Autoren: Elias Cuin und Florian Strobl, 12. Klasse

Lehrer: Herr Raake

Stand: Januar 2023

Inhaltsverzeichnis

[Beschreibung des Projektes 2](#_Toc124204343)

[ER-Modell 3](#_Toc124204344)

[Relationen-Modell 3](#_Toc124204345)

[Integritätsbedingungen 4](#_Toc124204346)

[SQLite CREATE Befehle 8](#_Toc124204347)

[Beispiel Testdaten 10](#_Toc124204348)

[SQLite INSERT INTO Befehle mit Testdaten 13](#_Toc124204349)

[Erklärungen zu der Einpflegen Funktion 14](#_Toc124204350)

[Erklärung zu der Anzeige Funktion 15](#_Toc124204351)

[SQLite Aufgaben mit Lösungen 16](#_Toc124204352)

[Link zu einem Repl und Anhang mit Sourcecode 18](#_Toc124204353)

# Beschreibung des Projektes

Auf einem Minecraft Server sind **Serverworlds**. Auf diesen verbinden sich verschiedene **Spieler** um miteinander spielen zu können. Diese 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), damit Spieler sich mit genau dem Server verbinden können, auf dem sie spielen wollen. Jede Welt hat darüber hinaus auch noch einen Namen (name) und ein Logo (icon) damit auch Menschen verschiedene Server einfacher unterscheiden können.

Auf den Serverworlds spielen (plays) verschiedene **Player**

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").

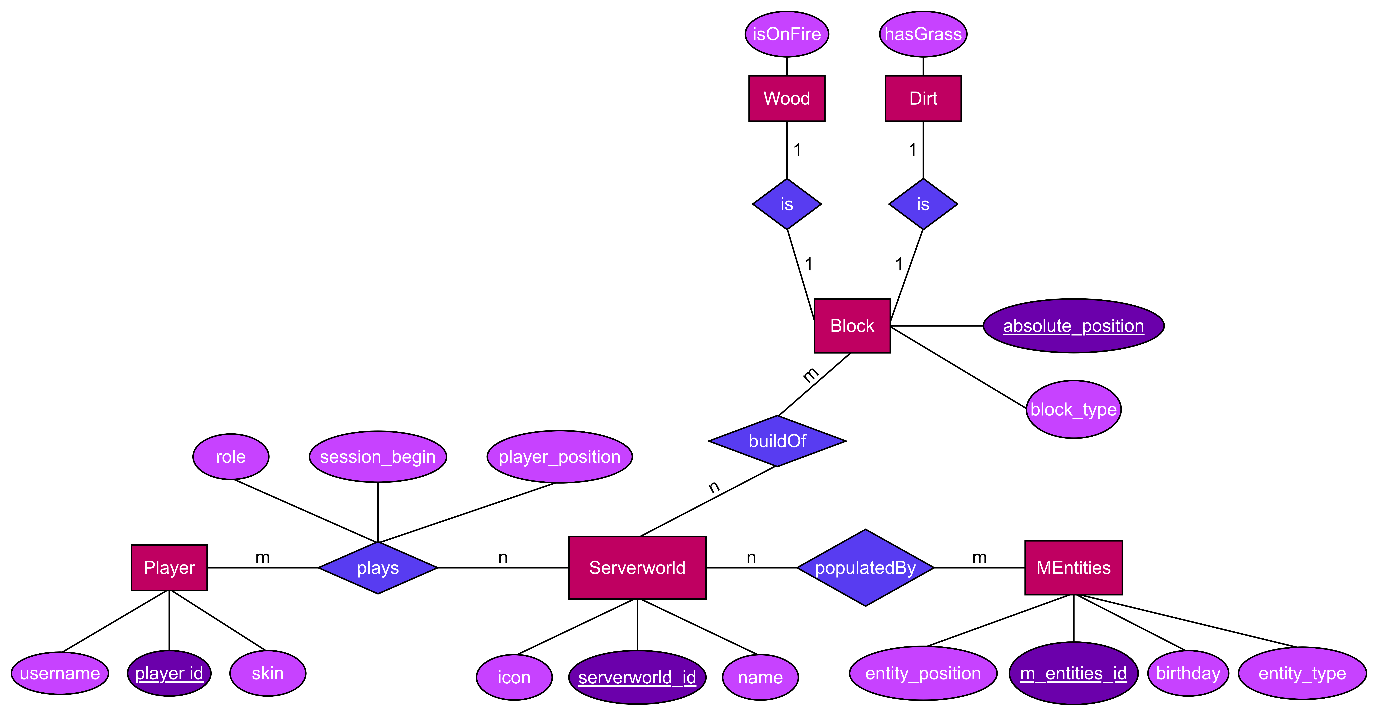
Das Spielen ("plays") in einer "Serverworld" muss die aktuelle absolute Position ("player\_position") des Spielers ("Player") speichern, wobei noch der Rang ("role") und die Uhrzeit des Spielbeginnes ("session\_begin") erfasst werden.

