CS307

Database Principles

Chapter 6

Most contents are from Stéphane Faroult's slides

6.1 NULL and Logical Operators

Most contents are from Stéphane Faroult’s slides

Arithmetic Operators:

col+NULL –> NULL
col–NULL –> NULL
col*NULL –> NULL
col/NULL –> NULL
...

Remember we have TRUE, FALSE and NULL for logical operations.

Logical operators:

(col > NULL) –> NULL

(col = NULL) –> NULL

…

col is NULL –> True or False

Logical operators

TRUE and NULL –> NULL
FALSE and NULL –> FALSE

TRUE or NULL –> TRUE
FALSE or NULL –> NULL

Throw a NULL in, we have a condition that is never true but because of OR it can just be ignored.

```
col in ('a', 'b', null)
```

=

```
(col = 'a'
 or col = 'b'
 or col = null)
```

If col is 'a', the result is:
 TRUE or FALSE or NULL -> TRUE

if col is 'c', the result is:

 FALSE or FALSE or NULL -> NULL

if col is NULL, the result is :
 NULL or NULL or NULL -> NULL

```
col not in
      ('a', 'b', null)
```

=

```
(col <> 'a'
 and col <> 'b'
 and col <> null)
```

If col is 'a', the result is:
 TRUE and FALSE and NULL ->
FALSE

if col is 'c', the result is:

 TRUE and TRUE and NULL -> NULL

if col is NULL, the result is :
 NULL and NULL and NULL -> NULL

6.2 Ordering

Most contents are from Stéphane FarouIt's slides

order by

There is one simple expression in SQL to order a result set, which is ORDER BY. It comes at the end of a query (although you can have it in subqueries, as you'll see). It is followed by the list of columns used as sort columns.

This will return all movies, starting with the oldest one.

```
select title, year_released
from movies
order by year_released
```

Sorts the result of the query

table unchanged

```
select title, year_released
from movies
where country = 'us'
order by year_released
```

We can apply it to any result set ...

```
select m.title,
       m.year_released
from movies m
where m.movieid in
  (select distinct c.movieid
   from credits c
        inner join people p
        on p.peopleid = c.peopleid
   where c.credited_as = 'A'
     and p.birth_year >= 1970)
order by m.year_released
```

… no matter how complicated the query.

```
select c.country_name,
       m.title,
       m.year_released
from movies m
     inner join countries c
     on c.country_code = m.country
where m.movieid in
  (select distinct c.movieid
   from credits c
        inner join people p
        on p.peopleid = c.peopleid
   where c.credited_as = 'A'
     and p.birth_year >= 1970)
order by m.year_released
```

and with joins you can sort by any column of any table in the join (remember the super wide table with all the columns from all tables involved)

```
order by col1 desc, col2 asc, ...
```

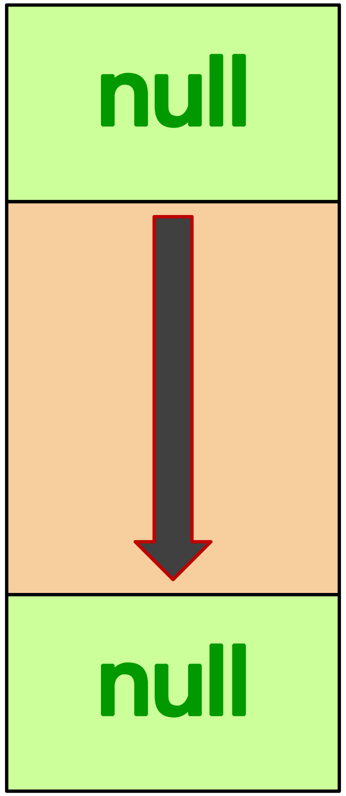
You can specify that a sort is descending by following the column name with DESC. You can also use ASC to say ascending, but as it's the default nobody uses it.

```
select c.country_name,                1
     m.title,                         2
     m.year_released                  3
from movies m
    inner join countries c
    on c.country_code = m.country
where m.movieid in
  (select distinct c.movieid
   from credits c
       inner join people p
       on p.peopleid = c.peopleid
   where c.credited_as = 'A'
    and p.birth_year >= 1970)
order by c.country_name,
        m.year_released desc, m.title
```

ordering depends on the data type

Remember that strings are sorted alphabetically, numbers numerically and dates and times chronologically. What happens when data is missing?

NULLs ?



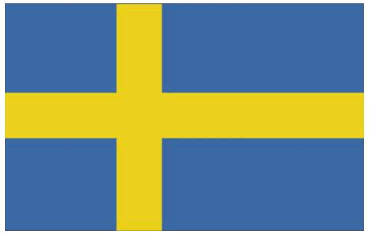
null

null

It depends on the DBMS. SQL Server, MySQL and SQLite consider by default that nothing is smaller than everything, and DB2, Oracle and PostgreSQL that it's greater than anything.

