Tabelle: address

Erstellungs-Statement:

```
CREATE TABLE address (
    address_ID INT PRIMARY KEY AUTO_INCREMENT,
    street VARCHAR(100) NOT NULL,
    homenumber VARCHAR(10) NOT NULL,
    postcode INT NOT NULL,
    city VARCHAR(50) NOT NULL
);
```

Dateneingabe:

```
INSERT INTO address (street, homenumber, postcode, city)
VALUES ('Reitgasse', '8', 35037, 'Marburg');
```

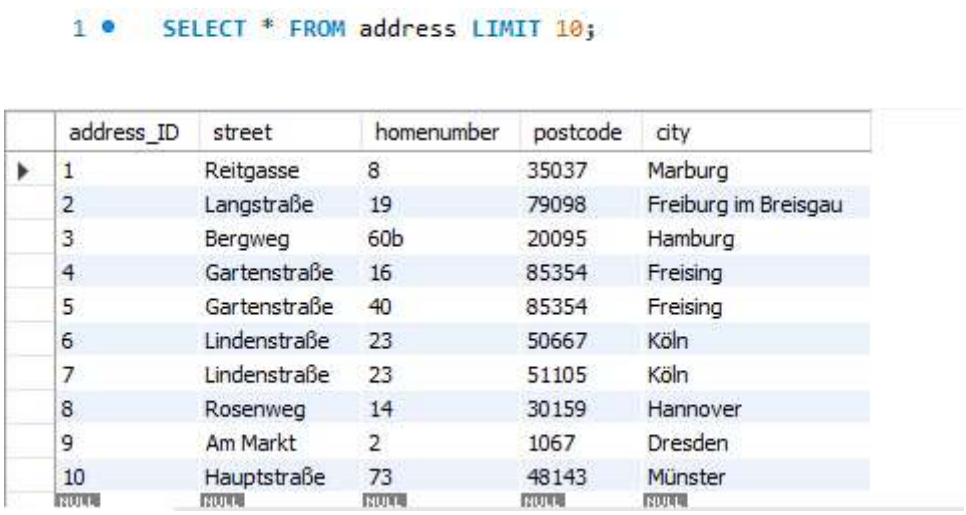
Testabfrage:

```
SELECT * FROM address LIMIT 10;
```

Hinweise/Design-Entscheidungen:

- VARCHAR für Hausnummer, da auch Buchstaben enthalten sein können (z. B. 60b)
- INT für Postleitzahl, da zunächst nur in Deutschland verfügbar
- „number“ in „homenumber“ umbenannt, da MySQL-Statement

Screenshots:



```
1 •   SELECT * FROM address LIMIT 10;
```

address_ID	street	homenumber	postcode	city
1	Reitgasse	8	35037	Marburg
2	Langstraße	19	79098	Freiburg im Breisgau
3	Bergweg	60b	20095	Hamburg
4	Gartenstraße	16	85354	Freising
5	Gartenstraße	40	85354	Freising
6	Lindenstraße	23	50667	Köln
7	Lindenstraße	23	51105	Köln
8	Rosenweg	14	30159	Hannover
9	Am Markt	2	1067	Dresden
10	Hauptstraße	73	48143	Münster
NULL	NULL	NULL	NULL	NULL

Tabelle: author

Erstellungs-Statement:

```
CREATE TABLE author (
   author_ID INT PRIMARY KEY AUTO_INCREMENT,
   lastname VARCHAR(255) NOT NULL,
   firstname VARCHAR(255) NOT NULL
);
```

Dateneingabe:

```
INSERT INTO author (lastname, firstname)
VALUES ('Tolkien', 'John Ronald Reuel');
```

Hinweise / Designentscheidungen:

- Volle Vornamen für historische Genauigkeit
- AUTO_INCREMENT für eindeutige Identifikation

Testabfragen:

- SELECT * FROM author WHERE author_ID = 3;

author_ID	lastname	firstname
3	Tolkien	John Ronald Reuel
NULL	NULL	NULL

- SELECT author_ID, firstname, lastname
FROM author
LIMIT 10;

author_ID	firstname	lastname
1	Sebastian	Fitzek
2	Sophie	Kinsella
3	John Ronald Reuel	Tolkien
4	Matt	Ruff
5	Ursula	Poznanski
6	Arno	Strobel
7	Hajime	Isayama
8	Peter	Ustinov
9	Mhairi	McFarlane
10	Michael	Ende

Tabelle: author_book

Erstellungs-Statement:

```
CREATE TABLE author_book (
    author_ID INT NOT NULL,
    book_ID INT NOT NULL,
    PRIMARY KEY (author_ID, book_ID),
    FOREIGN KEY (author_ID) REFERENCES author(author_ID) ON DELETE CASCADE,
    FOREIGN KEY (book_ID) REFERENCES book(book_ID) ON DELETE CASCADE
);
```

Dateneingabe:

```
INSERT INTO author_book (author_ID, book_ID) VALUES (3, 3);
```