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

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 damit die Werte auch valide bleiben. Hier wird erklärt, weshalb jedes Attribut ihre jeweiligen Bedingungen hat.

Standardmäßig soll die Datenbank keinerlei Null Werte besitzen. Alle Fremdschlüssel sollen auch mit ON UPDATE CASCADE und ON DELETE CASCADE markiert sein, damit eine Änderung in einer Tabelle auch dieselben Änderungen in allen anderen betroffenen Tabellen durchführt.

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

Serverworld

|  |  |  |
| --- | --- | --- |
| serverworld\_id | BIGINT PRIMARY KEY | Primärschlüssel,  ist eine positive natürliche Zahl ungleich 0. |
| name | TEXT NOT NULL | CONSTRAINT: name != ““, Namen von Servern dürfen nicht leer sein damit man über ihnen sprechen kann. |
| Icon | TEXT | Icons sind wie Profilbilder und müssen deshalb nicht notwendigerweise angegeben werden. Sie können also ein leerer String sein, oder Null. |

Player

|  |  |  |
| --- | --- | --- |
| player\_id | BIGINT PRIMARY KEY | Primärschlüssel,  ist eine positive natürliche Zahl ungleich 0. |
| username | TEXT NOT NULL | CONSTRAINT: username != “”,  Usernamen von Spielern dürfen nicht leer sein damit man sie ansprechen kann. |
| skin | TEXT NOT NULL | CONSTRAINT: skin != “”,  Jeder Spieler hat ein Aussehen, weshalb der Wert nicht leer gelassen werden darf. Ein Skin wird in Form einer URL zu dem Skin-Server gespeichert, welcher die Skins speichert. |

MEntities

|  |  |  |
| --- | --- | --- |
| m\_entities\_id | BIGINT PRIMARY KEY | Primärschlüssel,  ist eine positive natürliche Zahl ungleich 0. |
| entity\_position | TEXT NOT NULL | CONSTRAINT: entity\_position LIKE “(%,%,%)”,  Die x, y und z Positionen einer jeden Entität werden in einem nicht leeren String gespeichert da SQLite keinen Support für Tuples oder Listen hat. |
| birthday | DATE NOT NULL | CONSTRAINT: birthday > 0,  eine Entität kann nicht an einem negativen Zeitpunkt erstellt worden sein (vor dem Beginn des Servers). |
| entity\_type | INTEGER NOT NULL | CONSTRAINT: entity\_type >= 0 AND entity\_type < 10,  In dem Spiel gibt es 10 verschiedene Entitätstypen, die als Enum in entity\_type gespeichert werden. |

Block

|  |  |  |
| --- | --- | --- |
| absolute\_position | TEXT PRIMARY KEY | Primärschlüssel,  CONSTRAINT: absolute\_position LIKE "(%,%,%)",  Die x, y und z Position jedes Blockes wird in einem String in dem Format „(x, y, z)“ gespeichert. |
| block\_type | INTEGER NOT NULL | CONSTRAINT: block\_type >= 0 AND block\_type < 10,  In dem Spiel gibt es 10 verschiedene Blocktypen, die als Enum in block\_type gespeichert werden. |

