --single-transaction \
--master-data > full_dump.sql
```

```
USE test;
FLUSH TABLES WITH READ LOCK;
SHOW MASTER STATUS;
quit;
```

```
mysqldump -u root -pgeheim --opt test > testdb.sql
mysql -u root -pgeheim
UNLOCK TABLES;
quit;

GRANT REPLICATION SLAVE ON *.* TO 'replicationusr'@'%' IDENTIFIED BY
'geheim';
Query OK, 0 rows affected (0.00 sec)

show grants for 'replicationusr'@'%';
+-------------------------------------------------------------------------
-----------------------------------------+
| Grants for replicationusr@%                                             |
+-------------------------------------------------------------------------
-----------------------------------------+
| GRANT REPLICATION SLAVE ON *.* TO 'replicationusr'@'%' IDENTIFIED BY
PASSWORD
'*7B2F14D9BB629E334CD49A1028BD85750F7D3530' |
+-------------------------------------------------------------------------
-----------------------------------------+
1 row in set (0.00 sec)
```

Anschliessend muss man dem zukünftigen Slave mitteilen, wo sein Master zu finden ist und welcher User für die Replikation verwendet werden soll.

```
CHANGE MASTER TO MASTER_HOST='192.168.0.100', MASTER_USER='replicationusr',
MASTER_PASSWORD='geheim',        MASTER_LOG_FILE='        mysql-bin.000003',
MASTER_LOG_POS=1312;
```

Danach kann der auf dem Master gezogene initiale Dump auf dem Slave eingespielt werden.

/etc/my.cnf (des Slaves)
[mysqld]
server-id=2
master-host=192.168.0.100
master-user=replicationusr
master-password=geheim
master-connect-retry=60
replicate-do-db=test

```
mysql restart

mysql -u root -pgeheim
CREATE DATABASE test;
quit;
```

mysql -u root -pgeheim test < testdb.sql

Nach einem finalen überprüfen aller Einstellungen kann der Slave mit dem Befehl START SLAVE gestartet werden.
Der Slave verbindet sich nun automatisch mit dem Master und holt sich die ihm noch fehlenden Informationen ab.

Anhalten / Pausieren / Stoppen der Replikation

```
mysql -u root -pgeheim
SLAVE STOP;
```

Limitierungen

- Keine Fehlererkennung
- Bei temporären Tabellen kann es zu Problemen kommen, wenn sie zwischen zwei logfiles weiter bestehen muss. Wird der ein logfile eingespeist und der Client beendet, verschwindet die temporäre Tabelle, auch wenn Befehle im zweiten Logfile sich darauf beziehen. Hierfür gibt es Workarounds (direkte, sequentielle Einspeisung, Kombinieren der Files etc.).

MySQL-Replikation im täglichen Betrieb

Auch wenn der Slave zwischenzeitlich gestoppt wird, synchronisiert er sich automatisch, nachdem er wieder neu gestartet wird.
Im laufenden Betrieb ist zu überwachen dass der Slave sauber funktioniert und die Datenbank konsistent ist. Bei der Statment Based Replikation (SBR) kann es zu Inkonsistenzen kommen.
Auf dem Master müssen die Binary-Logs gelöscht werden.

