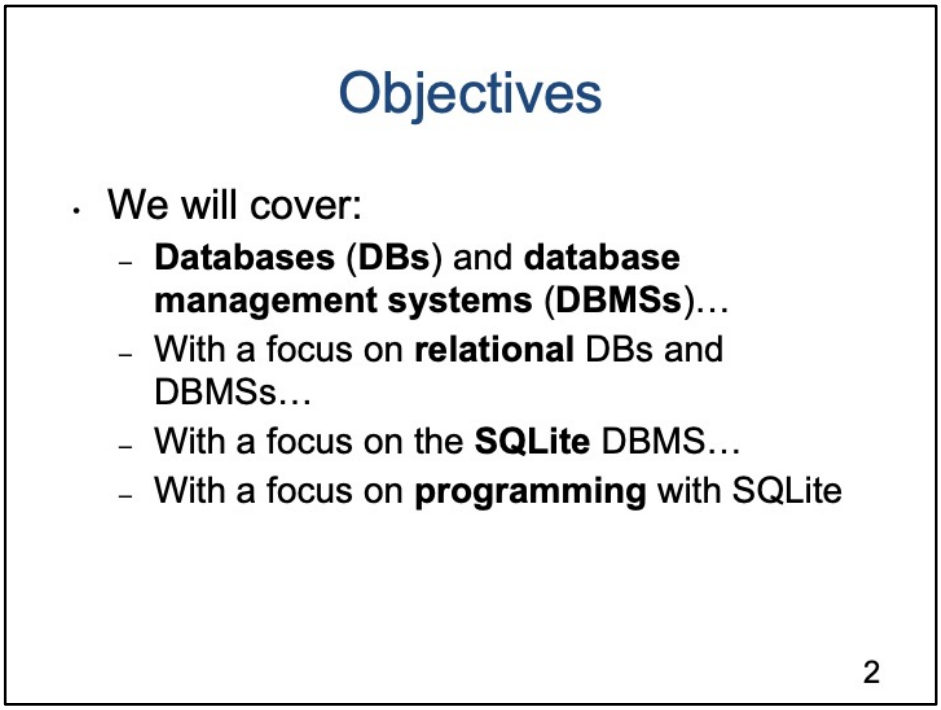
Database Programming (Part 2)

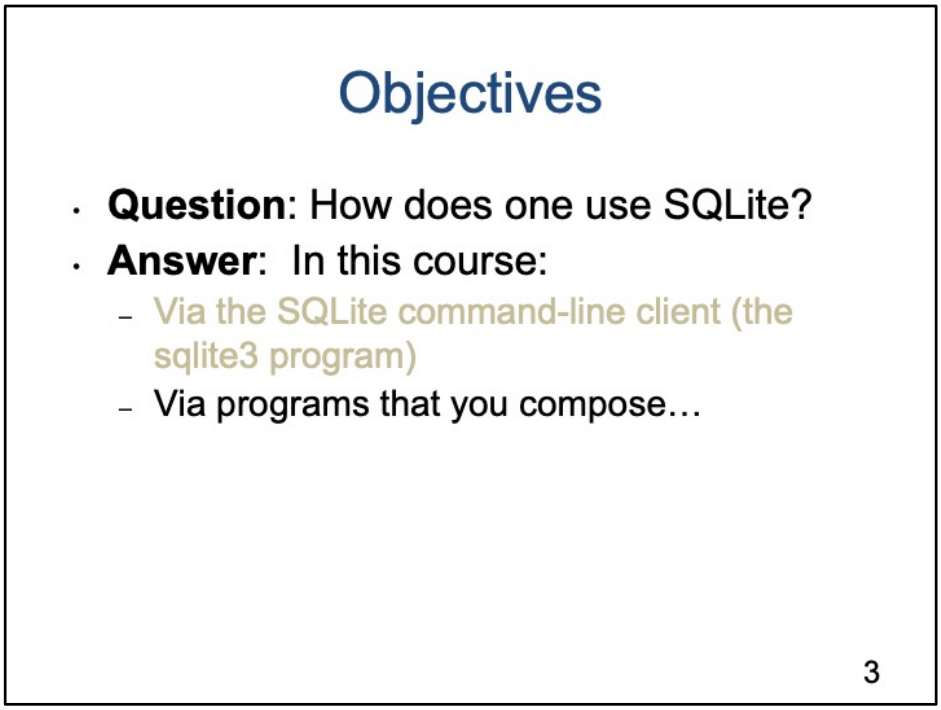
Robert M. Dondero, Ph.D.
Princeton University