Don't believe that things are simple with text, either. They are relatively simple in English, as long as you don't use a foreign word with an accent such as attaché.

In this case, you would probably think that é should sort with e (so do I), but that's not necessarily what internal encoding says. Besides, local habits may vary. Swedes think that ö should come after z. German speakers rather see it with o (Swedish is the default language for MySQL)



ö

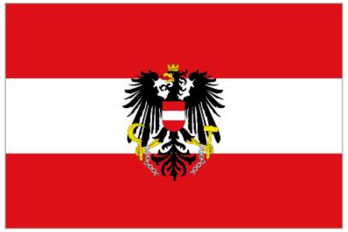
a

b

…

y

z



ö

Local text sorting rules are known as "collations". Some products allow you to specify how data in a column should be sorted when you create the table. It's also sometimes possible to specify how you want data to be sorted when you do it.

Collation

PostgreSQL

Microsoft SQL Server

MySQL

```
create table ... (
        some_text_column varchar(100)
          collate <collation name> not null,
        ...)

order by nls_sort(some_text_column,
                  '<collation name>')
```

ORACLE

How are Chinese text strings sorted?

How many collations can you choose for Chinese?

I've told you that usually dates are converted to a user-friendly format when returned, for instance with TO_CHAR() available in several products.

```
select to_char(a_date_column, 'MM/DD/YYYY')
                          as event_date, ...
from ...
where ...
order by event_date
```

No!

But if you sort by this column (text) the sort will be alphabetical! You should sort by the original, date column:

```
order by a_date_column
```

You can sort by a column that isn't returned.

movies

movied	title	country	year_released
1832	Gone With The Wind	us	1939

For instance suppose that we add producers to the movie database (credited_as = 'P')

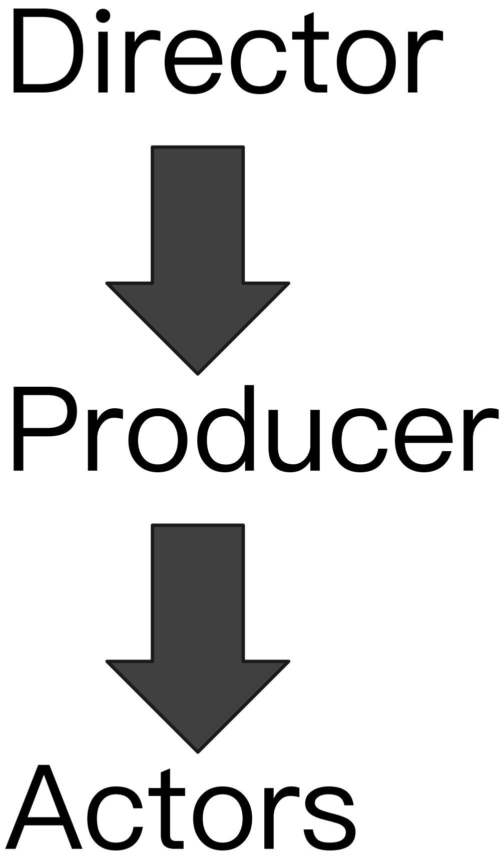
credits

movied	peopleid	credited_as
1832	237	A
1832	312	A
1832	742	P
1832	128	D

people

peopleid	first_name	surname	born	died
237	Clark	Gable	1901	1960
742	David	Selznick	1902	1965
312	Vivien	Leigh	1913	1967
128	Victor	Fleming	1889	1949

If we want to sort people by function first, with the director first, producer second and actors last ...



Director

Producer

Actors

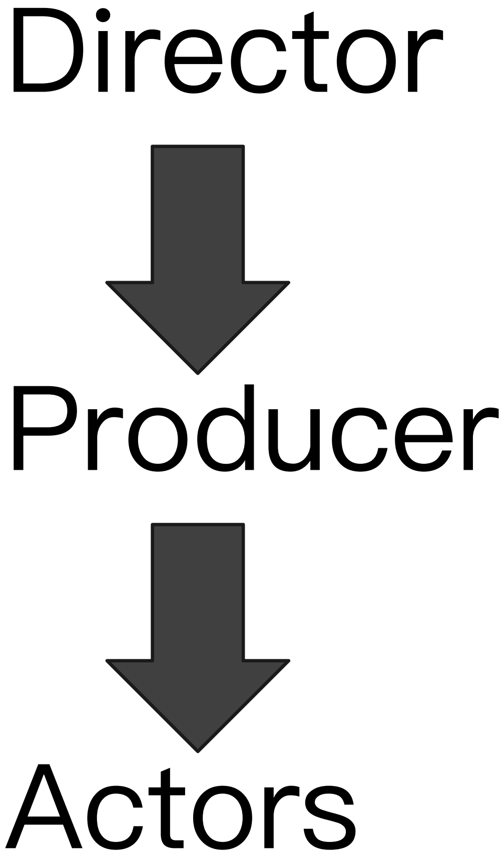
Actors

Director

Producer

... no matter whether CREDITED_AS is ascending or descending, sorting by it won't work.

