Welcome to
CompSci 316: Introduction to Database Systems!!
Fall 2020

About us…

• Instructor: Sudeepa Roy
  • At Duke CS since Fall 2015
  • PhD. UPenn, Postdoc: U. of Washington
  • Member of “Duke Database Devils” a.k.a. the database research group
  Research interests:
    • “data”
    • data management, database theory, data analysis, data science, causality and explanations, uncertain data, data provenance, crowdsourcing, ....

What are the goals of this course?

• Learn about “databases” or data management
Why do we care about data? (easy)

How big data can help find new mineral deposits

Data =
Money
Information
Power
Fun
in
Science, Business,
Politics, Security,
Sports, Education, ….

Wait… don’t we need to take a Machine Learning or Statistics course for those things? Yes, but…

Also think about building a new App or website based on data from scratch

• E.g., your own version of mini-Amazon* or a Book Selling Platform
• Large data! (think about all books in the world or even in English)

• How do we start?

* Many of you going to do this in the course projects!

Who are the key people?
(book-selling website)

Who are the key people?
(book-selling website)
What should the user be able to do?

• i.e. what the interface look like? (think about Amazon)

What should the platform do?

What are the desired and necessary properties of the platform?
That was the design phase
(a basic one though)

How about C++, Java, or Python?
On data stored in large files


Sounds simple!

• Text files – for books, customer, ...
• Books listed with title, author, price, and no. of copies
• Fields separated by #’s

Query by programming

James Morgan, Durham, NC

• James Morgan wants to buy “To Kill a Mockingbird”
• A simple script
  • Scan through the books file
  • Look for the line containing “To Kill a Mockingbird”
  • Check if the no. of copies is >= 1
  • Bill James $7.20 and reduce the no. of copies by 1

What if he changes the “query” and wants to buy a book by Victor Hugo?

Revisit: What are the desired and necessary properties of the platform?

• Should be able to handle a large amount of data
• Should be efficient and easy to use (e.g., search with authors as well as title)
• If there is a crash or loss of power, information should not be lost or inconsistent
  • Imagine a user was in the middle of a transaction when a crash happened, paid the money, but the book has not been purchased
• No surprises with multiple users logged in at the same time
  • Imagine one last copy of a book that two users are trying to purchase at the same time
• Easy to update and program
  • For the admin

Solution?

A DBMS takes care of all of the following (and more):

In an easy-to-code, efficient, and robust way

• DBMS = Database Management System

* We will learn these in the course!
DBMS helps the big ones!

Note: Not always the “standard” DBMS (called Relational DBMS), but we need to know pros and cons of all alternatives.

CompSci 316 gives an intro to DBMS

- How can a user use a DBMS (programmer/designer's perspective)
  - Run queries, update data (SQL, Relational Algebra)
  - Design a good database (ER diagram, normalization)
  - Use different types of data (Mostly relational, also XML/JSON)
- How does a DBMS work (system's or admin's perspective, also for programmers for writing better queries)
  - Storage, index
  - Query processing, join algorithms, query optimizations
  - Transactions: recovery and concurrency control
- Glimpse of advanced topics and other DBMS
  - NOSQL, Spark (big data)
  - Data mining, Parallel DBMS
- Hands-on experience in class projects by building an end-to-end website or an app that runs on a database

Misc. course info

- All information available on the Course Website: [https://www2.cs.duke.edu/courses/fall20/compsci316/](https://www2.cs.duke.edu/courses/fall20/compsci316/)
  - Course info; tentative schedule and reference sections in the book; lecture slides, assignments, help docs, ...

Projects

- **Fixed project Option:** Mini-amazon
- **Open project Option:** Your own ideal (More work, more fun)
  - From previous years:
    - RA: next-generation relational algebra interpreter
      - You may get to try it out for Homework #1!
    - Managing tent shifts and schedules!
    - Tutor-tutee matching
    - What's in my fridge and what can I cook?
    - Hearsay: manage your own musics
    - Dining at Duke (and deliver meals to students)
    - National Parklopedia: a website to find information about national parks
  - Project-details doc will be posted soon
  - More examples later - but we expect you to be creative with a new idea!

Let’s get started!

Relational Data Model

What is a good model to store data?
Tree? Nested data? Graph?

(just) Tables!

Edgar F. Codd (1923-2003)

- Pilot in the Royal Air Force in WW2
- Inventor of the relational model and algebra while at IBM
- Turing Award, 1981

RDBMS = Relational DBMS

The famous “Beers” database

Each has an address

Bars Serve Beers

At price $m$

Bars Each has an address

Drinkers Frequent Bars

“X” times a week

Drinkers Likes Beers

Bars Each has a brewer

Beers

Name

brewer

Budweiser

Anheuser-Busch Inc.

