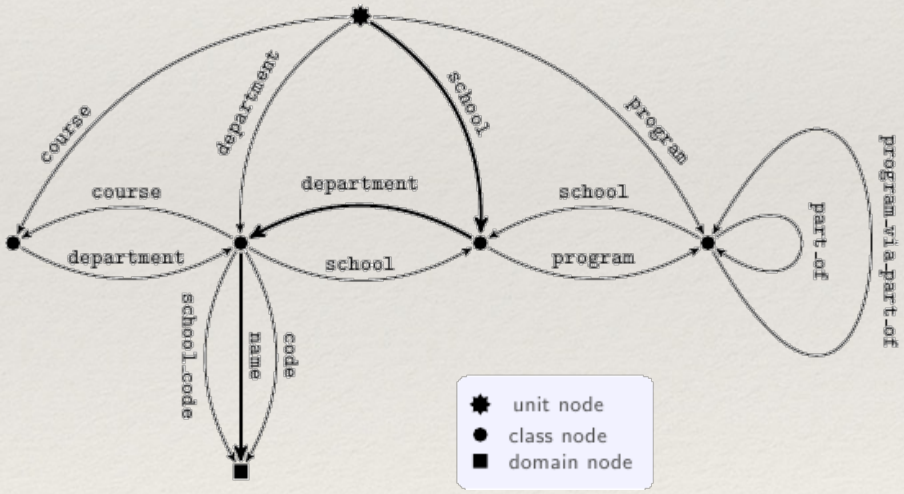
*Advanced Programming Techniques*

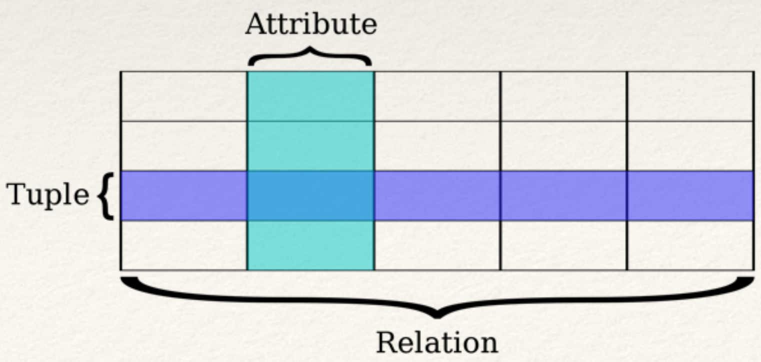
# Database Systems

Christopher Moretti

# History

- Pre-digital libraries
  - Organized by medium, size, shape, content, metadata
- Record managers (1800s-1950s)
  - manually- indexed punched cards
- Navigational DBs (1950s-)
  - records linked with references
- Relational DBs (1970s-)
  - split data into normalized tables
- NoSQL DBs (2000s-)



# Database Definitions

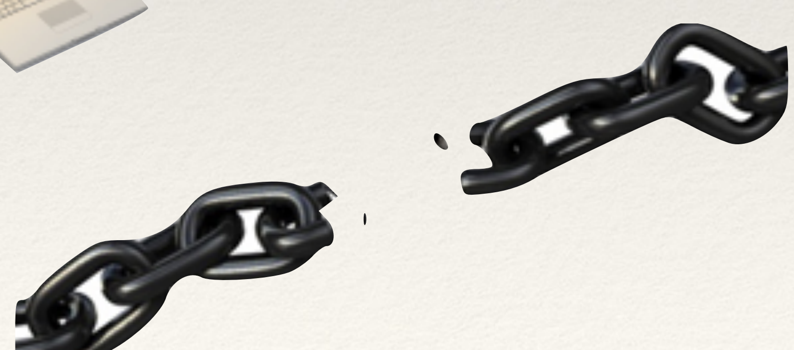
- Database (DB)
  - Structured collection of data
  - Abstract view of data collection
    - Data semantics may not be parallel to data storage
- Database Management System (DBMS)
  - Software infrastructure that constitutes a database
  - Typically client-server architecture

# Schema vs State

- Schema is a description of database
  - structure, types, constraints
  - changes only upon restructuring
- State is a snapshot of the data stored at a given time
  - individual records
  - changes potentially with every query

# Why Databases?

- Centralized control of data
  - Can reduce redundancy, increase efficiency
  - Guarantees important properties:



# Database ABCs, er … CABs


- CRUD - core database record operations
  - Create, Read, Update, Delete
- ACID - core properties of relational db transactions
  - Atomic, Consistent, Isolated, Durable
- BASE - a more relaxed db transaction paradigm
  - Basic Availability, Soft-state, Eventual-consistency

# Navigational Databases

- Hierarchical structure (IBM, early 1960's)
  - Data organized as a tree
  - User follows links from root to find data
  - Queries are biased by the root, link set
- Network structure (CODASYL, late 1960's)
  - Multi-parent as well as multi-child
  - User follows pointers among records to find data

# Relational Databases

- Edgar Cobb (early 1970's)
  - aim was to eliminate all links
- informally: set of tables
  - formally: set of predicates and constraints to define relationships
  - queries are unbiased, but can still be tuned based on anticipated/observed usage

