ISTM 6212 - Week 4
Intro Relational Databases & SQL

Daniel Chudnov, dchud@gwu.edu

2016-09-20

Agenda

- Notes on shell, filter scripts, and parallel
- Project 01 and reviews
- Relational databases
- Basic SQL
- Exercise 03 - next week

Notes on shell, filters, parallel

Shell, terminal, notebooks

- "the shell", "terminal", "command line" are synonymous
- "!" commands are sent to the shell from the notebook
- notebooks are web apps that communicate with a backend kernel
- Python notebooks use IPython kernels

IPython, an enhanced Python REPL

- REPL: Read - Evaluate - Print Loop
- you can use IPython in your shells
- (it's better than Python's shell)
- bash, Python, IPython, R all have REPLs

REPLs are friendly

- shell (bash, here), Python, IPython, R, sqlite3, …
- IDEs (Integrated Development Environment) like Spyder and RStudio combine editor, file explorer, shell, and code REPL
- innovative 50 years ago, still very useful

Why filters?

- "Small is beautiful", "Do one thing and do it well", "Make every program a filter" - Mike Gancarz, *The UNIX Philosophy*
- simpler to develop, maintain, and document
- simpler to understand, use, explain, and remember
- simpler to reuse, combine, connect

Filters in any language

- sort, grep, uniq are written in C
- csvkit and your filters are written in Python
- filters can be written in any language
- interface (input, output) is the key
- pipelines can combine filters in many languages

Why parallel?

- easy to use
- easy to scale up on one machine
- can scale up to many machines
- fits into repetitive workflows (big pipelines, shell scripts, scheduled jobs)

