L6
Application Programming

Eugene Wu
Fall 2015

Topics

Interfacing with applications
- Database APIs (DBAPIS)
- Cursors

Some uncommon SQL-based analyses
- Graph analysis
- Math

SQL != Programming Language

Not a general purpose programming language
- Tailored for data access/manipulation
- Easy to optimize and parallelize
- Can't perform "business logic"

Options
1. Extend SQL, make it Turing Complete
   goes from simple, easy to analyze to complex :(
2. Extend existing languages to understand SQL natively
3. Provide an API between programming languages and DBMSes

Many Database API options

Fully embed into language (embedded SQL)

Low-level library with core database calls (DBAPI)

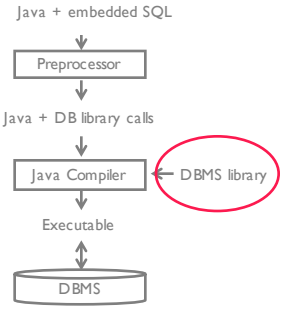
Object-relational mapping (ORM)
- Ruby on rails, django, Hibernate, sqlalchemy, etc
  - define database-backed classes
  - magically maps between database rows & objects
  - magic is a double edged sword

Embedded SQL

Extend host language (python) with SQL syntax
- e.g., EXEC SQL *sql-query*
- goes through a preprocessor

Compiled into program that interacts with DBMS directly

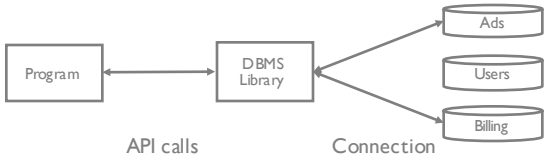
Embedded SQL




```
…
if (user == 'admin'){
    EXEC SQL select * …
} else {
…
```

What does a library need to do?

Single interface to possibly multiple DBMS engines
Connect to a database
Manage transactions (later)
Map objects between host language and DBMS
Manage query results



Overview

Library Components

Impedance Mismatches

1. Types
2. Classes/objects
3. Result sets
4. Functions

Engines

Abstraction for a database engine
tries 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: sqllite has no host name (sqlite:///)
```

http://docs.sqlalchemy.org/en/rel_1_0/core/engines.html

Connections

Before running queries need to create a connection

- Tells DBMS to allocate resources for the connection
- Relatively expensive to set up, libraries often cache connections for future use
- Defines scope of a transaction (later)

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

Should close connections when done! Otherwise resource leak.

Query Execution

```
conn1.execute("update table test set a = 1")
conn1.execute("update table test set s = 'wu'")
```

Query Execution

```
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?
Can we only pass data between DBMS and host language?

(Type) Impedance Mismatch

SQL standard defines mappings between SQL and several languages

Most libraries can deal with common types

SQL types	C types	Python types
CHAR(20)	char[20]	str
INTEGER	int	int
SMALLINT	short	int
REAL	float	float

What about complex objects { x: '1', y: 'hello' }

(Class) Impedance Mismatch

Programming languages usually have classes

Setting an attribute in User should save it

```
user.name  = "Dr Seuss"
user.job   = "writer"
```

```
class User { … }
class Employee extends User { … }
class Salaries {
    Employee worker;
    …
}
```

Object Relational Mappings designed to address this

Query Execution

How to pass values into a query?

```
Users(id int serial, name text)

name = "eugene"

conn1.execute("""
   INSERT INTO users(name)
   VALUES(<what to put here??>)""")
```

Query Execution

How to pass values into a query?

```
Users(id int serial, name text)

name = "eugene"

conn1.execute ("""
   INSERT INTO users(name)
   VALUES('{name}')""".format(name=name))
```

Why is this a *really* bad idea?

Detour: SQL Injections

http://w4111db1.cloudapp.net:8888

code on github:
syllabus/src/injection/


```
@app.route('/', methods=["POST", "GET"])
def index():
  if request.method == "POST":
    name = request.form['name']
    q = "INSERT INTO bad_table(name) VALUES('%s');" % name
    print q
    g.conn.execute(q)
```

Detour: SQL Injections

If we submit:
'); DELETE FROM bad_table; --

Query is
INSERT INTO bad_table(name) VALUES('');
DELETE FROM bad_table; --');

```
@app.route('/', methods=["POST", "GET"])
def index():
  if request.method == "POST":
    name = request.form['name']
    q = "INSERT INTO bad_table(name) VALUES('%s');" % name
    print q
    g.conn.execute(q)
```

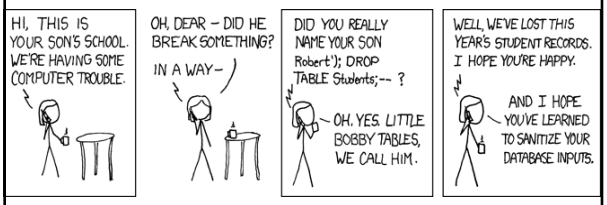
Detour: SQL Injections

Safe implementation

Pass form values as arguments to the `execute()` function
Library sanitizes inputs automatically (and correctly!)

```
@app.route('/safe/', methods=["POST", "GET"])
def safe_index():
  if request.method == "POST":
    name = request.form['name']
    q = "INSERT INTO bad_table(name) VALUES(%s);"
    print q
    g.conn.execute(q, (name,))
```

