Updated 1/12/2024

Topic 1: Data models and query languages
Unit 1: SQL
Lecture 1

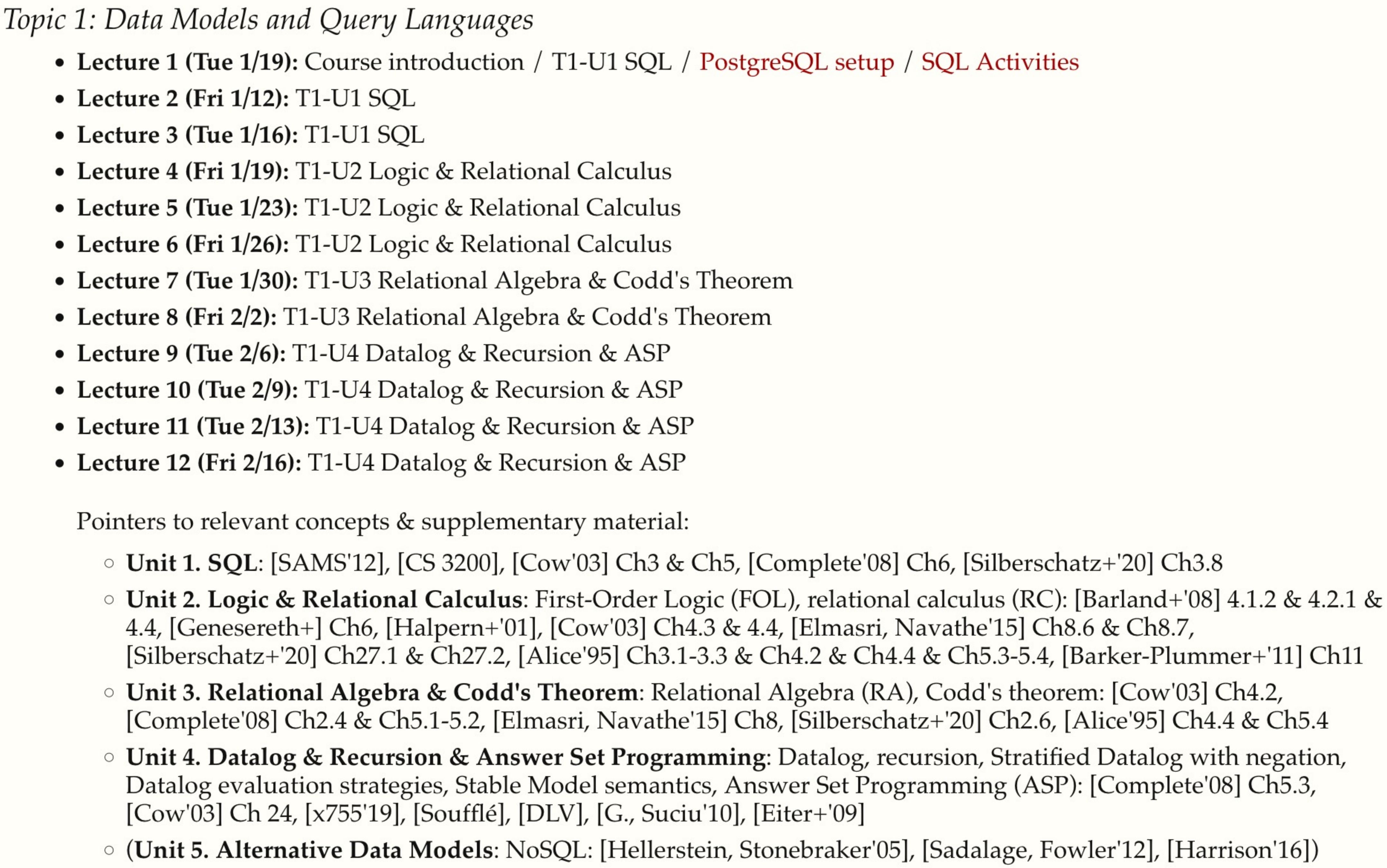
Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp24)

https://northeastern-datalab.github.io/cs7240/sp24/

1/9/2024

RELIMINARY



Topic 1: Data Models and Query Languages

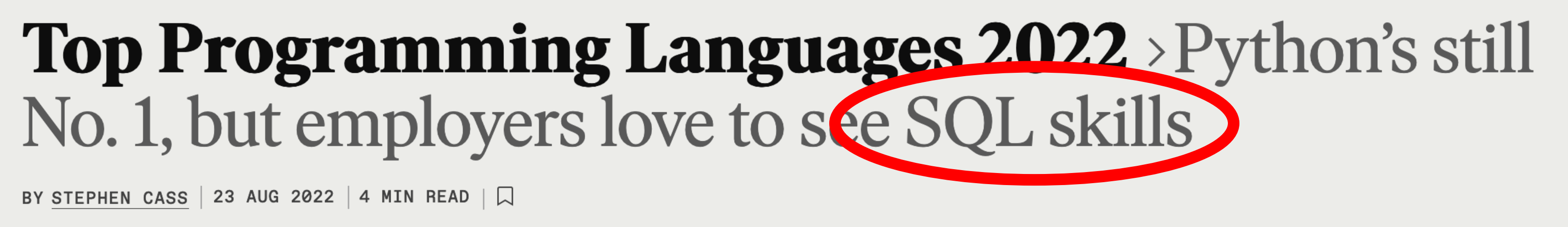
- **Lecture 1 (Tue 1/19):** Course introduction / T1-U1 SQL / PostgreSQL setup / SQL Activities
- **Lecture 2 (Fri 1/12):** T1-U1 SQL
- **Lecture 3 (Tue 1/16):** T1-U1 SQL
- **Lecture 4 (Fri 1/19):** T1-U2 Logic & Relational Calculus
- **Lecture 5 (Tue 1/23):** T1-U2 Logic & Relational Calculus
- **Lecture 6 (Fri 1/26):** T1-U2 Logic & Relational Calculus
- **Lecture 7 (Tue 1/30):** T1-U3 Relational Algebra & Codd's Theorem
- **Lecture 8 (Fri 2/2):** T1-U3 Relational Algebra & Codd's Theorem
- **Lecture 9 (Tue 2/6):** T1-U4 Datalog & Recursion & ASP
- **Lecture 10 (Tue 2/9):** T1-U4 Datalog & Recursion & ASP
- **Lecture 11 (Tue 2/13):** T1-U4 Datalog & Recursion & ASP
- **Lecture 12 (Fri 2/16):** T1-U4 Datalog & Recursion & ASP

Pointers to relevant concepts & supplementary material:

- **Unit 1. SQL**: [SAMS'12], [CS 3200], [Cow'03] Ch3 & Ch5, [Complete'08] Ch6, [Silberschatz+'20] Ch3.8
- **Unit 2. Logic & Relational Calculus**: First-Order Logic (FOL), relational calculus (RC): [Barland+'08] 4.1.2 & 4.2.1 & 4.4, [Genesereth+] Ch6, [Halpern+'01], [Cow'03] Ch4.3 & 4.4, [Elmasri, Navathe'15] Ch8.6 & Ch8.7, [Silberschatz+'20] Ch27.1 & Ch27.2, [Alice'95] Ch3.1-3.3 & Ch4.2 & Ch4.4 & Ch5.3-5.4, [Barker-Plummer+'11] Ch11
- **Unit 3. Relational Algebra & Codd's Theorem**: Relational Algebra (RA), Codd's theorem: [Cow'03] Ch4.2, [Complete'08] Ch2.4 & Ch5.1-5.2, [Elmasri, Navathe'15] Ch8, [Silberschatz+'20] Ch2.6, [Alice'95] Ch4.4 & Ch5.4
- **Unit 4. Datalog & Recursion & Answer Set Programming**: Datalog, recursion, Stratified Datalog with negation, Datalog evaluation strategies, Stable Model semantics, Answer Set Programming (ASP): [Complete'08] Ch5.3, [Cow'03] Ch 24, [x755'19], [Soufflé], [DLV], [G., Suciu'10], [Eiter+'09]
- (**Unit 5. Alternative Data Models**: NoSQL: [Hellerstein, Stonebraker'05], [Sadalage, Fowler'12], [Harrison'16])

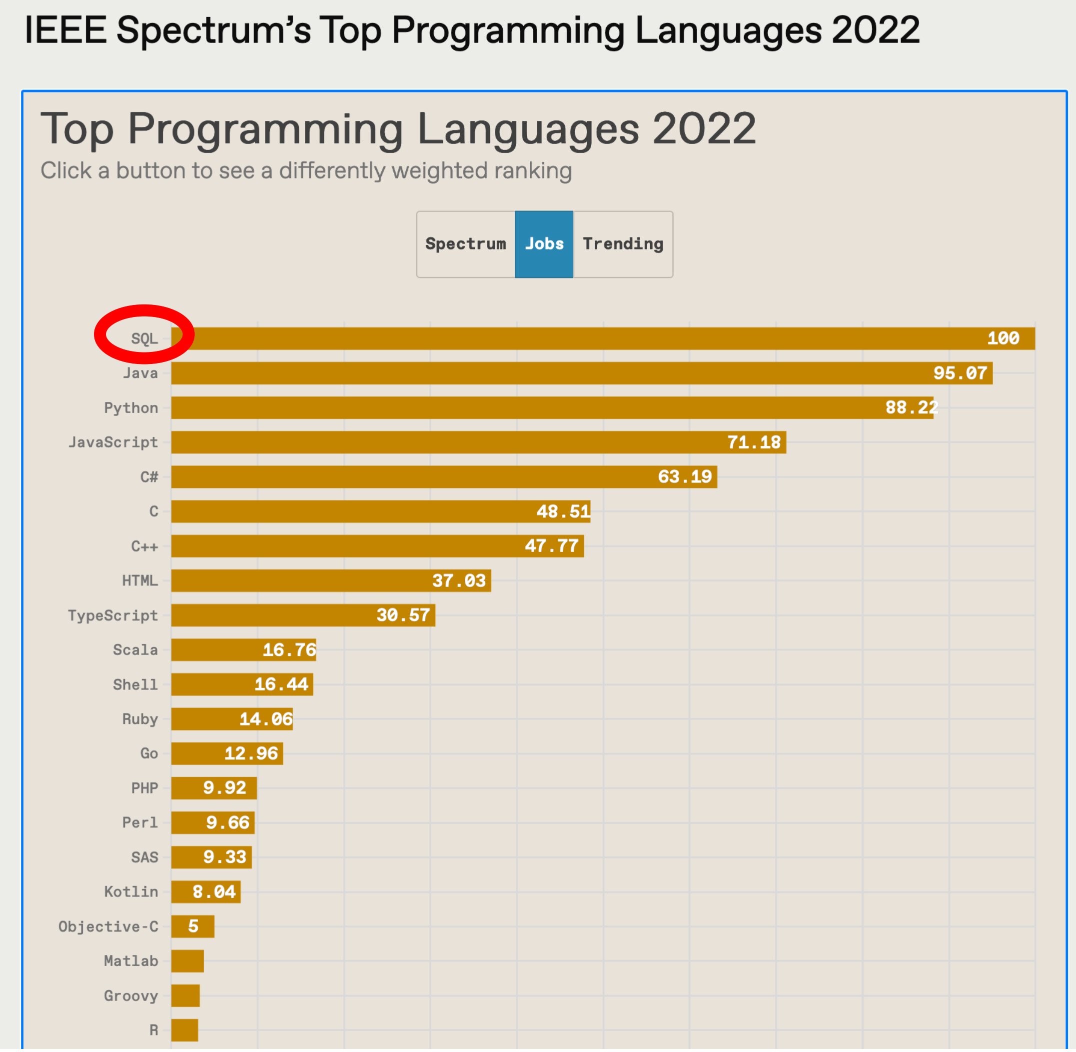
Outline: T1-U1: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Union and Theta Joins
  - Nulls & Outer joins
  - Top-k
  - [Recursion: moved to T1-U4: Datalog]

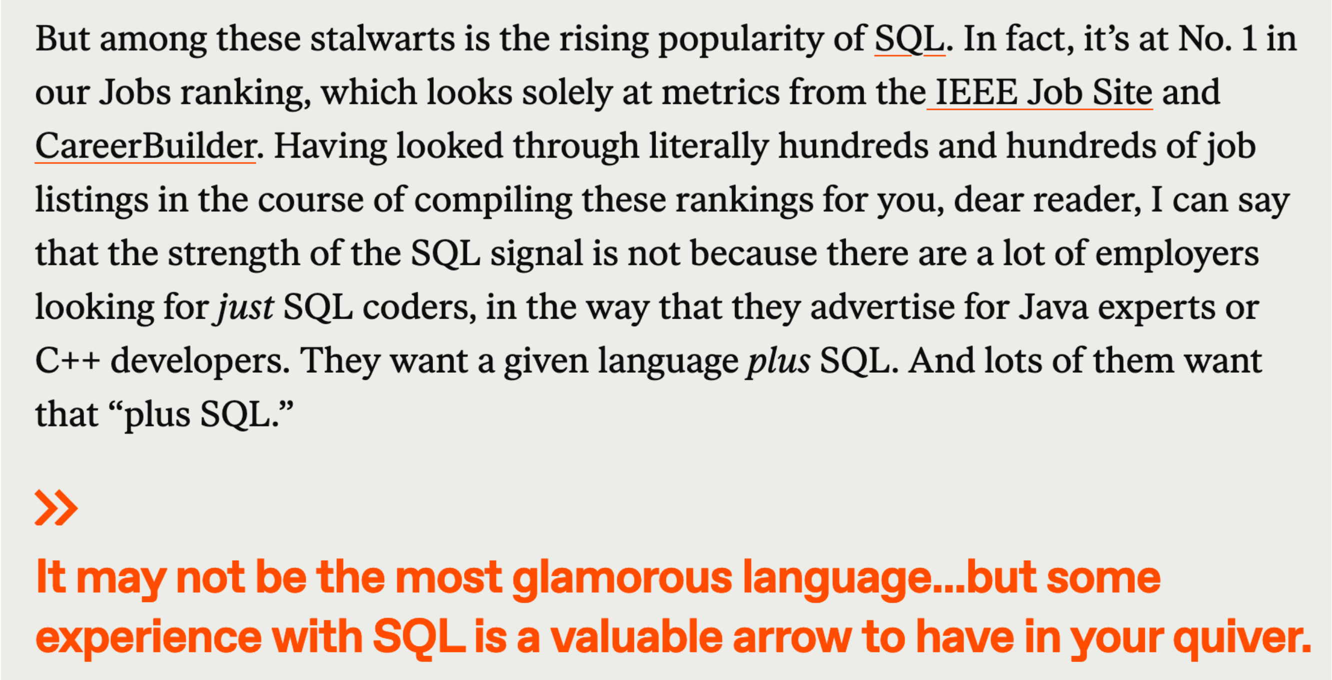


Top Programming Languages 2022 › Python's still No. 1, but employers love to see SQL skills

BY STEPHEN CASS | 23 AUG 2022 | 4 MIN READ |



IEEE Spectrum's Top Programming Languages 2022



But among these stalwarts is the rising popularity of SQL. In fact, it's at No. 1 in our Jobs ranking, which looks solely at metrics from the IEEE Job Site and CareerBuilder. Having looked through literally hundreds and hundreds of job listings in the course of compiling these rankings for you, dear reader, I can say that the strength of the SQL signal is not because there are a lot of employers looking for *just* SQL coders, in the way that they advertise for Java experts or C++ developers. They want a given language *plus* SQL. And lots of them want that "plus SQL."

»

It may not be the most glamorous language...but some experience with SQL is a valuable arrow to have in your quiver.

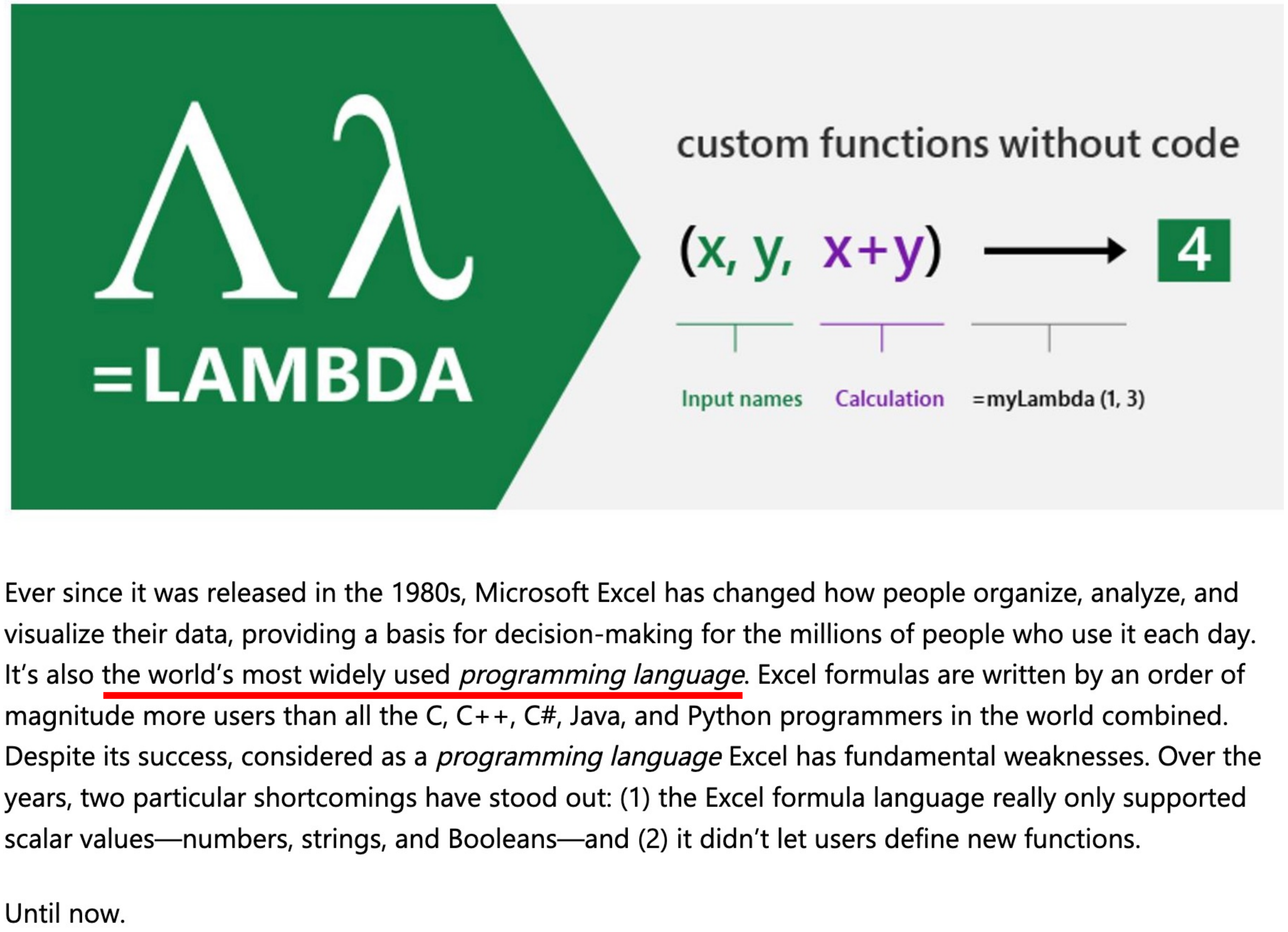
Source: https://spectrum.ieee.org/top-programming-languages-2022

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Fun question: What is the most popular PL?

?

Fun question: What is the most popular PL?



Ever since it was released in the 1980s, Microsoft Excel has changed how people organize, analyze, and visualize their data, providing a basis for decision-making for the millions of people who use it each day. It's also the world's most widely used *programming language*. Excel formulas are written by an order of magnitude more users than all the C, C++, C#, Java, and Python programmers in the world combined. Despite its success, considered as a *programming language* Excel has fundamental weaknesses. Over the years, two particular shortcomings have stood out: (1) the Excel formula language really only supported scalar values—numbers, strings, and Booleans—and (2) it didn't let users define new functions.

Until now.

Possibly interesting class scribe: Why is Excel Turing-complete?

Source: https://www.microsoft.com/en-us/research/blog/lambda-the-ultimatae-excel-worksheet-function/

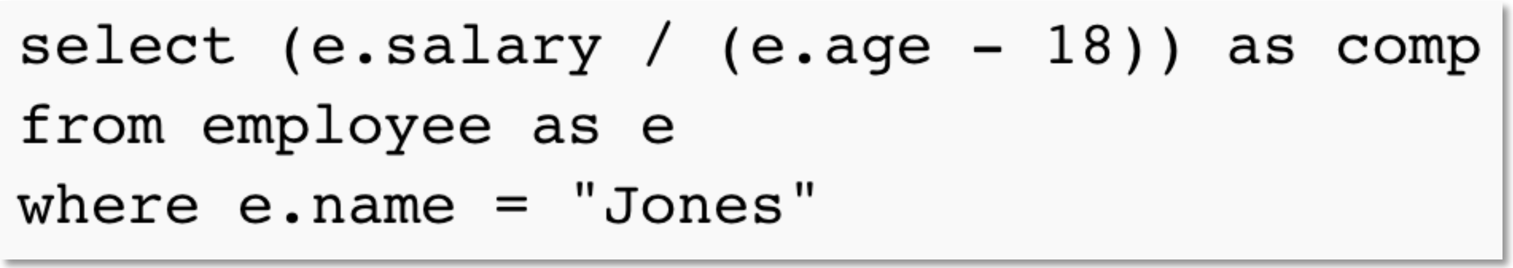
Structured Query Language: SQL

- Influenced by relational calculus (= First Order Logic)

- SQL is a declarative query language
  - We say what we want to get
  - We don’t say how we should get it ("separation of concerns")

SQL: Declarative Programming

SQL



```
select (e.salary / (e.age - 18)) as comp
from employee as e
where e.name = "Jones"
```

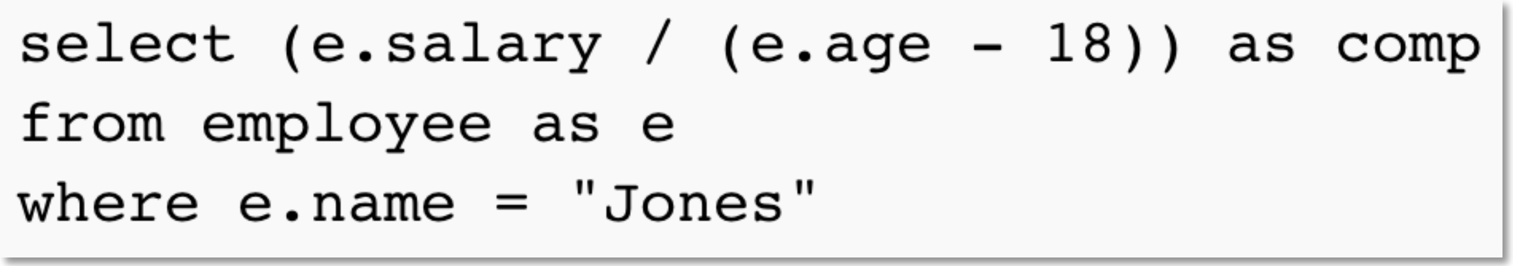
Declarative Language: you say what you want without having to say how to do it.

Procedural Language: you have to specify exact steps to get the result.

SQL: was not the only Attempt

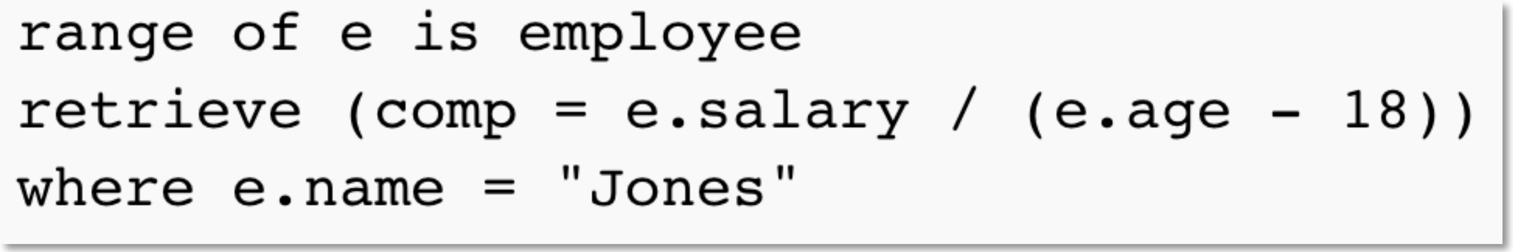
reading order:

SQL



3	`select (e.salary / (e.age - 18)) as comp`
1	`from employee as e`
2	`where e.name = "Jones"`

QUEL



1	`range of e is employee`
3	`retrieve (comp = e.salary / (e.age - 18))`
2	`where e.name = "Jones"`

Commercially not used anymore since ~1980

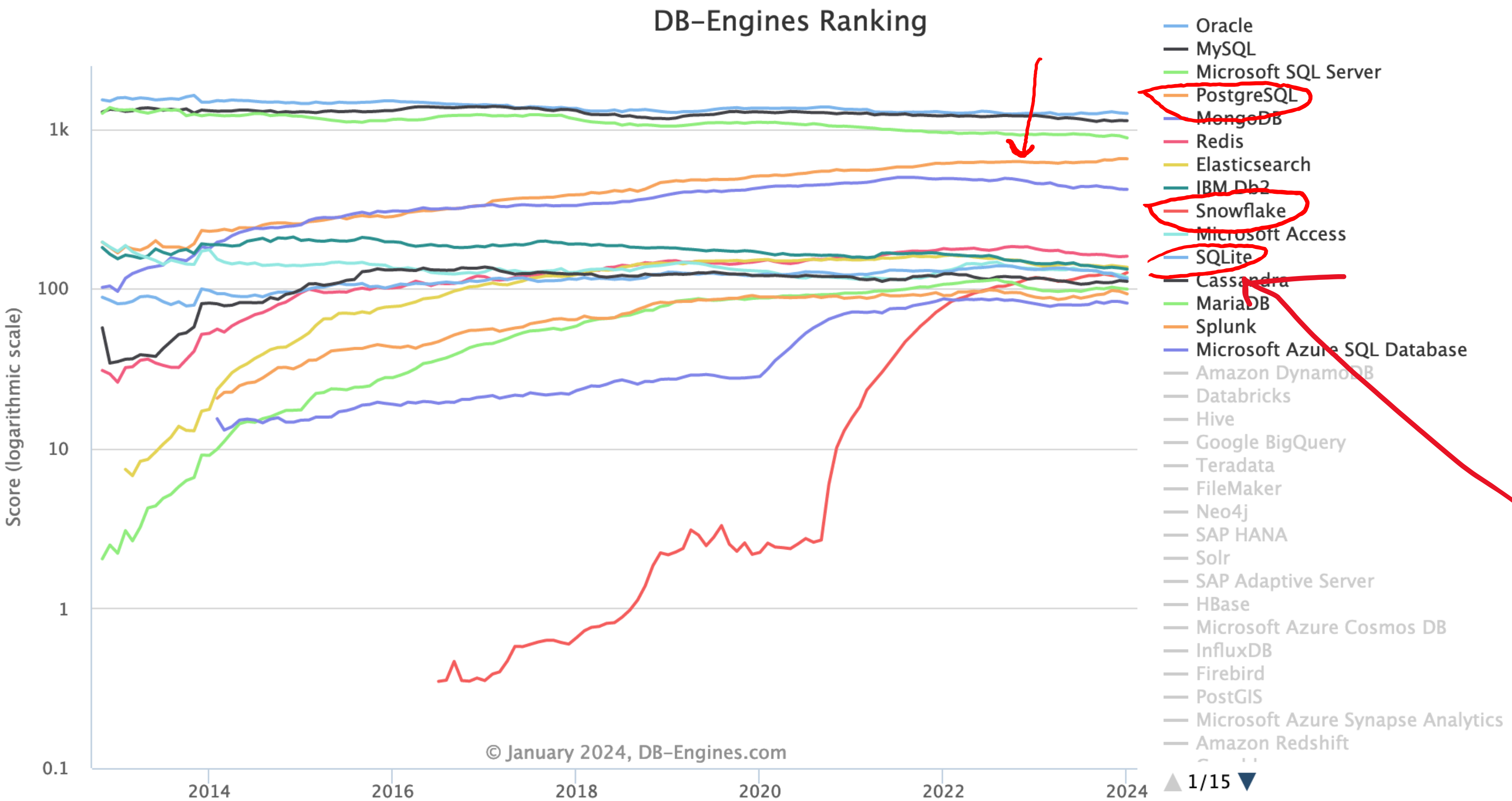
Source: http://en.wikipedia.org/wiki/QUEL_query_languages

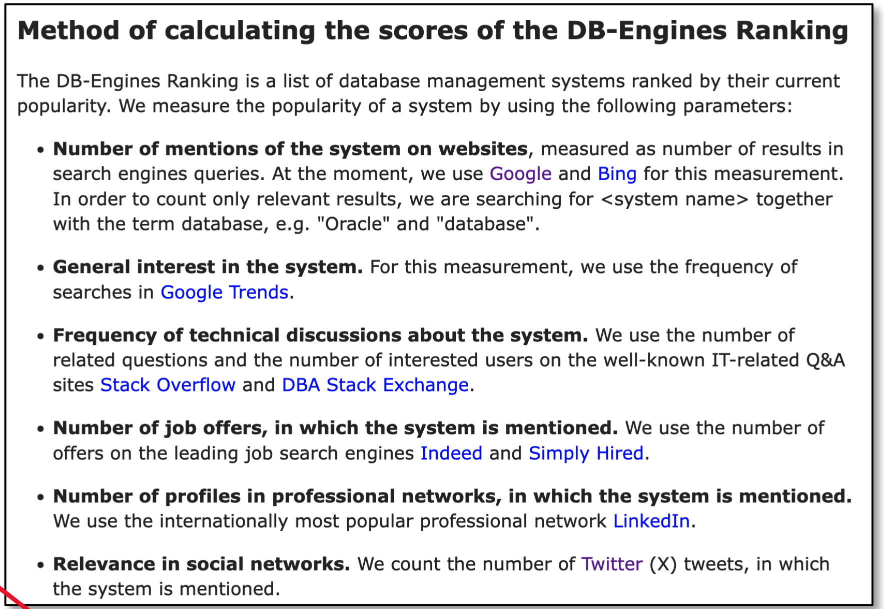
Why PostgreSQL instead of MariaDB (or MySQL)

PostgreSQL

MariaDB Foundation

MySQL

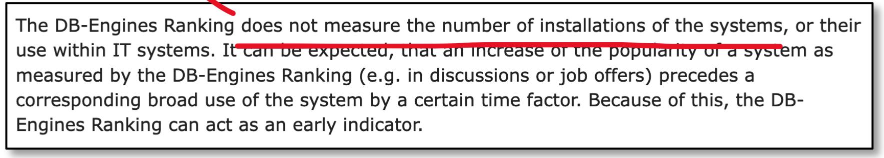




Method of calculating the scores of the DB-Engines Ranking

The DB-Engines Ranking is a list of database management systems ranked by their current popularity. We measure the popularity of a system by using the following parameters:

- **Number of mentions of the system on websites**, measured as number of results in search engines queries. At the moment, we use Google and Bing for this measurement. In order to count only relevant results, we are searching for <system name> together with the term database, e.g. "Oracle" and "database".
- **General interest in the system.** For this measurement, we use the frequency of searches in Google Trends.
- **Frequency of technical discussions about the system.** We use the number of related questions and the number of interested users on the well-known IT-related Q&A sites Stack Overflow and DBA Stack Exchange.
- **Number of job offers, in which the system is mentioned.** We use the number of offers on the leading job search engines Indeed and Simply Hired.
- **Number of profiles in professional networks, in which the system is mentioned.** We use the internationally most popular professional network LinkedIn.
- **Relevance in social networks.** We count the number of Twitter (X) tweets, in which the system is mentioned.

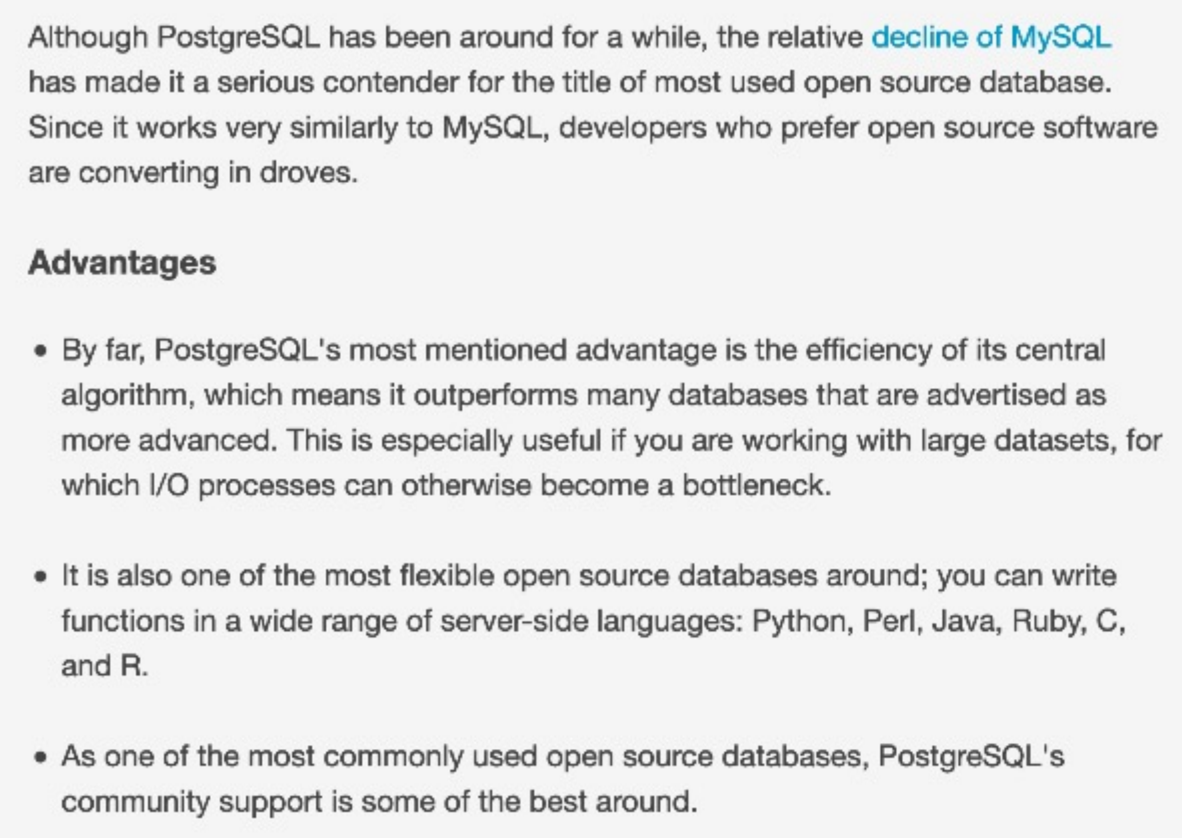


The DB-Engines Ranking does not measure the number of installations of the systems, or their use within IT systems. It ~~can be expected, that an increase of the popularity of a system~~ as measured by the DB-Engines Ranking (e.g. in discussions or job offers) precedes a corresponding broad use of the system by a certain time factor. Because of this, the DB-Engines Ranking can act as an early indicator.

SQLite likely has the most number of installations: its is an embedded serverless database (not a server-client databas)

Source: https://db-engines.com/en/ranking_trend

Why PostgreSQL instead of MariaDB (or MySQL)

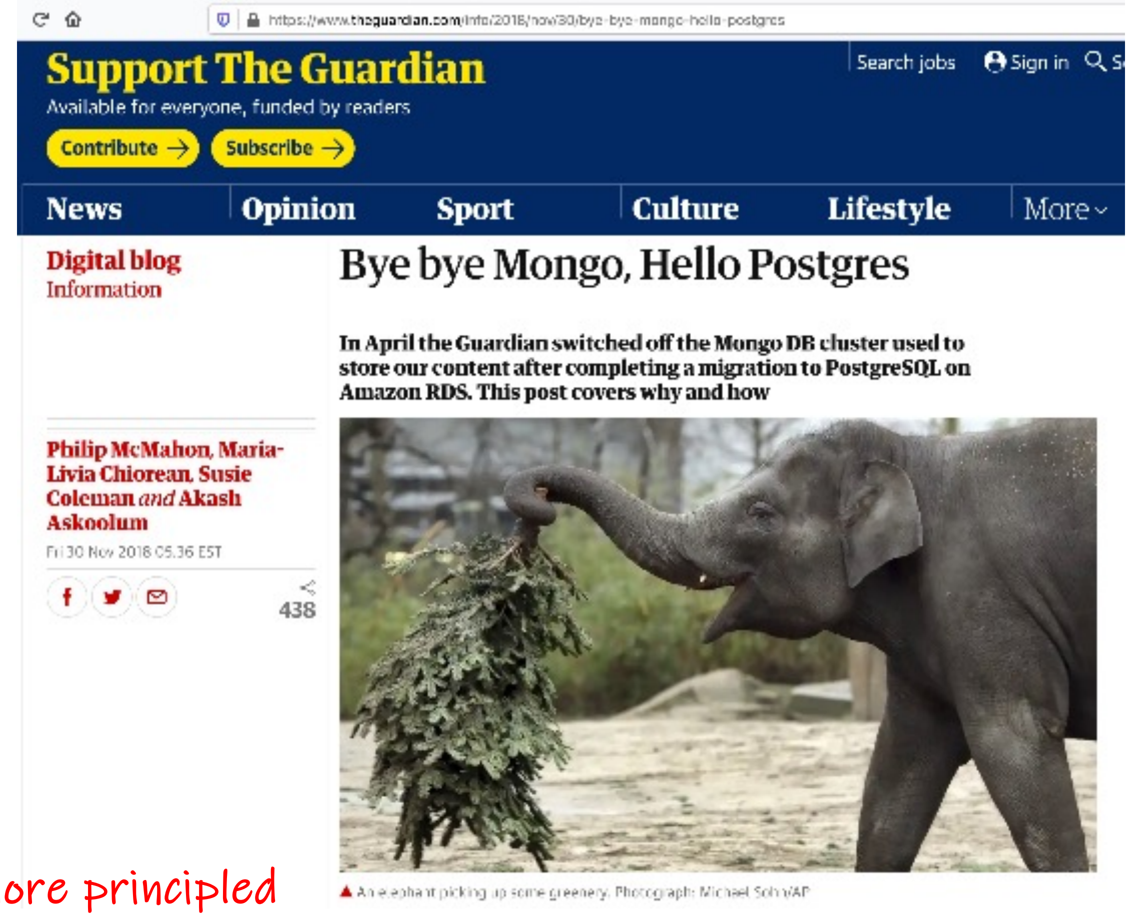


Although PostgreSQL has been around for a while, the relative decline of MySQL has made it a serious contender for the title of most used open source database. Since it works very similarly to MySQL, developers who prefer open source software are converting in droves.

Advantages

- By far, PostgreSQL's most mentioned advantage is the efficiency of its central algorithm, which means it outperforms many databases that are advertised as more advanced. This is especially useful if you are working with large datasets, for which I/O processes can otherwise become a bottleneck.
- It is also one of the most flexible open source databases around; you can write functions in a wide range of server-side languages: Python, Perl, Java, Ruby, C, and R.
- As one of the most commonly used open source databases, PostgreSQL's community support is some of the best around.

Source: https://db-engines.com/en/ranking_trend



Bye bye Mongo, Hello Postgres

In April the Guardian switched off the Mongo DB cluster used to store our content after completing a migration to PostgreSQL on Amazon RDS. This post covers why and how

Philip McMahon, Maria-Livia Chiorean, Susie Coleman *and* Akash Askoolum

Fri 30 Nov 2018 05.36 EST

An elephant picking up some greenery. Photograph: Michael Sohn/AP

Source: https://www.theguardian.com/info/2018/nov/30/bye-bye-mongo-hello-postgres

I also prefer PostgreSQL over MySQL because it has a more principled interpretation of SQL (and a powerful EXPLAIN command)

Simple SQL Query

Our friend here shows that you can follow along in Postgres.
Just install the database from the text file "302 - ..."
available in our sql folder from our course web page

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT pName, price
FROM   Product
WHERE  price > 100
```

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Simple SQL Query

Our friend here shows that you can follow along in Postgres. Just install the database from the text file "302 - ..." available in our sql folder from our course web page


302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
3  SELECT  pName, price
1  FROM    Product
2  WHERE   price > 100
```

PName	Price
SingleTouch	$149.99
MultiTouch	$203.99

Selection
& Projection

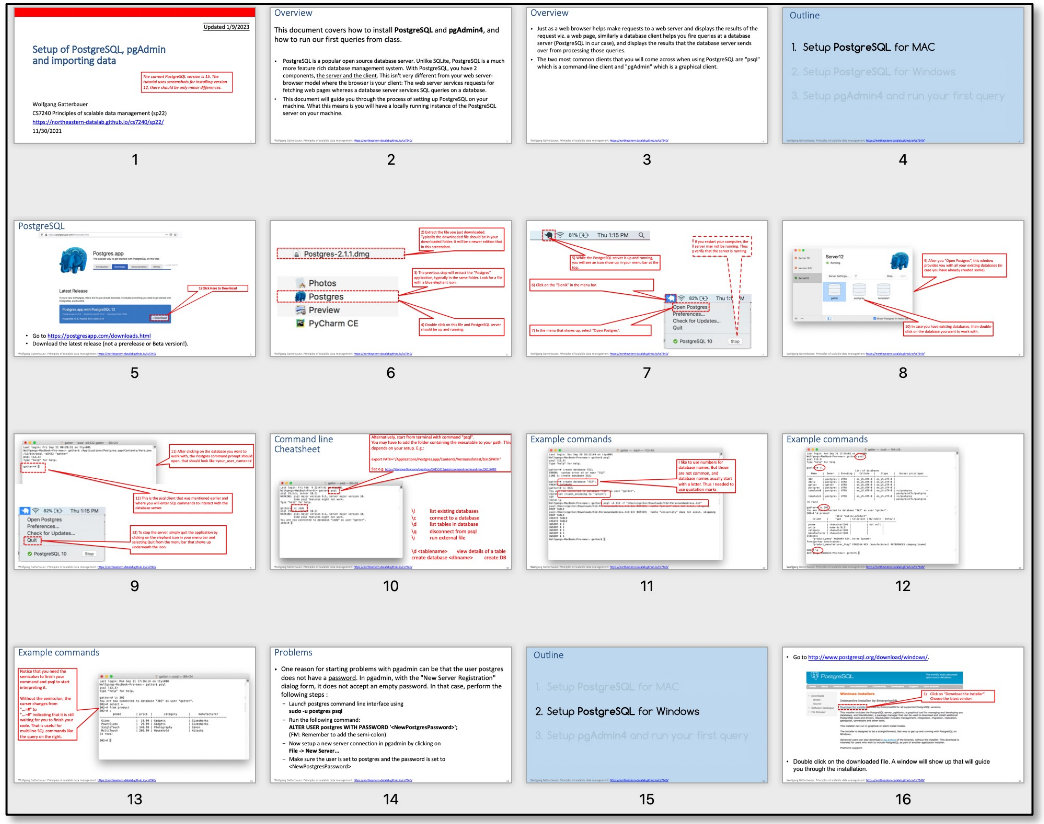
SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

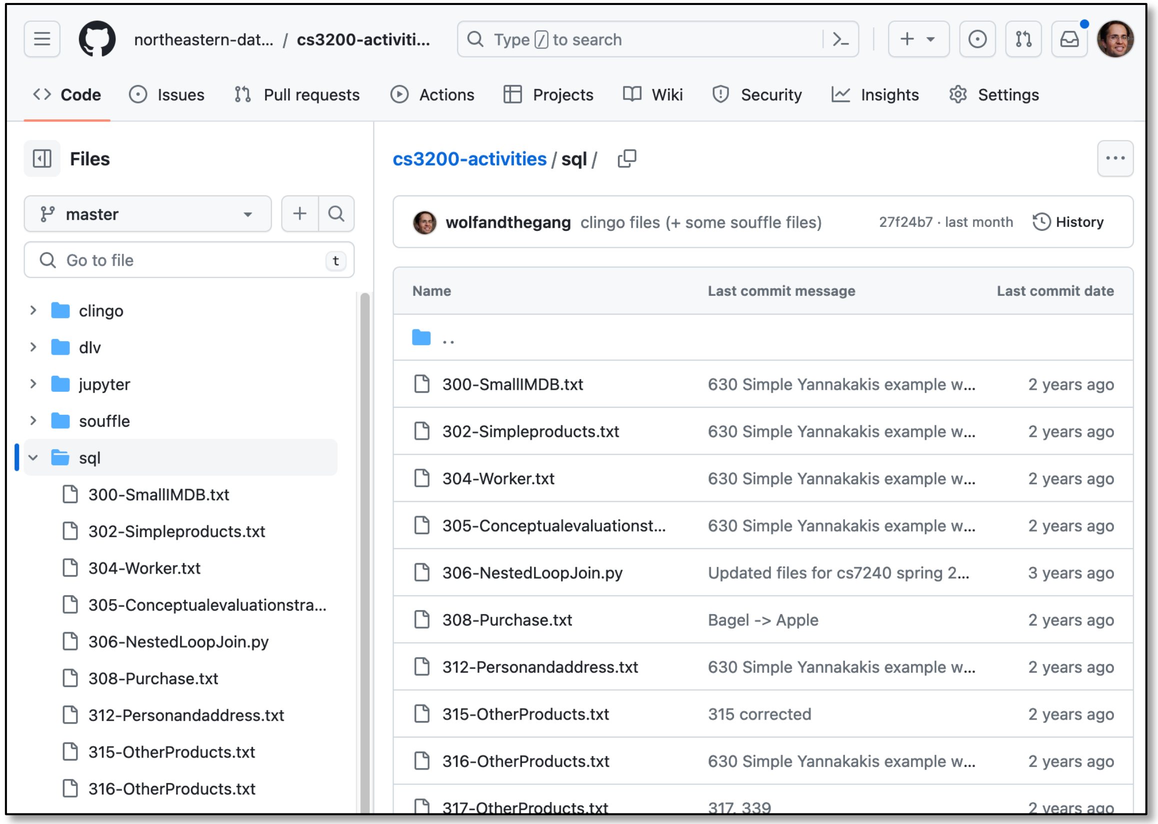
How to install PostgreSQL?

As always: if you find something that does not work, PLEASE let me know to fix it!

Topic 1: Data Models and Query Languages

- **Lecture 1 (Tue 1/19):** Course introduction / T1-U1 SQL / PostgreSQL setup / SQL Activities





cs3200-activities / sql /

wolfandthegang clingo files (+ some souffle files) 27f24b7 · last month

- clingo
- dlv
- jupyter
- souffle
- sql
  - 300-SmallIMDB.txt
  - 302-Simpleproducts.txt
  - 304-Worker.txt
  - 305-Conceptualevaluationstra...
  - 306-NestedLoopJoin.py
  - 308-Purchase.txt
  - 312-Personandaddress.txt
  - 315-OtherProducts.txt
  - 316-OtherProducts.txt

Name	Last commit message	Last commit date
..		
300-SmallIMDB.txt	630 Simple Yannakakis example w...	2 years ago
302-Simpleproducts.txt	630 Simple Yannakakis example w...	2 years ago
304-Worker.txt	630 Simple Yannakakis example w...	2 years ago
305-Conceptualevaluationst...	630 Simple Yannakakis example w...	2 years ago
306-NestedLoopJoin.py	Updated files for cs7240 spring 2...	3 years ago
308-Purchase.txt	Bagel -> Apple	2 years ago
312-Personandaddress.txt	630 Simple Yannakakis example w...	2 years ago
315-OtherProducts.txt	315 corrected	2 years ago
316-OtherProducts.txt	630 Simple Yannakakis example w...	2 years ago
317-OtherProducts.txt	317, 339	2 years ago

Selection vs. Projection

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT pName, price
FROM   Product
WHERE  price > 100
```

Where does the selection happen?

?

PName	Price
SingleTouch	$149.99
MultiTouch	$203.99

Selection
& Projection

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Selection vs. Projection

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

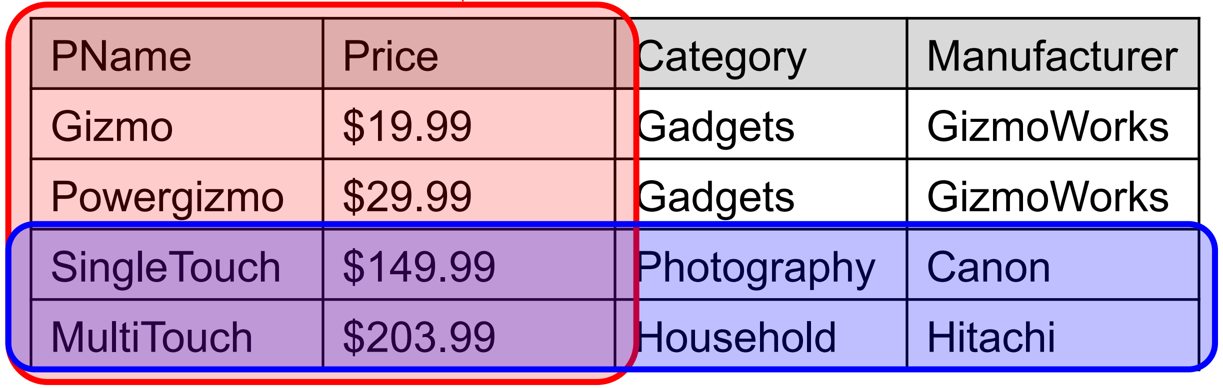
```
SELECT pName, price
FROM   Product
WHERE  price > 100
```

PName	Price
SingleTouch	$149.99
MultiTouch	$203.99

One **selects** certain entires=tuples (rows)
-> happens in the **WHERE** clause
-> acts like a **filter**

Selection vs. Projection

Product



PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

One **projects** onto some attributes (columns)
-> happens in the **SELECT** clause

One **selects** certain entires=tuples (rows)
-> happens in the **WHERE** clause
-> acts like a **filter**

```
SELECT pName, price
FROM   Product
WHERE  price > 100
```

PName	Price
SingleTouch	$149.99
MultiTouch	$203.99

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Eliminating Duplicates

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT category
FROM   Product
```

⇨ ?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Eliminating Duplicates

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Set vs. Bag semantics

```
SELECT category
FROM   Product
```

Category
Gadgets
Gadgets
Photography
Household

Think of a dictionary: keys mapping to # of occurences

Gadgets : 2
Photography :
Houshold : 1

?

Category
Gadgets
Photography
Household

underlying set also called the "support" of the bag

Eliminating Duplicates

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Set vs. Bag semantics

```
SELECT category
FROM   Product
```

Category
Gadgets
Gadgets
Photography
Household

Think of a dictionary: keys mapping to # of occurences

Gadgets : 2
Photography :
Houshold : 1

```
SELECT DISTINCT category
FROM   Product
```

Category
Gadgets
Photography
Household

underlying set also called the "support" of the bag

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Outline: T1-U1: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Union and Theta Joins
  - Nulls & Outer joins
  - Top-k
  - [Recursion: moved to T1-U4: Datalog]

Keys and Foreign Keys

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

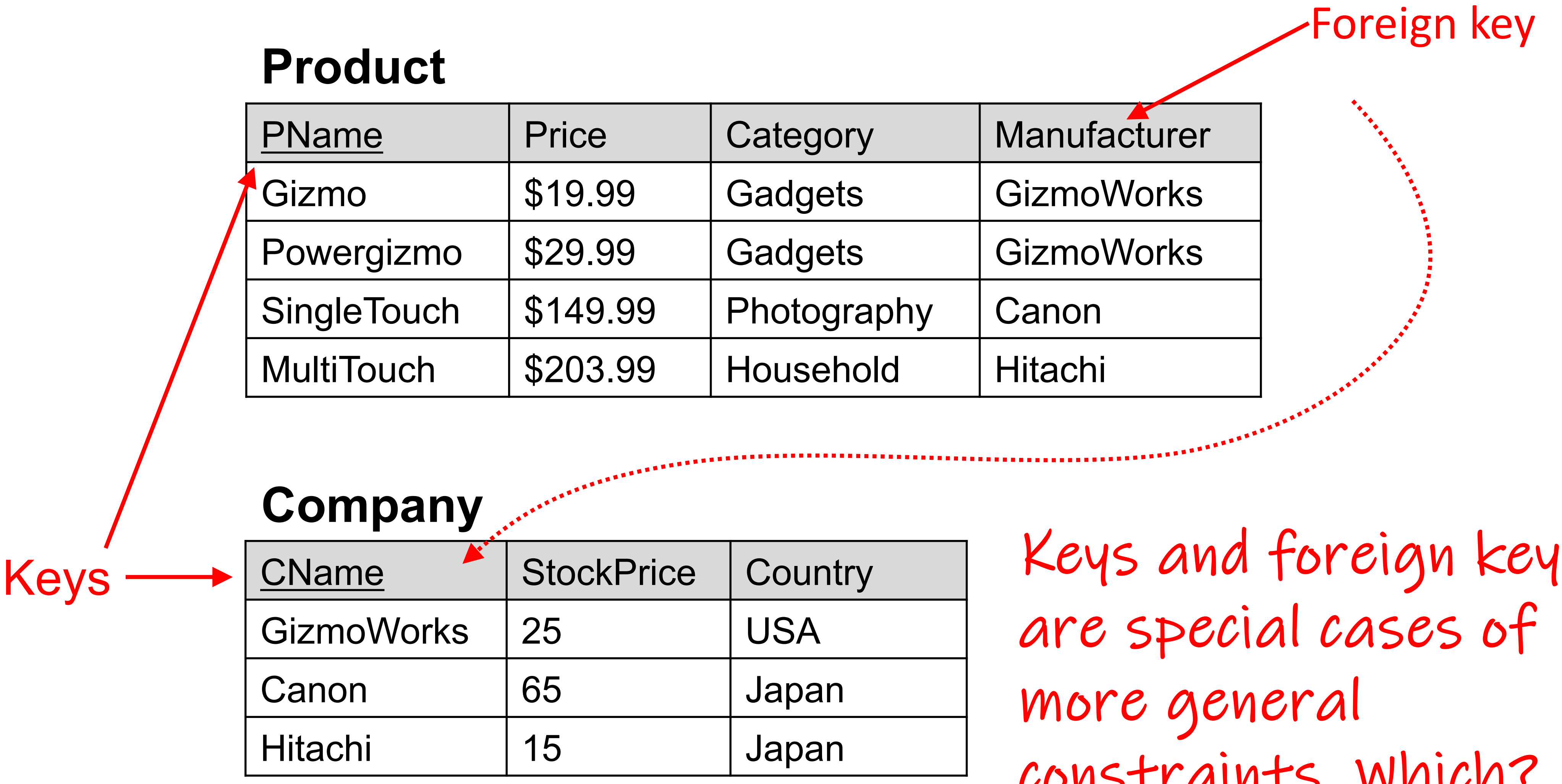
What is here
a key vs.
a foreign key?

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Keys and Foreign Keys

302



Foreign key

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Keys

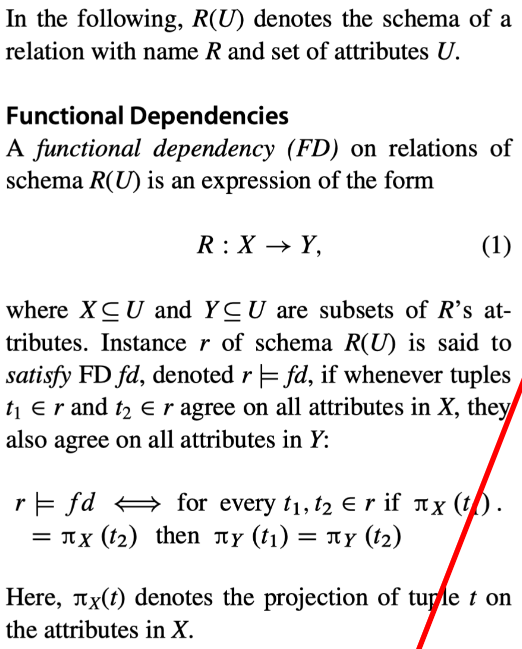
Keys and foreign keys are special cases of more general constraints. Which?

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Keys and Foreign Keys



In the following, $R(U)$ denotes the schema of a relation with name R and set of attributes U.