shell pipelines
can outperform
Hadoop clusters* **

* sometimes, with some data, with some problems

** we'll come back to this

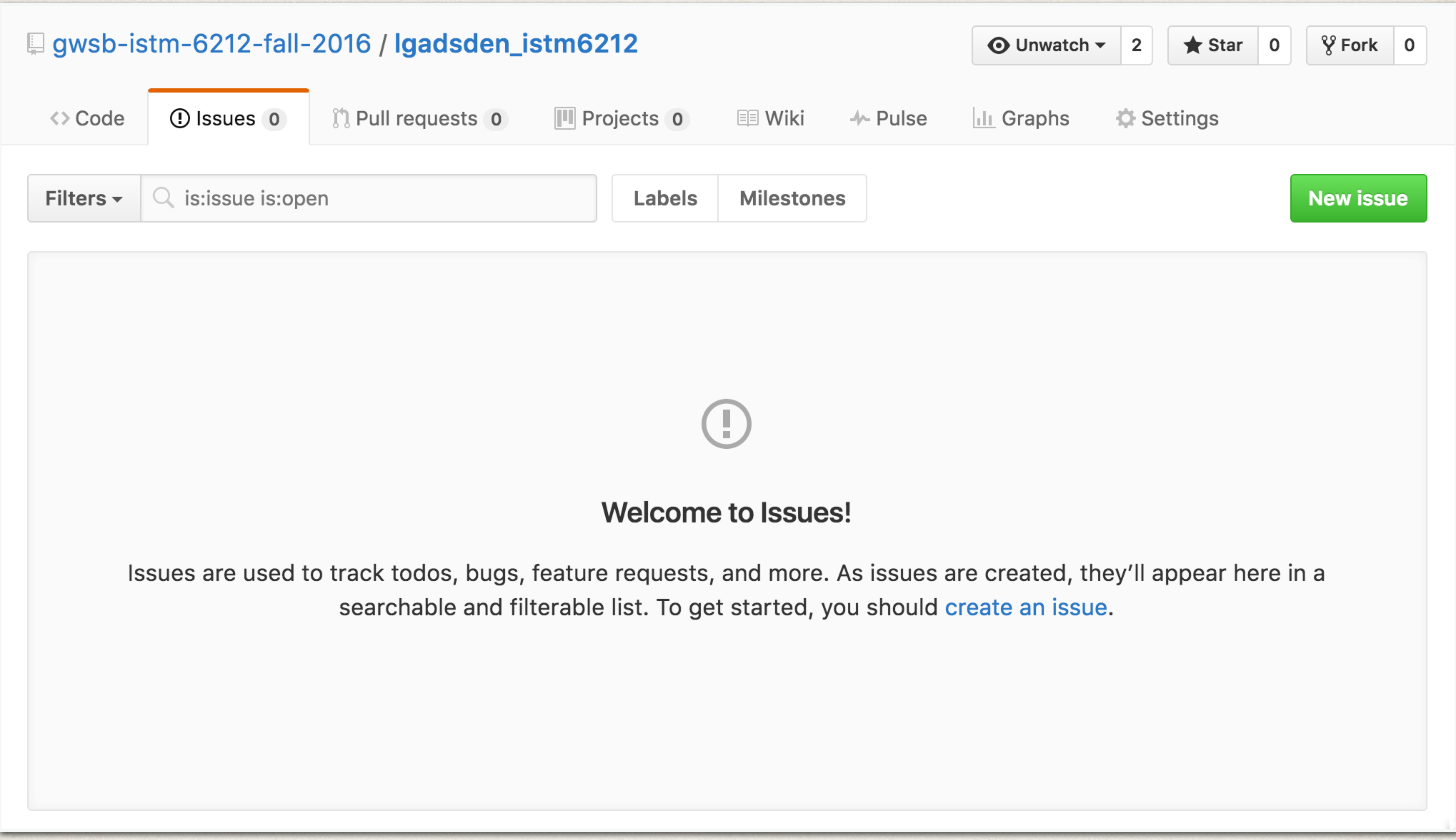
Project 01 and reviews

Project 01

- individual project, *not* group project
- use datanotebook.org if your VM doesn't work
- don't share solutions; don't copy each other
- if you do the readings and understand the exercises, this should not be difficult

Project 01 - Reviews

- goals:
  - get used to reading/running each others' code
  - learn from each other
- every student will review other two students' work
- clone/download and run their project notebooks
- use GitHub Issues to leave comments


gwsb-istm-6212-fall-2016 / lgadsden_istm6212
Unwatch 2
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Pulse
Graphs
Settings
Filters
is:issue is:open
Labels
Milestones
New issue
Welcome to Issues!
Issues are used to track todos, bugs, feature requests, and more. As issues are created, they'll appear here in a searchable and filterable list. To get started, you should create an issue.

gwsb-istm-6212-fall-2016 / lgadsden_istm6212

Unwatch 2

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Pulse | Graphs

review for project-01

Write | Preview

Hi Lawrence, this is my review for your Project 1 notebook.

I was able to run your notebook at datanotebook.org just fine.

I liked your solution for Part 2B. Your explanation of the pipeline you used was very clearly written and you brought up a great point about `csvgrep` vs regular `grep`. I hadn't thought of that before!

Your answer for Part 3B was different from mine. I found that I couldn't recreate the same results that we saw from the pipeline without Python filters, but in looking at your answer, I think you got it right and that I made a mistake. I didn't handle whitespace correctly but I see how you did it and I understand my mistake now.

Great job! Thanks, -Dan

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Styling with Markdown is supported

Submit new issue

Writing good reviews

- read and run their code - all of it!
- can you reproduce their results?
- did you get the same answers?
- do you understand their explanations?
- did you learn something?
- be friendly, constructive, supportive

Review assignments

- randomly chosen - two reviews per student
- each student should receive two reviews
- assignments go out after the deadline Friday
- reviews due by class next Tuesday

Review - time expectation

- should take >= 10 minutes to read each notebook
- should take >= 10 minutes to write each review
- plan on one hour
- do them soon, while you remember yours!

- questions about Project 01?

Relational databases

this section's slides borrow liberally from

Database System Concepts, 5th and 6th ed.

Silbershatz, Korth, and Sudarshan

highly recommended

see www.db-book.com for more

(today: chapters 1-3)

Why relational databases?

- many use cases require more than simple CSV files:
  - complex enterprises with inventory, scheduling, products, purchases, accounts, customers, employees, etc.
  - many simultaneous users and uses
  - large volumes of data and constant changes

