MySQL

- DB-Replikation
- Prozesse mit bestimmtem 'state' killen
- Waiting for table flush

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	`--single-transaction` nutzen	Keine `FLUSH TABLES WITH READ LOCK` auf InnoDB nutzen
Langläufer	Prozess mit `KILL <ID>` beenden	Indizes optimieren & Query-Timeouts setzen
Offene Transaktionen	Idle-Connections kappen	`interactive_timeout` & `wait_timeout` reduzieren