THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

11. Tips & Beyond RDBMS

CSCI 2541 Database Systems & Team Projects

Wood & Chaufournier

Includes slides from Prof. Bhagi Narahari

Upcoming…

Today:

- AWS / VS Code / Flask Tips

Wednesdays:

- Lab + Mentor meetings
- If majority of team members can't attend, you must schedule a separate meeting with your mentor

Monday April 5th: Phase 1 Preliminary Demo

Monday April 12th: Phase 1 DUE

Project / Teamwork

Some starter tasks…

1. ER Diagram and table schema
   - Consider normal forms!
2. Mockup of key pages
   - Make it functional, not pretty!
3. User stories
   - Break project into small steps (features) and build them one by one
4. Team Problems? Let me know ASAP!

AWS

Watch your usage!

- Check for emails with subject [AWS Educate]
- They will warn you if you are running low on credits

Only use the EC2 service

- Don't use AWS RDS - may create expensive VMs!

DO upgrade your VM type to medium

- Shut it down when you aren't using it to save credits

DO use Elastic IP to keep address constant

AWS Alternatives

Or you can do development locally and just use VM for hosting MySQL Database

- Be sure your laptop has Python 3.X
- Not hard to install flask. Learn about venv!

MySQL VM should be fine with micro/small size if you aren't running VS Code remote dev on it

We can group your student AWS accounts together if you want a single shared console

- You also can add keys to ~/.ssh/authorized_keys and allow other students to log into your VM

SSH Keys

Virtual Environments (venv)

Create a new virtual environment

```
# macOS/Linux
# You may need to run sudo apt-get install python3-venv first
python3 -m venv .venv

# Windows
# You can also use py -3 -m venv .venv
python -m venv .venv
```

Load the environment

```
# macOS/Linux
source .venv/bin/activate
# Windows
.venv\Scripts\activate.bat
```

Flask Auto-Reload

Do you save file, kill flask server, start flask server after every change? Try this!

```
FLASK_ENV=development python3 main.py
 * Serving Flask app "app" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 259-217-934
127.0.0.1 - - [19/Mar/2021 15:36:24] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [19/Mar/2021 15:36:24] "GET /favicon.ico HTTP/1.1" 404 -
 * Detected change in '/Users/timwood/flask-data/main.py', reloading
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 259-217-934
127.0.0.1 - - [19/Mar/2021 15:36:35] "GET / HTTP/1.1" 200 -
^C
```

VS Code Tips

Get familiar with this editor! It's pretty great!

Do you know how to?

1. Jump to the definition of a function?
2. Find all the references to a variable?
3. Rename a variable in all places?
4. Select the next occurrence of a highlighted word?
5. Quickly switch between files?
6. Autocomplete code snippets?
7. Share your environment with teammate?
8. Comment out the current line?
9. View two files side-by-side?
10. View the changes you've made to a file since last commit?

Python Debugger

Bold but true? Using a debugger is the single best way to quickly become a better developer and save yourself lots of time

Easy to use:

- Click left of line numbers to set a break point
- Press F5 to start debugger (or use menus)
- Step through code with buttons
- Use Debug Console to view/edit variables
- Sad: difficult to debug how data gets rendered in templates

Use your Database!

The database stores your information NOT your html or python files!

- Don't make an html file for each product!
- Don't have a python array of categories!
- Don't put data that is stored in your database anywhere else in your code!

This is a big change from coding you've done in the past where data, layout, and business were all intertwined

- html - structure of a page
- CSS - appearance of a page
- python - business logic for processing data
- mysql - actual data

Moving Data in Flask

1. Forms: all data from input fields are POSTed to the route which handles the action
   - Great if you have multiple pieces of information that need to be propagated or if data is user-specified
   - Data being sent is hidden from user

2. Query String: portion of the URL after ? is accessible in the route for processing

url.com/?id=123

   - Simple to use but can lead to ugly URLs

3. URL Converters: allow you to use the URL structure to define parameters for Flask

url.com/id/123

   - Cleaner and better than query string in all cases
   - Plan to use this a lot!

Examples

```python
@app.route('/query', methods = ['GET'])
def queryString():
  '''
  This example shows how to parse the query string to pass data to a
  new page (http://0.0.0.0:8080/query?id=456&fname=Tim&lname=Wood)
  '''
  data = {
    "id": request.args.get("id"),
    "fname": request.args.get("fname"),
    "lname": request.args.get("lname")
  }
  return render_template("results.html", data = data)
```

#2

Query String

#3

URL converter

```python
@app.route('/convert/<int:id>/<fname>/<lname>', methods = ['GET'])
def convert(id, fname, lname):
  '''
  This example shows how to use Flask's URL Converter functionality
  to get data from the URL (http://0.0.0.0:8080/789/Tim/Wood)
  '''
  data = {
    "id": id,
    "fname": fname,
    "lname": lname
  }
  return render_template("results.html", data = data)
```

Try it: https://replit.com/@twood02/flaskdata

Why Relational Databases are great… and awful

Relational Databases

Relational databases are the dominant form of database and apply to many data management problems.