Let's run amazon.com on CSV

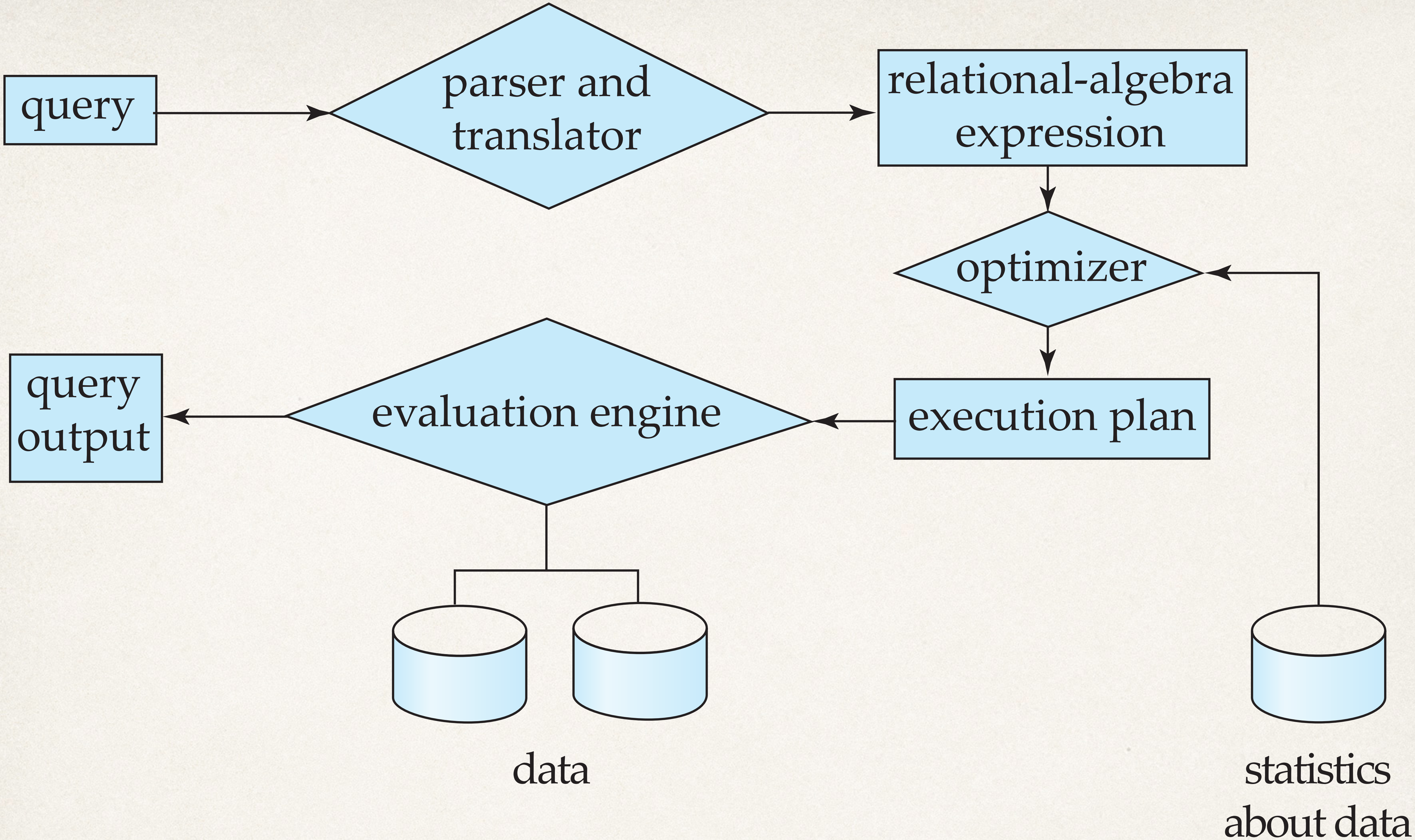
- what might go wrong?
- what might go right?

Databases provide:

- precise **models** of data and its interrelationships
- **isolation** of use cases and users
- **concurrent** reads and write
- **integrity** of data according to models & **constraints**
- **atomicity** of updates - failures handled consistently
- differential **access controls** to support **security** policies

Databases provide (2):

- separation of logical model (schema), physical storage
- Data Definition Language (DDL) for defining schema
- Data Dictionary: schema, relationships, constraints
- Data Manipulation Language (DML) for querying
- statistics about stored data
- optimized query processing



Relational database query processing

image (c) Silbershatz, Kortz, Sudarshan

Databases provide (3):

- **transactions** which group related operations into logical functions
- **consistency** according to schema; transactions complete successfully or fail completely (roll back)
- supports **concurrency** of multiple users, applications
- (think of bank example - deposit, purchase, account)

Types of database users

- **end users** - e.g. bank tellers, web/app users
- **application programmers** - develop code using database to support all users
- **sophisticated users** - analysts write queries, use results to support operations, planning
- **administrators** - manage system back-end

Types of database users

- **end users** - e.g. bank tellers, web/app users
- **application programmers** - develop code using database to support all users
- **sophisticated users** - analysts write queries, use results to support operations, planning
- **administrators** - manage system back-end

The relational model

attributes
(or columns)

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

tuples
(or rows)

attributes have **domain**

values are **atomic**

values may be **null**

image (c) Silbershatz, Kortz, Sudarshan

Relational schema

$A_1 A_2, \ldots, A_n$ Attributes

$R = (A_1, A_2, \ldots, A_n)$ Relation schema

e.g. *Instructor = (id, name, department, office)*

Given sets $D_1, D_2, \ldots, D_n$, a **relation** *r* is a subset of $D_1 \times D_2 \times \ldots \times D_n$, or a set of *n*-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$

