

MySQL

- [DB-Replikation](#)
- [Prozesse mit bestimmtem 'state' killen](#)
- [Waiting for table flush](#)
- [User auf neuen Server umziehen](#)
- [mysql_native_password vs. caching_sha2_password](#)

DB-Replikation

Einrichtung

Master

1. MySQL Konfigurationsdatei anpassen:

```
[mysqld]
server-id = <eindeutige ID zB 10>
log-bin = mysql-bin
binlog-format = ROW
```

2. Neustarten des MySQL-Dienstes auf dem Master-Server
3. Benutzer für die Replikation erstellen und Berechtigungen erteilen:

```
CREATE USER 'replica_user'@'%' IDENTIFIED BY 'secure_password';
GRANT REPLICATION SLAVE ON *.* TO 'replica_user'@'%';
FLUSH PRIVILEGES;
```

4. Erstellen eines Snapshot-Backups:
Dies ist notwendig, um sicherzustellen, dass der Slave-Server mit den aktuellen Daten startet.

```
FLUSH TABLES WITH READ LOCK;
SHOW MASTER STATUS;
-- Notieren Sie sich den Wert von 'File' und 'Position'
-- für die spätere Verwendung auf dem Slave-Server.
```

==> Backup erstellen (mysqldump) und auf dem Slave-Server ablegen.

5. Tabellen wieder entsperren:

```
UNLOCK TABLES;
```

Slave

1. MySQL Konfigurationsdatei anpassen:

```
[mysqld]
server-id = <eindeutige ID zB 11>
relay-log = mysql-relay-bin
```

2. Neustarten des MySQL-Dienstes auf dem Slave-Server
3. Datenbank-Dump auf dem Slave-Server importieren
4. Slave-Server konfigurieren, um mit dem Master zu synchronisieren:

```
CHANGE MASTER TO
  MASTER_HOST='master_host_ip',
  MASTER_USER='replica_user',
  MASTER_PASSWORD='secure_password',
  MASTER_LOG_FILE='mysql-bin.000001', -- Wert von SHOW MASTER STATUS
  MASTER_LOG_POS=1234; -- Wert von SHOW MASTER STATUS
```

5. Replikation starten:

```
START SLAVE;
```

6. Status der Replikation überprüfen:

```
SHOW SLAVE STATUS\G;
```

Prozesse mit bestimmtem 'state' killen

Bash-Einzeiler (auf der Linux-Konsole)

Wenn du SSH-Zugriff auf den Server oder den MySQL-Client hast, kannst du die Prozess-IDs mit `awk` auslesen und direkt zurück an MySQL zum Beenden pipe-en:

```
mysql -e "SELECT id FROM information_schema.processlist WHERE state = 'Waiting for table flush';" \
| awk '{if(NR>1) print "KILL \"$1\";"}' \
| mysql
```

```
mysql -e "SELECT id FROM information_schema.processlist WHERE state = 'User Sleep';" \
| awk '{if(NR>1) print "KILL \"$1\";"}' \
| mysql
```

`pt-kill` (Der professionelle Weg für Produktion)

In Produktionsumgebungen ist `pt-kill` (aus dem Percona Toolkit) der sicherste Standard. Es beendet gezielt nur Prozesse, die Kriterien wie den Status erfüllen:

```
pt-kill --match-state "Waiting for table flush" --kill --victim-order oldest --interval 5
```

- `--kill`: Beendet die Verbindungen.
- `--print` (optional statt `--kill`): Zeigt zuerst nur an, was gelöscht werden würde, ohne es auszuführen.

Waiting for table flush

Der Status `Waiting for table flush` entsteht meistens durch ein klassisches **Henne-Ei-Problem** im MySQL-Query-Locking: Ein Prozess fordert das Schließen/Leeren der Tabellencaches an (z. B. ein Backup, `FLUSH TABLES`, `ALTER TABLE` oder `ANALYZE TABLE`), wird aber selbst von einer lang laufenden `SELECT`-Query blockiert. Alle *darauffolgenden* Abfragen auf diese Tabelle müssen dann im Status `Waiting for table flush` warten.

Hier ist die Schritt-für-Schritt-Anleitung zur Ursachenforschung und Prävention:

1. Die genaue Ursache ermitteln

Um den "Übeltäter" (den Blocker) zu finden, musst du die Kette der abhängigen Abfragen analysieren.

Schritt A: Langläufer identifizieren (`SHOW FULL PROCESSLIST`)

Führe folgenden Befehl aus:

SQL

```
SELECT ID, USER, HOST, DB, COMMAND, TIME, STATE, INFO
FROM information_schema.PROCESSLIST
WHERE COMMAND != 'Sleep'
ORDER BY TIME DESC;
```

Wonach du suchen musst:

1. **Der Flush-Auslöser:** Suche nach Abfragen mit `FLUSH TABLES`, `ALTER TABLE`, `RENAME TABLE`, `OPTIMIZE TABLE` oder `ANALYZE TABLE`. Diese haben oft eine moderate `TIME`.
2. **Der eigentliche Blocker:** Suche nach **sehr alten** `SELECT`-Abfragen (hohe `TIME`), die **vor** dem Flush gestartet wurden. Das sind häufig schlechte Queries ohne passenden Index oder ungeschlossene Transaktionen.

