Overview
Relational Database
Hands-On SQL

BMI701 Introduction of Biomedical Informatics
Lab Session 1

Wei-Hung Weng

September 8, 2016

HMS DBMI — MGH LCS

HARVARD MEDICAL SCHOOL MASSACHUSETTS GENERAL HOSPITAL

Logistics

- Wei-Hung Weng, MD
- Research fellow in MGH Lab of Comp Sci
- NLP, medical ontology, machine learning, databases
- Every Thursday 12:30pm, about 1 hour
- Flexible, not mandatory
- Website
  - Github repository (codes and slides)
  - ckbjimmy@gmail.com

Schedule

Lab	Date	Topic
1	Sep 08	Overview / relational database / SQL
2	Sep 15	Database design /
		Database normalization / R basic
3	Sep 22	Ontology / database using R
4	Sep 26	**(Monday before & after Adam's class)**
		NoSQL / Project discussion
	Oct 06	**No lab (Presentation week)**
	Oct 13	**No lab (Columbus day)**
5	Oct 20	NLP using R or py / regular expression

Schedule

Lab	Date	Topic
6	Oct 27	MetaMap / cTAKES demonstration
7	Nov 03	Data visualization using ggplot2
8	Nov 10	Bioinformatics tools / GWAS
	Nov 17	**No lab (AMIA)**
9	Nov 24	ML using Weka, R or python (or unsupervised learning / feature engineering)
10	Dec 01	Deep learning on image classification (py)
	Dec 08	**No lab (Final week)**

Survey

- Background? (clinicians / scientists and engineers)
- CS courses? (introduction / data structure / algorithm)
- Math courses?
- Programming language?
  - C/C++, Java, Python, R, Matlab/Octave, Perl, ...

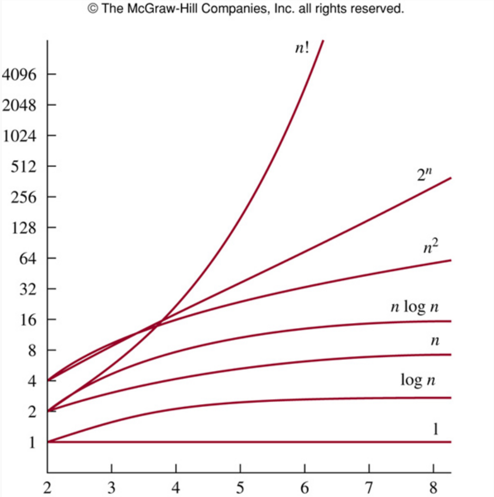
Today

- Database
- Relational database
- Simple SQL syntax
- MySQL

Why Database?

- I believe that you've learned a lot about database from Adam's class
- RAM is expensive and small $\rightarrow$ HD is cheaper and large
- You can't put all data into RAM (unless you are $$$$)
- But it's also not easy to manage data on your HD
- Therefore we need database solution
    - $\mathbf{O(1)} < \mathbf{O(log(n))} < \mathbf{O(n)} < O(nlog(n)) < O(n^m) < O(2^n)$

Computational Complexity



Courtesy by UCSB

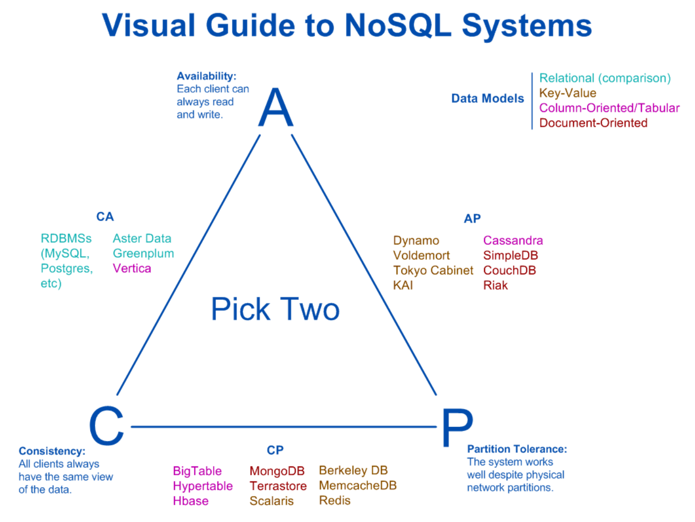
Database CAP

- Consistency
  - All nodes should see the same data at the same time (transaction)
- Availability/Speed
  - Every request receives a response about whether the transaction is passed or failed
- Partition tolerance
  - Able to continue operation even if there are partially loss/failure of the system
- Eric Brewer 2000
- You may only pick up two of three

