Updated 1/12/2026

Part 1: Query language foundations
Topic 1: SQL
Lecture 1

Wolfgang Gatterbauer

CS7575: A Seminar On Relational Language Design (sp26)

https://northeastern-datalab.github.io/cs7575/sp26/

1/8/2026

PART 1: Query Language Foundations

Introduces and compares the core relational languages: SQL, Domain and Tuple Relational Calculus, Relational Algebra, Datalog (including recursion with stratified negation). We also study Relational Diagrams, the semantic concept of Relational Patterns, and the Generalized Tuple Relational Calculus together with its Abstract Language Higraphs (a concept generalizing Abstract Syntax Trees). This part is led by the instructor and establishes a conceptual framework for studying new languages.

- **Lecture 1 (Thu, Jan 8)**
  Course introduction | T1-U1 SQL | PostgreSQL setup | SQL Activities
- **Lecture 2 (Mon, Jan 12)**
  P1-T1 SQL
- **Lecture 3 (Thu, Jan 15)**
  P1-T1 SQL
- **No class (Mon, Jan 19): Martin Luther King Day**
- **Lecture 4 (Thu, Jan 22): Remote class via Zoom**
  P1-T2 Logic & Relational Calculus
- **Lecture 5 (Mon, Jan 26)**
  P1-T2 Logic & Relational Calculus
- **Lecture 6 (Thu, Jan 29)**
  P1-T2 Logic & Relational Calculus
- **Lecture 7 (Mon, Feb 2)**
  P1-T3 Relational Algebra & Codd's Theorem
- **Lecture 8 (Thu, Feb 5)**
  P1-T4 Datalog & recursion
- **Lecture 9 (Mon, Feb 9) / S1 Project ideas**
  P1-T4 Datalog & recursion
- **Lecture 10 (Thu, Feb 12)**
  P1-T5 Relational Patterns & Relational Diagrams
- **No class (Mon, Feb 16): Presidents' Day**
- **Lecture 11 (Thu, Feb 19)**
  P1-T5 Abstract Relational Query Languages & Generalized Tuple Relational Calculus

Outline: T1-U1: SQL

- SQL
  - Schema, keys, referential integrity
  - Joins
  - Aggregates and grouping
  - Nested queries (Subqueries)
  - Union and Theta Joins
  - Nulls & Outer joins
  - Window Functions
  - 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

The Top Programming Languages 2024 ›

Typescript and Rust are among the rising stars

BY STEPHEN CASS | 22 AUG 2024 | 3 MIN READ |

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-2024 , Text on the right: https://spectrum.ieee.org/top-programming-languages-2022

The Top Programming Languages 2025 › Does AI mean the end for the Top Programming Languages?

BY STEPHEN CASS | 23 SEP 2025 | 6 MIN READ

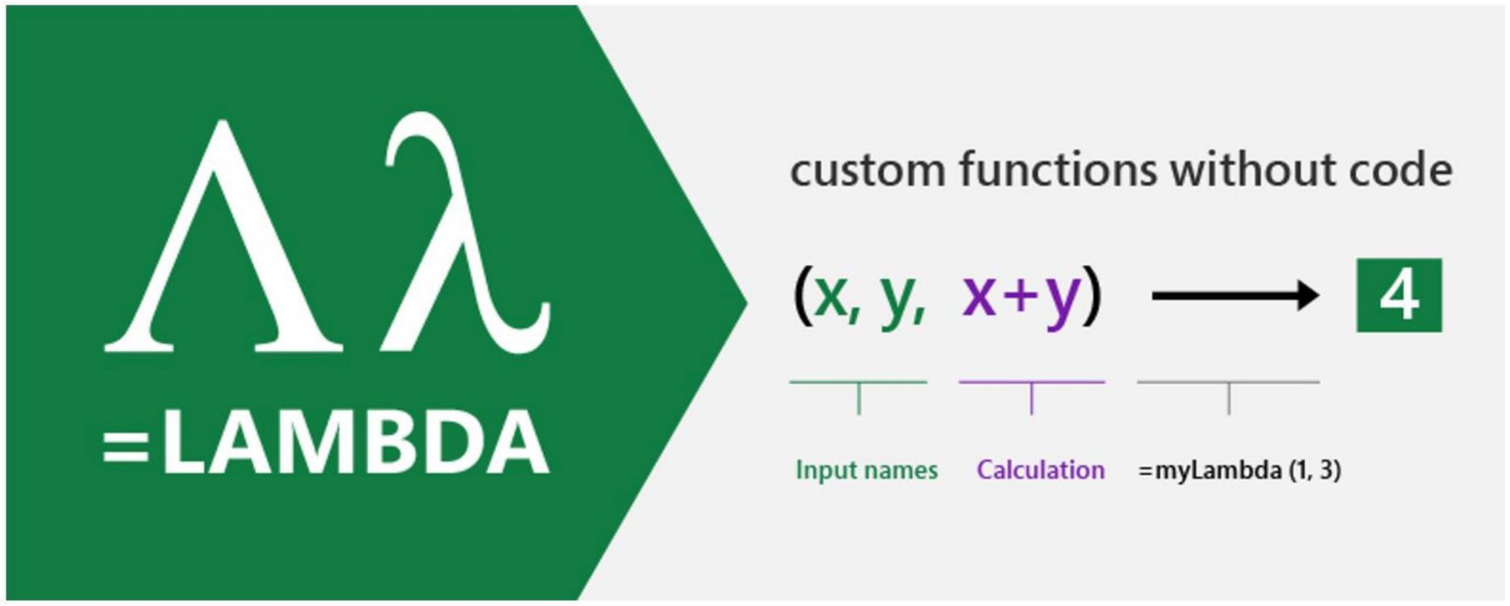
In the "*Spectrum*" default ranking, which is weighted with the interests of IEEE members in mind, we see that once again Python has the top spot, with the biggest change in the top five being JavaScript's drop from third place last year to sixth place this year. As JavaScript is often used to create web pages, and vibe coding is often used to create websites, this drop in the apparent popularity may be due to the effects of AI that we'll dig into in a moment. But first to finish up with this year's scores, in the "Jobs" ranking, which looks exclusively at what skills employers are looking for, we see that Python has also taken first place, up from second place last year, though SQL expertise remains an incredibly valuable skill to have on your resume.

Source: https://spectrum.ieee.org/top-programming-languages-2025

Fun question: What is the most popular PL?

?

Fun question: What is the most popular PL?



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

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.

