What is this course about? Why take it?

1. IBM's Watson beats humans in Jeapordy!

How did Watson achieve that?
Watson devoured LOTS of data!

Google also devours LOTS of data!

2. “Structured” data with Google search results

How does Google know that?
3. Amazon’s “spot-on” recommendations

How does Amazon know that?

You guessed it! LOTS and LOTS of data!

And innumerable “traditional” applications
Large-scale data management systems are the cornerstone of many digital applications, both modern and traditional.

Data data everywhere,
All the wallets did shrink!
Data data everywhere,
Nor any moment to think?
CSE 190 will get you thinking about the fundamentals of database systems

1. How are large structured datasets stored and organized?
2. How are “queries” handled?
3. How to make the system faster?
4. Deeper and more recent issues

And now for the (boring) logistics …

Course Administrivia

❖ Lectures: MWF 2:00-2:50pm, WLH 2205
   *Attending ALL lectures is mandatory!*
❖ Instructor: Arun Kumar; arunkk@eng.ucsd.edu
❖ Office hours: Wed 3-4pm, 3218 EBU3b (CSE building)
❖ TAs: Daniel Reznikov and Anirudh Shekhawat
❖ Piazza: https://piazza.com/class/jfecln5fnl47cn?cid=5
❖ CSE190A on TritonEd for project submissions
❖ Bring iClicker to all lectures; register it on TritonEd!

http://cseweb.ucsd.edu/classes/sp18/cse190-a

Prerequisites

❖ CSE 132A is essential
❖ CSE 120 is also a prerequisite but can be relaxed if you have sufficient systems experience
   
   Email the instructor with justification for waivers!
❖ C++ is needed for course projects; check course webpage for resources

http://cseweb.ucsd.edu/classes/sp18/cse190-a
Grading

❖ Midterm Exam: 15%
  Date: Friday, Date TBD, in-class
❖ Final Exam: 35% (cumulative)
  Date: TBD, Room TBD
❖ Project 1 (Buffer Manager): 15%
❖ Project 2 (B+ Tree Index): 25%
❖ Surprise Quizzes (in-class with iclicker): 10%

http://cseweb.ucsd.edu/classes/sp18/cse190-a

Course Projects

❖ BadgerDB: an RDBMS “skeleton” in C++
❖ Project 1: Buffer Manager
  Implement “clock algorithm” for buffer replacement
❖ Project 2: B+ Tree Index
  Implement data structure and insert/update/delete ops
❖ Teams of 2 or 1 only; no sharing of code across teams! The University has strict rules on plagiarism.
❖ No late days

http://cseweb.ucsd.edu/classes/sp18/cse190-a

Course Outline

1. How are large structured datasets stored and managed?
2. How are queries handled?
3. How to make the system faster? Deeper and more recent issues
4. Databases, Big Data, NoSQL, etc.

The primary focus will be the relational data model and Relational Database Management Systems (RDBMS)

http://cseweb.ucsd.edu/classes/sp18/cse190-a
Relational model in a nutshell

Basically, Relation:Table :: Pilot:Driver (okay, a bit more)

<table>
<thead>
<tr>
<th>RatingID</th>
<th>Rating</th>
<th>Date</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>23294</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>07/20/15</td>
<td>4232</td>
<td>293</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/15</td>
<td>54551</td>
<td>846</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

The model formalizes “operations” to manipulate relations

Relational model in a nutshell

Invented by E. F. Codd in 1970s at IBM Research in CA

Relational DBMS in a nutshell

A software system to implement the relational model, i.e., enable users to manage data stored as relations

Relational DBMS in a nutshell

First RDBMSs: System R (IBM) and Ingres (Berkeley) in 1970s

A rare photo of the original System R manual

Mike Stonebraker won the Turing Award in 2015!
Relational DBMS in a nutshell

RDBMS software is now a USD 20+ billions/year industry; many open source RDBMSs also exist

People still start companies about what are basically RDBMSs!

