MySQL-Umgebungen absichern mit Standby Datenbank

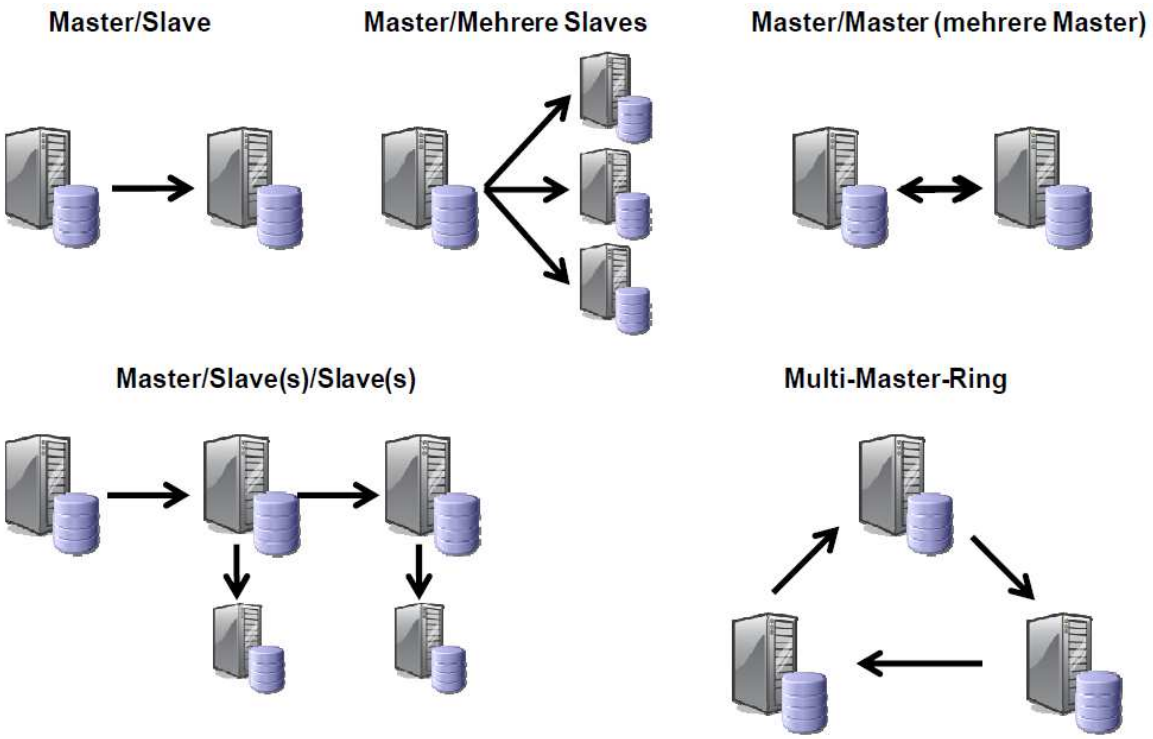
DOAG 20.11.2012

Franz Diegruber
Libelle AG

Libelle

MySQL-Umgebungen absichern mit Standby Datenbank – Topologien

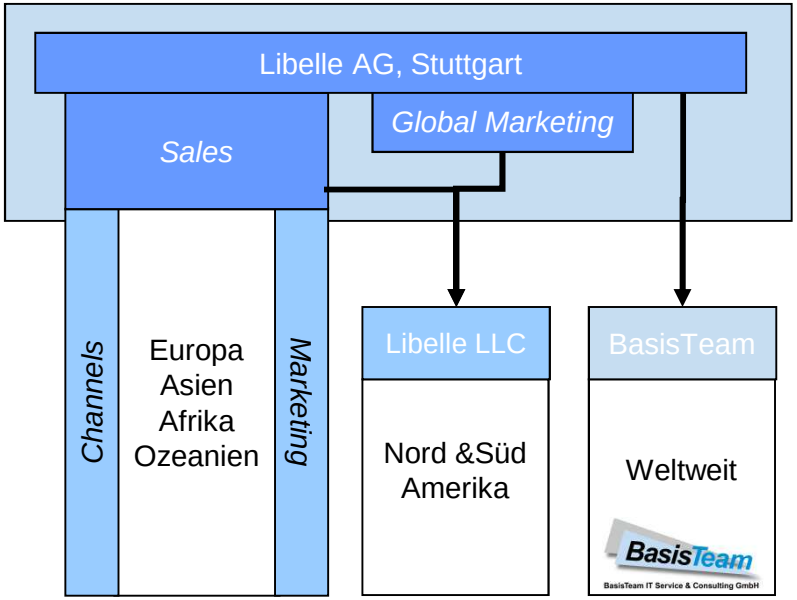
Replikationstopologien



Quelle: Oracle

Die Libelle AG

- Deutscher Softwarehersteller, gegründet 1994
- Alle zentralen Unternehmensfunktionen sitzen in Stuttgart, inklusive Entwicklung, Hotline und Support
- Das Lösungs- und Service-Portfolio fokussiert auf High Availability- und Disaster-Recovery-Architekturen u. a. im SAP-Umfeld
- Libelle ist Oracle Gold, IBM Advanced und SAP Partner
- Kernprodukt ist die Lösung Libelle **Business***Shadow*®, die SAP zertifizierte Verfügbarkeits- und Disaster Recovery-Lösung
- Eine weitere Lösung ist die ebenfalls SAP zertifizierte Libelle **System***Copy* als Werkzeug für Systemkopien
- Auf der Beratungsseite werden diese Lösungen durch das Dienstleistungsportfolio der seit 2010 zur Libelle AG gehörenden BasisTeam GmbH ergänzt



MySQL-Umgebungen absichern mit Standby Datenbank – Libelle Lösung

Optimale Lösung

Der Libelle **Business***Shadow*® besteht aus vier Komponenten:

- **DB***Shadow*®:
  Spiegelung von Datenbanken
- **FS***Shadow*®:
  Spiegelung von Filesystemen
- **Switch***Application*:
  Zum automatisierten Umgang mit IP-Adressen und Hostnames im Umschaltfall
- **Grafische Benutzeroberfläche**:
  Zur einfachen Steuerung aller angehängten Spiegel von einem beliebigen Ort aus.

MySQL-Umgebungen absichern mit Standby Datenbank – Inhaltsübersicht

Inhaltsübersicht

MySQL-Umgebungen absichern mit Standby Datenbank – Gründe für Standby System

Gründe für Standby System

Es gibt viele Gründe ein Spiegelsystem zu betreiben:

- Brand, Wasser, Terror
- organisatorische Erfordernisse
  - zentralisierte Auswertungen
  - Backup
- gesetzliche Bestimmungen
  - Basel II
  - SOX

Überlegungen für Standby System

Warum:

- Daten durch Anwender oder auch fehlerhafte Software gelöscht oder überschrieben
- Wiederherstellung zeitintensiv

Anforderung:

- **automatisiert** oder mit wenigen Mausklicks
- **einfach** und **schnell** vom Produktions- auf das Spiegelsystem umstellt
- entweder auf den **letzten bekannten Datenbestand** vor dem Ausfall, oder auf einen früheren Zustand, z.B. **vor Auftreten eines logischen Fehlers**
- Die hohe Kunst der Datenwiederherstellung ist es hierbei den **richtigen Zeitpunkt** zu erwischen

MySQL-Umgebungen absichern mit Standby Datenbank – Überlegungen zur Konzeptionierung des Standby System

Überlegungen für Standby System

Für welche **Szenarien** soll die Lösung zum Einsatz kommen:

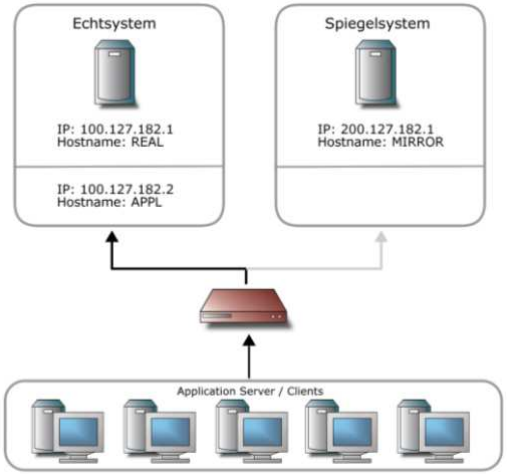
- Hardwareausfall
- Logische Fehler
  - Analyse des Fehlerzeitpunkts: mysqlbinlog
  - Export/Import von Tabellen; kein produktiver Betrieb
- Maintenance (Umschaltung ohne Verlust von Transaktionen)

SLA:

