Autoren: Elias Cuin und Florian Strobl, 12. Klasse

Lehrer: Herr Raake

Stand: Januar 2023

Inhaltsverzeichnis

[Beschreibung des Projektes 2](#_Toc124188517)

[ER-Modell 3](#_Toc124188518)

[Relationen-Modell 3](#_Toc124188519)

[Integritätsbedingungen 4](#_Toc124188520)

[SQLite CREATE Befehle 6](#_Toc124188521)

[Beispiel Testdaten 8](#_Toc124188522)

[SQLite INSERT INTO Befehle mit Testdaten 10](#_Toc124188523)

[Codestruktur mit allen Funktionssignaturen 11](#_Toc124188524)

[Erklärungen zu den Funktionen Anzeigen und Einpflegen 12](#_Toc124188525)

[SQLite Aufgaben mit Lösungen 13](#_Toc124188526)

[Link zu einem Repl und Anhang mit Sourcecode 14](#_Toc124188527)

# Beschreibung des Projektes

Auf einem Minecraft Server sind **Serverworlds** womit sich verschiedene **Spieler** verbinden können um miteinander spielen zu können. Diesen Welten bestehen aus **Blöcken** mit denen die Spieler interagieren können sowie Tiere und weitere **Entities** die die Welt bevölkern.

(Entity, Relation, Attribut)

"Serverworlds" haben eine ID ("serverworld\_id"), um sich mit ihnen verbinden zu können und einen Namen ("name") und Logo ("icon"), damit auch wir Menschen sie unterscheiden können.

In diesen "Serverworlds" spielen ("plays") verschiedene Spieler ("Player"). Darüber hinaus sind diese bevölkert ("populatedBy") durch verschiedene Entities ("MEntities"), also z.B. Tiere, und besteht aus ("buildOf") Blöcken ("Blocks"). Diese "Chunks" speichern immer deren absolute Position ("chunk position"), um später zu wissen, wo sie sich in der Welt ("Serverworld") befinden.

Jeder "Player" hat einen Namen ("username"), ein Avatar ("skin"), sowie eine eindeutige ID ("player\_id").

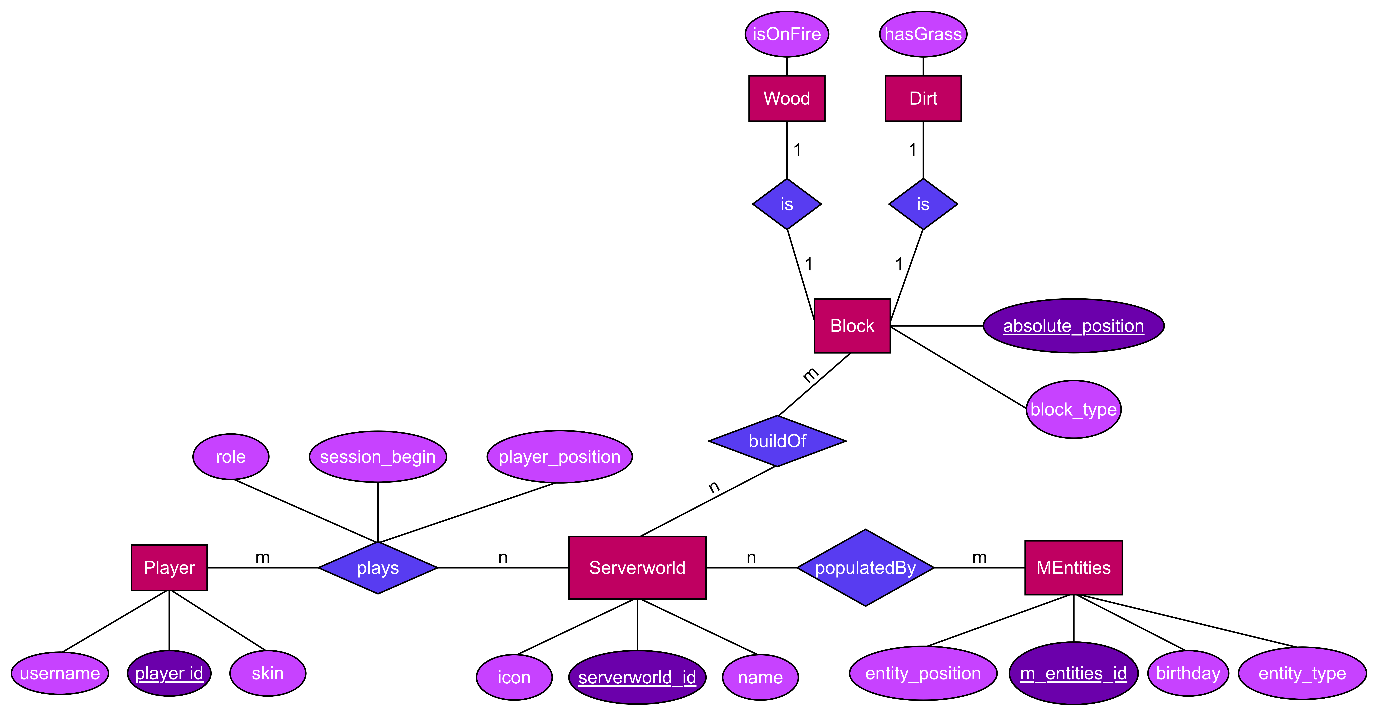
Das Spielen ("plays") in einer "Serverworld" muss die aktuelle absolute Position ("player\_position") des Spielers ("Player") speichern, wobei noch der Rang ("role") und die Uhrzeit des Spielbeginnes ("session\_begin") erfasst werden.

