CSE544
Data Management

Lectures 1&2:
Relational Data Model, SQL
Outline

• Introduction, class overview

• Database management systems (DBMS)

• The relational model
Course Staff

• Instructor: Dan Suciu
  – Office hours: Mondays, 11:30-12:20

• TA: Walter Cai
  – Office hours: TBD
Goals of the Class

• **Relational Data Model**
  - Data models, data independence, declarative query language.

• **Relational Database Systems**
  - Storage, query execution and optimization
  - Parallel data processing, column-oriented db etc.

• **Transactions**
  - Optimistic/pessimistic concurrency control
  - ARIES recovery system
A Note for Non-Majors

• For the Data Science option: take 414
• For the Advanced Data Science option: take 544

• 544 is an advanced class, not intended as an introduction to data management research
• Does not cover fundamentals systematically, yet there is an exam testing those fundamentals

• Unsure? Look at the short quiz on the website.
Readings

• Paper reviews
  – Mix of old seminal papers and new papers
  – Papers are available on class website

• Lecture notes (the slides)
  – Posted on class website after each lecture

• Background from:
  – Database Management Systems. Third Ed.
Class Resources

Website: lectures, assignments
  • http://www.cs.washington.edu/544

Canvas: zoom, videos

Ed: discussion board
Evaluation

- Assignments 40%
- Reviews 10%
- Project 40%
- Intangibles 10%
Assignments – 40%

- **HW1**: Use a DBMS
- **HW2**: Data analysis in the cloud
- **HW3**: Query Execution and SimpleDB
- **HW4**: Datalog
- [possibly a HW5 on transactions]
- See course calendar for deadlines
- Late assignments w/ **very** valid excuse
Paper reviews – 10%

- Recommended length: ½ page – 1 page
  - Summary of the main points of the paper
  - Critical discussion of the paper

- Grading: credit/partial-credit/no-credit

- Submit review before the lecture
Project – 40%

• Topic
  – Best: come up with your own, ideally related to your own research
  – Or choose from a list of mini-research topics
  – Can be related to a project in another course
  – Must be related to databases / data management
  – Must involve either research or significant engineering
  – Open ended

• Final deliverables
  – Short, conference-style presentation on Friday, March 12
  – Short, conference-style paper (6 pages)
Project – 40%

• Dates posted on the calendar page:
  – M1: form groups
  – M2: Project proposal
  – M3: Milestone report
  – M4: Poster presentation
  – M5: Project paper

• We will provide feedback throughout the quarter
Intangibles 10%

• Class participation

• Exceptionally good reviews, or homework, or project

• Etc, etc
How to Turn In

- Homeworks: gitlab
- Project: gitlab
- Reviews: google forms
Now onward to the world of databases!
Data Management

- **Entities**: employees, positions (ceo, manager, cashier), stores, products, sells, customers.

- **Relationships**: employee positions, staff of each store, inventory of each store.
Database Management System

• A DBMS is a software system designed to provide data management services

• Examples of DBMS
  – Oracle, DB2 (IBM), SQL Server (Microsoft),
  – PostgreSQL, MySQL, …
DBMS Functionality

1. Create & persistently store large datasets
2. Efficiently query & update
   1. Must handle complex questions about data
   2. Must handle sophisticated updates
   3. Performance matters
3. Change structure (e.g., add attributes)
4. Concurrency control: enable simultaneous updates
5. Crash recovery
6. Access control, security, integrity

Several types of architectures (next)
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, ...

Connection:
ODBC, JDBC

Database server
E.g. Oracle, DB2, ...

Applications:
Java
Three-tier Architecture

E.g. Web commerce

Application server
E.g. java, python, ruby-on-rails

Database server
E.g. Oracle

connection
(ODBC, JDBC)

Browser

http
Cloud Databases

E.g. large-scale analytics or…

Sharded database
E.g. Spark, Snowflake

ODBC, JDBC

http

App server

…social networks
Workloads

• OLTP – online transaction processing

• OLAP – online analytics processing, a.k.a. Decision Support

Most of this course
Relational Data Model
Relational Data Model

• A **Database** is a collection of relations

• A **Relation** is a set of tuples
  – Also called **Table**

• A **Tuple** \( t \) is an element of \( \text{Dom}_1 \times \text{Dom}_2 \times \ldots \times \text{Dom}_n \)
  – \( \text{Dom}_i \) is the domain of attribute \( i \)
  – \( n \) is number of attributes of the relation
  – Also called **Row** or **Record**
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**

• **Columns** in a tuple:
  – Ordering is significant
  – Applications refer to columns by their names

• **Domain** of each column is a primitive type
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 all relation instances
What is the schema? What is the instance?

