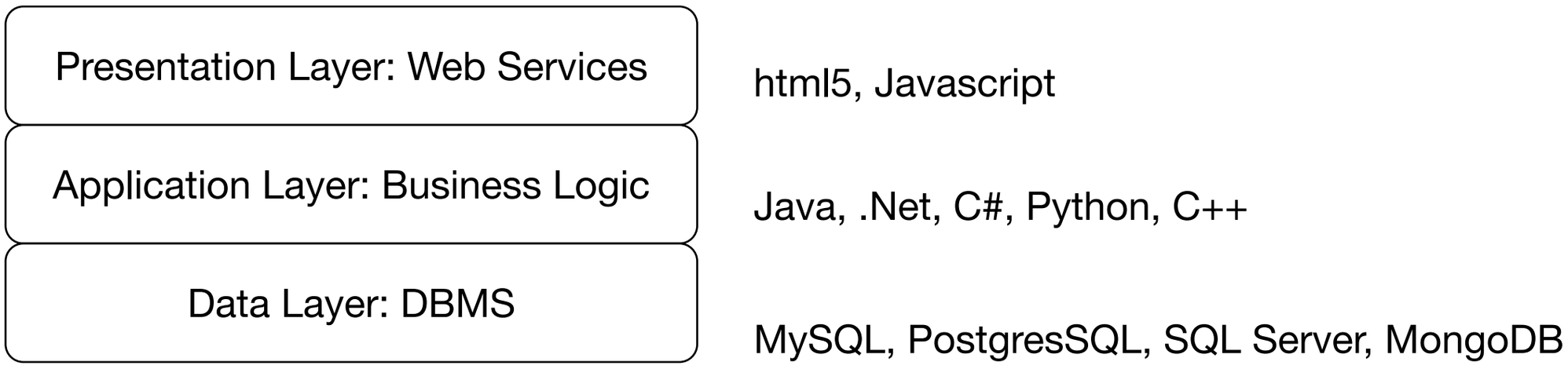
Database System Interactions

Thomas Schwarz, SJ

Three Tier Web Server Architecture

- Standard architecture for e-commerce sites
  - Tiered / layered architecture around since the THE operating system 1965



Three Tier Web Server Architecture

- Web Services Layer:
  - User interact with the site using a web browser
  - Forms, scripts, …
  - Requests are being routed to the application layer
    - Simple example: Embed PHP scripts into a web server
      - Download: LAMP / WAMP / XAMPP etc. With Apache, MySQL, PHP, Perl, …
      - Embed PHP script in HTML: <?php … ?>

Three Tier Web Server Architecture

- Application Tier
  - Simple system: Bypass application tier by directly translating web requests to database requests
  - Normally:
    - Integrate different databases
    - Implement business logic

Three Tier Web Server Architecture

- Database Tier:
  - Executes queries (including updates and inserts)

Integrating SQL with Application Layer

- Application layer uses languages like PHP, Python, Java, …
- Needs to interact with an application programming environment

SQL Environment

- SQL environment
  - Schemas: Tables, views, assertions, triggers, stored procedures, character sets, grant statements (for rights) maintained by a catalog
  - Servers / Clients
    - Clients need to connect to a server
    - Client/server connection is divided into Sessions
      - Each session selects a catalog and a schema

Integrating SQL with Application Layer

- **Impedance mismatch problem**
  - All languages / environment are Turing complete
  - Standard SQL is not:
    - Not everything that a computer can do can be done with SQL
      - E.g. cannot compute factorial with SQL
- Need to use both SQL (to interact with database) AND application level program

Integrating SQL with Application Layer

- Program sets up a connection to a database and closes it at the end
  - which might be automatic

Integrating SQL with Application Layer

- Central idea is the 'cursor'
  - Basically a pointer into the result table of an SQL query
  - Usually:
    - Can get result table row by row
    - Can get result table all at once
      - Could be hard on memory resources
    - Can get result table in tranches

Integrating Python with MySQL

- Solutions differ widely according to application tier environment and
    - Here: look at how to connect Python with MySQL
        - There are a variety of Python packages that will do that
            - I chose SQL-connector

Python — MySQL

Thomas Schwarz

Python 3 SQL connector

- Needed: Python 3
- Install MySQL Connector
    - Install with pip
        - Be careful for which Python you install
            - E.g. Mac has a Python 2.7 installed as part of the OS
- You will need to know your MySQL password
    - If necessary, just re-install MySQL

Python 3 MySQL Connector

- You can use
  - https://www.mysqltutorial.org/python-mysql/

MySQL connector

- Task 1: Connect to the mysql server / database

```
import mysql.connector

MYHOST = 'localhost'
MYUSER = 'root',
MYPASSWORD = '1EmilLie',

mydb = mysql.connector.connect(
    host = MYHOST,
    user = MYUSER,
    password = MYPASSWORD
    )
```

MySQL connector

- You interact by creating a cursor.

