Art der Arbeit	**Projekt – Finalisierungsphase**
Kursbezeichnung	DLBDSPBDM01_D – Projekt: Data-Mart-Erstellung in SQL
Studiengang	UPS-DPSQLEA – SQL Entwicklung und Administration
Datum	29.01.2026
Verfasserin	Stefanie Ellersiek
Matrikelnummer	UPS10788065

Erklärung des Datenbankdesigns und übergreifende Hinweise

- Es wird eine relationale Datenbank namens BUCHANGE mittels MySQL erstellt
  - SQL: CREATE DATABASE Buchange;
- Bei der Erstellung der Tabellen muss die korrekte Reihenfolge berücksichtigt werden, da die Tabellen sich referenzieren. Auch für das Einfügen der Testdaten ist dies relevant. Es ergibt sich folgende Reihenfolge:
  1. Anwender
  2. Adresse
  3. Abholinfo
  4. Autor
  5. Buch
  6. Anfrage
  7. Ausleihe
  8. Rueckgabe
  9. BuchBew
  10. AnwBew
- Die technischen IDs für den Primary Key sollen automatisch erzeugt werden, daher wird AUTO_INCREMENT angewendet
- Für die Testdaten und Durchführung der darauf basierenden Testfälle wird als aktuelles Datum der 20.12.2025 angenommen

Erklärung des Datenbankdesigns und übergreifende Hinweise

- Da die Performance für die Akzeptanz der App bei den Anwendern besonders wichtig ist, wird eine Indexierung eingesetzt
  - » Die Indexierung wird für die Fremdschlüssel jeder Tabelle vorgenommen, da diese die Verbindungen zwischen den Entitäten herstellen und häufig für SELECT-Statements oder Bedingungen genutzt werden
  - » Durch die Indexierung wird das Suchen und Finden in Tabellen beschleunigt, da nicht mehr jede Zeile durchsucht werden muss, sondern mit dem Index ein direkter Verweis auf die gesuchten Werte bereitgestellt wird
- Optional kann zusätzlich eine Partitionierung der Tabellen aufgebaut werden.
  - » Die Partitionierung bringt Performance-Vorteile, in dem Daten strukturiert gespeichert werden, bspw. die Gruppierung von Ausleihvorgänge nach Jahren
  - » Da bei Beginn der App noch nicht so viele Ausleihvorgänge vorliegen, wird dies zunächst nicht implementiert
  - » Wenn es trotz Indexierung zu Performance-Problemen kommt, wäre dies jedoch eine sinnvolle Weiterentwicklung der Datenbank
- Es werden Transaktionsmodelle aufgebaut, die die Datenintegrität sicherstellen
  - » Immer wenn aufgrund einer Anwenderaktivität mehrere SQL-Statements ausgeführt werden müssen, werden diese in einer Transaktion gebündelt. Dies stellt konsistente Daten in den Tabellen sicher, da die gesamte Transaktion abbricht, wenn ein Fehler in einem Statement auftritt
  - » Durch die Wahl eines geeigneten Isolationslevels ist eine Optimierung der App-Performance möglich, bspw. durch weniger Sperren

Entität ANWENDER

Es wird eine Tabelle namens Anwender erstellt:

- Die Tabelle speichert alle registrierten Anwender mit Vor- & Nachname, E-Mail, Profilfoto und ob er die Admin-Rolle besitzt
- Pflichtangaben für den Anwender sind Vorname, Nachname und E-Mail, daher dürfen die Attribute nicht NULL sein
- Für die Administrator-Rolle wird der DEFAULT-Wert FALSE definiert, da die meisten Anwender nicht Administrator sind

SQL-Statements:

- CREATE TABLE Anwender (…) erstellt die Tabelle
- INSERT INTO Anwender (Spalten) VALUES (….) fügt die Testdaten ein

Testfall:

- Ein **neuer Anwender** registriert sich und wird mit folgenden Daten in der Tabelle Anwender angelegt:

INSERT INTO Anwender (AID,PasswortHash, Vorname, Nachname, EMail, Profilfoto, Administrator)VALUES (11,'gehashtesPasswort', 'Alina', 'Nordmann', 'alina.nordmann@gmail.com', NULL, FALSE);

Screenshot nach Ausführung des Testfalls

AID	PasswortHash	Vorname	Nachname	EMail	Profilfoto	Administrator
4	hashed_pass_4	David	Meyer	david.meyer@web.de	NULL	0
5	hashed_pass_5	Eva	Wagner	eva.wagner@web.de	NULL	0
6	hashed_pass_6	Felix	Hoffmann	felix.hoffmann@web.de	NULL	0
7	hashed_pass_7	Greta	Klein	greta.klein@web.de	NULL	1
8	hashed_pass_8	Hannes	Fischer	hannes.fischer@web.de	NULL	0
9	hashed_pass_9	Isabel	Krüger	isabel.krueger@web.de	NULL	0
10	hashed_pass_10	Jonas	Vogel	jonas.vogel@web.de	NULL	0
11	gehashtesPass…	Alina	Nordmann	alina.nordmann@gmail.com	NULL	0

Entität Adresse

Es wird eine Tabelle namens Adresse erstellt

- Alle Attribute der Tabelle sind Pflichtbestandteile und dürfen nicht NULL sein
- Die Adresse wird als eigene Tabelle gespeichert und nicht in der Tabelle Anwender, da eine Adresse mehreren Anwendern zugeordnet werden kann. Ein Anwender darf aber nur eine Adresse haben, daher wird das Attribut mit UNIQUE definiert
- Zudem haben die einzelnen Attribute der Adresse eine Abhängigkeit untereinander, die eine Ausgliederung in eine eigene Tabelle im Sinne der Kriterien der dritten Normalform nahelegen

SQL-Statements:

- CREATE TABLE Adresse (…) erstellt die Tabelle
- INSERT INTO Adresse (…) fügt die Testdaten ein

Testfall:

- Alina gibt im Zuge der Registrierung ihre Adresse an.
- Diese wird in der Tabelle Adresse eingetragen:

```
INSERT INTO Adresse (AdressID,Straße, Hausnummer, PLZ, Ort, Land,
Breitengrad, Laengengrad, AID) VALUES ('Ebendiekstraße', 8, '48488', 'Listrup‘,
'Deutschland', 52.531400, 7.487800,11);
```

Screenshot nach Ausführung des Testfalls

AdressID	Straße	Hausnummer	PLZ	Ort	Land	Breitengrad	Laengengrad	AID
1	Dorfstraße	23	48488	Listrup	Deutschland	52.366700	7.516700	1
2	Alter Mühlenweg	5	48488	Listrup	Deutschland	52.367000	7.517000	2
3	Am Emstal	12	48488	Listrup	Deutschland	52.367200	7.516500	3
4	Am Haferkamp	8	48488	Listrup	Deutschland	52.366500	7.517500	4
5	Listruper Straße	15	48488	Listrup	Deutschland	52.367500	7.516000	5
6	Dorfstraße	83	48488	Listrup	Deutschland	52.366800	7.517200	6
7	Mühlenweg	7	48488	Listrup	Deutschland	52.367100	7.516800	7
8	Kirchstraße	4	48488	Listrup	Deutschland	52.366900	7.517100	8
9	Emsbürener Landstraße	22	48488	Listrup	Deutschland	52.367300	7.516300	9
10	Listruper Feldweg	10	48488	Listrup	Deutschland	52.366600	7.517300	10
11	Ebendiekstraße	8	48488	Listrup	Deutschland	52.531400	7.487800	11