Relation instances

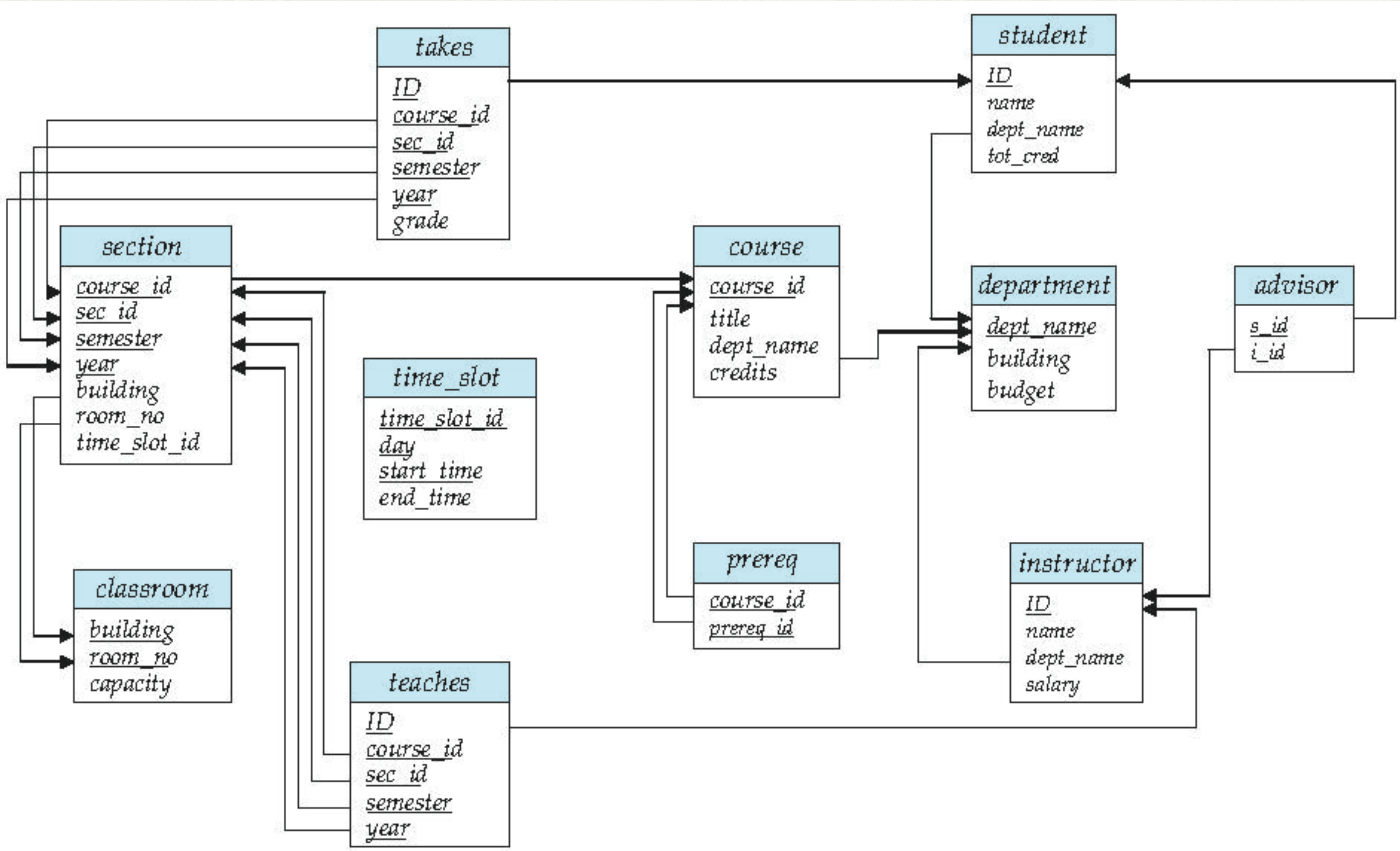
- a **relation instance** - the current values of a relation - are specified by a **table**
- an element *t* of *r* is a **tuple**, represented by a **row** in the table
- relation instances - the rows - are **not ordered**
- here, "tuple" is similar to a Python tuple
- here also, "row" is similar to a Tidy Data observation

Relational schemas

- contain multiple relations (product, customer, order)
- relations typically correspond to real-world concepts
- avoid repetition, null values, and are normalized

Keys

- used to identify a unique tuple of each relation *r(R)*
- one minimal (fewest possible attributes) key is the **primary key**
- a **foreign key** connects a value in one relation to its occurrence in another relation



course.***course_id*** - primary key

course_id in *takes, teaches, section, prereq* - foreign key

image (c) Silbershatz, Kortz, Sudarshan

Symbol (Name)	Example of Use
σ (Selection)	$\sigma_{\text{salary>=85000}}(instructor)$
	Return rows of the input relation that satisfy the predicate.
Π (Projection)	$\Pi_{ID,\ salary}(instructor)$
	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
⋈ (Natural Join)	*instructor* ⋈ *department*
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
× (Cartesian Product)	*instructor* × *department*
	Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes)
∪ (Union)	$\Pi_{name}(instructor) \cup \Pi_{name}(student)$
	Output the union of tuples from the two input relations.