Course Textbook

Prescribed:
“Database Management Systems” 3rd Edition
Raghu Ramakrishnan and Johannes Gehrke

Optional:
“Database Systems: The Complete Book”
H.G. Molina, J.D. Ullman, and J. Widom

Course Schedule

<table>
<thead>
<tr>
<th>Week</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction; Recap of Relational Algebra and SQL</td>
</tr>
<tr>
<td>1-2</td>
<td>Data Storage; Buffer Management; File Organization</td>
</tr>
<tr>
<td>3</td>
<td>Indexing (B+-Tree; Hash Index)</td>
</tr>
<tr>
<td>4</td>
<td>Sorting</td>
</tr>
<tr>
<td>4</td>
<td>Midterm Exam</td>
</tr>
<tr>
<td>5</td>
<td>No lectures (TA will present Project 2)</td>
</tr>
<tr>
<td>6-7</td>
<td>Relational Operator Implementations; Query Processing</td>
</tr>
<tr>
<td>7</td>
<td>Query Optimization</td>
</tr>
<tr>
<td>8</td>
<td>Transaction Management; Concurrency Control</td>
</tr>
<tr>
<td>9</td>
<td>Advanced: NoSQL stores; Column stores; In-memory RDBMSs</td>
</tr>
<tr>
<td>9</td>
<td>Advanced: MapReduce/Hadoop; Spark; ML Systems</td>
</tr>
</tbody>
</table>

General Dos and Do NOTs

Do:
❖ Raise your hand before asking questions during lectures
❖ Participate in class discussions and on Piazza
❖ Bring iclicker to every lecture
❖ Use “CSE190A:” as subject prefix for all emails to me/TAs

Do NOT:
❖ Use laptops, tablets, mobile phones, or any other electronic devices during lectures
❖ Use email as primary communication mechanism for doubts/questions instead of Office Hours
❖ Record or quote the instructor’s anecdotes out of class! 😊
Relational Model: Basic Terms

**What is a Relation?**
A glorified table!

**What are Attributes?**
These things

**What are Domains?**
The mathematical "domains" for the attributes
Integers, Real, ...

**What is Arity?**
Ratings
Number of attributes

**What are Tuples?**

**What is Cardinality?**
These things
Number of tuples

---

<table>
<thead>
<tr>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>79</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>07/20/15</td>
<td>4232</td>
<td>293</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/15</td>
<td>54551</td>
<td>846</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>79</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>07/20/15</td>
<td>4232</td>
<td>293</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/15</td>
<td>54551</td>
<td>846</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Referring to “tuples”: Two notations

1. Without using attribute names (positional/sequence)
2. Using attribute names (named/set)

<table>
<thead>
<tr>
<th>Ratings (R)</th>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>79</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>07/20/15</td>
<td>4232</td>
<td>293</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/15</td>
<td>54551</td>
<td>846</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

A tuple \( t \in R \)
\[ t[1] = 3.5 \]
\[ t.\text{NumStars} = 3.5 \]

Relational Model: Basic Terms

What is **Schema**?

<table>
<thead>
<tr>
<th>Ratings</th>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>79</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>07/20/15</td>
<td>4232</td>
<td>293</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/15</td>
<td>54551</td>
<td>846</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

The relation name, and the name and logical descriptions of the attributes (including domains)

Aka “metadata”

What is an **Instance**?

<table>
<thead>
<tr>
<th>Ratings</th>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>79</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>07/20/15</td>
<td>4232</td>
<td>293</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/15</td>
<td>54551</td>
<td>846</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Instance 2

A given relation populated with a set of tuples

(loose analogy: schema:instance::type:value in PL)

What is a **Relational Database**?

