INTRO APPWEB 2023

Databases… in Python

Storing data (on the server, too)

Luigi De Russis

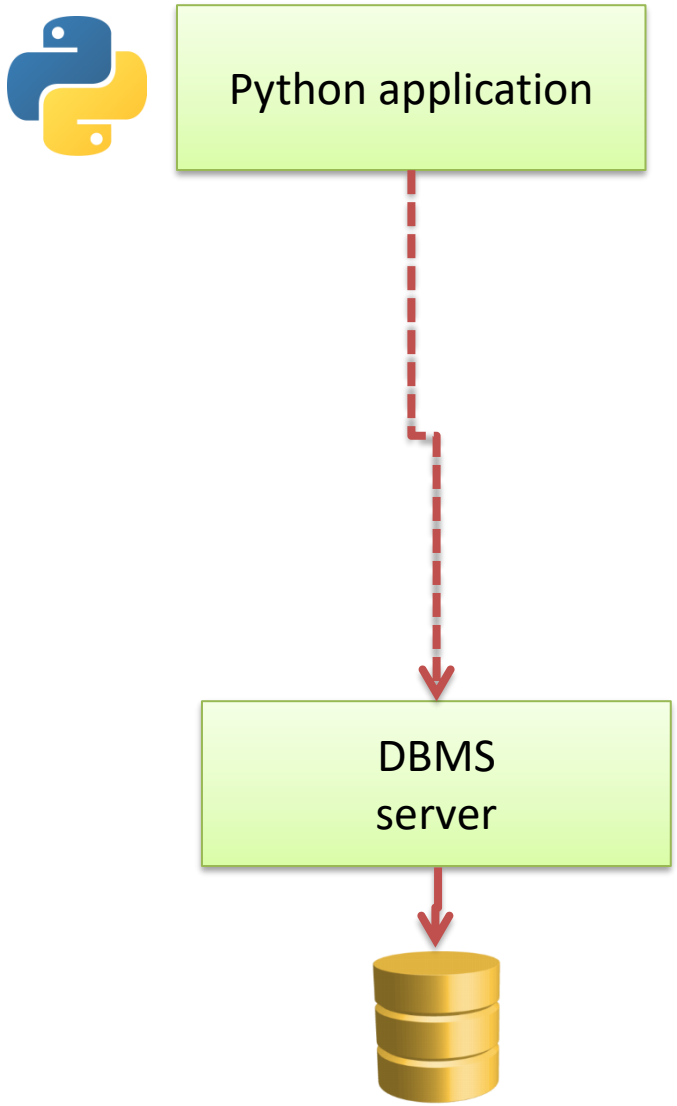
Photo by Tobias Fischer on Unsplash

Politecnico di Torino

Goal

- Making data ‘persistent’
  - When application restarts
- Manage big amounts of data
  - Not all in-memory
- Exploit the power of SQL
  - Complex data
  - Complex queries