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: 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'
```

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'
```

Commercially not used anymore since ~1980

QUEL

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

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

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'
```

Commercially not used anymore since ~1980

QUEL

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

Google's Pipe syntax

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

QUEL: http://en.wikipedia.org/wiki/QUEL_query_languages , [Shute+'24] SQL Has Problems. We Can Fix Them: Pipe Syntax In SQL, VLDB 2024, https://doi.org/10.14778/3685800.3685826

COMMUNICATIONS OF THE ACM | AUGUST 2024 | VOL. 67 | NO. 8

A discussion of the evolution of the database industry over the past half century, and why the relational database concepts introduced by E.F. Codd have proven to be so resilient over several decades.

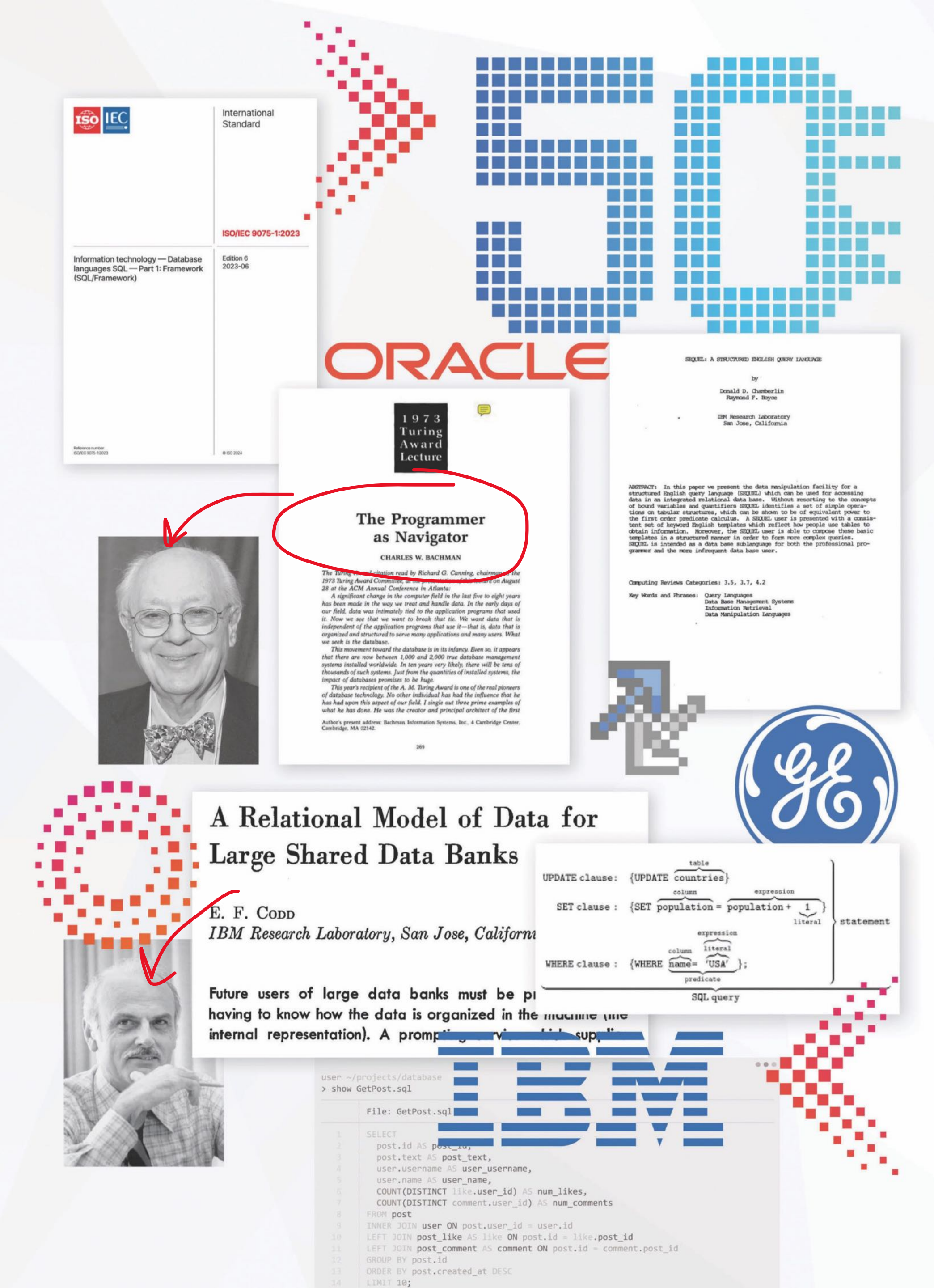
BY DONALD CHAMBERLIN

50 Years of Queries

E.F. CODD'S "A Relational Model of Data for Large Shared Data Banks"[10] is one of the most influential papers in all of computer science. In it, Codd defined concepts that are still in widespread use today, more than five decades later, including defining the theoretical foundation of the relational database industry.

When Codd's paper appeared in *Communications of the ACM* in June 1970, I was a student member of ACM, but I didn't receive the issue right away. I was driving cross-country from Stanford University to take a summer job at IBM's T.J. Watson Research Center in Yorktown Heights, New York. Before long, my summer job turned into a permanent IBM job, and I joined a group that was looking into the future of data management. My first task was to get up to speed on the current state of the art.

