Updated 10/3/2022

Topic 1: SQL
L08: SQL advanced

Wolfgang Gatterbauer

CS3200 Database design (fa22)

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

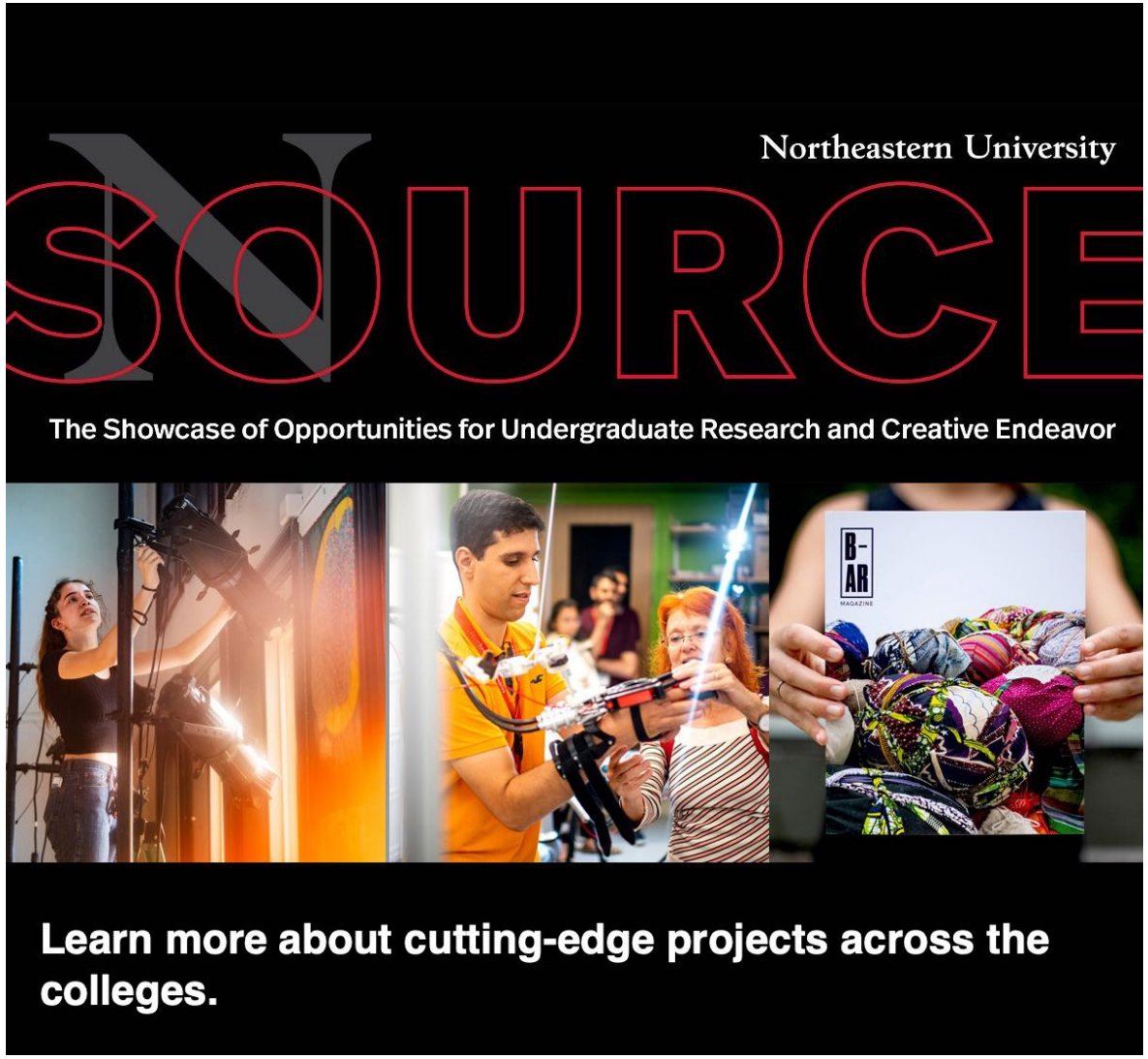
10/3/2022

Class warm-up

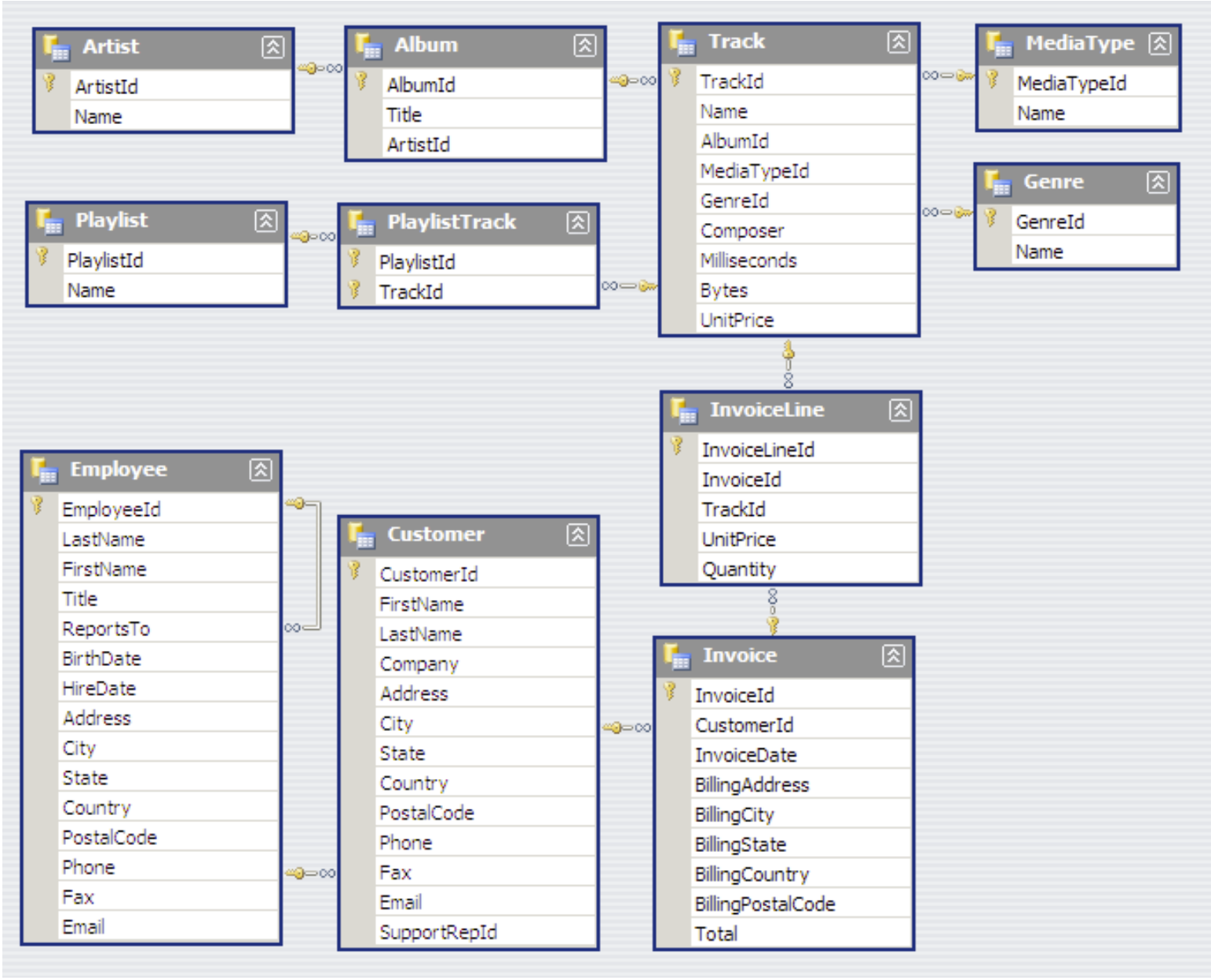
- Last class summary
- Exam WED next week in class
- Source
- Homework hands-on

- SQL today: Nulls & outer join practice, window functions

Undergraduate research @ NEU: "Source"



