Introduction, SQL

CSE 4/562: Database Systems | Lecture 1

DB. Sys.: T.C.B.: Ch. 1, 2.1-2.2

Background

- 15+ years studying database systems
- 2 major database systems (DBToaster, Vizier)

Research

- I make the database go brrr (optimizers, data structures)
- I make it easier to write compilers (query languages)
- I make it easier to understand errors (provenance, uncertain data)

Fun

- 🚴, 🍲, 🎲, ⚔️

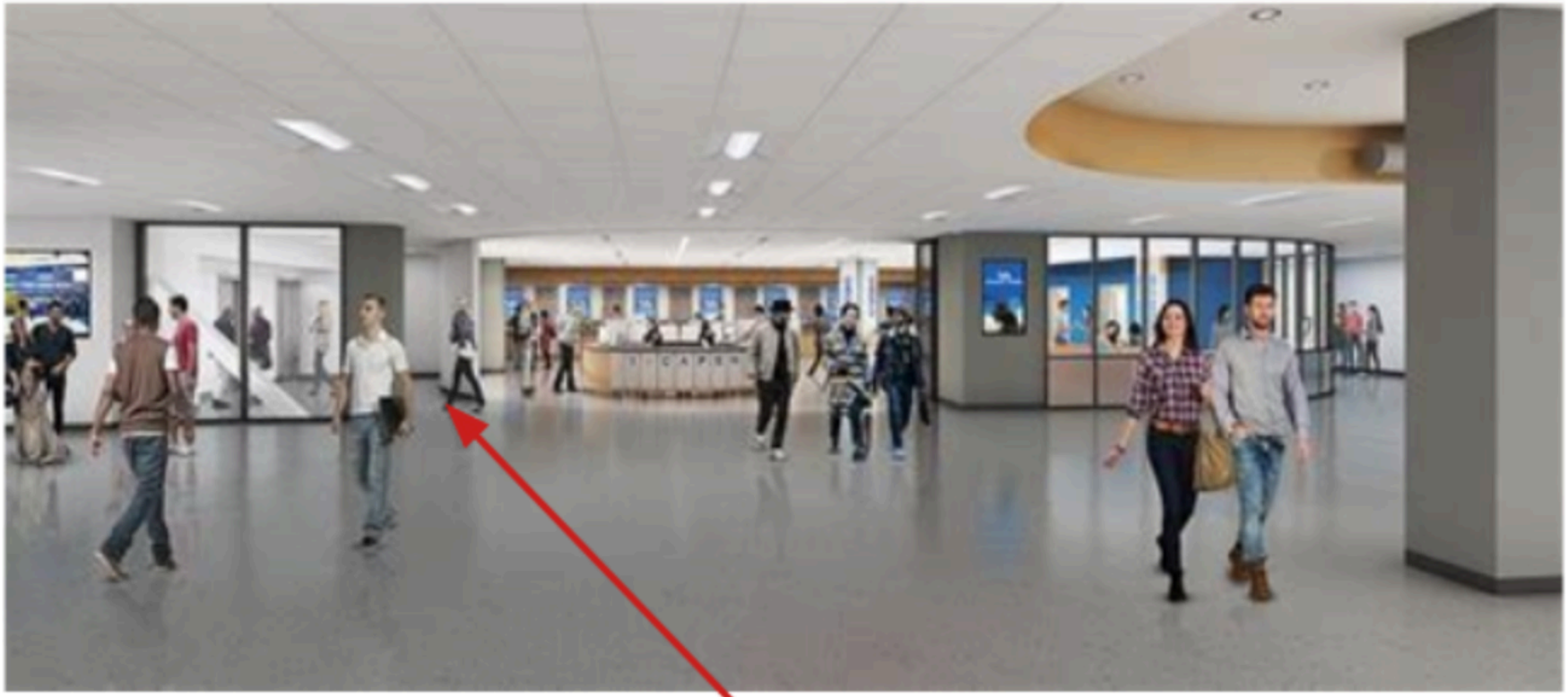
Contact Info

- **Office**: Capen 212
- **Email**: okennedy@buffalo.edu
- **Office Hours**: Tue 10-11 AM & Thu 1-3 PM, or by appointment

---

Put “[CSE 4/562]” in the subject line to make sure you get a timely response.

Finding Capen 211/212



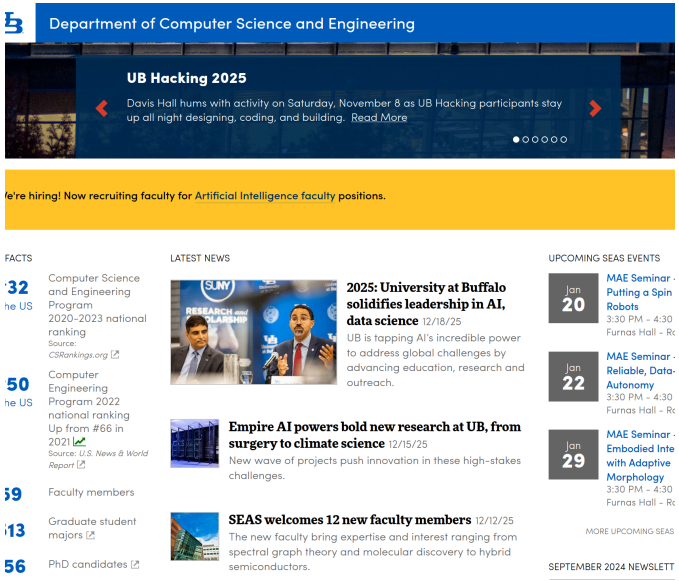
212 Capen: Take these elevators, then turn right.