Functional Dependencies

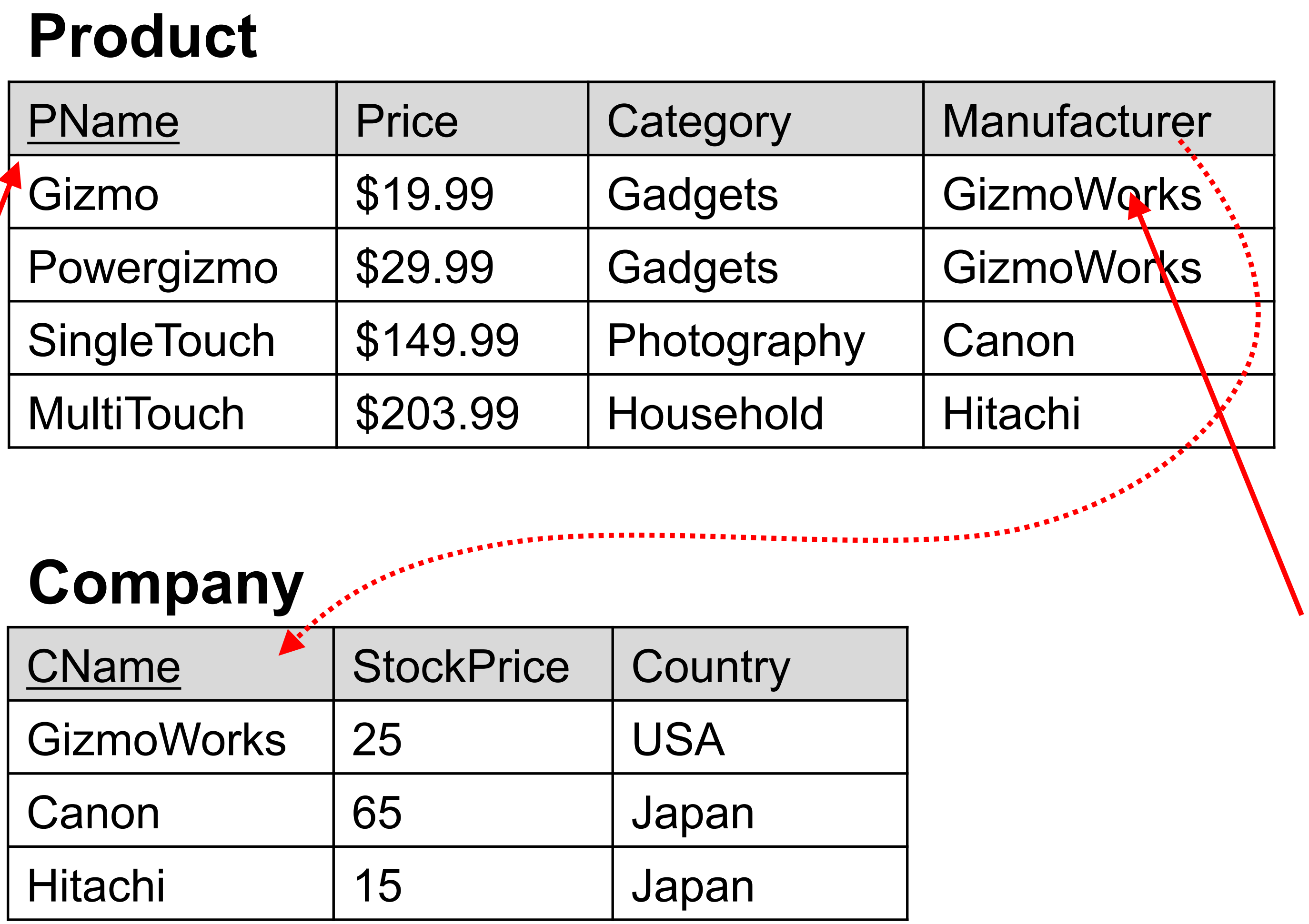
A *functional dependency (FD)* on relations of schema $R(U)$ is an expression of the form

$$R : X \to Y, \qquad (1)$$

where $X \subseteq U$ and $Y \subseteq U$ are subsets of R's attributes. Instance r of schema $R(U)$ is said to *satisfy* FD *fd*, denoted $r \models fd$, if whenever tuples $t_1 \in r$ and $t_2 \in r$ agree on all attributes in X, they also agree on all attributes in Y:

$$r \models fd \iff \text{for every } t_1, t_2 \in r \text{ if } \pi_X(t_1) = \pi_X(t_2) \text{ then } \pi_Y(t_1) = \pi_Y(t_2)$$

Here, $\pi_X(t)$ denotes the projection of tuple t on the attributes in X.



Product

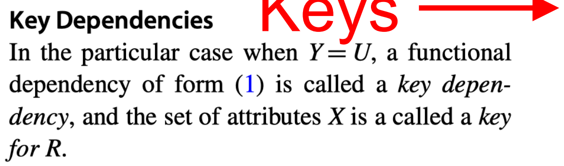
<u>PName</u>	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

<u>CName</u>	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

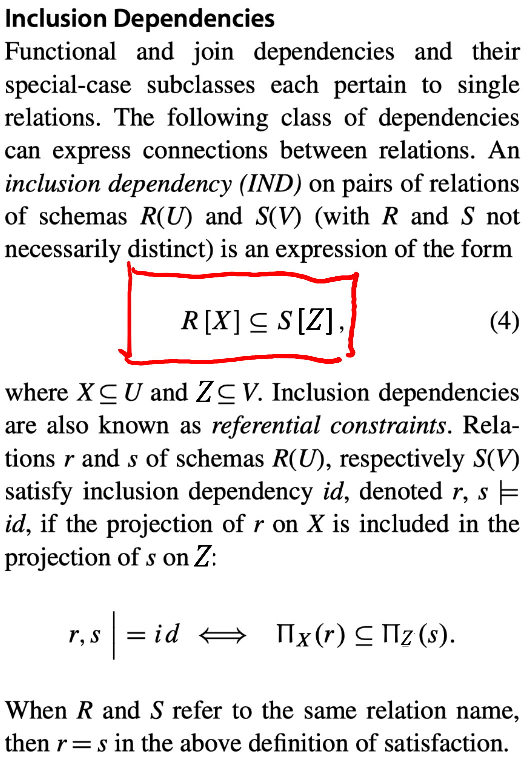
Keys

Foreign key



Key Dependencies

In the particular case when $Y = U$, a functional dependency of form (1) is called a *key dependency*, and the set of attributes X is a called a *key for R*.



Inclusion Dependencies

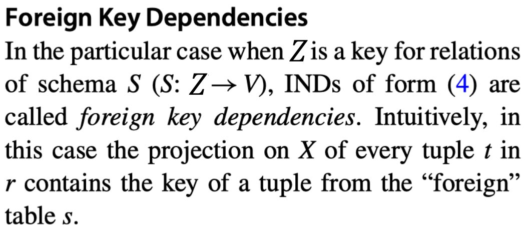
Functional and join dependencies and their special-case subclasses each pertain to single relations. The following class of dependencies can express connections between relations. An *inclusion dependency (IND)* on pairs of relations of schemas $R(U)$ and $S(V)$ (with R and S not necessarily distinct) is an expression of the form

$$\boxed{R[X] \subseteq S[Z],} \qquad (4)$$

where $X \subseteq U$ and $Z \subseteq V$. Inclusion dependencies are also known as *referential constraints*. Relations r and s of schemas $R(U)$, respectively $S(V)$ satisfy inclusion dependency *id*, denoted $r, s \models id$, if the projection of r on X is included in the projection of s on Z:

$$r, s \models id \iff \Pi_X(r) \subseteq \Pi_Z(s).$$

When R and S refer to the same relation name, then $r = s$ in the above definition of satisfaction.

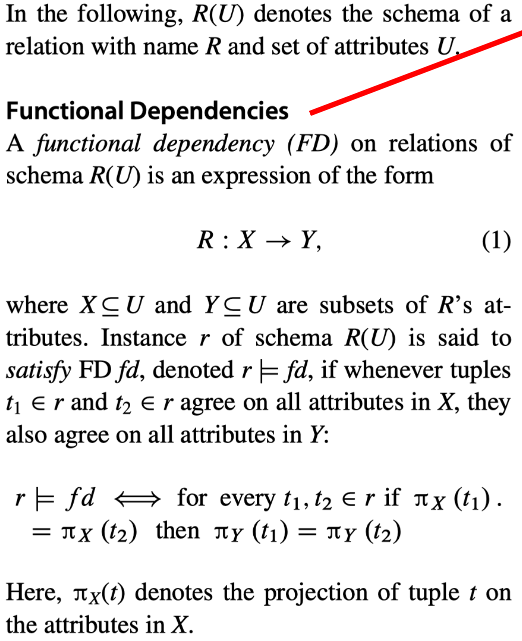


Foreign Key Dependencies

In the particular case when Z is a key for relations of schema S (S: $Z \to V$), INDs of form (4) are called *foreign key dependencies*. Intuitively, in this case the projection on X of every tuple t in r contains the key of a tuple from the "foreign" table s.

Screenshots from: Deutsch. FOL Modeling of Integrity Constraints (Dependencies). Encyclopedia of Database Systems, 2018. https://doi.org/10.1007/978-1-4614-8265-9_980
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Keys and Foreign Keys



In the following, $R(U)$ denotes the schema of a relation with name R and set of attributes U.

Functional Dependencies

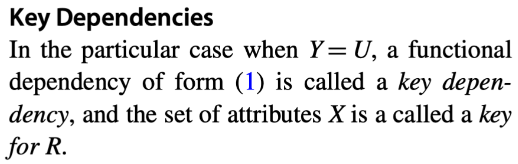
A *functional dependency (FD)* on relations of schema $R(U)$ is an expression of the form

$$R : X \to Y, \qquad (1)$$

where $X \subseteq U$ and $Y \subseteq U$ are subsets of R's attributes. Instance r of schema $R(U)$ is said to *satisfy* FD *fd*, denoted $r \models fd$, if whenever tuples $t_1 \in r$ and $t_2 \in r$ agree on all attributes in X, they also agree on all attributes in Y:

$$r \models fd \iff \text{for every } t_1, t_2 \in r \text{ if } \pi_X(t_1) . = \pi_X(t_2) \text{ then } \pi_Y(t_1) = \pi_Y(t_2)$$

Here, $\pi_X(t)$ denotes the projection of tuple t on the attributes in X.



Key Dependencies

In the particular case when $Y = U$, a functional dependency of form (1) is called a *key dependency*, and the set of attributes X is a called a *key for R*.

R[X] functionally determines R[Y]:
Y = f(X)

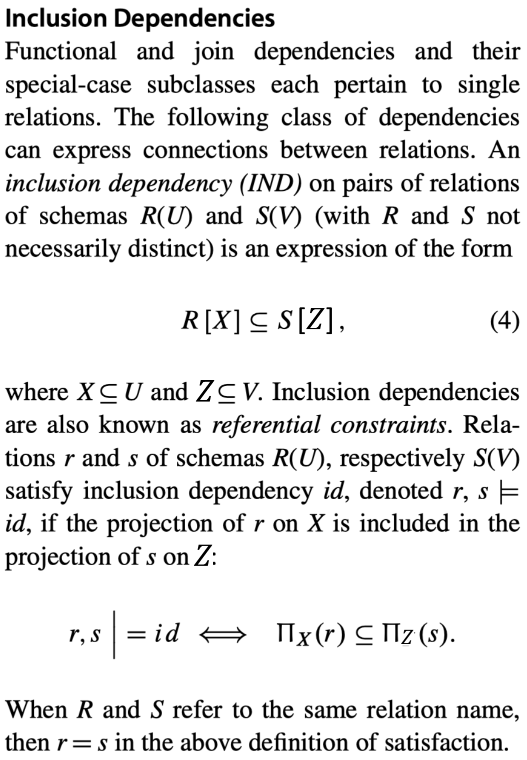
R

...	X	Y	...
...	1	7	...
...	1	7	...
...	2	5	...
...	3	7	...

R[X] is included in S[Z]:
R[X] ⊆ S[Z]

S

...	Z	...
...	1	...
...	2	...
...	2	...
...	3	...
...	4	...



Inclusion Dependencies

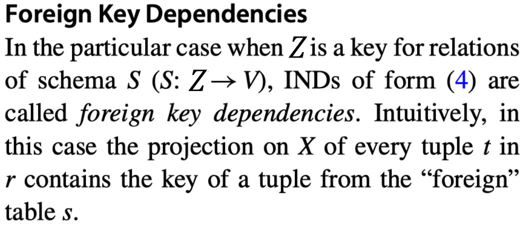
Functional and join dependencies and their special-case subclasses each pertain to single relations. The following class of dependencies can express connections between relations. An *inclusion dependency (IND)* on pairs of relations of schemas $R(U)$ and $S(V)$ (with R and S not necessarily distinct) is an expression of the form

$$R[X] \subseteq S[Z], \qquad (4)$$

where $X \subseteq U$ and $Z \subseteq V$. Inclusion dependencies are also known as *referential constraints*. Relations r and s of schemas $R(U)$, respectively $S(V)$ satisfy inclusion dependency *id*, denoted $r, s \models id$, if the projection of r on X is included in the projection of s on Z:

$$r, s \,\Big| = id \iff \Pi_X(r) \subseteq \Pi_Z(s).$$

When R and S refer to the same relation name, then $r = s$ in the above definition of satisfaction.



Foreign Key Dependencies

In the particular case when Z is a key for relations of schema S (S: $Z \to V$), INDs of form (4) are called *foreign key dependencies*. Intuitively, in this case the projection on X of every tuple t in r contains the key of a tuple from the "foreign" table s.

Screenshots from: Deutsch. FOL Modeling of Integrity Constraints (Dependencies). Encyclopedia of Database Systems, 2018. https://doi.org/10.1007/978-1-4614-8265-9_980
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Updated 1/12/2024

Topic 1: Data models and query languages
Unit 1: SQL (continued)
Lecture 2

Wolfgang Gatterbauer

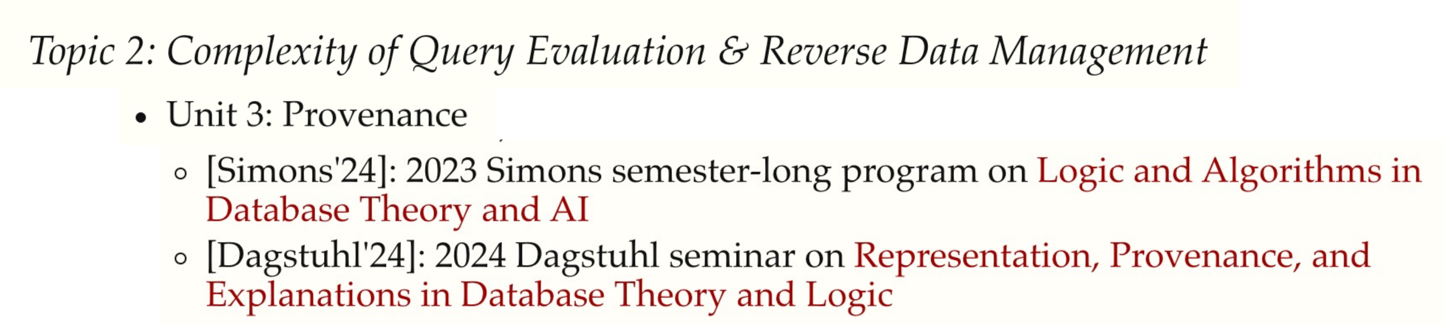
CS7240 Principles of scalable data management (sp24)

https://northeastern-datalab.github.io/cs7240/sp24/

1/12/2024

Pre-class conversations

- Last class summary
- Class procedures based on past suggestions:
  - Your experience Canvas vs Piazza?
  - Example past scribe posted on Canvas; Secondary posting of class scribes to Piazza (optionally anonymous). I comment on both Canvas and Piazza
  - Already installed Postgres?
  - A downside of HWs with self-determined deadlines: you are in charge
  - Next week TUE online, no class FRI; see links for current research
- Today:
  - SQL continued



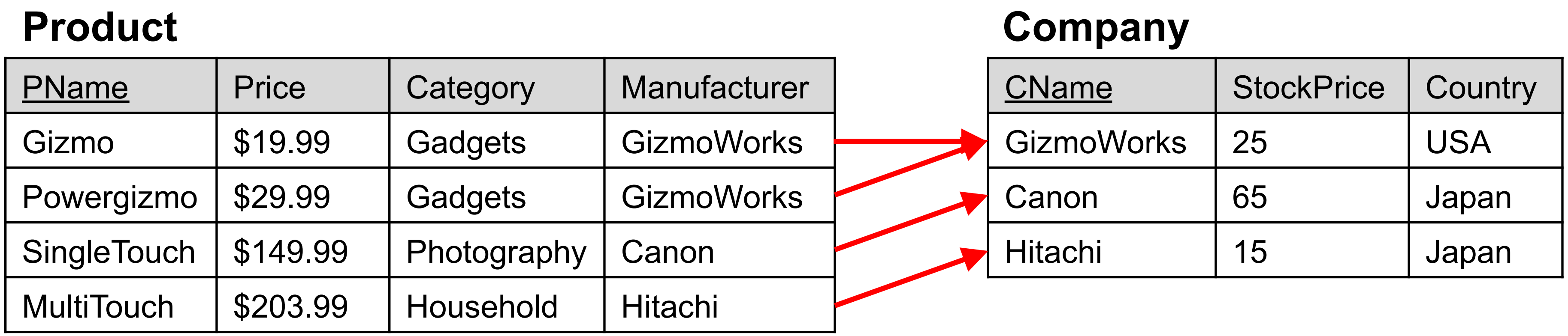
Topic 2: Complexity of Query Evaluation & Reverse Data Management

- Unit 3: Provenance
  - [Simons'24]: 2023 Simons semester-long program on Logic and Algorithms in Database Theory and AI
  - [Dagstuhl'24]: 2024 Dagstuhl seminar on Representation, Provenance, and Explanations in Database Theory and Logic

Outline: T1-U1: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Union and Theta Joins
  - Nulls & Outer joins
  - Top-k
  - [Recursion: moved to T1-U4: Datalog]

Referential Integrity



Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

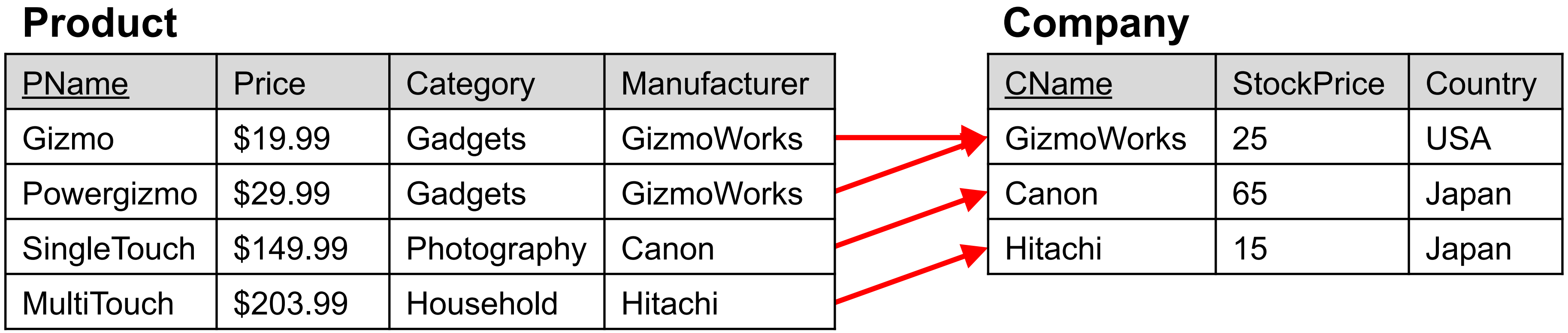
CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Key constraint: minimal subset of the attributes of a relation is a unique identifier for a tuple.

Foreign key: attribute in a relational table that matches a candidate key of another table

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Referential Integrity



Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Key constraint: minimal subset of the attributes of a relation is a unique identifier for a tuple.

```
Insert into Product values ('Gizmo', 14.99, 'Gadgets', 'Hitachi');
```

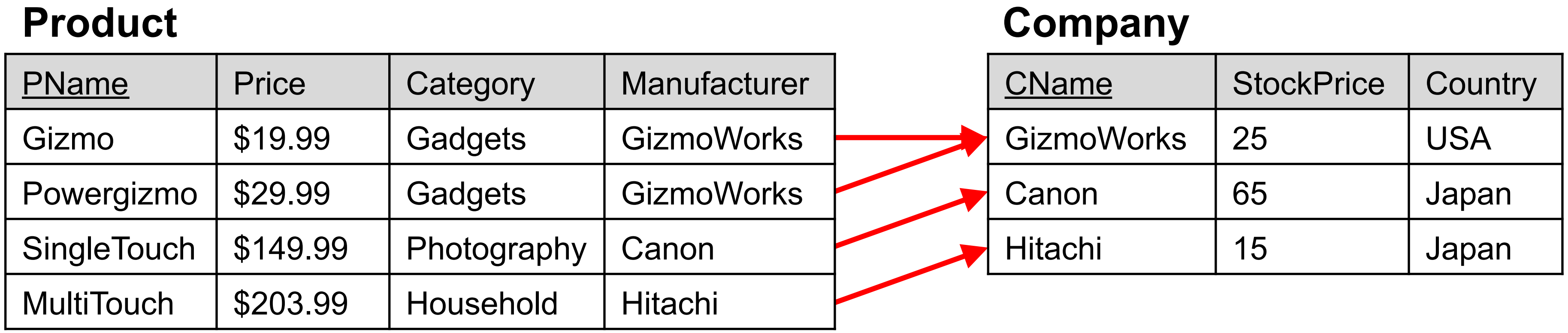
Gizmo	$14.99	Gadgets	Hitachi

?

Foreign key: attribute in a relational table that matches a candidate key of another table

Referential Integrity

302



Product

<u>PName</u>	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

<u>CName</u>	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Key constraint: minimal subset of the attributes of a relation is a unique identifier for a tuple.

tuple violates key constraint

```
Insert into Product values ('Gizmo', 14.99, 'Gadgets', 'Hitachi');
```

Gizmo	$14.99	Gadgets	Hitachi

Foreign key: attribute in a relational table that matches a candidate key of another table

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Referential Integrity

302

Product

<u>PName</u>	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

<u>CName</u>	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Key constraint: minimal subset of the attributes of a relation is a unique identifier for a tuple.

```
Insert into Product values ('Gizmo', 14.99, 'Gadgets', 'Hitachi');
```

tuple violates key constraint

Gizmo	$14.99	Gadgets	Hitachi

Foreign key: attribute in a relational table that matches a candidate key of another table

```
Insert into Product values ('SuperTouch', 249.99, 'Computer', 'NewCom');
```

SuperTouch	$249.99	Computer	NewCom

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Referential Integrity

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Key constraint: minimal subset of the attributes of a relation is a unique identifier for a tuple.

tuple violates key constraint

```
Insert into Product values ('Gizmo', 14.99, 'Gadgets', 'Hitachi');
```

Gizmo	$14.99	Gadgets	Hitachi

Foreign key: attribute in a relational table that matches a candidate key of another table

tuple violates foreign key constraint

```
Insert into Product values ('SuperTouch', 249.99, 'Computer', 'NewCom');
```

SuperTouch	$249.99	Computer	NewCom

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Referential Integrity

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
~~Canon~~	~~65~~	~~Japan~~
Hitachi	15	Japan

Key constraint: minimal subset of the attributes of a relation is a unique identifier for a tuple.

tuple violates key constraint

`Insert into Product values ('Gizmo', 14.99, 'Gadgets', 'Hitachi');`

Gizmo	$14.99	Gadgets	Hitachi

Foreign key: attribute in a relational table that matches a candidate key of another table

tuple violates foreign key constraint

`Insert into Product values ('SuperTouch', 249.99, 'Computer', 'NewCom');`

SuperTouch	$249.99	Computer	NewCom

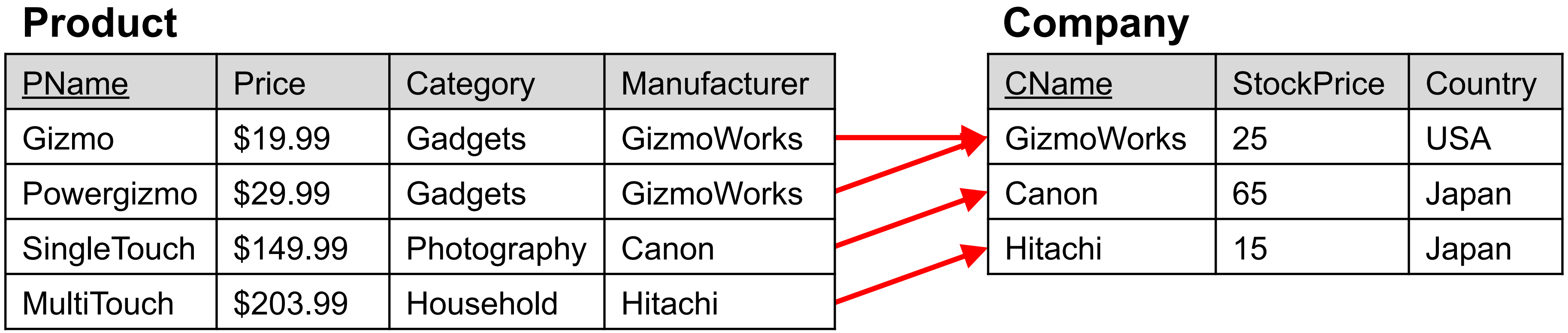
```
Delete from Company
where CName = 'Canon';
```

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Referential Integrity



Product

<u>PName</u>	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

<u>CName</u>	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Key constraint: minimal subset of the attributes of a relation is a unique identifier for a tuple.

tuple violates key constraint

```
Insert into Product values ('Gizmo', 14.99, 'Gadgets', 'Hitachi');
```

Gizmo	$14.99	Gadgets	Hitachi

Foreign key: attribute in a relational table that matches a candidate key of another table

tuple violates foreign key constraint

```
Insert into Product values ('SuperTouch', 249.99, 'Computer', 'NewCom');
```

SuperTouch	$249.99	Computer	NewCom

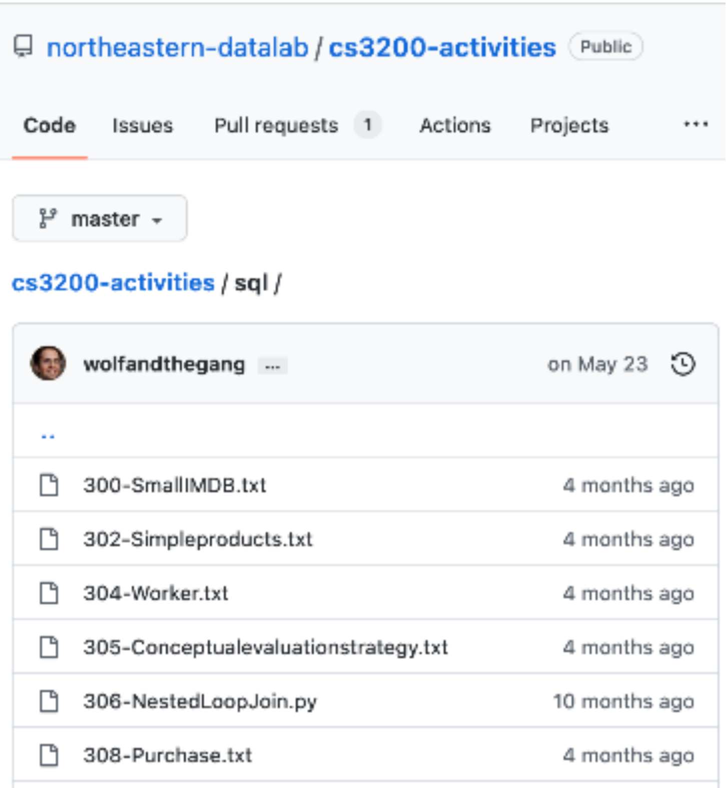
```
Delete from Company
where CName = 'Canon';
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Schema specification in SQL

302



northeastern-datalab / **cs3200-activities** Public

cs3200-activities / sql /

wolfandthegang ... on May 23

..

300-SmallIMDB.txt	4 months ago
302-Simpleproducts.txt	4 months ago
304-Worker.txt	4 months ago
305-Conceptualevaluationstrategy.txt	4 months ago
306-NestedLoopJoin.py	10 months ago
308-Purchase.txt	4 months ago

```
---------------------------
-- Create the tables
---------------------------

create table Company (
      CName char(20) PRIMARY KEY,
      StockPrice int,
      Country char(20) );

create table Product (
      PName char(20),
      Price decimal(9, 2),
      Category char(20),
      Manufacturer char(20),
PRIMARY KEY (PName),
FOREIGN KEY (Manufacturer) REFERENCES Company(CName) );



---------------------------
-- Populate the tables
---------------------------

insert into Company values ('GizmoWorks', 25, 'USA');
insert into Company values ('Canon', 65, 'Japan');
insert into Company values ('Hitachi', 15, 'Japan');

insert into Product values ('Gizmo', 19.99, 'Gadgets', 'GizmoWorks');
insert into Product values ('PowerGizmo', 29.99, 'Gadgets', 'GizmoWorks');
```

Example SQL code and database: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Outline: T1-U1: SQL

- SQL
    - Schema, keys, referential integrity
    - Joins
    - Aggregates and grouping
    - Nested queries (Subqueries)
    - Union and Theta Joins
    - Nulls & Outer joins
    - Top-k
    - [Recursion: moved to T1-U4: Datalog]

Joins

302

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Q: Find all products under $200 manufactured in Japan; return their names and prices!

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Joins

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Q: Find all products under $200 manufactured in Japan; return their names and prices!

```
SELECT  pName, price
FROM    Product, Company
WHERE   manufacturer=cName
    and country='Japan'
    and price <= 200
```

Join b/w Product and Company

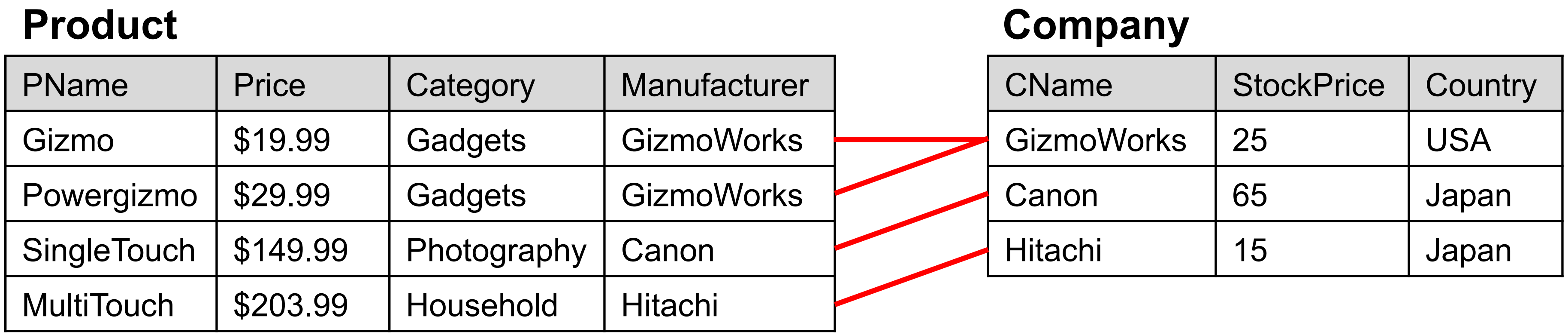
PName	Price
SingleTouch	$149.99

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Joins

302

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)



Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT  *
FROM    Product, Company
WHERE   manufacturer=cName
```

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Joins

302

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

PName	Price	Category	Manufacturer	CName	StockPrice	Country
Gizmo	$19.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
Powergizmo	$29.99	Gadgets	GizmoWorks	GizmoWorks	25	USA
SingleTouch	$149.99	Photography	Canon	Canon	65	Japan
MultiTouch	$203.99	Household	Hitachi	Hitachi	15	Japan

```
SELECT  *
FROM    Product, Company
WHERE   manufacturer=cName
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

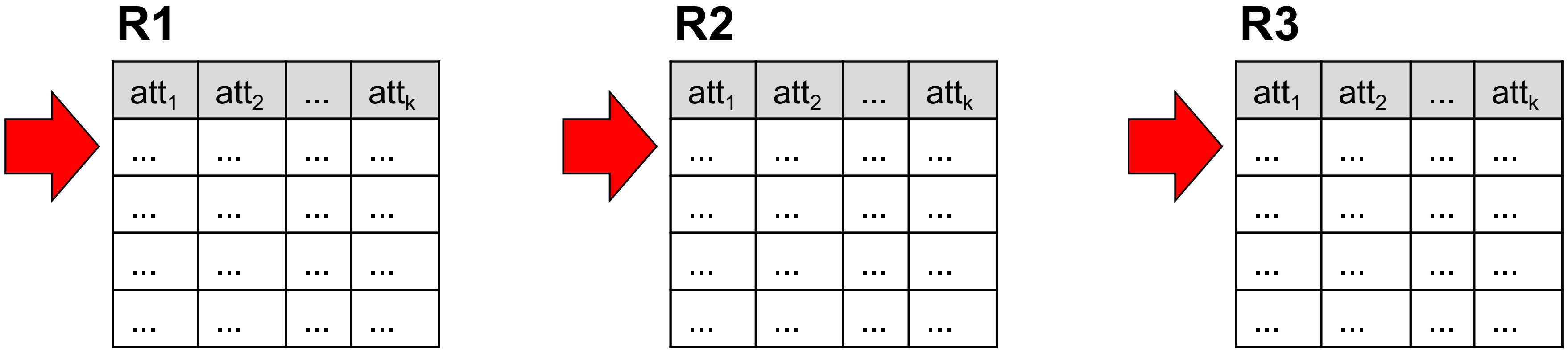
Meaning (Semantics) of SELECT-FROM-WHERE queries

```
3  SELECT  $a_1, a_2, \ldots, a_k$
1  FROM    $R_1$ as $x_1$, $R_2$ as $x_2$, …, $R_n$ as $x_n$
2  WHERE   Conditions
```

Conceptual evaluation strategy (nested for loops):

```
Answer = {}
for $x_1$ in $R_1$ do
    for $x_2$ in $R_2$ do
        …..
            for $x_n$ in $R_n$ do
                if Conditions
                    then Answer = Answer $\cup$ {$(a_1,\ldots,a_k)$}
return Answer
```

Meaning (Semantics) of SELECT-FROM-WHERE queries



R1

att_1	att_2	…	att_k
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

R2

att_1	att_2	…	att_k
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

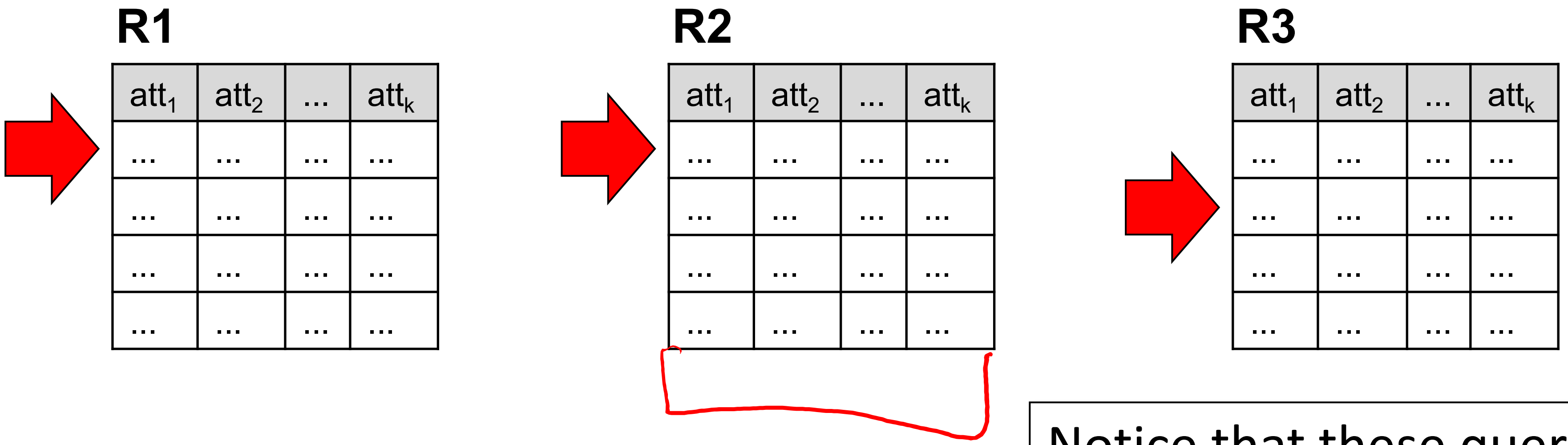
R3

att_1	att_2	…	att_k
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

Conceptual evaluation strategy (nested for loops):

```
Answer = {}
for $x_1$ in $R_1$ do
    for $x_2$ in $R_2$ do
        .....
            for $x_n$ in $R_n$ do
                if Conditions
                    then Answer = Answer $\cup$ {$(a_1,\ldots,a_k)$}
return Answer
```

Meaning (Semantics) of SELECT-FROM-WHERE queries



R1

att_1	att_2	…	att_k
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

R2

att_1	att_2	…	att_k
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

R3

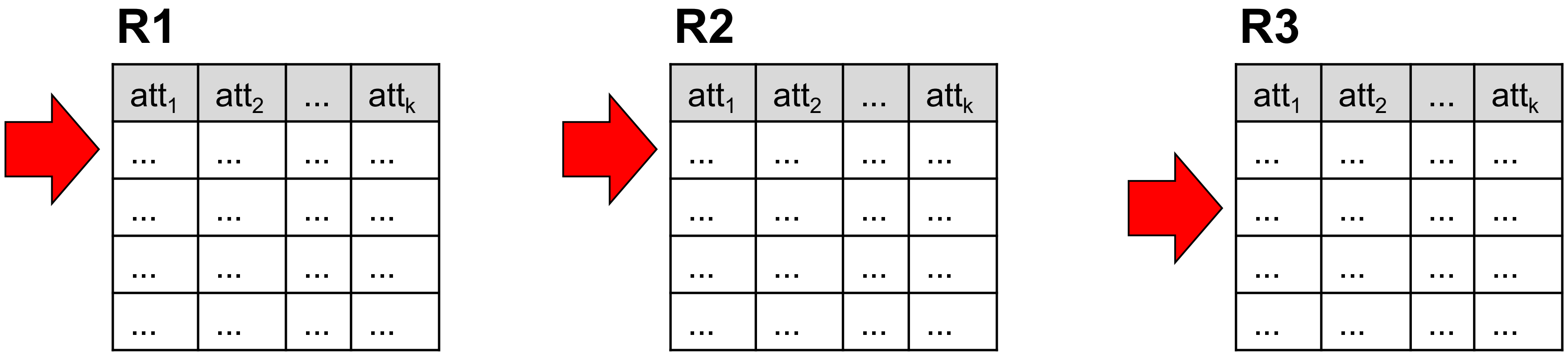
att_1	att_2	…	att_k
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

```
Answer = {}
for $x_1$ in $R_1$ do
    for $x_2$ in $R_2$ do
        .....
            for $x_n$ in $R_n$ do
                if Conditions
                    then Answer = Answer $\cup$ {($a_1$,…,$a_k$)}
return Answer
```

Notice that these queries are "monotone": whenever we add tuples to the input, the output can never decrease:

if $R_1 \subseteq R_1', R_2 \subseteq R_2', R_3 \subseteq R_3'$
then $Q(R_1, R_2, R_3) \subseteq Q(R_1', R_2', R_3')$

Meaning (Semantics) of conjunctive SQL Queries



R1

att$_1$	att$_2$	…	att$_k$
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

R2

att$_1$	att$_2$	…	att$_k$
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

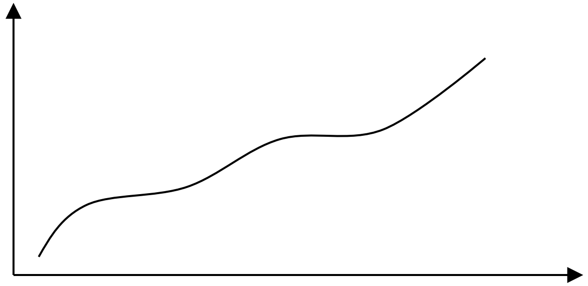
R3

att$_1$	att$_2$	…	att$_k$
…	…	…	…
…	…	…	…
…	…	…	…
…	…	…	…

DEFINITION: A function f(x) is "monotone" (or better "monotonically increasing") if:

if $x \leq y$

then $f(x) \leq f(y)$



Notice that these queries are "monotone": whenever we add tuples to the input, the output can never decrease:

if $R_1 \subseteq R_1', R_2 \subseteq R_2', R_3 \subseteq R_3'$

then $Q(R_1, R_2, R_3) \subseteq Q(R_1', R_2', R_3')$

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - **FROM**: Compute the cross-product of the relations. This is a new set of larger tuples.
  - **WHERE**: Only keep the tuples that pass the qualifications ("selection", filter)
  - **SELECT**: Delete attributes that are not in listed attributes
  - If **DISTINCT** is specified, eliminate duplicate rows.
- This strategy is probably the least efficient way to compute a query! An optimizer will find (algebraically equivalent but) more efficient strategies to compute the same answers.
- We say “semantics” not “execution order”. Why?

?

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - **FROM**: Compute the cross-product of the relations. This is a new set of larger tuples.
  - **WHERE**: Only keep the tuples that pass the qualifications ("selection", filter)
  - **SELECT**: Delete attributes that are not in listed attributes
  - If **DISTINCT** is specified, eliminate duplicate rows.
- This strategy is probably the least efficient way to compute a query! An optimizer will find (algebraically equivalent but) more efficient strategies to compute the same answers.
- We say "semantics" not "execution order". Why?
  - The preceding slides show what a join means (semantics = meaning): "the logic"
  - Not actually how the DBMS actually executes it (separation of concerns): algebra

Table Alias (Tuple Variables)

Person (pName, address, works_for)
University (uName, address)

```
SELECT  DISTINCT pName, address
FROM    Person, University
WHERE   works_for = uName
```

What will this query return ?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Table Alias (Tuple Variables)

Person (pName, address, works_for)
University (uName, address)

```
SELECT  DISTINCT pName, address
FROM    Person, University
WHERE   works_for = uName
```

which address?
Error!

```
SELECT  DISTINCT pName, University.address
FROM    Person, University
WHERE   Person.works_for = University.uName
```

```
SELECT  DISTINCT X.pName, Y.address
FROM    Person as X, University Y
WHERE   X.works_for = Y.uName
```

Notice that the use of "as" is not necessary, it is optional !!

Using the Formal Semantics

R(a), S(a), T(a)

R

a
1
2

S

a
1

T

a
2

What do these queries compute?

```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```

?

```
SELECT  R.a
FROM    R, S, T
WHERE   R.a=S.a
    or  R.a=T.a
```

?

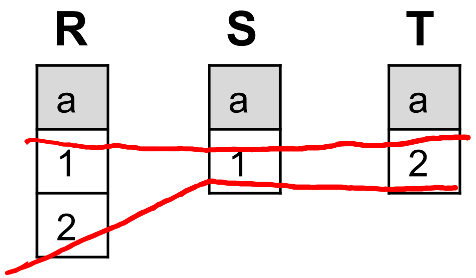
SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Using the Formal Semantics

R(a), S(a), T(a)



R

a
1
2

S

a
1

T

a
2

What do these queries compute?

```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```

a
1

Returns R ∩ S (intersection)

```
SELECT  R.a
FROM    R, S, T
WHERE   R.a=S.a
     or R.a=T.a
```

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Using the Formal Semantics

R(a), S(a), T(a)

R

a
1
2

S

a
1

T

a
2

What do these queries compute?

```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```

⇨

a
1

Returns R ∩ S
(intersection)

```
SELECT  R.a
FROM    R, S, T
WHERE   R.a=S.a
     or R.a=T.a
```

⇨

a
1
2

Returns R ∩ (S ∪ T)
if S ≠ ∅ and T ≠ ∅

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Using the Formal Semantics

R(a), S(a), T2(a)

Our colorful hands represent "team exercises" If we are online, please make a screenshot!

R

a
1
2

S

a
1

T2

a
~~2~~

Next, we are removing the input tuple "(2)"

What do these queries compute?

```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```

a
1

Returns R ∩ S (intersection)

?

```
SELECT  R.a
FROM    R, S, T2 as T
WHERE   R.a=S.a
     or R.a=T.a
```

a
1
2

Returns R ∩ (S ∪ T) if S ≠ ∅ and T ≠ ∅

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Using the Formal Semantics

R(a), S(a), T2(a)

R

a
1
2

S

a
1

T2

a
~~2~~

Next, we are removing the input tuple "(2)"

What do these queries compute?

```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```

a
1

Returns R ∩ S (intersection)

```
SELECT  R.a
FROM    R, S, T2 as T
WHERE   R.a=S.a
    or  R.a=T.a
```

a
1
2

Returns R ∩ (S ∪ T) if S ≠ ∅ and T ≠ ∅

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

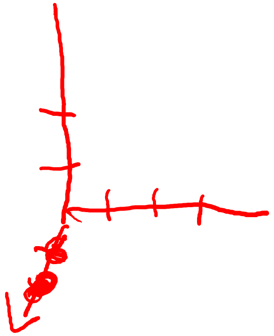
Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Using the Formal Semantics

305

R(a), S(a), T2(a)



What do these queries compute?

R

a
1
2

S

a
1
3

T2

a
~~2~~

Next, we are removing the input tuple "(2)"

```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```

a
1

Returns R ∩ S (intersection)

```
SELECT  R.a
FROM    R, S, T2 as T
WHERE   R.a=S.a
     or R.a=T.a
```

a

Returns ∅ if S = ∅ or T = ∅

Can seem counterintuitive! But remember conceptual evaluation strategy: Nested loops. If one table is empty -> no looping

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

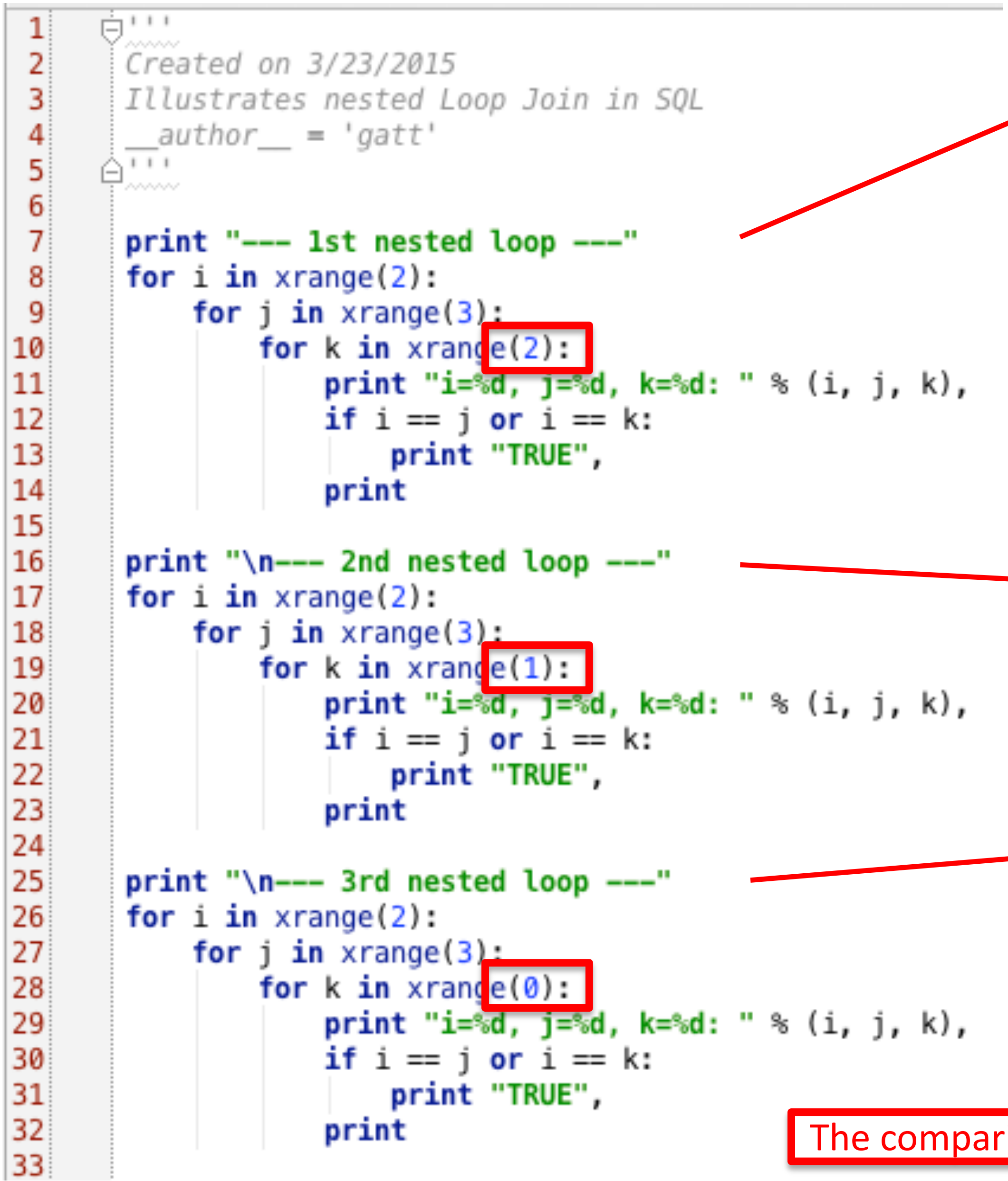
Example originally proposed in Garcia-Molina, Ullman, Widom. Database Systems. 2001. Ch. 6.2.4 Interpreting Multirelation Queries. http://infolab.stanford.edu/~ullman/dscb.html

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Illustration with Python

306

Python file



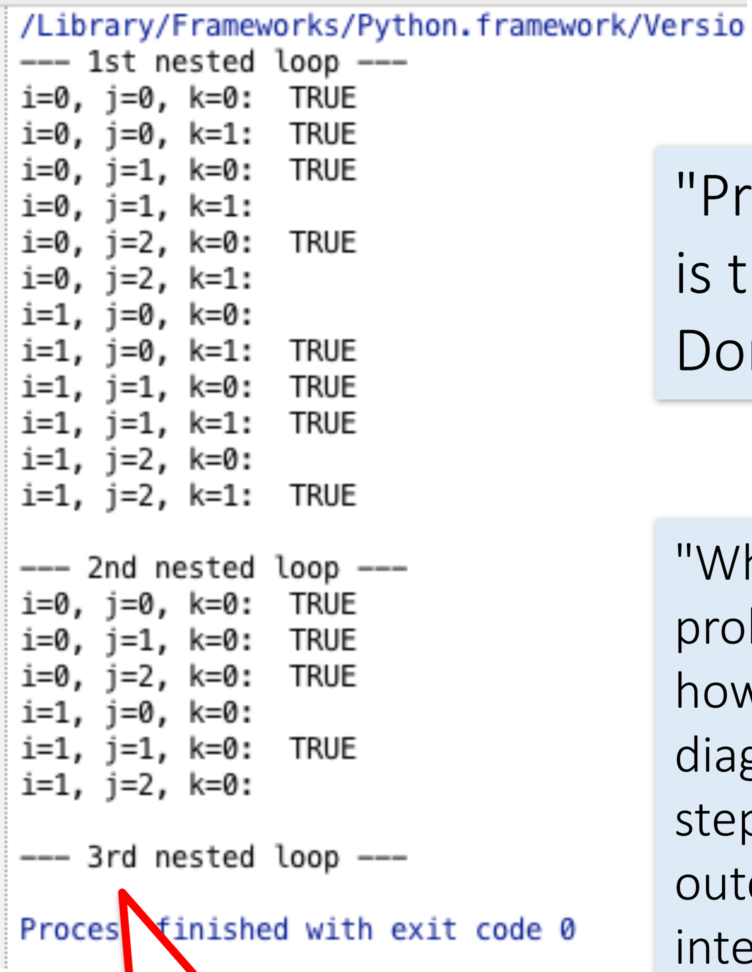
```
'''
Created on 3/23/2015
Illustrates nested Loop Join in SQL
__author__ = 'gatt'
'''

print "--- 1st nested loop ---"
for i in xrange(2):
    for j in xrange(3):
        for k in xrange(2):
            print "i=%d, j=%d, k=%d: " % (i, j, k),
            if i == j or i == k:
                print "TRUE",
            print

print "\n--- 2nd nested loop ---"
for i in xrange(2):
    for j in xrange(3):
        for k in xrange(1):
            print "i=%d, j=%d, k=%d: " % (i, j, k),
            if i == j or i == k:
                print "TRUE",
            print

print "\n--- 3rd nested loop ---"
for i in xrange(2):
    for j in xrange(3):
        for k in xrange(0):
            print "i=%d, j=%d, k=%d: " % (i, j, k),
            if i == j or i == k:
                print "TRUE",
            print

```



```
/Library/Frameworks/Python.framework/Versio
--- 1st nested loop ---
i=0, j=0, k=0:  TRUE
i=0, j=0, k=1:  TRUE
i=0, j=1, k=0:  TRUE
i=0, j=1, k=1:
i=0, j=2, k=0:  TRUE
i=0, j=2, k=1:
i=1, j=0, k=0:
i=1, j=0, k=1:  TRUE
i=1, j=1, k=0:  TRUE
i=1, j=1, k=1:  TRUE
i=1, j=2, k=0:
i=1, j=2, k=1:  TRUE

--- 2nd nested loop ---
i=0, j=0, k=0:  TRUE
i=0, j=1, k=0:  TRUE
i=0, j=2, k=0:  TRUE
i=1, j=0, k=0:
i=1, j=1, k=0:  TRUE
i=1, j=2, k=0:

--- 3rd nested loop ---

Process finished with exit code 0
```

The comparison gets never evaluated

> "Premature optimization is the root of all evil."
> Donald Knuth (1974)

> "When you are diagnosing problems, don't think about how you will solve them—just diagnose them. Blurring the steps leads to suboptimal outcomes because it interferes with uncovering the true problems."
> Ray Dalio (Principles, 2017)

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Quiz

Our colorful hands represent "team exercises"
If we are online, please make a screenshot!

302

pk

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

FK

pk

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT  DISTINCT cName
FROM
WHERE
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Quiz: Answer 1

Our colorful hands represent "team exercises"
If we are online, please make a screenshot!

302

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT  DISTINCT cName
FROM    Product as P, Company
WHERE   country = 'USA'
    and P.price < 20
    and P.price > 25
    and P.manufacturer = cName
```

What about this query?

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Quiz: Answer 1

Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT  DISTINCT cName
FROM    Product as P, Company
WHERE   country = 'USA'
    and P.price < 20
    and P.price > 25
    and P.manufacturer = cName
```

Wrong! Gives empty result: There is no product with price <20 and >25

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Quiz: Answer 2

Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT  DISTINCT cName
FROM    Product as P, Company
WHERE   country = 'USA'
    and (P.price < 20
    or  P.price > 25)
    and P.manufacturer = cName
```

What about this query?

?

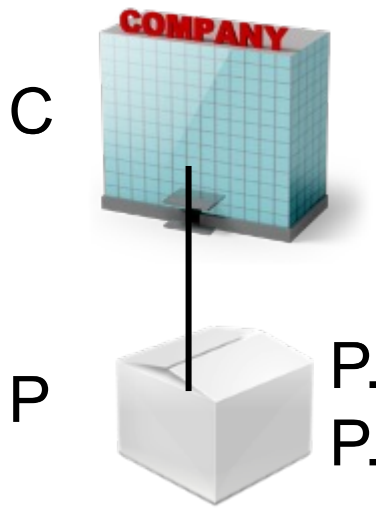
Quiz: Answer 2

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

Returns companies with single product w/price (<20 or >25)

```
SELECT  DISTINCT cName
FROM    Product as P, Company
WHERE   country = 'USA'
    and (P.price < 20
    or  P.price > 25)
    and P.manufacturer = cName
```



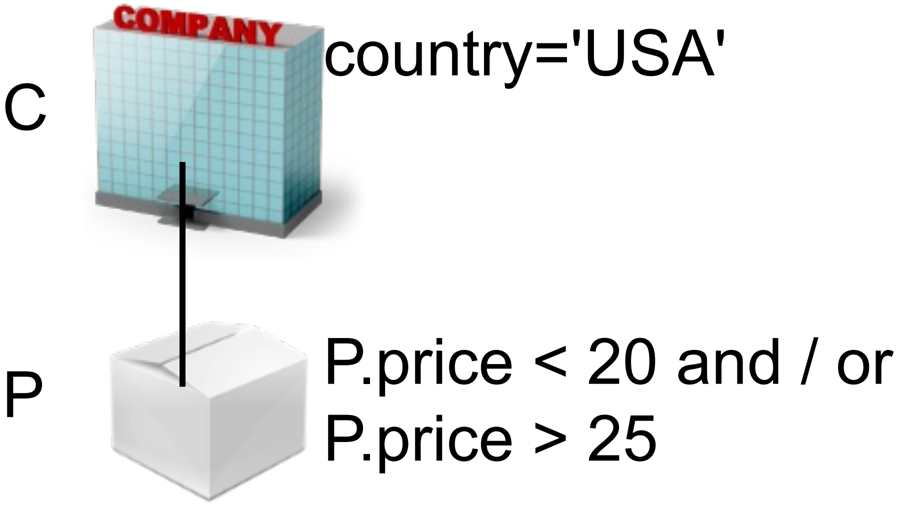
P.price<20 or
P.price>25

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Quiz: Answer 1

Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.



