Geodatenanalyse 2

Termin Big Data 4 - Modul 2

Weiterführender Umgang mit Datenbanken

Ca. 20-30 Minuten

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

Inhalt

- Erweiterte Funktionalität
- Beziehungen zwischen Tabellen
- Konvertierung in andere Datenformate

In [2]:
```
# Erstelle eine SQL Verbindung zur SQLite Datenbank her
con = sqlite3.connect("data/portal_mammals.sqlite")
cur = con.cursor()
```

Erweiterte Funktionalität

- Bei genauerer Betrachtung können die Abfragen sehr komplex ausgelegt werden
- Dabei gibt es viele Details, welche sehr gut eingestellt werden können
- Im Folgenden gibt es einen kleinen Überblick über die Feinheiten von Abfragen

LIMIT und OFFSET

- Mit Hilfe von LIMIT begrenzt man die Ergebnisse auf eine bestimmte Anzahl
- Mit Hilfe von OFFSET erhält man bestimmte Ergebnisse nach Reihenfolge
- OFFSET kann nur in Kombination mit LIMIT verwendet werden

In [3]:
```
# The result of a "cursor.execute" can be iterated over by row
for row in cur.execute('SELECT * FROM surveys LIMIT 10;'):
    print(row)
```

```
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None)
(2, 7, 16, 1977, 3, 'NL', 'M', 33.0, None)
(3, 7, 16, 1977, 2, 'DM', 'F', 37.0, None)
(4, 7, 16, 1977, 7, 'DM', 'M', 36.0, None)
(5, 7, 16, 1977, 3, 'DM', 'M', 35.0, None)
(6, 7, 16, 1977, 1, 'PF', 'M', 14.0, None)
(7, 7, 16, 1977, 2, 'PE', 'F', None, None)
(8, 7, 16, 1977, 1, 'DM', 'M', 37.0, None)
(9, 7, 16, 1977, 1, 'DM', 'F', 34.0, None)
(10, 7, 16, 1977, 6, 'PF', 'F', 20.0, None)
```

In [4]:
```
# The result of a "cursor.execute" can be iterated over by row
for row in cur.execute('SELECT * FROM surveys LIMIT 10 OFFSET 5;'):
    print(row)
```

```
(6, 7, 16, 1977, 1, 'PF', 'M', 14.0, None)
(7, 7, 16, 1977, 2, 'PE', 'F', None, None)
(8, 7, 16, 1977, 1, 'DM', 'M', 37.0, None)
(9, 7, 16, 1977, 1, 'DM', 'F', 34.0, None)
(10, 7, 16, 1977, 6, 'PF', 'F', 20.0, None)
(11, 7, 16, 1977, 5, 'DS', 'F', 53.0, None)
(12, 7, 16, 1977, 7, 'DM', 'M', 38.0, None)
(13, 7, 16, 1977, 3, 'DM', 'M', 35.0, None)
(14, 7, 16, 1977, 8, 'DM', None, None, None)
(15, 7, 16, 1977, 6, 'DM', 'F', 36.0, None)
```

ORDER BY

- Mit der ORDER BY Klausel können die Ergebnisse sortiert werden
- Dies kann aufwärts (ASC für 'ascending') oder abwärts (DESC für 'descending') erfolgen
- Zur Sortierung können auch mehrere Felder hintereinander verwendet werden (Engl. *multisort*)

In [5]:
```
for row in cur.execute("PRAGMA TABLE_INFO(surveys);"):
    print(row)
```

```
(0, 'record_id', 'BIGINT', 0, None, 0)
(1, 'month', 'BIGINT', 0, None, 0)
(2, 'day', 'BIGINT', 0, None, 0)
(3, 'year', 'BIGINT', 0, None, 0)
(4, 'plot_id', 'BIGINT', 0, None, 0)
(5, 'species_id', 'TEXT', 0, None, 0)
(6, 'sex', 'TEXT', 0, None, 0)
(7, 'hindfoot_length', 'FLOAT', 0, None, 0)
(8, 'weight', 'FLOAT', 0, None, 0)
```