Entität ABHOLINFO

Es wird eine Tabelle namens Abholinfo erstellt

- Bevor die Tabelle erstellt werden kann, ist darauf zu achten, dass die referenzierte Tabelle Adresse existiert
- Pflichtangaben sind nur, ob eine Abholung im Dorfgemeinschaftshaus (DGH) möglich ist sowie der Fremdschlüssel AdressID, daher dürfen diese Attribute nicht NULL sein. Die Attribute zu den Abholzeiten und Wochentagen müssen nur gefüllt sein, wenn eine Abholung im DGH nicht möglich also FALSE ist. Zu diesem Zweck wird der CONSTRAINT check_DGH im CREATE TABLE Statement eingefügt
- Da es zu einer Adresse unterschiedliche Abholinformationen geben kann (bspw. im Mehrparteienhaus mit unterschiedlichen Anwendern) ist keine UNIQUE Bedingung erforderlich

SQL-Statements:

- CREATE TABLE Abholinfo (…) erstellt Tabelle
- INSERT INTO Abholinfo (…) fügt Testdaten ein

Entität ABHOLINFO

Testfall:

- Nach der Angabe der Adresse fordert die App Alina auf, Infos zur Abholung der Bücher, die sind anbieten möchte, anzugeben
- Sie bietet die Hinterlegung im Dorfgemeinschaftshaus und die persönliche Abholung Donnerstags zwischen 18 und 19 Uhr an.

```
INSERT INTO Abholinfo (AbholID, DGH, AbholzeitAb, AbholzeitBis, Wochentage, AdressID) VALUES (11, TRUE, '18:00:00‘, '19:00:00', 'Do', 11);
```

Screenshot nach Ausführung des Testfalls

AbholID	DGH	AbholzeitAb	AbholzeitBis	Wochentage	AdressID
1	1	16:00:00	19:00:00	Mo,Mi,Fr	1
2	0	10:00:00	14:00:00	Sa	2
3	1	17:00:00	20:00:00	Di,Do	3
4	1	15:00:00	18:00:00	Mo-Fr	4
5	0	09:00:00	12:00:00	So	5
6	1	18:00:00	21:00:00	Mo,Do	6
7	1	14:00:00	17:00:00	Mi,Fr	7
8	0	11:00:00	15:00:00	Sa,So	8
9	1	16:30:00	19:30:00	Mo-Do	9
10	1	15:00:00	18:00:00	Fr	10
11	1	18:00:00	19:00:00	Do	11

Entität AUTOR

Es wird eine Tabelle namens Autor erstellt

- Pflichtangaben für den Autor sind Vorname und Nachname, daher dürfen die Attribute nicht NULL sein
- Alternativ hätte der Autor Teil der Tabelle BUCH sein können, allerdings hätte eine direkte Abhängigkeit zwischen den Attributen Vor- und Nachname den Anforderungen der 3. Normalform widersprochen
- Um Dubletten bei den Autoren zu verhindern, bspw. wenn 2 Anwender gleichzeitig ein Buch mit einem neuen Autor erfassen, wird ein Unique-Constraint für die Kombination aus Vor- und Nachnamen erstellt

SQL-Statements:

- CREATE TABLE Autor (…) erstellt die Tabelle
- INSERT INTO Autor (…) fügt die Testdaten ein

Testfall:

- Zur Ausleihe bietet Alina das Buch „Der Herr der Ringe“ an.
- Es wird ein neuer Autor in der Tabelle AUTOR eingetragen:

INSERT INTO Autor (AutorID, Vorname, Nachname)VALUES (12,'J.R.R', 'Tolkien‘);

Screenshot nach Ausführung des Testfalls

AutorID	Vorname	Nachname
1	Jane	Austen
2	Ernest	Hemingway
3	Jacques	Berndorf
4	Joachim	Gauck
5	Caroline	Wahl
6	John	Irving
7	Chloe	Dalton
8	Jochen	Gutsch
9	Maxim	Leo
10	Henning	Mankell
11	Martin	Suter
12	J.R.R	Tolkien

Entität BUCH

Es wird eine Tabelle namens Buch erstellt

- Bevor die Tabelle erstellt werden kann, ist darauf zu achten, dass die referenzierte Tabelle Autor und Abholinfo existiert
- Pflichtangaben für das Buch sind Titel, Genre, Jahr, Verlag, Zustand, Sprache, Leihdauer und Autor, daher dürfen die Attribute nicht NULL sein. Ebenso müssen Informationen zur Abholung angegeben werden. Da jedes Buch von einem Anwender erfasst wird, muss auch zwangsläufig ein Anwender zugeordnet werden können. Für die Fremdschlüssel gilt, dass sowohl Autor als auch Anwender mehrere Bücher zugeordnet werden können, daher wird kein UNIQUE ergänzt
- Die Informationen zum Autor werden gesondert gespeichert, da Vor- und Nachname nicht direkt von der BuchID abhängen, sondern vom Autor. Somit wird den Anforderungen der 3. Normalform entsprochen

SQL-Statements:

- CREATE TABLE Buch (…) erstellt die Tabelle
- INSERT INTO Buch (…) fügt Testdaten ein

Entität BUCH

Testfall:

- Da Alina das Buch „Herr der Ringe“ zur Ausleihe anbietet, werden folgende Informationen in die Tabelle Buch eingetragen:

  INSERT INTO Buch (BuchID, Titel, Genre, Jahr, Verlag, Zustand, Sprache, Leihdauer, AutorID, AID, AbholID) VALUES (12, 'Der Herr der Ringe', 'Klassiker', 1958, 'Klett-Cotta Verlag', 'wie neu', 'Deutsch', 30, 12, 11, 11);

Screenshot nach Ausführung des Testfalls

BuchID	Titel	Genre	Jahr	Verlag	Zustand	Sprache	Leihdauer	AutorID	AID	AbholID
1	Stolz und Vorurteil	Klassiker	1913	Reclam	wie neu	Deutsch	30	1	1	1
2	Der alte Mann und das Meer	Klassiker	1952	Rowohlt	leichte Gebrauchsspuren	Deutsch	21	2	2	2
3	Eifel-Blues	Krimi	1989	Grafit	leichte Gebrauchsspuren	Deutsch	21	3	3	3
4	Erschütterungen	Sachbuch	2023	Penguin	wie neu	Deutsch	14	4	4	4
5	Die Assistentin	Roman	2022	Rowohlt	wie neu	Deutsch	28	5	5	5
6	Straße der Wunder	Roman	1978	Diogenes	starke Gebrauchsspuren	Deutsch	30	6	6	6
7	Hase und Ich	Sachbuch	2023	Klett-Cotta	wie neu	Deutsch	14	7	7	7
8	Frankie	Roman	2023	Penguin	wie neu	Deutsch	21	8	8	8
9	Mörder ohne Gesicht	Krimi	1991	dtv	starke Gebrauchsspuren	Deutsch	21	9	9	9
10	Gott des Waldes	Krimi	2025	C.H.Beck	beschädigt	Deutsch	30	10	1	1
11	Melody	Roman	2023	Diagones	wie neu	Deutsch	20	11	10	10
12	Der Herr der Ringe	Klassiker	1958	Klett-Cott...	wie neu	Deutsch	30	12	11	11

View ViewVerfuegbareBuecher

Es wird eine View für die Anzeige der verfügbaren Bücher erstellt, die auf einer Karte abbildbar sind:

- Es wird eine View erstellt, da alle benötigten Daten bereits in Tabellen gespeichert sind. Somit wird auf die Speicherung von redundanten Daten verzichtet
- Es sollen nur Bücher angezeigt werden, die keine aktive Ausleihe haben. Hierdurch wird sichergestellt, dass keine Bücher doppelt ausgeliehen werden können
- Es dürfen ebenso nur Bücher angezeigt werden, deren Verleiher nicht gesperrt ist, aufgrund der Geschäftslogik, dass gesperrte Anwender keine Bücher verleihen dürfen (Achtung: Daher kann diese View erst erstellt werden, wenn die Sichten für Sperren erstellt worden sind, vgl. Folien 22 & 23)
- Es sollen alle für den Ausleihenden benötigten Informationen aus unterschiedlichen Tabellen angezeigt werden. Für die Abbildung auf der Karte muss der Längen- und Breitengrad in der View enthalten sein