319 Chinook database

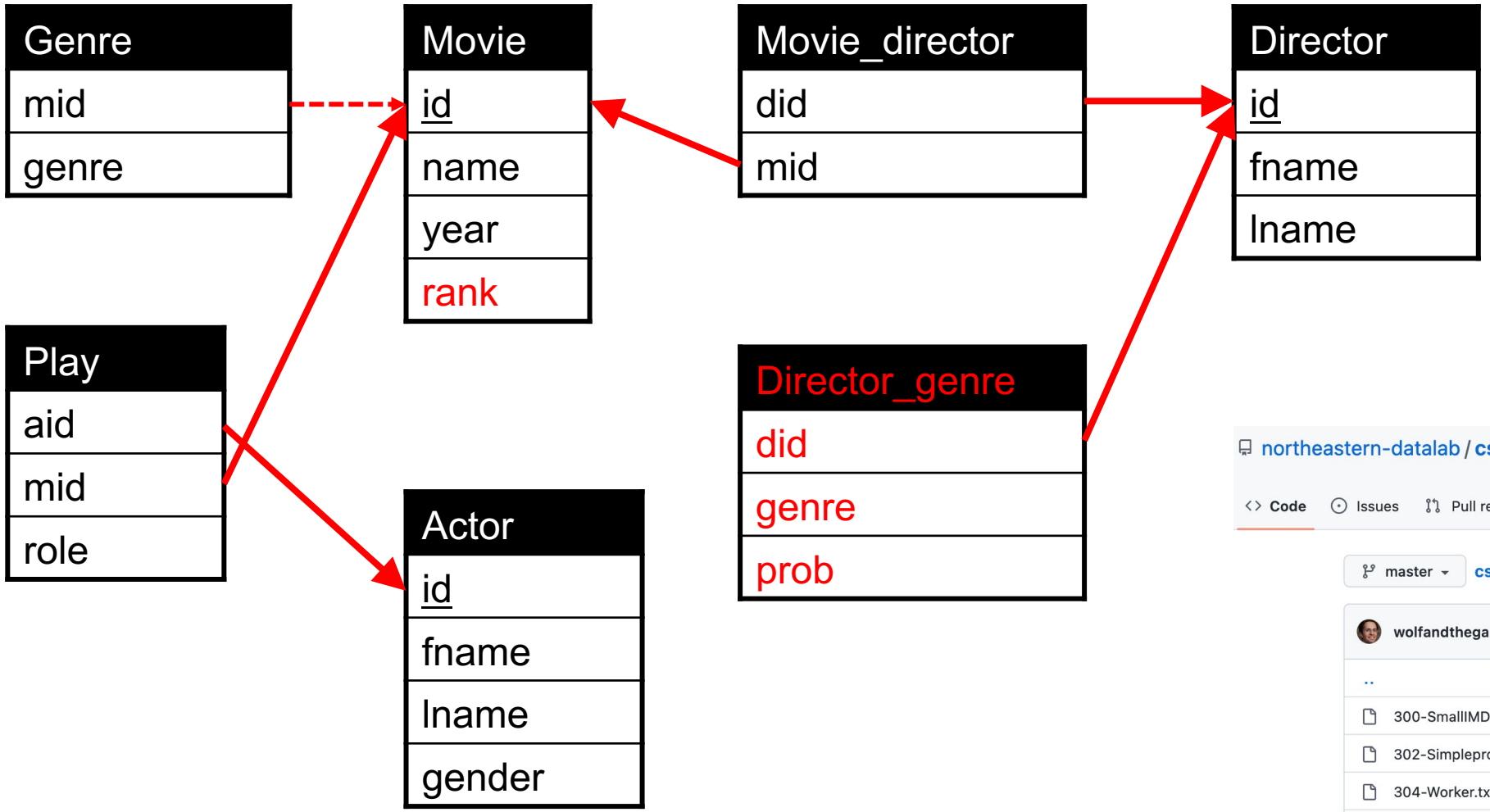


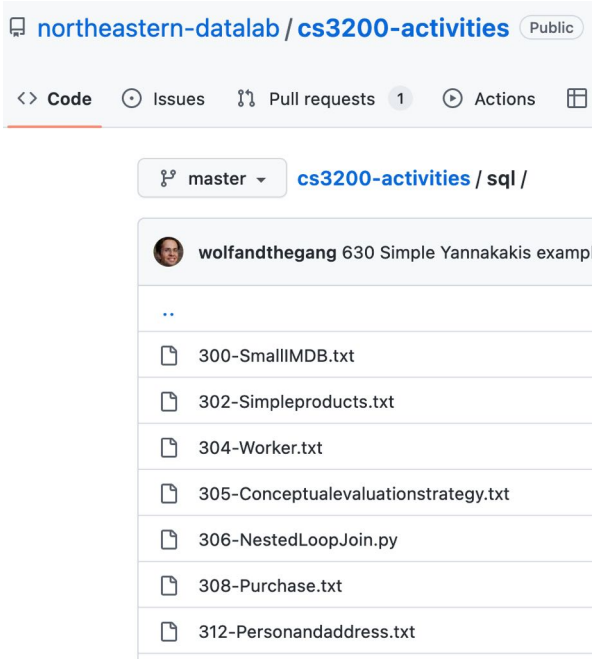
Source: https://chinookdatabase.codeplex.com/
Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

Small IMDB schema

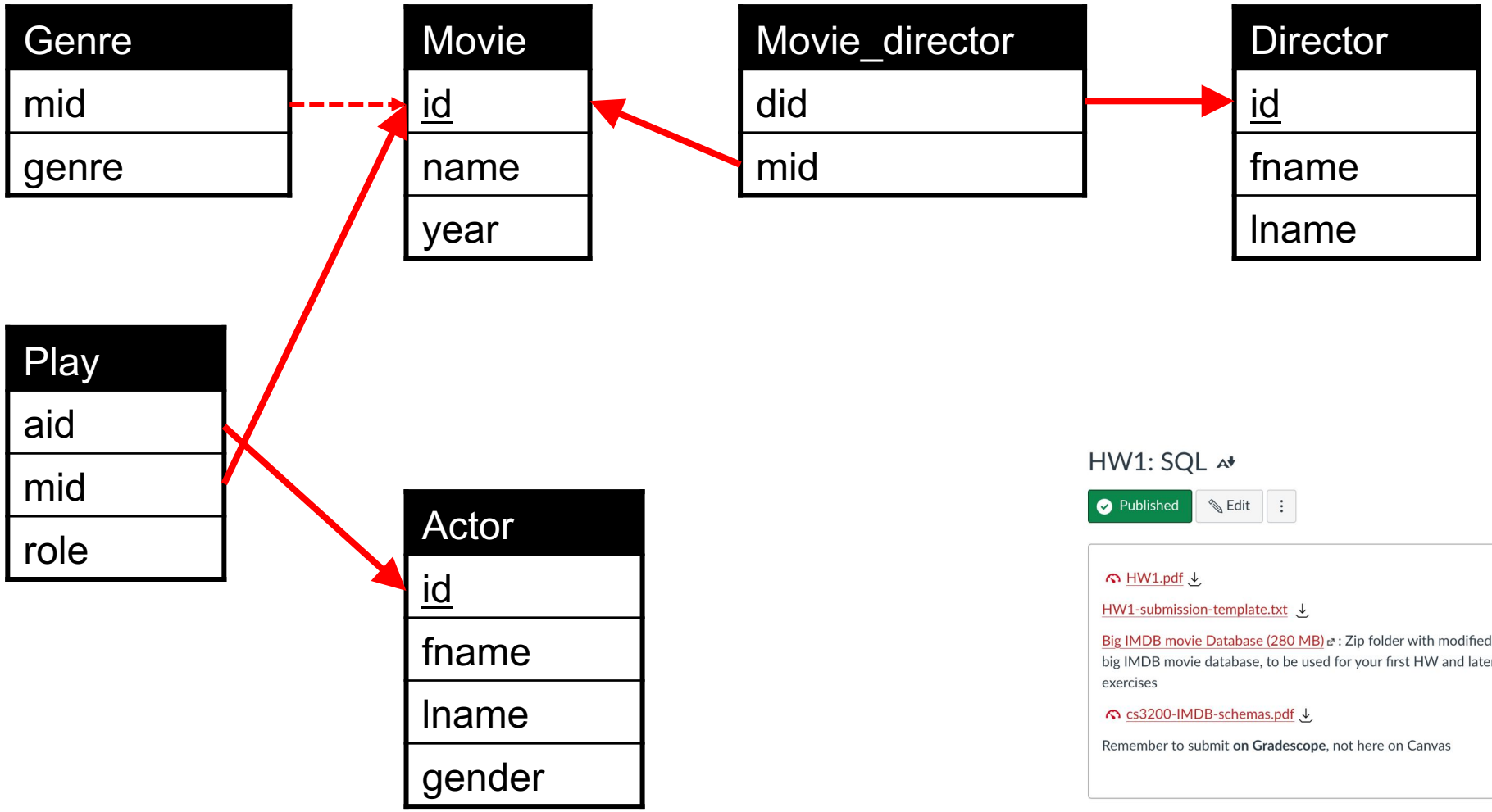
This is a far smaller movie database in our SQL folder

→ 300





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

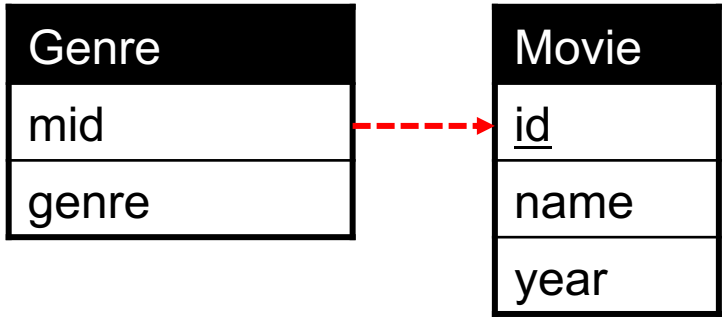
https://drive.google.com/file/d/1k4FwFQEEqygaQvNB38

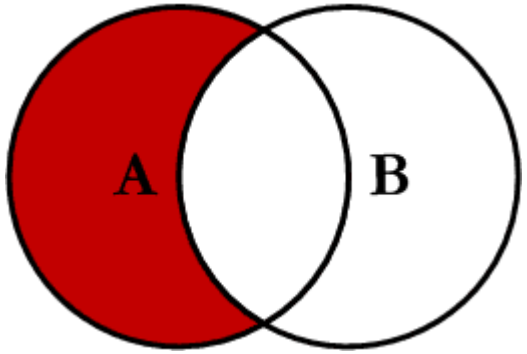
imdb-cs3200.zip

When our abstraction breaks

(when postgres does not know best)

Anti-join over the IMDB movie database





Large IMDB: # = 957k # = 2,384k

Small IMDB: # = 106 # = 36

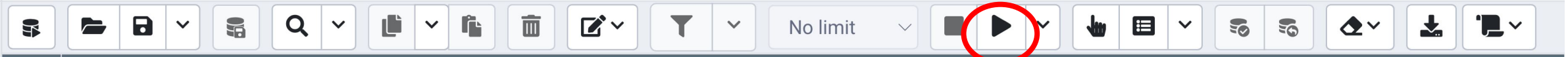
```
SELECT mid, genre
FROM Genre
LEFT JOIN Movie
ON mid = id
WHERE id IS NULL
```

time: ~4sec

result: 3973 rows

```
SELECT *
FROM Genre
WHERE mid NOT IN
   (SELECT id
   FROM Movie)
```

Does not complete within 15min



300-imdb/postgres@PostgreSQL 14

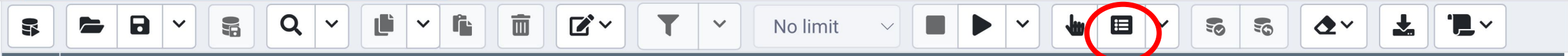
Query Editor | Query History

```
1  explain(
2  select count(*)
3  from Genre left join Movie
4  on mid = id
5  where id is null)
```

predicts the execution

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Aggregate (cost=19.45..19.46 rows=1 width=8)
2	[...] -> Hash Anti Join (cost=16.98..19.45 rows=1 width=0)
3	[...] Hash Cond: (genre.mid = movie.id)
4	[...] -> Seq Scan on genre (cost=0.00..2.07 rows=107 width=4)
5	[...] -> Hash (cost=13.10..13.10 rows=310 width=4)
6	[...] -> Seq Scan on movie (cost=0.00..13.10 rows=310 width=4)

