Praxisblock

MariaDB

INF1006 Praktische Informatik 2

von Prof. Dr. Weigel

Vorlesungsinhalte

❶ Server-seitige Programmierung

1. Eclipse vert.x
2. Routen und CORS
3. Webservices mit REST
4. Sessions
5. Praxisblock

❷ Verteilte Systeme

1. Definition und Architekturen
2. Kommunikationsmuster
3. Versionskontrollsysteme
4. Praxisblock

❸ Datenbanken

1. Datenbanksysteme
2. ER- & Relationenmodell
3. Normalformen
4. Structured Query Language
5. Praxisblock

❹ Betriebssysteme

1. Betriebssysteme und Rechnerarchitekturen
2. Prozesse
3. Speicherverwaltung
4. Praxisblock

❺ Cybersecurity

1. Bedrohungen & Gefährdungen
2. XSS, CSRF und SQL injection
3. Zugriffsrollen und Passwörter
4. Verschlüsselungen
5. Praxisblock

Aufgaben für heute

1. Verbinden mit der MariaDB Datenbank
2. Einrichten einer Trainerdatenbank
3. Queryanfragen an die Datenbank
4. Integration in vert.x mit JDBC

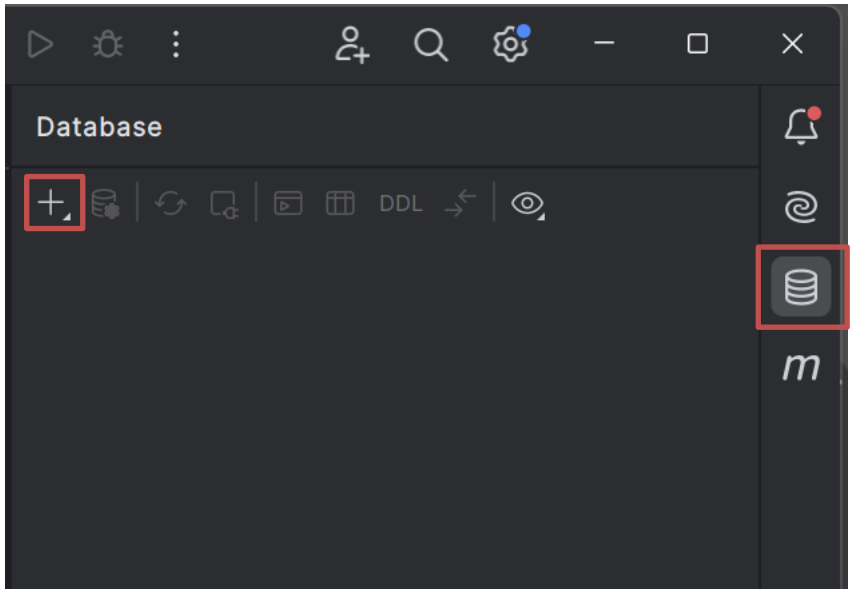
IntelliJ IDEA Ultimate

Die **Ultimate**-Version enthält ein Datenbank Plugin, welches wir in diesem Praxisblock verwenden

Alternativen

1. **IntelliJ DataGrip**
   GUI Progamm, identisch mit dem IDEA-Plugin
   https://www.jetbrains.com/datagrip/

1. **MariaDB Client**
   Commandozeilen Programm für MariaDB Datenbanken zum Testen und Skripten
   https://mariadb.com/docs/server/connect/clients/mariadb-client/



MariaDB für Übungen und Hausübung 3

Jede HÜ-Gruppe erhält eine Datenbank auf einem THM-gehosteten Server. Diese Datenbank können Sie für den Praxisblock, die Übung und die Hausübung 3 verwenden*

Achten Sie aber darauf, dass Sie **eindeutige Tabellennamen** (z.B. durch Präfix "homework_pokemon") verwenden

Zugriff nur aus dem THM-Netzwerk:

- an der THM per Eduroam
- zu Hause per VPN

Server

ip1-dbs.mni.thm.de

Port

3306

User

<THM-E-Mail laut Moodle>

Password

<THM-E-Mail laut Moodle>

Datenbank, z.B.