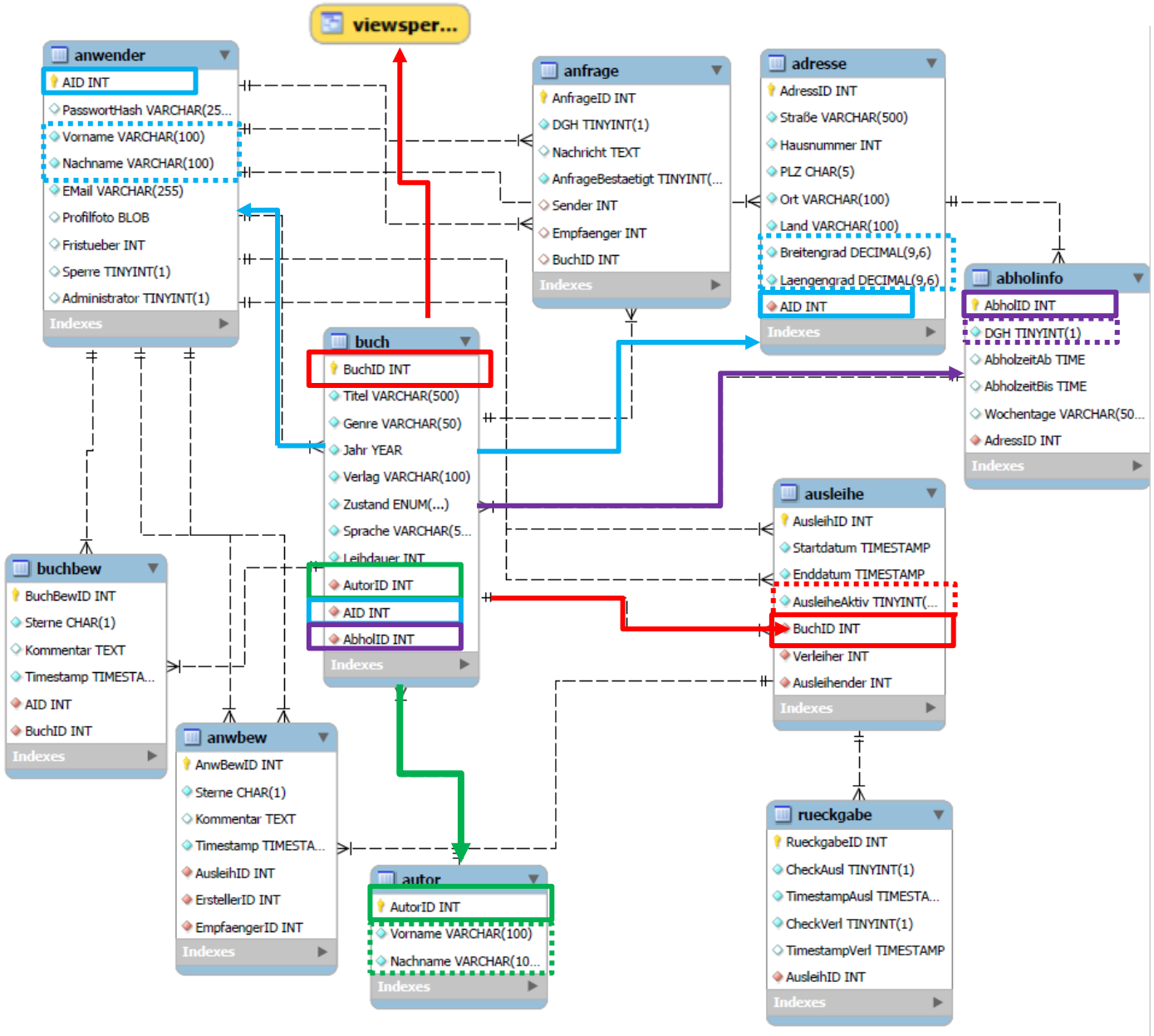
SQL-Statement:

Screenshot nach Ausführung des Statements

BuchID	Titel	Genre	Jahr	Verlag	Zustand	Sprache	Leihdauer	AutorName	Breitengrad	Laengengrad	Verleiher	DGH
2	Der alte Mann und das Meer	Klassiker	1952	Rowohlt	leichte Gebrauchsspuren	Deutsch	21	Ernest Hemingway	52.367000	7.517000	Ben Becker	0
6	Straße der Wunder	Roman	1978	Diogenes	starke Gebrauchsspuren	Deutsch	30	John Irving	52.366800	7.517200	Felix Hoffmann	1
8	Frankie	Roman	2023	Penguin	wie neu	Deutsch	21	Jochen Gutsch	52.366900	7.517100	Hannes Fischer	0
9	Mörder ohne Gesicht	Krimi	1991	dtv	starke Gebrauchsspuren	Deutsch	21	Maxim Leo	52.367300	7.516300	Isabel Krüger	1
11	Melody	Roman	2023	Diagones	wie neu	Deutsch	20	Martin Suter	52.366600	7.517300	Jonas Vogel	1

View ViewVerfuegbareBuecher

- Das Diagramm zeigt aus welchen Tabellen die benötigten Attribute mittels Join gezogen werden
- Für den Ausschluss von Büchern, deren Ausleihe noch aktiv ist, wird die Tabelle Ausleihe geprüft
- Für den Ausschluss von Büchern, deren Anwender gesperrt sind, wird zusätzlich die View ViewSperre geprüft



Entität Anfrage

Es wird eine Tabelle namens Anfrage erstellt

- Bevor die Tabelle Anfrage erstellt werden kann, muss sichergestellt sein, dass es die Tabellen Anwender und Buch gibt, da auf diese Tabellen referenziert wird
- Pflichtbestandteile der Anfrage sind die Info zur Abholung im Dorfgemeinschaftshaus und das Füllen der Anfragebestätigung. Initial ist diese FALSE und wird erst durch Aktivität des Verleihers auf TRUE gesetzt

SQL-Statements:

- CREATE TABLE Anfrage (…) erstellt die Tabelle
- INSERT INTO Anfrage (…) fügt Testdaten ein
- UPDATE Anfrage setzt AnfrageBestätigt auf TRUE

Testfall:

- Alina möchte das Buch „Erschütterungen “ ausleihen:

  INSERT INTO Anfrage (AnfrageID, DGH, Nachricht, AnfrageBestaetigt, Sender, BuchID) VALUES (20, TRUE, 'Ich habe das Buch empfohlen bekommen.', FALSE, 11, 4);

- David Meyer ist Besitzer des Buches und bestätigt die Anfrage von Alina:

  UPDATE Anfrage SET AnfrageBestaetigt = TRUE WHERE AnfrageID = 20;

Screenshot nach Ausführung des Testfalls

AnfrageID	DGH	Nachricht	AnfrageBestaetigt	Sender	BuchID
1	1	Ich würde das Buch gerne ausleihen.	0	10	1
2	0	Ist das Buch noch verfügbar?	0	7	2
3	1	NULL	0	6	3
4	1	Großes Interesse an dem Buch	1	5	1
5	0	NULL	1	4	2
6	1	NULL	1	2	3
7	1	Bin sehr vorsichtig mit Büchern	1	8	4
8	1	NULL	1	3	5
9	0	NULL	1	2	6
10	0	Grüß Dich!	1	1	7
11	1	NULL	1	7	1
12	1	NULL	1	6	9
13	0	Moin!	1	9	8
14	0	NULL	1	10	6
15	1	Tolles Buch!	1	7	2
16	1	NULL	1	5	7
17	1	NULL	1	8	4
18	1	NULL	1	3	5
19	1	NULL	1	9	1
20	1	Ich habe das Buch empfohlen beko…	1	11	4