order by credited_as desc


Director

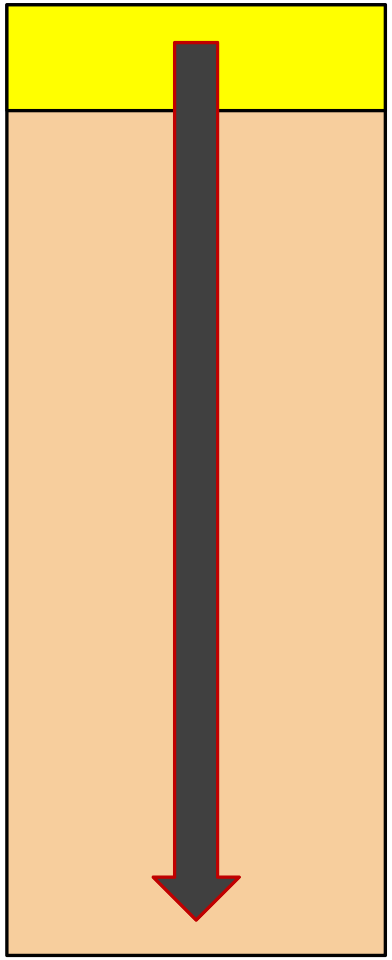
Producer

Actors

```
order by
  case credited_as
    when 'D' then 1
    when 'P' then 2
    when 'A' then 3
  end
```

The solution is to use CASE ... END to replace each code with a value that sorts as intended. This is frequently used for "custom sorts".

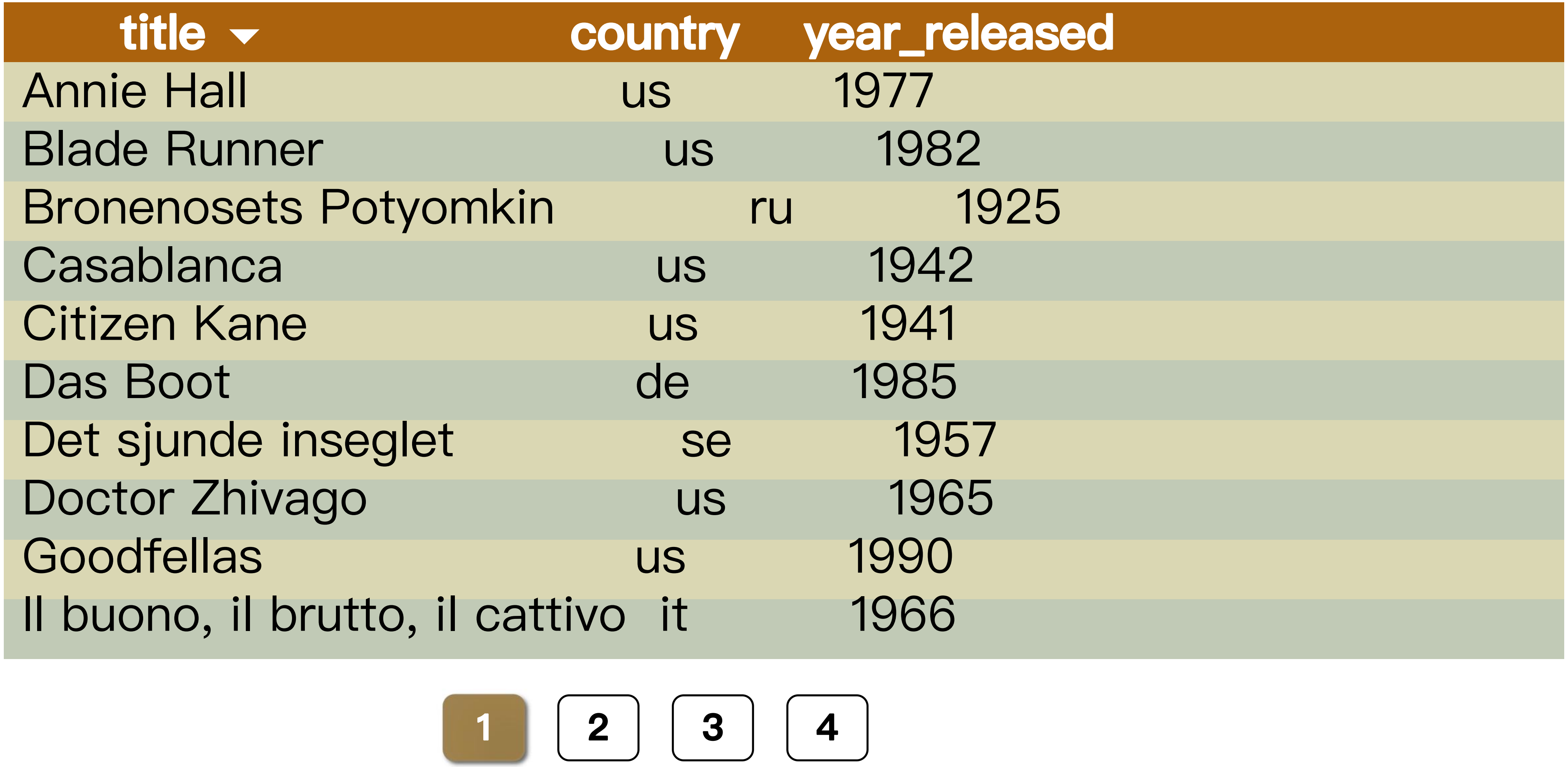
Three oldest ?



