Lecture 9
Databases

Database systems

- database: a structured collection of data
- provides an abstract view of data
  - separated from how it's stored in a file system
  - analogous to how file systems abstract from physical devices

- database management system: software that maintains a database
  -- usually running on a server, responding to client requests
- provides uniform access to information
  - by multiple clients simultaneously
- provides centralized control
- guarantees important properties
  - consistency
  - security
  - integrity
- can reduce redundancy while increasing speed

CRUD: basic data base operations

- **Create**
  - create a brand new record
- **Read**
  - read/ retrieve an existing record
- **Update**
  - change / modify / update all or part of an existing record
- **Delete**
  - guess what

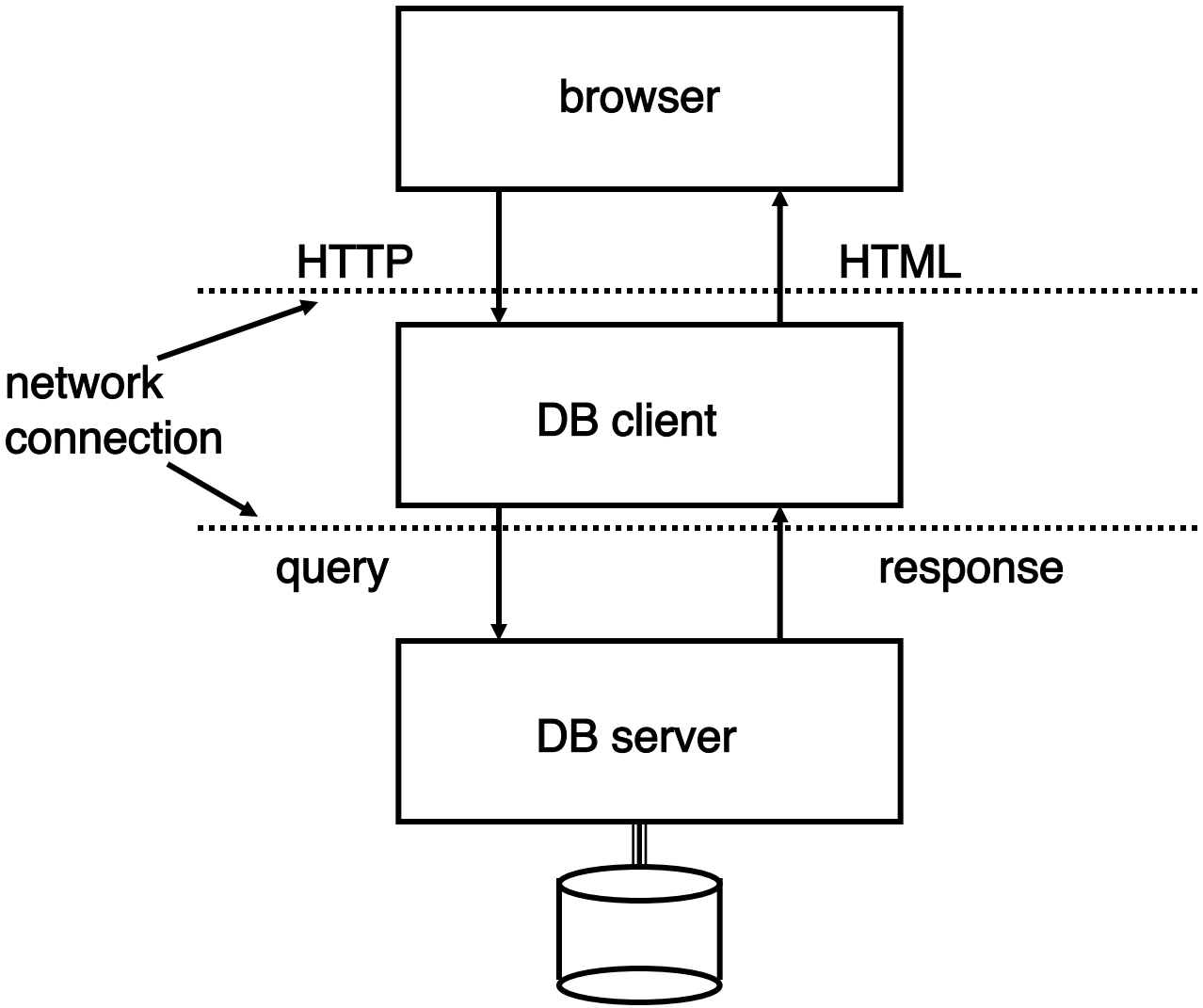
ACID: critical properties of a database system