- Over $30 billion annual market in 2017.

Relational databases are not the only way.

Other models:

- Hierarchical model
- Object-oriented
- XML
- Graphs
- Key-value stores
- Document models

Relational Database Model

Well developed data model – gained widespread acceptance…eventually!

- Started gaining acceptance in 80's…took off in 90's

Many benefits:

- Data-program independence
- Persistence of data – data 'stays' on storage
- Manage concurrency in transactions – **transaction processing**
- SQL programming became a standard
- Growth of online businesses / e-commerce meant greater demand for recording and reporting data

What is transaction processing?

A user's program may carry out many operations on the data retrieved from the database; but the DBMS is only concerned about what data is read/written from/to the database

A transaction (Xact) is the DBMS's abstract view of the user program: sequence of Read/Write to DB

- Ex: Withdraw from bank account: update balance in SQL

Concurrent execution of user programs essential for good performance.

- Keep CPU humming when disk IO takes place.
  - Recall your nightmares from CS2461 about memory hierarchy !!

Concurrency ….

Users submit Xacts; assume each Xact executes by itself

- Concurrency achieved by DBMS which interleaves actions (read/write of DB objects) of different Xacts
- Each Xact must leave the DB in a consistent state (if DB is consistent when Xact begins)

How to interleave operations from different Xacts (programs) which may share the same data

- Ex: Two (or more) students registering for same course

What happens if system crashes – how to recover to a consistent state?

Big idea: ACID Properties in RDBMs

Atomicity – a Xact either fails or fully succeeds

Consistency – a Xact won't break consistency

Isolated – One Xact cannot impact another

Durability – Data will be persisted if a Xact completes

What properties are important for Xacts?

Big idea: ACID Properties in RDBMs

Atomicty: all actions in Xact happen or none happen

Consistency: if each Xact is consistent, and DB starts in consistent state then it ends consistent

Isolation: Execution of one Xact isolated from others

Durability: if a Xact commits (completes), its effects persist

Meeting the **ACID** Test:

- Concurrency controller guarantees **consistency** and **isolation**
- Logging & recovery for **atomicity** and **durability**

Concurrency Control..How? Locks!

Conflict occurs when two Xacts try to access the same data item

Yay DBMS!

Associate a "**lock**" for each shared data item

- Similar to mutual exclusion (MUTEX)
- To access a data item, check if it is unlocked else wait
- Need to worry about the type of operation: Read or Write
  - Leads to Lock Modes: Shared Lock(S) for Reads only and Exclusive Lock(X) for Writes
- Providing both consistency and performance is hard!

You'll learn more in OS

Recoverability: Logging

Record the operations of each transaction into a log

- Only consider a transaction complete if a "commit" operation is appended to the log
- After a commit, we can update the actual data file

If system crashes, read from log file to rollback to a consistent state

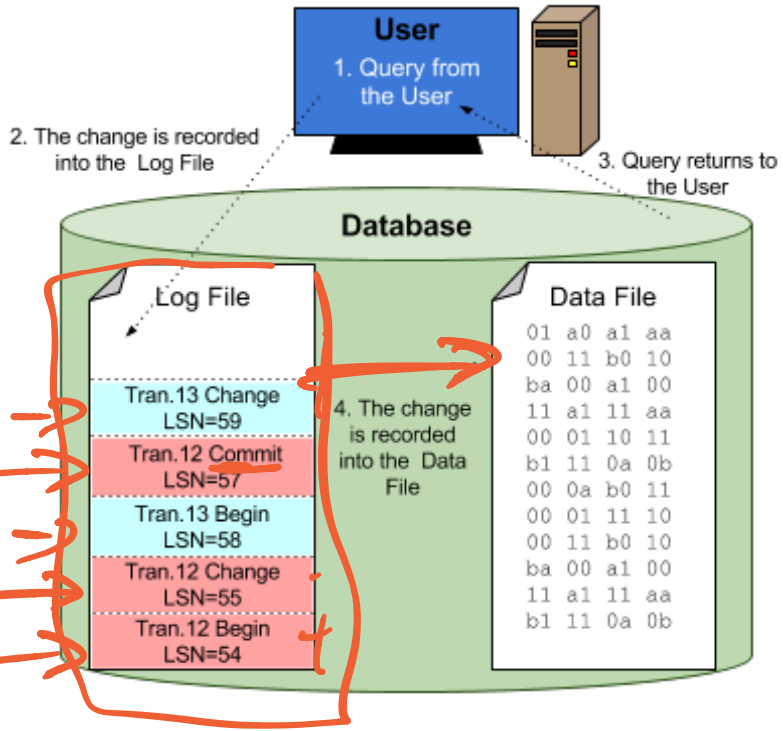


image from https://sqlbak.com/academy/transaction-log

So why do we need something other than Relational DBs?