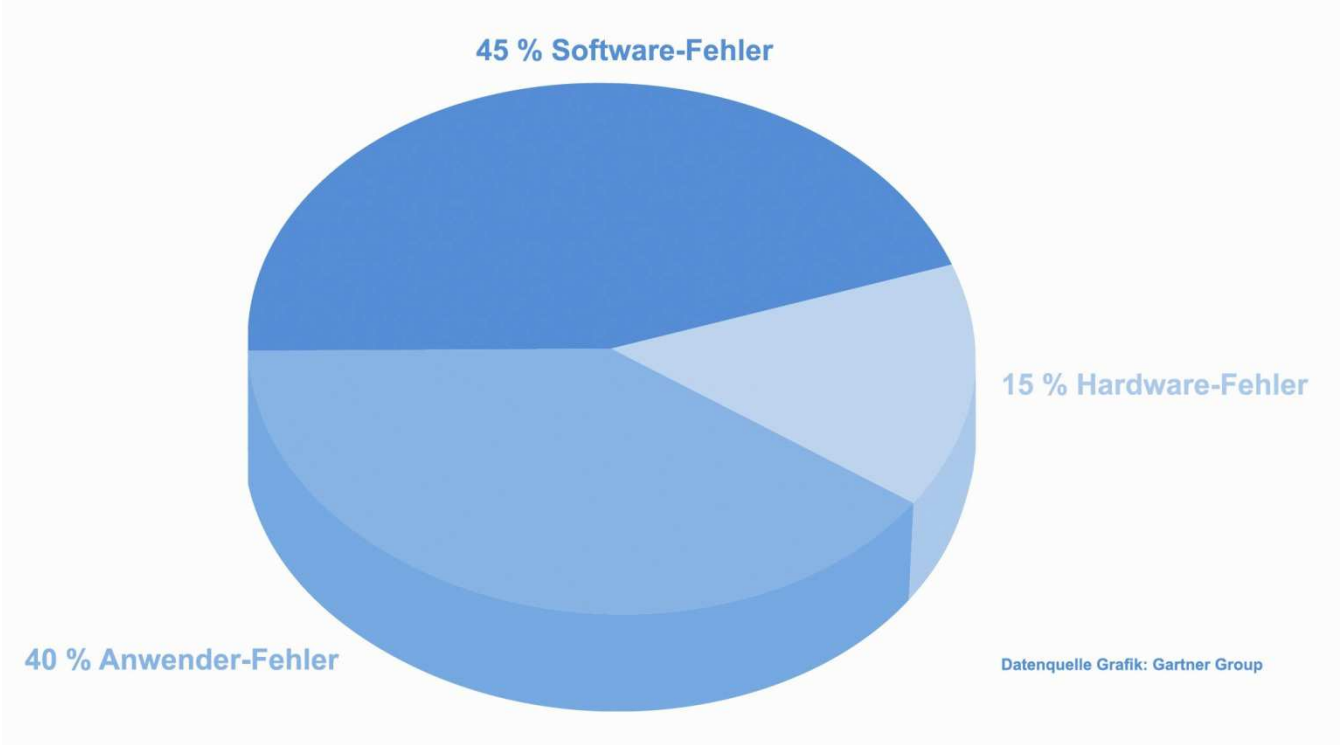
- RPO: Mit oder ohne Zeitversatz recovern
- RTO: Wie schnell muss das Standby System produktiv gehen können
- RCO: Abhängigkeiten von DBs untereinander

Neben der Datenbank- und Filesystemaktivierung gilt es noch zu berücksichtigen:

- Umschaltung IP-Adresse, Hostname, NetBIOS-Name für **connectivity**
  - Arbeiten über VIP
  - DNS, Routing
  - Geänderter Connect für User: VLAN, DNS, zweites Logon
  - Durchstarten der Prozesse von Applikationsservern
- Umzug SAN, **Storage**



MySQL-Umgebungen absichern mit Standby Datenbank – Gründe für Ausfälle



Studie

Systemausfälle basieren **immer seltener auf Hardwareproblemen**. Der Anteil ungeplanter Stillstände aufgrund **software- und anwenderbedingter Fehler** wird unter anderem bedingt durch steigende Komplexität **immer größer**.

