Updated 9/19/2022

Topic 1: SQL
L04: SQL intermediate

Wolfgang Gatterbauer

CS3200 Database design (fa22)

https://northeastern-datalab.github.io/cs3200/fa22s3/

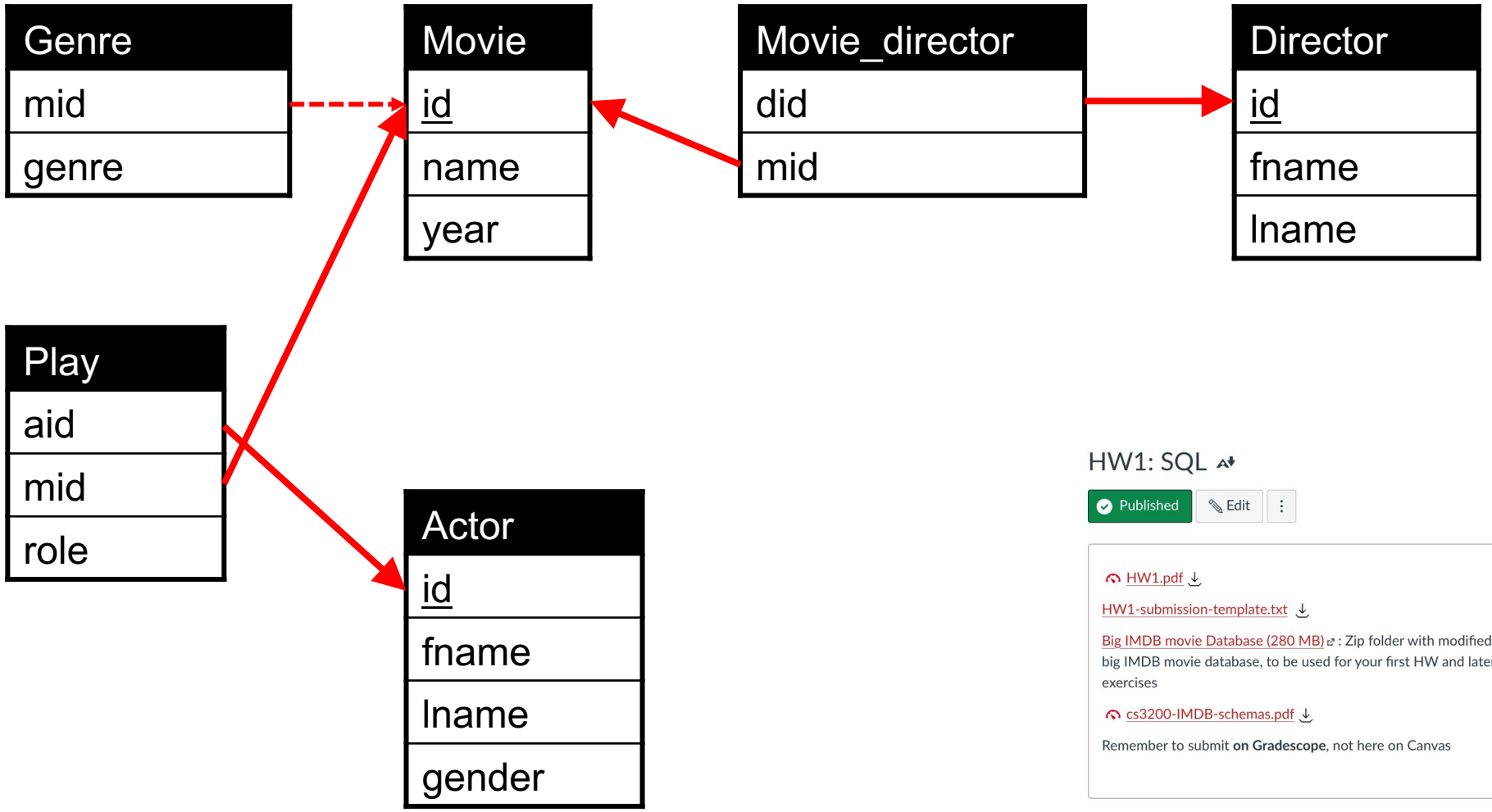
9/19/2022

Class warm-up

- Last class summary
- HW 1 groups: contacting best practice
- How to think about resources in the schedule?
- Thanks for posting and answering on Piazza!
- SQL today: aggregates, grouping, having clause
- SQL next time: nested queries
- SQL later: Nulls, outer joins, "witnesses" (traditionally students find this topic the conceptually most difficult)

