Computing on Data

Computing on Data

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Explain what computing on data means.
- Explain why computing on data is necessary for a database.
- Give examples of commutation framework for data.

What is “Computation”? Why Computing on Data?

- Computation--
  - A process that transforms one or more inputs to one or more outputs, following a well-defined “model”, or “algorithm.”
- Why computation?
  - It’s often the very purpose we use a computer.
  - That’s the subject of computer science!

- Why computing on data?
  - Because we cannot use data “as is” -- need computation to transform to what we want.
  - Because what we want from data vary in different situations.

Haven’t We Learned--
How to Compute on Data?

- We learned how to compute on numbers?
  - Arithmetic algebra: for numbers.
  - Defines: addition, subtraction, multiplication and division.

- But data is not just numbers.

- What is the closest we have?

Closest to “Data”: Maybe Matrices?

We can think a table as a matrix, and use linear algebra.

- It assumes “numbers”.

Beers

name	brewer	alcohol
Sam Adams	Boston Beer	4.9
Goose IPA	Goose Island	5.9
Summer Ale	Boston Beer	5.3

name	brewer	alcohol
1	1	4.9
2	2	5.9
3	1	5.3

Sells

bar	beer	price
John Bar	Sam Adams	5
Green Bar	Goose IPA	6
Purple Bar	Sam Adams	4

bar	beer	price
1	1	5
2	2	6
3	1	4

- Does it help transformation we need?
  - Find beers made by Boston Beer.
  - Which bars sell Boston Beer’s beers?

Computing on Data Depends on …

- What data?
  - Relations
  - Key-value pairs
  - Documents
  - Graphs

- What purpose?
  - Asking questions about the world we manage.
    - Which classes are taken by CS students?
    - Which bars sell Boston Beer’s beers?
  - Processing data for preparing another datasets
    - To normalize/reorganize/reformat into another datasets.

Examples of Data Computing Framework

- Relational data
  - Relational algebra: Foundation of the SQL query language.
  - Datalog: Popular mechanism for inferencing with relational data.
- Graph data
  - Graph is well-studied data structure, before it became a database model.
  - Finding neighbors, nodes matching paths, shortest paths, etc.
- Key-value data
  - MapReduce

Relational Algebra

Computing on Data

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Define what relational algebra is.
- Identify the operands and operators for relational algebra.

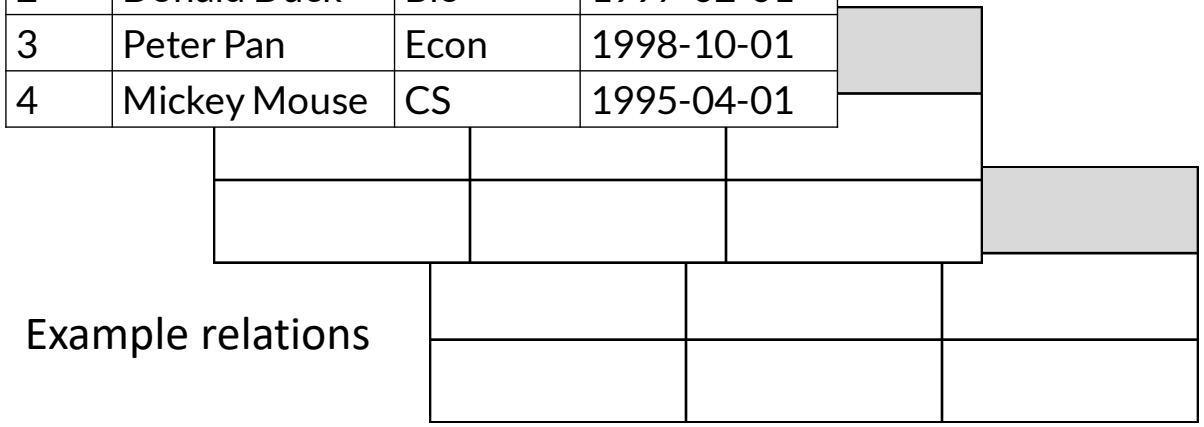
Algebra

- Mathematical system with:
  - **Operands**: Values to be operated upon (to generate new values)
  - **Operators***:* Symbols denoting operations that compute over operands and produce new values
- Examples
  - Arithmetic algebra
    - Operands: Numbers
    - Operators: Addition, subtraction, multiplication, division, etc.
  - Boolean algebra
    - Operands: Truth values false and true
    - Operators: And, or, not

Relational Algebra

- Operands: Relations, of course!

id	name	major	birthday
1	Bugs Bunny	CS	2004-11-06
2	Donald Duck	Bio	1997-02-01
3	Peter Pan	Econ	1998-10-01
4	Mickey Mouse	CS	1995-04-01



Example relations

- Operators
  - What do we want to compute over one or multiple relations?
  - Purpose: We want to compute on relations to “answer questions.”
    - What is the major of Bugs Bunny?
    - What majors do students have?
    - What courses are Bugs Bunny taking?

Relational Operators

- Operators: relations as input, new relation as output
- Basic operators
  - Reduction: Make one table smaller.
    - Selection
    - Projection
  - Combination: Combine two tables.
    - Set Operations
      - Union, difference
    - Cartesian Product
  - Renaming: Change attribute names.

