Leibniz Universität
IT Services

Leibniz
Universität
Hannover

SQLite – Nutzung in Python
Verknüpfung mit einer Datenbank

SQLite

- Datenbank, basierend auf Dateien mit der Endung „.sql“ oder „.db“.
- Programmbibliothek, die im Standard der Programmiersprache Python enthalten ist (Python\Python...\DLLs).
- Nutzung in eingebetteten Systemen wie zum Beispiel Android oder zur Speicherung von Lesezeichen im Firefox.
- Webseite: https://sqlite.org/

Import der Programmbibliothek

```
import sqlite3
```

- Bereitstellung der Funktionen, Methoden und Datentypen der Programmbibliothek sqlite3.
- Mit Hilfe des Schlüsselwortes import wird ein Modul in eine Code-Datei eingebunden.
- Das Modul sqlite3 wird ab der Python-Version 3.x genutzt.

Informationen zu der genutzten Version

```
import sqlite3

print("Version des Moduls: ", sqlite3.version)
print("Version der SQLite-Bibliothek zur Laufzeit: ",
        sqlite3.sqlite_version)
```

- sqlite3.version gibt die Version des Python-Moduls sqlite3 als String zurück.
- sqlite3.sqlite_version gibt die genutzte SQLite-Version zurück.

Nutzung von Funktionen aus dem Modul

sqlite3	.	version
Modul	.	Funktion / Attribut

- Das Modul ist mit Hilfe der Anweisung `import` in die Code-Datei importiert.
- Mit Hilfe des Punktoperators wird die Funktion oder das Attribut mit dem dazugehörigen Modul verbunden.
- Die Funktion oder das Attribut (rechts vom Punktoperator) ist in dem Modul (links vom Punktoperator) definiert.

Programmbibliothek

- Pfad: Python\Python36-32\DLLs
- Die Datei sqlite3.dll kann jederzeit durch eine andere Version ausgetauscht werden. Falls eine 32-Bit-Version von Python genutzt wird, muss auch eine 32-Bit-Version von SQLite genutzt werden.
- Download der verschiedenen Versionen: https://www.sqlite.org/download.html

Was ist eine „Datenbank“?

- Verwaltung von großen Datenmengen.
- Strukturierte Ablage von Daten.
- SQLite: Ablage von großen Datenmengen in einer Datei an einem beliebigen Speicherort.

Verbindung zu einer Datenbank

```
import sqlite3

datenbank = "chinook.db"
connection = sqlite3.connect(datenbank)
```

- Mit Hilfe der Methode .connect() wird eine Verbindung zu einer Datenbank hergestellt.
- Der Methode wird als Argument der Name der gewünschten Datenbank übergeben.
- Die Referenz auf die Verbindung wird in der Variablen connection gespeichert.

Hinweis

```
import sqlite3

datenbank = "myDatabase.db"
connection = sqlite3.connect(datenbank)
```

- Falls die Datenbank an dem angegebenen Speicherort nicht gefunden wird, wird eine leere Datenbank angelegt.
- Andernfalls wird die vorhandene Datenbank zum Lesen und Schreiben geöffnet.

Dateinamen

- SQLite-Datenbanken haben die Dateiendung „.db“, „.sqlite“, „.db3“ oder „.sqlite3“.
- Der Dateinamen befolgt die 8 + 3 – Regel. Der Dateiname selbst ist 1 bis 8 Zeichen lang. Die Dateiendung 1 bis 3 Zeichen lang.
- Die Dateiendung muss angegeben werden.
- Der Dateiname sollte nur aus den lateinischen Groß- und Kleinbuchstaben, den Ziffern 0 … 9 und den Unterstrich bestehen.

Angabe des Speicherortes

- Der Speicherort der Datenbank kann absolut angegeben werden.
- Der Speicherort kann relativ zum Speicherort des Programmcodes angegeben werden.
- Die Datenbank kann temporär gespeichert werden.