order by year_released

Another problem that isn't so easy is displaying only a limited number of oldest values, or successive "slices" in a long sorted list.

Top 10



title ▾	country	year_released
Annie Hall	us	1977
Blade Runner	us	1982
Bronenosets Potyomkin	ru	1925
Casablanca	us	1942
Citizen Kane	us	1941
Das Boot	de	1985
Det sjunde inseglet	se	1957
Doctor Zhivago	us	1965
Goodfellas	us	1990
Il buono, il brutto, il cattivo	it	1966

Successive pages are common on websites. Here titles are sorted.

Skip 10, Top 10

title ▾	country	year_released
Inglourious Basterds	us	2009
Jaws	us	1975
La Belle et la Bête	fr	1946
Ladri di biciclette	it	1948
Lawrence of Arabia	gb	1962
Le cinquième élément	fr	1997
Les Visiteurs du Soir	fr	1942
Mary Poppins	us	1964
On The Waterfront	us	1954
Pather Panchali	in	1955

1 2 3 4

First Page

```
select title,
       country,
       year_released
from movies
order by title
limit 10
```

PostgreSQL

MySQL

SQLite

Several products implement a LIMIT clause that is executed AFTER the sort; this syntax seems to be gaining in popularity.

First Page

```
select title,
       country,
       year_released
from movies
order by title
fetch first 10 rows only
```

DB2 has something slightly different, which was also (more recently) adopted by Oracle and Postgres.

First Page

```
select top 10
        title,
        country,
        year_released
from movies
order by title
```

SQL Server is frankly different, but the logic is the same: you sort, then discard everything but what you want.

Third Page

```
select title,
       country,
       year_released
from movies
order by title
limit 10      offset 20
```

Retrieving rows 20 to 30 in a sorted result is easy with PostgreSQL, MySQL and SQLite.

Third Page

```
select title,
       country,
       year_released
from movies
order by title
offset 20
fetch first 10 rows only
```