SQL

- Structured Query Language (higher C and A)
    - RDB
    - Can't deal with extremely large data
- NoSQL for big data (higher P)
- Big data?
    - Google/facebook/twitter-scale (10TB to 10PB)
    - AWS: No need to use NoSQL if your user $<$ 10M
- So we will talk about NoSQL later

Database Selection



Courtesy by digbigdata.com

Popular RDBMS

- MS SQL Server: you will probably use this in your lab/company
- Oracle: awesome if you are $$$$
- **MySQL** / MariaDB
  - Open-sourced, fast, easy to scale-up, community, BUT no JSON
- **PostgreSQL**
  - Open-sourced, fast enough, can scale-up, can use JSON, array, ...

Relational Database

- ACID
    - Atomicity (all or none / commit or rollback)
    - Consistency (transaction between two accounts)
    - Isolation (locking)
    - Durability (won't rollback once the transaction is done)
- PK, FK, index
- ER diagram
- Normalization/denormalization (next lab)

Relational Database

- A table has fixed columns
- A column has a datatype
  - Numeric, character, datetime, boolean, geometric, text, JSON, array (postgresql), ...
- Rows can grow (python tuple $\rightarrow$ list)

Primary Key (PK)

- Unique identifier in the table
- MUST be unique
- CAN'T be NULL
- Index is created

Foreign Key (FK)

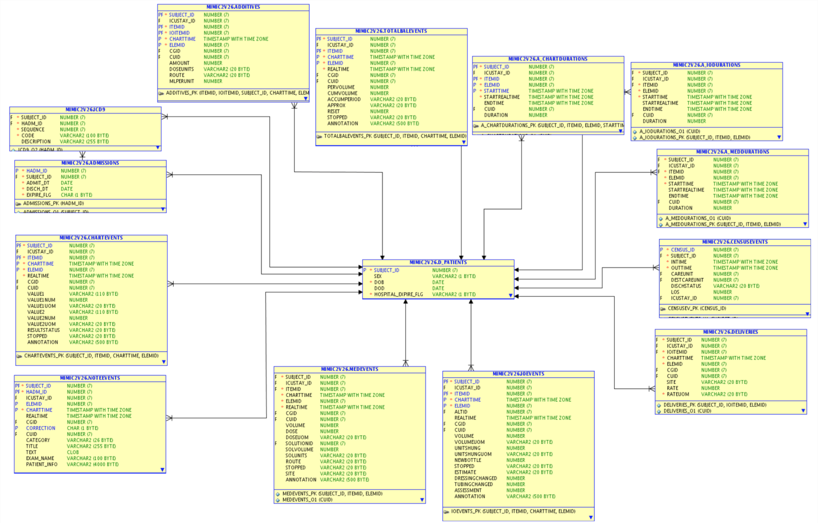
- For referential integrity
- The value MUST exist in other tables
- May be NULL or duplicated
- No index is created

Index

- Fast but takes space
- Slow insert/update but fast read/delete
- B+ tree

Task	No index	Index
Create	O(n)	O(n)
Read	O(n)	O(log(n))
Update	O(1)	O(log(n))
Delete	O(n)	O(log(n))

Entity-Relationship Diagram (ER Diagram)



MIMIC-II Database (Clifford, 2011)

SQL's CRUD

- CRUD: create, read, update, delete
- Insert
- Select
  - **Filter**: where, distinct
  - **Aggregate**: group by
  - **Join**
  - **Subquery**
  - Partition (splitting the data into manageable size)
- Update
- Delete

Hands-on SQL

- Install MySQL Server
- Install MySQL Workbench
- Small databases
- github.com/ckbjimmy/bmi701lab/blob/master/lab01.sql