- Derived operations
  - Intersection, complement
  - Joins (natural, equi-join, theta join, semi-join)

Relational Algebra: Basic Operators

Computing on Data

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

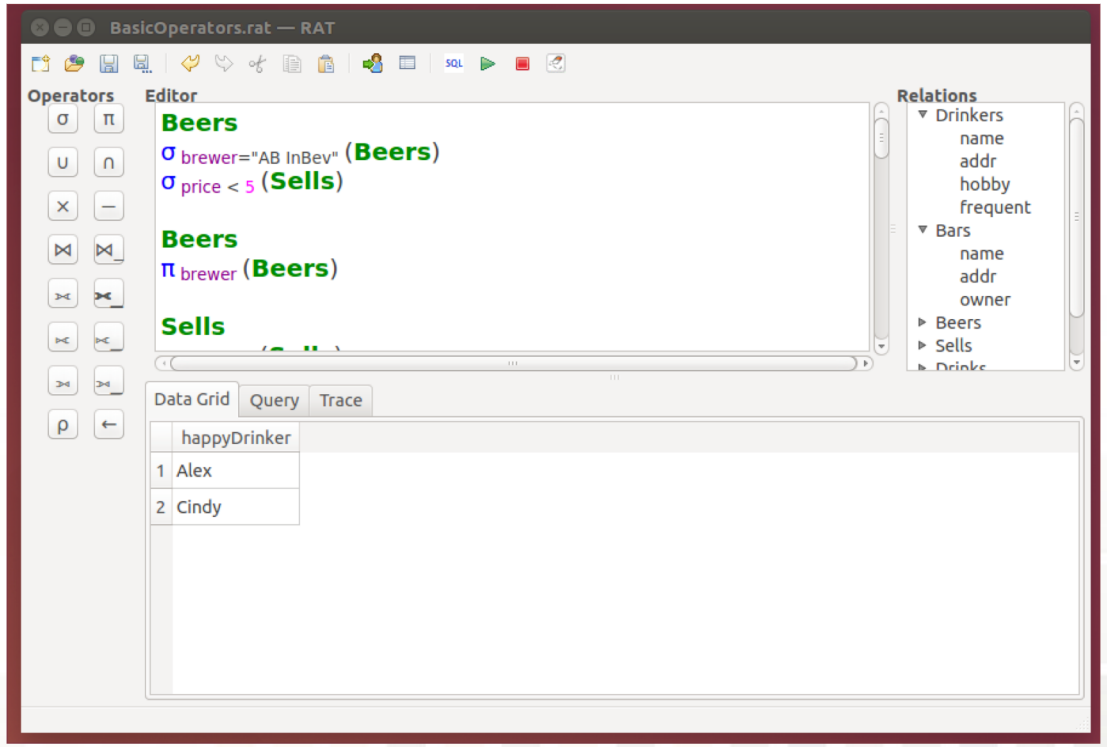
- Describe the operators and operands of relational algebra.
- Write a very simple relational algebra expression.
- State the general purposes of basic operations.
- Meet our new best friend RASQL!

Introducing RASQL

- One of the few softwares that speaks Relation Algebra (and SQL).
- Created by alumni of this class.
- Available at

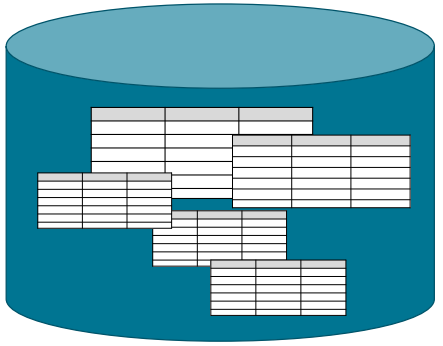
https://github.com/forward-uiuc/rasql

- We will use it to learn/demo RA!



Operators Operate upon Operands

- Relational algebra computes on relations.
- We refer to a relation as operand by its name.



Database is a set of tables

- A relation name is an operand, and a RA expression.

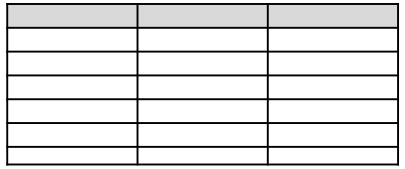
- What is the simplest relational algebra expression?

Basic Operators

- Reduction: *Make a table smaller.*
  - Selection σ
  - Projection π

What is the major of Bugs Bunny?

Students

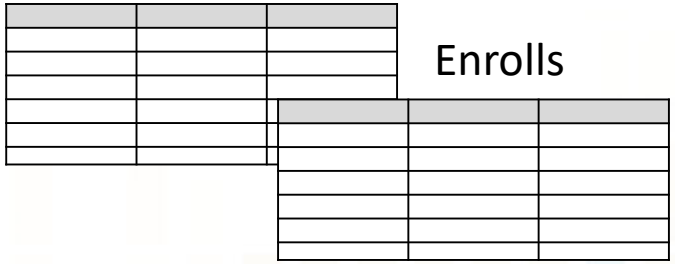


- Combination: *Combine two tables.*
  - Set Union $\cup$
  - Set Difference $-$
  - Cartesian Product $\times$

What courses are Bugs Bunny taking?

Students

Enrolls



- Renaming ρ: *Change attribute names*.

Relational Algebra:
Basic Operators for Reducing Relations

Computing on Data

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

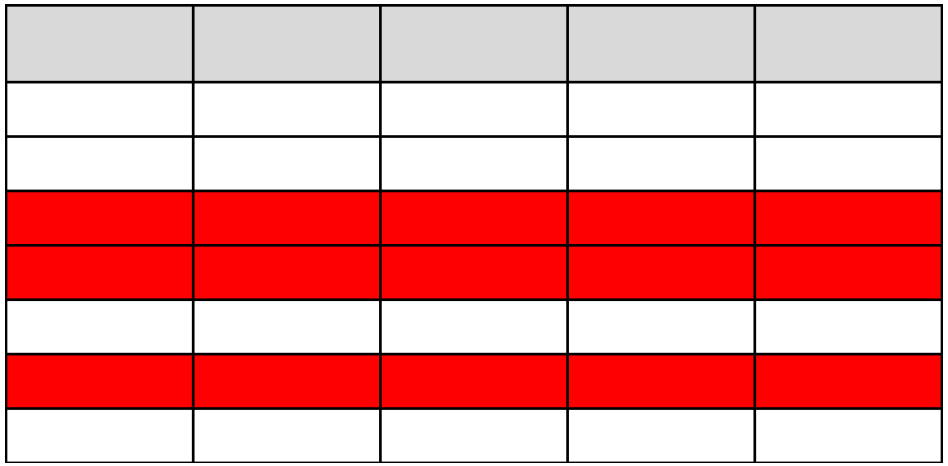
By the end of this video, you will be able to:

- Identify the basic operators that reduce tables to what we want.
- Use these operators to write RA expressions.

