Databases

RDBMS

• Relational Database Management Systems
• A way of saving and accessing data on persistent (disk) storage.

Why Use an RDBMS

• Data Safety
  – data is immune to program crashes
• Concurrent Access
  – atomic updates via transactions
• Fault Tolerance
  – replicated dbs for instant fail-over on machine/disk crashes
• Data Integrity
  – aids to keep data meaningful
• Scalability
  – can handle small/large quantities of data in a uniform manner
• Reporting
  – easy to write SQL programs to generate arbitrary reports

Three Tier Architecture

Browser  HTTP Server  Database Server
RDBMS Technology

• Client/Server Databases
  – Oracle, Sybase, MySQL, SQLServer

• Personal Databases
  – Access

• Embedded Databases
  – Pointbase

Client/Server Databases

MySQL Python Connector

• Standard SQL database access interface.
• Allows a Python program to issue SQL statements and process the results.
• Defines classes to represent constructs such as database connections, SQL statements, result sets, and database metadata.
API: Connection

```python
import mysql.connector

db_config = {
    'user': 'ece1779',
    'password': 'some_password',
    'host': '127.0.0.1',
    'database': 'ece1779'
}

db = mysql.connector.connect(user=db_config['user'],
                              password=db_config['password'],
                              host=db_config['host'],
                              database=db_config['database'])

db.close()
```

---

API: Executing Queries

- A query can return many rows, each with many attributes
- Steps are
  1. Send query to the database
     ```python
cursor = cnx.cursor()
query = 'SELECT * FROM courses'
cursor.execute(query)
```
  2. Retrieve one row at a time
  3. For each row, retrieve attributes
     ```python
     for row in cursor:
         print(row[0])
     ```

---

Trivial Example

**trivial.py**

```python
from flask import render_template
from app import db

import mysql.connector

@app.route('/trivial', methods=['GET'])
# Display an HTML list of all courses.
def trivial():
    cnx = mysql.connector.connect(user='ece1779',
                                   password='secret',
                                   host='127.0.0.1',
                                   database='ece1779')

cursor = cnx.cursor()
query = 'SELECT * FROM courses'
cursor.execute(query)
view = render_template('trivial.html', title='Courses Table', cursor=cursor)
return view
```
trivial.html

```html
<!DOCTYPE html>
<html>
  <head>
    <title>{{title}}</title>
  </head>
  <body>
    <h1>{{title}}</h1>
    <article>
      <table>
        <thead>
          <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Email</th>
            <th>Date of Birth</th>
            <th>Program of Study</th>
            <th>Course Code</th>
            <th>Course Title</th>
            <th>Time</th>
            <th>Location</th>
            <th>Maximum Enrollment</th>
            <th>Current Enrollment</th>
          </tr>
        </thead>
        <tbody>
          <!-- Table body content here -->
        </tbody>
      </table>
    </article>
  </body>
</html>
```

Database Design

- Create a database for keeping track of university courses, and students registration
- Each course can be offered in multiple sections
- Students register for a specific section
- Student info: id, name, email, date of birth, program of study
  - Section: id, course code, course title, time, location, maximum enrolment, current enrolment

ER Diagram
CRUD Design Pattern

- Common design pattern for single table data manipulation
- Create, Read, Update, Delete (CRUD)

<table>
<thead>
<tr>
<th>URL</th>
<th>Verb</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>/courses/</td>
<td>GET</td>
<td>Display a list of courses.</td>
</tr>
<tr>
<td>/courses/[id]</td>
<td>GET</td>
<td>Display details about a specific course.</td>
</tr>
<tr>
<td>/courses/edit/[id]</td>
<td>GET</td>
<td>Display editable form populated with course data.</td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td>Save the form changes for a particular course.</td>
</tr>
<tr>
<td>/courses/create</td>
<td>GET</td>
<td>Display an empty HTML form that allows users to define a new course.</td>
</tr>
<tr>
<td></td>
<td>POST</td>
<td>Create a new Course and save it.</td>
</tr>
<tr>
<td>/courses/delete/[id]</td>
<td>POST</td>
<td>Deletes the specified course</td>
</tr>
</tbody>
</table>

Transactions

- Definition: A transaction is a collection of DB modifications, which is treated as an atomic DB operation.
  - Transactions ensure that a collection of updates leaves the database in a consistent state (as defined by the application program); all updates take place or none do.
  - A sequence of read and write operations, terminated by a commit or abort

- Definition: Committed
  - A transaction that has completed successfully; once committed, a transaction cannot be undone

- Definition: Aborted
  - A transaction that did not complete normally

- Python Connector: By default in transactional mode: auto commit has been disabled, the method commit must be called explicitly; otherwise, database changes will not be saved.