- **Atomicity**
  - all or nothing: all steps of a transaction are completed
  - no partially completed transactions
- **Consistency**
  - each transaction maintains consistency of whole database
- **Isolation**
  - effects of a transaction not visible to other transactions until committed
- **Durability**
  - changes are permanent, survive system failure
  - consistency guaranteed

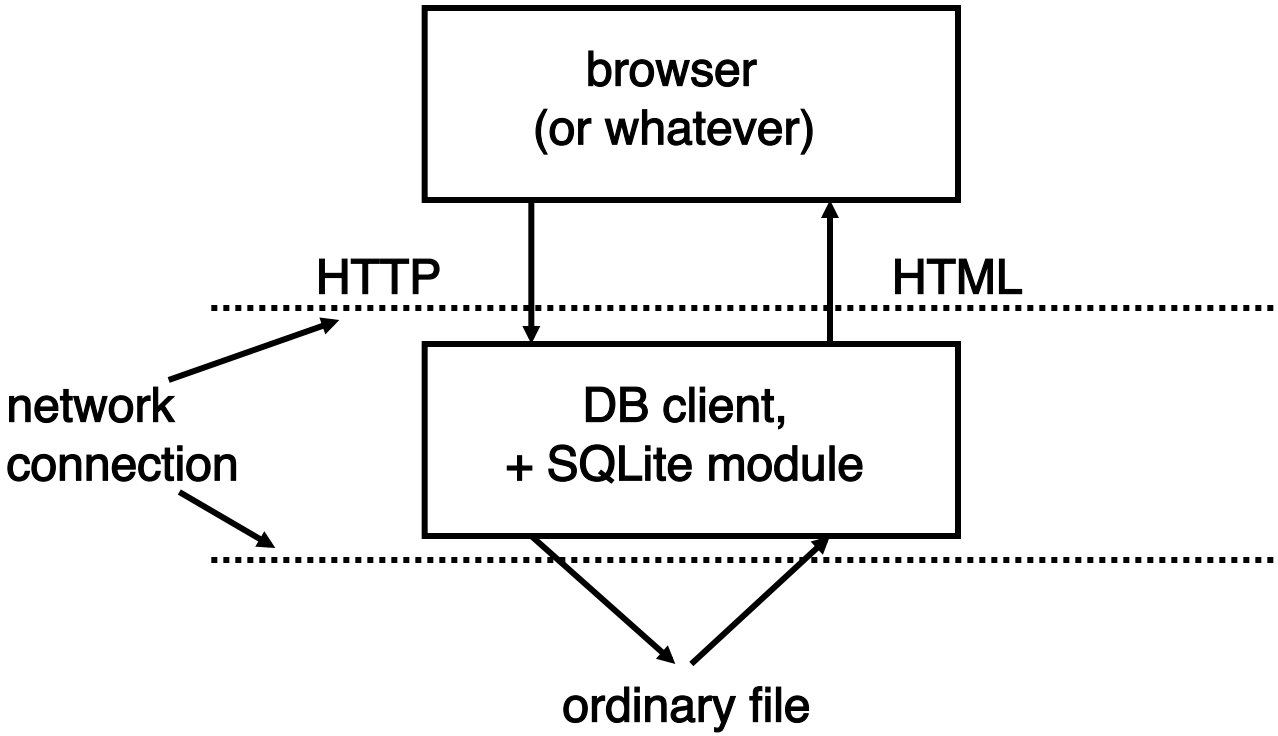
BASE: an alternate consistency model

- **Basically Available**
  - the database appears to work most of the time, but could return a failure if a request arrives when the system is in an inconsistent state
- **Soft state**
  - state of the system could change over time, even without input, because of eventual consistency
- **Eventual consistency**
  - data will eventually become consistent sometime, but not necessarily after each transaction
- **trading consistency for availability can improve scalability**