Objectives

Continuing from Part 1…

[see slide]



Objectives

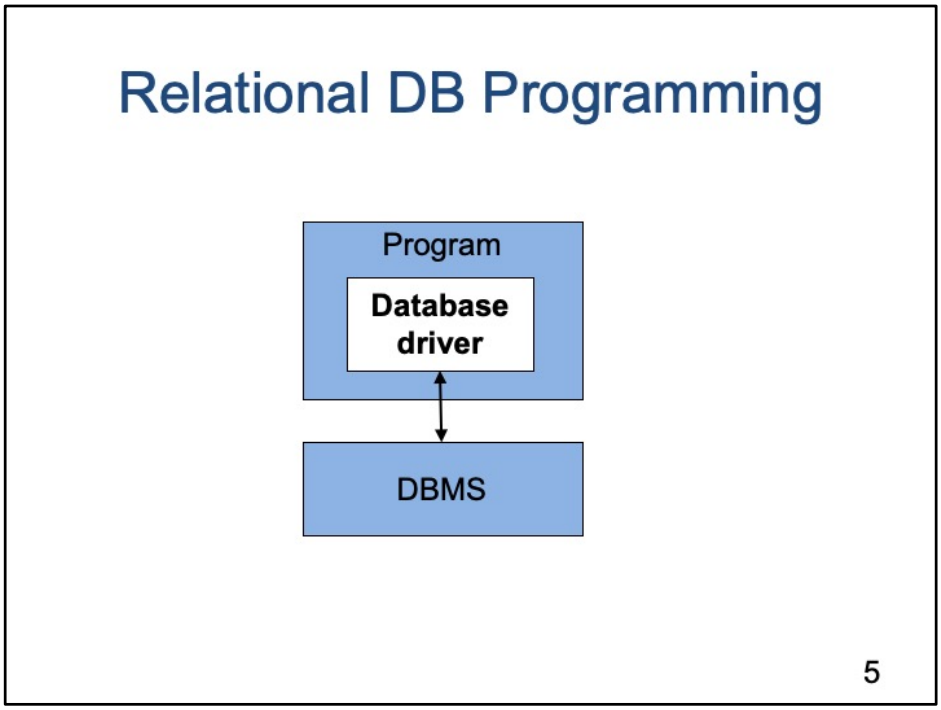
[see slide]

Previously we studied how to use SQLite via the sqlite3 command-line client program

Now we'll study how to use SQLite via (Python) programs that you compose

Agenda

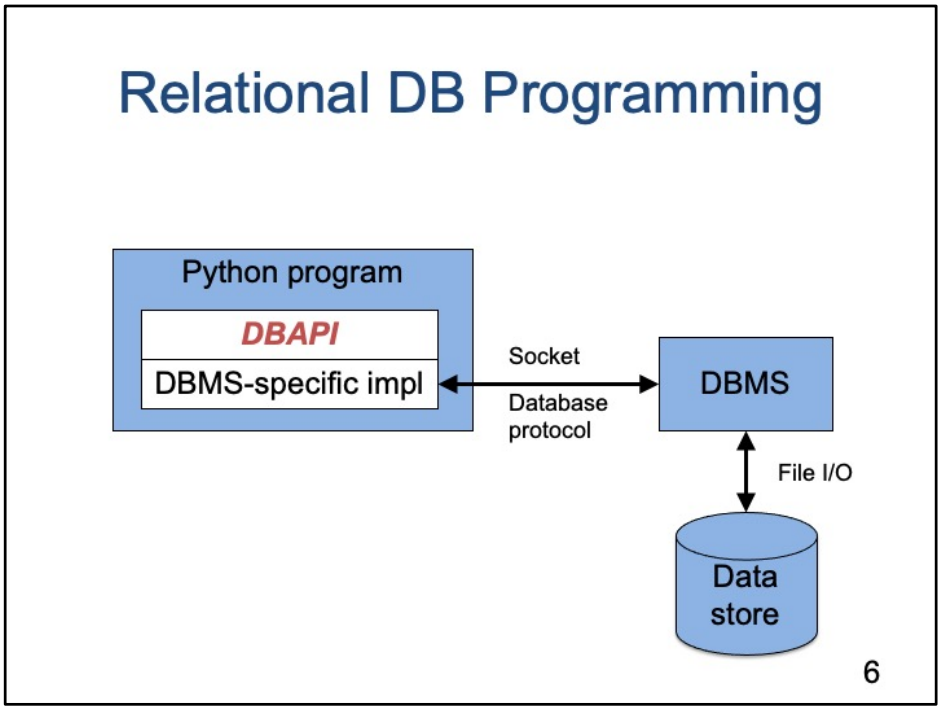
- **Relational DB pgmming**
- Relational DB pgmming: prepared stmts