MySQL-Umgebungen absichern mit Standby Datenbank – Absicherung der Applikation

Absicherung der Applikation

- 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.
- Für die **Spiegelung der Applikationsfiles** kann auf
  - Betriebssystemmittel
  - Storagemittel
  - Produkte von Drittanbietern zurückgegriffen werden
- Im laufenden Betrieb muss auf Filesystemebene nach **editierten** und **gelöschten** Files und Directories gesucht werden, genauso wie nach **Permission-** und **owner:group-Änderungen**.
- Wichtig ist auch, dass die Prüfung nach Veränderungen in möglichst **kleinen Zyklen** erfolgt, um im Falle einer Aktivierung auf dem Spiegelserver die DB und Applikation möglichst **konsistent** zueinander zur Verfügung stellen zu können.

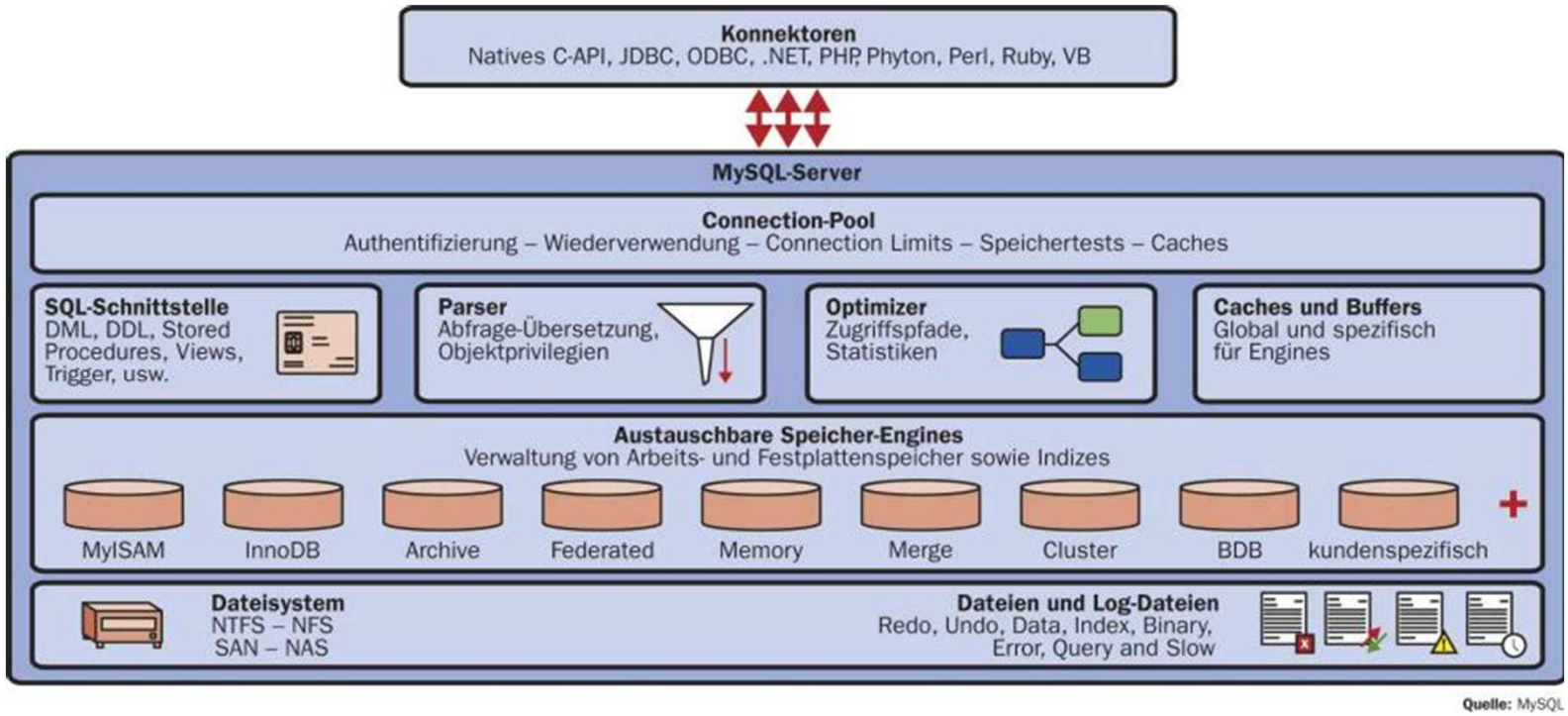
Libelle

MySQL – technische Übersicht 1

- Eine der wichtigsten Unterschiede von MySQL zu anderen Datenbank-Systemen ist die Möglichkeit, verschiedene Storage Engines zu benutzen. **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).

