

Geodatenanalyse 2

Termin 8 - Modul 1

Datenbanken und *Structured Query Language* (SQL)

Ca. 20-30 Minuten

Inhalt

- Was ist eine Datenbank?
- Verwendung von Datenbanken
- *Structured Query Language* (SQL)
- SQLite in Python
- Übersicht über die Arten von Abfragen

SQLite3 Installation

Für dieses Modul brauchen wir [SQLite3](#). Zur Installation bitte:

1 - *Anaconda Prompt* öffnen und folgendes eingeben:

2 - `conda activate geo` ENTER

3 - `conda install -c blaze sqlite3` ENTER

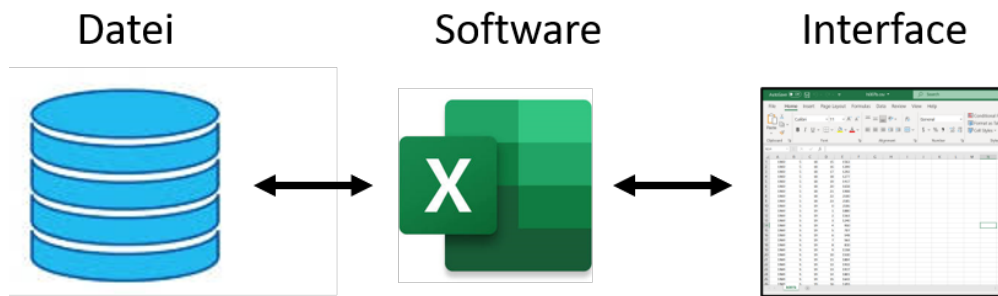
```
In [1]: import sqlite3
```

Was ist eine Datenbank?

- Eine Datenbank enthält elektronisch gespeicherte Daten
- Diese können **strukturiert** und **unstrukturiert** sein
- Zugriff auf diese Daten erfolgt immer über ein Interface

Beispiel Microsoft Excel

Eine *xlsx*-Datei enthält Daten und die Excel-Software erlaubt Zugriff und Manipulation



Wichtige Fragen ...

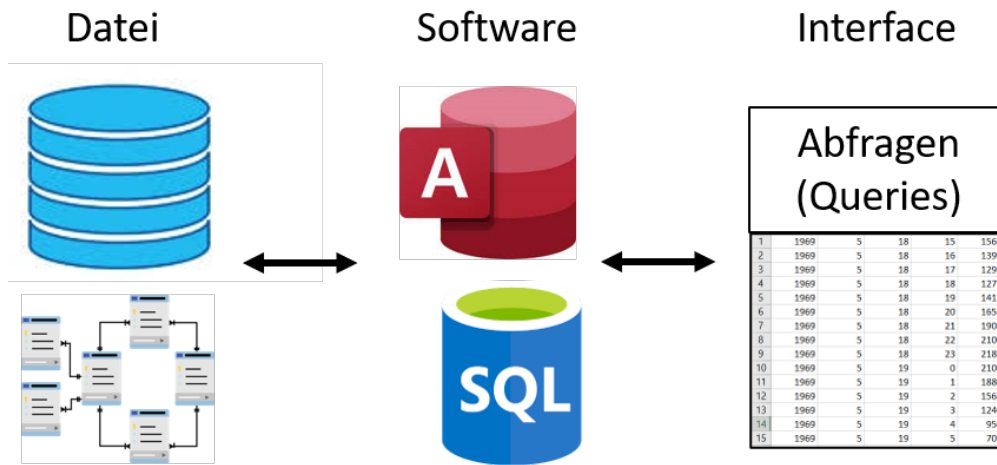
- Was passiert, wenn Daten zu groß werden?
- Was ist, wenn man mehrere Tabellen hat, welche zusammen gehören?
- Wie geht man mit mehrdimensionalen Daten um?
- Was ist, wenn mehrere Personen gleichzeitig darauf zugreifen?