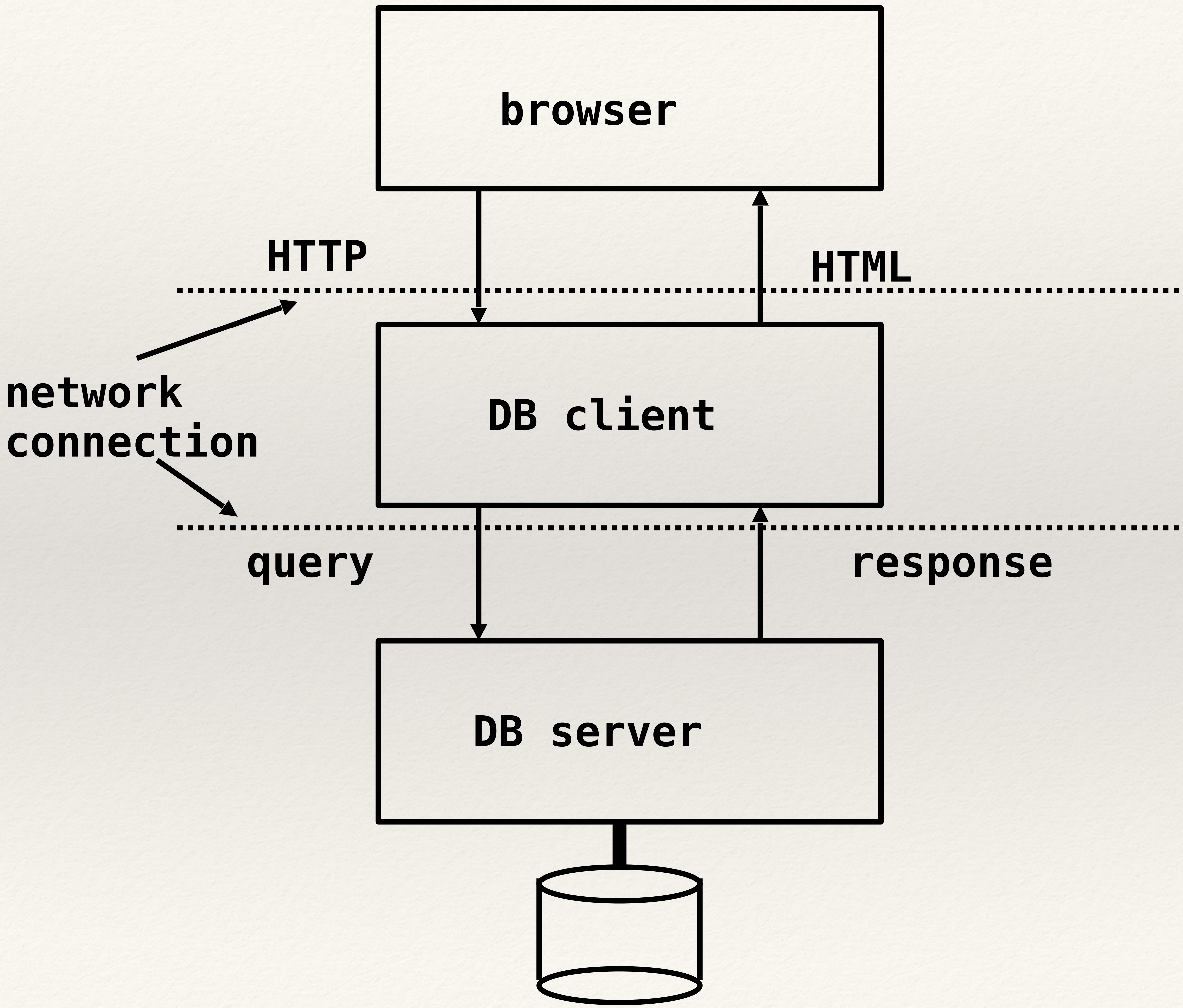
# Practical Options


Don't guarantee ACID
Don't guarantee BASE


MySQL, Oracle, PostgreS, etc.

NoSQL / NOSQL:
non-relational DBs,
document collections,
Key-Value and Column store

# Typical DBMS Architecture



# Relational Schema Example

- Simplest DB  has one table holding all data (e.g. spreadsheet)
- Relational: separate tables "related" by common attributes
  - e.g. custid in custs matches custid in sales
- Schema: content and structure of the tables
  - `books: isbn   title  author price`
  - `custs: custid name   adr`
  - `sales: isbn   custid date   price   qty`
  - `stock: isbn   count`
- Extract info via queries

# Example Books Table

| isbn | title | author | price |
|---|---|---|---|
| 1234 | MySQL | DuBois | 49.95 |
| 4321 | TPOP | K & P | 24.95 |
| 2468 | Ruby | Flanagan | 79.99 |
| 2467 | Java | Flanagan | 89.99 |

# A bit about database design …

Example thanks to Robert M. Dondero, Jr.

# DB0

- BOOKS: isbn, title, authors, quantity
- ORDERS: isbn, custid, custname, street, city, state, zipcode, quantity

**BOOKS**

| isbn | title | authors | quantity |
|---|---|---|---|
| 123 | The Practice of Programming | Kernighan,Pike | 500 |
| 234 | The C Programming Language | Kernighan,Ritchie | 800 |
| 345 | Algorithms in C | Sedgewick | 650 |

**ORDERS**

| isbn | custid | custname | street | city | state | zipcode | quantity |
|---|---|---|---|---|---|---|---|
| 123 | 222 | Harvard | 1256 Mass Ave | Cambridge | MA | 02138 | 20 |
| 345 | 222 | Harvard | 1256 Mass Ave | Cambridge | MA | 02138 | 100 |
| 123 | 111 | Princeton | 114 Nassau St | Princeton | NJ | 08540 | 30 |

- Note lack of atomicity (authors), redundancy (customer info)