MySQL-Umgebungen absichern mit Standby Datenbank – MySQL - technische Übersicht

MySQL – technische Übersicht 2



Quelle: MySQL

MySQL – Administration 1

- 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.
  - mysqlshowlog: Auslesen von Bin Logs.
- DB-Version ermitteln
  mysqld -V

MySQL – Administration 2

- 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

MySQL-Umgebungen absichern mit Standby Datenbank – DRBD und Cluster

DRBD und Cluster 1

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 DRBD ist ein **Linux Kernel Module** das auf einem verteilten storagesystem basiert.

DRBD und Cluster 2

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 Haupt-speicher gehalten werden (können). Dieses Konzept hat allerdings einen **Overhead** bei **Schreibzugriffen** zur Folge, insbesondere wenn viele Knotenrechner in einer Server-Farm koordiniert werden müssen.

MySQL-Umgebungen absichern mit Standby Datenbank – Übersicht über die Replikation einer MySQL-DB

Replikation einer MySQL-DB 1

- 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).
- Die Replikationsfunktionen in MySQL basierten ursprünglich auf der Weitergabe von SQL-Anweisungen vom Master an den Slave. Dies bezeichnete man als **anweisungsbasierte Replikation (Statement-Based Replication, SBR)**. Seit MySQL 5.1.5 gibt es noch eine andere Basis für die Replikation: die **datensatzbasierte Replikation (Row-Based Replication, RBR)**. Statt die SQL-Anweisungen an den Slave zu senden, schreibt der Master alle Ereignisse, die angeben, wie einzelne Datensätze in der Tabelle geändert werden, in sein Binärlog.
- Prüfung welche Replikationsmethode aktiv ist:

  mysql> SHOW VARIABLES LIKE 'binlog_format';

  Dieser heißt entweder STATEMENT (anweisungsbasierte Replikation) oder ROW (datensatzbasierte Replikation).
- 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.

Replikation einer MySQL-DB 2

- Vorteile der **anweisungsbasierten Replikation**
  - Bewährte Technologie
  - **Kleinere** Logdateien
  - Logdateien enthalten alle Anweisungen, mit denen Änderungen vorgenommen wurden, d. h., sie können zur Überwachung der Datenbank verwendet werden.
  - Logdateien können nicht nur zu Replikationszwecken, sondern auch zur **Point-in-Time-Wiederherstellung** verwendet werden.
  - Ein Slave kann eine **neuere MySQL-Version** mit einer anderen Datensatzstruktur verwenden.
- Nachteile der **anweisungsbasierten Replikation**
  - Nicht alle UPDATE-Anweisungen können repliziert werden: Jedes nichtdeterministische Verhalten (z. B. bei der Verwendung von Zufallsfunktionen in einer SQL-Anweisung) ist bei der anweisungsbasierten Replikation schwer zu replizieren.
- Vorteile der **datensatzbasierten Replikation**
  - Alles kann repliziert werden. Dies ist die sicherste Form der Replikation.
- Nachteile der **datensatzbasierten Replikation**
  - **Größere** Logdateien (in manchen Fällen sogar wesentlich größer).
  - Wenn die Anweisung viele Datensätze ändert, schreibt die datensatzbasierte Replikation **deutlich mehr Daten** in das Binärlog.
  - Sie können auf dem Slave **nicht nachprüfen**, welche Anweisungen vom Master empfangen und dann ausgeführt wurden.

MySQL-Umgebungen absichern mit Standby Datenbank – Übersicht über die Replikation einer MySQL-DB

Replikation einer MySQL-DB 3

- 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:

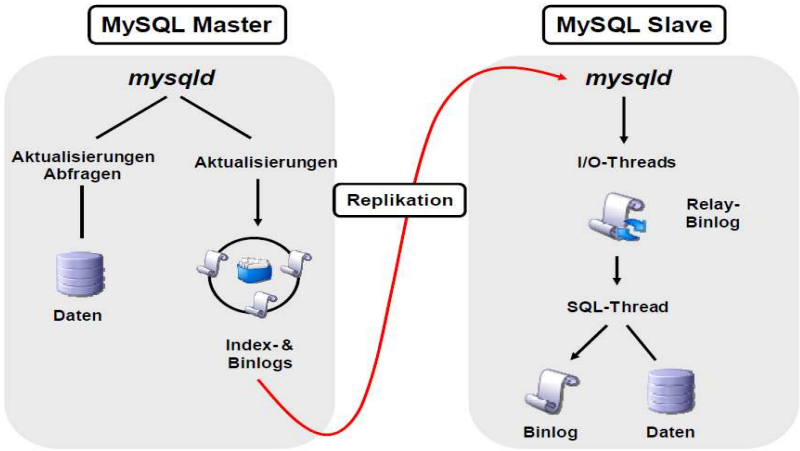
  #120823 9:35:36 server id 1 end_log_pos 108 Query thread_id=6 exec_time=0 error_code=0
  SET TIMESTAMP=1193384136/!*/;*
  insert into cluster values ("Max"),("Moritz")/!*/;*