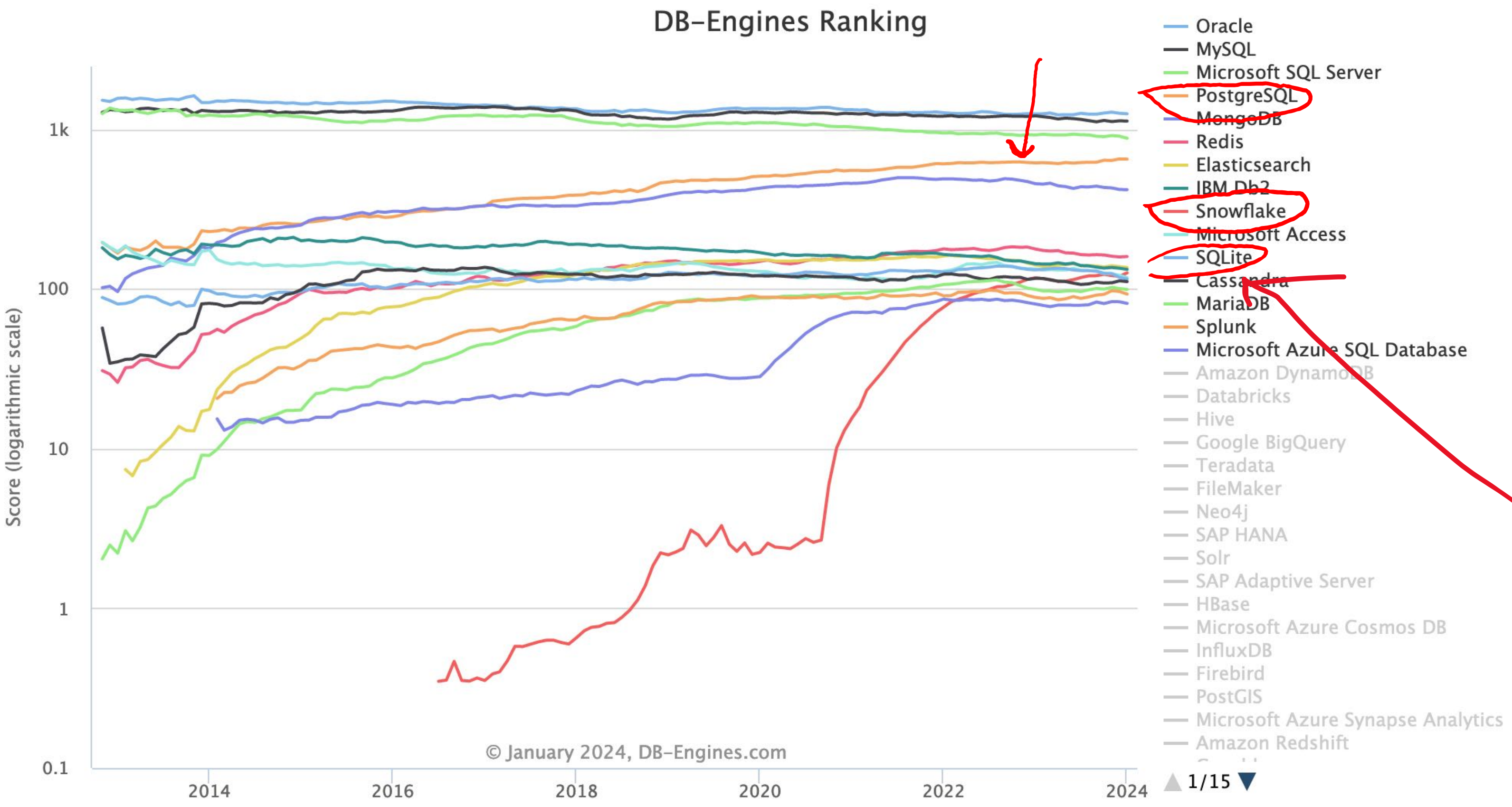
As part of my work in learning the state of the art in database management, I read Codd's 1970 paper. On first reading, I was not too impressed. The paper contained a lot of mathematical jargon. It introduced the concepts of data independence and normalization, defined a relation as a subset of the Cartesian product of a set of domains, proposed that the first-order predicate calculus could serve as a standard for measuring the expressive power of query languages, and introduced a set of operators that became known as the "relational algebra." My impression was that the paper was of some theoretical interest but was not grounded in practical engineering.



Source: Donald Chamberlin, 50 years of queries, CACM 2024. https://doi.org/10.1145/3649887
Wolfgang Gatterbauer. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/s

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.

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

Support The Guardian
Available for everyone, funded by readers
Contribute → Subscribe →
Search jobs | Sign in
News | Opinion | Sport | Culture | Lifestyle | More

Digital blog
Information

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
438

▲ A ne ephant picking up some greenery. Photograph: Michael Sohn/AP

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

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

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

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

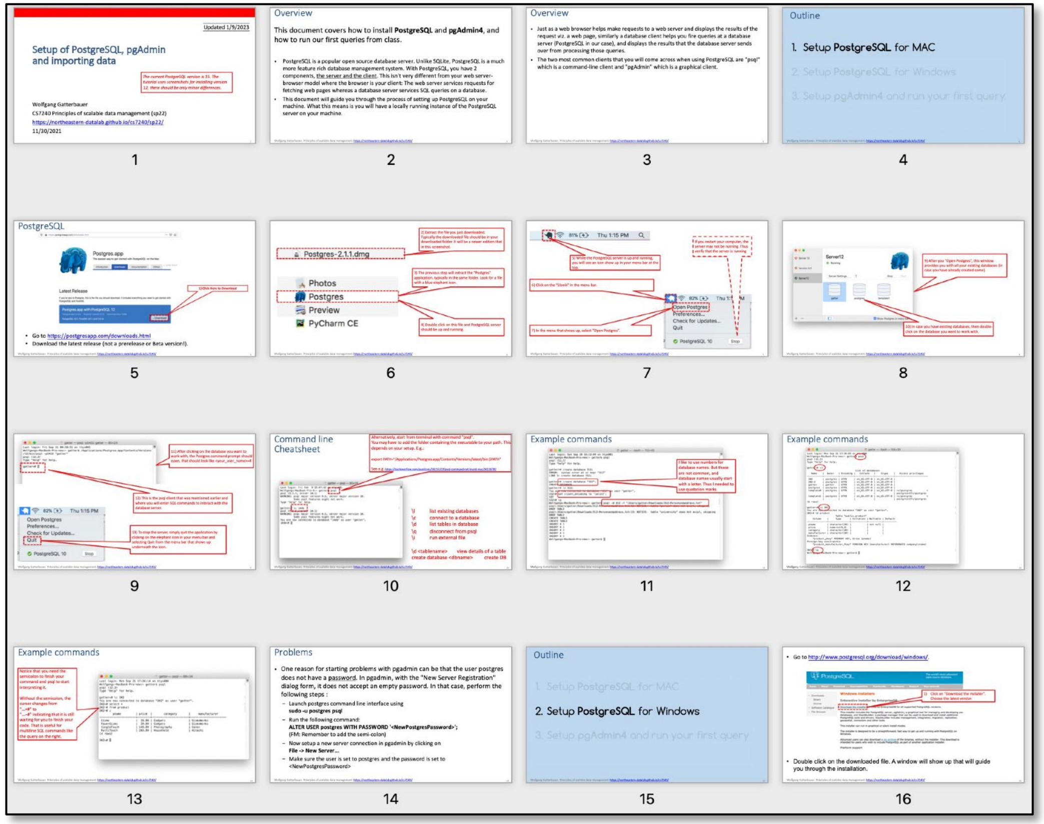
Selection
& Projection

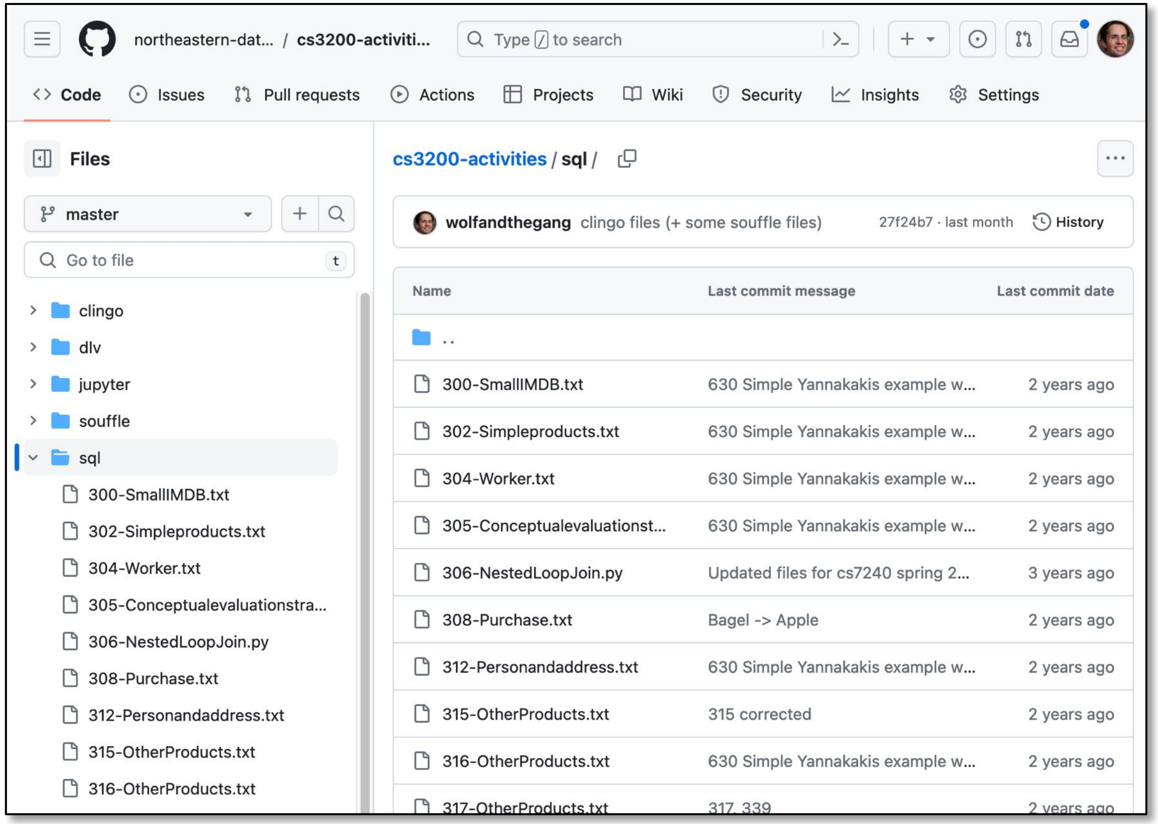
How to install PostgreSQL?

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





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
```