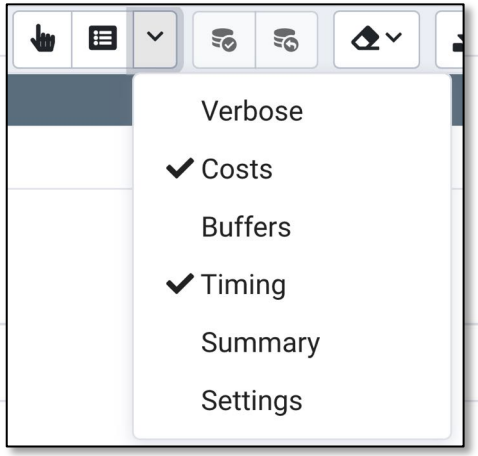


300-imdb/postgres@PostgreSQL 14

Query Editor | Query History

```
1  select count(*)
2  from Genre left join Movie
3  on mid = id
4  where id is null
```

actually runs the query



Data Output | Explain | Messages | Notifications

Graphical | Analysis | Statistics

	#	Node	Timings		Rows			Loops
			Exclusive	Inclusive	Rows X	Actual	Plan	
	1.	→ Aggregate (cost=19.45..19.46 rows=1 width=8) (actual=0.0…	0.002 ms	0.043 ms	↑ 1	1	1	1
	2.	→ Hash Anti Join (cost=16.98..19.45 rows=1 width=0) (a… Hash Cond: (genre.mid = movie.id)	0.011 ms	0.041 ms	↓ 4	4	1	1
	3.	→ Seq Scan on genre as genre (cost=0..2.07 rows=…	0.008 ms	0.008 ms	↑ 1	107	107	1
	4.	→ Hash (cost=13.1..13.1 rows=310 width=4) (actua… Buckets: 1024 Batches: 1 Memory Usage: 10 kB	0.02 ms	0.023 ms	↑ 8.62	36	310	1
	5.	→ Seq Scan on movie as movie (cost=0..13.1 r…	0.003 ms	0.003 ms	↑ 8.62	36	310	1

300-imdb/postgres@PostgreSQL 14

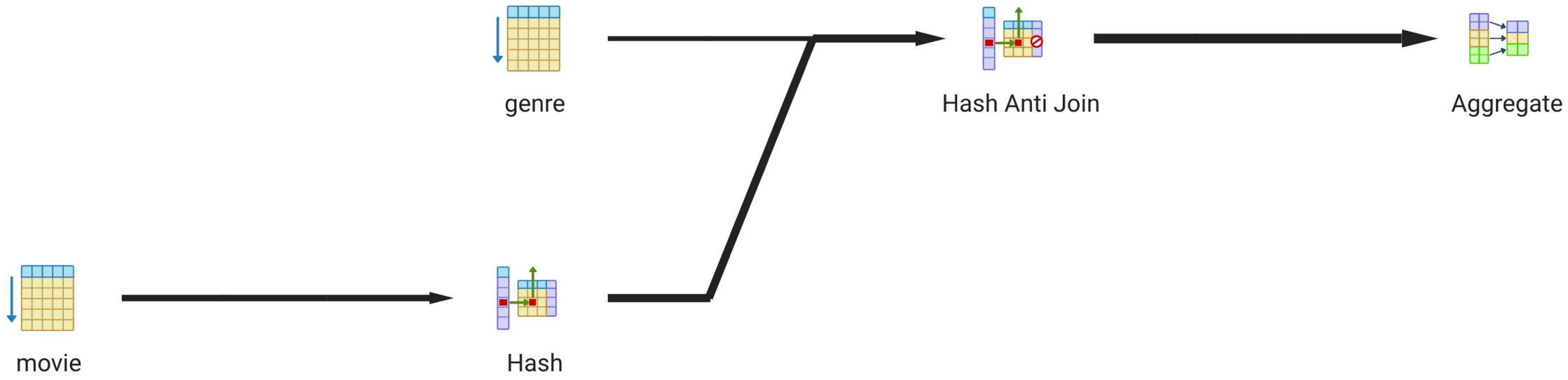
Query Editor | Query History

```
1  select count(*)
2  from Genre left join Movie
3  on mid = id
4  where id is null
```

actually runs the query

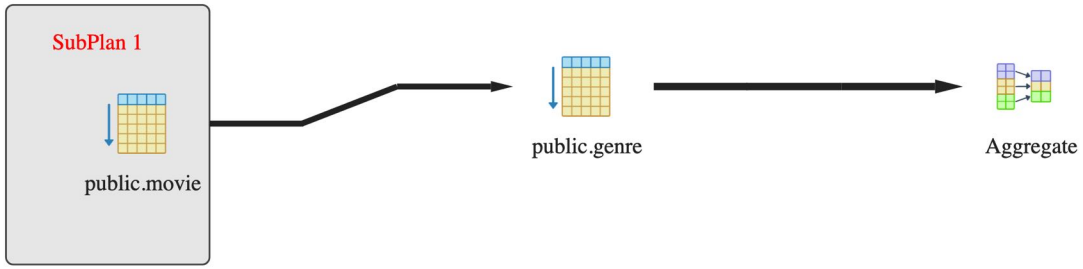
Data Output | Explain | Messages | Notifications

Graphical | Analysis | Statistics



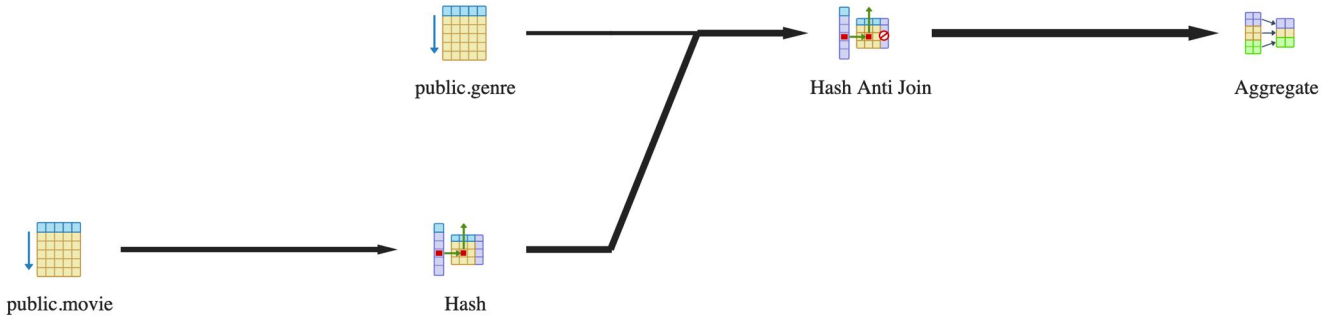
Small IMDB

	QUERY PLAN text
1	Aggregate (cost=16.35..16.36 rows=1 width=8)
2	[...] -> Seq Scan on genre (cost=13.88..16.21 rows=54 width=0)
3	[...] Filter: (NOT (hashed SubPlan 1))
4	[...] SubPlan 1
5	[...] -> Seq Scan on movie (cost=0.00..13.10 rows=310 width=4)




```
EXPLAIN(
  select count(*)
  from Genre
  where mid NOT IN
    (select id from Movie)
)
```

	QUERY PLAN text
1	Aggregate (cost=19.45..19.46 rows=1 width=8)
2	[...] -> Hash Anti Join (cost=16.98..19.45 rows=1 width=0)
3	[...] Hash Cond: (genre.mid = movie.id)
4	[...] -> Seq Scan on genre (cost=0.00..2.07 rows=107 width=4)
5	[...] -> Hash (cost=13.10..13.10 rows=310 width=4)
6	[...] -> Seq Scan on movie (cost=0.00..13.10 rows=310 width=4)




```
EXPLAIN(
  select count(*)
  from Genre left join Movie
  on mid = id
  where id is null
)
```

Read more on the EXPLAIN command: https://www.postgresql.org/docs/current/sql-explain.html

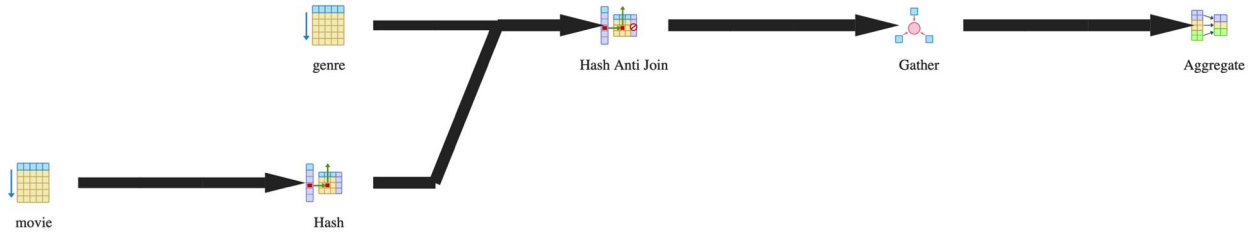
	QUERY PLAN text
1	Finalize Aggregate (cost=13475386273.99..13475386274.00 rows=1 width=8)
2	[...] -> Gather (cost=13475386273.78..13475386273.99 rows=2 width=8)
3	[...] Workers Planned: 2
4	[...] -> Partial Aggregate (cost=13475385273.78..13475385273.79 rows=1 width=8)
5	[...] -> Parallel Seq Scan on genre (cost=0.00..13475384775.51 rows=199307 width=0)
6	[...] Filter: (NOT (SubPlan 1))
7	[...] SubPlan 1
8	[...] -> Materialize (cost=0.00..61651.82 rows=2383721 width=4)
9	[...] -> Seq Scan on movie (cost=0.00..40421.21 rows=2383721 width=4)

?

```
EXPLAIN(
  select count(*)
  from Genre
  where mid NOT IN
    (select id from Movie)
)
```

Large IMDB

	QUERY PLAN text
1	Aggregate (cost=61451.43..61451.44 rows=1 width=8)
2	[...] -> Gather (cost=43811.38..61451.43 rows=1 width=0)
3	[...] Workers Planned: 2
4	[...] -> Parallel Hash Anti Join (cost=42811.38..60451.33 rows=1 width=0)
5	[...] Hash Cond: (genre.mid = movie.id)
6	[...] -> Parallel Seq Scan on genre (cost=0.00..9149.14 rows=398614 width=4)
7	[...] -> Parallel Hash (cost=26516.17..26516.17 rows=993217 width=4)
8	[...] -> Parallel Seq Scan on movie (cost=0.00..26516.17 rows=993217 width=4)




```
EXPLAIN(
  select count(*)
  from Genre left join Movie
  on mid = id
  where id is null
)
```

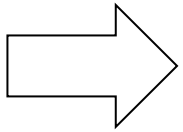
Read more on the EXPLAIN command: https://www.postgresql.org/docs/current/sql-explain.html
Wolfgang Gatterbauer. Database design: https://northeastern-datalab.github.io/cs3200/

Aggregates with null values

Aggregates with null value practice

Temporary table also requires attribute names

```
SELECT  A B, A C
FROM    (values (10), (20)) T(A)
```



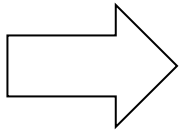
?