Verwendung von Datenbanken

- Eine Datenbank ist ein elektronisches Speicher- und Verwaltungssystem für strukturierte Daten
- Durch den Einsatz von *Structured Query Language* (SQL) wird eine Unabhängigkeit der Anwendungen vom eingesetzten Datenbankmanagementsystem erzielt
- SQL ist eine Sprache zur Manipulation (Abruf, Veränderung, Hinzufügung, etc.) von Daten aus einer Datenbank

Beispiel Software Schnittstelle

Eine Binärdatei enthält Daten und eine Software fungiert als Schnittstelle für Zugriff und Manipulation



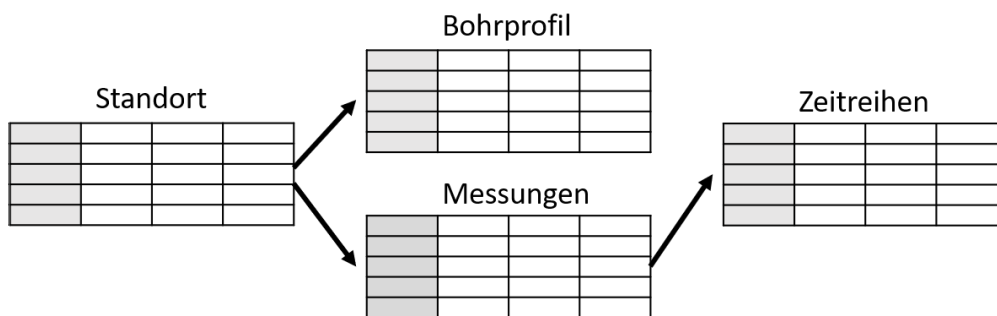
Relationale Datenbank

"Eine relationale Datenbank ist eine digitale Datenbank, die zur elektronischen Datenverwaltung in Computersystemen dient und auf einem tabellenbasierten relationalen Datenbankmodell beruht. Grundlage des Konzeptes relationaler Datenbanken ist die Relation."

Quelle: [Wikipedia](#)

Wozu braucht man das?

Für die Speicherung von **diversen Daten**, welche eine **Beziehung zueinander** haben



Übersicht über *SELECT* Abfragen

```
SELECT DISTINCT column_list
FROM table_list
    JOIN table ON join_condition
WHERE row_filter
ORDER BY column
LIMIT count OFFSET offset
GROUP BY column
HAVING group_filter;
```

Welche Tabellen gibt es?

Es gibt immer einen Phantom-Eintrag **sqlite_master**, welcher eine Liste der Tabellen enthält.

Eine Liste aller existierender Tabellen in der Datenbank:

```
In [3]: for row in cur.execute("SELECT name FROM sqlite_master WHERE type='table';"):
        print(row)

('surveys',)
('species',)
('plots',)
```

Eine detaillierte Übersicht über die Tabellenstruktur von *species*:

```
In [4]: for row in cur.execute("PRAGMA TABLE_INFO(species);"):
        print(row)

(0, 'species_id', 'TEXT', 0, None, 0)
(1, 'genus', 'TEXT', 0, None, 0)
(2, 'species', 'TEXT', 0, None, 0)
(3, 'taxa', 'TEXT', 0, None, 0)
```

Was steht in der Tabelle *species*?