Absoluter Pfad

```
C:\kurs_Python\beispiel\chinook.db
```

- Der Pfad wird von der Wurzel beginnend angegeben.
- Als Trennzeichen zwischen den Angaben wird der Schrägstrich genutzt.
- Ein Laufwerk (die Wurzel) wird durch einen Buchstaben plus den Doppelpunkt angegeben.
- Absolute Pfade sollten vermieden werden.

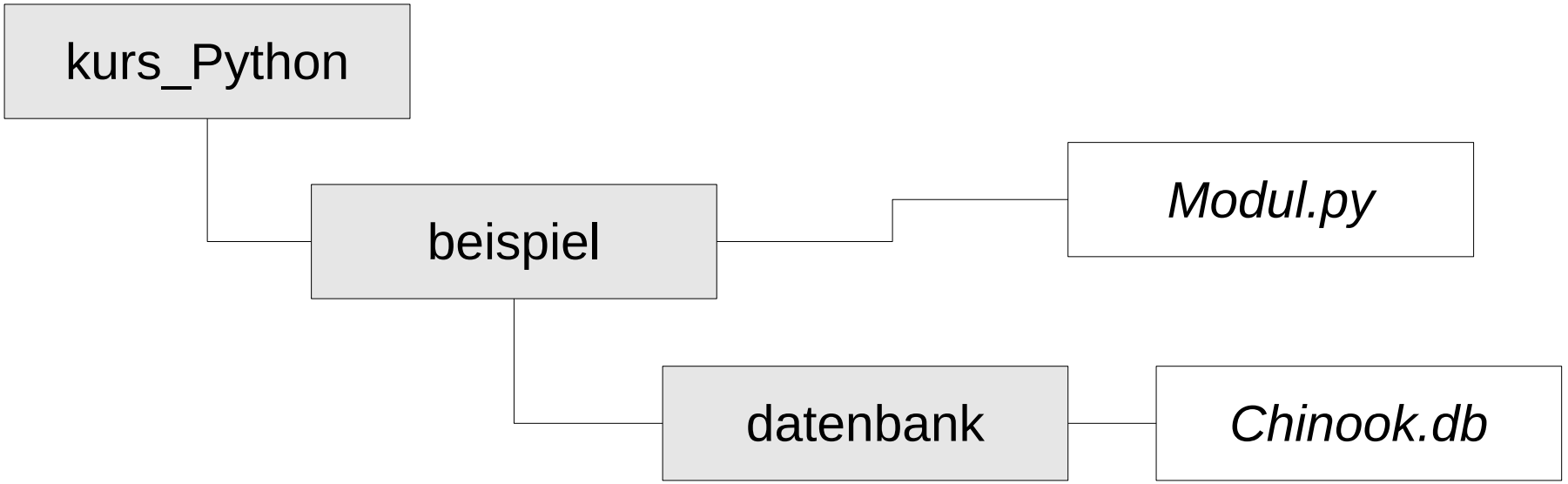
Relativer Pfad

```
datenbank/chinook.db


../datenbank/chinook.db
```

- Die Angaben des Pfades beziehen sich immer auf einen bestimmten Bezugspunkt.
- In Python wird der Speicherort einer Datenbank immer in Bezug zum Speicherort der Code-Datei angegeben.