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

Wolfgang Gatterbauer. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

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**

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

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

Wolfgang Gatterbauer. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

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
```

⇨ ?

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. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

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,
       manufacturer
FROM   Product
```

⇨

Category	Manufacturer
Gadgets	GizmoWorks
Gadgets	GizmoWorks
Photography	Canon
Household	Hitachi

```
SELECT DISTINCT category,
       manufacturer
FROM   Product
```

⇨

Category	Manufacturer
Gadgets	GizmoWorks
Photography	Canon
Household	Hitachi

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
  - Window Functions
  - 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

Wolfgang Gatterbauer. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

Keys and Foreign Keys

Foreign key

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

Keys

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

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

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

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

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*.

Keys

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

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:

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

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

Foreign key

Foreign Key Dependencies

In the particular case when Z is a key for relations of schema S (S: $Z \rightarrow V$), 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. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

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]:
$R[X] \subseteq S[Z]$

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:

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

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. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

Integrity constraints

- **Primary Keys** (PKs) are special cases of **function dependencies** (FDs)
- **Foreign keys** (FKs) are special cases of **inclusion dependencies** (IDs = referential integrity constraints)
- In SQL, **UNIQUE** specifies **candidate keys** (CKs = sets of attributes that can uniquely identify a tuple); A PK is a chosen CK
- SQL FKs may reference CKs (not only PKs)
- SQL has no general declarative mechanism to enforce arbitrary FDs or IDs; instead, there are two more powerful formalisms:
  - **General triggers**: widely supported (though syntax/behavior varies between databases); procedural and local (database just needs to understand "run this code now")
  - **Assertion constraints**: in the SQL standard, but not implemented in practice; declarative and global (difficult for database to enforce efficiently)

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

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

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

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.

```
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

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

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.

```
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

Wolfgang Gatterbauer. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

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');
```

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');
```

tuple violates foreign key constraint

SuperTouch	$249.99	Computer	NewCom

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

Wolfgang Gatterbauer. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

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');
```

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');
```

tuple violates foreign key constraint

SuperTouch	$249.99	Computer	NewCom

However, what is allowed by default is to add a null value:

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

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');
```

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');
```

tuple violates foreign key constraint

SuperTouch	$249.99	Computer	NewCom

However, what is allowed by default is to add a null value:

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

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

?

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');

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');

tuple violates foreign key constraint

SuperTouch	$249.99	Computer	NewCom

However, what is allowed by default is to add a null value:

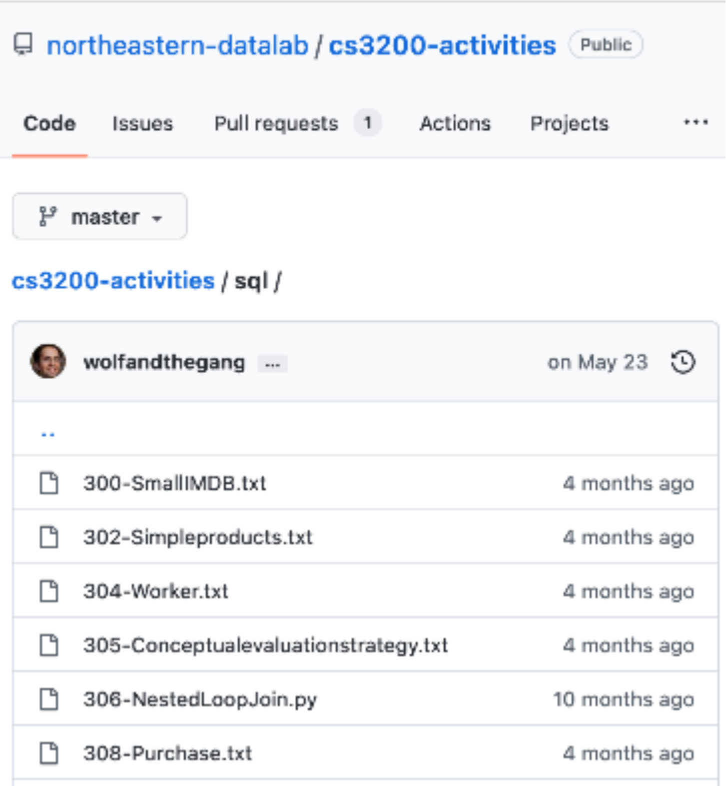
Insert into Product values ('SuperTouch', 249.99, 'Computer', null);

Delete from Company where CName = 'Canon';

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

Schema specification in SQL

302




```
---------------------------
-- 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) );
```

you can change the default behavior by appending instructions, e.g.
"foreign key (manufacturer) references company (cname) **on delete set null**;"

```
---------------------------
-- 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');
```

Outline: T1-U1: SQL

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

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!

?

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

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 (also called "join predicate)

"selection predicates"

PName	Price
SingleTouch	$149.99

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

Wolfgang Gatterbauer. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

Updated 1/12/2026

Part 1: Query language foundations
Topic 1: SQL
Lecture 2

Wolfgang Gatterbauer

CS7575: A Seminar On Relational Language Design (sp26)

https://northeastern-datalab.github.io/cs7575/sp26/

1/12/2026

Pre-class conversations

- Last class summary
- Admin
  - Already installed Postgres? Already on Piazza?
  - 2 example past mini projects from other classes posted on Canvas
  - Next week THU online
- Part 2 topics