<table>
<thead>
<tr>
<th>sno</th>
<th>sname</th>
<th>scity</th>
<th>sstate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s1</td>
<td>city 1</td>
<td>WA</td>
</tr>
<tr>
<td>2</td>
<td>s2</td>
<td>city 1</td>
<td>WA</td>
</tr>
<tr>
<td>3</td>
<td>s3</td>
<td>city 2</td>
<td>MA</td>
</tr>
<tr>
<td>4</td>
<td>s4</td>
<td>city 2</td>
<td>MA</td>
</tr>
</tbody>
</table>
What is the schema?  
What is the instance?

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

Supplier

<table>
<thead>
<tr>
<th>sno</th>
<th>sname</th>
<th>scity</th>
<th>sstate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s1</td>
<td>city 1</td>
<td>WA</td>
</tr>
<tr>
<td>2</td>
<td>s2</td>
<td>city 1</td>
<td>WA</td>
</tr>
<tr>
<td>3</td>
<td>s3</td>
<td>city 2</td>
<td>MA</td>
</tr>
<tr>
<td>4</td>
<td>s4</td>
<td>city 2</td>
<td>MA</td>
</tr>
</tbody>
</table>
Relational Query Language

• **Set-at-a-time:**
  – Query inputs and outputs are relations

• **Two variants of the query language:**
  – Relational algebra: specifies order of operations
  – Relational calculus / SQL: declarative
SQL

• Standard query language

• Introduced late 70’s, now it ballooned

• We briefly review “core SQL” (whatever that means); study more on you own!

• Read by Wed: A case against SQL
Structured Query Language: SQL

- **Data definition language: DDL**
  - Statements to create, modify tables and views
  - CREATE TABLE …,
  - CREATE VIEW …,
  - ALTER TABLE…

- **Data manipulation language: DML**
  - Statements to issue queries, insert, delete data
  - SELECT-FROM-WHERE…,
  - INSERT…,
  - UPDATE…,
  - DELETE…

Review on your own

We quickly review this
SQL Query

Basic form: (plus many many more bells and whistles)

```
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
Quick Review of SQL

```
SELECT DISTINCT  z.pno, z.pname
FROM    Supplier x, Supply y, Part z
WHERE    x.sno = y.sno
         and y.pno = z.pno
         and x.scity = ‘Seattle’
         and y.price < 100
```
Terminology

• **Selection**: return a subset of the rows:
  – `SELECT * FROM Supplier
    WHERE scity = 'Seattle'`

• **Projection**: return subset of the columns:
  – `SELECT DISTINCT scity FROM Supplier;`

• **Join**: refers to combining two or more tables
  – `SELECT * FROM Supplier, Supply, Part ...`
Self-Joins

Find the Parts numbers available both from suppliers in Seattle, and suppliers in Portland
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
```
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
Nested-Loop Semantics of SQL

**Example SQL Query**

```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
```

**Semantics**

```plaintext
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 ∪ {(a_1, ..., a_k)}
return Answer
```

This SEMANTICS! It is NOT how the engine computes the query!
More SQL: Aggregates

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

• SQL Aggregates = simple data analytics

• Semantics:
  1. FROM-WHERE (nested-loop semantics)
  2. Group answers by GROUP BY attrs
  3. Apply HAVING predicates on groups
  4. Apply SELECT aggregates on groups

• Aggregate functions:
  – count, sum, min, max, avg

• DISTINCT same as GROUP BY
Retrieve all product names, categories, and stores where they were purchased. Include products that never sold.
Outer joins

Retrieve all product names, categories, and stores where they were purchased. Include products that never sold.

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

Retrieve all product names, categories, and stores where they were purchased. Include products that never sold.

**SELECT**

\[
\begin{align*}
x \text{.} \text{name}, & \quad x \text{.} \text{category}, & \quad y \text{.} \text{store}\\
\end{align*}
\]

**FROM**

Product \(x\), Purchase \(y\)

**WHERE**

\[
x \text{.} \text{name} = y \text{.} \text{prodName}
\]

<table>
<thead>
<tr>
<th>Product</th>
<th>Purchase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>ProdName</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Gizmo</td>
</tr>
<tr>
<td>Camera</td>
<td>Camera</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Photo</td>
</tr>
<tr>
<td>gadget</td>
<td></td>
</tr>
<tr>
<td>Photo</td>
<td></td>
</tr>
</tbody>
</table>
Outer joins

Retrieve all product names, categories, and stores where they were purchased. Include products that never sold.