# First Normal Form

- Table is 1NF iff each column contains only atomic values

**BOOKS**

| isbn | title | authors | quantity |
|---|---|---|---|
| 123 | The Practice of Programming | Kernighan,Pike | 500 |
| 234 | The C Programming Language | Kernighan,Ritchie | 800 |
| 345 | Algorithms in C | Sedgewick | 650 |

**ORDERS**

| isbn | custid | custname | street | city | state | zipcode | quantity |
|---|---|---|---|---|---|---|---|
| 123 | 222 | Harvard | 1256 Mass Ave | Cambridge | MA | 02138 | 20 |
| 345 | 222 | Harvard | 1256 Mass Ave | Cambridge | MA | 02138 | 100 |
| 123 | 111 | Princeton | 114 Nassau St | Princeton | NJ | 08540 | 30 |

- DB0 is not in First Normal Form

# DB1

- BOOKS: isbn, title, quantity
- AUTHORS: isbn, author
- ORDERS: isbn, custid, custname, street, city, state, zipcode,quantity

**BOOKS**

| isbn | title | quantity |
|---|---|---|
| 123 | The Practice of Programming | 500 |
| 234 | The C Programming Language | 800 |
| 345 | Algorithms in C | 650 |

**AUTHORS**

| isbn | author |
|---|---|
| 123 | Kernighan |
| 123 | Pike |
| 234 | Kernighan |
| 234 | Ritchie |
| 345 | Sedgewick |

**ORDERS**

| isbn | custid | custname | street | city | state | zipcode | quantity |
|---|---|---|---|---|---|---|---|
| 123 | 222 | Harvard | 1256 Mass Ave | Cambridge | MA | 02138 | 20 |
| 345 | 222 | Harvard | 1256 Mass Ave | Cambridge | MA | 02138 | 100 |
| 123 | 111 | Princeton | 114 Nassau St | Princeton | NJ | 08540 | 30 |

- Now's as good as any to think about keys. What are DB1's candidates?

# DB1 Primary Keys

- Choose among candidate keys — in this case, there's only one choice

BOOKS

| isbn | title | quantity |
|---|---|---|
| 123 | The Practice of Programming | 500 |
| 234 | The C Programming Language | 800 |
| 345 | Algorithms in C | 650 |

AUTHORS

| isbn | author |
|---|---|
| 123 | Kernighan |
| 123 | Pike |
| 234 | Kernighan |
| 234 | Ritchie |
| 345 | Sedgewick |

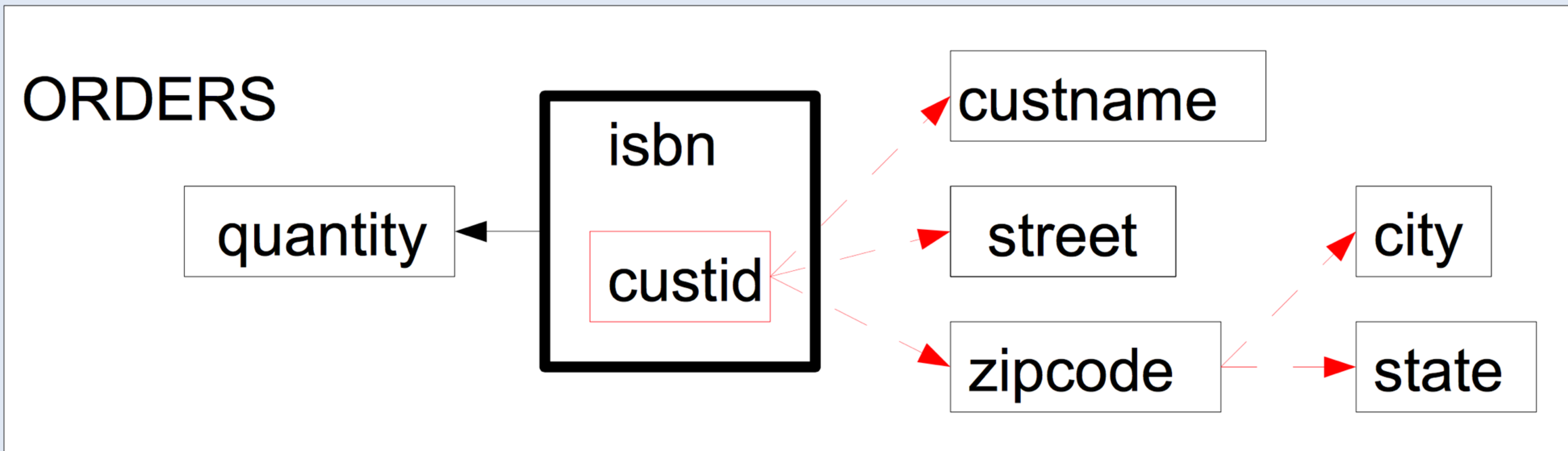
ORDERS

| isbn | custid | custname | street | city | state | zipcode | quantity |
|---|---|---|---|---|---|---|---|
| 123 | 222 | Harvard | 1256 Mass Ave | Cambridge | MA | 02138 | 20 |
| 345 | 222 | Harvard | 1256 Mass Ave | Cambridge | MA | 02138 | 100 |
| 123 | 111 | Princeton | 114 Nassau St | Princeton | NJ | 08540 | 30 |

- Great. That eliminated lack of atomicity. Is there still redundancy?