Slides & further readings
[redacted]
Setup PostgreSQL, L01-Introduction, L01-SQL
Setup Gradiance, L02-SQL, SAMS Ch 1-3, SDK 3.1-3.3, 3.4.4, 3.4.5, 3.9, 4.1, 4.4.5, 4.5.1
L03-SQL, SAMS 4 & 12
SAMS Ch 5-9, SDK 3.7
SAMS Ch 10-17, SDK 3.8

Big IMDB schema (Postgres)



HW1: SQL

Published | Edit

HW1.pdf

HW1-submission-template.txt

Big IMDB movie Database (280 MB) : Zip folder with modified big IMDB movie database, to be used for your first HW and later exercises

cs3200-IMDB-schemas.pdf

Remember to submit **on Gradescope**, not here on Canvas

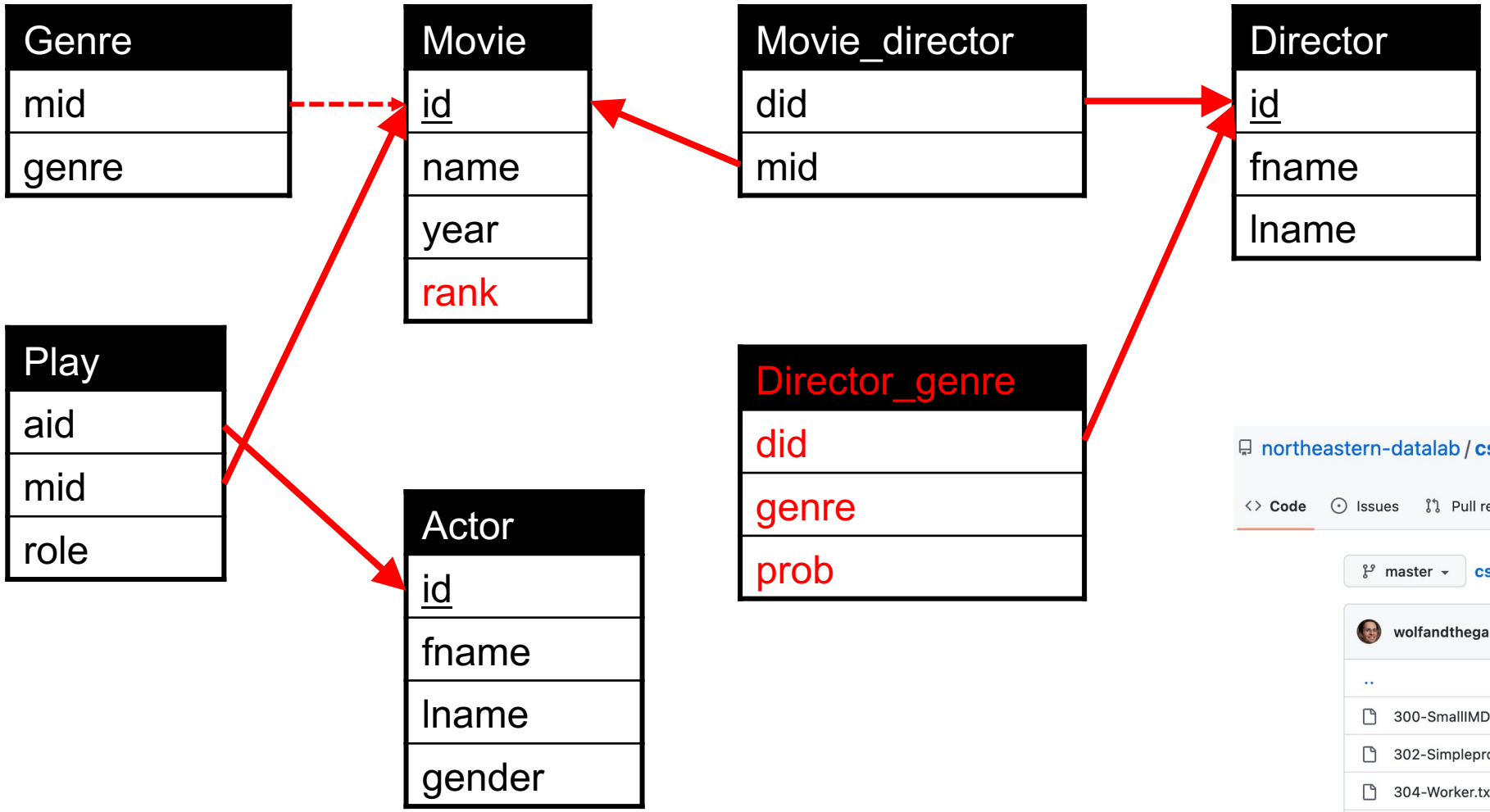
https://drive.google.com/file/d/1k4FwFQEEqygaQvNB38