Logistics

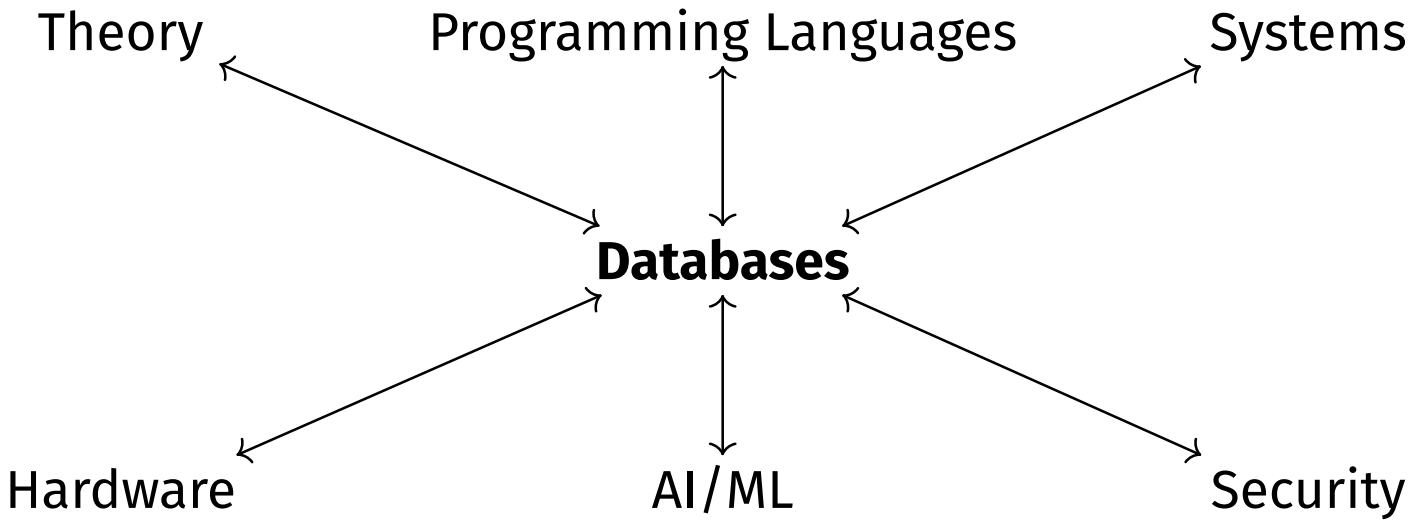
- **Course Info/Syllabus**
  https://odin.cse.buffalo.edu/teaching/cse-4562/2026sp/
- **Course Forums**: Piazza
  https://piazza.com/buffalo/spring2026/cse4562
- **Project Submission**: Autolab
  https://autolab.cse.buffalo.edu/courses/cse462-s26

Why are databases awesome?

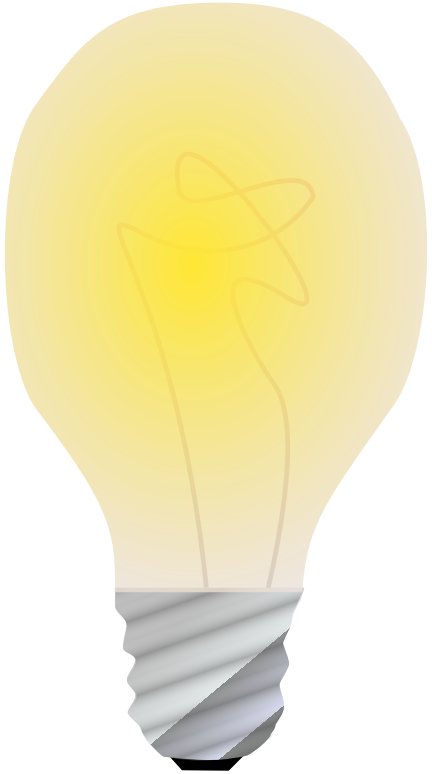
Databases are everywhere



Databases Tie Together Computer Science



Databases are a state of mind



What is this class about?

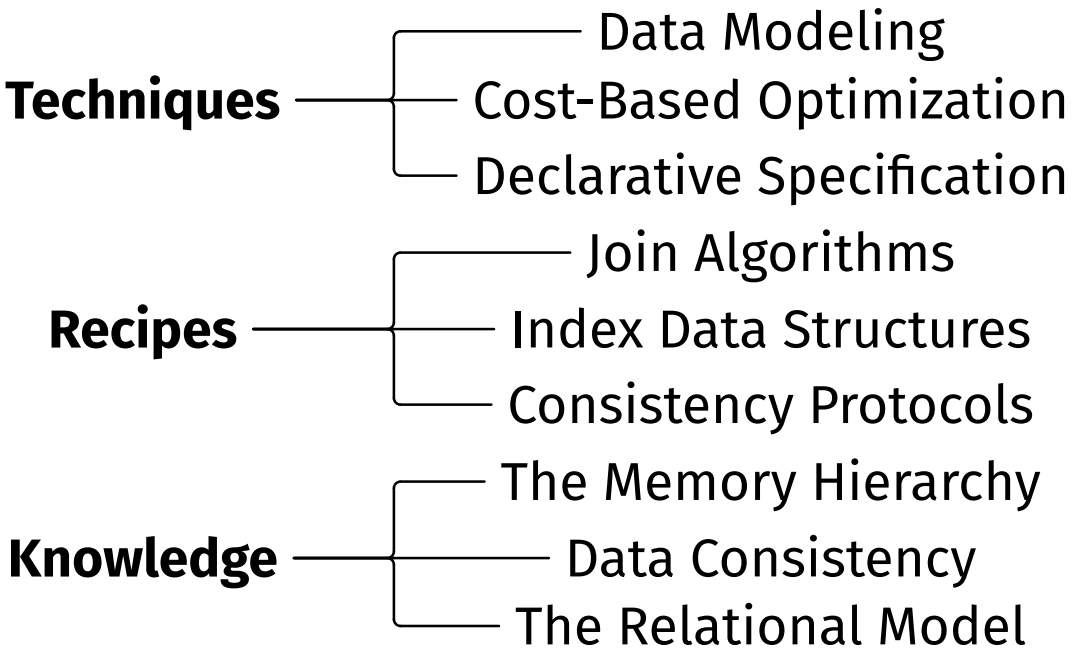
Data Access

- How do we efficiently compute with large amounts of data?
- How do we efficiently organize data to make access faster?
- How do we make it easier for users to specify data-centric computations?
- How do we scale up data-centric compute?

Data Management

- How do we safely coordinate updates to data?
- How can we safely modify persisted data?

Growing your toolbelt



Template for most of this class

For some data oriented problem X...

- Define language that you can use to precisely describe problem X
  (what does it mean for a solution to be correct)
- Define language that you can use to rank solutions to problem X
  (what does it mean for one solution to be better)
- Explore the design space for problems similar to problem X
  (algorithms, data structures, etc... that might work, and their pros/cons)
- Automating this process safely and correctly
  (how to write a query compiler/runtime)

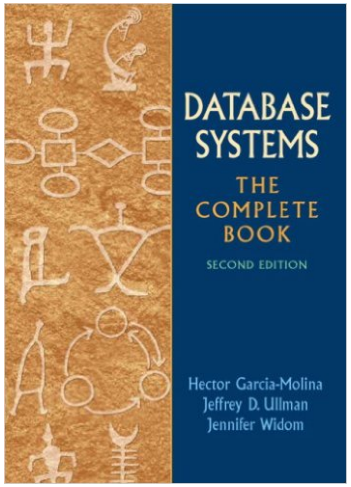
General Course Information

Expectations

I expect that you are familiar with...