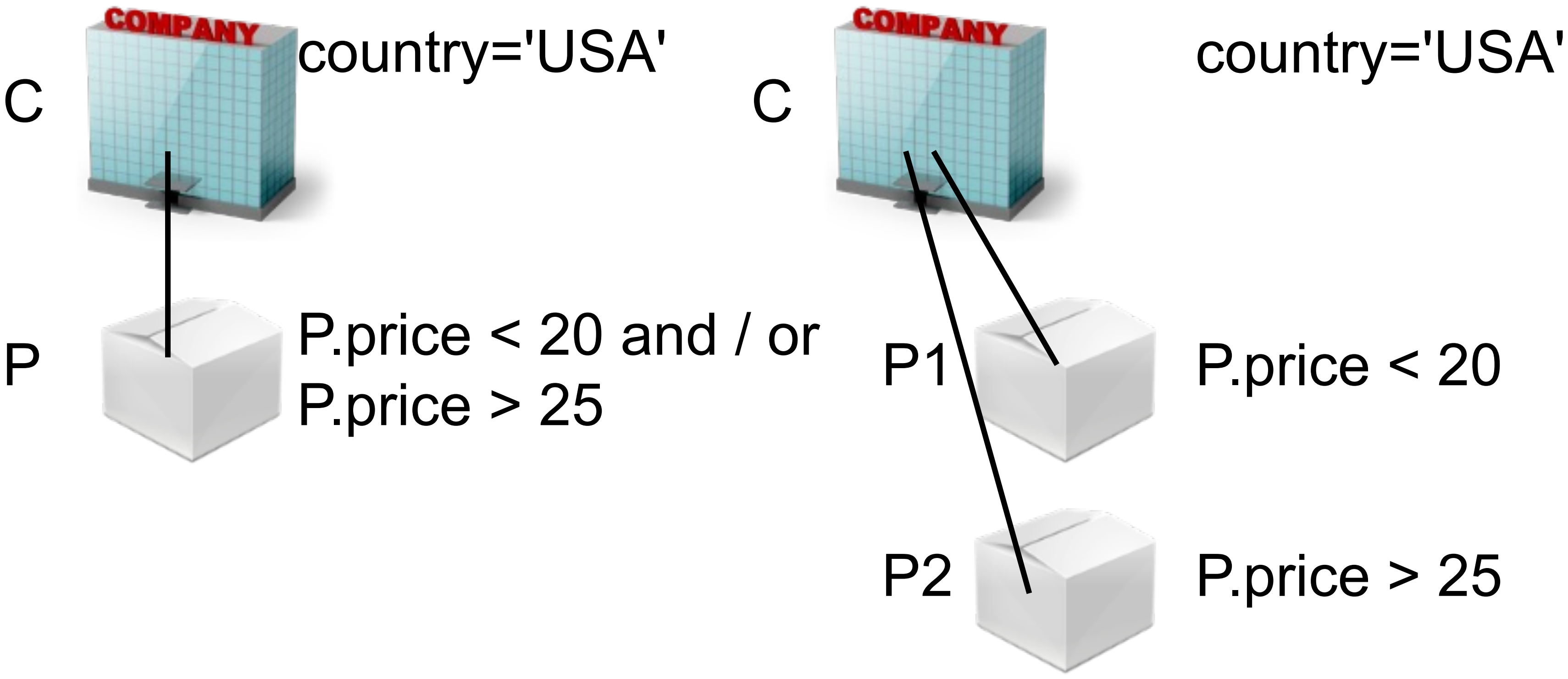
What do we actually want?

?

Quiz: Answer 1 vs. what we actually want

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

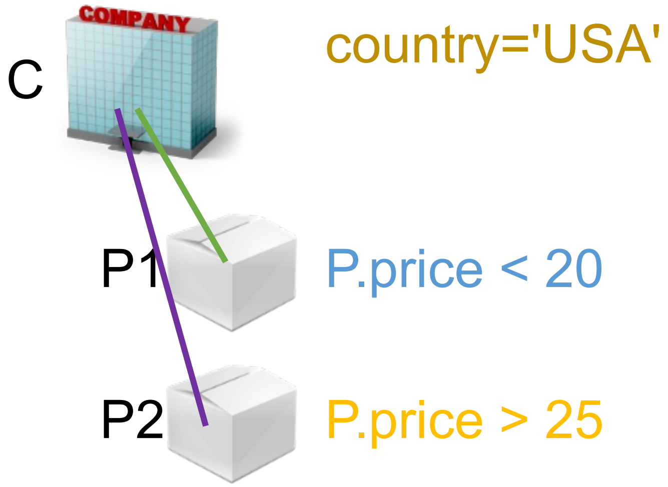
Q: Find all US companies that manufacture both a product below \$20 and a product above \$25.



Quiz: correct answer: we need "self-joins"!

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture both a product below $20 and a product above $25.



```
SELECT  DISTINCT cName
FROM    Product as P1, Product as P2, Company
WHERE   country = 'USA'
    and P1.price < 20
    and P2.price > 25
    and P1.manufacturer = cName
    and P2.manufacturer = cName
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Quiz response: we need "self-joins"!

P1

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

P2

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT  DISTINCT cName
FROM    Product as P1, Product as P2, Company
WHERE   country = 'USA'
   and  P1.price < 20
   and  P2.price > 25
   and  P1.manufacturer = cName
   and  P2.manufacturer = cName
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Quiz response: we need "self-joins"!

P1

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

P2

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

```
SELECT  DISTINCT cName
FROM    Product as P1, Product as P2, Company
WHERE   country = 'USA'
    and P1.price < 20
    and P2.price > 25
    and P1.manufacturer = cName
    and P2.manufacturer = cName
```

CName
GizmoWorks

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Outline: T1-U1: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Union and Theta Joins
  - Nulls & Outer joins
  - Top-k
  - [Recursion: moved to T1-U4: Datalog]

Grouping and Aggregation

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

⇨ ?

Q: For each product, find Total Quantities (TQ = sum of quantities) purchased, for all products with price >1.

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Grouping and Aggregation

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

⇨

Product	TQ
Bagel	?
Banana	?

Q: For each product, find Total Quantities (TQ = sum of quantities) purchased, for all products with price >1.

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Grouping and Aggregation

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
~~Banana~~	~~1~~	~~50~~
Banana	2	10
Banana	4	10

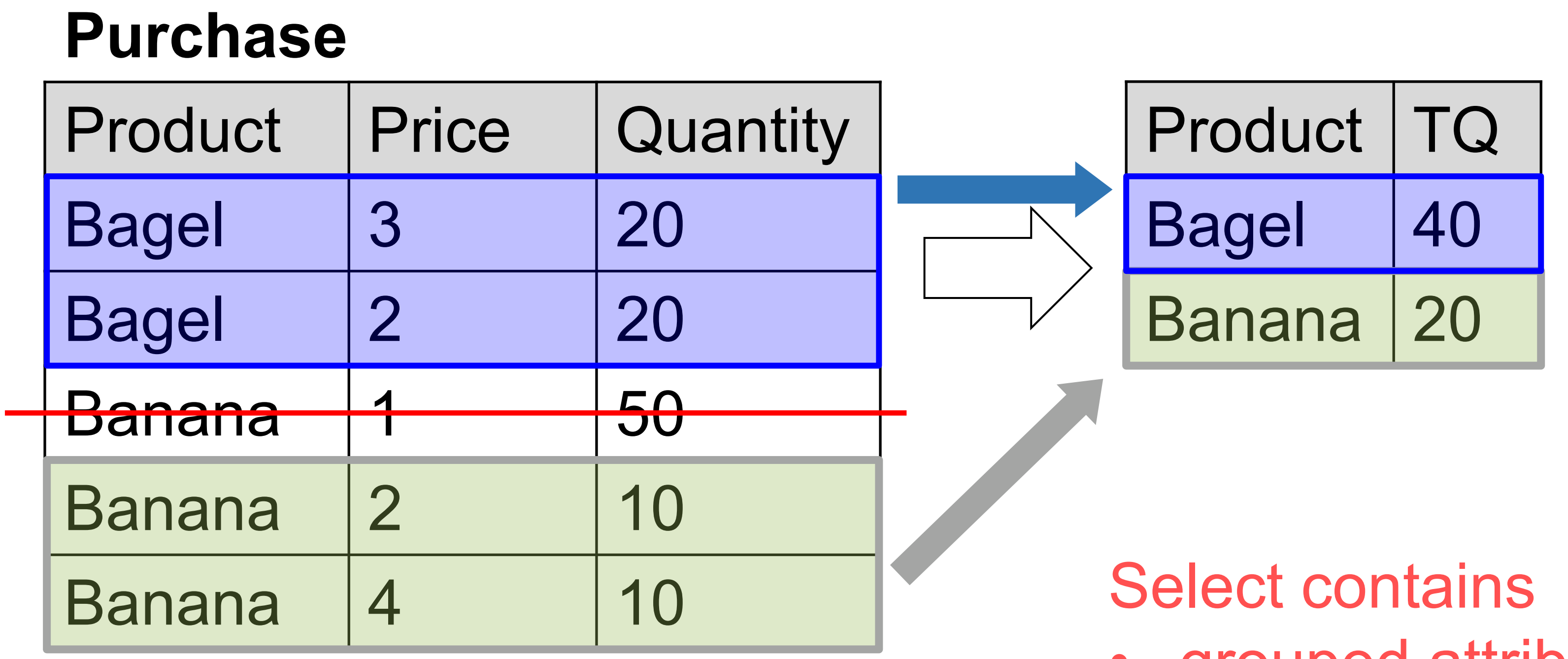
Product	TQ
Bagel	40
Banana	20

Q: For each product, find Total Quantities (TQ = sum of quantities) purchased, for all products with price >1.

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

From → Where → Group By → Select

308



Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
~~Banana~~	~~1~~	~~50~~
Banana	2	10
Banana	4	10

Product	TQ
Bagel	40
Banana	20

Select contains
- grouped attributes
- and aggregates

```
4  SELECT      product, sum(quantity) as TQ
1  FROM        Purchase
2  WHERE       price > 1
3  GROUP BY    product
```

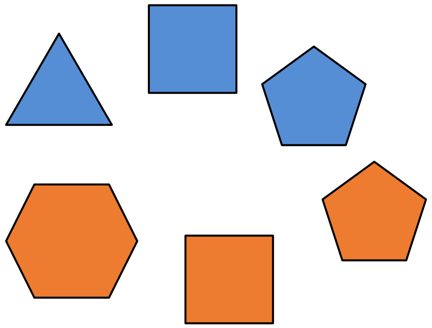
Tuples grouped together need to share the same value for attribute "product"

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Groupings illustrated with colored shapes

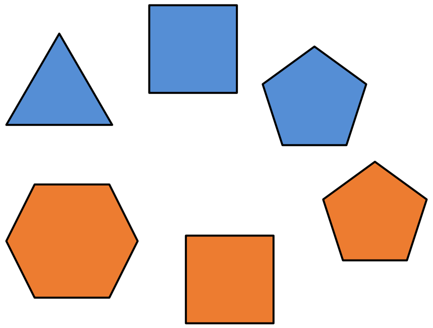
group by color



```
SELECT  color,
        avg(numc) anc
FROM    Shapes
GROUP BY color
```

⇨ ?

group by numc (# of corners)



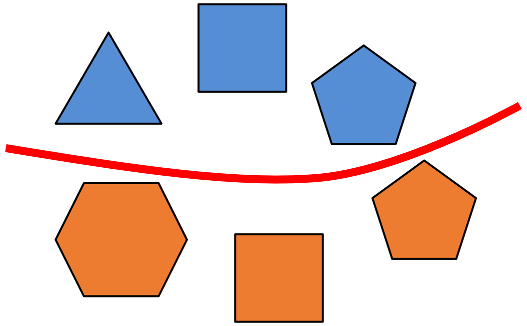
```
SELECT  numc
FROM    Shapes
GROUP BY numc
```

⇨ ?

Groupings illustrated with colored shapes

color	numc
blue	3
blue	4
blue	5
orange	4
orange	5
orange	6

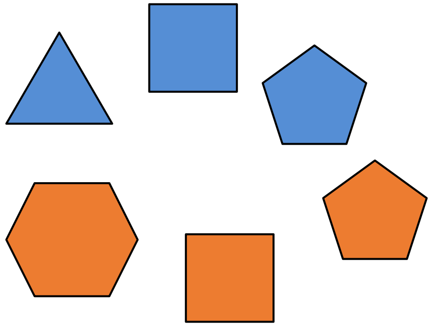
group by color



```
SELECT  color,
        avg(numc) anc
FROM    Shapes
GROUP BY color
```

⇨ ?

group by numc (# of corners)



```
SELECT  numc
FROM    Shapes
GROUP BY numc
```

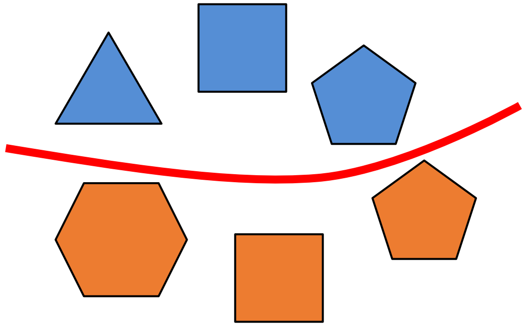
⇨ ?

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

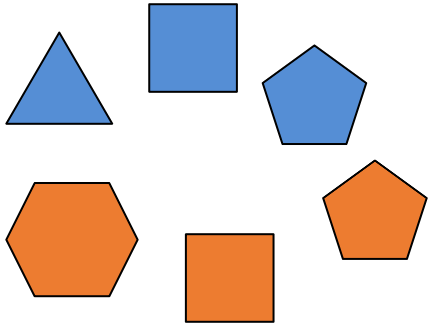
Groupings illustrated with colored shapes

color	numc
blue	3
blue	4
blue	5
orange	4
orange	5
orange	6

group by color



group by numc (# of corners)



```
SELECT  color,
        avg(numc) anc
FROM    Shapes
GROUP BY color
```

color	anc
blue	4
orange	5

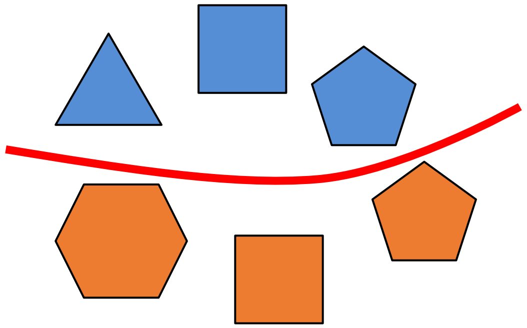
```
SELECT  numc
FROM    Shapes
GROUP BY numc
```

?

Groupings illustrated with colored shapes

color	numc
blue	3
blue	4
blue	5
orange	4
orange	5
orange	6

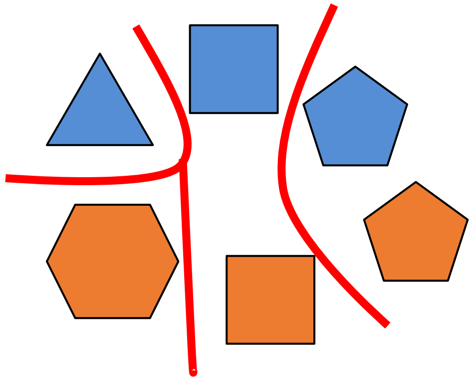
group by color



```
SELECT  color,
        avg(numc) anc
FROM    Shapes
GROUP BY color
```

color	anc
blue	4
orange	5

group by numc (# of corners)



```
SELECT  numc
FROM    Shapes
GROUP BY numc
```

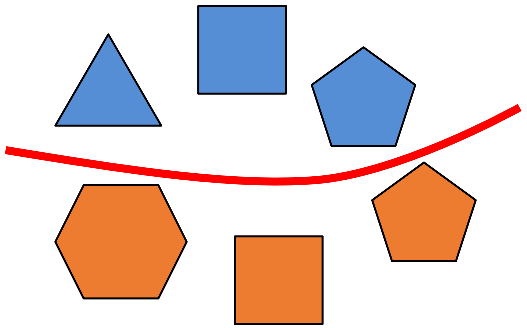
numc
3
4
5
6

Without group by ?

Groupings illustrated with colored shapes

color	numc
blue	3
blue	4
blue	5
orange	4
orange	5
orange	6

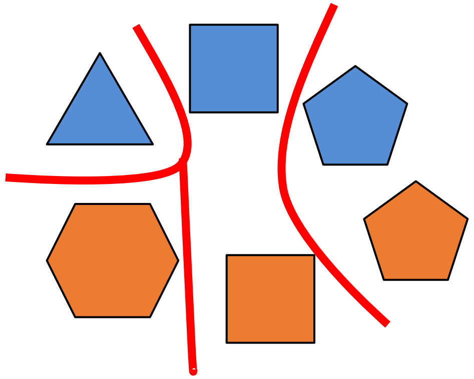
group by color



```
SELECT   color,
         avg(numc) anc
FROM     Shapes
GROUP BY color
```

color	anc
blue	4
orange	5

group by numc (# of corners)



```
SELECT   numc
FROM     Shapes
GROUP BY numc
```

numc
3
4
5
6

Same as:

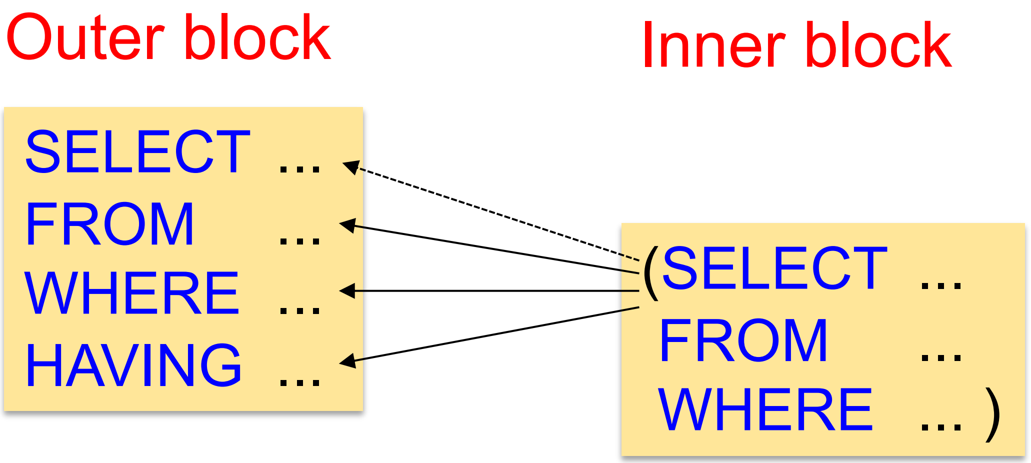
```
SELECT DISTINCT numc
FROM     Shapes
```

Without group by!

Outline: T1-U1: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Union and Theta Joins
  - Nulls & Outer joins
  - Top-k
  - [Recursion: moved to T1-U4: Datalog]

Subqueries = Nested queries



Outer block

SELECT ...
FROM ...
WHERE ...
HAVING ...

Inner block

(SELECT ...
FROM ...
WHERE ...)

We focus mainly on nestings in the WHERE clause, which are the most expressive type of nesting.

- We can nest queries because SQL is compositional:
  - Input & Output are represented as relations (multisets)
  - Subqueries also return relations; thus the output of one query can thus be used as the input to another (nesting)
- This is extremely powerful (think in terms of input/output)
- A complication: subqueries can be correlated (not just in-/output)

Subqueries in

SELECT clause

FROM clause (also called "derived tables")

WHERE clause

HAVING clause

Subqueries in FROM clause = Derived tables

308

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

⇨

Product	TQ
Bagel	40
Banana	70

⇨

MTQ
70

Q1: For each product, find total quantities (sum of quantities) purchased.

Q2: Find the maximal total quantities purchased across all products.

```
SELECT product, SUM(quantity) as TQ
FROM Purchase
GROUP BY   product
```

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Subqueries in FROM clause = Derived tables

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

⇨

X

Product	TQ
Bagel	40
Banana	70

⇨

MTQ
70

Q1: For each product, find total quantities (sum of quantities) purchased.

Q2: Find the maximal total quantities purchased across all products.

```
(
SELECT product, SUM(quantity) as TQ
FROM Purchase
GROUP BY   product
) X
```

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Subqueries in FROM clause = Derived tables

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

X

Product	TQ
Bagel	40
Banana	70

MTQ
70

Q1: For each product, find total quantities (sum of quantities) purchased.

Q2: Find the maximal total quantities purchased across all products.

```
(
SELECT product, SUM(quantity) as TQ
FROM Purchase
GROUP BY   product          ) X
```

```
SELECT MAX(TQ) as MTQ
FROM X
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Subqueries in FROM clause = Derived tables

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

```
SELECT MAX(TQ) as MTQ
FROM (SELECT product, SUM(quantity) as TQ
      FROM Purchase
      GROUP BY product) X
```

MTQ
70

Q1: For each product, find total quantities (sum of quantities) purchased.

Q2: Find the maximal total quantities purchased across all products.

```
SELECT product, SUM(quantity) as TQ
FROM Purchase
GROUP BY   product
```

```
SELECT MAX(TQ) as MTQ
FROM X
```

Common Table Expressions (CTE): WITH clause

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

```
SELECT MAX(TQ) as MTQ
FROM (SELECT product, SUM(quantity) as TQ
          FROM Purchase
          GROUP BY product) X
```

MTQ
70

CTE (Common Table Expression)

```
WITH  X as
          (SELECT product, SUM(quantity) as TQ
          FROM Purchase
          GROUP BY product)
```

Query using CTE

```
SELECT MAX(TQ) as MTQ
FROM X
```

The WITH clause defines a temporary relation that is available <u>only to the query in which it occurs</u>. Sometimes easier to read. Very useful for queries that need to access the same intermediate result multiple times

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Subqueries in

SELECT clause

FROM clause

WHERE clause (including IN, ANY, ALL)

HAVING clause

Subqueries in WHERE clause

R

a
1
2

W

a	b
2	0
3	0
4	0

What do these queries return?

```
SELECT   a
FROM     R
WHERE    a IN
         (SELECT a FROM W)
```

?

```
SELECT   a
FROM     R
WHERE    a < ANY
         (SELECT a FROM W)
```

?

```
SELECT   a
FROM     R
WHERE    a < ALL
         (SELECT a FROM W)
```

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Subqueries in WHERE clause

305

R

a
1
2

W

a	b
2	0
3	0
4	0

What do these queries return?

```
SELECT   a
FROM     R
WHERE    a IN
         (SELECT a FROM W)
```

⇨

a
2

Since 2 is in the set (bag) (2, 3, 4)

```
SELECT   a
FROM     R
WHERE    a < ANY
         (SELECT a FROM W)
```

⇨ ?

```
SELECT   a
FROM     R
WHERE    a < ALL
         (SELECT a FROM W)
```

⇨ ?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Subqueries in WHERE clause

R

a
1
2

W

a	b
2	0
3	0
4	0

What do these queries return?

```
SELECT   a
FROM     R
WHERE    a IN
         (SELECT a FROM W)
```

⇨

a
2

Since 2 is in the set (bag) (2, 3, 4)

```
SELECT   a
FROM     R
WHERE    a < ANY
         (SELECT a FROM W)
```

⇨

a
1
2

Since 1 and 2 are < than at least one ("any") of 2, 3 or 4

```
SELECT   a
FROM     R
WHERE    a < ALL
         (SELECT a FROM W)
```

⇨ ?

Subqueries in WHERE clause

305

R

a
1
2

W

a	b
2	0
3	0
4	0

What do these queries return?

SQLlite does not support "ANY" or "ALL" ☹

```
SELECT   a
FROM     R
WHERE    a IN
         (SELECT a FROM W)
```

⇨

a
2

Since 2 is in the set (bag) (2, 3, 4)

```
SELECT   a
FROM     R
WHERE    a < ANY
         (SELECT a FROM W)
```

⇨

a
1
2

Since 1 and 2 are < than at least one ("any") of 2, 3 or 4

```
SELECT   a
FROM     R
WHERE    a < ALL
         (SELECT a FROM W)
```

⇨

a
1

Since 1 is < than each ("all") of 2, 3, and 4

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Correlated subqueries

- In all previous cases, the nested subquery in the inner select block could be entirely evaluated before processing the outer select block.
  - Recall the "compositional" nature of relational queries
  - This is no longer the case for correlated nested queries.
- Whenever a condition in the WHERE clause of a nested query references some column of a table declared in the outer query, the two queries are said to be correlated.
  - The nested query is then evaluated once for each tuple (or combination of tuples) in the outer query (that's the conceptual evaluation strategy)

Correlated subquery (existential ∃)

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

slightly different product database!

Q$_1$: Find all companies that make some product(s) with price < 25

Using IN: Set / Bag membership

```
SELECT  DISTINCT C.cname
FROM    Company C
WHERE   C.cid IN ( SELECT  P.cid
                   FROM    Product P
                   WHERE   P.price < 25)
```

Is this a correlated nested query ?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Correlated subquery (existential ∃)

316

slightly different product database!

Product

<u>PName</u>	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

<u>cid</u>	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

Q_1: Find all companies that make <u>some</u> product(s) with price < 25

Using IN: Set / Bag membership

```
SELECT  DISTINCT C.cname
FROM    Company C
WHERE   C.cid IN ( SELECT  P.cid
                   FROM    Product P
                   WHERE   P.price < 25)
```

Not a correlated nested query!

```
SELECT  DISTINCT C.cname
FROM    Company C
WHERE   C.cid IN ( 1, 2 )
```

Inner query does not reference outer query! You could first evaluate the inner query by itself.

Correlated subquery (existential ∃)

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

Q_1: Find all companies that make some product(s) with price < 25

Using EXISTS: TRUE if the subquery's result is NOT empty

```
SELECT   DISTINCT C.cname
FROM     Company C
WHERE    EXISTS ( SELECT  *
                  FROM    Product P
                  WHERE   P.cid = C.cid
                  and     P.price < 25)
```

Is this a correlated nested query ?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Correlated subquery (existential ∃)

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

Q_1: Find all companies that make some product(s) with price < 25

Using EXISTS: TRUE if the subquery's result is NOT empty

```
SELECT  DISTINCT C.cname
FROM    Company C
WHERE   EXISTS ( SELECT  *
                 FROM    Product P
                 WHERE   P.cid = C.cid
                 and     P.price < 25)
```

This is a correlated nested query! Notice the additional join condition referencing a relation from the outer query.

Recall our conceptual evaluation strategy!

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Correlated subquery (existential ∃)

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

Q_1: Find all companies that make some product(s) with price < 25

Using ANY (also SOME): again set / bag comparison

```
SELECT  DISTINCT C.cname
FROM    Company C
WHERE   25 > ANY ( SELECT  price
                   FROM    Product P
                   WHERE   P.cid = C.cid)
```

But do we really need to write this query as nested query ?

SQLlite does not support "ANY" ☹

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Correlated subquery (existential ∃)

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

Q_1: Find all companies that make some product(s) with price < 25

```
SELECT  DISTINCT C.cname
FROM    Company C, Product P
WHERE   C.cid = P.cid
and     P.price < 25
```

We did not need to write nested queries;
we can "unnest" it!

Existential quantifiers are easy ☺

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Correlated subquery (universal ∀)

316

Product

PName	Price	Category	cid
Gizmo	$19.99	Gadgets	1
Powergizmo	$29.99	Gadgets	1
SingleTouch	$14.99	Photography	2
MultiTouch	$203.99	Household	3

Company

cid	CName	StockPrice	Country
1	GizmoWorks	25	USA
2	Canon	65	Japan
3	Hitachi	15	Japan

~~Q_1: Find all companies that make some product(s) with price < 25~~

Q_2: Find all companies that make only products with price < 25

≡ Q_2: Find all companies for which all products have price < 25

≡ Q_2: Find all companies that do not have any product with price >= 25

Universal quantifiers are more complicated ! ☹
(Think about the companies that should not be returned)

All three formulations are equivalent: a company with no product will be returned!

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Correlated subquery (universal ∀ = not exists ∄)

316

Q_2: Find all companies that make only products with price < 25

Step 1: Q_2': Find the other companies that make some product(s) with price ≥ 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE   C.cid IN        ( SELECT  P.cid
                          FROM    Product P
                          WHERE   P.price >= 25)
```

First think about the companies that should not be returned!

Step 2: Q_2: Find all companies that make no products with price ≥ 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE   C.cid NOT IN ( SELECT  P.cid
                       FROM    Product P
                       WHERE   P.price >= 25)
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Correlated subquery (universal ∀ = not exists ∄)

316

Q_2: Find all companies that make only products with price < 25

Step 1: Q_2': Find the other companies that make some product(s) with price ≥ 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE  EXISTS          ( SELECT  *
                          FROM     Product P
                          WHERE   C.cid = P.cid
                          and        P.price >= 25)
```

First think about the companies that should not be returned!

Step 2: Q_2: Find all companies that make no products with price ≥ 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE  NOT EXISTS ( SELECT  *
                          FROM     Product P
                          WHERE   C.cid = P.cid
                          and        P.price >= 25)
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Correlated subquery (universal ∀ = not exists ∄)


316

Q_2: Find all companies that make only products with price < 25

Step 1: Q_2': Find the other companies that make some product(s) with price ≥ 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE  25 <= ANY    ( SELECT P.price
                      FROM    Product P
                      WHERE  C.cid = P.cid)
```

First think about the companies that should not be returned!

Step 2: Q_2: Find all companies that make no products with price ≥ 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE  25 > ALL       ( SELECT P.price
                      FROM    Product P
                      WHERE  C.cid = P.cid)
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Correlated subquery (universal ∀ = not exists ∄)

316

Q_2: Find all companies that make only products with price < 25

Step 1: Q_2': Find the other companies that make some product(s) with price ≥ 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE  25 <= ANY     ( SELECT  P.price
                         FROM     Product P
                         WHERE   C.cid = P.cid)
```

First think about the companies that should not be returned!

Step 2: Q_2: Find all companies that make no products with price ≥ 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE  25 > ALL       ( SELECT  P.price
                         FROM     Product P
                         WHERE   C.cid = P.cid)
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

A natural question

Q_2: Find all companies that make only products with price < 25

- How can we unnest (no GROUP BY) the universal quantifier query ?

```
SELECT  ...
FROM    ...
WHERE   ...
```

?

Source: Dan Suciu

Updated 1/16/2024

Topic 1: Data models and query languages
Unit 1: SQL (continued)
Lecture 3

Wolfgang Gatterbauer

CS7240 Principles of scalable data management (sp24)

https://northeastern-datalab.github.io/cs7240/sp24/

1/16/2024

A natural question

Q_2: Find all companies that make only products with price < 25

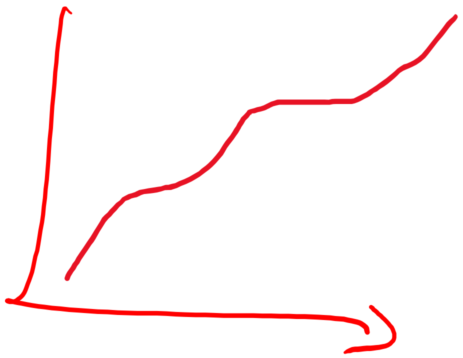
- How can we unnest (no GROUP BY) the universal quantifier query ?

```
SELECT  ...
FROM    ...
WHERE   ...
```

?

Source: Dan Suciu

Queries that must be nested



- Definition: A query Q is monotone if:
  - Whenever we add tuples to one or more of the tables…
  - … the answer to the query cannot contain fewer tuples
- Fact: all unnested queries are monotone
  - Proof: using the "nested for loops" semantics
- Fact: Query with universal quantifier is not monotone
  - Add one tuple violating the condition. Then "all" returns fewer tuples
- Consequence: we cannot unnest a query with a universal quantifier

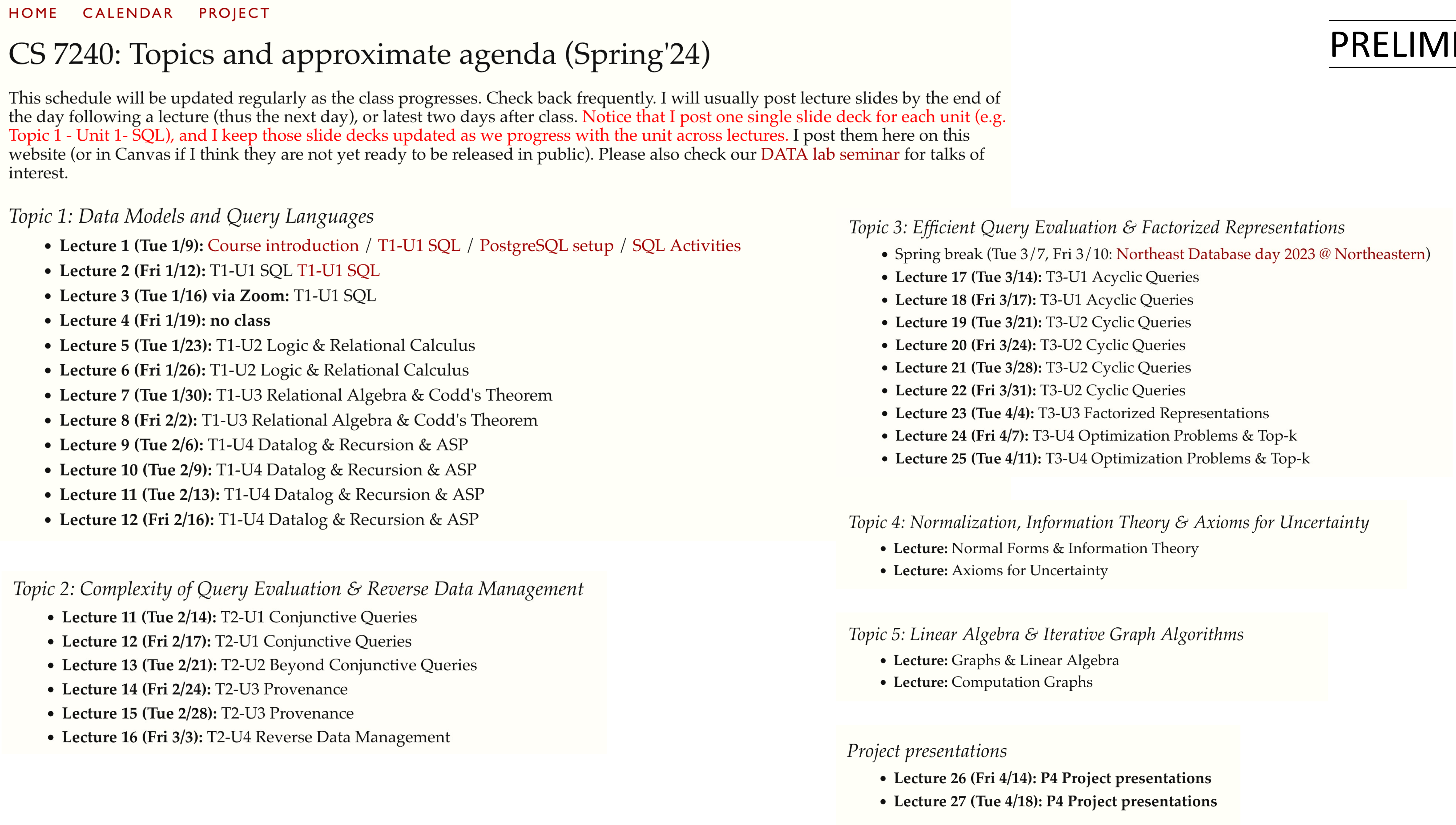
Source: Dan Suciu

Pre-class conversations

- Last class summary
- New class members: next time quick introduction
- Please point out any errors on slides, web page, Canvas, etc.
- Any questions on class policies

- Today:
  - SQL continued
  - Recall: no class this FRI, we resume TUE in person

PRELIMINARY



HOME CALENDAR PROJECT

CS 7240: Topics and approximate agenda (Spring'24)

This schedule will be updated regularly as the class progresses. Check back frequently. I will usually post lecture slides by the end of the day following a lecture (thus the next day), or latest two days after class. Notice that I post one single slide deck for each unit (e.g. Topic 1 - Unit 1- SQL), and I keep those slide decks updated as we progress with the unit across lectures. I post them here on this website (or in Canvas if I think they are not yet ready to be released in public). Please also check our DATA lab seminar for talks of interest.

Topic 1: Data Models and Query Languages

- **Lecture 1 (Tue 1/9):** Course introduction / T1-U1 SQL / PostgreSQL setup / SQL Activities
- **Lecture 2 (Fri 1/12):** T1-U1 SQL T1-U1 SQL
- **Lecture 3 (Tue 1/16) via Zoom:** T1-U1 SQL
- **Lecture 4 (Fri 1/19): no class**
- **Lecture 5 (Tue 1/23):** T1-U2 Logic & Relational Calculus
- **Lecture 6 (Fri 1/26):** T1-U2 Logic & Relational Calculus
- **Lecture 7 (Tue 1/30):** T1-U3 Relational Algebra & Codd's Theorem
- **Lecture 8 (Fri 2/2):** T1-U3 Relational Algebra & Codd's Theorem
- **Lecture 9 (Tue 2/6):** T1-U4 Datalog & Recursion & ASP
- **Lecture 10 (Tue 2/9):** T1-U4 Datalog & Recursion & ASP
- **Lecture 11 (Tue 2/13):** T1-U4 Datalog & Recursion & ASP
- **Lecture 12 (Fri 2/16):** T1-U4 Datalog & Recursion & ASP

Topic 2: Complexity of Query Evaluation & Reverse Data Management

- **Lecture 11 (Tue 2/14):** T2-U1 Conjunctive Queries
- **Lecture 12 (Fri 2/17):** T2-U1 Conjunctive Queries
- **Lecture 13 (Tue 2/21):** T2-U2 Beyond Conjunctive Queries
- **Lecture 14 (Fri 2/24):** T2-U3 Provenance
- **Lecture 15 (Tue 2/28):** T2-U3 Provenance
- **Lecture 16 (Fri 3/3):** T2-U4 Reverse Data Management

Topic 3: Efficient Query Evaluation & Factorized Representations

- Spring break (Tue 3/7, Fri 3/10: Northeast Database day 2023 @ Northeastern)
- **Lecture 17 (Tue 3/14):** T3-U1 Acyclic Queries
- **Lecture 18 (Fri 3/17):** T3-U1 Acyclic Queries
- **Lecture 19 (Tue 3/21):** T3-U2 Cyclic Queries
- **Lecture 20 (Fri 3/24):** T3-U2 Cyclic Queries
- **Lecture 21 (Tue 3/28):** T3-U2 Cyclic Queries
- **Lecture 22 (Fri 3/31):** T3-U2 Cyclic Queries
- **Lecture 23 (Tue 4/4):** T3-U3 Factorized Representations
- **Lecture 24 (Fri 4/7):** T3-U4 Optimization Problems & Top-k
- **Lecture 25 (Tue 4/11):** T3-U4 Optimization Problems & Top-k

Topic 4: Normalization, Information Theory & Axioms for Uncertainty

- **Lecture:** Normal Forms & Information Theory
- **Lecture:** Axioms for Uncertainty

Topic 5: Linear Algebra & Iterative Graph Algorithms

- **Lecture:** Graphs & Linear Algebra
- **Lecture:** Computation Graphs

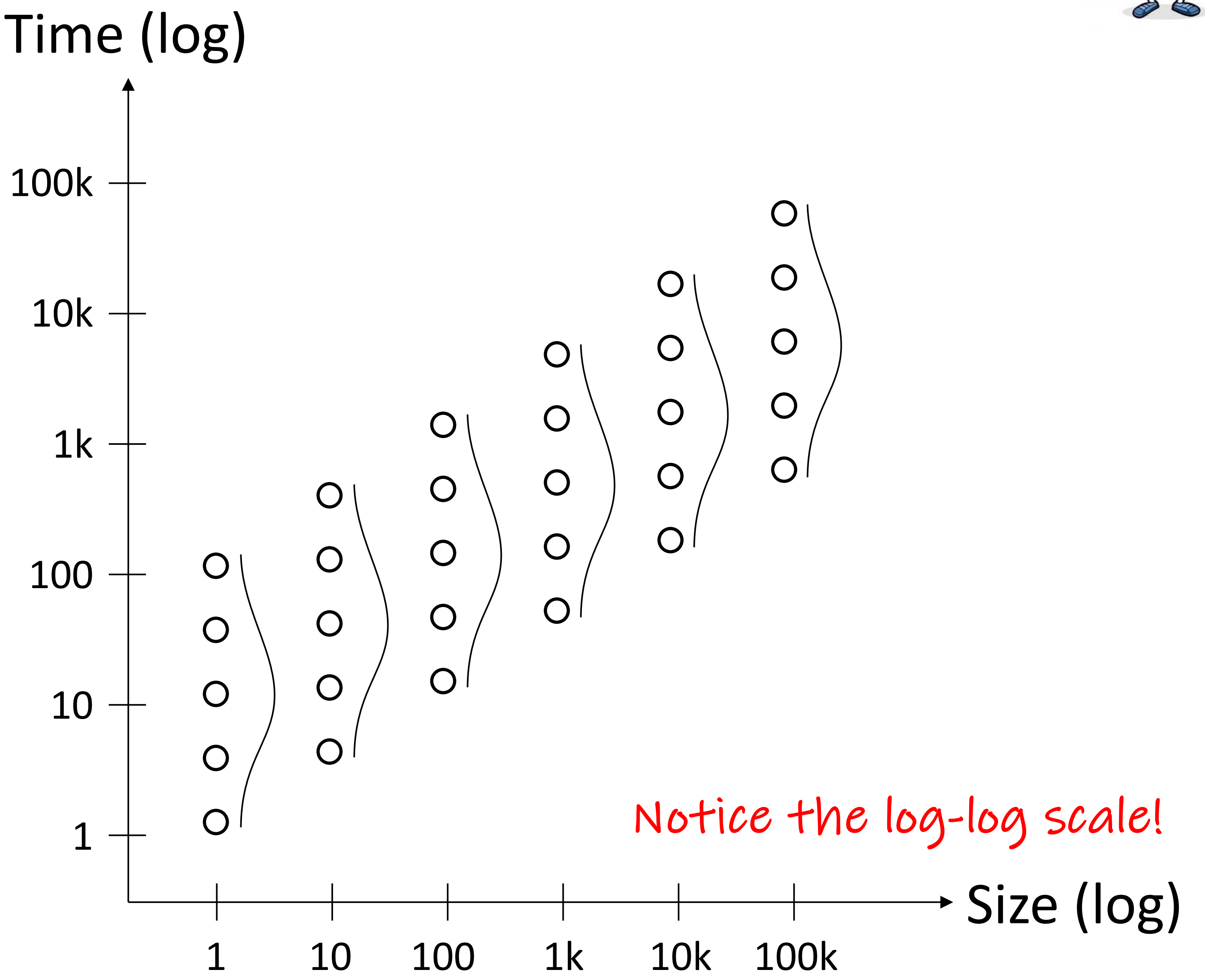
Project presentations

- **Lecture 26 (Fri 4/14): P4 Project presentations**
- **Lecture 27 (Tue 4/18): P4 Project presentations**

Revisiting our question from first class

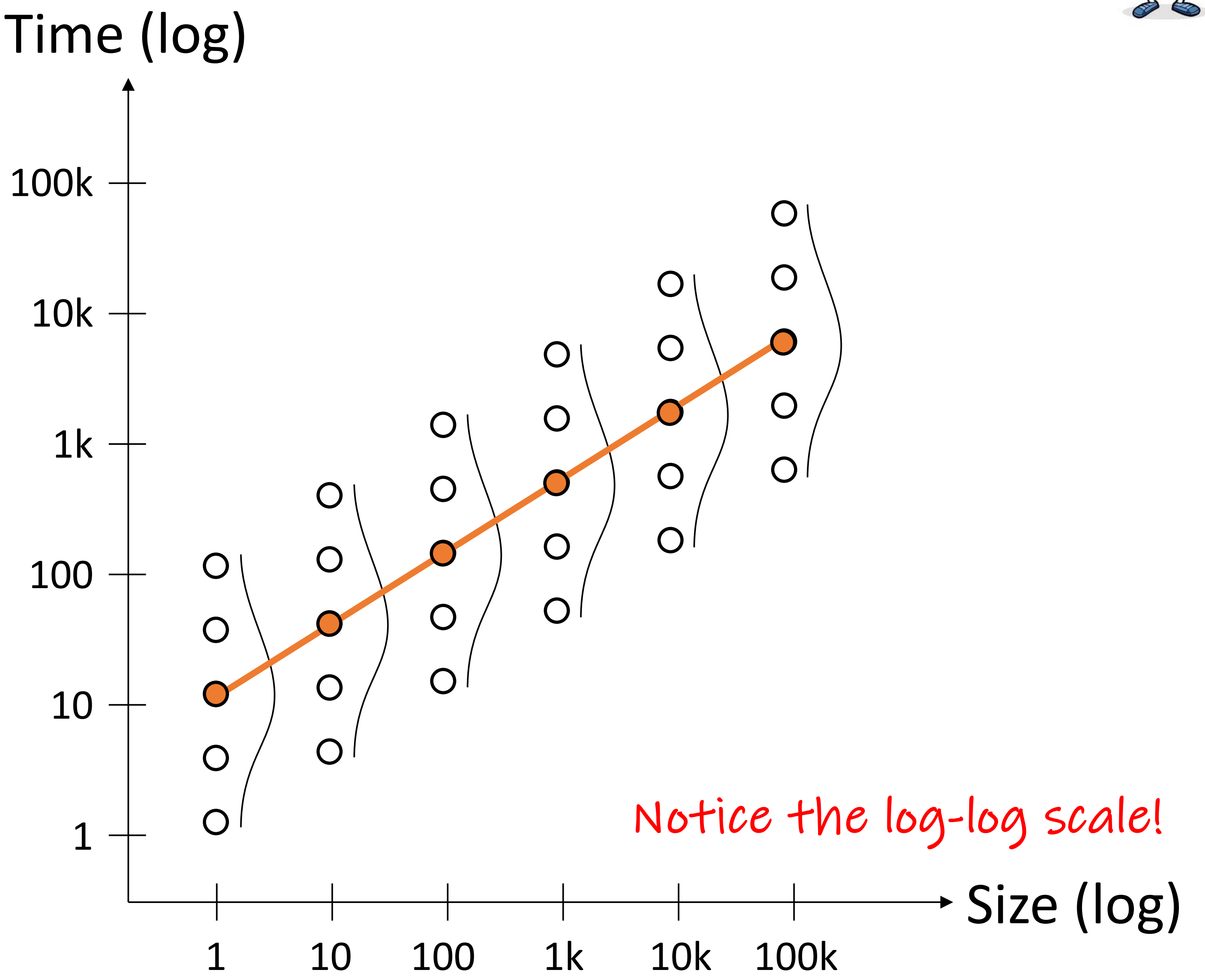
Question: How to deal with cut-offs when binning?

- These are the true points that you would get if you could run the experiments long enough.



Question: How to deal with cut-offs when binning?

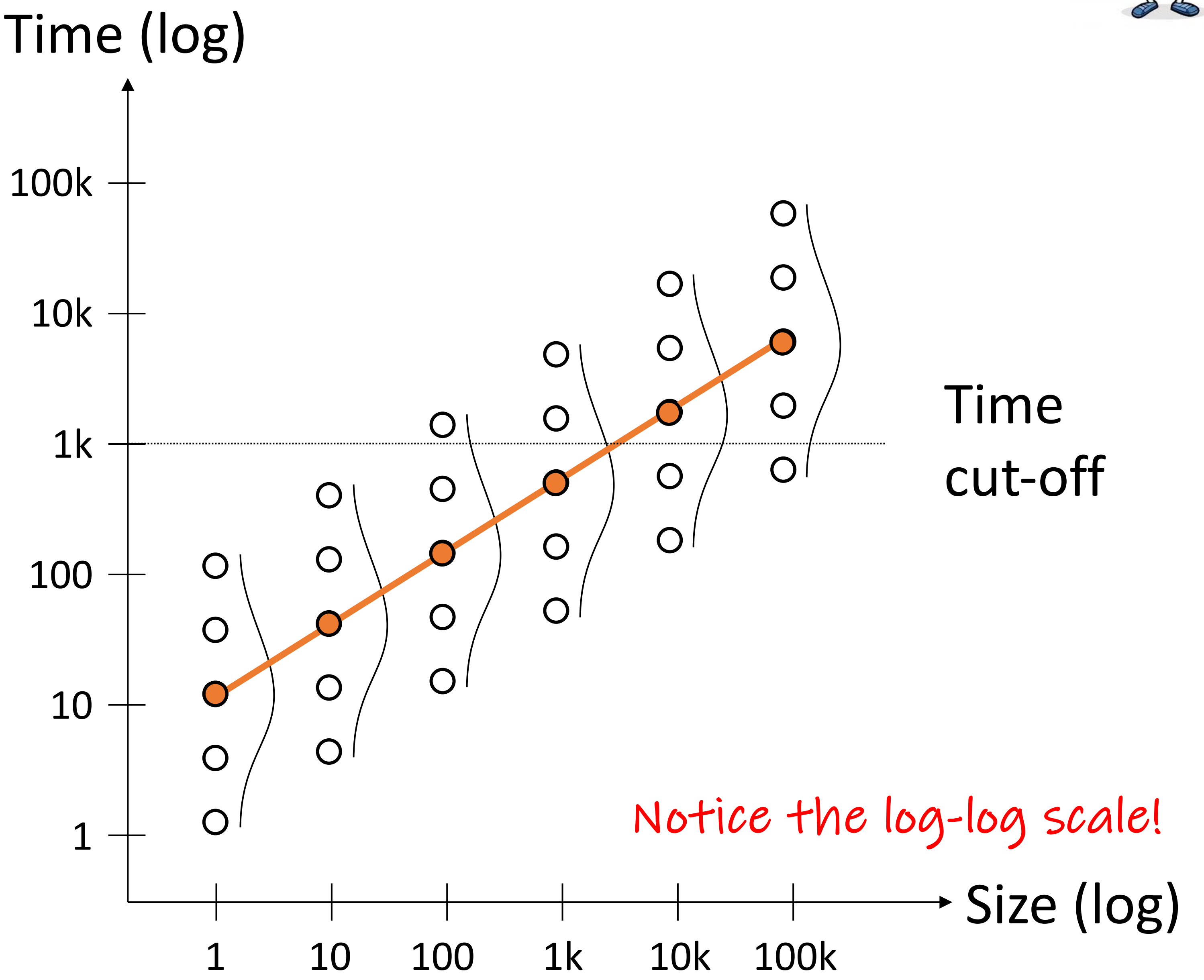
- These are the true points that you would get if you could run the experiments long enough.
- Here is what the aggregate would look like like if we could get all points and then aggregated for each size



Question: How to deal with cut-offs when binning?

- These are the true points that you would get if you could run the experiments long enough.
- Here is what the aggregate would look like like if we could get all points and then aggregated for each size
- However, some experiments take too long and we thus have to cut them off after some time.

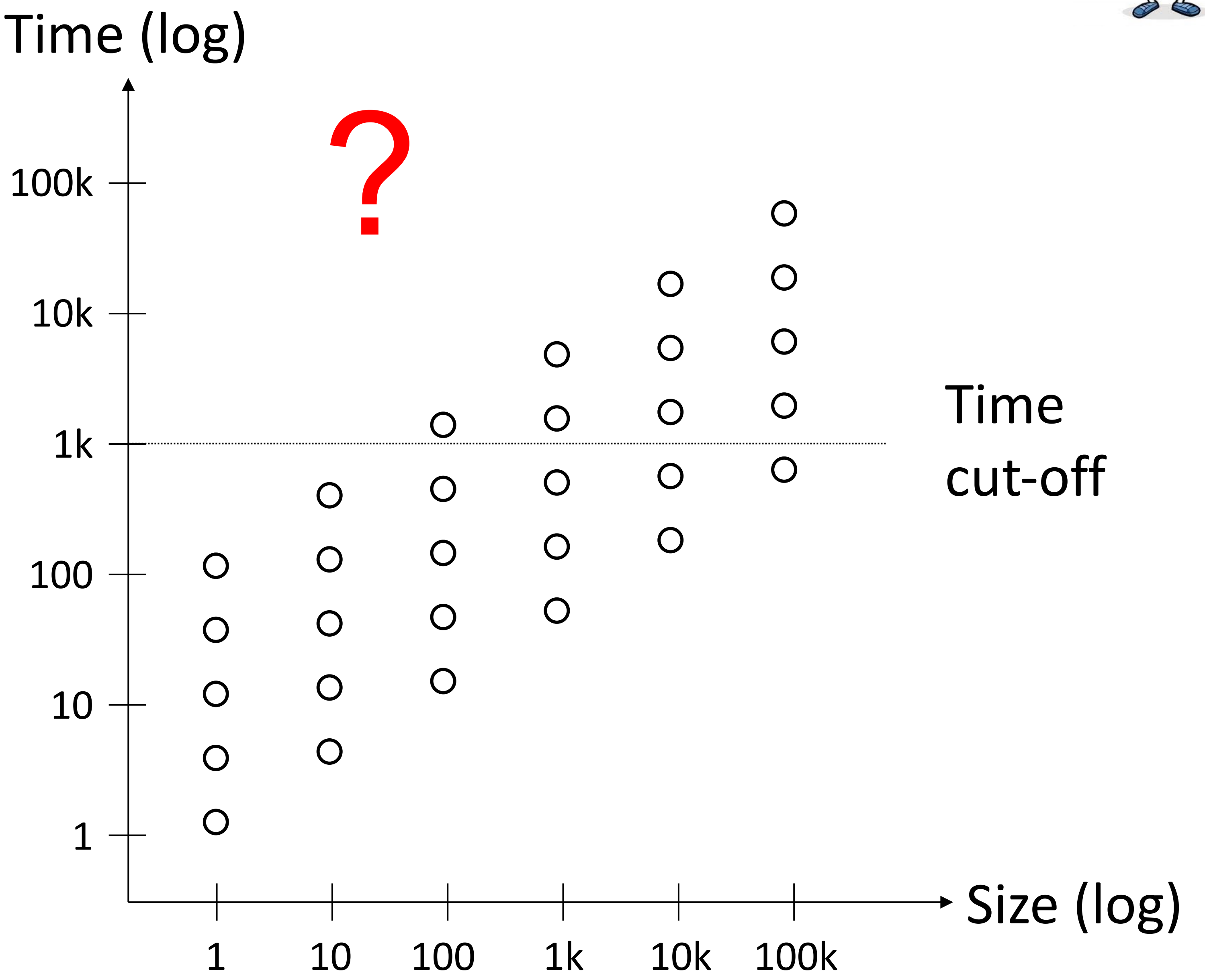
Question: There is an overall trend, yet big variation for each experiment. We still like to capture the overall trend with some smart aggregations. What can we do?



Notice the log-log scale!

Question: How to deal with cut-offs when binning?

- Option 1: What if we terminate experiments that take too long, and only average over the "seen points"?

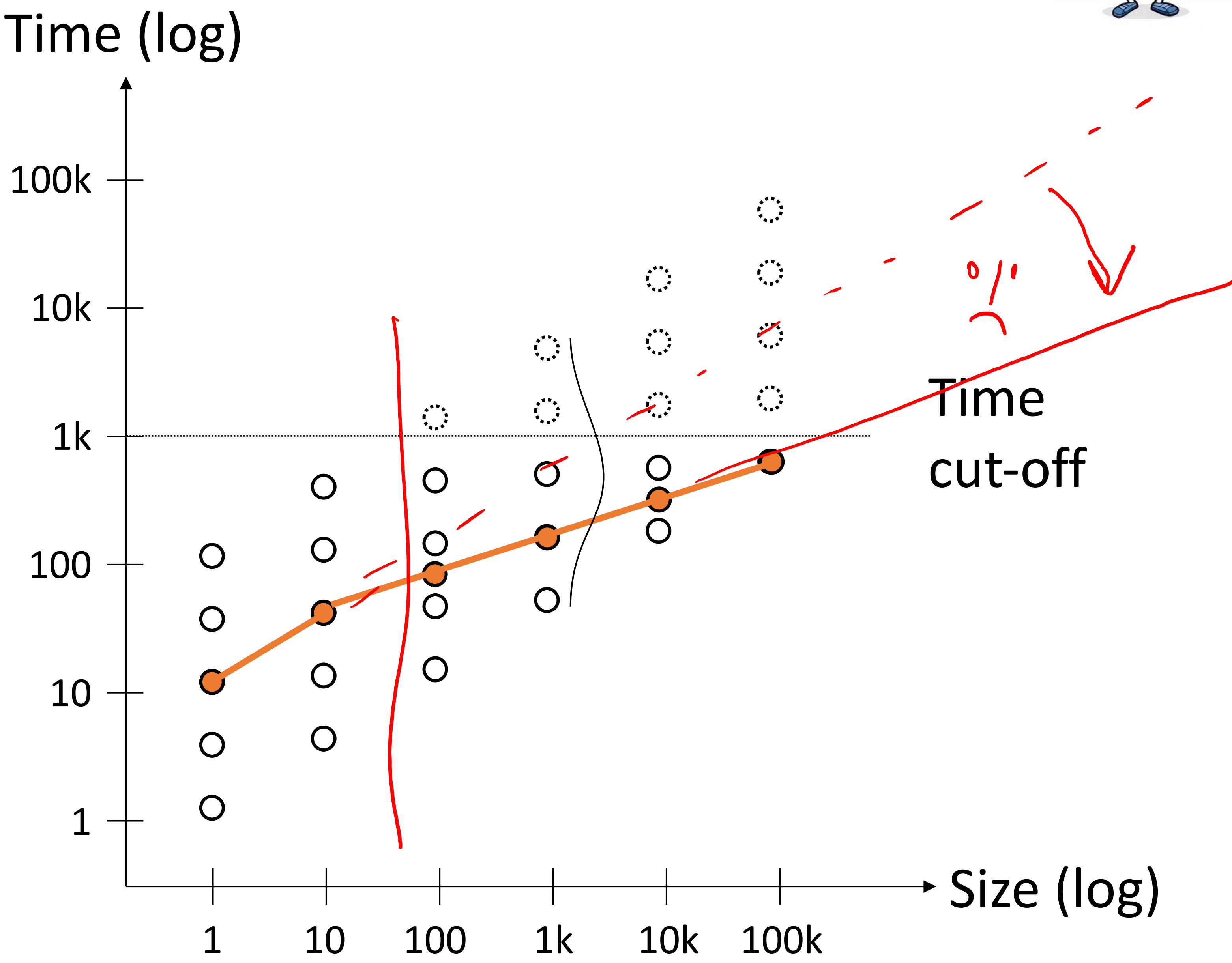


Question: How to deal with cut-offs when binning

- Option 1: What if we terminate experiments that take too long, and only average over the "seen points"?