Wood

|  |  |  |  |
| --- | --- | --- | --- |
| absolute\_position | STRING PRIMARY KEY | FOREIGN KEY REFERENCES Block | Primärschlüssel,  Selbe Constraints wie bei Block. |
| isOnFire | INTEGER NOT NULL | - | CONSTRAINT: isOnFire == 0 OR isOnFire == 1, DEFAULT 0,  Holz kann Brennen und da SQLite keine Booleans unterstützt, behelfen wir uns mit einer natürlichen Zahl die zwischen 0 und 1 ist. 0 bedeutet das Holz brennt nicht, 1 bedeutet es brennt. Standardmäßig brennt Holz nicht. |

Dirt

|  |  |  |  |
| --- | --- | --- | --- |
| absolute\_position | STRING PRIMARY KEY | FOREIGN KEY REFERENCES Block | Primärschlüssel,  Selbe Constraints wie bei Block. |
| hasGrass | INTEGER NOT NULL | - | CONSTRAINT: hasGrass == 0 OR hasGrass == 1,  DEFAULT: 0,  Erde kann Grass haben oder auch nicht und da SQLite keine Booleans unterstützt, behelfen wir uns mit einer natürlichen Zahl die zwischen 0 und 1 ist. 0 bedeutet das kein Grass auf der Erde gewachsen ist, 1 bedeutet schon. Standardmäßig hat Erde kein Grass. |

plays

|  |  |  |  |
| --- | --- | --- | --- |
| player\_id | BIGINT PRIMARY KEY | FOREIGN KEY REFERENCES Player | Primärschlüssel,  Selbe Constraints wie bei Player. |
| Serverworld\_id | BIGINT PRIMARY KEY | FOREIGN KEY REFERENCES Serverworld | Primärschlüssel,  Selbe Constraints wie bei Serverworld. |
| session\_begin | DATE NOT NULL | - | CONSTRAINT: session\_begin > 0,  ein Spieler kann sich nicht vor dem erstellen der Welt, in dieser spielen, weshalb man nicht einem negativen Zeitpunkt als Anfang speichern kann. |
| player\_position | TEXT NOT NULL | - | CONSTRAINT: player\_position LIKE “(%,%,%)”,  die x, y und z Koordinaten eines jeden Spielers werden in einem String in dem Format „(x, y, z)“ gespeichert. |
| role | TEXT NOT NULL | - | CONSTRAINT: role IN ("Admin", "Moderator", "Player"),  jeder Spieler kann entweder ein normaler Spieler sein, ein Moderator der andere Spieler moderieren kann, oder ein Administrator der alle Berechtigungen auf dem Server hat. |

populatedBy

|  |  |  |  |
| --- | --- | --- | --- |
| m\_entities\_id | BIGINT PRIMARY KEY | FOREIGN KEY REFERENCES MEntities | Primärschlüssel,  Selbe Constraints wie bei MEntities. |
| serverworld\_id | BIGINT PRIMARY KEY | FOREIGN KEY REFERENCES Serverworld | Primärschlüssel,  Selbe Constraints wie bei Serverworld. |

buildOf

|  |  |  |  |
| --- | --- | --- | --- |
| absolute\_position | TEXT PRIMARY KEY | FOREIGN KEY REFERENCES Block | Primärschlüssel,  Selbe Constraints wie bei Block. |
| serverworld\_id | BIGINT PRIMARY KEY | FOREIGN KEY REFERENCES Serverworld | Primärschlüssel,  Selbe Constraints wie bei Serverworld. |

# 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 den 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 folgenden Daten wurden mithilfe der GenData.py Datei 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 denselben 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 zu ihren 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 und dann ausgeführt.

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 (?, ?)