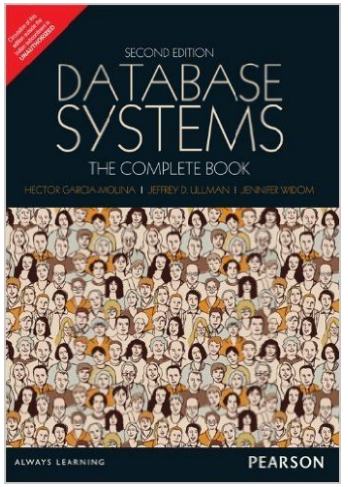
- ... **Data Structures**
  $O(\cdot)$ Analysis, Sort Algorithms, Search Trees, Heaps, Hash Tables
- ... **How to use a database**
  CSE 350, CSE 4/560, or similar
- ... **C++**
  Java, C++, Go, Rust, C#, C, or a similar systems language is usually sufficient.
- ... **Systems Programming**
  File IO, Casting structs to byte arrays, Manual memory management

Textbook



$175

Full Index and ToC



$45

Abbreviated Index and ToC

Grading

- 20% **Programming Assignments** (build a DB)
- 40% **Code/Journal Reviews** (more shortly)
- 15% **Midterm Exam** (03/05/26)
- 15% **Comprehensive Final Exam** (currently 05/11/26)
- 10% **In-Class Quizzes** (Random)

Failure to complete preliminary work in a timely manner will result in a grade of F:

- Checkpoint 0: **Jan 26**
- AI Quiz: **Jan 30**

I will bring candy to class. Correct question answers will be rewarded.

Especially good questions or insights *voiced during class* will also be rewarded.

TacoDB

Overview

- I hand you CSV data files and SQL.
- You hand me the results.
- Self-made groups of up-to-three

Checkpoints

0. **Setup**: Form groups and set up your build system.
1. **Intro**: C++ Primer; POSIX File IO.
2. **SQL**: Implement a subset of SQL.
3. **Optimizer**: Make SQL run fast.
4. **Constraints**: Make SQL run fast with constrained resources.
5. **Indexes**: Make SQL run even faster with prep time.

Starting with Checkpoint 2: Opt-in runtime leaderboard.

You may form groups of up to 3.

- Once selected, group changes will only be permitted in extreme circumstances.
- A groupfinding tool is available on Piazza.

Software Stack

- C++ / `cmake` / `ctest`
- Template code
  - SQL Parser
  - An AST framework / Expression language
- `git`
- I suggest `emacs` (with `clangd`) as an editor.
  - I will make every effort to assist you with other editors, but you are responsible for understanding how tools that you choose operate.

Typical Project Checkpoint

1. Merge in repository updates
2. Update objectives in your **journal** and explore design space.
3. Express objectives through one or more tests.
4. Get your code working.
5. Submit to **autolab**
   - If incomplete, reflect on it in your journal and develop a new test case.
   - Goto step 2 as needed.
6. Ensure that your journal documents your final design.
7. Schedule a code review meeting.

1-3 pages summarizing the project:

1. **Overview**: Provide a high-level overview of the system.
   - What does it mean for an implementation to be correct?
   - How can you subdivide the high-level objective into smaller ones.
   - What does it mean for each smaller objective to be correct?
   - What does it mean for an implementation/objective to be good?
2. **Design Space**: In your own words, what design choices do you have to make.
   - What algorithms, data structures, etc... could meet your correctness goals?
   - What are the pros/cons of each?
   - What did you ultimately choose?
3. **Iteration**: What challenges did you run into, and how did you address them?
   - What system component is behaving incorrectly?
   - Relate the observed problem to your correctness goals for the component.
   - How should you revise your idea of correctness and test for it?

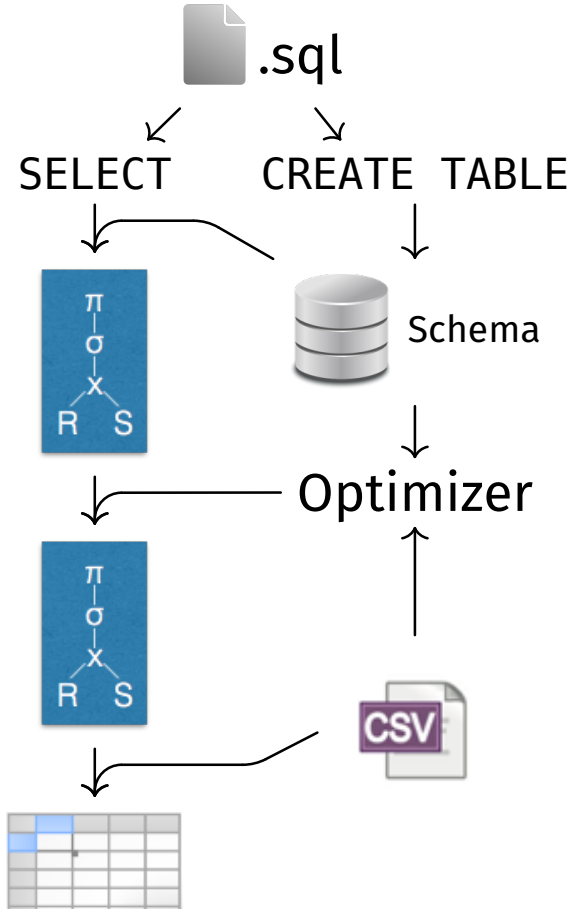
Project Grading

Checkpoint 0 must be completed by the deadline or you will receive a grade of F

Each checkpoint is worth 12 points total:

- **4 points**: Code submission (Team grading)
- **4 points**: In-person Code review (Individual grading)
- **4 points**: In-person Journal review (Individual grading)

Project Overview



General Advice

Ways to Fail

- Start your project at the last minute
- Never go to office hours
- Never ask questions (Piazza or in-class)
- Wait until the last minute to submit for the first time

... or ...

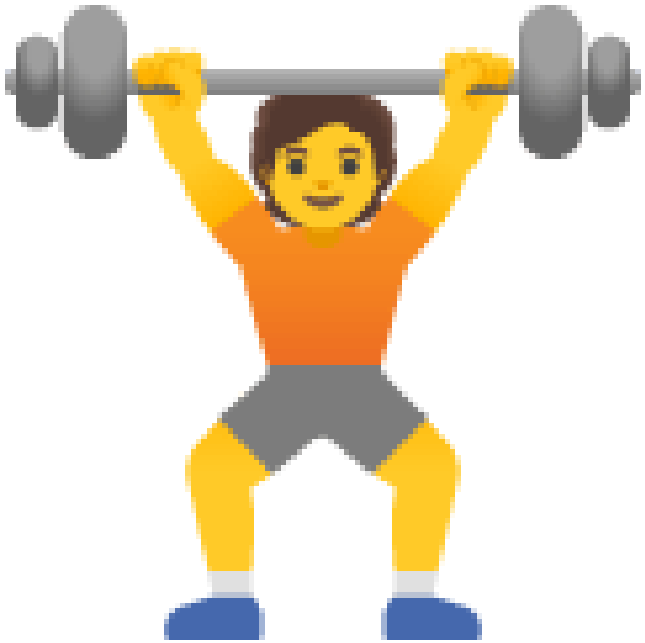
Academic Integrity

I’m here to help you learn…

… and to evaluate your understanding and ability.