General Architecture



Analyzed Databases

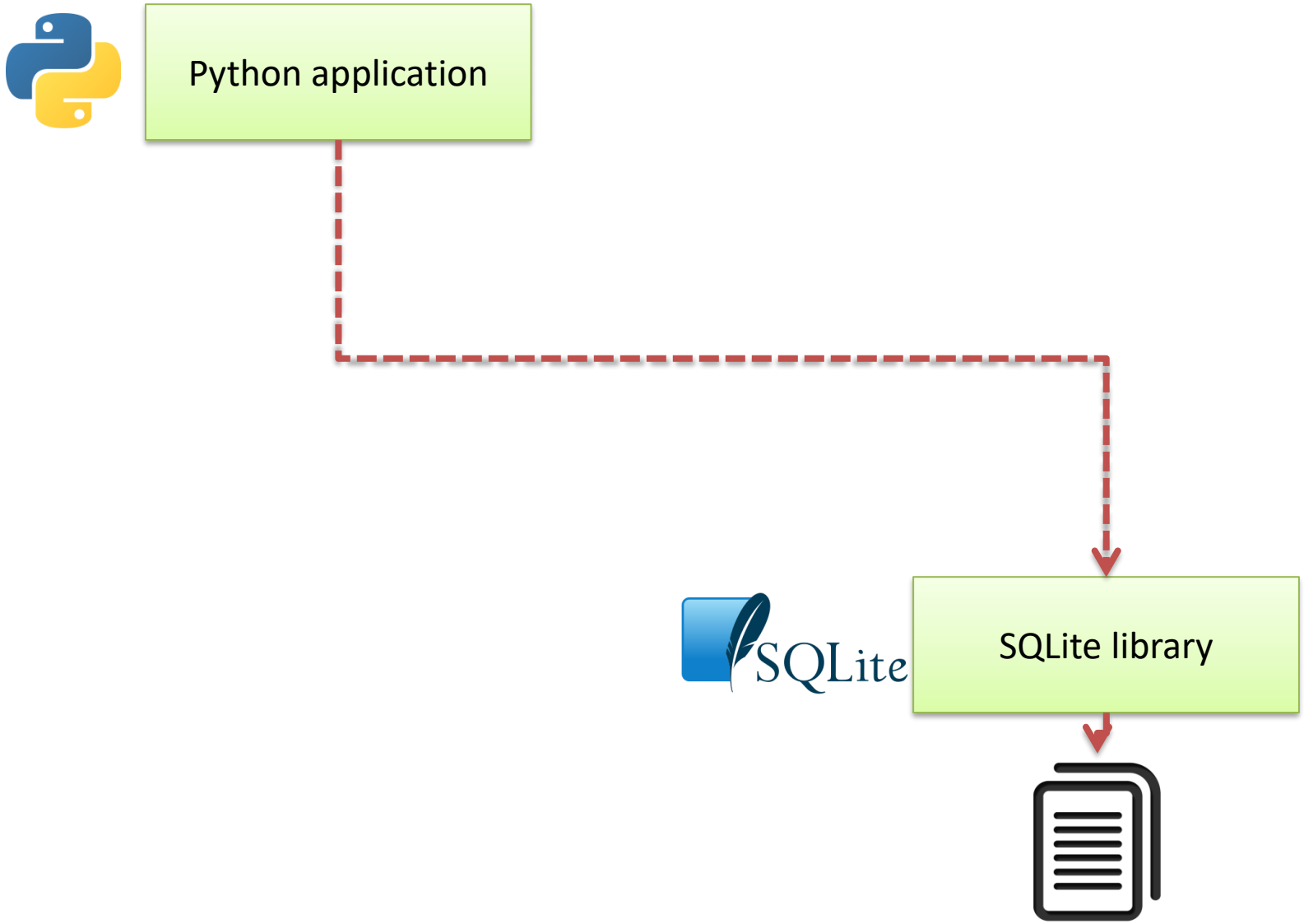
MySQL

- Open-source database server (from Oracle)
- Full featured
- Runs as a separate process (may be on a different computer)
- Allows concurrent access
- https://www.mysql.com

MariaDB

- Open-source fork of MySQL server
- Community-driven
- 99% compatible
- In some cases, faster
- On most Linux distributions
- http://mariadb.org/