Relational DB Programming

Any program must use a database driver to communicate with a DBMS

[see slide]



Relational DB Programming

Generally, this is the architecture of a Python program that uses a database

[see slide]

The Python community developed the DBAPI interface (alias API)

Each DBMS community provides an implementation of that interface
- PostgreSQL community provides an implementation that is specific to the PostgreSQL DBMS
- Oracle community provides an implementation that is specific to the Oracle DBMS
- Etc.

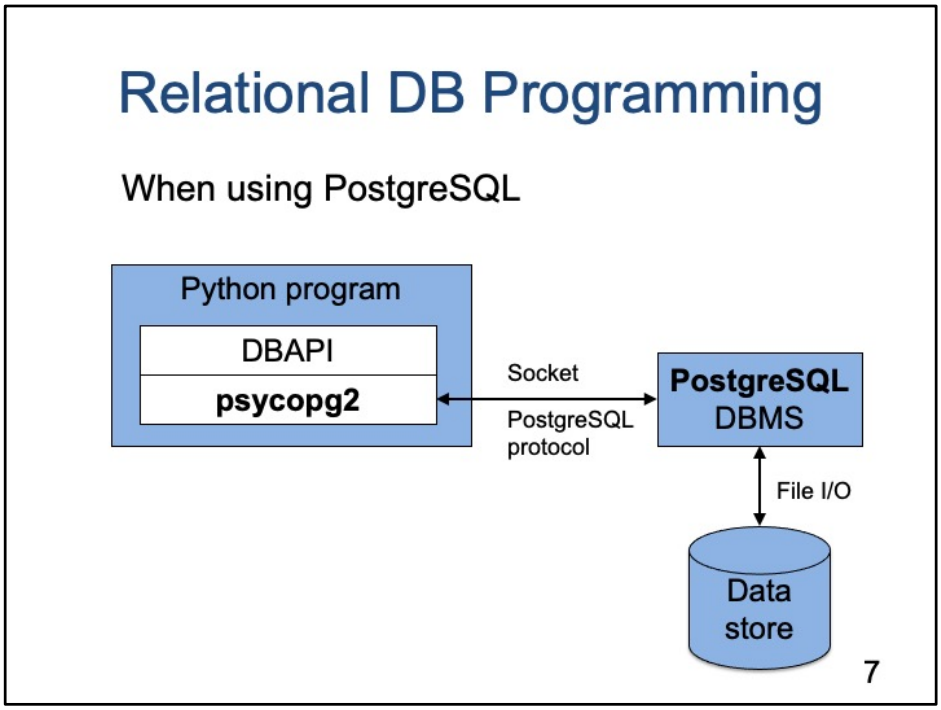
Your Python program communicates with the vendor-specific implementation by calling functions/methods of the DBAPI

The vendor-specific implementation communicates with the DBMS
- Typically over a network via a socket
- Using a protocol that is specific to the DBMS

The DBMS then uses ordinary file I/O to write data to and read data from a persistent

store

Nice architecture
To some extent, Python programs are independent of the particular DBMS that they use