<table>
<thead>
<tr>
<th>Ratings</th>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>79</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Users</th>
<th>UserID</th>
<th>Name</th>
<th>Age</th>
<th>JoinDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>Alice</td>
<td>23</td>
<td>01/10/13</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Movies</th>
<th>MovieID</th>
<th>Name</th>
<th>ReleaseDate</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Inception</td>
<td>07/13/2010</td>
<td>Christopher Nolan</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
“Write Operations” on a Relation

❖ **Insert**
   Add tuples to a relation

❖ **Delete**
   Remove tuples from a relation (typically based on “predicate” matches, e.g., “NumStars <= 4.5”)

❖ **Modify**
   Logically, deletes + inserts, but typically implemented as in-place updates to a relation instance

“Read Operations” on a Relation

❖ **“Select”**
   Select all tuples from **Ratings** with “**UserID == 19**”

❖ **“Project”**
   Select only **Director** attribute from **Movies**

❖ **“Aggregate”**
   Select **Average** of all **NumStars** in **Ratings**

   And a few more formal “algebraic” operations …

Recap: Relational Algebra

- **Select**
- **Project**
- **Rename**
- **Cross Product** (aka Cartesian Product)
- Set Operations:
  - **Union**
  - **Set Difference**
Select

**Ratings (R)**

<table>
<thead>
<tr>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>79</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>07/20/15</td>
<td>80</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/14</td>
<td>79</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>4.5</td>
<td>03/05/14</td>
<td>80</td>
<td>16</td>
</tr>
</tbody>
</table>

*Example:* Get all ratings with 4.0 or more stars

**Select**

\[ \sigma_{\text{NumStars} \geq 4.0} (R) \]

“Operator” “Selection condition/predicate”

**Complex Selection Conditions**

<table>
<thead>
<tr>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>79</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>07/20/15</td>
<td>80</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/14</td>
<td>79</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>4.5</td>
<td>03/05/14</td>
<td>80</td>
<td>16</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{NumStars} \neq 4.0 \land \text{MovieID} = 20} (R) \]

**Basic Relational Operations**

Select

Project

Rename

Cross Product (aka Cartesian Product)

Set Operations:

Union

Set Difference
### Project

#### Ratings (R)

<table>
<thead>
<tr>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>79</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>07/20/15</td>
<td>80</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/14</td>
<td>79</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>4.5</td>
<td>03/05/14</td>
<td>80</td>
<td>16</td>
</tr>
</tbody>
</table>

**Example:** Get all MovieIDs

\[ \pi_{\text{MovieID}}(R) \]

“Projection list”

### Composition of Relational Ops

#### Basic Relational Operations

- **Select**
- **Project**
- **Rename**
- **Cross Product** (aka Cartesian Product)
- **Set Operations:**
  - **Union**
  - **Set Difference**
### Rename

**Ratings (R)**

<table>
<thead>
<tr>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>79</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>07/20/15</td>
<td>80</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/14</td>
<td>79</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>4.5</td>
<td>03/05/14</td>
<td>80</td>
<td>16</td>
</tr>
</tbody>
</table>

**Example:** Rename Timestamp to RateDate

\[
\rho_{\text{RatingID, NumStars, RateDate, UserID, MovieID}}(R) \\
\rho_{\text{C}(2 \rightarrow \text{RateDate})}(R)
\]

### Basic Relational Operations

- **Select**
- **Project**
- **Rename**
- **Cross Product** (aka Cartesian Product)

**Set Operations:**
- **Union**
- **Set Difference**

### Cross Product

**Users (U)**

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Age</th>
<th>JoinDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>Alice</td>
<td>23</td>
<td>01/10/13</td>
</tr>
<tr>
<td>80</td>
<td>Bob</td>
<td>41</td>
<td>05/10/13</td>
</tr>
</tbody>
</table>

**Movies (M)**

<table>
<thead>
<tr>
<th>MovieID</th>
<th>Name</th>
<th>ReleaseYear</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Inception</td>
<td>2010</td>
<td>Christopher Nolan</td>
</tr>
<tr>
<td>16</td>
<td>Avatar</td>
<td>2009</td>
<td>Jim Cameron</td>
</tr>
</tbody>
</table>

\[
U \times M
\]

- Cartesian product (construct all pairs of tuples across tables)
- Schema of output “concatenates” the input schemas
- Be careful with attribute name conflicts! Use Rename op
Cross Product

<UserID> | Name  | Age  | JoinDate  
---|---|---|---
79 | Alice | 23 | 01/10/13 
80 | Bob | 41 | 05/10/13 