General Architecture



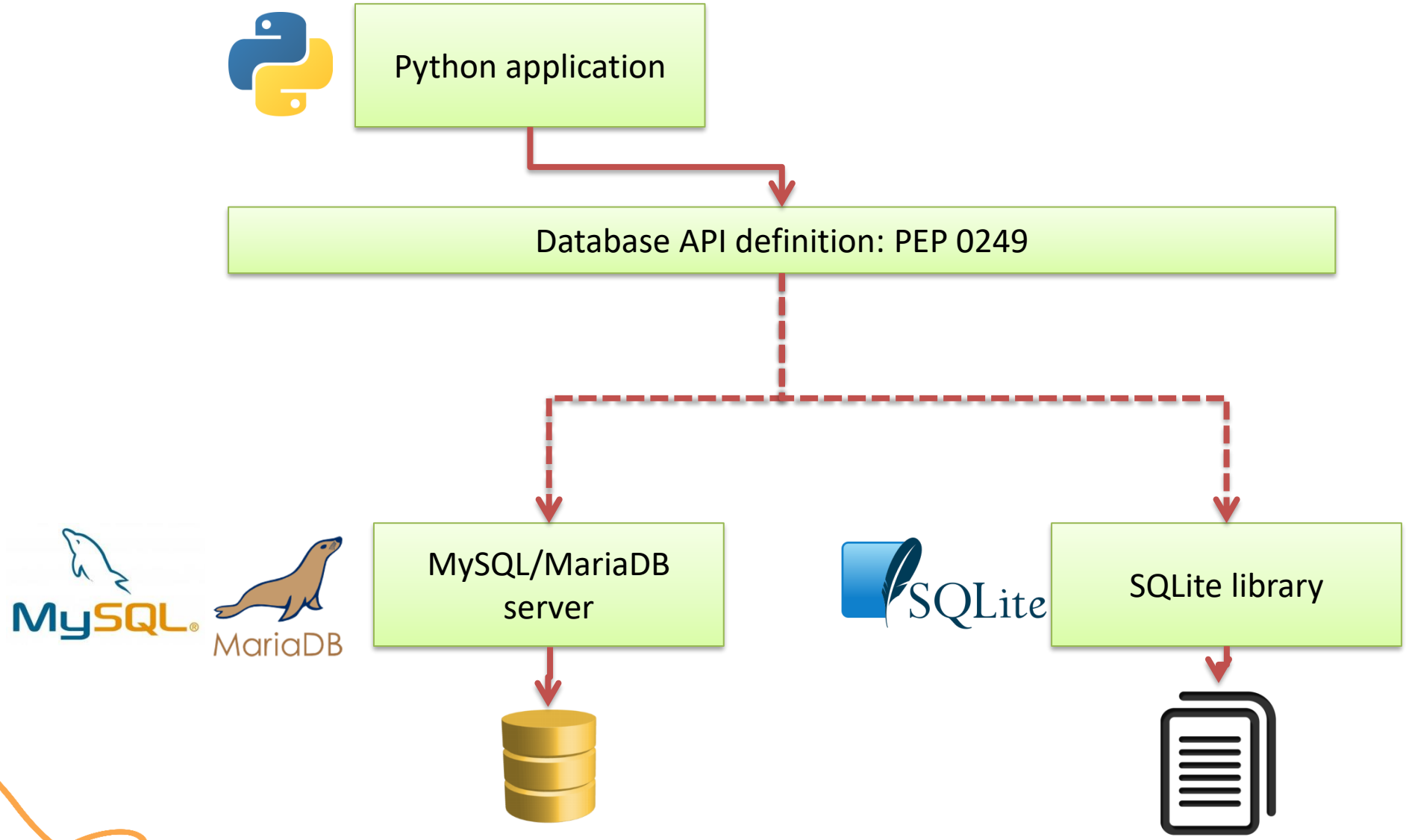
Analyzed Databases

MySQL / MariaDB

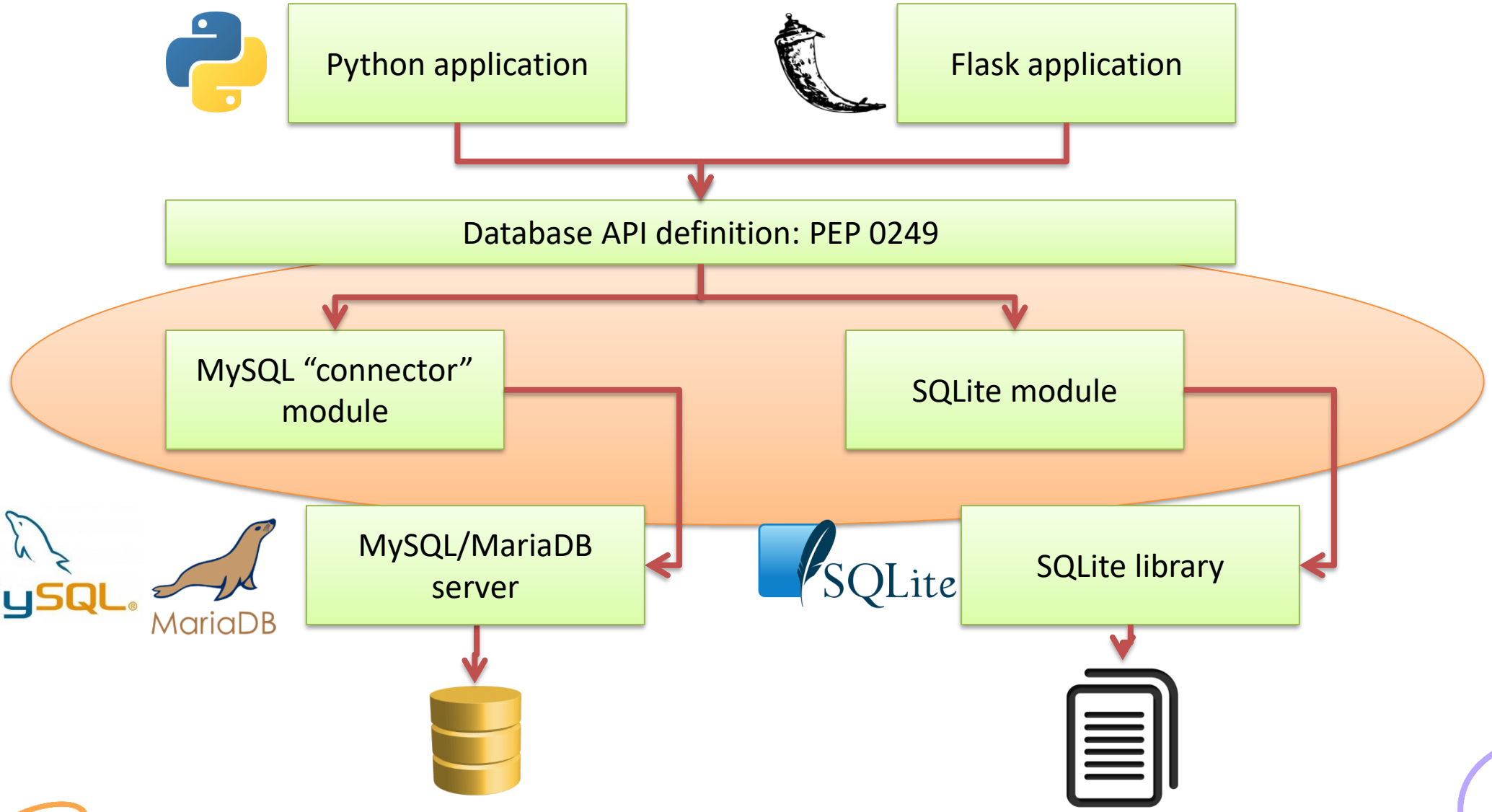
SQLite

- Open-source file-based storage
- Software library integrated in your program (serverless)
- Self-contained
- https://www.sqlite.org/

General Architecture



General Architecture



Other Options

- PostgreSQL – more complex, but more complete than MySQL/MariaDB
- Non-relational databases (NoSQL)
  - not considered here

PEP 0249

- Python Database API Specification v2.0
  - https://www.python.org/dev/peps/pep-0249/
- Specifies a standard API that Python modules that are used to access databases should implement
- Does not provide a library nor a module
- Third-party modules may adhere to these specifications

Main Concepts in PEP 249

- Access to database is provided through a **connect** method, that returns a **Connection** object
- For executing queries, you need a **Cursor** object, that can be obtained by the Connection
- A **cursor** may **execute**() a SQL query, with parameters
- A **cursor** may **fetch** the **results** of the query