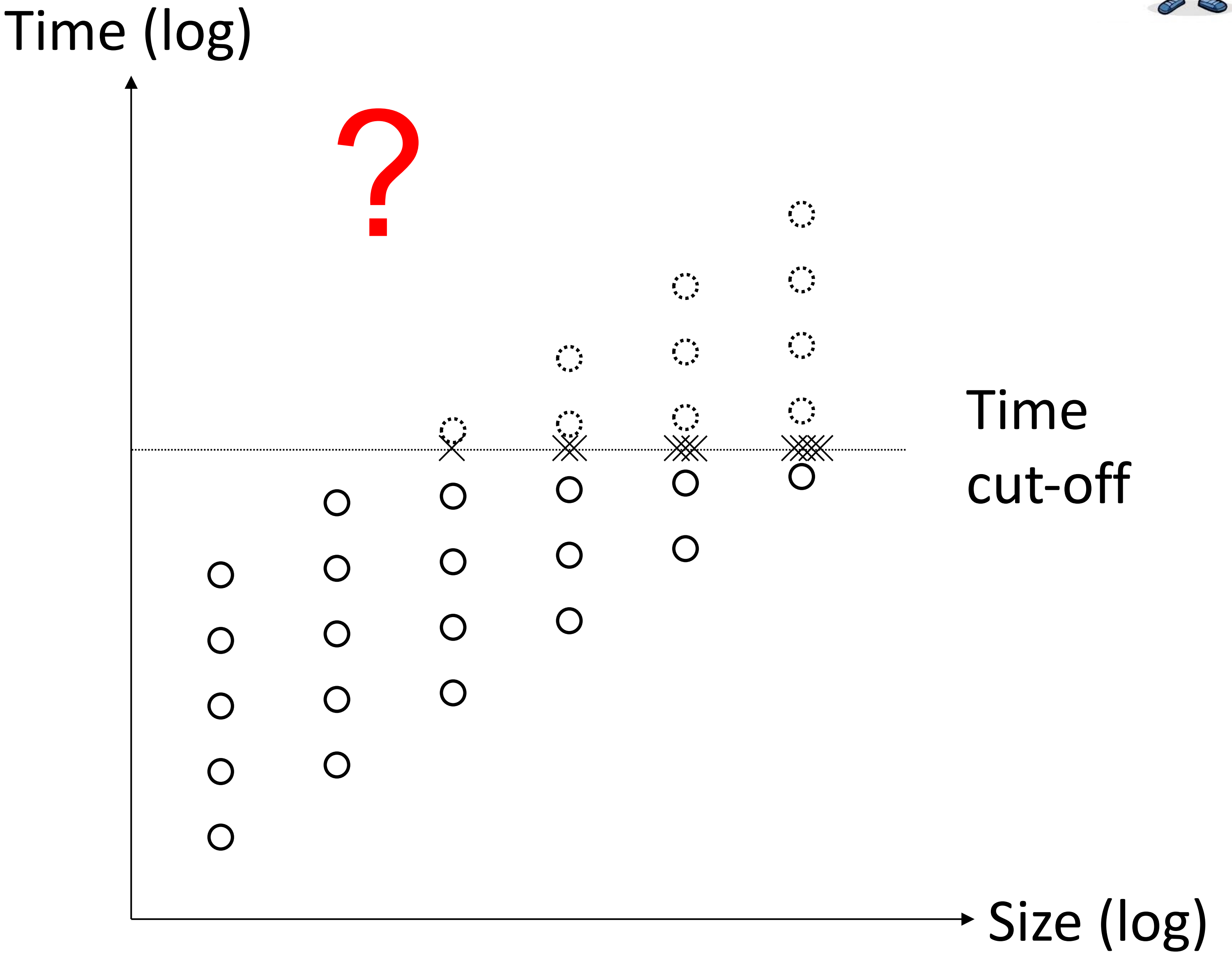
What would you do ?

We will discuss next class



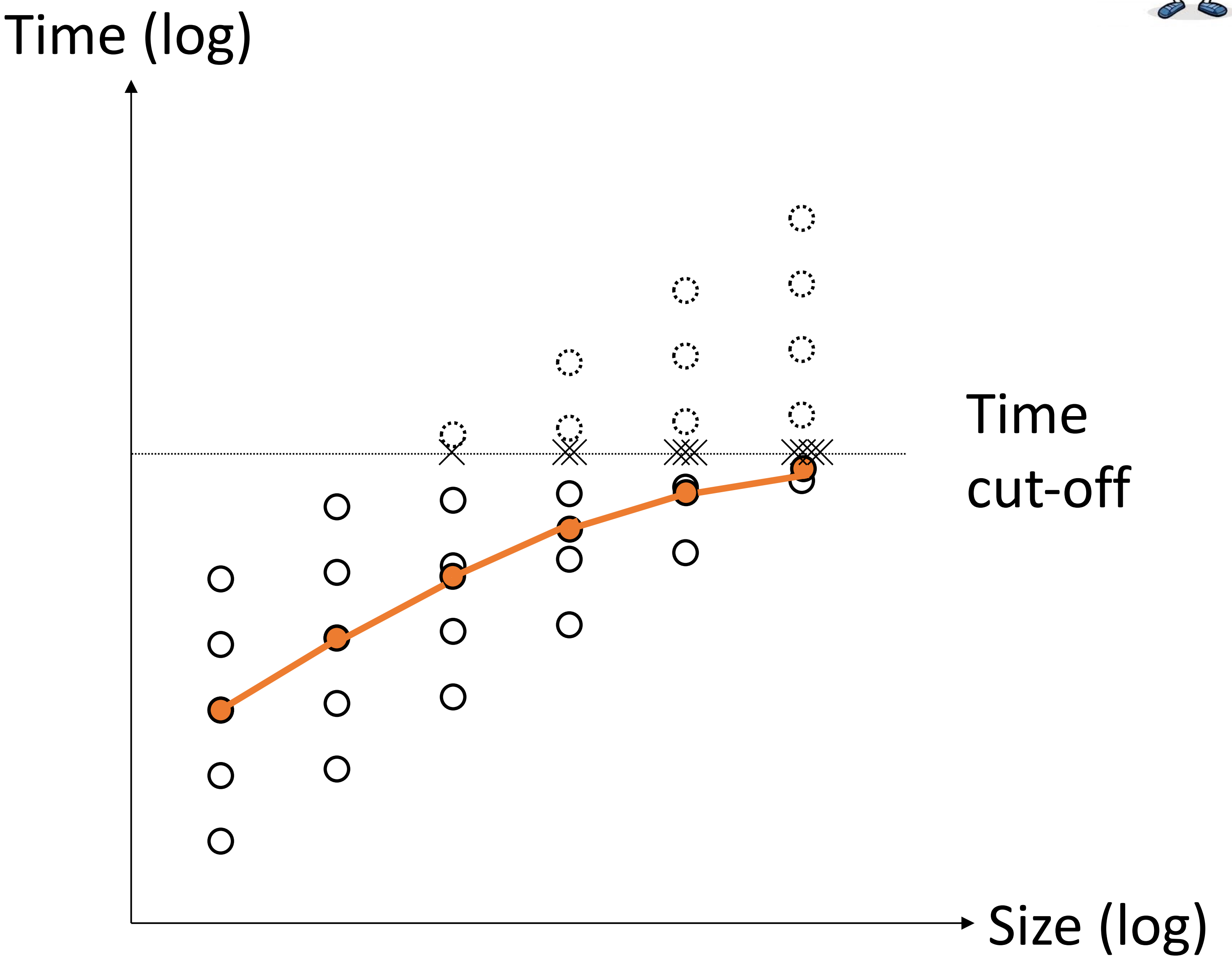
How to deal with cut-offs when binning: Option 1

- Option 1: What if we terminate experiments that take too long, and only average over the "seen points"?
- Option 2: What if we cut the points off and still use the cut-off points, and then average?



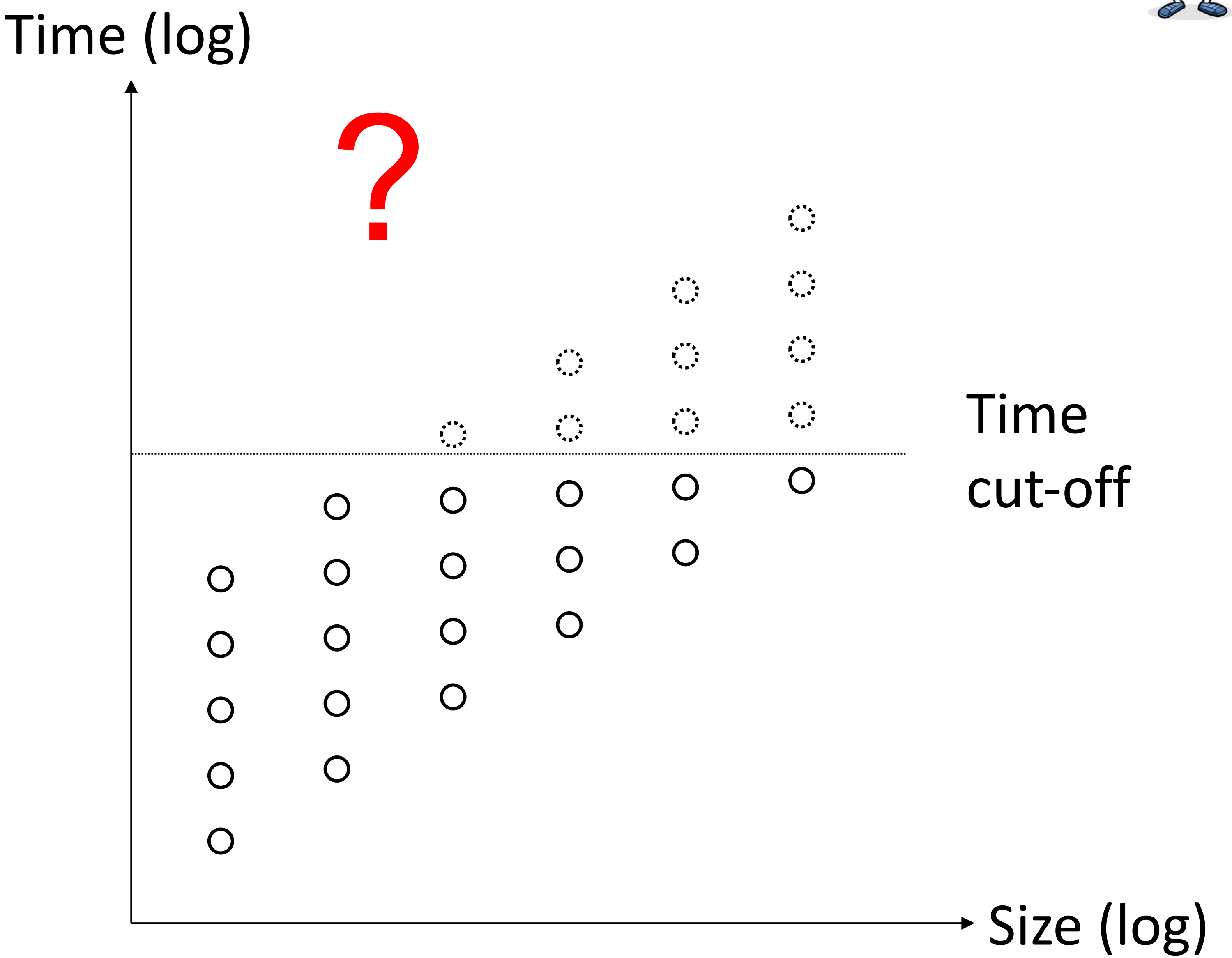
How to deal with cut-offs when binning: Option 1

- Option 1: What if we terminate experiments that take too long, and only average over the "seen points"?
- Option 2: What if we cut the points off and still use the cut-off points, and then average?



How to deal with cut-offs when binning: Option 2

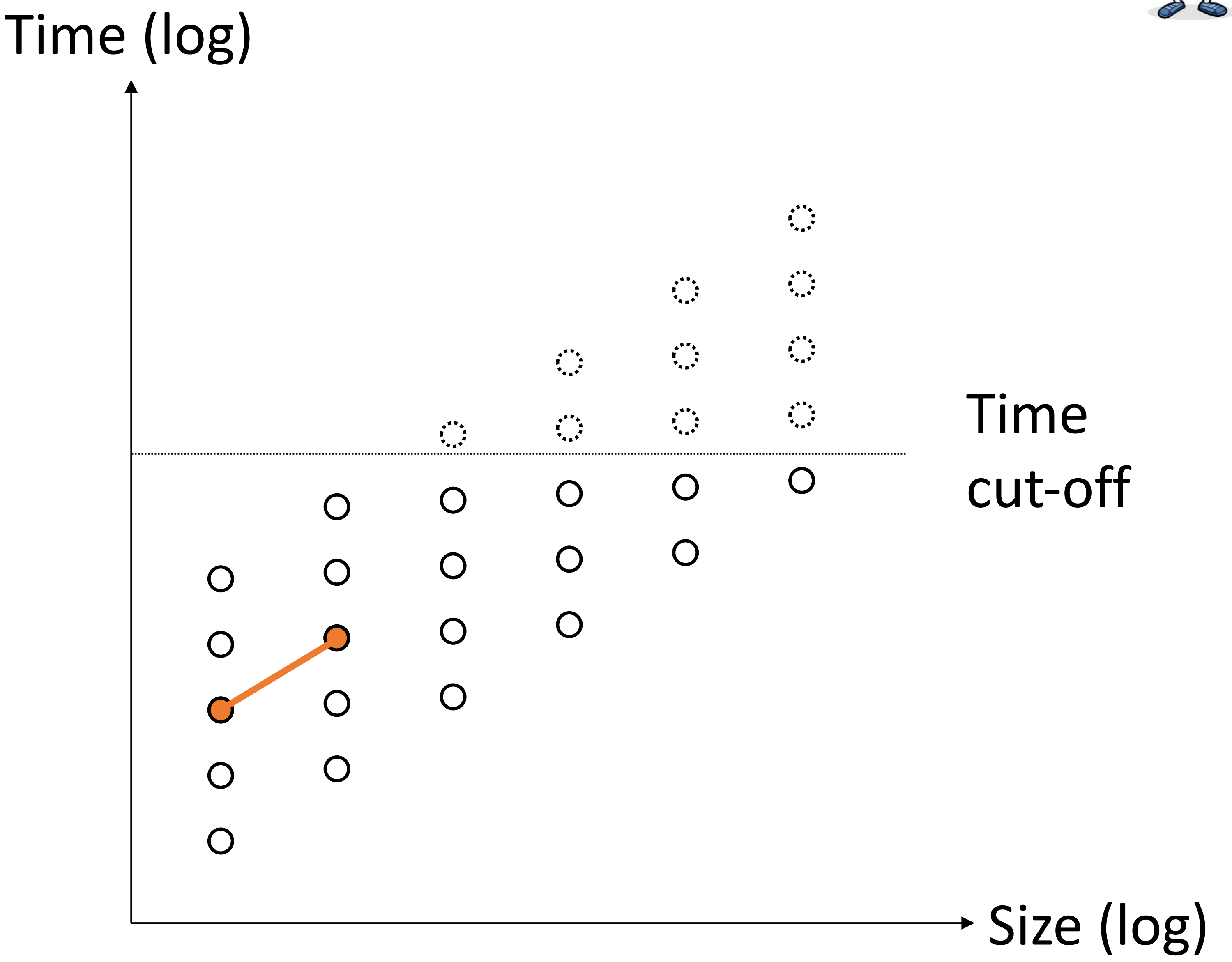
- Option 1: What if we terminate experiments that take too long, and only average over the "seen points"?
- Option 2: What if we cut the points off and still use the cut-off points, and then average?
- Option 3: What if we *only* use those sizes (x-axis) for which all experiments finish in time?



Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

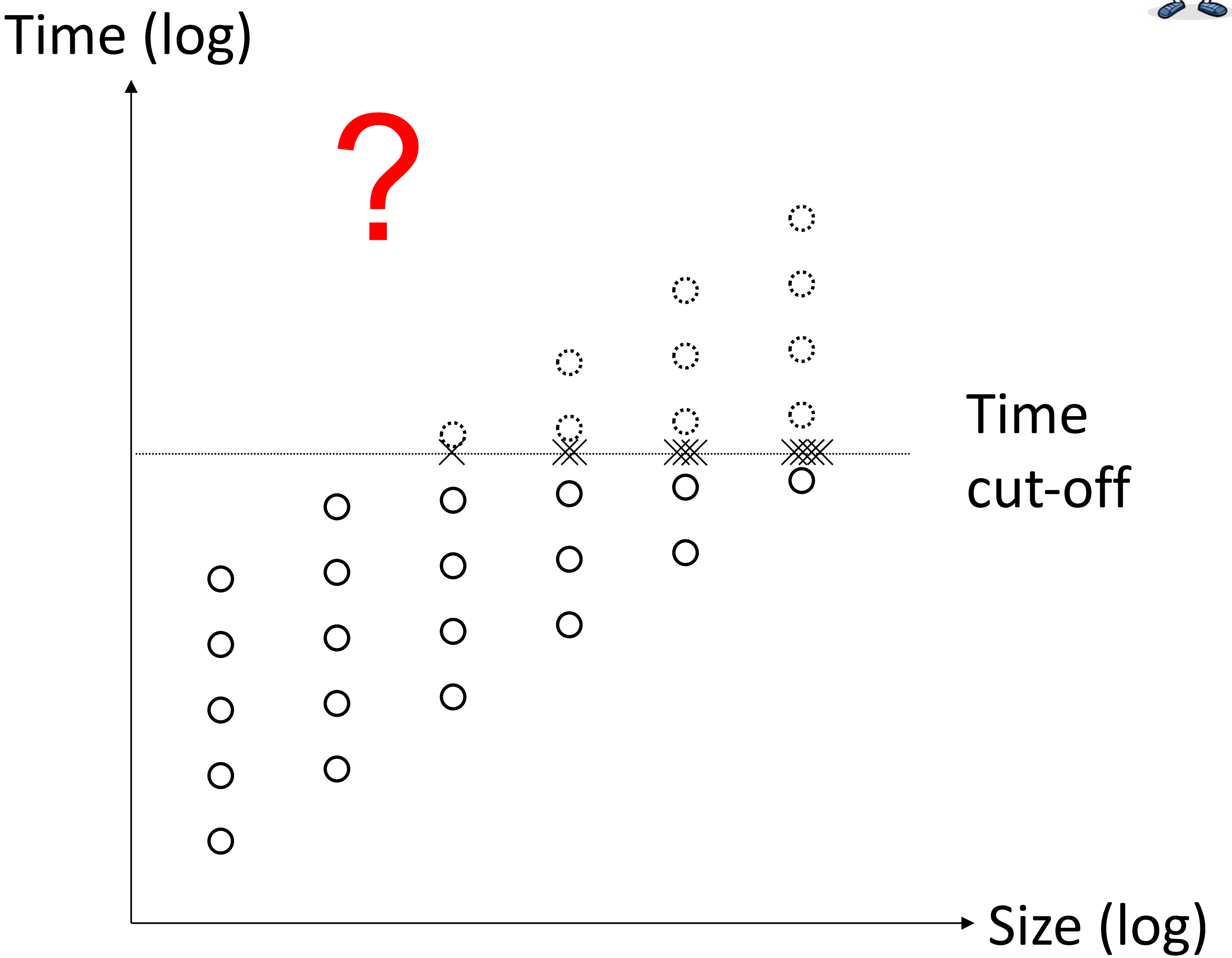
How to deal with cut-offs when binning: Option 2

- Option 1: What if we terminate experiments that take too long, and only average over the "seen points"?
- Option 2: What if we cut the points off and still use the cut-off points, and then average?
- Option 3: What if we *only* use those sizes (x-axis) for which all experiments finish in time?



How to deal with cut-offs when binning: Option 3

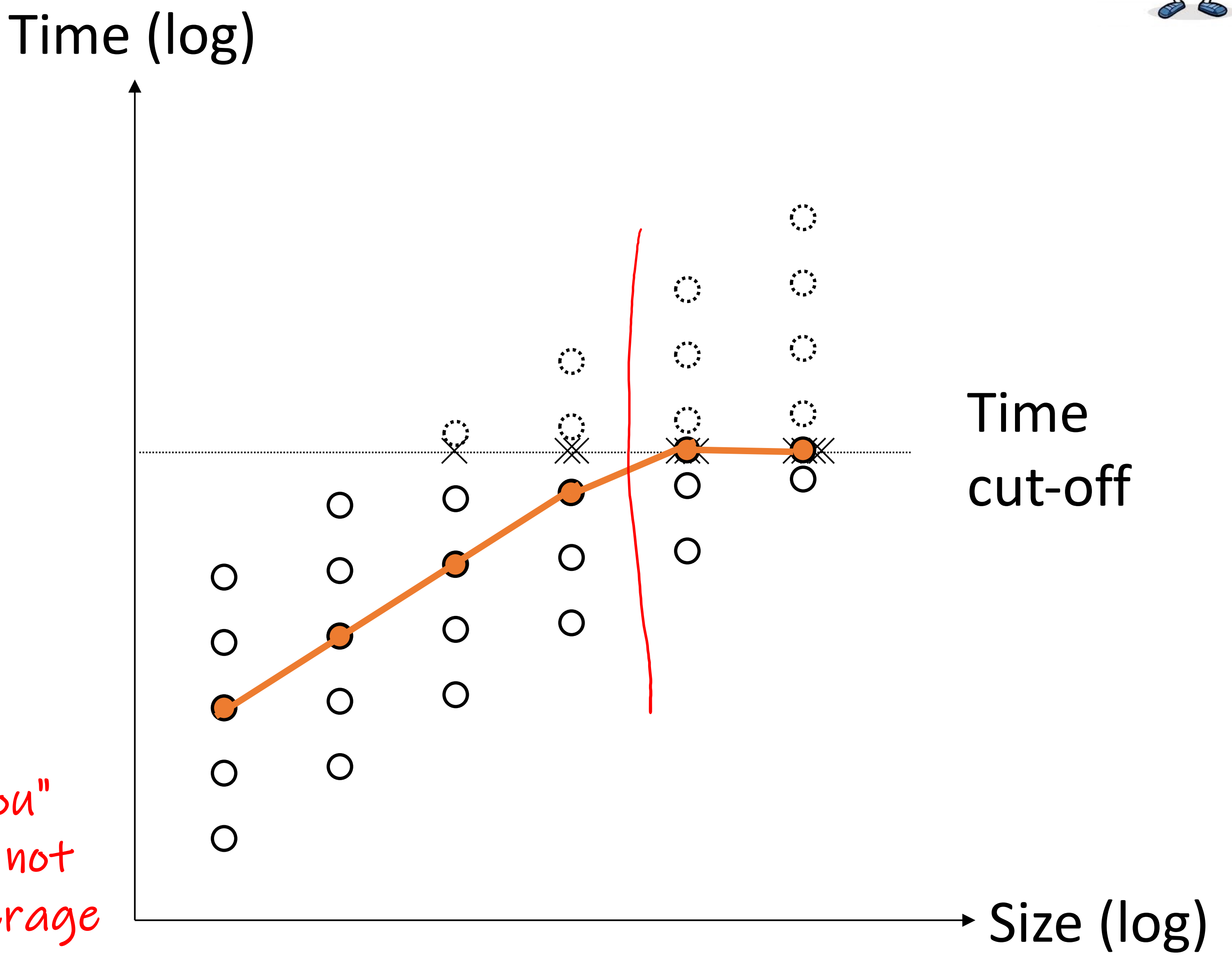
- Option 1: What if we terminate experiments that take too long, and only average over the "seen points"?
- Option 2: What if we cut the points off and still use the cut-off points, and then average?
- Option 3: What if we *only* use those sizes (x-axis) for which all experiments finish in time?
- Option 4: What if we take the median over all seen and cut-off points?



How to deal with cut-offs when binning: Option 3

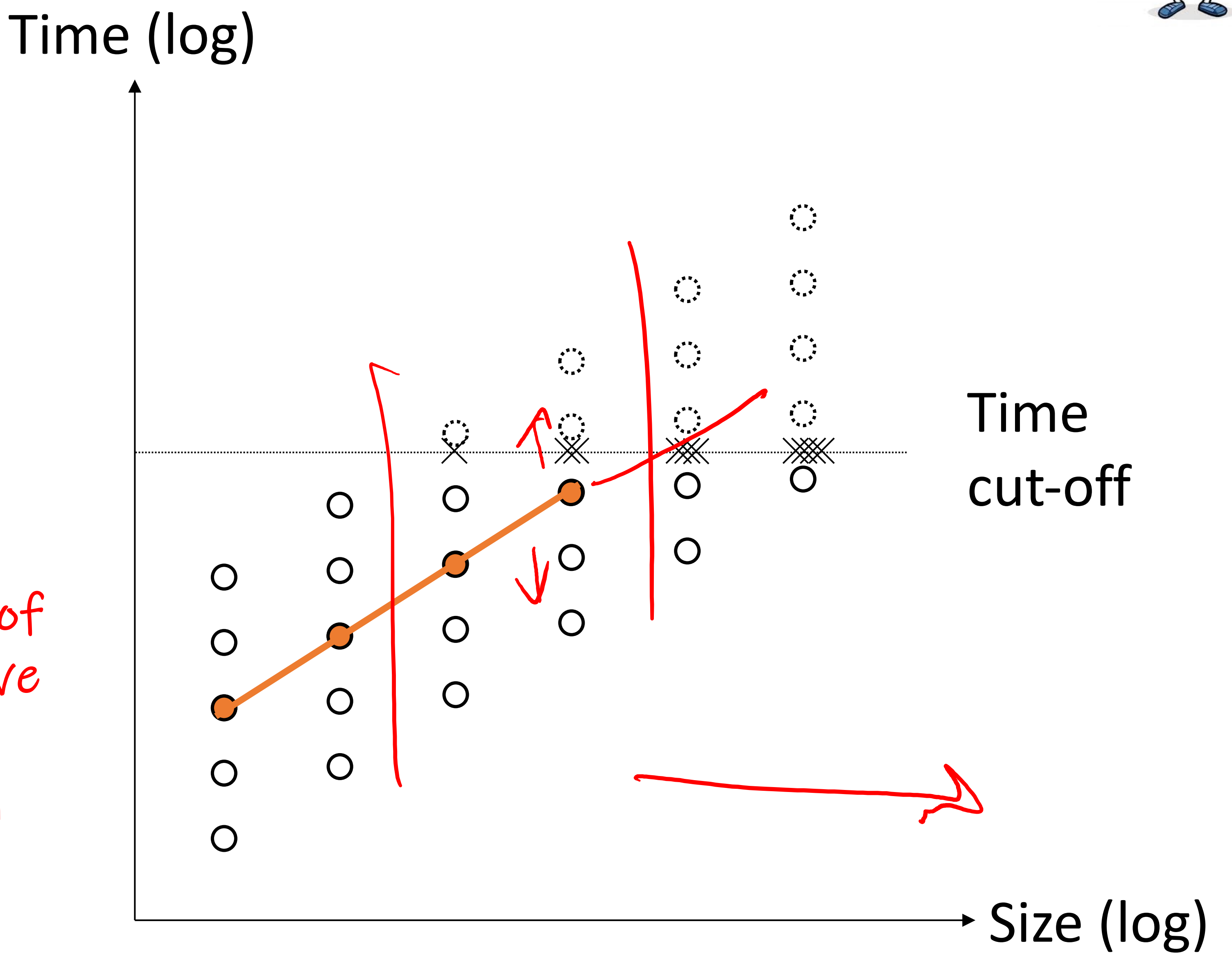
- Option 1: What if we terminate experiments that take too long, and only average over the "seen points"?
- Option 2: What if we cut the points off and still use the cut-off points, and then average?
- Option 3: What if we *only* use those sizes (x-axis) for which all experiments finish in time?
- Option 4: What if we take the median over all seen and cut-off points?

Notice the informal "semantics" of median: If more points are "above you" then you are pulled by their number, not by their distance (in contrast to average where distance is kind of a weight)



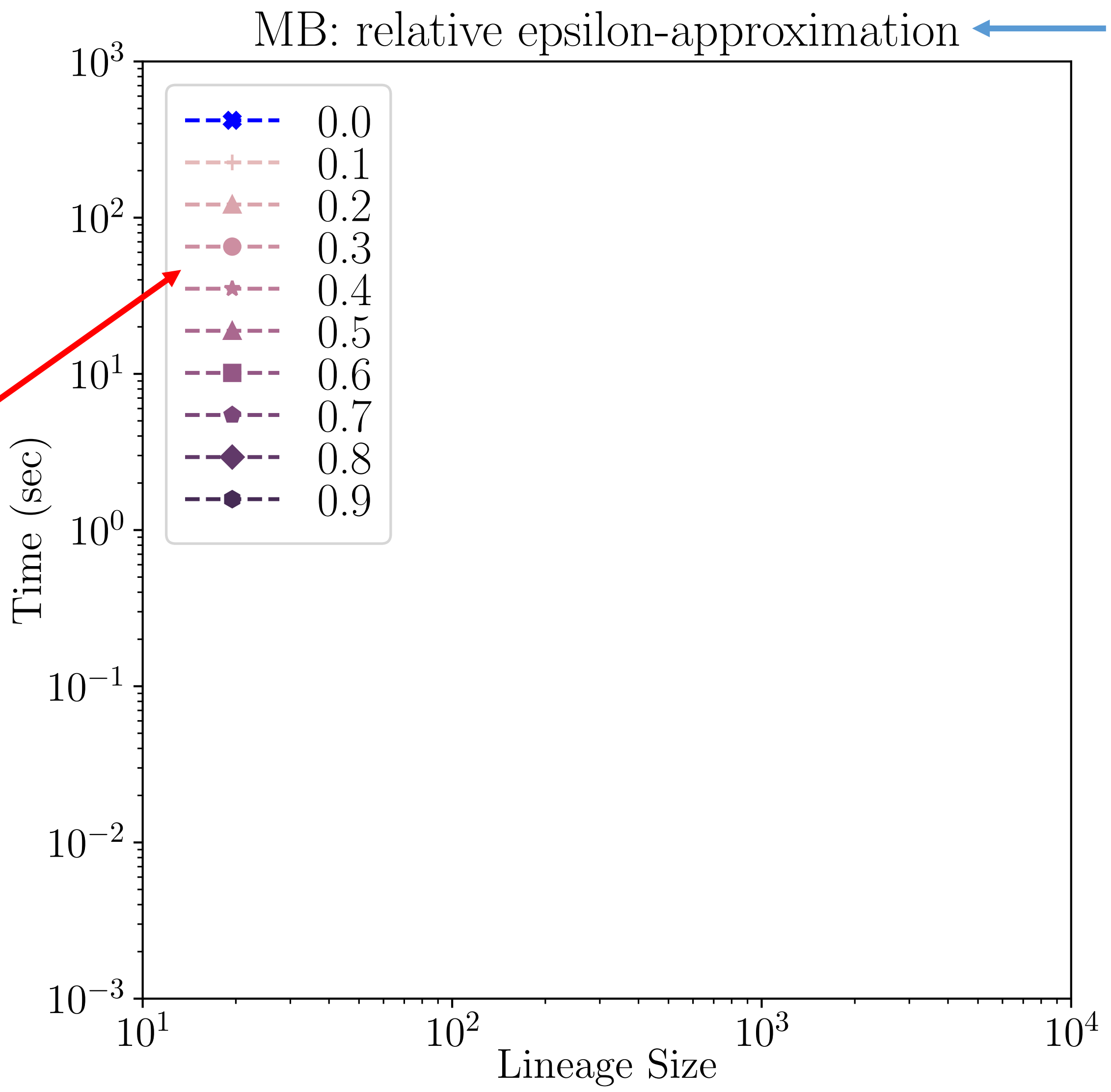
How to deal with cut-offs when binning: Suggestion

- Suggestion: Take the median over all seen and cut-off points, as long as there are <50% cut-off points!



Notice the informal "semantics" of median: If more points are "above you" then you are pulled by their number, not by their distance (in contrast to average where distance is kind of a weight)

Example: Experiments figures from [SIGMOD'19]



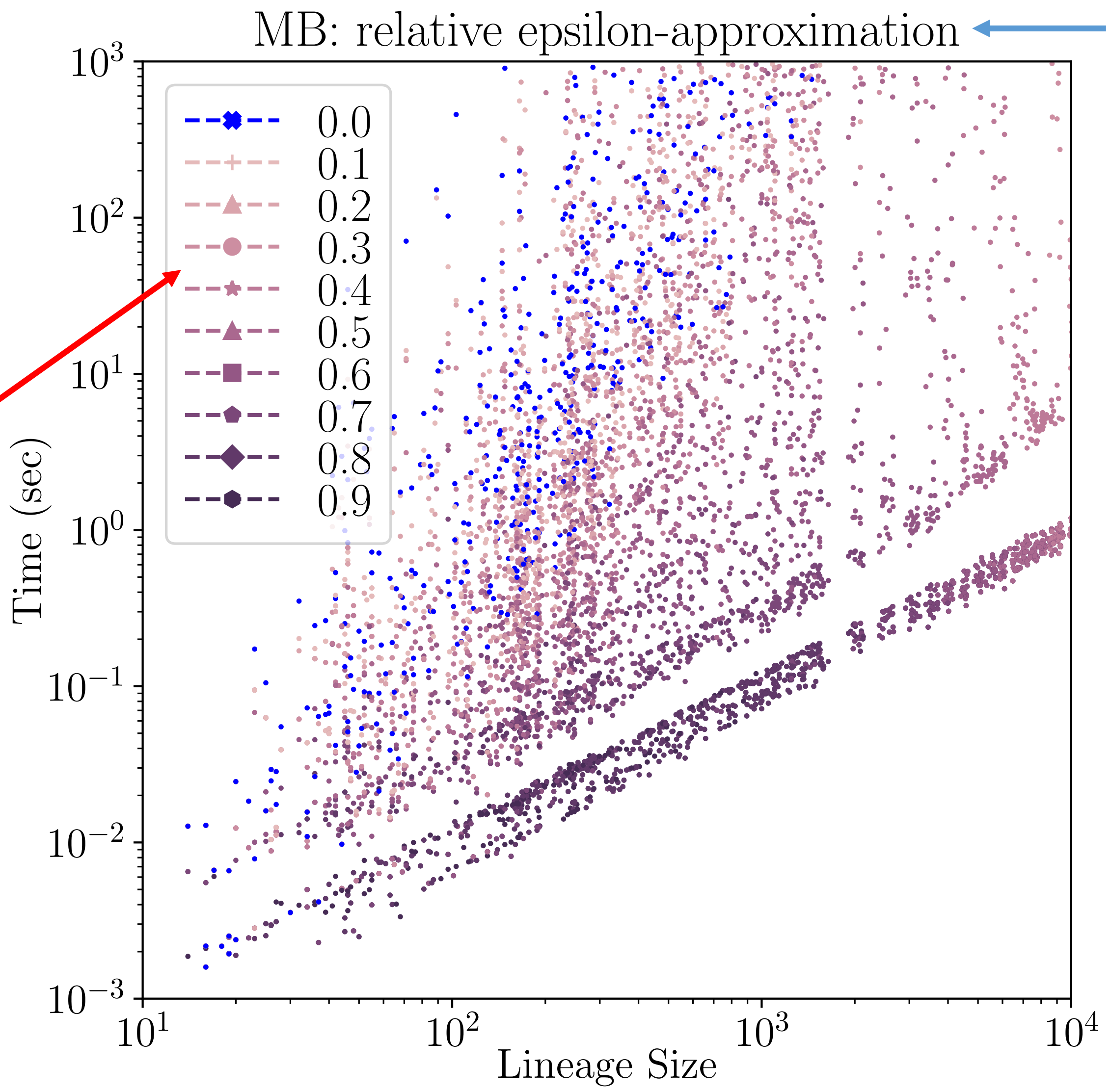
MB (prior): model-based
10 random bounds

Some type of error guarantees (smaller is better)

notice the log log scale!

Source: Van der Heuvel, Ivanov, Gatterbauer, Geerts, Theobald. Anytime approximation in probabilistic databases via scaled dissociations. SIGMOD 2019. https://doi.org/10.1145/3299869.3319900

Example: Experiments figures from [SIGMOD'19]



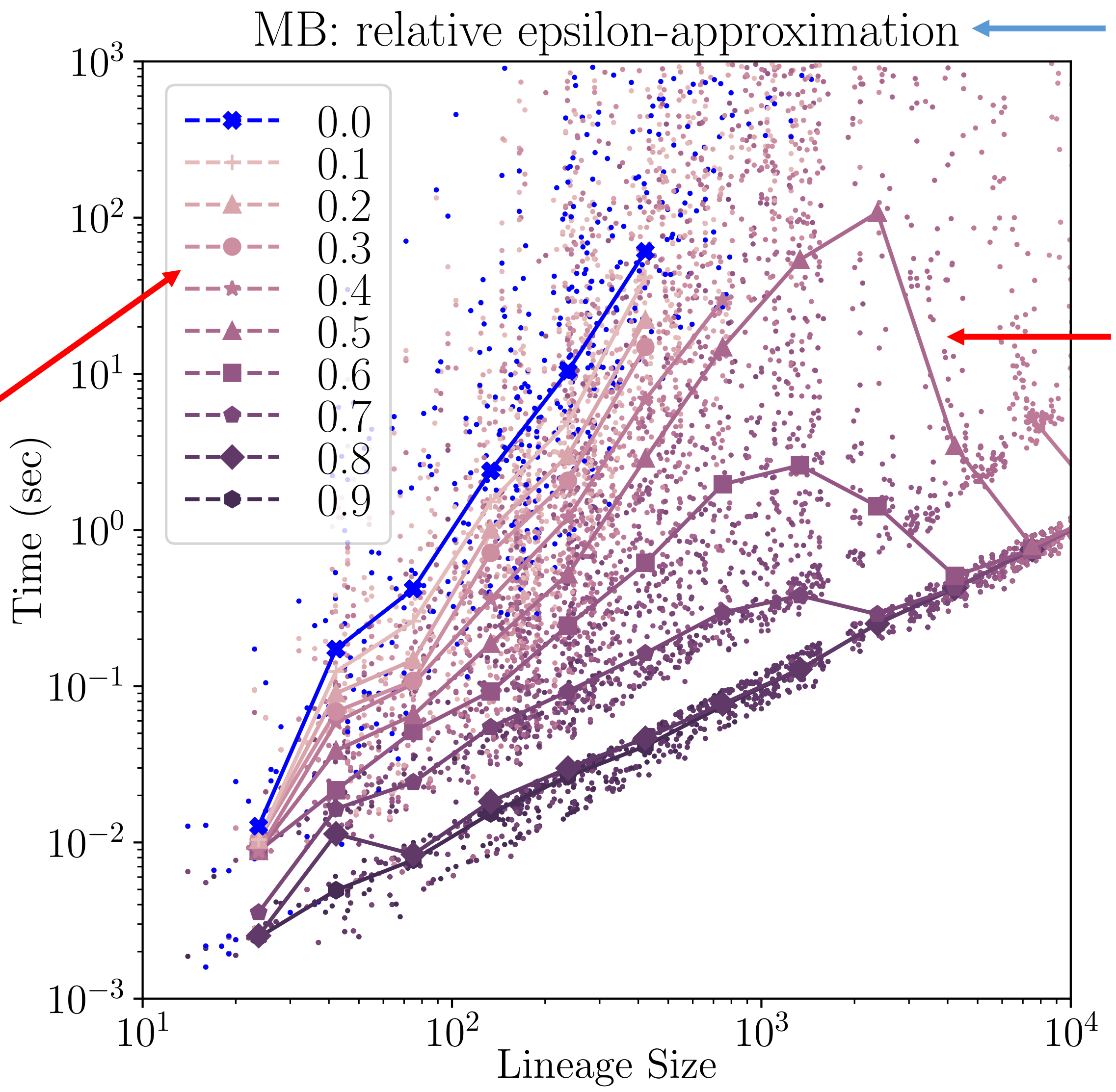
MB (prior): model-based
10 random bounds

Some type of error guarantees (smaller is better)

notice the log log scale!

Source: Van der Heuvel, Ivanov, Gatterbauer, Geerts, Theobald. Anytime approximation in probabilistic databases via scaled dissociations. SIGMOD 2019. https://doi.org/10.1145/3299869.3319900

Example: Experiments figures from [SIGMOD'19]



MB (prior): model-based
10 random bounds

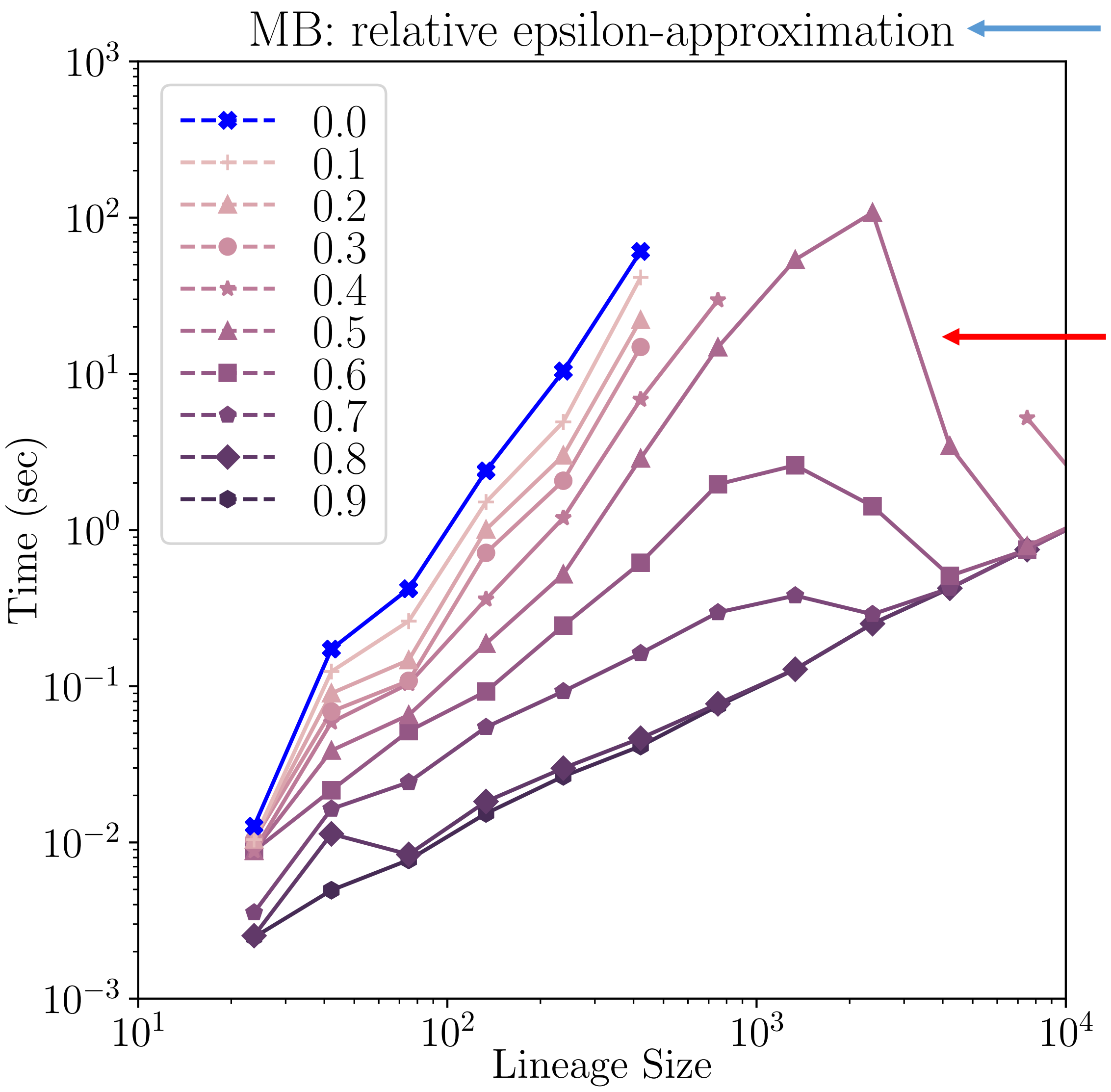
Median time to reach a certain error guarantee for fixed lin. size

Some type of error guarantees (smaller is better)

notice the log log scale!

Source: Van der Heuvel, Ivanov, Gatterbauer, Geerts, Theobald. Anytime approximation in probabilistic databases via scaled dissociations. SIGMOD 2019. https://doi.org/10.1145/3299869.3319900

Example: Experiments figures from [SIGMOD'19]

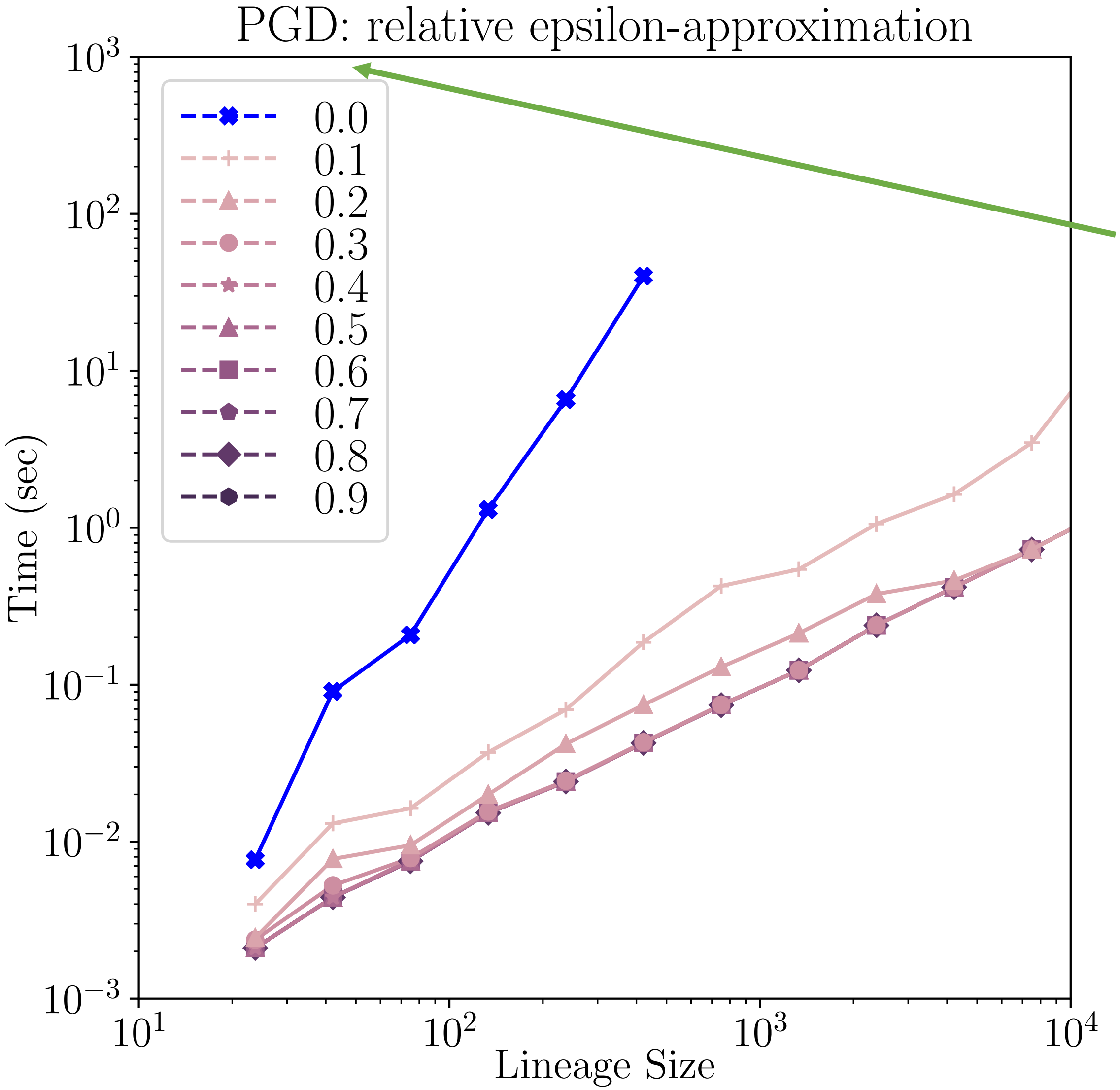


MB (prior): model-based
10 random bounds

Median time to reach a certain error guarantee for fixed lin. size

Source: Van der Heuvel, Ivanov, Gatterbauer, Geerts, Theobald. Anytime approximation in probabilistic databases via scaled dissociations. SIGMOD 2019. https://doi.org/10.1145/3299869.3319900

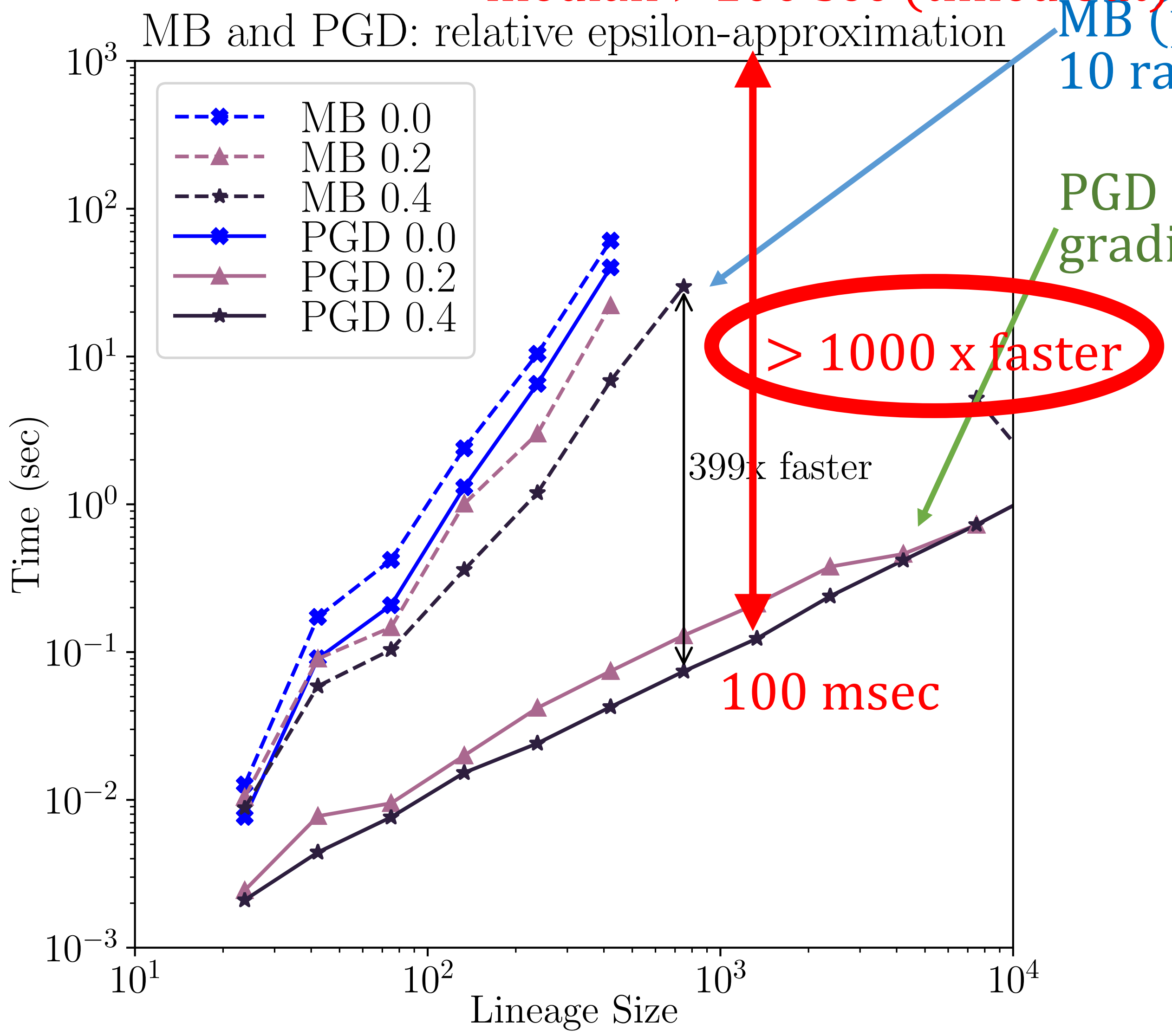
Example: Experiments figures from [SIGMOD'19]



PGD (our): projected gradient descent

Source: Van der Heuvel, Ivanov, Gatterbauer, Geerts, Theobald. Anytime approximation in probabilistic databases via scaled dissociations. SIGMOD 2019. https://doi.org/10.1145/3299869.3319900

Example: Experiments figures from [SIGMOD'19]



median >100 sec (timed out)

MB (prior): model-based
10 random bounds

PGD (our): projected
gradient descent

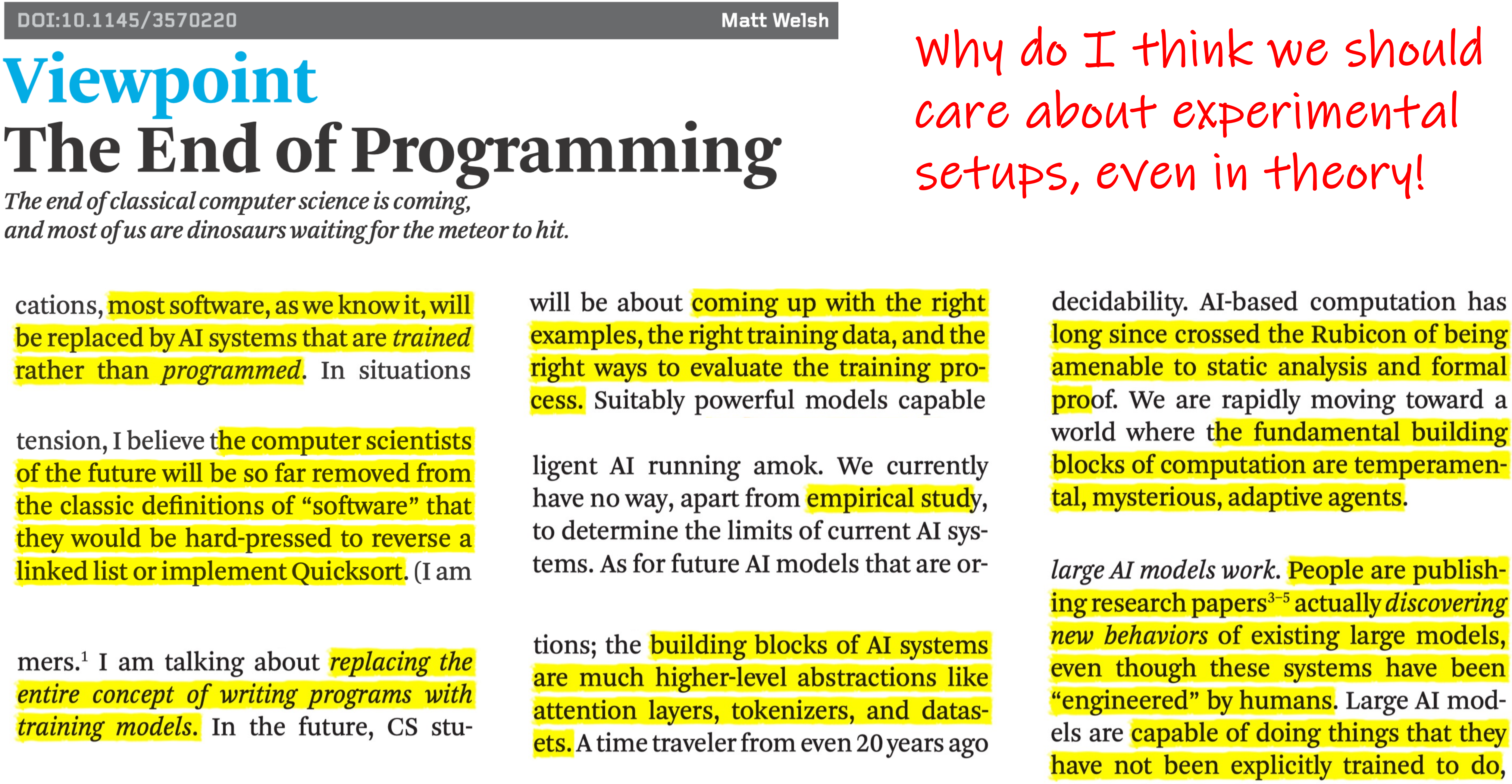
> 1000 x faster

100 msec

Take-away

- considerable speed-ups possible ☺

Source: Van der Heuvel, Ivanov, Gatterbauer, Geerts, Theobald. Anytime approximation in probabilistic databases via scaled dissociations. SIGMOD 2019. https://doi.org/10.1145/3299869.3319900



DOI:10.1145/3570220 **Matt Welsh**

Viewpoint

The End of Programming

The end of classical computer science is coming,
and most of us are dinosaurs waiting for the meteor to hit.

cations, most software, as we know it, will be replaced by AI systems that are *trained* rather than *programmed*. In situations

tension, I believe the computer scientists of the future will be so far removed from the classic definitions of "software" that they would be hard-pressed to reverse a linked list or implement Quicksort. (I am

mers.[1] I am talking about *replacing the entire concept of writing programs with training models*. In the future, CS stu-

will be about coming up with the right examples, the right training data, and the right ways to evaluate the training process. Suitably powerful models capable

ligent AI running amok. We currently have no way, apart from empirical study, to determine the limits of current AI systems. As for future AI models that are or-

tions; the building blocks of AI systems are much higher-level abstractions like attention layers, tokenizers, and datasets. A time traveler from even 20 years ago

decidability. AI-based computation has long since crossed the Rubicon of being amenable to static analysis and formal proof. We are rapidly moving toward a world where the fundamental building blocks of computation are temperamental, mysterious, adaptive agents.

large AI models work. People are publishing research papers[3–5] actually *discovering new behaviors* of existing large models, even though these systems have been "engineered" by humans. Large AI models are capable of doing things that they have not been explicitly trained to do,

Why do I think we should care about experimental setups, even in theory!

