SQL (Part 2)

February 5, 2019
Data Science CSCI 1951A
Brown University
Instructor: Ellie Pavlick
HTAs: Wennie Zhang, Maulik Dang, Gurnaaz Kaur

Follow up from last time

Follow up from last time

- Do Foreign Keys need to reference Primary Keys?

Follow up from last time

- Do Foreign Keys need to reference Primary Keys?
- NO!

Follow up from last time

- Do Foreign Keys need to reference Primary Keys?
- NO!
- But they do have to be unique. (More soon)

Follow up from last time

- Do Foreign Keys need to reference Primary Keys?
- NO!
- But they do have to be unique. (More soon)
- Also, NULLs are all considered distinct (i.e. NULL != NULL), so we'd want to have the FK reference a attribute that is not NULL too

Follow up from last time

- Do Foreign Keys need to reference Primary Keys?
- NO!
- But they do have to be unique. (More soon)
- Also, NULLs are all considered distinct (i.e. NULL != NULL), so we'd want to have the FK reference a attribute that is not NULL too
    - i.e. saying FK = NULL will not allow you to reference the other table

Follow up from last time

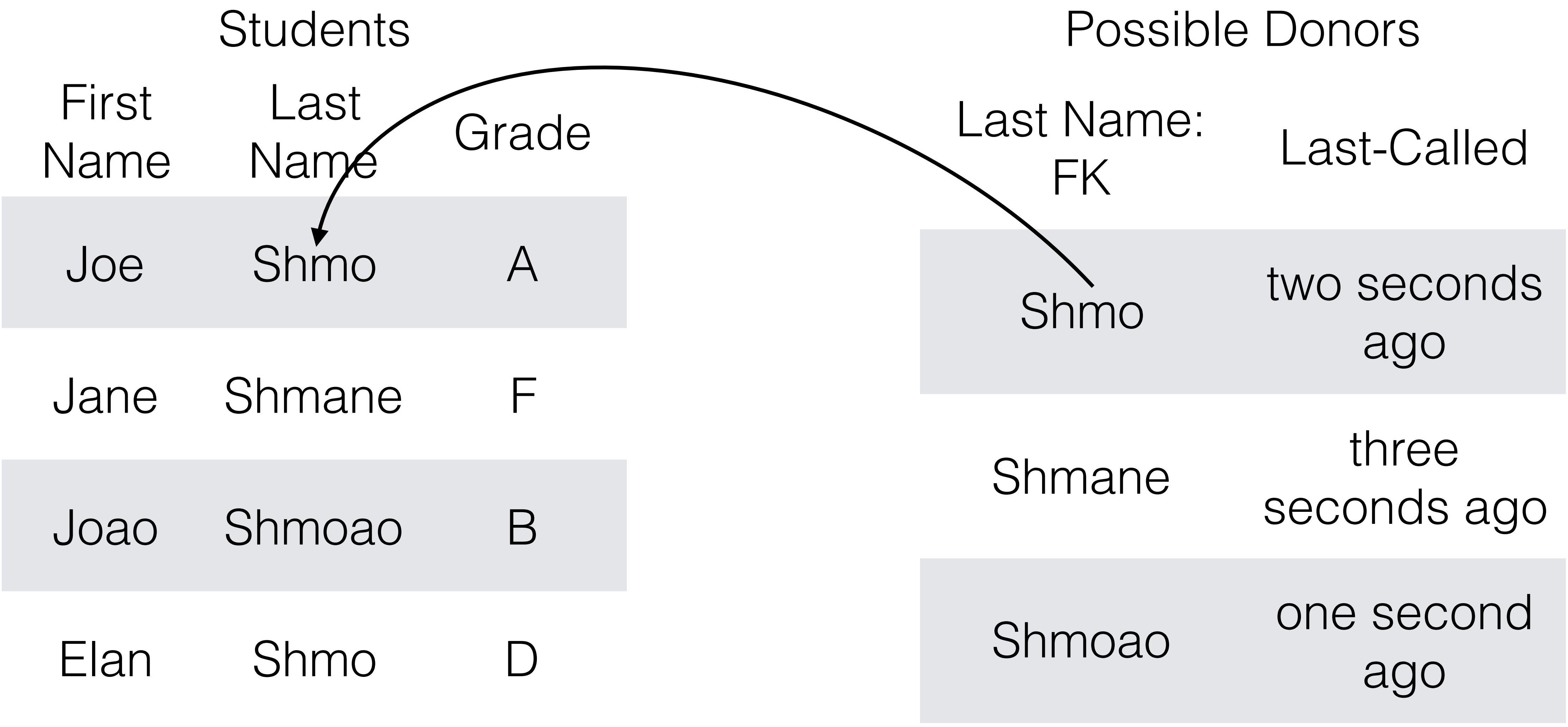
- Do Foreign Keys need to reference Primary Keys?
- NO!
- But they do have to be unique. (More soon)
- Also, NULLs are all considered distinct (i.e. NULL != NULL), so we'd want to have the FK reference a attribute that is not NULL too
    - i.e. saying FK = NULL will not allow you to reference the other table
- So! You should generally stick to the rule of making FK reference a PK

Follow up from last time

- Do Foreign Keys need to reference Primary Keys?
- NO!
- But they do have to be unique. (More soon)
- Also, NULLs are all considered distinct (i.e. NULL != NULL), so we'd want to have the FK reference a attribute that is not NULL too
    - i.e. saying FK = NULL will not allow you to reference the other table
- So! You should generally stick to the rule of making FK reference a PK
    - If you can't do this, try refactoring your DB to make it possible, if you are in a position to do this

Follow up from last time

Why do foreign keys have to be unique?



Students

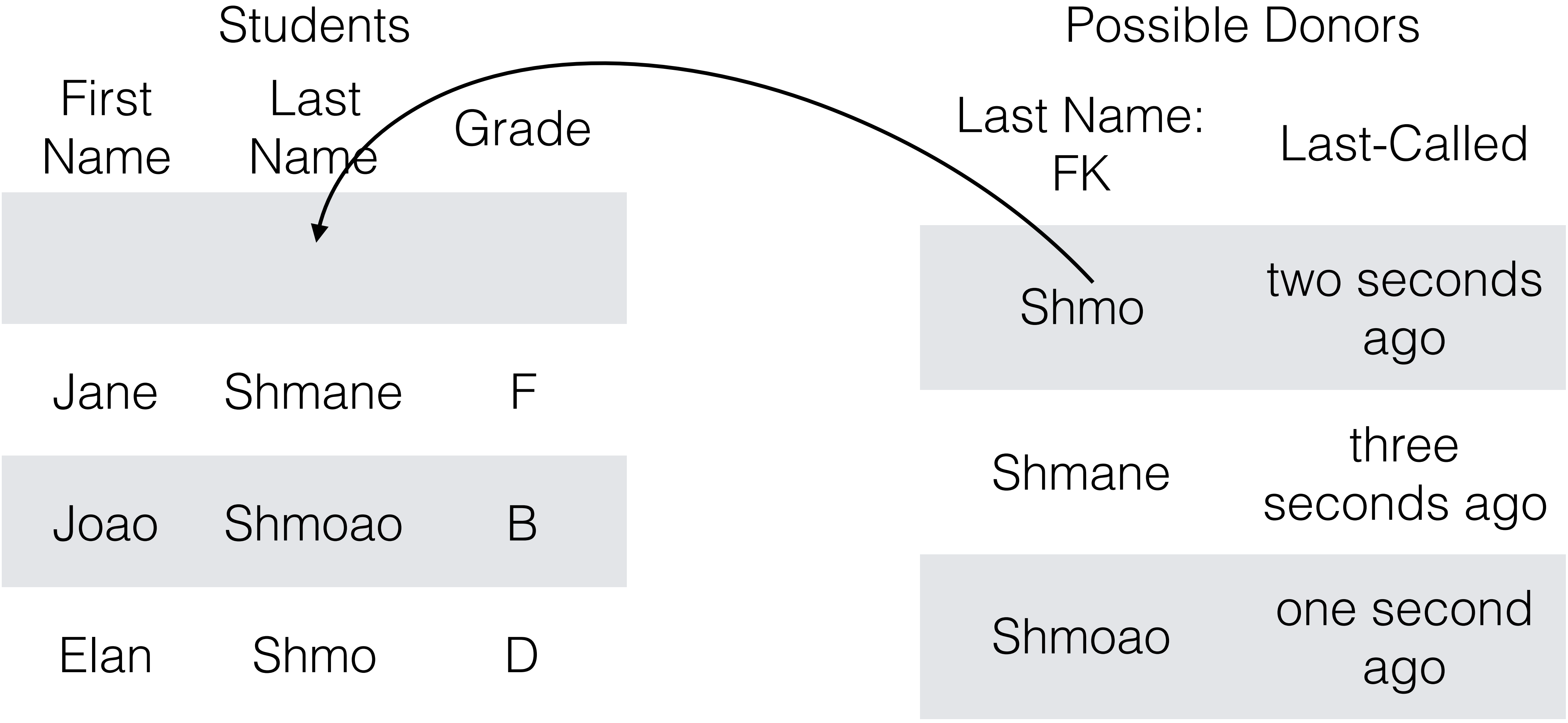
First Name	Last Name	Grade
Joe	Shmo	A
Jane	Shmane	F
Joao	Shmoao	B
Elan	Shmo	D

Possible Donors

Last Name: FK	Last-Called
Shmo	two seconds ago
Shmane	three seconds ago
Shmoao	one second ago

Follow up from last time

Why do foreign keys have to be unique?



Students

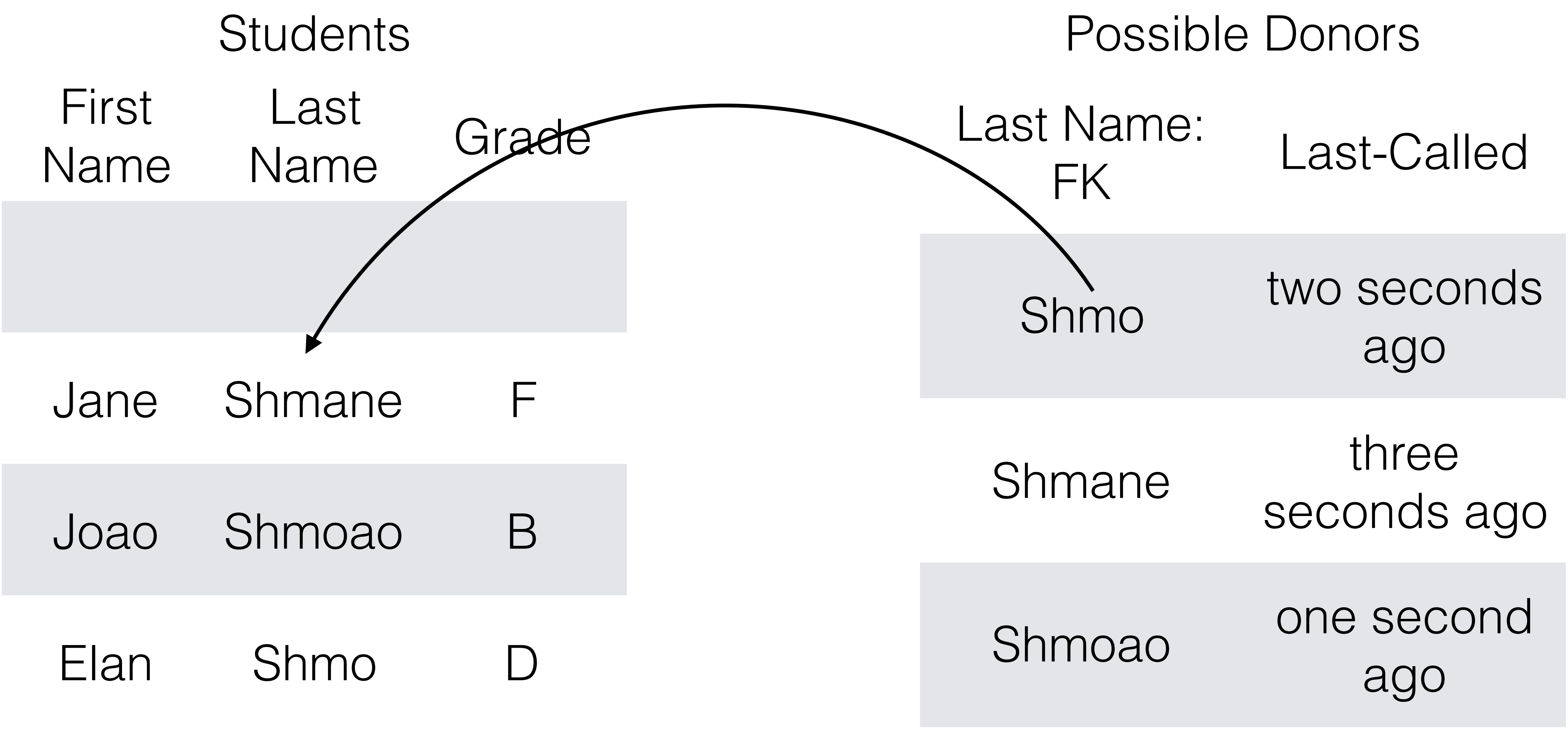
First Name	Last Name	Grade
Jane	Shmane	F
Joao	Shmoao	B
Elan	Shmo	D

Possible Donors

Last Name: FK	Last-Called
Shmo	two seconds ago
Shmane	three seconds ago
Shmoao	one second ago

Follow up from last time

Why do foreign keys have to be unique?



Students

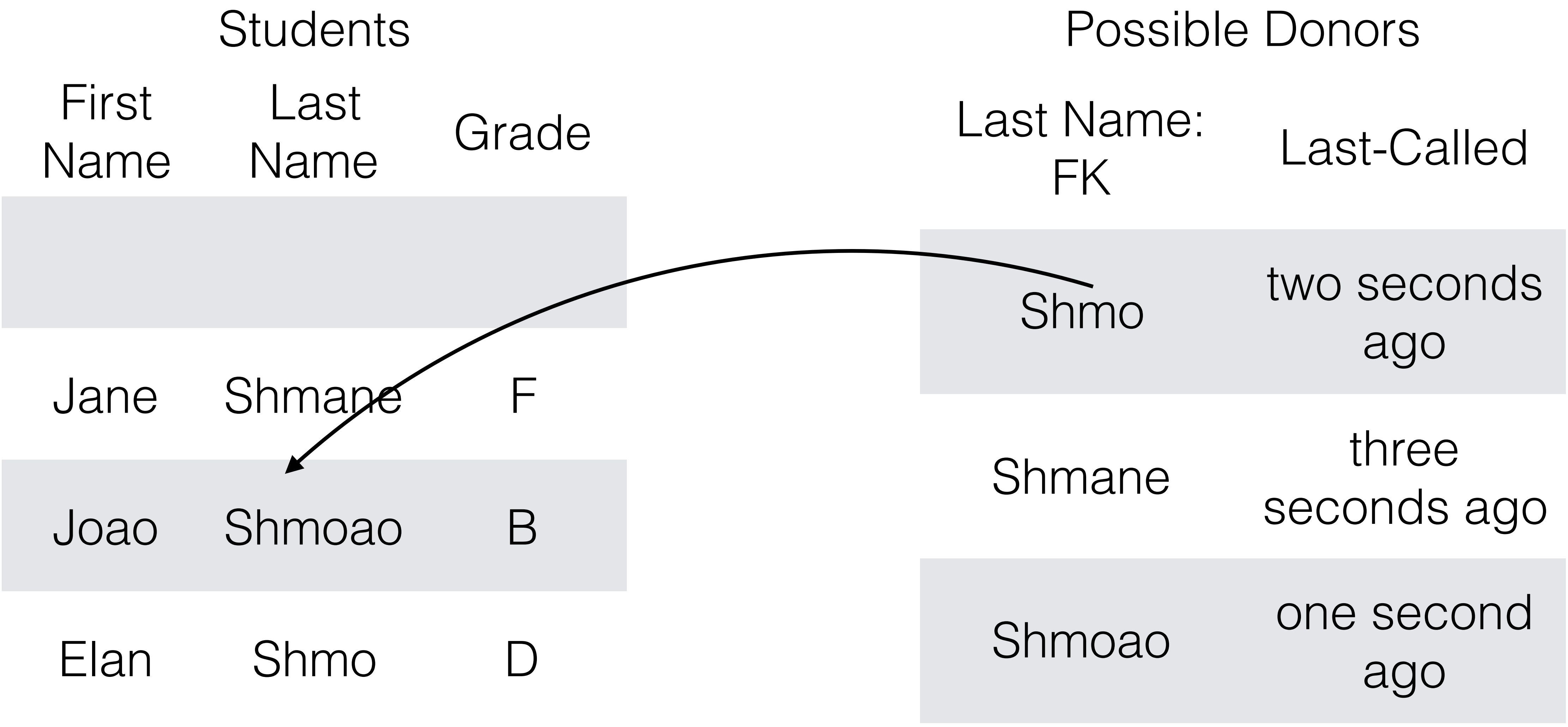
First Name	Last Name	Grade
Jane	Shmane	F
Joao	Shmoao	B
Elan	Shmo	D

Possible Donors

Last Name: FK	Last-Called
Shmo	two seconds ago
Shmane	three seconds ago
Shmoao	one second ago

Follow up from last time

Why do foreign keys have to be unique?



Students

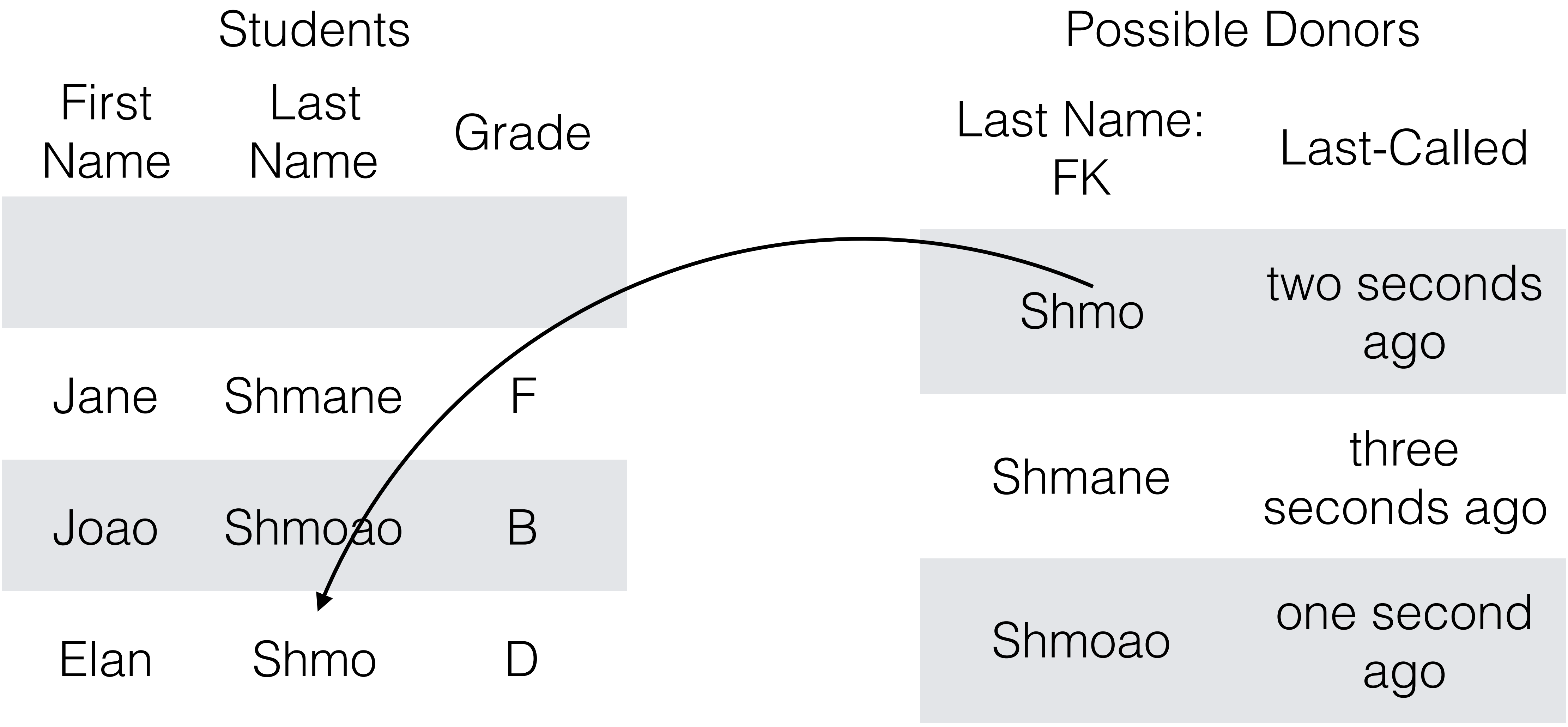
First Name	Last Name	Grade
Jane	Shmane	F
Joao	Shmoao	B
Elan	Shmo	D

Possible Donors

Last Name: FK	Last-Called
Shmo	two seconds ago
Shmane	three seconds ago
Shmoao	one second ago

Follow up from last time

Why do foreign keys have to be unique?



