Datenbank To-Do App

Einleitung

Im Rahmen meines Projekts habe ich mich mit der Implementierung eines hochverfügbaren MySQL-Datenbanksystems beschäftigt. Ziel war es, die Ausfallsicherheit und kontinuierliche Verfügbarkeit der Datenbank zu gewährleisten. Dazu habe ich zwei MySQL-Server in einer Master-Slave-Konfiguration eingerichtet und HAProxy als Load Balancer und Failover-Lösung implementiert.

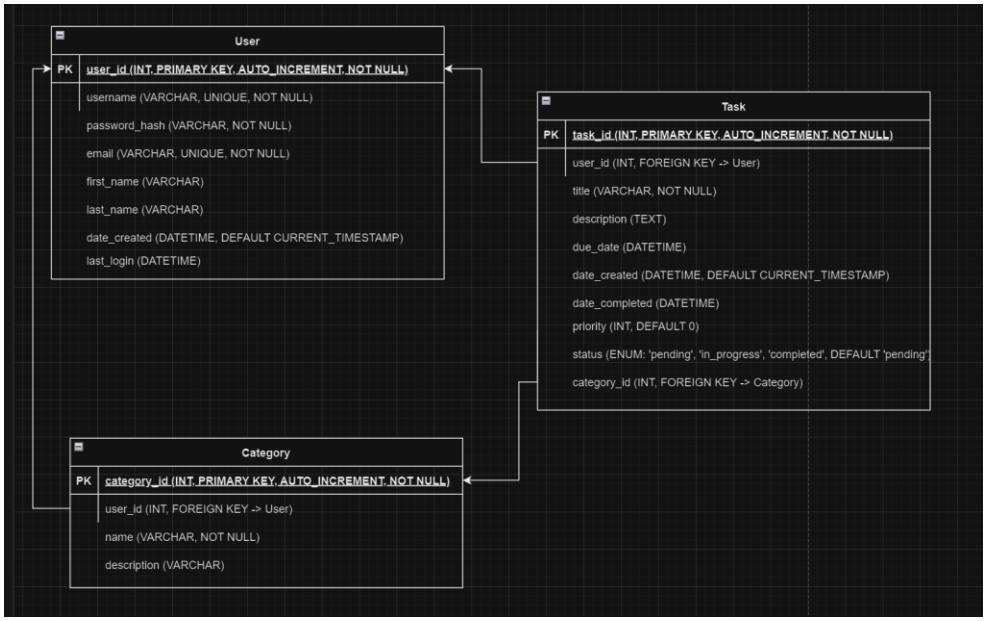
Datenbank

Datenbank definierung und ERD

Ich habe für die Gruppe zuerst eigenständig überlegt, welche Tabellen und Felder wir benötigen, um die Daten zu speichern die wir haben werden. Zuerst hatte ich Notification und Reminder Tabellen dinnen, jedoch haben wir uns nach einer Absprache entschieden diese auszulassen.

Wir arbeiten somit mit 3 Tabellen:

- User
    - Speichert alle Daten bezüglich den Usern
    - Password könnte mit Trigger gehashed, jedoch meinte Kevin das er dies von seiner Seite aus erledigen könnte.
- Task
    - Speicher alle Daten bezüglich den Tasks die man kreiren wird
    - Hat user als Foreign Key um den User zum spezifischen Task zuzuweisen
    - Hat die Kategorie als Foreign key und den Task in die dazugehörende Kategorie zu zuweisen
- Category
    - Speichert verschiedene Kategorien, die Users erstellen
    - Verwendet entsprechend den User Id als foreign key



Das Definieren der Schnittstelle

Für die Schnittstelle habe ich den Zugang zu meinem Server für alle IPs innerhalb des 10.3.32.0-Subnetzes gestattet und in der Datenbank zwei Benutzer mit allen Rechten erstellt. Ein Benutzer ist "backend_user", den Behan verwenden wird, und der andere ist "fastapi_user", den Kevin nutzen wird.

sudo ufw allow from 10.3.32.0/24 to any port 3306

mysql> CREATE USER 'backend_user'@'%' IDENTIFIED BY 'secure_password';

Query OK, 0 rows affected (0.02 sec)

mysql> GRANT ALL PRIVILEGES ON todo_app.* TO 'backend_user'@'%';