PI2_Fr_1_A

PI2_Mi_3_C

*) In der Praxis sollte man für jedes Projekt eine eigene Datenbank verwenden

Die Trainerdatenbank als echte Datenbank

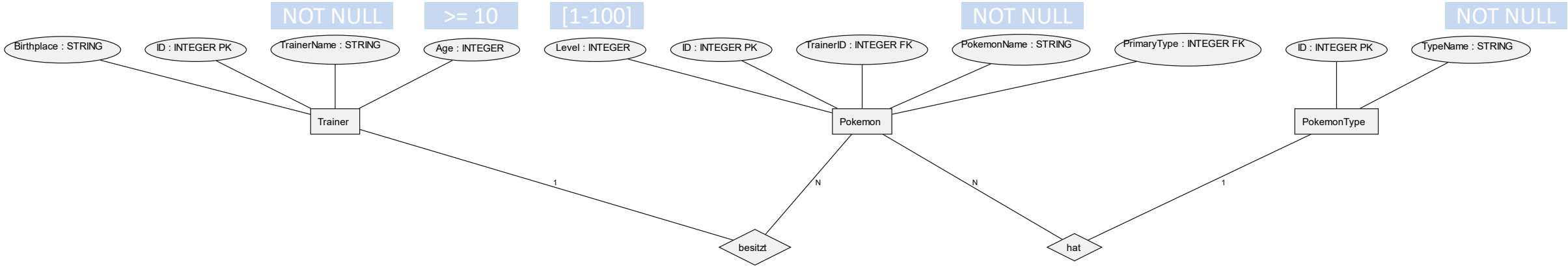
In Hausübung 1 haben wir eine Trainerdatenbank als REST-Server entwickelt,
die Daten jedoch nur im Arbeitsspeicher (RAM) gelagert.

Aufgabenstellung:

In der Trainerdatenbank sollen Trainer und ihre Pokemon gespeichert werden. Ein Trainer hat einen Namen, ein Alter und ein Geburtsort. Jeder Trainer bekommt eine eindeutige ID. Trainer besitzen mehrere Pokemon, welche einen Namen, ein Level und einen primären Typen besitzen. Jedes Pokemon soll eine eindeutige ID bekommen. Die Typen sollen in einer eigenen Tabelle definiert werden, sodass der Name des Typen nur an einer Stelle abgelegt werden muss.