# Second Normal Form

- Table is 2NF iff 1NF && every non-key is **functionally dependent** on primary key




ORDERS

| isbn | custid | custname | street | city | state | zipcode | quantity |
|---|---|---|---|---|---|---|---|
| 123 | 222 | Harvard | 1256 Mass Ave | Cambridge | MA | 02138 | 20 |
| 345 | 222 | Harvard | 1256 Mass Ave | Cambridge | MA | 02138 | 100 |
| 123 | 111 | Princeton | 114 Nassau St | Princeton | NJ | 08540 | 30 |

- DB1 is not in Second Normal Form

# DB2

- BOOKS: isbn, title, quantity
- AUTHORS: isbn, author
- CUSTOMERS: custid, custname, street, city, state ,zipcode
- ORDERS: isbn, custid, quantity

**BOOKS**

| isbn | title | quantity |
| --- | --- | --- |
| 123 | The Practice of Programming | 500 |
| 234 | The C Programming Language | 800 |
| 345 | Algorithms in C | 650 |

**AUTHORS**

| isbn | author |
| --- | --- |
| 123 | Kernighan |
| 123 | Pike |
| 234 | Kernighan |
| 234 | Ritchie |
| 345 | Sedgewick |

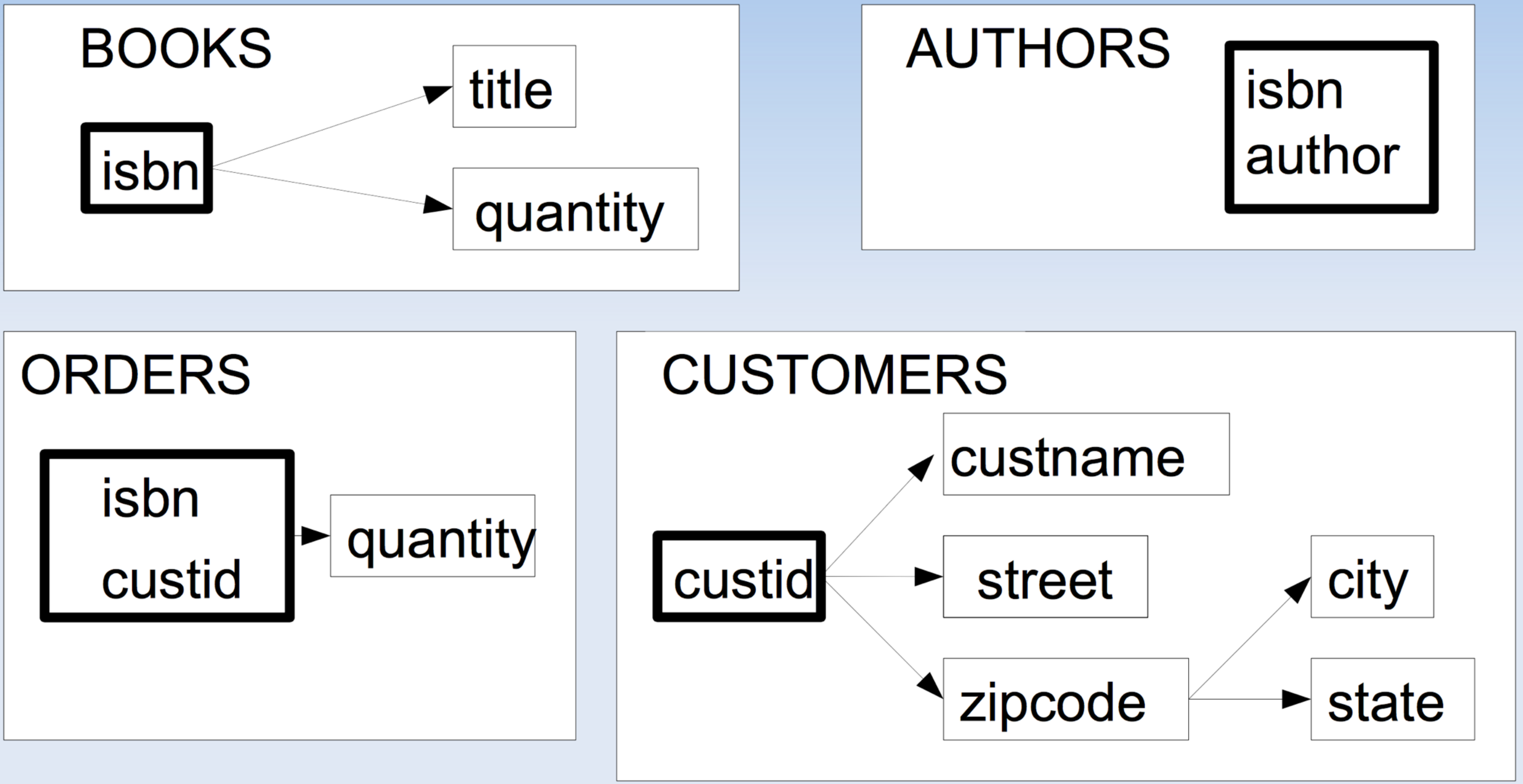
**ORDERS**

| isbn | custid | quantity |
| --- | --- | --- |
| 123 | 222 | 20 |
| 345 | 222 | 100 |
| 123 | 111 | 30 |

**CUSTOMERS**

| custid | custname | street | city | state | zipcode |
| --- | --- | --- | --- | --- | --- |
| 111 | Princeton | 114 Nassau St | Princeton | NJ | 08540 |
| 222 | Harvard | 1256 Mass Ave | Cambridge | MA | 02138 |
| 333 | MIT | 292 Main St | Cambridge | MA | 02142 |