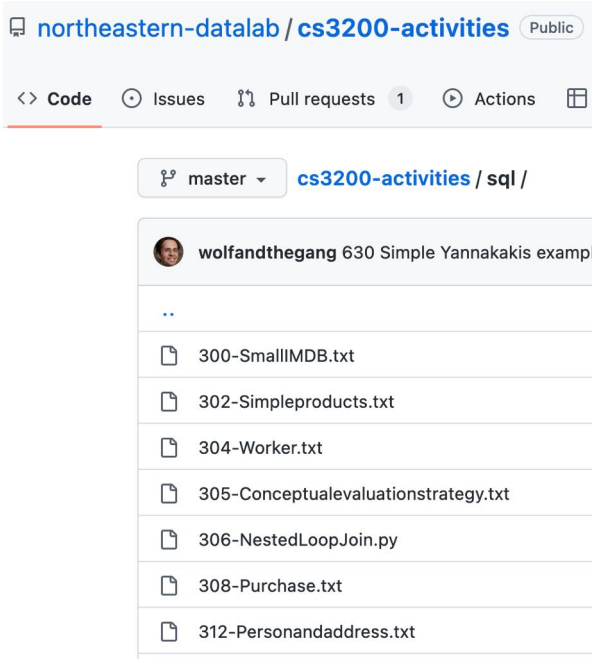
imdb-cs3200.zip

Small IMDB schema

This is a far smaller movie database in our SQL folder

→ 300





Character types

Name	Description
`character varying(n)`, `varchar(n)`	variable-length with limit
`character(n)`, `char(n)`	fixed-length, blank padded
`text`	variable unlimited length

- character varying = varchar(n)
  - variable-length character string (limited length)
- character(n) = char(n)
  - fixed length character string, blank padded (limited length)
  - in some databases faster than varchar for index look-ups (not postgres):
    - varchar may require more string manipulations; must perform some form of length checking
- text
  - variable-length character string (unlimited length)
  - Not part of SQL standard
    (e.g. Oracle does not know "text")

Tip

There is no performance difference among these three types, apart from increased storage space when using the blank-padded type, and a few extra CPU cycles to check the length when storing into a length-constrained column. While `character(n)` has performance advantages in some other database systems, there is no such advantage in PostgreSQL; in fact `character(n)` is usually the slowest of the three because of its additional storage costs. In most situations `text` or `character varying` should be used instead.

Source for tip: https://www.postgresql.org/docs/12/datatype-character.html

1. Aggregates
2. Groupings
3. Having

Simple Aggregation 1/3

Purchase (product, price, quantity)

Purchase

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

```
SELECT  sum(price * quantity)
FROM    Purchase
```

```
SELECT  sum(price * quantity)
FROM    Purchase
WHERE   product = 'Bagel'
```

What do these queries mean?

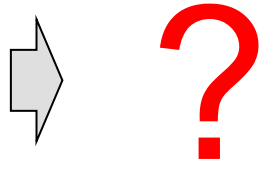
(next slides)

Simple Aggregation 2/3

Purchase

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

```
SELECT  sum(price * quantity)
FROM    Purchase
WHERE   product = 'Bagel'
```



Simple Aggregation 2/3

Purchase

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

3 * 20 = 60

2 * 20 = 40

sum: 100

```
SELECT  sum(price * quantity)
FROM    Purchase
WHERE   product = 'Bagel'
```

Database creates new attribute name

SUM
100

Simple Aggregation 3/3

Purchase

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

```
SELECT  sum(price) * sum(quantity)
FROM    Purchase
WHERE   product = 'Bagel'
```

?