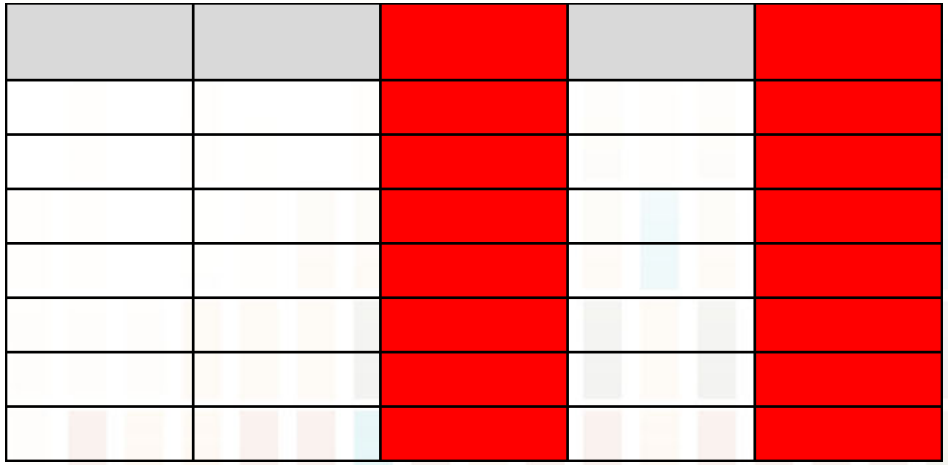
Reduction Operators

- Reducing rows: Selection: σ

Reducing rows from a table



- Reducing columns: Projection: π



Reducing columns from a table

Selection σ

- Notation: $\sigma_c(R)$
- Input: relation *R*
- Parameters: *c* is a condition: =, <, >, and, or, not.
- Output:
  - A relation as a subset of *R* that satisfies condition c
  - Schema: same as *R*

- What is the major of Bugs Bunny?
  - $\sigma_{\text{name}=\text{"Bugs Bunny"}}(\text{Students})$

id	name	major	birthday
1	Bugs Bunny	CS	2004-11-06
2	Donald Duck	Bio	1997-02-01
3	Peter Pan	Econ	1998-10-01
4	Mickey Mouse	CS	1995-04-01

Example relation Students

Selection Examples

- *Q1: Find beers that are made by “AB InBev”.*
- *Q2: Find beers that are made by ”Boston Beer”.*
- *Q3: Find beers that are sold for less than $5.*

Projection π

- Notation: $\pi_{A_1,\ldots,A_n}(R)$
- Input: relation $R(B_1, \ldots, B_m)$
- Parameters: $\{A_1, \ldots, A_n\} \subseteq \{B_1, \ldots, B_m\}$
- Output:
  - A relation of tuples with attribute $A_1, \ldots, A_n$ from *R*
  - Duplicates are removed.
  - Schema: $\{A_1, \ldots, A_n\}$

- What majors do students have?
  - $\pi_{\text{major}}(\text{Students})$

id	name	major	birthday
1	Bugs Bunny	CS	2004-11-06
2	Donald Duck	Bio	1997-02-01
3	Peter Pan	Econ	1998-10-01
4	Mickey Mouse	CS	1995-04-01

Example relation Students

Projection Examples

- *Q1: Find the brewers of beers.*

- *Q2: Find each bar-beer pair where a bar is selling a beer.*

Food for Thought

Sigma and Pi often are used together. Can you explain why?

Sigma Pi University of Illinois (SigmaPi, 1908)

References

- *SigmaPi, 1908*. Sigma Pi University of Illinois [Online image]. Retrieved from https://www.facebook.com/PhiofSigmaPi/.

Relational Algebra:
Basic Operators for Combining Relations

Computing on Data

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

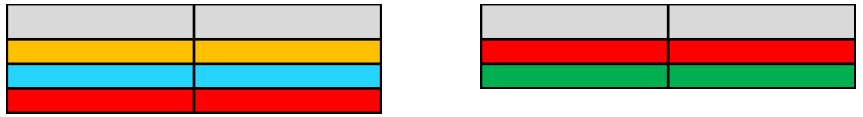
Learning Objectives

By the end of this video, you will be able to:

- Identify the basic operators that combine tables.
- Use these operators to write RA expressions.