Die Enitites ("MEntities") der Welten haben jeweils eine eindeutige ID ("m\_entities\_id"), ein Typ ("entity\_type"), wie beispielsweise "Kuh", eine absolute Position ("entity\_position") und zuletzt dessen Erstellungszeitpunkt ("birthday") gespeichert.

Die Blöcke ("Blocks") in den "Chunks" haben ihre relative Position ("absolute\_position") gespeichert und darüber hinaus noch aus was sie bestehen ("block type"). Blöcke ("Blocks") können aus Holz ("Wood") bestehen, welche brennen können ("isOnFire") oder sie sind aus Erde ("Dirt"), welche wiederum Gras auf ihrer Oberfläche wachsen lassen können ("hasGrass").

# ER-Modell

Hier das ER-Modell zu der oben modellierten Datenbank



# Relationen-Modell

Das Relationen-Modell zu dem gerade entworfenem ER-Modell

Serverworld (serverworld\_id, name, icon)

Player (player\_id, username, skin)

MEntities (m\_entities\_id, entity\_position, birthday, entity\_type)

Block (absolute\_position, block\_type)

Wood (^absolute\_position, isOnFire)

absolute\_position verweist auf den Primärschlüssel von Block

Dirt (^absolute\_position, hasGrass)

absolute\_position verweist auf den Primärschlüssel von Block

plays (^player\_id, ^serverworld\_id, session\_begin, player\_position, role)

player\_id verweist auf den Primärschlüssel von Player

serverworld\_id verweist auf den Primärschlüssel von Serverworld

populatedBy (^m\_entities\_id, ^serverworld\_id)

m\_entities\_id verweist auf den Primärschlüssel von MEntities

serverworld\_id verweist auf den Primärschlüssel von Serverworld

buildOf (^absolute\_position, ^serverworld\_id)

absolute\_position verweist auf den Primärschlüssel von Block

serverworld\_id verweist auf den Primärschlüssel von Serverworld

# Integritätsbedingungen

Die Relationen benötigen bei ihren Attributen Integritätsbedingungen. Hier wird erklärt weshalb jedes Attribute seine gegebenen Bedingungen hat.

|  |  |  |
| --- | --- | --- |
| Serverworld |  |  |
|  |  |  |
|  |  |  |

Serverworld

serverworld\_id: bigint, unique

name: string

icon: string or null

Player

player\_id: bigint, unique

username: string

skin: string

MEntities

m\_entities\_id: bigint, unique

entity\_position: string

birthday: integer

entity\_type: integer

CONSTRAINT:

0 <= entity\_type < 10

Block

absolute\_position: string, unique

block\_type: integer

Wood

^absolute\_position: string, unique reference to Block

ON DELETE cascade

ON UPDATE cascade

isOnFire: boolean, default 0

Dirt

^absolute\_position: string, unique reference to Block

ON DELETE cascade

ON UPDATE cascade

hasGrass: boolean, default 0

plays

^player\_id: bigint, unique in combination with serverworld\_id, references Player

ON DELETE cascade

ON UPDATE cascade

^serverworld\_id: bigint, unique in combination with player\_id, references Serverworld

ON DELETE cascade

ON UPDATE cascade

session\_begin: integer

player\_position: string

role: string

populatedBy

^m\_entities\_id: bigint, unique in combination with serverworld\_id, references MEntities

ON DELETE cascade

ON UPDATE cascade

^serverworld\_id: bigint, unique in combination with m\_entities\_id, references Serverworld

ON DELETE cascade

ON UPDATE cascade

buildOf

^absolute\_position: string, unique in combination with serverworld\_id, references Block

ON DELETE cascade

ON UPDATE cascade

^serverworld\_id: bigint, unique in combination with absolute\_position, references Serverworld

ON DELETE cascade

ON UPDATE cascade

*Notes:*

“boolean” steht für einen Integer der entweder 0 oder 1 sein kann.