- Today:
  - SQL continued

PART 1: Query Language Foundations

- Topic 1: SQL
  - [SAMS'19] SAMS: Teach yourself SQL in 10min by Forta. 5th ed. 2019. It is available for free for Northeastern students from Safari books eBook (you may have to first login from our library website, then try again the previous link). If the book is checked out online, you can use the 4th edition (there is almost no difference between 4th and 5th ed) as Safari books eBook, or as EBSCOhost eBook.
  - [CS 3200] PostgreSQL setup, PgAdmin 4 tutorial. Files to follow along our SQL lectures: SQL Activities, CS 3200 web pages with more detailed slides on SQL.
- Topic 2: Logic & Relational Calculus
  - [Barland+'08] Barland, Kolaitis, Vardi, Felleisen, Greiner. Intro to Logic, (alternative PDF version). 4.1.2 First-Order Logic: bound variables, free variables, 4.2.1 First-Order Logic: equivalences, 4.4 Exercises for First-Order Logic.
  - [Genesereth+] Genesereth et al. Introduction to logics. Ch 6: Relational Logic.
  - [Halpern+'01] Halpern, Harper, Immerman, Kolaitis, Vardi, Vianu. On the Unusual Effectiveness of Logic in Computer Science. Bulletin of Symbolic Logic 2001.
  - [Cow'03] Ramakrishnan, Gehrke. Database Management Systems. 3rd ed 2003. Ch 4.3: Relational calculus, Ch 4.4: Safety.
  - [Elmasri, Navathe'15] Fundamentals of Database Systems. 7th ed 2015. Ch 8.6: Tuple relational calculus, Ch 8.7: Domain relational calculus, Ch 8.6.8: Safe expressions.
  - [Silberschatz+'20] Silberschatz, Korth, Sudarshan. Database system concepts. 7th ed 2020. Ch 27.1: Tuple relational calculus, Ch 27.2: Domain relational calculus, Ch 27.1.3 & 27.2.3: Safe expressions.
  - [Alice'95] Abiteboul, Hull, Vianu. Foundations of Databases. 1995. Ch 3.1: Structure of the relational model, Ch 3.2: Named vs. unnamed perspective, Ch 3.3: Conventional vs. logic programming perspective, Ch 4.2: Logic-based perspective, Ch 5.3: Relational calculus, domain independence, Ch 5.4: Syntactic Restrictions for Domain Independence.
  - [Barker-Plummer+'11] Barker-Plummer, Barwise, Etchemendy. Language, Proof and Logic. 2nd ed. 2011. Ch 11: Multiple Quantifiers.
  - [Vardi'13] A Logical Revolution (slides), 2013. [Video @ MSR (1h 15min)]
- Topic 3: Relational Algebra & Codd's theorem
  - [Cow'03] Ramakrishnan, Gehrke. Database Management Systems. 3rd ed 2003. Ch 4.2: Relational algebra.
  - [Complete'08] Garcia-Molina, Ullman, Widom. Database Systems. 2nd ed 2008. Ch 2.4: Relational algebra, Ch 5.1-5.2: Relational Algebra and bags.
  - [Elmasri, Navathe'15] Fundamentals of Database Systems. 7th ed 2015. Ch 8: Relational Algebra.
  - [Silberschatz+'20] Silberschatz, Korth, Sudarshan. Database system concepts. 7th ed 2020. Ch 2.6: Relational algebra.
  - [Alice'95] Abiteboul, Hull, Vianu. Foundations of Databases. 1995. Ch 4.4: Algebraic perspectives, Ch 5.3-5.4: Codd's theorem: Equivalence of algebra and calculus.
- Topic 4: Datalog & Recursion
  - [Complete'08] Garcia-Molina, Ullman, Widom. Database Systems. 2nd ed. 2008. Ch 5.3-5.4: Datalog and bags, Ch 10.2: Recursion in SQL.
  - [Cow'03] Ramakrishnan, Gehrke. Database Management Systems. 3rd ed 2003. Ch 24.1-24.2: Datalog.
  - [Alice'95] Abiteboul, Hull, Vianu. Foundations of Databases. 1995. Ch 12.2: Datalog and model-theoretic semantics, Ch 12.3: Fixpoint semantics, Ch 13.1: Semi-naive evaluation, Ch 15.2: Stratified semantics and precedence graphs
  - [Koutris'19] Lecture notes from CS784 Wisconsin 2019. L9: Datalog evaluation, L10: Datalog negation
  - [Gatterbauer, Suciu'10] *Stable models*: Gatterbauer, Suciu. Data conflict resolution using trust mappings. SIGMOD 2010. [PPTX slides], [PDF slides], [video (24min)]
  - [x775'19] x775 blog post: A surprisingly nice introduction to Datalog
  - [Soufflé] Soufflé is an open-source Datalog engine. Also see Soufflé tutorial and Soufflé installation. Soufflé files to follow along our lectures on Datalog with stratified negation: Souffle Activities
  - [clingo] Postassco/Clingo is an is a disjunctive logic programming system, implementing the stable model semantics under the Answer Set Programming (ASP) paradigm. Download. Running clingo in the browser, More clingo resources. Clingo files to follow along our lectures on ASP: Clingo Activities. Clingo user guide.
  - [Eiter+'09] Eiter, Ianni, Krennwallner. Answer Set Programming: A Primer. Reasoning Web International Summer School 2009.
- Topic 5: Relational Patterns, Abstract Relational Calculus
  - [Gatterbauer,Dunne'24] On the Reasonable Effectiveness of Relational Diagrams: Explaining Relational Query Patterns and the Pattern Expressiveness of Relational Languages, SIGMOD 2024. Slides, Youtube, Project web page.
  - [Gatterbauer,Dunne'25] Relational Diagrams and the Pattern Expressiveness of Relational Languages, SIGMOD record 2025.
  - [Gatterbauer,Sabale'26] Database Research Needs and Abstract Relational Query Language, CIDR 2026.
  - [Gatterbauer'26] A Principled Solution to the Disjunction Problem of Diagrammatic Query Representations, SIGMOD 2026 (to appear).

PART 2: Old Proposals, New Proposals, and Ongoing Debates

- General debates about SQL and alternatives
  - [Chamberlin'24] 50 years of queries, CACM 2024. Video of SIGMOD 2023 keynote.
  - [Stonebraker,Hellerstein'13] What goes around comes around, Intro chapter to readings in database systems, 4th ed, 2005.
  - [Stonebraker,Pavlo'24] What goes around comes around... and around..., SIGMOD record 2024.
  - [Atzeni+'13] Atzeni, Jensen, Orsi, Ram, Tanca, Torlone. The relational model is dead, SQL is dead, and I don't feel so good myself, SIGMOD Record 2013.
  - [Date'84] A Critique of the SQL Database Language, SIGMOD Record 1984.
  - [Welty, Stemple'81] Human factors comparison of a procedural and a nonprocedural query language, TODS 1981.
  - [Chamberlin'82] On "Human Factors Comparison of a Procedural and a Nonprocedural Query Language", TODS 1982 Technical Correspondence.
  - [Reisner'81] Human Factors Studies of Database Query Languages: A Survey and Assessment, ACM Computing Surveys 1981.
- QUEL
  - QUEL query language, Wikipedia
  - [Stonebraker+'76] Stonebraker, Held, Wong, Kreps. The Design and Implementation of INGRES, TODS 1976.
  - [Stonebraker,Rowe'86] The Design of POSTGRES, SIGMOD Record 1986.
- GoogleSQL, Pipe Syntax
  - [Shute+'24] Shute, Bales, Brown, Browne, Dolphin, Kudtarkar, Litvinov, Ma, Morcos, Shen, Wilhite, Wu, Yu. SQL Has Problems. We Can Fix Them: Pipe Syntax In SQL, VLDB 2024. YouTube video
  - [Shute+'26] Shute, Zheng, Kudtarkar. Semantic Data Modeling, Graph Query, and SQL, Together at Last? CIDR 2026
- SaneQL
  - [Neumann,Leis'24] A Critique of Modern SQL and a Proposal Towards a Simple and Expressive Query Language, CIDR 2024. YouTube video
- Various other Pipe syntax / dataflow suggestions
  - [FunSQL'21]: FunSQL: a Julia library for compositional construction of SQL queries.
  - [RPQL'22]: RPQL: a simple, powerful, pipelined SQL replacement. PRQL ("Pipelined Relational QL" = "Prequel"). YouTube video
  - [Malloy'22]: Malloy: A modern open source language for analyzing, transforming, and modeling data. YouTube video
  - [Olston+'08]: Olston, Reed, Srivastava, Kumar, Tomkins. Pig Latin: a not-so-foreign language for data processing, SIGMOD 2008.
- Graph Query Languages
  - [Gheerbrant+'25] Gheerbrant, Libkin, Peterfreund, Rogova. GQL and SQL/PGQ: Theoretical Models and Expressive Power, PVLDB 2025.
  - [Francis+'23] Francis, Gheerbrant, Guagliardo, Libkin, Marsault, Martens, Murlak, Peterfreund, Rogova, Vrgoc. A researcher's digest of GQL, ICDT 2023.
- Rel from RelationalAI
  - [Aref+'25] Aref, Guagliardo, Kastrinis, Libkin, Marsault, Martens, McGrath, Murlak, Nystrom, Peterfreund, Rogers, Sirangelo, Vrgoč, Zhao, Zreika. Rel: A Programming Language for Relational Data, SIGMOD 2025.
- SQL++ and the nested relational model
  - [Carey+'24] Carey, Chamberlin, Goo, Ong, Papakonstantinou, Suver, Vemulapalli, Westmann.SQL++: We Can Finally Relax!, ICDE 2024
  - [Chamberlin+'18] SQL++ For SQL Users: A TUTORIAL, Couchbase 2018.
- Dataframes
  - ...
- [Desphande'25] Beyond Relations: A Case for Elevating to the Entity-Relationship Abstraction, CIDR 2025.
- [Dittrich'25] A Functional Data Model and Query Language is All You Need, arXiv 2025.
- [Dittrich'25] How to get Rid of SQL, Relational Algebra, the Relational Model, ERM, and ORMs in a Single Paper -- A Thought Experiment, arXiv 2025.
- [Shaikhha+'22] Shaikhha, Huot, Smith, Olteanu. Functional collection programming with semi-ring dictionaries. OOPSLA, 2022.
- [Nix,Dittrich'25]. Extending SQL to Return a Subdatabase, SIGMOD 2025.
- ...