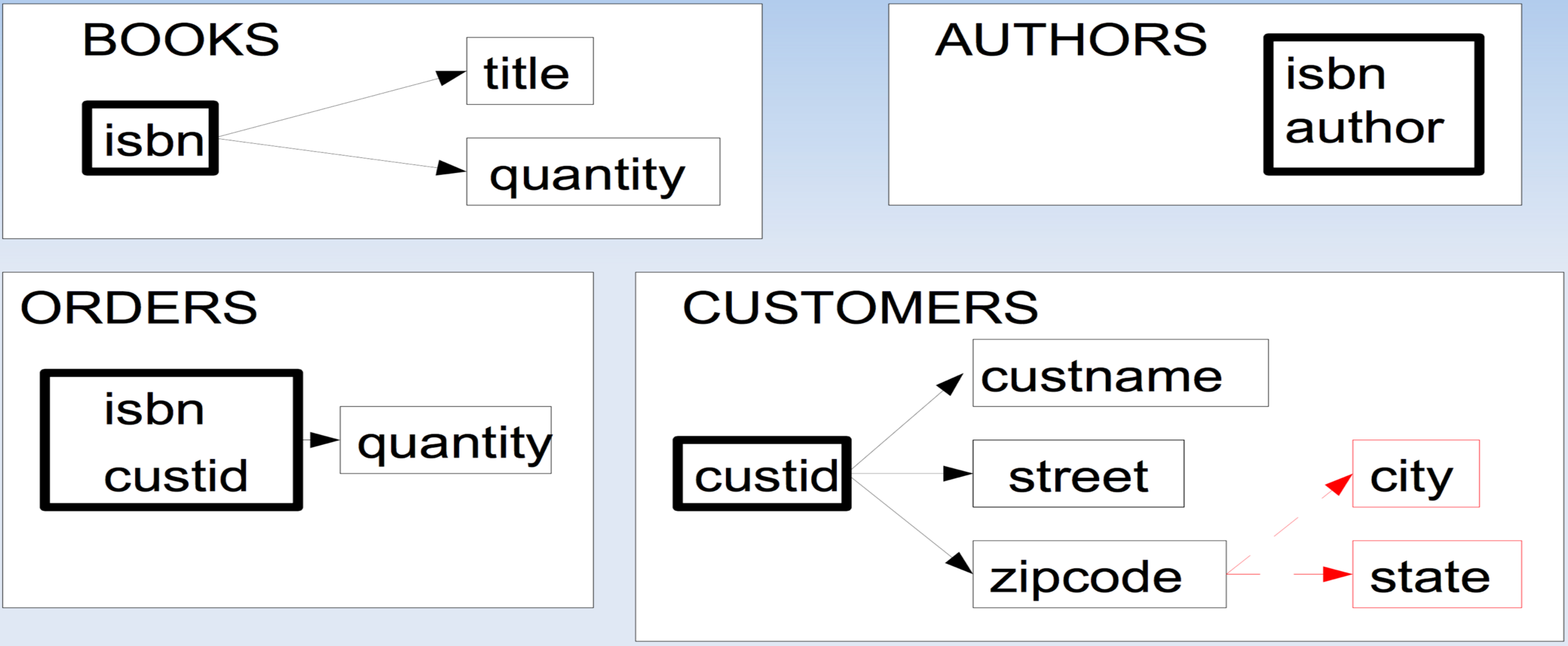
# DB2 is in Second Normal Form




- Great. That eliminated lots of redundancy. But is there still any?

# Third Normal Form

- Table is 3NF iff 2NF && every non-key is **non-transitively** dependent on primary key (not functionally dependent on something else first)




- DB2 is not in Third Normal Form

# DB3

- BOOKS: isbn, title, quantity
- AUTHORS: isbn, author
- CUSTOMERS: custid, custname, street, zipcode
- ZIPCODES: zipcode, city, state
- ORDERS: isbn, custid, quantity

**BOOKS**

| isbn | title | quantity |
|---|---|---|
| 123 | The Practice of Programming | 50 |
| 234 | The C Programming Language | 100 |
| 345 | Algorithms in C | 150 |

**AUTHORS**

| isbn | author |
|---|---|
| 123 | Kernighan |
| 123 | Pike |
| 234 | Kernighan |
| 234 | Ritchie |
| 345 | Sedgewick |

**ORDERS**

| isbn | custid | quantity |
|---|---|---|
| 123 | 222 | 20 |
| 345 | 222 | 100 |
| 123 | 111 | 30 |