Relational DB Programming

For example…

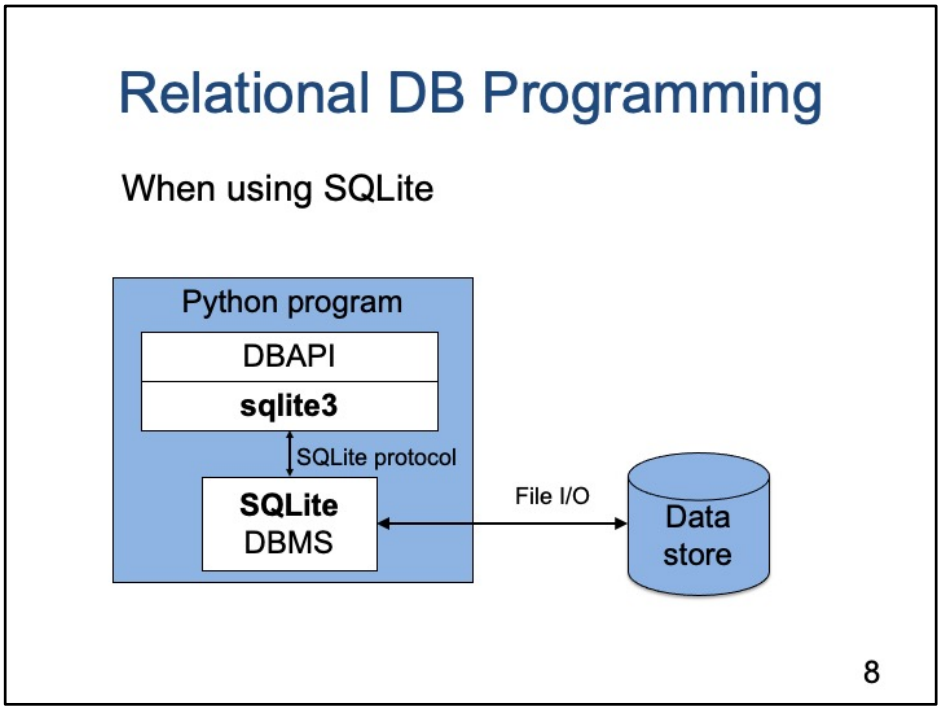
[see slide]

The PostgreSQL community provides an implementation of the DBAPI interface; its name is psycopg2

Your Python program communicates with psycopg2 by calling functions/methods of the DBAPI

psycopg2 communicates with the DBMS
- Over a network via a socket
- Using a protocol that is specific to PostgreSQL

The PosgtreSQL DBMS then uses ordinary file I/O to write data to and read data from a persistent store



Relational DB Programming

In our case…

Recall that SQLite is a module, not a program
It runs in the same process as your Python program does
The SQLite implementation of the DBAPI interface is stored in a module named sqlite3
sqlite3 module typically is bundled with Python itself

[see slide]

Your Python program communicates with sqlite3 by calling functions/methods of the DBAPI

sqlite3 communicates with the DBMS
    Using function/method calls

The SQLite DBMS then uses ordinary file I/O to write data to and read data from a persistent store

Relational DB Programming

- Code structure

Baseline:

```
connection = connect(…)
…
…
connection.close()
```

Better:

```
connection = connect(…)
try:
    …
    …
finally:
    connection.close()
```

Better still:

```
with connect(…) as connection:
    …
    …
```

Relational DB Programming

- Code structure

Baseline:

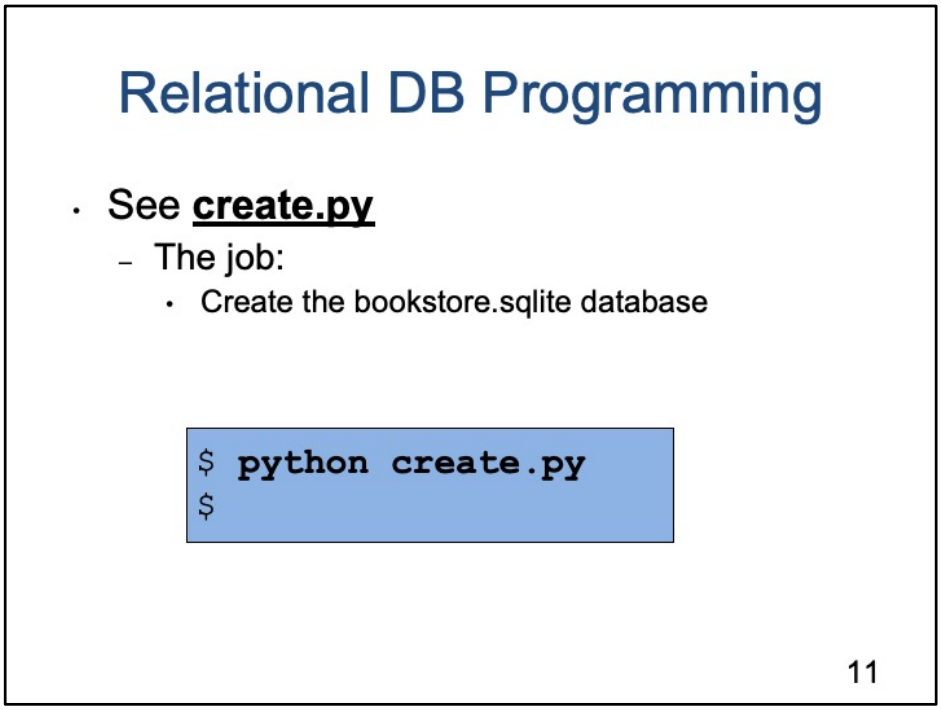
```
cursor = connection.cursor()
…
…
cursor.close()
```