Students

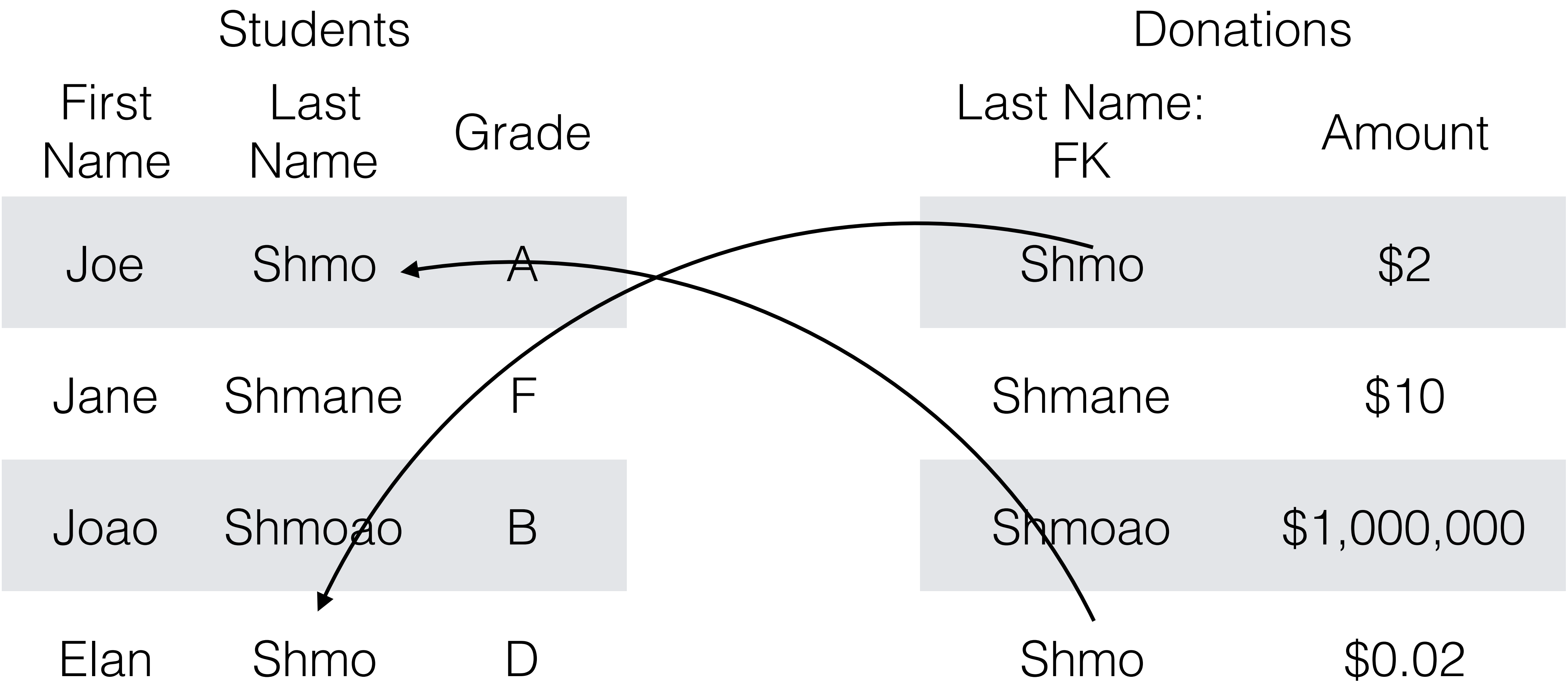
First Name	Last Name	Grade
Jane	Shmane	F
Joao	Shmoao	B
Elan	Shmo	D

Possible Donors

Last Name: FK	Last-Called
Shmo	two seconds ago
Shmane	three seconds ago
Shmoao	one second ago

Follow up from last time

Why do foreign keys have to be unique?



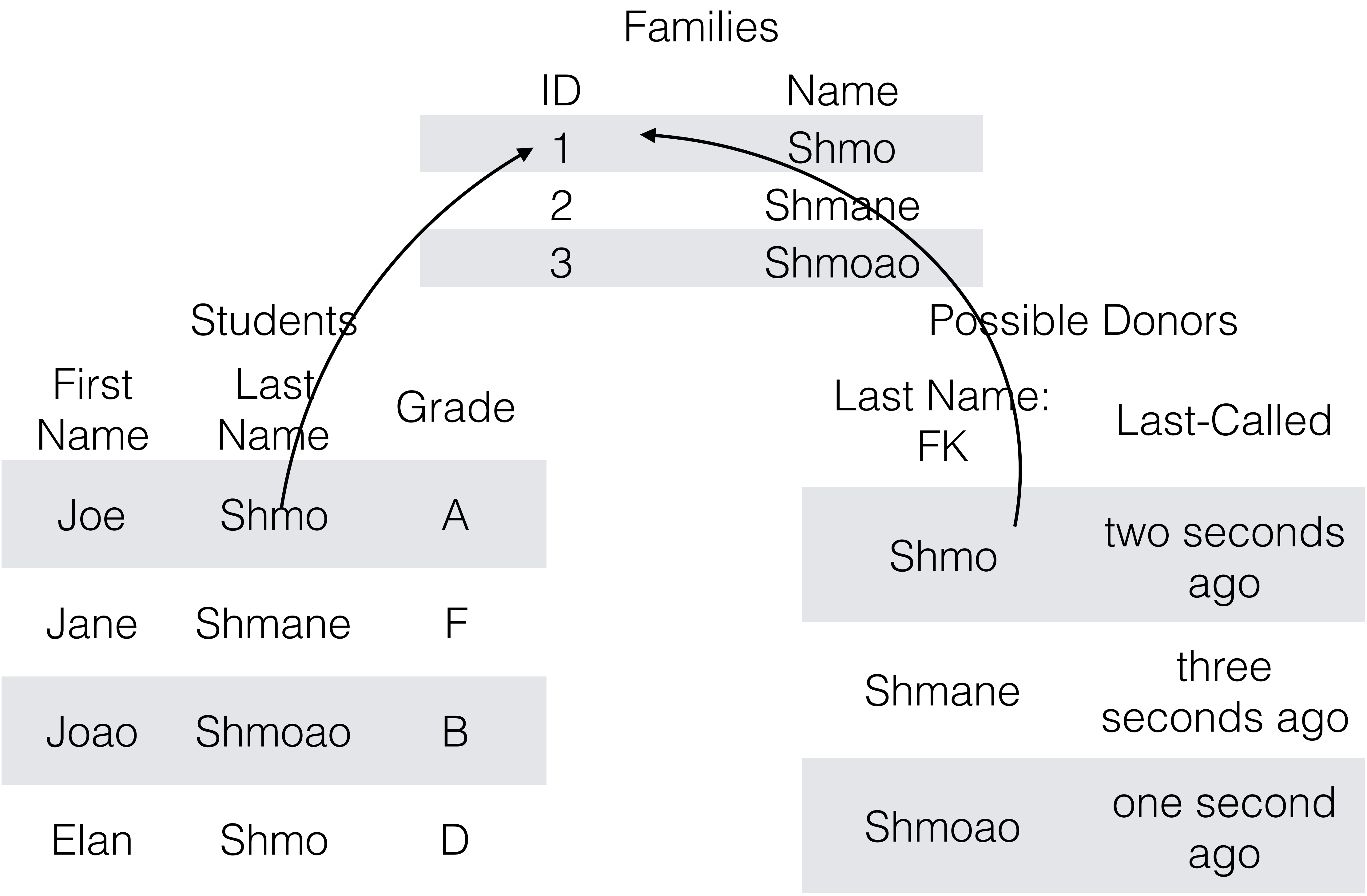
Students

First Name	Last Name	Grade
Joe	Shmo	A
Jane	Shmane	F
Joao	Shmoao	B
Elan	Shmo	D

Donations

Last Name: FK	Amount
Shmo	$2
Shmane	$10
Shmoao	$1,000,000
Shmo	$0.02

Follow up from last time



Families

ID	Name
1	Shmo
2	Shmane
3	Shmoao

Students

First Name	Last Name	Grade
Joe	Shmo	A
Jane	Shmane	F
Joao	Shmoao	B
Elan	Shmo	D

Possible Donors

Last Name: FK	Last-Called
Shmo	two seconds ago
Shmane	three seconds ago
Shmoao	one second ago

Follow up from last time

- Why would I ever use CHAR(n) as opposed to VARCHAR(n)? Are there any benefits?
- CHAR(n) is faster
  - Can use static memory allocation
  - No length checks in operations, so less overhead
- VARCHAR(n) uses less space on average

Announcements

- Have pen/paper or sit by someone who does—this will help for working through longer in-class exercises
- Please don't leave early! 3 minutes per day = one whole lecture! 😩
- Final projects: Start thinking about teams, watch Piazza, the HTAs are trying to help orchestrate

Outline

- Catchup up from last lecture (more SQL keywords)
- NULLs
- Execution Order, Optimization
- Nested Queries, More optimization
- ~~NoSQL~~ (no NoSQL = SQL??? 🤯)

Outline

- Catchup up from last lecture (more SQL keywords)
- NULLs
- Execution Order, Optimization
- Nested Queries, More optimization

ORDER BY

TWEET

ID	Time	Text
782138	2019-01-04 15:04:57	1951A 4 lyfe
389472	2019-01-01 12:34:56	hey
123794	2019-01-01 12:34:57	lol
127890	2019-01-04 17:30:07	hey
893110	2019-01-06 12:21:53	i <3 1951A
596208	2019-01-02 3:14:15	:-D
173902	2019-01-05 3:34:18	i <3 1951A

```
SELECT Text
FROM Tweet
ORDER BY Time
```

Text
hey
lol
:-D
1951A 4 lyfe
hey
i <3 1951A
i <3 1951A

ORDER BY

TWEET

ID	Time	Text
782138	2019-01-04 15:04:57	1951A 4 lyfe
389472	2019-01-01 12:34:56	hey
123794	2019-01-01 12:34:57	lol
127890	2019-01-04 17:30:07	hey
893110	2019-01-06 12:21:53	i <3 1951A
596208	2019-01-02 3:14:15	:-D
173902	2019-01-05 3:34:18	i <3 1951A

```
SELECT Text
FROM Tweet
ORDER BY ID
```

Text
lol
hey
i <3 1951A
hey
:-D
1951A 4 lyfe
i <3 1951A

GROUP BY

TWEET

ID	Likes	Text
782138	1,000	1951A 4 lyfe
389472	10	hey
123794	100	lol
127890	0	hey
893110	8,000,000	i <3 1951A
596208	1	:-D
173902	1,000,000,000	i <3 1951A

```
SELECT Text,
Count(*), AVG(Likes)
FROM Tweet
GROUP BY Text
```

Text	Count(*)	AVG(Likes)
lol	1	100
hey	2	5
i <3 1951A	2	504,000,000
:-D	1	1
1951A 4 lyfe	1	1,000

GROUP BY

TWEET

ID	Likes	Text
782138	1,000	1951A 4 lyfe
389472	10	hey
123794	100	lol
127890	0	hey
893110	8,000,000	i <3 1951A
596208	1	:-D
173902	1,000,000,000	i <3 1951A

```
SELECT Text,
Count(*), AVG(Likes)
FROM Tweet
GROUP BY Text
```

Text	Count(*)	AVG(Likes)
lol	1	100
hey	2	5
i <3 1951A	2	504,000,000
:-D	1	1
1951A 4 lyfe	1	1,000

SUM, MIN, MAX,
COUNT, AVG

HAVING

TWEET

ID	Likes	Text
782138	1,000	1951A 4 lyfe
389472	10	hey
123794	100	lol
127890	0	hey
893110	8,000,000	i <3 1951A
596208	1	:-D
173902	1,000,000,000	i <3 1951A

```
SELECT Text,
Count(*), AVG(Likes)
FROM Tweet
GROUP BY Text
HAVING COUNT(*) > 1
```

Text	Count(*)	AVG(Likes)
hey	2	5
i <3 1951A	2	504,000,000

SUM, MIN, MAX,
COUNT, AVG

LIKE

TWEET

ID	Likes	Text
782138	1,000	1951A 4 lyfe
389472	10	hey
123794	100	lol
127890	0	hey
893110	8,000,000	i <3 1951A
596208	1	:-D
173902	1,000,000,000	i <3 1951A

```
SELECT Text, Count(*),
AVG(Likes)
FROM Tweet
WHERE Text LIKE ‘%1951A%’
GROUP BY Text
```

Text	Count(*)	AVG(Likes)
1951A 4 lyfe	1	1,000
i <3 1951A	2	504,000,000

IN

STUDENT

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Jens
5	Erin

GRADES

Student	Course	Grade
1	32	A
2	1951A	A
6	32	A

```
SELECT Name
FROM STUDENT
WHERE ID IN
   (SELECT Student
    FROM GRADES
    WHERE Course = 1951A
    )
```

Find names of students in 1951A

IN

"Subquery"
(More later, get excited)

STUDENT

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Jens
5	Erin

GRADES

Student	Course	Grade
1	32	A
2	1951A	A
6	32	A

```
SELECT Name
FROM STUDENT
WHERE ID IN
  (SELECT Student
   FROM GRADES
   WHERE Course = 1951A
   )
```

Find names of students in 1951A

IN

STUDENT

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Jens
5	Erin

GRADES

Student	Course	Grade
1	32	A
2	1951A	A
6	32	A

Returns "bag" of student IDs

```
SELECT Name
FROM STUDENT
WHERE ID IN
  (SELECT Student
   FROM GRADES
   WHERE Course = 1951A
   )
```

Find names of students in 1951A

IN

STUDENT

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Jens
5	Erin

GRADES

Student	Course	Grade
1	32	A
2	1951A	A
6	32	A

Returns True if ID is in that bag

```
SELECT Name
FROM STUDENT
WHERE ID IN
   (SELECT Student
    FROM GRADES
    WHERE Course = 1951A
    )
```

Find names of students in 1951A

ALL/ANY

STUDENT

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Jens
5	Erin

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

```
SELECT Grade
FROM GRADES
WHERE Course = “1951A”
   AND Grade >= ALL
  (SELECT Grade
   FROM GRADES
   WHERE Course = 1951A
   )
```

What is the highest grade in 1951A?

ALL/ANY

STUDENT

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Jens
5	Erin

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

Returns True if condition holds for all tuples in bag

```
SELECT Grade
FROM GRADES
WHERE Course = “1951A”
   AND Grade >= ALL
  (SELECT Grade
   FROM GRADES
   WHERE Course = 1951A
   )
```

What is the highest grade in 1951A?

ALL/ANY

STUDENT

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Jens
5	Erin

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

```
SELECT Grade
FROM GRADES
WHERE Course = “1951A”
   AND Grade > ANY
  (SELECT Grade
   FROM GRADES
   WHERE Course = 1951A
   )
```

???

ALL/ANY

STUDENT

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Jens
5	Erin

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

```
SELECT Grade
FROM GRADES
WHERE Course = “1951A”
   AND Grade > ANY
  (SELECT Grade
   FROM GRADES
   WHERE Course = 1951A
   )
```

Return all grades
except the lowest one.

ALL/ANY

STUDENT

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Jens
5	Erin

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

```
SELECT Grade
FROM GRADES
WHERE Course = “1951A”
   AND Grade > NOT ANY
  (SELECT Grade
   FROM GRADES
   WHERE Course = 1951A
   )
```

Return the lowest grade.

ALL/ANY

STUDENT

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Jens
5	Erin

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

```
SELECT Grade
FROM GRADES
WHERE Course = “1951A”
   AND Grade >= ALL
  (SELECT Grade
   FROM GRADES
   WHERE Course = 1951A
   )
```

Grade
…

ALL/ANY

STUDENT

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Jens
5	Erin

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

```
SELECT Grade
FROM GRADES
WHERE Course = “1951A”
   AND Grade >= ALL
  (SELECT Grade
   FROM GRADES
   WHERE Course = 1951A
   )
```

Grade
3.5
3.5

DISTINCT

STUDENT

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Jens
5	Erin

```
SELECT DISTINCT Grade
FROM GRADES
WHERE Course = “1951A”
   AND Grade >= ALL
  (SELECT Grade
   FROM GRADES
   WHERE Course = 1951A
   )
```

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

Grade
3.5

DISTINCT

STUDENT

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Jens
5	Erin

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

```
SELECT DISTINCT Grade
FROM GRADES
WHERE Course = “1951A”
   AND Grade >= ALL
  (SELECT Grade
   FROM GRADES
   WHERE Course = 1951A
   )
```

Grade
3.5

Set operations (Union, Intersection, etc.) remove duplicates by default.

EXISTS

STUDENT

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Jens
5	Erin

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

```
SELECT NAME
FROM STUDENT s
WHERE NOT EXISTS
   (SELECT *
    FROM GRADES
    WHERE Course = 1951A
    AND Student = s.ID
    )
```

???

EXISTS

STUDENT

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Jens
5	Erin

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

True as long as bag is not empty

```
SELECT NAME
FROM STUDENT s
WHERE NOT EXISTS
   (SELECT *
    FROM GRADES
    WHERE Course = 1951A
    AND Student = s.ID
    )
```

???

EXISTS

STUDENT

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Jens
5	Erin

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

```
SELECT NAME
FROM STUDENT s
WHERE NOT EXISTS
   (SELECT *
    FROM GRADES
    WHERE Course = 1951A
    AND Student = s.ID
    )
```

Students who are
not in 1951A

Outline

- Catchup up from last lecture (more SQL keywords)
- NULLs
- Execution Order, Optimization
- Nested Queries, More optimization
- ~~NoSQL~~ (no NoSQL = SQL??? ::mindblown::)

NULL!

- Black hole! NULL is NULL is NULL and there is no coming back from it…
- If an operand is NULL, the result is NULL:
    - NULL + 1 = NULL
    - NULL * 0 = NULL
- Comparisons: All comparisons that involve a null value, evaluate to unknown
    - NULL = NULL -> Unknown
    - NULL < 13 -> Unknown
    - NULL > NULL -> Unknown

NULL!

p	q	p OR q	p AND q	p = q
TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	TRUE	FALSE	FALSE
FALSE	TRUE	TRUE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE

NULL!

p	q	p OR q	p AND q	p = q
TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	TRUE	FALSE	FALSE
FALSE	TRUE	TRUE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE
TRUE	UNK	TRUE	UNK	UNK
FALSE	UNK	UNK	FALSE	UNK
UNK	TRUE	TRUE	UNK	UNK
UNK	FALSE	UNK	FALSE	UNK
UNK	UNK	UNK	UNK	UNK

NULL!

WHERE: Only tuples which evaluate to true are part of the query result. (I.e. unknown and false treated equivalently.)

TWEET

ID	Text	Likes
389472	NULL	100
123794	NULL	3
596208	:-D	NULL
782138	1951A 4 lyfe	NULL
173902	i <3 1951A	19
893110	i <3 1951A	7539

```
SELECT COUNT(*)
FROM TWEET
WHERE Likes != 10
```

↓

Count(*)
4

NULL!

GROUP BY: If NULL exists, then there is a group for NULL.

TWEET

ID	Text	Likes
389472	NULL	100
123794	NULL	3
596208	:-D	NULL
782138	1951A 4 lyfe	NULL
173902	i <3 1951A	19
893110	i <3 1951A	7539

```
SELECT Text, COUNT(*)
FROM TWEET
GROUP BY Text
```

↓

Text	Count(*)
NULL	2
:-D	1
1951A 4 lyfe	1
i <3 1951A	2

NULL!

For predicates with NULL, use IS (e.g. not “=“)

TWEET

ID	Text	Likes
389472	NULL	100
123794	NULL	3
596208	:-D	NULL
782138	1951A 4 lyfe	NULL
173902	i <3 1951A	19
893110	i <3 1951A	7539

```
SELECT Text ID
FROM TWEET
WHERE Text = NULL
```

↓

ID

NULL!