Entität Ausleihe

Es wird eine Tabelle namens Ausleihe erstellt

- Alle Attribute der Tabelle sind verpflichtend zu füllen. Initital wird AusleiheAktiv mit TRUE befüllt, da nach Bestätigung der Anfrage die Tabelle Ausleihe befüllt wird und dann die Ausleihe anfänglich aktiv ist
- Das Attribut AusleiheAktiv wird auf FALSE gesetzt, wenn in der Tabelle Rückgabe die Attribut CheckAusl und CheckVerl auf TRUE gesetzt sind. Das UPDATE-Statement wird durch die Bestätigung des Verleihers zum Erhalt der Rückgabe in der App ausgelöst
- Es gibt keine Attribute für die Erinnerungszeitpunkte vor Ablauf der Frist, da diese aus dem Enddatum ermittelt werden können
- Bei den Fremdschlüsselbeziehungen ist keine UNIQUE-Bedingung erforderlich, da alle Fremdschlüssel mehrfach vorkommen können
- Es werden die Attribute BuchID, Verleiher und Ausleihender explizit gespeichert anstatt bspw. einfach die AnfrageID als Fremdschlüssel. Der Grund ist, dass der Ausleihprozess die zentrale Informationsquelle ist, welche Bücher, von welchen Anwendern ausgeliehen oder verliehen wurden. Die Speicherung der Attribute in dieser Tabelle vereinfacht die Abfrage der Informationen sehr bspw. für die Anzeige der Statistiken für Buchausleihen etc.

SQL-Statements:

- CREATE TABLE Ausleihe (…) erstellt Tabelle
- INSERT INTO Ausleihe (…) fügt Testdaten ein
- UPDATE Ausleihe

Entität Ausleihe

Testfall:

- Nachdem David die Anfrage bestätigt hat, wird der Ausleihprozess gestartet und folgende Daten in der Tabelle Ausleihe eingetragen:

  INSERT INTO Ausleihe (AusleihID, Startdatum, Enddatum, AusleiheAktiv, BuchID, Verleiher, Ausleihender) VALUES (17, '2025-11-10', '2025-10-31‘, TRUE, 4, 4, 11);

Screenshot nach Ausführung des Testfalls

AusleihID	Startdatum	Enddatum	AusleiheAktiv	BuchID	Verleiher	Ausleihender
1	2025-10-05	2025-10-21	0	1	1	5
2	2025-10-07	2025-11-06	0	2	2	4
3	2025-10-12	2025-11-09	0	3	3	2
4	2025-10-15	2025-11-12	0	4	4	8
5	2025-10-18	2025-11-15	1	5	5	3
6	2025-10-18	2025-11-14	0	6	6	2
7	2025-10-20	2025-11-03	1	7	7	1
8	2025-10-22	2025-11-21	0	1	1	7
9	2025-11-10	2025-12-01	0	9	9	6
10	2025-11-11	2025-12-02	0	8	8	9
11	2025-11-27	2025-12-27	0	6	6	10
12	2025-12-10	2025-12-31	0	2	2	7
13	2025-12-11	2025-12-25	1	7	7	5
14	2025-12-18	2026-01-01	1	4	4	8
15	2025-12-19	2026-01-16	1	5	5	3
16	2025-12-19	2026-01-18	1	1	1	9
17	2025-11-10	2025-10-31	1	4	4	11

Entität Rückgabe

Es wird eine Tabelle namens Ausleihe erstellt

- Bevor die Tabelle Rückgabe erstellt werden kann, muss die Tabelle Ausleihe erstellt worden sein
- Alle Attribute der Tabelle sind verpflichtend zu füllen mit Ausnahme des Datums für den Check des Verleihers. Initital wird CheckAusl mit TRUE gefüllt, da der Eintrag in der Tabelle mit der Bestätigung des Ausleihenden erfolgt, dass die Rückgabe erfolgt ist. CheckVerl wird initial mit FALSE gefüllt, da die Bestätigung in der Regel noch erfolgen muss
- Jede AusleihID kann in der Tabelle nur einmal vorkommen, daher ist sie UNIQUE. Zu jeder Rückgabe muss es eine AusleihID geben
- Wenn der Verleiher die Rückgabe des Buches bestätigt, wird in der Tabelle Rueckgabe das Attribut CheckVerl = TRUE gesetzt und auch das Datum der Bestätigung gesetzt

SQL-Statements:

- CREATE TABLE Rueckgabe (...) erstellt die Tabelle
- INSERT INTO Rueckgabe (…) fügt Testdaten ein
- UPDATE Rueckgabe (…)

Entität Rückgabe

Testfall:

- Alina gibt das Buch am 25.10.2025 zurück. Daraufhin wird folgender Eintrag in der Tabelle Rueckgabe gesetzt:

  INSERT INTO Rueckgabe (RueckgabeID, CheckAusl, TimestampAusl, CheckVerl, TimestampVerl, AusleihID) VALUES(12, TRUE, '2025-10-25 15:00:00‘, FALSE, NULL, 17);

- David bestätigt den Erhalt der Rückgabe am 27.10.2025. Daraufhin wird die Tabelle Rueckgabe wie folgt aktualisiert:

  UPDATE Rueckgabe SET CheckVerl = TRUE, TimestampVerl = '2025-10-25 16:14:23‘ WHERE RueckgabeID = 12 AND AusleihID = 17;

- Mit der Bestätigung von David wird auch eine Aktualisierung der Tabelle Ausleihe ausgelöst:

  UPDATE Ausleihe SET AusleiheAktiv = FALSE WHERE AusleihID = 17 AND EXISTS (SELECT 1 FROM Rueckgabe WHERE Rueckgabe.AusleihID = Ausleihe.AusleihID AND Rueckgabe.CheckAusl = TRUE AND Rueckgabe.CheckVerl = TRUE);

RueckgabeID	CheckAusl	TimestampAusl	CheckVerl	TimestampVerl	AusleihID
1	1	2025-10-21	1	2025-10-22	1
2	1	2025-11-06	1	2025-11-07	2
3	1	2025-11-09	1	2025-11-11	3
4	1	2025-11-12	1	2025-11-13	4
5	1	2025-11-08	0	NULL	5
6	1	2025-11-15	1	2025-11-15	8
7	1	2025-11-14	1	2025-11-15	6
8	1	2025-12-02	1	2025-12-03	9
9	1	2025-12-11	1	2025-12-12	10
10	1	2025-12-18	1	2025-12-18	11
11	1	2025-12-18	1	2025-12-18	12
12	1	2025-10-25	1	2025-10-25	17

AusleihID	Startdatum	Enddatum	AusleiheAktiv	BuchID	Verleiher	Ausleihender
1	2025-10-05	2025-10-21	0	1	1	5
2	2025-10-07	2025-11-06	0	2	2	4
3	2025-10-12	2025-11-09	0	3	3	2
4	2025-10-15	2025-11-12	0	4	4	8
5	2025-10-18	2025-11-15	1	5	5	3
6	2025-10-18	2025-11-14	0	6	6	2
7	2025-10-20	2025-11-03	1	7	7	1
8	2025-10-22	2025-11-21	0	1	1	7
9	2025-11-10	2025-12-01	0	9	9	6
10	2025-11-11	2025-12-02	0	8	8	9
11	2025-11-27	2025-12-27	0	6	6	10
12	2025-12-10	2025-12-31	0	2	2	7
13	2025-12-11	2025-12-25	1	7	7	5
14	2025-12-18	2026-01-01	1	4	4	8
15	2025-12-19	2026-01-16	1	5	5	3
16	2025-12-19	2026-01-18	1	1	1	9
17	2025-11-10	2025-10-31	0	4	4	11

Entität Buch-Bewertung

Es wird eine Tabelle namens BUCHBEW erstellt

- Vor der Tabelle BuchBew müssen die Tabellen Anwender und Buch erstellt sein
- Alle Attribute der Tabelle sind verpflichtend zu füllen mit Ausnahme des Kommentars zur Bewertung
- Alle Fremdschlüssel dürfen mehrfach in der Tabelle vorkommen, da es sowohl von einem Anwender als auch zu einem Buch mehrere Bewertungen geben kann