Frage alle Informationen ab:

```
In [5]: # The result of a "cursor.execute" can be iterated over by row
        for row in cur.execute('SELECT * FROM species;'):
            print(row)
```

```
('AB', 'Amphispiza', 'bilineata', 'Bird')
('AH', 'Ammospermophilus', 'harrisi', 'Rodent')
('AS', 'Ammodramus', 'savannarum', 'Bird')
('BA', 'Baiomys', 'taylori', 'Rodent')
('CB', 'Campylorhynchus', 'brunneicapillus', 'Bird')
('CM', 'Calamospiza', 'melanocorys', 'Bird')
('CQ', 'Callipepla', 'squamata', 'Bird')
('CS', 'Crotalus', 'scutalatus', 'Reptile')
('CT', 'Cnemidophorus', 'tigris', 'Reptile')
('CU', 'Cnemidophorus', 'uniparens', 'Reptile')
('CV', 'Crotalus', 'viridis', 'Reptile')
('DM', 'Dipodomys', 'merriami', 'Rodent')
('DO', 'Dipodomys', 'ordii', 'Rodent')
('DS', 'Dipodomys', 'spectabilis', 'Rodent')
('DX', 'Dipodomys', 'sp.', 'Rodent')
('EO', 'Eumeces', 'obsoletus', 'Reptile')
('GS', 'Gambelia', 'silus', 'Reptile')
('NL', 'Neotoma', 'albigula', 'Rodent')
('NX', 'Neotoma', 'sp.', 'Rodent')
('OL', 'Onychomys', 'leucogaster', 'Rodent')
('OT', 'Onychomys', 'torridus', 'Rodent')
('OX', 'Onychomys', 'sp.', 'Rodent')
('PB', 'Chaetodipus', 'baileyi', 'Rodent')
('PC', 'Pipilo', 'chlorurus', 'Bird')
('PE', 'Peromyscus', 'eremicus', 'Rodent')
('PF', 'Perognathus', 'flavus', 'Rodent')
('PG', 'Poocetes', 'gramineus', 'Bird')
('PH', 'Perognathus', 'hispidus', 'Rodent')
('PI', 'Chaetodipus', 'intermedius', 'Rodent')
('PL', 'Peromyscus', 'leucopus', 'Rodent')
('PM', 'Peromyscus', 'maniculatus', 'Rodent')
('PP', 'Chaetodipus', 'penicillatus', 'Rodent')
('PU', 'Pipilo', 'fuscus', 'Bird')
('PX', 'Chaetodipus', 'sp.', 'Rodent')
('RF', 'Reithrodontomys', 'fulvescens', 'Rodent')
('RM', 'Reithrodontomys', 'megalotis', 'Rodent')
('RO', 'Reithrodontomys', 'montanus', 'Rodent')
('RX', 'Reithrodontomys', 'sp.', 'Rodent')
('SA', 'Sylvilagus', 'audubonii', 'Rabbit')
('SB', 'Spizella', 'breweri', 'Bird')
('SC', 'Sceloporus', 'clarki', 'Reptile')
('SF', 'Sigmodon', 'fulviventer', 'Rodent')
('SH', 'Sigmodon', 'hispidus', 'Rodent')
('SO', 'Sigmodon', 'ochrognathus', 'Rodent')
('SS', 'Spermophilus', 'spilosoma', 'Rodent')
('ST', 'Spermophilus', 'tereticaudus', 'Rodent')
('SU', 'Sceloporus', 'undulatus', 'Reptile')
('SX', 'Sigmodon', 'sp.', 'Rodent')
('UL', 'Lizard', 'sp.', 'Reptile')
('UP', 'Pipilo', 'sp.', 'Bird')
('UR', 'Rodent', 'sp.', 'Rodent')
('US', 'Sparrow', 'sp.', 'Bird')
('ZL', 'Zonotrichia', 'leucophrys', 'Bird')
('ZM', 'Zenaida', 'macroura', 'Bird')
```

Frage selektiv Informationen ab:

```
In [6]: # The result of a "cursor.execute" can be iterated over by row
for row in cur.execute('SELECT * FROM species WHERE taxa="Bird" LIMIT 5;'):
    print(row)

('AB', 'Amphispiza', 'bilineata', 'Bird')
('AS', 'Ammodramus', 'savannarum', 'Bird')
('CB', 'Campylorhynchus', 'brunneicapillus', 'Bird')
('CM', 'Calamospiza', 'melanocorys', 'Bird')
('CQ', 'Callipepla', 'squamata', 'Bird')
```

Übersicht über INSERT Abfragen

```
INSERT INTO table (column1,column2 ,...)
VALUES( value1, value2 ,...);
```