Permitted Resources

- **Cited References** (Journal only)
  - Wikipedia, Wikibooks, CPPReference: **Ok**
- **Code Not Provided by Course Staff**
  - StackOverflow, ChatGPT, Copilot: **Not Ok**
- ***Discussing* concepts/ideas with classmates**
  - “A hash index has O(1) lookups”: **Ok** (except during exams 😇)
- ***Sharing* code or answers with anyone**
  - “Look at how I implemented it”: **Not Ok**



Official Policy

You may not use Generative AI when completing any deliverable for this class.

- No Chatbots (ChatGPT, Gemini)
- No Coding Assistants (Copilot, Gemini)

Enforcement

In-Person deliverables drive your grade

- 2/3 of your project grade is your ability to explain and justify your design.
- Significant differences in demonstrated understanding between in-person and take-home work will be grounds for an academic integrity sanction.

Zero Tolerance

If there is “a preponderance of evidence” to suggest that you violated academic integrity **you will fail the class**.

Group Responsibility

If your group submits work that violates academic integrity, **the entire group will be penalized**.

Share Code, Share Blame

If someone else submits your work or other course materials, **you will be penalized as well**.

Questions or concerns?

What are data management systems?

Analysis: Answering user-provided questions about data

What kind of tools can we give users?

- **Declarative Languages**: Separate the “what?” from the “how?”
- **Organizational Data Structures**: Indexes, Column Layouts
- **Optimizers**: Find faster ways to answer the same question

Manipulation: Safely persisting and sharing data updates

What kind of tools can we give users?

- **Transactions**: Declarative frameworks for data manipulation
- **Concurrency Primitives**: Rules for “correct” parallel updates
- **Incremental View Maintenance**: Tricks for monitoring complex properties

Filesystems vs Databases

vs

Filesystems vs Databases

VS

```
{"firstName": "John",
 "lastName": "Smith",
 "age": 25,
 "address": { "streetAddress": "21 2nd Street",
              "city": "New York",
              "state": "NY",
              "postalCode": 10021 },
 "phoneNumbers": [ { "type": "home",
                     "number": "212 555-1234" },
                   { "type": "fax",
                     "number": "646 555-4567" }]}
```

Databases exploit the data's structure

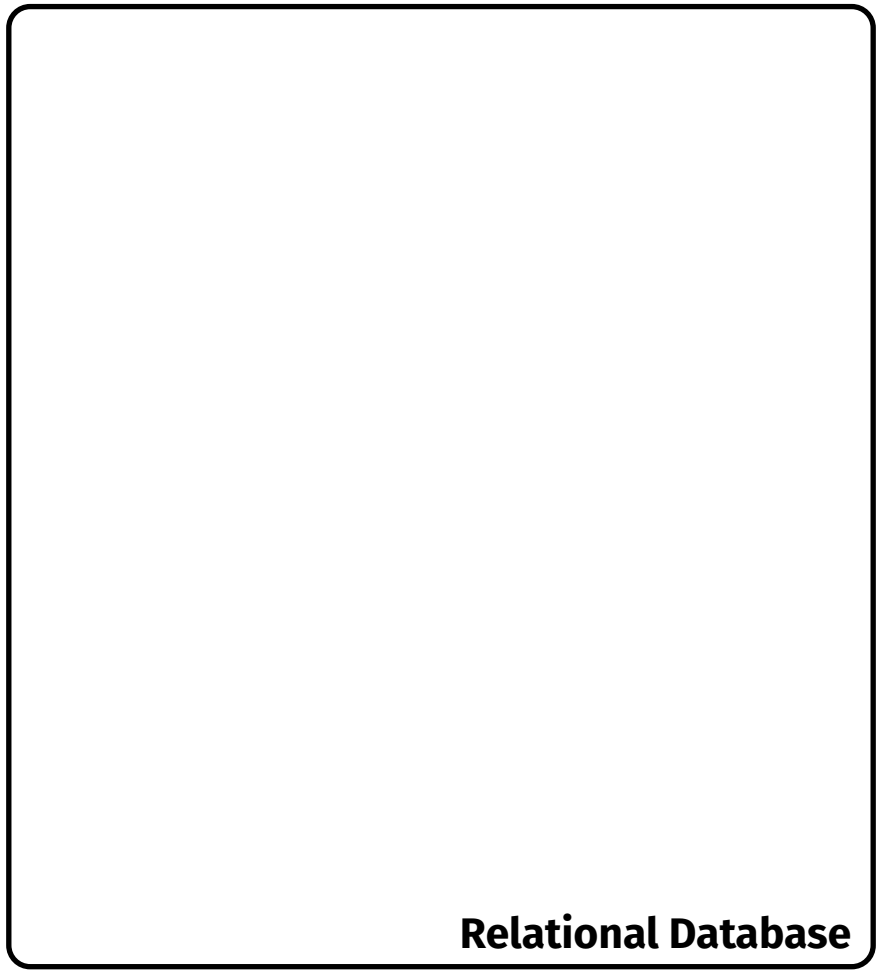
Singletons

- **Primitive**: Basic building blocks (Int, Float, Char, String).
- **Tuple**: Several 'fields' of different types (N-tuple has N fields)
  - A tuple has a "schema" defining the types (and names) of each field
  - Sometimes you'll hear me call this a "record"