Combination Operators

- Combining two relations R_1 and R_2 :



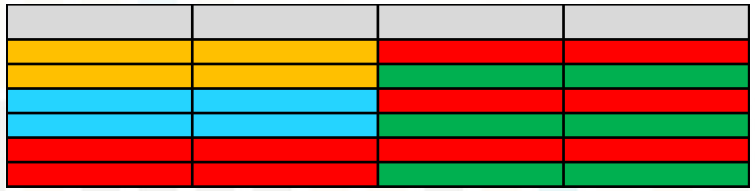
Two relations

- Set operations
  - Union: $R_1 \cup R_2$ (addition)
  - Difference: $R_1 - R_2$ (like subtraction)

Union

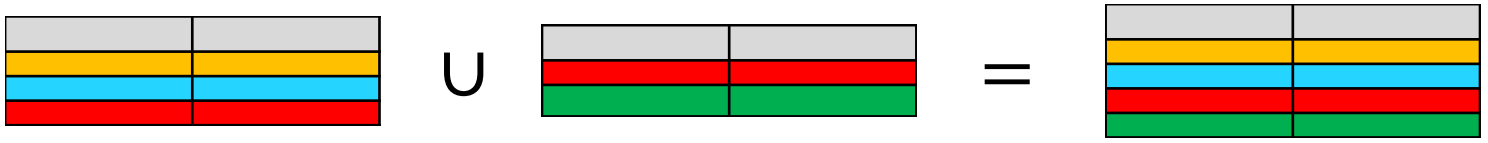
Difference

- Cartesian product (multiplication)
  - $R_1 \times R_2$



Cartesian product

Set Union



Union of two relations

- Notation: $R_1 \cup R_2$
- Input: R_1 and R_2, which have the same schema.
- Output:
  - A relation with all tuples in R_1 and R_2 (no duplicates)
  - Schema: same as R_1 and R_2

- *What are all the people in the Academic World?*
  - $\pi_{\text{name}}(\text{Students}) \cup \pi_{\text{name}}(\text{Professors})$

Union Examples

- *Q1: Find beers that are drunk or favorited by some drinkers.*

- *Q2: Find beer-drinker pairs where a bar is drunk or favorited by a drinker.*

Set Difference

Difference of two relations

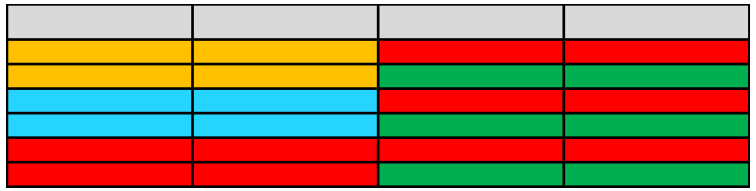
- Notation: $R_1 - R_2$
- Input: R_1 and R_2, which have the same schema.
- Output:
  - A relation with all tuples in R_1 but not in R_2
  - Schema: same as R_1 and R_2

- *What students took CS411 but not CS412?*
  - $\pi_{\text{name}}(\sigma_{\text{number}=\text{“CS411”}}\text{Enrolls}) - \pi_{\text{name}}(\sigma_{\text{number}=\text{“CS412”}}\text{Enrolls})$

Difference Examples

- *Q1: Find beers that are drunk by some but not favorited by any drinkers.*

Cartesian Product



Cartesian product of two relations

- Notation: $R_1 \times R_2$
- Input: $R_1(A_1, \ldots, A_n), R_2(B_1, \ldots, B_m)$.
- Output:
  - A relation consisting of all pairs of each tuple from R_1 and each tuple from R_2
  - Schema: $(A_1, \ldots, A_n, B_1, \ldots, B_m)$
    - Assume no attribute name clash. Or, use $R_1.A$ vs. $R_2.A$ to distinguish.

Cartesian Product: Example

Students S

id	name	major	birthday
1	Bugs Bunny	CS	2004-11-06
2	Donald Duck	ECE	1997-02-01
3	Peter Pan	SS	1998-10-01
4	Mickey Mouse	Music	1995-04-01

X

Enrolls E

id	number	term	grade
1	411	Fall 2017	A+
4	411	Fall 2017	B
1	426	Fall 2017	A

=

S.id	name	major	birthday	E.id	number	term	grade
1	Bugs Bunny	CS	2014-11-06	1	411	Fall 2017	A+
1	Bugs Bunny	CS	2014-11-06	4	411	Fall 2017	B
1	Bugs Bunny	CS	2014-11-06	1	426	Fall 2017	A
2	Donald Duck	ECE	1997-02-01	1	411	Fall 2017	A+
2	Donald Duck	ECE	1997-02-01	4	411	Fall 2017	B
2	Donald Duck	ECE	1997-02-01	1	426	Fall 2017	A
…	…	…	…	…	…	…	…

- Is the result of this operation useful?

Cartesian product of Students and Enrolls

Cartesian Product: Useful for Collecting Data into One Table

- $SE := \text{Students} \times \text{Enrolls} =$

S.id	name	major	birthday	E.id	number	term	grade
1	Bugs Bunny	CS	2014-11-06	1	411	Fall 2017	A+
1	Bugs Bunny	CS	2014-11-06	4	411	Fall 2017	B
1	Bugs Bunny	CS	2014-11-06	1	426	Fall 2017	A
2	Donald Duck	ECE	1997-02-01	1	411	Fall 2017	A+
2	Donald Duck	ECE	1997-02-01	4	411	Fall 2017	B
2	Donald Duck	ECE	1997-02-01	1	426	Fall 2017	A
…	…	…	…	…	…	…	…

Cartesian product of Students and Enrolls

- Upon A, we can now ask:

  What courses did Bugs Bunny take?

$$BB := \sigma_{\text{S.id=E.id AND name="Bugs Bunny"}}(SE)$$

$$Answer := \pi_{\text{number}}(BB) = \{(411), (426)\}$$

Overall:

$$\pi_{\text{number}}\ \sigma_{\text{S.id=E.student AND name="Bugs Bunny"}}(\text{Students} \times \text{Enrolls})$$

Cartesian-Product Examples

- *Q1: Find "happy drinkers": those drinkers who live on the same street (address equal) as some bars.*

Relational Algebra:
Basic Operator for Renaming Schema

Computing on Data

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Identify the basic operator that renames tables and attributes.
- Use it and other basic operators to write RA expressions.