Query OK, 0 rows affected (0.01 sec)

mysql> CREATE USER 'fastapi_user'@'10.3.32.24' IDENTIFIED BY 'secure_password';

Query OK, 0 rows affected (0.01 sec)

mysql> GRANT ALL PRIVILEGES ON todo_app.* TO 'fastapi_user'@'10.3.32.24';

Query OK, 0 rows affected (0.00 sec)

Server und Master/Slave Relationship

Serverstruktur

Die Datenbank bildet das Herzstück jeder datenintensiven Anwendung. Um sicherzustellen, dass die Daten jederzeit verfügbar sind und keine Verluste auftreten, habe ich mich für eine Master-Slave-Replikation entschieden:

- Master-Server (Server A, IP: 10.3.32.11): Dieser Server ist für Schreiboperationen zuständig und dient als Hauptdatenquelle.
- Slave-Server (Server B, IP: 10.3.32.24): Dieser Server repliziert die Daten vom Master und steht als Backup bereit.

Gründe für diese Struktur:

- Redundanz: Die Replikation stellt sicher, dass bei einem Ausfall des Masters keine Daten verloren gehen.
- Hochverfügbarkeit: Fällt der Master aus, kann der Slave nahtlos übernehmen.
- Lastverteilung: Leseanfragen können auf beide Server verteilt werden, um die Leistung zu steigern.

Einrichtung der MySQL-Server

Beide Datenbaken laufen auf den Lernmaas. Server A ist meine VM und server B ist Kevins eine VM.

Server A (Master)

Auf Server A habe ich MySQL installiert und so konfiguriert, dass es Remote-Verbindungen zulässt. Dazu habe ich die bind-address in der MySQL-Konfigurationsdatei auf 0.0.0.0 gesetzt, um Verbindungen von allen Netzwerkadressen zu ermöglichen.

Des Weiteren habe ich einen Replikationsbenutzer erstellt und die notwendigen Berechtigungen vergeben. Die Server-ID wurde auf 1 gesetzt, und das Binärlog wurde aktiviert, um die Replikation zu ermöglichen.

Server B (Slave)

Auf Server B wurde ebenfalls MySQL installiert und für die Replikation vorbereitet. Die bind-address wurde ebenfalls auf 0.0.0.0 gesetzt. Die Server-ID wurde auf 2 gesetzt, und das Relay-Log wurde aktiviert. Mit den Informationen vom Master-Server habe ich die Replikation eingerichtet, sodass Server B die Daten von Server A erhält.