```
PURGE BINARY LOGS
```

Konsistenz der Standby-DB prüfen. Schwierig, da manche Datenbanken im Speicher und nicht auf der Festplatte liegen. Mittels „checksum“ auf Master und Slave kann dies geprüft werden.

Aktivierung der Standby Datenbank im Fehlerfall

Im Fehlerfall sollten Datenbank und Applikation möglichst schnell und weitgehend automatisiert auf dem Standby-System aktiviert werden können. Dabei muss aber vorab schon definiert sein wie auf welche Szenarien reagiert werden soll. Solche Szenarien können sein:

- Hardwareausfall
- Logische Fehler (z.B. korrupte Daten, fehlerhafte Software-Updates)
- Maintenance

Bei Hardwareausfällen empfiehlt es sich auf einen möglichst aktuellen Zeitpunkt zu recovern. Auf welchen Zeitpunkt maximal recovert werden kann ist hier abhängig, wann die letzten Änderungsdaten vor dem Fehler übertragen wurden.
Bei logischen Fehlern soll lediglich bis kurz vor den Zeitpunkt des logischen Fehlers recovert werden. Hierbei stellt die Analyse des Zeitpunkts des logischen Fehlers eine der grössten Herausforderungen dar. Auch ist es nicht immer erforderlich auf dem Standby-System produktiv zu gehen. Ist man mit der Datenbankstruktur vertraut und sind somit die Abhängigkeiten der Tabellen bekannt, kann die Standby-Datenbank read only zur Verfügung gestellt werden. Konsistente Daten des Spiegelsystems können mittels export/import auf dem Produktiv-System wieder eingespielt werden.

Die Binary-Logs können mittels des Tools mysqlbinlog ausgelesen werden damit möglicherweise die fehlerhafte Transaktion ermittelt werden. Somit kann dann der Zeitpunkt für den idealen Recover-Point-In-Time festgelegt werden.
Aufruf mittels mysqlbinlog <Dateiname>, um ein Logfile zu dekodieren, das heißt, es in für Menschen lesbarer Form auf die Kommandozeile auszugeben.
Mittels --start-datetime=<Datum, Uhrzeit> lässt sich ein Startpunkt für die Ausgabe, mittels –stop-datetime=<Datum, Uhrzeit> lässt sich ein Endzeitpunkt für die Ausgabe spezifizieren.

```
mysql –start-datetime=“2012-09-12 11:12:13“ mysql-bin.000013
```

Dieser Befehl gibt das binlog.00017 ab dem 12.09.2012, Zeitpunkt 11Uhr,12 Minuten und 13 Sekunden aus.

Für das Point-In-Time-Recovery werden alle Logs abgefahren, die älter als der gewünschte Zeitpunkt sind. Dann wird das letzte Log bis zu dem gewünschten Zeitpunkt abgefahren.
Abfahren der binärlogs mittels:

```
mysqlbinlog –D mysql-bin.000013 –stop-datetime=“2012-09-12 11:13:00“|mysql
```

Um den Slave als produktiv zu starten müssen folgende Schritte durchlaufen werden:

```
STOP SLAVE
RESET MASTER
```

DB durchstarten

Wurde erfolgreich auf das Standby-System umgeschaltet, die Applikation aktiviert und die Benutzer mit diesem verbunden, ist die Grundlage für die **Weiterführung des Geschäftsbetriebes** hergestellt. Jedoch sollte auch bedacht werden wie der Umzug zurück auf das Produktiv-System realisiert werden kann. Meist hat das Standby-System eine geringere Performance und/oder die Konnektivität ist eingeschränkt. Durch entsprechende Vorarbeiten im normalen Betrieb kann vermieden werden, dass die Spiegelung eine reine **Einbahnlösung** ist.
So müssen Master und alle Slaves umkonfiguiert werden, damit die Spiegelung nun vom neuen Master aus auf alle Slaves aufgebaut werden kann.

Zusammenfassung
Eine MySQL Spiegelung lässt sich mit recht einfachen Mitteln einrichten. Bisher war die Überwachung auf fehlerfreie Übertragung der Änderungen und konsistenten Stand der Standby-DB noch einigermassen rudimentär. Auch war es bisher nicht standardmässig vorgesehen die Standby-DB mit einem Zeitversatz zu fahren, um sich gegen logische Fehler zu schützen. Hier findet gerade ein grosser Umbruch statt.
Hier gilt es vorab sauber zu definieren, welche Ziele mit der Spiegelung erreicht werden sollen, um die optimale Mischung aus Performance, Kosten und Verfügbarkeit umsetzen zu können.

Kontaktadresse:
Franz Diegruber
Libelle AG
Gewerbestr. 42
D-70565 Stuttgart

Telefon:	+49 (0) 711-78335 312
Fax:	+49 (0) 711-78335 340
E-Mail	fdiegruber@libelle.com
Internet:	www.libelle.com