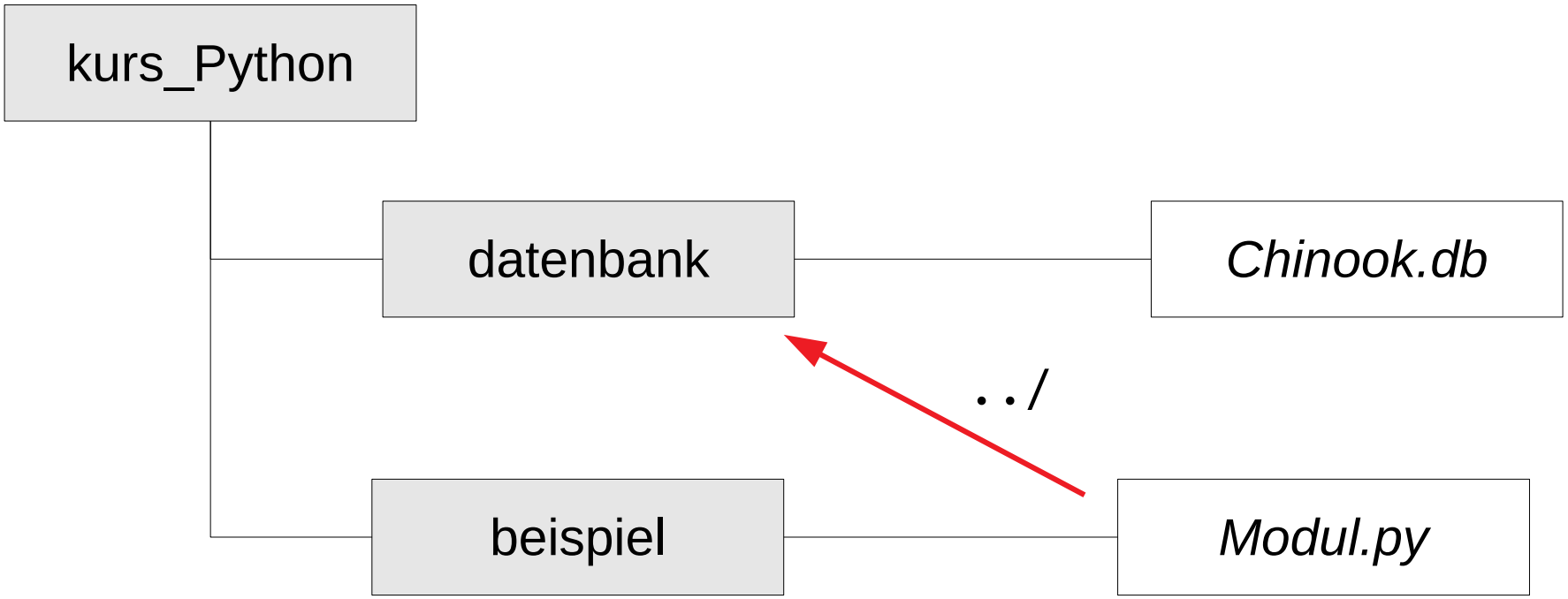
Beispiel 1

datenbank/chinook.db



Beispiel 2

../datenbank/chinook.db



Temporäre Datenbank

```
connection = sqlite3.connect(":memory:")
```

- Durch das Argument ":memory:" wird die Datenbank im RAM des Rechners abgelegt.
- Die Datenbank wird temporär genutzt.

Schließen einer Datenbank

```
import sqlite3

datenbank = "chinook.db"
connection = sqlite3.connect(datenbank)

connection.close()
```

- Mit Hilfe der Methode .close() wird die Verbindung zur Datenbank getrennt.
- Die Datenbank, auf die die Instanz verweist, wird geschlossen.

Datenbank-Schema

- Struktur der Datenbank.
- Logischer Entwurf einer relationalen Datenbank in Abhängigkeit des genutzten Systems.
- Welche Tabellen sind in der Datenbank vorhanden?

Tabelle sqlite_master

- Systemtabelle.
- Diese Tabelle bildet das Schema einer SQLite-Datenbank ab.
- Die Struktur der Datenbank wird in dieser Tabelle abgebildet.
- Siehe https://www.techonthenet.com/sqlite/sys_tables/index.php

Informationen in der Tabelle

- Die Spalte `name` speichert den Namen des Datenbankobjekts.
- Die Spalte `type` speichert die Objektart. Der Typ `table` verweist auf Tabellen. `index` verweist auf einen Index.
- Die Spalte `sql` speichert die Definition des Objekts

Auslesen der Datenbankstruktur

```
import sqlite3

datenbank = "chinook.db"
connection = None

try:
    connection = sqlite3.connect(datenbank)
    cursor = connection.cursor()

    cursor.execute("SELECT name
                    FROM sqlite_master
                    WHERE type='table';")
    print(cursor.fetchall())
```

Cursor-Objekt

```
connection = sqlite3.connect(datenbank)
cursor = connection.cursor()
```

- Das Objekt connection wird mit der Methode .cursor() über den Punkt-Operator verbunden.
- Ein Zeiger auf Element in der Datenbank wird erzeugt. Das Objekt connection verweist auf die gewünschte Datenbank.

