Zusammenfassung der wichtigsten Erkenntnisse:

Punkt	Beschreibung
Ursache	Tritt auf, wenn zwei oder mehr Transaktionen dieselbe Ressource in einer Datenbank anfordern und sich gegenseitig blockieren.
Symptome	Fehler: `"Deadlock found when trying to get lock; try restarting transaction"`.
Lösung	Identifizieren von Deadlocks, Analyse von Transaktionsmustern, Optimierung der Abfragen und Anpassung von Isolationseinstellungen.
Best Practices	Einheitliche Sperr-Reihenfolge, Minimierung von Ressourcenkonflikten, regelmäßiges Backup & Testen.

Schritt-für-Schritt-Anleitung zum Beheben des Fehlers: "Deadlock found when trying to get lock"

1. Was bedeutet der Fehler?

Ein "Deadlock" entsteht, wenn zwei oder mehr Transaktionen in einer Datenbank gegenseitig auf Ressourcen warten, die von der jeweils anderen gehalten werden. Dies führt zu einer Blockade. Um solche Zustände aufzulösen, beendet die Datenbank eine der Transaktionen (automatisches "Rollback"), was zu der genannten Fehlermeldung führt.

---

2. Ursachen und Identifikation des Problems

a. Ursachen verstehen

- **Uneinheitliche Sperrreihenfolge**: Wenn Transaktionen Ressourcen in unterschiedlicher Reihenfolge sperren.
- **Lange Transaktionszeiten**: Transaktionen, die länger als nötig offenbleiben, erhöhen das Risiko von Deadlocks.
- **Ressourcenkonflikte**: Hohe Konkurrenz auf dieselben Daten, z. B. bei gleichzeitigen Lese-/Schreibvorgängen.

b. Deadlocks identifizieren

1. **Datenbanklogs analysieren**:
   - Überprüfen Sie die Logs mit Query-Tools (z. B. MySQL Workbench).
   - Suchen Sie nach spezifischen Deadlock-Protokollen oder Konfliktinformationen.
2. **Prozessliste anzeigen**:
   - Führen Sie den Befehl aus:

     ```
     SHOW ENGINE INNODB STATUS\G
     ```

   - Hier finden Sie Details zu den gesperrten Ressourcen und beteiligten Transaktionen.

   Experten-Tipp: Nutze spezialisierte Monitoring-Tools wie Datadog oder New Relic zur Echtzeitüberwachung von Datenbankvorgängen.

---

3. Analyse der Transaktionen

a. Bestimmung der Sperr-Reihenfolge

- **Konsistenz herstellen**: Alle Transaktionen sollten Ressourcen immer in derselben Reihenfolge sperren. Beispiel:
  - **Falsch**:
    1. Transaktion A sperrt `Tabelle 1`, dann `Tabelle 2`.
    2. Transaktion B sperrt `Tabelle 2`, dann `Tabelle 1`.
  - **Richtig**:
    - Beide Transaktionen sperren zuerst `Tabelle 1`, dann `Tabelle 2`.

b. Priorisieren von Row-Level-Locks

- Verwenden Sie spezifische Abfragen mit `WHERE`-Klauseln, um Row-Level-Locks durchzuführen, z. B.:

```
DELETE FROM kunden WHERE id = 10;
```

c. Prüfung komplexer/Delete-Queries

- Falls ein Fehler bei einer DELETE-Operation auftritt, verwenden Sie besser eine **geordnete Sortierung**:

```
DELETE FROM daten
WHERE bedingung IS TRUE
ORDER BY spalte ASC;
```

---

4. Optimierung von Abfragen

a. Vermeidung von Full Table Scans

- Indexe korrekt setzen, um unnötige vollständige Tabellen-Scans und damit auch Sperren auf große Tabellen zu vermeiden.

b. Kurzlebige Transaktionen

- Halten Sie Transaktionen so kurz wie möglich – jede Transaktion bleibt nur so lange aktiv, wie unbedingt nötig.
  Beispiel:

```
START TRANSACTION;
UPDATE table_name SET column_name = value WHERE some_condition;
COMMIT;
```

c. Reduzierung der betroffenen Zeilen

- Nutzen Sie gezielte Abfragen, um nur die wirklich relevanten Zeilen zu sperren, z. B. mit `LIMIT`:

```
SELECT * FROM kundendaten WHERE status = 'aktiv' LIMIT 100;
```

---

5. Einstellungen in der Datenbank anpassen

a. Transaktions-Isolationsebene ändern

Die Isolationsebene beeinflusst, wie intensiv Transaktionen Ressourcen sperren:

- Wechsle von `SERIALIZABLE` (höchste Sperrstufe) zu weniger restriktiven Modi wie `READ COMMITTED`:

```
SET GLOBAL TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

b. Deadlock-Timeout anpassen

- Erhöhe den Timeout, um Deadlocks seltener auftreten zu lassen:

```
SET innodb_lock_wait_timeout = 20;
```

---

6. Backups und Plugins

a. Backups anlegen

- Vor Änderungen: Erstelle ein vollständiges Backup deiner Datenbank. Nutze Tools wie MiniTool ShadowMaker.

b. Aktualisiere Software

- Aktualisiere die verwendeten Plugins oder Themes, die möglicherweise ineffiziente SQL-Abfragen ausführen.

  Hinweis: Nutze EaseUS MS SQL Recovery, falls nach Updates Daten beschädigt wurden.

---

7. Test deiner Änderungen

Teste deine Datenbank nach folgenden Kriterien:

- Deadlocks behoben?
- Leistungsverbesserungen spürbar?
- Keine neuen Fehler eingeführt?

Häufig gestellte Fragen (FAQ)

1. **Was ist ein Deadlock in einer Datenbank?**
   Ein Deadlock tritt auf, wenn zwei oder mehr Transaktionen gegenseitig auf das Freigeben von Ressourcen warten. Dadurch entsteht eine Blockade.

2. **Wie kann ich Deadlocks vermeiden?**

   - Konsistente Sperrreihenfolge.
   - Optimierte Abfragen.
   - Nutzung von Row-Level-Locks.
   - Anpassen der Isolationseinstellungen.

3. **Was passiert, wenn Deadlocks regelmäßig auftreten?**

Wenn wiederholte Deadlocks auftreten, solltest du genauere Überwachungstools nutzen, um Transaktionsmuster zu analysieren, und deine Abfragen sowie die Datenbankstruktur optimieren.

4. **Warum tritt der Fehler häufig bei DELETE/UPDATE-Operationen auf?**
   Diese Operationen greifen oft auf mehrere Zeilen zurück, wodurch mehr Ressourcen gesperrt werden. Reduziere betroffene Zeilen mit spezifischen Bedingungen oder einer ORDER-BY-Klausel.

Siehe auch:

- Hostinger für optimiertes Hosting – Verbessere die Leistung deiner Datenbank durch zuverlässiges Hosting.
- MiniTool Partition Wizard für optimale Datenverwaltung.