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for source to send event
                  Master_Host: 10.3.32.11 Server A
                  Master_User: replica_user
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000010
          Read_Master_Log_Pos: 8358
               Relay_Log_File: mysql-relay-bin.000034
                Relay_Log_Pos: 8480
        Relay_Master_Log_File: mysql-bin.000010
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
```

Erstellung von MySQL-Benutzern für den Zugriff

Um den Zugriff auf die Datenbank zu steuern und Sicherheit zu gewährleisten, habe ich spezifische MySQL-Benutzer erstellt:

- Anwendungsbenutzer (app_user): Dieser Benutzer hat die notwendigen Berechtigungen, um auf die Datenbank zuzugreifen und Operationen durchzuführen. Dadurch können Anwendungen sicher und kontrolliert auf die Daten zugreifen.
- HAProxy-Check-Benutzer (haproxy_check): Dieser Benutzer wird von HAProxy für Gesundheitsprüfungen verwendet. Er hat minimale Berechtigungen, um die Sicherheit zu erhöhen und sicherzustellen, dass nur autorisierte Prüfungen durchgeführt werden.

Implementierung von HAProxy für Hochverfügbarkeit

HAProxy wurde auf Server B installiert, um als Load Balancer und Failover-Lösung zu dienen. Die Konfiguration von HAProxy umfasst:

Frontend: Hört auf Anfragen auf Port 3306 auf der externen IP-Adresse von Server B (10.3.32.24).

Backend: Enthält die Definition beider MySQL-Server. Server A dient als primärer Server, während Server B als Backup fungiert.

Durch diese Konfiguration leitet HAProxy die Anfragen standardmäßig an Server A weiter. Fällt dieser aus, werden die Anfragen automatisch an Server B umgeleitet.

Herausforderungen und Fehlerbehebung

Portkonflikt auf Port 3306

Problem: HAProxy konnte nicht starten, da es nicht an Port 3306 binden konnte. Die Fehlermeldung wies darauf hin, dass der Socket nicht gebunden werden kann.

Analyse: Der Konflikt entstand, weil MySQL bereits auf 127.0.0.1:3306 lauschte und HAProxy versuchte, auf 0.0.0.0:3306 zu binden. Da 0.0.0.0 alle Schnittstellen umfasst, einschließlich 127.0.0.1, kam es zu einem Konflikt.

Lösung: Ich habe die HAProxy-Konfiguration angepasst, sodass HAProxy nur auf der externen IP-Adresse 10.3.32.24 auf Port 3306 bindet. Dadurch wurde der Konflikt vermieden, und HAProxy konnte erfolgreich starten.

Blockierung des Hosts aufgrund vieler Verbindungsfehler

Beim Versuch, eine Verbindung zur MySQL-Datenbank herzustellen, erhielt ich die Fehlermeldung:

- **ERROR 1129 (HY000): Host '10.3.32.24' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'**

Analyse: MySQL blockiert Hosts automatisch nach einer bestimmten Anzahl fehlgeschlagener Verbindungsversuche, um Sicherheitsrisiken zu minimieren.

Lösung: Durch Ausführen von FLUSH HOSTS auf dem MySQL-Server habe ich den Host entsperrt. Um die Ursache der Verbindungsfehler zu beheben, habe ich die Zugangsdaten überprüft und sichergestellt, dass der MySQL-Benutzer die korrekten Berechtigungen hat.

HAProxy startet nicht trotz korrekter Konfiguration

Problem: Trotz angepasster Konfiguration startete HAProxy nicht und zeigte wiederholt Fehlermeldungen an.

Analyse: Neben dem Portkonflikt könnte ein anderer Prozess den Port 3306 nutzen, oder HAProxy hatte nicht die notwendigen Berechtigungen.

```
ubuntu@m321-24-ap21d:~$ systemctl status haproxy.service
● haproxy.service - HAProxy Load Balancer
     Loaded: loaded (/etc/systemd/system/haproxy.service; enabled; vendor preset: enabled)
     Active: failed (Result: exit-code) since Thu 2024-11-07 08:02:59 UTC; 11s ago
       Docs: man:haproxy(1)
             file:/usr/share/doc/haproxy/configuration.txt.gz
    Process: 2710422 ExecStartPre=/usr/sbin/haproxy -Ws -f $CONFIG -c -q $EXTRAOPTS (code=exited, status=0/SUCCESS)
    Process: 2710424 ExecStart=/usr/sbin/haproxy -Ws -f $CONFIG -p $PIDFILE $EXTRAOPTS (code=exited, status=1/FAILURE)
   Main PID: 2710424 (code=exited, status=1/FAILURE)