Minimal Example

```
1  sql = "SELECT id, original, modified FROM translation"

2  conn = mysql.connector.connect(user='root', password='',
   host='localhost', database='funnyecho')

3  cursor = conn.cursor()
   cursor.execute(sql)

4  translations = cursor.fetchall()

5  cursor.close()
   conn.close()

6  return translations
```

Minimal Example

```
1  sql = "SELECT id, original, modified FROM translation"

2  conn = mysql.connector.connect(user='root', password='',
   host='localhost', database='funnyecho')

3  cursor = conn.cursor()
   cursor.execute(sql)

4  translations = cursor.fetchall()

5  cursor.close()
   conn.close()

6  return translations
```

The **only** step that depends on the type of database

Step 1: Query Definition

- Write a correct SQL statement, stored as a Python string
  - `sql = "SELECT id, original, modified FROM translation"`
- Variable arguments may be specified with '%s' or '?' placeholders
  - according to the underlying database/library
  - `sql = "INSERT INTO translation (original, modified) VALUES (%s, %s)"`
  - `sql = "INSERT INTO translation (original, modified) VALUES (?, ?)"`

Placeholders

- **Never** use string concatenation over SQL statements. N.E.V.E.R. Huge security problems (SQL Injection)
- SQL statement “templates” that include placeholders
- Actual values passed in `.execute()`
- Different libraries use different types of placeholder

Placeholder Syntax

MySQL/MariaDB

- C-like format string
- `...WHERE name=%s`
- Beware: always use %s, even for numeric data – not %d or %f

SQLite

- Question mark
- `...WHERE name=?`

Step 2: Database Connection

- Depending on the library, use the provided ‘connect’ method
- The method parameters are dependent on the module implementation (non-standard)
  - `conn = mysql.connector.connect(user='root', password='', host='localhost', database='funnyecho')`
  - `conn = sqlite3.connect('example.db')`

Step 3: Query Execution

- First, obtain a cursor from the connection
  - `cursor = conn.cursor()`
- Then, execute the query
  - `cursor.execute(sql)`
- Query parameters (%s/? placeholders) are specified as a ‘tuple’ argument
  - `cursor.execute(sql, (txtbefore, txtafter) )`
  - `cursor.execute(sql, (txtid,) )`
  - Beware: one-element tuples require trailing `,`

Step 4 (SELECT): Result Analysis

- Only if the query was a SELECT
- Use various methods of **cursor**:
  - `cursor.fetchone()` # next result
  - `cursor.fetchall()` # all remaining results
  - They return tuples, corresponding to the SELECT'ed columns
  - https://www.python.org/dev/peps/pep-0249/#cursor-methods

Step 4 (UPDATE): Commit the Change

- For INSERT, UPDATE and DELETE there is no result
- The change is not applied immediately to the database, but needs to be "committed"
- `conn.commit()`
  - Will commit all pending executed queries in the connection
- Must be called before `conn.close()`
- DO NOT forget, or you will lose your data

Step 5 (a): Clean Up

- When the cursor is no longer needed
- `cursor.close()`

Step 5 (b): Clean Up

- Do not forget to close the connection, thus freeing up resources on the database server
  - `conn.close()`
- Write the close statement *immediately*, otherwise you will forget it
- Remember not to ‘return’ the function before cleaning up

Step 6: Use the Results

- Analyze the returned data and do what the application requires for them
- If further queries are needed, go back to step 3
  - re-use the same Connection, creating new Cursors

Using SQLite

- SQLite is a simple file-based storage library
- Since Python 2.5, it is included by default, in the “sqlite3” package
  - https://docs.python.org/3/library/sqlite3.html
  - Developed at https://github.com/ghaering/pysqlite
- The “connection” just means specifying the file name
  - `import sqlite3`
  - `conn = sqlite3.connect('example.db')`
- Remember: placeholder = ?