PlantUML: TrainerDB.puml

Modellierung als Entity-Relationship-Modell

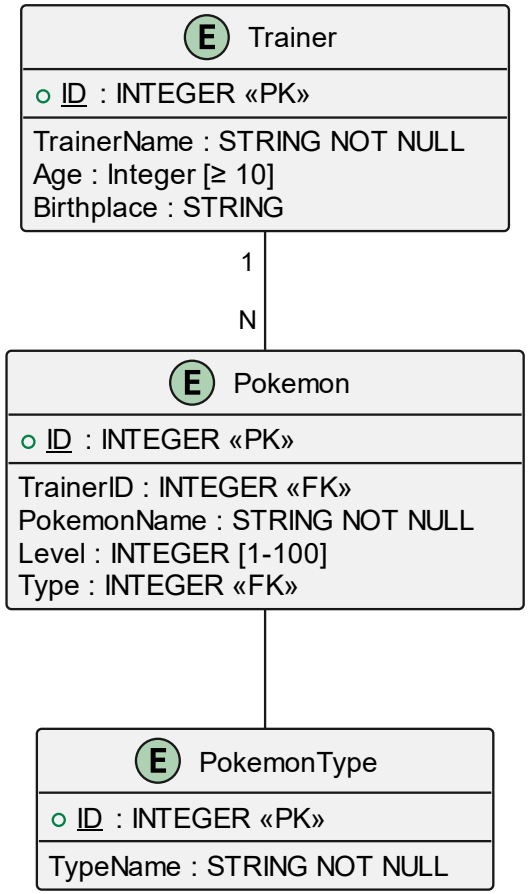


PlantUML: ERM-TrainerDB.puml

Umwandlung zum Relationenmodell

Im nächsten Schritt wandeln wir das ERM in ein Relationenmodell um. Dabei sollen die Entitäten und ggf. die Relationen, sowie zusammengesetzte und mehrwertige Attribute in eigene Tabellen umgeformt werden.

Anmerkung: Die Typen sind hier noch abstrakt als Domänen angegeben und nicht als konkreter SQL-Typ (z.B. VARCHAR)



PlantUML: RM-TrainerDB.puml

Anlegen der Tabellen in MariaDB

```
CREATE TABLE trainers (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    trainer_name VARCHAR(50) NOT NULL,
    age INT UNSIGNED CHECK (age >= 10),
    birthplace VARCHAR(50));

CREATE TABLE pokemon_types (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    type_name VARCHAR(50) NOT NULL);

CREATE TABLE pokemon (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    trainer_id INT UNSIGNED REFERENCES trainers(id),
    pokemon_name VARCHAR(50) NOT NULL,
    level INT UNSIGNED DEFAULT 1 CHECK (level >= 1 AND level <= 100),
    primary_type INT UNSIGNED REFERENCES pokemon_types(id));
```

Einträge der Pokemon Typen erstellen

```
INSERT INTO pokemon_types (id, type_name) VALUES
(1, 'Feuer'),
(2, 'Wasser'),
(3, 'Elektro'),
(4, 'Pflanze'),
(5, 'Psycho'),
(6, 'Flug'),
(7, 'Gestein'),
(8, 'Eis'),
(9, 'Boden'),
(10, 'Gift');
```

Einträge der Trainer und Pokemon erstellen

```
INSERT INTO trainers (id, trainer_name, age, birthplace) VALUES
(1, 'Ash Ketchum', 10, 'Alabastia'),
(2, 'Misty', 12, 'Azuria City'),
(3, 'Rocko', 15, 'Mamoria City'),
(4, 'Maike', 10, 'Blütenburg City'),
(5, 'Lucia', 10, 'Zweiblattdorf');

INSERT INTO pokemon (id, trainer_id, pokemon_name, level, primary_type) VALUES
(1, 1, 'Pikachu', 25, 3),
(2, 1, 'Glurak', 36, 1);
```

Wie lesen wir alle Trainer aus die genau 10 Jahre alt sind?

```
CREATE TABLE trainers (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    trainer_name VARCHAR(50) NOT NULL,
    age INT UNSIGNED CHECK (age >= 10),
    birthplace VARCHAR(50));

CREATE TABLE pokemon_types (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    type_name VARCHAR(50) NOT NULL);

CREATE TABLE pokemon (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    trainer_id INT UNSIGNED REFERENCES trainers(id),
    pokemon_name VARCHAR(50) NOT NULL,
    level INT UNSIGNED DEFAULT 1 CHECK (level >= 1 AND level <= 100),
    primary_type INT UNSIGNED REFERENCES pokemon_types(id));
```

Wie lesen wir die Pokemon vom Trainer mit id=1 aus?

```
CREATE TABLE trainers (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    trainer_name VARCHAR(50) NOT NULL,
    age INT UNSIGNED CHECK (age >= 10),
    birthplace VARCHAR(50));

CREATE TABLE pokemon_types (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    type_name VARCHAR(50) NOT NULL);

CREATE TABLE pokemon (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    trainer_id INT UNSIGNED REFERENCES trainers(id),
    pokemon_name VARCHAR(50) NOT NULL,
    level INT UNSIGNED DEFAULT 1 CHECK (level >= 1 AND level <= 100),
    primary_type INT UNSIGNED REFERENCES pokemon_types(id));
```