Hinweise/Designentscheidungen:

- Tabelle erlaubt Mehrfachautoren durch mehrere Einträge pro Buch
- ON DELETE CASCADE für sauberes Löschen von Autoren oder Büchern
- Abweichend zu ERM Primärschlüssel nicht auto increment, sondern zusammengesetzt aus den beiden referenzierten Fremdschlüsseln (klarere und sinnvollere Zuordnung)

Testabfrage:

```
SELECT ab.author_ID, a.firstname, a.lastname, ab.book_ID, b.title
FROM author_book ab
JOIN author a ON ab.author_ID = a.author_ID
JOIN book b ON ab.book_ID = b.book_ID
LIMIT 10;
```

author_ID	firstname	lastname	book_ID	title
1	Sebastian	Fitzek	1	Passagier 23
2	Sophie	Kinsella	2	Sag's nicht weiter, Liebling!
3	John Ronald Reuel	Tolkien	3	Der Herr der Ringe
3	John Ronald Reuel	Tolkien	18	Der Herr der Ringe
4	Matt	Ruff	4	Bad Monkeys
5	Ursula	Poznanski	5	Erebos
5	Ursula	Poznanski	6	Fremd
6	Arno	Strobel	6	Fremd
7	Hajime	Isayama	7	Attack on Titan 1
7	Hajime	Isayama	8	Attack on Titan 2

Tabelle: available

Erstellungs-Statement:

```
CREATE TABLE available (
    available_ID INT PRIMARY KEY AUTO_INCREMENT,
    state ENUM ('verfügbar', 'reserviert', 'verliehen') NOT NULL,
    available_from DATE NOT NULL,
    available_until DATE NOT NULL
);
```

Dateineingabe:

```
INSERT INTO available (state, available_from, available_until)
VALUES ('verfügbar', '2025-09-25', '2025-10-25');
```

Hinweise / Designentscheidungen:

- Datumsangaben im DATE-Format
- state beschreibt aktuellen Zustand

Testanfragen:

- SELECT * FROM available WHERE state='verfügbar';

available_ID	state	available_from	available_until
1	verfügbar	2025-09-01	2025-12-31
2	verfügbar	2025-09-25	2025-12-01
5	verfügbar	2025-08-10	2025-08-31
6	verfügbar	2025-09-10	2025-09-28
8	verfügbar	2025-09-10	2025-09-28
10	verfügbar	2025-10-15	2025-10-31
12	verfügbar	2025-09-01	2025-09-08

- SELECT * FROM available LIMIT 10;

available_ID	state	available_from	available_until
1	verfügbar	2025-09-01	2025-12-31
2	verfügbar	2025-09-25	2025-12-01
3	reserviert	2025-09-20	2025-09-23
4	verliehen	2025-08-01	2025-10-15
5	verfügbar	2025-08-10	2025-08-31
6	verfügbar	2025-09-10	2025-09-28
7	reserviert	2025-09-24	2025-09-27
8	verfügbar	2025-09-10	2025-09-28
9	verliehen	2025-09-01	2025-10-31
10	verfügbar	2025-10-15	2025-10-31

Tabelle: book

Erstellungs-Statement:

```
CREATE TABLE book (
    book_ID INT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    publisher VARCHAR(100) NOT NULL,
    ISBN VARCHAR(20) NOT NULL,
    release_date DATE NOT NULL,
    language_b VARCHAR(50) NOT NULL,
    pages INT
);
```

Dateneingabe:

```
INSERT INTO book (title, publisher, ISBN, release_date, language_b, pages)
VALUES ('Passagier 23', 'Droemer HC', '9783426199190', '2014-10-30', 'deutsch', 432);
```

Hinweise / Designentscheidungen:

- ISBN als VARCHAR gespeichert, da Ziffern teilweise führende Nullen enthalten
- release_date als DATE für einfache Abfrage von Erscheinungsjahr oder Zeitraum
- pages kann NULL sein, daher optional

Testabfragen:

- SELECT title, publisher, release_date FROM book WHERE book_ID = 1;

title	publisher	release_date
Passagier 23	Droemer HC	2014-10-30

- SELECT book_ID, title, publisher, ISBN, release_date, language_b, pages
FROM book
LIMIT 10;

book_ID	title	publisher	ISBN	release_date	language_b	pages
1	Passagier 23	Droemer HC	9783426199190	2014-10-30	deutsch	432
2	Sag's nicht weiter, Liebling!	Goldmann Verlag	9783442456321	2004-02-01	deutsch	384
3	Der Herr der Ringe	Klett-Cotta	9783608952124	1987-02-01	deutsch	NULL
4	Bad Monkeys	Deutscher Taschenbuch Verlag	9783423211796	2009-11-01	deutsch	272
5	Erebos	Loewe Verlag	9783785569573	2010-01-07	deutsch	485
6	Fremd	Rowohlt Taschenbuch Verlag	9783499270918	2016-08-26	deutsch	416
7	Attack on Titan 1	Carlsen Manga	9783551742339	2014-03-18	deutsch	192
8	Attack on Titan 2	Carlsen Manga	9783551742346	2014-05-01	deutsch	192
9	Der Alte Mann und Mr. Smith	Econ	9783430192781	1991-01-01	deutsch	320
10	Wir in drei Worten	Knaur TB	9783426514535	2013-10-01	deutsch	496
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Tabelle: book_genre

