Lecture 1
Design of a Relational DBMS
Course Staff

• Instructor: Dan Suciu
  suciu@cs.washington.edu

• TA: Remy Wang
  remywang@cs.washington.edu

• TA: Zechariah Cheung
  zachcheu@gmail.com
Coarse Aims

- Study design of big data systems
  - Historical perspective
  - Sample of modern systems
  - Breadth of designs (relational, streaming, graph, etc.)
- Study key scalable data processing algorithms
- Gain hands-on experience with big data systems
  - Demonstrations and tutorials in sections
  - Assignments and projects
Coarse Content

- Query processing: single-server, distributed
- MapReduce, legacy, successors
- Some important “Big data” algorithms
- Misc: streaming, column stores, graph engines
Course Format

• 5pm-7:50pm: Lectures
  – Discuss system architecture & algorithms

• 8pm-8:50pm: Hands-on tutorials
  – Learn how to use big data systems
  – Jump start your homeworks
  – Bring your laptop!
Grading (subject to change!)

• 15%: Reading assigned papers
  – Write short statement/review
• 60%: Homework assignments
  – Redshift, Spark, Snowflake, others
• 25%: Final project
Choose a topic:

• Don’t worry about novelty!
• Highly recommended: Benchmark projects
  – Analyze the performance of some features
  – Compare the performance of different systems
  – Try to implement an interesting workload
• I will post a few ideas, but you are strongly encouraged to come up with your own
Project

1. Project proposal (1 page)
2. Project milestone (2-3 pages)
3. Project presentation (in class)
4. Project final report (4-5 pages)
Web Services

- HW1: Amazon Redshift – attend today’s section!
- HW2: Spark/AWS
- HW3: Snowflake – see Remy’s post
- HW4: mini-homeworks – stay tuned

Azure: optional, for the project
Communication

• Course webpage: all important stuff
  https://courses.cs.washington.edu/courses/csed516/20au/

• Discussion Board: ED. Say “hello”!

• Class email: only for important announcements
How to Turn In

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

- Your own repository
- Pull to get homework instructions, starter files
- Push homework solutions, project reports

Reviews: we use google forms

- Typically around ½ page
- Goal is only for us to check that you have read the paper
Relational Database Management Systems
Quick Review

• Database is a collection of files
• Database management system (DBMS) is a piece of software to help manage that data
• History:
  – Origins in the 1960’s
  – Relational model 1970
  – First relational DBMSs (Ingres and System R): 1970’s
  – Parallel DBMSs: 1980’s
DBMS Functionality

1. Describe real-world entities in terms of a data model
2. Create & persistently store large datasets
3. Efficiently query & update
   1. Must handle complex questions about data
   2. Must handle sophisticated updates
   3. Performance matters
4. Change structure (e.g., add attributes)
5. Concurrency control: enable simultaneous updates
6. Crash recovery
7. Access control, security, integrity
Relational Data Model

- A **Database** is a collection of relations.

- A **Relation** is a subset of $\text{Dom}_1 \times \text{Dom}_2 \times \ldots \times \text{Dom}_n$
  - Where $\text{Dom}_i$ is the domain of attribute $i$
  - $n$ is the number of attributes of the relation
  - A relation $R$ is a set of tuples

- A **Tuple** $t$ is an element of $\text{Dom}_1 \times \text{Dom}_2 \times \ldots \times \text{Dom}_n$

Other names: relation = *table*; tuple = *row*
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 tuples (also called records) matching the schema
  - The *cardinality* of the relation = # tuples
    (a.k.a. size)

- **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
Note

• We will review Relational Algebra and SQL today

• In addition: please review at home:
  – Review material from DATA514/CSED514
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…

We focus on this
SQL Query

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

**SELECT**  <attributes>
**FROM**   <one or more relations>
**WHERE**  <conditions>
Quick Review of SQL

Supplier(sno, sname, scity, sstate)
Supply(sno, pno, qty, price)
Part(pno, pname, psize, pcolor)
Quick Review of SQL

What does this query compute?

```
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
```

Simple Analytics

For each part, compute its minimum and maximum price from all suppliers.

```
SELECT z.pno, z.pname, min(y.price) as p1, max(y.price) as p2
FROM    Supply y, Part z
WHERE   y.pno = z.pno
GROUP BY z.pno, z.pname
```
Terminology

• Online Analytical Processing (OLAP) a.k.a. Data Analytics queries
  – GROUP-BY + aggregates
  – No updates
  – Touch most of, or all the data
  – Very important in data science!

Data Science
Terminology

• Online Analytical Processing (OLAP) a.k.a. Data Analytics queries
  – GROUP-BY + aggregates
  – No updates
  – Touch most of, or all the data
  – Very important in data science!