More details on CAST function: https://www.postgresql.org/docs/current/sql-expressions.html#SQL-SYNTAX-TYPE-CASTS

Aggregates with null value practice

Temporary table also requires attribute names

```
SELECT  A B, A C
FROM    (values (10), (20)) T(A)
```



B	C

?

More details on CAST function: https://www.postgresql.org/docs/current/sql-expressions.html#SQL-SYNTAX-TYPE-CASTS

Aggregates with null value practice

Temporary table also requires attribute names

```
SELECT  A B, A C
FROM    (values (10), (20)) T(A)
```

B	C
10	10
20	20

```
SELECT  count(A) B, sum(A) C
FROM    (values (10), (20)) T(A)
```

?

```
SELECT  count(A) B, sum(A) C
FROM    (values (10), (null)) T(A)
```

?

alternatively: "sum(cast(A as int)) as C"

```
SELECT  count(A) B, sum(A) C
FROM    (values (null::int), (null)) T(A)
```

?

also: "cast(null as int)"

More details on CAST function: https://www.postgresql.org/docs/current/sql-expressions.html#SQL-SYNTAX-TYPE-CASTS

Aggregates with null value practice

Temporary table also requires attribute names

```
SELECT  A B, A C
FROM    (values (10), (20)) T(A)
```

B	C
10	10
20	20

```
SELECT  count(A) B, sum(A) C
FROM    (values (10), (20)) T(A)
```

B	C
2	30

```
SELECT  count(A) B, sum(A) C
FROM    (values (10), (null)) T(A)
```

?

```
SELECT  count(A) B, sum(A) C
FROM    (values (null::int), (null)) T(A)
```

?

More details on CAST function: https://www.postgresql.org/docs/current/sql-expressions.html#SQL-SYNTAX-TYPE-CASTS

Aggregates with null value practice

Temporary table also requires attribute names

```
SELECT  A B, A C
FROM    (values (10), (20)) T(A)
```

B	C
10	10
20	20

```
SELECT  count(A) B, sum(A) C
FROM    (values (10), (20)) T(A)
```

B	C
2	30

```
SELECT  count(A) B, sum(A) C
FROM    (values (10), (null)) T(A)
```

B	C
1	10

aggregate functions ignore null values

```
SELECT  count(A) B, sum(A) C
FROM    (values (null::int), (null)) T(A)
```

?

More details on CAST function: https://www.postgresql.org/docs/current/sql-expressions.html#SQL-SYNTAX-TYPE-CASTS

Aggregates with null value practice

Temporary table also requires attribute names

```
SELECT  A B, A C
FROM    (values (10), (20)) T(A)
```

B	C
10	10
20	20

```
SELECT  count(A) B, sum(A) C
FROM    (values (10), (20)) T(A)
```

B	C
2	30

```
SELECT  count(A) B, sum(A) C
FROM    (values (10), (null)) T(A)
```

B	C
1	10

aggregate functions ignore null values

```
SELECT  count(A) B, sum(A) C
FROM    (values (null::int), (null)) T(A)
```

B	C
0	null

count initialized with 0 [size of empty set], all other aggregates with null [what should otherwise be "max({null})"?

More details on CAST function: https://www.postgresql.org/docs/current/sql-expressions.html#SQL-SYNTAX-TYPE-CASTS

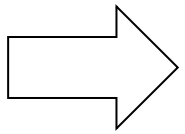
Practice with groupings

Grouping variants

Person

L	F	M	V
Smith	Alice	C.	1
Smith	Alice	NULL	2
Smith	Alice	NULL	3
Smith	Bob	NULL	4
Tiger	Alice	NULL	5

```
SELECT    L, F, M, max(V) MV
FROM      Person
GROUP BY  L, F, M
```



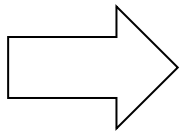
?

Grouping variants

Person

L	F	M	V
Smith	Alice	C.	1
Smith	Alice	NULL	2
Smith	Alice	NULL	3
Smith	Bob	NULL	4
Tiger	Alice	NULL	5

```
SELECT    L, F, M, max(V) MV
FROM      Person
GROUP BY  L, F, M
```



?

Grouping variants

Person

L	F	M	V
Smith	Alice	C.	1
Smith	Alice	NULL	2
Smith	Alice	NULL	3
Smith	Bob	NULL	4
Tiger	Alice	NULL	5

```
SELECT    L, F, M, max(V) MV
FROM      Person
GROUP BY  L, F, M
```

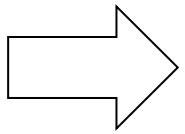
L	F	M	MV
Smith	Alice	C.	1
Smith	Alice	NULL	3
Smith	Bob	NULL	4
Tiger	Alice	NULL	5

Grouping variants

Person

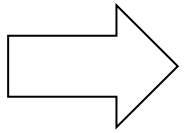
L	F	M	V
Smith	Alice	C.	1
Smith	Alice	NULL	2
Smith	Alice	NULL	3
Smith	Bob	NULL	4
Tiger	Alice	NULL	5

```
SELECT    L, F, M, max(V) MV
FROM      Person
GROUP BY  L, F, M
```



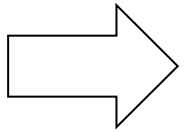
L	F	M	MV
Smith	Alice	C.	1
Smith	Alice	NULL	3
Smith	Bob	NULL	4
Tiger	Alice	NULL	5

```
SELECT    L, F, max(V) MV
FROM      Person
GROUP BY  L, F
```



?

```
SELECT    L, max(V) MV
FROM      Person
GROUP BY  L
```



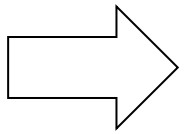
?

Grouping variants

Person

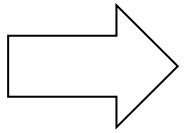
L	F	M	V
Smith	Alice	C.	1
Smith	Alice	NULL	2
Smith	Alice	NULL	3
Smith	Bob	NULL	4
Tiger	Alice	NULL	5

```
SELECT    L, F, M, max(V) MV
FROM      Person
GROUP BY  L, F, M
```



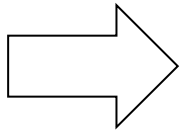
L	F	M	MV
Smith	Alice	C.	1
Smith	Alice	NULL	3
Smith	Bob	NULL	4
Tiger	Alice	NULL	5

```
SELECT    L, F, max(V) MV
FROM      Person
GROUP BY  L, F
```



L	F	MV
Smith	Alice	3
Smith	Bob	4
Tiger	Alice	5

```
SELECT    L, max(V) MV
FROM      Person
GROUP BY  L
```



?

Grouping variants

Person

L	F	M	V
Smith	Alice	C.	1
Smith	Alice	NULL	2
Smith	Alice	NULL	3
Smith	Bob	NULL	4
Tiger	Alice	NULL	5

```
SELECT    L, F, M, max(V) MV
FROM      Person
GROUP BY  L, F, M
```

L	F	M	MV
Smith	Alice	C.	1
Smith	Alice	NULL	3
Smith	Bob	NULL	4
Tiger	Alice	NULL	5

```
SELECT    L, F, max(V) MV
FROM      Person
GROUP BY  L, F
```

L	F	MV
Smith	Alice	3
Smith	Bob	4
Tiger	Alice	5

```
SELECT    L, max(V) MV
FROM      Person
GROUP BY  L
```

L	MV
Smith	4
Tiger	5

?

MV
5

Grouping variants

Person

L	F	M	V
Smith	Alice	C.	1
Smith	Alice	NULL	2
Smith	Alice	NULL	3
Smith	Bob	NULL	4
Tiger	Alice	NULL	5

```
SELECT   L, F, M, max(V) MV
FROM     Person
GROUP BY L, F, M
```

L	F	M	MV
Smith	Alice	C.	1
Smith	Alice	NULL	3
Smith	Bob	NULL	4
Tiger	Alice	NULL	5

```
SELECT   L, F, max(V) MV
FROM     Person
GROUP BY L, F
```

L	F	MV
Smith	Alice	3
Smith	Bob	4
Tiger	Alice	5

```
SELECT   L, max(V) MV
FROM     Person
GROUP BY L
```

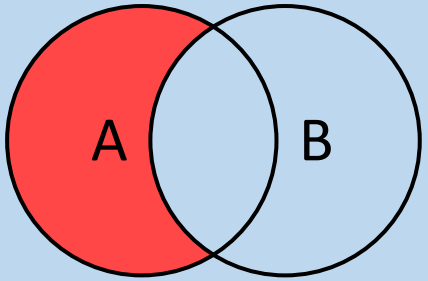
L	MV
Smith	4
Tiger	5

Intuition: group by Ø: empty set: tuples grouped together need to share no attributes.

```
SELECT  max(V) MV
FROM    Person
```

MV
5

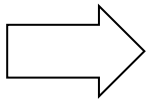
Anti-joins and null values



Remember NULL values

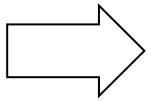
true or false?

```
SELECT 1 in (2, 3)
```



?

```
SELECT 1 NOT in (2, 3)
```



?

Remember NULL values

true or false?

SELECT 1 in (2, 3) ⇨ false

SELECT 1 NOT in (2, 3) ⇨ true

SELECT 1 in (2, 3, NULL) ⇨ ?

SELECT 1 NOT in (2, 3, NULL) ⇨ ?

Remember NULL values

true or false?

SELECT 1 in (2, 3) ⇨ false

SELECT 1 NOT in (2, 3) ⇨ true

SELECT 1 in (2, 3, NULL) ⇨ NULL

SELECT 1 NOT in (2, 3, NULL) ⇨ NULL

SELECT NULL NOT in (2, 3) ⇨ ?