Daten hinzufügen:

```
In [7]: cur.execute('INSERT INTO species (genus, species, taxa) VALUES("Crocodile", "sp.
```

```
Out[7]: <sqlite3.Cursor at 0x1832dcf99d0>
```

```
In [8]: # The result of a "cursor.execute" can be iterated over by row
for row in cur.execute('SELECT * FROM species WHERE taxa="Reptile";'):
    print(row)
```

```
('CS', 'Crotalus', 'scutalatus', 'Reptile')
('CT', 'Cnemidophorus', 'tigris', 'Reptile')
('CU', 'Cnemidophorus', 'uniparens', 'Reptile')
('CV', 'Crotalus', 'viridis', 'Reptile')
('EO', 'Eumeces', 'obsoletus', 'Reptile')
('GS', 'Gambelia', 'silus', 'Reptile')
('SC', 'Sceloporus', 'clarki', 'Reptile')
('SU', 'Sceloporus', 'undulatus', 'Reptile')
('UL', 'Lizard', 'sp.', 'Reptile')
(None, 'Crocodile', 'sp.', 'Reptile')
```

Übersicht über UPDATE Abfragen

```
UPDATE table
SET column_1 = new_value_1,
    column_2 = new_value_2
WHERE
    search_condition
ORDER column_or_expression
LIMIT row_count OFFSET offset;
```

```
In [9]: cur.execute('UPDATE species SET species_id="UL" WHERE taxa="Reptile";')
```

```
Out[9]: <sqlite3.Cursor at 0x1832dcf99d0>
```

```
In [10]: # The result of a "cursor.execute" can be iterated over by row
for row in cur.execute('SELECT * FROM species WHERE taxa="Reptile";'):
    print(row)
```

```
( 'UL', 'Crotalus', 'scutalatus', 'Reptile')
( 'UL', 'Cnemidophorus', 'tigris', 'Reptile')
( 'UL', 'Cnemidophorus', 'uniparens', 'Reptile')
( 'UL', 'Crotalus', 'viridis', 'Reptile')
( 'UL', 'Eumeces', 'obsoletus', 'Reptile')
( 'UL', 'Gambelia', 'silus', 'Reptile')
( 'UL', 'Sceloporus', 'clarki', 'Reptile')
( 'UL', 'Sceloporus', 'undulatus', 'Reptile')
( 'UL', 'Lizard', 'sp.', 'Reptile')
( 'UL', 'Crocodile', 'sp.', 'Reptile')
```

Übersicht über DELETE Abfragen

```
DELETE FROM table
WHERE search_condition;
```

```
In [11]: cur.execute('DELETE FROM species WHERE genus="Crocodile";')
```

```
Out[11]: <sqlite3.Cursor at 0x1832dcf99d0>
```

```
In [12]: # The result of a "cursor.execute" can be iterated over by row
for row in cur.execute('SELECT * FROM species WHERE taxa="Reptile";'):
    print(row)
```

```
( 'UL', 'Crotalus', 'scutalatus', 'Reptile')
( 'UL', 'Cnemidophorus', 'tigris', 'Reptile')
( 'UL', 'Cnemidophorus', 'uniparens', 'Reptile')
( 'UL', 'Crotalus', 'viridis', 'Reptile')
( 'UL', 'Eumeces', 'obsoletus', 'Reptile')
( 'UL', 'Gambelia', 'silus', 'Reptile')
( 'UL', 'Sceloporus', 'clarki', 'Reptile')
( 'UL', 'Sceloporus', 'undulatus', 'Reptile')
( 'UL', 'Lizard', 'sp.', 'Reptile')
```

Übersicht über CREATE TABLE Abfragen

```
CREATE TABLE [IF NOT EXISTS] [schema_name].table_name (
    column_1 data_type PRIMARY KEY,
    column_2 data_type NOT NULL,
    column_3 data_type DEFAULT 0,
    table_constraints
) [WITHOUT ROWID];
```