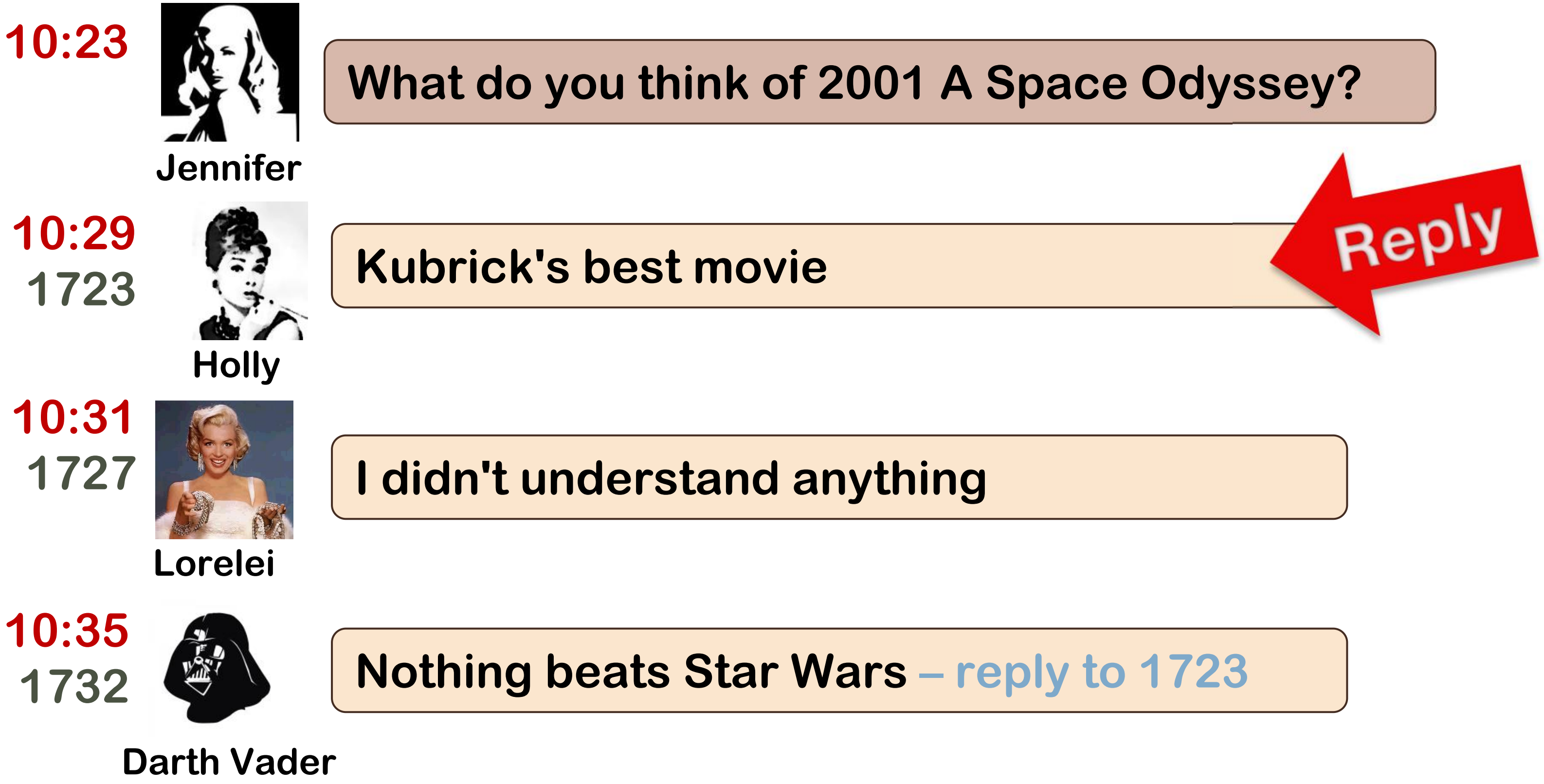
IBM DB2

ORACLE

PostgreSQL

What when order is a bit more subtle?

There are many cases when plain ordering isn't satisfying.



10:23 Jennifer

What do you think of 2001 A Space Odyssey?