Alternative SQLite Libraries

- Another Python SQLite Wrapper
  - https://github.com/rogerbinns/apsw/
- More powerful and complete than the built-in library
- It does not follow the PEP 249
  - No interchangeability with other database drivers ☹

Extra, for the curious

USING MYSQL OR MARIADB

Using MySQL

- Pre-requisite: a working installation of the mysql server
  - http://dev.mysql.com/downloads/mysql/
- Pre-requisite: a working installation of the mariadb server
  - https://mariadb.org/download/

MySQL Connectors

Official connector (Oracle)

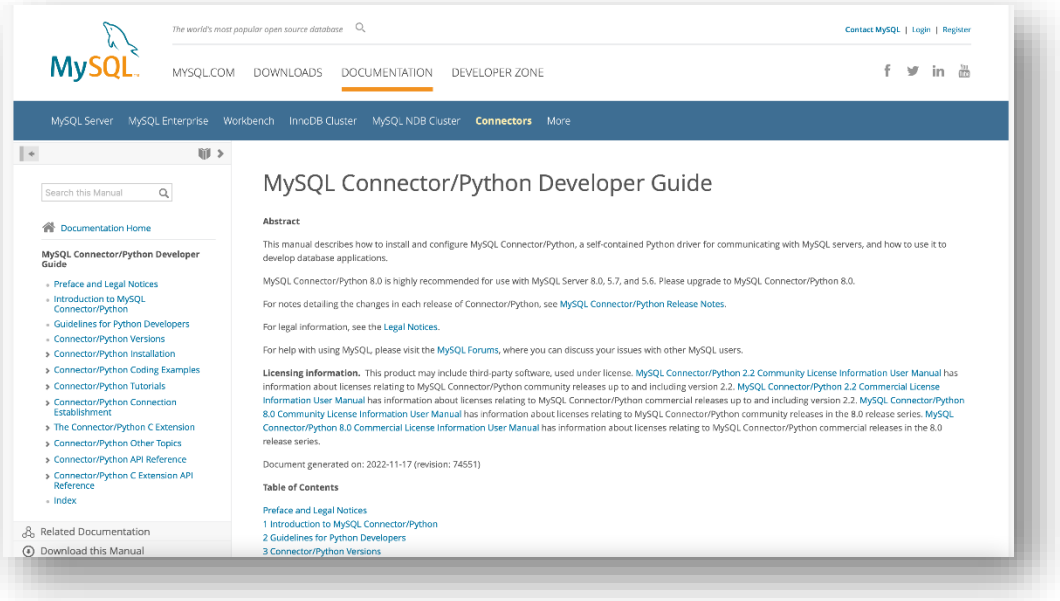
- Download and install the “MySQL Connector for Python”
  - http://dev.mysql.com/downloads/connector/python/
  - Provides the package “mysql.connector”

Alternative (from pip)

- Pure Python implementation
  - https://github.com/PyMySQL/PyMySQL/
  - pip install PyMySQL
  - Provides the package “pymysql”
- Nearly drop-in replacement
- Easier to install

MySQL Python Connector

- To use: import mysql.connector
- Well-done documentation at
  - http://dev.mysql.com/doc/connector-python/en/index.html



Connecting With mysql (Oracle)

- Basic form
  - `import mysql.connector`
  - `cnx = mysql.connector.connect (`
    - `user='joe',`
    - `password='xxx',`
    - `database='test',`
    - `host='localhost' )`

- Additional parameters
  - http://dev.mysql.com/doc/connector-python/en/connector-python-connectargs.html

Connecting with PyMySQL

```
– import pymysql
– cnx = pymysql.connect ( ... )
– cursor = cnx.cursor()
```

- … Same connection parameters
- … Same placeholder (%s)
- … When in doubt, check the Oracle documentation

License

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
  - **Share** — copy and redistribute the material in any medium or format
  - **Adapt** — remix, transform, and build upon the material
  - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
  - **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - **NonCommercial** — You may not use the material for commercial purposes.
  - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- https://creativecommons.org/licenses/by-nc-sa/4.0/