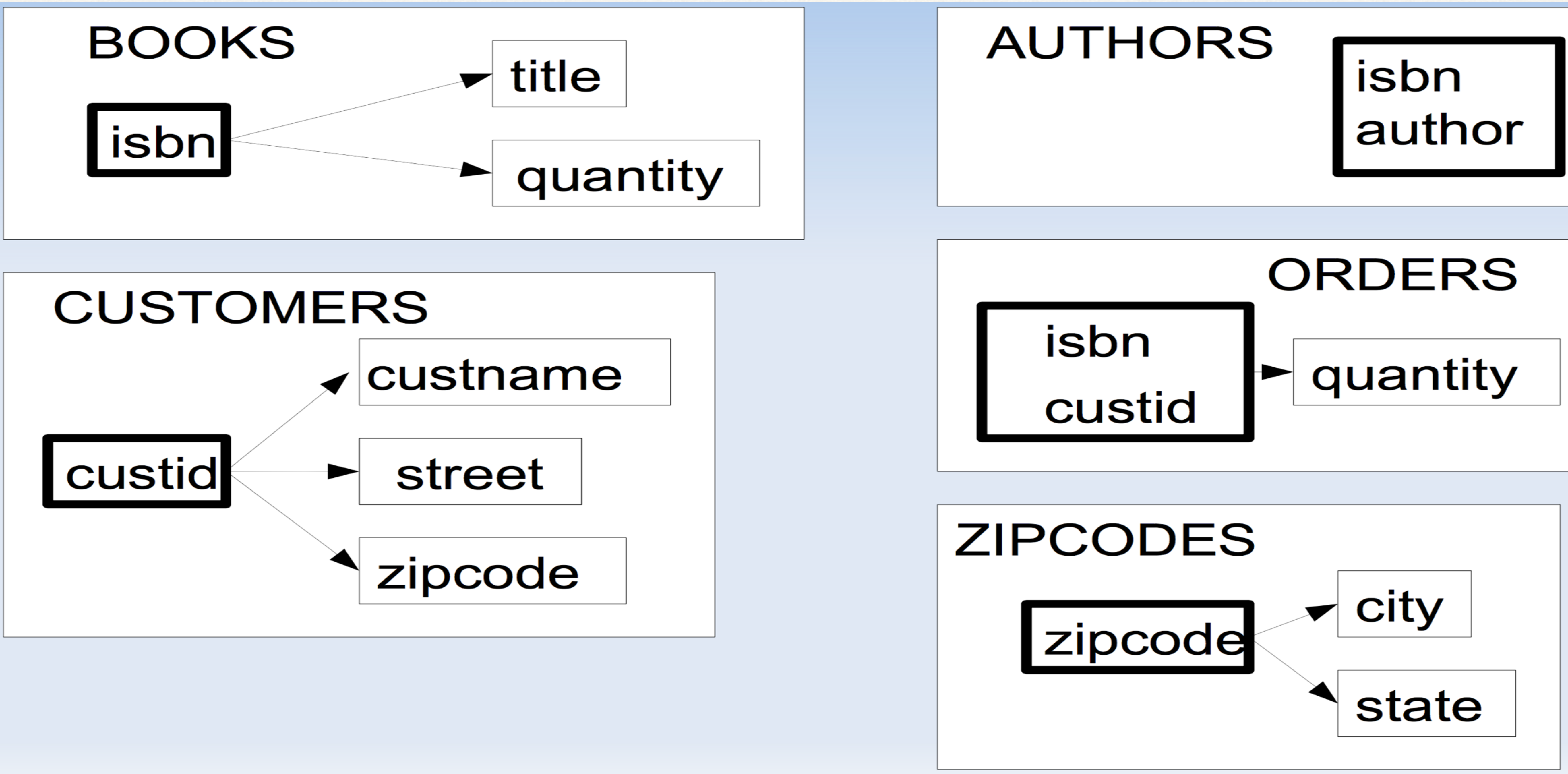
**CUSTOMERS**

| custid | custname | street | zipcode |
|---|---|---|---|
| 111 | Princeton | 114 Nassau St | 08540 |
| 222 | Harvard | 1256 Mass Ave | 02138 |
| 333 | MIT | 292 Main St | 02142 |

**ZIPCODES**

| zipcode | city | state |
|---|---|---|
| 08540 | Princeton | NJ |
| 02138 | Cambridge | MA |
| 02142 | Cambridge | MA |

# DB3 is in Third Normal Form




- And so on … (next would be reduce same authors on different books)

# Structured Query Language (SQL)

- General (`select`) query format:
```
select column-names  from tables  where condition ;
```

- So:
```
select * from books;
select name, adr from custs;
select title, price from books where price > 50;
select * from books where author = "Flanagan";
select author, title from books where author like "F%";
select author, title from books order by author;
select author, count(*) from books group by author;
select author, count(*) as n from books group by author
   order by n desc;
```

- Query result is, itself, a table


> SELECT * FROM users WHERE clue > 0
0 rows returned

# Multiple-table Queries / Joins

- If desired data comes from multiple tables, this implies "joining" the tables together into a new big table from which to produce the output

```
select title, count from books, stock
   where books.isbn = stock.isbn;

select * from books, sales
   where books.isbn = sales.isbn
     and books.author like "F%";

select custs.name, books.title
  from books, custs, sales
    where custs.id = sales.custid
      and sales.isbn = books.isbn;

select price, count(*) as count from books
   where author like 'F%'
     group by author order by count desc;
```