In [6]:
```
# Ergebnisse nach Gewicht sortiert
for row in cur.execute('SELECT * FROM surveys ORDER BY weight DESC LIMIT 20;'):
    print(row)
```

```
(33049, 11, 17, 2001, 12, 'NL', 'M', 33.0, 280.0)
(12871, 5, 28, 1987, 2, 'NL', 'M', 32.0, 278.0)
(15459, 1, 11, 1989, 9, 'NL', 'M', 36.0, 275.0)
(2133, 10, 25, 1979, 2, 'NL', 'F', 33.0, 274.0)
(12729, 4, 26, 1987, 2, 'NL', 'M', 32.0, 270.0)
(13114, 7, 26, 1987, 2, 'NL', 'M', None, 269.0)
(30175, 1, 8, 2000, 2, 'NL', 'M', 34.0, 265.0)
(4962, 11, 22, 1981, 12, 'NL', 'F', None, 264.0)
(12602, 4, 6, 1987, 2, 'NL', 'M', 34.0, 260.0)
(13025, 7, 1, 1987, 2, 'NL', 'M', 33.0, 260.0)
(8869, 2, 5, 1984, 15, 'NL', 'M', 33.0, 259.0)
(12458, 3, 2, 1987, 2, 'NL', 'M', 33.0, 259.0)
(8427, 10, 15, 1983, 18, 'NL', 'M', 34.0, 256.0)
(12299, 2, 1, 1987, 2, 'NL', 'M', 32.0, 253.0)
(5846, 5, 21, 1982, 12, 'NL', 'M', 34.0, 252.0)
(31862, 3, 24, 2001, 2, 'NL', 'F', 32.0, 252.0)
(4373, 5, 3, 1981, 24, 'NL', 'F', 33.0, 251.0)
(9031, 4, 10, 1984, 3, 'NL', 'M', 34.0, 250.0)
(15686, 3, 12, 1989, 12, 'NL', 'M', 34.0, 249.0)
(5400, 2, 23, 1982, 5, 'NL', 'F', 32.0, 248.0)
```

GROUP BY

- Hiermit können identische Daten mit Hilfe einiger Funktionen in Gruppen geordnet werden
- Wenn eine bestimmte Spalte gleiche Werte in verschiedenen Zeilen hat, werden diese Zeilen in einer Gruppe angeordnet

In [7]:
```python
# Anzahl der 'surveys' pro Jahr
for row in cur.execute('SELECT year, COUNT(year) FROM surveys GROUP BY year;'):
    print(row)
```

```
(1977, 503)
(1978, 1048)
(1979, 719)
(1980, 1415)
(1981, 1472)
(1982, 1978)
(1983, 1673)
(1984, 981)
(1985, 1438)
(1986, 942)
(1987, 1671)
(1988, 1469)
(1989, 1569)
(1990, 1311)
(1991, 1347)
(1992, 1038)
(1993, 750)
(1994, 668)
(1995, 1222)
(1996, 1706)
(1997, 2493)
(1998, 1610)
(1999, 1135)
(2000, 1552)
(2001, 1610)
(2002, 2229)
```

Beziehungen zwischen Tabellen

- Tabellen können untereinander eine Beziehung haben (Engl. *relational database*)
- Diese wird meistens über eine einzigartige Bezeichnung (Engl. *unique identifier*) gekennzeichnet
- Abfragen können tabellenübergreifend gemacht werden, wobei die Beziehung berücksichtigt werden kann

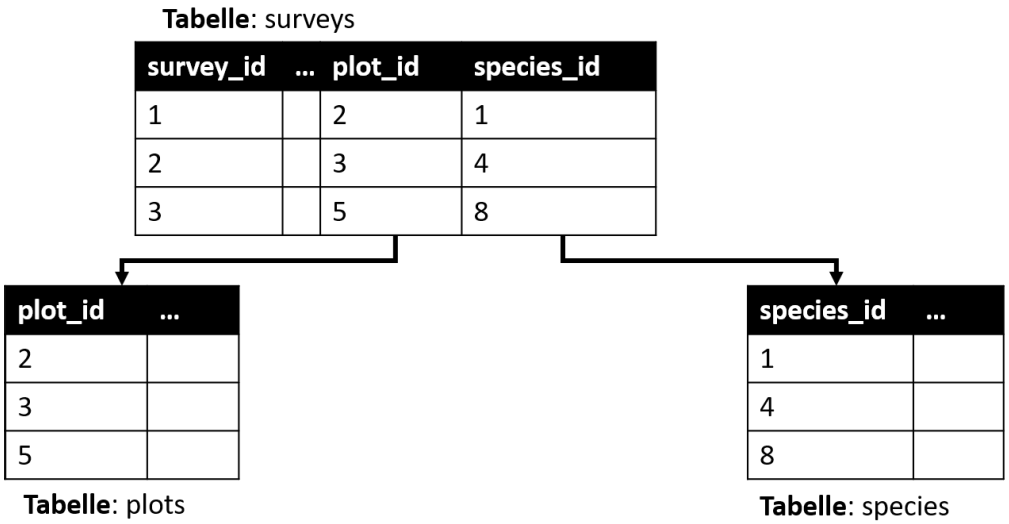
In [8]:
```
for row in cur.execute("SELECT name FROM sqlite_master WHERE type='table';"):
    print(row)
```