Source: Matt Welsh, CAMC 2023. https://cacm.acm.org/magazines/2023/1/267976-the-end-of-programming/fulltext, https://doi.org/10.1145/3570220

Outline: T1-U1: SQL

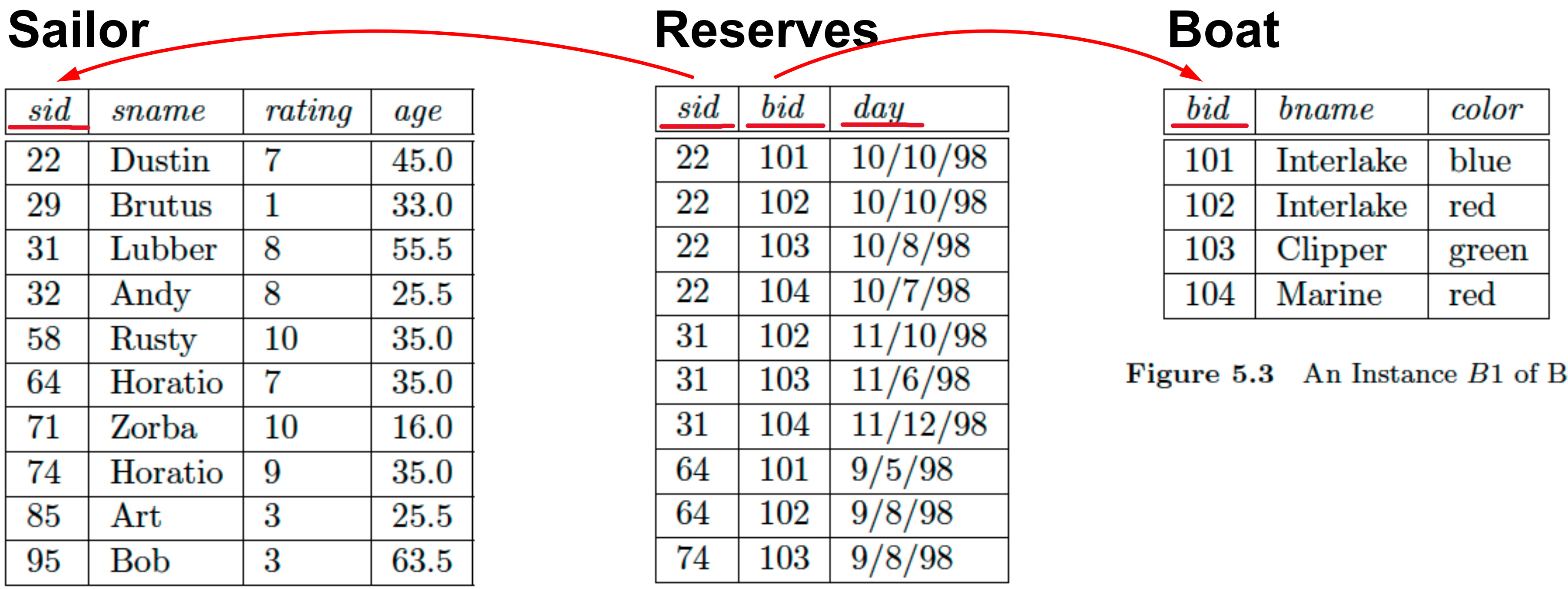
- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Union and Theta Joins
  - Nulls & Outer joins
  - Top-k
  - [Recursion: moved to T1-U4: Datalog]

Understanding nested queries with "Relational Diagrams"

The sailors database

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340



Sailor

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 5.1 An Instance *S*3 of Sailors

Reserves

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 5.2 An Instance *R*2 of Reserves

Boat

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 5.3 An Instance *B*1 of Boats

Schema and several of the following queries taken from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

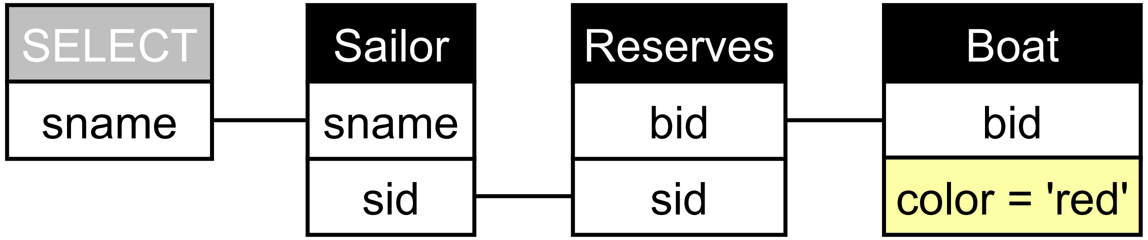
Nested query 1

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

Q: ?

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE S.sid IN
   (SELECT R.sid
   FROM Reserves R
   WHERE R.bid IN
      (SELECT B.bid
      FROM Boat B
      WHERE B.color = 'red'))
```



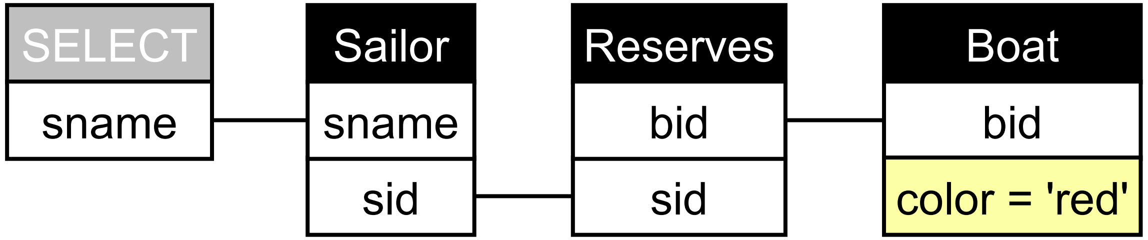
Nested query 1

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

Q: Find the names of sailors who have reserved a red boat.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE S.sid IN
  (SELECT R.sid
   FROM Reserves R
   WHERE R.bid IN
     (SELECT B.bid
      FROM Boat B
      WHERE B.color = 'red'))
```



$\{S.sname \mid \exists S \in Sailor.(\exists R \in Reserves.(R.sid=S.sid \wedge \exists B \in Boat.(B.bid=R.bid \wedge B.color='red')))\}$

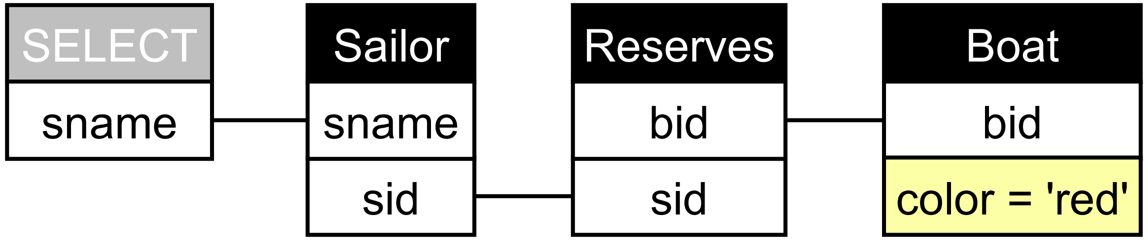
Nested query 1

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

Q: Find sailors who have reserved a red boat.

```sql
SELECT DISTINCT S.sname
FROM Sailor S
WHERE EXISTS
   (SELECT R.sid
   FROM Reserves R
   WHERE R.sid=S.sid
   AND EXISTS
      (SELECT B.bid
      FROM Boat B
      WHERE B.bid = R.bid
      AND B.color = 'red'))
```



This is an alternative way to write the previous query with EXISTS and correlated nested queries that matches the Relational Calculus below.

$$\{S.sname \mid \exists S \in Sailor.(\exists R \in Reserves.(R.sid=S.sid \land \exists B \in Boat.(B.bid=R.bid \land B.color='red')))\}$$

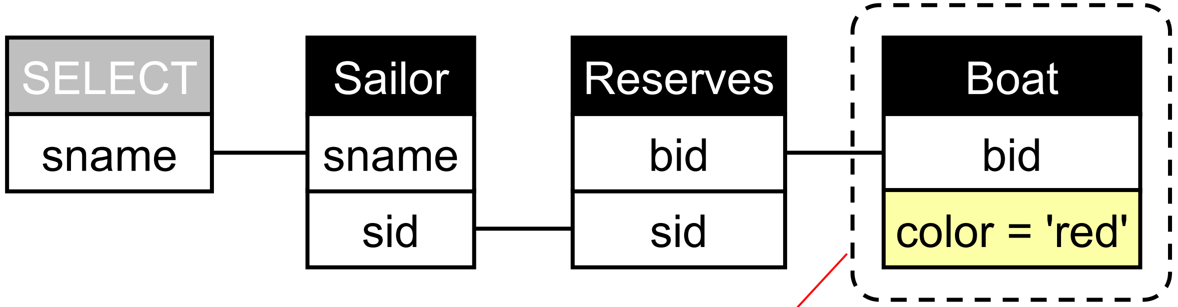
Nested query 2

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

Q: ?

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE EXISTS
   (SELECT R.sid
   FROM Reserves R
   WHERE R.sid=S.sid
   AND NOT EXISTS
      (SELECT B.bid
      FROM Boat B
      WHERE B.bid = R.bid
      AND B.color = 'red'))
```



Dashed lines represent not exists ∄

$$\{S.sname \mid \exists S \in Sailor.(\exists R \in Reserves.(R.sid=S.sid \land \nexists B \in Boat.(B.bid=R.bid \land B.color='red')))\}$$

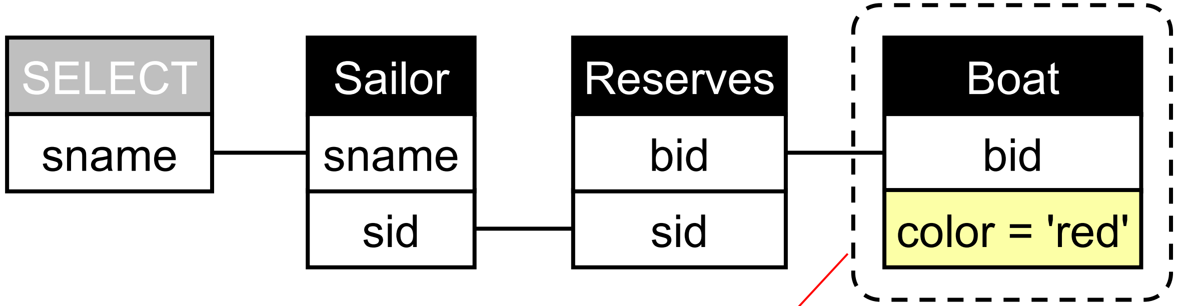
Nested query 2

Sailor (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boat (<u>bid</u>, bname, color)

340

Q: Find sailors who have reserved a boat that is not red.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE EXISTS
  (SELECT R.sid
  FROM Reserves R
  WHERE R.sid=S.sid
  AND NOT EXISTS
     (SELECT B.bid
     FROM Boat B
     WHERE B.bid = R.bid
     AND B.color = 'red'))
```



Dashed lines represent not exists ∄

They must have reserved at least one boat in another color. They can also have reserved a red boat in addition.

$$\{S.sname \mid \exists S \in Sailor.(\exists R \in Reserves.(R.sid=S.sid \wedge \nexists B \in Boat.(B.bid=R.bid \wedge B.color='red')))\}$$

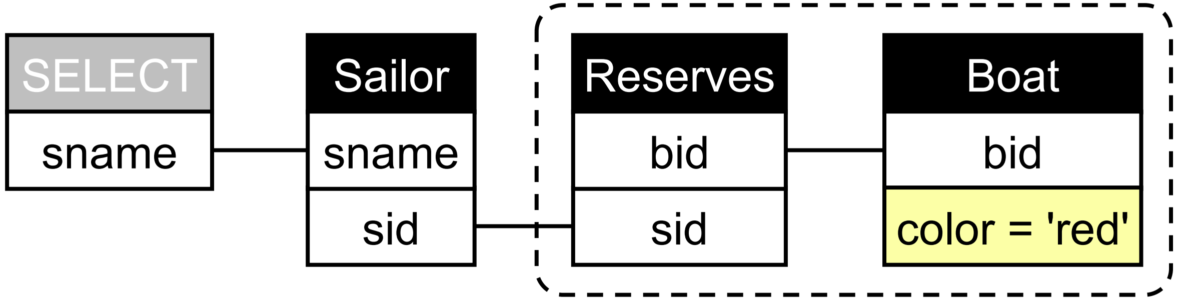
Nested query 3

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

Q: ?

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS
   (SELECT R.sid
   FROM Reserves R
   WHERE R.sid=S.sid
   AND EXISTS
      (SELECT B.bid
      FROM Boat B
      WHERE B.bid = R.bid
      AND B.color = 'red'))
```



$\{S.sname \mid \exists S \in Sailor.(\nexists R \in Reserves.(R.sid=S.sid \land \exists B \in Boat.(B.bid=R.bid \land B.color='red')))\}$

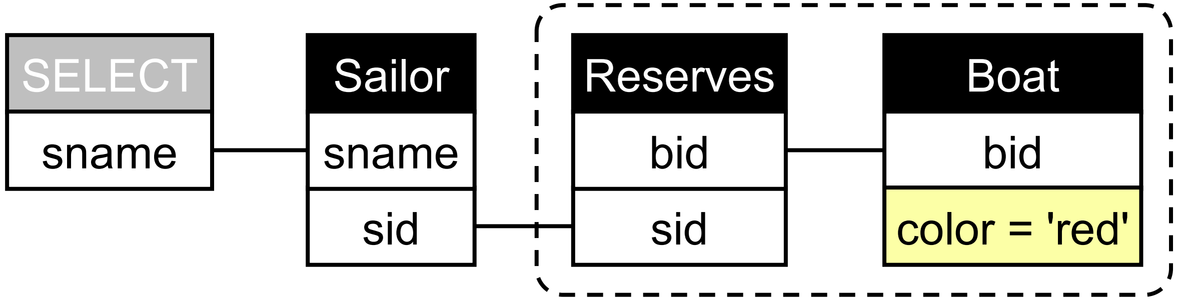
Nested query 3

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

Q: Find sailors who have not reserved a red boat.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS
   (SELECT R.sid
   FROM Reserves R
   WHERE R.sid=S.sid
   AND EXISTS
      (SELECT B.bid
      FROM Boat B
      WHERE B.bid = R.bid
      AND B.color = 'red'))
```



They can have reserved 0 or more boats in another color, but must not have reserved any red boat.

$$\{S.sname \mid \exists S \in Sailor.(\nexists R \in Reserves.(R.sid=S.sid \land \exists B \in Boat.(B.bid=R.bid \land B.color='red')))\}$$

Quiz: Dustin?

340

Sailor

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 5.1 An Instance *S3* of Sailors

Reserves

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 5.2 An Instance *R2* of Reserves

Boat

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 5.3 An Instance *B1* of Boats

Should Dustin be in the output of either of the two queries?

Q2: Find sailors who have reserved a boat that is not red.

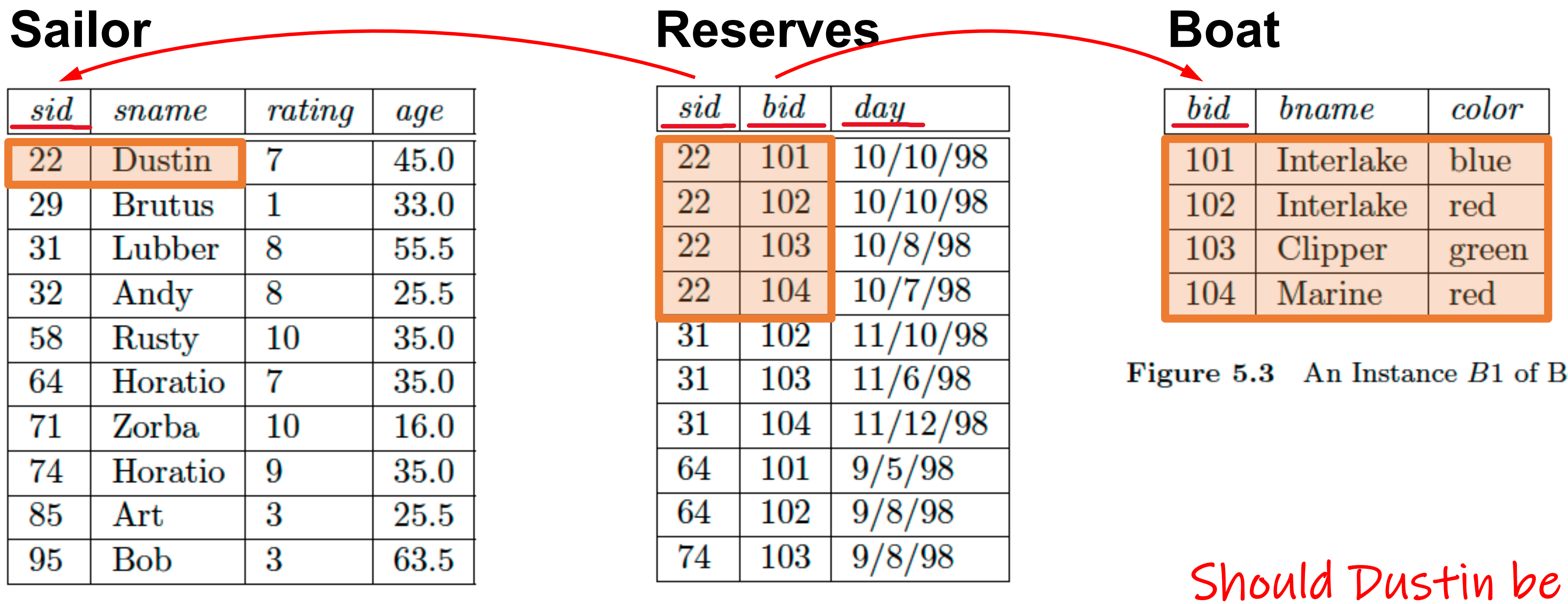
Q3: Find sailors who have not reserved a red boat.

?

Schema and several of the following queries taken from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

Quiz: Dustin?

340



Sailor

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 5.1 An Instance *S*3 of Sailors

Reserves

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 5.2 An Instance *R*2 of Reserves

Boat

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 5.3 An Instance *B*1 of Boats

Should Dustin be in the output of either of the two queries?

Q2: Find sailors who have reserved a boat that is not red.	Yes!
Q3: Find sailors who have not reserved a red boat.	No!

Schema and several of the following queries taken from: Ramakrishnan, Gehrke: Database management systems, 2nd ed (2000). http://pages.cs.wisc.edu/~dbbook/

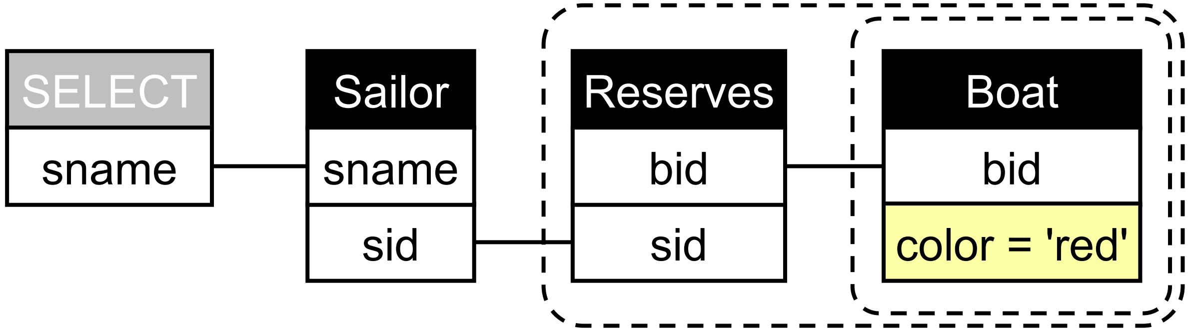
Nested query 4

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

Q: ?

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS
   (SELECT R.sid
   FROM Reserves R
   WHERE R.sid=S.sid
   AND NOT EXISTS
      (SELECT B.bid
      FROM Boat B
      WHERE B.bid = R.bid
      AND B.color = 'red'))
```



$$\{S.sname \mid \exists S \in Sailor.(\nexists R \in Reserves.(R.sid=S.sid \;\wedge\; \nexists B \in Boat.(B.bid=R.bid \wedge B.color='red')))\}$$

Nested query 4

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

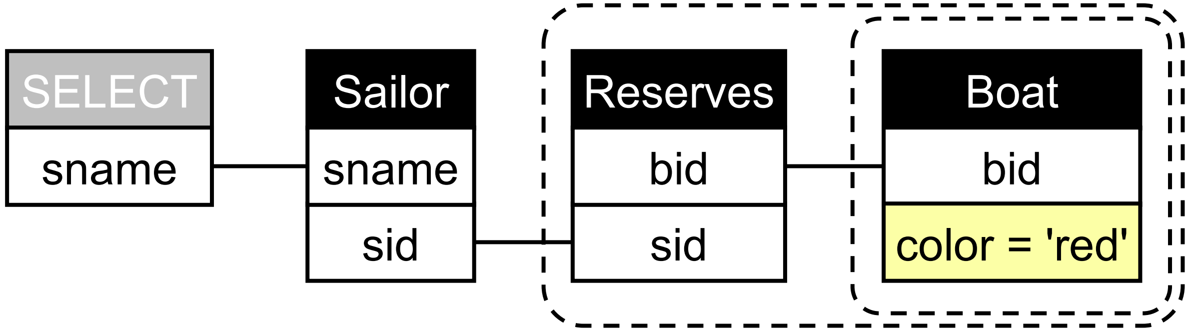
340

They can have reserved 0 or more boats in red, just no other color.

= Find sailors who have reserved only red boats

Q: Find sailors who have not reserved a boat that is not red.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS
   (SELECT R.sid
   FROM Reserves R
   WHERE R.sid=S.sid
   AND NOT EXISTS
      (SELECT B.bid
      FROM Boat B
      WHERE B.bid = R.bid
      AND B.color = 'red'))
```



They can have reserved 0 or more boats in red, just no other color.

$\{S.sname \mid \exists S \in Sailor.(\nexists R \in Reserves.(R.sid=S.sid \land \nexists B \in Boat.(B.bid=R.bid \land B.color='red')))\}$

Nested query 4 (universal)

They can have reserved 0 or more boats in red, just no other color.

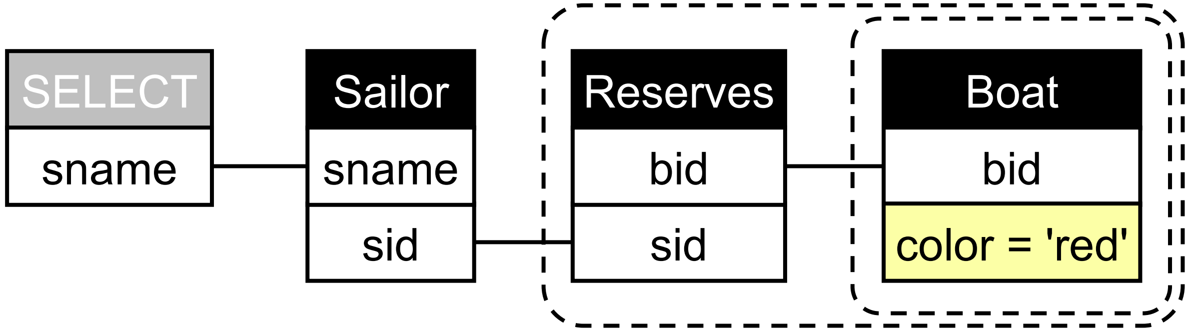
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

= Find sailors who have reserved only red boats

Q: Find sailors who have not reserved a boat that is not red.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS
   (SELECT R.sid
   FROM Reserves R
   WHERE R.sid=S.sid
   AND NOT EXISTS
      (SELECT B.bid
      FROM Boat B
      WHERE B.bid = R.bid
      AND B.color = 'red'))
```



They can have reserved 0 or more boats in red, just no other color.

$$\{S.sname \mid \exists S{\in}Sailor.(\forall R{\in}Reserves.(R.sid{=}S.sid \rightarrow \exists B{\in}Boat.(B.bid{=}R.bid \land B.color{=}\text{'red'})))\}$$

$$\{S.sname \mid \exists S{\in}Sailor.(\nexists R{\in}Reserves.(R.sid{=}S.sid \land \nexists B{\in}Boat.(B.bid{=}R.bid \land B.color{=}\text{'red'})))\}$$

Nested query 4 (another variant)

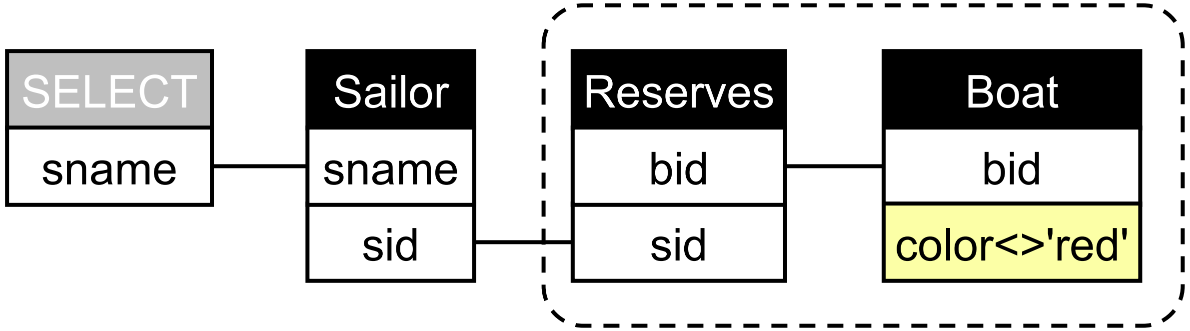
Sailor (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boat (<u>bid</u>, bname, color)

340

= Find sailors who have reserved only red boats

Q: Find sailors who have not reserved a boat that is not red.

```sql
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS
   (SELECT R.sid
   FROM Reserves R
   WHERE R.sid=S.sid
   AND EXISTS
      (SELECT B.bid
      FROM Boat B
      WHERE B.bid = R.bid
      AND B.color <> 'red'))
```



SELECT
sname

Sailor
sname
sid

Reserves
bid
sid

Boat
bid
color<>'red'

They can have reserved <u>0 or more boats in red</u>, just no other color.

Equivalence with previous variant only because of FK-PK constraint!

$$\{S.sname \mid \exists S \in Sailor.(\nexists R \in Reserves.(R.sid=S.sid \wedge \exists B \in Boat.(B.bid=R.bid \wedge B.color<>\text{'red'})))\}$$

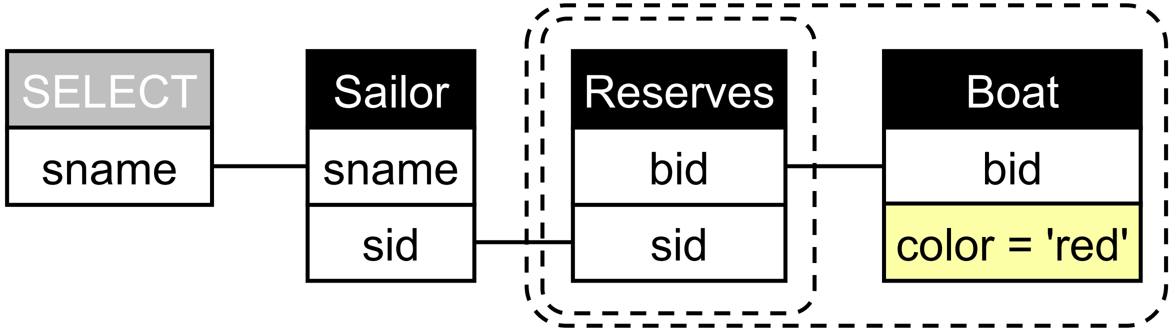
Nested query 5

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

Q: ?

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS
   (SELECT B.bid
   FROM Boat B
   WHERE B.color = 'red'
   AND NOT EXISTS
      (SELECT R.bid
      FROM Reserves R
      WHERE R.bid = B.bid
      AND R.sid = S.sid))
```



{S.sname | ∃S∈Sailor.(∄B∈Boat.(B.color='red' ∧ ∄R∈Reserves.(B.bid=R.bid ∧ R.sid=S.sid)))}

Nested query 5

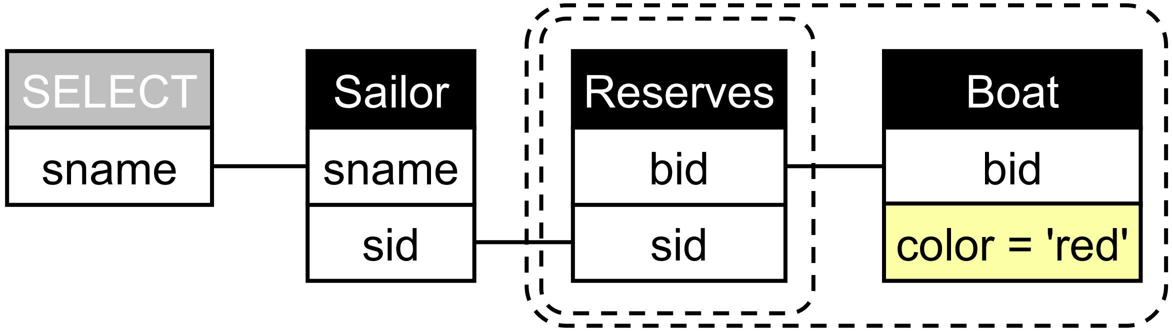
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

= Find sailors who have reserved all red boats

Q: Find sailors so there is no red boat that is not reserved by the sailor.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS
   (SELECT B.bid
   FROM Boat B
   WHERE B.color = 'red'
   AND NOT EXISTS
      (SELECT R.bid
      FROM Reserves R
      WHERE R.bid = B.bid
      AND R.sid = S.sid))
```



I don't know of a way to write that query with IN instead of EXISTS and without an explicit cross product between sailors and red boats. (More on that in a moment and also later when we discuss this query in relational algebra.)

$\{S.sname \mid \exists S \in Sailor.(\nexists B \in Boat.(B.color='red' \land \nexists R \in Reserves.(B.bid=R.bid \land R.sid=S.sid)))\}$

Nested query 5 (universal)

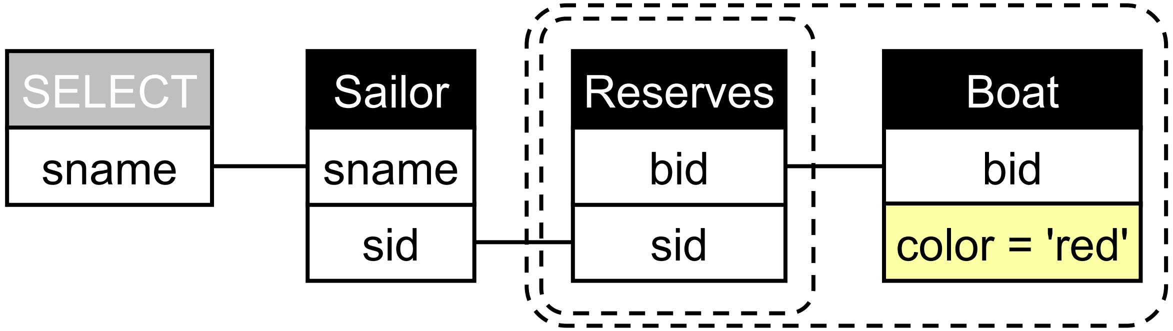
Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

340

= Find sailors who have reserved all red boats

Q: Find sailors so there is no red boat that is not reserved by the sailor.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS
   (SELECT B.bid
   FROM Boat B
   WHERE B.color = 'red'
   AND NOT EXISTS
      (SELECT R.bid
      FROM Reserves R
      WHERE R.bid = B.bid
      AND R.sid = S.sid))
```



I don't know of a way to write that query with IN instead of EXISTS and without an explicit cross product between sailors and red boats. (More on that in a moment and also later when we discuss this query in relational algebra.)

$\{S.sname \mid \exists S \in Sailor.(\forall B \in Boat.(B.color='red' \rightarrow \exists R \in Reserves.(B.bid=R.bid \wedge R.sid=S.sid))))\}$

$\{S.sname \mid \exists S \in Sailor.(\nexists B \in Boat.(B.color='red' \wedge \nexists R \in Reserves.(B.bid=R.bid \wedge R.sid=S.sid)))\}$

Nested query 5 (w/o correlation)

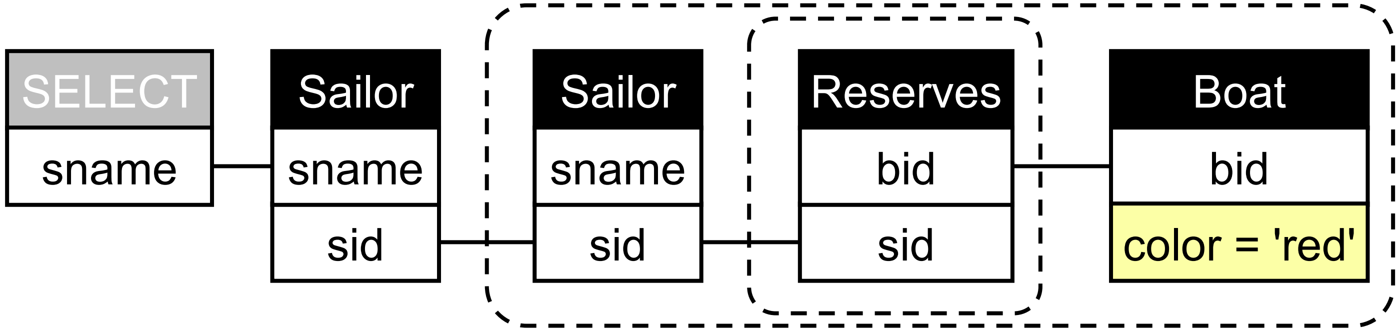
Sailor (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boat (<u>bid</u>, bname, color)

340

= Find sailors who have reserved all red boats

Q: Find sailors so there is no red boat that is not reserved by the sailor.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE S.sid NOT IN
  (SELECT S2.sid
  FROM Sailor S2, Boat B
  WHERE B.color = 'red'
  AND (S2.sid, B.bid) NOT IN
    (SELECT R.sid, R.bid
    FROM Reserves R))
```



$\{S.\text{sname} \mid \exists S{\in}\text{Sailor}.(\forall S2{\in}\text{Sailor}, \forall B{\in}\text{Boat}.(B.\text{color}=\text{'red'} \wedge S2.\text{sid}=S.\text{sid} \rightarrow \exists R{\in}\text{Reserves}.(B.\text{bid}=R.\text{bid} \wedge R.\text{sid}=S2.\text{sid}))))\}$

$\{S.\text{sname} \mid \exists S{\in}\text{Sailor}.(\nexists S2{\in}\text{Sailor}, \nexists B{\in}\text{Boat}.(B.\text{color}=\text{'red'} \wedge S2.\text{sid}=S.\text{sid} \;\wedge \nexists R{\in}\text{Reserves}.(B.\text{bid}=R.\text{bid} \wedge R.\text{sid}=S.\text{sid})))\}$

Nested query 5 (w/o correlation)

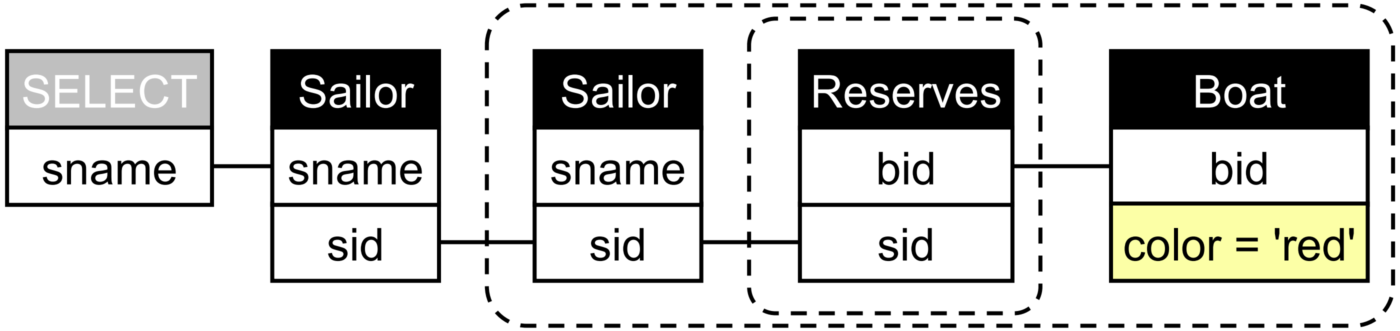
Sailor (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boat (<u>bid</u>, bname, color)

340

= Find sailors who have reserved all red boats

Q: Find sailors so there is no red boat that is not reserved by the sailor.

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS
   (SELECT *
   FROM Sailor S2, Boat B
   WHERE B.color = 'red'
   AND S.sid = S2.sid
   AND NOT EXISTS
      (SELECT *
      FROM Reserves R
      WHERE B.bid = R.bid
      AND S2.sid = R.sid))
```



$\{S.sname \mid \exists S \in Sailor.(\forall S2 \in Sailor, \forall B \in Boat.(B.color='red' \wedge S2.sid=S.sid \rightarrow \exists R \in Reserves.(B.bid=R.bid \wedge R.sid=S2.sid))))\}$

$\{S.sname \mid \exists S \in Sailor.(\nexists S2 \in Sailor, \nexists B \in Boat.(B.color='red' \wedge S2.sid=S.sid \ \wedge \nexists R \in Reserves.(B.bid=R.bid \wedge R.sid=S.sid)))\}$

Towards SQL patterns

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

SQL

Sailors who **have not** reserved a red boat

```sql
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Reserves R, Boat B
  WHERE R.sid = S.sid
  AND R.bid = B.bid
  AND B.color = 'red')
```

Sailors who reserved **only** red boats

```sql
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Reserves R
  WHERE R.sid = S.sid
  AND NOT EXISTS(
    SELECT *
    FROM Boat B
    WHERE R.bid = B.bid
    AND B.color = 'red'))
```

Sailors who reserved **all** red boats

```sql
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Boat B
  WHERE B.color = 'red'
  AND NOT EXISTS(
    SELECT *
    FROM Reserves R
    WHERE R.bid = B.bid
    AND R.sid = S.sid))
```

Towards SQL patterns

Sailor (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boat (<u>bid</u>, bname, color)

	Sailors who **have not** reserved a red boat	Sailors who reserved **only** red boats	Sailors who reserved **all** red boats

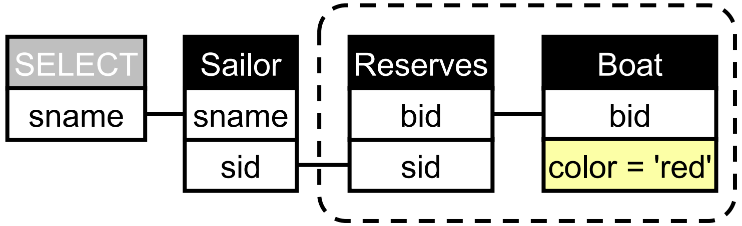
SQL

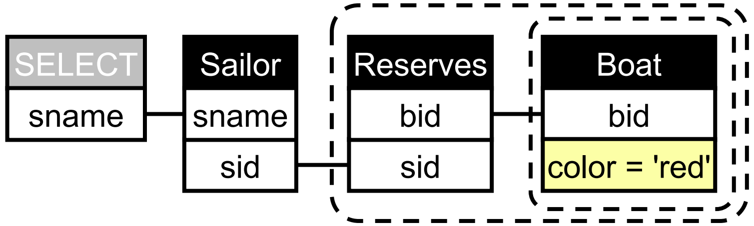
```sql
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Reserves R, Boat B
  WHERE R.sid = S.sid
  AND R.bid = B.bid
  AND B.color = 'red')
```

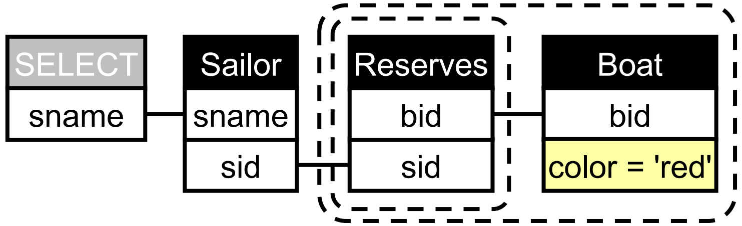
```sql
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Reserves R
  WHERE R.sid = S.sid
  AND NOT EXISTS(
    SELECT *
    FROM Boat B
    WHERE R.bid = B.bid
    AND B.color = 'red'))
```

```sql
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Boat B
  WHERE B.color = 'red'
  AND NOT EXISTS(
    SELECT *
    FROM Reserves R
    WHERE R.bid = B.bid
    AND R.sid = S.sid))
```

RD







Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

	not	**only**	**all**
Sailors renting boats	have not reserved a red boat	reserved only red boats	reserved all red boats

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

Student (sid, sname)
Takes (sid, cid, semester)
Course (cid, cname, department)

	not	**only**	**all**
Sailors renting boats	have not reserved a red boat	reserved only red boats	reserved all red boats
Students taking classes	took no art class	took only art classes	took all art classes

Sailor (sid, sname, rating, age)
Reserves (sid, bid, day)
Boat (bid, bname, color)

Student (sid, sname)
Takes (sid, cid, semester)
Course (cid, cname, department)

Actor (aid, aname)
Plays (aid, mid, role)
Movie (mid, mname, director)

	not	**only**	**all**
Sailors renting boats	have not reserved a red boat	reserved only red boats	reserved all red boats
Students taking classes	took no art class	took only art classes	took all art classes
Actors playing in movies	did not play in a Hitchcock movie	played only Hitchcock movies	played in all Hitchcock movies

Sailor (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boat (<u>bid</u>, bname, color)

Student (<u>sid</u>, sname)
Takes (<u>sid, cid, semester</u>)
Course (<u>cid</u>, cname, department)

Actor (<u>aid</u>, aname)
Plays (<u>aid, mid, role</u>)
Movie (<u>mid</u>, mname, director)

Sailors

not

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Reserves R, Boat B
  WHERE R.sid = S.sid
  AND R.bid = B.bid
  AND B.color = 'red')
```

only

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Reserves R
  WHERE R.sid = S.sid
  AND NOT EXISTS(
    SELECT *
    FROM Boat B
    WHERE R.bid = B.bid
    AND B.color = 'red'))
```

all

```
SELECT DISTINCT S.sname
FROM Sailor S
WHERE NOT EXISTS(
  SELECT *
  FROM Boat B
  WHERE B.color = 'red'
  AND NOT EXISTS(
    SELECT *
    FROM Reserves R
    WHERE R.bid = B.bid
    AND R.sid = S.sid))
```

Students

not

```
SELECT DISTINCT S.sname
FROM Student S
WHERE NOT EXISTS(
  SELECT *
  FROM Takes T, Class C
  WHERE T.sid = S.sid
  AND T.cid = C.bid
  AND C.department = art')
```

only

```
SELECT DISTINCT S.sname
FROM Student S
WHERE NOT EXISTS(
  SELECT *
  FROM Takes T
  WHERE T.sid = S.sid
  AND NOT EXISTS(
    SELECT *
    FROM Class C
    WHERE T.cid = C.cid
    AND C.department= 'art'))
```

all

```
SELECT DISTINCT S.sname
FROM Student S
WHERE NOT EXISTS(
  SELECT *
  FROM Class C
  WHERE C.department = 'art'
  AND NOT EXISTS(
    SELECT *
    FROM Takes T
    WHERE T.cid = C.cid
    AND T.sid = S.sid))
```

Actors

not

```
SELECT DISTINCT A.aname
FROM Actor A
WHERE NOT EXISTS(
  SELECT *
  FROM Plays P, Movie M
  WHERE P.aid = A.aid
  AND P.mid = M.mid
  AND M.director= 'Hitchcock')
```

only

```
SELECT DISTINCT A.aname
FROM Actor A
WHERE NOT EXISTS(
  SELECT *
  FROM Plays P
  WHERE P.aid = A.sid
  AND NOT EXISTS(
    SELECT *
    FROM Movie M
    WHERE P.mid = M.mid
    AND M.director= 'Hitchcock'))
```

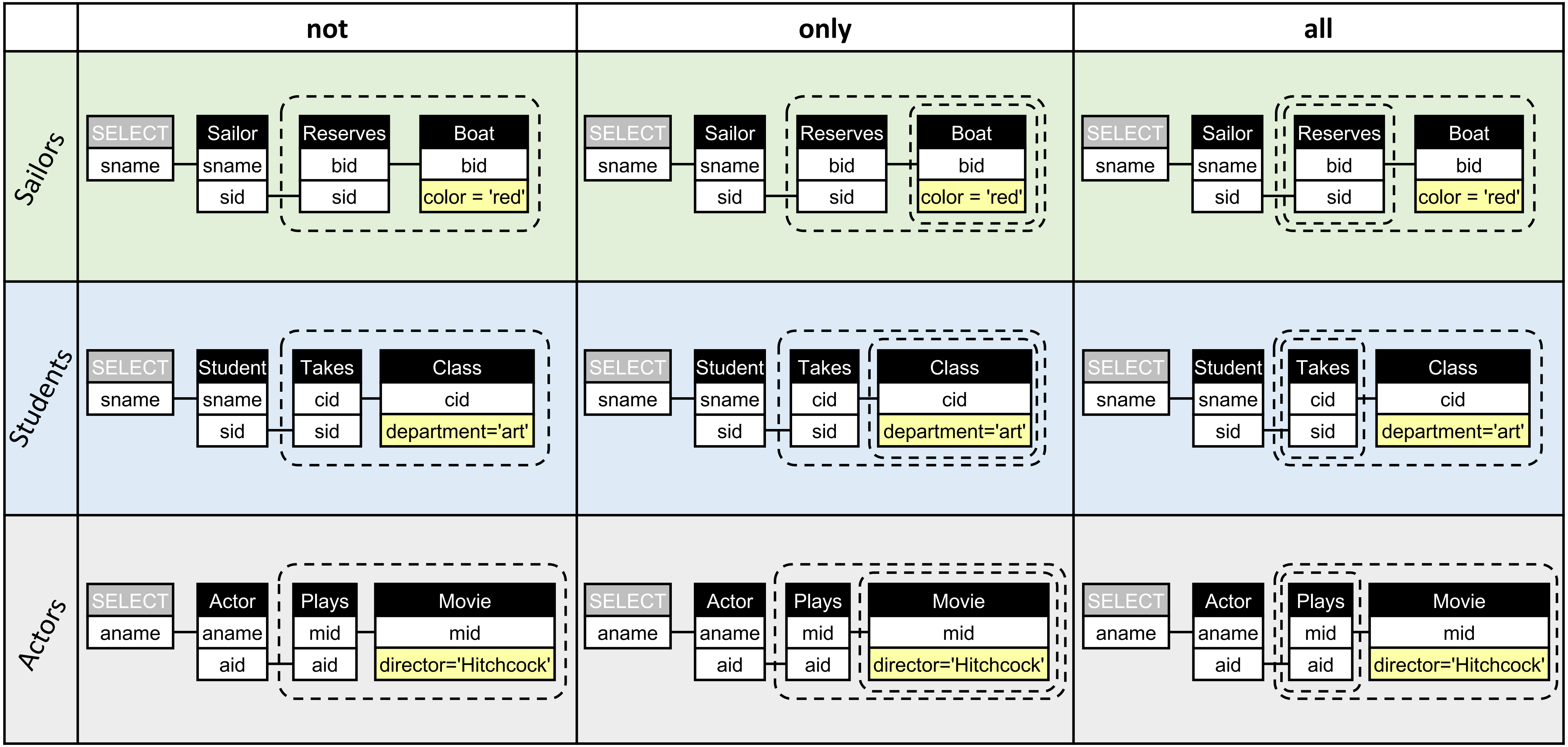
all

```
SELECT DISTINCT A.aname
FROM Actor A
WHERE NOT EXISTS(
  SELECT *
  FROM Movie M
  WHERE M.director= 'Hitchcock'
  AND NOT EXISTS(
    SELECT *
    FROM Plays P
    WHERE P.mid = M.mid
    AND P.aid = A.aid))
```

Sailor (<u>sid</u>, sname, rating, age)
Reserves (<u>sid, bid, day</u>)
Boat (<u>bid</u>, bname, color)

Student (<u>sid</u>, sname)
Takes (<u>sid, cid, semester</u>)
Course (<u>cid</u>, cname, department)

Actor (<u>aid</u>, aname)
Plays (<u>aid, mid, role</u>)
Movie (<u>mid</u>, mname, director)



Logical SQL Patterns

Logical patterns are the building blocks of most SQL queries.

Patterns are very hard to extract from the SQL text.

A pattern can appear across different database schemas.

Think of queries like:

- Find sailors who reserved all red boats
- Find students who took all art classes
- Find actors who played in all movies by Hitchcock

For a formal definition of relational query patterns see: Gatterbauer, Dunne. On the Reasonable Effectiveness of Relational Diagrams: Explaining Relational Query Patterns and the Pattern Expressiveness of Relational Languages, SIGMOD 2024, https://arxiv.org/pdf/2401.04758 , https://relationaldiagrams.com

What does this query return?

Likes(drinker,beer)

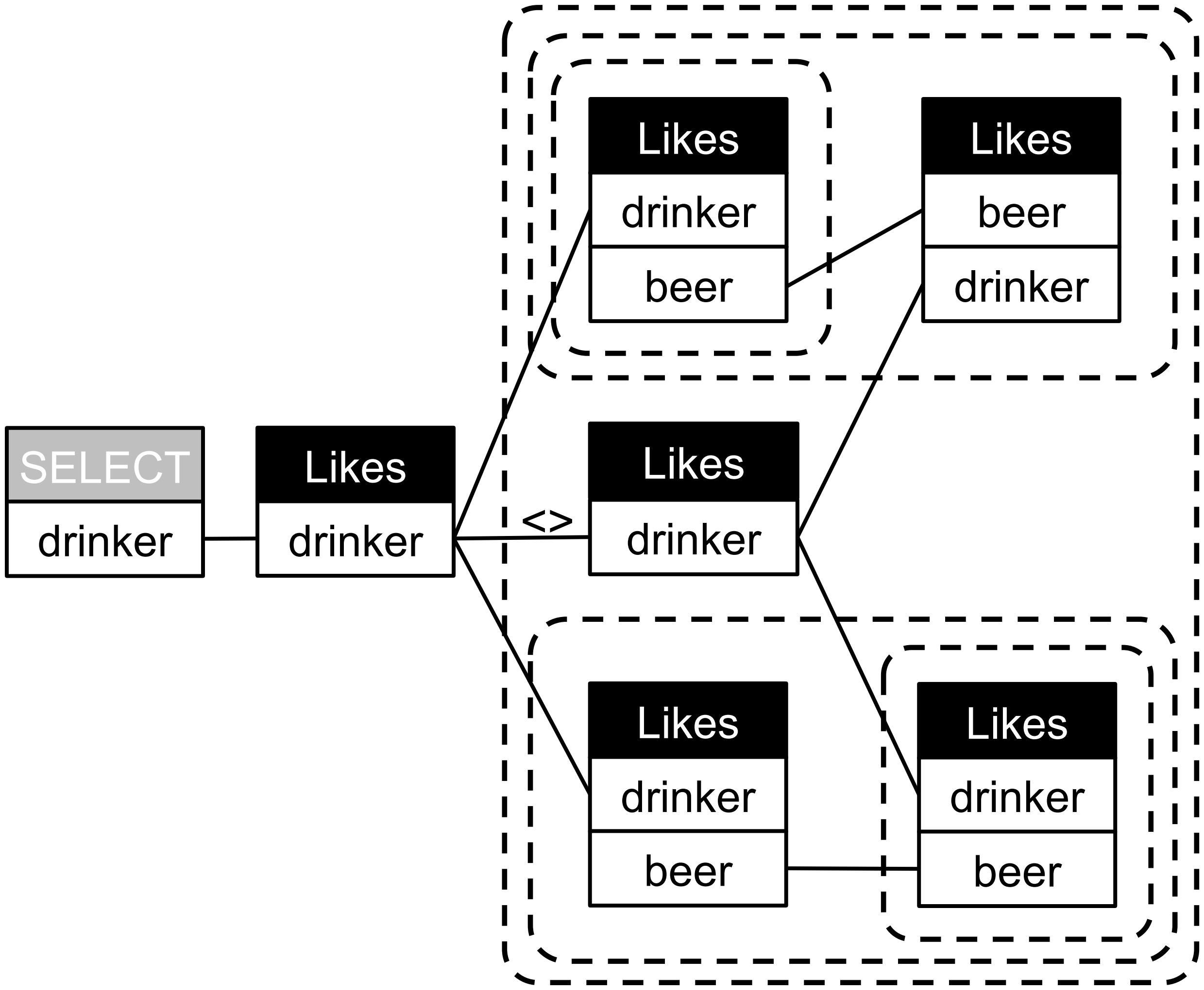
```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
  FROM Likes L2
  WHERE L1.drinker <> L2.drinker
  AND not exists
    (SELECT  *
    FROM Likes L3
    WHERE L3.drinker = L2.drinker
    AND not exists
      (SELECT *
      FROM Likes L4
      WHERE L4.drinker = L1.drinker
      AND L4.beer = L3.beer))
  AND not exists
    (SELECT *
    FROM Likes L5
    WHERE L5. drinker = L1. drinker
    AND not exists
      (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer= L5.beer)))
```

What does this query return?