PART 3: Beyond "traditional" Query Languages

- ASP & Disjunctive Logic Programming
  - [clingo] Postassco/Clingo is an is a disjunctive logic programming system, implementing the stable model semantics under the Answer Set Programming (ASP) paradigm. Download. Running clingo in the browser, More clingo resources. Clingo files to follow along our lectures on ASP: Clingo Activities. Clingo user guide.
  - [Eiter+'09] Eiter, Ianni, Krennwallner. Answer Set Programming: A Primer. Reasoning Web International Summer School 2009.
- miniKanren
  - [miniKanren] miniKanren project website
  - [Byrd'09] Relational programming in miniKanren- Techniques, applications, and implementations, PhD thesis 2009.
  - [Byrd+'12] Byrd, Holk, Friedman. miniKanren, live and untagged: Quine generation via relational interpreters (programming pearl), Scheme 2012.
  - [Byrd+'17] Byrd, Ballantyne, Rosenblatt, Might. A Unified Approach to Solving Seven Programming Problems (Functional Pearl) (interactive paper), ICFP 2017.
  - [Friedman+'18] Friedman, Byrd, Kiselyov, Hemann. The Reasoned Schemer, MIT press 2018.
- ...

Parameterized queries

606

northeastern-datalab / **cs200-activities**
Public

Code · Issues · Pull requests

← Files · master

cs3200-activities / sql /

wolfandthegang 2 weeks ago

Name	Last commit date
..	
300-SmallIMDB.txt	4 years ago
302-Simpleproducts.txt	4 years ago
304-Worker.txt	4 years ago
305-Conceptualevaluationstra...	4 years ago
306-NestedLoopJoin.py	11 months ago
605-top-k.txt	4 years ago
606-top-k.txt	4 years ago
607-Recursion.txt	4 years ago

cs3200-activities / sql / 606-top-k.txt

```
---------------------------
-- Create function to populate the tables
---------------------------
CREATE OR REPLACE FUNCTION addTuples(
    Relation_name varchar,
    Att0 varchar,
    Att1 varchar,
    n int)
RETURNS void
LANGUAGE plpgsql
as
$func$
DECLARE
    i int;

BEGIN
i = 1;

WHILE i <= n LOOP
    EXECUTE 'INSERT INTO ' || Relation_name || '(' || Att1 || ',' || Att0 || ', W) VALUES ('  || i || ', 0 ,' || i || ');';
    i := i + 1;
END LOOP;

END;
$func$;
```

Example SQL code and database: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql
Wolfgang Gatterbauer. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

Joins

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

Wolfgang Gatterbauer. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

Joins

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

Also called Select-Project-Join (SPJ) 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 $\cup$ $\{(a_1,\ldots,a_k)\}$ ← YIELD
return Answer

Meaning (Semantics) of SELECT-FROM-WHERE queries

Also called Select-Project-Join (SPJ) 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

Also called Select-Project-Join (SPJ) 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 ∪ {(a_1,…,a_k)}
return Answer

Notice the conceptual evaluation strategy for SPJ-queries implies that all SPJ-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

Also called Select-Project-Join (SPJ) queries

R1

att_1	att_2	...	att_k
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...

R2

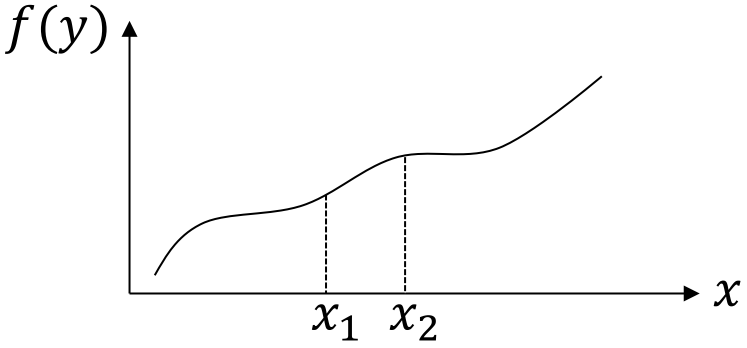
att_1	att_2	...	att_k
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...

R3

att_1	att_2	...	att_k
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...

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

if $x_1 \leq x_2$
then $f(x_1) \leq f(x_2)$



Notice the conceptual evaluation strategy for SPJ-queries implies that all SPJ-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

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

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)

What do these queries compute?

R
a
1
2

S
a
1

T
a
2

```
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. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

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. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

Using the Formal Semantics

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

What do these queries compute?

R

a
1
2

S

a
1

T

a
2

```
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. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

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. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

Using the Formal Semantics

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

What do these queries compute?

R

a
1
2

S

a
1

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
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. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

Using the Formal Semantics

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

R
a
1
2

S
a
1
3

T2
a
~~2~~

What do these queries compute?

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. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

Illustration with Python

Python file

```python
'''
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!

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
WHERE
```

Quiz: Answer 1

302

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.

```sql
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?

?

Quiz: Answer 1

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
```

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

Quiz: Answer 2

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
    or   P.price > 25)
    and  P.manufacturer = cName
```

What about this query?

?

Quiz: Answer 2

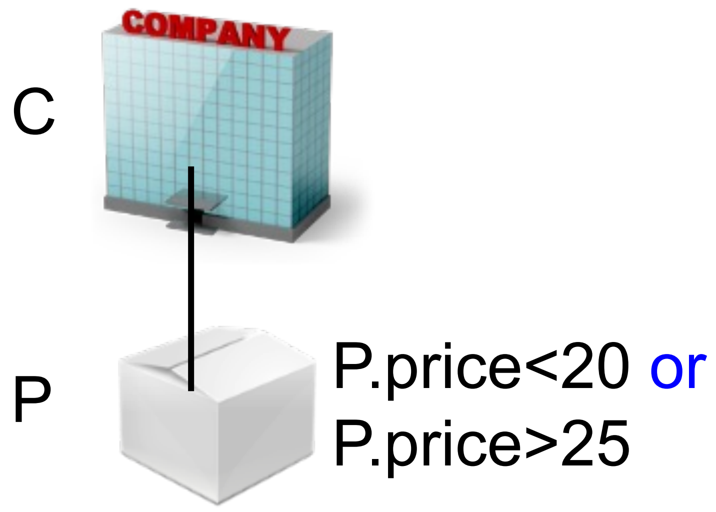
302

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.

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
```



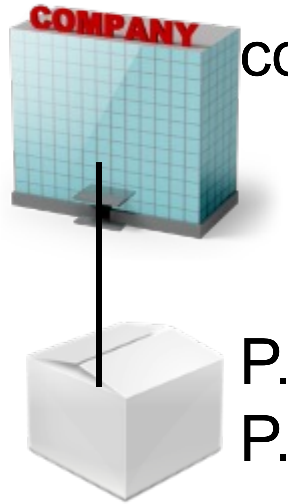
SQL example available at: https://github.com/northeastern-datalab/cs3200-activities/tree/master/sql

Quiz: Problem with Answers 1 & 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.