SELECT NULL NOT in (2, 3, NULL) ⇨ ?

Remember NULL values

true or false?

```
SELECT 1 in (2, 3)
```
⇨ false

```
SELECT 1 NOT in (2, 3)
```
⇨ true

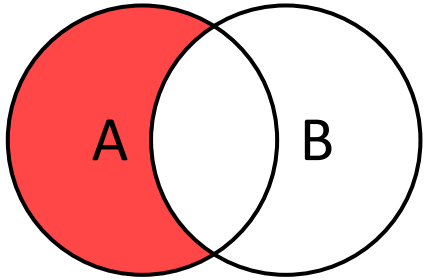
```
SELECT 1 in (2, 3, NULL)
```
⇨ NULL

```
SELECT 1 NOT in (2, 3, NULL)
```
⇨ NULL

```
SELECT NULL NOT in (2, 3)
```
⇨ NULL

```
SELECT NULL NOT in (2, 3, NULL)
```
⇨ NULL

Anti-joins with NULLs



L'

A	B
a	1
b	2
c	3

R

B	C
2	d
3	e
4	f

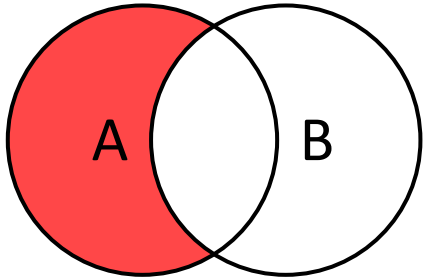
```
SELECT distinct A, L.B
FROM L
LEFT JOIN R
ON L.B = R.B
WHERE R.B IS NULL
```

⇨ ?

```
SELECT distinct *
FROM L
WHERE B NOT IN
    (SELECT B
    FROM R)
```

⇨ ?

Anti-joins with NULLs



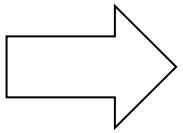
L'

A	B
a	1
b	2
c	3

R

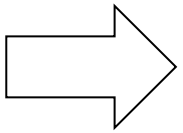
B	C
2	d
3	e
4	f

```
SELECT distinct A, L.B
FROM L
LEFT JOIN R
ON L.B = R.B
WHERE R.B IS NULL
```



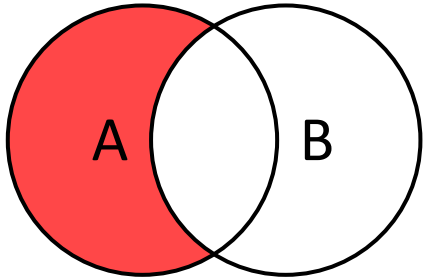
A	B
a	1

```
SELECT distinct *
FROM L
WHERE B NOT IN
    (SELECT B
    FROM R)
```



A	B
a	1

Anti-joins with NULLs



L

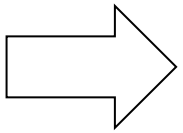
A	B
a	1
b	2
c	NULL

R

B	C
2	d
3	e
4	f

330

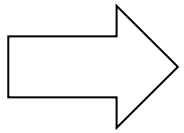
```
SELECT distinct A, L.B
FROM L
LEFT JOIN R
ON L.B = R.B
WHERE R.B IS NULL
```



A	B
a	1

?

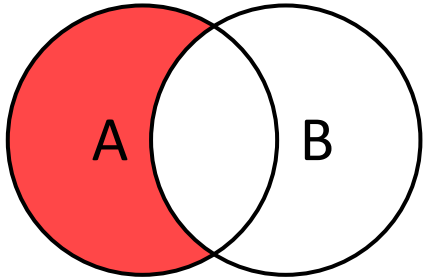
```
SELECT distinct *
FROM L
WHERE B NOT IN
   (SELECT B
   FROM R)
```



A	B
a	1

?

Anti-joins with NULLs



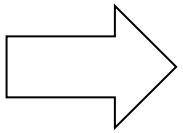
L

A	B
a	1
b	2
c	NULL

R

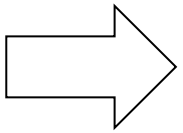
B	C
2	d
3	e
4	f

```
SELECT distinct A, L.B
FROM L
LEFT JOIN R
ON L.B = R.B
WHERE R.B IS NULL
```



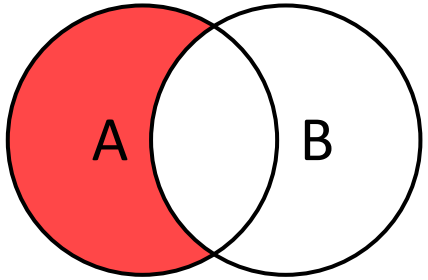
A	B
a	1
c	NULL

```
SELECT distinct *
FROM L
WHERE B NOT IN
   (SELECT B
   FROM R)
```



A	B
a	1

Anti-joins with NULLs



L

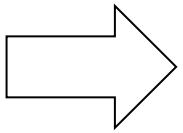
A	B
a	1
b	2
c	NULL

R'

B	C
2	d
3	e
NULL	f

330

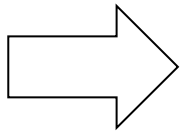
```
SELECT distinct A, L.B
FROM L
LEFT JOIN R
ON L.B = R.B
WHERE R.B IS NULL
```



A	B
a	1
c	NULL

?

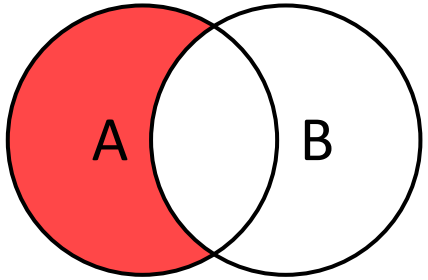
```
SELECT distinct *
FROM L
WHERE B NOT IN
    (SELECT B
    FROM R)
```



A	B
a	1

?

Anti-joins with NULLs



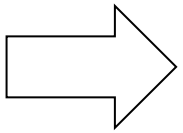
L

A	B
a	1
b	2
c	NULL

R'

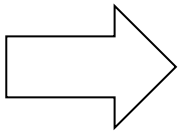
B	C
2	d
3	e
NULL	f

```
SELECT distinct A, L.B
FROM L
LEFT JOIN R
ON L.B = R.B
WHERE R.B IS NULL
```



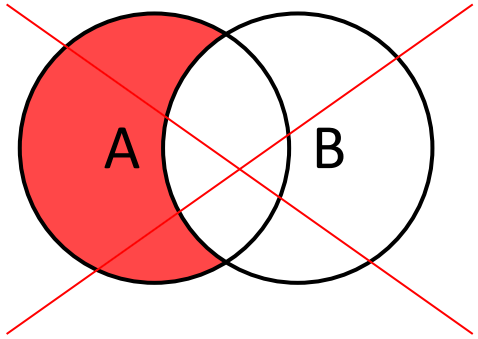
A	B
a	1
c	NULL

```
SELECT distinct *
FROM L
WHERE B NOT IN
   (SELECT B
   FROM R)
```



A	B

Incorrect Anti-joins



330

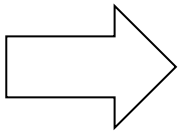
L

A	B
a	1
b	2
c	NULL

R

B	C
2	d
3	e
4	f

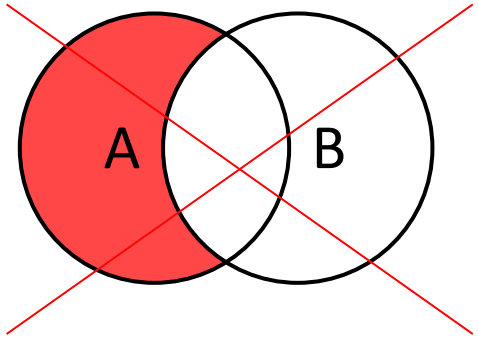
```
SELECT distinct *
FROM L, R
WHERE L.B NOT IN
    (SELECT B
    FROM R)
```



A	B
a	1

?

Incorrect Anti-joins



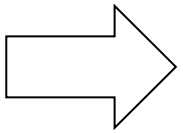
L

A	B
a	1
b	2
c	NULL

R

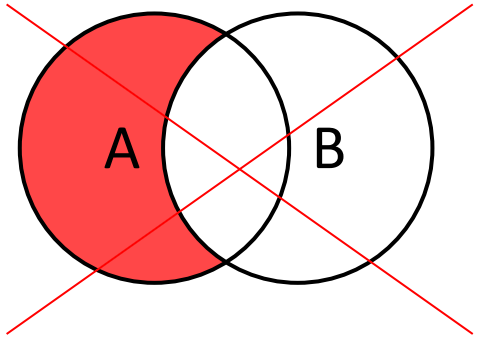
B	C
2	d
3	e
4	f

```
SELECT distinct *
FROM L, R
WHERE L.B NOT IN
    (SELECT B
    FROM R)
```



A	L.B	R.B	C
a	1	2	d
a	1	3	e
a	1	4	f

Incorrect Anti-joins



L

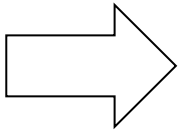
A	B
a	1
b	2
c	NULL

R

B	C
2	d
3	e
4	f

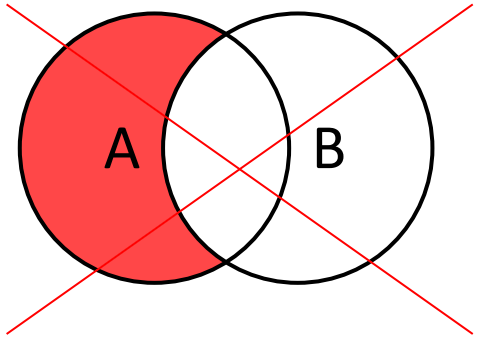
```
SELECT distinct *
FROM L, R
WHERE L.B != R.B
```

?