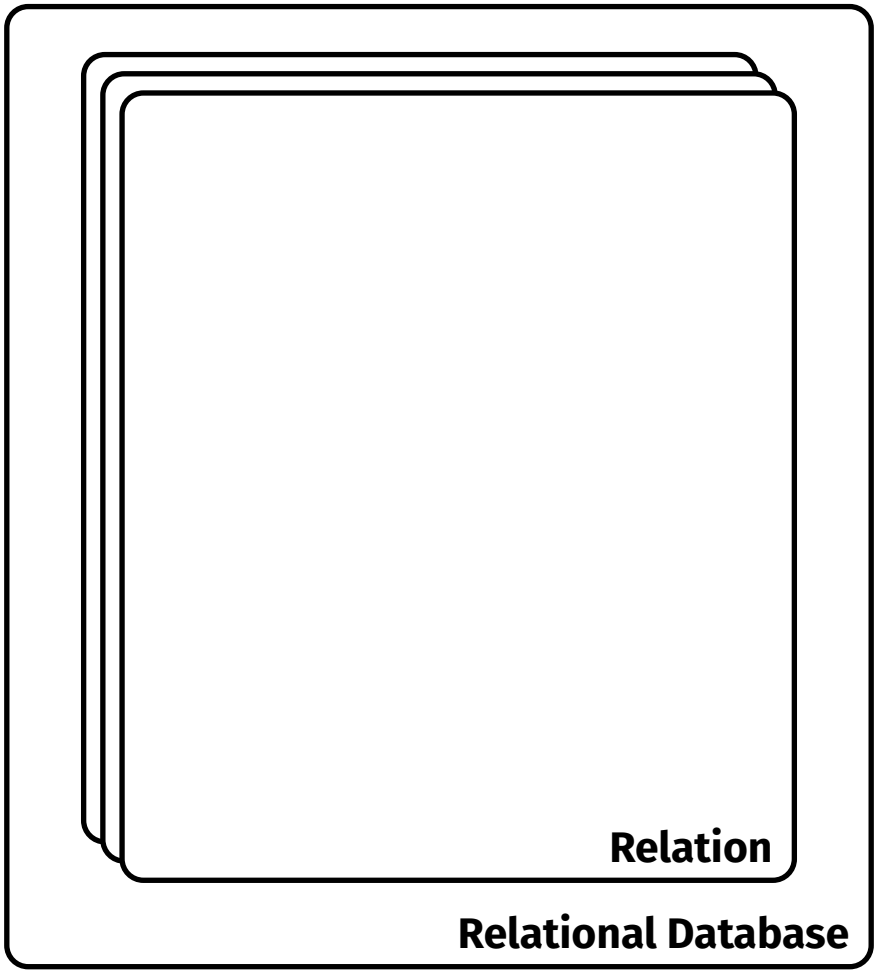
Collections

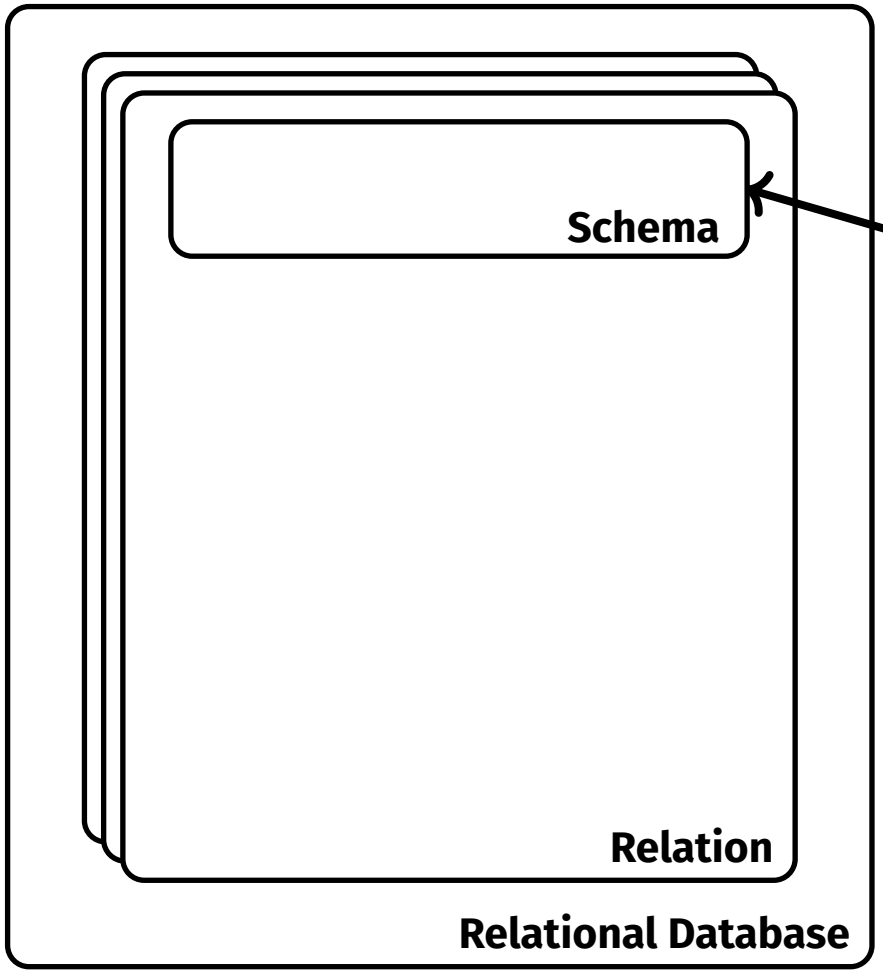
A collection of elements of the same type

- **Set**: All of the records are unique, but have no order.
- **Bag**: No constraints at all.
- **List**: Elements of the collection are arranged in a specific order.
- **Map**: Each element is identified by a unique "key"


Relational Database

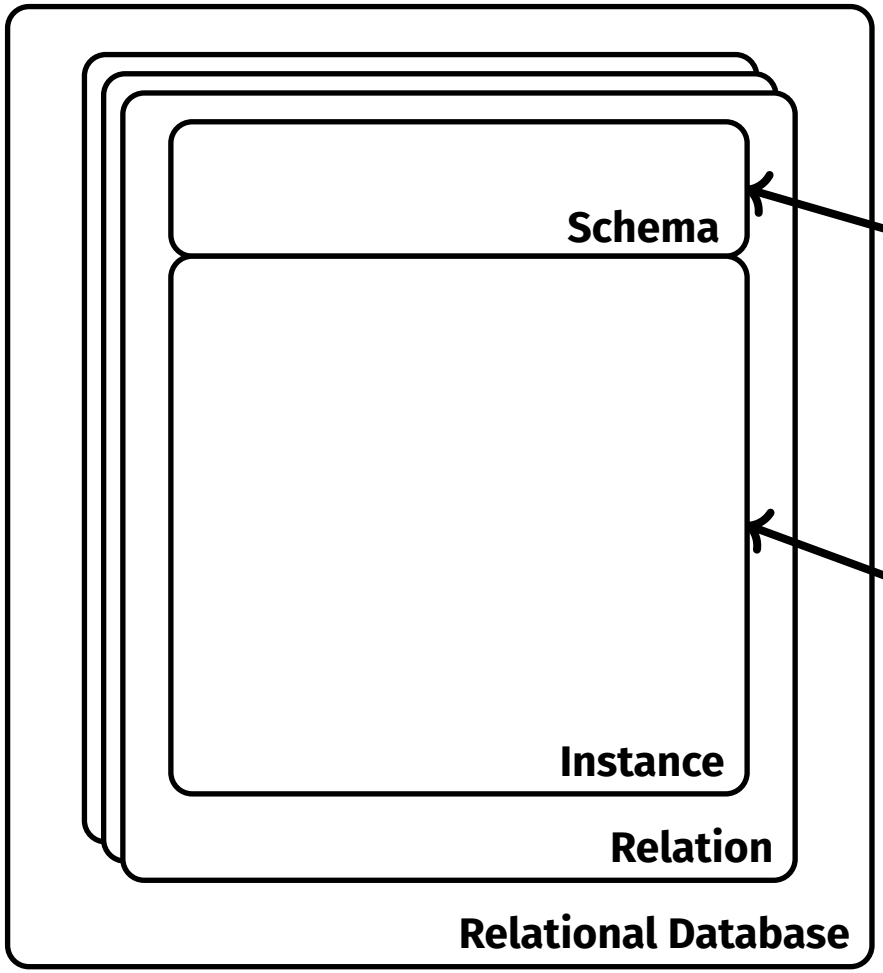
Relational Databases





Specifies the name of the relation, name and type of each column, and any other constraints

```
Officers(
  firstname string,
  lastname string,
  id int
)
```