Schritt B: Exakte Sperren über das Performance Schema auslesen

Falls MySQL 5.7/8.0+ im Einsatz ist, zeigt dir das `performance_schema` genau, welcher Prozess auf welche Metadata Lock (MDL) wartet:

SQL

```
SELECT
    waiting_p.ID AS waiting_thread_id,
    waiting_p.INFO AS waiting_query,
    blocking_p.ID AS blocking_thread_id,
    blocking_p.INFO AS blocking_query
FROM performance_schema.metadata_locks waiting_ml
JOIN information_schema.PROCESSLIST waiting_p
    ON waiting_ml.OWNER_THREAD_ID = waiting_p.ID
JOIN performance_schema.metadata_locks blocking_ml
    ON waiting_ml.OBJECT_SCHEMA = blocking_ml.OBJECT_SCHEMA
    AND waiting_ml.OBJECT_NAME = blocking_ml.OBJECT_NAME
JOIN information_schema.PROCESSLIST blocking_p
    ON blocking_ml.OWNER_THREAD_ID = blocking_p.ID
WHERE waiting_p.STATE = 'Waiting for table flush'
    AND blocking_p.ID != waiting_p.ID;
```

2. Typische Ursachen & wie man sie behebt

Ursache 1: Automatische Backups (`mysqldump` / `mariadb-dump`)

Klassischer Auslöser: Ein Backup-Job führt `FLUSH TABLES WITH READ LOCK` aus. Wenn parallel eine lange `SELECT`-Abfrage läuft, blockiert das Backup alle schreibenden und lesenden Zugriffe.

- **Lösung (InnoDB):** Verwende bei `mysqldump` immer das Flag `--single-transaction`. Dadurch wird die Tabelle nicht mehr global gesperrt:

Bash

```
mysqldump --single-transaction --quick -u root -p meinedatenbank > backup.sql
```

Ursache 2: Langsame `SELECT`-Queries ohne Index

Ein automatischer Wartungsjob (z. B. `ANALYZE TABLE`) oder DDL-Befehl möchte ein Table-Flush ausführen. Ein schlechter `SELECT`, der 5 Minuten für einen Full-Table-Scan braucht, blockiert den Flush.

- **Lösung:**

- Analysiere langsame Abfragen mit `EXPLAIN` und setze fehlende Indizes.
- Begrenze die maximale Laufzeit von Leseabfragen über Timeouts (z. B. in MySQL 8.0+):
SQL

```
SET GLOBAL max_execution_time = 30000; -- bricht SELECTs nach 30 Sekunden ab
```

Ursache 3: Nicht committete Transaktionen

Ein Entwickler oder eine Anwendung hat `BEGIN` ausgeführt, ein `SELECT` gemacht und vergessen, `COMMIT` oder `ROLLBACK` aufzurufen. Der Thread schläft (`Sleep`), hält aber die Metadata-Lock.

- **Lösung:** Aktive Transaktionen identifizieren:

SQL

```
SELECT * FROM information_schema.innodb_trx;
```

Findest du dort alte Transaktionen (`trx_started`), kannst du diese gezielt mit `KILL [trx_mysql_thread_id]` beenden.

Ursache 4: Zu kleine Table Caches

Wenn MySQL ständig Tabellen schließen muss, weil der Cache voll ist, steigt die Wahrscheinlichkeit für Flush-Konflikte.

- **Lösung:** Überprüfe in der `my.cnf` / `my.ini` folgende Variablen und erhöhe sie gegebenenfalls:

Ini, TOML

```
table_open_cache = 4096  
table_definition_cache = 4096
```

Zusammenfassung zur Prävention

Problem	Schnelle Abhilfe	Langfristige Prävention
Backups	<code>--single-transaction</code> nutzen	Keine <code>FLUSH TABLES WITH READ LOCK</code> auf InnoDB nutzen
Langläufer	Prozess mit <code>KILL <ID></code> beenden	Indizes optimieren & Query-Timeouts setzen
Offene Transaktionen	Idle-Connections kappen	<code>interactive_timeout</code> & <code>wait_timeout</code> reduzieren

User auf neuen Server umziehen

Um einen MySQL-Benutzer inklusive seines verschlüsselten Passworts und aller Rechte 1:1 auf einen neuen Server zu übertragen, nutzt du am besten das MySQL-Kommando `SHOW CREATE USER`.