Wie erhalten wir das höchste Pokemon Level?

```sql
CREATE TABLE trainers (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    trainer_name VARCHAR(50) NOT NULL,
    age INT UNSIGNED CHECK (age >= 10),
    birthplace VARCHAR(50));

CREATE TABLE pokemon_types (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    type_name VARCHAR(50) NOT NULL);

CREATE TABLE pokemon (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    trainer_id INT UNSIGNED REFERENCES trainers(id),
    pokemon_name VARCHAR(50) NOT NULL,
    level INT UNSIGNED DEFAULT 1 CHECK (level >= 1 AND level <= 100),
    primary_type INT UNSIGNED REFERENCES pokemon_types(id));
```

Wie erhalten wir die Namen der drei Pokemon mit dem höchsten Level?

```sql
CREATE TABLE trainers (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    trainer_name VARCHAR(50) NOT NULL,
    age INT UNSIGNED CHECK (age >= 10),
    birthplace VARCHAR(50));

CREATE TABLE pokemon_types (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    type_name VARCHAR(50) NOT NULL);

CREATE TABLE pokemon (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    trainer_id INT UNSIGNED REFERENCES trainers(id),
    pokemon_name VARCHAR(50) NOT NULL,
    level INT UNSIGNED DEFAULT 1 CHECK (level >= 1 AND level <= 100),
    primary_type INT UNSIGNED REFERENCES pokemon_types(id));
```

Wie erhalte ich eine Tabelle mit Trainernamen und dazugehörigen Pokemonnamen?

```sql
CREATE TABLE trainers (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    trainer_name VARCHAR(50) NOT NULL,
    age INT UNSIGNED CHECK (age >= 10),
    birthplace VARCHAR(50));

CREATE TABLE pokemon_types (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    type_name VARCHAR(50) NOT NULL);

CREATE TABLE pokemon (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    trainer_id INT UNSIGNED REFERENCES trainers(id),
    pokemon_name VARCHAR(50) NOT NULL,
    level INT UNSIGNED DEFAULT 1 CHECK (level >= 1 AND level <= 100),
    primary_type INT UNSIGNED REFERENCES pokemon_types(id));
```

Wie bekommen wir eine Tabelle mit den IDs der Trainer, welche mindestens ein Pokemon auf oder über Level 30 besitzen?

```sql
CREATE TABLE trainers (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    trainer_name VARCHAR(50) NOT NULL,
    age INT UNSIGNED CHECK (age >= 10),
    birthplace VARCHAR(50));

CREATE TABLE pokemon_types (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    type_name VARCHAR(50) NOT NULL);

CREATE TABLE pokemon (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    trainer_id INT UNSIGNED REFERENCES trainers(id),
    pokemon_name VARCHAR(50) NOT NULL,
    level INT UNSIGNED DEFAULT 1 CHECK (level >= 1 AND level <= 100),
    primary_type INT UNSIGNED REFERENCES pokemon_types(id));
```

Wie bekommen wir eine Tabelle mit den IDs der Trainer, welche mindestens drei Pokemon besitzen?

```
CREATE TABLE trainers (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    trainer_name VARCHAR(50) NOT NULL,
    age INT UNSIGNED CHECK (age >= 10),
    birthplace VARCHAR(50));

CREATE TABLE pokemon_types (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    type_name VARCHAR(50) NOT NULL);

CREATE TABLE pokemon (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    trainer_id INT UNSIGNED REFERENCES trainers(id),
    pokemon_name VARCHAR(50) NOT NULL,
    level INT UNSIGNED DEFAULT 1 CHECK (level >= 1 AND level <= 100),
    primary_type INT UNSIGNED REFERENCES pokemon_types(id));
```

Integration in vert.x mit JDBC

Siehe Projekt "TrainerDB" im GitLab (Link im Moodle)