Detour: SQL Injections



Project: You'll need to protect against simple SQL injections

Query Execution

Pass *sanitized* values to the database

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

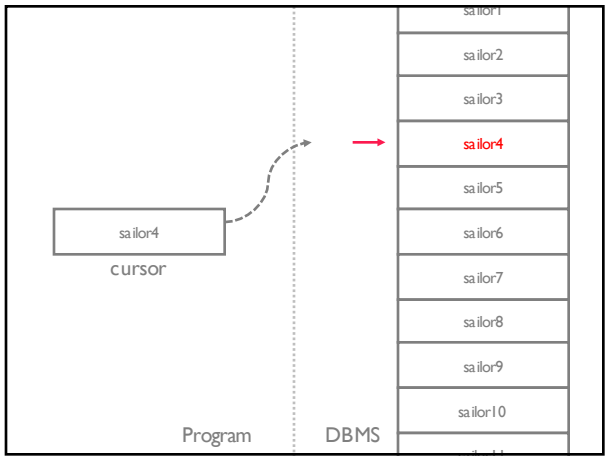
Pass in a tuple of query arguments
DBAPI library will *properly escape* input values
Most libraries support this
Never construct raw SQL strings

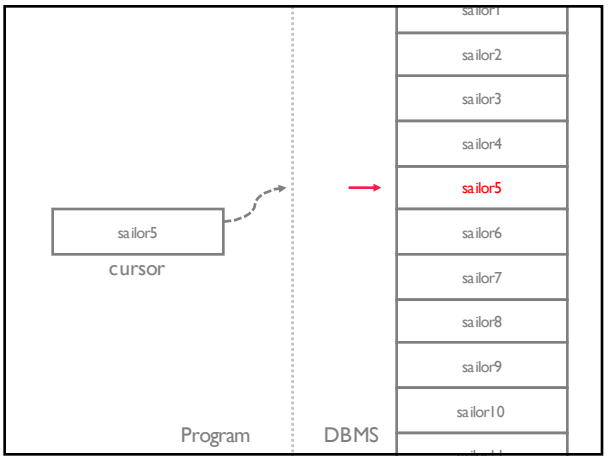
(results) Impedance Mismatch

SQL relations and results are sets of records
What is the type of `table`?

```
table = execute("SELECT * FROM big_table")
```

Cursor over the Result Set
- similar to an iterator interface
- Note: relations are unordered!
  - Cursors have no ordering guarantees
  - Use ORDER BY to ensure an ordering





(results) Impedance Mismatch

Cursor similar to an iterator (next() calls)

```
cursor = execute("SELECT * FROM bigtable")
```

Cursor attributes/methods (logical)

```
rowcount
keys()
previous()
next()
get(idx)
```

(results) Impedance Mismatch

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

(functions) Impedance Mismatch

What about functions?

```
def add_one(val):
  return val + 1

conn1.execute("SELECT add_one(1)")
```

Would need to embed a language runtime into DBMS
- Many DBMSes support runtimes e.g., python
- Can register User Defined Functions (UDFs)

(constraints) Impedance Mismatch

DB-style constraints often as conditionals or exceptions
Constraints often duplicated throughout program

JS
```
age = get_age_input();
if (age > 100 or age < 18)
    show_error("age should be 18 – 100");
```

DBMS
```
CREATE TABLE Users (
    …
    age int CHECK(age >= 18 and age <= 100)
    …
)
```

(constraints) Impedance Mismatch

Some ORMs try to have one place to define constraints

```
class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30, null=True)
```

```
CREATE TABLE myapp_person (
    "id" serial NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30)
);
```

Some Useful Names

DBMS vendors provide libraries for most libraries

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.* (recommended)

Modern Database APIs

DryadLinq, SparkSQL

DBMS executor in same language (dotNET, Spark) as app code
- what happens to language impedance?
- what happens to exception handling?
- what happens to host language functions?

```
val lines = spark.textFile("logfile.log")
val errors = lines.filter(_ startswith "Error")
val msgs = errors.map(_.split("\t")(2))

msgs.filter(_ contains "foo").count()
```

Some Tricky Queries

Lets write some tricky queries
- social graph analysis
  - how many friends?
  - clustering coefficient
- statistics
  - median

Social Network

```
-- A directed friend graph. Store each link once
CREATE TABLE Friends(
    fromID integer,
    toID integer,
    since date,
    PRIMARY KEY (fromID, toID),
    FOREIGN KEY (fromID) REFERENCES Users,
    FOREIGN KEY (toID) REFERENCES Users,
    CHECK (fromID < toID));

-- Return edges in both directions
CREATE VIEW BothFriends AS
    SELECT * FROM Friends
    UNION
    SELECT F.toID, F.fromID, F.since
    FROM Friends F;
```