• Online Transaction Processing (OLTP):
  – Point queries: return account 12345
  – Often have updates
Other use of Relational Data

- Sparse vectors, matrices
- Graph databases
Sparse Matrics

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

How can we represent it as a relation?
Sparse Matrics

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

<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

\[ C = A + B \]

```
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

\[ C = A + B \]

```
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;
Discussion

• Outer joins: includes a tuple even if it doesn’t join with anything in the other table
• Left outer join, right outer join, full outer join – what do they mean?
• Note distinction between ON and WHERE
## WHERE v.s. ON

Sparse vectors:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>50</td>
<td>0</td>
<td>-30</td>
<td>60</td>
<td>-80</td>
<td>0</td>
<td>-90</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Y</td>
<td>-55</td>
<td>0</td>
<td>65</td>
<td>-15</td>
<td>0</td>
<td>35</td>
<td>-75</td>
<td>15</td>
<td>25</td>
</tr>
</tbody>
</table>
WHERE v.s. ON

Sparse vectors:

\[ X = \begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
50 & 0 & -30 & 60 & -80 & 0 & -90 & 10 & 0
\end{array} \]

\[ Y = \begin{array}{cccccccccc}
-55 & 0 & 65 & -15 & 0 & 35 & -75 & 15 & 25
\end{array} \]

```
SELECT x.pos, x.val, y.val
FROM x left outer join y
ON x.pos = y.pos and y.val > 0;
```
WHERE v.s. ON

Sparse vectors:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>50</td>
<td>0</td>
<td>-30</td>
<td>60</td>
<td>-80</td>
<td>0</td>
<td>-90</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Y</td>
<td>-55</td>
<td>0</td>
<td>65</td>
<td>-15</td>
<td>0</td>
<td>35</td>
<td>-75</td>
<td>15</td>
<td>25</td>
</tr>
</tbody>
</table>

```
SELECT x.pos, x.val, y.val
FROM x left outer join y
ON x.pos = y.pos and y.val > 0;
```

v.s.

```
SELECT x.pos, x.val, y.val
FROM x left outer join y
ON x.pos = y.pos
WHERE y.val > 0;
```
**Sparse vectors:**

\[ X = \begin{pmatrix} 50 & 0 & -30 & 60 & -80 & 0 & -90 & 10 & 0 \end{pmatrix} \]

\[ Y = \begin{pmatrix} -55 & 0 & 65 & -15 & 0 & 35 & -75 & 15 & 25 \end{pmatrix} \]

**SELECT** x.pos, x.val, y.val

**FROM** x left outer join y

**ON** x.pos = y.pos and y.val > 0;

v.s.

**SELECT** x.pos, x.val, y.val

**FROM** x left outer join y

**ON** x.pos = y.pos

**WHERE** y.val > 0;

---

<table>
<thead>
<tr>
<th>x.pos</th>
<th>x.val</th>
<th>y.val</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>Null</td>
</tr>
<tr>
<td>3</td>
<td>-30</td>
<td>65</td>
</tr>
<tr>
<td>4</td>
<td>60</td>
<td>Null</td>
</tr>
<tr>
<td>5</td>
<td>-80</td>
<td>Null</td>
</tr>
<tr>
<td>7</td>
<td>-90</td>
<td>Null</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>15</td>
</tr>
</tbody>
</table>
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:

<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:

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
Other Representation

Adding edge labels
Adding node labels…

```
<table>
<thead>
<tr>
<th>src</th>
<th>dst</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>Bob</td>
<td>3</td>
</tr>
<tr>
<td>Bob</td>
<td>Alice</td>
<td>1</td>
</tr>
<tr>
<td>Bob</td>
<td>Chris</td>
<td>2</td>
</tr>
<tr>
<td>Alice</td>
<td>David</td>
<td>9</td>
</tr>
<tr>
<td>Chris</td>
<td>David</td>
<td>5</td>
</tr>
<tr>
<td>Bob</td>
<td>Eve</td>
<td>1</td>
</tr>
<tr>
<td>David</td>
<td>Eve</td>
<td>1</td>
</tr>
</tbody>
</table>
```
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. python
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>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
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>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>

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

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

Switched (following Mitchell)
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>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>

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

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

Train weights \(w_0, w_1, w_2, w_3\) to minimize loss:

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

Gradient Descent:

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

Tom Mitchell: Machine Learning

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>
</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

Tom Mitchell: *Machine Learning*

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);

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:

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

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>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

SELECT

\[
W.w0+0.01*\text{sum}(d.Y - 1 + 1/(1+\exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3))) \text{ as } w0, \\
W.w1+0.01*\text{sum}(d.X1*(d.Y - 1 + 1/(1+\exp(W.w0+W.w1*d.X1+W.w2*d.X2+W.w3*d.X3)))) \text{ as } w1, \\
\]

FROM data d, W
WHERE W.k=1

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>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
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

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, 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
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**: \( (U) \)

• **Set difference**: \(-\),

• **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** ($\tau$)
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

\[
\delta
\]

\[
\Pi_{\text{snname, scity}}
\]

\[
\bowtie \bowtie \bowtie
\]

\[
\text{SELECT DISTINCT x.snname, x.scity FROM Supplier x, Supply y, Part z WHERE x.sno=y.sno and y.pno=z.pno and z.psize > 10;}
\]
Query Optimizer