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

<table>
<thead>
<tr>
<th>Product</th>
<th>Purchase</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>ProdName</strong></td>
<td><strong>Name</strong></td>
</tr>
<tr>
<td>Gizmo</td>
<td>Gizmo</td>
<td>Gizmo</td>
</tr>
<tr>
<td>Camera</td>
<td>Camera</td>
<td>Camera</td>
</tr>
<tr>
<td>OneClick</td>
<td>Camera</td>
<td>Camera</td>
</tr>
</tbody>
</table>

*missing*
Outer joins

Retrieve all product names, categories, and stores where they were purchased. Include products that never sold.

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

<table>
<thead>
<tr>
<th>Product</th>
<th></th>
<th>Purchase</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Category</strong></td>
<td><strong>ProdName</strong></td>
<td><strong>Store</strong></td>
</tr>
<tr>
<td>Gizmo</td>
<td>gadget</td>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
<td>Camera</td>
<td>Wiz</td>
</tr>
</tbody>
</table>

**Now it's present**

prodName is foreign Key.
Joins

• **Inner join** = includes only matching tuples (i.e. regular join)

• **Left outer join** = includes everything from the left

• **Right outer join** = includes everything from the right

• **Full outer join** = includes everything
ON v.s. WHERE

- Outer join condition in the ON clause
- Different from the WHERE clause
- Compare:

SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
       AND y.price < 10

SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
       WHERE y.price < 10
**ON v.s. WHERE**

- Outer join condition in the **ON** clause
- Different from the **WHERE** clause
- Compare:

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
       AND y.price < 10
```

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
WHERE  y.price < 10
```

Includes products that were never purchased with price < 10
ON v.s. WHERE

- Outer join condition in the **ON** clause
- Different from the **WHERE** clause
- Compare:

```sql
SELECT x.name, y.store
FROM Product x
LEFT OUTER JOIN Purchase y
ON x.name = y.prodName
AND y.price < 10
```

Includes products that were never purchased with price < 10

```sql
SELECT x.name, y.store
FROM Product x
LEFT OUTER JOIN Purchase y
ON x.name = y.prodName
WHERE y.price < 10
```

Includes products that were never purchased, *then* checks price < 10.
ON v.s. WHERE

- Outer join condition in the **ON** clause
- Different from the **WHERE** clause
- Compare:

```sql
SELECT x.name, y.store
FROM Product x
LEFT OUTER JOIN Purchase y
ON x.name = y.prodName
AND y.price < 10
```

Includes products that were never purchased with price < 10

```sql
SELECT x.name, y.store
FROM Product x
LEFT OUTER JOIN Purchase y
ON x.name = y.prodName
WHERE y.price < 10
```

Includes products that were never purchased, then checks price <10

Same as inner join!
NULLs in SQL

• A NULL value means missing, or unknown, or undefined, or inapplicable
• We can specify whether attributes may or may not be NULL:

```sql
CREATE TABLE product
(
  pid int NOT NULL,  
  pname text NOT NULL,  
  price int -- may be NULL
);
```
Three-Valued Logic

• False=0, Unknown=0.5, True=1
• Result of a comparison A=B is
  – False or True when both A, B are not null
  – Unknown otherwise
• AND, OR, NOT are min, max.
• Return tuples whose condition is True
Three-Valued Logic

- False=0, Unknown=0.5, True=1
- Result of a comparison A=B is
  - False or True when both A, B are not null
  - Unknown otherwise
- AND, OR, NOT are min, max.
- Return tuples whose condition is True

```sql
SELECT * FROM Product WHERE (price <= 100) OR (price > 100)
```

<table>
<thead>
<tr>
<th>pid</th>
<th>Pname</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>iPhone</td>
<td>500</td>
</tr>
<tr>
<td>2</td>
<td>iPod</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>iPad</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Three-Valued Logic

- False=0, Unknown=0.5, True=1
- Result of a comparison A=B is
  - False or True when both A, B are not null
  - Unknown otherwise
- AND, OR, NOT are min, max.
- Return tuples whose condition is True

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

where (price <= 100) or (price > 100)
  or isNull(price)
```

<table>
<thead>
<tr>
<th>pid</th>
<th>Pname</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>iPhone</td>
<td>500</td>
</tr>
<tr>
<td>2</td>
<td>iPod</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>iPad</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Likbkin’s Critique Of SQL

• Libkin’s slides: *A Case Against SQL*
• In class: discuss some of the main inconsistencies in SQL
Other use of Relational Data

- Sparse vectors, matrices
- Graph databases
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

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

<table>
<thead>
<tr>
<th>Row</th>
<th>Col</th>
<th>Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>-2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>7</td>
</tr>
</tbody>
</table>
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
• Many operations can be written in SQL
• E.g. 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

\[ C = A + B \]