Simple Aggregation 3/3

Purchase

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

	3		20	
	2		20	
sum: 5	*	sum: 40	=	200

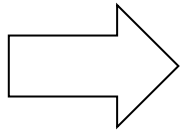
```
SELECT  sum(price) * sum(quantity)
FROM    Purchase
WHERE   product = 'Bagel'
```

?column?
200

Grouping and Aggregation

Purchase

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



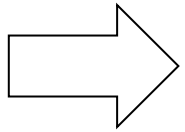
?

Q: Find total quantities for all purchases with price above $1 grouped by product.

Grouping and Aggregation

Purchase

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



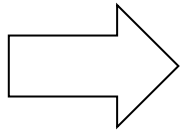
Product	TotalQuantities
Bagel	?
Banana	?

Q: Find total quantities for all purchases with price above $1 grouped by product.

Grouping and Aggregation

Purchase

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



Product	TotalQuantities
Bagel	40
Banana	20

Q: Find total quantities for all purchases with price above $1 grouped by product.

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	TotalQuantities
Bagel	40
Banana	20

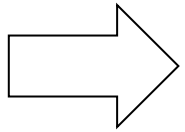
Select contains
- grouped attributes
- and aggregates

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

Let's confuse the database engine

Purchase

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



Product	Quantity
Bagel	?
Banana	?

What quantity should the database return for Banana?

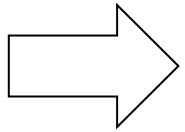
?

```
SELECT    product, quantity
FROM      Purchase
GROUP BY  product
```

Let's confuse the database engine

Purchase

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



Product	Quantity
Bagel	?
Banana	?

What quantity should the database return for Banana?

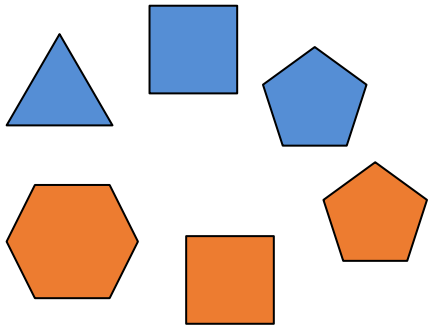
```
SELECT    product, quantity
FROM      Purchase
GROUP BY  product
```

The DB engine is confused, and rightly so: there is no single quantity for banana (it's an ill-defined query).
It should thus return an error (only SQLite misbehaves and returns something, but which makes no sense). Please think through this example carefully!

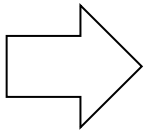
Groupings illustrated with colored shapes

Our colorful hands represent "team exercises"

group by color

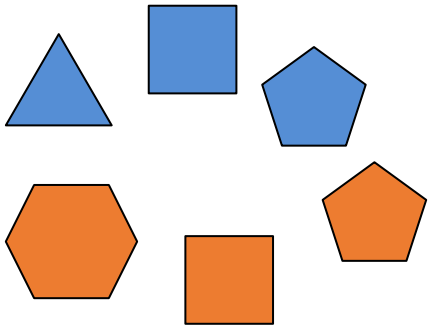


```
SELECT  color,
        avg(numc) anc
FROM    Shapes
GROUP BY color
```

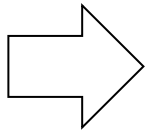


?

group by numc (# of corners)



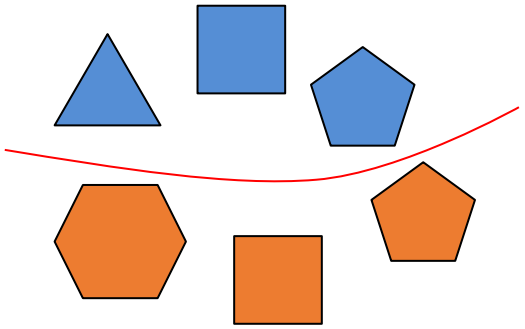
```
SELECT  numc
FROM    Shapes
GROUP BY numc
```



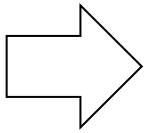
?