Nov 07 08:02:59 m321-24-ap21d systemd[1]: haproxy.service: Main process exited, code=exited, status=1/FAILURE
Nov 07 08:02:59 m321-24-ap21d systemd[1]: haproxy.service: Failed with result 'exit-code'.
Nov 07 08:02:59 m321-24-ap21d systemd[1]: Failed to start HAProxy Load Balancer.
Nov 07 08:02:59 m321-24-ap21d systemd[1]: haproxy.service: Scheduled restart job, restart counter is at 5.
Nov 07 08:02:59 m321-24-ap21d systemd[1]: Stopped HAProxy Load Balancer.
Nov 07 08:02:59 m321-24-ap21d systemd[1]: haproxy.service: Start request repeated too quickly.
Nov 07 08:02:59 m321-24-ap21d systemd[1]: haproxy.service: Failed with result 'exit-code'.
Nov 07 08:02:59 m321-24-ap21d systemd[1]: Failed to start HAProxy Load Balancer.
ubuntu@m321-24-ap21d:~$
```

Fazit Failover

Ich konnte das Failover leider schlussendlich nicht implementiere, da ich nicht herausfinden konnte welcher Prozess den Port 3306 verwendet hat. Die Daten auf Server A werden auf Server B repliziert, jedoch wird nicht automatisch zu Server B gewechselt, wenn Server A runtergeht.

Testen der Datenbankoperationen

Im Rahmen der Sicherstellung der Robustheit und Integrität der todo_app-Datenbank habe ich mehrere Tests an den Tabellen User, Category und Task durchgeführt. Diese Tests umfassten sowohl erfolgreiche Operationen als auch absichtliche Fehler, um zu überprüfen, ob die Einschränkungen und Beziehungen korrekt durchgesetzt werden.

Überblick über die Tests

- User-Tabelle:
    - Testen der Einfügung neuer Benutzer mit gültigen Daten.
    - Testen der Einfügung mit doppelten Benutzernamen oder E-Mails.
    - Testen der Einfügung mit fehlenden Pflichtfeldern.
- Category-Tabelle:
    - Testen der Einfügung von Kategorien mit gültigen Benutzer-IDs.
    - Testen der Einfügung mit nicht vorhandenen Benutzer-IDs.
    - Testen der Löschung von Benutzern und der Kaskadeneffekte auf Kategorien.
- Task-Tabelle:
    - Testen der Einfügung von Aufgaben mit gültigen Benutzer- und Kategorie-IDs.
    - Testen der Einfügung mit ungültigen Benutzer- oder Kategorie-IDs.
    - Testen der Löschung von Kategorien und der Auswirkungen auf Aufgaben.
    - Testen der Aktualisierung des Aufgabenstatus und der Priorität.

Detaillierte Testfälle

Tests der User-Tabelle

Test 1: Erfolgreiche Einfügung eines neuen Benutzers

Aktion:

Einfügen eines neuen Benutzers mit allen erforderlichen Feldern, die korrekt ausgefüllt sind.

SQL-Anweisung:

INSERT INTO User (username, password_hash, email, first_name, last_name)

VALUES ('jdoe', 'gehashtes_passwort', 'jdoe@example.com', 'John', 'Doe');

Erwartetes Ergebnis:

- Der Benutzer wird erfolgreich eingefügt.
- user_id wird automatisch inkrementiert.
- date_created wird auf den aktuellen Zeitstempel gesetzt.

Überprüfung:

SELECT * FROM User WHERE username = 'jdoe';

Ergebnis:

- Eine einzelne Zeile mit den angegebenen Benutzerdaten.

Test 2: Einfügung mit doppeltem Benutzernamen

Aktion:

Versuch, einen weiteren Benutzer mit demselben Benutzernamen wie ein bestehender Benutzer einzufügen.

SQL-Anweisung:

INSERT INTO User (username, password_hash, email)

VALUES ('jdoe', 'ein_anderes_gehashtes_passwort', 'johndoe2@example.com');

Erwartetes Ergebnis:

- Die Einfügung schlägt fehl.
- Eine Fehlermeldung weist auf einen doppelten Eintrag für username hin.

Fehlermeldung:

ERROR 1062 (23000): Duplicate entry 'jdoe' for key 'User.username'

Erläuterung:

- Das Feld username hat eine UNIQUE-Einschränkung.
- Doppelte Benutzernamen sind nicht erlaubt, um die Eindeutigkeit der Benutzer sicherzustellen.

Test 3: Einfügung mit fehlendem Pflichtfeld

Aktion:

Versuch, einen Benutzer ohne Angabe des Feldes email einzufügen.

SQL-Anweisung:

INSERT INTO User (username, password_hash)