<MovieID> | Name  | ReleaseYear | Director  
---|---|---|---
20 | Inception | 2010 | Christopher Nolan 
16 | Avatar | 2009 | Jim Cameron 

\[ \rho_C(1 \rightarrow U.\text{Name}(U)) \times \rho_C(1 \rightarrow M.\text{Name}(M)) \]

Basic Relational Operations

Select
Project
Rename
Cross Product (aka Cartesian Product)
Set Operations:
- **Union**
- **Set Difference**

Union

**R1**

<table>
<thead>
<tr>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>79</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>07/20/15</td>
<td>80</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/14</td>
<td>79</td>
<td>16</td>
</tr>
</tbody>
</table>

**R2**

<table>
<thead>
<tr>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/14</td>
<td>79</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>4.5</td>
<td>03/05/14</td>
<td>80</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>5.0</td>
<td>06/09/13</td>
<td>135</td>
<td>20</td>
</tr>
</tbody>
</table>

**R’ = R1 \cup R2**

<table>
<thead>
<tr>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>79</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>07/20/15</td>
<td>80</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/14</td>
<td>79</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>4.5</td>
<td>03/05/14</td>
<td>80</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>5.0</td>
<td>06/09/13</td>
<td>135</td>
<td>20</td>
</tr>
</tbody>
</table>
Basic Relational Operations

Select

Project

Rename

Cross Product (aka Cartesian Product)

Set Operations:

Union

Set Difference

Set Difference

R1

<table>
<thead>
<tr>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>79</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>07/20/15</td>
<td>80</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/14</td>
<td>79</td>
<td>16</td>
</tr>
</tbody>
</table>

R2

<table>
<thead>
<tr>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/14</td>
<td>79</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>4.5</td>
<td>03/05/14</td>
<td>80</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>5.0</td>
<td>06/09/13</td>
<td>135</td>
<td>20</td>
</tr>
</tbody>
</table>

R’ = R1 – R2

Basic Relational Operations

✔ Select

✔ Project

✔ Rename

✔ Cross Product (aka Cartesian Product)

Set Operations:

✔ Union

✔ Set Difference
Derived and Other Relational Ops

Set Operation:

- **Intersection**
- **Join**
- **Group By Aggregate**

Intersection

Set intersection of sets of tuples (instances)

Inputs must be "Union-compatible"

<table>
<thead>
<tr>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>79</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>07/20/15</td>
<td>80</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/14</td>
<td>79</td>
<td>16</td>
</tr>
</tbody>
</table>

\[ R_1 \cap R_2 \]

Derived and Other Relational Ops

Set Operation:

- **Intersection**
- **Join**
- **Group By Aggregate**
Join

\[ R \bowtie_{\text{JoinCondition}} M \]

- Equivalent Select on Cross Product, but “bypasses” full \( X \)
  \[ \sigma_{\text{JoinCondition}}(R \times M) \]
- Perhaps the most intensively studied Rel Op!
- Several “types” of Joins:
  
  Natural Join and Equi-Join
  
  Condition Join (aka Theta Join)
  
  Semi-Join, Inner Join, Outer Join, Anti-Join, etc.

Natural Join

\[ R \bowtie M \]

- Attribute names of “join attributes” need not be the same
- EqualityCondition is a general boolean expression (logical “and”, and/or “or”) of terms with equality predicates only
- Join attributes from both \( R \) and \( M \) in output (no Project)

Equi-Join

\[ R \bowtie_{\text{EqualityCondition}} M \]

- Generalization of the Natural Join

Perhaps the most important and common type of Join!