Groupings illustrated with colored shapes

group by color

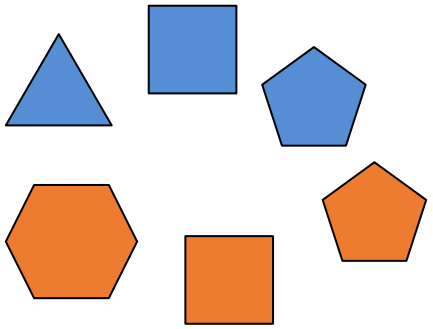


```
SELECT  color,
        avg(numc) anc
FROM    Shapes
GROUP BY color
```

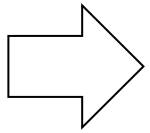


color	anc
blue	4
orange	5

group by numc (# of corners)



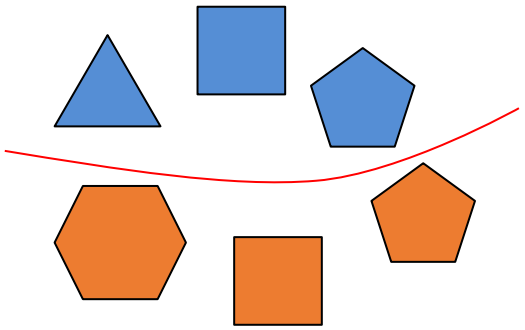
```
SELECT  numc
FROM    Shapes
GROUP BY numc
```



?

Groupings illustrated with colored shapes

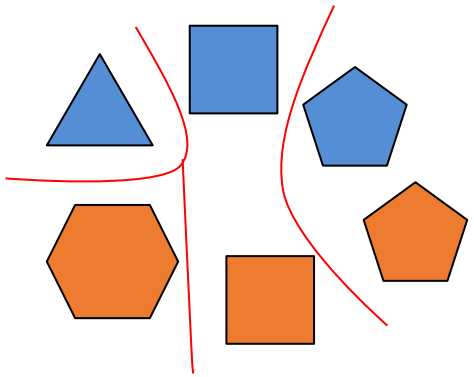
group by color



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

color	anc
blue	4
orange	5

group by numc (# of corners)

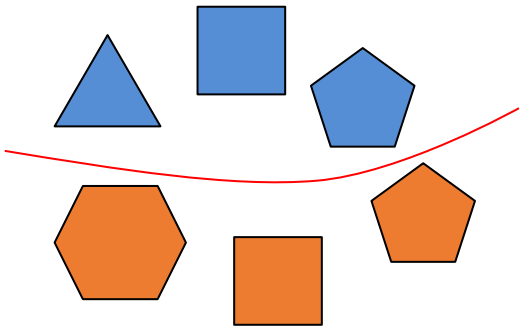


```
SELECT  numc
FROM    Shapes
GROUP BY numc
```

numc
3
4
5
6

Groupings illustrated with colored shapes

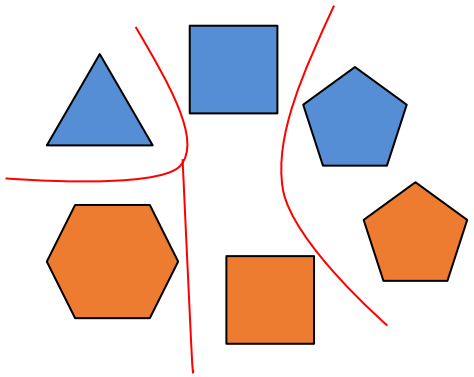
group by color



```
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

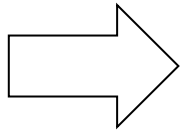
Same as:

```
SELECT  DISTINCT numc
FROM    Shapes
```

Another Example

Purchase

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

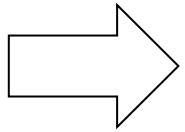


```
SELECT   product,
         sum(quantity) as SumQ,
         max(price) as MaxP
FROM     Purchase
GROUP BY product
```

Another Example

Purchase

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



Product	SumQ	MaxP
Bagel	40	3
Banana	70	4

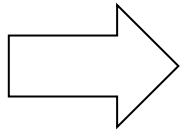
```
SELECT   product,
         sum(quantity) as SumQ,
         max(price) as MaxP
FROM     Purchase
GROUP BY product
```

Next, focus only on products with at least 50 total quantity

Group qualification

Q: Similar to before, but only products with at least 50 sales.

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



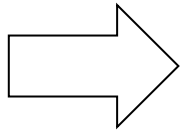
Product	SumQ	MaxP
Banana	70	4

```
SELECT   product,
         sum(quantity) as SumQ,
         max(price) as MaxP
FROM     Purchase
GROUP BY product
HAVING   sum(quantity) > 50
```

Quiz

What does this query return over the given database?

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



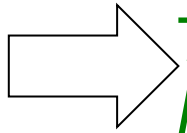
?

