What is this course about? Why take it?

A bit about the instructor

- Joined UC San Diego CSE in Fall’16
- Research: Databases, data science, machine learning
- PhD from University of Wisconsin-Madison (Summer’16)
- Thesis: Accelerating machine learning over joins
- Originally from southern India
1. IBM’s Watson beats humans in Jeopardy!

How did Watson achieve that?

Watson devoured LOTS of data!

2. “Structured” data with Google search results
How does Google know that?

Google also devours LOTS of data!

Knowledge Vault fuses all these signals together

- Data from web
  - Unstructured text
  - Semi-structured DCMs
  - Structured WebTables
- "Prior" data from FB

Details in a paper submitted to WWW'14 (Dong et al)

How does Amazon know that?

3. Amazon’s “spot-on” recommendations
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
The Age of “Big Data”/“Data Science”

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 6.00-6.50pm, PCYNH 106
  Attending ALL lectures is mandatory!
❖ Instructor: Arun Kumar; arunkk@eng.ucsd.edu
  Office hours: Thu 2-3pm, 3218 EBU3b (CSE building)
❖ TA: TBD; Office Hours: TBD
❖ Piazza: TBD
❖ CSE190 on TritonEd for course material/submissions
  http://cseweb.ucsd.edu/classes/sp17/cse190-d/

Prerequisites

❖ CSE 132A is absolutely essential
  Additionally, CSE 120, 132B might be helpful
  Otherwise, email the instructor with justification
❖ C++ is absolutely essential for course projects;
  check webpage for resources
  http://cseweb.ucsd.edu/classes/sp17/cse190-d/

Grading

❖ Midterm Exam: 20%
  Date: Friday, 04/28, in-class
❖ Final Exam: 30% (cumulative)
  Date: TBD, Room TBD
❖ Project 1 (Buffer Manager): 20%
❖ Project 2 (B+-Tree Index): 30%
  http://cseweb.ucsd.edu/classes/sp17/cse190-d/

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/sp17/cse190-d/
Course Outline

1. Storage, file structure, and database organization
2. Indexing, sorting, relational operator implementation, and query processing
3. Query optimization
4. Transactions, Big Data, NoSQL, etc.

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

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>

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

The model formalizes “operations” to manipulate relations
Relational DBMS in a nutshell

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

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

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

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>5-6</td>
<td>Relational Operator Implementations; Query Processing</td>
</tr>
<tr>
<td>6</td>
<td>Query Optimization</td>
</tr>
<tr>
<td>7</td>
<td>– No class (SIGMOD week) –</td>
</tr>
<tr>
<td>8</td>
<td>Transaction Management; Concurrency Control</td>
</tr>
<tr>
<td>9</td>
<td>Advanced: NoSQL stores; In-memory RDBMSs; Streaming DBMSs</td>
</tr>
<tr>
<td>9</td>
<td>Advanced: MapReduce/Hadoop; Spark; Advanced Analytics 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
- Use “CSE190D:” as subject prefix for all emails to me/TA

**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! 😊

Questions?

Recap: Relational Model
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?

What are Tuples?

What is Cardinality?

Referring to “tuples”: Two notations
1. Without using attribute names (positional/sequence)
2. Using attribute names (named/set)

What is Schema?
The relation name, and the name and logical descriptions of the attributes (including domains)
Aka “metadata”
### Relational Model: Basic Terms

#### What is an Instance?

<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>...</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).