```sql
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 1: Outer Joins

\[ C = A + B \]

```sql
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

\[ C = A + B \]

```sql
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

\[ C = A + B \]

```sql
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

\[ C = A + B \]

```sql
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

- Graph databases systems are a niche category of products specialized for processing large graphs
- E.g. Neo4J, TigerGraph
- A graph is a special case of a relation, and can be processed using SQL
Graph Databases

A graph:
Graph Databases

A graph:

A relation:

Edge

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
Graph Databases

A graph:

A relation:

Edge

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Find nodes at distance 2: \{(x, z) \mid \exists y \ Edge(x, y) \land Edge(y, z)\}
Graph Databases

A graph:

A relation:

Edge

<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Node</th>
<th>Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>dst</td>
</tr>
<tr>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td>Bob</td>
<td>Alice</td>
</tr>
<tr>
<td>Chris</td>
<td>Bob</td>
</tr>
<tr>
<td>David</td>
<td>Chris</td>
</tr>
<tr>
<td>Eve</td>
<td>David</td>
</tr>
<tr>
<td>Frank</td>
<td>Eve</td>
</tr>
</tbody>
</table>
Other Representation

Adding edge labels
Adding node labels...

<table>
<thead>
<tr>
<th>Node</th>
<th>Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>dst</td>
</tr>
<tr>
<td>Alice</td>
<td>Bob</td>
</tr>
<tr>
<td>Bob</td>
<td>Alice</td>
</tr>
<tr>
<td>Chris</td>
<td>Bob</td>
</tr>
<tr>
<td>David</td>
<td>Alice</td>
</tr>
<tr>
<td>Eve</td>
<td>David</td>
</tr>
<tr>
<td>Frank</td>
<td>Eve</td>
</tr>
<tr>
<td>David</td>
<td>Eve</td>
</tr>
</tbody>
</table>
Discussion: SQL and Logic

• First Order Logic is the language consisting of: $\forall, \exists, \lor, \land, \neg, \Rightarrow$

• In class: what do these sentences say?
  – $\forall x\forall y(E(x, y) \Rightarrow E(y, x))$
  – $\exists x(E(\text{"Alice"}, x) \land E(\text{"Bob"}, x))$
  – $\forall x\forall y\forall z(E(x, y) \land E(y, z) \Rightarrow E(x, z))$
  – $\forall x\forall y(E(x, y) \Rightarrow (x \neq y) \land \exists z(E(x, z) \land E(z, y))$

• **Theorem:** every FO sentence can be written in SQL
Limitations of SQL

• No recursion! Examples requiring recursion:
  – Gradient descent
  – Connected components in a graph
• Advanced systems _do_ support recursion
• Practical solution: use some external driver, e.g. pyton
Example: Logistic Regression


Data

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Example: Logistic Regression

Tom Mitchell: Machine Learning

Data

\[
\begin{array}{|c|c|c|c|}
\hline
X1 & X2 & X3 & Y \\
\hline
3 & 9 & 3 & 0 \\
3 & 5 & 7 & 1 \\
6 & 2 & 2 & 0 \\
3 & 6 & 3 & 0 \\
5 & 5 & 9 & 1 \\
9 & 3 & 3 & 1 \\
\ldots & \ldots & \ldots & \ldots \\
\ldots & \ldots & \ldots & \ldots \\
\hline
\end{array}
\]

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

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

Switched (following Mitchell)
Example: Logistic Regression

Data

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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

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

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

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

Switched (following Mitchell)
Data

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Gradient Descent:

\[
\hat{w}_i \leftarrow w_i + \eta \sum_{\ell=1,N} X_i^\ell (Y^\ell - P(\text{Y} = 1|X^\ell))
\]
Example: Logistic Regression

Tom Mitchell: Machine Learning

Data

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Gradient Descent:

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

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]

Data

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Gradient Descent:

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

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

**Gradient Descent:**

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

**Data**

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

**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:

\[
\mathbf{w}_i \leftarrow \mathbf{w}_i + \eta \sum_{\ell=1,N} \mathbf{X}_\ell (\mathbf{Y}_\ell - \mathbb{P}(\mathbf{Y} = 1|\mathbf{X}_\ell))
\]

Data

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

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**


### Data

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

### Gradient Descent:

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

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

SELECT 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,
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,
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,
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