C



country='USA'

P

P.price < 20 and / or
P.price > 25

What do we actually want?

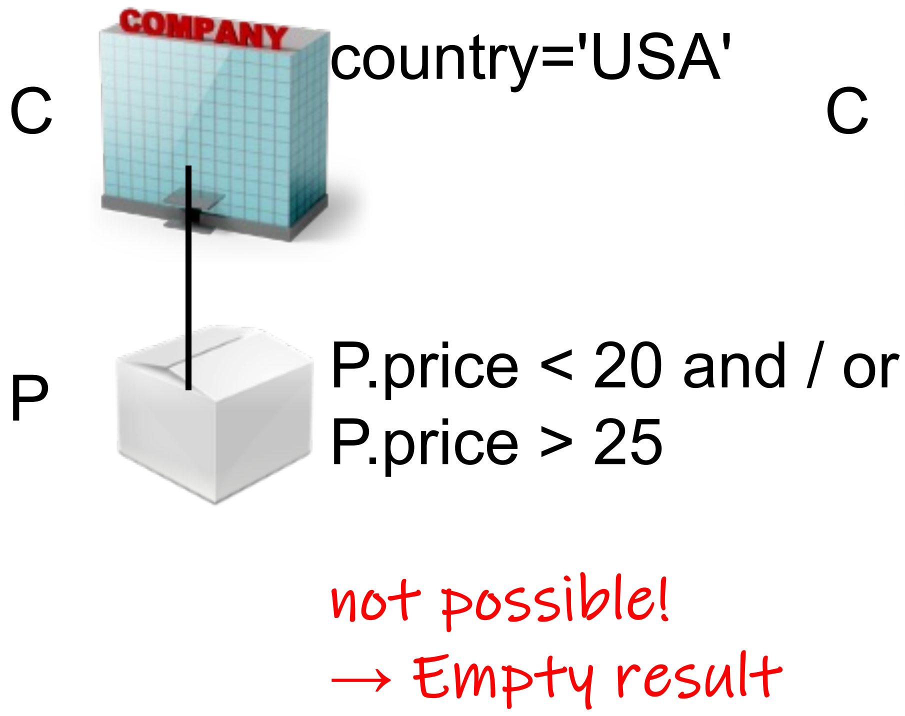
?

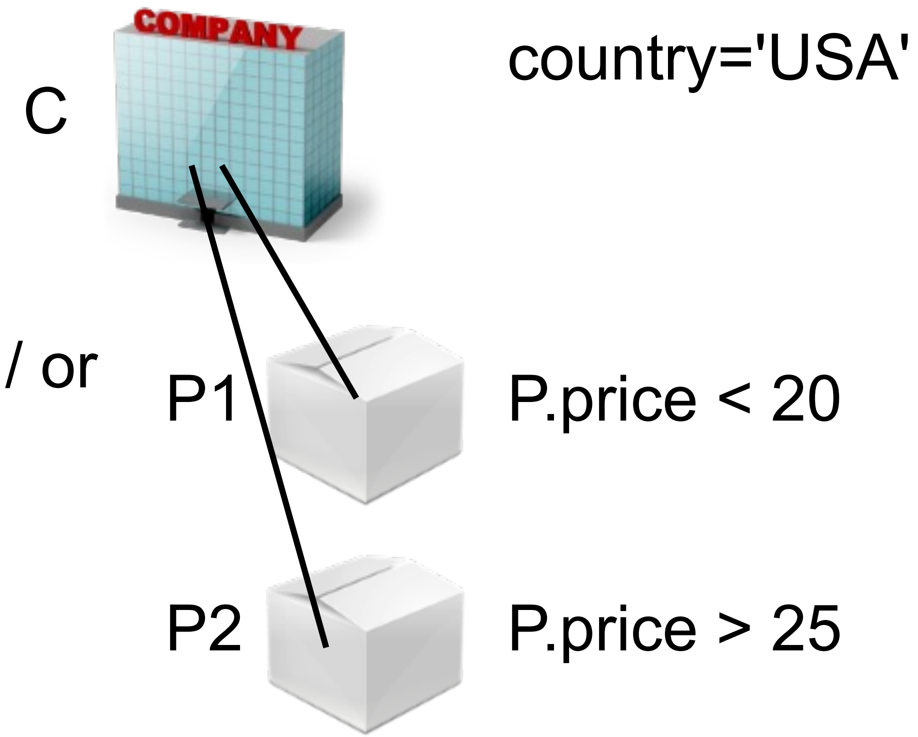
not possible!
→ Empty result

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.



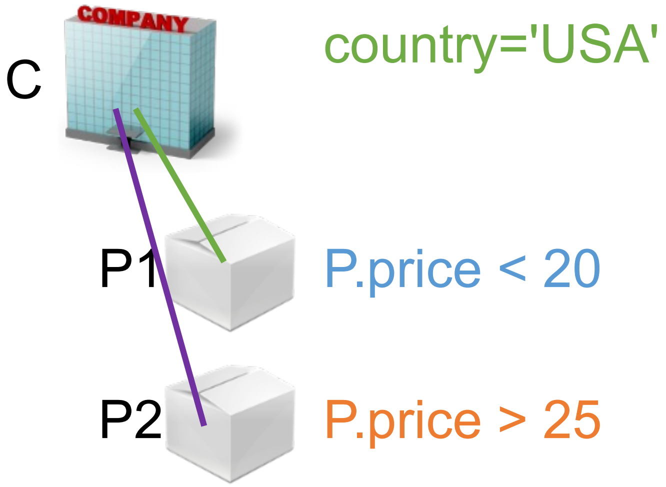


Quiz answer: we need "self-joins" (table aliases)!

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 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 answer: we need "self-joins" (table aliases)!

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

Wolfgang Gatterbauer. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

Quiz answer: we need "self-joins" (table aliases)!

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
  - Window Functions
  - Top-k
  - [Recursion: moved to T1-U4: Datalog]

Grouping and Aggregation

308

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

Wolfgang Gatterbauer. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

From → Where → Group By → Select

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
```

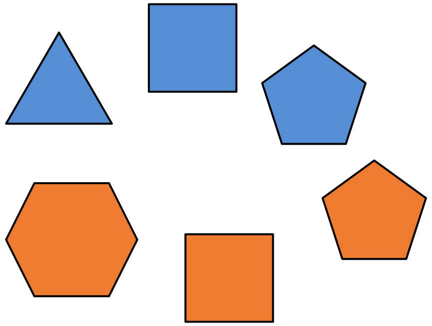
Tuples grouped together need to share the same value for attribute "product"

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

Wolfgang Gatterbauer. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

Groupings illustrated with colored shapes

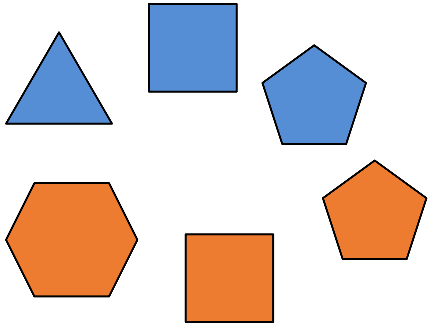
group by color



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

⇨ ?

group by numc (# of corners)

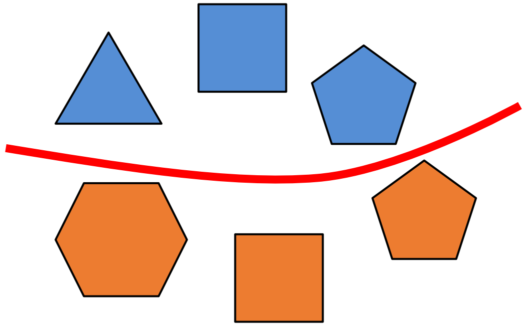