Renaming

- Notation: $\rho_{S(B_1,\ldots,B_n)}(R)$
- Input: $R(A_1, \ldots, A_n)$
- Output:
  - The same relation instance
  - Schema $S(B_1, \ldots, B_n)$
- Why?
  - So that names are more meaningful.
    - $\rho_{(\text{student, advisor})}(\pi_{\text{name, advisor}}\text{Students})$
    - or: $\rho_{\text{name}\to\text{student}}(\pi_{\text{name, advisor}}\text{Students})$
  - So that names do not clash.
    - $\rho_{\text{id}\to\text{student}}(\text{Students}) \times \rho_{\text{id}\to\text{professor}}(\text{Professors})$

Cartesian-Product Examples

- *Q1: Rename Drinkers(name, addr, hobby, frequent) to Drinkers(drinker, dAddr, hobby, frequent).*

- *Q2: Rename Bars(name, addr, owner) to Bars(bar, bAddr, owner).*

- *Q3: Find drinkers who have a bar on the same street they live (addr equal). Rename these drinkers to "happyDrinker".*

Food for Thought

Can you imagine a "question" that cannot be expressed using the basic operators in relational algebra?

Relational Algebra: Derived Operators

Computing on Data

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

By the end of this video, you will be able to:

- Identify the derived operators of relational algebra.
- Perform computation on relations using basic/derived operators.

Derived Operations

- Some operations are very often used.
- But they can be expressed in terms of the basic operations.
- We thus define ”shorthand” for these operations.
- We will introduce:
  - Set Intersection $R_1 \cap R_2$
  - Theta Join $R_1 \bowtie_\theta R_2$
  - Natural Join $R_1 \bowtie R_2$

Set Intersection

Intersection of two relations

- Notation: $R_1 \cap R_2$
- Input: R_1 and R_2, which have the same schema.
- Output:
  - A relation with all tuples common in R_1 and R_2
  - Schema: same as R_1 and R_2

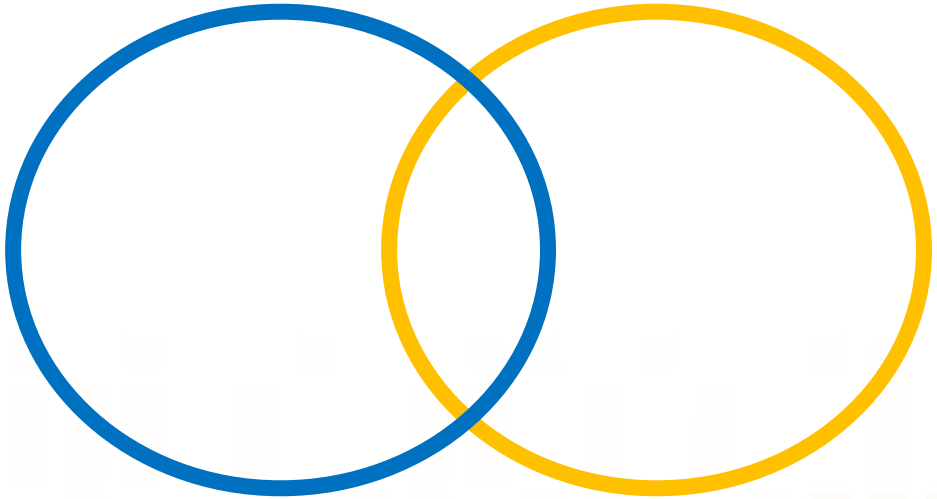
id	number	term	grade
1	411	Fall 2017	A+
4	411	Fall 2017	B
1	426	Fall 2017	A
2	426	Spring 2017	A-
2	225	Spring 2017	B

Example Enrolls relation

- Which students take both 411 and 426?
  - $\pi_{\text{id}}(\sigma_{\text{number="411"}}\text{Enrolls}) \cap \pi_{\text{id}}(\sigma_{\text{number="426"}},\text{Enrolls})$
    $= \{(1),(4)\} \cap \{(1),(2)\} = \{(1)\}.$

Intersection: Derivable from the Basics

- Intersection can be derived from Difference.
- $R_1 \cap R_2 = R_1 - (R_1 - R_2)$



Intersection of two relations

Intersection Examples

- *Q1: Find beers that are drunk and favorited by some drinkers.*

- *Q2: Find ”really-happy drinkers” who have bars on the same streets they live and the bars sell beers that they drink.*

Theta Join

- Notation: $R_1 \bowtie_\theta R_2$, where θ is a condition.
- Input: $R_1(A_1, \ldots, A_n), R_2(B_1, \ldots, B_m)$
- Parameters: θ is a condition over R_1 and/or R_2's attributes.
- Output:
  - A relation of the *product* of R_1 and R_2 *filtered* by condition θ.
  - I.e., can be expressed by $\sigma_\theta(R_1 \times R_2)$.
  - Schema: $(A_1, \ldots, A_n, B_1, \ldots, B_m)$
    - Assume no attribute name clash. Or, use $R_1.A$ vs. $R_2.A$ to distinguish.
- *What courses did Bugs Bunny take?*
  - $\pi_{\text{number}}\, \sigma_{\text{S.id=E.id AND name="Bugs Bunny"}}(\text{Students} \times \text{Enrolls})$
  - $\pi_{\text{number}}\, (\text{Students} \bowtie_{\textbf{S.id=E.id AND name="Bugs Bunny"}} \text{Enrolls})$

Theta-Join Examples

- *Q1: Find ”really-happy drinkers” who have bars on the same streets they live and the bars sell beer that they drink. Now use theta-joins instead of Cartesian products.*