Better:

```
cursor = connection.cursor(…)
try:
    …
    …
finally:
    cursor.close()
```

Better still:

```
from contextlib import closing
…
with closing(connection.cursor()) as cursor:
    …
    …
```



Relational DB Programming

[see slide]

Code notes:

DATABASE_URL = 'file:bookstore.sqlite?mode=rwc'
rwc => read/write/create
The database will be created if it doesn't already exist

connect(DATABASE_URL, uri=True)
Connect to DB

connection.cursor()
Create a ***cursor***
Cursor: "A control structure that enables traversal over the records [rows] in a database [table]. Cursors facilitate subsequent processing in conjunction with the traversal, such as retrieval, addition and removal of database records. The database cursor characteristic of traversal makes cursors akin to the programming language concept of iterator." -- Wikipedia *Cursor (Databases)* article

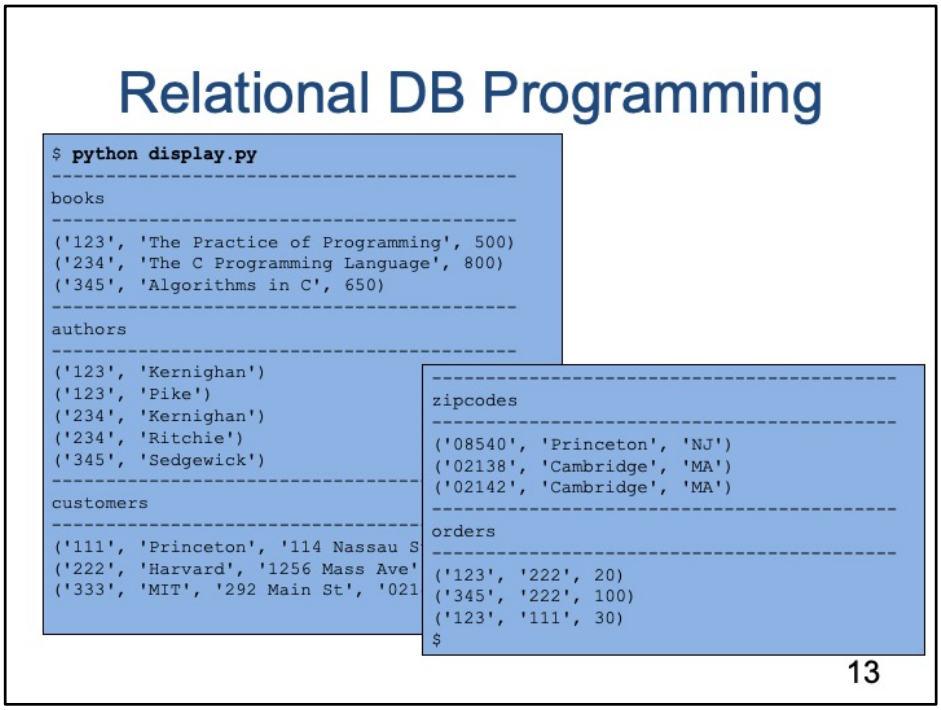
cursor.execute()

Called to execute standard SQL statements
DROP TABLE statements
CREATE TABLE statements
INSERT statements

connection.commit()
Commit changes to the database
(More later)

Relational DB Programming

- See **display.py**
  - The job:
    - Write to stdout the raw contents of the bookstore.sqlite database



Relational DB Programming

[see slide]

Code notes:

DATABASE_URL = 'file:bookstore.sqlite?mode=ro'

ro => read only

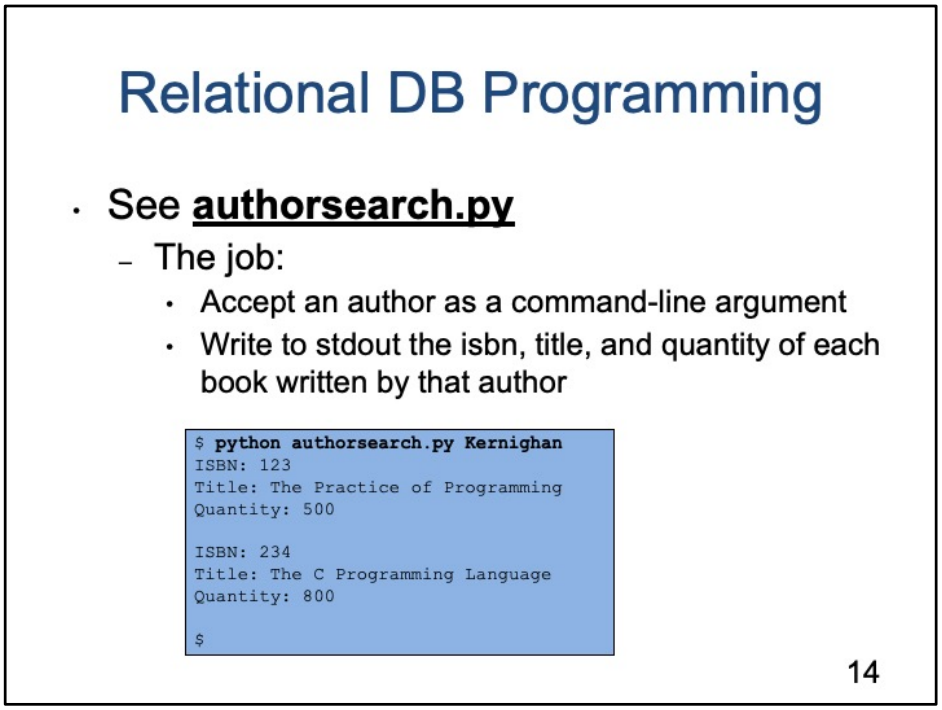
If database file doesn't already exist, driver will throw an exception

Important! "rwc" would create an empty database file; subsequent attempts to SELECT would fail with cryptic exception

Execution of `SELECT` statement

Use of cursor to iterate over result table

Cursor provides each row as tuple



Relational DB Programming

[see slide]

Notes:

Execution of SELECT statement with WHERE clause

SQLite provides result table as a cursor
Within a cursor, SQLite provides each row as a tuple
Use of conversion functions really isn't necessary

Commentary:

Don't execute SELECT * statements from code
In principle, in the relational model columns are unordered
In principle, row[0] would be dangerous

sqlite3 module should provide each row as a dict, not a tuple
Then pgmmer would be required to write row['isbn'] instead of row[0]
Clearer, and safer

Relational DB Programming

- See **order.py**
  - The job:
    - Accept isbn and custid as command-line arguments
    - Increment quantity in appropriate row of ORDERS table

Relational DB Programming

```
$ python display.py
…
----------------------------------------
orders
----------------------------------------
('123', '222', 20)
('345', '222', 100)
('123', '111', 30)
$ python order.py 123 222
$ python display.py
…
----------------------------------------
orders
----------------------------------------
('123', '222', 21)
('345', '222', 100)
('123', '111', 30)
$
```

Agenda

- Relational DB pgmming
- **Relational DB pgmming: prepared stmts**

DB Pgmming: Prepared Stmts

- Problem (via an example)...
- Consider this SQL statement from authorsearch.py:

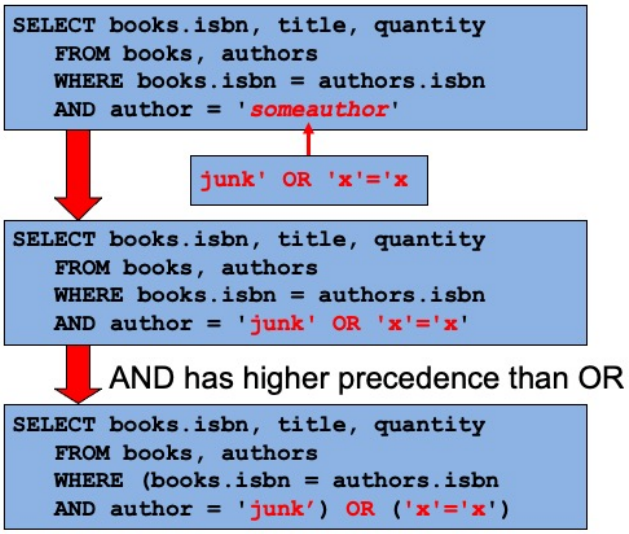
```
SELECT books.isbn, title, quantity
   FROM books, authors
   WHERE books.isbn = authors.isbn
   AND author = 'someauthor'
```

