DATA516/CSED516
Scalable Data Systems and Algorithms

Lecture 1

Relational Model, SQL

Course Staff

- Instructor: Jack Khuu
  jackkhuu@cs.washington.edu

- TA: Punya Prakash Shetty
  punya97@cs.washington.edu

- TA: Nayan Kaushal
  nakaush@uw.edu

COVID-19 UW Policy

- UW recommends everyone wearing a mask in the classroom.

Course Aims

- Study design of big data systems
  - Historical perspective
  - Sample of modern systems
  - Breadth of designs (relational, streaming, graph, etc.)

- Study scalable data processing algorithms

- Gain hands-on experience with big data systems

Course Content

- Query processing: single-sever, distributed
- MapReduce, successors
- Streaming, Column Stores, Graph engines
- See the calendar on the course website (subject to change)

Course Format

- 5pm-7:50pm: Lectures
- 8pm-8:50pm: Section
  - Bring your laptop!
- Office hours: by zoom only

See the course website

Grading

- 15%: Reading assigned papers
- 60%: Homework assignments
- 25%: Final project

Homeworks

- HW1: Amazon Redshift
- HW2: Spark/AWS
- HW3: Snowflake
- HW4: mini-homeworks – stay tuned

Save free credits for the project!

Project

Choose a topic:

- Don't worry about novelty
- Recommended: Benchmark projects
- Other ideas are welcome too
- I posted a few ideas, but you are encouraged to come up with your own

See the course website

Communication

- Course webpage: all important stuff https://courses.cs.washington.edu/courses/csed516/22au/
- Discussion Board: Canvas
- Class email: only for important announcements

How to Turn In

Homework and project:

- https://gitlab.cs.washington.edu/

Reviews

- Canvas

See the course website

Ice Breaker

Now onward to the world of databases!

Quick Review

- Database = a collection of files
  - Examples: products database; movies database

- Database management system (DBMS) = a piece of software to help manage that data
  - Examples: Postgres, Oracle, sqlite

DBMS Functionality

- DBMS does many things:
  - Complex queries, updates, concurrency, recovery, access control, integrity checks, data distribution, etc, etc

- Some DBMS are more specialized for some tasks than others

DBMS Architectures and Workloads

Single Client

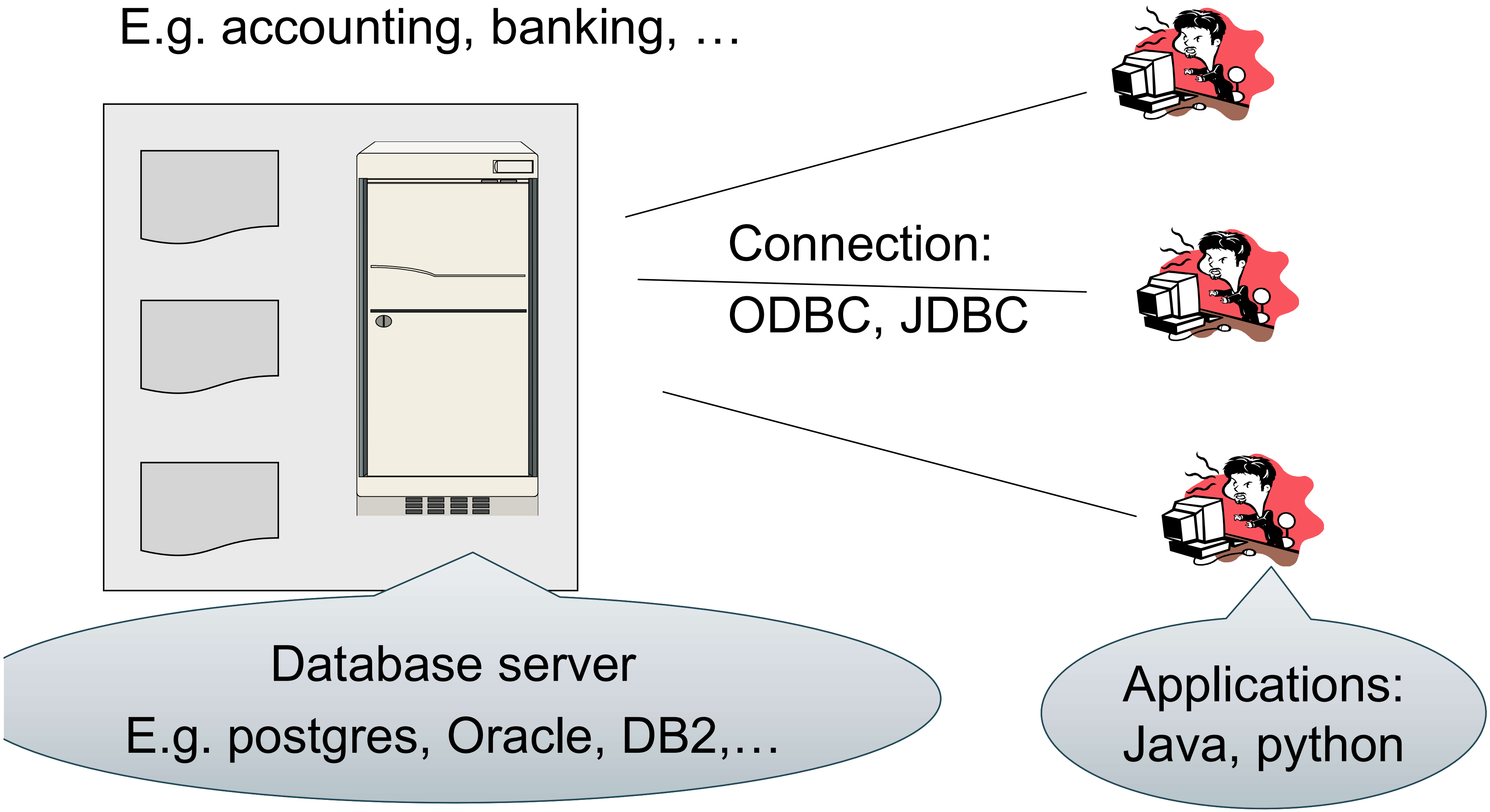
E.g. data analytics

Application and database
on the same computer

E.g. sqlite, postgres

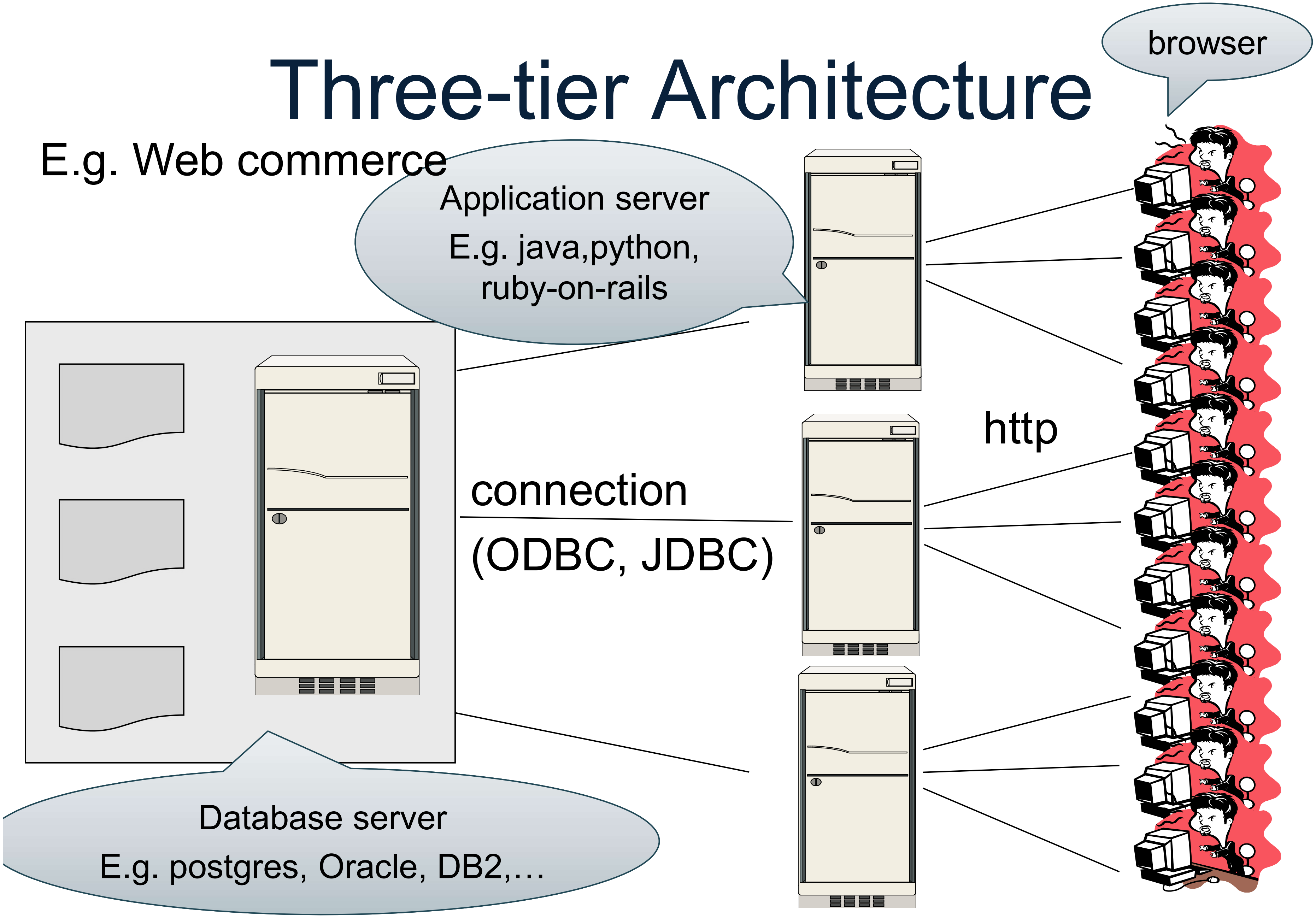
Two-tier Architecture
Client-Server

E.g. accounting, banking, …

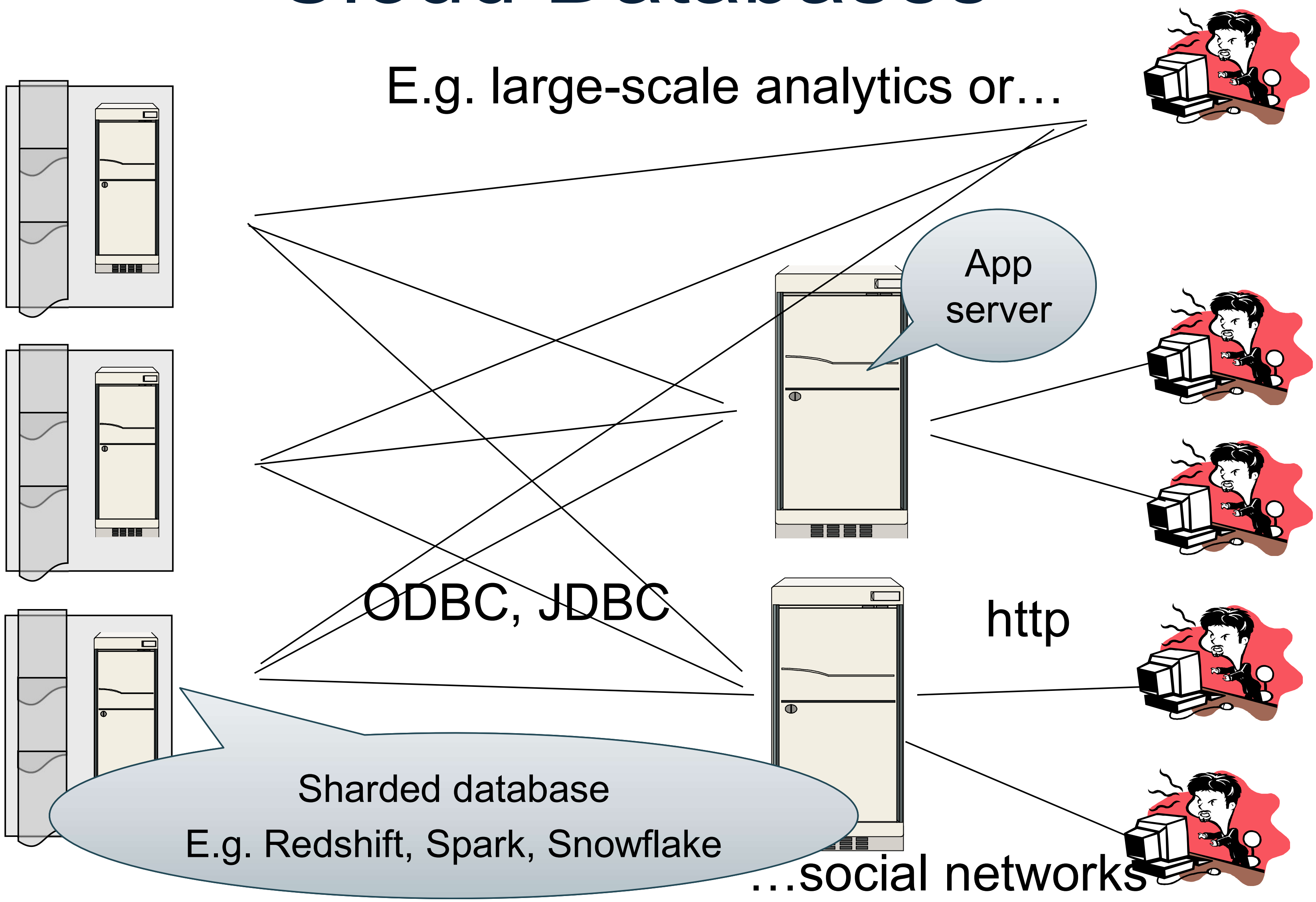


Three-tier Architecture

E.g. Web commerce



Cloud Databases



Workloads

- OLTP – online transaction processing
  - Not interesting for data science
- OLAP – online analytics processing, a.k.a. Decision Support
  - Critical for scalable data science

Relational Data Model

Relational Data Model

Modeling the data: schema + data

- Database = collection of relations
- Relation (a.k.a. table) = a set of tuples
- A Tuple (row, record) = $(v_1, \ldots, v_n)$

Modeling the query:

- Set-at-a-time, relational query language

Schema

- Relation schema: describes column heads
  - Relation name
  - Name of each field (or column, or attribute)
  - Domain of each field
  - The *arity* of the relation = # attributes

- Database schema: set of all relation schemas

Instance

- Relation instance: concrete table content
  - Set of records matching the schema
  - The *cardinality* or *size* of the relation = # tuples
- Database instance: set of relation instances

What is the schema?
What is the instance?

Supplier

sno	**sname**	**scity**	**sstate**
1005	ACME	Seattle	WA
1006	Freddie	Austin	TX
1007	Joe's	Seattle	WA
1008	ACME	Austin	TX

What is the schema?
What is the instance?

Schema

Supplier(sno: integer, sname: string, scity: string, sstate: string)

Supplier

sno	sname	scity	sstate
1005	ACME	Seattle	WA
1006	Freddie	Austin	TX
1007	Joe’s	Seattle	WA
1008	ACME	Austin	TX

instance

What is the schema?
What is the instance?

Schema

Supplier(<u>sno: integer</u>, sname: string, scity: string, sstate: string)

Supplier

sno	**sname**	**scity**	**sstate**
1005	ACME	Seattle	WA
1006	Freddie	Austin	TX
1007	Joe's	Seattle	WA
1008	ACME	Austin	TX

instance

In class: discuss keys, foreign keys, FD

Discussion

- Rows in a relation:
  - Ordering immaterial (a relation is a set)
  - All rows are distinct – set semantics
  - Query answers may have duplicates – bag semantics

Data independence!

Discussion

Data independence!

- Rows in a relation:
  - Ordering immaterial (a relation is a set)
  - All rows are distinct – set semantics
  - Query answers may have duplicates – bag semantics

- Columns in a tuple:
  - Ordering is immaterial

    Or is it?
  - Applications refer to columns by their names

Discussion

- Rows in a relation:
  - Ordering immaterial (a relation is a set)
  - All rows are distinct – set semantics
  - Query answers may have duplicates – bag semantics

Data independence!

- Columns in a tuple:
  - Ordering is immaterial
  - Applications refer to columns by their names

Or is it?

- Each Domain = a primitive type; no nesting!

Relational Query Language

- Set-at-a-time:
  - Inputs and outputs are relations
  - Contrast with python/Julia/java/etc: tuple-at-a-time

- Examples:
  - SQL, Relational Algebra, datalog, various graph query languages (Sparql, TigerGraph)

SQL

SQL

- Standard query language
- Introduced late 70's, now it ballooned
- We briefly review "core SQL" (whatever that means); study more on your own!
- Review: *A case against SQL*

Structured Query Language: SQL

- Data definition language: DDL
  - CREATE TABLE …,
    CREATE VIEW …,
    ALTER TABLE…

Review
on your own

- Data manipulation language: DML
  - SELECT-FROM-WHERE…,
    INSERT…,
    UPDATE…,
    DELETE…

Our focus

SQL Query

```
SELECT  <attributes>
FROM    <one or more relations>
WHERE   <conditions>
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Quick Review of SQL

```
SELECT   *
FROM     Part
WHERE    pcolor = 'red'
```

What do these queries compute?

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Quick Review of SQL

```
SELECT  *
FROM    Part
WHERE   pcolor = 'red'
```

```
SELECT  x.sno, x.name
FROM    Supplier x
WHERE   x.sstate = 'WA'
```

What do these queries compute?

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Quick Review of SQL

```
SELECT DISTINCT  z.pno, z.pname, x.scity
FROM       Supplier x, Supply y, Part z
WHERE    x.sno = y.sno
         and y.pno = z.pno
         and x.sstate = 'WA'
         and y.price < 100
```

What does this query compute?

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Terminology

- Selection/filter: e.g. … WHERE scity='Seattle'
- Projection: e.g. SELECT sname …
- Join: e.g. …FROM Supplier, Supply, Part …

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y.pno
FROM       Supplier x, Supply y
WHERE      x.scity = 'Seattle'
           and x.scity = 'Portland'
           and x.sno = y.sno
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y.pno
FROM      Supplier x, Supply y
WHERE     x.scity = 'Seattle'
          and x.scity = 'Portland'
          and x.sno = y.sno
```

This doesn't work…

Why?

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y.pno
FROM     Supplier x, Supply y
WHERE    (x.scity = 'Seattle'
         or x.scity = 'Portland')
       and x.sno = y.sno
```

Does this work?

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y.pno
FROM      Supplier x, Supply y
WHERE     (x.scity = 'Seattle'
            or x.scity = 'Portland')
          and x.sno = y.sno
```

Does this work?

Nope!

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

Need TWO Suppliers and TWO Supplies

```
SELECT DISTINCT  y1.pno
FROM     Supplier x1, Supplier x2, Supply y1, Supply y2
WHERE    x1.scity = 'Seattle'
         and x1.sno = y1.sno
         and x2.scity = 'Portland'
         and x2.sno = y2.sno
         and y1.pno = y2.pno
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

```sql
SELECT DISTINCT  y1.pno
FROM     Supplier x1, Supplier x2, Supply y1, Supply y2
WHERE     x1.scity = 'Seattle'
        and x1.sno = y1.sno
        and x2.scity = 'Portland'
        and x2.sno = y2.sno
        and y1.pno = y2.pno
```

Need TWO Suppliers and TWO Supplies

one in Seattle the other in Portland

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y1.pno
FROM      Supplier x1, Supplier x2, Supply y1, Supply y2
WHERE     x1.scity = 'Seattle'
          and x1.sno = y1.sno
          and x2.scity = 'Portland'
          and x2.sno = y2.sno
          and y1.pno = y2.pno
```

Need TWO Suppliers and TWO Supplies

one in Seattle the other in Portland

the SAME part

Semantics

Semantics

- What does a SQL query compute?
- Simple semantics:
  - *Nested Loop Semantics*
- Allows optimizations
  - *Physical data independence*

Nested-Loop Semantics of SQL

```
SELECT a_1, a_2, …, a_k
FROM   R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE  Conditions
```

Nested-Loop Semantics of SQL

```
SELECT a_1, a_2, …, a_k
FROM   R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE  Conditions
```

SELECT $a_1, a_2, \ldots, a_k$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE Conditions

Answer = {}

Nested-Loop Semantics of SQL

SELECT $a_1, a_2, \ldots, a_k$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE Conditions

Answer = {}
for x_1 **in** R_1 **do**

Nested-Loop Semantics of SQL

SELECT $a_1, a_2, \ldots, a_k$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE Conditions

Answer = {}
for x_1 **in** R_1 **do**
    for x_2 **in** R_2 **do**

Nested-Loop Semantics of SQL

```
SELECT a_1, a_2, …, a_k
FROM   R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE  Conditions
```

SELECT $a_1, a_2, \ldots, a_k$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE Conditions

Answer = {}
for x_1 **in** R_1 **do**
    for x_2 **in** R_2 **do**
        …..
            for x_n **in** R_n **do**

Nested-Loop Semantics of SQL

```
SELECT a1, a2, …, ak
FROM   R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE  Conditions
```

SELECT $a_1, a_2, \ldots, a_k$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE Conditions

Answer = {}
for x_1 **in** R_1 **do**
    for x_2 **in** R_2 **do**
        …..
            for x_n **in** R_n **do**
                if Conditions
                    then Answer = Answer $\cup$ $\{(a_1,\ldots,a_k)\}$
return Answer

Nested-Loop Semantics of SQL

```
SELECT a_1, a_2, ..., a_k
FROM   R_1 AS x_1, R_2 AS x_2, ..., R_n AS x_n
WHERE  Conditions
```

This SEMANTICS!
It says what it means.
Doesn't say how to get it

Answer = {}
for x_1 **in** R_1 **do**
    for x_2 **in** R_2 **do**
        .....
            for x_n **in** R_n **do**
                if Conditions
                    then Answer = Answer $\cup$ $\{(a_1,\ldots,a_k)\}$
return Answer

Nested-Loop Semantics of SQL

SELECT $a_1, a_2, \ldots, a_k$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE Conditions

This SEMANTICS!
It says what it means.
Doesn't say how to get it

Data Independence

Answer = {}
for x_1 **in** R_1 **do**
  for x_2 **in** R_2 **do**
    …..
      for x_n **in** R_n **do**
        if Conditions
          then Answer = Answer $\cup$ $\{(a_1,\ldots,a_k)\}$
return Answer

Data Independence

```
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Physical Data Independence

- The query is written independently of how it will be evaluated
- We write what data we want; optimizer decides how to get it

Discuss in class how

```
SELECT.  *
FROM     Supply y, Part z
WHERE    y.price = 100 and z.pcolor = 'red' and y.pno = z.pno
```

Discussion

- Data independence is the main reason why the relational data model is the dominant data model today

- Reading next week: What Goes Around

NULL

`Part(pno,pname,price,psize,pcolor)`

NULLs in SQL

- A NULL value means missing, or unknown, or undefined, or inapplicable
- Common in Data Science
- The key should never be NULL

pno	pname	price	psize	pcolor
1	iPad	500	13	blue
2	Scooter	99	NULL	NULL
3	Charger	NULL	NULL	red
4	iPad	50	2	NULL

`Part(pno,pname,price,psize,pcolor)`

NULLs in WHERE Clause

Predicate in WHERE Clause

- Atomic: e.g. pcolor = ‘red’
- AND / OR / NOT

When is the WHERE condition satisfied?

`Part(pno,pname,price,psize,pcolor)`

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- pcolor = ‘red’
  - False or True when pcolor is not NULL
  - Unknown when pcolor is NULL
- AND, OR, NOT are min, max, 1- …

WHERE condition: returns the tuple when True

`Part(pno,pname,price,psize,pcolor)`

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- pcolor = ‘red’
  - False or True when pcolor is not NULL
  - Unknown when pcolor is NULL
- AND, OR, NOT are min, max, 1- …

WHERE condition: returns the tuple when True

```
select *
from Part
where price < 100
 and (psize=2 or pcolor=‘red’)
```

`Part(pno,pname,price,psize,pcolor)`

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- pcolor = 'red'
  - False or True when pcolor is not NULL
  - Unknown when pcolor is NULL
- AND, OR, NOT are min, max, 1- …

WHERE condition: returns the tuple when True

```
select *
from Part
where price < 100
 and (psize=2 or pcolor='red')
```

pno	pname	price	psize	pcolor
1	iPad	500	13	blue
2	Scooter	99	NULL	NULL
3	Charger	NULL	NULL	red
4	iPad	50	2	NULL

`Part(pno,pname,price,psize,pcolor)`

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- pcolor = ‘red’
  - False or True when pcolor is not NULL
  - Unknown when pcolor is NULL
- AND, OR, NOT are min, max, 1- …

WHERE condition: returns the tuple when True

```
select *
from Part
where price < 100
 and (psize=2 or pcolor=‘red’)
```

pno	pname	price	psize	pcolor
1	iPad	500	13	blue
2	Scooter	99	NULL	NULL
3	Charger	NULL	NULL	red
4	iPad	50	2	NULL

Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- pcolor = ‘red’
  - – False or True when pcolor is not NULL
  - – Unknown when pcolor is NULL
- AND, OR, NOT are min, max, 1- …

WHERE condition: returns the tuple when True

```
select *
from Part
where price < 100
 and (psize=2 or pcolor=‘red’)
```

pno	pname	price	psize	pcolor
1	iPad	500	13	blue
2	Scooter	99	NULL	NULL
3	Charger	NULL	NULL	red
4	iPad	50	2	NULL

`Part(pno,pname,price,psize,pcolor)`

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- pcolor = ‘red’
  - False or True when pcolor is not NULL
  - Unknown when pcolor is NULL
- AND, OR, NOT are min, max, 1- …

WHERE condition: returns the tuple when True

```
select *
from Part
where price < 100
 and (psize=2 or pcolor=‘red’)
```

pno	pname	price	psize	pcolor
1	iPad	500	13	blue
2	Scooter	99	NULL	NULL
3	Charger	NULL	NULL	red
4	iPad	50	2	NULL

`Part(pno,pname,price,psize,pcolor)`

Three-Valued Logic

- Problem: A or not(A) ≠ true

```
select *
from Part
where (price <= 100) or (price > 100)
```

Does it return all parts?

`Part(pno,pname,price,psize,pcolor)`

Three-Valued Logic

- Problem: A or not(A) ≠ true

```
select *
from Part
where (price <= 100) or (price > 100)
```

Does it return all parts?

```
-- solution to return all parts:
select *
from Part
where (price <= 100) or (price > 100) or isNull(price)
```

Aggregates

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Examples

What do they compute?

```
SELECT count(*)
FROM   Part
```

Supplier(sno,sname,scity,sstate)

Supply(sno,pno,qty,price)

Part(pno,pname,psize,pcolor)

Examples

What do they compute?

```
SELECT count(*)
FROM   Part
```

```
SELECT x.scity, avg(psize)
FROM   Supplier x, Supply y, Part z
WHERE  x.sno = y.sno  and y.pno = z.pno
GROUP BY x.scity
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Examples

What do they compute?

```
SELECT count(*)
FROM   Part
```

```
SELECT x.scity, avg(psize)
FROM   Supplier x, Supply y, Part z
WHERE  x.sno = y.sno  and y.pno = z.pno
GROUP BY x.scity
```

```
SELECT x.scity, avg(psize)
FROM   Supplier x, Supply y, Part z
WHERE  x.sno = y.sno  and y.pno = z.pno
GROUP BY x.scity
HAVING count(*) > 200
```

Discussion

- Aggregates = important for data science!
- Semantics:
  1. FROM-WHERE (nested-loop semantics)
  2. GROUP BY attrs
  3. Apply HAVING predicates on groups
  4. Apply SELECT aggregates on groups
- count, sum, min, max, avg
- DISTINCT is special case of GROUP BY

Outer Joins

Product(name, category)

Purchase(prodName, store)

prodName is foreign Key

Outer joins

Retrieve all products and stores.
Include products that never sold

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

```
Product(name, category)
Purchase(prodName, store)
```

prodName is foreign Key

Outer joins

Retrieve all products and stores.
Include products that never sold

```
SELECT x.name, x.category, y.store
FROM   Product x, Purchase y
WHERE  x.name = y.prodName
```

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

```
Product(name, category)
Purchase(prodName, store)
```

prodName is foreign Key

Outer joins

Retrieve all products and stores.
Include products that never sold

```
SELECT x.name, x.category, y.store
FROM   Product x, Purchase y
WHERE  x.name = y.prodName
```

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

missing

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Output

Name	Category	Store
Gizmo	gadget	Wiz
Camera	Photo	Ritz
Camera	Photo	Wiz

```
Product(name, category)
Purchase(prodName, store)
```

prodName is foreign Key

Outer joins

Retrieve all products and stores.
Include products that never sold

```
SELECT x.name, x.category, y.store
FROM   Product x LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
```

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Output

Name	Category	Store
Gizmo	gadget	Wiz
Camera	Photo	Ritz
Camera	Photo	Wiz
OneClick	Photo	NULL

Now it's present

Left Outer Join (Details)

from R left outer join S on C1 where C2

1. Compute cross product R×S
2. Filter on C1
3. Add all R records without a match
4. Filter on C2

Joins

- Inner join
- Left outer join
- Right outer join
- Full outer join

SQL: Beyond Relations

Beyond Relations

- Sparse vectors, matrices
- Graph databases
- Important to data science!

Sparse Matrix

$$A = \begin{bmatrix} 5 & 0 & -2 \\ 0 & 0 & -1 \\ 0 & 7 & 0 \end{bmatrix}$$

How can we represent it as a relation?

Sparse Matrix

$$A = \begin{bmatrix} 5 & 0 & -2 \\ 0 & 0 & -1 \\ 0 & 7 & 0 \end{bmatrix}$$

Row	Col	Val
1	1	5
1	3	-2
2	3	-1
3	2	7

Matrix Multiplication in SQL

$C = A \cdot B$

Matrix Multiplication in SQL

$$C = A \cdot B$$

$$C_{ik} = \sum_{j} A_{ij} \cdot B_{jk}$$

Matrix Multiplication in SQL

$$C = A \cdot B$$

$$C_{ik} = \sum_j A_{ij} \cdot B_{jk}$$

```
SELECT A.row, B.col, sum(A.val*B.val)
FROM A, B
WHERE A.col = B.row
GROUP BY A.row, B.col;
```

Discussion

Matrix multiplication = join + group-by

- Try at home: write in SQL

  $$Tr(A \cdot B \cdot C)$$

  where the trace is defined as:

  $$Tr(X) = \sum_i X_{ii}$$

Surprisingly, $A + B$ is a bit harder…

Matrix Addition in SQL

$$C = A + B$$

Matrix Addition in SQL

$$C = A + B$$

```
SELECT A.row, A.col, A.val + B.val as val
FROM   A, B
WHERE  A.row = B.row and A.col = B.col
```

Matrix Addition in SQL

$$C = A + B$$

```
SELECT A.row, A.col, A.val + B.val as val
FROM     A, B
WHERE   A.row = B.row and A.col = B.col
```

Why is this wrong?

Solution 1: Outer Joins

$$C = A + B$$

```
SELECT



FROM A full outer join B ON A.row = B.row and A.col = B.col;
```

Solution 1: Outer Joins

$$C = A + B$$

```
SELECT


 (CASE WHEN A.val is null THEN 0 ELSE A.val END) +
 (CASE WHEN B.val is null THEN 0 ELSE B.val END)  as val
FROM A full outer join B ON A.row = B.row and A.col = B.col;
```

Solution 1: Outer Joins

$$C = A + B$$

```
SELECT
 (CASE WHEN A.row is null THEN B.row ELSE A.row END) as row,

 (CASE WHEN A.val is null THEN 0 ELSE A.val END) +
 (CASE WHEN B.val is null THEN 0 ELSE B.val END)  as val
FROM A full outer join B ON A.row = B.row and A.col = B.col;
```

Solution 1: Outer Joins

$$C = A + B$$

```
SELECT
 (CASE WHEN A.row is null THEN B.row ELSE A.row END) as row,
 (CASE WHEN A.col is null THEN B.col ELSE A.col END) as col,
 (CASE WHEN A.val is null THEN 0 ELSE A.val END) +
 (CASE WHEN B.val is null THEN 0 ELSE B.val END)  as val
FROM A full outer join B ON A.row = B.row and A.col = B.col;
```

Solution 2: Group By

$$C = A + B$$

```
SELECT m.row, m.col, sum(m.val)
FROM (SELECT * FROM A
              UNION ALL
           SELECT * FROM B) as m
GROUP BY m.row, m.col;
```

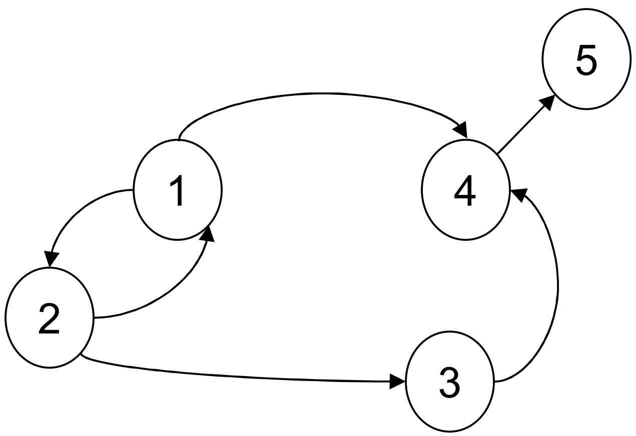
Graph Databases

A graph is a simple relational database

- Niche area: graph databases/languages
  - E.g. Neo4J, TigerGraph, Sparql
- Do we need specialized graph engines?
  - Dan's answer: NO
  - We may need better languages: datalog

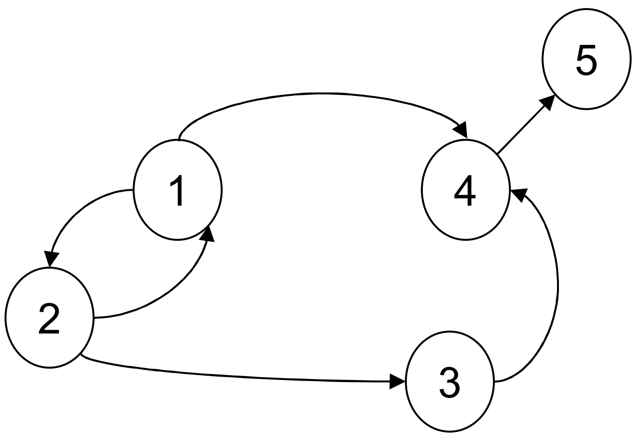
Graph Databases

A graph:



Graph Databases

A graph:



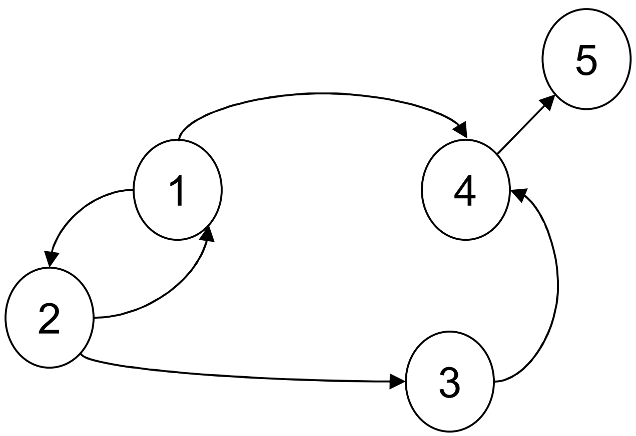
A relation:

Edge

src	dst
1	2
2	1
2	3
1	4
3	4
4	5

Graph Databases

A graph:



A relation:

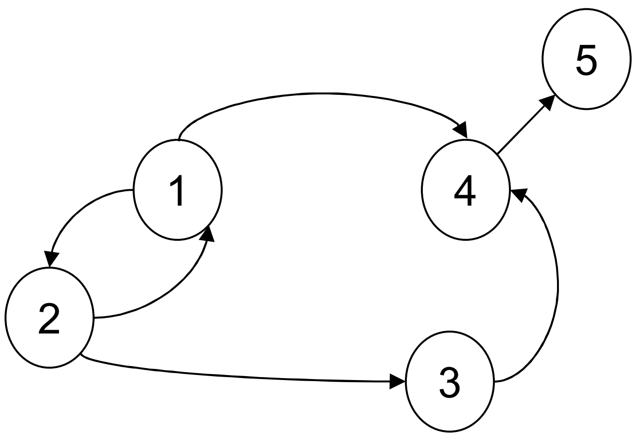
Edge

src	dst
1	2
2	1
2	3
1	4
3	4
4	5

Find nodes at distance 2: $\{(x,z) | \exists y \; Edge(x,y) \wedge Edge(y,z)\}$

Graph Databases

A graph:



A relation:

Edge

src	dst
1	2
2	1
2	3
1	4
3	4
4	5

Find nodes at distance 2: $\{(x, z) | \exists y\ Edge(x, y) \wedge Edge(y, z)\}$

```
SELECT DISTINCT e1.src as X, e2.dst as Z
FROM Edge e1, Edge e2
WHERE e1.dst = e2.src;
```

Crash Course in Formal Logic

- The Relational Data Model is *founded* on first order logic ("What goes around")

- SQL was designed as a more friendly language than FO

- Complex SQL queries are sometimes best understood in the framework of FO

Crash Course in Formal Logic

Atomic predicates:

- Likes(x,y)
- Product(x,y,z)
  -- pid, name, color
- Product(x,y,'red')

Connectives: $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\exists$, $\forall$

Crash Course in Formal Logic

Atomic predicates:

- Likes(x,y)
- Product(x,y,z)
  -- pid, name, color
- Product(x,y,'red')

Connectives: ∧, ∨, ¬, ⇒, ∃, ∀

- ∃x P(x):
  there exists x s.t. P(x) is true
- ∀x P(x):
  for every x, P(x) is true

Crash Course in Formal Logic

Atomic predicates:

- Likes(x,y)
- Product(x,y,z)
  -- pid, name, color
- Product(x,y,'red')

Connectives: ∧, ∨, ¬, ⇒, ∃, ∀

- ∃x P(x):
  there exists x s.t. P(x) is true
- ∀x P(x):
  for every x, P(x) is true

What do these sentences say?

∃x(Likes('Alice',x)∧Likes('Bob',x))

Crash Course in Formal Logic

Atomic predicates:

- Likes(x,y)
- Product(x,y,z)
  -- pid, name, color
- Product(x,y,'red')

Connectives: $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\exists$, $\forall$

- $\exists x\, P(x)$:
  there exists x s.t. P(x) is true
- $\forall x\, P(x)$:
  for every x, P(x) is true

What do these sentences say?

$\exists x(\text{Likes}(\text{'Alice'},x) \wedge \text{Likes}(\text{'Bob'},x))$

There is somebody liked
by both Alice and Bob

Crash Course in Formal Logic

Atomic predicates:

- Likes(x,y)
- Product(x,y,z)
  -- pid, name, color
- Product(x,y,'red')

Connectives: $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\exists$, $\forall$

- $\exists x\, P(x)$:
  there exists x s.t. P(x) is true
- $\forall x\, P(x)$:
  for every x, P(x) is true

What do these sentences say?

$\exists x(\text{Likes}(\text{'Alice'},x) \wedge \text{Likes}(\text{'Bob'},x))$

There is somebody liked by both Alice and Bob

$\forall x\, (\text{Likes}(\text{'Alice'},x) \Rightarrow \text{Likes}(\text{'Bob'},x))$

Crash Course in Formal Logic

Atomic predicates:

- Likes(x,y)
- Product(x,y,z)
  -- pid, name, color
- Product(x,y,'red')

Connectives: ∧, ∨, ¬, ⇒, ∃, ∀

- ∃x P(x):
  there exists x s.t. P(x) is true
- ∀x P(x):
  for every x, P(x) is true

What do these sentences say?

∃x(Likes('Alice',x)∧Likes('Bob',x))

There is somebody liked by both Alice and Bob

∀x (Likes('Alice',x) ⇒Likes('Bob',x))

Everybody liked by Alice, is also liked by Bob

Crash Course in Formal Logic

Atomic predicates:

- Likes(x,y)
- Product(x,y,z)
  -- pid, name, color
- Product(x,y,'red')

Connectives: $\wedge, \vee, \neg, \Rightarrow, \exists, \forall$

- $\exists x\ P(x)$:
  there exists x s.t. P(x) is true
- $\forall x\ P(x)$:
  for every x, P(x) is true

What do these sentences say?

$\exists x(\text{Likes}(\text{'Alice'},x) \wedge \text{Likes}(\text{'Bob'},x))$

There is somebody liked by both Alice and Bob

$\forall x\ (\text{Likes}(\text{'Alice'},x) \Rightarrow \text{Likes}(\text{'Bob'},x))$

Everybody liked by Alice, is also liked by Bob

$\forall x\ (\exists y\ \text{Likes}(x,y) \Rightarrow \text{Likes}(x,\text{'Alice'}))$

Crash Course in Formal Logic

Atomic predicates:

- Likes(x,y)
- Product(x,y,z)
  -- pid, name, color
- Product(x,y,'red')

Connectives: $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\exists$, $\forall$

- $\exists x\, P(x)$:
  there exists x s.t. P(x) is true
- $\forall x\, P(x)$:
  for every x, P(x) is true

What do these sentences say?

$\exists x(\text{Likes}(\text{'Alice'},x) \wedge \text{Likes}(\text{'Bob'},x))$

There is somebody liked by both Alice and Bob

$\forall x\, (\text{Likes}(\text{'Alice'},x) \Rightarrow \text{Likes}(\text{'Bob'},x))$

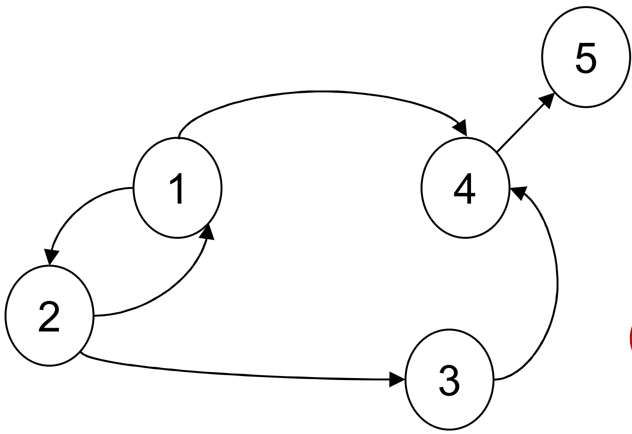
Everybody liked by Alice, is also liked by Bob

$\forall x\, (\exists y\, \text{Likes}(x,y) \Rightarrow \text{Likes}(x,\text{'Alice'}))$

Everybody who likes somebody also likes Alice

Graph Databases

A graph:



A relation:

Now this should be clear

Edge

src	dst
1	2
2	1
2	3
1	4
3	4
4	5

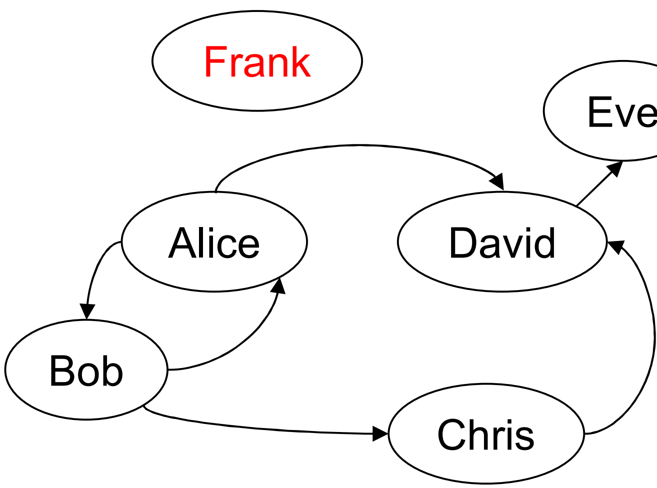
Find nodes at distance 2: $\{(x,z) | \exists y\ Edge(x,y) \wedge Edge(y,z)\}$

```
SELECT DISTINCT e1.src as X, e2.dst as Z
FROM Edge e1, Edge e2
WHERE e1.dst = e2.src;
```

Other Representation

Representing nodes separately;

needed for "isolated nodes" e.g. Frank



Node

src
Alice
Bob
Chris
David
Eve
Frank

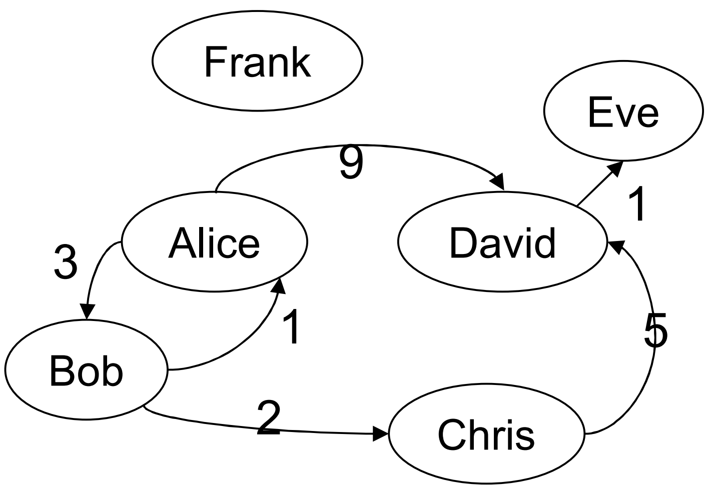
Edge

src	dst
Alice	Bob
Bob	Alice
Bob	Chris
Alice	David
Chris	David
David	Eve

Other Representation

Adding edge labels

Adding node labels…



Node

src
Alice
Bob
Chris
David
Eve
Frank

Edge

src	dst	weight
Alice	Bob	3
Bob	Alice	1
Bob	Chris	2
Alice	David	9
Chris	David	5
David	Eve	1

Limitations of SQL

- No recursion!
- Data Science often requires recursion
- Datalog is designed for recursion
  - later in the quarter
- Practical solution
  - Use some external driver, e.g. pyton

Example: Logistic Regression

Tom Mitchell: Machine Learning

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	0
3	6	3	0
5	5	9	1
9	3	3	1
…	…	…	
…	…	…	

Example: Logistic Regression

Tom Mitchell: Machine Learning

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	0
3	6	3	0
5	5	9	1
9	3	3	1
…	…	…	
…	…	…	

Switched (following Mitchell)

$$P(Y=0|X) = \frac{1}{1 + exp\left(w_0 + \sum_{i=1,3} w_i X_i\right)}$$

$$P(Y=1|X) = \frac{exp\left(w_0 + \sum_{i=1,3} w_i X_i\right)}{1 + exp\left(w_0 + \sum_{i=1,3} w_i X_i\right)}$$

Example: Logistic Regression

Tom Mitchell: Machine Learning

Switched (following Mitchell)

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	0
3	6	3	0
5	5	9	1
9	3	3	1
…	…	…	
…	…	…	

$$P(Y=0|X) = \frac{1}{1+exp\left(w_0 + \sum_{i=1,3} w_i X_i\right)}$$

$$P(Y=1|X) = \frac{exp\left(w_0 + \sum_{i=1,3} w_i X_i\right)}{1+exp\left(w_0 + \sum_{i=1,3} w_i X_i\right)}$$

Train weights w_0, w_1, w_2, w_3 to minimize loss:

$$L(w_0, \ldots, w_3) = \sum_{\ell=1,N} \left(Y^\ell \cdot \ln P\left(Y=1|X^\ell\right) + (1-Y^\ell) \cdot \ln P\left(Y=0|X^\ell\right)\right)$$

Example: Logistic Regression

Tom Mitchell: Machine Learning

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	0
3	6	3	0
5	5	9	1
9	3	3	1
…	…	…	
…	…	…	

Gradient Descent:

$$w_i \leftarrow w_i + \eta \sum_{\ell=1,N} X_i^{\ell}\left(Y^{\ell} - P(Y = 1|X^{\ell})\right)$$

Example: Logistic Regression

Tom Mitchell: Machine Learning

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	[illegible]
3	6	3	[illegible]
5	5	9	1
9	3	3	1
…	…	…	
…	…	…	

Gradient Descent:

$$w_i \leftarrow w_i + \eta \sum_{\ell=1,N} X_i^\ell \left(Y^\ell - P(Y = 1|X^\ell)\right)$$

```sql
CREATE TABLE W (k int primary key, w0 real, w1 real, w2 real, w3 real);
INSERT INTO W VALUES (1, 0, 0, 0, 0);
```

Example: Logistic Regression

Tom Mitchell: Machine Learning

Gradient Descent:

$$w_i \leftarrow w_i + \eta \sum_{\ell=1,N} X_i^{\ell}\left(Y^{\ell} - P(Y = 1|X^{\ell})\right)$$

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	[illegible]
3	6	3	[illegible]

```
CREATE TABLE W (k int primary key, w0 real, w1 real, w2 real, w3 real);
INSERT INTO W VALUES (1, 0, 0, 0, 0);
```

```
FROM data d, W
WHERE W.k=1
```

Example: Logistic Regression

Tom Mitchell: [Machine Learning]

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	[illegible]
3	6	3	[illegible]

Gradient Descent:

$$w_i \leftarrow w_i + \eta \sum_{\ell=1,N} X_i^{\ell}\left(Y^{\ell} - P(Y = 1|X^{\ell})\right)$$

```
CREATE TABLE W (k int primary key, w0 real, w1 real, w2 real, w3 real);
INSERT INTO W VALUES (1, 0, 0, 0, 0);
```

```
SELECT
   W.w0+0.01*sum(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3))) as w0,



FROM data d, W
WHERE W.k=1
```

Example: Logistic Regression

Tom Mitchell: Machine Learning

Gradient Descent:

$$w_i \leftarrow w_i + \eta \sum_{\ell=1,N} X_i^{\ell}\left(Y^{\ell} - P(Y = 1|X^{\ell})\right)$$

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	[illegible]
3	6	3	[illegible]

```
CREATE TABLE W (k int primary key, w0 real, w1 real, w2 real, w3 real);
INSERT INTO W VALUES (1, 0, 0, 0, 0);
```

```
SELECT
  W.w0+0.01*sum(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3))) as w0,
  W.w1+0.01*sum(d.X1*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w1,


FROM data d, W
WHERE W.k=1
```

Example: Logistic Regression

Tom Mitchell: Machine Learning

Gradient Descent:

$$w_i \leftarrow w_i + \eta \sum_{\ell=1,N} X_i^{\ell}\left(Y^{\ell} - P(Y=1|X^{\ell})\right)$$

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	0
3	6	3	0

```
CREATE TABLE W (k int primary key, w0 real, w1 real, w2 real, w3 real);
INSERT INTO W VALUES (1, 0, 0, 0, 0);
```

```
SELECT
  W.w0+0.01*sum(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3))) as w0,
  W.w1+0.01*sum(d.X1*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w1,
  W.w2+0.01*sum(d.X2*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w2,
  W.w3+0.01*sum(d.X3*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w3
FROM data d, W
WHERE W.k=1
```

Example: Logistic Regression

Tom Mitchell: Machine Learning

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	[illegible]
3	6	3	[illegible]

Gradient Descent:

$$w_i \leftarrow w_i + \eta \sum_{\ell=1,N} X_i^\ell \left(Y^\ell - P(Y = 1 | X^\ell)\right)$$

```
CREATE TABLE W (k int primary key, w0 real, w1 real, w2 real, w3 real);
INSERT INTO W VALUES (1, 0, 0, 0, 0);
```

```
SELECT
  W.w0+0.01*sum(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3))) as w0,
  W.w1+0.01*sum(d.X1*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w1,
  W.w2+0.01*sum(d.X2*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w2,
  W.w3+0.01*sum(d.X3*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w3
FROM data d, W
WHERE W.k=1
GROUP BY W.k, W.w0, W.w1, W.w2, W.w3;
```

Example: Logistic Regression

Tom Mitchell: Machine Learning

Data

X1	X2	X3	Y
3	9	3	0
3	5	7	1
6	2	2	[illegible]
3	6	3	[illegible]

Gradient Descent:

$$w_i \leftarrow w_i + \eta \sum_{\ell=1,N} X_i^{\ell}\left(Y^{\ell} - P(Y=1|X^{\ell})\right)$$

```
CREATE TABLE W (k int primary key, w0 real, w1 real, w2 real, w3 real);
INSERT INTO W VALUES (1, 0, 0, 0, 0);
```

```
SELECT
  W.w0+0.01*sum(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3))) as w0,
  W.w1+0.01*sum(d.X1*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w1,
  W.w2+0.01*sum(d.X2*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w2,
  W.w3+0.01*sum(d.X3*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) as w3
FROM data d, W
WHERE W.k=1
GROUP BY W.k, W.w0, W.w1, W.w2, W.w3;
```

Update W, then repeat this
e.g. using python

Lecture Summary

- One line takeaway:
  – Relational model → data independence
- What you should do next:
  – Review SQL
  – Write reviews for next lecture
  – Start working on HW1 (redshift)