Lots of R&D on efficient implementations!
Equi-Join: Example

\[ T(J, K, P, Q) = R_1 \bowtie_{K=P} R_2 \]

<table>
<thead>
<tr>
<th>R_1(J,K)</th>
<th>R_2(P,Q)</th>
<th>T(J,K,P,Q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>J \ K</td>
<td>P \ Q</td>
<td>J \ K \ P \ Q</td>
</tr>
<tr>
<td>J K</td>
<td>P Q</td>
<td>J K P Q</td>
</tr>
<tr>
<td>10 x</td>
<td>x 4</td>
<td>10 x x 4</td>
</tr>
<tr>
<td>20 y</td>
<td>y 9</td>
<td>20 y y 9</td>
</tr>
<tr>
<td>30 x</td>
<td>x 8</td>
<td>30 x x 8</td>
</tr>
</tbody>
</table>

(Primary) Key-Foreign Key Join

- A special kind of equi-join
- One of the join attributes is the (Primary) Key of an input relation; the other is a Foreign Key in the other relation

Ratings

<table>
<thead>
<tr>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UID</th>
<th>MovielID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Users

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Age</th>
<th>JoinDate</th>
<th>Users</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Also a common and important (sub) type of join with even more specialized efficient implementations!

Condition Join (aka “Theta” Join)

- Generalization of the Equi-Join
  \[ R \bowtie_{JoinCondition} M \]
- Instead of just equality predicates, the JoinCondition can be any SelectionCondition involving >, ≥, <, ≤, =, and/or ≠
- Note: In both SelectionCondition and JoinCondition, simple legal “arithmetic expressions” are permissible
  \[ A \geq B, 2 * A \neq B, A + 1 < B / 2, A * B = C - 10, \text{etc.} \]

Condition Join: Example

\[ T(J, K, P, Q) = R_1 \bowtie_{J/2>Q} R_2 \]

<table>
<thead>
<tr>
<th>R_1(J,K)</th>
<th>R_2(P,Q)</th>
<th>T(J,K,P,Q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>J \ K</td>
<td>P \ Q</td>
<td>J \ K \ P \ Q</td>
</tr>
<tr>
<td>J K</td>
<td>P Q</td>
<td>J K P Q</td>
</tr>
<tr>
<td>10 x</td>
<td>x 4</td>
<td>10 x x 4</td>
</tr>
<tr>
<td>20 y</td>
<td>y 9</td>
<td>20 y y 9</td>
</tr>
<tr>
<td>30 x</td>
<td>x 12</td>
<td>30 x x 12</td>
</tr>
</tbody>
</table>

Perhaps the most difficult type of Join to implement efficiently!
Join Expressions

Can compose many joins into a single complex expression

Ratings
- RatingID
- NumStars
- Timestamp
- UserID
- MovieID

Users
- UserID
- UName
- Age
- JoinDate

Movies
- MovieID
- Name
- ReleaseYear
- Director

Users \( \bowtie \) Ratings \( \bowtie \) Movies

Q. What do we get as the output?

Taxonomy of Joins

All kinds of joins
- Outer joins
- Semi joins
- Anti joins
- Inner joins
- Theta joins
- Equi joins
- Key-Foreign Key Joins
- “Snowflake” joins
- “Star” joins
- Natural joins

Derived and Other Relational Ops

Set Operation:
- Intersection

Join

Group By Aggregate

NOT a part of relational algebra, but “Extended RA”!

Useful for “analytics” queries that aggregate numerical data

Set Operation:
- Intersection

Join

Group By Aggregate

NOT a part of relational algebra, but “Extended RA”!

Useful for “analytics” queries that aggregate numerical data

RatingID
- NumStars
- Timestamp
- UserID
- MovieID

What is the average rating for each movie?
- How many movies has each user rated?

Standard 5 numerical aggregations supported in SQL:
- Count, Sum, Average, Maximum, and Minimum
- Extra: Median, Mode, Variance, Standard Deviation, etc.
Group By Aggregate

\[ \gamma [X \text{Agg}(Y)] (R) \]

**Grouping Attributes**  A numerical attribute in R
(Subset of R’s attributes) **Aggregate Function**  (SUM, COUNT, AVG, MAX, MIN)

- Output schema will have X and an extra numerical attribute (result of the aggregate function)
- Can list multiple aggregate functions in the same operation

\[ R \]

<table>
<thead>
<tr>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>79</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>07/20/15</td>
<td>80</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/14</td>
<td>79</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>4.5</td>
<td>03/05/14</td>
<td>80</td>
<td>16</td>
</tr>
</tbody>
</table>

What is the average rating for each movie?

\[ R' \]

<table>
<thead>
<tr>
<th>MovieID</th>
<th>AVG(NumStars)</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>3.75</td>
</tr>
<tr>
<td>16</td>
<td>3.5</td>
</tr>
</tbody>
</table>

Group By Aggregate

The set of Grouping Attributes can be empty too!