DB Pgmming: Prepared Stmts

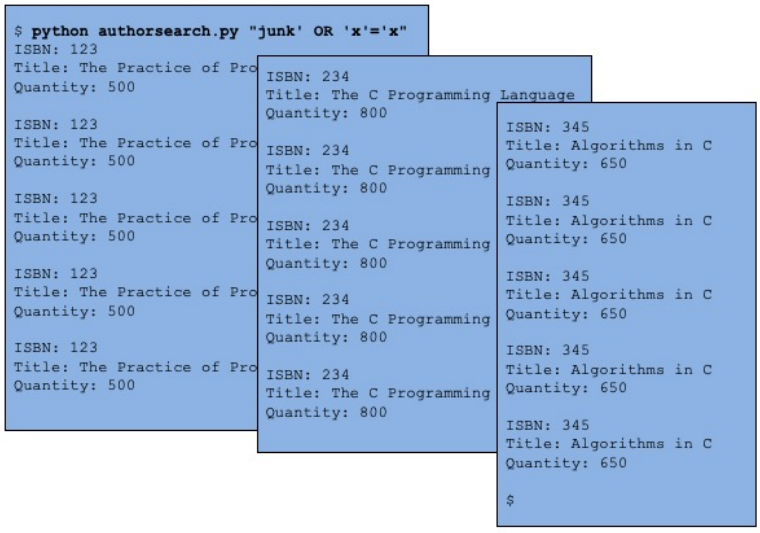
- Suppose (malicious?) user provides this someauthor:

```
junk' OR 'x'='x
```

DB Pgmming: Prepared Stmts



DB Pgmming: Prepared Stmts



DB Pgmming: Prepared Stmts

- Resulting statement:
  - Selects all rows of Cartesian product
  - Accesses data that user is not authorized to access???
  - Causes denial-of-service???
  - Implements a ***SQL injection attack***

DB Pgmming: Prepared Stmts

- The problem (via another example)...
- Consider this SQL statement from order.py:

```
UPDATE orders SET quantity = quantity+1
   WHERE isbn='someisbn' AND custid='somecustid'
```

DB Pgmming: Prepared Stmts

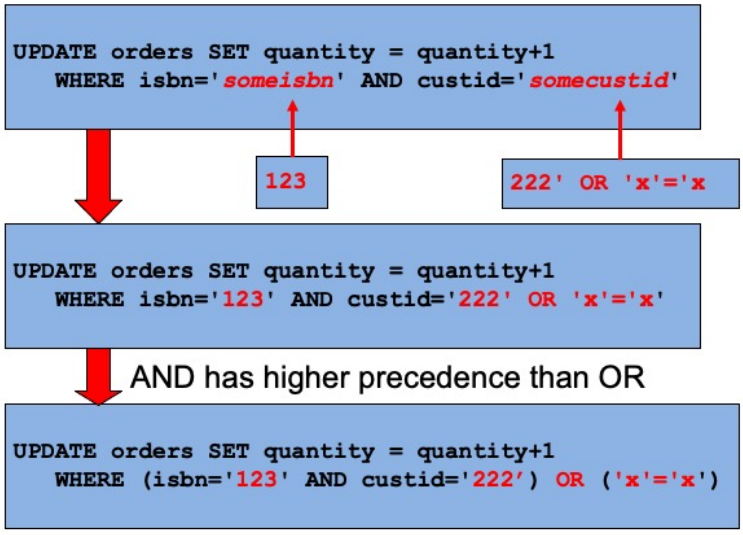
- Suppose (malicious?) user provides this someisbn

```
123
```