```
SELECT   product, sum(quantity) as SumQ
FROM     Purchase
WHERE    quantity > 15
GROUP BY product
HAVING   sum(quantity) > 40
```

Quiz

What does this query return over the given database?

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



Product	SumQ
~~Bagel~~	~~40~~
Banana	50

```
SELECT   product, sum(quantity) as SumQ
FROM     Purchase
WHERE    quantity > 15
GROUP BY product
HAVING   sum(quantity) > 40
```

General form of SQL Query

The clauses of a SQL query always appear in this order; you can't reorder them

5	SELECT	S
1	FROM	$R_1,\ldots,R_n$
2	WHERE	C1
3	GROUP BY	$a_1,\ldots,a_k$
4	HAVING	C2
6	ORDER BY	S2

S: "projection": may contain attributes $a_1,\ldots,a_k$ and/or any aggregates but no other attributes

C1: is any condition ("comparison predicate" or "join condition") on the attributes in $R_1,\ldots,R_n$

C2: is any condition on aggregates and on attributes $a_1,\ldots,a_k$ (except if PK)

Evaluation ("logical order")

1. Evaluate FROM
2. WHERE, apply condition C1
3. GROUP BY the attributes $a_1,\ldots,a_k$
4. Apply condition C2 to each group (may have aggregates)
5. Compute aggregates in S and return the result
6. Sort rows by ORDER BY clause

The logical order is useful for under-standing, but not always correct. The ANSI SQL standard does not require a specific processing order and leaves that to the implementation. Recall our intro example with SELECT DISTINCT and order by! Notice that that example can't be explained with the order shown here (you need to assume that the ORDER clause still has access to all attributes)

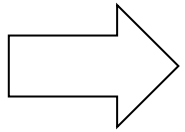
Conceptual Evaluation Strategy with GROUP BY

- The cross-product of relation-list is computed (FROM), tuples that fail qualification are discarded (WHERE), then:
- GROUP BY: the remaining tuples are partitioned into groups by the value of attributes in grouping-list.
- HAVING: the group-qualification is applied to eliminate some groups
  - Expressions in group-qualification must have a single value per group!
  - In effect, an attribute in group-qualification that is not an argument of an aggregate op must also appear in grouping-list. (exception: Some DBMSs can exploit primary key semantics!)
- One answer tuple is generated per qualifying group.

Don't use new Alias in HAVING clause

What does this query return over the given database?

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



?

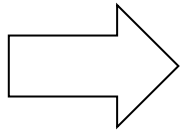
```
SELECT   product, sum(quantity) as SumQ
FROM     Purchase
WHERE    quantity > 15
GROUP BY product
HAVING   SumQ > 35
```

Source: More information: http://stackoverflow.com/questions/2068682/why-cant-i-use-alias-in-a-count-column-and-reference-it-in-a-having-clause

Don't use new Alias in HAVING clause

What does this query return over the given database?

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



Product	SumQ
Bagel	40
Banana	50

Error in PostgreSQL and (SQL server)! Reason: HAVING is evaluated before SELECT! (However, SQLite works: different implementation)

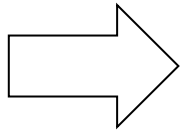
```
SELECT   product, sum(quantity) as SumQ
FROM     Purchase
WHERE    quantity > 15
GROUP BY product
HAVING   SumQ > 35
```

Source: More information: http://stackoverflow.com/questions/2068682/why-cant-i-use-alias-in-a-count-column-and-reference-it-in-a-having-clause

Don't use new Alias in HAVING clause

What does this query return over the given database?

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



Product	SumQ
Banana	50
Bagel	40

Works! Notice the new sorting

```
SELECT   product, sum(quantity) as SumQ
FROM     Purchase
WHERE    quantity > 15
GROUP BY product
HAVING   sum(quantity)  > 35
ORDER BY sumQ desc
```

Some more history

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

SQL

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

QUEL

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

Commercially not used anymore since ~1980

Source: http://en.wikipedia.org/wiki/QUEL_query_languages
Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

Arithmetic & Date functions

Arithmetic expressions

```
SELECT  3+2
```

⇨ ?