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Gradient Descent:

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

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

<table>
<thead>
<tr>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>9</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

**Gradient Descent:**

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

---

**SELECT**

\[
\begin{align*}
W.w0 + 0.01 & \times \text{sum(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3))))} \\
W.w1 + 0.01 & \times \text{sum(d.X1*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3))))} \\
W.w2 + 0.01 & \times \text{sum(d.X2*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3))))} \\
W.w3 + 0.01 & \times \text{sum(d.X3*(d.Y - 1 + 1/(1+exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3))))}
\end{align*}
\]

**FROM** data d, W
**WHERE** W.k=1
**GROUP BY** W.k, W.w0, W.w1, W.w2, W.w3;

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

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

SQL in Data Science:

• Used primarily to prepare the data
  – ETL – Extract/Transform/Load
  – Join tables, process columns, filter rows

• Can also be used in training
  – Much less convenient than ML packages
  – But can be the best option if data is huge
SQL – Summary

• Very complex: >1000 pages,
  – No vendor supports full standard; (in practice, people use postgres as de facto standard)
  – Much more than DML

• It is a **declarative** language:
  – we say what we want
  – we don’t say how to get it

• Relational algebra says how to get it
Relational Algebra

• Queries specified in an operational manner
  – A query gives a step-by-step procedure

• Relational operators
  – Take one or two relation instances as input
  – Return one relation instance as result
  – Easy to compose into relational algebra expressions
Five Basic Relational Operators

- **Selection**: $\sigma_{\text{condition}}(S)$
  - Condition is Boolean combination ($\land, \lor$) of atomic predicates ($<, \leq, =, \neq, \geq, >$)
- **Projection**: $\pi_{\text{list-of-attributes}}(S)$
- **Union** ($\cup$)
- **Set difference** ($\setminus$),
- **Cross-product/cartesian product** ($\times$),
  Join: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$

Other operators: anti-semijoin, renaming
Extended Operators of Relational Algebra

• **Duplicate elimination** ($\delta$)
  – Since commercial DBMSs operate on multisets not sets

• **Group-by/aggregate** ($\gamma$)
  – Min, max, sum, average, count
  – Partitions tuples of a relation into “groups”
  – Aggregates can then be applied to groups

• **Sort operator** ($\tau$)
Logical Query Plans

```
SELECT DISTINCT x.sname, x.scity
FROM Supplier x, Supply y, Part z
WHERE x.sno=y.sno
    AND y.pno=z.pno
    AND z.psize > 10;
```
Logical Query Plans

SELECT DISTINCT x.sname, x.scity
FROM Supplier x, Supply y, Part z
WHERE x.sno = y.sno
  and y.pno = z.pno
  and z.psize > 10;

\[\delta \]
\[\Pi_{sname, scity}\]
\[\bowtie\]
\[\bowtie\]
\[\bowtie\]
\[\sigma_{psize > 10}\]

\(\delta\) - Delta Join
\(\Pi\) - Selection
\(\bowtie\) - Equi Join
\(\sigma\) - Selection
Query Optimizer

• Rewrite one relational algebra expression to a better one
• Very brief review now, more details next lectures
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid AND y.cid = z.cid AND
x.price > 100 AND z.city = 'Seattle'
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
  x.price > 100 and z.city = 'Seattle'
Optimization

SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
    x.price > 100 and z.city = 'Seattle'

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

More about this next lecture
Benefits of Relational Model

• Physical data independence
  – Can change how data is organized on disk without affecting applications

• Logical data independence
  – Can change the logical schema without affecting applications (not 100%... consider updates)
Physical Data Independence

**Supplier**

<table>
<thead>
<tr>
<th>sno</th>
<th>sname</th>
<th>scity</th>
<th>sstate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s1</td>
<td>city 1</td>
<td>WA</td>
</tr>
<tr>
<td>2</td>
<td>s2</td>
<td>city 1</td>
<td>WA</td>
</tr>
<tr>
<td>3</td>
<td>s3</td>
<td>city 2</td>
<td>MA</td>
</tr>
<tr>
<td>4</td>
<td>s4</td>
<td>city 2</td>
<td>MA</td>
</tr>
</tbody>
</table>

SELECT DISTINCT sname FROM Supplier WHERE scity = 'Seattle'

How is the data stored on disk? (e.g. row-wise, column-wise)

Is there an index on scity? (e.g. no index, unclustered index, clustered index)

The SQL query works the same, regardless of the answers to these questions
Lecture on Monday

• Data model – what’s so hard about it?

• Review “What goes around…”