Database application trends?

Data trends?

Any guesses for how data or applications have been changing in last 10 years?

How to store a Customer…?

In 1990 FName, LName, Addr, Phone, Sex ↑ M F

In 2000 + Email

In 2005 + Cell phone

In 2020 + twitter + Gender

For each change:

- ALTER TABLE Customer… add columns
- Take DB offline, change schema, repopulate DB, fix any inconsistencies…

Instead of adding Columns…

How could we add new information such as mobile phone to our DB without adding columns to an existing table?

Instead of adding Columns…

Could create separate tables and use Joins to combine them

- Customer JOIN Phone JOIN MobilePhone JOIN Gender JOIN Email JOIN Twitter JOIN Instagram JOIN …

But doing lots of joins is expensive and messy

- Lots of fields may be NULL, need to be careful about consistency

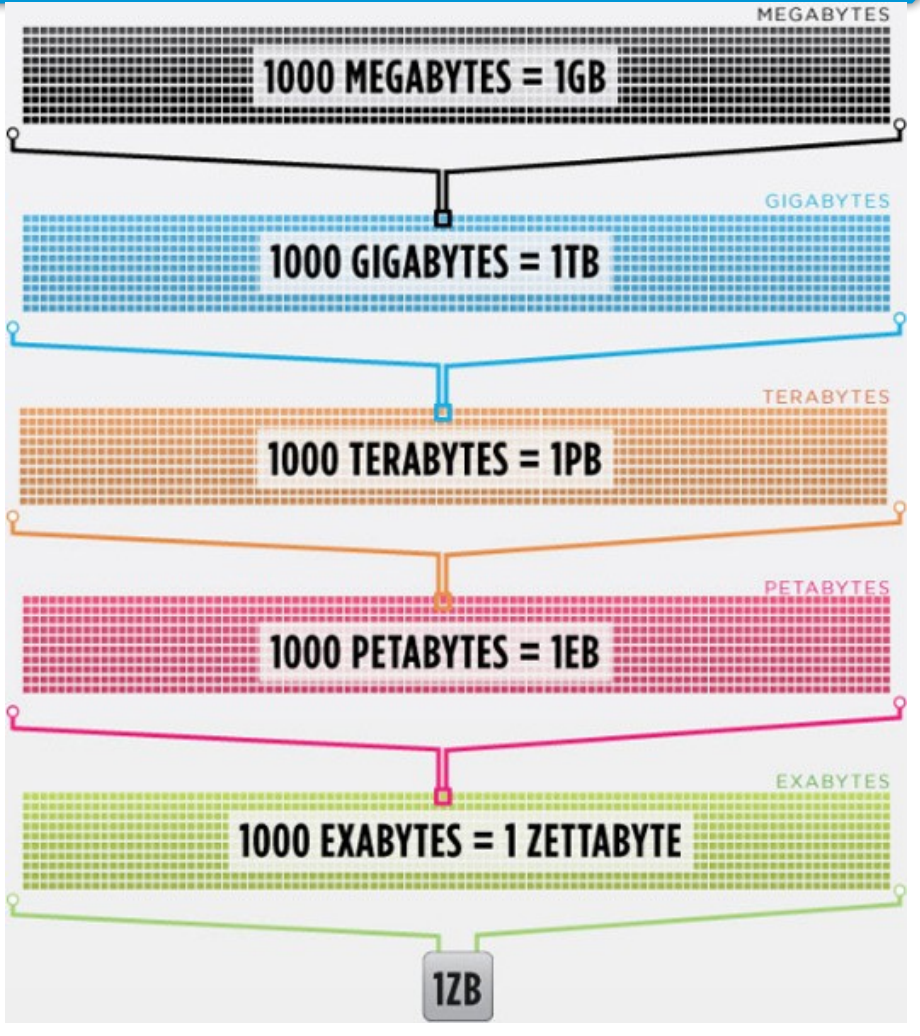
If our data is constantly evolving or every record has a variable structure, RDBMS may not be the right choice!