VALUES ('asmith', 'gehashtes_passwort_für_asmith');

Erwartetes Ergebnis:

- Die Einfügung schlägt fehl.
- Eine Fehlermeldung weist darauf hin, dass email nicht NULL sein darf.

Fehlermeldung:

ERROR 1048 (23000): Column 'email' cannot be null

Erläuterung:

- Das Feld email ist als NOT NULL definiert.

- Alle Pflichtfelder müssen angegeben werden.

Tests der Category-Tabelle

Test 4: Erfolgreiche Einfügung einer neuen Kategorie

Aktion:

Einfügen einer neuen Kategorie, die einem bestehenden Benutzer zugeordnet ist.

SQL-Anweisung:

INSERT INTO Category (user_id, name, description)

VALUES (1, 'Arbeit', 'Aufgaben im Zusammenhang mit Arbeitsprojekten');

Erwartetes Ergebnis:

- Die Kategorie wird erfolgreich eingefügt.
- category_id wird automatisch inkrementiert.

Überprüfung:

SELECT * FROM Category WHERE name = 'Arbeit';

Ergebnis:

- Eine einzelne Zeile mit den angegebenen Kategorieninformationen.

Test 5: Einfügung mit nicht existierender Benutzer-ID

Aktion:

Versuch, eine Kategorie mit einer user_id einzufügen, die nicht in der User-Tabelle existiert.

SQL-Anweisung:

INSERT INTO Category (user_id, name)

VALUES (999, 'Persönlich');

Erwartetes Ergebnis:

- Die Einfügung schlägt fehl.
- Eine Fehlermeldung weist auf einen Fremdschlüsseleinschränkungsverstoß hin.

Fehlermeldung:

ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`todo_app`.`Category`, CONSTRAINT `Category_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `User` (`user_id`) ON DELETE CASCADE ON UPDATE CASCADE)

Erläuterung:

- Die user_id in Category ist ein Fremdschlüssel, der auf User(user_id) verweist.
- Der Wert 999 entspricht keiner existierenden user_id.

Test 6: Löschung eines Benutzers und Kaskadeneffekt auf Kategorien

Aktion:

Löschen eines Benutzers und Beobachtung der Auswirkungen auf die zugehörigen Kategorien.

SQL-Anweisung:

DELETE FROM User WHERE user_id = 1;

Erwartetes Ergebnis:

- Der Benutzer mit user_id = 1 wird gelöscht.
- Alle Kategorien, die user_id = 1 zugeordnet sind, werden ebenfalls gelöscht aufgrund von ON DELETE CASCADE.

Überprüfung:

SELECT * FROM Category WHERE user_id = 1;

Ergebnis:

- Keine Zeilen werden zurückgegeben; die zugehörigen Kategorien wurden gelöscht.

Erläuterung:

- Die Fremdschlüsseleinschränkung spezifiziert ON DELETE CASCADE, was bedeutet, dass abhängige Kategorien gelöscht werden, wenn ein Benutzer gelöscht wird.

Tests der Task-Tabelle

Test 7: Erfolgreiche Einfügung einer neuen Aufgabe

Aktion:

Einfügen einer neuen Aufgabe mit gültiger user_id und category_id.

Voraussetzungen:

- Neuerstellung des Benutzers mit user_id = 1:INSERT INTO User (user_id, username, password_hash, email)

VALUES (1, 'jdoe', 'gehashtes_passwort', 'jdoe@example.com');

- Neuerstellung der Kategorie mit category_id = 1:
- INSERT INTO Category (category_id, user_id, name)

VALUES (1, 1, 'Arbeit');

SQL-Anweisung:

INSERT INTO Task (user_id, title, description, due_date, priority, status, category_id)

VALUES (1, 'Präsentation vorbereiten', 'Folien für das Meeting erstellen', '2023-11-30 10:00:00', 1, 'in_progress', 1);

Erwartetes Ergebnis:

- Die Aufgabe wird erfolgreich eingefügt.

- task_id wird automatisch inkrementiert.
- date_created wird auf den aktuellen Zeitstempel gesetzt.

Überprüfung:

SELECT * FROM Task WHERE title = 'Präsentation vorbereiten';

Ergebnis:

- Eine einzelne Zeile mit den angegebenen Aufgabendaten.

Test 8: Einfügung mit ungültiger Benutzer-ID

Aktion:

Versuch, eine Aufgabe mit einer user_id einzufügen, die nicht existiert.

SQL-Anweisung:

INSERT INTO Task (user_id, title)

VALUES (999, 'Aufgabe mit ungültiger Benutzer-ID');

Erwartetes Ergebnis:

- Die Einfügung schlägt fehl.