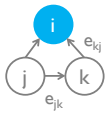
How many friends of friends do I have?

```
SELECT    count(distinct F3.toID)
FROM      BothFriends F1,
          BothFriends F2,
          BothFriends F3
WHERE     F1.toID = F2.fromID AND
          F2.toID = F3.fromID AND
          F1.fromID = <myid>;
```

friends of friends for each user?

```
SELECT    F1.fromID,  count(distinct F3.toID)
FROM      BothFriends F1,
          BothFriends F2,
          BothFriends F3
WHERE     F1.toID = F2.fromID AND
          F2.toID = F3.fromID
GROUP BY  F1.fromID;
```

Clustering Coefficient



$$C_i = 2|\{e_{jk}\}| / k_i(k_i-1)$$

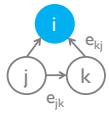
friends that are actually friends — max possible edges between friends

K_i # neighbors of node i

e_{jk} edge between nodes j and k (j < k)

Cliqui-ness: % of your friends that are friends with each other
Clustering coefficient of graph = avg cliqui-ness of all nodes

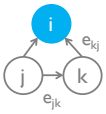
Clustering Coefficient



$$C_i = 2|\{e_{jk}\}| / k_i(k_i-1)$$

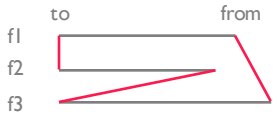
```
CREATE VIEW NEIGHBOR_COUNT AS
SELECT     fromID AS nodeID, count(*) AS friend_cnt
FROM       BothFriends
GROUP BY   nodeID;
```

Clustering Coefficient

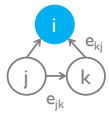


$$C_i = 2|\{e_{jk}\}| / k_i(k_i-1)$$

```
CREATE VIEW TRIANGLES AS
SELECT  F1.toID as root, F1.fromID AS f1, F2.fromID  AS f1
FROM    BothFriends  F1, BothFriends  F2, Friends F3
WHERE   F1.toID = F2.toID          /* j,k both point to i */   AND
        F1.fromID  = F3.fromID     /* j two outgoing edges  */ AND
        F3.toID = F2.fromID        /* j and k are friends  */ ;
```



Clustering Coefficient



$$C_i = 2|\{e_{jk}\}| / k_i(k_i-1)$$

```
CREATE VIEW NEIGHBOR_EDGE_COUNT  AS
SELECT     root, COUNT(*) as cnt
FROM       TRIANGLES
GROUP BY   root;

CREATE VIEW CC_PER_NODE  AS
SELECT  NE.root,
        2.0*NE.cnt  / (N.friend_cnt*(N.friend_cnt-1))   AS CC
FROM    NEIGHBOR_EDGE_COUNT  NE,
        NEIGHBOR_COUNT  N
WHERE   NE.root  = N.nodeID;

SELECT AVG(cc) FROM CC_PER_NODE;
```

Median

Given n values in sorted order, value at idx n/2
- if n is even, can take lower of middle 2

Robust statics compared to avg
- if want avg to equal 0, what fraction of values need to be corrupted?
- if want median to be 0, what fraction?

Breakdown point of a statistic
- crucial if there are outliers
- helps with over-fitting

Median

Given n values in sorted order, value at idx n/2

```
SELECT     T.c
FROM       T
ORDER BY   T.c
LIMIT      1
OFFSET     (SELECT  COUNT(*)/2
            FROM T AS T2)
```

Median

Given n values in sorted order, value at idx n/2

```
SELECT   c AS median
FROM     T
WHERE
   (SELECT  COUNT(*)  FROM T AS T1
    WHERE  T1.c < T.c)
   =
   (SELECT  COUNT(*)  FROM T AS T2
    WHERE  T2.c > T.c);
```

Faster Median

```
SELECT   x.c as median
FROM     T x, T y
GROUP BY x.c
HAVING
   SUM(CASE WHEN y.c <= x.c THEN 1 ELSE 0 END)
    >= (COUNT(*)+1)/2
   AND
   SUM(CASE WHEN y.c >= x.c THEN 1 ELSE 0 END)
    >= (COUNT(*)/2)+1  ;
```

Window Functions

How to run queries over ordered data
- O(n logn)
- Works with even # of items

```
CREATE  VIEW twocounters  AS
(SELECT     x,
            ROW_NUMBER()  OVER (ORDER  BY x ASC)  AS RowAsc,
            ROW_NUMBER()  OVER (ORDER  BY x DESC)  AS RowDesc
 FROM numbers  );


SELECT  AVG(x)
FROM  twocounters
WHERE  RowAsc  IN (RowDesc,  RowDesc  - 1, RowDesc  + 1);
```

Summary

DBAPIs
- Impedance mismatch
- Cursors
- SQL injection

Some hard queries
- More in the HW

Windows are optional material
SQL Injection: only what's in slides