MySQL connector

- Task 2: Create a data base.

```
mycursor = mydb.cursor()
mycursor.execute("DROP DATABASE mydatabase")
```

```
try:
    mycursor.execute("CREATE DATABASE IF NOT EXISTS mydatabase")
except:
    print('could not create database')
```

MySQL connector

```
mycursor.execute("SHOW DATABASES")
for item in mycursor:
    print(item)
```

- Some databases are part of mysql and always there

```
('classicmodels',)
('information_schema',)
('mydatabase',)
('mysql',)
('performance_schema',)
('sys',)
```

MySQL connector

- You can also connect directly to a database

```
MYHOST = 'localhost'
MYUSER = 'root',
MYPASSWORD = '1EmilLie',
MYDATABASE = 'mydatabase'

mydb = mysql.connector.connect(
    host = MYHOST,
    user = MYUSER',
    password = MYPASSWORD,
    database = MYDATABASE
    )
mycursor = mydb.cursor()
```

MySQL connector

- Creating a table:

```
try:
    mycursor.execute(
"""CREATE TABLE animals (name VARCHAR(63), food VARCHAR(63))"""
        )
except:
    print('did not create table')


mycursor.execute("SHOW TABLES")
print('Showing Tables')
for item in mycursor:
    print(item)
```

MySQL connector

- Insertion / Deletion

```
mycursor.execute(
"""
DELETE FROM animals;
"""
)
mycursor.execute(
    """INSERT INTO animals Values ('elephant', 'bananas');
    """
    )
```

Excursion

- Quotes in Python:
  - Use single quotes for strings ', "
    - The other type of quote can be part of the string
  - Use triple quotes to include new-line characters
  - Use triple quotes to generate manual entries for functions and classes

MySQL connector

```
mycursor.execute(
    """
INSERT INTO
        animals
VALUES
        ('monkey', 'roti'),
        ('dog', 'meat'),
        ('cat', 'miece'),
        ('mouse', 'grain');
    """
    )
```

MySQL connector

- You can of course construct the statements in a more pythonesque way

```
food_list = []
food_list.append(('cat','fish'))
food_list.append(('goose','corn'))
food_list.append(('donkey','hay'))
food_list.append(('grizzly','cat'))
food_list.append(('grizzly','fish'))
food_list.append(('pheasant','insects'))
food_list.append(('fish','worm'))
food_list.append(('fish','fish'))
food_list.append(('fish','plancton'))
food_list.append(('grizzly','berries'))

for ani, prey in food_list:
    sql_statement = ('INSERT INTO animals\n' +
                     f"VALUES ('{ani}', '{prey}');")
    print(sql_statement)
    mycursor.execute(sql_statement)
```

MySQL connector

- Do not forget to commit:

```
mydb.commit()
```

SELECT statements

- Retrieval with select:

```
mycursor.execute("SELECT * FROM animals")
myresult = mycursor.fetchall()
for item in myresult:
    print(item)
```

```
print('Contained queries')
mycursor.execute(
"""SELECT a1.*, a2.food
FROM animals a1, animals a2
WHERE a1.food = a2.name
"""
)
myresult = mycursor.fetchall()
for item in myresult:
    print(item)
```

SELECT statements

- Once we have executed a select

```
cursor.execute("SELECT * FROM books")
```

  - we need to fetch the data with
    - `fetchone`
    - `fetchmane`
    - `fetchall`
  - access the cursor data structure