Corona

Grupo Modelo

Dixie

Dixie Brewing

Drinker

bar

times_a_week

Ben

Satisfaction

2

Dan

The Edge

1

Dan

Satisfaction

2

Ben

Budweiser

Corona

Budweiser

2.25

The Edge

Corona

2.00

Satisfaction

Budweiser

2.15

See online database for more tuples

### “Beers” as a Relational Database

<table>
<thead>
<tr>
<th>Bar</th>
<th>address</th>
<th>Serves</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Edge</td>
<td>108 Morris Street</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The Edge</td>
<td>915 W. Main Street</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Serves</th>
<th>name</th>
<th>brewer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Edge</td>
<td>Budweiser</td>
<td>Anheuser-Busch Inc.</td>
<td>2.50</td>
</tr>
<tr>
<td>The Edge</td>
<td>Corona</td>
<td>Grupo Modelo</td>
<td>3.00</td>
</tr>
<tr>
<td>Satisfaction</td>
<td>Budweiser</td>
<td>2.15</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drinker</th>
<th>name</th>
<th>address</th>
<th>Likes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ben</td>
<td>Amy</td>
<td>100 W. Main Street</td>
<td></td>
</tr>
<tr>
<td>Dan</td>
<td>Ben</td>
<td>101 W. Main Street</td>
<td></td>
</tr>
<tr>
<td>Dan</td>
<td>Amy</td>
<td>300 N. Duke Street</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Likes</th>
<th>name</th>
<th>address</th>
<th>Serves</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy</td>
<td>Ben</td>
<td>100 W. Main Street</td>
<td></td>
</tr>
<tr>
<td>Ben</td>
<td>Dan</td>
<td>101 W. Main Street</td>
<td></td>
</tr>
<tr>
<td>Dan</td>
<td>Amy</td>
<td>300 N. Duke Street</td>
<td></td>
</tr>
</tbody>
</table>

Relational data model

- A database is a collection of relations (or tables)
- Each relation has a set of attributes (or columns)
- Each attribute has a name and a domain (or type)
  - Set-valued attributes are not allowed
- Each relation contains a “set” of tuples (or rows)
  - Each tuple has a value for each attribute of the relation
  - Duplicate tuples are not allowed (Two tuples are duplicates if they agree on all attributes)
  - Ordering of rows doesn’t matter (even though output is always in some order)
- However, SQL supports “bag” or duplicate tuples (why?)
  - Simplicity is a virtue
  - not a weakness!

### Schema vs. instance

- **Schema**
  - Beer (name string, brewer string)
  - Serves (bar string, beer string, price float)
  - Frequents (drinker string, bar string, times_a_week int)
- **Instance**
  - Actual tuples or records
  - Compare to types vs. collections of objects of these types in a programming language

SQL: Querying a RDBMS

- **SQL**: Structured Query Language
  - Pronounced “S-Q-L” or “sequel”
  - The standard query language supported by most DBMS
  - First developed at IBM System R
  - Follows ANSI standards

**SQL is Declarative:**

- Programmer specifies what answers a query should return, but not how the query is executed
- DBMS picks the best execution strategy based on availability of indexes, data/workload characteristics, etc.
  - Provides **physical data independence**
- Not a “Procedural” or “Operational” language like C++, Java, Python

Basic queries: SFW statement

- **SELECT** $A_1, A_2, \ldots, A_n$
  FROM $R_1, R_2, \ldots, R_m$
  WHERE condition

- **SELECT, FROM, WHERE** are often referred to as SELECT, FROM, WHERE “clauses”
Example: reading a table

- `SELECT *` FROM `Serves`

- Single-table query
- `WHERE` clause is optional
- `*` is a short hand for “all columns”

<table>
<thead>
<tr>
<th>beer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Edge Budweiser</td>
<td>2.50</td>
</tr>
<tr>
<td>The Edge Corona</td>
<td>3.00</td>
</tr>
<tr>
<td>Satisfaction Budweiser</td>
<td>2.25</td>
</tr>
</tbody>
</table>

Example: selecting few rows

- `SELECT beer AS mybeer` FROM `Serves` WHERE `price < 2.75`

- `SELECT beer` FROM `Serves` WHERE `bar = 'The Edge'`

- `SELECT list can contain expressions` Can also use built-in functions such as `SUBSTR`, `ABS`, etc.
- `String literals (case sensitive) are enclosed in single quotes`
- "AS" is optional
- Do not want duplicates? Write `SELECT DISTINCT beer` ...

Example: Join

- Find addresses of all bars that ‘Dan’ frequents

- Which tables do we need?