Relational operations

image (c) Silbershatz, Kortz, Sudarshan

Operation: select tuples

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

relation *r*

A	B	C	D
α	α	1	7
β	β	23	10

select tuples with *A*=*B* and *D*>5

$$\sigma_{A=B, D>5}(r)$$

Operation: select columns

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

relation r

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

select (project) A and C

$$\Pi_{A,C}(r)$$

Operation: join relations (Cartesian Product)

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

relations *r, s*

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

$$r \times s$$

Operation: set difference

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

relations *r, s*

A	B
α	1
β	1

$r - s$

Operation: set intersection

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

relations *r, s*

A	B
α	2

$r \cap s$

- these relational operations underlie everything we'll do next
- SQL is the standard language for implementing them in relational databases
- code libraries like dplyr for R, pandas for Python use same concepts (in part)

Basic SQL

SQL - Structured Query Language

- grew out of IBM's Sequel language
- international standard since 1986
- 1992 version was first widely implemented
- basic features the same, others vary per database

What does SQL do?

- Data Definition Language (DDL) - create and manage schema for each relation, attribute value domains, key integrity constraints, indexes, security, storage
- Data Manipulation Language (DML) - query!
- Today DML, next week DDL, too

Basic SQL DML query structure

- query, insert, delete, update tuples
- typical form: SELECT $A_1, A_2, \ldots, A_n$
  FROM $r_1, r_2, \ldots, r_n$
  WHERE P;
- A_i - attribute, R_i - relation, P - predicate
- result of SQL query is another relation

SQL projection: SELECT

- SELECT clause lists **attributes** to show in query result
- this is the **projection** (Π) operation
- e.g.: SELECT *name*
  FROM *instructor;*

SQL product: FROM

- FROM clause lists **relations** involved in query
- this is the **Cartesian product** (X) operation
- e.g.: SELECT *
  FROM *instructor, teaches;*

SQL selection: WHERE

- WHERE clause specifies **conditions** result must satisfy
- this is the **selection** (σ) operation
- e.g.: SELECT *name*
  FROM *instructor*
  WHERE *dept_name = 'DNSC';*

SQL projection: DISTINCT

- DISTINCT eliminates **duplicates** in query results
- modifies the **projection** operation
- e.g.: SELECT DISTINCT *name*
  FROM *instructor;*

SQL projection: AS

- AS renames **attributes** in query results
- modifies the **projection** and **selection** operations
- e.g.: SELECT *I.name AS last_name*
  FROM *instructor AS I;*

SQL selection: wildcards

- wildcards match strings in attribute values
- '%' - match any substring, '_' - match one character
- modifies the **selection** operation
- e.g.: SELECT *name*
  FROM *instructor*
  WHERE *name* LIKE '%oye%';

SQL projection: ORDER BY

- specify the ordering of result relation by attributes
- may be ASCending (default) or DESCending order
- may be more than one attribute
- e.g.: SELECT *name, dept_name*
  FROM *instructor*
  ORDER BY *dept_name, name;*

SQL: NULL values

- a schema can define attributes that allow NULL values
- NULL indicates an unknown or non-extant value
- operations w/NULL result in NULL: 5+NULL=NULL
- use IS NULL, IS NOT NULL to check, not '='
- e.g.:

  SELECT *name, dept_name*
  FROM *instructor*
  WHERE *cell_number* IS NULL;

SQL: aggregate functions

- AVG(), MIN(), MAX(), SUM(), COUNT() operate on multiple sets of values from a relation column at once
- each returns one value, as expected
- e.g.: SELECT AVG(*total*) AS *avg_total*
  FROM *orders*
  WHERE *customer_state* = *'DC'*;

SQL: aggregates and grouping

- GROUP BY collects like values into separate groups for aggregation
- typically returns one value per group
- e.g.: SELECT *customer_state*, AVG(*total*) AS *avg_total*
  FROM *orders*
  GROUP BY *customer_state;*

Practice!
Exercise 03 next week