Likes(drinker,beer)

```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
  FROM Likes L2
  WHERE L1.drinker <> L2.drinker
  AND not exists
    (SELECT  *
    FROM Likes L3
    WHERE L3.drinker = L2.drinker
    AND not exists
      (SELECT *
      FROM Likes L4
      WHERE L4.drinker = L1.drinker
      AND L4.beer = L3.beer))
  AND not exists
    (SELECT *
    FROM Likes L5
    WHERE L5. drinker = L1. drinker
    AND not exists
      (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer= L5.beer)))
```

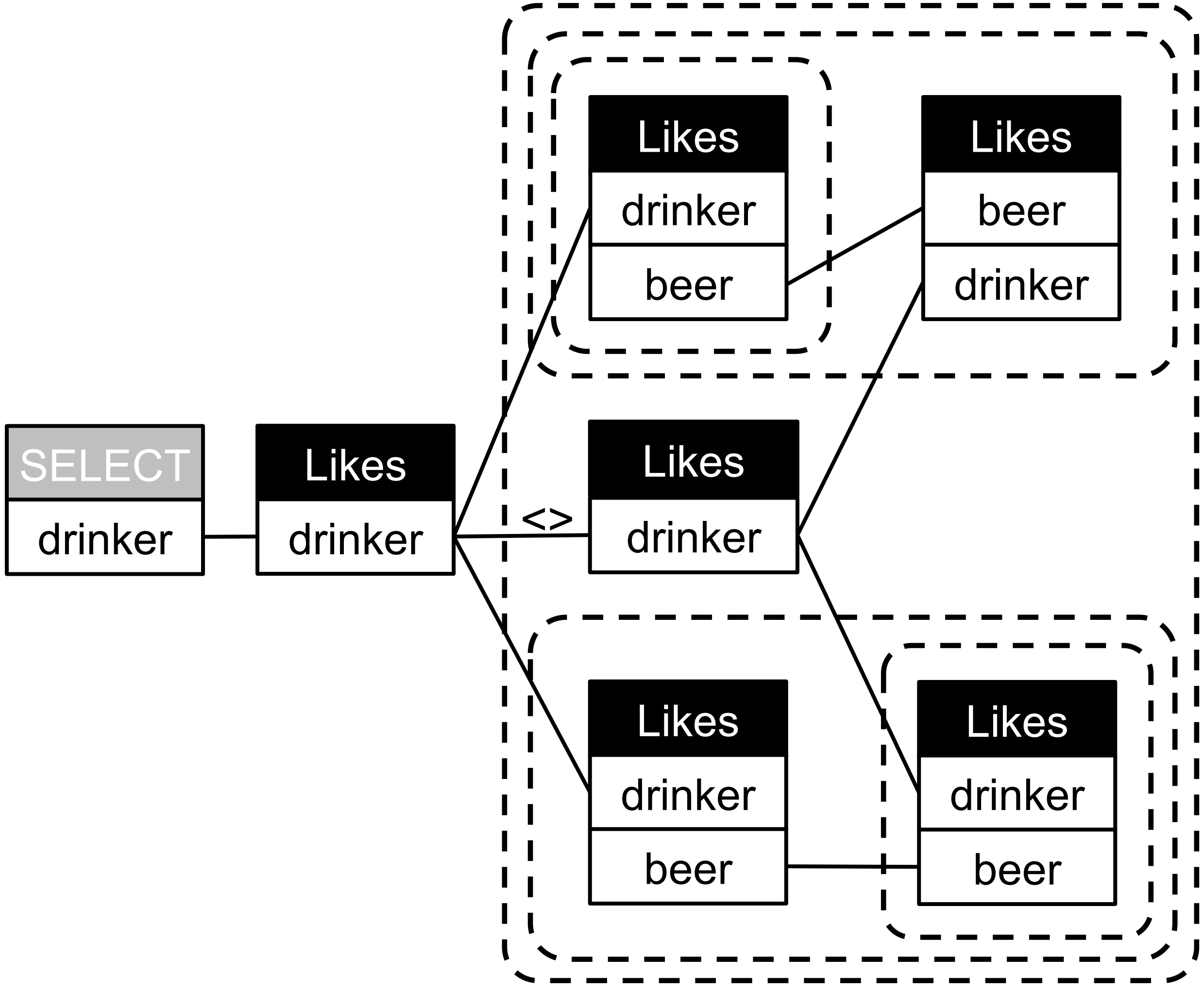
Relational Diagrams scoping



Q: Finder drinkers with a unique beer taste

Likes(drinker,beer)

```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
  FROM Likes L2
  WHERE L1.drinker <> L2.drinker
  AND not exists
    (SELECT  *
    FROM Likes L3
    WHERE L3.drinker = L2.drinker
    AND not exists
      (SELECT *
      FROM Likes L4
      WHERE L4.drinker = L1.drinker
      AND L4.beer = L3.beer))
  AND not exists
    (SELECT *
    FROM Likes L5
    WHERE L5. drinker = L1. drinker
    AND not exists
      (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer= L5.beer)))
```



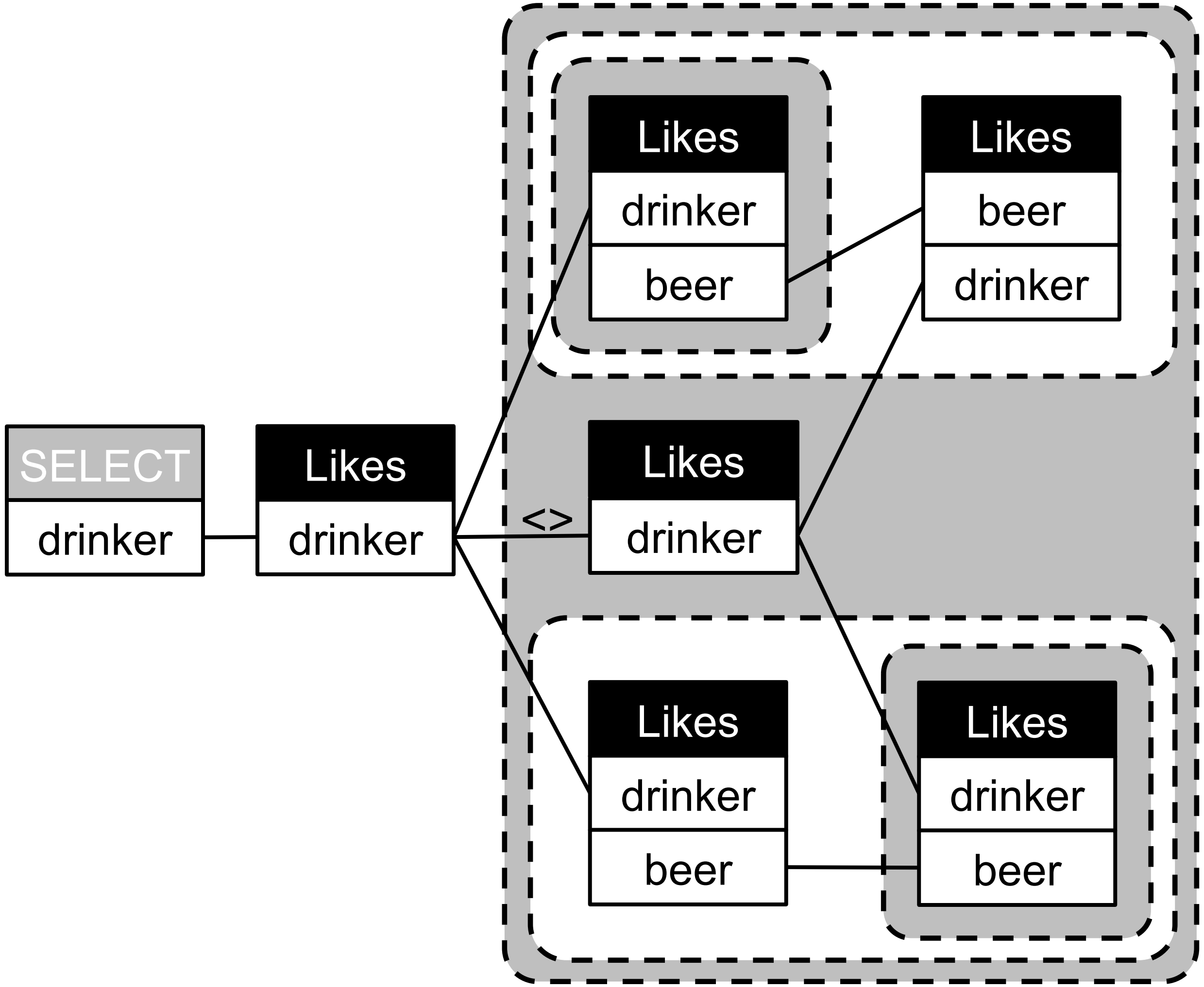
Relational Diagrams scoping

Q: Finder drinkers with a unique beer taste

Likes(drinker,beer)

```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
  FROM Likes L2
  WHERE L1.drinker <> L2.drinker
  AND not exists
    (SELECT  *
    FROM Likes L3
    WHERE L3.drinker = L2.drinker
    AND not exists
      (SELECT *
      FROM Likes L4
      WHERE L4.drinker = L1.drinker
      AND L4.beer = L3.beer))
  AND not exists
    (SELECT *
    FROM Likes L5
    WHERE L5. drinker = L1. drinker
    AND not exists
      (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer= L5.beer)))
```

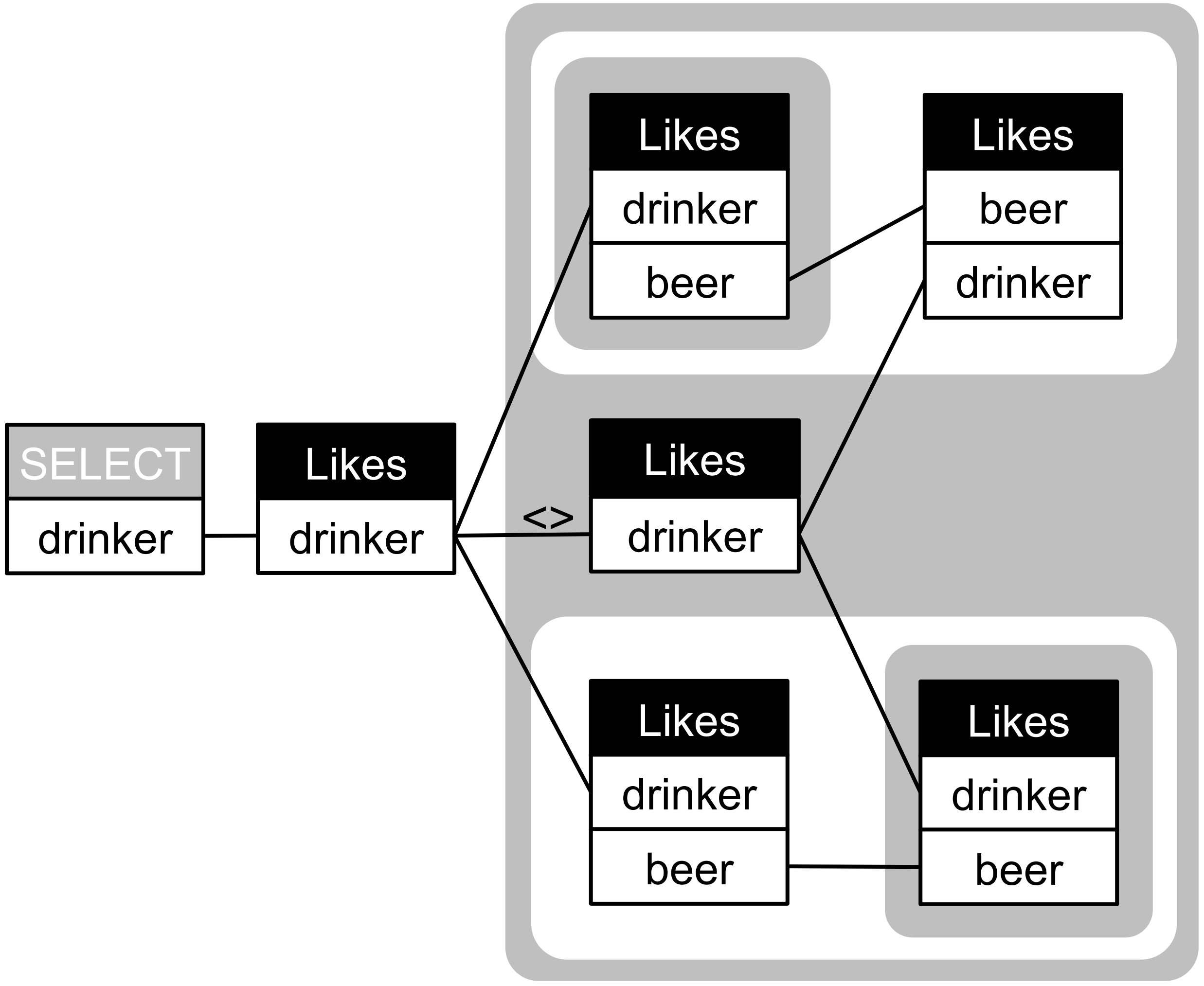
Relational Diagrams scoping



Q: Finder drinkers with a unique beer taste

Likes(drinker,beer)

```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
  FROM Likes L2
  WHERE L1.drinker <> L2.drinker
  AND not exists
    (SELECT  *
    FROM Likes L3
    WHERE L3.drinker = L2.drinker
    AND not exists
      (SELECT *
      FROM Likes L4
      WHERE L4.drinker = L1.drinker
      AND L4.beer = L3.beer))
  AND not exists
    (SELECT *
    FROM Likes L5
    WHERE L5. drinker = L1. drinker
    AND not exists
      (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer= L5.beer)))
```



Relational Diagrams scoping

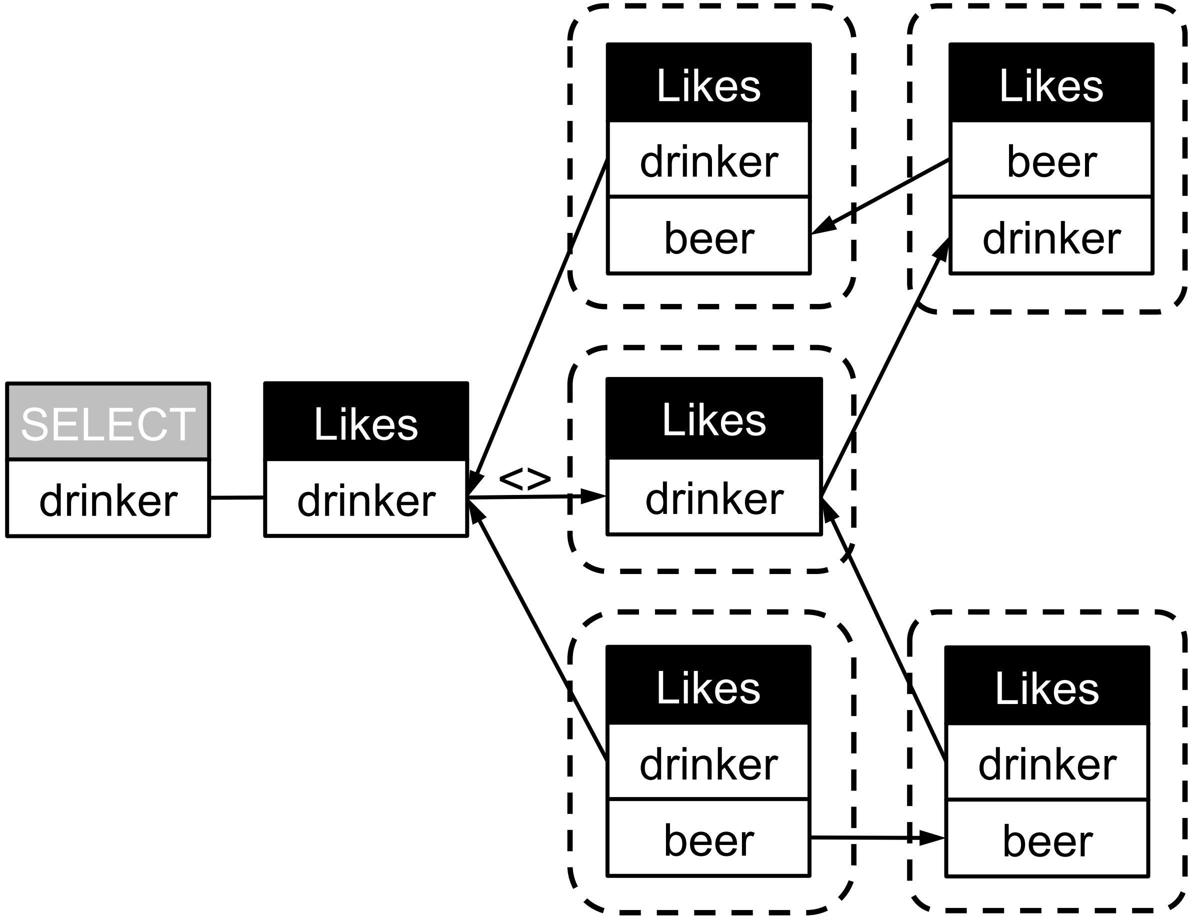
Q: Finder drinkers with a unique beer taste

Likes(drinker,beer)

```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
  (SELECT *
  FROM Likes L2
  WHERE L1.drinker <> L2.drinker
  AND not exists
    (SELECT  *
    FROM Likes L3
    WHERE L3.drinker = L2.drinker
    AND not exists
      (SELECT *
      FROM Likes L4
      WHERE L4.drinker = L1.drinker
      AND L4.beer = L3.beer))
  AND not exists
    (SELECT *
    FROM Likes L5
    WHERE L5. drinker = L1. drinker
    AND not exists
      (SELECT *
      FROM Likes L6
      WHERE L6.drinker = L2.drinker
      AND L6.beer= L5.beer)))
```

QueryVis scoping

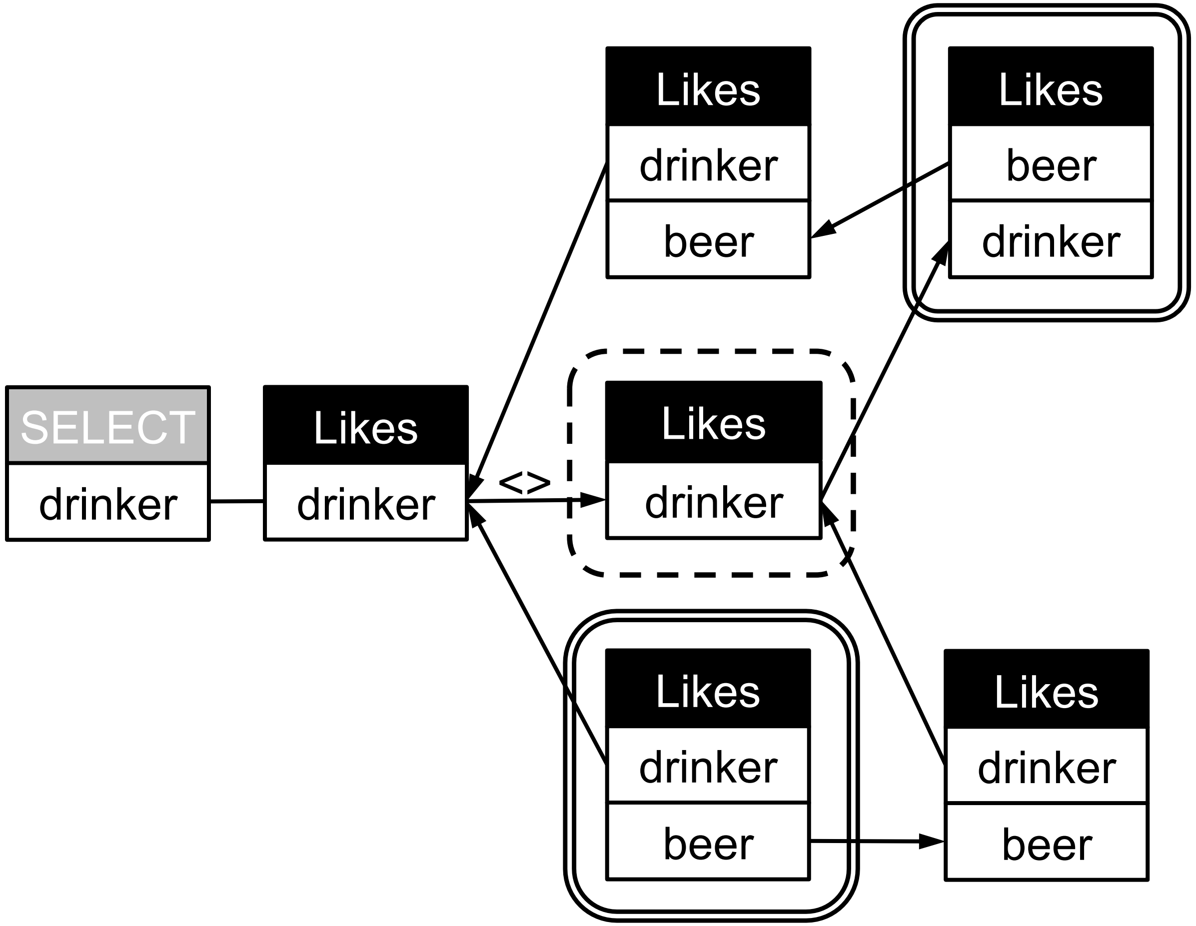
Relational Diagrams scoping



Q: Finder drinkers with a unique beer taste

Likes(drinker,beer)

```
SELECT L1.drinker
FROM Likes L1
WHERE not exists
   (SELECT *
   FROM Likes L2
   WHERE L1.drinker <> L2.drinker
   AND not exists
     (SELECT  *
     FROM Likes L3
     WHERE L3.drinker = L2.drinker
     AND not exists
       (SELECT *
       FROM Likes L4
       WHERE L4.drinker = L1.drinker
       AND L4.beer = L3.beer))
   AND not exists
     (SELECT *
     FROM Likes L5
     WHERE L5. drinker = L1. drinker
     AND not exists
       (SELECT *
       FROM Likes L6
       WHERE L6.drinker = L2.drinker
       AND L6.beer= L5.beer)))
```



QueryVis scoping

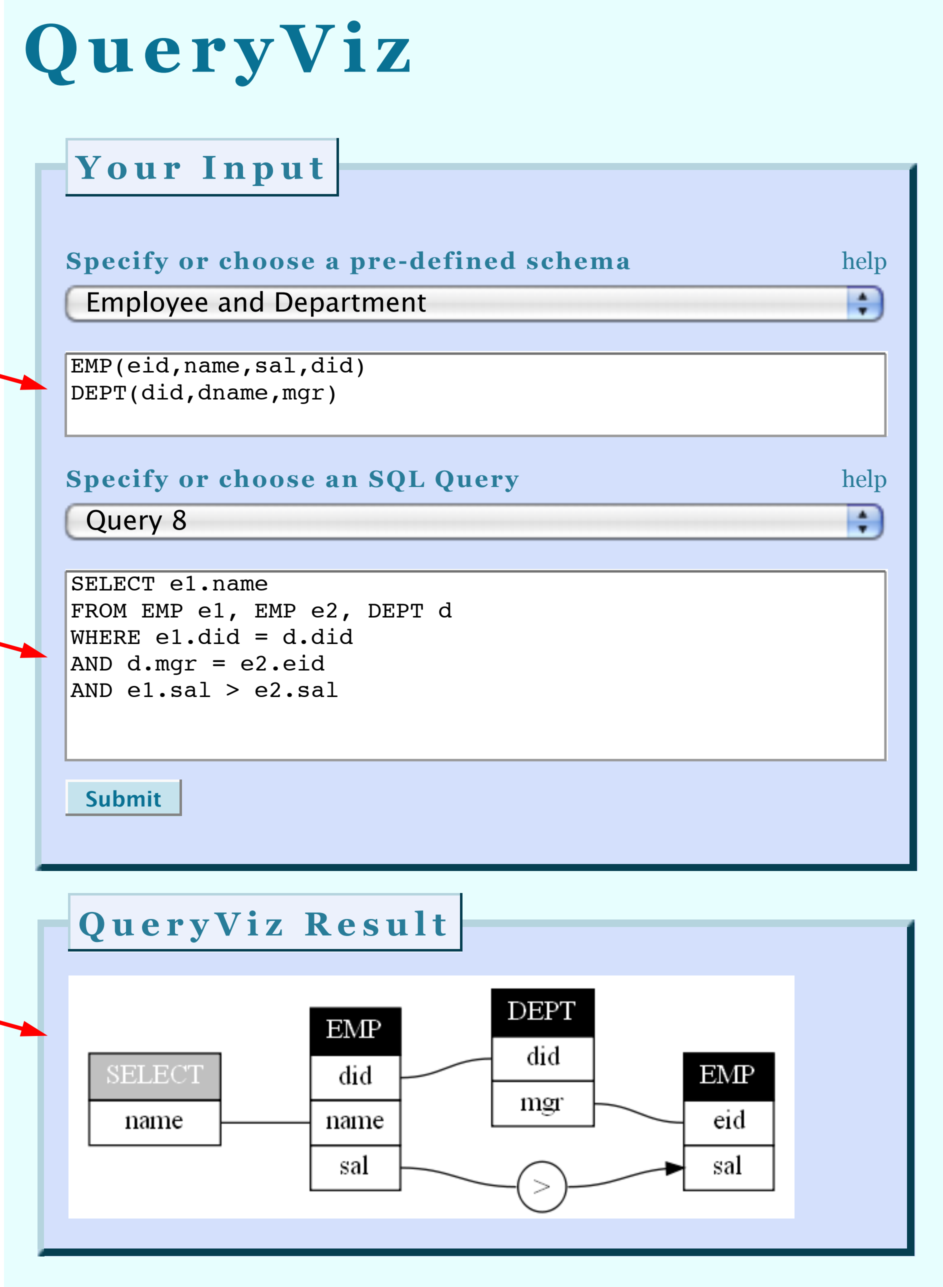
Relational Diagrams scoping

https://demo.queryvis.com

Input: Schema

Input Query

Output: Visualization



```
EMP(eid,name,sal,did)
DEPT(did,dname,mgr)
```

```
SELECT e1.name
FROM EMP e1, EMP e2, DEPT d
WHERE e1.did = d.did
AND d.mgr = e2.eid
AND e1.sal > e2.sal
```

https://queryvis.com/

http://www.youtube.com/watch?v=kVFnQRGAQls

Source: Danaparamita, Gatterbauer: QueryViz: Helping users understand SQL queries and their patterns. EDBT 2011. https://doi.org/10.14778/3402755.3402805

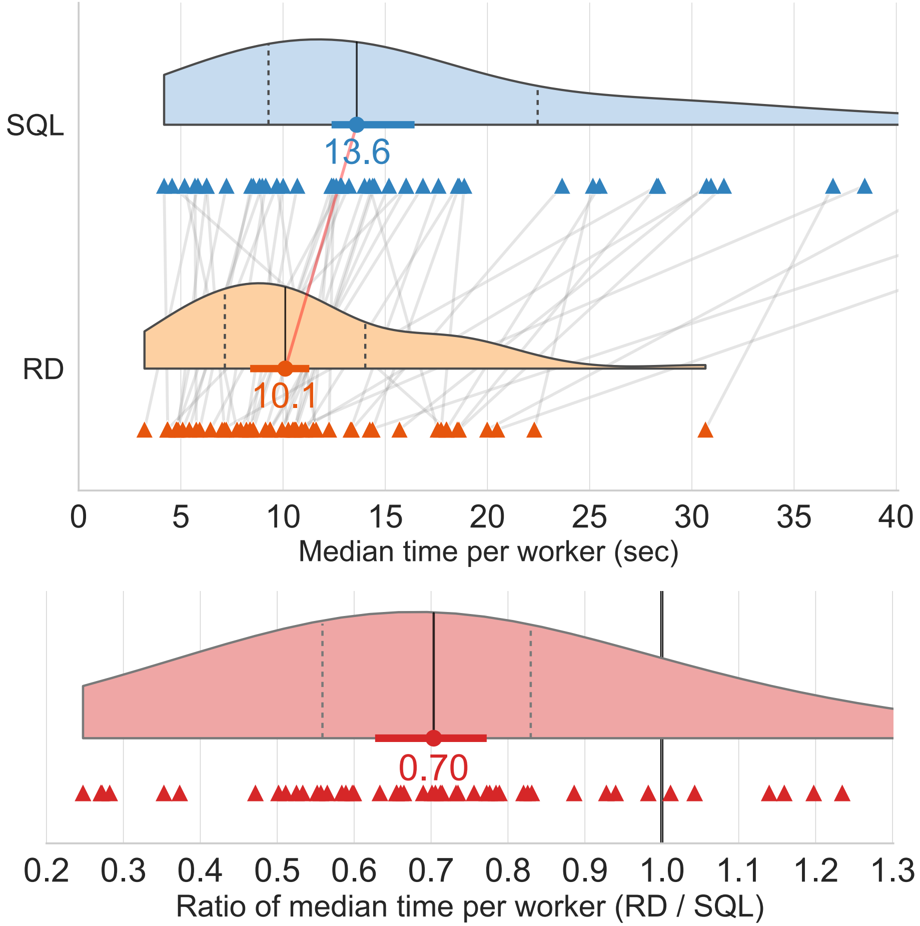
See also: Gatterbauer, Dunne, Jagadish, Riedewald: Principles of Query Visualization. IEEE Debull 2023. http://sites.computer.org/debull/A22sept/p47.pdf

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

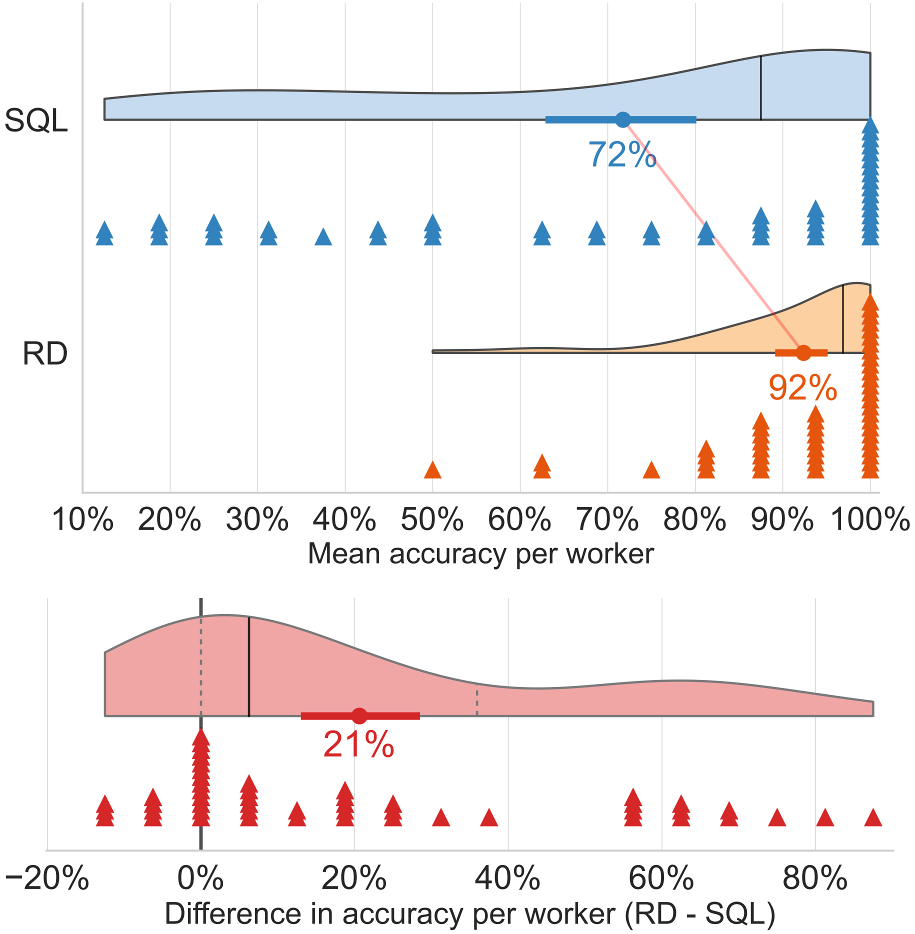
Preregistered, randomized user study on AMT

n = 50 participants, preregistration: https://osf.io/4zpsk

Speed



Accuracy



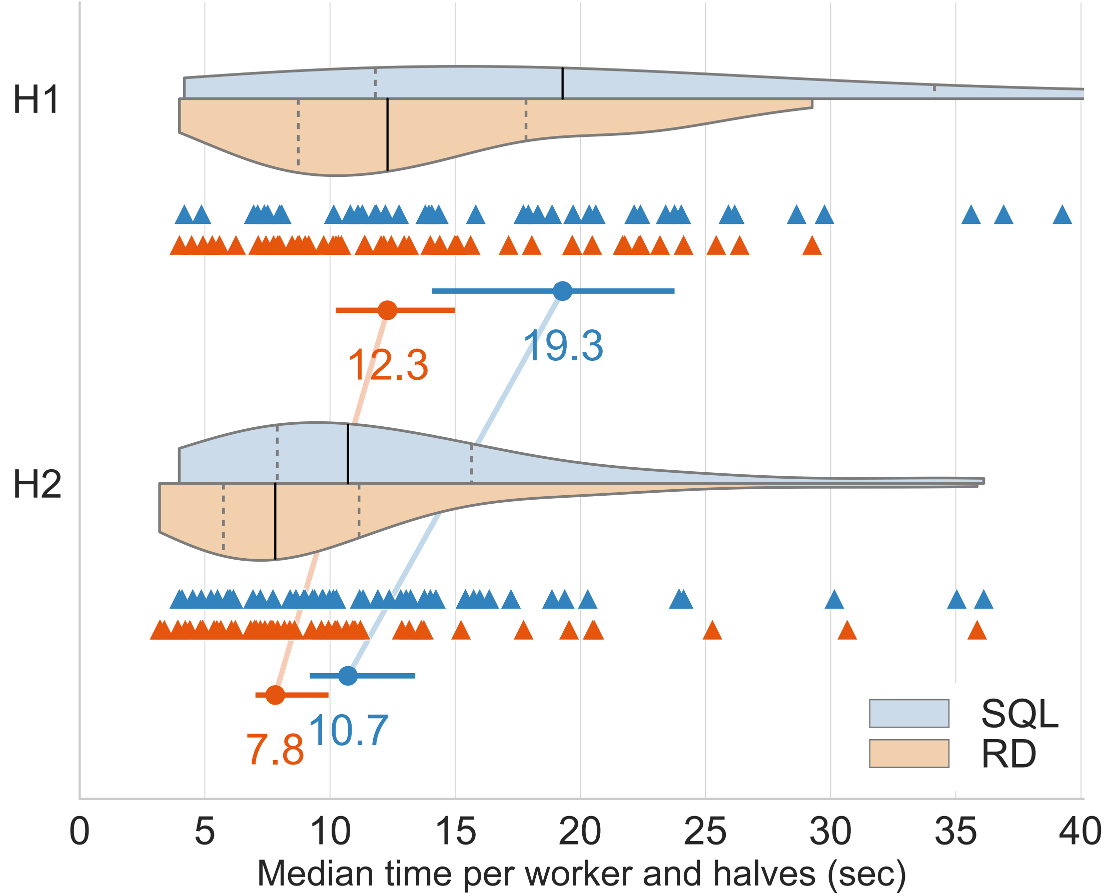
Source: Gatterbauer, Dunne. On the Reasonable Effectiveness of Relational Diagrams: Explaining Relational Query Patterns and the Pattern Expressiveness of Relational Languages, SIGMOD 2024, https://arxiv.org/pdf/2401.04758 ,

Preregistered, randomized user study on AMT

n = 50 participants, preregistration: https://osf.io/4zpsk

Learning

H1 = first 16 questions
H2 = second 16 questions



Source: Gatterbauer, Dunne. On the Reasonable Effectiveness of Relational Diagrams: Explaining Relational Query Patterns and the Pattern Expressiveness of Relational Languages, SIGMOD 2024, https://arxiv.org/pdf/2401.04758 ,

Outline: T1-U1: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Union and Theta Joins
  - Nulls & Outer joins
  - Top-k
  - [Recursion: moved to T1-U4: Datalog]

Union

R

a
1
2

U

a
2
3
4

305

```
SELECT a
FROM   R
UNION
SELECT a
FROM   U
```

⇒ ?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Union uses set semantics

305

R

a
1
2

U

a
2
3
4

What if we wanted also the duplicates

```
SELECT a
FROM   R
UNION
SELECT a
FROM   U
```

⇨

a
1
2
3
4

?

⇨

a
1
2
2
3
4

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Union ALL uses bag semantics

R

a
1
2

U

a
2
3
4

```
SELECT a
FROM   R
UNION ALL
SELECT a
FROM   U
```

⇨

a
1
2
2
3
4

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Theta joins

R

a
1
2

U

a
2
3
4

305

What do these queries compute?

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a < U.a
```

⇨ ?

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a >= U.a
```

⇨ ?

A **Theta-join** allows for arbitrary comparison relationships (such as ≥).
An **equijoin** is a theta join using the equality operator.

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Theta joins


305

R

a
1
2

U

a
2
3
4

What do these queries compute?

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a < U.a
```

⇨

a	b
1	2
1	3
1	4
2	3
2	4

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a >= U.a
```

⇨ ?

A **Theta-join** allows for arbitrary comparison relationships (such as ≥).
An **equijoin** is a theta join using the equality operator.

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Theta joins

305

R

a
1
2

U

a
2
3
4

What do these queries compute?

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a < U.a
```

a	b
1	2
1	3
1	4
2	3
2	4

∪

```
SELECT   R.a, U.a as b
FROM     R, U
WHERE    R.a >= U.a
```

a	b
2	2

×

Think about these two queries as a partition of the Cartesian product

A **Theta-join** allows for arbitrary comparison relationships (such as ≥).
An **equijoin** is a theta join using the equality operator.

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Outline: T1-U1: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Union and Theta Joins
  - Nulls & Outer joins
  - Top-k
  - [Recursion: moved to T1-U4: Datalog]

3-valued logic example

- Three logicians walk into a bar. The bartender asks: "Do all of you want a drink?"
- The 1st logician says: "I don't know."
- The 2nd logician says: "I don't know."
- The 3rd logician says: "Yes!"

What is going on here ?

Nulls in SQL

- Whenever we don't have a value, we can put a NULL
- Can mean many things, e.g.:

?

Nulls in SQL

- Whenever we don't have a value, we can put a NULL
- Can mean many things, e.g.:
  - Value exists but is unknown
  - Value not applicable (yet)

A new class without a grade

Student	Class	Semester	grade
Alice	cs3200	Fall 2023	B+
Bob	cs3200	Spring 2024	null

- The schema specifies for each attribute if it can be NULL (nullable attribute) or not ("NOT NULL")
- Lots of ongoing research on NULLs
- Next: How does SQL cope with tables that have NULLs ?

Null Values

- In SQL there are three Boolean values ("ternary logic")
  - FALSE, TRUE, UNKNOWN

- If x = NULL then
  - Boolean conditions are also NULL. E.g: x = 'Joe'
  - Arithmetic operations produce NULL. E.g: 4*(3-x)/7
  - But aggregates ignore NULL values (exception: count(*))

- Logical reasoning:
  - FALSE = 0
  - TRUE = 1
  - UNKNOWN = 0.5

x AND y = min(x,y)

1 0.5 → 0.5

0 0.5 → 0

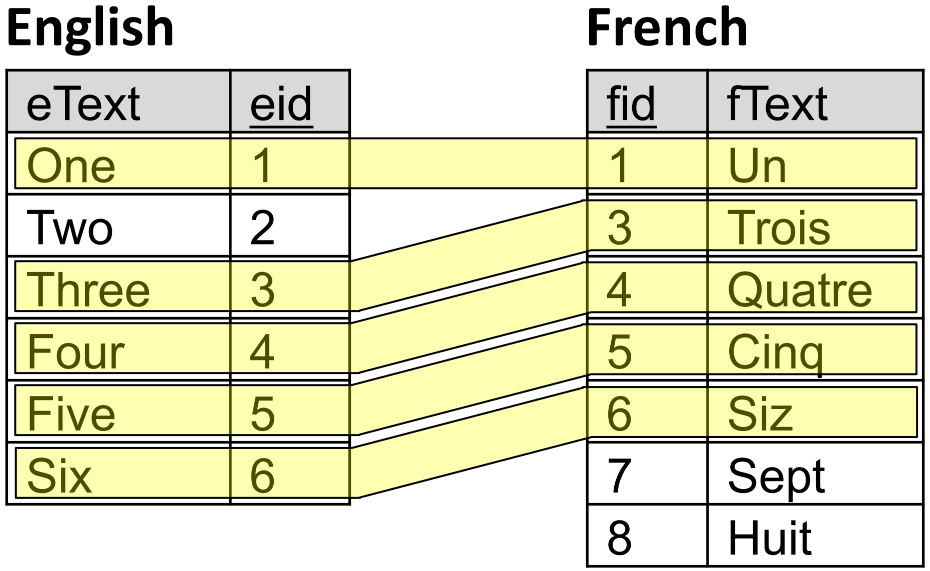
x OR y = max(x,y)

NOT x = (1 – x)

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Join Illustration

361



English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

An "inner join":

```
SELECT  *
FROM    English, French
WHERE   eid = fid
```

Same as (alternative join syntax):

```
SELECT  *
FROM    English JOIN French
ON      eid = fid
```

shortform of "INNER JOIN"

etext	eid	fid	ftext
One	1	1	Un
Three	3	3	Trois
Four	4	4	Quatre
Five	5	5	Cinq
Six	6	6	Siz

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Join Illustration

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

Null also sometimes just shown as empty

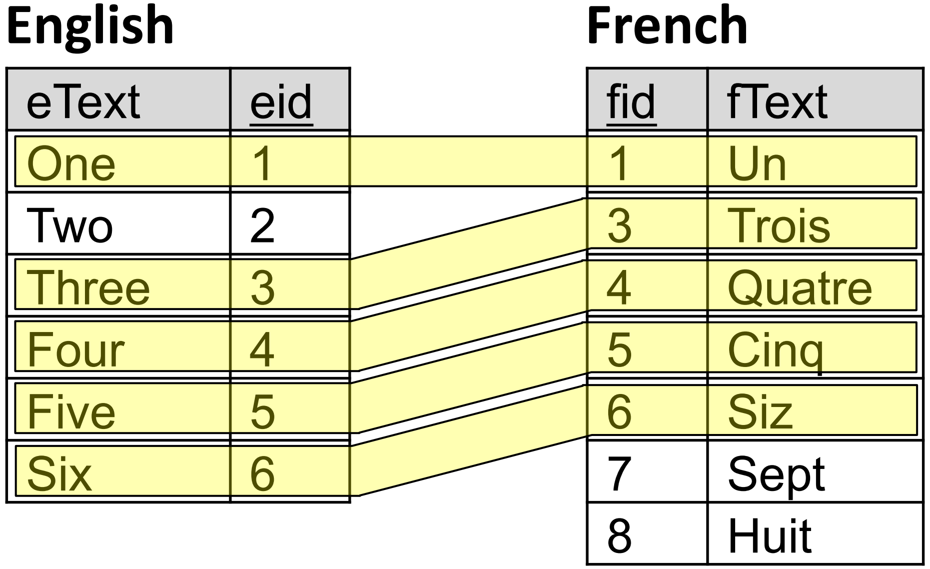
etext	eid	fid	ftext
One	1	1	Un
Two	2	NULL	NULL
Three	3	3	Trois
Four	4	4	Quatre
Five	5	5	Cinq
Six	6	6	Siz
NULL	NULL	7	Sept
NULL	NULL	8	Huit

How do we get a join with the full data ?

~~SELECT *~~
~~FROM English JOIN French~~
~~ON eid = fid~~

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Join Illustration



English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

shortform of:
"FULL OUTER JOIN"

```
SELECT  *
FROM    English FULL JOIN French
ON      English.eid = French.fid
```

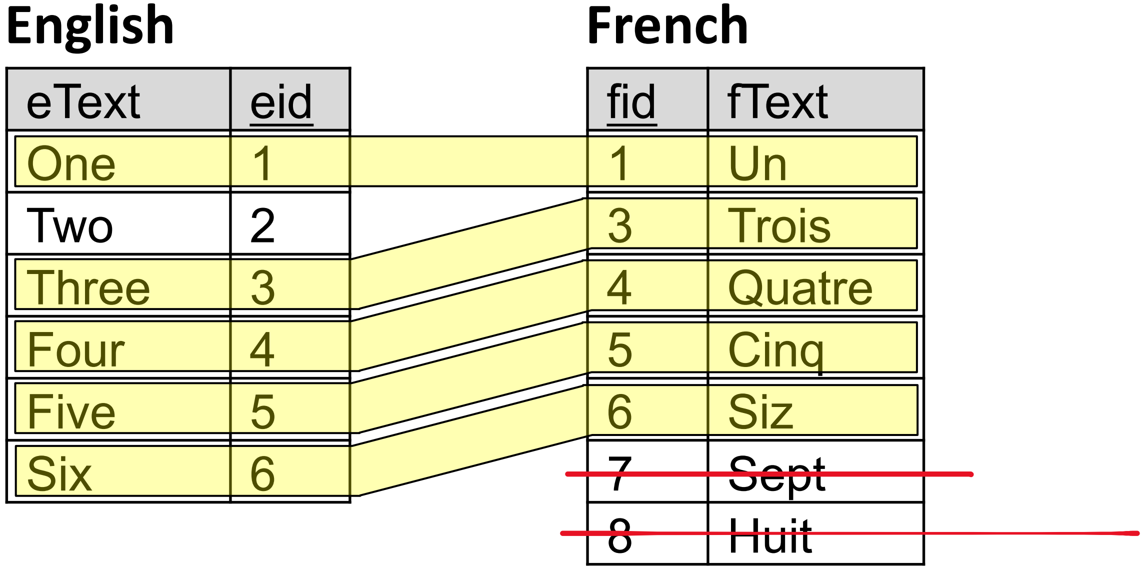
~~SELECT *~~
~~FROM English JOIN French~~
~~ON eid = fid~~

Null also sometimes just shown as empty

etext	eid	fid	ftext
One	1	1	Un
Two	2	NULL	NULL
Three	3	3	Trois
Four	4	4	Quatre
Five	5	5	Cinq
Six	6	6	Siz
NULL	NULL	7	Sept
NULL	NULL	8	Huit

Join Illustration

361



English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
~~7~~	~~Sept~~
~~8~~	~~Huit~~

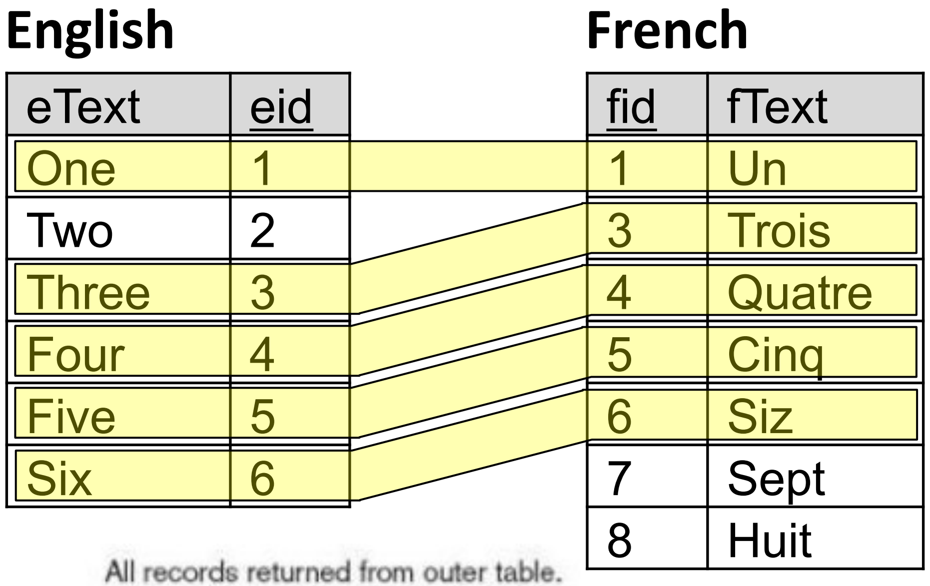
```
SELECT  *
FROM    English LEFT JOIN French
ON      English.eid = French.fid
```

etext	eid	fid	ftext
One	1	1	Un
Two	2	NULL	NULL
Three	3	3	Trois
Four	4	4	Quatre
Five	5	5	Cinq
Six	6	6	Siz

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Join Illustration

361

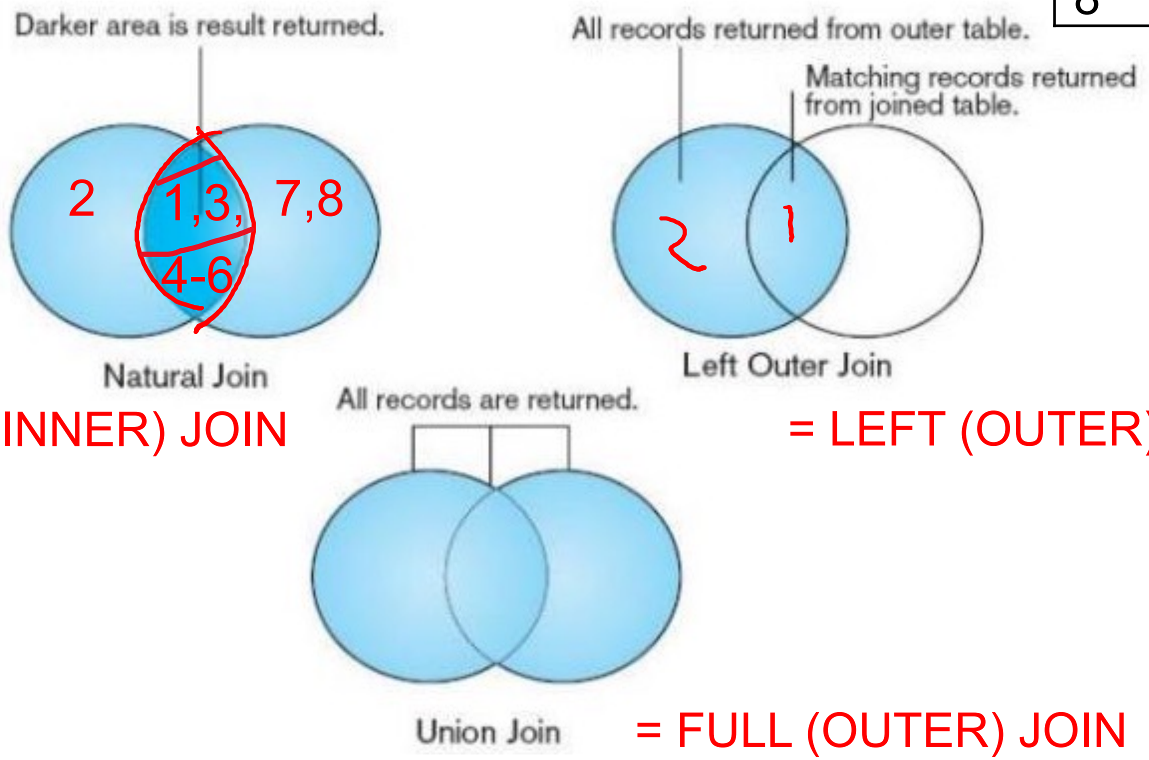


English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit



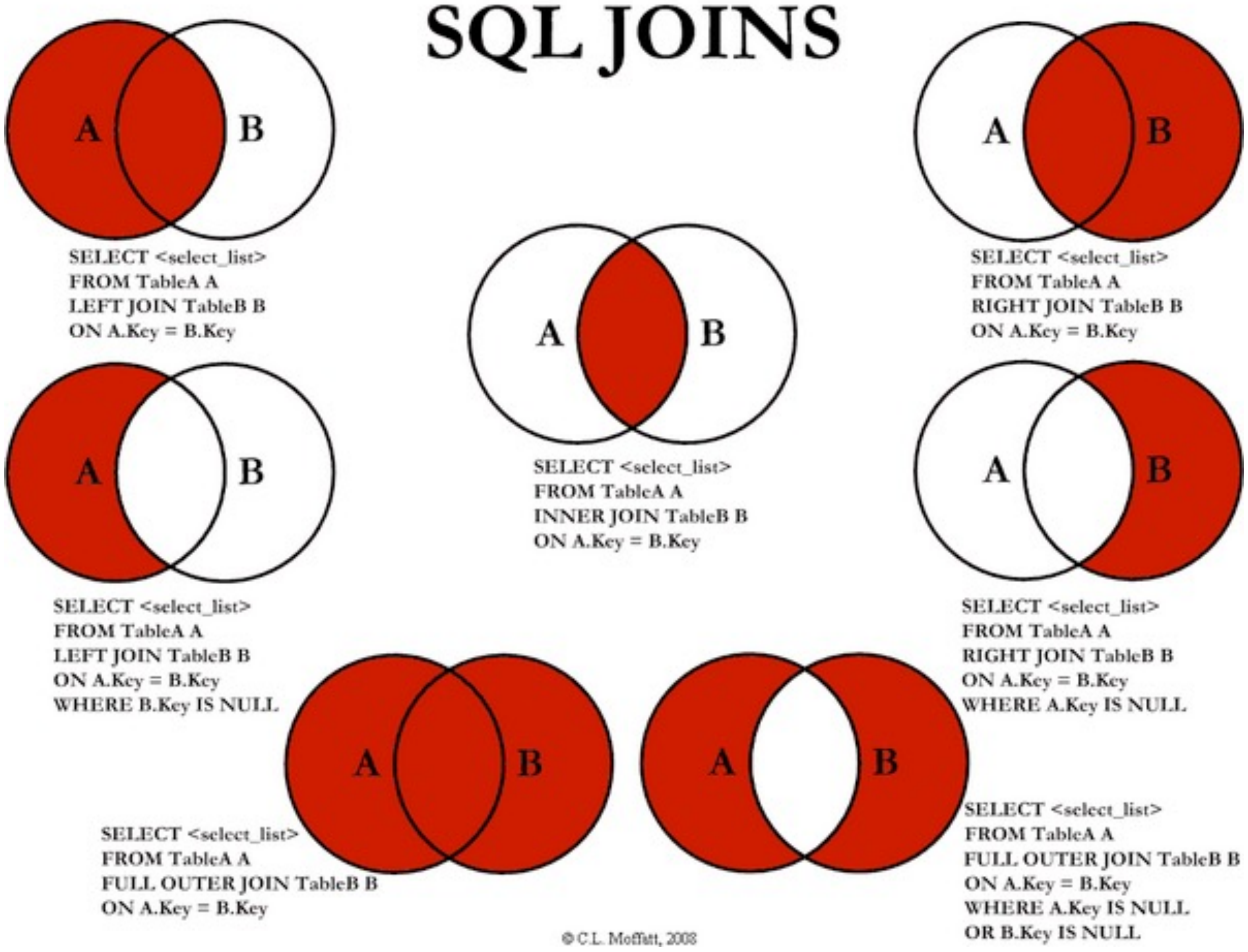
= (INNER) JOIN

= LEFT (OUTER) JOIN

= FULL (OUTER) JOIN

Source: Fig. 7-2, Hoffer et al., Modern Database Management, 10ed ed, 2011.

Detailed Illustration with Examples (follow the link)



```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

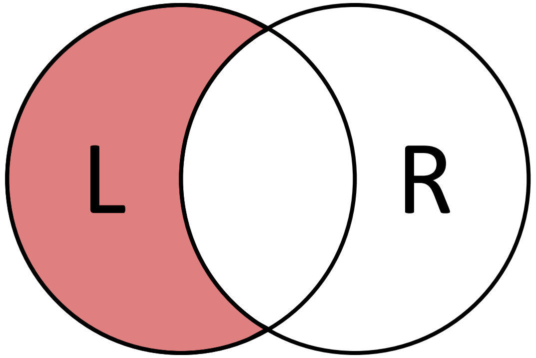
also called "anti-join"

Check this web page for illustrating examples

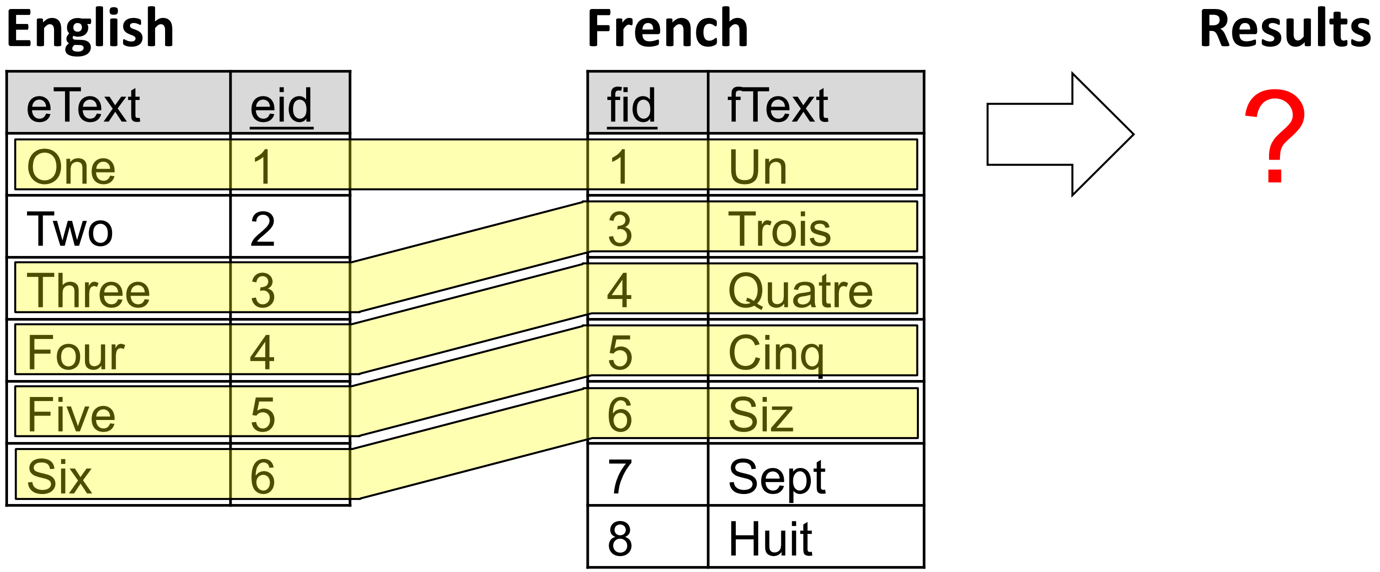
Source: http://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins

Let's practice anti-joins

361



```
SELECT <select_list>
FROM L
LEFT JOIN R
ON L.key = R.key
WHERE R.key IS NULL
```



English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

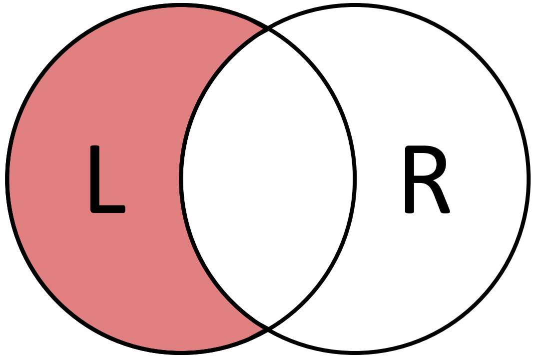
Results

?