10:29 1723 Holly

Kubrick's best movie

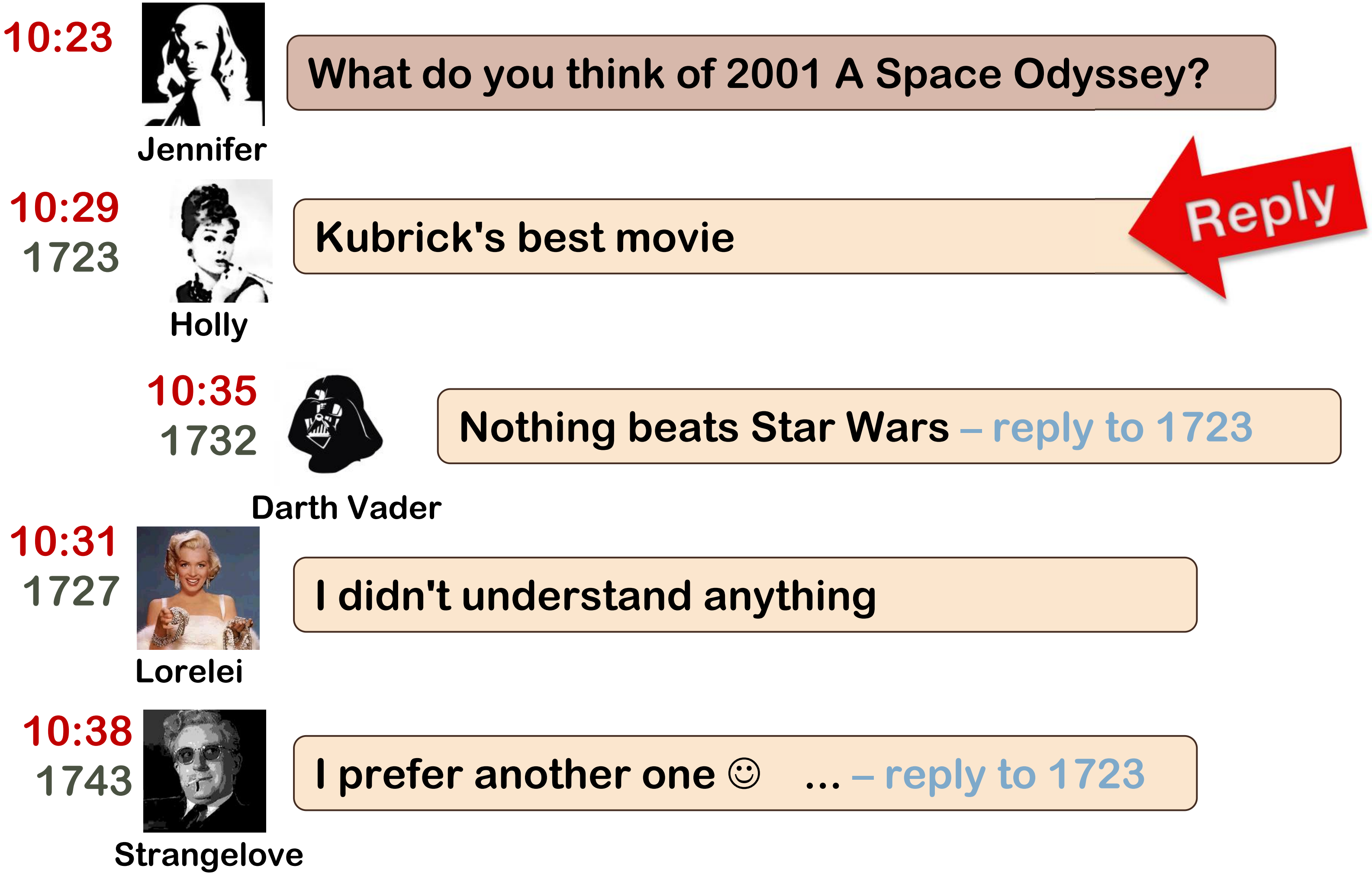
10:31 1727 Lorelei

I didn't understand anything

10:35 1732 Darth Vader

Nothing beats Star Wars – reply to 1723

Such a case is a forum. Somebody posts a topic, then people post their comments in sequence. Things turn ugly when somebody starts posting an answer to a comment rather than to the original topic. Some forums always keep a sequential order and force users to add say @Holly or @1723 (the post id) to help others understand what they are reacting at.



10:23 Jennifer
What do you think of 2001 A Space Odyssey?

10:29 1723 Holly
Kubrick's best movie

10:35 1732 Darth Vader
Nothing beats Star Wars – reply to 1723

10:31 1727 Lorelei
I didn't understand anything

10:38 1743 Strangelove
I prefer another one ☺ ... – reply to 1723

A better solution (for visitors, not developers) is to maintain "threads"

10:23

Jennifer

What do you think of 2001 A Space Odyssey?

10:29
1723

Holly

Kubrick's best movie

10:35
1732

Darth Vader

Nothing beats Star Wars – reply to 1723

Reply

10:38
1743

Strangelove

I prefer another one ☺ ... – reply to 1723

10:31
1727

Lorelei

I didn't understand anything

10:36
1733

Harry Lime

Are you kidding? – reply to 1732

But threads can develop into complicated hierarchies.

10:23

Jennifer

What do you think of 2001 A Space Odyssey?

10:29
1723

Holly

Kubrick's best movie

10:35
1732

Darth Vader

Nothing beats Star Wars – reply to 1723

Reply

10:36
1733

Harry Lime

Are you kidding? – reply to 1732

10:38
1743

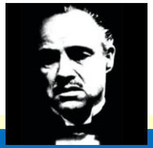
Strangelove

I prefer another one ☺ ... – reply to 1723

10:31
1727

Lorelei

I didn't understand anything

10:40
1747

Vito

Darth, you'll stop trolling if I ask you gently.
– reply to 1732

Nobody's perfect, and the area where SQL database management systems struggle a bit is the management of hierarchies (sometimes referred to as the BOM problem – Bill Of Materials). This is something you encounter everywhere you have to deal with items that can be divided in subitems that can also be subdivided and indefinite number of times. A few example:

* Cars, made of components that can themselves have subcomponents

* Chemistry. Ingredients rarely are "pure" ingredients but already the result of chemical processes

* Financial participations. You can have parts in two companies, one of which also has parts in the other (also known as "financial exposure")

10:23

Jennifer

What do you think of 2001 A Space Odyssey?