For predicates with NULL, use IS (e.g. not “=“)

TWEET

ID	Text	Likes
389472	NULL	100
123794	NULL	3
596208	:-D	NULL
782138	1951A 4 lyfe	NULL
173902	i <3 1951A	19
893110	i <3 1951A	7539

```
SELECT Text ID
FROM TWEET
WHERE Text IS NULL
```

↓

ID
389472
123794

NULL!

- `count(att)`: NULL is ignored
- `sum(att)`: NULL is ignored
- `avg(att)`: results from SUM and COUNT
- `min(att)` and `max(att)`: NULL is ignored
- Exception! If NULL is the only value in the column, then `sum/avg/min/max` all return “NULL”

Clicker Question!

```
SELECT COUNT(*)
FROM TWEET
```

→

Count(*)
100

```
SELECT COUNT(*)
FROM TWEET
WHERE Text = “:)”
```

→

Count(*)
15

What will be the result of this query? →

```
SELECT COUNT(*)
FROM TWEET
WHERE Text != “:)”
```

(a)

Count(*)
100

(b)

Count(*)
85

(c)

I…don’t…know…

Clicker Question!

```
SELECT COUNT(*)
FROM TWEET
```

→

Count(*)
100

```
SELECT COUNT(*)
FROM TWEET
WHERE Text = “:)”
```

→

Count(*)
15

Can't say how many are NULL

What will be the result of this query? →

```
SELECT COUNT(*)
FROM TWEET
WHERE Text != “:)”
```

(a)

Count(*)
100

(b)

Count(*)
85

[✓ (c)]

I...don't...know...

Clicker Question!

RUNNERS

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Haomo

RACES

Event_ID	Event	Winner_ID
1	Wennie	2
2	Maulik	3
3	Gurnaaz	2
4	Haomo	NULL

What will be the result of the below query?

```
SELECT COUNT(*)
FROM RUNNERS
WHERE ID NOT IN SELECT(Winner_ID FROM RACES)
```

(a)

Count(*)
0

(b)

Count(*)
1

(c)

Count(*)
2

Clicker Question!

RUNNERS

ID	Name
1	Wennie
2	Maulik
3	Gurnaaz
4	Haomo

RACES

Event_ID	Event	Winner_ID
1	Wennie	2
2	Maulik	3
3	Gurnaaz	2
4	Haomo	NULL

What will be the result of the below query?

```
SELECT COUNT(*)
FROM RUNNERS
WHERE ID NOT IN SELECT(Winner_ID FROM RACES)
```

ID NOT IN (2,3,NULL) is the same as ID!=2 AND ID!=3 and ID!=NULL

[✓ (a)]

Count(*)
0

(b)

Count(*)
1

(c)

Count(*)
2

Outline

- Catchup up from last lecture (more SQL keywords)
- NULLs
- Execution Order, Optimization
- Nested Queries, More optimization
- ~~NoSQL~~ (no NoSQL = SQL??? ::mindblown::)

Relational Algebra Recap

- $\sigma_{<condition>}(S)$: select, return a relation containing just the tuples in S that meet condition

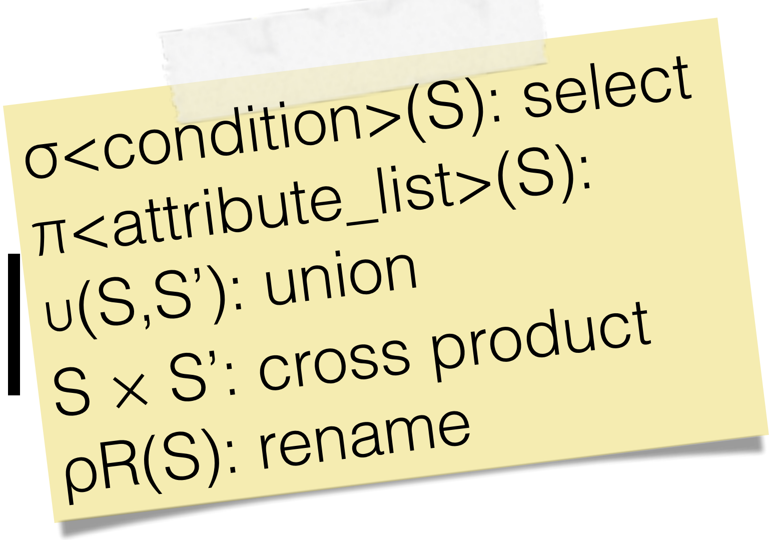
- $\pi_{<attribute\_list>}(S)$: project, return a relation S’ containing the following: for each tuple t in S there is a tuple t’ in S’ that contains the attributes of t that are in attribute list

- $\cup(S,S')$: union, typical set-theoretic definitions (same for intersection, minus)

- $S \times S'$: cross product, return a new relation S’’ such that, for every t in S and t’ in S’, (t, t’) is in S’’.

- $\rho_R(S)$: rename the relation S as to R

SQL -> Relational Al



σ<condition>(S): select
π<attribute_list>(S):
∪(S,S'): union
S × S': cross product
ρR(S): rename

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

↓

ID	Text
389472	hey
123794	lol
596208	:-D
782138	1951A 4 lyfe
173902	i <3 1951A
893110	i <3 1951A

SQL

```
SELECT ID, Text
FROM TWEET
```

Relational Algebra

```
???
```

SQL -> Relational Al

σ<condition>(S): select
π<attribute_list>(S):
∪(S,S'): union
S × S': cross product
ρR(S): rename

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

↓

ID	Text
389472	hey
123794	lol
596208	:-D
782138	1951A 4 lyfe
173902	i <3 1951A
893110	i <3 1951A

SQL

```
SELECT ID, Text
FROM TWEET
```

Relational Algebra

$\pi_{<ID,Text>}(TWEET)$

SQL -> Relational Al

σ<condition>(S): select
π<attribute_list>(S):
∪(S,S’): union
S × S’: cross product
ρR(S): rename

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

↓

ID	Text
389472	hey

SQL

```
SELECT ID, Text
FROM TWEET
WHERE Text = “hey”
```

Relational Algebra

```
???
```

SQL -> Relational Al

σ<condition>(S): select
π<attribute_list>(S):
∪(S,S'): union
S × S': cross product
ρR(S): rename

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

↓

ID	Text
389472	hey

SQL

```
SELECT ID, Text
FROM TWEET
WHERE Text = "hey"
```

Relational Algebra

$$\pi_{<ID,Text>}(\sigma_{Text=\text{"hey"}}(TWEET))$$

Clicker Question!

Do these queries return the same relation?

(a) Yep (b) Nah

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

```
SELECT Text FROM TWEET
```

$\Pi_{\text{<Text>}}(\text{TWEET})$

Clicker Question!

Do these queries return the same relation?

(a) Yep [✓ (b) Nah]

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

```
SELECT Text FROM TWEET
```

$\Pi_{<\text{Text}>}(\text{TWEET})$

Clicker Question!

Do these queries return the same relation?

[✓ (a) Yep] (b) Nah

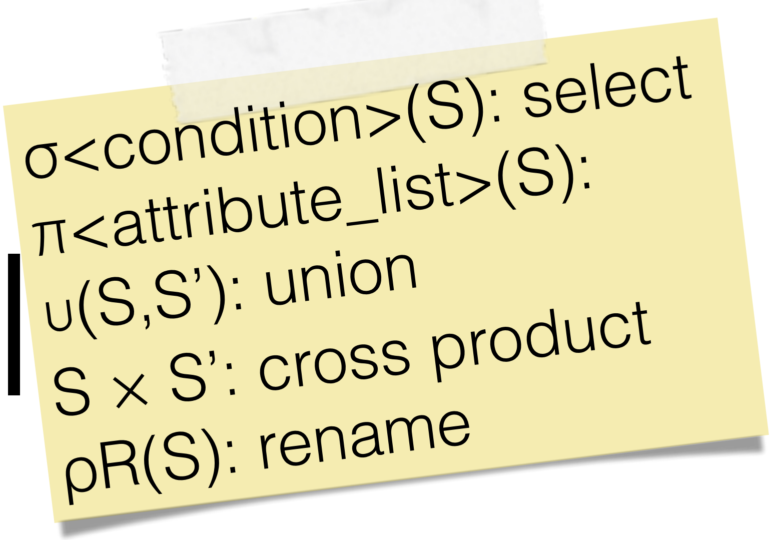
TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

```
SELECT DISTINCT
Text FROM TWEET
```

$\Pi_{<\mathrm{Text}>}(\mathrm{TWEET})$

SQL -> Relational Al



σ<condition>(S): select
π<attribute_list>(S):
∪(S,S’): union
S × S’: cross product
ρR(S): rename

PERSON

Handle	Name
m	Maulik
w	Wennie
g	Gurnaaz

RETWEET

Person	Tweet
m	1
m	2
w	1

SQL

```
SELECT Name
FROM PERSON, RETWEET
WHERE PERSON.Handle =
     RETWEET.Person
```

Relational Algebra

???

SQL -> Relational Al

σ<condition>(S): select
π<attribute_list>(S):
$\cup$(S,S'): union
S × S': cross product
ρR(S): rename

PERSON

Handle	Name
m	Maulik
w	Wennie
g	Gurnaaz

RETWEET

Person	Tweet
m	1
m	2
w	1

SQL

```
SELECT Name
FROM PERSON, RETWEET
WHERE PERSON.Handle =
      RETWEET.Person
```

Relational Algebra

$$\pi_{<\mathrm{Name}>}(\sigma_{\mathrm{PERSON.Handle} = \mathrm{RETWEET.Person}}(\mathrm{PERSON} \times \mathrm{RETWEET}))$$

SQL -> Relational Al

σ<condition>(S): select
π<attribute_list>(S):
∪(S,S'): union
S × S': cross product
ρR(S): rename

PERSON

Handle	Name
m	Maulik
w	Wennie
g	Gurnaaz

RETWEET

Person	Tweet
m	1
m	2
w	1

SQL

```
SELECT Name
FROM PERSON AS p,
     RETWEET AS r
WHERE r.Person = p.Handle
```

Relational Algebra

```
???
```

SQL -> Relational Al

σ<condition>(S): select
π<attribute_list>(S):
∪(S,S’): union
S × S’: cross product
ρR(S): rename

PERSON

Handle	Name
m	Maulik
w	Wennie
g	Gurnaaz

RETWEET

Person	Tweet
m	1
m	2
w	1

SQL

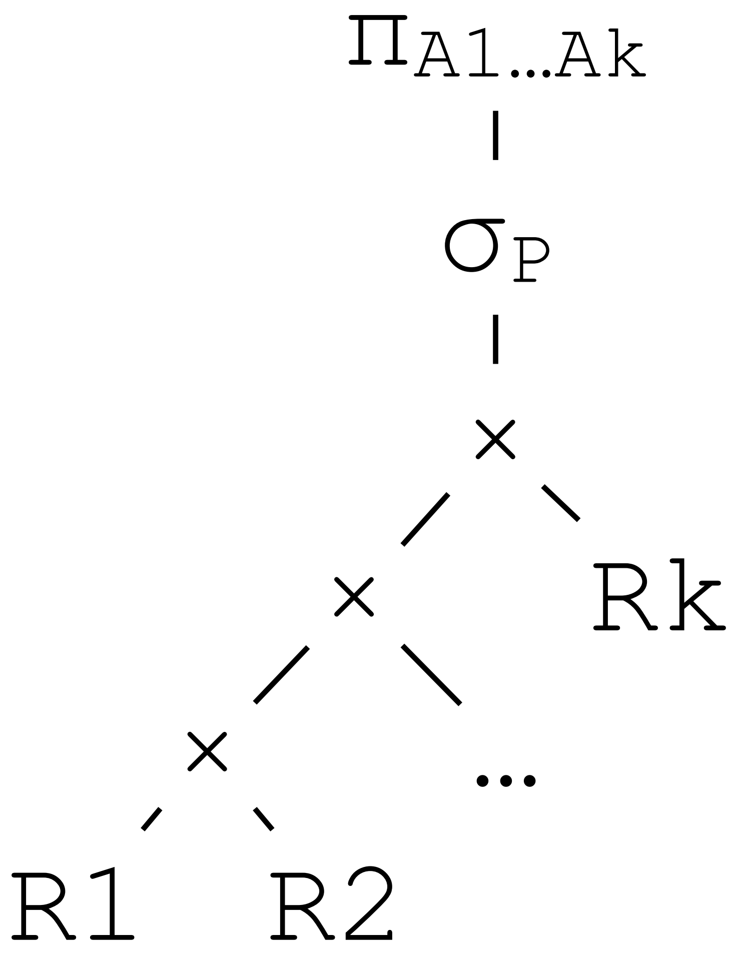
```
SELECT Name
FROM PERSON AS p,
     RETWEET AS r
WHERE r.Person = p.Handle
```

Relational Algebra

$$\pi_{\texttt{Name}}(\sigma_{\texttt{p.Handle = r.Person}}(\rho_{\texttt{p}}(\texttt{PERSON}) \times \rho_{\texttt{r}}(\texttt{RETWEET})))$$

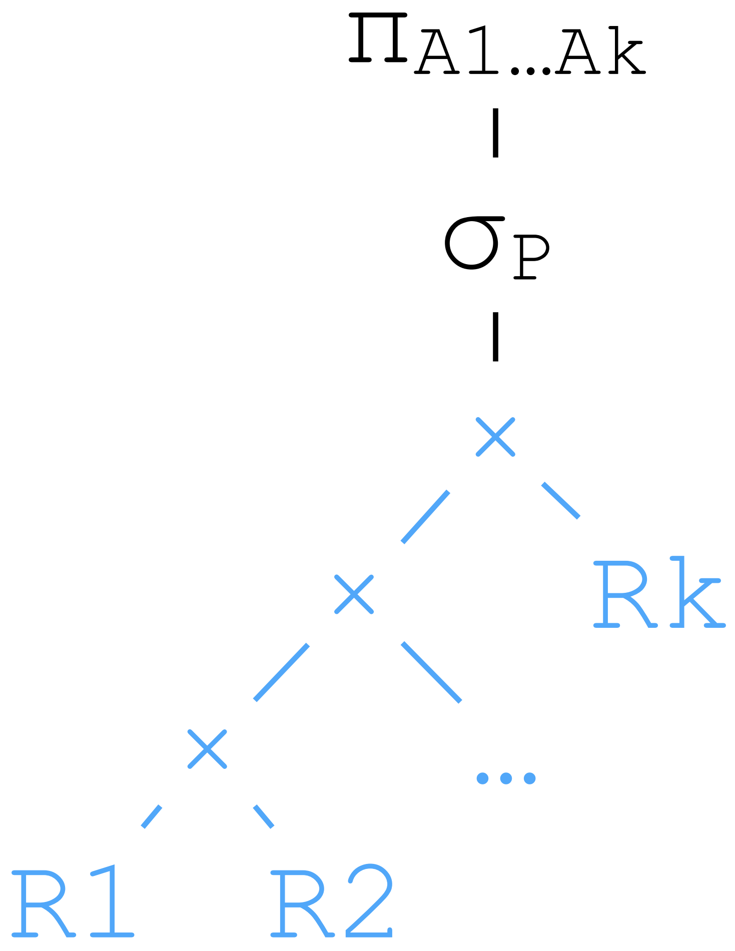
Execution Order

```
SELECT A1…An
FROM R1…Rk
WHERE P
```



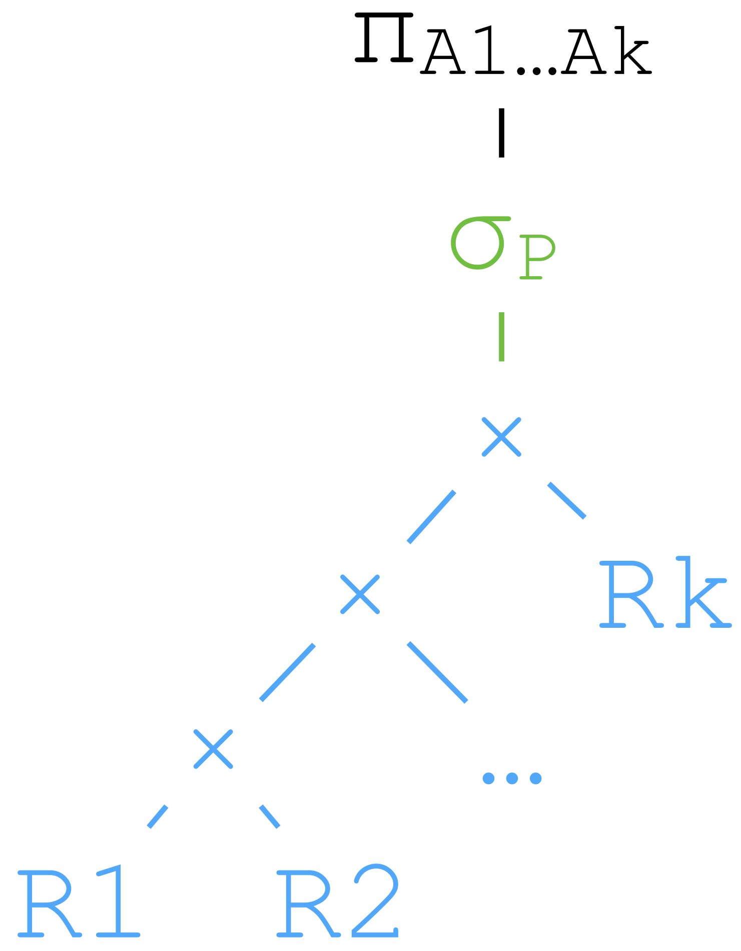
Execution Order

```
SELECT A1…An
FROM R1…Rk
WHERE P
```



Execution Order

```
SELECT A1…An
FROM R1…Rk
WHERE P
```



$\pi_{A1…Ak}$

σ_P

×

×

Rk

×

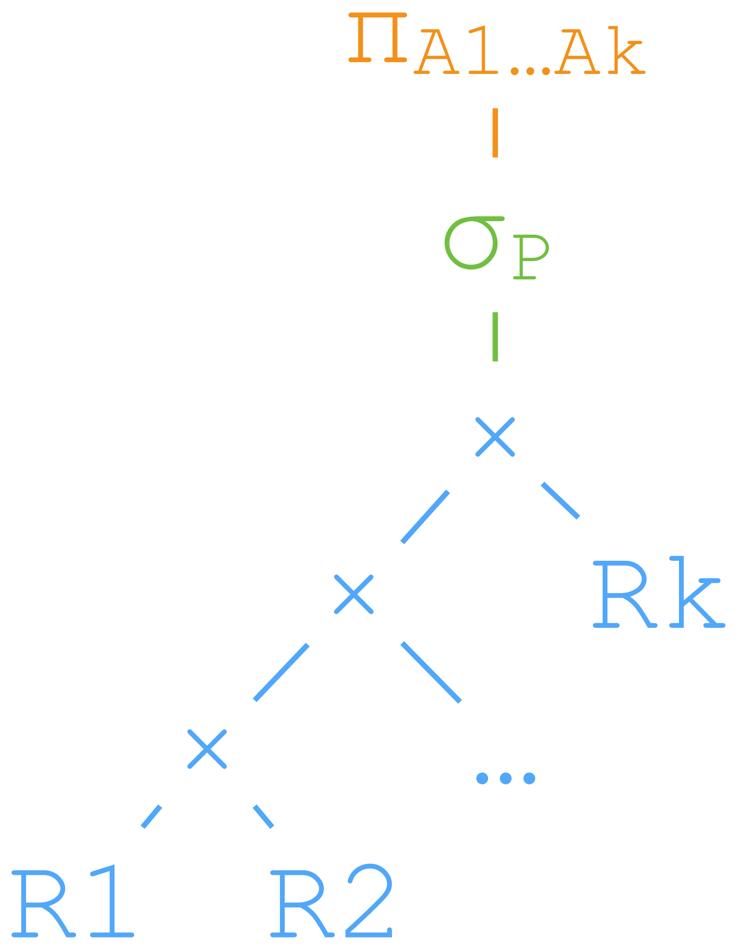
…

R1

R2

Execution Order

```
SELECT A1…An
FROM R1…Rk
WHERE P
```



SQL -> Relational Algebra

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

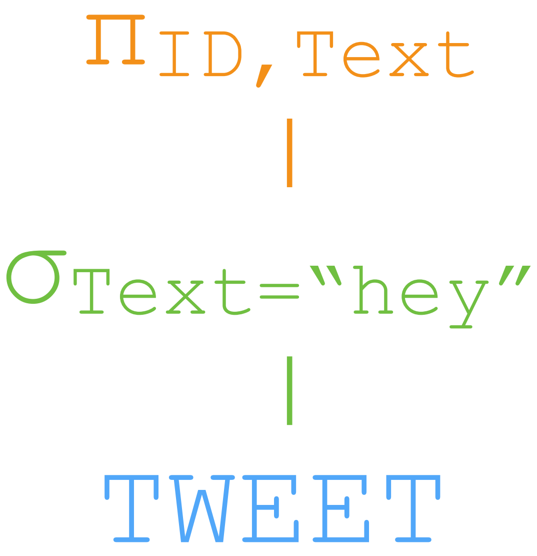
↓

ID	Text
389472	hey

SQL

```
SELECT ID, Text
FROM TWEET
WHERE Text = “hey”
```

Relational Algebra



$\pi_{\texttt{ID,Text}}$

$\sigma_{\texttt{Text=“hey”}}$

TWEET

SQL -> Relational Algebra

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

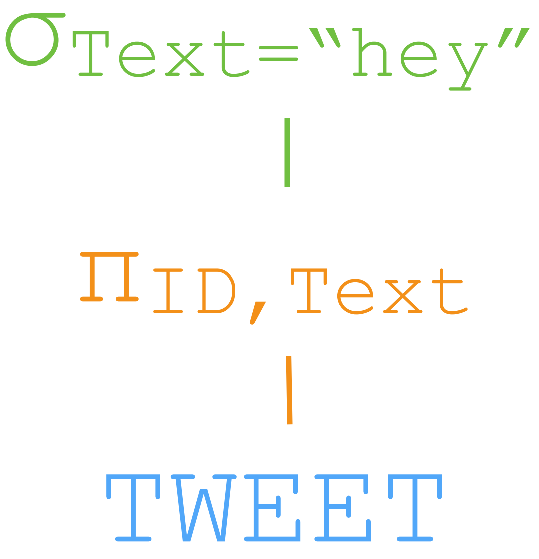
ID	Text
389472	hey

A query can have multiple "equivalent" trees

SQL

```
SELECT ID, Text
FROM TWEET
WHERE Text = "hey"
```

Relational Algebra



$\sigma_{Text=\text{"hey"}}$

$\pi_{ID,Text}$

TWEET

Clicker Question!

