CS W4111.001
Introduction to Databases
Fall 2022

Computer Science Department
Columbia University
Application Programming
(material not included in exams)
SQL ≠ Programming Language

SQL is not a general purpose programming language

• SQL is tailored for data access and manipulation
• SQL queries are easy to optimize and parallelize
• SQL can’t perform “business logic” that’s often needed

Options:

• Extend existing programming languages to understand SQL natively
• Provide an API between programming languages and DBMS
Several Options

• Fully embed into language: **Embedded SQL**
• Use low-level library with core database calls: **DB API**
• Object-Relational Mapping, **ORM**:  
  • Ruby on Rails, Django, Hibernate, SQLAlchemy, …
  • Defines database-backed classes
  • “Magically” maps between database rows and objects in programming language
  • “Magic” is a double-edged sword; we will **not** use in our class
Embedded SQL

- Host programming language (e.g., Java, C) is extended with special SQL syntax/directives
  Example: `EXEC SQL sql-query`
- Program goes through a preprocessor
- Finally, program is compiled into a program that interacts with the DBMS directly
Embedded SQL

Java + Embedded SQL

Preprocessor

Java + DB library calls

Java Compiler

Executable

DBMS library

... if (user == 'admin') {
    EXEC SQL SELECT * ...
} else {
...

DBMS
Database API: Vendor-Specific or Standardized (ODBC, JDBC)

• Write a program using classes implemented by the DBMS vendor, which implement a standard set of database interfaces
• Pass SQL statements as arguments to functions
• Process SQL statements at runtime, and send to DBMS via a driver provided by the DBMS
What Does a Library Need to Do?

• Provide single interface to possibly multiple DBMS engines
• Connect to a database
• Manage transactions
• Map objects between host language and DBMS
• Manage query results
Library/API Overview

• Library components
• Impedance mismatches
  • Types
  • Result sets
“Engines”

Abstraction for a database engine; attempts to hide DBMS language differences

driver://username:password@host:port/database

from sqlalchemy import create_engine
db1 = create_engine("postgresql://localhost:5432/testdb")

db2 = create_engine("sqlite:///testdb.db")
// note: sqlite has no host name (sqlite:///)

https://docs.sqlalchemy.org/en/14/core/engines.html
“Connections”

Before running queries, need to create a connection with database

- Tells DBMS to allocate resources for connection
- Are relatively expensive to set up, so libraries often cache connections for future use
- Defines scope of a transaction

```
conn1 = db1.connect()
conn2 = db2.connect()
```

Should close connections when done!
conn1.execute("UPDATE TABLE test SET a = 1")
conn1.execute("UPDATE TABLE test SET s = 'smith'")
foo = conn1.execute("SELECT * FROM big_table")

Challenges:
• What is the return type of execute()?  
• Type impedance  
• How to pass data between DBMS and host language?
(Type) Impedance Mismatch

• SQL standard defines mappings between SQL and several languages
• Most libraries can handle common types

<table>
<thead>
<tr>
<th>SQL types</th>
<th>C types</th>
<th>Python types</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR(20)</td>
<td>char[20]</td>
<td>str</td>
</tr>
<tr>
<td>INTEGER</td>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>short</td>
<td>int</td>
</tr>
<tr>
<td>REAL</td>
<td>float</td>
<td>float</td>
</tr>
</tbody>
</table>

What about complex objects (e.g., { x: ‘1’, y: ‘hello’ })?
Query Execution: Avoiding “SQL Injection”

• Pass only “sanitized” values to database

```python
args = ('Dr Seuss', '40')
conn1.execute(
    "INSERT INTO users(name, age) VALUES(%s, %s)",
    args)
```

• Pass in a tuple of query arguments
• DBAPI library will properly escape input values
• Most libraries support this
• Never construct raw SQL strings
(Results) Impedance Mismatch

- SQL relations and results are sets of records
- What is the type of table?
  
  ```python
  table = execute("SELECT * FROM big_table")
  ```

- **Cursor** over result set, similar to an iterator interface
- To have ordering guarantees, use ORDER BY clause in queries
Program

sailor4

cursor

DBMS

sailor1

sailor2

sailor3

sailor4

sailor5

sailor6

sailor7

sailor8

sailor9

sailor10

sailor11
(Results) Impedance Mismatch

- Cursor similar to an iterator (next() calls)

  \[
  \text{cursor} = \text{execute("SELECT * FROM bigtable")}
  \]

- Cursor attributes/methods (logical)

  \[
  \text{rowcount}
  \text{keys()}
  \text{previous()}
  \text{next()}
  \]
Cursor similar to an iterator (next() calls)

cursor = execute("SELECT * FROM bigtable")
cursor.rowcount() # 1000000
cursor.fetchone() # (0, 'foo', …)
for row in cursor: # iterate over the rest
    print row

Actual cursor methods vary depending on implementation
Some Useful Names

- DBMS vendors provide libraries for most languages
- Two heavyweights in enterprise world
  - ODBC: Open DataBase Connectivity (Microsoft defined for Windows libraries)
  - JDBC: Java DataBase Connectivity (Sun developed as set of Java interfaces); java.sql.*, javax.sql.*
What to Understand

- Impedance mismatch
- Different uses of a DBAPI
- Why Embedded SQL is no good
- What good are cursors?

Will use **SQLAlchemy** with Python for Part 3 of Project 1 (for Web Front-End Option), but **without ORM component**