Trend 1

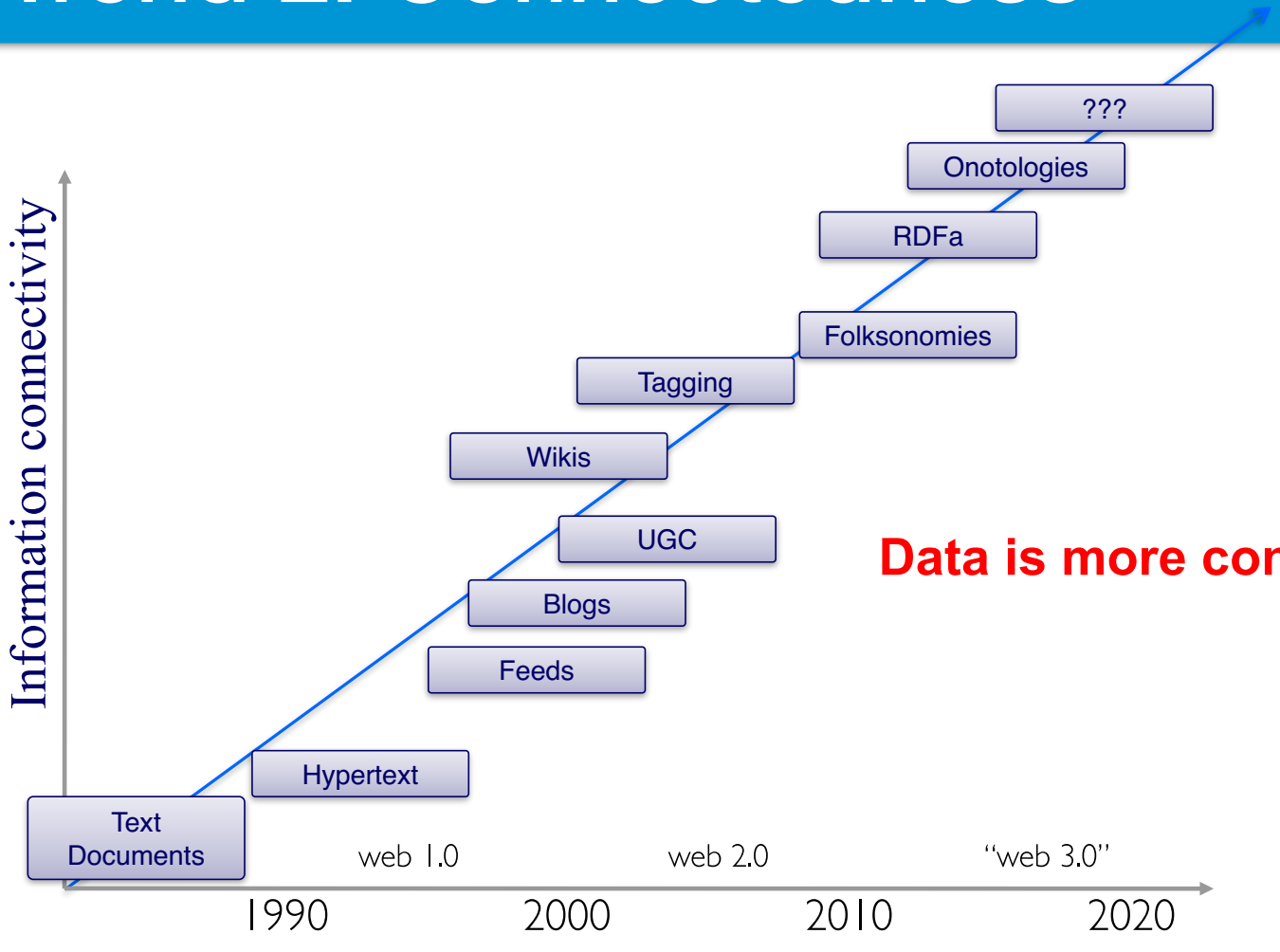
Data is getting bigger:

"Every 2 days we create as much information as we did up to 2003" – Eric Schmidt, Google in **2010**

Facebook generates
4 Petabytes per day! (2020)



Trend 2: Connectedness



Data is more connected

Trend 3: Data is often Semi-Structured (or no structure)

If you tried to collect all the data of every movie ever made, how would you model it?

Actors, Characters, Locations, Dates, Costs, Ratings, Showings, Ticket Sales, etc.

Relational Databases Challenges

Some features of relational databases make them "challenging" for certain problems:

- 1) Fixed schemas – defined ahead of time, changes are difficult, and lots of real-world data is “messy”. Relational design requires lots of Joins. **So get rid of schemas**
- 2) Complicated queries – SQL is declarative and powerful but may be overkill. **Instead, do the work in application code**
- 3) Transaction overhead – Not all data and query answers need to be perfect. **Close enough is sometimes good enough**
- 4) Scalability – Relational databases may not scale sufficiently to handle high data and query loads or this scalability comes with a very high cost. **Find new ways to scale**