A	L.B	R.B	C
a	1	2	d
a	1	3	e
a	1	4	f

Incorrect Anti-joins



330

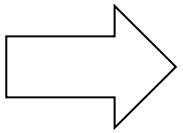
L

A	B
a	1
b	2
c	NULL

R

B	C
2	d
3	e
4	f

```
SELECT distinct *
FROM L, R
WHERE L.B != R.B
```



A	L.B	R.B	C
a	1	2	d
a	1	3	e
a	1	4	f
b	2	3	e
b	2	4	f

Outer Joins, Coalesce, and non-distributivity

Coalesce function

M

a
1
2

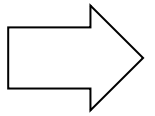
N

a
2
3

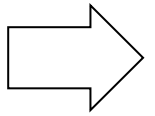
```
SELECT M.a, N.a, COALESCE(M.a, N.a) as b
FROM M
FULL JOIN N
ON M.a = N.a
```

COALESCE: takes first non-NULL value,

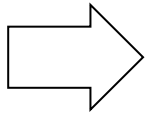
```
SELECT COALESCE(1, NULL)
```



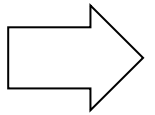
```
SELECT COALESCE(NULL, 3)
```



```
SELECT COALESCE(1, 2)
```



```
SELECT COALESCE(NULL, NULL)
```



?

Also see use of COALESCE across programming languages: https://en.wikipedia.org/wiki/Null_coalescing_operator

Coalesce function

M

a
1
2

N

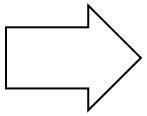
a
2
3

```
SELECT M.a, N.a, COALESCE(M.a, N.a) as b
FROM M
FULL JOIN N
ON M.a = N.a
```

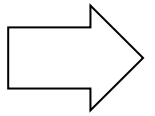
COALESCE: takes first non-NULL value,
C(x,y,z)=C(x,C(y,z))=C(C(x,y),z)

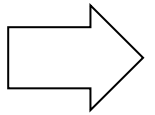
Result

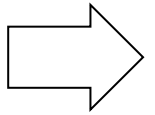
M.a	N.a	b

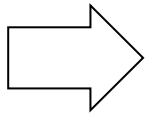


? ?

```
SELECT COALESCE(1, NULL)
```

1

```
SELECT COALESCE(NULL, 3)
```

3

```
SELECT COALESCE(1, 2)
```

1

```
SELECT COALESCE(NULL, NULL)
```

NULL

Also see use of COALESCE across programming languages: https://en.wikipedia.org/wiki/Null_coalescing_operator

Coalesce function

M

a
1
2

N

a
2
3

```
SELECT M.a, N.a, COALESCE(M.a, N.a) as b
FROM M
FULL JOIN N
ON M.a = N.a
```

COALESCE: takes first non-NULL value,
C(x,y,z)=C(x,C(y,z))=C(C(x,y),z)

Result

M.a	N.a	b
1	NULL	1
2	2	2
NULL	3	3

```
SELECT COALESCE(1, NULL)
```
⇨ 1

```
SELECT COALESCE(NULL, 3)
```
⇨ 3

```
SELECT COALESCE(1, 2)
```
⇨ 1

```
SELECT COALESCE(NULL, NULL)
```
⇨ NULL

Also see use of COALESCE across programming languages: https://en.wikipedia.org/wiki/Null_coalescing_operator

Coalesce, Natural Outer Join, Union

M

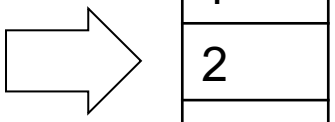
a
1
2

N

a
2
3

```
SELECT *
FROM M
NATURAL FULL JOIN N
```

Result



a
1
2
3

Natural full join models "coalesce"

Join vs. Union – it is actually the same:
Union is a special case of a join ☺
(under set semantics)

Quick recap: Commutativity & Associativity

Multiplication

$3 \bullet 2 \bullet 4 = 24$

?

Multiplication is associative ☺

Matrix multiplication

$$\begin{bmatrix} 2 & 3 \end{bmatrix} \bullet \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \bullet \begin{bmatrix} 3 \\ 1 \end{bmatrix} =$$

Quick recap: Commutativity & Associativity

Multiplication

$(3 \cdot 2) \cdot 4 = 24$

Order of operations can be exchanged:

$3 \cdot (2 \cdot 4) = 24$

Multiplication is associative ☺

?

and commutative ☺

Matrix multiplication

$$\begin{bmatrix} 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 1 \end{bmatrix} =$$

Quick recap: Commutativity & Associativity

Multiplication

$$(3 \cdot 2) \cdot 4 = 24$$

Order of operations can be exchanged:

$$3 \cdot (2 \cdot 4) = 24$$

Multiplication is associative ☺

Order of operands can be exchanged:

$$4 \cdot 2$$

and commutative ☺

Matrix multiplication

$$\begin{bmatrix} 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 1 \end{bmatrix} =$$

?

Quick recap: Commutativity & Associativity

Multiplication

$(3 \cdot 2) \cdot 4 = 24$

Order of operations can be exchanged:

$3 \cdot (2 \cdot 4) = 24$

Multiplication is associative ☺

Order of operands can be exchanged:

$4 \cdot 2$

and commutative ☺

Matrix multiplication

$$\left(\begin{bmatrix}1&1\\2&3\end{bmatrix} \cdot \begin{bmatrix}1&2\\3&4\end{bmatrix}\right) \cdot \begin{bmatrix}3\\1\end{bmatrix} = \begin{bmatrix}18\\49\end{bmatrix}$$

$$\begin{bmatrix}4&6\\11&16\end{bmatrix}$$

$$\begin{bmatrix}1&1\\2&3\end{bmatrix} \cdot \left(\begin{bmatrix}1&2\\3&4\end{bmatrix} \cdot \begin{bmatrix}3\\1\end{bmatrix}\right) = \begin{bmatrix}18\\49\end{bmatrix}$$

$$\begin{bmatrix}5\\13\end{bmatrix}$$

Matrix multipl. is associative ☺

$$\begin{bmatrix}3\\1\end{bmatrix} \cdot \begin{bmatrix}1&2\\3&4\end{bmatrix}$$

#col ≠ #row

... but *not* commutative ☹

It turns out this is mainly a problem of syntax, not semantics. Einstein notation solves that. See e.g. Laue et al. *A Simple and Efficient Tensor Calculus*. AAAI 2020. https://arxiv.org/abs/2010.03313

Commutativity & Associativity

Outer joins

R

A	B
1	2

S

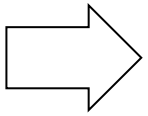
B	C
2	3

T

A	C
4	5

```
SELECT A, B, C
FROM (R
NATURAL FULL JOIN S)
NATURAL FULL JOIN T
```

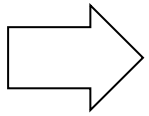
Result



?

```
SELECT  A, B, C
FROM R
NATURAL FULL JOIN (S
NATURAL FULL JOIN T)
```

Result



?

Commutativity & Associativity

Outer joins

R

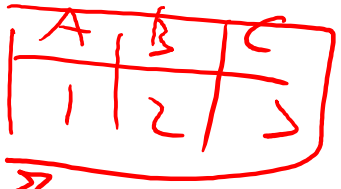
A	B
1	2

S

B	C
2	3

T

A	C
4	5



```
SELECT A, B, C
FROM (R
NATURAL FULL JOIN S)
NATURAL FULL JOIN T
```

Result

A	B	C
1	2	3
4	NULL	5

```
SELECT  A, B, C
FROM R
NATURAL FULL JOIN (S
NATURAL FULL JOIN T)
```

Result

?

Commutativity & Associativity

Outer joins

R

A	B
1	2

S

B	C
2	3

T

A	C
4	5

A	B	C
1	2	3
4	L	5

```
SELECT A, B, C
FROM (R
NATURAL FULL JOIN S)
NATURAL FULL JOIN T
```

Result

A	B	C
1	2	3
4	NULL	5

```
SELECT  A, B, C
FROM R
NATURAL FULL JOIN (S
NATURAL FULL JOIN T)
```

Result

A	B	C
1	2	NULL
NULL	2	3
4	NULL	5

Thus outer joins are not associative! (but they are commutative)

Outer Joins: summary

- Left (outer) join:
  - Include the left tuple even if there's no match
- Right (outer) join:
  - Include the right tuple even if there's no match
- Full (outer) join:
  - Include both left and right tuples even if there's no match

- (inner) join:
  - Include only the matches

Processing Multiple Tables–Joins

- **Join**: a relational operation that causes two or more tables with a common domain to be combined into a single table or view
- **Equi-join**: a join in which the joining condition is based on equality between values in the common columns; common columns appear redundantly in the result table
- **Natural join**: an equi-join in which one of the duplicate columns is eliminated in the result table
- A **Theta-join** allows for arbitrary comparison relationships (e.g., ≥). An equijoin is a theta join using the equality operator.

The common columns in joined tables are usually the primary key of the dominant table and the foreign key of the dependent table in 1:M relationships

Processing Multiple Tables–Joins

- **Left Outer join**: a join in which rows from the left table that do not have matching values in common columns are nonetheless included in the result table (as opposed to inner join, in which rows must have matching values in order to appear in the result table)

- Union join ("**Full outer join**"): includes all columns from each table in the join, and an instance for each row of each table

Set & Multiset operations (UNION, INTERSECT, EXCEPT)

Set Operations

- We can apply union, intersection and difference to two (or more) queries
  - (Q_1) UNION (Q_2)
  - (Q_1) INTERSECT (Q_2)
  - (Q_1) EXCEPT (Q_2)
- Subqueries must be union compatible in a weak sense
  - Same #attributes
  - Types of corresponding attributes must be convertible to each other (e.g., int to float)
  - The output adopts names of 1st subquery

Bag or Set Semantics?

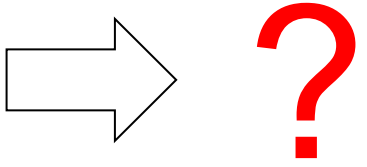
- Default is set semantics:
  1. Eliminate duplicates
  2. Apply operator
  3. Eliminate duplicates
- For bag semantics, use the keyword ALL
  - (Q_1) UNION ALL (Q_2)
  - (Q_1) INTERSECT ALL (Q_2)
  - (Q_1) EXCEPT ALL (Q_2)

Union

R

a	b
1	A
2	B
3	C

```
(select a e from R)
UNION
(select c from S)
order by e
```

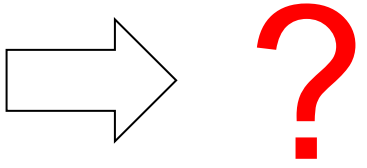


?