  - 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).

MySQL-Umgebungen absichern mit Standby Datenbank – Übersicht über die Replikation einer MySQL-DB

Replikation einer MySQL-DB 4

- Die MySQL-Replikation wird unter Verwendung von drei Threads implementiert:
  - einem auf dem Master
    Binlog Dump der die Inhalte der Binärlogs an den Slave sendet
  - zweien auf dem Slave.
    I/O-Thread, der eine Verbindung zum Master herstellt und ihn auffordert, die in seinen Binärlogs aufgezeichneten Änderungen zu senden und kopiert diese in lokale Dateien, (Relay-Logs)
    SQL-Thread, um die Relay-Logs zu lesen und die enthaltenen Aktualisierungen auszuführen.



Quelle: Oracle

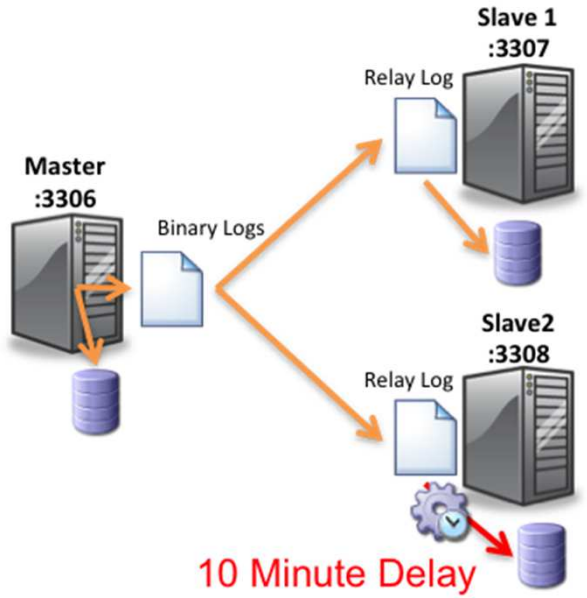
Einsatzgebiete Replikation

- 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 auch für ein **HA oder DR** Konzept zur Verfügung.
- In vielen Fällen schlägt die Replikation von einem **neueren Master** auf einen **älteren Slave** fehl. Generell können Slaves, die unter MySQL 5.1.x laufen, mit älteren Mastern (sogar solchen unter MySQL 3.23, 4.0 oder 4.1) verwendet werden, nicht jedoch umgekehrt.

Libelle

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.
  Für die Einrichtung der semisynchronen Replikation muss auf dem Master und allen Slaves ein Plug-In installiert werden.
- Mit der Version 5.6 wurde die Möglichkeit geschaffen die SQLs auf dem Slave **zeitversetzt einzuspielen**.
  Dies wird über ein commando
  „*change master to master_delay = <x> s*“
  auf dem Slave durchgeführt.
  Bei mehreren Slaves können so unterschiedliche „recover delays“ definiert werden.



Quelle: Oracle

MySQL-Umgebungen absichern mit Standby Datenbank – MySQL-Replikation einrichten

Replikation einrichten 1

- **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.