Which is better?

(a) $\sigma_{\texttt{<condition>}}(\pi_{\texttt{<attr\_list>}}(\texttt{R}))$

(b) $\pi_{\texttt{<attr\_list>}}(\sigma_{\texttt{<condition>}}(\texttt{R}))$

Clicker Question!

Which is better?

(a) $\sigma_{<condition>}(\Pi_{<attr\_list>}(R))$

[✓ (b)] $\Pi_{<attr\_list>}(\sigma_{<condition>}(R))$

σ<condition>(S): select
π<attribute_list>(S):
∪(S,S’): union
S × S’: cross product
ρR(S): rename

SQL -> Relational Al

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

SQL

```
SELECT ID, Text
FROM TWEET
WHERE Text = “hey”
```

Relational Algebra

$$\sigma_{\text{Text=“hey”}}(\pi_{\text{<ID,Text>}}(\text{TWEET}))$$

$$\pi_{\text{<ID,Text>}}(\sigma_{\text{Text=“hey”}}(\text{TWEET}))$$

SQL -> Relational Al

σ<condition>(S): select
π<attribute_list>(S):
∪(S,S'): union
S × S': cross product
ρR(S): rename

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

SQL

```
SELECT ID, Time
FROM TWEET
WHERE Text = “hey”
```

Relational Algebra

$$\sigma_{\texttt{Text=“hey”}}(\pi_{\texttt{<ID,Time>}}(\texttt{TWEET}))$$

$$\pi_{\texttt{<ID,Time>}}(\sigma_{\texttt{Text=“hey”}}(\texttt{TWEET}))$$

SQL -> Relational Al

σ<condition>(S): select

π<attribute_list>(S):

∪(S,S’): union

S × S’: cross product

ρR(S): rename

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

SQL

```
SELECT ID, Time
FROM TWEET
WHERE Text = “hey”
```

Relational Algebra

$\sigma_{\text{Text=“hey”}}(\pi_{\text{<ID,Time>}}(\text{TWEET}))$

$\pi_{\text{<ID,Time>}}(\sigma_{\text{Text=“hey”}}(\text{TWEET}))$

SQL -> Relational Al

σ<condition>(S): select
π<attribute_list>(S):
∪(S,S’): union
S × S’: cross product
ρR(S): rename

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

↓

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

SQL

```
SELECT ID, Time
FROM TWEET
WHERE Text = “hey”
```

Relational Algebra

$$\sigma_{Text=\text{“hey”}}(\pi_{<ID,Time>}(TWEET))$$

$$\pi_{<ID,Time>}(\sigma_{Text=\text{“hey”}}(TWEET))$$

SQL -> Relational Al

σ<condition>(S): select
π<attribute_list>(S):
∪(S,S'): union
S × S': cross product
ρR(S): rename

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

↓

ID	Time
389472	12:34:56
123794	12:34:57
596208	3:14:15
782138	15:04:57
173902	3:34:18
893110	12:21:53

SQL

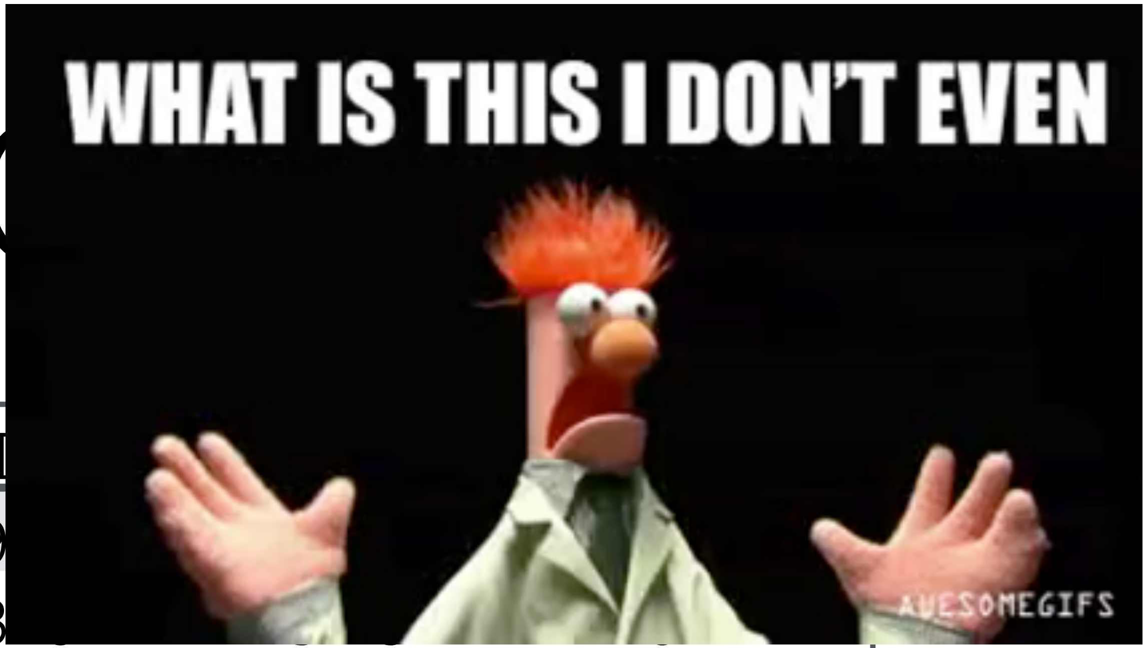
```
SELECT ID, Time
FROM TWEET
WHERE Text = “hey”
```

Relational Algebra

$\sigma_{Text=“hey”}(\pi_{<ID,Time>}(TWEET))$

$\pi_{<ID,Time>}(\sigma_{Text=“hey”}(TWEET))$

S nal Al



σ<condition>(S): select
π<attribute_list>(S):
∪(S,S'): union
S × S': cross product
ρR(S): rename

I		
389		
123		
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

↓

ID	Time
389472	12:34:56
123794	12:34:57
596208	3:14:15
782138	15:04:57
173902	3:34:18
893110	12:21:53

SQL

```
SELECT ID, Time
FROM TWEET
WHERE Text = "hey"
```

Relational Algebra

$\sigma_{Text=\text{“hey”}}(\Pi_{<ID,Time>}(TWEET))$

$\Pi_{<ID,Time>}(\sigma_{Text=\text{“hey”}}(TWEET))$

SQL -> Relational Al

σ<condition>(S): select
π<attribute_list>(S):
∪(S,S'): union
S × S': cross product
ρR(S): rename

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

↓

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

SQL

```
SELECT ID, Time
FROM TWEET
WHERE Text = “hey”
```

Relational Algebra

$\sigma_{Text=“hey”}(\Pi_{<ID,Time>}(TWEET))$

$\Pi_{<ID,Time>}(\sigma_{Text=“hey”}(TWEET))$

SQL -> Relational Al

σ<condition>(S): select
π<attribute_list>(S):
∪(S,S’): union
S × S’: cross product
ρR(S): rename

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

↓

ID	Time	Text
389472	12:34:56	hey

SQL

```
SELECT ID, Time
FROM TWEET
WHERE Text = “hey”
```

Relational Algebra

$\sigma_{Text=“hey”}(\Pi_{<ID,Time>}(TWEET))$

$\Pi_{<ID,Time>}(\sigma_{Text=“hey”}(TWEET))$

SQL -> Relational Al

σ<condition>(S): select
π<attribute_list>(S):
∪(S,S’): union
S × S’: cross product
ρR(S): rename

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

ID	Time
389472	12:34:56

SQL

```
SELECT ID, Time
FROM TWEET
WHERE Text = “hey”
```

Relational Algebra

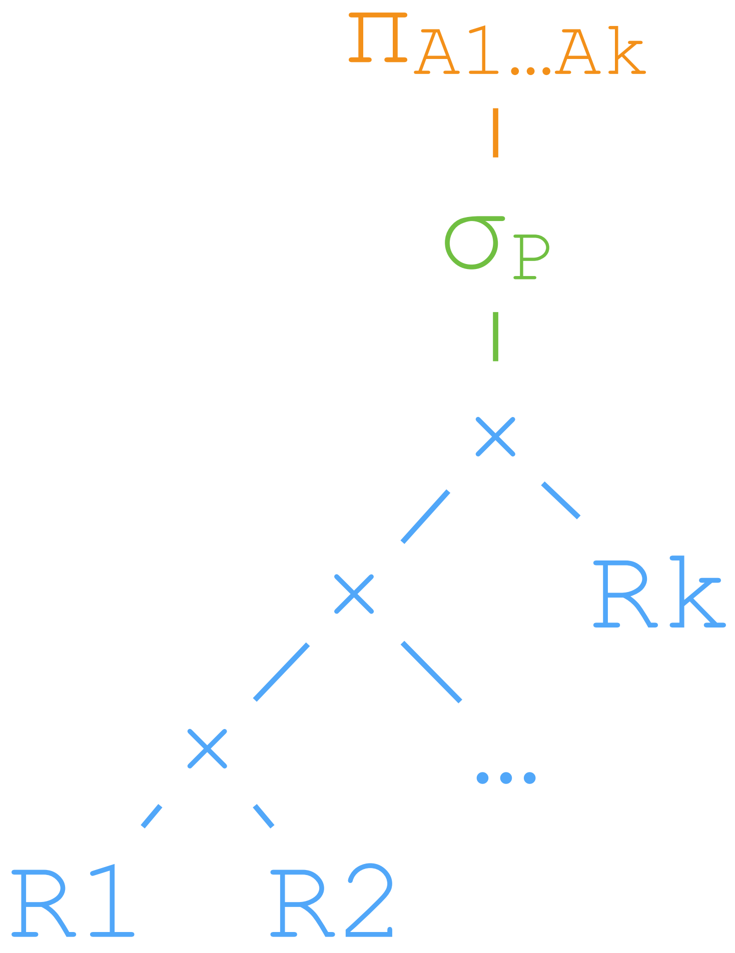
$\sigma_{\text{Text=“hey”}}(\Pi_{\text{<ID,Time>}}(\text{TWEET}))$

$\Pi_{\text{<ID,Time>}}(\sigma_{\text{Text=“hey”}}(\text{TWEET}))$

Execution Order

```
SELECT A1…An
FROM R1…Rk
WHERE P
```

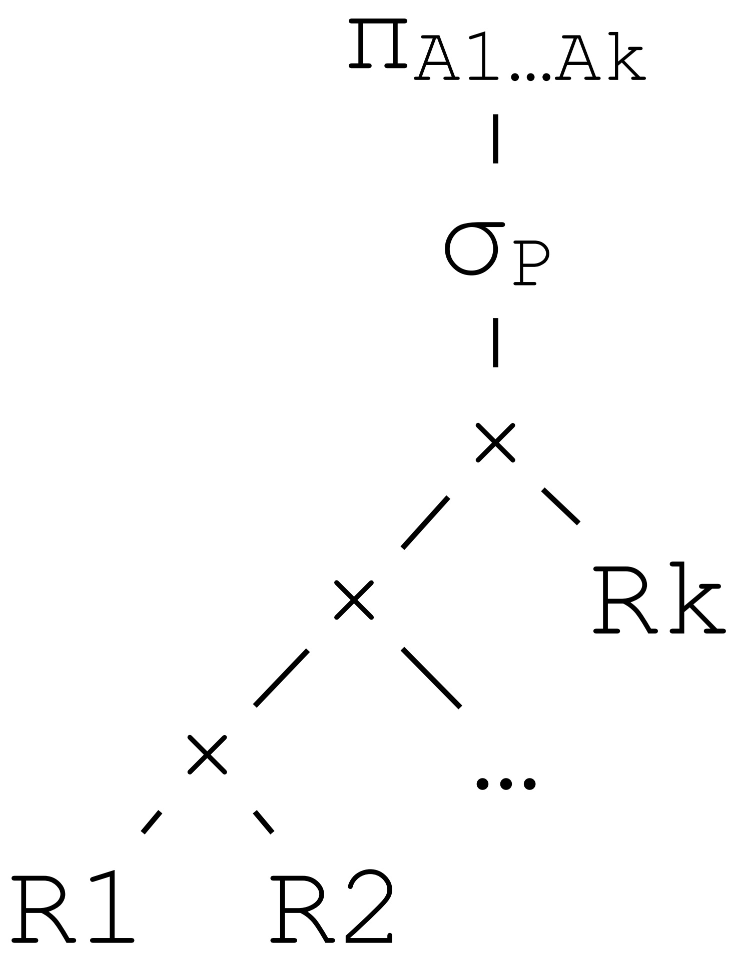
“Canonical Execution Order”
(FROM WHERE SELECT)



Clicker Question!
How much memory do I need?

say each R has
O(m) tuples

```
SELECT A1…An
FROM R1…Rk
WHERE P
```



Clicker Question!
How much memory do I need?

say each R has
O(m) tuples

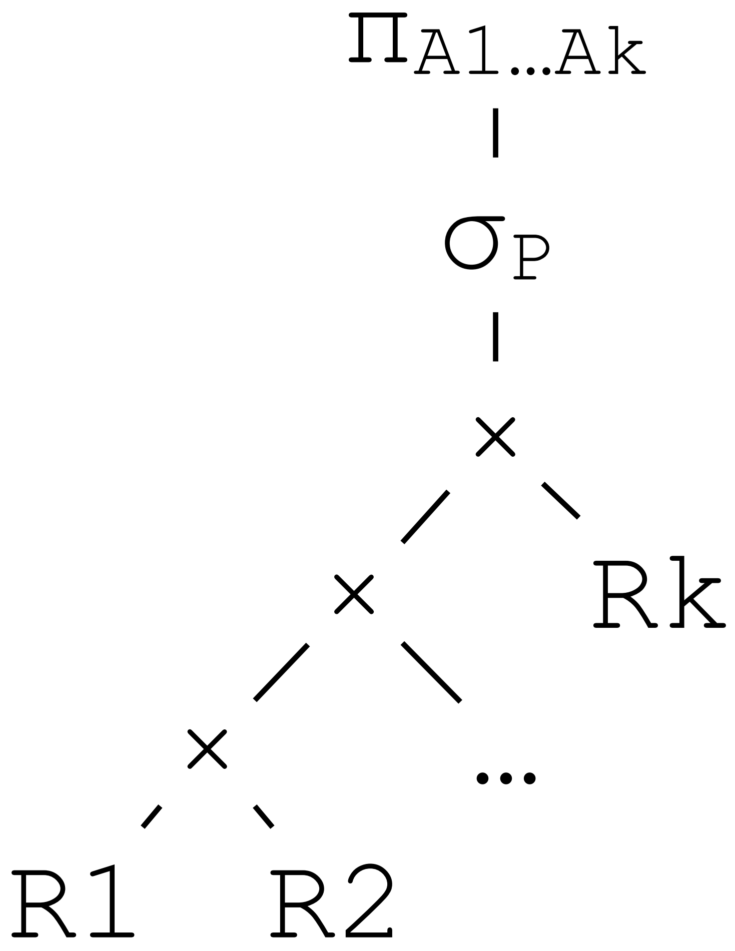
```
SELECT A1…An
FROM R1…Rk
WHERE P
```

(a) $O(m^k)$

(b) $O(m \times k)$

(c) $O(m + k)$

(d) $O(m^{k-n})$



Clicker Question!
How much memory do I need?

say each R has
O(m) tuples

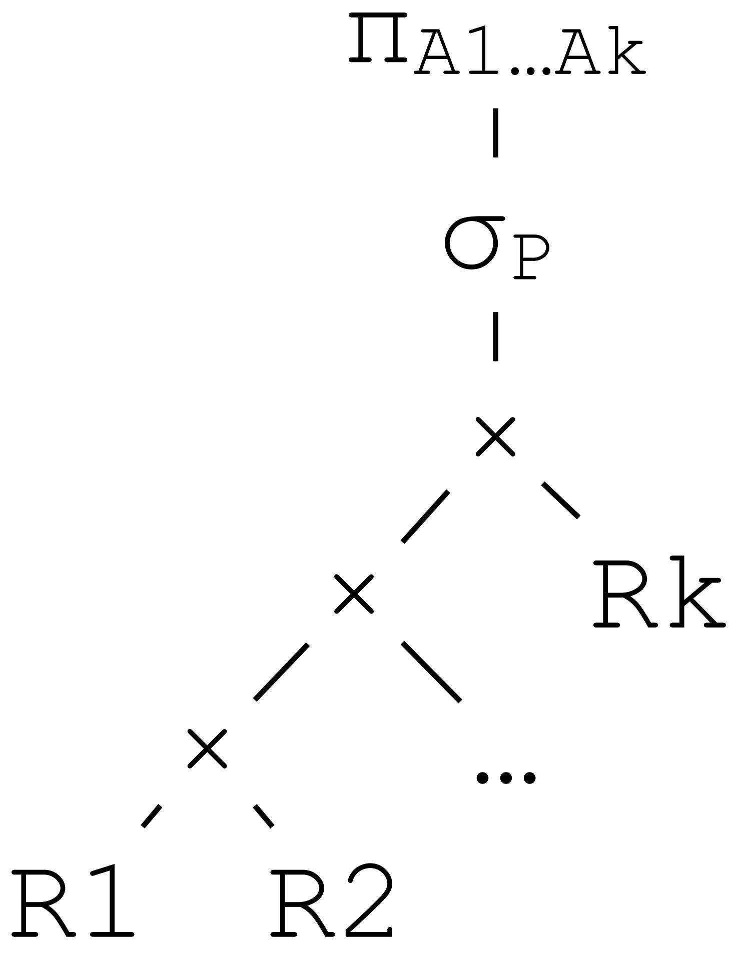
```
SELECT A1…An
FROM R1…Rk
WHERE P
```

(a) $O(m^k)$

[✓ (b)] $O(m \times k)$

(c) $O(m + k)$

(d) $O(m^{k-n})$



Clicker Question!
How much memory do I need?