Filter

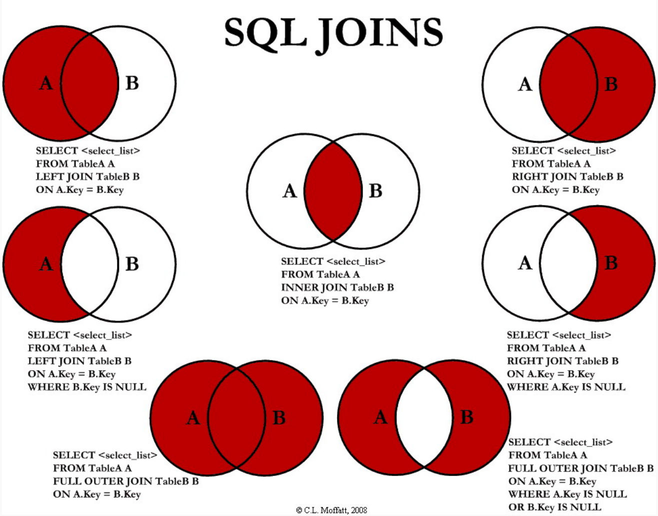
- `SELECT * FROM gwas LIMIT 5`
- `SELECT gene FROM gwas WHERE chr_id = 22`
- `SELECT distinct gene, trait, p_value FROM gwas WHERE allele like 'A' ORDER BY p_value`
- `SELECT gene, trait FROM gwas WHERE trait like '%diabetes_'`
- `SELECT gene, trait FROM gwas WHERE trait like '%diabetes%'`
- `SELECT gene, trait FROM gwas WHERE trait like 'diabetes'`

- `SELECT GROUP_CONCAT(DISTINCT gene SEPARATOR ',') FROM gwas WHERE chr_id = 3`

Aggregation

- `SELECT count(*) FROM sitka`
- `SELECT avg(size) FROM sitka`
- `SELECT max(size) FROM sitka`
- `SELECT min(size) FROM sitka`
- `SELECT stddev(size) FROM sitka`
- `SELECT tree, avg(size) FROM sitka GROUP BY tree`

Join



Courtesy by codeproject.com

Join

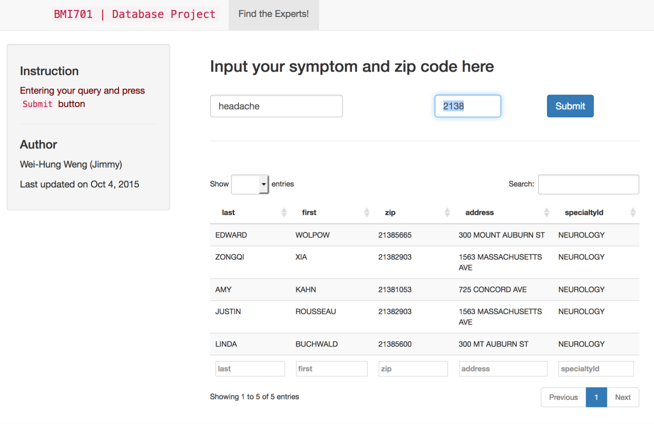
- ON col1 = col2
- Inner (default), left, right, full, cross (inner join on TRUE)
- Using PK
- `SELECT curr_description_f.conceptid, curr_description_f.term, curr_textdefinition_f.term FROM curr_description_f LEFT JOIN curr_textdefinition_f ON curr_description_f.conceptid LIKE curr_textdefinition_f.conceptid WHERE curr_description_f.conceptid LIKE '%777%' LIMIT 30`

Subquery

- `SELECT * FROM sitka WHERE treat LIKE '%ozo%'`
- `SELECT * FROM (SELECT * FROM sitka WHERE treat LIKE '%ozo%') oz WHERE size > 5.0`

Demonstration

- My database project (MySQL, R, shiny, AWS)



Preparation for Next Next Week

- Install R
- Install RStudio
- Register UTS Service
- Download SNOMED (need to wait for UTS permission)

Take Home Message

- Every Thursday 12:30pm
- `SELECT (DISTINCT) ... FROM ... JOIN ... ON ... WHERE ... GROUP BY ... ORDER BY ... LIMIT ...`
- Install R/RStudio, register UTS, download SNOMED
- Contact
  - Github repository
  - ckbjimmy@gmail.com
  - Linkedin: Wei-Hung Weng