Example: Join

- Find addresses of all bars that ‘Dan’ frequents

- Which tables do we need?

Example: Join

- Find addresses of all bars that ‘Dan’ frequents

- Which tables do we need?

Which tables do we need?

How do we combine them?

Try some SQL queries yourself on pgweb!

(See how to access the pgweb interface for a small “Beers” database on the slides posted on the course website)

Next: semantics of SFW statements in SQL
Semantics of SFW

- SELECT E₁, E₂, ..., Eₘ
  FROM R₁, R₂, ..., Rₘ
  WHERE condition

  - For each t₁ in R₁;
  - For each t₂ in R₂; ... ... For each tₘ in Rₘ;
    1. Apply “FROM” Form cross-product of R₁, ..., Rₘ.
    2. Apply “WHERE” Only consider satisfying rows
    3. Apply “SELECT” Compute and output E₁, E₂, ..., Eₘ as a row
    4. Output the desired columns

Step 1: Illustration of Semantics of SFW

- NOTE: This is “NOT HOW” the DBMS outputs the result, but “WHAT” is outputs!
  Form Cross product of two relations

```
<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>drinker</th>
<th>bar</th>
<th>times_a_week</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Edge</td>
<td>108 Morris Street</td>
<td>Dan</td>
<td>The Edge</td>
<td>2</td>
</tr>
<tr>
<td>The Edge</td>
<td>109 W. Main Street</td>
<td>Dan</td>
<td>The Edge</td>
<td>3</td>
</tr>
<tr>
<td>The Edge</td>
<td>109 W. Main Street</td>
<td>Dan</td>
<td>The Edge</td>
<td>2</td>
</tr>
<tr>
<td>Satisfaction</td>
<td>905 W. Main Street</td>
<td>Dan</td>
<td>Satisfaction</td>
<td>3</td>
</tr>
</tbody>
</table>
```

Step 2: Illustration of Semantics of SFW

- NOTE: This is “NOT HOW” the DBMS outputs the result, but “WHAT” is outputs!
  Discard rows that do not satisfy WHERE condition

```
<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>drinker</th>
<th>bar</th>
<th>times_a_week</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Edge</td>
<td>108 Morris Street</td>
<td>Dan</td>
<td>The Edge</td>
<td>2</td>
</tr>
<tr>
<td>The Edge</td>
<td>109 W. Main Street</td>
<td>Dan</td>
<td>The Edge</td>
<td>3</td>
</tr>
<tr>
<td>The Edge</td>
<td>109 W. Main Street</td>
<td>Dan</td>
<td>The Edge</td>
<td>2</td>
</tr>
<tr>
<td>Satisfaction</td>
<td>905 W. Main Street</td>
<td>Dan</td>
<td>Satisfaction</td>
<td>3</td>
</tr>
</tbody>
</table>
```

Step 3: Illustration of Semantics of SFW

- NOTE: This is “NOT HOW” the DBMS outputs the result, but “WHAT” is outputs!
  Output the “address” output of rows that survived

```
<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>drinker</th>
<th>bar</th>
<th>times_a_week</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Edge</td>
<td>108 Morris Street</td>
<td>Dan</td>
<td>The Edge</td>
<td>2</td>
</tr>
<tr>
<td>The Edge</td>
<td>109 W. Main Street</td>
<td>Dan</td>
<td>The Edge</td>
<td>2</td>
</tr>
<tr>
<td>The Edge</td>
<td>109 W. Main Street</td>
<td>Dan</td>
<td>The Edge</td>
<td>3</td>
</tr>
<tr>
<td>Satisfaction</td>
<td>905 W. Main Street</td>
<td>Dan</td>
<td>Satisfaction</td>
<td>2</td>
</tr>
</tbody>
</table>
```

Final output: Illustration of Semantics of SFW

- NOTE: This is “NOT HOW” the DBMS outputs the result, but “WHAT” is outputs!
  Output the “address” output of rows that survived

```
<table>
<thead>
<tr>
<th>name</th>
<th>address</th>
<th>drinker</th>
<th>bar</th>
<th>times_a_week</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Edge</td>
<td>108 Morris Street</td>
<td>Dan</td>
<td>The Edge</td>
<td>2</td>
</tr>
<tr>
<td>Satisfaction</td>
<td>905 W. Main Street</td>
<td>Dan</td>
<td>Satisfaction</td>
<td>2</td>
</tr>
</tbody>
</table>
```

Announcements (Tue, 08/18)

- You are/will be on Sakai, Piazza, Gradescope by the next class
- You will receive instructions on installing the VM
  - Please follow Piazza posts, all notifications will be posted there and you should receive emails right away
- Office hours start from today
- First homework to be released soon