WICHTIG: Ein PRIMARY KEY ist immer sinnvoll, denn er erstellt automatisch eine eindeutige Nummer für jeden Eintrag. Damit wird eine Verwechslung von Einträgen verhindert.

```
In [13]: cur.execute('CREATE TABLE occurrence \
    (oc_id INTEGER PRIMARY KEY, \
    continent TEXT);')
```

```
Out[13]: <sqlite3.Cursor at 0x1832dcf99d0>
```



```
In [14]: for row in cur.execute("SELECT name FROM sqlite_master WHERE type='table';"):
          print(row)

('surveys',)
('species',)
('plots',)
('occurrence',)
```

Weitere Abfragen

- Die gegebenen Beispiele waren nur ein kleiner Teil der möglichen Abfragen an eine Datenbank
- Weitere Beispiele könnt ihr z.B. über [SQLITE Tutorial](#) anschauen und lernen

Der Vollständigkeit halber löschen wir die vorher erstellte Tabelle wieder:

```
In [15]: for row in cur.execute("DROP TABLE occurrence;"):
          print(row)
```

```
In [16]: for row in cur.execute("SELECT name FROM sqlite_master WHERE type='table';"):
          print(row)

('surveys',)
('species',)
('plots',)
```

Verwendung von Funktionen

SQL hat viele praktische Funktionen eingebaut:

- Mit `MIN()` oder `MAX()` fragt man Extremwerte ab
- Mit `COUNT()`, `SUM()` oder `AVG()` kann man die Ergebnisse direkt auswerten

Verbindung beenden

- Eine Verbindung muss immer beendet werden!
- Damit werden alle Abfragen permanent gespeichert

```
In [17]: con.close()
```


SQL Cheat Sheet

Achtung: SQLite hat limitierte Funktionalität!

Quelle: [SQL Tutorials](http://www.sqltutorial.org)

SQL CHEAT SHEET

<http://www.sqltutorial.org>



QUERYING DATA FROM A TABLE

SELECT c1, c2 FROM t;
Query data in columns c1, c2 from a table

SELECT * FROM t;
Query all rows and columns from a table

**SELECT c1, c2 FROM t
WHERE condition;**
Query data and filter rows with a condition

**SELECT DISTINCT c1 FROM t
WHERE condition;**
Query distinct rows from a table

**SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];**
Sort the result set in ascending or descending order

**SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;**
Skip offset of rows and return the next n rows

**SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;**
Group rows using an aggregate function

**SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;**
Filter groups using HAVING clause

QUERYING FROM MULTIPLE TABLES

**SELECT c1, c2
FROM t1
INNER JOIN t2 ON condition;**
Inner join t1 and t2

**SELECT c1, c2
FROM t1
LEFT JOIN t2 ON condition;**
Left join t1 and t2

**SELECT c1, c2
FROM t1
RIGHT JOIN t2 ON condition;**
Right join t1 and t2

**SELECT c1, c2
FROM t1
FULL OUTER JOIN t2 ON condition;**
Perform full outer join

**SELECT c1, c2
FROM t1
CROSS JOIN t2;**
Produce a Cartesian product of rows in tables

**SELECT c1, c2
FROM t1, t2;**
Another way to perform cross join

**SELECT c1, c2
FROM t1 A
INNER JOIN t2 B ON condition;**
Join t1 to itself using INNER JOIN clause

USING SQL OPERATORS

**SELECT c1, c2 FROM t1
UNION [ALL]
SELECT c1, c2 FROM t2;**
Combine rows from two queries

**SELECT c1, c2 FROM t1
INTERSECT
SELECT c1, c2 FROM t2;**
Return the intersection of two queries

**SELECT c1, c2 FROM t1
MINUS
SELECT c1, c2 FROM t2;**
Subtract a result set from another result set

**SELECT c1, c2 FROM t1
WHERE c1 [NOT] LIKE pattern;**
Query rows using pattern matching %, _

**SELECT c1, c2 FROM t
WHERE c1 [NOT] IN value_list;**
Query rows in a list

**SELECT c1, c2 FROM t
WHERE c1 BETWEEN low AND high;**
Query rows between two values

**SELECT c1, c2 FROM t
WHERE c1 IS [NOT] NULL;**
Check if values in a table is NULL or not

ENDE