SQL-Statements:

- CREATE TABLE BuchBew (...) erstellt die Tabelle
- INSERT INTO BuchBew (…) fügt Testdaten ein

Testfall:

- Alina fand das Buch „Erschütterungen“ sehr interessant.
- Daher vergibt sie eine 5 Sterne-Bewertung:

  INSERT INTO BuchBew (BuchBewID, Sterne, Kommentar, Timestamp, AID, BuchID) VALUES (11, ‚5‘, ‘Super spannend!‘,`'2025-10-28 22:02:14‘, 11, 4);

Screenshot nach Ausführung des Testfalls

BuchBewID	Sterne	Kommentar	Timestamp	AID	BuchID
1	5	NULL	2025-10-21 14:30:00	5	10
2	4	NULL	2025-11-06 16:45:00	4	9
3	5	Hervorragend	2025-11-09 11:20:00	2	8
4	2	NULL	2025-11-12 17:10:00	8	6
5	5	NULL	2025-11-08 15:25:00	3	2
6	5	NULL	2025-11-14 12:35:00	2	7
7	2	Enttäschend	2025-11-15 18:50:00	7	6
8	2	NULL	2025-12-11 16:15:00	9	1
9	4	Erwartungen getroffen	2025-12-18 13:30:00	1	2
10	4	NULL	2025-12-18 17:55:00	7	3
11	5	Super spannend!	2025-10-28 22:02:14	11	4

Entität Anwender-Bewertung

Es wird eine Tabelle namens AnwBew erstellt

- Vor der Tabelle BuchBew müssen die Tabellen Anwender und Ausleihe erstellt sein
- Alle Attribute der Tabelle sind verpflichtend zu füllen mit Ausnahme des Kommentars zur Bewertung
- Alle Fremdschlüssel dürfen mehrfach in der Tabelle vorkommen, da es von einem Anwender mehrere Bewertungen geben kann und er auch von anderen Anwendern mehrere Bewertungen erhalten kann. Auch die Ausleih-ID kann mehrfach vorkommen, da es zu einer Ausleihe von beiden Anwendern Bewertungen geben kann
- Die Attribute ErstellerID und EmpfaengerID könnten auch weggelassen werden und stattdessen über die AusleihID per Join ermittelt werden. Da die empfangenen und vergebenen Bewertungen aber im Profil der Anwender angezeigt werden sollen, ist die Speicherung der Daten direkt in der Tabelle effizient

SQL-Statements:

- CREATE TABLE AnwBew (…) erstellt die Tabelle
- INSERT INTO AnwBew (…) fügt Testdaten ein

Entität Anwender-Bewertung

Testfall:

- David Meyer ist ebenfalls sehr zufrieden mit dem Ausleihprozess und schreibt für Alina eine Bewertung:

```
INSERT INTO AnwBew (AnwBewID, Sterne, Kommentar, Timestamp, AusleihID, ErstellerID, EmpfaengerID)
VALUES (11, '5', 'Sehr nett', '2025-10-28 23:00:05‘, 17, 4, 11);
```

Screenshot nach Ausführung des Testfalls

AnwBewID	Sterne	Kommentar	Timestamp	AusleihID	ErstellerID	EmpfaengerID
1	5	Einwandfrei	2025-10-21 14:30:00	1	5	5
2	5	NULL	2025-11-06 16:45:00	2	4	4
3	5	Super nett	2025-11-09 11:20:00	3	2	2
4	5	NULL	2025-11-12 17:10:00	4	8	8
5	5	NULL	2025-11-05 18:50:00	8	7	7
6	5	Unkompliziert	2025-11-14 12:35:00	6	2	2
7	5	NULL	2025-12-10 16:15:00	9	6	6
8	5	NULL	2025-12-02 17:55:00	10	9	9
9	5	NULL	2025-12-27 13:30:00	11	1	1
10	5	NULL	2025-12-24 14:30:00	12	7	7
11	5	Sehr nett	2025-10-28 23:00:05	17	4	11

VIEW ViewFristueber

Es wird eine View für die Fristüberschreitungen erstellt:

- Die Anzahl der Fristüberschreitungen soll öffentlich im Profil der Anwender abgebildet werden. Ebenso soll geprüft werden, ob eine aktuelle Fristüberschreitung vorliegt. Dies ist relevant, da dies zu einer Sperre des Anwenders führt
- Die Fristüberschreitungen beziehen sich zwar auf den Anwender sind aber dynamische Zustände. Daher werden sie nicht in der Stammdatentabelle zum Anwender gespeichert, sondern zur Laufzeit berechnet
- Dadurch wird sichergestellt, dass die Fristüberschreitungen eines Anwenders jederzeit aktuell und konsistent mit den zugrunde liegenden Ausleihdaten sind
- Die Aktuelle Fristüberschreitung wird gesetzt, wenn für einen Ausleihvorgang das Attribut AusleiheAktiv = TRUE ist und das aktuelle Datum > dem Enddatum ist
- Die Anzahl der Fristüberschreiten wird gezählt, wenn für einen Ausleihvorgang des Anwenders das Rückgabedatum durch den Ausleihenden > dem Enddatum der Ausleihe ist oder die Bedingungen der aktuellen Fristüberschreitungen erfüllt sind

SQL-Statement: CREATE ViewFristueber (…) erstellt die View

Screenshot nach Ausführung des Statements

	AID	AnzFristueber	AktFristueber
▶	1	1	1
	2	0	0
	3	1	1
	4	0	0
	5	0	0
	6	1	0
	7	0	0
	8	0	0
	9	1	0
	10	0	0

VIEW ViewSperre

Es wird eine View für die Sperren erstellt, die sich aufgrund von Fristüberschreitungen ergeben:

- Die Sperren beziehen sich analog der Fristüberschreitungen zwar auf den Anwender, ergeben sich aber dynamisch aufgrund aktueller Zustände. Daher wird keine physische Tabelle erstellt, sondern eine View
- Sperren werden gemäß der definierten Geschäftslogik gesetzt, wenn entweder eine aktuelle Fristüberschreitung vorliegt (-> AktSperre) oder die Anzahl der Fristüberschreitungen insgesamt 10 übersteigt (->MaxSperre)
- Um die Prüfung auf eine vorliegende Sperre des Anwenders (entweder AktSperre oder MaxSperre) zu vereinfachen werden beide Informationen in „Sperre“ konsolidiert

SQL-Statement:

- CREATE VIEW ViewSperre (…) erstellt die View

Screenshot nach Ausführung des Statements

AID	MaxSperre	AktSperre	Sperre
1	0	1	1
2	0	0	0
3	0	1	1
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	0	0	0
9	0	0	0
10	0	0	0

SELECT-Statements für Anzeige von Statistiken

Alina möchte jetzt auswerten, welche Ausleihen sie bisher getätigt hat:

- Bevorzugt sie Bücher eines bestimmten Genres? Sie möchte sehen, wie viele Bücher sie je Genre ausgeliehen hat:

  SELECT b.Genre, COUNT(*) AS Anzahl FROM Ausleihe auJOIN Buch b ON b.BuchID = au.BuchID WHERE au.Ausleihender = 11 GROUP BY b.Genre;

	Genre	Anzahl
▶	Sachbuch	1

- Bevorzugt sie Bücher eines bestimmten Autors? Sie möchte sehen, wie viele Bücher sie je Autor ausgeliehen hat:

  SELECT aut.AutorID AS ID, aut.Vorname AS Autor_Vorname, aut.Nachname AS Autor_Nachname, COUNT(*) AS Anzahl FROM Ausleihe au JOIN Buch b ON b.BuchID = au.BuchID JOIN Autor aut ON aut.AutorID = b.AutorID WHERE au.Ausleihender = 11 GROUP BY aut.AutorID, aut.Vorname, aut.Nachname ORDER BY Anzahl DESC;