Natural Join

id	name	major	birthday
1	Bugs Bunny	CS	2004-11-06
2	Donald Duck	ECE	1997-02-01
3	Peter Pan	SS	1998-10-01
4	Mickey Mouse	Music	1995-04-01

X

id	number	term	grade
1	411	Fall 2017	A+
4	411	Fall 2017	B
1	426	Fall 2017	A

=

S.id	name	major	birthday	E.id	number	term	grade
1	Bugs Bunny	CS	2014-11-06	1	411	Fall 2017	A+
1	Bugs Bunny	CS	2014-11-06	4	411	Fall 2017	B
1	Bugs Bunny	CS	2014-11-06	1	426	Fall 2017	A
2	Donald Duck	ECE	1997-02-01	1	411	Fall 2017	A+
2	Donald Duck	ECE	1997-02-01	4	411	Fall 2017	B
2	Donald Duck	ECE	1997-02-01	1	426	Fall 2017	A
…	…	…	…	…	…	…	…

Cartesian product of two tables

- Notation: $R_1 \bowtie R_2$
- Input: $R_1(A_1, \ldots, A_n), R_2(B_1, \ldots, B_m)$
- Output:
  - A relation combining all pairs of tuples in R_1 and R_2 that agree on their "common" attributes $\{A_1, \ldots, A_n\} \cap \{B_1, \ldots, B_m\}$, the *join attributes.*
  - I.e., can be expressed by $\sigma_\theta(R_1 \times R_2)$.
  - Schema: $\{A_1, \ldots, A_n\} \cup \{B_1, \ldots, B_m\}$, i.e., merging the two schemas and removing duplicates.
- *What courses did Bugs Bunny take?*
  - π_{number} (Students $\bowtie_{\text{S.id=E.id AND name="Bugs Bunny"}}$ Enrolls)
  - π_{number} ($\sigma_{\text{names="Bugs Bunny"}}$ **Students ⋈ Enrolls**)

Natural-Join Examples

- *Q1: Find ”really-happy drinkers” who have bars on the same streets they live and the bars sell beer that they drink. Now use natural joins if possible.*

Food for Thought

Joins are expensive. One major advantage that document-model databases claim is -- they do not need joins. Can you explain why?

mongoDB

CouchDB relax

```
{
  "_id": "<ObjectId1>",
  "name": "Samuel Adams",
  "brewer": {
    "name": "Boston Beer Company",
    "location": "Boston, Massachusetts"
  },
  "alcohol": 4.9,
  "type": "larger",
  "year introduced": 1984,
  "variants": [
    "<ObjectId2>",
    "<ObjectId3>"
  ]
}
```

Document model data

MapReduce for Key-Value Data

Computing on Data

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Kevin C.C. Chang, Professor

Computer Science @ Illinois

Learning Objectives

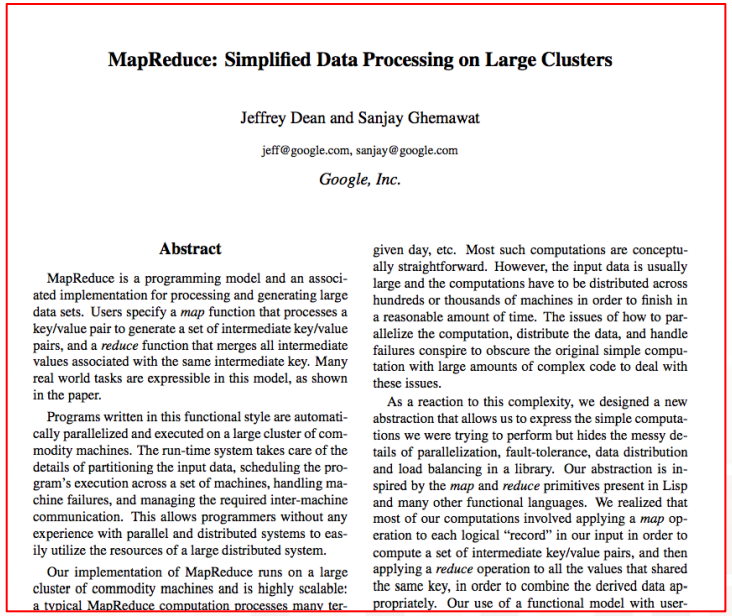
By the end of this video, you will be able to:

- Describe how MapReduce computes on key-value data.
- Design map and reduce functions for a computation task.
- Specify how relational operations can be performed by MapReduce.

MapReduce for Kev-Value Data

- A programming model for processing key-value data.
- For a **simplified** data processing framework over **large** clusters.
- Created at Google in 2004.
- Many implementations
  - E.g., Apache Hadoop

Apache Hadoop



MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many ter-

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then applying a *reduce* operation to all the values that shared the same key, in order to combine the derived data appropriately. Our use of a functional model with user-

Jeffrey Dean, Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters. OSDI 2004: 137-150

Map and Reduce in Functional Programming

- MapReduce: Inspired by map and reduce in functional programming.
- Map:
  - Input: Function M, list $L = [e_1, \dots, e_n]$
  - Output: map $M\ L = [M(e_1), \dots, M(e_n)]$
    - Apply given function to every element of the list.
  - E.g., map square [1, 2, 3, 4, 5] = [1, 4, 9, 16, 25]

- Reduce:
  - Input: Function R, list $L = [e_1, \dots, e_n]$
  - Output: reduce $R\ L = R(e_1, \dots, e_n)$.
    - Apply given function to aggregate all elements of the list.
  - E.g., reduce sum [1, 4, 9, 16, 25] = 55

Computing on K-V Data with MapReduce

- Database DB = $[e_1, \ldots, e_n]$, where $e_i = (k_i, v_i)$, a key-value pair.
- Programmer gives map function M and reduce function R.
- Execute M and R in MapReduce system with steps

  Map → Group → Reduce.