say each R has
O(m) tuples

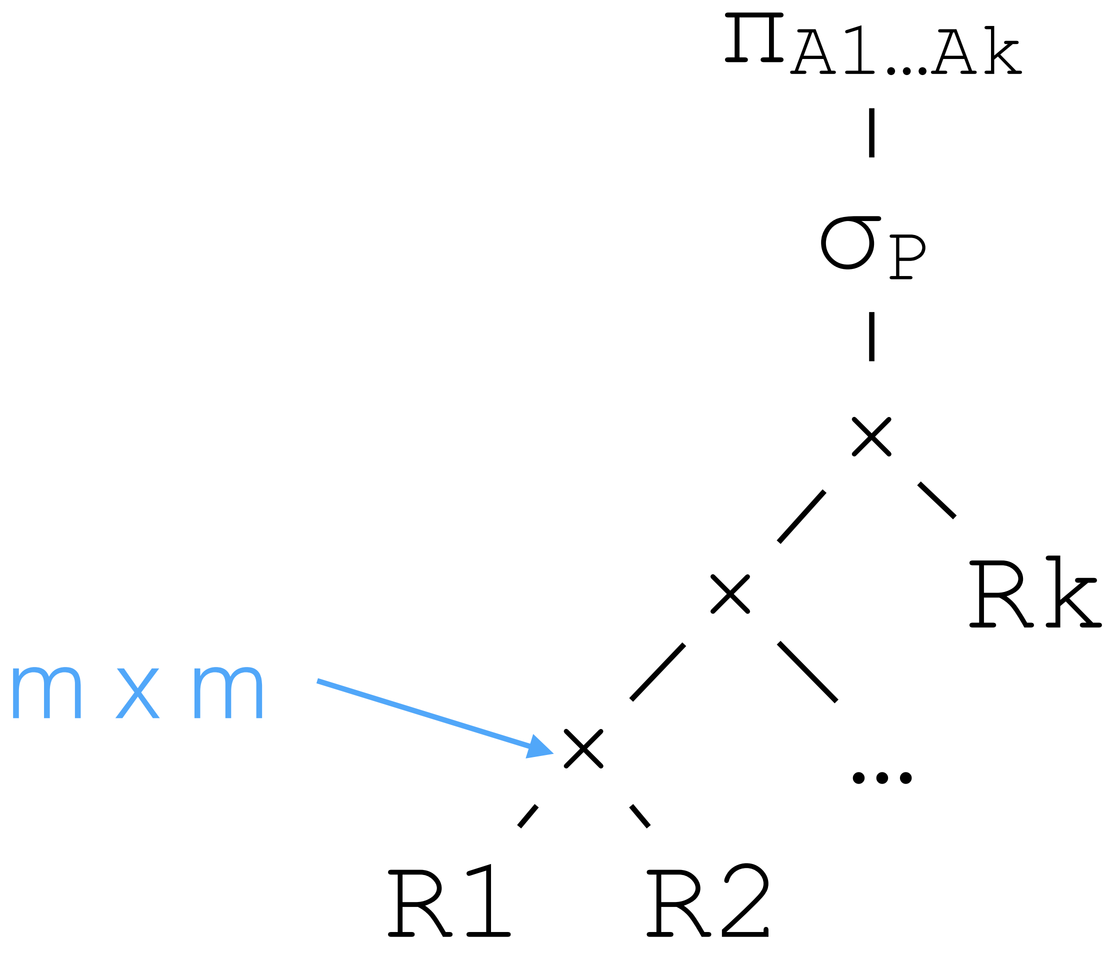
```
SELECT A1…An
FROM R1…Rk
WHERE P
```

(a) $O(m^k)$

[✓ (b) $O(m \times k)$]

(c) $O(m + k)$

(d) $O(m^{k-n})$



Clicker Question!
How much memory do I need?

say each R has
O(m) tuples

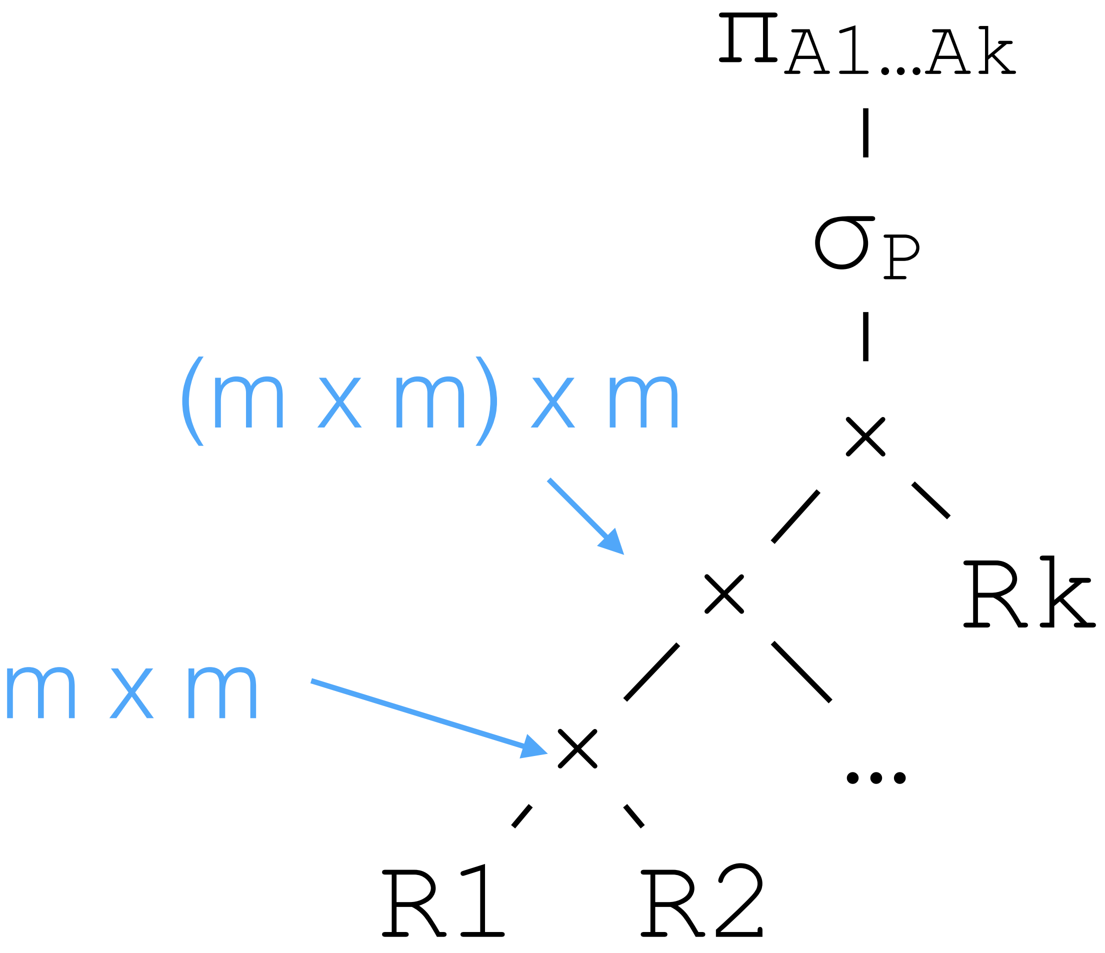
```
SELECT A1…An
FROM R1…Rk
WHERE P
```

(a) $O(m^k)$

[✓ (b) $O(m \times k)$]

(c) $O(m + k)$

(d) $O(m^{k-n})$



Clicker Question!
How much memory do I need?

say each R has
O(m) tuples

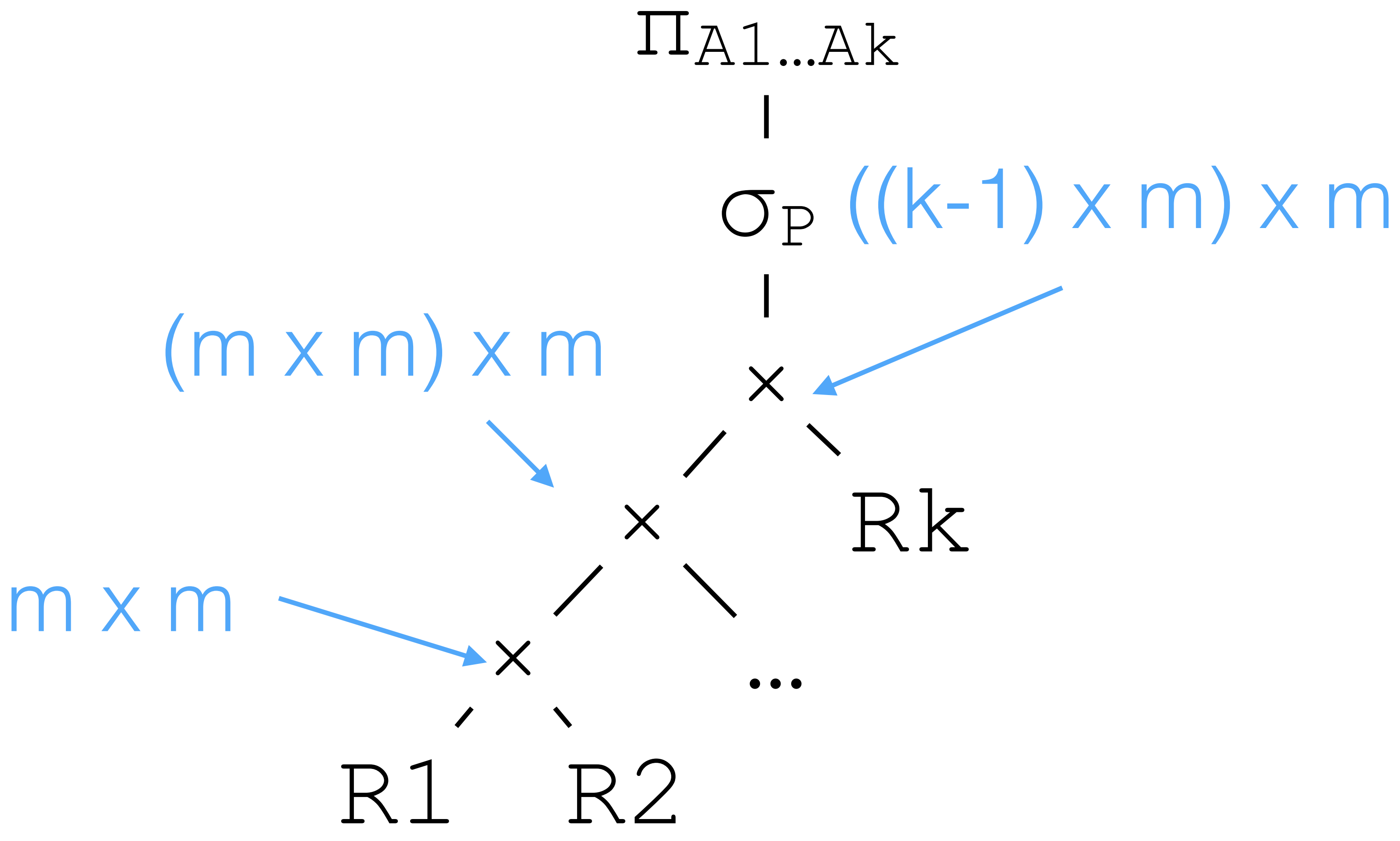
```
SELECT A1…An
FROM R1…Rk
WHERE P
```

[✓ (a)] $O(m^k)$

(b) $O(m \times k)$

(c) $O(m + k)$

(d) $O(m^{k-n})$



Clicker Question!
How much memory do I need?

say each R has
O(m) tuples

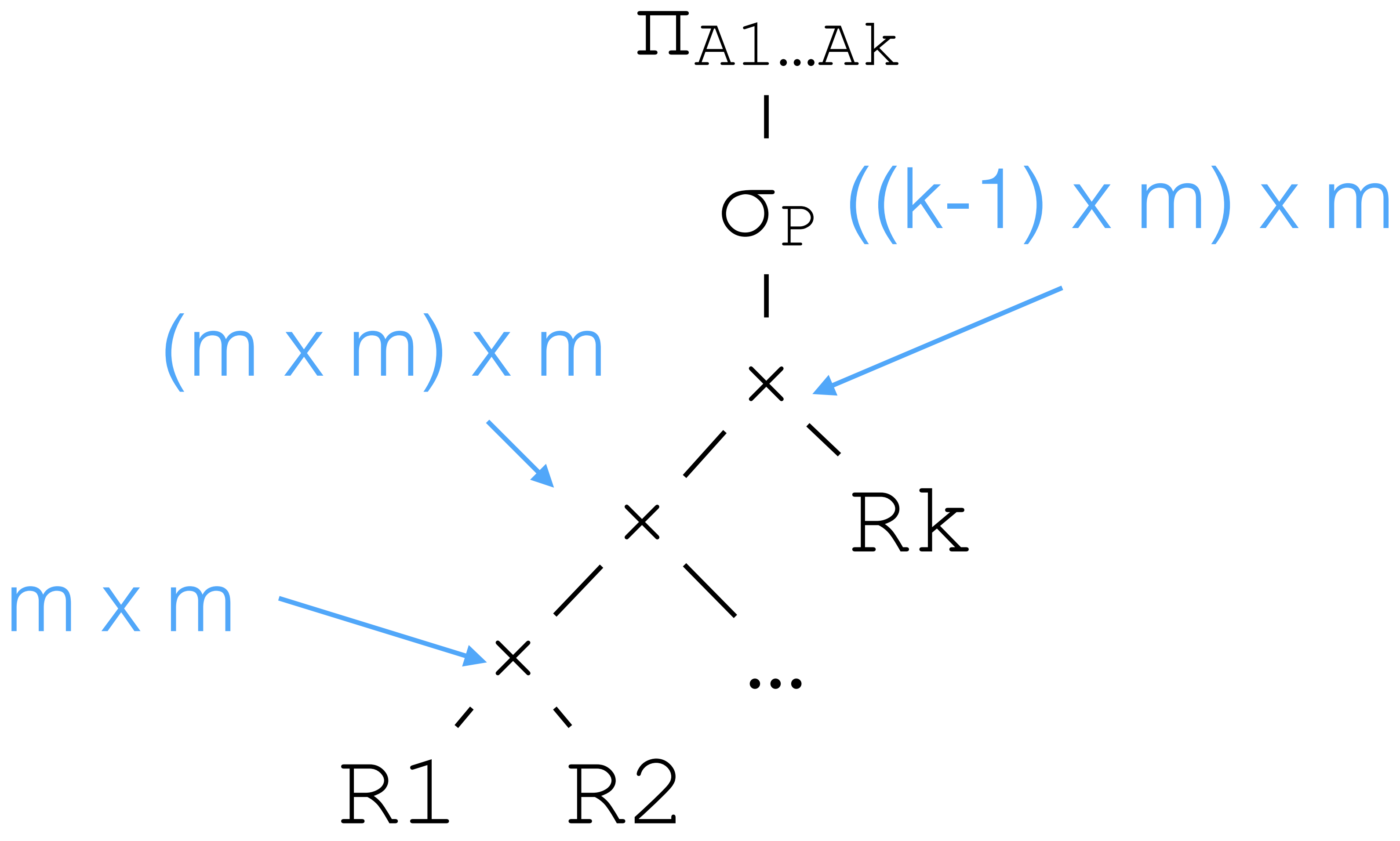
```
SELECT A1…An
FROM R1…Rk
WHERE P
```

[✓ (a)] $O(m^k)$

(b) $O(m \times k)$

(c) $O(m + k)$

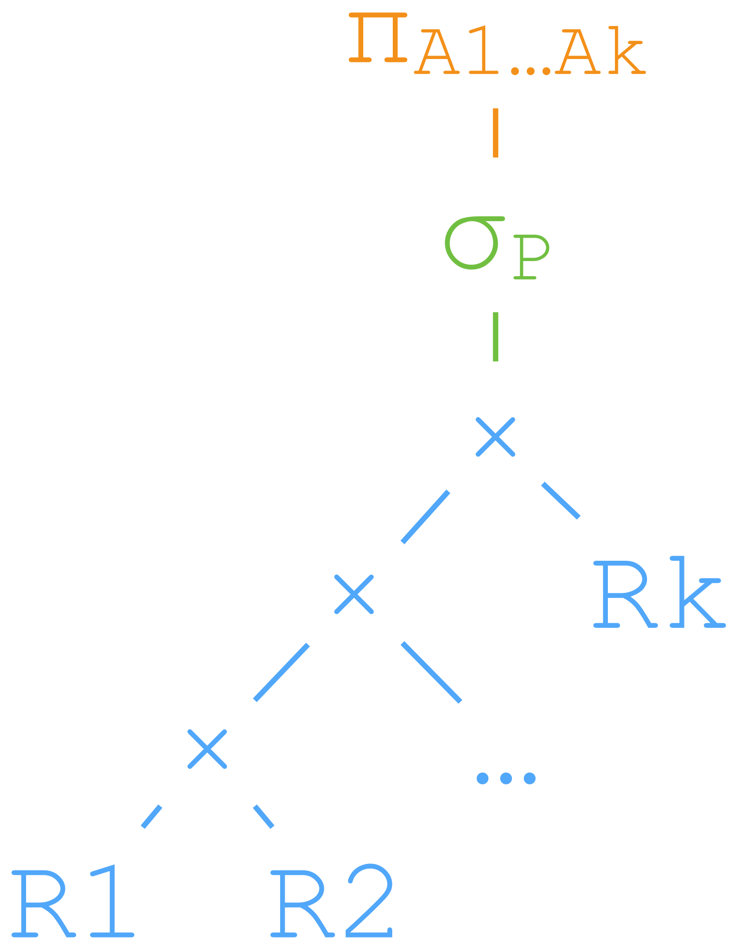
(d) $O(m^{k-n})$



m = 1000, k = 3 —> 1 billion tuples

Execution Order

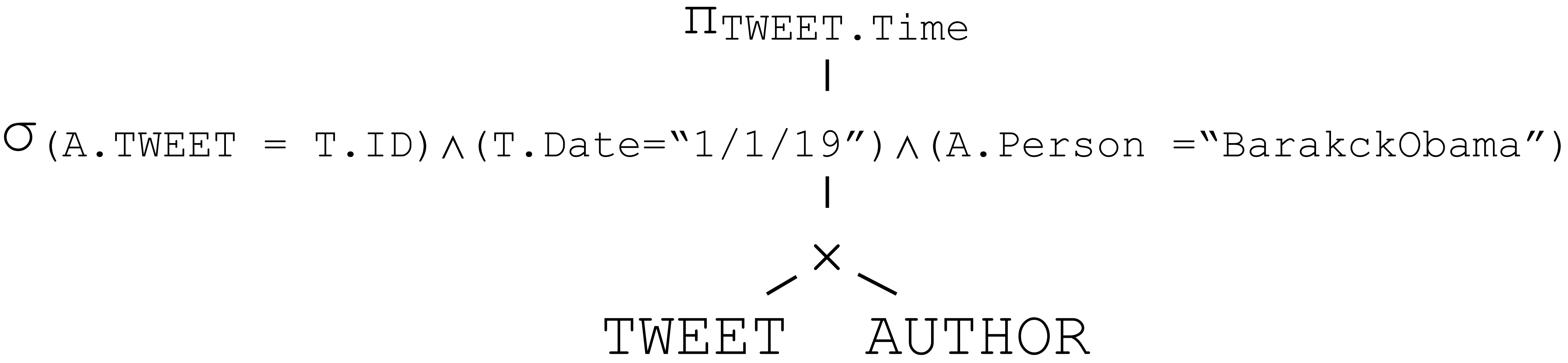
```
SELECT A1…An
FROM R1…Rk
WHERE P
```



“Canonical Execution Order” (FROM WHERE SELECT)

Execution Order

```
SELECT TWEET.Time
FROM TWEET, AUTHOR
WHERE AUTHOR.TWEET = TWEET.ID
      and TWEET.Date == ’01/01/2019‘
      and AUTHOR.Person = “BarackObama”
```



$\pi_{\texttt{TWEET.Time}}$

$\sigma_{\texttt{(A.TWEET = T.ID)} \wedge \texttt{(T.Date=“1/1/19”)} \wedge \texttt{(A.Person =“BarakckObama”)}}$

