Minecraft Datenbank - Ein Python und SQLite Projekt

Autoren: Elias Cuin und Florian Strobl, 12. Klasse
Lehrer: Herr Raake
Stand: Januar 2023

Inhaltsverzeichnis

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").

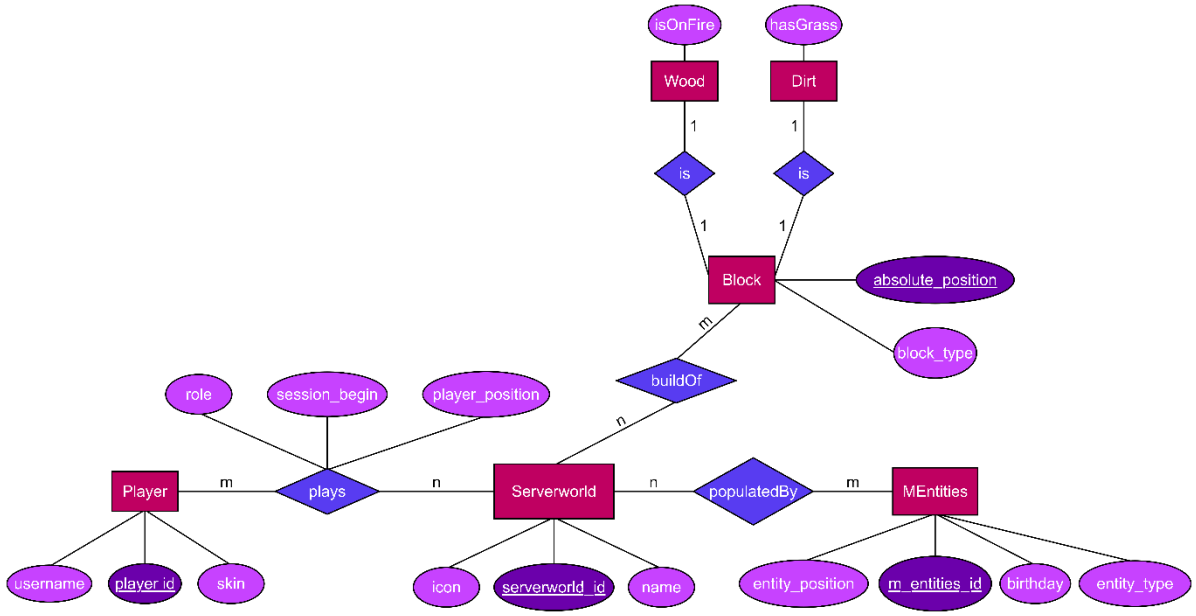
Das Spielen ("plays") in einer "Serverworld" muss die aktuelle absolute Position ("player_position") des Spielers ("Player") speichern, wobei noch der Rang ("role") und die Uhrzeit des Spielbeginnes ("session_begin") erfasst werden.

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 (?, ?)
```

```python
# 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