Erstellungs-Statement:

```
CREATE TABLE book_genre (
    book_ID INT NOT NULL,
    genre_ID INT NOT NULL,
    PRIMARY KEY (book_ID, genre_ID),
    FOREIGN KEY (book_ID) REFERENCES book(book_ID) ON DELETE CASCADE,
    FOREIGN KEY (genre_ID) REFERENCES genre(genre_ID) ON DELETE CASCADE
);
```

Dateneingabe:

```
INSERT INTO book_genre (book_ID, genre_ID) VALUES (1, 1), (1, 3);
```

Hinweise / Designentscheidungen:

- Mehrfachgenres pro Buch möglich
- ON DELETE CASCADE für sauberes Löschen von Büchern oder Genres
- Abweichend von ERM Primärschlüssel nicht auto increment, sondern zusammengesetzt aus den beiden referenzierten Fremdschlüsseln (klarere und sinnvollere Zuordnung)

Testabfragen:

-
```
SELECT bg.book_ID, b.title, bg.genre_ID, g.category
FROM book_genre bg
JOIN book b ON bg.book_ID = b.book_ID
JOIN genre g ON bg.genre_ID = g.genre_ID
LIMIT 10;
```

book_ID	title	genre_ID	category
1	Passagier 23	1	Psychothriller
1	Passagier 23	3	Thriller allgemein
2	Sag's nicht weiter, Liebling!	15	Romantic Comedy
3	Der Herr der Ringe	4	Fantasy
4	Bad Monkeys	1	Psychothriller
4	Bad Monkeys	3	Thriller allgemein
4	Bad Monkeys	5	Science Fiction
5	Erebos	3	Thriller allgemein
5	Erebos	17	Young Adult
6	Fremd	1	Psychothriller

-
```
SELECT b.title, g.category
FROM book_genre bg
JOIN book b ON bg.book_ID = b.book_ID
JOIN genre g ON bg.genre_ID = g.genre_ID
WHERE b.book_ID = 1;
```

title	category
Passagier 23	Psychothriller
Passagier 23	Thriller allgemein

Tabelle: conflict

Erstellungs-Statement:

```
CREATE TABLE conflict (
    conflict_ID INT PRIMARY KEY AUTO_INCREMENT,
    loan_ID INT,
    opponent_ID INT,
    reported_by_ID INT,
    reported_time TIMESTAMP NOT NULL,
    state ENUM('offen', 'geklärt') NOT NULL,
    FOREIGN KEY (loan_ID) REFERENCES loan(loan_ID) ON DELETE SET NULL,
    FOREIGN KEY (opponent_ID) REFERENCES user(user_ID) ON DELETE SET NULL,
    FOREIGN KEY (reported_by_ID) REFERENCES user(user_ID) ON DELETE SET NULL
);
```

Dateneingabe:

```
INSERT INTO conflict (loan_ID, opponent_ID, reported_by_ID, reported_time, state) VALUES
(null, 7, 2, '2025-09-25 14:00:00', 'offen');
```

Hinweise / Designentscheidungen:

- Konflikt kann ohne Leihe oder Gegner existieren
- ENUM für Status, TIMESTAMP für exakte Zeitangabe der Meldung

Testabfrage:

```
SELECT c.conflict_ID, l.loan_ID, r.nickname AS reporter, o.nickname AS opponent, c.state
FROM conflict c
LEFT JOIN loan l ON c.loan_ID = l.loan_ID
LEFT JOIN user r ON c.reported_by_ID = r.user_ID
LEFT JOIN user o ON c.opponent_ID = o.user_ID
LIMIT 10;
```

conflict_ID	loan_ID	reporter	opponent	state
1	1	BuchFan88	Leseratte99	offen
2	NULL	Leseratte99	PageTurner	geklärt
3	3	PageTurner	NULL	offen
4	NULL	BookWorm	NovelAddict	geklärt
5	5	NovelAddict	NULL	offen
6	NULL	ReadLover	StorySeeker	geklärt
7	7	StorySeeker	NULL	offen
8	NULL	Bibliophile	BookNerd	geklärt
9	9	BookNerd	NULL	offen
10	NULL	LitLover	NovelSeeker	geklärt

Tabelle: genre

Erstellungs-Statement:

```
CREATE TABLE genre (
    genre_ID INT PRIMARY KEY AUTO_INCREMENT,
    category VARCHAR(50) NOT NULL
);
```

Dateneingabe:

```
INSERT INTO genre (category) VALUES ('Krimi allgemein');
```

Hinweise / Designentscheidungen:

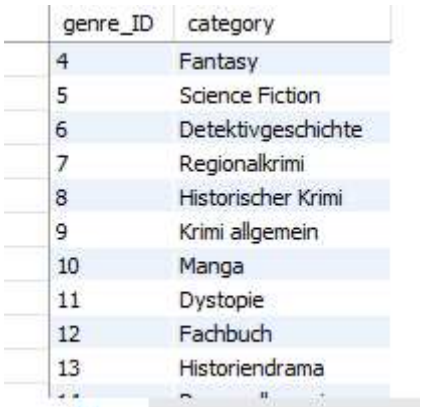
- Varchar für flexible Genre-Bezeichnungen
- Nutzerfreundliche Begriffe verwendet, daher z.T. umgangssprachliche Bezeichnung und geläufigste Genres

Testabfragen:

- SELECT * FROM genre WHERE genre_ID = 3;

genre_ID	category
3	Thriller allgemein
NULL	NULL

- SELECT * FROM genre;



genre_ID	category
4	Fantasy
5	Science Fiction
6	Detektivgeschichte
7	Regionalkrimi
8	Historischer Krimi
9	Krimi allgemein
10	Manga
11	Dystopie
12	Fachbuch
13	Historiendrama

Tabelle: loan

Erstellungs-Statement:

```
CREATE TABLE loan (
    loan_ID INT PRIMARY KEY AUTO_INCREMENT,
    request_ID INT NOT NULL,
    loan_start DATE NOT NULL,
    loan_end DATE NOT NULL,
    state ENUM('laufend','beendet') NOT NULL,
    FOREIGN KEY (request_ID) REFERENCES request(request_ID) ON DELETE CASCADE
);
```

Dateneingabe:

INSERT INTO loan (request_ID, loan_start, loan_end, state) VALUES (1, '2025-09-26', '2025-10-03', 'laufend');

Hinweise / Designentscheidungen:

- DATE für Start- und Enddatum, genauer Zeitstempel ist nicht erforderlich
- ON DELETE CASCADE für sauberes Löschen von Requests

Testabfragen:

- SELECT l.loan_ID, r.request_ID, u.nickname AS borrower, l.loan_start, l.loan_end, l.state
FROM loan l
JOIN request r ON l.request_ID = r.request_ID
JOIN user u ON r.requested_by_ID = u.user_ID
LIMIT 10;

loan_ID	request_ID	borrower	loan_start	loan_end	state
1	1	Leseratte99	2025-09-01	2025-09-15	beendet
2	2	PageTurner	2025-09-03	2025-09-17	laufend
3	3	BuchFan88	2025-09-05	2025-09-19	beendet
4	4	NovelAddict	2025-09-02	2025-09-16	laufend
5	5	ReadLover	2025-09-04	2025-09-18	beendet
6	6	StorySeeker	2025-09-06	2025-09-20	laufend
7	7	Bibliophile	2025-09-07	2025-09-21	beendet
8	8	BookNerd	2025-09-08	2025-09-22	laufend
9	9	LitLover	2025-09-09	2025-09-23	beendet
10	10	NovelSeeker	2025-09-10	2025-09-24	laufend

- SELECT * FROM loan WHERE state = 'laufend';

loan_ID	request_ID	loan_start	loan_end	state
2	2	2025-09-03	2025-09-17	laufend
4	4	2025-09-02	2025-09-16	laufend
6	6	2025-09-06	2025-09-20	laufend
8	8	2025-09-08	2025-09-22	laufend
10	10	2025-09-10	2025-09-24	laufend
12	12	2025-09-12	2025-09-26	laufend

Tabelle: location

Erstellungs-Statement:

```
CREATE TABLE location (
    location_ID INT PRIMARY KEY AUTO_INCREMENT,
    street VARCHAR(100),
    homenumber VARCHAR(10),
    place VARCHAR(50),
    postcode INT NOT NULL,
    city VARCHAR(50) NOT NULL,
    latitude DECIMAL(10,6) NOT NULL,
    longitude DECIMAL(10,6) NOT NULL
);
```

Dateneingabe:

INSERT INTO location (street, homenumber, postcode, city, latitude, longitude) VALUES (null, null, ’Café Extrablatt‘, 48143, 'Münster', 51.9624, 7.6257);

Hinweise / Designentscheidungen:

- Latitude und Longitude für genauere Speicherung
- Hausnummernangabe als VARCHAR, um z. B. 60b abzudecken
- street, homenumber und place nullable, damit Ort flexibel gewählt werden kann
- number in homenumber umbenannt, weil MySQL-Statement

Testabfragen:

- SELECT * FROM location WHERE city='Münster';

location_ID	street	homenumber	place	postcode	city	latitude	longitude
11	NULL	NULL	Café Extrablatt Münster	48143	Münster	51.962400	7.625700