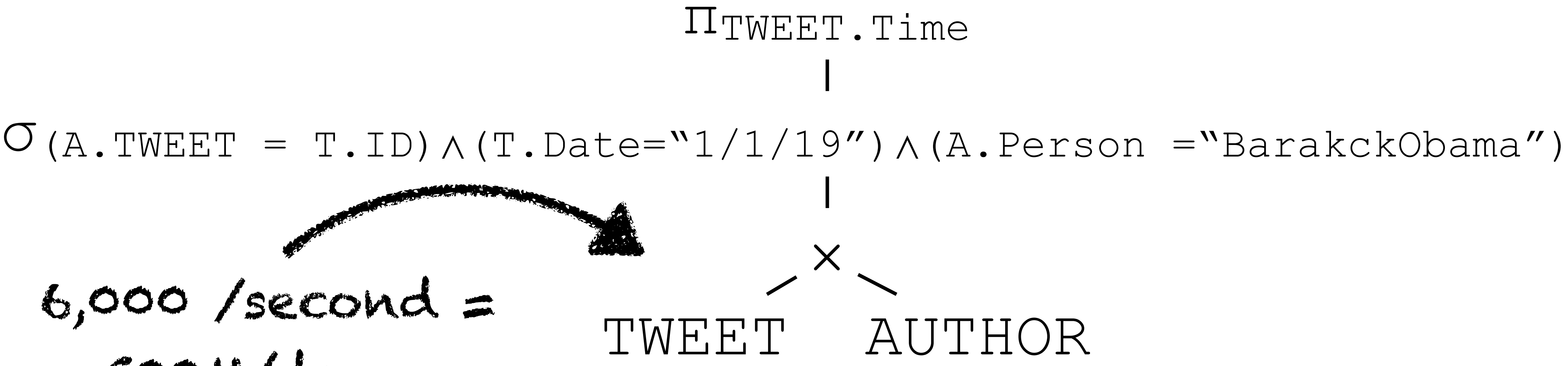
×

TWEET AUTHOR

“Canonical Execution Order” (FROM WHERE SELECT)

Execution Order

```
SELECT TWEET.Time
FROM TWEET, AUTHOR
WHERE AUTHOR.TWEET = TWEET.ID
      and TWEET.Date == ’01/01/2019‘
      and AUTHOR.Person = “BarackObama”
```



$\pi_{\texttt{TWEET.Time}}$

$\sigma_{\texttt{(A.TWEET = T.ID)} \wedge \texttt{(T.Date=“1/1/19”)} \wedge \texttt{(A.Person =“BarakckObama”)}}$

$\times$

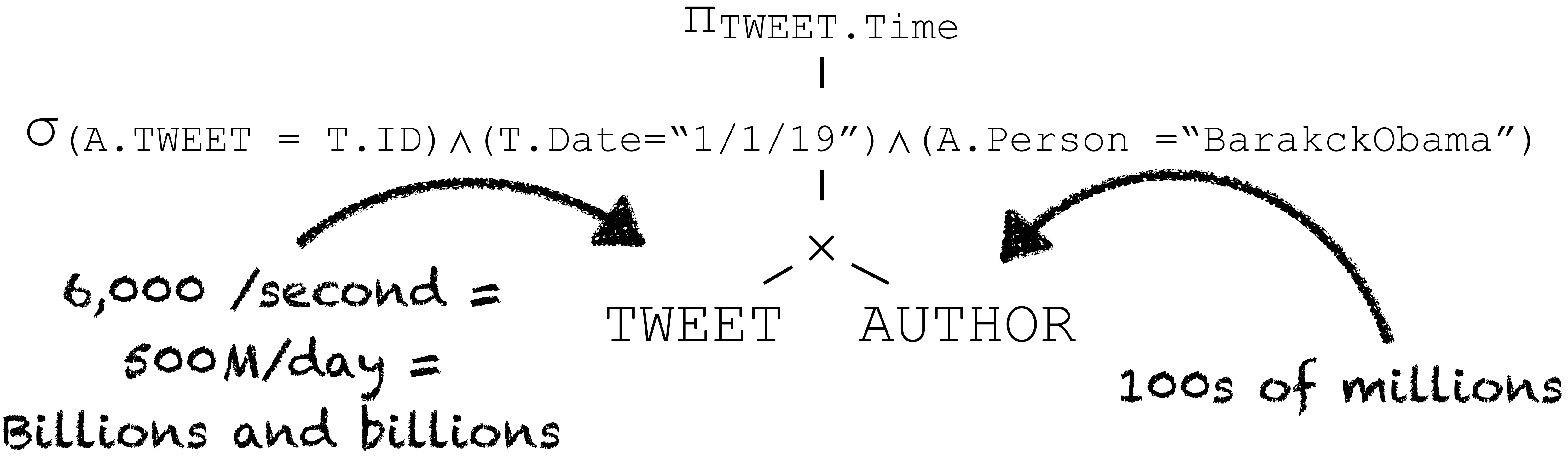
TWEET AUTHOR

6,000 /second =
500M/day =
Billions and billions

http://www.internetlivestats.com/twitter-statistics/
https://www.omnicoreagency.com/twitter-statistics/

Execution Order

```
SELECT TWEET.Time
FROM TWEET, AUTHOR
WHERE AUTHOR.TWEET = TWEET.ID
      and TWEET.Date == ’01/01/2019‘
      and AUTHOR.Person = “BarackObama”
```



$\pi_{\texttt{TWEET.Time}}$

σ (A.TWEET = T.ID)∧(T.Date=“1/1/19”)∧(A.Person =“BarakckObama”)

×

TWEET AUTHOR

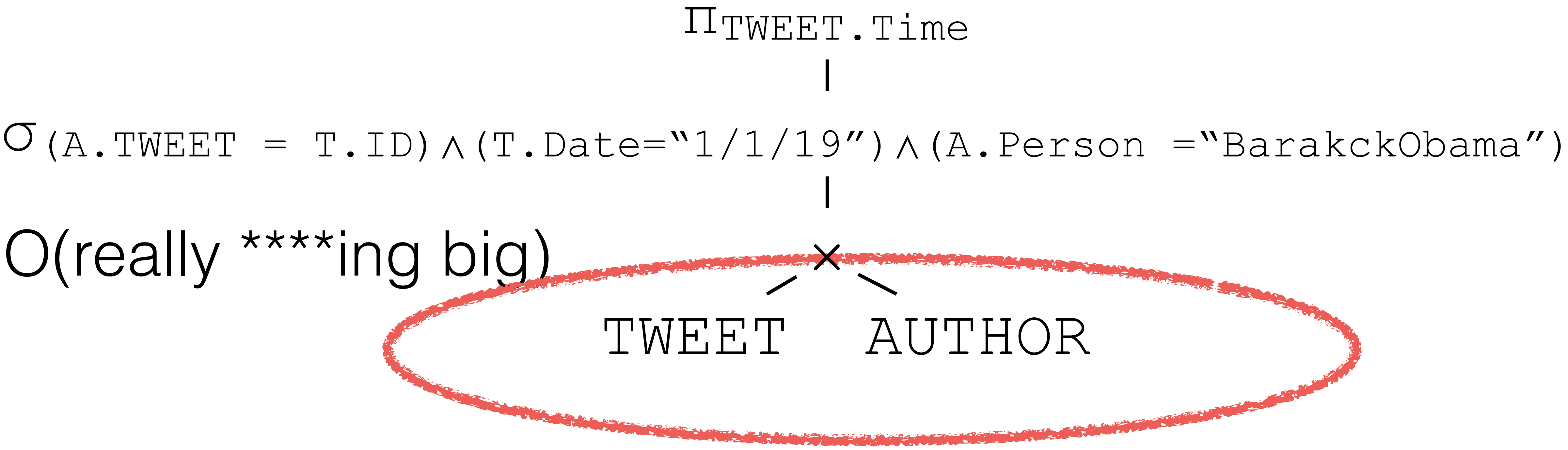
6,000 /second = 500M/day = Billions and billions

100s of millions

http://www.internetlivestats.com/twitter-statistics/
https://www.omnicoreagency.com/twitter-statistics/

Execution Order

```
SELECT TWEET.Time
FROM TWEET, AUTHOR
WHERE AUTHOR.TWEET = TWEET.ID
      and TWEET.Date == ’01/01/2019‘
      and AUTHOR.Person = “BarackObama”
```



$\pi_{\text{TWEET.Time}}$

$\sigma_{\text{(A.TWEET = T.ID)} \wedge \text{(T.Date=“1/1/19”)} \wedge \text{(A.Person =“BarakckObama”)}}$

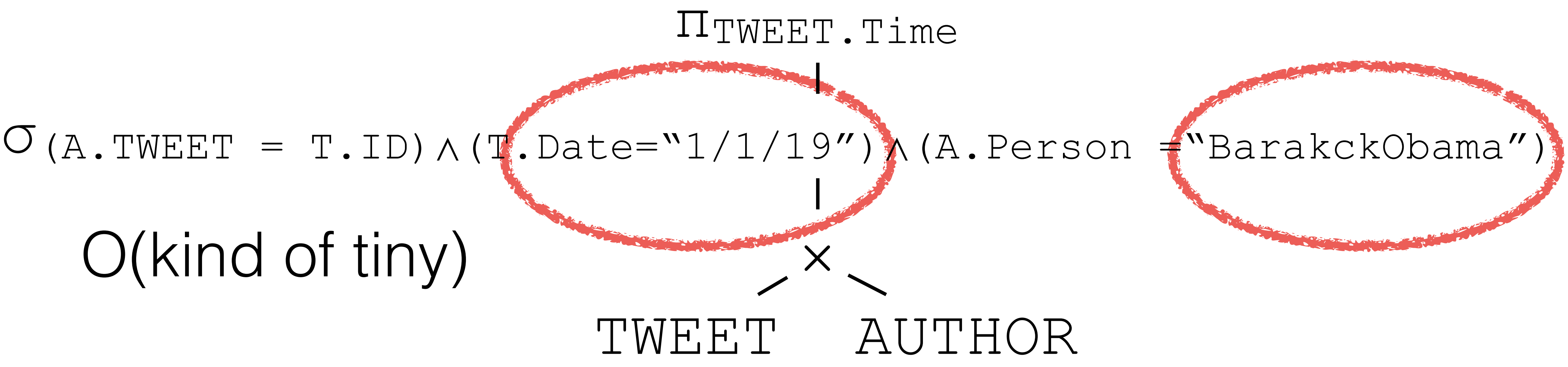
×

TWEET AUTHOR

O(really ****ing big)

Execution Order

```
SELECT TWEET.Time
FROM TWEET, AUTHOR
WHERE AUTHOR.TWEET = TWEET.ID
      and TWEET.Date == ’01/01/2019‘
      and AUTHOR.Person = “BarackObama”
```



$\pi_{\text{TWEET.Time}}$

$\sigma_{(\text{A.TWEET = T.ID}) \wedge (\text{T.Date=“1/1/19”}) \wedge (\text{A.Person =“BarakckObama”})}$

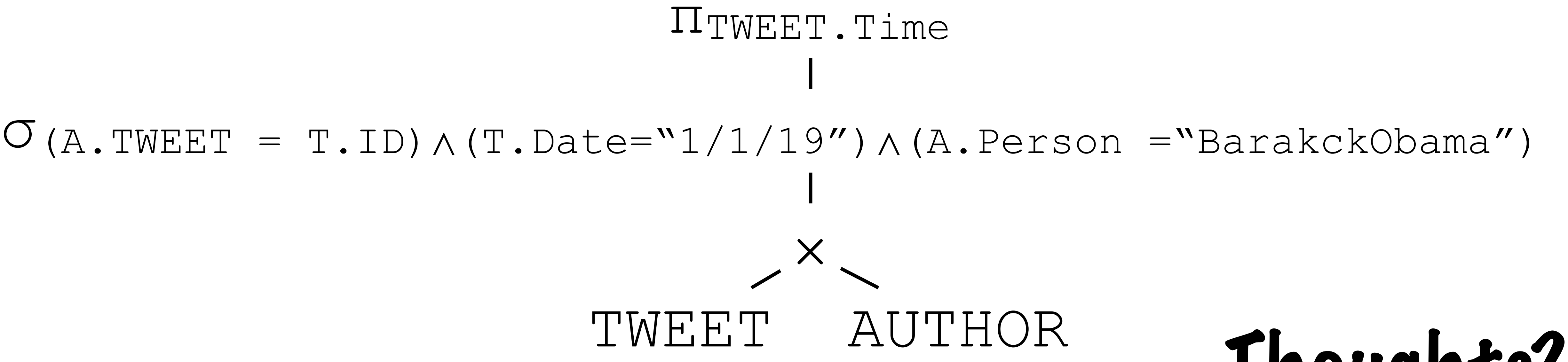
O(kind of tiny)

×

TWEET AUTHOR

Execution Order

```
SELECT TWEET.Time
FROM TWEET, AUTHOR
WHERE AUTHOR.TWEET = TWEET.ID
      and TWEET.Date == ’01/01/2019‘
      and AUTHOR.Person = “BarackObama”
```



$\pi_{\text{TWEET.Time}}$

$\sigma_{\text{(A.TWEET = T.ID)} \wedge \text{(T.Date=“1/1/19”)} \wedge \text{(A.Person =“BarakckObama”)}}$

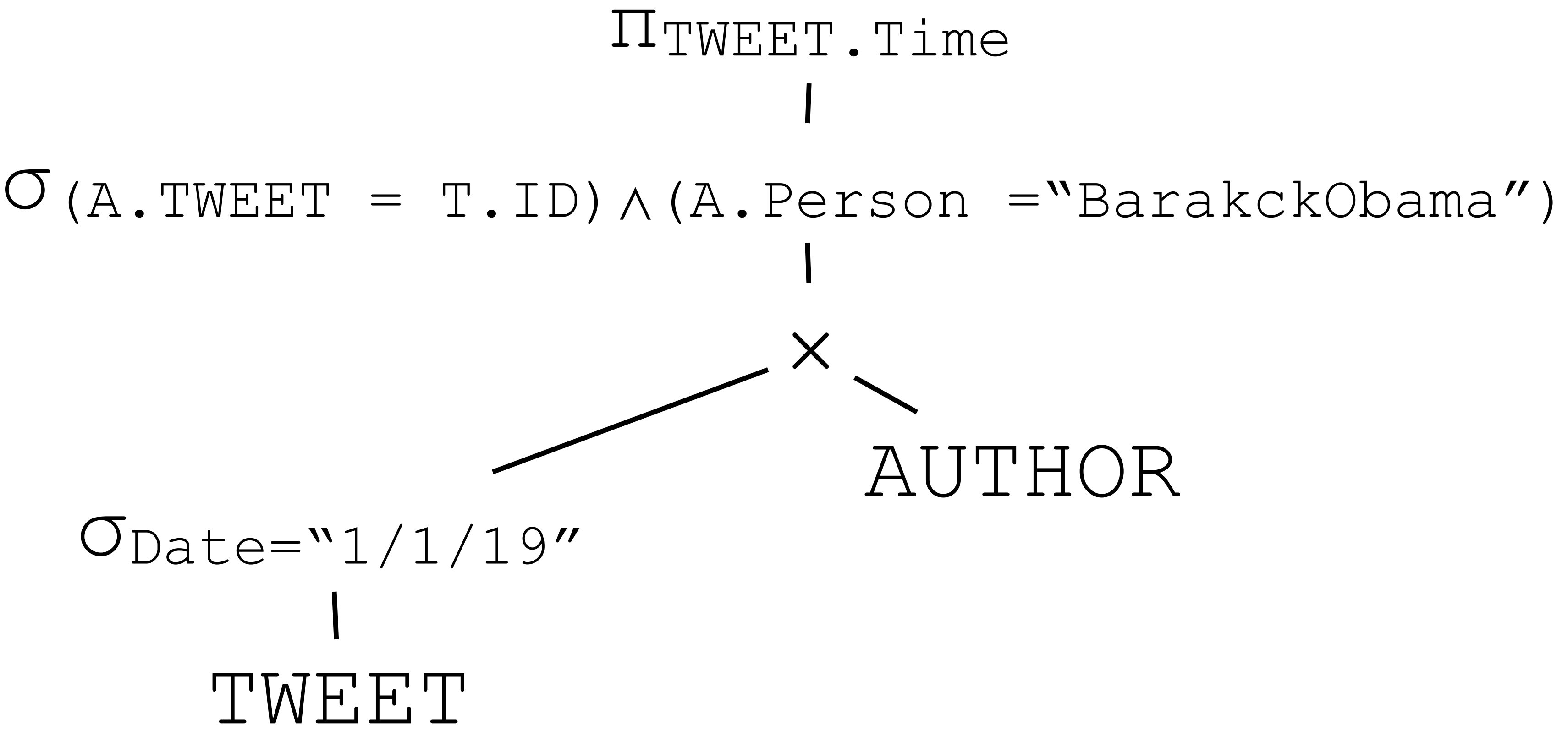
×

TWEET AUTHOR

Thoughts??

Execution Order

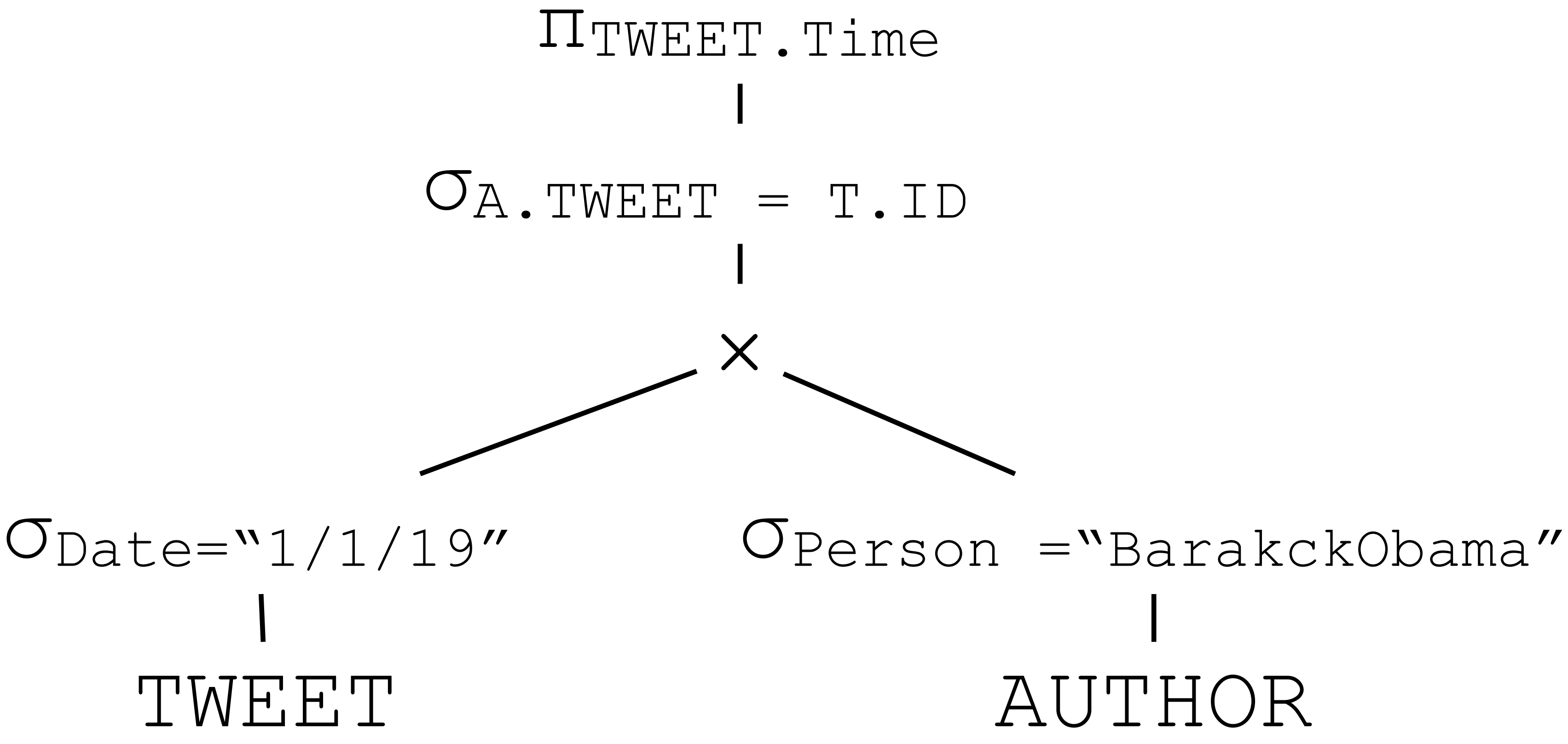
```
SELECT TWEET.Time
FROM TWEET, AUTHOR
WHERE AUTHOR.TWEET = TWEET.ID
      and TWEET.Date == ’01/01/2019‘
      and AUTHOR.Person = “BarackObama”
```



$\pi_{\text{TWEET.Time}}$
$\sigma_{(\text{A.TWEET = T.ID}) \wedge (\text{A.Person =“BarakckObama”})}$
×
AUTHOR
$\sigma_{\text{Date=“1/1/19”}}$
TWEET

Execution Order

```
SELECT TWEET.Time
FROM TWEET, AUTHOR
WHERE AUTHOR.TWEET = TWEET.ID
      and TWEET.Date == ’01/01/2019‘
      and AUTHOR.Person = “BarackObama”
```



$\pi_{\texttt{TWEET.Time}}$

$\sigma_{\texttt{A.TWEET = T.ID}}$

×

$\sigma_{\texttt{Date=“1/1/19”}}$

$\sigma_{\texttt{Person =“BarakckObama”}}$

TWEET

AUTHOR

Clicker Question! (Demand?)