- Eine Fehlermeldung weist auf einen Fremdschlüsseleinschränkungsverstoß hin.

Fehlermeldung:

ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`todo_app`.`Task`, CONSTRAINT `Task_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `User` (`user_id`) ON DELETE CASCADE ON UPDATE CASCADE)

Erläuterung:

- Die user_id in Task ist ein Fremdschlüssel, der auf User(user_id) verweist.
- Der Wert 999 entspricht keiner existierenden user_id.

Test 9: Löschung einer Kategorie, auf die Aufgaben verweisen

Aktion:

Löschen einer Kategorie, die von Aufgaben referenziert wird, und Beobachtung der Auswirkungen auf category_id in Task.

SQL-Anweisung:

DELETE FROM Category WHERE category_id = 1;

Erwartetes Ergebnis:

- Die Kategorie mit category_id = 1 wird gelöscht.
- In der Task-Tabelle wird category_id für Aufgaben, die auf diese Kategorie verweisen, auf NULL gesetzt aufgrund von ON DELETE SET NULL.

Überprüfung:

SELECT task_id, title, category_id FROM Task WHERE task_id = 1;

Ergebnis:

- category_id ist NULL für die Aufgabe, die zuvor auf die gelöschte Kategorie verwiesen hat.

Erläuterung:

- Die Fremdschlüsseleinschränkung spezifiziert ON DELETE SET NULL, sodass beim Löschen einer Kategorie category_id in abhängigen Aufgaben auf NULL gesetzt wird.

Test 10: Aktualisierung des Aufgabenstatus und der Priorität

Aktion:

Aktualisieren des Status und der Priorität einer bestehenden Aufgabe.

SQL-Anweisung:

- UPDATE Task
- SET status = 'completed', priority = 2, date_completed = CURRENT_TIMESTAMP
- WHERE task_id = 1;

Erwartetes Ergebnis:

- Die Aufgabe wird erfolgreich aktualisiert.
- status wird auf 'completed' gesetzt.
- priority wird auf 2 aktualisiert.
- date_completed wird auf den aktuellen Zeitstempel gesetzt.

Überprüfung:

SELECT * FROM Task WHERE task_id = 1;

Ergebnis:

- Die Aufgabe zeigt den aktualisierten Status und die aktualisierte Priorität.

Zusätzliche Tests

Test 11: Einfügung mit ungültigem Statuswert

Aktion:

Versuch, eine Aufgabe mit einem ungültigen status-Wert einzufügen.

SQL-Anweisung:

INSERT INTO Task (user_id, title, status)

VALUES (1, 'Aufgabe mit ungültigem Status', 'not_started');

Erwartetes Ergebnis:

- Die Einfügung schlägt fehl.
- Eine Fehlermeldung weist darauf hin, dass der status-Wert nicht zulässig ist.

Fehlermeldung:

ERROR 1265 (01000): Data truncated for column 'status' at row 1

Erläuterung:

- Das Feld status ist als ENUM mit den zulässigen Werten 'pending', 'in_progress' und 'completed' definiert.
- 'not_started' ist kein zulässiger Wert.

Test 12: Löschung eines Benutzers mit zugehörigen Aufgaben

Aktion:

Löschen eines Benutzers, der zugehörige Aufgaben hat, und Beobachtung der Auswirkungen.

SQL-Anweisung:

DELETE FROM User WHERE user_id = 1;

Erwartetes Ergebnis:

- Der Benutzer wird gelöscht.
- Alle Aufgaben, die user_id = 1 zugeordnet sind, werden aufgrund von ON DELETE CASCADE gelöscht.

Überprüfung:

SELECT * FROM Task WHERE user_id = 1;

Ergebnis:

- Keine Zeilen werden zurückgegeben; die zugehörigen Aufgaben wurden gelöscht.

Erläuterung:

- Die Fremdschlüsseleinschränkung in Task spezifiziert ON DELETE CASCADE für user_id.

Zusammenfassung der Testergebnisse

- Durchsetzung von Einschränkungen: Die Datenbank erzwingt korrekt UNIQUE-, NOT NULL- und Fremdschlüsseleinschränkungen.
- Kaskadenoperationen: ON DELETE CASCADE und ON DELETE SET NULL funktionieren wie erwartet und erhalten die referenzielle Integrität.
- Datenintegrität: Versuche, ungültige Daten einzufügen, werden verhindert, wodurch die Datenkonsistenz sichergestellt wird.
- Aktualisierungen und Löschungen: Daten können angemessen aktualisiert und gelöscht werden, wobei abhängige Datensätze gemäß den definierten Einschränkungen behandelt werden.