Cursor

- Verarbeitung von SQL-Anweisung in einer relationalen Datenbank.
- Verweis auf einen Datensatz oder einen kleinen Block von Datensätzen, der von einer Auswahlabfrage zurückgegeben wird.
- Datensatz-Zeiger. Verweis auf einen bestimmten Datensatz in einer Tabelle.

Informationen aus einer Tabelle auswählen

```
SELECT name
FROM sqlite_master
WHERE type='table';
```

- Mit Hilfe einer Auswahlabfrage werden die Informationen ausgewählt.
- Die SQL-Anweisung setzt sich aus Befehlen, Tabellennamen und so weiter zusammen.

SQL-Anweisung

```
SELECT name
FROM sqlite_master
WHERE type='table';
```

- Beginn mit einem englischsprachigen Verb (hier: SELECT, Wähle aus).
- Das Verb am Anfang einer SQL-Anweisung beschreibt die Verarbeitung der Daten.
- Entsprechend des SQL-Befehls am Anfang wird die Anweisung aufgebaut.
- Beendigung mit einem Semikolon.

SQL-Befehle

```
SELECT name
FROM sqlite_master
WHERE type='table';
```

- SQL-Befehle beginnen immer mit einem Buchstaben.
- Die Befehle werden häufig groß geschrieben.
- Befehle in Verb-Form beschreiben die Handlung, die in der Datenbank ausgeführt werden soll.
- Befehle definieren den Ablageort der Daten.
- Befehle definieren Filter- und Sortierkriterien.

Auswahlabfragen

```
SELECT name
FROM sqlite_master
WHERE type='table';
```

- Auswahlanweisungen beginnen immer mit SELECT (wähle aus).
- Es werden alle oder einige Felder aus einer Tabelle ausgewählt.
- Das Resultat kann gefiltert und / oder sortiert werden.

Aufbau

SELECT	Wähle
feld01, feld02, ...	die Felder
FROM tabelle	von der Tabelle aus
WHERE [Bedingung]	Filtere diese
ORDER BY feld01, feld02 DESC, ...	Sortiere diese

… ausführen

```
cursor.execute("SELECT name
                FROM sqlite_master
                WHERE type='table';")
```

- Die Methode .execute() wird durch den Punkt-Operator mit einem Cursor-Objekt verbunden.
- Der Methode wird als Parameter eine SQL-Anweisung als String übergeben.

Hinweise

```
SELECT name
FROM sqlite_master
WHERE type='table'
```

- Der Methode .execute() werden SQL-Anweisungen immer als String übergeben.
- In Python werden Strings durch das Anführungszeichen oder das Apostroph begrenzt.
- In einer SQL-Anweisung werden aber auch Literale vom Typ „Text“ genutzt. Falls das Argument durch das Anführungszeichen begrenzt wird, muss das Text-Literal durch das Apostroph begrenzt werden und umgekehrt.

Resultat der Methode

```
cursor.execute("SELECT name
                FROM sqlite_master
                WHERE type='table';")
```

- Die Methode .execute() liefert als Resultat x Datensätze entsprechend der Auswahlabfrage zurück.
- In diesem Beispiel werden die Namen der, in der Datenbank gespeicherten Tabellen durch die SQL-Anweisung ausgewählt.