Typical database system organization



SQLite database system organization



Types of database systems

- ordinary files
  - sometimes ok, but this is not a database except in informal sense
    e.g., doesn't guarantee the ACID properties
- relational / SQL
  - MySQL, MariaDB, SQLite, Postgres, Oracle, DB2, …
  - tables, rows, attributes
  - very structured, organized

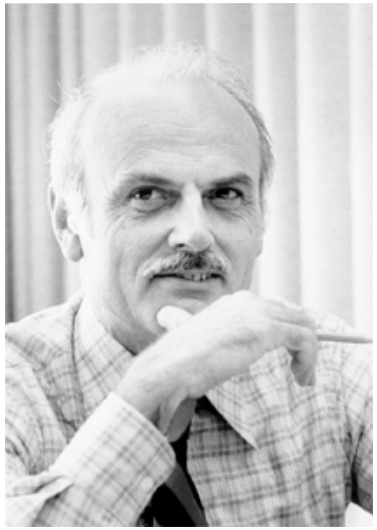
- non-relational (sometimes "no-SQL")
  - MongoDB, CouchDB, ...
  - collections, documents, fields
  - more intuitive, more flexible for some things
  - don't provide all the mechanisms and guarantees of SQL databases
  - may run better on clusters of servers
- key-value & column stores
  - Redis, Berkeley DB, memcached, BigTable, …

Relational Database Management Systems

- e.g.: MySQL, MariaDB, Postgres, SQLite, Oracle, DB2, …
- a database is a collection of tables (formally, "relations")
- each table has a variable number of rows ("tuples")
  - each row is a "record" that contains data
- each table has a fixed number of columns ("attributes")
  - each column is an "attribute" common to all rows

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
2466	Javascript	Flanagan	99.99
1357	Networks	Peterson	105.00
1111	Practical Ethics	Singer	25.00
4320	C Prog Lang	K & R	40.00

Relational model (Edgar Codd, IBM ~1968)



- simplest database has one table holding all the data
  - e.g., Excel spreadsheet
- relational model: data in 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 desired info by queries
- query processing figures out what info comes from what tables, extracts it efficiently

Sample relational database

- books [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
```

- custs [custid, name, adr]

```
11    Brian    Princeton
22    Bob      Princeton
33    Bill     Redmond
44    Bob      Palo Alto
```

- sales [isbn, custid, date, price, qty]

```
4321    11     2019-02-28     45.00      1
2467    22     2019-01-01     60.00     10
2467    11     2019-02-05     57.00      3
4321    33     2019-02-05     45.00      1
```

- stock [isbn, count]

```
1234    100
4321     20
2468      5
2467      0
```

Retrieving data from a single table

- SQL ("Structured Query Language") is the standard language for expressing queries
  - all major database systems support it

- select is the most common command:

select *column-names* **from** *tables* **where** *condition* ;

```
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;
```

- result is a table

Multiple tables and joins

- if desired info comes from multiple tables, this implies a "join" operator to relate data in different tables
  - in effect join makes a big table for later selection

```
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;
```

MySQL, MariaDB (a fork of MySQL)

Michael "Monty" Widenius

- relational database systems

  `www.mysql.com, www.mariadb.com`
- "LAMP" stack
  - Linux
  - Apache
  - MySQL
  - P*: Perl, Python, PHP
- command-line interface:
  - connect to server using command interface

    `mysql -h publicdb -u bwk -p  [or similar]`
  - type commands, read responses