Schritt 1: Benutzer und Passworthash auf dem alten Server auslesen

Melde dich auf dem **alten Server** in MySQL an:

Bash

```
mysql -u root -p
```

Führe folgenden Befehl für den gewünschten Benutzer aus (ersetze `benutzername` und `localhost` bzw. `%` entsprechend):

SQL

```
SHOW CREATE USER 'benutzername'@'localhost';
```

Ausgabe:

MySQL gibt dir ein fertiges `CREATE USER`-Statement zurück, das das bereits gehashte Passwort enthält. Das sieht ungefähr so aus:

SQL

```
CREATE USER 'benutzername'@'localhost' IDENTIFIED WITH 'caching_sha2_password' AS '$A$005$...'  
FAILED_LOGIN_ATTEMPTS 0 PASSWORD_LOCK_TIME 0 ACCOUNT UNLOCK PASSWORD EXPIRE DEFAULT;
```

(Kopiere dieses gesamte `CREATE USER ...`-Statement).

Schritt 2: Berechtigungen (Grants) auslesen

Lies direkt danach die zugewiesenen Rechte des Benutzers aus:

SQL

```
SHOW GRANTS FOR 'benutzername'@'localhost';
```

Ausgabe:

Du erhältst eine oder mehrere Zeilen mit `GRANT`-Befehlen, zum Beispiel:

SQL

```
GRANT ALL PRIVILEGES ON `meine_datenbank`.* TO `benutzername`@`localhost`;
```

(Kopiere alle ausgegebenen `GRANT`-Zeilen).

Schritt 3: Benutzer auf dem neuen Server anlegen

Melde dich auf dem **neuen Server** in MySQL an:

Bash

```
mysql -u root -p
```

1. Füge den in **Schritt 1** kopierten `CREATE USER`-Befehl ein und führe ihn aus.
2. Füge die in **Schritt 2** kopierten `GRANT`-Befehle ein und führe sie aus.
3. Aktualisiere die Rechematrix:

SQL

```
FLUSH PRIVILEGES;
```

Alternative: Automatisiert über die Kommandozeile (für viele User)

Wenn du den Benutzer direkt über ein Terminal-Einzeiler exportieren möchtest, kannst du `pt-show-grants` aus den **Percona Toolkit** nutzen oder `pt-show-grants` / `mysqldump` verwenden:

Bash

```
# Exportiert CREATE USER und GRANT Statements direkt in eine SQL-Datei
mysql -u root -p -BN -e "SHOW CREATE USER 'benutzername'@'localhost';" > user_export.sql
mysql -u root -p -BN -e "SHOW GRANTS FOR 'benutzername'@'localhost';" >> user_export.sql
```

Diese `user_export.sql` kannst du auf den neuen Server übertragen und mit `mysql -u root -p < user_export.sql` importieren.

mysql_native_password vs. caching_sha2_password

Nein, man kann den Passwort-Hash nicht umwandeln.

Das liegt an der Funktionsweise von kryptografischen Hashes: Sowohl `mysql_native_password` (SHA1-basiert) als auch `caching_sha2_password` (SHA-256-basiert) sind Einweg-Verschlüsselungen. Aus dem Hash auf dem alten Server lässt sich das ursprüngliche Klartext-Passwort mathematisch nicht wiederherstellen, um daraus den neuen SHA-256-Hash zu berechnen.

Du hast in dieser Situation jedoch **zwei praktische Wege**:

Option 1: Das alte Verfahren auf dem neuen Server aktivieren (Empfohlen, wenn Passwort unbekannt)

MySQL 8.0 unterstützt `mysql_native_password` weiterhin (in MySQL 8.0/8.4 ist es zwar deprecated oder als Plugin verfügbar, kann aber problemlos verwendet werden). Du kannst den User also **trotzdem mit seinem alten Hash anlegen**:

SQL

```
CREATE USER 'benutzername'@'localhost'
  IDENTIFIED WITH mysql_native_password AS '*2470C0C21F792A758062C1D2543813E071372E97';
```

(Ersetze den Stern-String mit dem Hash aus deinem alten `SHOW CREATE USER`-Befehl).

“ **Hinweis:** Falls MySQL auf dem neuen Server meckert (`Plugin 'mysql_native_password' is not loaded`), musst du das Plugin in der Server-Konfiguration (`my.cnf` / `mysqld.cnf`) unter `[mysqld]` aktivieren:

Ini, TOML

```
[mysqld]
mysql_native_password=ON
```

Danach den MySQL-Dienst neu starten (`systemctl restart mysql`).

Option 2: Neues Passwort mit `caching_sha2_password` vergeben

Wenn dir das Passwort im Klartext bekannt ist (oder du ein neues vergeben kannst), erstelle den User direkt mit dem neuen Standard-Verfahren:

SQL

```
CREATE USER 'benutzername'@'localhost'  
IDENTIFIED WITH caching_sha2_password BY 'DeinNeuesOderAltesKlartextPasswort';
```

Fazit & Vorgehen

- **Kennst du das Passwort im Klartext?** → Nutze **Option 2** (sicherer und zukunftssicherer).
- **Kennst du das Passwort NICHT?** → Nutze **Option 1**, um den alten Hash direkt zu übernehmen.