- **Map** on each $e_i = (k_i, v_i)$ with map function M.
  - $M(k_i, v_i) \rightarrow [(k_{i1}, v_{i1}), \ldots, (k_{im}, v_{im})]$
- **Group**
  - Organize (k_{ij}, v_{ij}) pairs by key k_{ij} -- each group is $(k_{ij}, \ [values\ with\ key\ k_{ij}])$.
- **Reduce** on each group $(k_{ij}, \ [values])$ by reduce function R.
  - Output: $[k_{ij}, R(values\ values\ with\ key\ k_{ij})]$

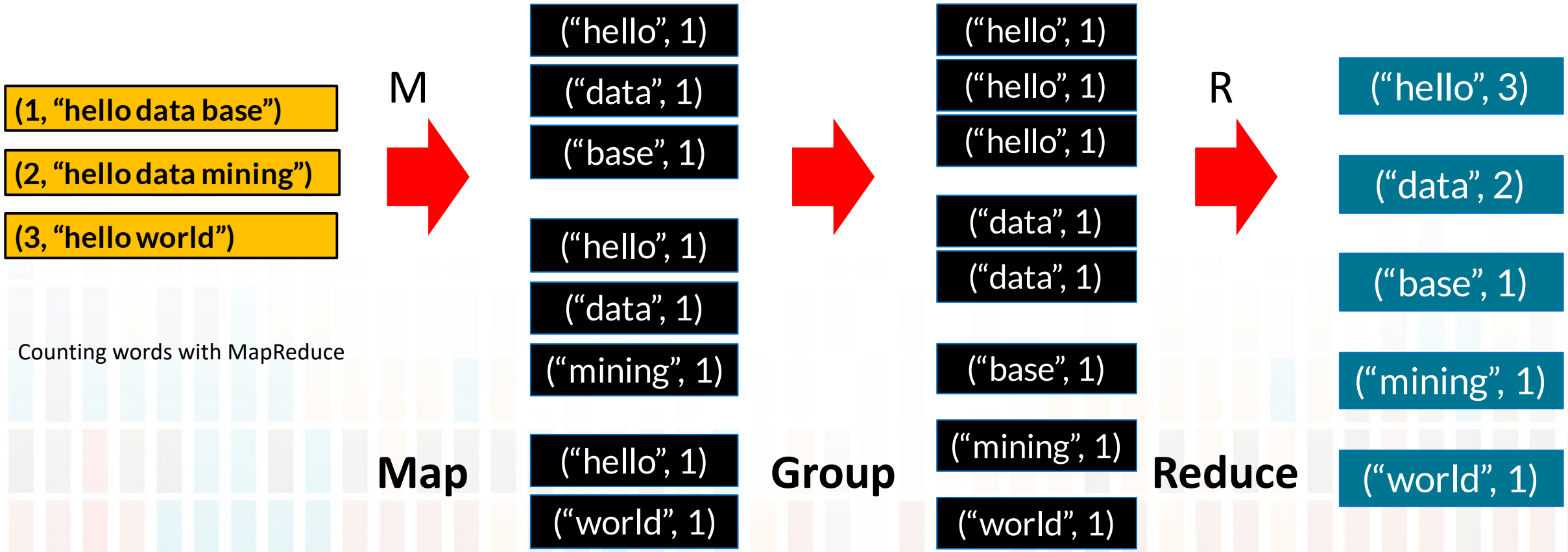
Example: Generating Word Cloud

- Application: Generating word cloud by counting words in database

Generating word cloud by counting words in database

The Standard Word Counting Example

- **function** M(k, v): **for each** word w **in** v: emit (w, 1)
- **function** R (k, values): emit (k, sum(values))

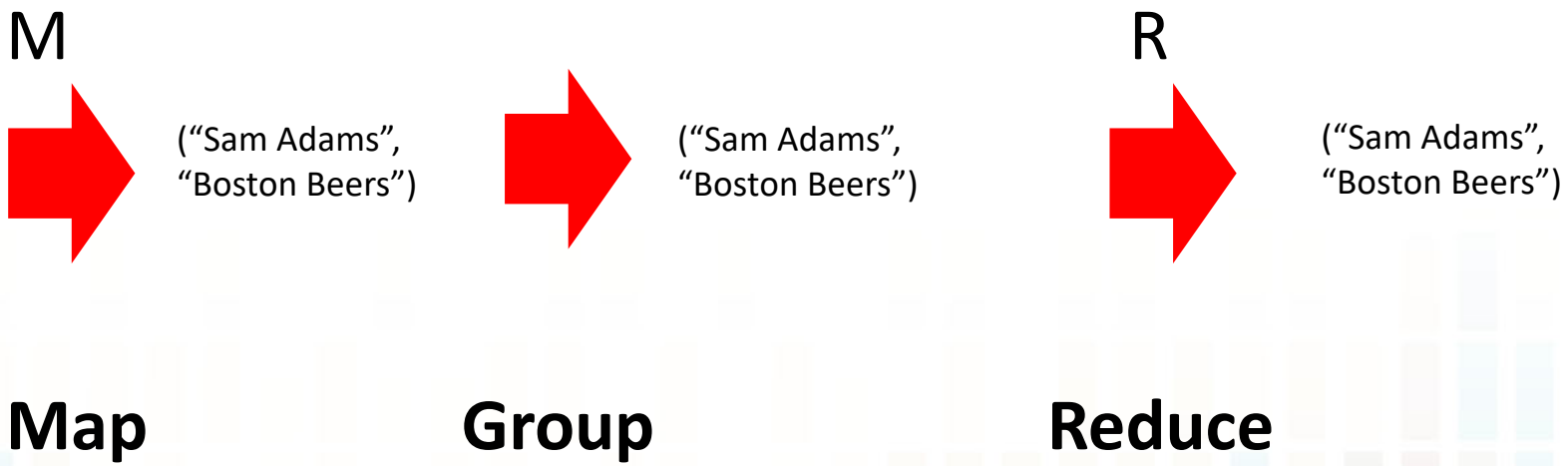


Counting words with MapReduce

Relational Operation with MapReduce?