```
SELECT   numc
FROM     Shapes
GROUP BY numc
```

⇨ ?

Groupings illustrated with colored shapes

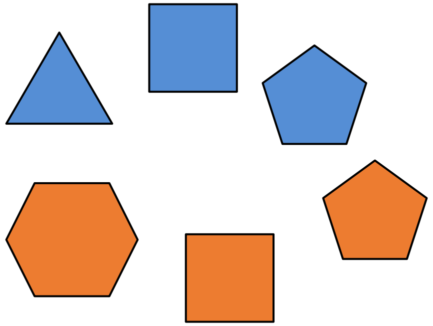
group by color



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

⇨ ?

group by numc (# of corners)



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

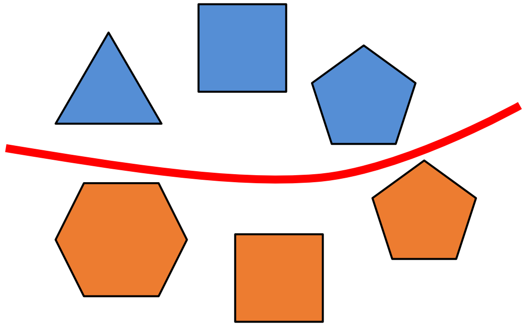
⇨ ?

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

Groupings illustrated with colored shapes

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

group by color

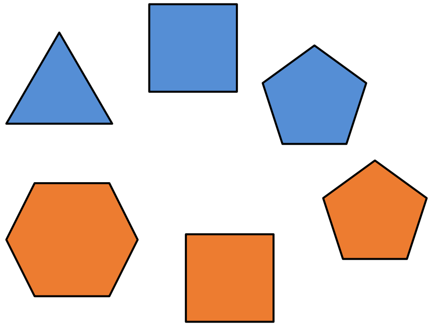


```
SELECT   color,
         avg(numc) anc
FROM     Shapes
GROUP BY color
```

⇨

color	anc
blue	4
orange	5

group by numc (# of corners)



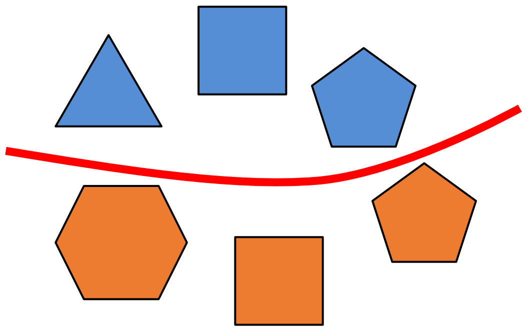
```
SELECT   numc
FROM     Shapes
GROUP BY numc
```

⇨ ?

Groupings illustrated with colored shapes

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

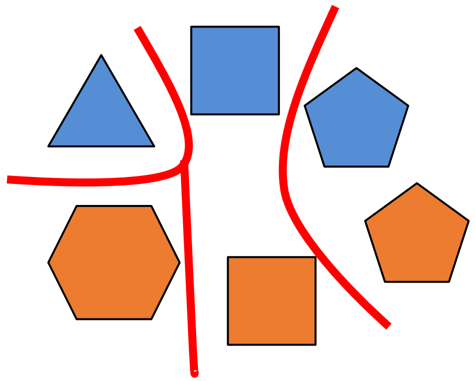
group by color



```
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
```

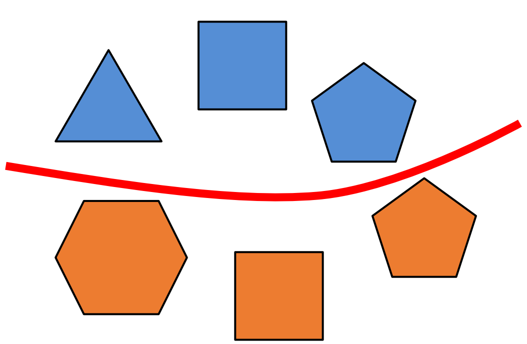
numc
3
4
5
6

Without group by ?

Groupings illustrated with colored shapes

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

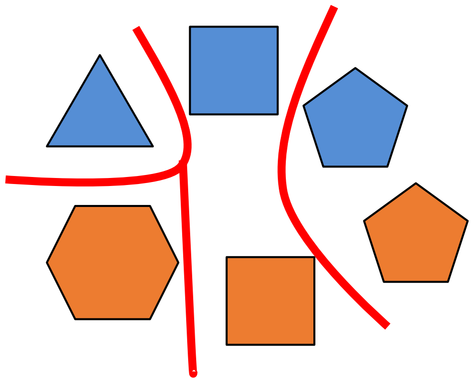
group by color



```
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:

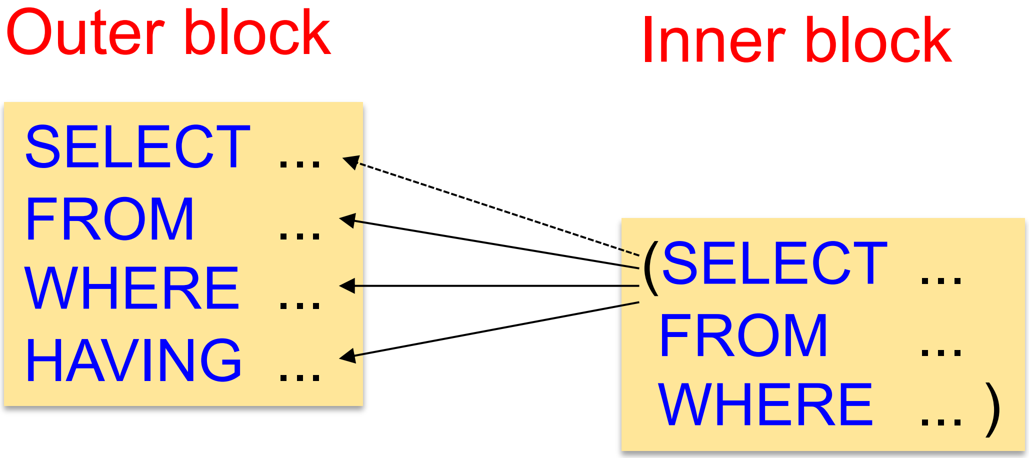
```
SELECT DISTINCT numc
FROM     Shapes
```

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
  - Window Functions
  - Top-k
  - [Recursion: moved to T1-U4: Datalog]

Subqueries = Nested queries



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

Wolfgang Gatterbauer. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

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

Wolfgang Gatterbauer. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

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

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.

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

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

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

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

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. Required for recursive queries (we discuss later with Datalog)

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

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

Wolfgang Gatterbauer. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

Subqueries in WHERE clause

What do these queries return?

R

a
1
2

W

a	b
2	0
3	0
4	0

```
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. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

Subqueries in WHERE clause

What do these queries return?

R

a
1
2

W

a	b
2	0
3	0
4	0

```
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)
```

⇒ ?

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

Wolfgang Gatterbauer. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/

Subqueries in WHERE clause

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 ∃)


316

slightly different product database!

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 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. A seminar on relational language design: https://northeastern-datalab.github.io/cs7575/sp26/