<table>
<thead>
<tr>
<th>UserID</th>
<th>Name</th>
<th>Age</th>
<th>JoinDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>Alice</td>
<td>23</td>
<td>01/10/13</td>
</tr>
<tr>
<td>80</td>
<td>Bob</td>
<td>41</td>
<td>05/10/13</td>
</tr>
<tr>
<td>123</td>
<td>Carol</td>
<td>19</td>
<td>08/09/14</td>
</tr>
<tr>
<td>420</td>
<td>Dan</td>
<td>20</td>
<td>03/01/15</td>
</tr>
</tbody>
</table>

What is the average age of the users?

\[ \gamma [AVG(Age)] (U) \]

AVG(Age)

25.75

Derived and Other Relational Ops

Set Operation:
- ✓ Intersection
- ✓ Join
- ✓ Group By Aggregate
Recap: SQL

Basic Form of an SQL Query

```
SELECT [DISTINCT] target-list
FROM relation-list
[WHERE condition]
```

- Optional List of attributes to project
- List of relations (possibly with “aliases”)
- Selection/join condition (optional)
- List of relations

What does it mean logically?

1. Cross-product of relations in `relation-list`
2. If `condition` given, apply it to filter out tuples
3. Remove attributes not present in `target-list`
4. If `DISTINCT` given, deduplicate tuples in result

The above is only a logical interpretation. It is NOT a “plan” an RDBMS would use in general to run an SQL query!

Example SQL Query

```
SELECT M.Name
FROM Movies M
WHERE M.Year = 2013
```

Example: Get the names of movies released in 2013

<table>
<thead>
<tr>
<th>MovieID</th>
<th>Name</th>
<th>Year</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Inception</td>
<td>2010</td>
<td>Christopher Nolan</td>
</tr>
<tr>
<td>16</td>
<td>Avatar</td>
<td>2009</td>
<td>Jim Cameron</td>
</tr>
<tr>
<td>53</td>
<td>Gravity</td>
<td>2013</td>
<td>Alfonso Cuaron</td>
</tr>
<tr>
<td>74</td>
<td>Blue Jasmine</td>
<td>2013</td>
<td>Woody Allen</td>
</tr>
</tbody>
</table>
Example SQL Query

### Movies (M)

<table>
<thead>
<tr>
<th>MovieID</th>
<th>Name</th>
<th>Year</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Inception</td>
<td>2010</td>
<td>Christopher Nolan</td>
</tr>
<tr>
<td>16</td>
<td>Avatar</td>
<td>2009</td>
<td>Jim Cameron</td>
</tr>
<tr>
<td>53</td>
<td>Gravity</td>
<td>2013</td>
<td>Alfonso Cuaron</td>
</tr>
<tr>
<td>74</td>
<td>Blue Jasmine</td>
<td>2013</td>
<td>Woody Allen</td>
</tr>
</tbody>
</table>

**Example**: Get the names of movies from years other than 2013

```sql
SELECT M.Name
FROM Movies M
WHERE M.Year ≠ 2013
```

### Movies (M)

<table>
<thead>
<tr>
<th>MovieID</th>
<th>Name</th>
<th>Year</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Inception</td>
<td>2010</td>
<td>Christopher Nolan</td>
</tr>
<tr>
<td>16</td>
<td>Avatar</td>
<td>2009</td>
<td>Jim Cameron</td>
</tr>
<tr>
<td>53</td>
<td>Gravity</td>
<td>2013</td>
<td>Alfonso Cuaron</td>
</tr>
<tr>
<td>74</td>
<td>Blue Jasmine</td>
<td>2013</td>
<td>Woody Allen</td>
</tr>
</tbody>
</table>