- SELECT * FROM location LIMIT 10;

location_ID	street	homenumber	place	postcode	city	latitude	longitude
1	Universitätsstraße	10	NULL	35037	Marburg	50.810000	8.770000
2	NULL	NULL	Stadtbibliothek Freiburg	79098	Freiburg	47.995900	7.852200
3	NULL	NULL	Englischer Garten, Nähe See	80538	München	48.150000	11.590000
4	Bahnhofstraße	3	NULL	50667	Köln	50.941300	6.958300
5	Am Markt	1	Treffpunkt Brunnen	1067	Dresden	51.050400	13.737300
6	NULL	NULL	Hauptbahnhof Hamburg, Ditsch-Stand	20095	Hamburg	53.552600	10.006700
7	Lindenstraße	23	NULL	51105	Köln	50.922000	7.000500
8	NULL	NULL	Uni-Bibliothek Tübingen	72070	Tübingen	48.521600	9.057600
9	Gartenstraße	16	NULL	85354	Freising	48.399600	11.748600
10	Holstenstraße	NULL	Bushaltestelle	24103	Kiel	54.323300	10.122800

Tabelle: messages

Erstellungs-Statement:

```
CREATE TABLE messages (
    message_ID INT PRIMARY KEY AUTO_INCREMENT,
    sender_ID INT,
    receiver_ID INT,
    time_sent TIMESTAMP NOT NULL,
    content TEXT NOT NULL,
    FOREIGN KEY (sender_ID) REFERENCES user(user_ID) ON DELETE SET NULL,
    FOREIGN KEY (receiver_ID) REFERENCES user(user_ID) ON DELETE SET NULL
);
```

Dateneingabe:

```
INSERT INTO messages (sender_ID, receiver_ID, time_sent, content)
VALUES (1, 2, '2025-09-25 12:00:00', 'Hallo, ich würde das Buch gerne ausleihen :)');
```

Hinweise / Designentscheidungen:

- TIMESTAMP für Zeitstempel
- ON DELETE SET NULL für Nutzerlöschung
- Sender und Empfänger nullable, um Nachrichten bei Löschung eines Nutzers nicht zu verlieren

Testabfrage:

```
SELECT m.message_ID, u1.nickname AS sender, u2.nickname AS receiver, m.content
FROM messages m
JOIN user u1 ON m.sender_ID = u1.user_ID
JOIN user u2 ON m.receiver_ID = u2.user_ID;
```

message_ID	sender	receiver	content
1	BuchFan88	Leseratte99	Hallo, hast du das Buch schon zurückgegeben?
2	Leseratte99	PageTurner	Ja, morgen bringe ich es vorbei.
3	PageTurner	BookWorm	Super, danke!
4	BookWorm	NovelAddict	Kannst du mir dein Exemplar leihen?
5	NovelAddict	ReadLover	Klar, ich bringe es mit.
6	ReadLover	StorySeeker	Danke!
7	StorySeeker	Bibliophile	Wann ist Abholung?
8	Bibliophile	BookNerd	Morgen Nachmittag.
9	BookNerd	LitLover	Perfekt!
10	LitLover	NovelSeeker	Bis morgen.

Tabelle: photo

Erstellungs-Statement:

```
CREATE TABLE photo (
    photo_ID INT PRIMARY KEY AUTO_INCREMENT,
    user_book_ID INT,
    loan_ID INT,
    conflict_ID INT,
    created_by_ID INT NOT NULL,
    creation_time TIMESTAMP NOT NULL,
    FOREIGN KEY (user_book_ID) REFERENCES user_book(user_book_ID) ON DELETE SET
NULL,
    FOREIGN KEY (loan_ID) REFERENCES loan(loan_ID) ON DELETE SET NULL,
    FOREIGN KEY (conflict_ID) REFERENCES conflict(conflict_ID) ON DELETE SET NULL,
    FOREIGN KEY (created_by_ID) REFERENCES user(user_ID) ON DELETE CASCADE
);
```

Dateneingabe:

```
INSERT INTO photo (user_book_ID, created_by_ID, creation_time) VALUES (1, 1, '2025-09-25
13:00:00');
```

Hinweise / Designentscheidungen:

- Optionale FK erlauben Fotos ohne Konflikt/Leihe
- TIMESTAMP für Erstellzeit, damit im Konfliktfall belegbar

Testabfrage:

```
SELECT p.photo_ID, ub.user_book_ID, l.loan_ID, p.creation_time, u.nickname AS created_by
FROM photo p
LEFT JOIN user_book ub ON p.user_book_ID = ub.user_book_ID
LEFT JOIN loan l ON p.loan_ID = l.loan_ID
JOIN user u ON p.created_by_ID = u.user_ID
LIMIT 10;
```

photo_ID	user_book_ID	loan_ID	creation_time	created_by
1	1	1	2025-09-01 10:00:00	BuchFan88
2	2	2	2025-09-01 11:00:00	Leseratte99
3	3	3	2025-09-01 12:00:00	PageTurner
4	4	4	2025-09-01 13:00:00	BookWorm
5	5	5	2025-09-01 14:00:00	NovelAddict
6	6	6	2025-09-01 15:00:00	ReadLover
7	7	7	2025-09-01 16:00:00	StorySeeker
8	8	8	2025-09-01 17:00:00	Bibliophile
9	9	9	2025-09-01 18:00:00	BookNerd
10	10	10	2025-09-01 19:00:00	LitLover

Tabelle: request

Erstellungs-Statement:

```
CREATE TABLE request (
   request_ID INT PRIMARY KEY AUTO_INCREMENT,
   user_book_loc_ID INT NOT NULL,
   requested_by_ID INT NOT NULL,
   state ENUM('offen', 'angenommen', 'abgelehnt') NOT NULL,
   FOREIGN KEY (user_book_loc_ID) REFERENCES user_book_location(user_book_loc_ID) ON
DELETE CASCADE,
   FOREIGN KEY (requested_by_ID) REFERENCES user(user_ID)
);
```

Dateneingabe:

```
INSERT INTO request (user_book_loc_ID, requested_by_ID, state) VALUES (1, 2, 'open');
```

Hinweise / Designentscheidungen:

ENUM für Status, um nur gültige Werte zuzulassen

Testabfrage:

```
SELECT r.request_ID, u.nickname AS requested_by, ub.user_book_ID, r.state
FROM request r
JOIN user u ON r.requested_by_ID = u.user_ID
JOIN user_book_location ubl ON r.user_book_loc_ID = ubl.user_book_loc_ID
JOIN user_book ub ON ubl.user_book_ID = ub.user_book_ID
LIMIT 10;
```

request_ID	requested_by	user_book_ID	state
1	Leseratte99	1	offen
2	PageTurner	2	angenommen
3	BuchFan88	3	offen
4	NovelAddict	4	offen
5	ReadLover	5	angenommen
6	StorySeeker	6	offen
7	Bibliophile	7	offen
8	BookNerd	8	abgelehnt
9	LitLover	9	offen
10	NovelSeeker	10	angenommen

Tabelle: reviews

Erstellungs-Statement:

```
CREATE TABLE reviews (
    review_ID INT PRIMARY KEY AUTO_INCREMENT,
    reviewer_ID INT NOT NULL,
    reviewed_user_ID INT NOT NULL,
    loan_ID INT,
    friendly_contact TINYINT(1) NOT NULL,
    reliability TINYINT(1),
    condition_as_described TINYINT(1),
    review_total DECIMAL(3,1),
    FOREIGN KEY (reviewer_ID) REFERENCES user(user_ID) ON DELETE CASCADE,
    FOREIGN KEY (reviewed_user_ID) REFERENCES user(user_ID) ON DELETE CASCADE,
    FOREIGN KEY (loan_ID) REFERENCES loan(loan_ID) ON DELETE SET NULL
);
```

Dateneingabe:

```
INSERT INTO review (reviewer_ID, reviewed_user_ID, loan_ID, friendly_contact, reliability,
condition_as_described, review_total)
VALUES (2,1,1,5,5,5,5);
```

Hinweise / Designentscheidungen:

- Loan optional, daher SET NULL bei Löschung
- Werte „reliability“ und „condition_as_described“ nullable, damit review auch ohne Ausleihe abgegeben werden kann (daher „friendly_contact“ NN, weil auch bei Nachrichten bewertbar)
- TINYINT statt SMALLINT aus Gründen der Speicherkapazität (mehr ist nicht erforderlich
- Spalte „notes“ wurde gestrichen, da ohnehin optional und anhand der simplen Bewertungsmöglichkeit überflüssig

Testabfrage:

```
SELECT rev.review_ID, u.nickname AS reviewer, rev.review_total, rev.friendly_contact,
rev.reliability
FROM reviews rev
JOIN user u ON rev.reviewer_ID = u.user_ID
LIMIT 10;
```

review_ID	reviewer	review_total	friendly_contact	reliability
1	BuchFan88	5.0	5	5
2	Leseratte99	4.0	4	5
3	PageTurner	5.0	5	4
4	BookWorm	3.0	3	4
5	NovelAddict	5.0	5	5
6	ReadLover	4.0	4	4
7	StorySeeker	5.0	5	5
8	Bibliophile	4.0	4	5
9	BookNerd	5.0	5	4
10	LitLover	3.0	3	4

Tabelle: user

Erstellungs-Statement:

```
CREATE TABLE user (

    user_ID INT PRIMARY KEY AUTO_INCREMENT,
    address_ID INT NOT NULL,
    nickname VARCHAR(50) NOT NULL,
    lastname VARCHAR(100) NOT NULL,
    firstname VARCHAR(100) NOT NULL,
    password_hash VARCHAR(64) NOT NULL,
    mail VARCHAR(255) NOT NULL,
    about_me TEXT,
    FOREIGN KEY (address_ID) REFERENCES address(address_ID)
);
```

Dateneingabe:

```
INSERT INTO user (address_ID, nickname, lastname, firstname, password_hash, mail, about_me)
VALUES (1, 'BuchFan88', 'Müller', 'Lena', 'hashed_password', 'lena.mueller@example.com',
'Lieblingsgenre: Thriller');
```

Hinweise / Designentscheidungen:

- Fremdschlüssel address_ID referenziert address – nur eine Adresse pro user zu Identifikation- und ggf. Nachverfolgungszwecken bei Konflikten. Abholung soll separat angegeben werden können (Datenschutz, Sicherheit, damit ggf. auch mehr Teilnahme)
- VARCHAR-Längen bewusst großzügig gewählt für mögliche längere Namen oder E-Mail-Adressen

Testabfrage:

```
SELECT u.user_ID, u.nickname, u.firstname, u.lastname, a.city, a.street
FROM user u
JOIN address a ON u.address_ID = a.address_ID
LIMIT 10;
```

user_ID	nickname	firstname	lastname	city	street
1	BuchFan88	Lena	Müller	Marburg	Reitgasse
2	Leseratte99	Max	Schmidt	Freiburg im Breisgau	Langstraße
3	PageTurner	Anna	Fischer	Hamburg	Bergweg
4	BookWorm	Jonas	Weber	Freising	Gartenstraße
5	NovelAddict	Sophie	Becker	Freising	Gartenstraße
6	ReadLover	Tim	Schneider	Köln	Lindenstraße
7	StorySeeker	Laura	Hoffmann	Köln	Lindenstraße
8	Bibliophile	Paul	Richter	Hannover	Rosenweg
9	BookNerd	Marie	Klein	Dresden	Am Markt
10	LitLover	Niklas	Wolf	Münster	Hauptstraße

Result 23

Tabelle: user_book

Erstellungs-Statement:

```
CREATE TABLE user_book (
    user_book_ID INT PRIMARY KEY AUTO_INCREMENT,
    user_ID INT NOT NULL,
    book_ID INT NOT NULL,
    condition_book VARCHAR(100) NOT NULL,
    description_book TEXT NOT NULL,
    notes VARCHAR(255),
    FOREIGN KEY (user_ID) REFERENCES user(user_ID) ON DELETE CASCADE,
    FOREIGN KEY (book_ID) REFERENCES book(book_ID) ON DELETE CASCADE
);
```

Dateneingabe:

```
INSERT INTO user_book (user_ID, book_ID, condition_book, description_book, notes)
VALUES (1, 1, 'gut', 'Gebrauchtes Buch, gepflegt', 'leichte Knicke im Umschlag');
```

Hinweise / Designentscheidungen:

- ON DELETE CASCADE, damit Bücher oder Nutzer ohne FK-Probleme gelöscht werden können
- notes optional, da nicht immer notwendig

Testabfragen:

-
```
SELECT ub.user_book_ID, u.nickname, b.title
FROM user_book ub
JOIN user u ON ub.user_ID = u.user_ID
JOIN book b ON ub.book_ID = b.book_ID
WHERE ub.user_book_ID = 1;
```

▶	1	BuchFan88	Passagier 23

-
```
SELECT ub.user_book_ID, u.nickname, b.title, ub.condition_book
FROM user_book ub
JOIN user u ON ub.user_ID = u.user_ID
JOIN book b ON ub.book_ID = b.book_ID
LIMIT 10;
```

	user_book_ID	nickname	title	condition_book
▶	1	BuchFan88	Passagier 23	gut
	2	BuchFan88	Erebos	gut
	3	Leseratte99	Sag's nicht weiter, Liebling!	in Ordnung
	4	PageTurner	Bad Monkeys	neu
	5	BookWorm	Der Herr der Ringe	gebraucht
	6	NovelAddict	Attack on Titan 1	gut
	7	NovelAddict	Attack on Titan 2	wie neu
	8	ReadLover	Passagier 23	wie neu
	9	StorySeeker	Attack on Titan 1	wie neu
	10	Bibliophile	Die unendliche Geschichte	gebraucht

Tabelle: user_book_location

Erstellungs-Statement:

```
CREATE TABLE user_book_location (
   user_book_loc_ID INT PRIMARY KEY AUTO_INCREMENT,
   user_book_ID INT NOT NULL,
   location_ID INT,
   available_ID INT,
   FOREIGN KEY (user_book_ID) REFERENCES user_book(user_book_ID) ON DELETE
CASCADE,
   FOREIGN KEY (location_ID) REFERENCES location(location_ID) ON DELETE SET NULL,
   FOREIGN KEY (available_ID) REFERENCES available(available_ID) ON DELETE SET NULL
);
```

Dateineingabe:

```
INSERT INTO user_book_location (user_book_ID, location_ID, available_ID) VALUES (1, 1, 1);
```

Hinweise / Designentscheidungen:

- Verknüpft Buch, Nutzer, Abholort und Verfügbarkeit
- ON DELETE CASCADE für sauberes Löschen des ganzen user_book_location-Objekts, wenn das user_book entfernt wird
- location_ID und available_ID auf nullable umgestellt, um ON DELETE SET NULL zu ermöglichen, bei Löschen der Location oder der Verfügbarkeit soll user-book-location-Objekts bestehen bleiben und der gelöschte Bereich nur auf null gesetzt werden können