#### “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

#### What is a Relational Database?

<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>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

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

A collection of relations; similarly, schema vs. 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

**Basic Relational Operations**

- **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**

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

**R'**

- Schema preserved
- Subset of tuples (satisfying selection condition)
Complex Selection Conditions

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

Basic Relational Operations

Select

Project

Rename

Cross Product (aka Cartesian Product)

Set Operations:

Union

Set Difference

Project

Ratings (R)

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

“Projection list”

Example: Get all MovieIDs

Tuple values “deduplicated”

(slightly different semantics in SQL)
**Composition of Relational Ops**

<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 UserID and NumStars of ratings with less than 3 stars

\[ \pi_{\text{UserID}, \text{NumStars}}(\sigma_{\text{NumStars} < 3.0}(R)) \]

**Basic Relational Operations**

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

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

**Rename**

<table>
<thead>
<tr>
<th>Ratings (R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RatingID</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

**Example:** Rename Timestamp to RateDate

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

\[ \rho C(2 \rightarrow \text{RateDate})(R) \]
Basic Relational Operations

Select
Project
Rename
Cross Product (aka Cartesian Product)

Set Operations:
Union
Set Difference

Cross Product

<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>

<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

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>

\[ R1 \cup R2 \]

*Union of sets of tuples (instances)*

*Inputs must have identical schema: “Union-compatible”*

---

**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>

\[ R1 \setminus R2 \]

*Set difference of sets of tuples (instances)*

*Inputs must be “Union-compatible”*
Set Difference

R1  
RatingID | NumStars | Timestamp  | UserID | MovieID  
--- | --- | --- | --- | ---  
1 | 3.5 | 08/27/15 | 79 | 20  
2 | 4.0 | 07/20/15 | 80 | 20  
3 | 2.5 | 08/02/14 | 79 | 16  

R2  
RatingID | NumStars | Timestamp  | UserID | MovieID  
--- | --- | --- | --- | ---  
3 | 2.5 | 08/02/14 | 79 | 16  
4 | 4.5 | 03/05/14 | 80 | 16  
5 | 5.0 | 06/09/13 | 135 | 20  

R' = R1 – R2  
RatingID | NumStars | Timestamp  | UserID | MovieID  
--- | --- | --- | --- | ---  
1 | 3.5 | 08/27/15 | 79 | 20  
2 | 4.0 | 07/20/15 | 80 | 20  

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

R1  
RatingID | NumStars | Timestamp  | UserID | MovieID  
--- | --- | --- | --- | ---  
1 | 3.5 | 08/27/15 | 79 | 20  
2 | 4.0 | 07/20/15 | 80 | 20  
3 | 2.5 | 08/02/14 | 79 | 16  

R2  
RatingID | NumStars | Timestamp  | UserID | MovieID  
--- | --- | --- | --- | ---  
3 | 2.5 | 08/02/14 | 79 | 16  
4 | 4.5 | 03/05/14 | 80 | 16  
5 | 5.0 | 06/09/13 | 135 | 20  

R1 ∩ R2  
Set intersection of sets of tuples (instances)

Inputs must be "Union-compatible"
### Intersection

#### 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 \cap 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>
</tbody>
</table>

### Derived and Other Relational Ops

#### Set Operation:

- **Intersection**

#### Join

- Group By Aggregate

### Natural Join

#### 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>

#### 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>

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

- 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

\[ R \bowtie M \]

<table>
<thead>
<tr>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
<th>Name</th>
<th>ReleaseYear</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.5</td>
<td>08/27/15</td>
<td>79</td>
<td>20</td>
<td>Inception</td>
<td>2010</td>
<td>Christopher Nolan</td>
</tr>
<tr>
<td>2</td>
<td>4.0</td>
<td>07/20/15</td>
<td>80</td>
<td>20</td>
<td>Inception</td>
<td>2010</td>
<td>Christopher Nolan</td>
</tr>
<tr>
<td>3</td>
<td>2.5</td>
<td>08/02/14</td>
<td>79</td>
<td>16</td>
<td>Avatar</td>
<td>2009</td>
<td>Jim Cameron</td>
</tr>
<tr>
<td>4</td>
<td>4.5</td>
<td>03/05/14</td>
<td>80</td>
<td>16</td>
<td>Avatar</td>
<td>2009</td>
<td>Jim Cameron</td>
</tr>
</tbody>
</table>

