SQL: Programming
Introduction to Databases
CompSci 316 Fall 2015
Announcements (Tue., Oct. 1)

- **Homework #2** due Tuesday
  - Sample solution to be posted by Wednesday noon
  - Submissions later than Wednesday noon receive no credit

- **Midterm in class next Thursday**
  - Open-book, open-notes
  - Same format as the sample midterm (posted on Sakai)

- **Project milestone #1** due next Thursday
Motivation

• Pros and cons of SQL
  • Very high-level, possible to optimize
  • Not intended for general-purpose computation

• Solutions
  • Augment SQL with constructs from general-purpose programming languages
    • E.g.: SQL/PSM
  • Use SQL together with general-purpose programming languages
    • E.g.: Python DB API, JDBC, embedded SQL
  • Extend general-purpose programming languages with SQL-like constructs
    • E.g.: LINQ (Language Integrated Query for .NET)
An “impedance mismatch”

- SQL operates on a set of records at a time
- Typical low-level general-purpose programming languages operate on one record at a time

Solution: cursor

- Open (a result table): position the cursor before the first row
- Get next: move the cursor to the next row and return that row; raise a flag if there is no such row
- Close: clean up and release DBMS resources

Found in virtually every database language/API
  - With slightly different syntaxes

Some support more positioning and movement options, modification at the current position, etc.
Augmenting SQL: SQL/PSM

• PSM = Persistent Stored Modules

• `CREATE PROCEDURE` `proc_name(param_decls)`
  `local_decls`
  `proc_body`;

• `CREATE FUNCTION` `func_name(param_decls)`
  `RETURNS` `return_type`
  `local_decls`
  `func_body`;

• `CALL` `proc_name(params)`;

• Inside procedure body:
  `SET` `variable` `= CALL` `func_name(params)`;
SQL/PSM example

CREATE FUNCTION SetMaxPop(IN newMaxPop FLOAT) RETURNS INT
-- Enforce newMaxPop; return # rows modified.
BEGIN
  DECLARE rowsUpdated INT DEFAULT 0;
  DECLARE thisPop FLOAT;

  -- A cursor to range over all users:
  DECLARE userCursor CURSOR FOR
      SELECT pop FROM User
    FOR UPDATE;

  -- Set a flag upon “not found” exception:
  DECLARE noMoreRows INT DEFAULT 0;
  DECLARE CONTINUE HANDLER FOR NOT FOUND
      SET noMoreRows = 1;

  ... (see next slide) ...

  RETURN rowsUpdated;
END
-- Fetch the first result row:
OPEN userCursor;
FETCH FROM userCursor INTO thisPop;
-- Loop over all result rows:
WHILE noMoreRows <> 1 DO
    IF thisPop > newMaxPop THEN
        -- Enforce newMaxPop:
        UPDATE User SET pop = newMaxPop
        WHERE CURRENT OF userCursor;
        -- Update count:
        SET rowsUpdated = rowsUpdated + 1;
    END IF;
    -- Fetch the next result row:
    FETCH FROM userCursor INTO thisPop;
END WHILE;
CLOSE userCursor;
Other SQL/PSM features

• Assignment using scalar query results
  • SELECT INTO
• Other loop constructs
  • FOR, REPEAT UNTIL, LOOP
• Flow control
  • GOTO
• Exceptions
  • SIGNAL, RESIGNAL

... 

• For more PostgreSQL-specific information, look for “PL/pgSQL” in PostgreSQL documentation
  • Link available from course website (under Help: PostgreSQL Tips)
Interfacing SQL with another language

- **API approach**
  - SQL commands are sent to the DBMS at runtime
  - Examples: Python DB API, JDBC, ODBC (C/C++/VB)
  - These API’s are all based on the SQL/CLI (Call-Level Interface) standard

- **Embedded SQL approach**
  - SQL commands are embedded in application code
  - A precompiler checks these commands at compile-time and converts them into DBMS-specific API calls
  - Examples: embedded SQL for C/C++, SQLJ (for Java)
import psycopg2
conn = psycopg2.connect(dbname='beers')
cur = conn.cursor()

# list all drinkers:
cur.execute('SELECT * FROM Drinker')
for drinker, address in cur:
    print drinker + ' lives at ' + address

# print menu for bars whose name contains “a”:
cur.execute('SELECT * FROM Serves WHERE bar LIKE %s , (%a%,))
for bar, beer, price in cur:
    print bar + ' serves ' + beer\
        + ' at ${:,.2f}'.format(price)
cur.close()
conn.close()
More psycopg2 examples

```sql
# “commit” each change immediately—need to set this option just once at the start of the session
conn.set_session(autocommit=True)
#
bar = raw_input('Enter the bar to update: ').strip()
beer = raw_input('Enter the beer to update: ').strip()
price = float(raw_input('Enter the new price: '))

try:
    cur.execute('''
UPDATE Serves
SET price = %s
WHERE bar = %s AND beer = %s''', (price, bar, beer))
    if cur.rowcount != 1:
        print '{} row(s) updated: correct bar/beer?'.format(cur.rowcount)
except Exception as e:
    print e
```