	ID	Autor_Vorname	Autor_Nachname	Anzahl
▶	4	Joachim	Gauck	1

- Leiht sie häufig von den gleichen Personen Bücher aus? Sie möchte sehen, wie viele Bücher sie von anderen Anwendern ausgeliehen hat:

  SELECT besitzer.AID AS ID, besitzer.Vorname AS Vorname, besitzer.Nachname AS Nachname, COUNT(au.AusleihID) AS Anzahl FROM Ausleihe au INNER JOIN Buch b ON b.BuchID = au.BuchID INNER JOIN Anwender ausleiher ON ausleiher.AID = au.Ausleihender INNER JOIN Anwender besitzer ON besitzer.AID = b.AID WHERE au.Ausleihender = 11 GROUP BY besitzer.AID, besitzer.Vorname, besitzer.Nachname ORDER BY Anzahl DESC;

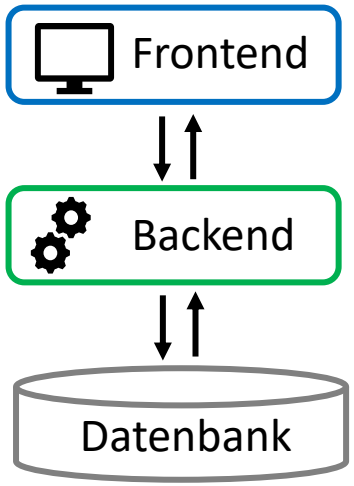
	ID	Vorname	Nachname	Anzahl
▶	4	David	Meyer	1

Sicherstellung der Datenkonsistenz durch Architektur

Weitere Prüfungen sollten im Frontend sowie im Backend zur Datenbank verortet werden:

- Bspw. leere Dimensionen und Leerzeichen (z.B. ‘‘ oder ‘ ‘) können auf Tabellenebene nur sehr aufwändig mit Triggern oder CHECK(TRIM(...) <> ,‘) für jede Spalte bzw. Tabelle eingesetzt werden
- Durch die eindeutige Verortung von Verantwortlichkeiten wird die Wartbarkeit und Sicherheit des Systems erhöht

Das folgende Architektur-Schaubild verdeutlicht die jeweiligen Verantwortlichkeiten:



» Ablauflogik für Benutzereingaben
» Definition von Pflichtfeldern

» Validierung der Eingabe, z. B. Trimmen, leere Dimensionen, E-Mailformat
» Eingabelogik, z. B. Leihdauer > 0, Ausleiher ≠ Verleiher, Startdatum < Enddatum

» Persistierung der Daten
» Sicherstellung der Datenkonsistenz durch Fremdschlüssel, Constraints, Trigger, etc.

Sicherstellung der Datenkonsistenz durch Randfalltests

Um die Robustheit der bisher aufgeführten Tabellen sicherzustellen, werden ergänzende Tests durchgeführt:

- NULL-Werte für Pflichtfelder: z. B. Vor- und Nachname sind Pflichtfelder
  - » INSERT INTO Anwender (AID,PasswortHash, Vorname, Nachname, EMail, Profilfoto, Administrator) VALUES (12,'PWHash', NULL , NULL, 'test@gmail.com', NULL, FALSE);
- UNIQUE-Verletzungen: z. B. Ein Anwender darf nur eine Adresse haben
  - » INSERT INTO Adresse (AdressID,Straße, Hausnummer, PLZ, Ort, Land, Breitengrad, Laengengrad, AID) VALUES (13,'Pumuckelstr.', 8, '48431', 'Rheine', 'Deutschland', 52.531400, 7.487800, 11);
  - » INSERT INTO Autor (AutorID, Vorname, Nachname)VALUES (13,'Joachim', 'Gauck');
- CONSTRAINT-Verletzungen: z. B. Abholzeiten müssen gefüllt sein, wenn DGH = FALSE
  - » INSERT INTO Abholinfo (AbholID, DGH, AbholzeitAb, AbholzeitBis, Wochentage, AdressID) VALUES (13, FALSE, NULL, NULL, NULL, 11);
- Datentyp-Verletzungen: z. B. Sterne erlauben nur CHAR(1)
  - » INSERT INTO BuchBew (BuchBewID, Sterne, Kommentar, Timestamp, AID, BuchID) VALUES (12, '10', 123,'2025-10-28', 11, 4);
- Massendaten-Test: Da beim Ausführen vieler einzelner INSERT-Statements die Datenbank-Verbindung abbricht, wird ein rekursives Statement genutzt und das Rekursionslimit kurzfristig erhöht
  - » INSERT INTO Anwender (…) WITH RECURSIVE seq AS (…) , Vgl. SQL-Skript “03_TEST_MASSDATA”

Sicherstellung der Datenkonsistenz durch Trigger vor Einfügen von Anwender-Bewertungen

Geschäftsregel:

- Um die Qualität der Anwender-Bewertungen sicherzustellen, darf eine Bewertung nur nach abgeschlossener Ausleihe erfolgen. Daher muss eine Prozessregel mit folgender Logik eingeführt werden: Prüfe, ob in der Tabelle Ausleihe der Status „AusleiheAktiv“ FALSE beträgt, bevor ein Eintrag in der Tabelle AnwBew erlaubt wird.

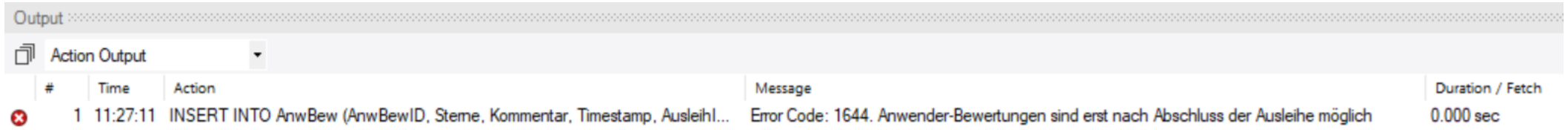
SQL-Statements:

- CREATE TRIGGER CheckAusleihe…

Testfall:

- David Meyer hat von Hannes Fischer das Buch Frankie ausgeliehen. Die Ausleihe ist noch aktiv. Dennoch möchte Hannes bereits eine Bewertung zu David schreiben. Dies ist nicht erlaubt, da die Ausleihe vor Abgabe der Bewertung abgeschlossen sein muss: INSERT INTO AnwBew (AnwBewID, Sterne, Kommentar, Timestamp, AusleihID, ErstellerID, EmpfaengerID) VALUES (12, '5', 'Sehr nett', '2025-12-28 16:08:45', 14, 8, 4);

Screenshot nach Ausführung des Testfalls



Sicherstellung der Datenkonsistenz durch Trigger Prüfung von Sperren

Geschäftsregel:

- Um sicherzustellen, dass kein gesperrter Anwender eine Ausleihe tätigt, wird folgende Prozesslogik definiert: Prüfe, ob in der View Anwenderstatus die Sperre FALSE ist, bevor eine Anfrage zur Ausleihe erlaubt wird

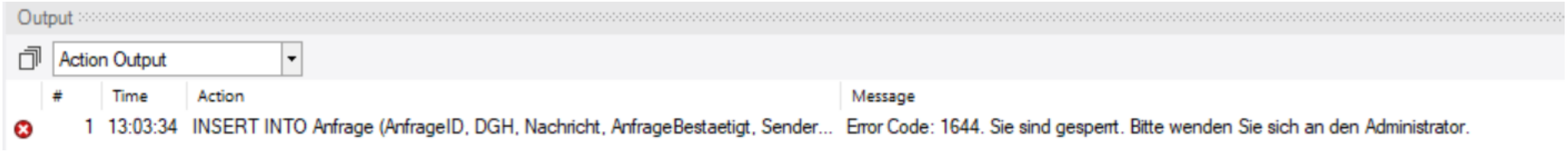
SQL-Statements:

- CREATE TRIGGER CheckSperre…

Testfall:

- Clara Schmidt möchte das Buch „Der alte Mann und das Meer“ ausleihen und daher eine Anfrage zur Ausleihe stellen. Dies ist ihr nicht erlaubt, da zu ihrer letzten Ausleihe noch keine Bestätigung des Verleihers zur erfolgten Rückgabe vorliegt:
- INSERT INTO Anfrage (AnfrageID, DGH, Nachricht, AnfrageBestaetigt, Sender, BuchID) VALUES (21, FALSE, NULL , FALSE, 3, 2);

Screenshot nach Ausführung des Testfalls



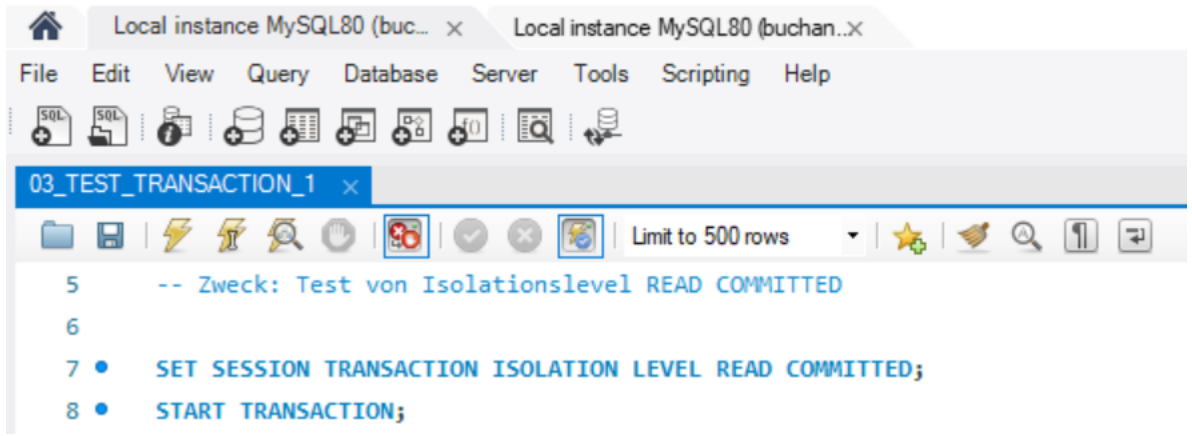
Sicherstellung der Datenkonsistenz durch Transaktionsmodelle

- Wie in den übergreifenden Hinweisen zur Datenbank begründet (vgl. Folie 3) werden zur Sicherstellung der Datenkonsistenz für die unten genannten Anwender-Aktivitäten Transaktionen definiert
- Im Sinne der Anwenderakzeptanz wird die flüssige App-Anwendung gegenüber jederzeitiger Datenkonsistenz bevorzugt. Daher wird READ COMMITTED als Isolationslevel für alle Transaktionen definiert. Somit müssen nur committed-Daten gelesen und weniger Sperren gesetzt werden

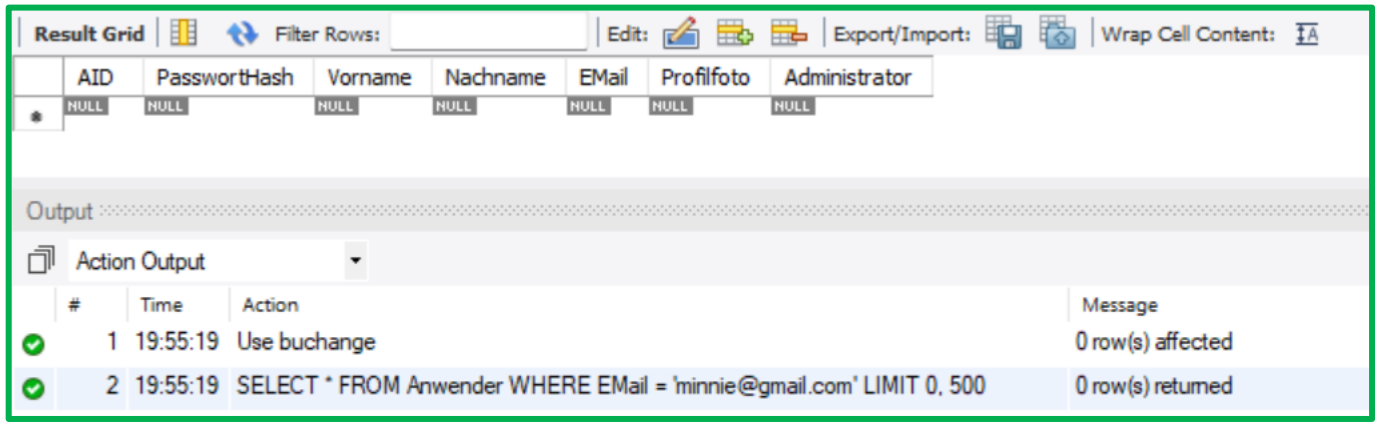
1. Registrierung eines neuen Anwenders
    - » führt zu Eintrag in Tabellen Anwender, Adresse und Abholinfo
    - » Kaum Konfliktrisiko durch READ COMMITTED, da nur neue Einträge erfolgen
2. Erfassung eines neuen Buches
    - » führt zu Eintrag in Tabelle Buch und ggf. Autor, wenn der Autor in der Tabelle Autor noch nicht vorhanden ist
    - » Falls 2 Nutzer parallel ein Buch mit einem nicht vorhandenen Autor einfügen, verhindert der Unique-Constraint der Tabellendefinition Dubletten.
3. Bestätigung einer Anfrage
    - » führt zu Aktualisierung der Tabelle Anfrage und Eintrag in Tabelle Ausleihe
    - » Kein Risiko durch Isolationslevel, da eine Anfrage immer nur von einem Anwender bestätigt werden kann
4. Bestätigung einer Buchrückgabe
    - » führt zu Aktualisierung der Tabellen Rückgabe und Ausleihe
    - » Kein Risiko durch Isolationslevel, da eine Anfrage immer nur von einem Anwender bestätigt werden kann

Beurteilung des Einsatzes von Transaktionen mit Isolationslevel READ COMMITED

Es wird getestet, ob bei einer Registrierung eines Anwenders alle zugehörigen Tabellen konsistent aktualisiert werden und dass keine unvollständigen Zwischenstände angezeigt werden:



- Es werden bei Sessions in der Workbench mit Verbindung zur Datenbank geöffnet
- In TEST_TRANSACTION_1 läuft eine Anwenderregistrierung, die noch nicht abgeschlossen ist (COMMIT fehlt)
- In TEST_TRANSACTION_2 soll der Anwender bereits mit einem SELECT-Statement angezeigt werden



- Das SELECT-Statement blockiert nicht, sondern kann ausgeführt werden
- Es werden aber auch keine inkonsistenten Daten angezeigt, da einfach der letzte gesicherte Datenstand angezeigt wird

Beurteilung der Indexierung durch Performance-Messung

Es wird eine Performance-Messung von ausgewählten Select-Statements durchgeführt, um zu prüfen, ob die eingesetzte Indexierung tatsächlich zu Verbesserungen führt. Das Vorgehen wird auf den Folgefolien dargelegt.