„cascade“ steht, wie in SQLite, für “übernehme diese Aktion“.

Ein Teil der Integritätsbedingungen werden von SQLite3 übernommen, ein anderer Teil wird durch den Python code umgesetzt.

# SQLite CREATE Befehle

Die folgenden SQLite3 Befehle wurden in Python als String gespeichert und werden mit der Funktion createTable(cursor, tableCreateStr) ausgeführt. Sie erstellen die Tabellen der Datenbank mit gegebenen Zwangsbedingungen („Constraints“).

CREATE TABLE IF NOT EXISTS Serverworld (

  serverworld\_id bigint PRIMARY KEY

    CHECK (serverworld\_id > 0),

  name TEXT NOT NULL

    CHECK (name != ""),

  icon TEXT

);

CREATE TABLE IF NOT EXISTS Player (

  player\_id bigint PRIMARY KEY

    CHECK (player\_id > 0),

  username TEXT NOT NULL

    CHECK (username != ""),

  skin TEXT NOT NULL

    CHECK (skin != "")

);

CREATE TABLE IF NOT EXISTS MEntities (

  m\_entities\_id bigint PRIMARY KEY

    CHECK (m\_entities\_id > 0),

  entity\_position TEXT NOT NULL

    CHECK (entity\_position != ""

      AND entity\_position LIKE "(%,%,%)"),

  birthday DATE NOT NULL

    CHECK (birthday > 0),

  entity\_type INTEGER NOT NULL

    CHECK (entity\_type >= 0 AND entity\_type < 10)

);

CREATE TABLE IF NOT EXISTS Block (

  absolute\_position TEXT PRIMARY KEY

    CHECK (absolute\_position != "" AND absolute\_position LIKE "(%,%,%)"),

  block\_type INTEGER NOT NULL

    CHECK (block\_type >= 0 AND block\_type < 10)

);

CREATE TABLE IF NOT EXISTS Wood (

  absolute\_position TEXT NOT NULL UNIQUE

    CHECK (absolute\_position != "" AND absolute\_position LIKE "(%,%,%)"),

  isOnFire INTEGER DEFAULT 0

    CHECK (isOnFire == 0 OR isOnFire == 1),

  PRIMARY KEY(absolute\_position),

  FOREIGN KEY(absolute\_position) REFERENCES Block(absolute\_position)

    on UPDATE cascade

    on DELETE cascade

);

CREATE TABLE IF NOT EXISTS Dirt (

  absolute\_position TEXT NOT NULL UNIQUE

    CHECK (absolute\_position != "" AND absolute\_position LIKE "(%,%,%)"),

  hasGrass INTEGER DEFAULT 0

    CHECK (hasGrass == 0 OR hasGrass == 1),

  PRIMARY KEY(absolute\_position),

  FOREIGN KEY(absolute\_position) REFERENCES Block(absolute\_position)

    on UPDATE cascade

    on DELETE cascade

);

CREATE TABLE IF NOT EXISTS plays (

  player\_id bigint NOT NULL

    CHECK (player\_id > 0),

  serverworld\_id bigint NOT NULL

    CHECK (serverworld\_id > 0),

  session\_begin DATE NOT NULL

    CHECK (session\_begin > 0),

  player\_position TEXT NOT NULL

    CHECK (player\_position != "" AND player\_position LIKE "(%,%,%)"),

  role TEXT NOT NULL

    CHECK (role != "" AND role IN ("Admin", "Moderator", "Player")),

  PRIMARY KEY(player\_id, serverworld\_id),

  FOREIGN KEY(player\_id) REFERENCES Player(player\_id)

    on UPDATE cascade

    on DELETE cascade,

  FOREIGN KEY(serverworld\_id) REFERENCES Serverworld(serverworld\_id)

  on UPDATE cascade

  on DELETE cascade

);

CREATE TABLE IF NOT EXISTS populatedBy (

  m\_entities\_id bigint NOT NULL

    CHECK (m\_entities\_id > 0),

  serverworld\_id bigint NOT NULL

    CHECK (serverworld\_id > 0),

  PRIMARY KEY(m\_entities\_id, serverworld\_id),

  FOREIGN KEY(m\_entities\_id) REFERENCES MEntities(m\_entities\_id)

    on UPDATE cascade

    on DELETE cascade,

  FOREIGN KEY(serverworld\_id) REFERENCES Serverworld(serverworld\_id)

  on UPDATE cascade

  on DELETE cascade

);