SELECT statements

- After a select statement, the cursor contains the rows as a tuple of strings

```
try:
    mycursor.execute("SELECT * FROM customers;")
    for item in mycursor:
        print(item)
except:
    print('Could not execute select statement')
```

```
(103, 'Atelier graphique', 'Schmitt', 'Carine ', '40.32.2555', '54, rue Royale', None, 'Nantes', None, '44000',
'France', 1370, Decimal('21000.00'))
(112, 'Signal Gift Stores', 'King', 'Jean', '7025551838', '8489 Strong St.', None, 'Las Vegas', 'NV', '83030', 'USA',
1166, Decimal('71800.00'))
(114, 'Australian Collectors, Co.', 'Ferguson', 'Peter', '03 9520 4555', '636 St Kilda Road', 'Level 3', 'Melbourne',
'Victoria', '3004', 'Australia', 1611, Decimal('117300.00'))
(119, 'La Rochelle Gifts', 'Labrune', 'Janine ', '40.67.8555', '67, rue des Cinquante Otages', None, 'Nantes', None,
'44000', 'France', 1370, Decimal('118200.00'))
```

SELECT statements

- Using fetchone:
  - Obtain the next result of the row

```
try:
    mycursor.execute("SELECT * FROM customers;")
    rows = mycursor.fetchall()
    print(f'We got {mycursor.rowcount} rows')
    for row in rows:
        print(row)

except Exception as e:
    print('Could not execute select statement')
    print(e)
```

SELECT statements

- fetchmany(nr_rows) allows to limit the number of rows retrieved
  - Returns array, that can be empty

```
try:
    mycursor.execute("SELECT city FROM offices;")
    while True:
        rows = mycursor.fetchmany(3)
        if not rows:
            break
        for row in rows:
            print(row)
        print(10*'-')
```

SELECT statements

- fetchmany( )

```
('San Francisco',)
('Boston',)
('NYC',)
----------
('Paris',)
('Tokyo',)
('Sydney',)
----------
('London',)
----------
```

- If there are no more rows, the result is an empty list

SELECT statements

- fetchone( ) returns a single row as a tuple of field values

```
try:
    mycursor.execute("SELECT city FROM offices;")
    while True:
        row = mycursor.fetchone()
        if not row:
            break
        print(row[0])
```

Calling procedures

- Use the callproc function
  - Takes as parameters the name of the procedure and the arguments as a list
  - stored_results( ) method in cursor contains the results
    - Each result is a list of tuples

Calling procedures

- Example:
  - Create a stored procedure getEmp that gives the names and their office city:

```
CREATE PROCEDURE 'getEmp'( )
BEGIN
    SELECT CONCAT(firstName, ' ', lastName) AS author, city
     FROM employees JOIN offices USING(officeCode);
END
```

Calling procedures

- Example (cont.)

Result Grid | Filter Rows:

author	city
Diane Murphy	San Francisco
Mary Patterson	San Francisco
Jeff Firrelli	San Francisco
Anthony Bow	San Francisco
Leslie Jennings	San Francisco
Leslie Thompson	San Francisco
Julie Firrelli	Boston
Steve Patterson	Boston
Foon Yue Tseng	NYC
George Vanauf	NYC
Gerard Bondur	Paris
Loui Bondur	Paris
Gerard Hernan…	Paris
Pamela Castillo	Paris
Martin Gerard	Paris
Mami Nishi	Tokyo
Yoshimi Kato	Tokyo
William Patterson	Sydney
Andy Fixter	Sydney
Peter Marsh	Sydney
Tom King	Sydney
Larry Bott	London
Barry Jones	London

Calling procedures

- Example (cont.)
  - To access the result, we use a **double** loop

```
try:
    mycursor.callproc('getEmp')
    for item in mycursor.stored_results():
        for row in item:
            print(f"{row[0]} in {row[1]}")
```

Calling procedures

- Example 2:
  - Find the total number of orders

```
CREATE PROCEDURE 'total_order'(IN customer_name VARCHAR(16) )
BEGIN
    DECLARE tos INT DEFAULT 0;

     SELECT COUNT(*)
     INTO tos
     FROM orders JOIN customers USING(customerNumber)
     WHERE customerName LIKE CONCAT('%', customer_name, '%');

     SELECT tos;
END
```

Calling procedures

- Example 2 (cont.)
  - We need to specify the arguments as an array

```
try:
    mycursor.callproc('total_order', ['Euro'])
    for item in mycursor.stored_results():
        for row in item:
            print(row[0])
```