Die Ergebnisse zeigen, dass sich die Abfragezeiten bspw. für die SELECT STATISTICS nur bei Tabellen mit großen Datenmengen verbessern:

Statement	Max Time in Mikrosekunden ohne Index/Transaktionen	Max Time in Mikrosekunden mit Index/Transaktionen
SELECT b.Genre, COUNT(*) AS Anzahl …	665,10	463,20
SELECT aut.AutorID AS ID, …	461,50	466,90
SELECT a.Vorname FROM Anwender a	720,50	351,10

Die Messergebnisse lassen sich durch folgende Überlegungen begründen:

» Eine Verbesserung kann insbesondere für die Tabelle Anwender beobachtet werden, da hier viele Daten eingefügt wurden

» In den Tabellen mit wenigen Daten bringt die Indexierung keinen Vorteil, da die Daten ohnehin schnell gefunden werden

» Indexierung kann teilweise zu einer Verschlechterung führen, da zusätzliche Verwaltungs- und Speicherzugriffe nötig sind

FAZIT: Die Indexierung wird beibehalten, da im Verlauf der Jahre mit wachsenden Datenmengen zu rechnen ist und die aktuellen Ausführungszeiten bereits sehr schnell sind.

Vorgehen zur Performance-Messung ohne Indexierung

1. Ausführen CREATE DATABASE
2. Ausführen CREATE TABLE
3. **Weglassen CREATE INDEX**
4. Ausführen CREATE TRIGGER
5. Ausführen START TRANSACTION
6. Ausführen CREATE VIEW
7. Einfügen der Testdaten durch Ausführung INSERT INTO
8. Einfügen der Massendaten durch Ausführung TEST_MASSDATA
9. Ausführen SELECT STATISTICS
10. Aufruf Performance-Reports

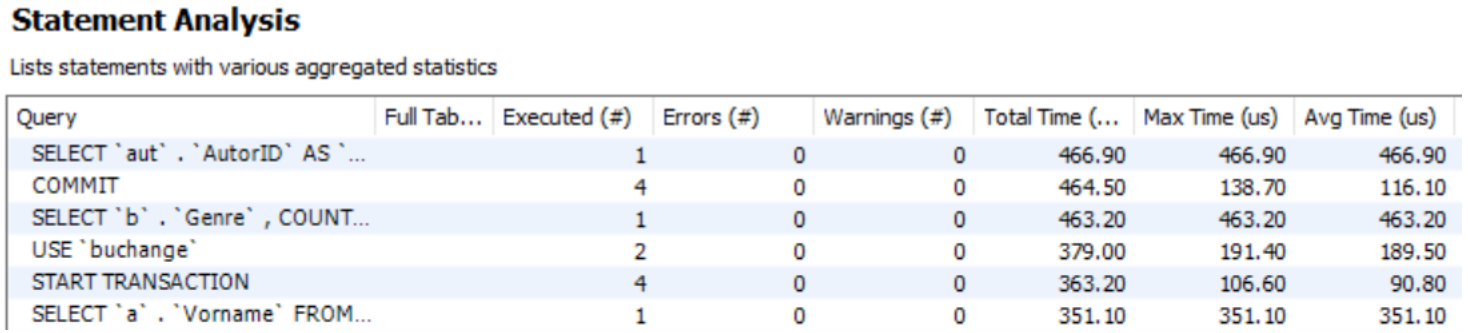
Statement Analysis

Lists statements with various aggregated statistics

Query	Full Tab...	Executed (#)	Errors (#)	Warnings (#)	Total Time (...	Max Time (us)	Avg Time (us)
TRUNCATE TABLE `performanc...		1	0	0	779.70	779.70	779.70
SELECT `a` . `Vorname` FROM...	*	1	0	0	720.50	720.50	720.50
TRUNCATE TABLE `performanc...		1	0	0	686.30	686.30	686.30
SELECT `b` . `Genre` , COUNT...		1	0	0	665.10	665.10	665.10
SELECT `besitzer` . `AID` AS `...		1	0	0	543.50	543.50	543.50
SELECT `aut` . `AutorID` AS `...		1	0	0	461.50	461.50	461.50

Vorgehen zur Performance-Messung mit Indexierung

1. Löschen der Datenbank durch DROP DATABASE
2. Löschen der Performance-Reports durch Ausführung PERFORMANCE_TABLENAMES und PERFORMANCE_DELETE
3. Ausführen CREATE DATABASE
4. Ausführen CREATE TABLE
5. **Ausführen CREATE INDEX**
6. Ausführen CREATE TRIGGER
7. Ausführen START TRANSACTION
8. Ausführen CREATE VIEW
9. Einfügen der Testdaten durch Ausführung INSERT INTO
10. Einfügen der Massendaten durch Ausführung TEST_MASSDATA
11. Ausführen SELECT STATISTICS
12. Aufruf Performance-Reports



Statement Analysis

Lists statements with various aggregated statistics

Query	Full Tab...	Executed (#)	Errors (#)	Warnings (#)	Total Time (...	Max Time (us)	Avg Time (us)
SELECT `aut` . `AutorID` AS `...		1	0	0	466.90	466.90	466.90
COMMIT		4	0	0	464.50	138.70	116.10
SELECT `b` . `Genre` , COUNT...		1	0	0	463.20	463.20	463.20
USE `buchange`		2	0	0	379.00	191.40	189.50
START TRANSACTION		4	0	0	363.20	106.60	90.80
SELECT `a` . `Vorname` FROM...		1	0	0	351.10	351.10	351.10

Zusammenfassung

Ziel:

- Aufbau einer relationalen Datenbank zur Verwaltung von Buchausleihen
- Abbildung sämtlicher Stammdaten und Bewegungsdaten von der Anfrage bis zur Rückgabe und Bewertung
- Konsistenz der Daten unter Berücksichtigung der definierten Geschäftslogiken

Methode:

- Es wurde mittels MySQL eine Datenbank „buchange“ implementiert
- Für die zentralen Objekte und Prozesse wurde jeweils eine eigene Entität definiert
- Zur Nachvollziehbarkeit der Prozesse werden in den Tabellen Status und Zeitstempel eingesetzt
- Die Entitäten stehen mittels Fremdschlüsseln in Beziehungen zueinander, die Datenqualität wird zusätzlich durch den Einsatz von Bedingungen durch Constraints und Trigger gesichert
- Eine Indexierung wird eingesetzt, mit Hilfe von Test-Massendaten kann gezeigt werden, dass dies zu Performance-Verbesserungen führt
- Für die Sicherstellung der Datenkonsistenz und zur Vermeidung von Blockaden während der App-Ausführung werden Transaktionen mit Isolationslevel READ COMMITTED definiert
- Nutzung von Views, mit denen aufwändige SELECT-Statements zentral gekapselt und mehrfach nutzbar gemacht werden
- Der modulare Aufbau der Entitäten und Views soll eine flexible Erweiterung und einfache Wartbarkeit der Datenbank ermöglichen

Installationsanleitung zu SQL-Dateien

1. Installiere MySQL inkl. der MySQL Workbench (Link: MySQL :: Download MySQL Installer, hier verwendete Version ist 8.0.44)
2. Ausführen CREATE DATABASE
3. Doppelklicke unter SCHEMAS die Datenbank buchange und führe CREATE TABLE aus
4. Ausführen CREATE INDEX
5. Ausführen CREATE TRIGGER
6. Ausführen START TRANSACTION
7. Ausführen CREATE VIEW
8. OPTIONAL: Einfügen der Testdaten durch Ausführung INSERT INTO
9. OPTIONAL: Ausführung eines Testfalls durch iteratives Ausführen der Statements in TESTS
10. OPTIONAL: Einspielen von Massendaten für Tabelle Anwender durch Ausführen TEST_MASSDATA
11. OPTIONAL: Ausführen STATISTICS für Anzeige der Nutzungsstatistiken
12. OPTIONAL: Ausführen der Transaktionstests mit Hilfe von TEST_TRANSACTION_1 und TEST_TRANSACTION_2 in 2 Workbench-Sessions
13. OPTIONAL: Durchführung von Performance-Tests in 2 Schritten
    1. Ermittlung Namen der in SQL integrierten Performance-Reports
    2. Löschen der Tabelleninhalte (falls Datenbank verändert wurde)
14. OPTIONAL: Verwendung UPDATE TABLE für weitere Anwendungsfälle
15. OPTIONAL: Ausführen von METADATA

Zusammenfassung

Ergebnis: