

Einführung in Datenbanken

März 2025

Plan für die Woche

Montag

- Was sind Datenbanken?
- Normalisierung

Dienstag

- Was ist SQL?
- DDL und DML

Mittwoch

- DDL und DML-Fortsetzung

Donnerstag

- JOIN

Freitag

- Aggregat

Plan für heute

- Was ist SQL?
- DDL und DML
- CREATE; ALTER; DROP; INSERT; SELECT

SQL, DDL, DML

Was ist SQL?

- SQL steht für „Structured Query Language“
- ist eine Datenbanksprache zum Definieren und Ansprechen einer relationalen Datenbank
 - Daten können verwaltet, bearbeitet und abgerufen werden

DDL – Data Definition Language

- DDL steht für Data Definition Language
- umfasst SQL-Befehle, die zum Erstellen und Verwalten von Datenbankstrukturen verwendet werden
 - Erstellen, Ändern und Löschen von Tabellen und Datenbanken
 - CREATE – Erstellen
 - ALTER – Ändern
 - DROP – Löschen

CREATE DATABASE

– wird genutzt, um eine Datenbank zu erstellen

– Syntax:

```
CREATE DATABASE databasename;
```

– Beispiel:

```
CREATE DATABASE Hospital;
```

USE DATABASE

- wird genutzt, um zu signalisieren, welche Datenbank genutzt werden soll
- muss einmal vorher ausgeführt werden (wenn mehrere Datenbanken existieren)

– Syntax:

```
USE databasename;
```

– Beispiel:

```
USE Hospital;
```


DEMO CREATE

Aufgabe

Erzeuge in deinem Datenbankmanagementsystem eine Datenbank SchoolManagementSystem

```
CREATE DATABASE databasename;
```



CREATE TABLE

- wird genutzt, um eine Tabelle innerhalb einer Datenbank zu erstellen
- Syntax:

```
CREATE TABLE table_name (  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
    ....  
);
```

Wichtige Datentypen

- jede Spalte hat einen Datentyp
- man kann sich eine Spalte wie ein Datenfeld in einer Java-Klasse vorstellen
- Beispiel in Java: Klasse Patient

```
public class Patient {  
    private String vorname;  
    private String nachname;  
    private String email;  
    private LocalDate geburtsdatum;  
    private int alter;  
    private double gewicht;  
}
```

```
CREATE TABLE table_name (  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
    ....
```

Wichtige Datentypen

);

SQL-Datentyp	Beschreibung	Vergleich mit Java
BOOL / BOOLEAN	Speichert Wahrheitswerte (true/false)	boolean
INT/INTEGER	Ganzzahl, Größe ist optional	int, long
FLOAT	Gleitkommazahl mit fester Größe und Dezimalstellen	float
DOUBLE	Präzisere Gleitkommazahl als FLOAT	double
DECIMAL(size,d)	Dezimalzahl mit genauer Präzision	BigDecimal
DATE	Speichert ein Datum (YYYY-MM-DD)	LocalDate
DATETIME	Speichert Datum und Uhrzeit (YYYY-MM-DD HH:MM:SS)	LocalDateTime
YEAR	Speichert ein Jahr (YYYY)	int
CHAR(size)	Zeichenkette fester Länge	char[] oder String
VARCHAR(size)	Zeichenkette variabler Länge	String

Welche Datentypen wählen?

– Beispiel: Tabelle Patienten

- Vorname:
- Nachname:
- EMail:
- Geburtsdatum:
- Alter:
- Gewicht:

SQL-Datentyp	Beschreibung	Vergleich mit Java
BOOL / BOOLEAN	Speichert Wahrheitswerte (true/false)	<u>boolean</u>
INT/INTEGER	Ganzzahl, Größe ist optional	int, long
FLOAT	Gleitkommazahl mit fester Größe und Dezimalstellen	float
DOUBLE	Präzisere Gleitkommazahl als FLOAT	double
DECIMAL(size,d)	Dezimalzahl mit genauer Präzision	BigDecimal
DATE	Speichert ein Datum (YYYY-MM-DD)	LocalDate
DATETIME	Speichert Datum und Uhrzeit (YYYY-MM-DD HH:MM:SS)	LocalDateTime
YEAR	Speichert ein Jahr (YYYY)	int
CHAR(size)	Zeichenkette fester Länge	char[] oder String
VARCHAR(size)	Zeichenkette variabler Länge	String



Wichtige Datentypen

- Beispiel: Tabelle Patienten
 - Vorname: VARCHAR
 - Nachname: VARCHAR
 - EMail: VARCHAR
 - Geburtsdatum: DATE
 - Alter: INT
 - Gewicht: DOUBLE

```
CREATE TABLE Patienten (  
    PatientenID INT,  
    Vorname VARCHAR(50),  
    Nachname VARCHAR(50),  
    Email VARCHAR(100),  
    Geburtsdatum DATE,  
    Alter INT,  
    Gewicht DOUBLE(5,2)  
);
```



Definition eines Primärschlüssels

- es ist wichtig, dass jeder Kunde eindeutig identifiziert werden kann
 - Dazu braucht es Primärschlüssel
- es ist möglich, eine Spalte als Primärschlüssel zu definieren
- Syntax:

```
CREATE TABLE Patienten (  
    PatientenID INT PRIMARY KEY,  
    Vorname VARCHAR(50),  
    Nachname VARCHAR(50),  
    Email VARCHAR(100),  
    Geburtsdatum DATE,  
    Alter INT,  
    Gewicht DOUBLE(5,2)  
);
```


AUTO_INCREMENT

- man kann das Hochzählen eines Primärschlüssel automatisieren
 - Beim Einfügen eines Datensatzes in die Tabelle wird automatisch der Primärschlüssel hochgezählt
 - Funktioniert **NUR** bei Primärschlüssel vom Datentyp INT/INTEGER
- Schlüsselwort: AUTO_INCREMENT
- Syntax:

```
CREATE TABLE Patienten (  
    PatientenID INT PRIMARY KEY AUTO_INCREMENT,  
    Vorname VARCHAR(50),  
    Nachname VARCHAR(50),  
    Email VARCHAR(100),  
    Geburtsdatum DATE,  
    Alter INT,  
    Gewicht DOUBLE(5,2)  
);
```

NOT NULL

- man kann angeben, welche Spalten keine Nullwerte enthalten dürfen
- Schlüsselwort: NOT NULL
- Syntax:

```
CREATE TABLE Patienten(  
  PatientenID INT PRIMARY KEY AUTO_INCREMENT NOT NULL,  
  Firstname VARCHAR(50) NOT NULL,  
  Lastname VARCHAR(50) NOT NULL,  
  Email VARCHAR(100),  
  DayOfBirth DATE NOT NULL,  
  Age INT,  
  Weight DECIMAL(5,2)  
);
```

DEMO CREATE

Aufgabe

```
CREATE TABLE table_name (  
    column1 datatype,  
    column2 datatype,  
    column3 datatype,  
    ....  
);
```

Erzeuge in deiner Datenbank eine Tabelle Student mit folgenden Daten:

studentID
firstname
lastname
dayOfBirth
eMail
grade

SQL-Datentyp	Beschreibung	Vergleich mit Java
BOOL / BOOLEAN	Speichert Wahrheitswerte (true/false)	<u>boolean</u>
INT/INTEGER	Ganzzahl, Größe ist optional	int, long
FLOAT	Gleitkommazahl mit fester Größe und Dezimalstellen	float
DOUBLE	Präzisere Gleitkommazahl als FLOAT	double
DECIMAL(size,d)	Dezimalzahl mit genauer Präzision	BigDecimal
DATE	Speichert ein Datum (YYYY-MM-DD)	LocalDate
DATETIME	Speichert Datum und Uhrzeit (YYYY-MM-DD HH:MM:SS)	LocalDateTime
YEAR	Speichert ein Jahr (YYYY)	int
CHAR(size)	Zeichenkette fester Länge	char[] oder String
VARCHAR(size)	Zeichenkette variabler Länge	String



ALTER TABLE

- wird genutzt, um eine Tabelle zu verändern
 - Spalten hinzufügen/löschen (ADD/DROP)
 - Datentypen von Spalten ändern (MODIFY)

- Syntax:

```
ALTER TABLE table_name  
ADD column_name datatype;
```

- Beispiel:

```
ALTER TABLE Patienten ADD Telefonnummer VARCHAR(20);
```

DEMO ALTER

Aufgabe

Füge der Tabelle Student eine weitere Spalte phoneNumber hinzu.

```
ALTER TABLE table_name
ADD column_name datatype;
```

SQL-Datentyp	Beschreibung	Vergleich mit Java
BOOL / BOOLEAN	Speichert Wahrheitswerte (true/false)	<u>boolean</u>
INT/INTEGER	Ganzzahl, Größe ist optional	int, long
FLOAT	Gleitkommazahl mit fester Größe und Dezimalstellen	float
DOUBLE	Präzisere Gleitkommazahl als FLOAT	double
DECIMAL(size,d)	Dezimalzahl mit genauer Präzision	BigDecimal
DATE	Speichert ein Datum (YYYY-MM-DD)	LocalDate
DATETIME	Speichert Datum und Uhrzeit (YYYY-MM-DD HH:MM:SS)	LocalDateTime
YEAR	Speichert ein Jahr (YYYY)	int
CHAR(size)	Zeichenkette fester Länge	char[] oder String
VARCHAR(size)	Zeichenkette variabler Länge	String



DROP TABLE/DATABASE

– wird genutzt, um eine Datenbank oder Tabelle zu löschen

– Syntax:

```
DROP DATABASE databasename;
```

```
DROP TABLE table_name;
```

– Beispiel:

```
DROP TABLE Patienten;
```


DEMO DROP

Aufgabe

Füge eine weitere Tabelle deiner Datenbank hinzu. Lösche sie direkt.

```
DROP TABLE table_name;
```



Aufgabe (ca. 20-25 Minuten)

Erstelle eine Datenbank mit dem Namen „Library“.

In dieser Datenbank soll eine Tabelle mit dem Namen „Books“ angelegt werden. Die Tabelle soll folgende Anforderungen erfüllen:

- Jede Zeile in der Tabelle muss eindeutig identifizierbar sein. Dafür soll eine Spalte mit einer eindeutigen ID erstellt werden.
- Die Tabelle soll folgende Spalten enthalten: den Titel des Buches, den Namen des Autors, das Erscheinungsjahr und das Genre.
- Optional kann auch eine Spalte hinzugefügt werden, die die ISBN-Nummer des Buches speichert.



DML – Data Manipulation Language

- DML steht für Data Manipulation Language
- wird verwendet, um Daten innerhalb einer Datenbank zu verwalten
 - Es werden Daten einer Tabelle manipuliert
- Befehle:
 - INSERT – Einfügen neuer Daten in eine Tabelle
 - SELECT – Aufrufen von Daten aus einer oder mehrerer Tabellen
 - UPDATE – Ändern bestehender Daten einer Tabelle
 - DELETE – Löschen von Daten einer Tabelle

INSERT INTO

- wird genutzt, um Daten in eine Tabelle einzupflegen
- Syntax:
 - Wenn man spezifische Spalten einfügen möchte (z.B. Spalte mit Primary Key auslassen):

```
INSERT INTO table_name (column1, column2, column3, ...)  
VALUES (value1, value2, value3, ...);
```

- Wenn man Werte in alle Spalten der Tabelle hinzufügen möchte:

```
INSERT INTO table_name  
VALUES (value1, value2, value3, ...);
```

Zum Hinzufügen mehrerer
Datensätze auf einmal:

```
... VALUES(v1, v2,v3),  
          (v4,v5,v6),  
          (v7, v8, v9);
```

INSERT INTO - Beispiel

```
CREATE TABLE Patienten (  
    PatientenID INT PRIMARY KEY AUTO_INCREMENT,  
    Vorname VARCHAR(50),  
    Nachname VARCHAR(50),  
    Email VARCHAR(100),  
    Geburtsdatum DATE,  
    Alter INT,  
    Gewicht DOUBLE(5,2)  
);
```

```
INSERT INTO Patienten (Vorname, Nachname, Email, Geburtsdatum, Alter, Gewicht)  
VALUES ('Max', 'Mustermann', 'max.mustermann@email.com', '1990-05-15', 34, 78.5);
```

DEMO INSERT

– Wenn man spezifische Spalten einfügen möchte (z.B. Spalte mit Primary Key auslassen):

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

– Wenn man Werte in alle Spalten der Tabelle hinzufügen möchte:

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

Aufgabe

Zum Hinzufügen mehrerer
Datensätze auf einmal:
... VALUES(v1, v2,v3),
 (v4,v5,v6),
 (v7, v8, v9);

Füge folgende Bücher zur Tabelle „Book“ hinzu:

- 1.Titel: "1984", Autor: George Orwell, Erscheinungsjahr: 1949, Genre: Dystopie, ISBN: 9780451524935
- 2.Titel: "Pride and Prejudice", Autor: Jane Austen, Erscheinungsjahr: 1813, Genre: Romanze, ISBN: 9780141439518
- 3.Titel: "To Kill a Mockingbird", Autor: Harper Lee, Erscheinungsjahr: 1960, Genre: Drama, ISBN: 9780060935467
- 4.Titel: "The Great Gatsby", Autor: F. Scott Fitzgerald, Erscheinungsjahr: 1925, Genre: Klassiker, ISBN: 9780743273565
- 5.Titel: "The Catcher in the Rye", Autor: J.D. Salinger, Erscheinungsjahr: 1951, Genre: Coming-of-Age, ISBN: 9780316769488



Plan für die Woche

Montag

- Was sind Datenbanken?
- Normalisierung

Dienstag

- Was ist SQL?
- DDL und DML

Mittwoch

- DDL und DML-Fortsetzung

Donnerstag

- JOIN

Freitag

- Aggregat

FOREIGN KEY

- wird genutzt, um die Spalte mit den Fremdschlüsseln in einer Tabelle zu markieren
- Syntax:

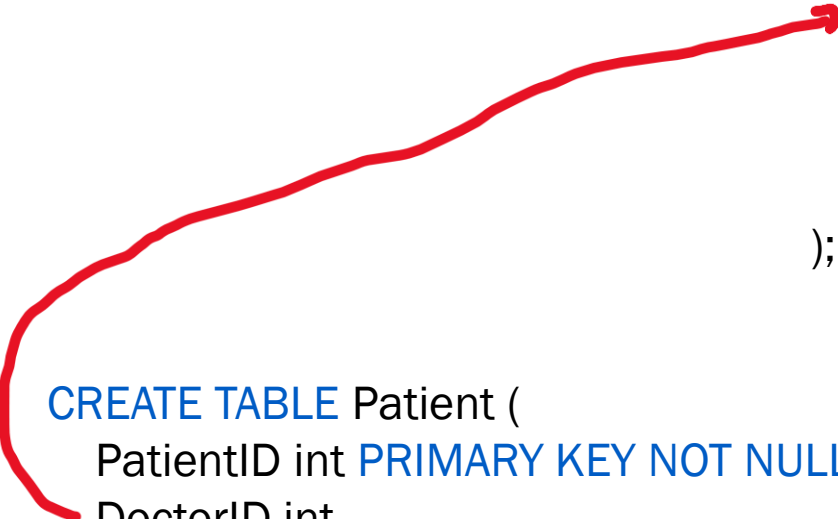
```
CREATE TABLE tablename (  
    column_name datatype,  
    .... ,  
    FOREIGN KEY (ForeignKeyOfThisTable) REFERENCES OtherTable(PrimaryKey)  
);
```

FOREIGN KEY

– Beispiel:

```
CREATE TABLE Doctor (  
  DoctorID int PRIMARY KEY NOT NULL,  
  Firstname VARCHAR(50),  
  Lastname VARCHAR(50),  
  SpecialField VARCHAR(50)  
);
```

```
CREATE TABLE Patient (  
  PatientID int PRIMARY KEY NOT NULL,  
  DoctorID int,  
  Firstname VARCHAR(50),  
  Lastname VARCHAR(50),  
  FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID)  
);
```



Jeder Patient hat einen
Doktor zugewiesen.

DEMO FOREIGN KEY UND WIEDERHOLUNG

Aufgabe

Erstelle eine Tabelle Author. Ein Autor hat eine ID, einen Vornamen und einen Nachnamen.

Zu jedem Buch gibt es einen Autor.

Wie müsste die Struktur der Buchtabelle nun aussehen?

Ändere die Tabelle Books so, dass sie mit der Autortabelle verknüpft wird.



SELECT (1)

- wird genutzt, um Daten aus einer Tabelle zu holen
- Syntax:
 - Wenn man alle Daten haben möchte:

```
SELECT *  
FROM table_name;
```

```
SELECT * FROM Patienten;
```

- Wenn man spezifische Spalten aus einer Tabelle haben möchte:

```
SELECT column1, column2, ...  
FROM table_name;
```

```
SELECT Lastname FROM Patient;
```

Sortieren

- Datensätze können sortiert ausgegeben werden

```
SELECT column1, column2, ...  
FROM table_name  
ORDER BY column1, column2, ... ASC | DESC;
```

Limitieren

- es kann eine Anzahl an Datensätzen ausgegeben werden

```
SELECT column_name(s)  
FROM table_name  
WHERE condition  
LIMIT number;
```


DEMO SELECT

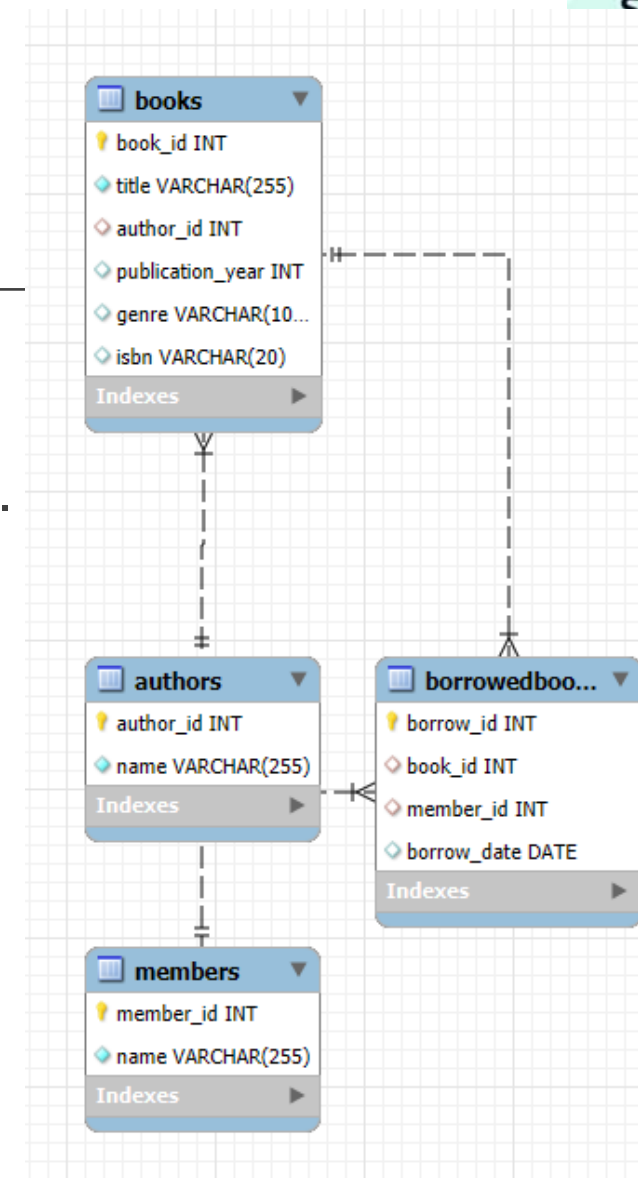
Aufgabe

Gebe alle Bücher deiner Datenbank aus.

Gebe alle Titel der Tabelle Books in absteigender Reihenfolge aus.

Gebe Titel und Jahr der Tabelle Books aus in aufsteigender Reihenfolge des Jahres.

Es sollen nur 5 Einträge ausgegeben werden.



SELECT (2)

- SELECT-Anfragen können spezifiziert werden
 - Somit kann man noch konkrete Daten aus der Datenbank abfragen
 - Ähnlich wie If-Abfragen in Java: mittels Bedingung!

- Syntax:

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition;
```

WHERE

- mittels WHERE kann man die Datensätze filtern
 - Somit können Datensätze aus der Tabelle entnommen werden, die eine bestimmte Bedingung erfüllen
- Beispiel:

SELECT * FROM Patient WHERE Lastname = 'Mustermann';

Vgl. zu Java:

```
for(Patient p: patientList) {  
    if (p.nachname.equals("Mustermann")) {  
        System.out.println(p);  
    }  
}
```

WHERE

– mögliche einfache Operationen:

Operator	Beschreibung
=	Überprüft den Inhalt auf Gleichheit
>, <, >=, <=	Vergleichsoperatoren
<>, !=	Ungleich
BETWEEN	Zwischen einer bestimmten Spanne
NOT	Negiert eine Bedingung

WHERE Klauseln verknüpfen

- Bedingungen können mittels OR und AND verknüpft werden
- Beispiel:

```
SELECT * FROM Patient WHERE Age NOT BETWEEN 18 AND 65;
```

```
SELECT * FROM Patient WHERE Lastname = 'Mustermann' AND Age > 10;
```

```
SELECT * FROM Patient WHERE Lastname = 'Mustermann' OR Lastname = 'Musterfrau';
```

```
SELECT * FROM Patient WHERE Age > 10 AND (Lastname = 'Mustermann' OR Lastname = 'Musterfrau');
```

```
SELECT * FROM Patient WHERE NOT Lastname = 'Mustermann';
```

Vgl. zu Java:

```
for(Patient p: patientList) {  
    if ((p.nachname.equals("Mustermann") && p.alter > 10) ||  
        (p.nachname.equals("Musterfrau") && p.alter > 10)) {  
        System.out.println(p);  
    }  
}
```

IS NULL / IS NOT NULL

- es können Datensätze geholt werden, in denen eine Spalte Nullwerte enthält oder nicht

```
SELECT column_names  
FROM table_name  
WHERE column_name IS NULL;
```

```
SELECT column_names  
FROM table_name  
WHERE column_name IS NOT NULL;
```

Vgl. zu Java:

```
for(Patient p: patientList) {  
    if (p.nachname!=null) {  
        System.out.println(p);  
    }  
}
```

LIKE

- es können Datensätze geholt werden, in denen Spalten mit einem gewissen Buchstaben/ einer gewissen Zeichenkette anfangen

```
SELECT column1, column2, ...  
FROM table_name  
WHERE columnN LIKE pattern;
```

Vgl. zu Java:

```
for(Patient p: patientList) {  
    if (p.nachname.startsWith („Mu“)) {  
        System.out.println(p);  
    }  
}
```

```
for(Patient p: patientList) {  
    if (p.nachname.endsWith („Mu“)) {  
        System.out.println(p);  
    }  
}
```

```
for(Patient p: patientList) {  
    if (p.nachname.contains („stern“)) {  
        System.out.println(p);  
    }  
}
```


WILDCARDS

- um etwas sinnvolles als Pattern anzugeben, müssen Wildcards genutzt werden

WHERE Lastname LIKE 'Blau%'	Finds any values that start with “Blau”
WHERE Lastname LIKE '%schmied'	Finds any values that end with “schmied”
WHERE Lastname LIKE '%or%'	Finds any values that have "or" in any position
WHERE Lastname LIKE '_r%'	Finds any values that have "r" in the second position
WHERE Lastname LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

DEMO SELECT WHERE

Aufgabe (ca. 20-30 Minuten)

Rufe alle Informationen zu dem Buch mit dem Titel „1984“ ab.

Finde alle Bücher, die im Genre „Romanze“ eingeordnet sind.

Erstelle eine Abfrage, die alle Bücher anzeigt, die vor dem Jahr 1950 veröffentlicht wurden.

Suche alle Bücher, die zwischen 1950 und 2000 veröffentlicht wurden und zum Genre „Drama“ oder „Klassiker“ gehören.

Rufe alle Bücher ab, bei denen das Erscheinungsjahr entweder zwischen 1800 und 1900 oder das Genre „Dystopie“ ist.

Finde alle Bücher, deren Titel mit "The" beginnen (Verwendung von Wildcards)

Zeige alle Bücher, bei denen das Erscheinungsjahr angegeben ist



Subqueries

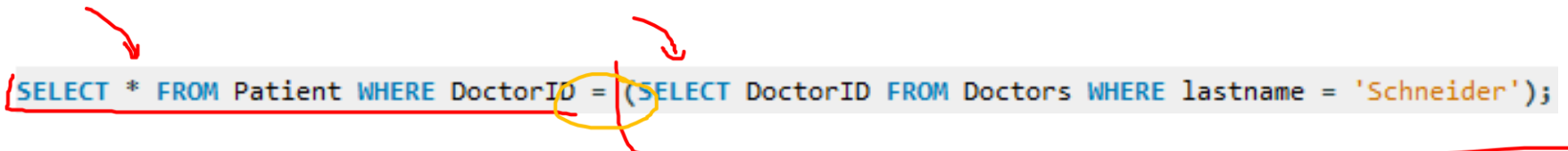
- Query bedeutet Abfrage
- Subquery ist eine Unterabfrage, die innerhalb einer anderen Abfrage eingebettet ist
- Beispiel: Wir möchten alle Patienten finden, die von Dr. Schneider betreut werden
- Schritte:
 - Wie kriegt man alles vom Patienten?
 - `SELECT * FROM Patient WHERE DoctorID ?`
 - Wie kriegt man die ID des Doktors mit dem Nachnamen Schneider?
 - `SELECT DoctorID FROM Doctors WHERE Lastname = 'Schneider';`



Wir kriegen eine ID

Subqueries

- Query bedeutet Abfrage
- Subquery ist eine Unterabfrage, die innerhalb einer anderen Abfrage eingebettet ist
- Beispiel: Wir möchten alle Patienten finden, die von Dr. Schneider betreut werden



```
SELECT * FROM Patient WHERE DoctorID = (SELECT DoctorID FROM Doctors WHERE lastname = 'Schneider');
```

IN / NOT IN

Wird auch in
Subqueries genutzt

- durch das Schlüsselwort kann die Spalte mehrere Werte (nicht) „annehmen“
- Beispiel:
 - Alle Patienten, die entweder Mustermann, Musterfrau, Becker, Schmidt oder Hoffmann im Nachnamen heißen

```
SELECT * FROM PATIENT WHERE LASTNAME IN ('Mustermann', 'Musterfrau', 'Becker', 'Schmidt', 'Hoffmann');
```

- Alle Patienten, die weder Mustermann, Musterfrau, Becker, Schmidt noch Hoffmann im Nachnamen heißen

```
SELECT * FROM PATIENT WHERE LASTNAME NOT IN ('Mustermann', 'Musterfrau', 'Becker', 'Schmidt', 'Hoffmann');
```

DEMO SUBQUERY

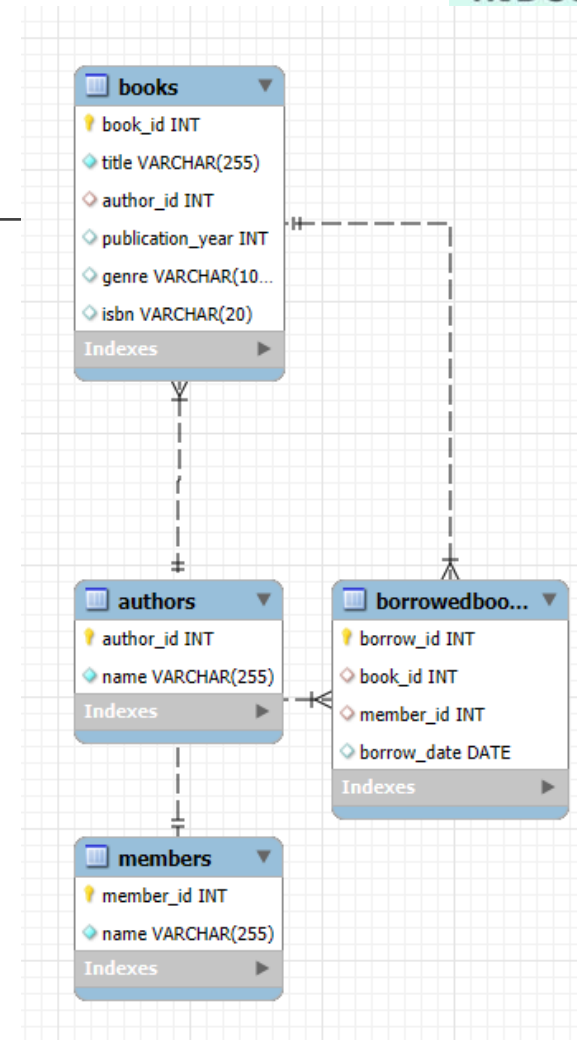
Aufgabe (ca. 20 Minuten)

Zeige alle Bücher, die von **J.K. Rowling** geschrieben wurden.

Zeige alle Bücher, die von **Agatha Christie** geschrieben wurden.

Zeige alle Bücher, die nach dem **1. März 2024** ausgeliehen wurden.

Zeige alle Mitglieder, die das Buch „**1984**“ ausgeliehen haben.



UPDATE

– UPDATE-Anfragen aktualisieren einen spezifischen Datensatz

– Syntax:

```
UPDATE table_name  
SET column1 = value1, column2 = value2, ...  
WHERE condition;
```

– Beispiel:

```
UPDATE Patient SET Email = 'max.neu@email.com' WHERE PatientID = 1;
```

Vgl. zu Java:

```
for(Patient p: patientList) {  
    if (p.id == 1){  
        p.setEmail("max.neu@email.com");  
    }  
}
```

DEMO UPDATE

UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;

Aufgabe

Ändere das Erscheinungsjahr des Buches „1984“ auf 1950.

Ändere die ISBN für das Buch „Pride and Prejudice“ auf „9780000000002“.

Setze für alle Bücher, die vor 1900 veröffentlicht wurden, das Genre auf „Klassiker“.

Ändere den Titel des Buches „The Great Gatsby“ in „The Greatest Gatsby“.



DELETE

– DELETE-Anfragen löschen einen spezifischen Datensatz

– Syntax:

```
DELETE FROM table_name WHERE condition;
```

– Beispiel:

```
DELETE FROM Patienten WHERE PatientID = 1;
```

DEMO DELETE

```
DELETE FROM table_name WHERE condition;
```

Aufgabe

Lösche das Buch mit dem Titel „The Catcher in the Rye“ aus der Tabelle.

Entferne alle Bücher, die vor dem Jahr 1900 veröffentlicht wurden.

Lösche alle Bücher, deren Genre „Dystopie“ ist oder deren Erscheinungsjahr nach 2000 liegt.

Entferne alle Bücher, bei denen sowohl das Genre „Romanze“ als auch das Erscheinungsjahr vor 1850 ist.