- “Join attributes”: attributes that determine matching tuples
  - Have same name in both inputs! “MovieID” in R and M
- Implicit equality condition on join attributes
  - If > 1 pair, implicit logical “and” of all equality terms
- Output schema concatenates input schemas
  - But join attributes appear only once in output (Project)

Equi-Join

- Generalization of the Natural Join

\[ R \bowtie_{EqualityCondition} M \]

- 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: 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</td>
<td>P</td>
<td>J</td>
</tr>
<tr>
<td>10</td>
<td>x</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>y</td>
<td>20</td>
</tr>
<tr>
<td>30</td>
<td>x</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>Q</td>
<td>K</td>
</tr>
<tr>
<td>x</td>
<td>4</td>
<td>x</td>
</tr>
<tr>
<td>y</td>
<td>9</td>
<td>y</td>
</tr>
<tr>
<td>x</td>
<td>8</td>
<td>x</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

\[ Ratings \bowtie_{UID=UserID} Users \]

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
  \[ R \bowtie \text{JoinCondition} M \]
- Instead of just equality predicates, the JoinCondition can be any SelectionCondition involving \(>, \geq, <, \leq, =, \text{and/or} \neq \)
- Note: In both SelectionCondition and JoinCondition, simple legal “arithmetic expressions” are permissible
  \[ A \geq B, 2 \times A \neq B, A + 1 < B / 2, A \times B = C - 10, \text{etc.} \]

Condition Join: Example

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

\[
\begin{array}{c|c}
J & K \\
\hline
10 & x \\
20 & y \\
30 & x \\
\end{array}
\begin{array}{c|c}
P & Q \\
\hline
x & 4 \\
y & 9 \\
x & 12 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
J & K & P & Q \\
\hline
10 & x & x & 4 \\
20 & y & x & 4 \\
20 & y & y & 9 \\
30 & x & x & 4 \\
30 & x & y & 9 \\
30 & x & x & 12 \\
\end{array}
\]

Perhaps the most difficult type of Join to implement efficiently!

Join Expressions

Can compose many joins into a single complex expression

Ratings

<table>
<thead>
<tr>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>UserID</th>
<th>MovieID</th>
</tr>
</thead>
</table>

Users

<table>
<thead>
<tr>
<th>UserID</th>
<th>UName</th>
<th>Age</th>
<th>JoinDate</th>
<th>Users</th>
</tr>
</thead>
</table>

Movies

<table>
<thead>
<tr>
<th>MovieID</th>
<th>Name</th>
<th>ReleaseYear</th>
<th>Director</th>
</tr>
</thead>
</table>

AllStuff

<table>
<thead>
<tr>
<th>UserID</th>
<th>UName</th>
<th>Age</th>
<th>JoinDate</th>
<th>RatingID</th>
<th>NumStars</th>
<th>Timestamp</th>
<th>MovieID</th>
<th>Name</th>
<th>ReleaseYear</th>
<th>Director</th>
</tr>
</thead>
</table>

Q. What do we get as the output?

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

Ratings
RatingID | NumStars | Timestamp | UserID | MovieID
---------|----------|-----------|--------|--------
1         | 3.5      | 08/27/15  | 79     | 20     
2         | 4.0      | 07/20/15  | 80     | 20     
3         | 2.5      | 08/02/14  | 79     | 16     
4         | 4.5      | 03/05/14  | 80     | 16     

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.