Die Python Funktion

# 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

# Erklärungen zu der Einpflegen Funktion

Beim einpflegen werden sämtliche Tkinter Inputfelder des GUI durchlaufen (mithilfe der tableDataBodyWidgets Variable) und dann mit den, vom User eingegebenen, erhaltenen Werten mithilfe der hier zuvor definierten insertIntoTable() Methode in die Datenbank speichern.

# Einpflegen

import sqlite3

# Given variables:

# - tableDataBodyWidgets -> Contains all input fields as widgets in a table 2d array

# - insertIntoTable: Function that inserts data into a table

cursor: sqlite3.Connection.cursor = sqlite3.connect("minecraftDatabase.db").cursor()

def einpflegen(tableName: str, columnNames: str) -> None:

    # Function that gets the data from the inout fields

    def getTablesInputs():

        result = []

        # Every row is gone through

        for widgetRow in tableDataBodyWidgets:

            subresult = []

            # It will go through each element except for the last one and ui subresult appended

# -> because the last one is the trash can object

            for widget in range(len(widgetRow) - 1):

                subresult.append(widgetRow[widget].get())

            # This row input is appended to the result array

            result.append(subresult)

        return result

    # Example for maintenance in Serverworld

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

    insertIntoTable(cursor, string, getTablesInputs())

# Erklärung zu der Anzeige Funktion

Um Daten aus der Datenbank anzuzeigen, muss zunächst eine SELECT Anfrage abgefragt werden und die Spaltennamen der Antwort müssen ermittelt werden. Es gibt zwei Funktionen, um die Daten zu visualisieren: "anzeigen\_body()" und "anzeigen\_columns()". Die Funktion "anzeigen\_body()" durchläuft die Variable "tableBody" und für jede Iteration wird ein Eingabefeld auf dem Viewport generiert und der entsprechende Dateninhalt wird in diesem Eingabefeld angezeigt. Die Funktion "anzeigen\_columns()" geht jede Spalte (die als Parameter übergeben wurde) durch und zeigt sie in Form eines Labels über den Eingabefeldern im Grid an.

import customtkinter

import sqlite3

# Variables given:

# - widthCurrentFrame

# - WidthforEntry

# - WidthforColumn

# - scrollFrame.viewPort -> is like "Frame" but with scrollBar

# - onRemove -> Delete a row

# - clearTableDataHeaderWidgets() -> Everything is cleared

# - x - X coordinate of the table frame

def anzeigen\_columns(columns):

    # The old columns are cleared

    clearTableDataHeaderWidgets()

    # Each element of the given array is iterated and a label is added

    # created above the table for the corresponding columns

    startX = x + 5

    for i in range(len(arr)):

        subWidth = widthforColumn

        myLabel1 = customtkinter.CTkLabel(

            master=app,

            text=columns[i],

            font=("Helvetica", 12, "bold"),

            fg\_color="#2b2b2b",

            anchor="w",

        )

        myLabel1.place(x=startX, y=y, width=subWidth)

        tableDataHeaderWidgets.append(myLabel1)

        startX += subWidth

def anzeigen\_body(tableBody):

    # Creation of the cursor

    for row in range(len(tableBody)):

        # Each column of the row is traversed

        lastCol = 0

        for col in range(len(row)):

            lastCol = col

            # Create an Entry with sharp corners,

            # place it on the ViewFrame and give it the width widthForEntry

            myEntry = customtkinter.CTkEntry(

                scrollFrame.viewPort, corner\_radius=0, width=widthForEntry

            )

            # Place the entry on a grid on the view port

            myEntry.grid(row=row, column=col)

            # try to place the corresponding text from tableBody in the entry

            # If this doesn't work, put "Data not found" in the input field

            try:

                myEntry.insert(0, tableBody[row][col])

            except:

                myEntry.insert(0, "Data not found")

        # When all columns for the unput fields from the row have been created,

        # a delete button is also created

        # who is responsible for deleting the respective row

        deleteButton = customtkinter.CTkButton(

            scrollFrame.viewPort,

            text="🗑",

            command=lambda row=row: self.onRemove(row),

            corner\_radius=0,

            width=self.actionColumnWidth,

        )