CREATE TABLE IF NOT EXISTS buildOf (

  absolute\_position TEXT NOT NULL

    CHECK (absolute\_position != "" AND absolute\_position LIKE "(%,%,%)"),

  serverworld\_id bigint NOT NULL

    CHECK (serverworld\_id > 0),

  PRIMARY KEY(absolute\_position, serverworld\_id),

  FOREIGN KEY(absolute\_position) REFERENCES Block(absolute\_position)

    on UPDATE cascade

    on DELETE cascade,

  FOREIGN KEY(serverworld\_id) REFERENCES Serverworld(serverworld\_id)

  on UPDATE cascade

  on DELETE cascade

);

Die Python Funktion, um die SQLite3 Befehle auszuführen:

# tableCreateStr: an SQLite3 CREATE TABLE code

def createTable(cursor, tableCreateStr: str) -> None:

  try:

    # try execute the create command

    cursor.execute(tableCreateStr)

  except:

    # did not execute properly, show an error and return

    Logger.error(f"Error while trying to create the table {tableCreateStr}")

    return

  # commit the changes to the database since the command executed without errors

  cursor.connection.commit()

Sie wird einmalig beim Starten der Datenbank, mithilfe einer Schleife, für alle CREATE Befehle ausgeführt.

# Beispiel Testdaten

Die Daten wurden in der GenData.py generiert und in der Datenbank anschließend als Standardwerte für jede Tabelle gespeichert.

Die Tabellen Wood und Dirt haben nur halb so viele Einträge, da sie beide Fremdschlüssel zu Block besitzen und somit nicht zwei Mal dieselben Primärschlüssel referenzieren dürfen. Sprich, die erste Hälfte der Primärschlüssel von Block sind bei Wood ein Fremdschlüssel, und die zweite Hälfte bei Dirt.

Die Tabellen plays, populatedBy und buildOf haben bei ihren Fremdschlüsseln jeweils existierende Primärschlüssel ihrer jeweiligen referenzierten Tabellen.

Serverworld

|  |  |  |
| --- | --- | --- |
| serverworld\_id | name | icon |
| 4265536020256728 | Imperial Japan | https://mc-icons/6ab4c42f87984158820dd285bba7a4e7 |
| 4335715097468013 | The Lion`s Pride | null |
| 1976886614033885 | Mongol Empire | https://mc-icons/814575a13d9342739481f25a188c8c5a |
| 2825036397637892 | Frosty Fortress | https://mc-icons/00e8486d5d3b42799c5c421ceb14ba64 |
| 1945457973125363 | Neo Tokyo | https://mc-icons/0c693bff8ace4b739e7d6405cad87ff0 |

Player

|  |  |  |
| --- | --- | --- |
| player\_id | Username | skin |
| 469244117836675 | Terminator | https://mc-skin/eccc8472b01a47e095d54c78eaf15e0d |
| 2010747666924997 | Shooter | https://mc-skin/29a7cbe4a5ba44638f8935f9f01d53f2 |
| 567010368764946 | Sick Rebellious Names | https://mc-skin/dbf7f8a5be5448ed885f1024b3b2a273 |
| 2084141176415482 | Dropkick | https://mc-skin/72c2787f15594776b95a6ae0cc9d8712 |
| 923845784918491 | Bowler | https://mc-skin/142c396991ba43e087c6eb0eb1432771 |

MEntities

|  |  |  |  |
| --- | --- | --- | --- |
| m\_entities\_id | entity\_position | Birthday | entity\_type |
| 1249116255128559 | (5784561, 22617947, 4422489) | 1680627754 | 3 |
| 1477335842422153 | (4513773, 3364412, 6775063) | 1670022822 | 5 |
| 309176951353911 | (22950435, 7109287, 17234290) | 1679859647 | 3 |
| 3618547799519888 | (3763394, 8301312, 13256823) | 1666251670 | 2 |
| 1072600461443545 | (25415578, 9483589, 29479241) | 1672899056 | 7 |

Block

|  |  |
| --- | --- |
| absolute\_position | block\_type |
| (21730538, 7257256, 12166861) | 6 |
| (29681001, 22411625, 5323664) | 5 |
| (5643271, 7661161, 21081451) | 2 |
| (2213165, 20874322, 16722080) | 1 |
| (6771785, 2416044, 20519029) | 7 |