10:29
1723

Holly

NULL

order by concat(coalesce(path, ''),
<formated id>)

10:35
1732

Darth Vader

000001723

10:36
1733

Harry Lime

000001723000001732

10:40
1747

Vito

000001723000001732

10:38
1743

Strangelove

000001723

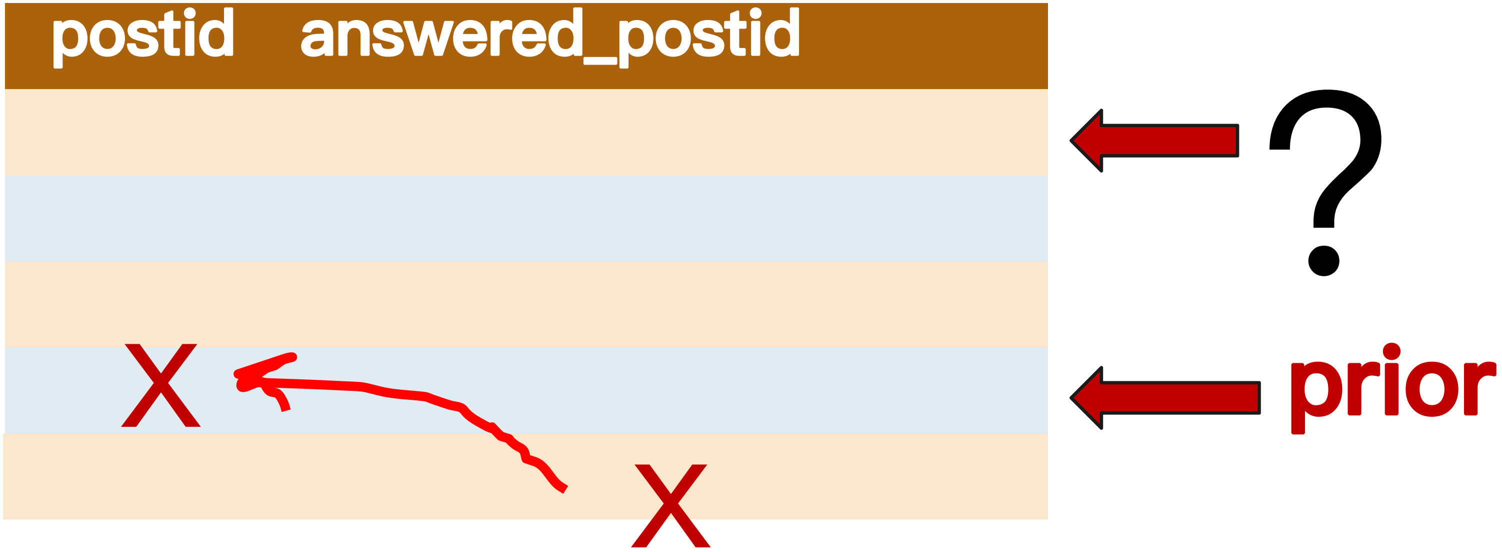
10:31
1727

Lorelei

NULL

One way to try to solve the problem is the "materialized path", turning the "ancestry" into an attribute.

```
select message, ....
from forum_posts ...
connect by answered_postid = prior postid
start with answered_postid is null
       and topicid = ...
order siblings by postid
```



postid answered_postid

?

prior

X

X

Oracle has long (since the first half of the 1980s) implemented a way to refer to a 'prior row' in a kind of "dynamic ordering"

Most products handle hierarchies through recursive queries, that we'll see in some detail next time.

6.3 Window Functions

Most contents are from Stéphane Faroult’s slides

Another very important (but not available in MySQL before 8 or SQLite) set of functions for ordering/reporting are window functions. They bear different names, Oracle calls them analytic functions, DB2 calls them OLAP (OnLine Analytical Processing) functions. They are of two kinds, we'll start with non-ranking functions.

Non-ranking

Window Functions

Ranking

We have seen so far two categories of functions: functions that operate on values in the current row (called scalar functions), and aggregate functions, that operate on sets of rows.

Year of oldest movie per country?

```
select country,
       min(year_released) earliest_year
from movies
group by country
```

The problem with aggregate functions is that details just vanish. If I ask for the year of the oldest movie per country, I get a country, a year, and nothing else.

TITLE and year of the earliest movie per country

If I want some detail, for instance which was the title of this oldest movie, the only option with aggregate functions is to join their output to the very same table that has been aggregated to retrieve the lost detail.