… und holen

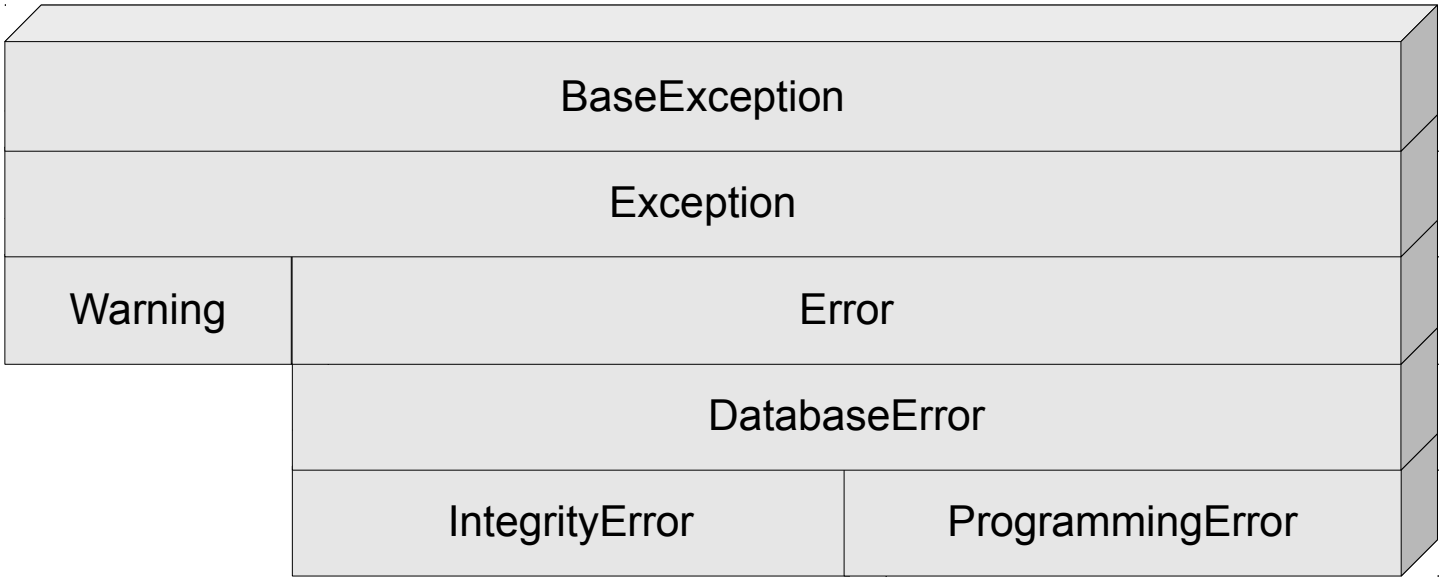
```
cursor.execute("SELECT name FROM sqlite_master
                    WHERE type='table';")
print(cursor.fetchall())
```

- Mit Hilfe der Methode .fetchall() des Cursor-Objekts wird das Resultat der SQL-Anweisung vollständig geholt.
- Die gewählten Datensätze werden als Liste zurückgegeben. Falls keine passenden Datensätze vorhanden sind, wird eine leere Liste ausgegeben.
- In diesem Beispiel werden alle Tabellennamen in der Datenbank auf der Standardausgabe mit Hilfe von print() ausgegeben.

Exception

- Ausnahme von der Regel.
- Fehler, die zur Laufzeit des Programms auftreten können.
- Fehler, die im Code abgefangen und repariert werden können.

Hierarchie in Bezug auf SQLite



Mögliche Fehler

- DatabaseError. Fehler, die die Datenbank betreffen.
- IntegrityError. Fremdschlüssel ist nicht definiert. Fehler in Bezug auf die referentielle Integrität einer Datenbank.
- ProgrammingError. Eine Tabelle existiert nicht. Fehler in einer SQL-Anweisung.
- OperationalError. Der Datenbank-Name wird nicht gefunden und so weiter.

Exception-Handling

```
try:
    connection = sqlite3.connect(":memory:")

except sqlite3.Error as e:
    print(e)

except:
    exceptionTupel = sys.exc_info()
    print("Error-Type: ", exceptionTupel[0])
    print("Error-Wert / Beschreibung: ", exceptionTupel[1])

finally:
    if not(connection is None):
        connection.close()
```

Aufbau

try: connection = sqlite3.connect(":memory:")	Versuche
except sqlite3.Error as e: print(e)	Abfangen von speziellen Fehlern
except: print("Allgemeiner Laufzeitfehler")	Abfangen von allen anderen Fehlern
else: print("Kein Fehler")	Wenn kein Fehler aufgetreten ist ...
finally: connection.close()	Aufräumarbeiten

Versuche die Anweisungen auszuführen

```
try:
    connection = sqlite3.connect(":memory:")
```

- Beginn mit dem Schlüsselwort try.
- Dem Schlüsselwort folgt der Doppelpunkt. In der nächsten Zeile beginnt der, zu dem Befehl gehörende Codeblock.
- In dem Codeblock können Laufzeitfehler auftreten, müssen aber nicht.