# Beyond “select”

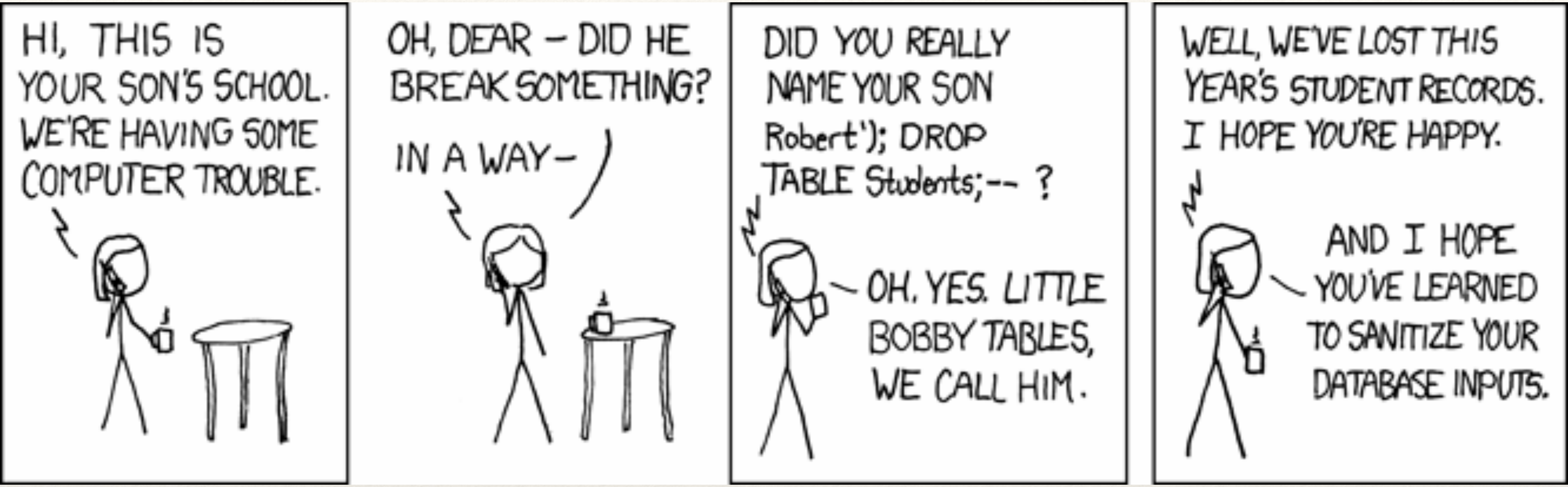
- SQL can, of course, do much more than simply select data from an existing table
  - Warning: different DBs have annoying little inconsistencies about syntax, semantics, performance, but in general garden-variety SQL will work fine.

```
insert into sales values(‘1234','44','2008-03-06','27.95');

update books set price = 99.99 where author = "Flanagan";

delete from books where author = "Singer";
```

http://xkcd.com/327/




Suppose a system does this query:

```
select * from books where author = '{{form_content}}';
```

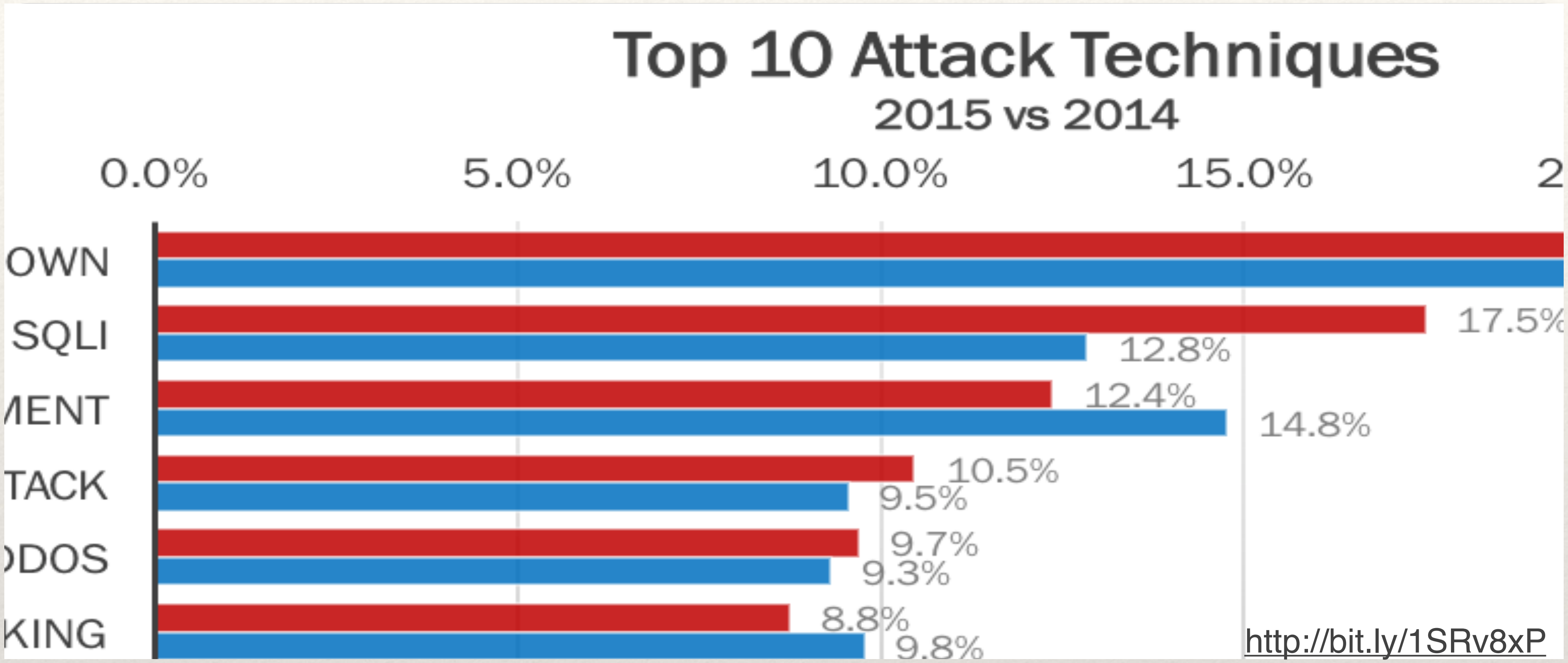
Let's specially construct form_content to do our bidding (a la COS217's buffer overflow):

```
x'; update books set price = $1.00 where author like 'K%'; --'
```

Our construction yields this effective query:

```
select * from books where author = 'x';
update books set price = $1.00 where author like 'K%'; --'
```

