CSCI 403 Database Management

20 – Programming Against the Database

This Lecture

An overview of various approaches to writing software programs that connect to a SQL database.

COMMONALITIES

Things All Software Must Do

- Load necessary libraries
- Establish a connection to the desired DB
- Communicate a query and data to the DB
- (If SELECT) receive and interpret result
- (If modification) commit or rollback changes

Libraries

- Language dependent
- DBMS dependent
- Often adhere to some standard
  - Java -> JDBC
  - Windows -> ODBC
  - Python -> DB-API
  - Etc.

Establishing a Connection

- You do this when you use psql or SQuirreL:
  - Supply hostname (and optionally port), database name, username, password
  - Different ways to supply these for each library
  - URIs increasingly popular

Communicate a Query

We'll see some examples of this in a bit…

Communicate Data

- Something we don't see in query tools like psql: query parameters
  - In software, often want to re-use queries
  - Also, security issues if we write data directly into our queries
    - Lecture on SQL injection attacks coming soon!
  - So, use "prepared" queries
    - Data not included – just placeholders (parameters)
    - Data is passed separately
- Example (Python):

```
query = "SELECT * FROM mines_courses WHERE
course_id = $1";
cursor.execute(query, ("CSCI403",))
```


Interpret Results

- Data is returned as some kind of collection
  - Python list
  - Specialized object (Java ResultSet)
  - Must extract rows/columns you need
- Data from DB has types
  - In dynamic languages (Python, Javascript) these will probably all be strings, but easily converted
  - In statically typed languages (Java) you must conform to types some how – this is a pain

Commit/Rollback

- A topic we've avoided until now: transactions
- A transaction wraps up groups of queries
  - Provides *atomicity*
  - Gives you the choice whether or not to make changes permanent
- Can choose to "autocommit" (the default setting for psql and SQuirreL)
- Otherwise, must commit (to make permanent) or rollback (to undo) transaction
  - This gets more complicated when working at higher-level abstractions!

PARADIGMS

Low Level and High Level

- Software is all about abstractions
  - SQL is an abstraction we use to talk to DB
  - Programming languages abstract common functionality into libraries
- Database connection libraries – lots of choices
  - SQL
  - Functional mapping
  - Object-relational mapping (ORM)

Greater abstraction ↓

SQL Is the Foundation

- At the base level of abstraction is SQL
  - Libraries transmit SQL strings and data to DB
  - Results returned in dynamic structures
  - No lower level access (this may be surprising)
- Other libraries build on top
  - E.g., higher level abstractions translate to SQL first

CS@Mines

Example (Python)

```
connection = …
cursor = connection.cursor()
query = "SELECT * FROM
mines_courses WHERE course_id =
'CSCI403'"
cursor.execute(query)
results = cursor.fetchall()
```

CS@Mines

Functional Mapping

- SQL is not part of (most) programming languages
  - So need some way to communicate SQL
  - Base level is creating string SQL commands – some people find this messy/unintuitive
- Simple SQL can be replaced with function calls
  - Works for many use cases
  - Must fall back to raw SQL for complex queries

CS@Mines

Example (massive.js)

massive.js is a Javascript library for querying PostgreSQL databases.

```
results = db.mines_courses.find(
    { course_id: "CSCI403" }
);
```

This is equivalent to executing:

```
SELECT * FROM mines_courses
WHERE course_id = 'CSCI403';
```

CS@Mines

Object-Relational Mapping

- Object-oriented programming
  - Very popular programming paradigm
  - Outside the scope of this course ☹
- Object-relational "impedance mismatch"
  - Objects store relationships with other objects
  - SQL databases allow ad-hoc relationships
  - This difference causes friction at the interface
- Object-oriented programmers want to deal with data as objects
  - Special purpose "OODBMSes", not very popular/successful
  - ORM tries to treat data in RDBMS as objects

CS@Mines

Example (SqlAlchemy)

SqlAlchemy is a Python ORM; this snippet really doesn't show all that goes into an ORM.

Guest lecture on this topic in a few weeks!

```
session.query(Course).filter(Course
.id == 'CSCI403').all()
```

CS@Mines

Next Time

Programming against the database in Python.

CS@Mines