Replikation einrichten 2

- Replikationsuser anlegen (*CREATE USER*) und Replikationsberechtigungen vergeben (*GRANT REPLICATION SLAVE ON*)
- Sperren des Master (*FLUSH TABLES WITH READ LOCK*) und Binlog-Position merken (*SHOW MASTER STATUS*)
- Master backupen (mysqldump)
- Die DB auf dem Master wieder beschreibbar machen (*UNLOCK TABLES*)
- Auf dem Slave DB anlegen (*CREATE DATABASE*) und den auf dem Master gezogenen **initialen Dump** auf dem Slave eingespielen (mysql -u root -pgeheim test < testdb.sql)

```
CHANGE MASTER TO
  ->    MASTER_HOST='fink',
  ->    MASTER_USER='root',
  ->    MASTER_PASSWORD='geheim',
  ->    MASTER_LOG_FILE='<recorded_log_file_name>',
  ->    MASTER_LOG_POS=<recorded_log_position>;
```

Libelle

Replikation einrichten 3

- 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.
- Prüfen ob die Replikation läuft (*SHOW SLAVE STATUS*)
- Anhalten / Pausieren / Stoppen der Replikation

  mysql -u root -pgeheim
  SLAVE STOP;

MySQL-Umgebungen absichern mit Standby Datenbank – Limitierungen

Limitierungen

- Keine automatische **Fehlererkennung**. Logfiles müssen ausgewertet werden (log-error oder datadir Verzeichnis). Es gibt auch Enterprise Monitor, aber nur mit bestimmten kostenpflichtigen MySQL-Versionen erhältlich.
- Bei **temporären Tabellen** kann es zu Problemen kommen, wenn sie zwischen zwei logfiles weiter bestehen muss. Wird 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.).

```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql
```

Ein weiterer Ansatz besteht darin, alle Logs in eine einzelne Datei zu schreiben und diese Datei dann zu verarbeiten:

```
shell> mysqlbinlog binlog.000001 >  /tmp/statements.sql
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql
shell> mysql -e "source /tmp/statements.sql"
```

Deshalb sollte man vor dem Stoppen des Slaves (mysqladmin shutdown) Prüfen ob noch temporäre Tabellen aktiv sind (SHOW STATUS den Wert der Variablen Slave_open_temp_tables ungleich 0).

MySQL-Umgebungen absichern mit Standby Datenbank – Im täglichen Betrieb

Täglicher 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.
  Hierzu kann ein SHOW MASTER STATUS und SHOW SLAVE STATUS ausgewertet werden.
  Wichtige Felder des SHOW SLAVE STATUS:
  Slave_IO_State, Slave_IO_Running, Slave_SQL_Running, Last_Error, Seconds_Behind_Master
  (Achtung: „0“ kann auch bedeuten, dass es einen Verbindungsfehler gibt)
- Auf dem **Master** müssen die **Binary-Logs gelöscht werden**.
  PURGE BINARY LOGS
  Auf dem **Slave** werden die **logs automatisch** nach dem Einspielen **gelöscht**.
- **Konsistenz der Standby-DB prüfen**. Schwierig, da manche Datenbanken im Speicher und nicht auf der Festplatte liegen. Mittels Ermittlung der „checksum“ auf Master und Slave kann dies geprüft werden.

MySQL-Umgebungen absichern mit Standby Datenbank – Produktivgang

Produktivgang 1

- 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
- Während es für den **Hardwareausfall** und **Maintenance** das Ziel ist möglichst alle Änderungen zu recovern, ist beim **Logischen Fehler** die hohe Kunst den Fehlerzeitpunkt zu ermitteln und die DB mit Stand vor dem Fehler zur Verfügung zu stellen.
- 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-Umgebungen absichern mit Standby Datenbank – Produktivgang

Produktivgang 2

- Dieser Befehl gibt das binlog.00013 ab dem 12.09.2012, Zeitpunkt 11Uhr,12 Minuten und 13 Sekunden aus.

  mysql –start-datetime="2012-09-12 11:12:13" mysql-bin.000013
- 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
  log-bin setzen (my.ini bzw. my.cnf)
  DB durchstarten

MySQL-Umgebungen absichern mit Standby Datenbank – Wiederherstellung Normalbetrieb

Wiederherstellung Normalbetrieb

- 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. Durch entsprechende Vorarbeiten im normalen Betrieb kann vermieden werden, dass die Spiegelung eine **reine Einbahnlösung** ist.