# SQL Injection Attacks




```
select * from books where author = ‘’ or ‘1’==‘1’

select * from books where author = ‘x’; drop table books; -- ’

select * from books where author = 'x';
update books set price = $1.00 where author like 'K%'; -- ’
```

# SQL Injection Protection

Prepared statements and parameterized queries:

Details vary by language, DB library: `?` for SQlite, `%s` for MySQL, etc.

```
query='select * from books where author = ?'
mycursor.execute(query, param)
```

Use functions for escaping, e.g.:

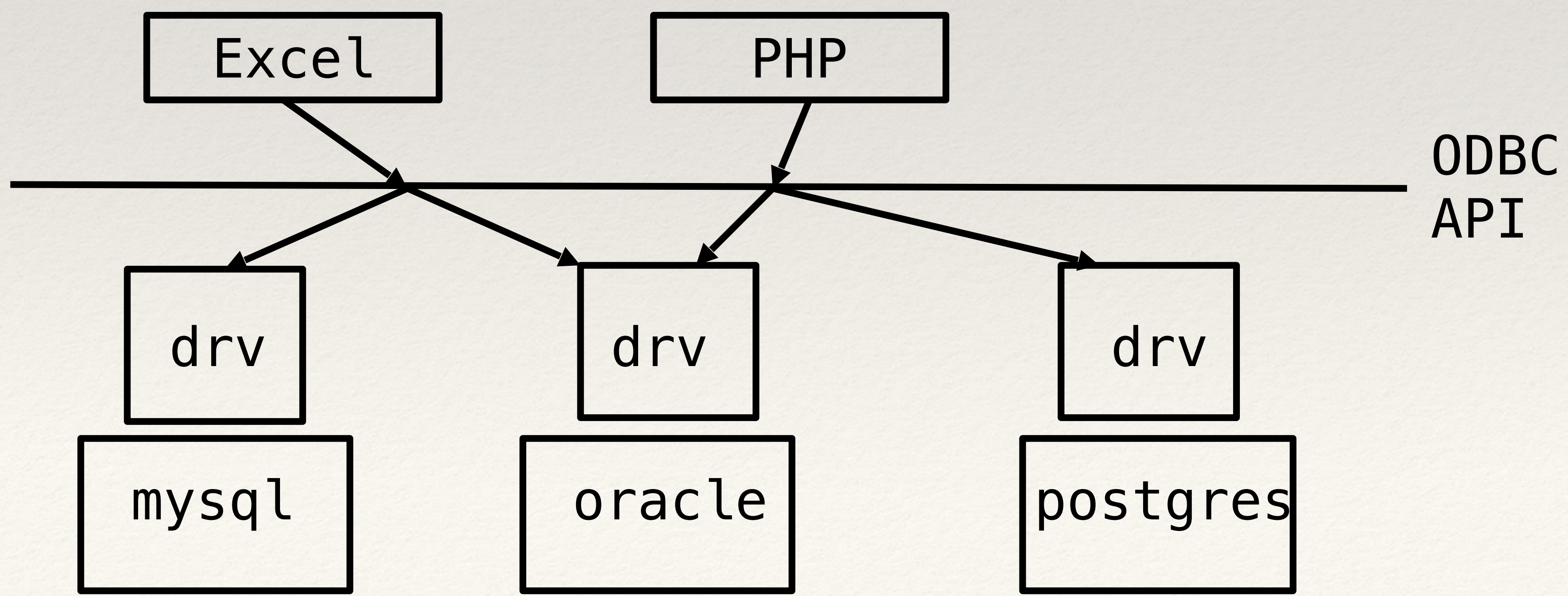
```
mysql_real_escape_string
```

Django and other frameworks generally do this for you.

www.bobby-tables.com

# Database Access in Programs

- There are standard interfaces
  - MS: ODBC (“Open Database Connectivity”)
  - Java JDBC
  - Drivers exist for all major databases, making applications relatively independent of underlying DB



# MySQL Program Interface

- MySQL interface exposes about 50 functions across many languages
  - https://dev.mysql.com/doc/connector-python/en/
  - https://github.com/felixge/node-mysql

```
import sys, fileinput, _mysql
db = _mysql.connect(host="…", user="…", db="…", passwd="…")
db.query("…")
res = db.store_result()
row = res.fetch_row()
while len(row) != 0:
  print row
  row = res.fetch_row()
```

```
import java.sql.*;

public class mysql {
  public static void main(String args[]) {
   String url = “jdbc:mysql://…”;
   try {
      Class.forName("com.mysql.jdbc.Driver");
   } catch(java.lang.ClassNotFoundException e) {
      System.err.print("ClassNotFoundException: " + e.getMessage());
   }
   try {
      Connection con = DriverManager.getConnection(url, “…”, ”…”);
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery("select * from books");
      while (rs.next())
         System.out.println(rs.getString("title") + " "
                                          + rs.getString("author"));
      stmt.close();
      con.close();
   } catch(SQLException ex) {
      System.err.println("SQLException: " + ex.getMessage());
   }
  }
}
```

# MongoDB Program Interface (Flaskr)

```
from pymongo import Connection
db = Connection()['dbfile']
blog = db['blog']

def show_entries():
    entries = [dict(title=cur['title'], text=cur['text'])
                       for cur in blog.find()]
    return render_template('show_entries.html', entries=entries)

def add_entry():
    blog.insert({"title": request.form['title'],
                 "text": request.form['text']}) # BUG: injection?
    return redirect(url_for('show_entries'))
def clear():
    blog.remove()
    return redirect(url_for('show_entries'))
```

http://openmymind.net/2011/3/28/The-Little-MongoDB-Book/