Optimize this.

Find grades of students taking 1951A ahead of schedule

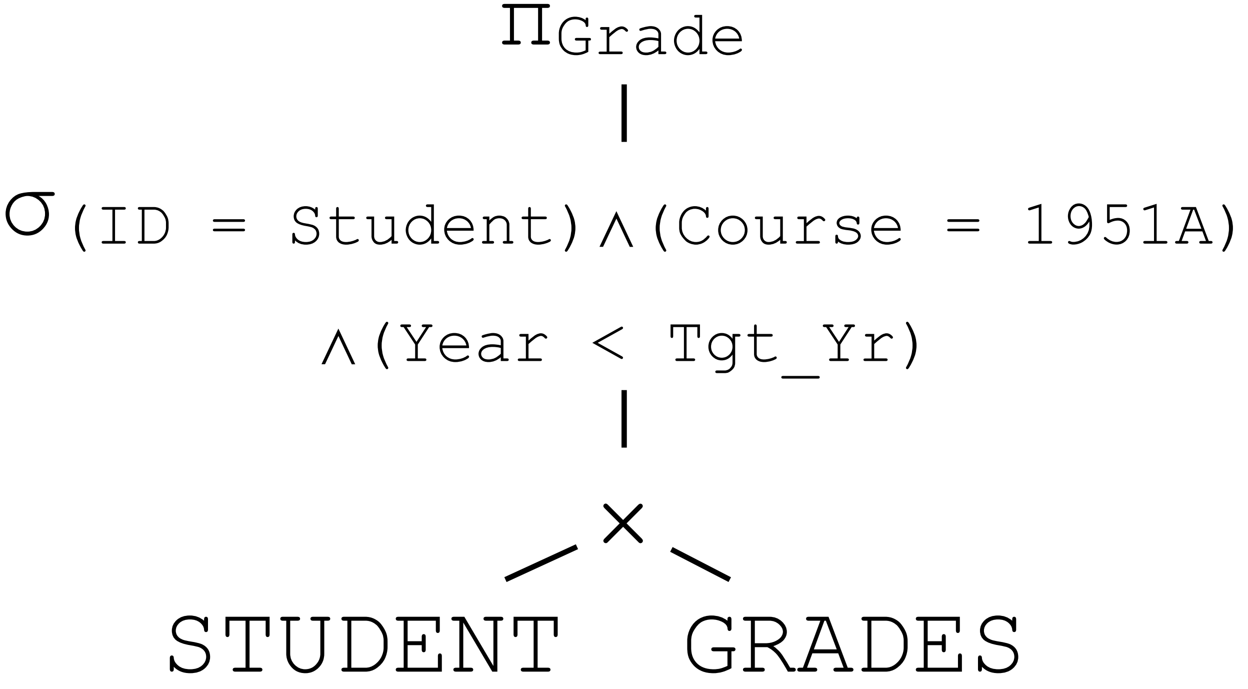
STUDENT

ID	Name	Year
1	Wennie	4
2	Maulik	5
3	Gurnaa	5
4	Jens	4
5	Erin	4

GRADES

Student	Course	Grade	Tgt_Yr
1	32	A	1
2	1951A	A	3
6	32	A	1

```
SELECT Grade
FROM STUDENT, GRADES
WHERE STUDENT.ID = GRADES.Student
  and GRADES.Course == ’1951A‘
  and STUDENT.Year < GRADES.Tgt_Yr
```



π_{Grade}

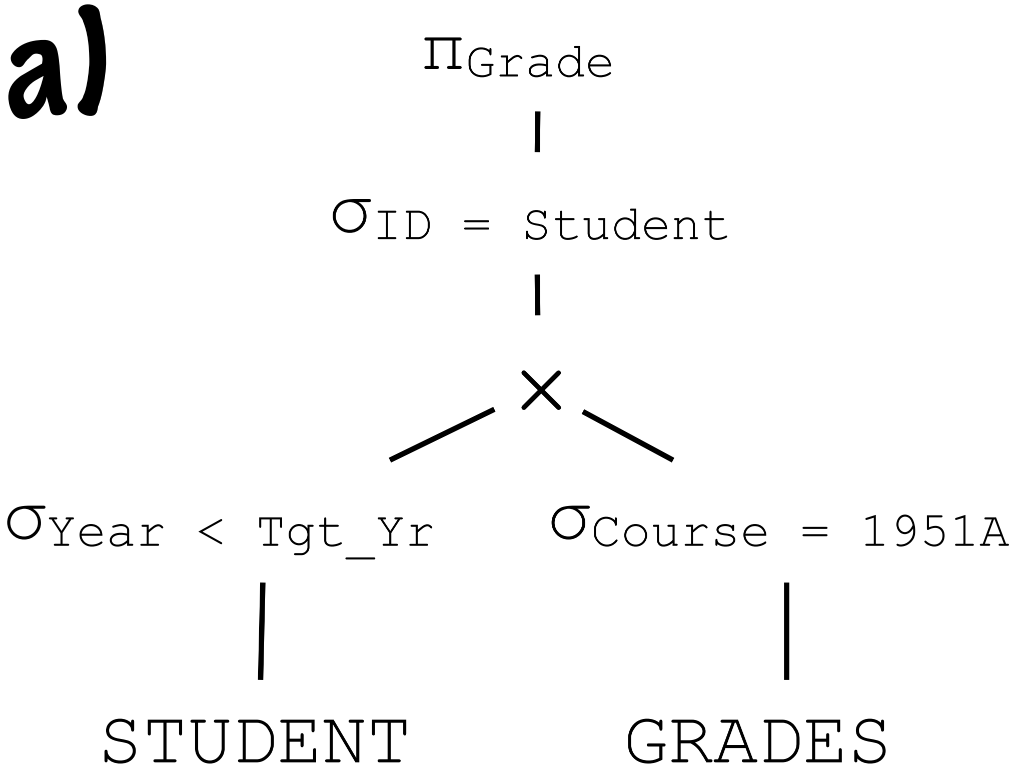
$\sigma_{(\text{ID} = \text{Student}) \wedge (\text{Course} = 1951\text{A}) \wedge (\text{Year} < \text{Tgt\_Yr})}$

$\times$

STUDENT GRADES

Clicker Question!

(a)



π_{Grade}

$\sigma_{\text{ID = Student}}$

$\times$

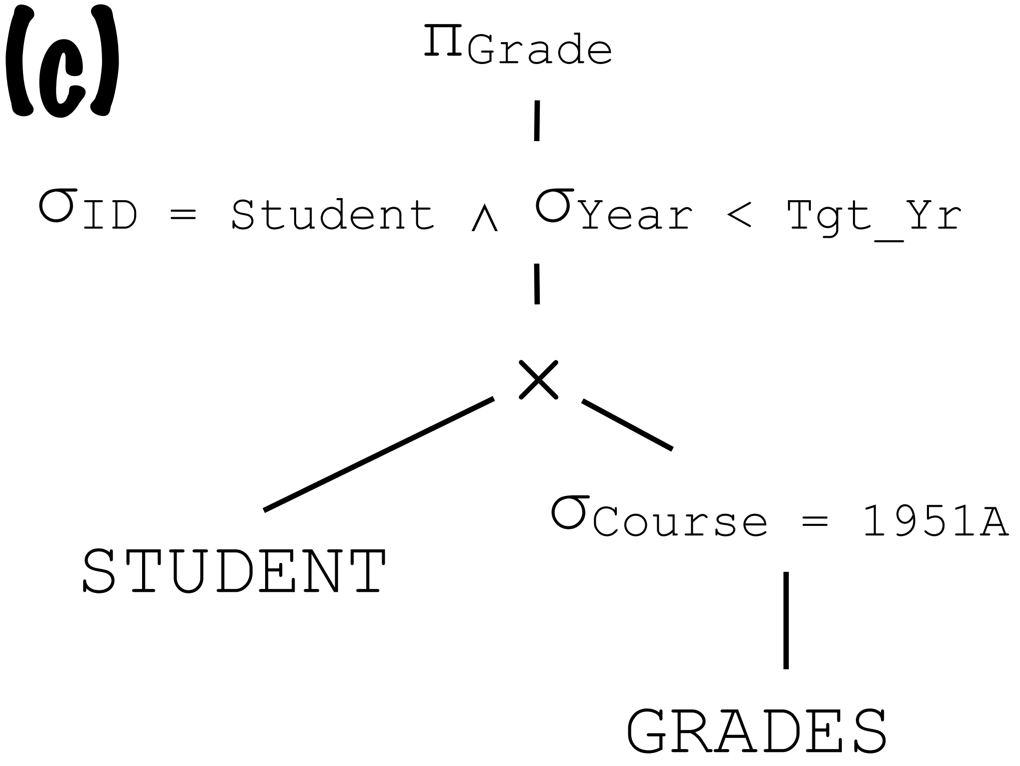
$\sigma_{\text{Year < Tgt\_Yr}}$

$\sigma_{\text{Course = 1951A}}$

STUDENT

GRADES

(c)



π_{Grade}

$\sigma_{\text{ID = Student}} \wedge \sigma_{\text{Year < Tgt\_Yr}}$

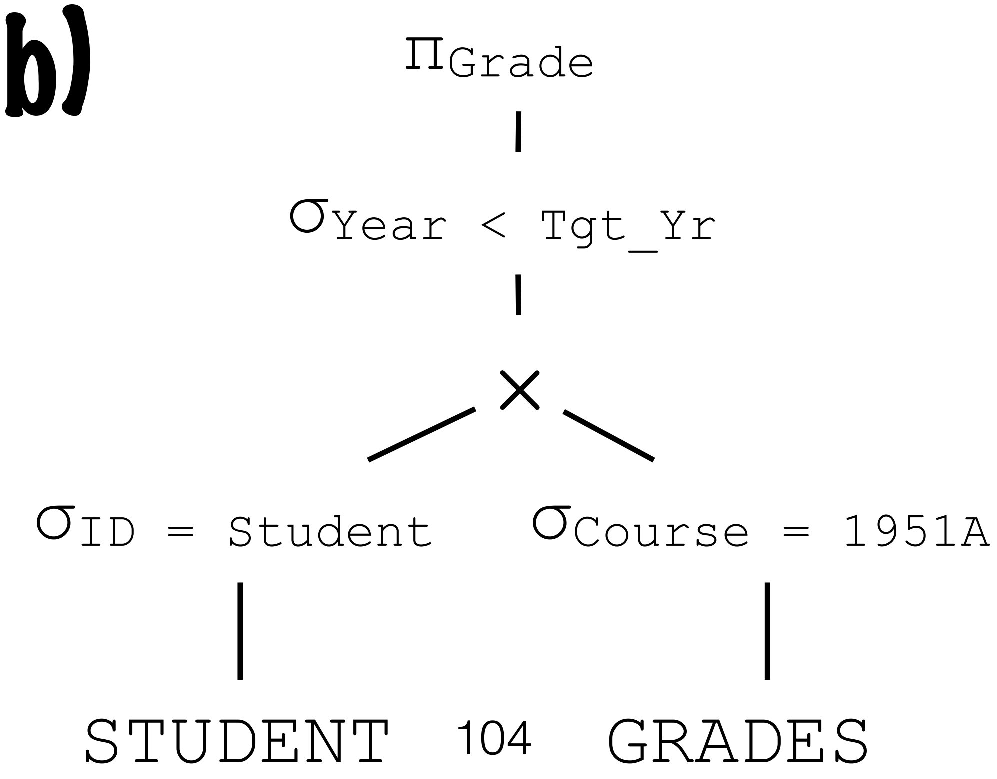
$\times$

STUDENT

$\sigma_{\text{Course = 1951A}}$

GRADES

(b)



π_{Grade}

$\sigma_{\text{Year < Tgt\_Yr}}$

$\times$

$\sigma_{\text{ID = Student}}$

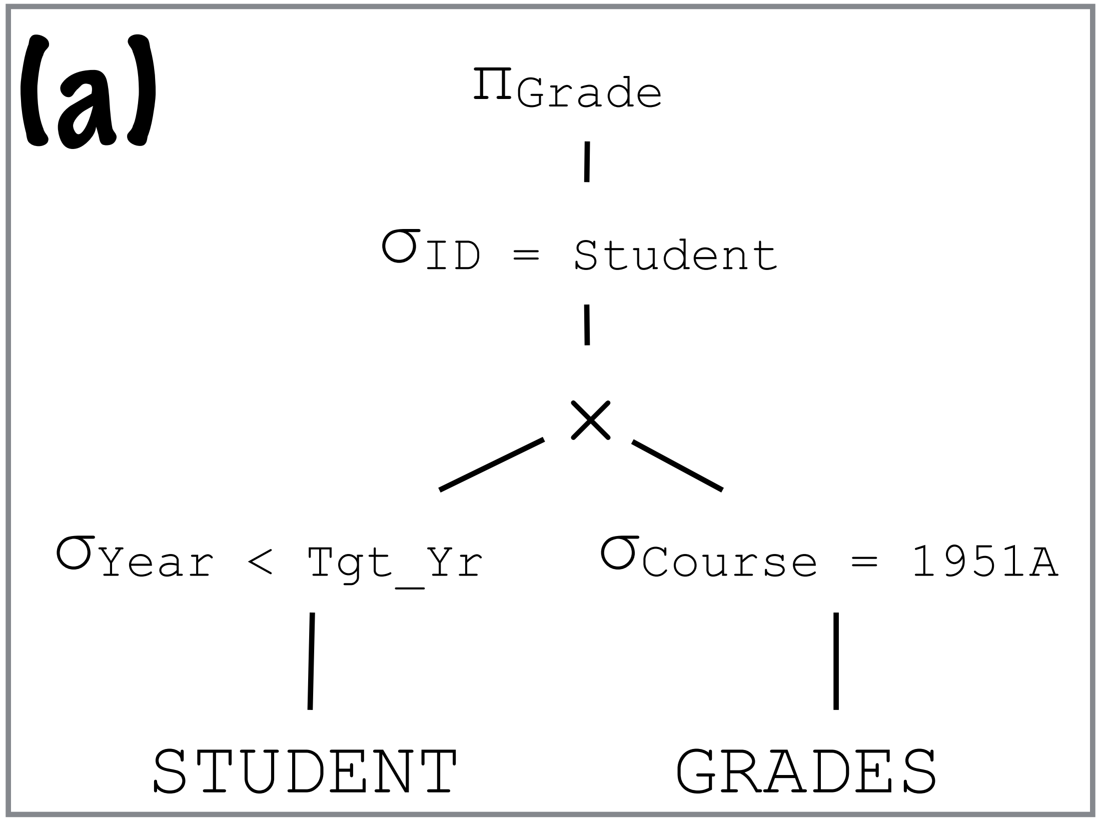
$\sigma_{\text{Course = 1951A}}$

STUDENT

GRADES

Clicker Question!

(a)



π_{Grade}

$\sigma_{ID = Student}$

$\times$

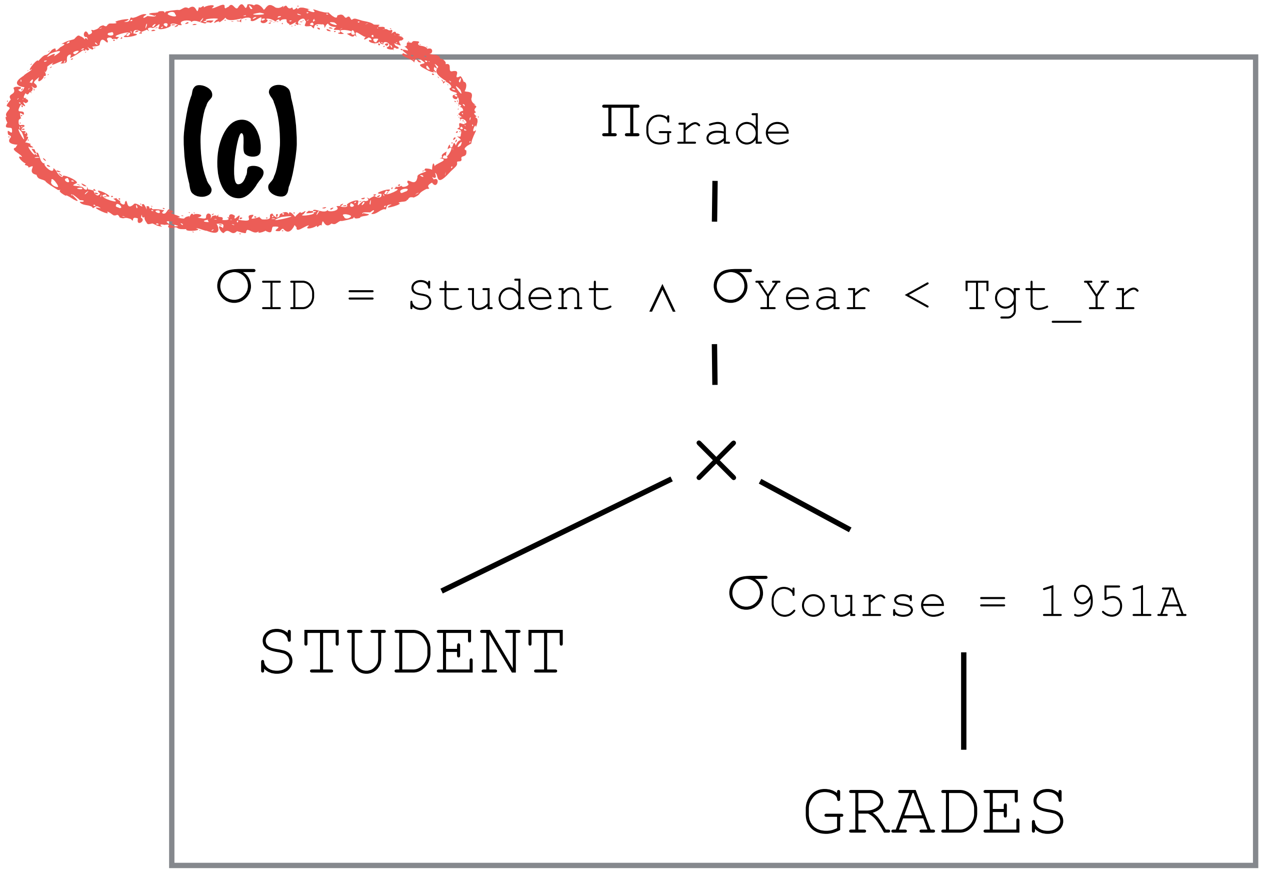
$\sigma_{Year < Tgt\_Yr}$

$\sigma_{Course = 1951A}$

STUDENT

GRADES

[✓ (c)]