- $\sigma_{\text{brewer}=\text{“Boston Beer”}}\text{Beers}$
- **function** M(k, v):. **if** v = “Boston Beer”: **emit** (k, v)
- **function** R (k, v): emit (k, v)

Key	Value
“Sam Adams”	“Boston Beer”
“Bud”	”AB InBev”
“Bud Lite”	”AB InBev”
“Coors”	“Coors”



Selection operation with MapReduce

Food for Thought

*Can we use MapReduce to perform **θ-join** over relations organized as key-value data?*

$$\textbf{Brewers} \bowtie_{\text{brewer="AB InBev" AND price<5.0}} \textbf{Price}$$

Brewer

Key	Value
"Sam Adams"	(brewer, "Boston Beer")
"Bud"	(brewer, "AB InBev")
"Bud Lite"	(brewer, "AB InBev")
"Coors"	(brewer, "Coors")

Brewer relation in the key-value model

Price

Key	Value
"Sam Adams"	(price, 5.0)
"Bud"	(price, 3.0)
"Bud Lite"	(price, 6.5)
"Coors"	(price, 2.5)

Price relation in the key-value model

The End

Figure

id	name	major	birthday
1	Bugs Bunny	CS	2004-11-06
2	Donald Duck	Bio	1997-02-01
3	Peter Pan	Econ	1998-10-01
4	Mickey Mouse	CS	1995-04-01

Figure

Beers

name	brewer	alcohol
Sam Adams	Boston Beer	4.9
Goose IPA	Goose Island	5.9
Summer Ale	Boston Beer	5.3

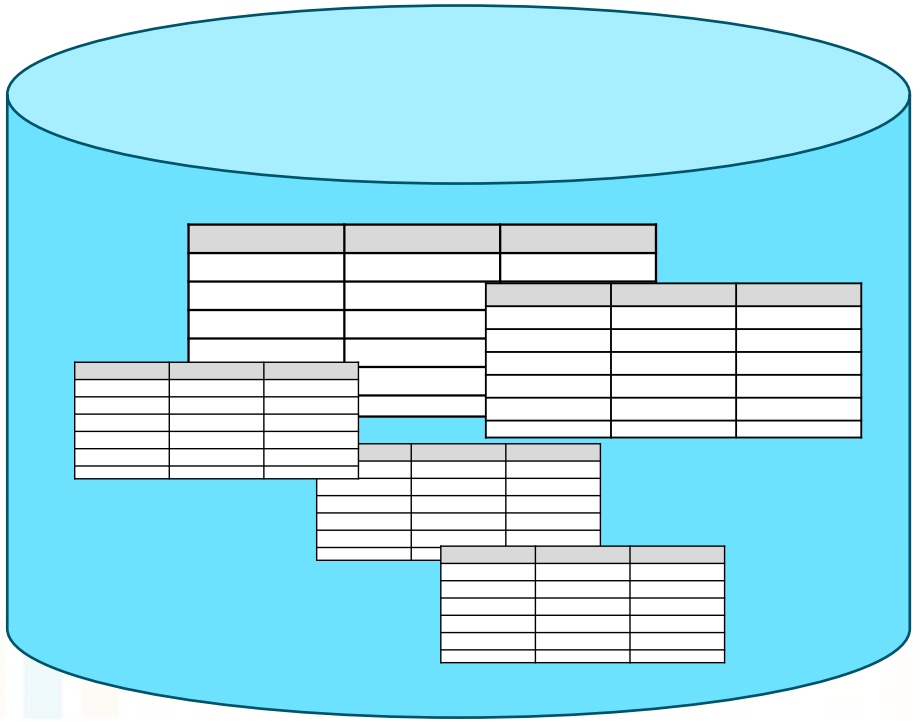
name	brewer	alcohol
1	1	4.9
2	2	5.9
3	1	5.3

Sells

bar	beer	price
John Bar	Sam Adams	5
Green Bar	Goose IPA	6
Purple Bar	Sam Adams	4

bar	beer	price
1	1	5
2	2	6
3	1	4

Figure



Figure


(1, “hello data base”)
(2, “hello data mining”)
(3, “hello world”)
(“hello”, 1)
(“data”, 1)
(“base”, 1)
(“hello”, 1)
(“data”, 1)
(“mining”, 1)
(“hello”, 1)
(“world”, 1)
(“hello”, 1)
(“hello”, 1)
(“hello”, 1)
(“data”, 1)
(“data”, 1)
(“base”, 1)
(“mining”, 1)
(“world”, 1)
(“hello”, 3)
(“data”, 2)
(“base”, 1)
(“mining”, 1)
(“world”, 1)

Figure

Key	Value
“Sam Adams”	“Boston Beer”
“Bud”	”AB InBev”
“Bud Lite”	”AB InBev”
“Coors”	“Coors”

Key	Value
“Sam Adams”	(brewer, “Boston Beer”)
“Bud”	(brewer, ”AB InBev”)
“Bud Lite”	(brewer, ”AB InBev”)
“Coors”	(brewer, “Coors”)

Key	Value
“Sam Adams”	(price, 5.0)
“Bud”	(price, 3.0)
“Bud Lite”	(price, 6.5)
“Coors”	(price, 2.5)