CS1116/CS5018

Web Development 2

Dr Derek Bridge

School of Computer Science & Information Technology
University College Cork

Relational databases

- A **database** is a repository of integrated data
- A **relational database** organises the data into **tables**
- Mathematically speaking, the tables are **n-ary relations**
- **SQL** (Structured Query Language) is a 'standard' language for defining and manipulating relational databases:
  - it is a **data definition language** — it has commands for creating and deleting tables; and
  - it is a **data manipulation language** — it has commands for inserting rows into tables, deleting rows and retrieving rows
- We use MySQL — one of the most popular database management systems, especially on the Web

Database of past and future gigs

```
CREATE TABLE gigs
(
    num INT AUTO_INCREMENT,
    band VARCHAR(50),
    gig_date DATE,
    PRIMARY KEY (num)
);
```

```
INSERT INTO gigs (band, gig_date)
    VALUES ('Decaying Shroom', DATE('2020-01-12')),
           ('Belated Tonic', DATE('2020-01-14'));
```

Band names produced by www.bandnamemaker.com

Querying the database from Python: the essentials

```
#!/usr/local/bin/python3

import pymysql as db

print('Content-Type: text/html')
print()

connection = db.connect('localhost', 'userid', 'password', 'database_name')

cursor = connection.cursor(db.cursors.DictCursor)

cursor.execute("""SELECT band, gig_date FROM gigs
                  WHERE gig_date >= CURDATE()""")

for row in cursor.fetchall():
    print(row['band'], row['gig_date'])

cursor.close()
connection.close()
```

Notes

1. Import the database module for your database server:
   - Different modules for different DBMS (mySQL, Oracle, ...)
   - But many of these modules provide the same set of functions, which makes your code portable
2. Create a **connection**:
   - `localhost`: the machine on which the DBMS runs
   - `userid`: your usual CompSci user id, e.g. `db12`
   - `password`: the database password you used in CS1106/CS5021, **not** your usual Linux password
   - `database_name`: the database name you used in CS1106/CS5021, e.g. `2020_db12`
3. Create a **cursor** for executing SQL statements and traversing their results.
   - Several different types of cursor, e.g. `DictCursor`

Notes

4. Use the cursor to ask the DBMS to execute an SQL statement.
5. Use the cursor to fetch various results:
   - `cursor.rowcount`: the number of rows in the result
   - `cursor.fetchone()`: returns the next row, or `None` if no more rows are available
   - `cursor.fetchall()`: returns a list of all rows (or all remaining rows), or the empty list if there are no remaining rows

   Note: Because we created a `DictCursor`, each row is a dictionary
   - Q: What will the **keys** of the dictionary be?
   - Q: What will the **values** of the dictionary be?
6. Close the cursor and the connection

Improving the program

```
#!/usr/local/bin/python3

from cgitb import enable
enable()

import pymysql as db

print('Content-Type: text/html')
print()

result = ''
try:
    connection = db.connect('localhost', 'userid', 'password', 'database_name')
    cursor = connection.cursor(db.cursors.DictCursor)
    cursor.execute("""SELECT band, gig_date FROM gigs
                      WHERE gig_date >= CURDATE()""")
    result = """<table>
                <tr><th colspan="2">Upcoming Gigs</th></tr>
                <tr><th>Band</th><th>Date</th></tr>"""
    for row in cursor.fetchall():
        result += '<tr><td>%s</td><td>%s</td></tr>' % (row['band'], row['gig_date'])
    result += '</table>'
    cursor.close()
    connection.close()
except db.Error:
    result = '<p>Sorry! We are experiencing problems at the moment. Please call back later.</p>'

print("""
    <!DOCTYPE html>
    <html lang="en">
        <head>
            <meta charset="utf-8" />
            <title>Upcoming Gigs</title>
        </head>
        <body>
            %s
        </body>
    </html>""" % (result))
```

Notes

- Creating a connection may fail
  Q: Why?
- `try`/`except` (simplified):
  - If it successfully creates a connection,
    - it executes the `try` block
    - it skips the `except` block
  - If it fails to create a connection,
    - it skips the rest of the `try` block
    - it executes the `except` block

A different program

```
#!/usr/local/bin/python3

from cgitb import enable
enable()

from cgi import FieldStorage
from html import escape
import pymysql as db

print('Content-Type: text/html')
print()

form_data = FieldStorage()
bandname = ''
result = ''
if len(form_data) != 0:
    try:
        bandname = escape(form_data.getfirst('bandname'))
        connection = db.connect('localhost', 'userid', 'password', 'database_name')
        cursor = connection.cursor(db.cursors.DictCursor)
        cursor.execute("""SELECT gig_date FROM gigs
                          WHERE band = '%s'""" % (bandname))
        result = """<table>
                    <tr><th>Gig Dates</th></tr>"""
        for row in cursor.fetchall():
            result += '<tr><td>%s</td></tr>' % row['gig_date']
        result += '</table>'
        cursor.close()
        connection.close()
    except db.Error:
        result = '<p>Sorry! We are experiencing problems at the moment. Please call back later.</p>

print("""
    <!DOCTYPE html>
    <html lang="en">
        <head>
            <meta charset="utf-8" />
            <title>Gigs by band</title>
        </head>
        <body>
            <form action="band_gigs.py" method="get">
                <label for="bandname">Band: </label>
                <input type="text" name="bandname" vale="%s" size="50" maxlength="50" id="bandname"
                <input type="submit" value="Search for gigs" />
            </form>
            %s
        </body>
    </html>""" % (bandname, result))
```

Notes

- A self-processing page.
- Suppose you type `Belated Tonic` into the form.
  Q: What SQL statement will the cursor execute?

```
cursor.execute("""SELECT gig_date FROM gigs
                  WHERE band = '%s'""" % (bandname))
```

- **Note:** We will improve on this in the next lecture