• Rewrite one relational algebra expression to a better one
• Very brief review now, more details next lecture
Optimization

SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.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 = y.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

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

More about this next lecture

Push selections down
Relational Model -- Summary

• Schema v.s. Data
Relational Model -- Summary

• Schema v.s. Data
• Data is *normalized* (what is that?)
Relational Model -- Summary

- Schema v.s. Data
- Data is *normalized* (what is that?)
  - $1^{st}$ NF: relations are flat (also unordered)
  - BCNF (or $3^{rd}$ or $4^{th}$ NF…): split large table into many small (why?), need to join back
Relational Model -- Summary

• Schema v.s. Data
• Data is normalized (what is that?)
  – 1\textsuperscript{st} NF: relations are flat (also unordered)
  – BCNF (or 3\textsuperscript{rd} or 4\textsuperscript{th} NF...): split large table into many small (why?), need to join back
  – (Consequence: joins are really important)
Relational Model -- Summary

• Schema v.s. Data
• Data is *normalized* (what is that?)
  – 1\textsuperscript{st} NF: relations are flat (also unordered)
  – BCNF (or 3\textsuperscript{rd} or 4\textsuperscript{th} NF…): split large table into many small (why?), need to join back
    – (Consequence: joins are *really* important)
• Query language is SQL, or something equivalent, like relational algebra
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>

The SQL query works the same, regardless of the answers to these questions:

```
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)
How to Implement a Relational DBMS?
DBMS Architecture

Process Manager
- Admission Control
- Connection Mgr

Query Processor
- Parser
- Query Rewrite
- Optimizer
- Executor

Storage Manager
- Access Methods
- Lock Manager
- Buffer Manager
- Log Manager

Shared Utilities
- Memory Mgr
- Disk Space Mgr
- Replication Services
- Admin Utilities

Storage Manager
Disks

- Data resides persistently on disks
  - Your local disk, or a Network Attached Storage (NAS), or Amazon’s S3
- For processing, data must reside in main memory
Disks v.s. Main Memory

Disk
- Unit of data = 1 block
  4KB or 8KB or 16KB
- Access time\(^*\) = seek time + rotational latency + transfer rate
  \(\approx 12\text{ms} + 5\text{ms} + 150\text{MB/s}\)
  - Random access \(\approx 17\text{ ms}\)
  - Sequential access \(\approx 50\mu\text{s}\)
- Organization:
  - Heap file
  - Index file

Main memory
- Unit of data = 1 byte
- Access time\(^**\) = 50\text{ns}
- Organization:
  - Lists, arrays, hash tables, ...

Disks v.s. Main Memory

Disk
- Unit of data = 1 block 4KB or 8KB or 16KB
- Access time* = seek time + rotational latency + transfer rate ≈ 12ms + 5ms + 150MB/s
  - Random access ≈ 17 ms
  - Sequential access ≈ 50μs
- Organization:
  - Heap file
  - Index file

Main memory
- Unit of data = 1 byte
- Access time** = 50ns
- Organization:
  - Lists, arrays, hash tables, …

Orders of magnitude slower

Heap File

Data on disk is stored in files
Files consist of pages filled with records
A heap file is not sorted on any attribute

Student(sid: int, age: int, …)

<table>
<thead>
<tr>
<th>Page</th>
<th>Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>18 ...</td>
</tr>
<tr>
<td>70</td>
<td>21</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>40</td>
<td>19</td>
</tr>
<tr>
<td>80</td>
<td>19</td>
</tr>
<tr>
<td>60</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>21</td>
</tr>
<tr>
<td>50</td>
<td>22</td>
</tr>
</tbody>
</table>

1 record
1 page
Index

An index is a data structure stored on disk. It is stored in a file consisting of pages & records. But records are (search key value, record ID).

Index File

Data File (Sequential file)

1 data entry

1 page

Pointer to a record on disk
Clustered vs. Unclustered Index

Clustered = records close in index are close in data
Buffer Manager

Page requests from higher-level code

Access methods
Buffer pool manager

Buffer pool

Disk page

Free frame

Main memory

Disk

1 page corresponds to 1 disk block

Disk is a collection of blocks
Discussion

• Disks are necessary both for persistent storage, and in order to process data larger than main memory
• They are slow! Buffer pool mitigates this
• ”Cold v.s. Hot execution”: first time you execute the query it is slow; if you repeat it, then it is faster (WHY?)
Discussion

• Main idea for Distributed data processing: if we distribute the data to many servers, then the data will fit in main memory

• For that reason, they often do not implement “out of core” algorithm

• Our focus in this course is on distributed data processing, not on out-of-core.
Summary

• RDMBS are complex systems
• Need to know some of their basics inner workings in order to understand query performance

Next week: we start review query processing, optimization, and start discussion distributed query processing