```
SELECT  (2*3 + 4*5) as name
```

⇨ ?

Arithmetic expressions

```
SELECT  3+2
```

⇨

(no column name)
5

```
SELECT  (2*3 + 4*5) as name
```

⇨ ?

Arithmetic expressions

```
SELECT  3+2
```

(no column name)
5

```
SELECT  (2*3 + 4*5) as name
```

name
26

Date functions are database-specific

Worker

Name	Birthdate
Max	1980-01-01
Fred	1979-02-01
Susan	1990-01-31
Tilda	1988-01-01

Assuming 2013. We can specify the output column names

age
33
34
23
25

```
SELECT date('now')-date(birthdate) as age
FROM Worker
```

This is here SQLite semantics. Date functions are different between different databases. You do not need to remember those. In real life, you may need to look up how your DB handles date functions, e.g., http://www.sqlite.org/lang_datefunc.html , https://www.postgresql.org/docs/current/functions-datetime.html

postgres

```
SELECT extract(year from age(birthdate)) as age
FROM Worker
```

```
SELECT date_part('year', age(birthdate)) as age
FROM Worker
```

Practicing more Joins

Exercise

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

Q: Find all US companies that manufacture at least two different products.

```
SELECT  DISTINCT cName
FROM
WHERE
```

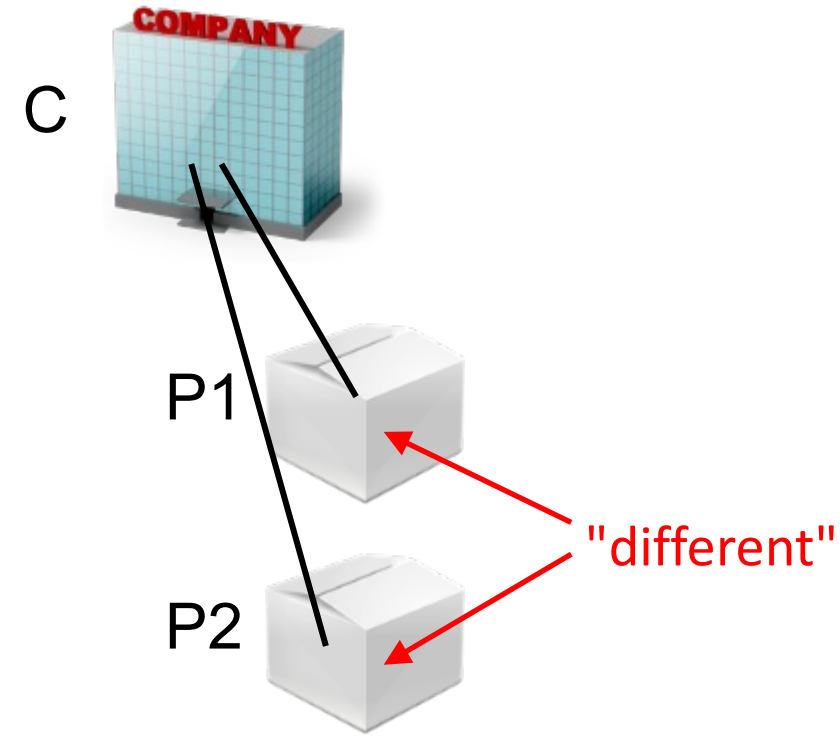
?

Exercise

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

Q: Find all US companies that manufacture at least two different products.