S

c	d
1	A
4	D
5	E

```
(select a e from R)
UNION ALL
(select c from S)
order by e
```



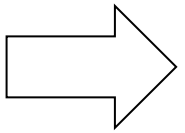
?

Union

R

a	b
1	A
2	B
3	C

```
(select a e from R)
UNION
(select c from S)
order by e
```

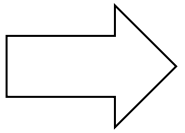


e
1
2
3
4
5

S

c	d
1	A
4	D
5	E

```
(select a e from R)
UNION ALL
(select c from S)
order by e
```



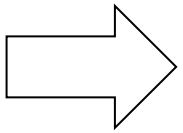
e
1
1
2
3
4
5

Do we need a "union"?

R

a	b
1	A
2	B
3	C

```
(select a e from R)
UNION
(select c from S)
order by e
```

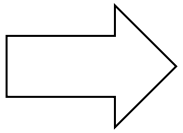


e
1
2
3
4
5

S

c	d
1	A
4	D
5	E

?



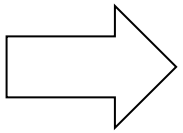
e
1
2
3
4
5

Do we need a "union"?

R

a	b
1	A
2	B
3	C

```
(select a e from R)
UNION
(select c from S)
order by e
```

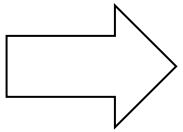


e
1
2
3
4
5

S

c	d
1	A
4	D
5	E

```
select
  COALESCE(R.a,S.c) e
from  R FULL JOIN S
on  R.a = S.c
```



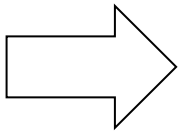
e
1
2
3
4
5

Do we need a "union"?

R

a	b
1	A
2	B
3	C

```
(select a e from R)
UNION
(select c from S)
order by e
```

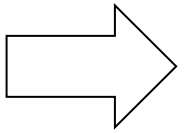


e
1
2
3
4
5

S

c	d
1	A
4	D
5	E

```
select
  COALESCE(R.a,S.c) e, *
from  R FULL JOIN S
on  R.a = S.c
```



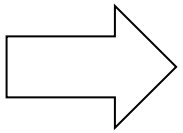
e	a	b	c	d
1	1	A	1	A
2	2	B	null	null
3	3	C	null	null
4	null	null	4	D
5	null	null	5	E

Do we need a "union"?

R

a	b
1	A
2	B
3	C

```
(select a e, b f from R)
UNION
(select c, d from S)
order by e
```

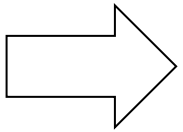


e	f
1	A
2	B
3	C
4	D
5	E

S

c	d
1	A
4	D
5	E

```
select
  COALESCE(R.a,S.c) as e,
  COALESCE(R.b,S.d) as f, *
from  R FULL JOIN S
on  R.a = S.c and  R.b = S.d
```



e	f	a	b	c	d
1	A	1	A	1	A
2	B	2	B	null	null
3	C	3	C	null	null
4	D	null	null	4	D
5	E	null	null	5	E

UNION (∪), INTERSECT, EXCEPT (Difference) (–)

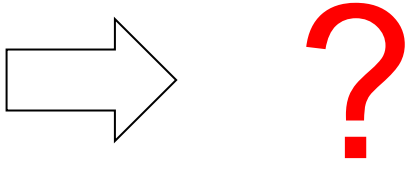
R

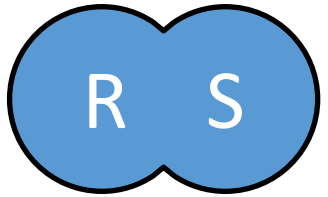
a	b
1	A
2	B
3	C

S

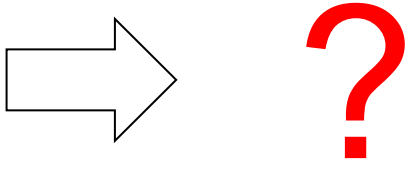
c	d
1	A
4	D
5	E

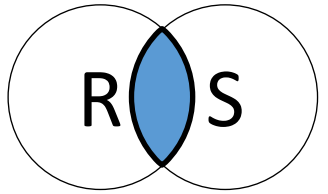
```
(select a e, b f from R)
UNION
(select c, d from S)
```



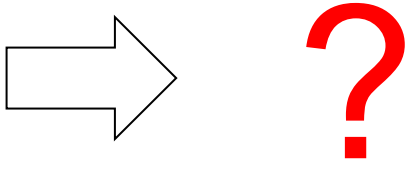


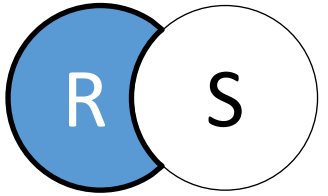

```
(select a e, b f from R)
INTERSECT
(select c, d from S)
```






```
(select a e, b f from R)
EXCEPT
(select c, d from S)
```





Not in all DBMSs. (SQLlite does not like the parentheses, Oracle used until recently "MINUS" instead of "EXCEPT")

UNION (∪), INTERSECT, EXCEPT (Difference) (–)

R

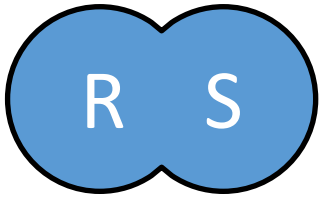
a	b
1	A
2	B
3	C

S

c	d
1	A
4	D
5	E

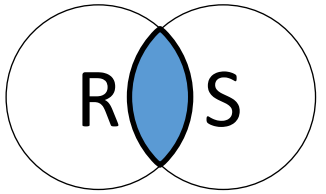
```
(select a e, b f from R)
UNION
(select c, d from S)
```

e	f
1	A
2	B
3	C
4	D
5	E



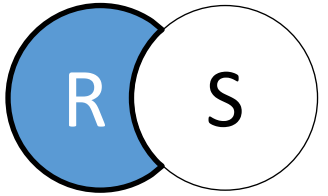

```
(select a e, b f from R)
INTERSECT
(select c, d from S)
```

e	f
1	A



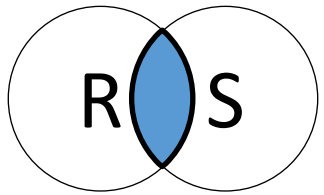

```
(select a e, b f from R)
EXCEPT
(select c, d from S)
```

e	f
2	B
3	C

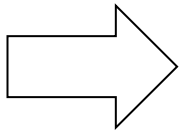


Not in all DBMSs. (SQLlite does not like the parentheses, Oracle used until recently "MINUS" instead of "EXCEPT")

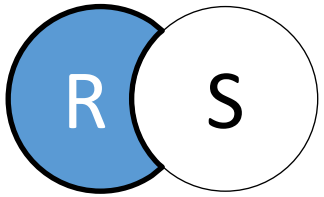
UNION (∪), INTERSECT, EXCEPT (Difference) (–)



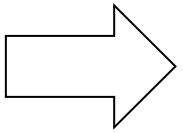

```
(select a e, b f
from R)
INTERSECT
(select c, d
from S)
```



?

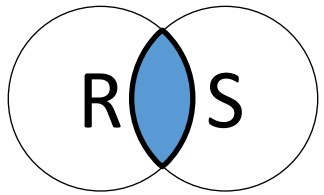



```
(select a e, b f
from R)
EXCEPT
(select c, d
from S)
```

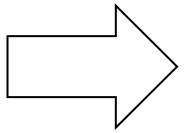


?

UNION (∪), INTERSECT, EXCEPT (Difference) (–)

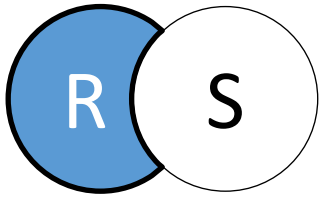



```
(select a e, b f
from R)
INTERSECT
(select c, d
from S)
```

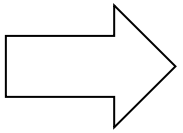


```
select distinct a e, b f
from R
where EXISTS
  (select *
  from S
  where R.a=S.c and R.b=S.d)
```

can be unnested




```
(select a e, b f
from R)
EXCEPT
(select c, d
from S)
```



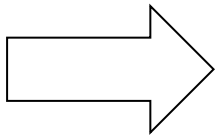
```
select distinct a e, b f
from R
where NOT EXISTS
  (select *
  from S
  where R.a=S.c and R.b=S.d)
```

Alternative representation for Multisets (Bags)

R0 (with duplicates)

a	b
1	A
1	A
2	B
2	B
3	C
3	C
3	C

equivalent representations of a **multiset**



R (with counts)

a	b	λ
1	A	2
2	B	2
3	C	3

$\lambda(\text{tuple})$= "Count of tuple in R0"
(Items not listed have implicit count 0)

?

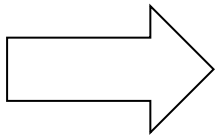
Note: In a set all counts are {0,1}.

Alternative representation for Multisets (Bags)

R0 (with duplicates)

a	b
1	A
1	A
2	B
2	B
3	C
3	C
3	C

equivalent representations of a **multiset**



R (with counts)

a	b	λ
1	A	2
2	B	2
3	C	3

$\lambda(\text{tuple})$= "Count of tuple in R0"
(Items not listed have implicit count 0)

```
SELECT *, count(*)
FROM R0
GROUP BY a, b
```

Note: In a set all counts are {0,1}.