The set of Grouping Attributes can be empty too!

```
\gamma_{\text{MovieID, AVG(NumStars)}} (R)
```

```
\gamma_{\text{Avg(Age)}} (U)
```

What is the average age of the users?
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]
```

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 all movies released in 2013**

\[
\pi_{\text{Name}}(\sigma_{\text{Year}=2013}(M))
\]

**Example:**

Get the names of movies released in 2013

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

**Example SQL Query**

**Select movies from years other than 2013**

\[
\pi_{\text{Name}}(\sigma_{\text{Year} \neq 2013}(M))
\]

**Example:**

Get the names of movies from years other than 2013

```
SELECT M.Name
FROM Movies M
WHERE M.Year \neq 2013
```
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:* For which years do we have movie data?

\[
\pi_{\text{Year}}(M) = \{2010, 2009, 2013, 2013\}
\]

### DISTINCT in SQL

**SELECT DISTINCT**

```
SELECT DISTINCT M.Year
FROM Movies M
```

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
```

Why bother with the alias? Not needed here!
### Aliases in SQL – Useful for Joins!

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

```
SELECT M.Name
FROM Movies M, Directors D
WHERE D.Name = "Jim Cameron"
  AND M.DirectorID = D.DID
```

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

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

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

“%” matches any number of characters; “_” matches one

### ORDER BY in SQL

**Example:**

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

Useful for data readability

Ordering defined by domain semantics

Can specify DESC; multiple attributes
**LIMIT in SQL**

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

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**

```
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
```

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.Year = 2013
UNION ALL
```

---

**Movies (M)**

<table>
<thead>
<tr>
<th>MID</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>

**Ratings (R)**

<table>
<thead>
<tr>
<th>RID</th>
<th>Stars</th>
<th>UID</th>
<th>MID</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>

<table>
<thead>
<tr>
<th>MID</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>42</td>
<td>Life of Pi</td>
<td>2012</td>
<td>Ang Lee</td>
</tr>
<tr>
<td>53</td>
<td>Gravity</td>
<td>2013</td>
<td>Alfonso Cuaron</td>
</tr>
</tbody>
</table>
### INTERSECT in SQL

<table>
<thead>
<tr>
<th>RID</th>
<th>Stars</th>
<th>UID</th>
<th>MID</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>

<table>
<thead>
<tr>
<th>MID</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>42</td>
<td>Life of Pi</td>
<td>2012</td>
<td>Ang Lee</td>
</tr>
<tr>
<td>53</td>
<td>Gravity</td>
<td>2013</td>
<td>Alfonso Cuaron</td>
</tr>
</tbody>
</table>

```sql
```

### EXCEPT (Set Difference) in SQL

<table>
<thead>
<tr>
<th>RID</th>
<th>Stars</th>
<th>UID</th>
<th>MID</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>

<table>
<thead>
<tr>
<th>MID</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>42</td>
<td>Life of Pi</td>
<td>2012</td>
<td>Ang Lee</td>
</tr>
<tr>
<td>53</td>
<td>Gravity</td>
<td>2013</td>
<td>Alfonso Cuaron</td>
</tr>
</tbody>
</table>

```sql
```

### The Contentious Bag Semantics!

**UNION ALL**

- Add the number of repetitions

- **EXCEPT ALL**

- Subtract the number of repetitions
The Contention Bag Semantics!

Minimum of the number of repetitions

Aggregate Functions in SQL

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?

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)
Aggregate Functions in SQL

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 directors do we have?

SELECT COUNT(DISTINCT M.Director)
FROM Movies M

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?

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

Aggregate Functions in SQL

Group By Aggregate in SQL

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

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

target-list must be in this form:

- \( X' \), Agg(Y)
- Subset of X

Condition on each group in aggregate
### 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>

*What is the average rating for each movie?*

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

### Surprise Review Question!

**Movies (M)**

<table>
<thead>
<tr>
<th>MovieID</th>
<th>Name</th>
<th>Year</th>
<th>Director</th>
</tr>
</thead>
</table>

*Get the number of 5-star ratings for each movie directed by “Christopher Nolan”*

```
SELECT R.MovieID, COUNT(R.Stars) AS NumHighRatings
FROM Ratings R, Movies M
WHERE M.Director = "Christopher Nolan"
AND R.MovieID = M.MovieID
AND R.Stars = 5
GROUP BY R.MovieID
```