π_{Grade}

$\sigma_{ID = Student \wedge} \sigma_{Year < Tgt\_Yr}$

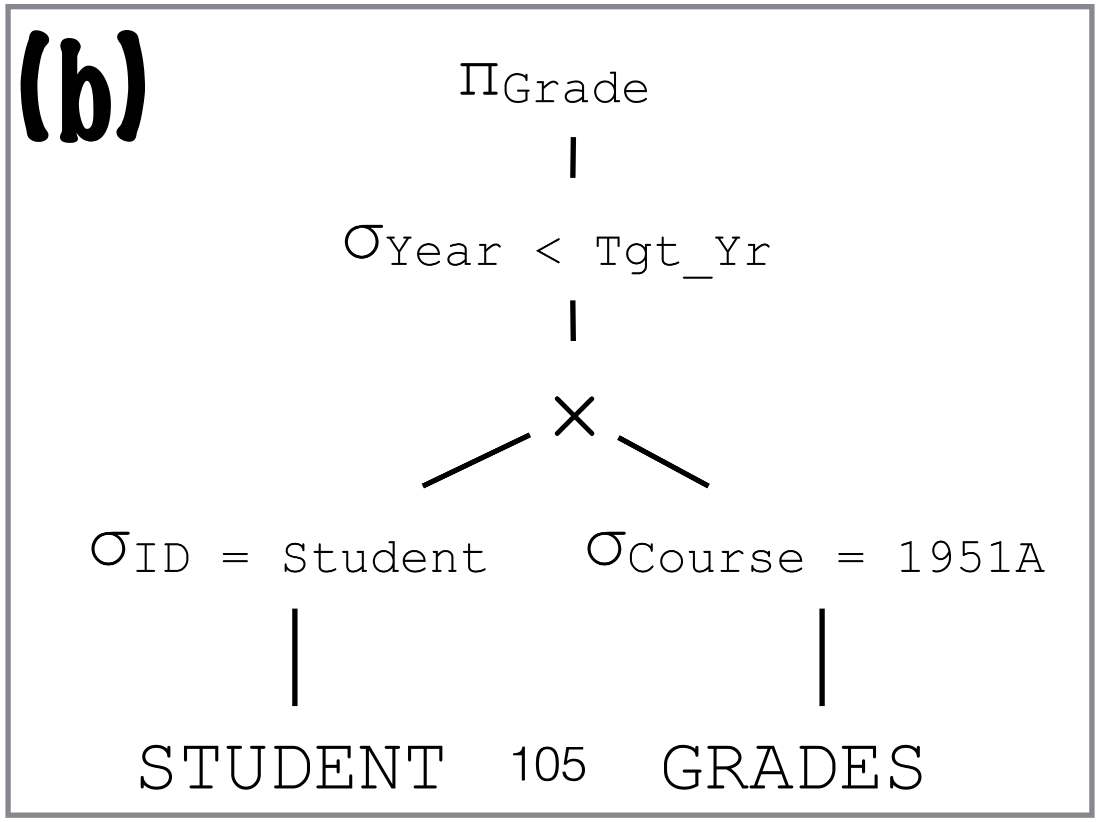
$\times$

STUDENT

$\sigma_{Course = 1951A}$

GRADES

(b)



π_{Grade}

$\sigma_{Year < Tgt\_Yr}$

$\times$

$\sigma_{ID = Student}$

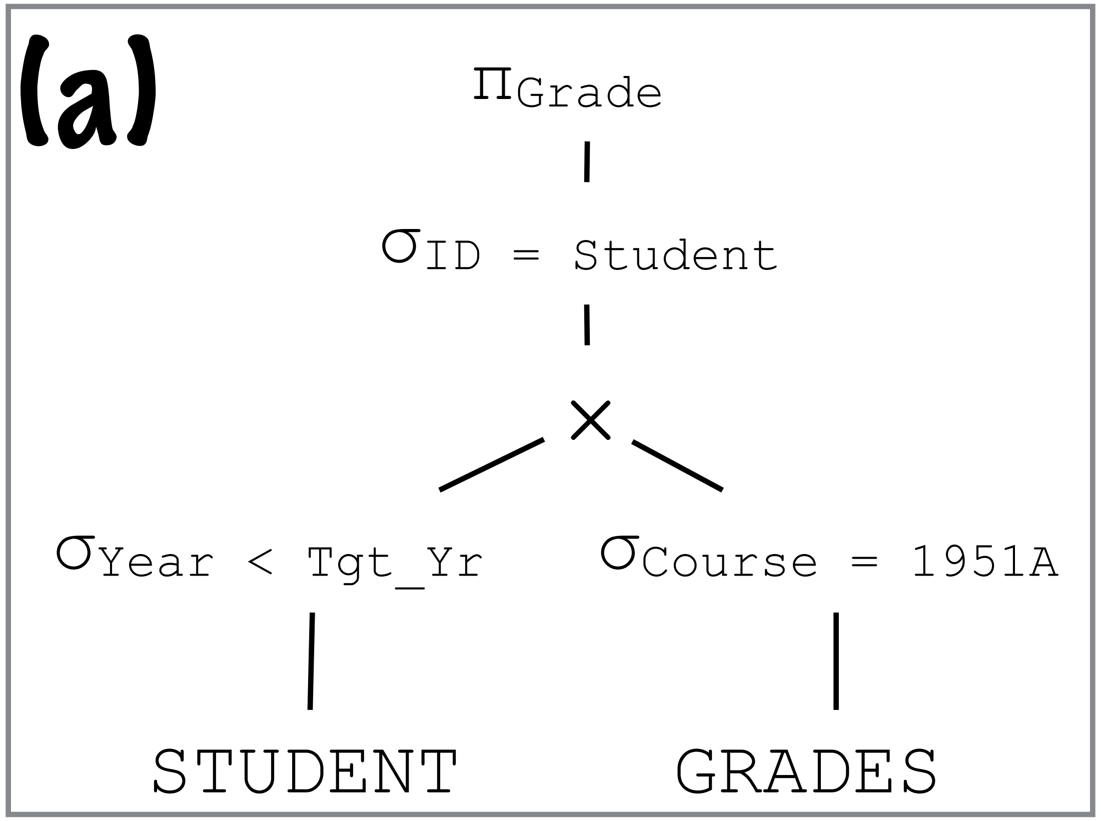
$\sigma_{Course = 1951A}$

STUDENT

GRADES

Clicker Question!

(a)



π_{Grade}

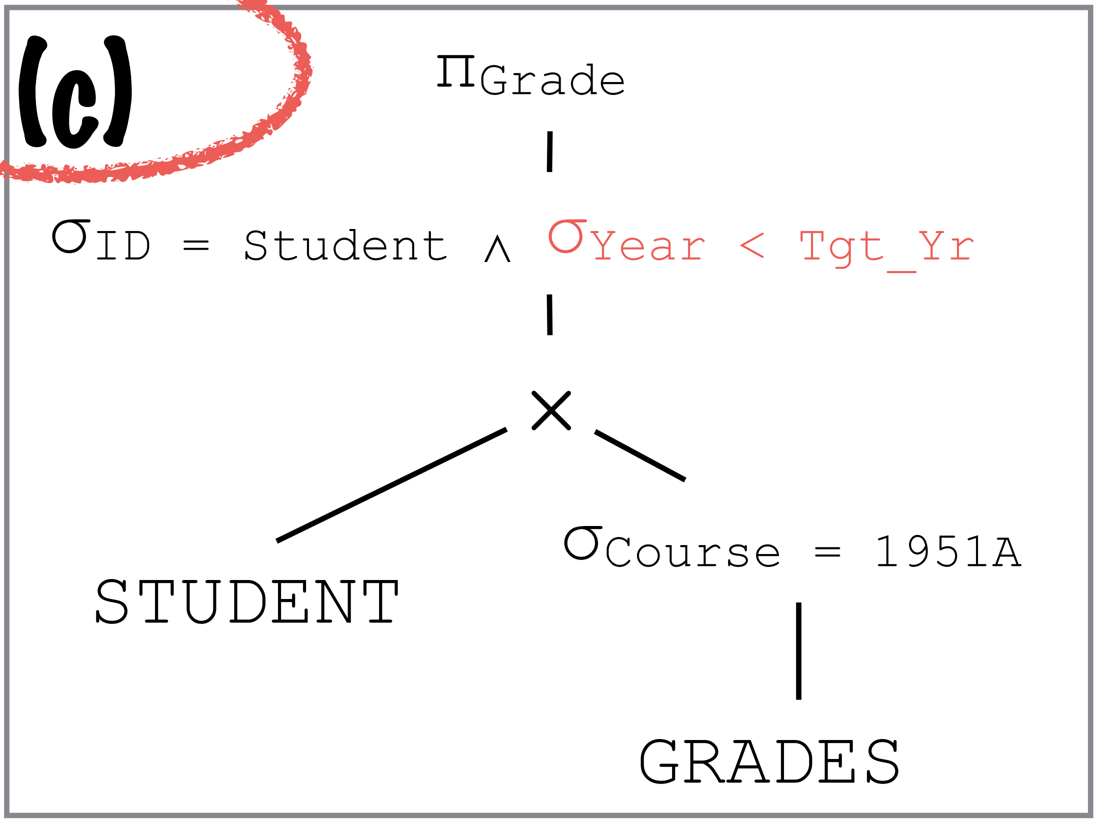
$\sigma_{ID\ =\ Student}$

×

$\sigma_{Year\ <\ Tgt\_Yr}$ $\sigma_{Course\ =\ 1951A}$

STUDENT GRADES

(c)



π_{Grade}

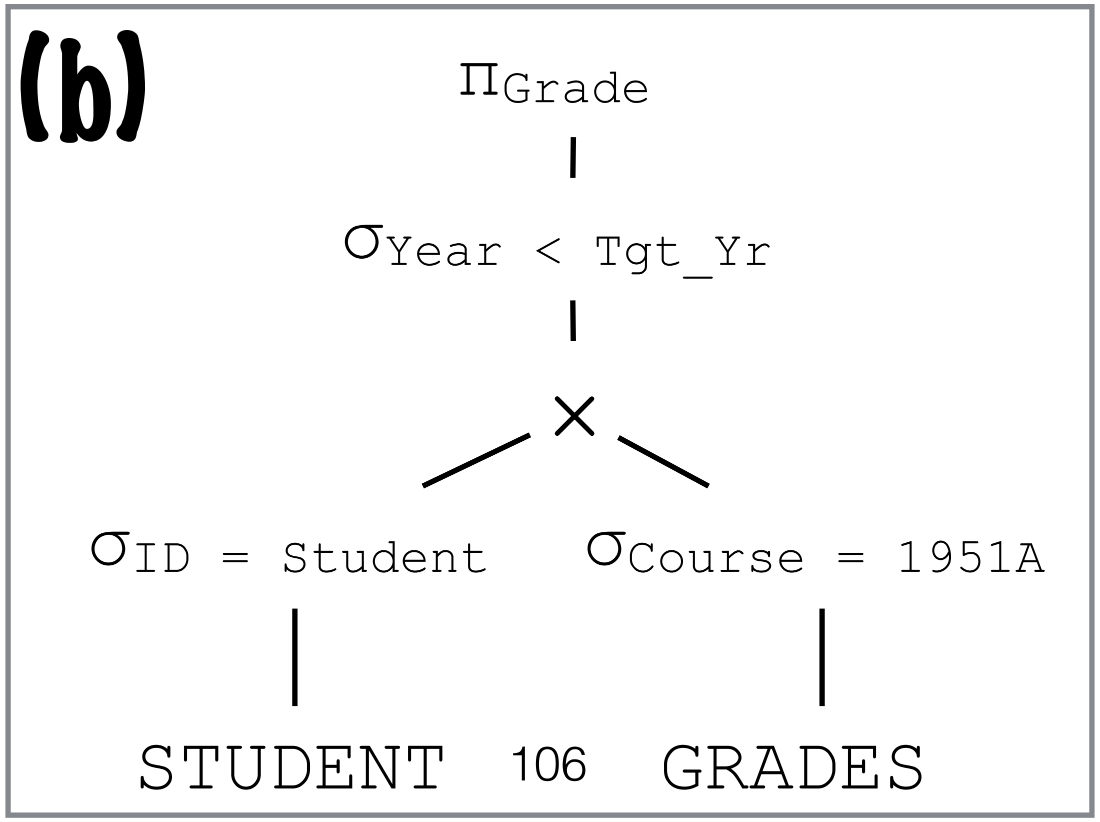
$\sigma_{ID\ =\ Student} \land \sigma_{Year\ <\ Tgt\_Yr}$

×

STUDENT $\sigma_{Course\ =\ 1951A}$

GRADES

(b)



π_{Grade}

$\sigma_{Year\ <\ Tgt\_Yr}$

×

$\sigma_{ID\ =\ Student}$ $\sigma_{Course\ =\ 1951A}$

STUDENT GRADES

Depends on output of join

Outline

- Catchup up from last lecture (more SQL keywords)
- NULLs
- Execution Order, Optimization
- Correlated Subqueries, More optimization

Nested Queries

STUDENT

ID	Name	Year
1	Wennie	4
2	Maulik	5
3	Gurnaa	5
4	Jens	4
5	Erin	4

GRADES

Studen	Cours	GPA	Tgt_Yr
1	32	4.0	1
2	1951A	3.5	3
6	32	2.8	1

```
SELECT s.Name
FROM STUDENT s
WHERE NOT EXISTS(
    SELECT *
    FROM GRADES
    WHERE s.ID = STUDENT.ID
  )
```

Find names students who are not in any classes.

Nested Queries

STUDENT

ID	Name	Year
1	Wennie	4
2	Maulik	5
3	Gurnaa	5
4	Jens	4
5	Erin	4

GRADES

Studen	Cours	GPA	Tgt_Yr
1	32	4.0	1
2	1951A	3.5	3
6	32	2.8	1

Outer Query

```
SELECT s.Name
FROM STUDENT s
WHERE NOT EXISTS(
    SELECT *
    FROM GRADES
    WHERE s.ID = STUDENT.ID
  )
```

Inner Query

Find names students who are not in any classes.

Nested Queries

STUDENT

ID	Name	Year
1	Wennie	4
2	Maulik	5
3	Gurnaa	5
4	Jens	4
5	Erin	4

GRADES

Studen	Cours	GPA	Tgt_Yr
1	32	4.0	1
2	1951A	3.5	3
6	32	2.8	1

Correlated! Inner query will return differently for every row...

```
SELECT s.Name
FROM STUDENT s
WHERE NOT EXISTS(
    SELECT *
    FROM GRADES
    WHERE s.ID = GRADES.Student
  )
```

Find names students who are not in any classes.

Nested Queries

STUDENT

ID	Name	Year
1	Wennie	4
2	Maulik	5
3	Gurnaa	5
4	Jens	4
5	Erin	4

GRADES

Studen	Cours	GPA	Tgt_Yr
1	32	4.0	1
2	1951A	3.5	3
6	32	2.8	1

Not correlated! Inner query will always return the same thing.

```
SELECT s.Name
FROM STUDENT s
WHERE s.ID NOT IN(
    SELECT Student
    FROM GRADES
 )
```

Find names students who are not in any classes.

Nested Queries

STUDENT

ID	Name	Year
1	Wennie	4
2	Maulik	5
3	Gurnaa	5
4	Jens	4
5	Erin	4

GRADES

Studen	Cours	GPA	Tgt_Yr
1	32	4.0	1
2	1951A	3.5	3
6	32	2.8	1

How many courses is each student taking?

```
SELECT s.ID, s.Name,
   (SELECT COUNT(*) as num_courses
    FROM GRADES g
    WHERE s.ID = g.Student)
FROM STUDENT s
```

Clicker Question!

STUDENT

ID	Name	Year
1	Wennie	4
2	Maulik	5
3	Gurnaa	5
4	Jens	4
5	Erin	4

GRADES

Studen	Cours	GPA	Tgt_Yr
1	32	4.0	1
2	1951A	3.5	3
6	32	2.8	1

How many courses is each student taking?

```
SELECT s.ID, s.Name,
   (SELECT COUNT(*) as num_courses
    FROM GRADES g
    WHERE s.ID = g.Student)
FROM STUDENT s
```

Is this query correlated?

(a) uh huh   (b) nuh uh

Clicker Question!

STUDENT

ID	Name	Year
1	Wennie	4
2	Maulik	5
3	Gurnaa	5
4	Jens	4
5	Erin	4

GRADES

Studen	Cours	GPA	Tgt_Yr
1	32	4.0	1
2	1951A	3.5	3
6	32	2.8	1

How many courses is each student taking?

```
SELECT s.ID, s.Name,
    (SELECT COUNT(*) as num_courses
     FROM GRADES g
     WHERE s.ID = g.Student)
FROM STUDENT s
```

Yes! This value will be different for every row (i.e. for every s.ID)

Is this query correlated?

[✓ (a) uh huh] (b) nuh uh

Nested Queries

STUDENT

ID	Name	Year
1	Wennie	4
2	Maulik	5
3	Gurnaa	5
4	Jens	4
5	Erin	4

GRADES

Studen	Cours	GPA	Tgt_Yr
1	32	4.0	1
2	1951A	3.5	3
6	32	2.8	1

How many courses is each student taking?

```
SELECT s.ID, s.Name, c.num_courses
FROM STUDENT s,
    (SELECT Student,
     COUNT(*) AS num_courses
     FROM GRADES
     GROUP BY Student) c
WHERE s.ID = c.Student
```

Clicker Question!

STUDENT

ID	Name	Year
1	Wennie	4
2	Maulik	5
3	Gurnaa	5
4	Jens	4
5	Erin	4

GRADES

Studen	Cours	GPA	Tgt_Yr
1	32	4.0	1
2	1951A	3.5	3
6	32	2.8	1

How many courses is each student taking?

```
SELECT s.ID, s.Name, c.num_courses
FROM STUDENT s,
    (SELECT Student,
     COUNT(*) AS num_courses
     FROM GRADES
     GROUP BY Student) c
WHERE s.ID = c.Student
```

Is this query correlated?

(a) yeah sure (b) not really

Clicker Question!

STUDENT

ID	Name	Year
1	Wennie	4
2	Maulik	5
3	Gurnaa	5
4	Jens	4
5	Erin	4

GRADES

Studen	Cours	GPA	Tgt_Yr
1	32	4.0	1
2	1951A	3.5	3
6	32	2.8	1

How many courses is each student taking?

```
SELECT s.ID, s.Name, c.num_courses
FROM STUDENT s,
    (SELECT Student,
     COUNT(*) AS num_courses
     FROM GRADES
     GROUP BY Student) c
WHERE s.ID = c.Student
```

This value is always the same, regardless of the row

Is this query correlated?

(a) yeah sure [✓ (b) not really]

Rewriting Queries

How many courses is each student taking?

```
SELECT s.ID, s.Name,
   (SELECT COUNT(*) as num_courses
    FROM GRADES g
    WHERE s.ID = g.Student)
FROM STUDENT s
```

Rewriting Queries

How many courses is each student taking?

```
SELECT s.ID, s.Name,
   (SELECT COUNT(*) as num_courses
    FROM GRADES g
    WHERE s.ID = g.Student)
FROM STUDENT s
```

Executed for every row

Rewriting Queries

How many courses is each student taking?

```
SELECT s.ID, s.Name,
   (SELECT COUNT(*) as num_courses
    FROM GRADES g
    WHERE s.ID = g.Student)
FROM STUDENT s
```

Only executed once

```
SELECT s.ID, s.Name, c.num_courses
FROM STUDENT s,
   (SELECT Student, COUNT(*) as num_courses
    FROM GRADES
    GROUP BY Student) c
WHERE s.ID = c.Student
```

(non)Clicker Question!
Rewrite to remove the subquery altogether?

STUDENT

ID	Name	Year
1	Wennie	4
2	Maulik	5
3	Gurnaa	5
4	Jens	4
5	Erin	4

GRADES

Studen	Cours	GPA	Tgt_Yr
1	32	4.0	1
2	1951A	3.5	3
6	32	2.8	1

```
SELECT s.Name
FROM STUDENT s
WHERE EXISTS(
    SELECT * FROM GRADES
    WHERE s.ID = GRADES.Student
    AND s.Year < GRADES.Tgt_Yr
)
```

Find students taking courses that are above their level.

(non)Clicker Question!
Rewrite to remove the subquery altogether?

STUDENT

ID	Name	Year
1	Wennie	4
2	Maulik	5
3	Gurnaa	5
4	Jens	4
5	Erin	4

GRADES

Studen	Cours	GPA	Tgt_Yr
1	32	4.0	1
2	1951A	3.5	3
6	32	2.8	1

```
SELECT s.Name
FROM STUDENT s
WHERE EXISTS(
    SELECT * FROM GRADES
    WHERE s.ID = GRADES.Student
    AND s.Year < GRADES.Tgt_Yr
)
```

HINT! Use a Join Condition

Find students taking courses that are above their level.

(non)Clicker Question!
Rewrite to remove the subquery altogether?

STUDENT

ID	Name	Year
1	Wennie	4
2	Maulik	5
3	Gurnaa	5
4	Jens	4
5	Erin	4

GRADES

Studen	Cours	GPA	Tgt_Yr
1	32	4.0	1
2	1951A	3.5	3
6	32	2.8	1

```
SELECT s.Name
FROM STUDENT s, GRADES g
WHERE s.ID = g.Student
    AND s.Year < g.Tgt_Yr
```

Find students taking courses that are above their level.