Abfangen aller Laufzeitfehler

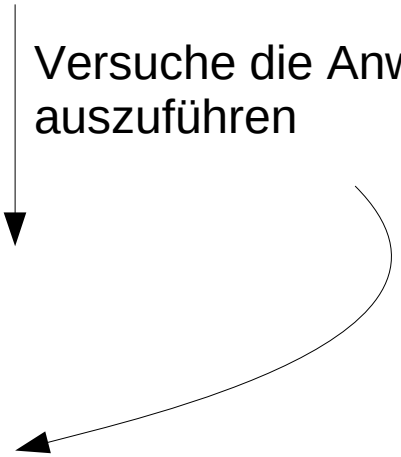
```
try:
    connection = sqlite3.connect(":memory:")

except:
    print("Ein unbekannter Fehler ist aufgetreten")
```

- Dem Schlüsselwort except folgt der Doppelpunkt.
- In der nächsten Zeile beginnt der, zu dem Befehl gehörende Codeblock. Dieser Codeblock fängt alle erzeugten Laufzeitfehler ab.

Ablauf

```
try:
  connection = sqlite3.connect(":memory:")
```

```
except:
  print("Allgemeiner Laufzeitfehler")
```



Spezifizierung von Exceptions

```
try:
    connection = sqlite3.connect(":memory:")

except sqlite3.Error as e:
    print(e)
```

- Dem Schlüsselwort except folgt der Name des abzufangenden Laufzeitfehlers.
- In diesem Beispiel werden Fehler, die im Modul sqlite3 definiert sind, abgefangen.

Aufbau

except	sqlite3.Error	:
except		:
except	Name	:

- Dem Befehl except kann der Name eines Laufzeitfehlers folgen, muss aber nicht.
- Die abzufangende Exception kann in einem Modul definiert oder benutzerdefiniert sein.

Name einer Exception

- Abfangen einer Standard-Expection. Der Name wird direkt angegeben.
- Abfangen einer Exception, die in einem Modul definiert ist. Je nach dem wie das Modul importiert wird, wird ein qualifizierter Name oder ein unqualifizierter Name genutzt.

Nutzung von qualifizierte Namen

```
import sqlite3

try:
    connection = sqlite3.connect(":memory:")

except sqlite3.Error as e:
    print(e)
```

- Das Modul ist vollständig mit Hilfe des Schlüsselwortes `import` geladen.
- Qualifizierter Name: `Modul.Element`. In diesem Beispiel ist das Element Error in dem Modul sqlite3 definiert. Das Modul wird mit dem gewünschten Element durch den Punktoperator verbunden.

Reihenfolge von except-Anweisungen

```
try:
    connection = sqlite3.connect(":memory:")

except sqlite3.Error as e:
    print(e)

except:
    print("Ein unbekannter Fehler ist aufgetreten")
```

Nutzung eines Alias

except	sqlite3.Error	as	e	:
except	Exception	as	Alias	:

- Der Alias ist frei wählbar. Häufig wird der Bezeichner e oder err genutzt.
- Der Alias ist ein Platzhalter für die abgefangene Ausnahme.
- Mit Hilfe des Platzhalters kann eine entsprechende Meldung ausgegeben werden. Die Ausnahme kann mit Hilfe von raise weitergereicht werden.

Aufräumarbeiten

```
finally:
    if not(connection is None):
        connection.close()
```

- Der Codeblock finally wird immer ausgeführt, egal ob eine Ausnahme aufgetreten ist oder nicht.
- In diesem Block sollte die Verbindung zu der Datenbank geschlossen werden.