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

In [9]:
```
for row in cur.execute("PRAGMA TABLE_INFO(surveys);"):
    print(row)
```

```
(0, 'record_id', 'BIGINT', 0, None, 0)
(1, 'month', 'BIGINT', 0, None, 0)
(2, 'day', 'BIGINT', 0, None, 0)
(3, 'year', 'BIGINT', 0, None, 0)
(4, 'plot_id', 'BIGINT', 0, None, 0)
(5, 'species_id', 'TEXT', 0, None, 0)
(6, 'sex', 'TEXT', 0, None, 0)
(7, 'hindfoot_length', 'FLOAT', 0, None, 0)
(8, 'weight', 'FLOAT', 0, None, 0)
```

Zusammenfassung: Übersicht über die Datenbank *data/portal_mammals.sqlite*:



CROSS JOIN

- Gleicht jede Zeile der ersten Tabelle mit jeder Zeile der zweiten Tabelle ab
- Wenn die Eingabetabellen jeweils x und y Zeilen haben, hat die Ergebnistabelle x*y Zeilen
- Sollte vorsichtig verwendet werden, denn es erzeugt extrem große Tabellen

ACHTUNG: Desweiteren wird hier immer ein Limit verwendet zwecks besserer Übersicht

In [10]:
```
# Beispiel für einen CROSS JOIN
for row in cur.execute('SELECT * FROM surveys CROSS JOIN plots LIMIT 10;'):
    print(row)
```

```
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 1, 'Spectab exclosure')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 2, 'Control')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 3, 'Long-term Krat Exclosure')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 4, 'Control')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 5, 'Rodent Exclosure')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 6, 'Short-term Krat Exclosure')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 7, 'Rodent Exclosure')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 8, 'Control')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 9, 'Spectab exclosure')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 10, 'Rodent Exclosure')
```

INNER JOIN

- Erzeugt eine neue Ergebnistabelle durch die Kombination von Spaltenwerten zweier Tabellen (**table1** und **table2**) basierend auf dem Join-Prädikat
- Die Abfrage vergleicht jede Zeile von **table1** mit jeder Zeile von **table2**, um alle Zeilenpaare zu finden, die das Join-Prädikat erfüllen
- Wenn das Join-Prädikat erfüllt ist, werden die Spaltenwerte für jedes übereinstimmende Paar von Zeilen von A und B in einer Ergebniszeile kombiniert

In [11]:
```
# Beispiel für einen INNER JOIN
for row in cur.execute('SELECT * FROM surveys INNER JOIN plots ON surveys.plot_i
    print(row)
```

```
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 2, 'Control')
(2, 7, 16, 1977, 3, 'NL', 'M', 33.0, None, 3, 'Long-term Krat Exclosure')
(3, 7, 16, 1977, 2, 'DM', 'F', 37.0, None, 2, 'Control')
(4, 7, 16, 1977, 7, 'DM', 'M', 36.0, None, 7, 'Rodent Exclosure')
(5, 7, 16, 1977, 3, 'DM', 'M', 35.0, None, 3, 'Long-term Krat Exclosure')
(6, 7, 16, 1977, 1, 'PF', 'M', 14.0, None, 1, 'Spectab exclosure')
(7, 7, 16, 1977, 2, 'PE', 'F', None, None, 2, 'Control')
(8, 7, 16, 1977, 1, 'DM', 'M', 37.0, None, 1, 'Spectab exclosure')
(9, 7, 16, 1977, 1, 'DM', 'F', 34.0, None, 1, 'Spectab exclosure')
(10, 7, 16, 1977, 6, 'PF', 'F', 20.0, None, 6, 'Short-term Krat Exclosure')
```

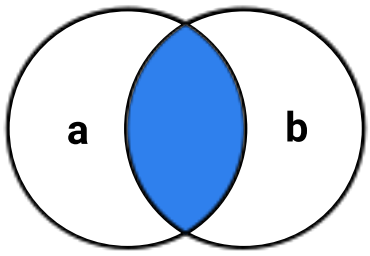
LEFT JOIN

- Die Abfrage vergleicht jede Zeile von table1 mit jeder Zeile von table2, um alle Zeilenpaare zu finden, die das Join-Prädikat erfüllen
- Wenn das Join-Prädikat erfüllt ist, werden die Spaltenwerte für jedes Paar von Zeilen von A welche einen übereinstimmenden Wert in B hat, in einer Ergebniszeile kombiniert