- … and this somecustid:

```
222' OR 'x'='x
```

DB Pgmming: Prepared Stmts

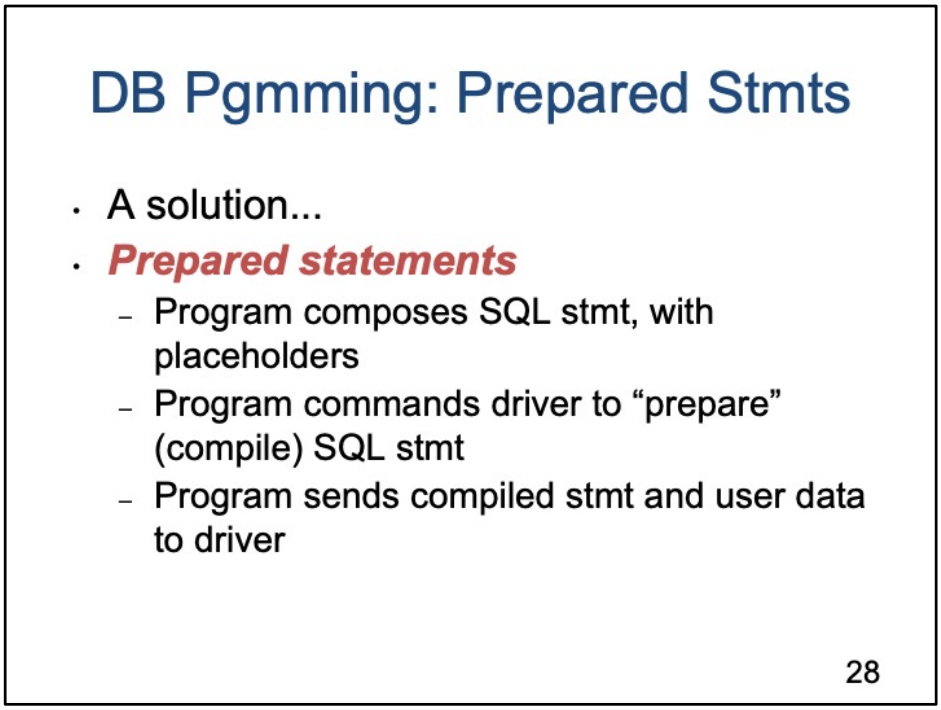


DB Pgmming: Prepared Stmts

```
$ python display.py
…
-----------------------------------------
orders
-----------------------------------------
('123', '222', 20)
('345', '222', 100)
('123', '111', 30)
$ python order.py 123 "222' OR 'x'='x"
$ python display.py
…
-----------------------------------------
orders
-----------------------------------------
('123', '222', 21)
('345', '222', 101)
('123', '111', 31)
```

DB Pgmming: Prepared Stmts

- Resulting statement:
  - Updates **all** rows of ORDERS table
  - Corrupts the DB!!!
  - Implements a ***SQL injection attack***
- For more examples:
  - http://unixwiz.net/techtips/sql-injection.html



Relational DB Programming

A solution...

Prepared statements

Program composes SQL stmt, with placeholders

Program commands driver to “prepare” (compile) SQL stmt
SQL stmt compilation is unrelated to user data
E.g. Presence of quotes or “OR” in user data does not affect compilation

Program sends compiled stmt and user data to driver
Driver replaces placeholders with user data

DB Pgmming: Prepared Stmts

- See **authorsearchprep.py**
  - The job:
    - Same as authorsearch.py, except…
    - Thwarts SQL injection attacks

DB Pgmming: Prepared Stmts

```
$ python authorsearchprep.py Kernighan
ISBN: 123
Title: The Practice of Programming
Quantity: 500

ISBN: 234
Title: The C Programming Language
Quantity: 800

$ python authorsearchprep.py "junk' OR 'x'='x"
$
```

DB Pgmming: Prepared Stmts

- See **orderprep.py**
  - The job:
    - Same as order.py, except…
    - Thwarts SQL injection attacks

DB Pgmming: Prepared Stmts

```
$ python display.py
…
----------------------------------------
orders
----------------------------------------
('123', '222', 20)
('345', '222', 100)
('123', '111', 30)
$ python orderprep.py 123 222
$ python display.py
…
----------------------------------------
orders
----------------------------------------
('123', '222', 21)
('345', '222', 100)
('123', '111', 30)
$
```

DB Pgmming: Prepared Stmts

```
$ python display.py
…
----------------------------------------
orders
----------------------------------------
('123', '222', 20)
('345', '222', 100)
('123', '111', 30)
$ python orderprep.py 123 "junk' OR 'x'='x"
$ python display.py
…
----------------------------------------
orders
----------------------------------------
('123', '222', 20)
('345', '222', 100)
('123', '111', 30)
$
```

Summary

- We have covered:
  - Relational DB pgmming
  - Relational DB pgmming: prepared stmts