Exceptions can be thrown (e.g., if positive-price constraint is violated)
Prepared statements: motivation

```python
while True:
    # Input bar, beer, price...
    cur.execute('''
UPDATE Serves
SET price = %s
WHERE bar = %s AND beer = %s''', (price, bar, beer))
    # Check result...
```

- Every time we send an SQL string to the DBMS, it must perform parsing, semantic analysis, optimization, compilation, and finally execution.
- A typical application issues many queries with a small number of patterns (with different parameter values).
- Can we reduce this overhead?
Prepared statements: example

```python
cur.execute('''
PREPARE update_price AS
UPDATE Serves
SET price = $1
WHERE bar = $2 AND beer = $3''')
```

```python
while True:
    # Input bar, beer, price...
    cur.execute('EXECUTE update_price(%s, %s, %s)', (price, bar, beer))

    # Check result...
```

- The DBMS performs parsing, semantic analysis, optimization, and compilation only once, when it “prepares” the statement
- At execution time, the DBMS only needs to check parameter types and validate the compiled plan
- Most other API’s have better support for prepared statements than psycopg2
  - E.g., they would provide a `cur.prepare()` method

See `/opt/dbcourse/examples/psycopg2/ on your VM for a complete code example`
“Exploits of a mom”

The school probably had something like:
```
cur.execute("SELECT * FROM Students " + \
    "WHERE (name = '" + name + "')")
```
where `name` is a string input by user

Called an **SQL injection attack**

Guarding against SQL injection

• Escape certain characters in a user input string, to ensure that it remains a single string
  • E.g., ' ', which would terminate a string in SQL, must be replaced by '' (two single quotes in a row) within the input string

• Luckily, most API’s provide ways to “sanitize” input automatically (if you use them properly)
  • E.g., pass parameter values in psycopg2 through %s’s
If one fails to learn the lesson...

... P.S. To Ashley Madison’s Development Team: You should be embarrassed [sic] for your train wreck of a database (and obviously security), not sanitizing your phone numbers to your database is completely amateur, it’s as if the entire site was made by Comp Sci 1XX students.

— Creators of CheckAshleyMadison.com

Augmenting SQL vs. API

• Pros of augmenting SQL:

• Cons of augmenting SQL:
A brief look at other approaches

• “Embed” SQL in general-purpose programming languages
  • E.g.: embedded SQL

• Extend general-purpose programming languages with SQL-like constructs
  • E.g.: LINQ (Language Integrated Query for .NET)
Embedded SQL

• Embed SQL inside code written in a general-purpose language
  • Special keywords mark code sections containing SQL or variables holding data to be passed to/from SQL
• A “pre-compiler” parses the program and automatically convert the special sections to code with appropriate API calls
  • Pros: more compile-time checking, and potentially more optimization opportunities
• Cons: DBMS-specific:
  • Different pre-compilers for different DBMS vendors
  • Program executable not portable across DBMS’s
  • Difficult for a program to talk to DBMS’s from different vendors
Embedded SQL example (in C)

```c
EXEC SQL BEGIN DECLARE SECTION;
int thisUid; float thisPop;
EXEC SQL END DECLARE SECTION;
EXEC SQL DECLARE ABCMember CURSOR FOR
    SELECT uid, pop FROM User
    WHERE uid IN (SELECT uid FROM Member WHERE gid = 'abc')
    FOR UPDATE;
EXEC SQL OPEN ABCMember;
EXEC SQL WHENEVER NOT FOUND DO break;
while (1) {
    EXEC SQL FETCH ABCMember INTO :thisUid, :thisPop;
    printf("uid %d: current pop is %f\n", thisUid, thisPop);
    printf("Enter new popularity: ");
    scanf("%f", &thisPop);
    EXEC SQL UPDATE User SET pop = :thisPop
        WHERE CURRENT OF ABCMember;
}
EXEC SQL CLOSE ABCMember;
```

Declared variables to be “shared” between the application and DBMS

Specify a handler for NOT FOUND exception
Adding SQL to a language

• Example: LINQ (Language Integrated Query) for Microsoft .NET languages (e.g., C#)

```csharp
int someValue = 5;
var results = from c in someCollection
              let x = someValue * 2
              where c.SomeProperty < x
              select new {c.SomeProperty, c.OtherProperty};
foreach (var result in results) {
    Console.WriteLine(result);
}
```

• Automatic data mapping and query translation

• But syntax may vary for different host languages