MySQL-Umgebungen absichern mit Standby Datenbank – Zusammenfassung

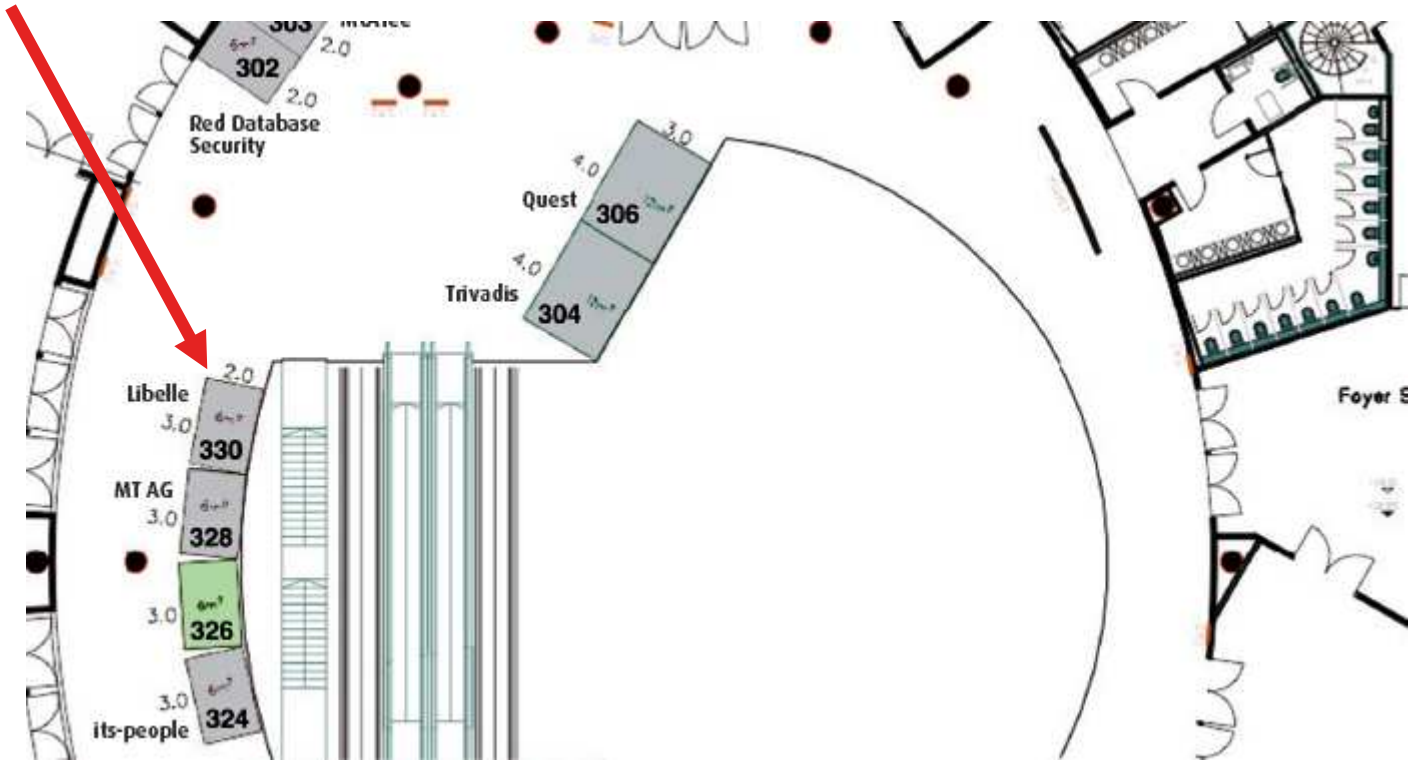
Zusammenfassung

- 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.
- Es gilt vorab sauber zu definieren, **welche Ziele mit der Spiegelung** erreicht werden sollen, um die optimale Mischung aus Performance, Kosten und Verfügbarkeit umsetzen zu können.
- Jedoch gilt es **auch die Applikationsebene** zu betrachten und sicher zu stellen dass DB und Applikationsfiles einen konsistenten Stand darstellen.
- **Prozesse und Prüfungen genau zu definieren** und schriftlich festzuhalten. Hierbei ist es eine Frage des Geschmacks wie weit **automatisiert** wird (möglicherweise schneller) oder ob **manuell** jeder Schritt durchlaufen werden soll (flexibler um auf spezielle Situationen und Fehler reagieren zu können).

MySQL-Umgebungen absichern mit Standby Datenbank – Mehr Information

Libelle

Sie finden Libelle auf der DOAG auf Ebene 3



MySQL-Umgebungen absichern mit Standby Datenbank – Referent

Franz Diegruber
Technical Consultant

Libelle AG
Gewerbestr. 42
70565 Stuttgart, Germany

T +49 711 / 78335-312
F +49 711 / 78335-148
M +49 172 / 71 93 550

Franz.Diegruber@libelle.com
www.libelle.com