Alternative representation: UNION ALL

R

a	b	λ
1	A	2
2	B	2
3	C	3

U

S

a	b	λ
1	A	5
3	C	2
4	D	1

=

a	b	λ
1	A	7
2	B	2
3	C	5
4	D	1

$$\boldsymbol{\lambda(\mathbf{OUT}) = \lambda(r) + \lambda(s)}$$

```
(select *
from R0)
UNION ALL
(select *
from S0)
```

Alternative representation: UNION ALL

R

a	b	λ
1	A	2
2	B	2
3	C	3
4	D	0

U

S

a	b	λ
1	A	5
2	B	0
3	C	2
4	D	1

=

a	b	λ
1	A	7
2	B	2
3	C	5
4	D	1

$$\boldsymbol{\lambda(\mathbf{OUT}) = \lambda(r) + \lambda(s)}$$

```
(select *
from R0)
UNION ALL
(select *
from S0)
```

```
select a, b, sum(lambda)
from
  ((select * from R)
  UNION ALL
  (select * from S)) X
group by a, b
```

```
select
  COALESCE(R.a,S.a) a,
  COALESCE(R.b,S.b) b,
  COALESCE(R.lambda, 0)
    + COALESCE(S.lambda, 0) lambda
from R full join S
on R.a = S.a and R.b = S.b
```

Alternative representation: INTERSECT ALL

R

a	b	λ
1	A	2
2	B	2
3	C	3

$\cap$

S

a	b	λ
1	A	5
3	C	2
4	D	1

$=$

a	b	λ
1	A	2
3	C	2
2	B	0
4	D	0

$$\boldsymbol{\lambda(\mathbf{OUT}) = \min\{\lambda(\mathbf{r}), \lambda(\mathbf{s})\}}$$

```
(select *
from R0)
INTERSECT ALL
(select*
from S0)
```

```
select R.a, R.b,
  CASE
    WHEN R.lambda>S.lambda THEN S.lambda
    ELSE R.lambda
  END AS lambda
from R, S
where R.a=S.a and R.b=S.b
```

Alternative representation: EXCEPT ALL

R

a	b	λ
1	A	2
2	B	2
3	C	3

$-$

S

a	b	λ
1	A	5
3	C	2
4	D	1

$=$

a	b	λ
2	B	2
3	C	1
1	A	0
4	D	0

$$\boldsymbol{\lambda(\mathbf{OUT})} = \max\{\boldsymbol{\lambda(r)} - \boldsymbol{\lambda(s)}, 0\}$$

```
(select *
from R0)
EXCEPT ALL
(select *
from S0)
```

```
select R.a, R.b,
  CASE
    WHEN R.lambda>coalesce(S.lambda,0)
    THEN R.lambda-coalesce(S.lambda,0)
    ELSE 0
  END AS lambda
from R left join S
on R.a=S.a and R.b=S.b
```

Alternative multiset representation: EXCEPT ALL

R

a	b	λ
1	A	2
2	B	2
3	C	3

—

S

a	b	λ
1	A	5
3	C	2
4	D	1

=

a	b	λ
2	B	2
3	C	1
1	A	0
4	D	0

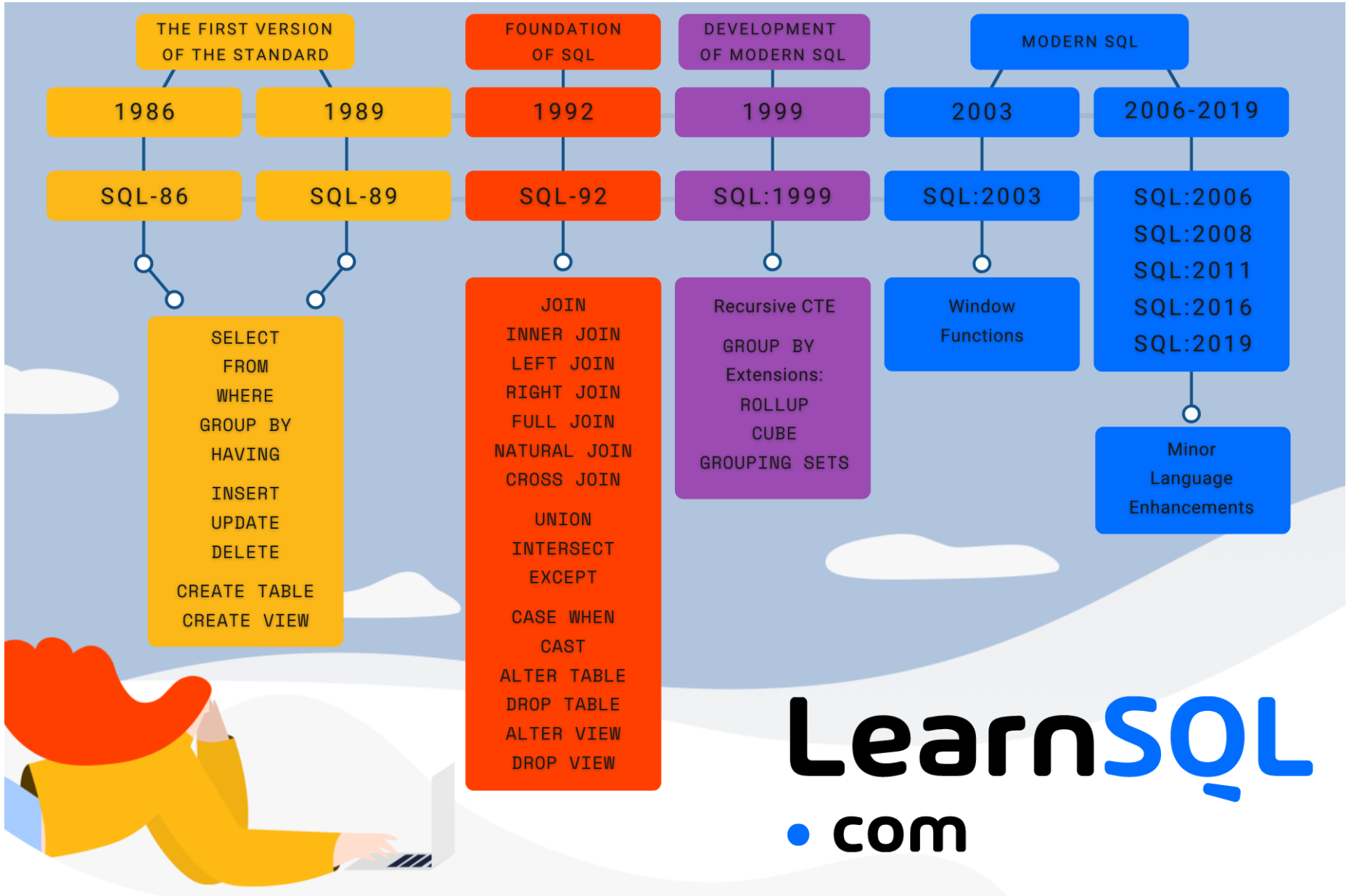
$$\boldsymbol{\lambda(\mathbf{OUT})} = \max\{\boldsymbol{\lambda(r)} - \boldsymbol{\lambda(s)}, 0\}$$

```
(select *
from R0)
EXCEPT ALL
(select *
from S0)
```

```
select * from
(select R.a, R.b,
  CASE
    WHEN R.lambda>coalesce(S.lambda,0)
    THEN R.lambda-coalesce(S.lambda,0)
    ELSE 0
  END AS lambda
from R left join S
on R.a=S.a and R.b=S.b) X
where lambda>0
```

Window functions

The History of SQL Standards



Picture source: https://learnsql.com/blog/history-of-sql-standards/

Differences to minimum price

what we want

would be easy if we had that

Purchase

product	price	quantity	delta	minp
Apple	3	20	2	1
Apple	2	20	1	1
Banana	1	50	0	1
Banana	2	10	1	1
Banana	4	10	3	1

?

X

minp
1

Differences to minimum price

what we want

would be easy if we had that

Purchase

product	price	quantity	delta	minp
Apple	3	20	2	1
Apple	2	20	1	1
Banana	1	50	0	1
Banana	2	10	1	1
Banana	4	10	3	1

```
SELECT  min(price) minp
FROM  Purchase
```

?

X

minp
1

Differences to minimum price

what we want

would be easy if we had that

Purchase

product	price	quantity	delta	minp
Apple	3	20	2	1
Apple	2	20	1	1
Banana	1	50	0	1
Banana	2	10	1	1
Banana	4	10	3	1

```
WITH  X as(
   SELECT  min(price) minp
   FROM  Purchase
)SELECT  P.*, price-minp delta
FROM  Purchase P, X
```

X

minp
1

?

Window functions allow us to calculate aggregates w/o reducing rows (Thus each of the original rows is returned)

Differences to minimum price

what we want

would be easy if we had that

Purchase

product	price	quantity	delta	minp
Apple	3	20	2	1
Apple	2	20	1	1
Banana	1	50	0	1
Banana	2	10	1	1
Banana	4	10	3	1

```
WITH X as(
  SELECT min(price) minp
  FROM Purchase
)SELECT P.*, price-minp delta
FROM Purchase P, X
```

X

minp
1

operate over a "window frame", i.e. a set of rows related to the current row

```
SELECT *, price - min(price) over() delta
FROM Purchase
```

Window functions allow us to calculate aggregates w/o reducing rows (Thus each of the original rows is returned)

"over()" indicates window function

Differences to minimum price

Purchase

product	price	quantity	delta	minp
Apple	3	20	1	2
Apple	2	20	0	2
Banana	1	50	0	1
Banana	2	10	1	1
Banana	4	10	3	1

?

Differences to minimum price

Purchase

product	price	quantity	delta	minp
Apple	3	20	1	2
Apple	2	20	0	2
Banana	1	50	0	1
Banana	2	10	1	1
Banana	4	10	3	1

?

X

product	minp
Apple	2
Banana	1

Differences to minimum price

Purchase

product	price	quantity	delta	minp
Apple	3	20	1	2
Apple	2	20	0	2
Banana	1	50	0	1
Banana	2	10	1	1
Banana	4	10	3	1

```
SELECT  product, min(price) minp
FROM  Purchase
GROUP BY  product
```

X

product	minp
Apple	2
Banana	1

Differences to minimum price

Purchase

product	price	quantity	delta	minp
Apple	3	20	1	2
Apple	2	20	0	2
Banana	1	50	0	1
Banana	2	10	1	1
Banana	4	10	3	1

```
WITH  X as(
   SELECT  product, min(price) minp
   FROM  Purchase
   GROUP BY  product
)SELECT P.*, price-minp delta
FROM  Purchase P, X
WHERE  P.product = X.product
```

X

product	minp
Apple	2
Banana	1

```
SELECT  *, price - min(price) over(partition by product) delta
FROM  Purchase
```

"partition by" corresponds to "group by"

Grouping vs Windows

Purchase

product	price
Apple	3
Apple	2
Banana	1
Banana	2
Banana	4

group by product

product	minp
Apple	1
Banana	2

over(partition by product)

product	price	minp
Apple	3	2
Apple	2	2
Banana	1	1
Banana	2	1
Banana	4	1

Window functions can gather both aggregate and non-aggregate values at once (so we don't need to collapse each group to one row)