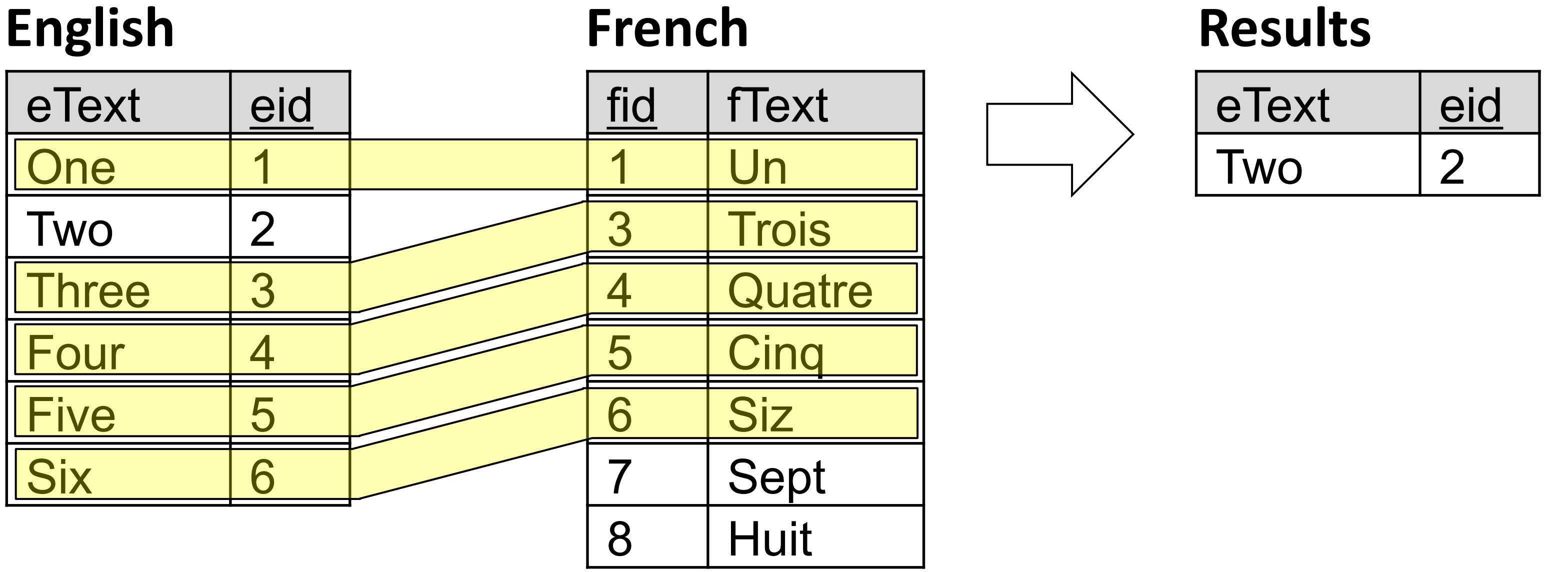
SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Let's practice anti-joins

361



```
SELECT <select_list>
FROM L
LEFT JOIN R
ON L.key = R.key
WHERE R.key IS NULL
```



English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

Results

eText	eid
Two	2

How to write in SQL?

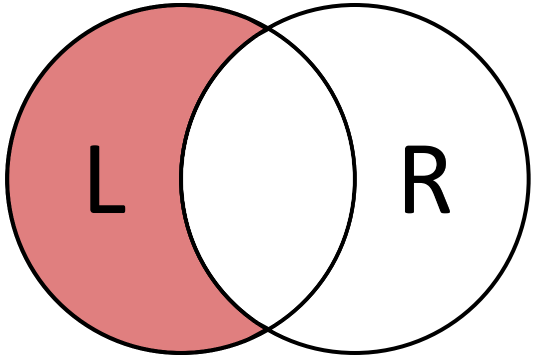
?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

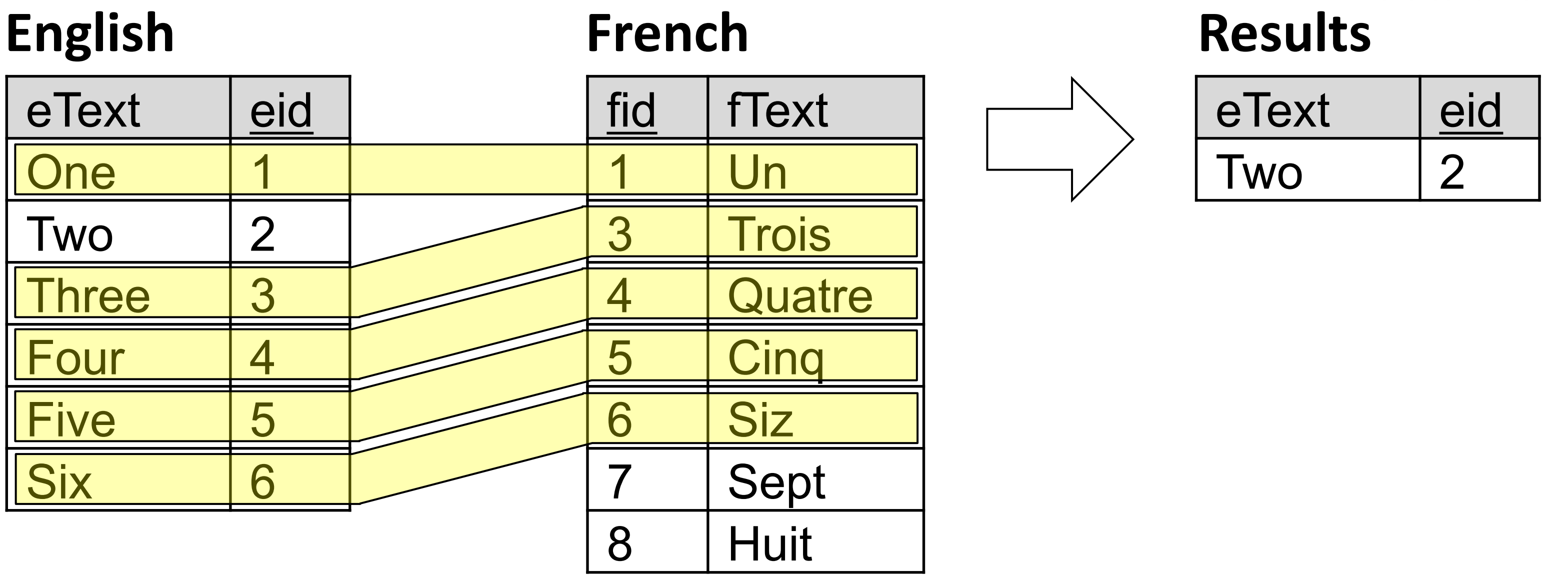
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Let's practice anti-joins

361



```
SELECT <select_list>
FROM L
LEFT JOIN R
ON L.key = R.key
WHERE R.key IS NULL
```



English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

Results

eText	eid
Two	2

How to write in SQL?

```
SELECT eText, eid
FROM English
LEFT JOIN French
ON eid = fid
WHERE fid IS NULL
```

Any alternative?

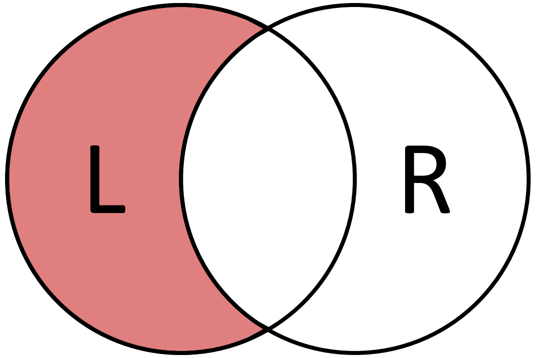
?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

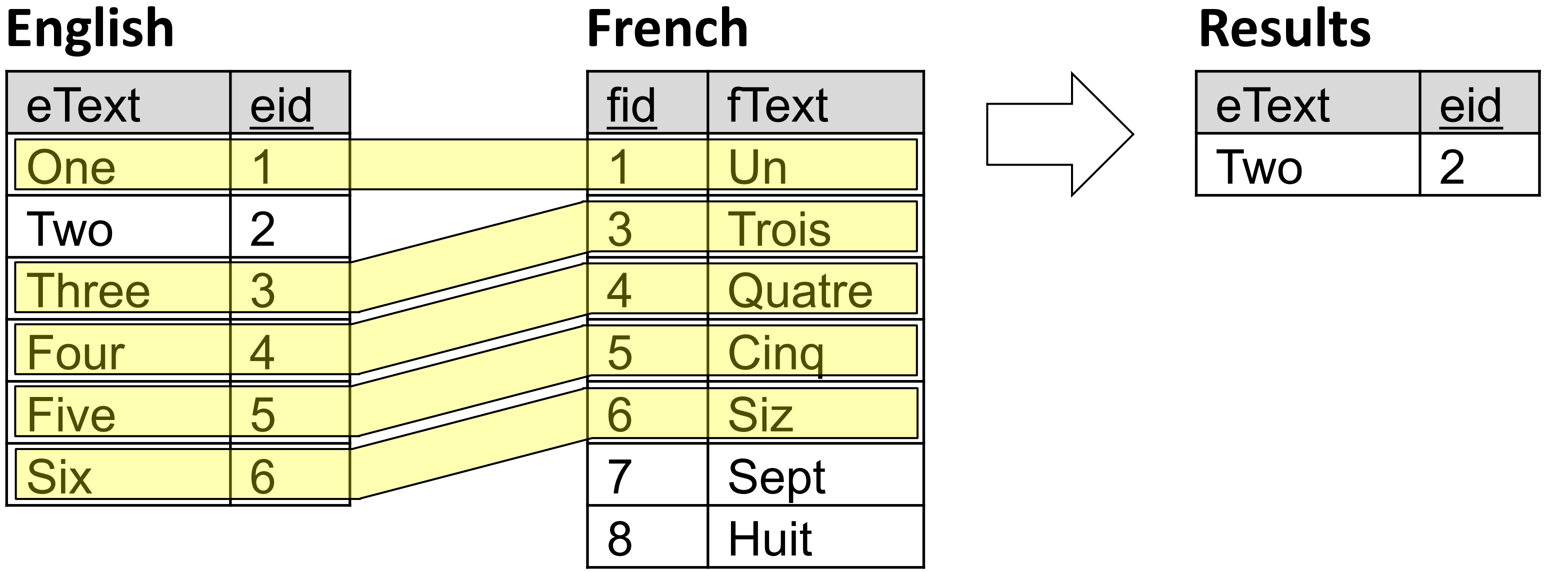
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Let's practice anti-joins

361



```
SELECT <select_list>
FROM L
LEFT JOIN R
ON L.key = R.key
WHERE R.key IS NULL
```



English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

Results

eText	eid
Two	2

How to write in SQL?

```
SELECT eText, eid
FROM English
LEFT JOIN French
ON eid = fid
WHERE fid IS NULL
```

Any alternative?

```
SELECT *
FROM English
WHERE eid NOT IN
    (SELECT fid
    FROM French)
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Semi-joins: kind of the anti-anti-joins…

361

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

Results

eText	eid
One	1
Three	3
Four	4
Five	5
Six	6

What do we have to change to these queries to get the tuples in English that have a partner in French?

?

```
SELECT eText, eid
FROM English
LEFT JOIN French
ON eid = fid
WHERE fid IS NULL
```

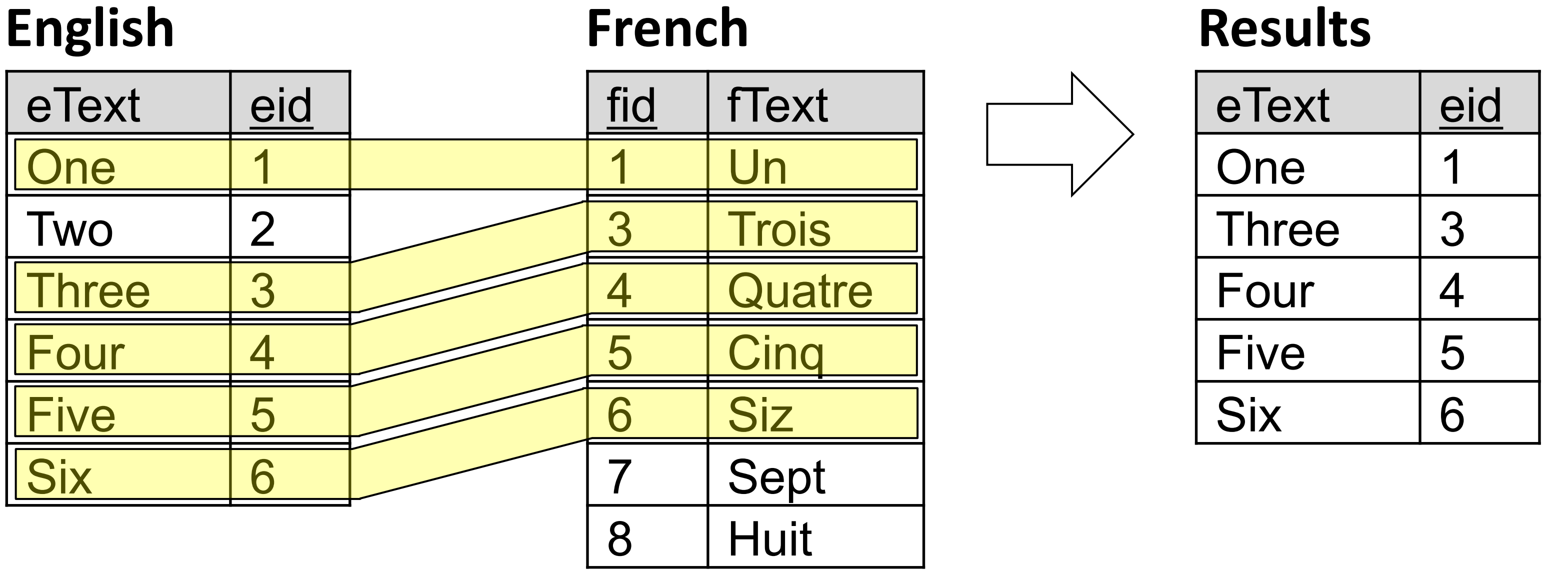
```
SELECT *
FROM English
WHERE eid NOT IN
    (SELECT fid
    FROM French)
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Semi-joins: kind of the anti-anti-joins...

361



English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

Results

eText	eid
One	1
Three	3
Four	4
Five	5
Six	6

What do we have to change to these queries to get the tuples in English that have a partner in French?

```sql
SELECT eText, eid
FROM English
LEFT JOIN French
ON eid = fid
WHERE fid IS NOT NULL
```

```sql
SELECT *
FROM English
WHERE eid IN
    (SELECT fid
     FROM French)
```

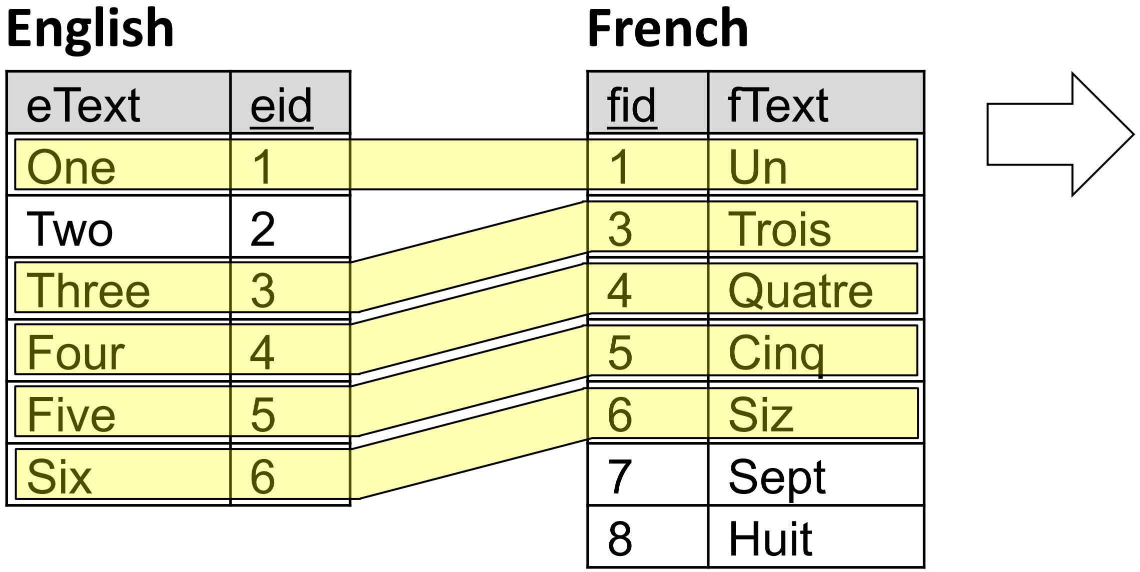
What if fid is not a key?

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Semi-joins: kind of the anti-anti-joins...

361



English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

Results

eText	eid
One	1
Three	3
Four	4
Five	5
Six	6

What do we have to change to these queries to get the tuples in English that have a partner in French?

What if fid is not a key?

DISTINCT

```
SELECT eText, eid
FROM English
LEFT JOIN French
ON eid = fid
WHERE fid IS NOT NULL
```

These queries express "semi-joins"

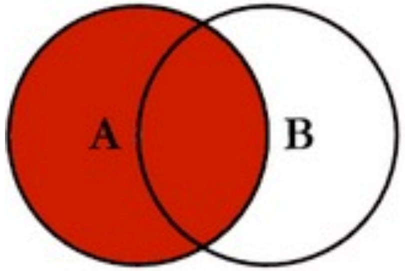
```
SELECT *
FROM English
WHERE eid IN
    (SELECT fid
    FROM French)
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

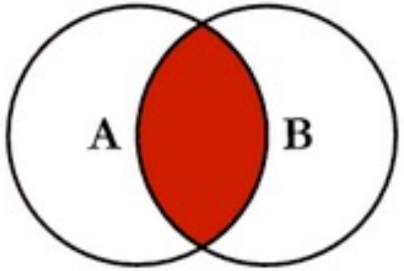
Semi-joins: kind of the anti-anti-joins...

Which of these two VENN diagrams from earlier correspond to a semi-join?

?



```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

Get the tuples in English that have a partner in French?

These queries express "semi-joins"

DISTINCT

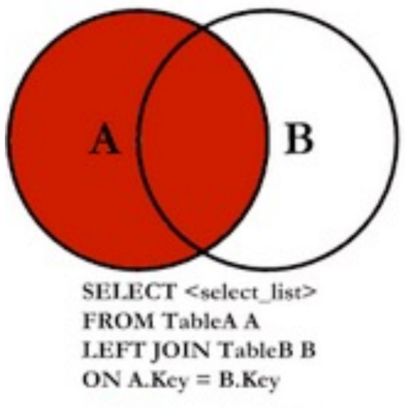
```
SELECT eText, eid
FROM English
LEFT JOIN French
ON eid = fid
WHERE fid IS NOT NULL
```

```
SELECT *
FROM English
WHERE eid IN
   (SELECT fid
   FROM French)
```

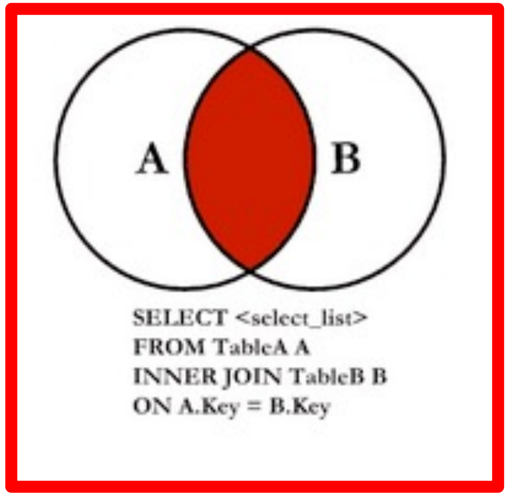
Semi-joins: kind of the anti-anti-joins...

361

Which of these two VENN diagrams from earlier correspond to a semi-join?



SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key



SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key

Only preserve tuples from A that also appear in B...

Get the tuples in English that have a partner in French?

DISTINCT

```
SELECT eText, eid
FROM English
LEFT JOIN French
ON eid = fid
WHERE fid IS NOT NULL
```

These queries express "semi-joins"

```
SELECT *
FROM English
WHERE eid IN
    (SELECT fid
    FROM French)
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Another look at Outer Joins

361

```
SELECT  *
FROM    English FULL JOIN French
ON      English.eid = French.fid
```

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

etext	eid	fid	ftext
One	1	1	Un
Two	2	NULL	NULL
Three	3	3	Trois
Four	4	4	Quatre
Five	5	5	Cinq
Six	6	6	Siz
NULL	NULL	7	Sept
NULL	NULL	8	Huit

FULL JOIN can be written as union of inner join with anti-joins

?

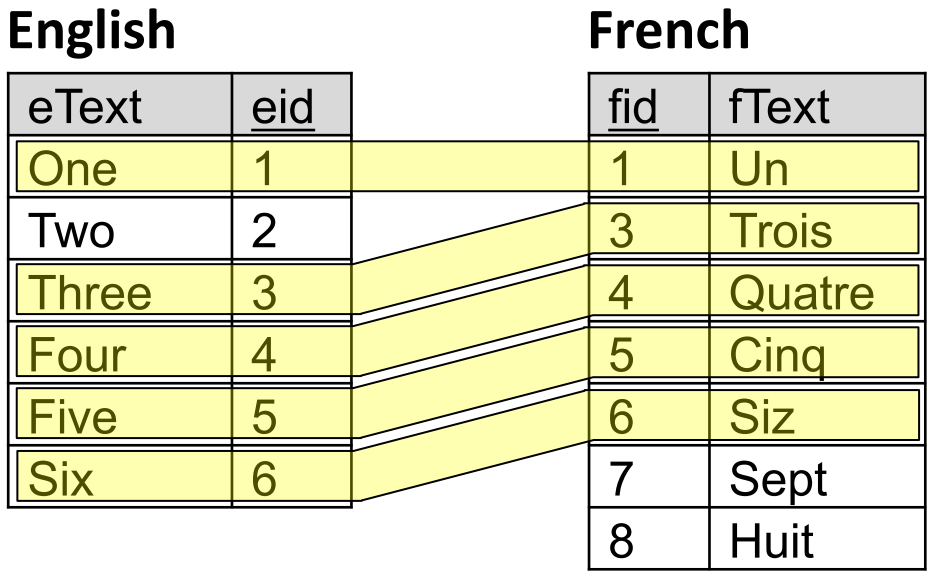
SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Another look at Outer Joins

```
SELECT  *
FROM    English FULL JOIN French
ON      English.eid = French.fid
```

```
SELECT    etext,eid, fid, ftext
FROM      English INNER JOIN French
ON        English.eid = French.fid
UNION ALL
SELECT    etext, eid, NULL, NULL
FROM      English
WHERE     NOT EXISTS(
          SELECT *
          FROM French
          WHERE eid=fid)
UNION ALL
SELECT    NULL, NULL, fid, ftext
FROM      French
WHERE     NOT EXISTS(
          SELECT *
          FROM English
          WHERE eid=fid)
```

anti-join



English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

etext	eid	fid	ftext
One	1	1	Un
Two	2	NULL	NULL
Three	3	3	Trois
Four	4	4	Quatre
Five	5	5	Cinq
Six	6	6	Siz
NULL	NULL	7	Sept
NULL	NULL	8	Huit

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Are these two queries equivalent?

316

Q_2: Find all companies that make only products with price < 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE   C.cid NOT IN ( SELECT  P.cid
                       FROM    Product P
                       WHERE   P.price >= 25)
```

Is the following query identical to the one above?

```
SELECT C.name
FROM Company C, Product P
WHERE C.cid=P.cid
GROUP BY cname
HAVING MAX(P.price) < 25
```

Are these two queries equivalent?


316

Q_2: Find all companies that make only products with price < 25

```
SELECT DISTINCT C.cname
FROM    Company C
WHERE   C.cid NOT IN ( SELECT  P.cid
                       FROM    Product P
                       WHERE   P.price >= 25)
```

!

Is the following query identical to the one above?

```
SELECT C.name
FROM Company C, Product P
WHERE C.cid=P.cid
GROUP BY cname
HAVING MAX(P.price) < 25
```

Almost, but not really. The upper query would return a company that has no product, the one below would not.

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Outer Joins, Coalesce, and non-associativity

Coalesce function

333

M

a
1
2

N

a
2
3

```
SELECT M.a, N.a, COALESCE(M.a, N.a) as b
FROM M
FULL JOIN N
ON M.a = N.a
```

COALESCE: takes first non-NULL value,

(N N 9 8)
1 2 3 4

```
SELECT COALESCE(1, NULL)
```

```
SELECT COALESCE(NULL, 3)
```

```
SELECT COALESCE(1, 2)
```

```
SELECT COALESCE(NULL, NULL)
```

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Also see use of COALESCE across programming languages: https://en.wikipedia.org/wiki/Null_coalescing_operator

Coalesce function

M

a
1
2

N

a
2
3

```sql
SELECT M.a, N.a, COALESCE(M.a, N.a) as b
FROM M
FULL JOIN N
ON M.a = N.a
```

COALESCE: takes first non-NULL value,

$C(x,y,z)=C(x,C(y,z))=C(C(x,y),z)$

Result

M.a	N.a	b

? ?

```sql
SELECT COALESCE(1, NULL)
```
→ 1

```sql
SELECT COALESCE(NULL, 3)
```
→ 3

```sql
SELECT COALESCE(1, 2)
```
→ 1

```sql
SELECT COALESCE(NULL, NULL)
```
→ NULL

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Also see use of COALESCE across programming languages: https://en.wikipedia.org/wiki/Null_coalescing_operator

Coalesce function

M

a
1
2

N

a
2
3

```
SELECT M.a, N.a, COALESCE(M.a, N.a) as b
FROM M
FULL JOIN N
ON M.a = N.a
```

COALESCE: takes first non-NULL value,

$C(x,y,z)=C(x,C(y,z))=C(C(x,y),z)$

Result

M.a	N.a	b
1	NULL	1
2	2	2
NULL	3	3

`SELECT COALESCE(1, NULL)` ⇨ 1

`SELECT COALESCE(NULL, 3)` ⇨ 3

`SELECT COALESCE(1, 2)` ⇨ 1

`SELECT COALESCE(NULL, NULL)` ⇨ NULL

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Also see use of COALESCE across programming languages: https://en.wikipedia.org/wiki/Null_coalescing_operator

Coalesce, Natural Outer Join, Union

333

M

a
1
2

N

a
2
3

```
SELECT *
FROM M
NATURAL FULL JOIN N
```

Result

a
1
2
3

Natural full join models "coalesce"

Join vs. Union – it is actually the same:
Union is a special case of a join ☺
(under set semantics)

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Do we need "union"?

R

a
1
2

U

a
2
3
4

305

```
SELECT a
FROM   R
UNION
SELECT a
FROM   U
```

a
1
2
3
4

?

Do we need "union"?

305

R

a
1
2

U

a
2
3
4

Coalse (R.A, U.A)

1	N
2	2
N	3
N	4

```
SELECT a
FROM   R
UNION
SELECT a
FROM   U
```

a
1
2
3
4

```
SELECT COALESCE(R.a,U.a) as a
FROM   R FULL JOIN U
on     R.a = U.a
```

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Quick recap: Commutativity & Associativity

Multiplication

$$3 \bullet 2 \bullet 4 = 24$$

?

Multiplication is associative ☺

Matrix multiplication

$$\begin{array}{|c|c|}\hline 1 & 1 \\ \hline 2 & 3 \\ \hline \end{array} \bullet \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} \bullet \begin{array}{|c|}\hline 3 \\ \hline 1 \\ \hline \end{array} =$$

Quick recap: Commutativity & Associativity

Multiplication

$$[3 \bullet 2] \bullet 4 = 24$$

Order of operations can be exchanged:

$$3 \bullet [2 \bullet 4] = 24$$

Multiplication is associative ☺

?

and commutative ☺

Matrix multiplication

$$\begin{bmatrix} 1 & 1 \\ 2 & 3 \end{bmatrix} \bullet \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \bullet \begin{bmatrix} 3 \\ 1 \end{bmatrix} =$$

Quick recap: Commutativity & Associativity

Multiplication

$$(3 \bullet 2) \bullet 4 = 24$$

Order of operations can be exchanged:

$$3 \bullet (2 \bullet 4) = 24$$

Multiplication is associative ☺

Order of operands can be exchanged:

$$4 \bullet 2$$

and commutative ☺

Matrix multiplication

$$\begin{bmatrix}1 & 1\\ 2 & 3\end{bmatrix} \bullet \begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} \bullet \begin{bmatrix}3\\ 1\end{bmatrix} =$$

?

Quick recap: Commutativity & Associativity

Multiplication

$$(3 \bullet 2) \bullet 4 = 24$$

Order of operations can be exchanged:

$$3 \bullet (2 \bullet 4) = 24$$

Multiplication is associative ☺

Order of operands can be exchanged:

$$4 \bullet 2$$

and commutative ☺

Matrix multiplication

$$\left(\begin{bmatrix}1&1\\2&3\end{bmatrix} \bullet \begin{bmatrix}1&2\\3&4\end{bmatrix}\right) \bullet \begin{bmatrix}3\\1\end{bmatrix} = \begin{bmatrix}18\\49\end{bmatrix}$$

$$\begin{bmatrix}1&1\\2&3\end{bmatrix} \bullet \left(\begin{bmatrix}1&2\\3&4\end{bmatrix} \bullet \begin{bmatrix}3\\1\end{bmatrix}\right) = \begin{bmatrix}18\\49\end{bmatrix}$$

Matrix multipl. is associative ☺

$$\begin{bmatrix}3\\1\end{bmatrix} \bullet \begin{bmatrix}1&2\\3&4\end{bmatrix}$$

#col ≠ #row

... but *not* commutative ☹

It turns out this is mainly a problem of syntax, not semantics. Einstein notation (and similar more recent extensions like "EINSUM") solves that. See e.g. Laue et al. *A Simple and Efficient Tensor Calculus*. AAAI 2020. https://arxiv.org/abs/2010.03313 . Alternatively, think about the relational join operator as a commutative notation for sparse matrix multiplication (also Cartesian product under named or unnamed perspective)

The power of associativity

Option 1: $\left(\begin{bmatrix}1&1\\2&3\end{bmatrix} \bullet \begin{bmatrix}1&2\\3&4\end{bmatrix}\right) \bullet \begin{bmatrix}3\\1\end{bmatrix} = \begin{bmatrix}18\\49\end{bmatrix}$

Option 2: $\begin{bmatrix}1&1\\2&3\end{bmatrix} \bullet \left(\begin{bmatrix}1&2\\3&4\end{bmatrix} \bullet \begin{bmatrix}3\\1\end{bmatrix}\right) = \begin{bmatrix}18\\49\end{bmatrix}$

Which option would you choose to evaluate this matrix multiplication ?

The power of associativity

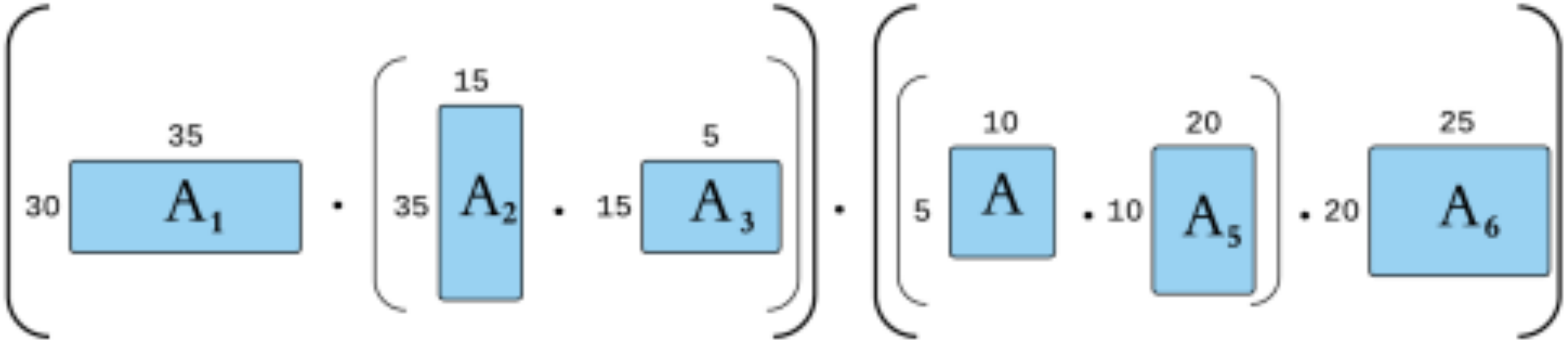
Option 1: $\underbrace{\left(\begin{bmatrix}1&1\\2&3\end{bmatrix} \bullet \begin{bmatrix}1&2\\3&4\end{bmatrix}\right)}_{\begin{bmatrix}4&6\\11&16\end{bmatrix}} \bullet \begin{bmatrix}3\\1\end{bmatrix} = \begin{bmatrix}18\\49\end{bmatrix}$

Option 2: $\begin{bmatrix}1&1\\2&3\end{bmatrix} \bullet \underbrace{\left(\begin{bmatrix}1&2\\3&4\end{bmatrix} \bullet \begin{bmatrix}3\\1\end{bmatrix}\right)}_{\begin{bmatrix}5\\13\end{bmatrix}} = \begin{bmatrix}18\\49\end{bmatrix}$

All variants give the same result. But some are faster.
Intuition: we like to have small intermediate result sizes!

Matrix chain multiplication

Given n matrices, what is the optimal sequence to multiply them? ?



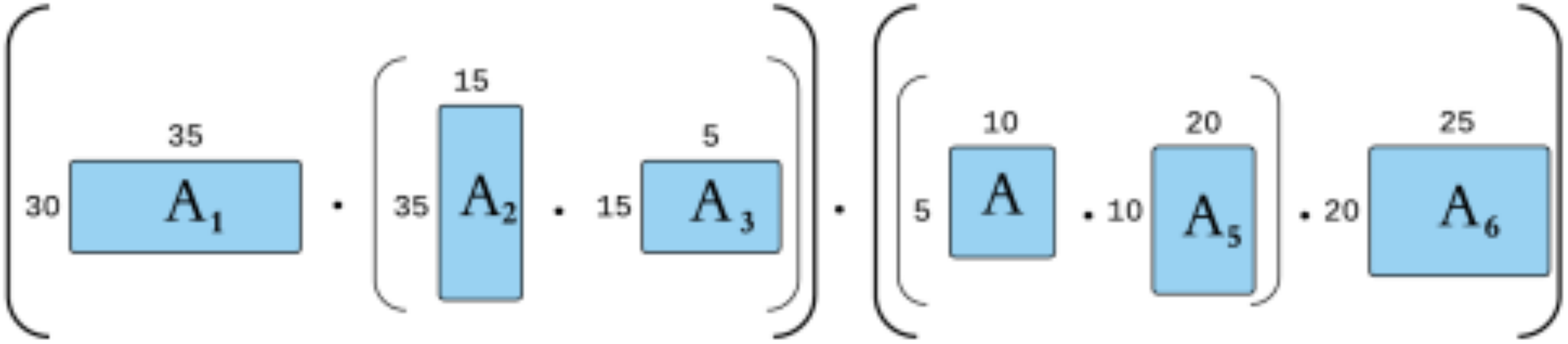
This is an example optimal factorization. ?
What is its cost?

See also https://en.wikipedia.org/wiki/Catalan_number , https://en.wikipedia.org/wiki/Matrix_chain_multiplication , https://en.wikipedia.org/wiki/Matrix_multiplication#Associativity

Source figure: https://bruceoutdoors.wordpress.com/2015/11/24/matrix-chain-multiplication-with-c-code-part-3-extracting-the-sequence/

Matrix chain multiplication

Given n matrices, what is the optimal sequence to multiply them? ?



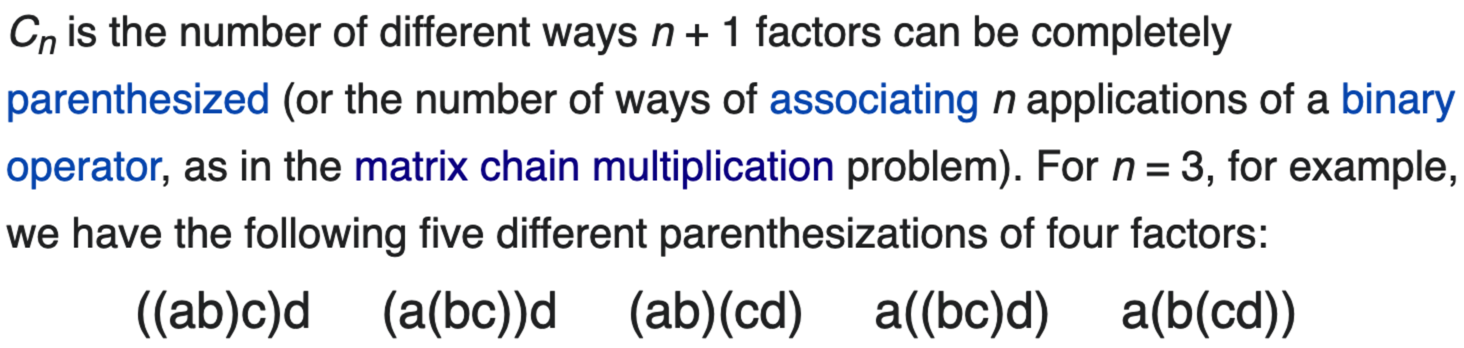
This is an example optimal factorization. ?
What is its cost?

MinCost: (30*35*5 + (35*15*5)) + 30*5*25 + (5*10*20) + 5*20*25)

Nave method: all possible way to place closed parentheses: "Catalan numbers"

Via Dynamic programming: $O(n^3)$

Best known: $O(n \log n)$



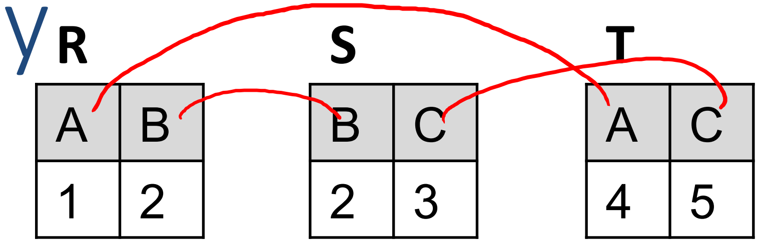
C_n is the number of different ways $n + 1$ factors can be completely parenthesized (or the number of ways of associating n applications of a binary operator, as in the matrix chain multiplication problem). For $n = 3$, for example, we have the following five different parenthesizations of four factors:

((ab)c)d (a(bc))d (ab)(cd) a((bc)d) a(b(cd))

See also https://en.wikipedia.org/wiki/Catalan_number , https://en.wikipedia.org/wiki/Matrix_chain_multiplication , https://en.wikipedia.org/wiki/Matrix_multiplication#Associativity
Source figure: https://bruceoutdoors.wordpress.com/2015/11/24/matrix-chain-multiplication-with-c-code-part-3-extracting-the-sequence/

Commutativity & Associativity

333



R

A	B
1	2

S

B	C
2	3

T

A	C
4	5

Outer joins

```
SELECT A, B, C
FROM (R
NATURAL FULL JOIN S)
NATURAL FULL JOIN T
```

```
SELECT  A, B, C
FROM R
NATURAL FULL JOIN (S
NATURAL FULL JOIN T)
```

Result

⇨ ?

Result

⇨ ?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Commutativity & Associativity

333

R

A	B
1	2

S

B	C
2	3

T

A	C
4	5

Outer joins

A	B	C
1	2	3

```
SELECT A, B, C
FROM (R
NATURAL FULL JOIN S)
NATURAL FULL JOIN T
```

```
SELECT  A, B, C
FROM R
NATURAL FULL JOIN (S
NATURAL FULL JOIN T)
```

Result

⇨ ?

Result

⇨ ?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Commutativity & Associativity

333

R

A	B
1	2

S

B	C
2	3

T

A	C
4	5

Outer joins

A	B	C
1	2	3

```
SELECT A, B, C
FROM (R
NATURAL FULL JOIN S)
NATURAL FULL JOIN T
```

```
SELECT  A, B, C
FROM R
NATURAL FULL JOIN (S
NATURAL FULL JOIN T)
```

Result

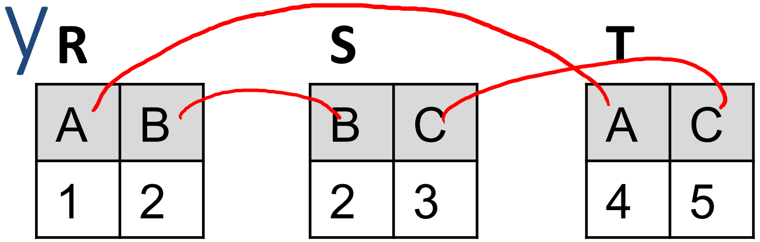
A	B	C
1	2	3
4	NULL	5

Result

?

Commutativity & Associativity

333



R

A	B
1	2

S

B	C
2	3

T

A	C
4	5

Outer joins

A	B	C
1	2	3

```
SELECT A, B, C
FROM (R
NATURAL FULL JOIN S)
NATURAL FULL JOIN T
```

Result

A	B	C
1	2	3
4	NULL	5

A	B	C
1	2	3
4	⊥	5

```
SELECT  A, B, C
FROM R
NATURAL FULL JOIN (S
NATURAL FULL JOIN T)
```

Result

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Commutativity & Associativity


333

R

A	B
1	2

S

B	C
2	3

T

A	C
4	5

Outer joins

A	B	C
1	2	3

```
SELECT A, B, C
FROM (R
NATURAL FULL JOIN S)
NATURAL FULL JOIN T
```

Result

A	B	C
1	2	3
4	NULL	5

A	B	C
1	2	3
4	⊥	5

```
SELECT  A, B, C
FROM R
NATURAL FULL JOIN (S
NATURAL FULL JOIN T)
```

Result

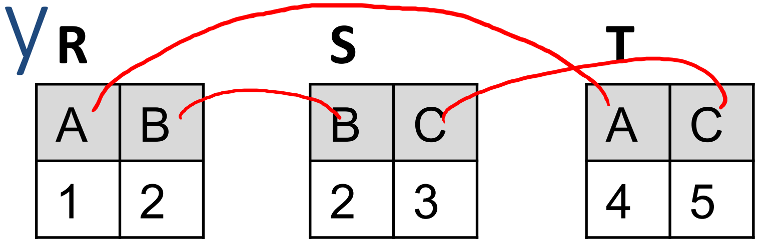
A	B	C
1	2	NULL
NULL	2	3
4	NULL	5

Thus outer joins are not associative! (but they are commutative)

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Commutativity & Associativity

333



R

A	B
1	2

S

B	C
2	3

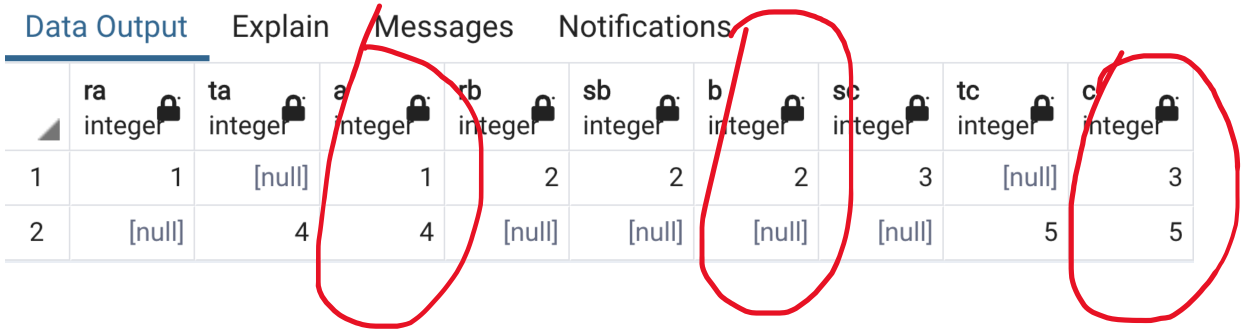
T

A	C
4	5

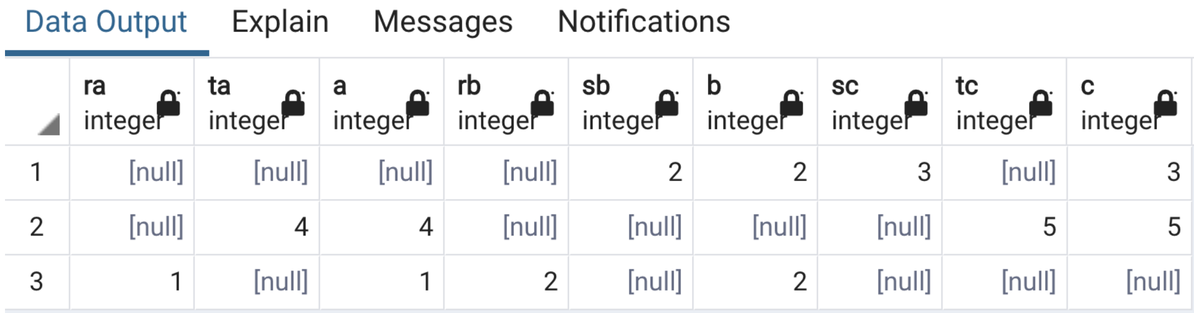
Outer joins

```
SELECT R.a RA, T.a TA, coalesce(R.a, T.a) a,
       R.b RB, S.b SB, coalesce(R.b, S.b) b,
       S.c SC, T.c TC, coalesce(S.c, T.c) c
FROM (R
FULL JOIN S on R.B=S.B)
FULL JOIN T on S.C=T.C AND R.A = T.A
```

```
SELECT R.a RA, T.a TA, coalesce(R.a, T.a) a,
       R.b RB, S.b SB, coalesce(R.b, S.b) b,
       S.c SC, T.c TC, coalesce(S.c, T.c) c
FROM R
FULL JOIN (S
FULL JOIN T on S.C=T.C) on R.B=S.B AND R.A = T.A
```



	ra integer	ta integer	a integer	rb integer	sb integer	b integer	sc integer	tc integer	c integer
1	1	[null]	1	2	2	2	3	[null]	3
2	[null]	4	4	[null]	[null]	[null]	[null]	5	5



	ra integer	ta integer	a integer	rb integer	sb integer	b integer	sc integer	tc integer	c integer
1	[null]	[null]	[null]	[null]	2	2	3	[null]	3
2	[null]	4	4	[null]	[null]	[null]	[null]	5	5
3	1	[null]	1	2	[null]	2	[null]	[null]	[null]

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example: Data Sources on Tourist Information

Climates

Country	Climate
Canada	diverse
Bahamas	tropical
UK	temperate

Accommodations

Country	City	Hotel	Stars
Canada	Toronto	Plaza	4
Canada	London	Ramada	3
Bahamas	Nassau	Hilton	

Sites

Country	City	Site
Canada	London	Air show
Canada		Mount Logan
UK	London	Buckingham
UK	London	Hyde Park

```sql
SELECT  *
FROM (Accommodations
NATURAL FULL JOIN Climates)
NATURAL FULL JOIN Sites
```

Result

⇨

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example from: Cohen, Fadida, Kanza, Kimelfeld, Sagiv. "Full Disjunctions: Polynomial-Delay Iterators in Action", VLDB 2006. http://vldb.org/conf/2006/p739-cohen.pdf

Example: Data Sources on Tourist Information

Climates

Country	Climate
Canada	diverse
Bahamas	tropical
UK	temperate

Accommodations

Country	City	Hotel	Stars
Canada	Toronto	Plaza	4
Canada	London	Ramada	3
Bahamas	Nassau	Hilton	

Sites

Country	City	Site
Canada	London	Air show
Canada		Mount Logan
UK	London	Buckingham
UK	London	Hyde Park

```
SELECT  *
FROM (Accommodations
NATURAL FULL JOIN Climates)
NATURAL FULL JOIN Sites
```

Result

Country	City	Climate	Hotel	Stars	Site
Canada	Toronto	diverse	Plaza	4	
Canada	London	diverse	Ramada	3	Air Show
Canada					Mount Logan
UK	London				Buckingham
UK	London				Hyde Park
UK		temperate			
Bahamas	Nassau	Tropical	Hilton		

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Example from: Cohen, Fadida, Kanza, Kimelfeld, Sagiv. "Full Disjunctions: Polynomial-Delay Iterators in Action", VLDB 2006. http://vldb.org/conf/2006/p739-cohen.pdf
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Example: Data Sources on Tourist Information

Climates

Country	Climate
Canada	diverse
Bahamas	tropical
UK	temperate

Accommodations

Country	City	Hotel	Stars
Canada	Toronto	Plaza	4
Canada	London	Ramada	3
Bahamas	Nassau	Hilton	

Sites

Country	City	Site
Canada	London	Air show
~~Canada~~		~~Mount Logan~~
UK	London	Buckingham
UK	London	Hyde Park

```
SELECT  *
FROM (Accommodations
NATURAL FULL JOIN Climates)
NATURAL FULL JOIN Sites
```

Result

Country	City	Climate	Hotel	Stars	Site
Canada	Toronto	diverse	Plaza	4	
Canada	London	diverse	Ramada	3	Air Show
Canada					Mount Logan
UK	London				Buckingham
UK	London				Hyde Park
UK		temperate			
Bahamas	Nassau	Tropical	Hilton		

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example from: Cohen, Fadida, Kanza, Kimelfeld, Sagiv. "Full Disjunctions: Polynomial-Delay Iterators in Action", VLDB 2006. http://vldb.org/conf/2006/p739-cohen.pdf

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Example: Data Sources on Tourist Information

Climates

Country	Climate
Canada	diverse
Bahamas	tropical
~~UK~~	~~temperate~~

Accommodations

Country	City	Hotel	Stars
Canada	Toronto	Plaza	4
Canada	London	Ramada	3
Bahamas	Nassau	Hilton	

Sites

Country	City	Site
Canada	London	Air show
Canada		Mount Logan
UK	London	Buckingham
UK	London	Hyde Park

```
SELECT  *
FROM (Accommodations
NATURAL FULL JOIN Climates)
NATURAL FULL JOIN Sites
```

Result

Country	City	Climate	Hotel	Stars	Site
Canada	Toronto	diverse	Plaza	4	
Canada	London	diverse	Ramada	3	Air Show
Canada					Mount Logan
UK	London				Buckingham
UK	London				Hyde Park
UK		temperate			
Bahamas	Nassau	Tropical	Hilton		

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example from: Cohen, Fadida, Kanza, Kimelfeld, Sagiv. "Full Disjunctions: Polynomial-Delay Iterators in Action", VLDB 2006. http://vldb.org/conf/2006/p739-cohen.pdf

Example: Data Sources on Tourist Information

Climates

Country	Climate
~~Canada~~	~~diverse~~
Bahamas	tropical
UK	temperate

Accommodations

Country	City	Hotel	Stars
Canada	Toronto	Plaza	4
Canada	London	Ramada	3
Bahamas	Nassau	Hilton	

Sites

Country	City	Site
Canada	London	Air show
~~Canada~~		~~Mount Logan~~
UK	London	Buckingham
UK	London	Hyde Park

```
SELECT  *
FROM Accommodations
NATURAL FULL JOIN (Climates
NATURAL FULL JOIN Sites)
```

Result

Country	City	Climate	Hotel	Stars	Site
Canada	Toronto		Plaza	4	
Canada	London	diverse	Ramada	3	Air Show
Canada		diverse			Mount Logan
UK	London	temperate			Buckingham
UK	London	temperate			Hyde Park
Bahamas		Tropical			
Bahamas	Nassau		Hilton		

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Example from: Cohen, Fadida, Kanza, Kimelfeld, Sagiv. "Full Disjunctions: Polynomial-Delay Iterators in Action", VLDB 2006. http://vldb.org/conf/2006/p739-cohen.pdf
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Example: Data Sources on Tourist Information

Climates

Country	Climate
Canada	diverse
Bahamas	tropical
UK	temperate

Accommodations

Country	City	Hotel	Stars
Canada	Toronto	Plaza	4
Canada	London	Ramada	3
Bahamas	Nassau	Hilton	

Sites

Country	City	Site
Canada	London	Air show
Canada		Mount Logan
UK	London	Buckingham
UK	London	Hyde Park

```
SELECT  *
FROM Accommodations
NATURAL FULL JOIN (Climates
NATURAL FULL JOIN Sites)
```

Result

Country	City	Climate	Hotel	Stars	Site
Canada	Toronto		Plaza	4	
Canada	London	diverse	Ramada	3	Air Show
Canada		diverse			Mount Logan
UK	London	temperate			Buckingham
UK	London	temperate			Hyde Park
Bahamas		Tropical			
Bahamas	Nassau		Hilton		

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example from: Cohen, Fadida, Kanza, Kimelfeld, Sagiv. "Full Disjunctions: Polynomial-Delay Iterators in Action", VLDB 2006. http://vldb.org/conf/2006/p739-cohen.pdf

Full disjunction

Climates

Country	Climate
Canada	diverse
Bahamas	tropical
UK	temperate

Accommodations

Country	City	Hotel	Stars
Canada	Toronto	Plaza	4
Canada	London	Ramada	3
Bahamas	Nassau	Hilton	

Sites

Country	City	Site
Canada	London	Air show
Canada		Mount Logan
UK	London	Buckingham
UK	London	Hyde Park

```
SELECT  *
FROM FULL DISJUNCTION(Climates,
(Accommodations, Sites)
```

FD: variation of the join operator that maximally combines join consistent tuples from connected relations, while preserving all information in the relations.

Not available in SQL! We may discuss later in class in more detail (or skip this year)

Result

Country	City	Climate	Hotel	Stars	Site
Canada	Toronto	diverse	Plaza	4	
Canada	London	diverse	Ramada	3	Air Show
Canada		diverse			Mount Logan
UK	London	temperate			Buckingham
UK	London	temperate			Hyde Park
Bahamas	Nassau	tropical	Hilton		

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example from: Cohen, Fadida, Kanza, Kimelfeld, Sagiv. "Full Disjunctions: Polynomial-Delay Iterators in Action", VLDB 2006. http://vldb.org/conf/2006/p739-cohen.pdf

Full disjunction: definition

Climates

t_1 t_2

Country	Climate
Canada	diverse
Bahamas	tropical
UK	temperate

t_3

Accommodations

Country	City	Hotel	Stars
Canada	Toronto	Plaza	4
Canada	London	Ramada	3
Bahamas	Nassau	Hilton	

Sites

t_4

Country	City	Site
Canada	London	Air show
Canada		Mount Logan
UK	London	Buckingham
UK	London	Hyde Park

- Two tuples (max one from each relation) are join consistent if they agree on common attributes, e.g. t_1/t_2, t_3/t_4. A set of tuples is join consistent if every pair is join consistent.
- Set of tuples (max one from each relation) is connected if the schema is connected, thus share attributes
- A tuple is in the Full disjunction if it is the inner join from tuples that are connected, join consistent, and there is no superset with both conditions (related to "subsumption").