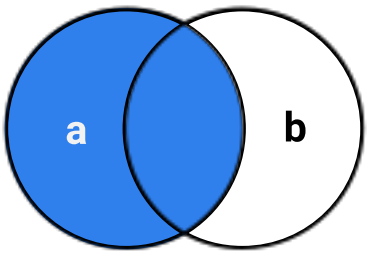
In [12]:
```
# Beispiel für ein LEFT JOIN
for row in cur.execute('SELECT * FROM surveys LEFT JOIN species LIMIT 10;'):
    print(row)
```

```
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 'AB', 'Amphispiza', 'bilineata', 'Bird
')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 'AH', 'Ammospermophilus', 'harrisi', '
Rodent')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 'AS', 'Ammodramus', 'savannarum', 'Bir
d')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 'BA', 'Baiomys', 'taylori', 'Rodent')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 'CB', 'Campylorhynchus', 'brunneicapil
lus', 'Bird')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 'CM', 'Calamospiza', 'melanocorys', 'B
ird')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 'CQ', 'Callipepla', 'squamata', 'Bird
')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 'CS', 'Crotalus', 'scutalatus', 'Repti
le')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 'CT', 'Cnemidophorus', 'tigris', 'Rept
ile')
(1, 7, 16, 1977, 2, 'NL', 'M', 32.0, None, 'CU', 'Cnemidophorus', 'uniparens', 'R
eptile')
```

Überblick über Verbindungstypen (abgeändert von Dataquest)



`a INNER JOIN b`



`a LEFT JOIN b`

Konvertierung in andere Datenformate

- Oftmals besteht die Aufgabe darin, bestimmte Daten aus einer Datenbank zu extrahieren
- Dafür muss eine passende Abfrage (Engl. *query*) gemacht werden
- Das Ergebnis kann dann als Tabelle gespeichert werden, z.B. im *csv*-Format

Pandas kann viele Datenformate lesen

Auszug aus dem Pandas Handbuch

Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf

Datenexport von SQL mit Hilfe von *Pandas*

In [13]:
```
import pandas as pd
```

In [14]:
```
# eine Verbindung erstellen
dbcon = sqlite3.connect('data/portal_mammals.sqlite')

# Daten abfragen ...
df = pd.read_sql_query("SELECT * FROM surveys LEFT JOIN species ON surveys.speci

# Daten als CSV speichern
df.to_csv('data/sqlite_export.csv')

df
```

Out[14]:

	record_id	month	day	year	plot_id	species_id	sex	hindfoot_length	weigh
0	1	7	16	1977	2	NL	M	32.0	NaN
1	2	7	16	1977	3	NL	M	33.0	NaN
2	3	7	16	1977	2	DM	F	37.0	NaN
3	4	7	16	1977	7	DM	M	36.0	NaN
4	5	7	16	1977	3	DM	M	35.0	NaN
...	...	...	...	...	...	...	...	...	.
35544	35545	12	31	2002	15	AH	None	NaN	NaN
35545	35546	12	31	2002	15	AH	None	NaN	NaN
35546	35547	12	31	2002	10	RM	F	15.0	14.(
35547	35548	12	31	2002	7	DO	M	36.0	51.(
35548	35549	12	31	2002	5	None	None	NaN	NaN

35549 rows × 13 columns

Überblick über andere SQL-Systeme

- **MySQL** ist ein relationales Datenbankmanagementsystem, das auf SQL basiert. Die Anwendung wird für eine Vielzahl von Zwecken eingesetzt, darunter Data Warehousing, E-Commerce und Protokollierungsanwendungen. Die häufigste Verwendung für mySQL ist jedoch der Zweck einer Web-Datenbank.
- **PostgreSQL** ist ein objektrelationales Datenbankmanagementsystem (ORDBMS), während MySQL ein Community-getriebenes DBMS-System ist. PostgreSQL unterstützt moderne Anwendungsfunktionen wie JSON, XML usw., während MySQL nur JSON unterstützt.
- **Microsoft SQL** ist ein relationales Datenbankmanagementsystem (RDBMS), das eine Vielzahl von Transaktionsverarbeitungs-, Business Intelligence- und Analyseanwendungen in IT-Umgebungen von Unternehmen unterstützt.

ENDE