Testabfrage:

```
SELECT ubl.user_book_loc_ID, u.nickname, b.title, l.city, a.state
FROM user_book_location ubl
JOIN user_book ub ON ubl.user_book_ID = ub.user_book_ID
JOIN user u ON ub.user_ID = u.user_ID
JOIN book b ON ub.book_ID = b.book_ID
JOIN location l ON ubl.location_ID = l.location_ID
JOIN available a ON ubl.available_ID = a.available_ID
LIMIT 10;
```

user_book_loc_ID	nickname	title	city	state
1	BuchFan88	Passagier 23	Marburg	verfügbar
2	BuchFan88	Erebos	Marburg	verfügbar
3	Leseratte99	Sag's nicht weiter, Liebling!	Freiburg	reserviert
4	PageTurner	Bad Monkeys	Hamburg	verliehen
5	BookWorm	Der Herr der Ringe	Freising	verfügbar
6	NovelAddict	Attack on Titan 1	Freising	verfügbar
7	NovelAddict	Attack on Titan 2	Freising	verfügbar
8	ReadLover	Passagier 23	Köln	verfügbar
9	StorySeeker	Attack on Titan 1	Köln	verfügbar
10	Bibliophile	Die unendliche Geschichte	München	reserviert

Tabelle: wishlist

Erstellungs-Statement:

```
CREATE TABLE wishlist (
   wishlist_ID INT PRIMARY KEY AUTO_INCREMENT,
   saved_by_ID INT NOT NULL,
   name_wishlist VARCHAR(20),
   FOREIGN KEY (saved_by_ID) REFERENCES user(user_ID) ON DELETE CASCADE
);
```

Dateneingabe:

```
INSERT INTO wishlist (saved_by_ID, name_wishlist) VALUES (1, 'Sommer 2025');
```

Hinweise / Designentscheidungen:

- Name optional
- ON DELETE CASCADE für sauberes Löschen

Testabfragen:

- SELECT * FROM wishlist LIMIT 10;

wishlist_ID	saved_by_ID	name_wishlist
13	1	Thriller-Favoriten
14	2	Fantasy-Abenteuer
15	3	Sommerromane
16	4	Manga-Sammlung
17	5	Historische Romane
18	6	NULL
19	7	Ratgeber & Self-Help
20	8	Science-Fiction Hits
21	9	Detektivgeschichten
22	10	Psychothriller

- SELECT * FROM wishlist WHERE saved_by_ID=1;

wishlist_ID	saved_by_ID	name_wishlist
13	1	Thriller-Favoriten
NULL	NULL	NULL

Tabelle: wishlist_user

Erstellungs-Statement:

```
CREATE TABLE wishlist_user (
    wishlist_ID INT NOT NULL,
    user_book_ID INT NOT NULL,
    PRIMARY KEY (wishlist_ID, user_book_ID),
    FOREIGN KEY (wishlist_ID) REFERENCES wishlist(wishlist_ID) ON DELETE CASCADE,
   FOREIGN KEY (user_book_ID) REFERENCES user_book(user_book_ID) ON DELETE CASCADE
);
```

Dateneingabe:

```
INSERT INTO wishlist_user (wishlist_ID, user_book_ID) VALUES (13, 7);
```

Hinweise/ Designentscheidungen:

- ON DELETE CASCADE für beide: Beim Löschen einer Wunschliste oder eines Buchs in user_book werden die zugehörigen Einträge automatisch entfernt
- - Abweichend von ERM Primärschlüssel nicht auto increment, sondern zusammengesetzt aus den beiden referenzierten Fremdschlüsseln (klarere und sinnvollere Zuordnung)

Testabfrage:

```
- SELECT wu.wishlist_ID, w.name_wishlist AS name_wishlist,
 ub.user_book_ID,
 u.nickname AS user_nickname,
 b.title AS book_title
FROM wishlist_user wu
JOIN wishlist w ON wu.wishlist_ID = w.wishlist_ID
JOIN user_book ub ON wu.user_book_ID = ub.user_book_ID
JOIN user u ON ub.user_ID = u.user_ID
JOIN book b ON ub.book_ID = b.book_ID
LIMIT 10;
```

wishlist_ID	name_wishlist	user_book_ID	user_nickname	book_title
13	Thriller-Favoriten	1	BuchFan88	Passagier 23
14	Fantasy-Abenteuer	2	BuchFan88	Erebos
15	Sommerromane	3	Leseratte99	Sag's nicht weiter, Liebling!
16	Manga-Sammlung	4	PageTurner	Bad Monkeys
17	Historische Romane	5	BookWorm	Der Herr der Ringe
18	NULL	6	NovelAddict	Attack on Titan 1
19	Ratgeber & Self-Help	7	NovelAddict	Attack on Titan 2
20	Science-Fiction Hits	8	ReadLover	Passagier 23
21	Detektivgeschichten	9	StorySeeker	Attack on Titan 1