**Example**: For which years do we have movie data?

```sql
SELECT M.Year
FROM Movies M
WHERE M.Year <> 2013
```
Example SQL Query

**SELECT M.Year FROM Movies M**

<table>
<thead>
<tr>
<th>MovieID</th>
<th>Name</th>
<th>Year</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Inception</td>
<td>2010</td>
<td>Christopher Nolan</td>
</tr>
<tr>
<td>16</td>
<td>Avatar</td>
<td>2009</td>
<td>Jim Cameron</td>
</tr>
<tr>
<td>53</td>
<td>Gravity</td>
<td>2013</td>
<td>Alfonso Cuaron</td>
</tr>
<tr>
<td>74</td>
<td>Blue Jasmine</td>
<td>2013</td>
<td>Woody Allen</td>
</tr>
</tbody>
</table>

SQL allows repetitions of tuples in a relation!
Not the same semantics as RA’s Project
 Called “bag semantics” vs. RA’s set semantics

DISTINCT in SQL

**SELECT DISTINCT M.Year FROM Movies M**

<table>
<thead>
<tr>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
</tr>
<tr>
<td>2009</td>
</tr>
<tr>
<td>2013</td>
</tr>
</tbody>
</table>

DISTINCT needed to achieve set semantics of RA’s Project in SQL

Aliases in SQL

**SELECT M.Name FROM Movies M**

WHERE M.Year = 2013

**SELECT Name FROM Movies**

WHERE Year = 2013

Why bother with the alias? Not needed here!

Aliases in SQL – Useful for Joins!

**SELECT M.Name FROM Movies M, Directors D**

WHERE D.Name = “Jim Cameron”

AND M.DirectorID = D.DID

Example: Get names of movies directed by “Jim Cameron”

Aliases help disambiguate attributes with the same name from multiple relations (or even a self-join!)
More SQL Examples

Example: Get names of movies released in 2013 by Woody Allen or some other director 50 years or older

```
SELECT M.Name
FROM Movies M, Directors D
WHERE (D.Name = "Woody Allen" OR D.Age >= 50) AND M.Year = 2013
AND M.DirectorID = D.DID
```

LIKE in SQL

```
SELECT DISTINCT M.Director
FROM Movies M
WHERE M.Name LIKE "Blue%"
```

```
SELECT M.Name FROM Movies M
WHERE M.Year = 2013
ORDER BY M.Name
```

```
SELECT M.Name FROM Movies M
WHERE M.Year >= 2010
ORDER BY M.Year
```

ORDER BY in SQL

Example: Get the directors of movies that start with “Blue”

```
SELECT M.Name FROM Movies M
WHERE M.Name LIKE "Blue%"
```

```
SELECT M.Name FROM Movies M
WHERE M.Year = 2013
ORDER BY M.Name
```

```
SELECT M.Name FROM Movies M
WHERE M.Year >= 2010
ORDER BY M.Year
```

LIMIT in SQL

```
SELECT M.Name FROM Movies M
WHERE M.Year >= 2010
ORDER BY M.Year
LIMIT 2
```

Useful for data readability
Ordering defined by domain semantics
Can specify DESC; multiple attributes

Also useful for data readability
Prevents “flooding” of screen with data
Be wary of using it without ORDER BY!
**UNION in SQL**

Get the IDs of users that have rated a movie directed by “Ang Lee” or a movie that released in 2013

```
SELECT R.UID FROM Ratings R, Movies M WHERE R.MID = M.MID AND M.Director = "Ang Lee"
UNION
SELECT R.UID FROM Ratings R, Movies M WHERE R.MID = M.MID AND M.Year = 2013
```

**Semantics of UNION in SQL**

**UNION implicitly deduplicates tuples (unlike SELECT)!**

*Q. How to retain duplicates with UNION?*

```
SELECT R.UID FROM Ratings R, Movies M WHERE R.MID = M.MID AND M.Director = "Ang Lee"
UNION ALL
SELECT R.UID FROM Ratings R, Movies M WHERE R.MID = M.MID AND M.Year = 2013
```

**INTERSECT in SQL**
EXCEPT (Set Difference) in SQL

```
SELECT R.UID
FROM Ratings R, Movies M
WHERE R.MID = M.MID AND M.Director = "Ang Lee"
EXCEPT
SELECT R.UID
FROM Ratings R, Movies M
WHERE R.MID = M.MID AND M.Year = 2013
```

The Contentious Bag Semantics!