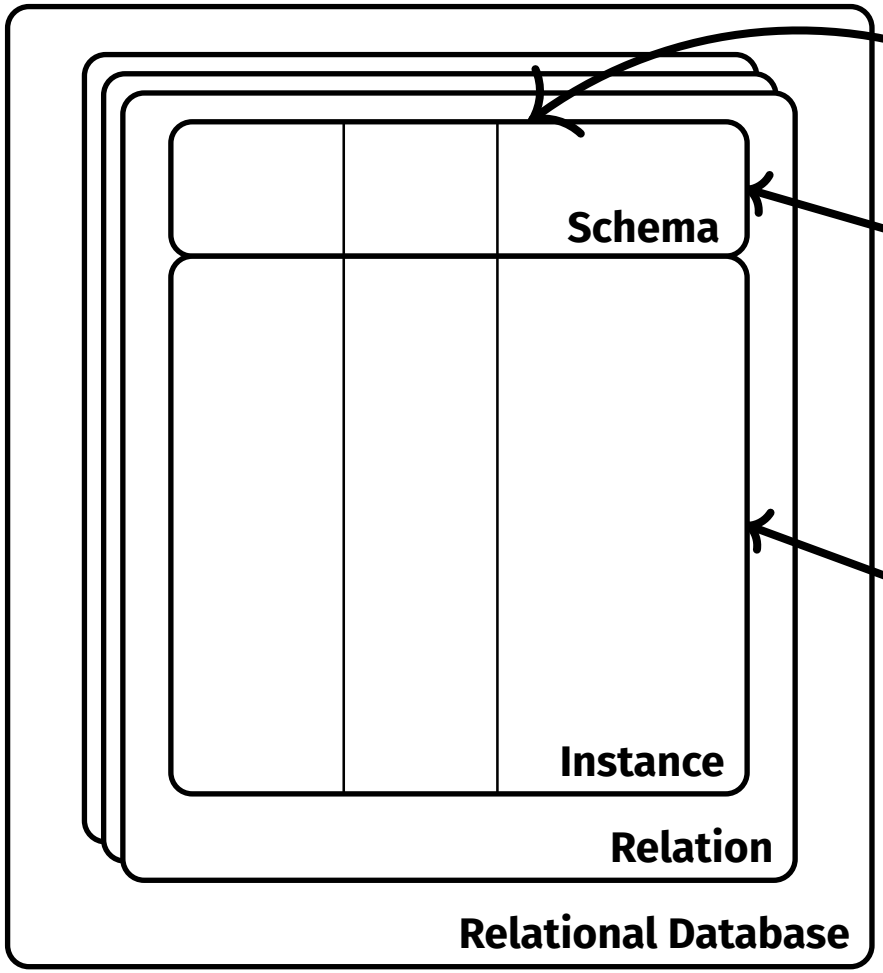


Specifies the name of the relation, name and type of each column, and any other constraints

```
Officers(
  firstname string,
  lastname string,
  id int
)
```

The Data

```
[Jean Luc, Picard, 2360]
[Benjamin, Sisko, 2365]
[Kathryn, Janeway, 2370]
```



Columns (degree/arity)

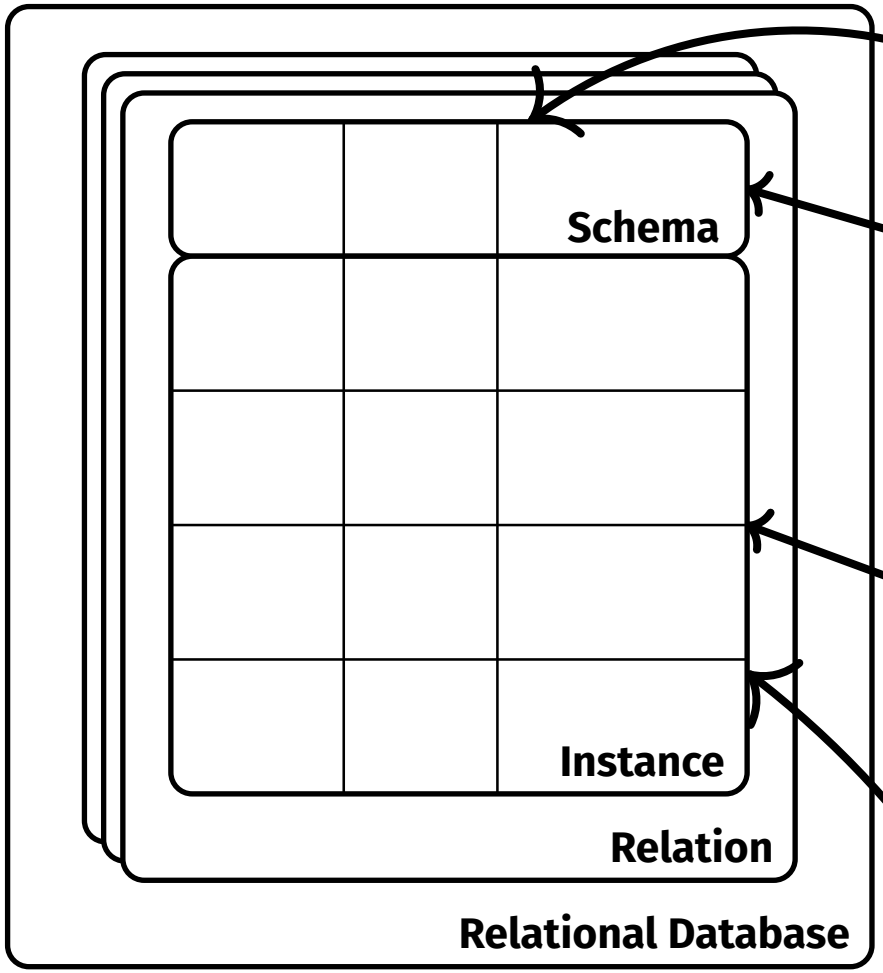
Specifies the name of the relation, name and type of each column, and any other constraints

```
Officers(
  firstname string,
  lastname string,
  id int
)
```

The Data

```
[Jean Luc, Picard, 2360]
[Benjamin, Sisko, 2365]
[Kathryn, Janeway, 2370]
```

Relational Databases



Columns (degree/arity)

Specifies the name of the relation, name and type of each column, and any other constraints

```
Officers(
  firstname string,
  lastname string,
  id int
)
```

The Data

```
[Jean Luc, Picard, 2360]
[Benjamin, Sisko, 2365]
[Kathryn, Janeway, 2370]
```

Rows (cardinality)

Example

Your data is currently an **Unordered Set** of **Tuples** with 100 attributes each.

Tomorrow, you will repeatedly be asked ...

- ...for the value of 1 specific attribute
- ...but only from 5 specific tuples identified by a second attribute

Can you do better than a CSV file?

Better Idea:

Rewrite the data into a 99-Tuple of Maps, with the first attribute as the key.

Declarative Query Languages

Write the query without thinking about:

- How the data is organized
- What algorithms you will use to answer the question
- What supplemental data structures you need/have available

Write the logic once, and the database adapts it to the current data organization.

SQL

- Developed by IBM (for System R) in the 1970s
- Standard used by many database implementations
  - **SQL-86**: Original standard
  - **SQL-89**: Minor revisions (e.g., integrity constraints)
  - **SQL-92**: Major revisions, basis for modern SQL
  - **SQL-99**: Support for XML, Window Queries, *Recursive CTEs*
  - **SQL-2003**: Major changes to XML support
  - **SQL-2008**: Minor extensions
  - **SQL-2011**: Temporal databases
  - **SQL-2016**: Support for JSON, listagg
  - **SQL-2023**: Better JSON standardization, Property Graphs

```
SELECT [DISTINCT]   target-list

FROM                relation-list

WHERE               condition
```

```
SELECT [DISTINCT]   target-list
```

A list of attributes from relations in `relation-list`

```
FROM                relation-list

WHERE               condition
```

```
SELECT [DISTINCT]   target-list
```

A list of attributes from relations in `relation-list`

```
FROM                relation-list
```

A list of relation names (with optional range variables)

```
WHERE               condition
```

```
SELECT [DISTINCT]   target-list
```

A list of attributes from relations in `relation-list`

```
FROM                relation-list
```

A list of relation names (with optional range variables)

```
WHERE               condition
```

Comparisons and other boolean predicates, combined with `AND`, `OR`, and `NOT`

(optional) keyword indicating the answer should be a set

```
SELECT [DISTINCT]   target-list
```

A list of attributes from relations in `relation-list`

```
FROM                relation-list
```

A list of relation names (with optional range variables)

```
WHERE               condition
```

Comparisons and other boolean predicates, combined with `AND`, `OR`, and `NOT`

```sql
CREATE TABLE R(
  A int,
  B int,
);
INSERT INTO R VALUES(1, 1);
INSERT INTO R VALUES(2, 2);
INSERT INTO R VALUES(3, 2);
```

```sql
CREATE TABLE S(
  B int,
  C char,
);
INSERT INTO S VALUES(2, 'a');
INSERT INTO S VALUES(2, 'b');
INSERT INTO S VALUES(3, 'c');
```

```sql
SELECT R.A, S.C
FROM R, S
WHERE R.B = S.B
```

SELECT: Step 1

Compute every combination of tuples from `relation-list`.

$R \bowtie S$	R.A	R.B	S.B	S.C
	1	1	2	a
	1	1	2	b
	1	1	3	c
	2	2	2	a
	2	2	2	b
	2	2	3	c
	3	2	2	a
	3	2	2	b
	3	2	3	c

SELECT: Step 2

Apply the rule from `condition`.

$R \bowtie S$	R.A	R.B	S.B	S.C
	~~1~~	~~1~~	~~2~~	~~a~~
	~~1~~	~~1~~	~~2~~	~~b~~
	~~1~~	~~1~~	~~3~~	~~c~~
	2	**2**	**2**	**a**
	2	**2**	**2**	**b**
	~~2~~	~~2~~	~~3~~	~~c~~
	3	**2**	**2**	**a**
	3	**2**	**2**	**b**
	~~3~~	~~2~~	~~3~~	~~c~~

SELECT: Step 3

Keep only columns listed in `target-list`

$R \bowtie S$	R.A	S.C
	2	a
	2	b
	3	a
	3	b

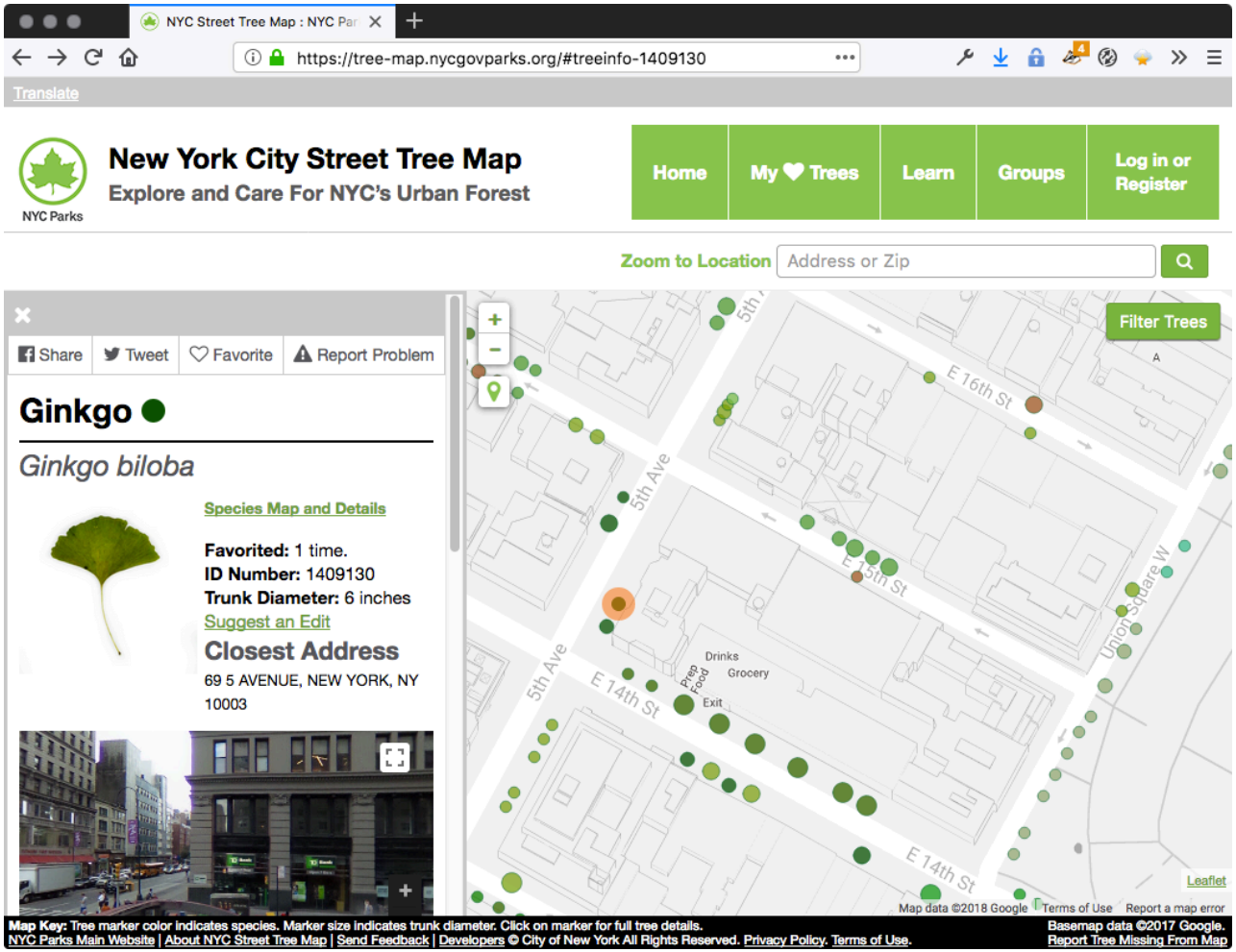
1. Compute every combination of tuples from `relation-list`.
2. Apply the rule from `condition`.
3. Keep only columns listed in `target-list`
4. Delete duplicates if `DISTINCT` is specified.