B

1	2
1	NULL

SUBSUMTION

Result

Country	City	Climate	Hotel	Stars	Site
Canada	Toronto	diverse	Plaza	4	
Canada	London	diverse	Ramada	3	Air Show
Canada		diverse			Mount Logan
UK	London	temperate			Buckingham
UK	London	temperate			Hyde Park
Bahamas	Nassau	tropical	Hilton		

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Example from: Cohen, Fadida, Kanza, Kimelfeld, Sagiv. "Full Disjunctions: Polynomial-Delay Iterators in Action", VLDB 2006. http://vldb.org/conf/2006/p739-cohen.pdf

Outline: T1-U1: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Union and Theta Joins
  - Nulls & Outer joins
  - Top-k
  - [Recursion: moved to T1-U4: Datalog]

Sorting & Top-*k* evaluation with SQL

605

R

A	*B*
1	0
2	0
3	0
4	1

S

B	*C*
0	1
0	1
0	1
0	2

T

C	*D*
1	1
1	2
2	3
2	4

```
select  A, R.B, S.C, D

from    R, S, T
where   R.B=S.B and S.C=T.C
```

Result

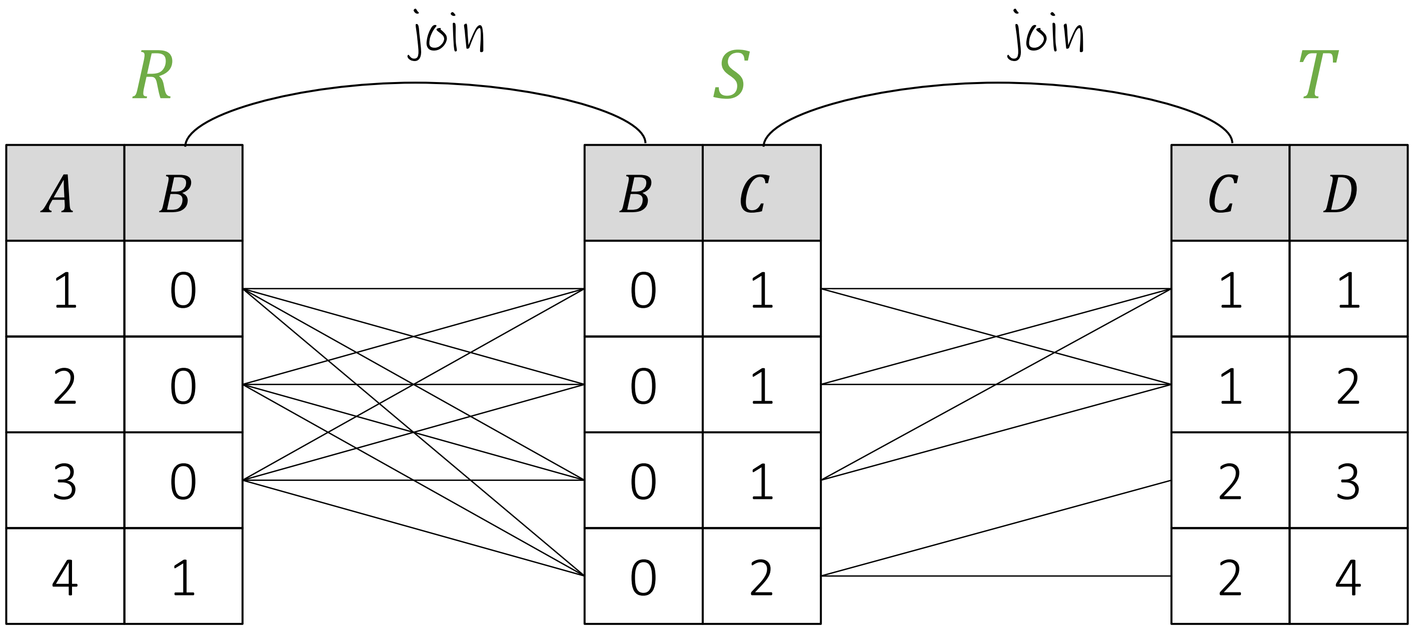
How many results
do we get?

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Sorting & Top-*k* evaluation with SQL

605



A	*B*
1	0
2	0
3	0
4	1

B	*C*
0	1
0	1
0	1
0	2

C	*D*
1	1
1	2
2	3
2	4

```
select   A, R.B, S.C, D

from     R, S, T
where    R.B=S.B and S.C=T.C
```

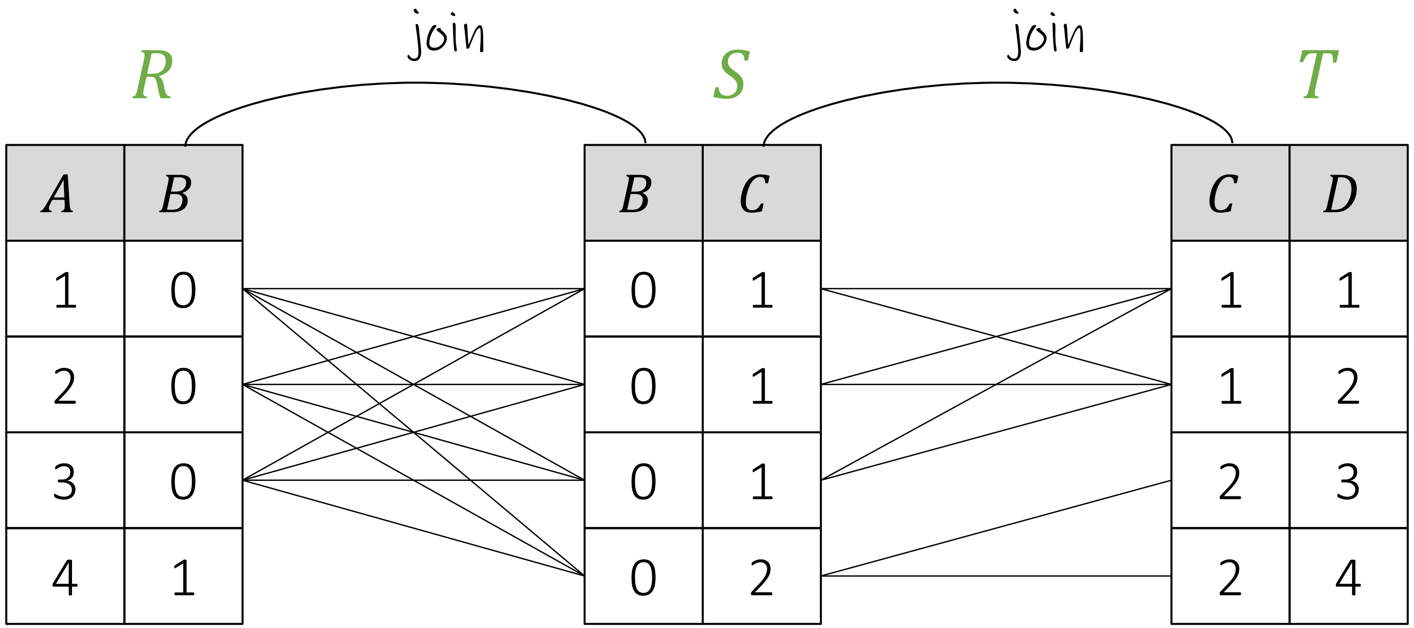
Result

How many results do we get?

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Sorting & Top-*k* evaluation with SQL



R

A	*B*
1	0
2	0
3	0
4	1

S

B	*C*
0	1
0	1
0	1
0	2

T

C	*D*
1	1
1	2
2	3
2	4

```
select   A, R.B, S.C, D

from     R, S, T
where    R.B=S.B and S.C=T.C
```

Result

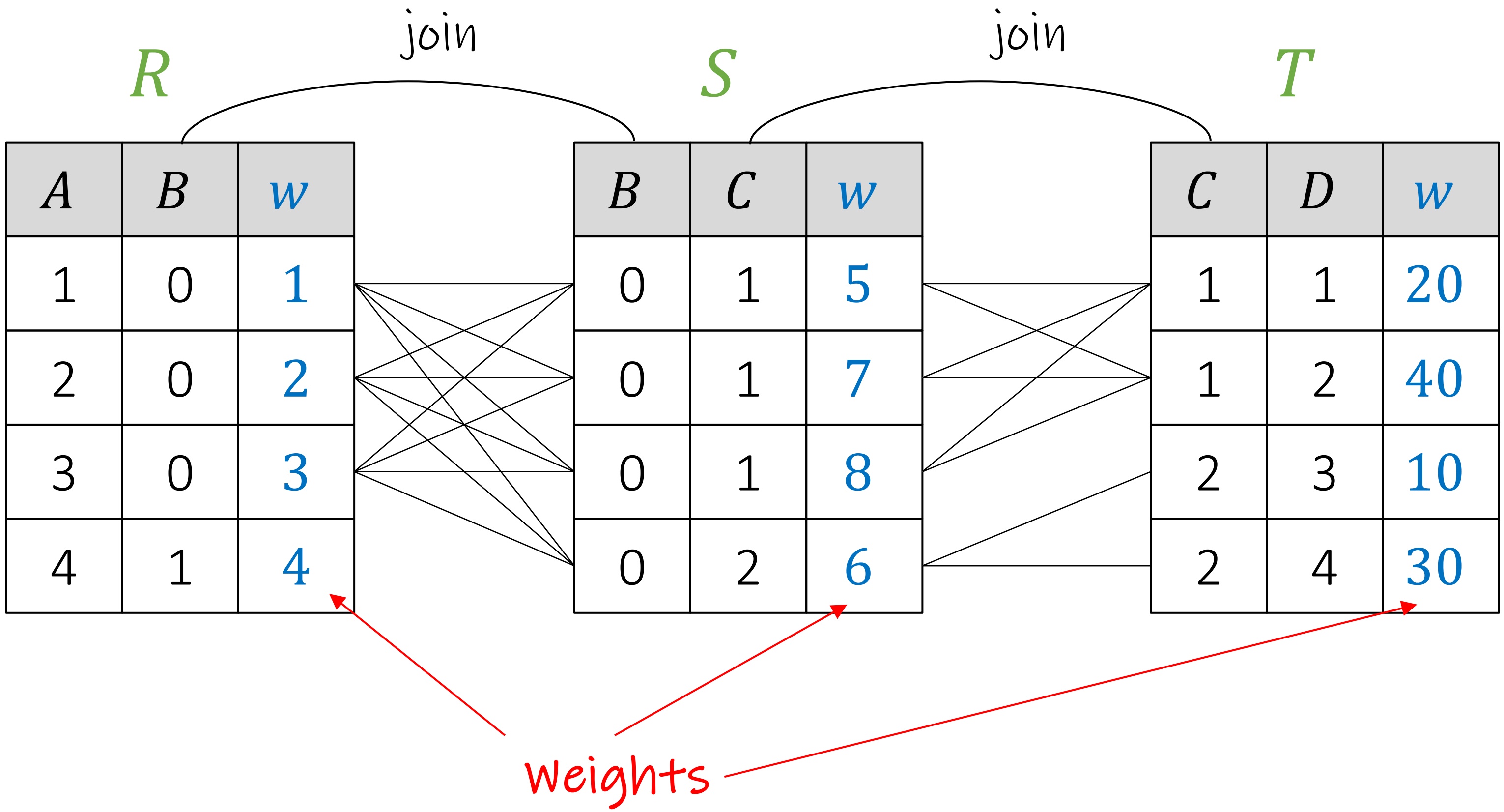
A	B	C	D
1	0	2	3
2	0	2	3
3	0	2	3
1	0	1	1
2	0	1	1
3	0	1	1
1	0	1	1
...	...	...	...

24 total results

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Sorting & Top-*k* evaluation with SQL

605



R

A	*B*	*w*
1	0	1
2	0	2
3	0	3
4	1	4

S

B	*C*	*w*
0	1	5
0	1	7
0	1	8
0	2	6

T

C	*D*	*w*
1	1	20
1	2	40
2	3	10
2	4	30

```
select   A, R.B, S.C, D,
         R.w + S.w + T.w as weight
from     R, S, T
where    R.B=S.B and S.C=T.C
order by weight ASC
```

Result

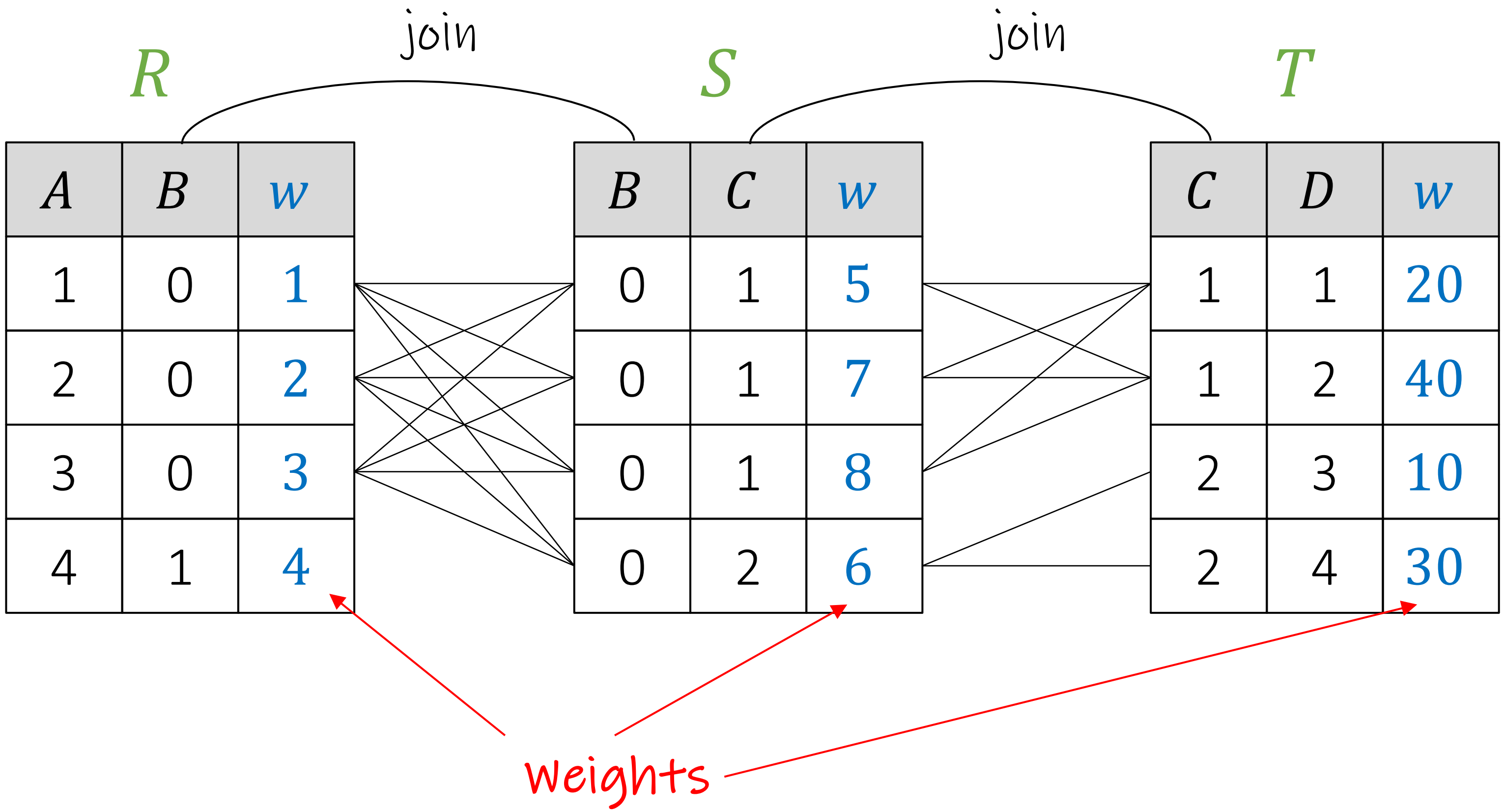
A	B	C	D	weight

What do we get now?

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Sorting & Top-*k* evaluation with SQL



A	*B*	*w*
1	0	1
2	0	2
3	0	3
4	1	4

B	*C*	*w*
0	1	5
0	1	7
0	1	8
0	2	6

C	*D*	*w*
1	1	20
1	2	40
2	3	10
2	4	30

```
select   A, R.B, S.C, D,
         R.w + S.w + T.w as weight
from     R, S, T
where    R.B=S.B and S.C=T.C
order by weight ASC
```

Return all 24 results in order of sum of weights

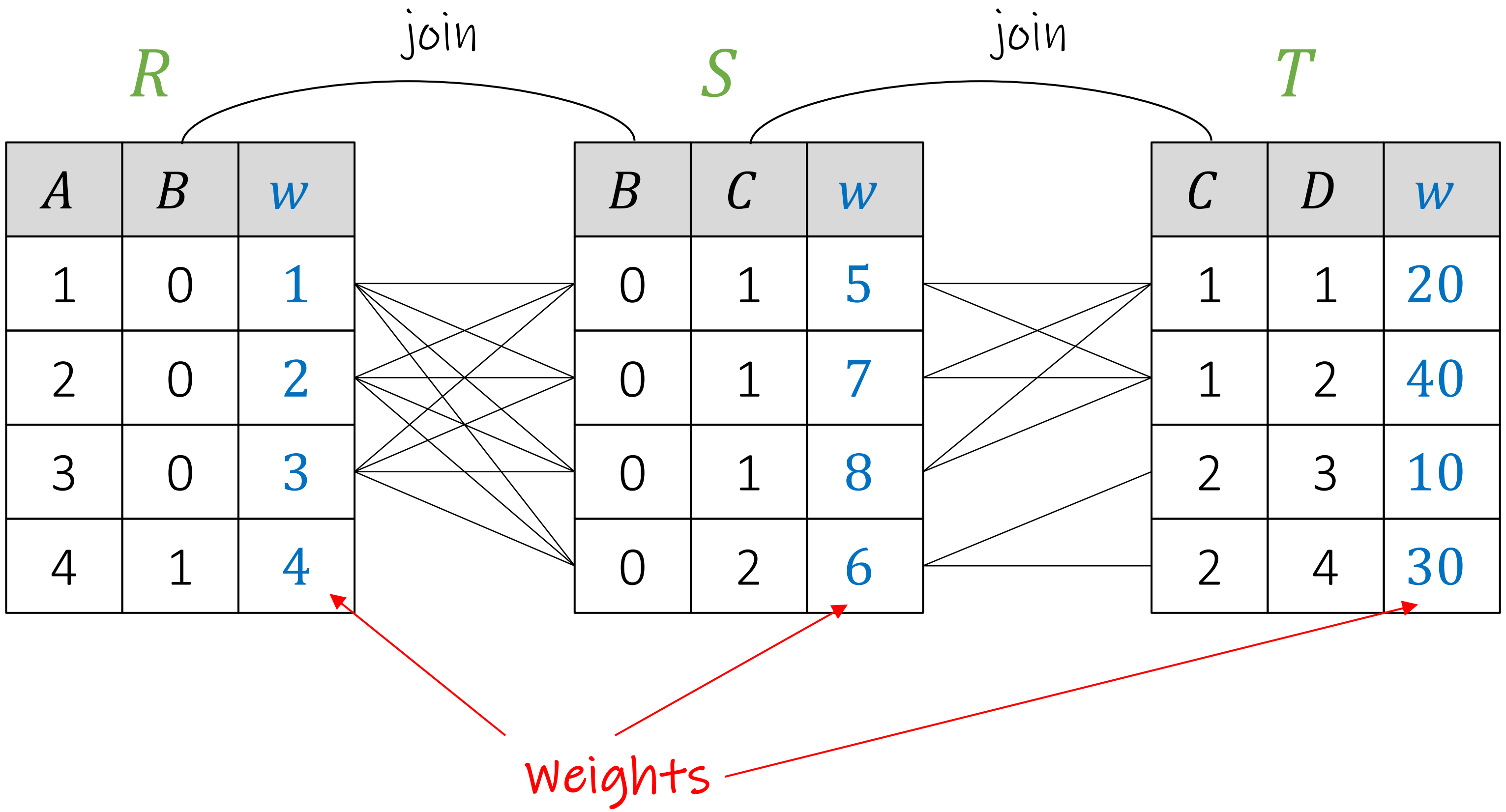
Result

A	B	C	D	weight
1	0	2	3	17
2	0	2	3	18
3	0	2	3	19
1	0	1	1	26
2	0	1	1	27
3	0	1	1	28
1	0	1	1	28
…	…	…	…	…

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Sorting & Top-*k* evaluation with SQL

605



R

A	*B*	*w*
1	0	1
2	0	2
3	0	3
4	1	4

S

B	*C*	*w*
0	1	5
0	1	7
0	1	8
0	2	6

T

C	*D*	*w*
1	1	20
1	2	40
2	3	10
2	4	30

```
select   A, R.B, S.C, D,
         R.w + S.w + T.w as weight
from     R, S, T
where    R.B=S.B and S.C=T.C
order by weight ASC
limit    6
```

What do we get now? ?

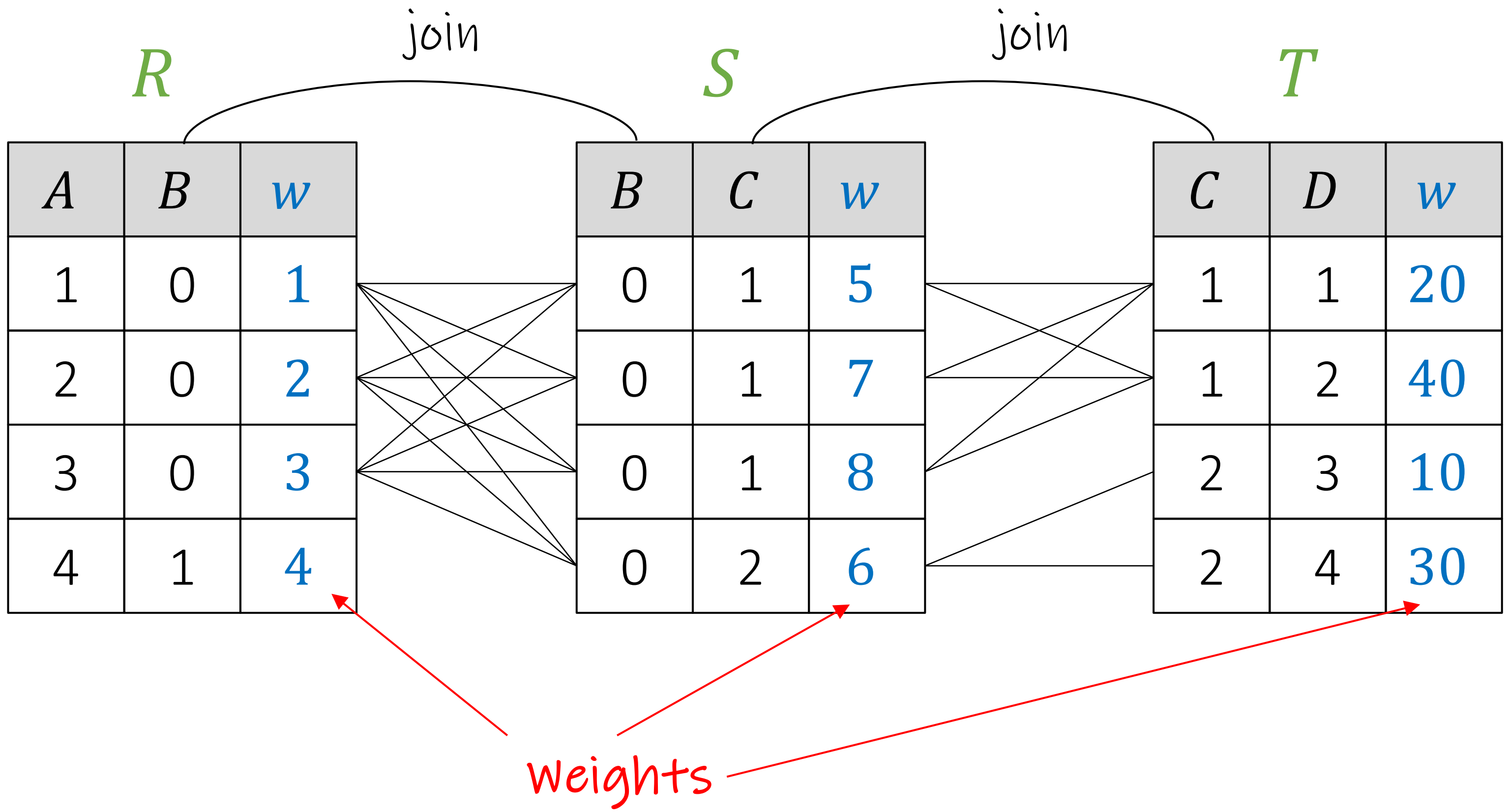
Result

A	B	C	D	weight
1	0	2	3	17
2	0	2	3	18
3	0	2	3	19
1	0	1	1	26
2	0	1	1	27
3	0	1	1	28
1	0	1	1	28
...	...	...	...	...

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Sorting & Top-*k* evaluation with SQL

605



R

A	*B*	*w*
1	0	1
2	0	2
3	0	3
4	1	4

S

B	*C*	*w*
0	1	5
0	1	7
0	1	8
0	2	6

T

C	*D*	*w*
1	1	20
1	2	40
2	3	10
2	4	30

```
select   A, R.B, S.C, D,
         R.w + S.w + T.w as weight
from     R, S, T
where    R.B=S.B and S.C=T.C
order by weight ASC
limit    6
```

What do we get now?

Result

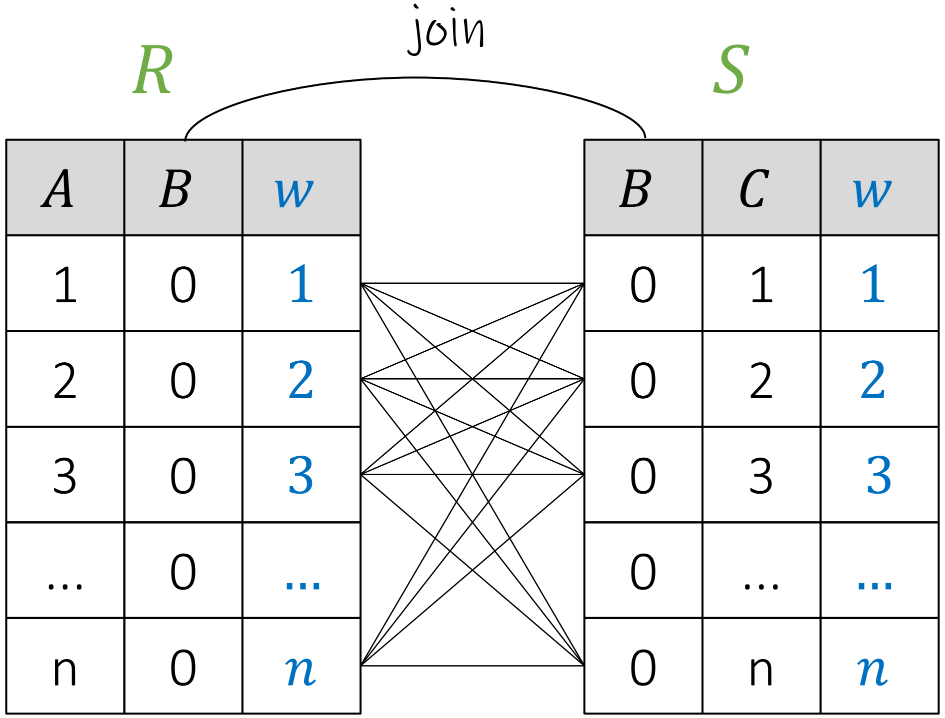
A	B	C	D	weight
1	0	2	3	17
2	0	2	3	18
3	0	2	3	19
1	0	1	1	26
2	0	1	1	27
3	0	1	1	28
1	0	1	1	28
...	...	...	...	...

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Wolfgang Gatterbauer. Principles of scalable data management: https://northeastern-datalab.github.io/cs7240/

Top-*k* is evaluated inefficiently by modern DBMS's

606



R

A	*B*	*w*
1	0	1
2	0	2
3	0	3
…	0	…
n	0	*n*

S

B	*C*	*w*
0	1	1
0	2	2
0	3	3
0	…	…
0	n	*n*

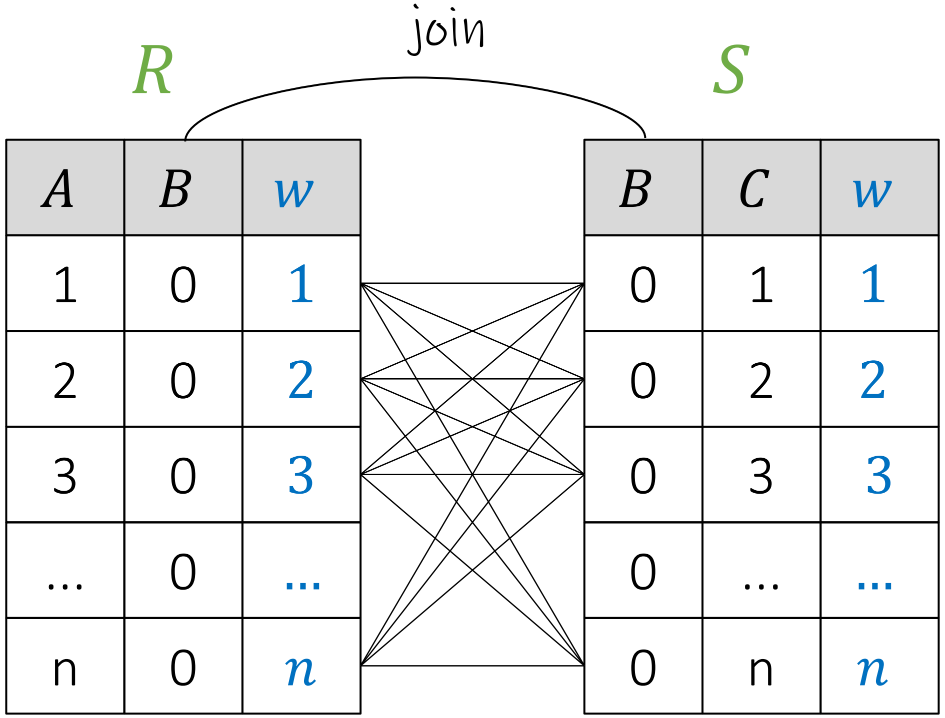
```
select   A, R.B, S.C,
         R.w + S.w as weight
from     R, S
where    R.B=S.B
order by weight ASC
limit    1
```

What will this query return?

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Top-*k* is evaluated inefficiently by modern DBMS's



R

A	B	w
1	0	1
2	0	2
3	0	3
…	0	…
n	0	n

S

B	C	w
0	1	1
0	2	2
0	3	3
0	…	…
0	n	n

```
3  select   A, R.B, S.C,
            R.w + S.w as weight
1  from     R, S
2  where    R.B=S.B
4  order by weight ASC
5  limit    1
```

Result

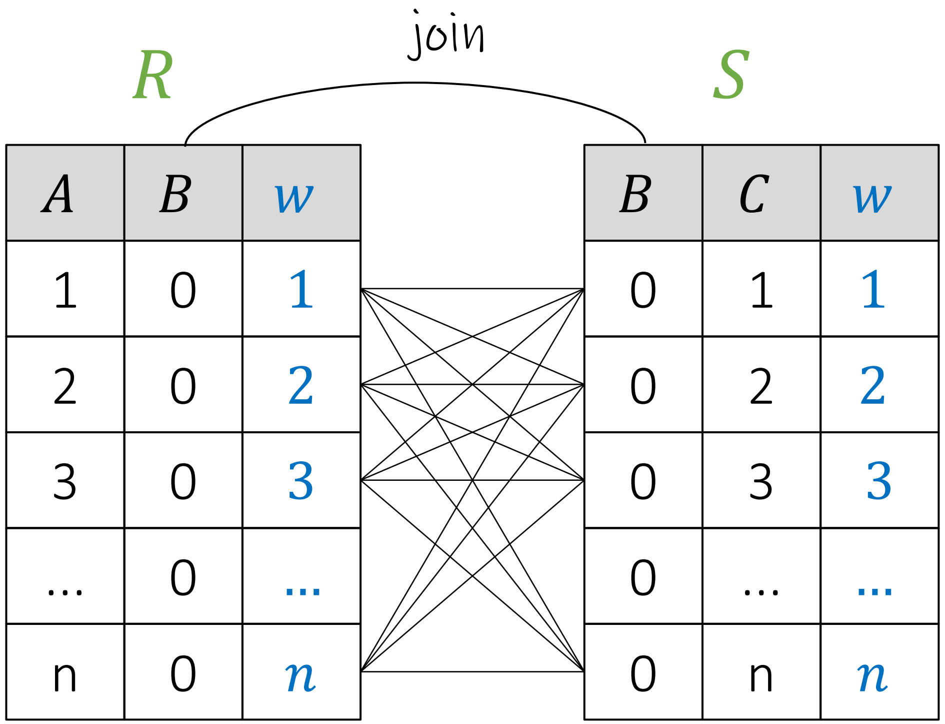
A	B	C	weight
1	0	1	2
1	0	2	3
2	0	1	3
3	0	1	4
2	0	2	4
1	0	3	4
4	0	1	5
…	…	…	…
n	0	n	n*n

Can you see any possible problem of this query as n gets bigger?

?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Top-*k* is evaluated inefficiently by modern DBMS's



R

A	*B*	*w*
1	0	1
2	0	2
3	0	3
…	0	…
n	0	*n*

S

B	*C*	*w*
0	1	1
0	2	2
0	3	3
0	…	…
0	n	*n*

```
select    A, R.B, S.C,
          R.w + S.w as weight
from      R, S
where     R.B=S.B
order by weight ASC
limit     1
```

Result

A	B	C	weight
1	0	1	2
1	0	2	3
2	0	1	3
3	0	1	4
2	0	2	4
1	0	3	4
4	0	1	5
…	…	…	…
n	0	n	n*n

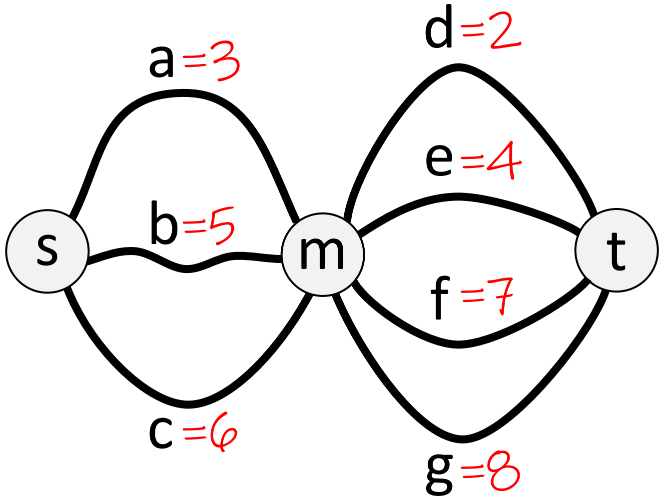
n^2 total results. But we are only interested in the top-1.

Problem: Database first calculates all n^2 results before sorting.

Question: is there any way to push the sorting behind the join? ?

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

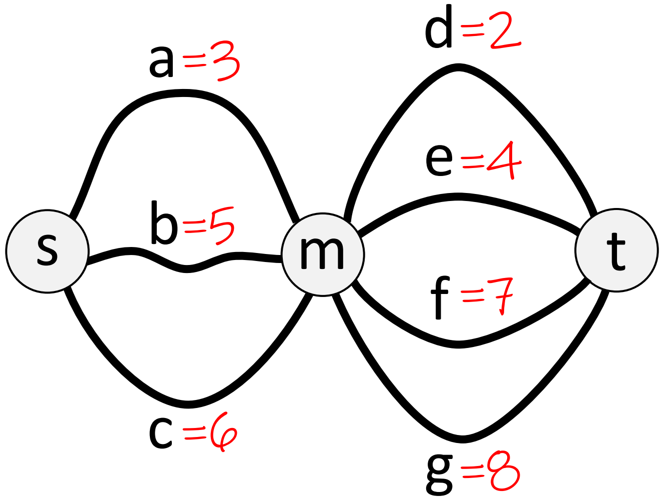
Digression: Distributivity = efficient factorization



What is the shortest path from s to t?

?

Digression: Distributivity = efficient factorization



min [a + d, a + e, a + f, a + g, ..., c + g]

min[3+2, 3+4, 3+7, 3+8, ..., 6+8]

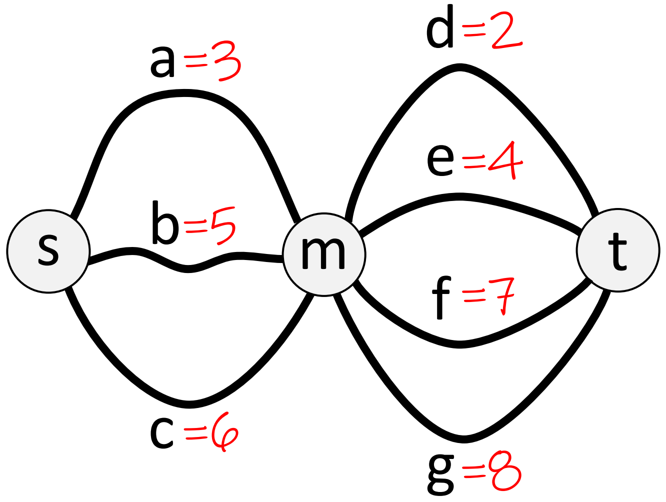
?

What is the shortest path from s to t?

Answer: 5 = 3 + 2

Digression: Distributivity = efficient factorization

Principle of optimality from Dynamic Programming: *irrespective of the initial state and decision, an optimal solution continues optimally from the resulting state*



min [a + d, a + e, a + f, a + g, ..., c + g]

min[3+2, 3+4, 3+7, 3+8, ..., 6+8]

= min [a, b, c] + min [d, e, f, g]

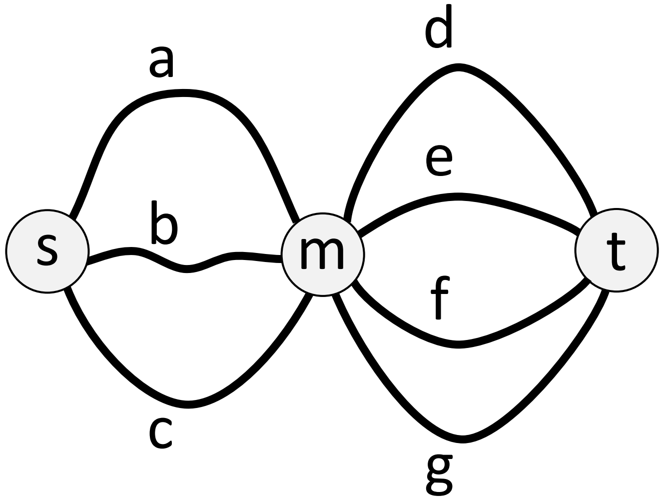
min[3,5,6] + min[2,4,7,8]

What is the shortest path from s to t?

Answer: 5 = 3 + 2

$\min[x,y]+z = \min[(x+z), (y+z)]$
(+ distributes over min)

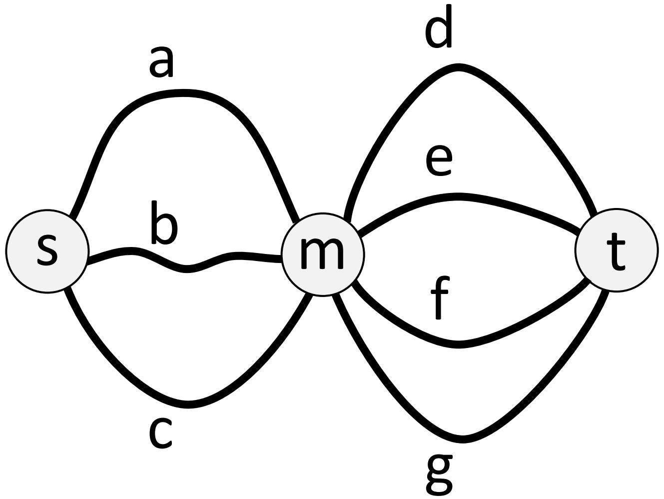
Digression: Distributivity = efficient factorization



How many paths are there from s to t?

?

Digression: Distributivity = efficient factorization

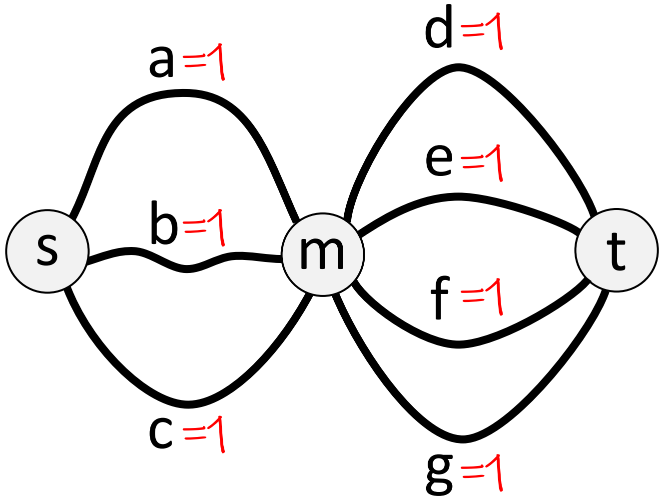


How many paths are
there from s to t?

Answer: $12 = 3 \cdot 4$

Digression: Distributivity = efficient factorization

The more general algebraic structure behind these two examples are "semirings" (much more on those later in class)



count [a·d, a·e, a · f, a · g, ..., c · g]

$$\text{count}[\underbrace{1\cdot1, 1\cdot1, 1\cdot1, 1\cdot1, \ldots, 1\cdot1}_{12}]$$

= count [a, b, c] · count [d, e, f, g]

count[1,1,1] · count[1,1,1,1]

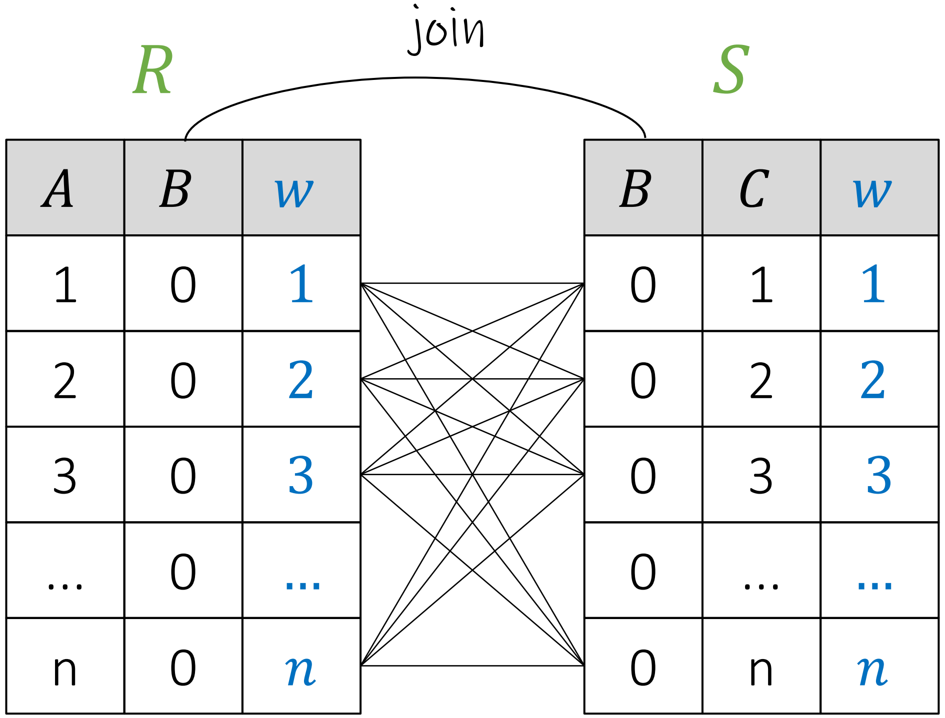
How many paths are there from s to t?

Answer: 12 = 3 · 4

$+[x,y] \cdot z = +[x\cdot z, y\cdot z]$

(· distributes over +)

Top-*k* is evaluated inefficiently by modern DBMS's



A	B	w
1	0	1
2	0	2
3	0	3
…	0	…
n	0	n

B	C	w
0	1	1
0	2	2
0	3	3
0	…	…
0	n	n

```
----------------------------
-- Query 1
----------------------------
SELECT   A, R.B, S.C,
            R.W + S.W as weight
FROM     R, S
WHERE   R.B=S.B
ORDER BY weight ASC
LIMIT   1;
```

```
----------------------------
-- Query 2
----------------------------
```

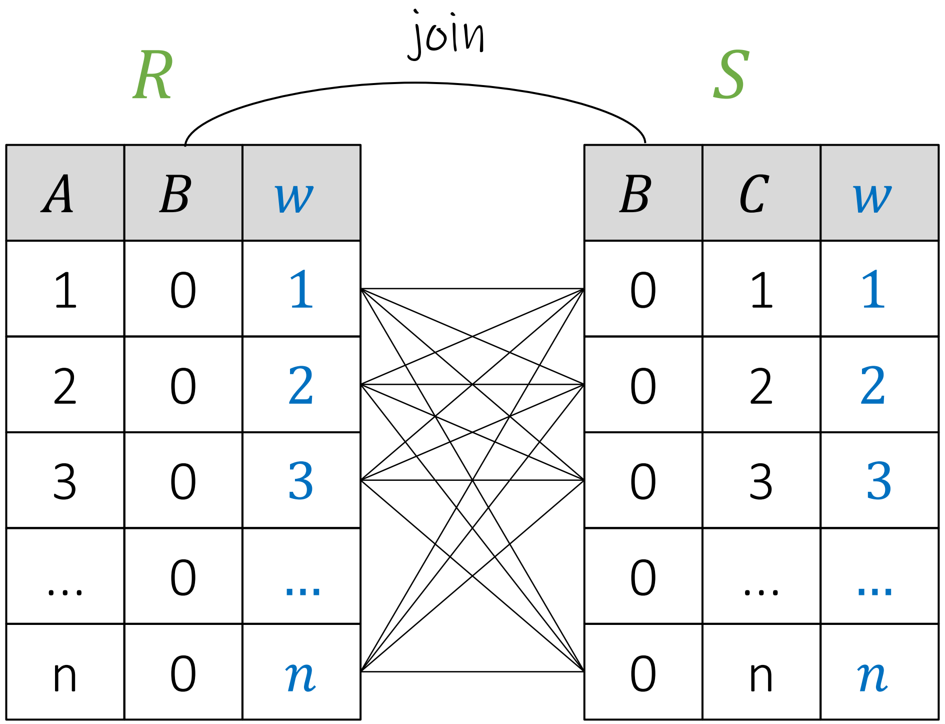
?

n=1000: t_{Q1}= 0.88 sec t_{Q2}=2 msec

n=5000: t_{Q1}=18.6 sec t_{Q2}=8 msec

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Top-*k* is evaluated inefficiently by modern DBMS's



A	B	w
1	0	1
2	0	2
3	0	3
…	0	…
n	0	*n*

B	C	w
0	1	1
0	2	2
0	3	3
0	…	…
0	n	*n*

Maximal intermediate
result size is O(n) ☺
What is this algorithm
called?

?

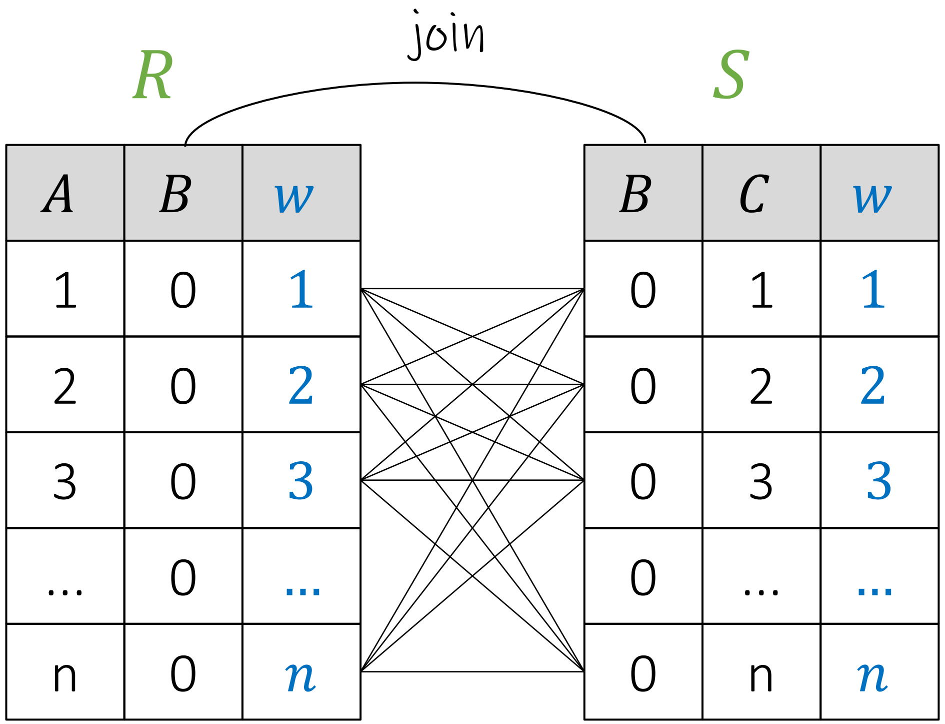
```
---------------------------
-- Query 1
---------------------------
SELECT   A, R.B, S.C,
            R.W + S.W as weight
FROM     R, S
WHERE   R.B=S.B
ORDER BY weight ASC
LIMIT   1;
```

```
---------------------------
-- Query 2
---------------------------
SELECT R.A, X.B, S.C, X.W as weight
FROM R, S,
    (SELECT T1.B, W1, W2, W1+W2 W
    FROM
        (SELECT B, MIN(W) W1
        FROM R
        GROUP BY B) T1,
        (SELECT B, MIN(W) W2
        FROM S
        GROUP BY B) T2
    WHERE T1.B = T2.B
    ORDER BY W ASC
    LIMIT 1) X
WHERE X.B = R.B
AND X.W1 = R.W
AND X.B = S.B
AND X.W2 = S.W
LIMIT   1;
```

n=1000:	t_{Q1}= 0.88 sec	t_{Q2}=2 msec
n=5000:	t_{Q1}=18.6 sec	t_{Q2}=8 msec

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Top-*k* is evaluated inefficiently by modern DBMS's



A	*B*	*w*
1	0	1
2	0	2
3	0	3
…	0	…
n	0	*n*

B	*C*	*w*
0	1	1
0	2	2
0	3	3
0	…	…
0	n	*n*

Maximal intermediate result size is $O(n)$ ☺
What is this algorithm called?

Dynamic programming

```
---------------------------
-- Query 1
---------------------------
SELECT   A, R.B, S.C,
            R.W + S.W as weight
FROM     R, S
WHERE   R.B=S.B
ORDER BY weight ASC
LIMIT   1;
```

$O(n^2)$

```
---------------------------
-- Query 2
---------------------------
SELECT R.A, X.B, S.C, X.W as weight
FROM R, S,
    (SELECT T1.B, W1, W2, W1+W2 W
    FROM
        (SELECT B, MIN(W) W1
        FROM R
        GROUP BY B) T1,
        (SELECT B, MIN(W) W2
        FROM S
        GROUP BY B) T2
    WHERE T1.B = T2.B
    ORDER BY W ASC
    LIMIT 1) X
WHERE X.B = R.B
AND X.W1 = R.W
AND X.B = S.B
AND X.W2 = S.W
LIMIT   1;
```

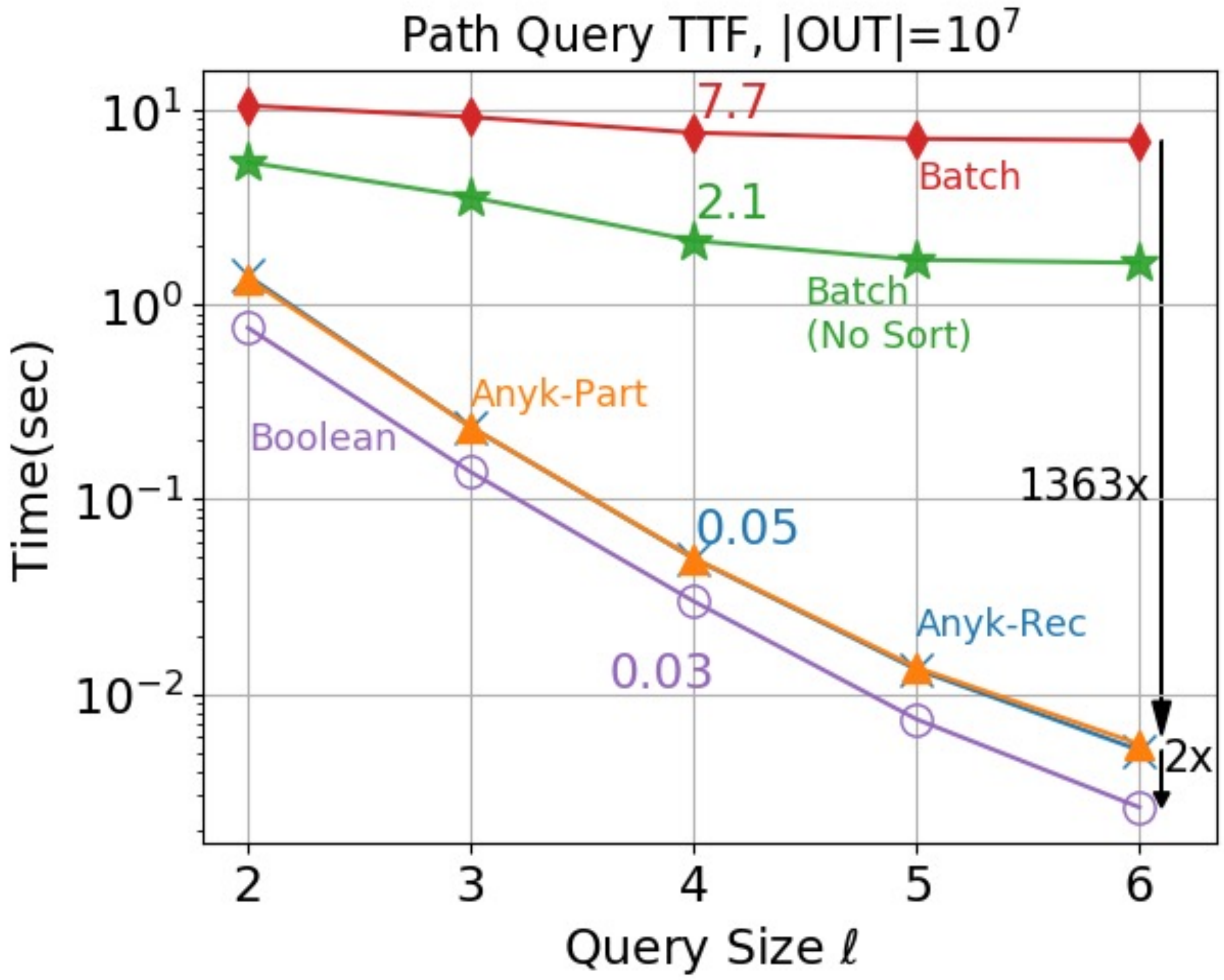
$O(n)$

n=1000: t_{Q1}= 0.88 sec t_{Q2}=2 msec

n=5000: t_{Q1}=18.6 sec t_{Q2}=8 msec

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Any-*k*: Faster and more versatile than Top-*k*

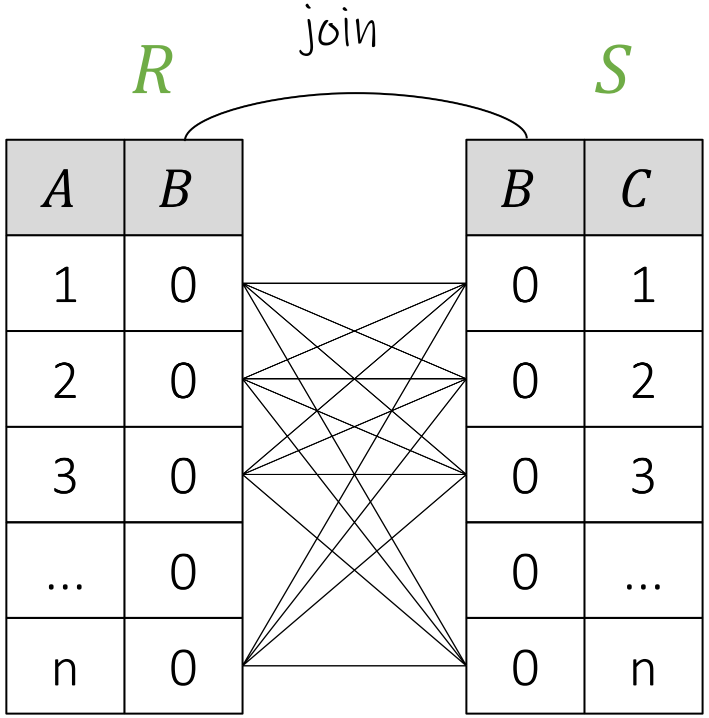


Path query with constant size output and increasing query size

https://northeastern-datalab.github.io/anyk/
https://northeastern-datalab.github.io/topk-join-tutorial/
https://northeastern-datalab.github.io/responsive-dbms-tutorial/
https://www.youtube.com/watch?v=KpUQayBuaQI&list=PL_72ERGKF6DR7kvGNwwjWlbpScKtGjt9R&index=2
Tziavelis, Ajwani, Gatterbauer, Riedewald, Yang. Optimal Algorithms for Ranked Enumeration of Answers to Full Conjunctive Queries. PVLDB 2020. https://doi.org/10.14778/3397230.3397250

Even Grouping Aggregates can be improved

608



R

A	B
1	0
2	0
3	0
…	0
n	0

S

B	C
0	1
0	2
0	3
0	…
n	0

```
---------------------------
-- Query 1
---------------------------
SELECT    count(*) CT
INTO record1
FROM        R, S
WHERE     R.B=S.B;
```

```
---------------------------
-- Query 2
---------------------------
SELECT SUM(C) as CT
INTO record2
FROM
    (SELECT T1.B, C1, C2, C1*C2 C
    FROM
        (SELECT B, COUNT(*) C1
        FROM R
        GROUP BY B) T1,
        (SELECT B, COUNT(*) C2
        FROM S
        GROUP BY B) T2
    WHERE T1.B = T2.B) X;
```

n=1000: t_{Q1}= 0.374 sec t_{Q2}=3 msec

n=5000: t_{Q1}= 10.021 sec t_{Q2}=5 msec

SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Outline: T1-U1: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Union and Theta Joins
  - Nulls & Outer joins
  - Top-k
  - [Recursion: moved to T1-U4: Datalog]