        # Set the delete button one column further than

        # the last input field placed on the row

        deleteButton.grid(row=row, column=lastCol + 1)

# Here, data is fetched from the Block table, for example

cursor: sqlite3.Connection.cursor = sqlite3.connect("minecraftDatabase.db").cursor()

cursor.execute("SELECT \* FROM Block")

resultData = cursor.fetchall()

anzeigen\_body(resultData)

columnNames = list(map(lambda x: x[0], cursor.description))

anzeigen\_columns(columnNames)

# SQLite Aufgaben mit Lösungen

Aufgabe 1)

|  |
| --- |
| Aufgabe: Wie lauten die IDs der Spieler mit dem Namen „Terminator“? |
| Antwort bei den Standardwerten: 469244117836675 |
| Lösung SQLite: SELECT player\_id FROM player WHERE username == "Terminator" |
| Lösung Python: username == "Terminator" && player\_id |

Aufgabe 2)

|  |
| --- |
| Aufgabe: Wie viele Blöcke gibt es? |
| Antwort bei den Standardwerten: 5 |
| Lösung SQLite: SELECT count(\*) AS Anzahl FROM Block |
| Lösung Python: - |

Aufgabe 3)

|  |
| --- |
| Aufgabe: Wie lauten die Koordinaten und der Type der ältesten Entität? |
| Antwort bei den Standardwerten:  entity\_position entity\_type  “(5784561, 22617947, 4422489)” 3 |
| Lösung SQLite: SELECT position, entity\_type from MEntities ORDER BY birthday DESC LIMIT 1 |
| Lösung Python: birthday2 - birthday1 && entity\_position, entity\_type && slice 0; 1 |

Aufgabe 4)

|  |
| --- |
| Aufgabe: Wie viele Spieler sind auf Welten, bei denen die Welt mindestens einen Spieler hat? |
| Antwort bei den Standardwerten:  Serverworld\_id Spieleranzahl  2825036397637892 1  4265536020256728 1  4335715097468013 3 |
| Lösung SQLite: SELECT serverworld\_id, count(player\_id) AS Spieleranzahl FROM plays GROUP BY serverworld\_id |
| Lösung Python: - |

Aufgabe 5)

|  |
| --- |
| Aufgabe: Berechnen sie die Differenz zwischen dem Zeitpunkt des ersten Spieler joins und dem letzten Spieler joins der Plays Tabelle. |
| Antwort bei den Standardwerten: 16038598 |
| Lösung SQLite: SELECT (SELECT session\_begin FROM plays ORDER BY session\_begin DESC LIMIT 1) - (SELECT session\_begin FROM plays ORDER BY session\_begin ASC LIMIT 1) |
| Lösung Python: - |

Aufgabe 6)

|  |
| --- |
| Aufgabe: Auf welcher Serverwelt lebt die Entität „1249116255128559“? |
| Antwort bei den Standardwerten: Frosty Fortress |
| Lösung SQLite: SELECT Serverworld.name FROM Serverworld JOIN populatedBy ON populatedBy.serverworld\_id == Serverworld.serverworld\_id JOIN MEntities ON MEntities.m\_entities\_id == populatedBy.m\_entities\_id WHERE Serverworld.serverworld\_id == 2825036397637892 |
| Lösung Python: - |

# Link zu einem Repl und Anhang mit Sourcecode

Einladungslink zu dem Replit: <https://replit.com/join/blhbwelbgi-florianstrobl>

(Der „Run“ Button Funktioniert, sowie die meisten Features)

Der Sourcecode ist im Anhang in der Zip-Datei.

Der Entrypoint des Programmes ist in *Backend/main.py*

Tipp: copy-pasten Sie nicht unbedingt Code in die Software hinein, da Word meistens unsichtbare extra Zeichen mitspeichert, die die Software nicht richtig versteht.