```
show databases;
use bwk;
show tables;
select now(), version(), user();
```

- these commands are specific to MySQL

Creating and loading a table

- create table

```
create table books (
    isbn varchar(15) primary key,
    title varchar(35), author varchar(20),
    price decimal(10,2)
);
```

- insert records

```
insert into books
     values('2464','AWK','Flanagan','89.99');
```

Other statements

- generic SQL
  - ought to be the same for all db systems
  - (though they are not always)

```
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";
```

- MySQL-specific
  - other db's have analogous but different meta statements

```
use bwk;
show tables;
describe books;
drop tables if exists books, custs;
```

SQLite: an alternative (www.sqlite.org)

- small, fast, simple, embeddable
  - no configuration
  - no server
  - single cross-platform database file
- most suitable for
  - embedded devices (cellphones)
  - web sites with modest traffic & rapid processing
    <100K hits/day, 10 msec transaction times
  - ad hoc file system or format replacement
  - internal or temporary databases
- probably not right for
  - large scale client server
  - high volume web sites
  - gigabyte databases
  - high concurrency
- "SQLite is not designed to replace Oracle.
  It is designed to replace fopen()."

Program interfaces to MySQL

- original and basic interface is in C
  - about 50 functions
  - other interfaces build on this

- APIs exist for most other languages
  - Perl, Python, PHP, Ruby, C++, Java, …
  - can use MySQL from Excel, etc., with ODBC module

- basic structure for APIs is

```
db_handle = connect to database
repeat {
    stmt_handle = prepare an SQL statement
    execute (stmt_handle)
    fetch result
} until tired
disconnect (db_handle)
```

SQL injection

- one of the most common attacks on web servers
- malicious SQL statements within queries
  can reveal database contents
  and perhaps modify contents or do other damage

- if text from a form is handed directly to SQL engine,
  the database is vulnerable

```
select * from books
    where author = 'something from a form';

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

Defenses

- always watch out for this
- don't try to roll your own with regular expressions
  - it's too hard to get it right
- use parameterized queries
  - query is processed before insertion

```
cmd = "update people set name=%s where id=%s"
db.execute(cmd, (name, id))
```

- details vary among systems (e.g., %s for MySQL, ? for SQlite)

- Django and other frameworks generally do this for you

- www.unixwiz.net/techtips/sql-injection.html
- www.bobby-tables.com

Non-relational databases (e.g., MongoDB)

- intended for scalability, performance
  - can be distributed across multiple computers more easily than relational
- may not have fixed schema
  - easier to reorganize or augment data organization than with SQL
- no join operator: you have to do it yourself
- may not guarantee ACID properties
  - "eventually consistent" instead
- no standardization
  - different access methods for different db's

MongoDB using mongo commandline interface

```
$ mongod &    # start mongo daemon
$ mongoimport --jsonArray courses.json
$ mongo
show collections
courses
use courses
db.courses.find()
 { "_id" : ObjectId("58d16c4703c287213c5ec5b4"),
  "profs" : [ { "uid" : "960030209", "name" :
  "Christopher L. Hedges" } ], "title" : "...",
   "area" : "EM", ...} }
db.courses.count()
1222
db.courses.find({area: {$eq: "EM"}})
 ...
db.courses.find({area: {$eq: "EM"}}).count()
34
```

Database design

- two different possible table structures:

```
books
   isbn   title   author   price
booktitle, bookauthor, bookprice
   isbn  title
   isbn  author
   isbn  price
```

- they need different SQL queries:

```
select title, author, price from books;
select title, author, price
  from booktitle, bookauthor,bookprice
    where booktitle.isbn = bookauthor.isbn
     and bookauthor.isbn = bookprice.isbn;
```

- most of the program should be independent of the specific table organization
  - shouldn't know or care which one is being used

```
getList(title, author, price)
```