- **EXCEPT ALL**: Subtract the number of repetitions
- **UNION ALL**: Add the number of repetitions
- **INTERSECT ALL**: Minimum of the number of repetitions
**Aggregate Functions in SQL**

```sql
SELECT COUNT(*)
FROM Movies M
WHERE M.Year > 2010
```

Movies (M)

<table>
<thead>
<tr>
<th>MovieID</th>
<th>Name</th>
<th>Year</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Inception</td>
<td>2010</td>
<td>Christopher Nolan</td>
</tr>
<tr>
<td>16</td>
<td>Avatar</td>
<td>2009</td>
<td>Jim Cameron</td>
</tr>
<tr>
<td>53</td>
<td>Gravity</td>
<td>2013</td>
<td>Alfonso Cuaron</td>
</tr>
<tr>
<td>91</td>
<td>Interstellar</td>
<td>2014</td>
<td>Christopher Nolan</td>
</tr>
</tbody>
</table>

How many movies came out after 2010?

```sql
SELECT COUNT(*)
FROM Movies M
WHERE M.Year > 2010
```

COUNT(*)

2

---

**5 Native Aggregate Functions in SQL**

- `COUNT ([DISTINCT] attribute)`
- `AVG ([DISTINCT] attribute)`
- `SUM ([DISTINCT] attribute)`
- `MAX (attribute)`
- `MIN (attribute)`

---

**How many directors do we have?**

```sql
SELECT COUNT(DISTINCT M.Director)
FROM Movies M
```

COUNT(DISTINCT M.Director)

5

---

**How many movies came out after 2010?**

```sql
SELECT COUNT(*)
FROM Movies M
WHERE M.Year > 2010
```
Aggregate Functions in SQL

### Ratings (R)

<table>
<thead>
<tr>
<th>RatingID</th>
<th>Stars</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>79</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>80</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>79</td>
<td>53</td>
</tr>
<tr>
<td>4</td>
<td>4.5</td>
<td>123</td>
<td>42</td>
</tr>
</tbody>
</table>

Which MovieID(s) have the highest rating?

```sql
SELECT DISTINCT R.MovieID
FROM Ratings R
WHERE R.Stars = (SELECT MAX(R2.Stars)
                 FROM Ratings R2)
```

Other attributes NOT allowed in the target-list as such!

### Group By Aggregate in SQL

\[ \gamma_{X,Agg(Y)}(R) \]

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

target-list must be in this form:
- \( X' \), \( \text{Agg}(Y) \)
- Subset of \( X \)

Condition on each group in aggregate

What is the average rating for each movie?

```sql
SELECT R.MovieID, AVG(R.Stars) AS AvgRating
FROM Ratings R
GROUP BY R.MovieID
```
### Group By Aggregate in SQL

#### Ratings (R)

<table>
<thead>
<tr>
<th>RatingID</th>
<th>Stars</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>79</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>80</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>79</td>
<td>53</td>
</tr>
<tr>
<td>4</td>
<td>4.5</td>
<td>123</td>
<td>42</td>
</tr>
</tbody>
</table>

```sql
SELECT R.MovieID, AVG(R.Stars) AS AvgRating
FROM Ratings R
GROUP BY R.MovieID
```

<table>
<thead>
<tr>
<th>MovieID</th>
<th>AvgRating</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>4.0</td>
</tr>
<tr>
<td>42</td>
<td>4.0</td>
</tr>
<tr>
<td>53</td>
<td>2.5</td>
</tr>
</tbody>
</table>

*One tuple in output per unique value of R.MovieID (aka “group”)*