In general, **do not do this.**

We use an inefficient language to define the “what” because it is general.

A good optimizer will often find a better way (the “how”) if one exists.

Example Data



Example 1

Wildcards (`*`, `tablename.*`) are special targets that select all attributes.

```
SELECT * FROM Trees;
```

T	TREE_ID	BLOCK_ID	THE_GEOM	TREE_DBH	STUMP_DIAM	CURB_LOC	STATUS	HEALTH	SPC_LATIN	SPC_COMMON	STEWARD	GUARDS	SIDEWALK	USER_TYPE	PROBLEMS	ROOT_STONE	ROOT_GR
'	180683	348711	'POINT (-73.84421521958048 40.723091773924274)'	3	0	'OnCurb'	'Alive'	'Fair'	'Acer rubrum'	'red maple'	'None'	'None'	'NoDamage'	'TreesCount Staff'	'None'	'No'	'No'
'	200540	315986	'POINT (-73.81867945834878 40.79411066708779)'	21	0	'OnCurb'	'Alive'	'Fair'	'Quercus palustris'	'pin oak'	'None'	'None'	'Damage'	'TreesCount Staff'	'Stones'	'Yes'	'No'
'	204026	218365	'POINT (-73.93660770459083 40.717580740099116)'	3	0	'OnCurb'	'Alive'	'Good'	'Gleditsia triacanthos var. inermis'	'honeylocust'	'1or2'	'None'	'Damage'	'Volunteer'	'None'	'No'	'No'
'	204337	217969	'POINT (-73.93445615919741 40.713537494833226)'	10	0	'OnCurb'	'Alive'	'Good'	'Gleditsia triacanthos var. inermis'	'honeylocust'	'None'	'None'	'Damage'	'Volunteer'	'Stones'	'Yes'	'No'
'	189565	223043	'POINT (-73.97597938483258 40.66677775537875)'	21	0	'OnCurb'	'Alive'	'Good'	'Tilia americana'	'American linden'	'None'	'None'	'Damage'	'Volunteer'	'Stones'	'Yes'	'No'

... and 683783 more

Example 2

In English, what does the following query compute?

```
SELECT tree_id, spc_common, boroname
FROM Trees
WHERE boroname = 'Brooklyn'
```

Example 2

What is the ID, Common Name and Borough of Trees in Brooklyn?

Q	TREE_ID	SPC_COMMON	BORONAME
	204026	'honeylocust'	'Brooklyn'
	204337	'honeylocust'	'Brooklyn'
	189565	'American linden'	'Brooklyn'

... and 177289 more

Example 3

In English, what does the following query compute?

```
SELECT latitude, longitude
FROM Trees, SpeciesInfo
WHERE Trees.spc_common = SpeciesInfo.name
  AND SpeciesInfo.has_unpleasant_smell = 'Yes';
```

Example 3

What are the coordinates of Trees with bad smells?

Q	LATITUDE	LONGITUDE
	40.59378755	−73.9915968
	40.69149917	−73.97258754
	40.74829709	−73.98065645
	40.68767857	−73.96764605
	40.739991	−73.86526993

... and more

Example 3

Range variables make it easier to refer to tables.

```
SELECT Trees.latitude, Trees.longitude
FROM Trees, SpeciesInfo
WHERE Trees.spc_common = SpeciesInfo.name
  AND SpeciesInfo.has_unpleasant_smell = 'Yes';
```

Example 3

Range variables make it easier to refer to tables.

```
SELECT Trees.latitude, Trees.longitude
FROM Trees, SpeciesInfo
WHERE Trees.spc_common = SpeciesInfo.name
  AND SpeciesInfo.has_unpleasant_smell = 'Yes';
```

... is the same as ...

```
SELECT T.latitude, T.longitude
FROM Trees T, SpeciesInfo S
WHERE T.spc_common = S.name
  AND S.has_unpleasant_smell = 'Yes';
```

Range variables make it easier to refer to tables.

```
SELECT Trees.latitude, Trees.longitude
FROM Trees, SpeciesInfo
WHERE Trees.spc_common = SpeciesInfo.name
  AND SpeciesInfo.has_unpleasant_smell = 'Yes';
```

... is the same as ...

```
SELECT T.latitude, T.longitude
FROM Trees T, SpeciesInfo S
WHERE T.spc_common = S.name
  AND S.has_unpleasant_smell = 'Yes';
```

... is (usually) the same as ...

```
SELECT latitude, longitude
FROM Trees, SpeciesInfo
WHERE spc_common = name
  AND has_unpleasant_smell = 'Yes';
```

Primitive-valued **expressions** can appear as targets. They define new columns.

```
SELECT tree_id,
       stump_diam / 2 AS stump_radius,
       stump_area = 3.14 * stump_diam * stump_diam / 4
FROM Trees;
```

Use `=` or `AS` to assign names to these attributes.

If you do not specify a name, the database will pick one for you. Often this is meant to be human-readable, but the SQL spec only says what to do when there is a bare reference to an attribute.

```
SELECT tree_id, spc_common FROM Trees WHERE spc_common LIKE '%maple'
```

SQL uses **single quotes** for string literals. Use `''` to escape a quote.

`LIKE` is used for string matches (`%` matches 0 or more characters)

`UNION` merges the rows of two queries.

```
SELECT tree_id FROM Trees WHERE spc_common = 'red maple'
UNION [ALL]
SELECT tree_id FROM Trees WHERE spc_common = 'sycamore maple'
```

By the spec, the queries must be **union compatible**, although some engines are more flexible.

`UNION` computes a **set** (no duplicates), while `UNION ALL` computes a bag (duplicates allowed).

Aggregate Queries

```
SELECT [DISTINCT] target-list
FROM relation-list
WHERE condition
GROUP BY grouping-list
HAVING group-condition
```

The `target-list` now contains **grouped attributes** and **aggregate expressions**.

Grouped attributes must appear in the `grouping-list`.

`group-condition` is applied **after** aggregation, and may use aggregate expressions.

Aggregate Queries

```
SELECT spc_common, count(*) FROM Trees GROUP BY spc_common
```

Q	SPC_COMMON	COUNT
	'Schubert chokecherry'	4888
	'American beech'	273
	'American elm'	7975
	'American hophornbeam'	1081
	'American hornbeam'	1517

... and more

Reminders

Reminders

- In-class lectures start with the second lecture next **Tuesday**.
- Complete the AI Quiz on Autolab.
- Make a group and complete Checkpoint 0.