Wood

|  |  |
| --- | --- |
| absolute\_position | isOnFire |
| (5643271, 7661161, 21081451) | 1 |
| (2213165, 20874322, 16722080) | 0 |

Dirt

|  |  |
| --- | --- |
| absolute\_position | hasGrass |
| (29681001, 22411625, 5323664) | 0 |
| (21730538, 7257256, 12166861) | 0 |

plays

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| player\_id | serverworld\_id | session\_begin | player\_position | role |
| 2084141176415482 | 2825036397637892 | 1666602795 | (5958751, 19326618, 13565303) | Moderator |
| 923845784918491 | 4335715097468013 | 1674296095 | (20308392, 7050927, 1940677) | Player |
| 469244117836675 | 4265536020256728 | 1682641393 | (15059814, 13646139, 28813591) | Admin |
| 469244117836675 | 4335715097468013 | 1676081651 | (28917752, 21778988, 19790312) | Moderator |
| 2010747666924997 | 4335715097468013 | 1677866115 | (16220625, 1713882, 26215383) | Admin |

populatedBy

|  |  |
| --- | --- |
| m\_entities\_id | serverworld\_id |
| 1249116255128559 | 4335715097468013 |
| 1249116255128559 | 1976886614033885 |
| 3618547799519888 | 4335715097468013 |
| 1477335842422153 | 1976886614033885 |
| 1477335842422153 | 2825036397637892 |

buildOf

|  |  |
| --- | --- |
| absolute\_position | serverworld\_id |
| (5643271, 7661161, 21081451) | 1945457973125363 |
| (29681001, 22411625, 5323664) | 4265536020256728 |
| (29681001, 22411625, 5323664) | 1945457973125363 |
| (21730538, 7257256, 12166861) | 4265536020256728 |
| (5643271, 7661161, 21081451) | 4265536020256728 |

# SQLite INSERT INTO Befehle mit Testdaten

Um generierte Daten oder vom User eingegebene Daten speichern zu können, werden die INSERT INTO Befehle von SQLite benötigt. Diese werden mit den zu speichernden Daten, in der insertIntoTable(cursor, insertIntoStr, toSaveData) Funktion, ergänzt.

INSERT INTO Serverworld (serverworld\_id, name, icon) VALUES (?, ?, ?)

INSERT INTO Player (player\_id, username, skin) VALUES (?, ?, ?)

INSERT INTO MEntities (m\_entities\_id, entity\_position, birthday, entity\_type) VALUES (?, ?, ?, ?)

INSERT INTO Block (absolute\_position, block\_type) VALUES (?, ?)

INSERT INTO Wood (absolute\_position, isOnFire) VALUES (?, ?)

INSERT INTO Dirt (absolute\_position, hasGrass) VALUES (?, ?)

INSERT INTO plays (player\_id, serverworld\_id, session\_begin, player\_position, role) VALUES (?, ?, ?, ?, ?)

INSERT INTO populatedBy (m\_entities\_id, serverworld\_id) VALUES (?, ?)

INSERT INTO buildOf (absolute\_position, serverworld\_id) VALUES (?, ?)

# insertIntoStr: an SQLite3 INSERT INTO code

# toSaveData: a list of lists of data to be inserted into the table

# -> returns True if successful in saving data, False if not

def insertIntoTable(cursor, insertIntoStr: str, toSaveData: list[list]) -> bool:

    BULK\_INSERT\_LIMIT = (

        5000  # switch between inserting data one by one and doing a bulk insert

    )

    \_data = None  # data which could not be saved for potential error message

    try:

        if len(toSaveData) < BULK\_INSERT\_LIMIT:

            # single insert for better error messages

            for data in toSaveData:

                \_data = data  # set \_data for later use

                # if the cursor.execute throws an error

                cursor.execute(insertIntoStr, data)

        else:

            # bulk insert for better performance:

            # saidly no precise error message anymore

            \_data = toSaveData

            # execute the command for each element of the array

            cursor.executemany(

                insertIntoStr,

                toSaveData,

            )

    except:

        # print an error to the user and return

        Logger.error(

            f"Error while inserting data with {insertIntoCommand} with the data: {\_data}"

        )

        return False  # return unsuccessful

    # commit change to database if it did not error

    cursor.connection.commit()

    return True  # return successful

# Codestruktur mit allen Funktionssignaturen

# Erklärungen zu den Funktionen Anzeigen und Einpflegen

# SQLite Aufgaben mit Lösungen

# Link zu einem Repl und Anhang mit Sourcecode