```
SELECT  DISTINCT cName
FROM    Product P1, Product P2, Company
WHERE   country = 'USA'
    and P1.manufacturer = cName
    and P2.manufacturer = cName
    and "the product should be different"
```

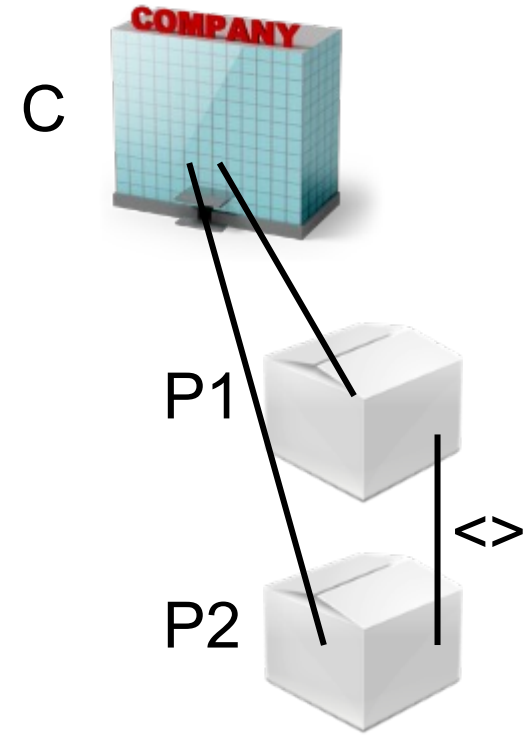


Exercise

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

Q: Find all US companies that manufacture at least two different products.

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



Exercise

P1

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
...	...	...	...

<>

P2

PName	Price	Category	Manufacturer
...	...	...	...
Powergizmo	$29.99	Gadgets	GizmoWorks

Company

CName	StockPrice	Country
GizmoWorks	25	USA
...	...	...

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

Cname
GizmoWorks

Question: what about 3, 4, 5, etc. different products? ?

Exercise

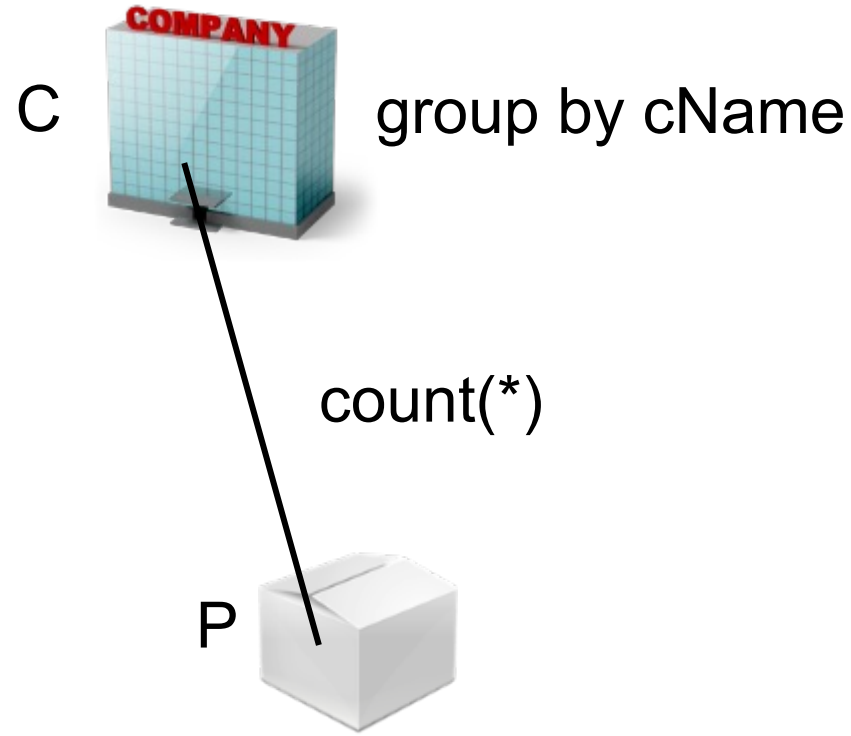
Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture at least two different products.

```
SELECT   cName
FROM     Product P, Company C
WHERE    manufacturer = cName
     and country = 'USA'
GROUP by cName
HAVING   count(*) >= 2
```

what about:

```
...
HAVING   count(DISTINCT pname) >= 2
```



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  cName
FROM    Product P, Company C
WHERE   manufacturer = cName
    and   country = 'USA'
GROUP by cName
HAVING  count(*) >= 2
```

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  cName
FROM    Product P, Company C
WHERE   manufacturer = cName
    and   country = 'USA'
GROUP by cName
HAVING  count(*) >= 2
```

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   cName
FROM     Product P, Company C
WHERE    manufacturer = cName
    and  country = 'USA'
GROUP by cName
HAVING   count(*) >= 2
```

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   cName
FROM     Product P, Company C
WHERE    manufacturer = cName
     and country = 'USA'
GROUP by cName
HAVING   count(*) >= 2
```

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

count(*) >= 2 ✓

```
SELECT   cName
FROM     Product P, Company C
WHERE    manufacturer = cName
     and country = 'USA'
GROUP by cName
HAVING   count(*) >= 2
```

Cname
GizmoWorks