For instance, by joining with movies I can retrieve the title(s) of the movie(s) released in this country that year. Intuitively, we feel that we visited MOVIES twice and that perhaps we could have done better.

```
select a.country,
       a.title,
       a.year_released
from movies a
     inner join
     (select country,
        min(year_released) minyear
        from movies
        group by country) b
  on b.country = a.country
  and b.minyear = a.year_released
```

Window functions hold the middle-ground between scalar and aggregate functions. Like scalar functions, they return a result for a single row; but like aggregate functions, this result is computed out of several rows. The syntax is as follows

```
func(parameters) over (magic clause)
```

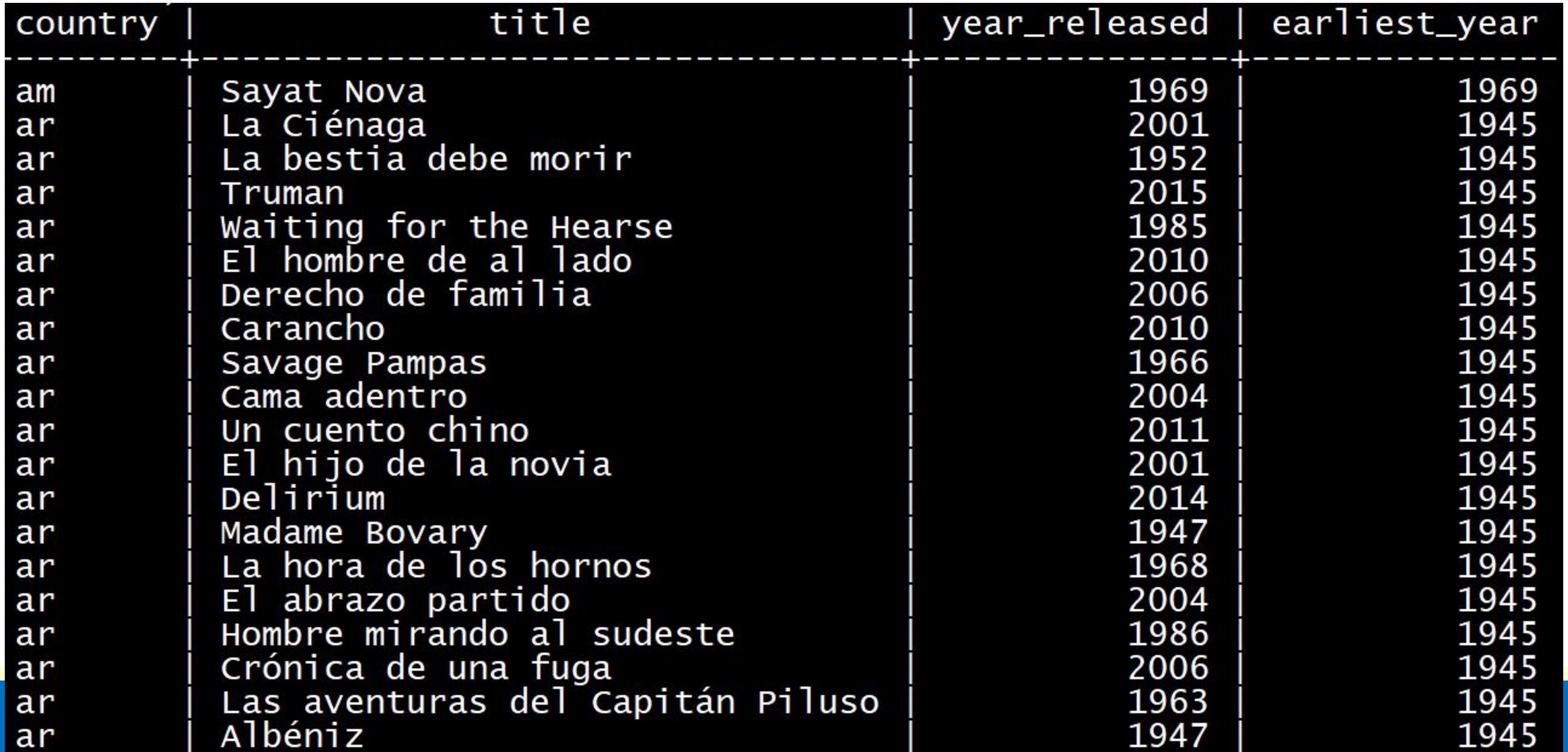
With DBMS products that support window functions, every aggregate function can be used as a window function. Instead of specifying with GROUP BY the subset on which the result is computed, you say OVER (PARTITION BY ...)

```
min(year_released)
    over (partition by country)
```

```
select country,
       title,
       year_released,
       min(year_released)
       over (partition by country)
              earliest_year
from movies
```

Thus, this query returns two years for every movie: the one when this particular movie was released, and the one when the earliest movie for the same country was released. You get both detail and an aggregate value on the same row.

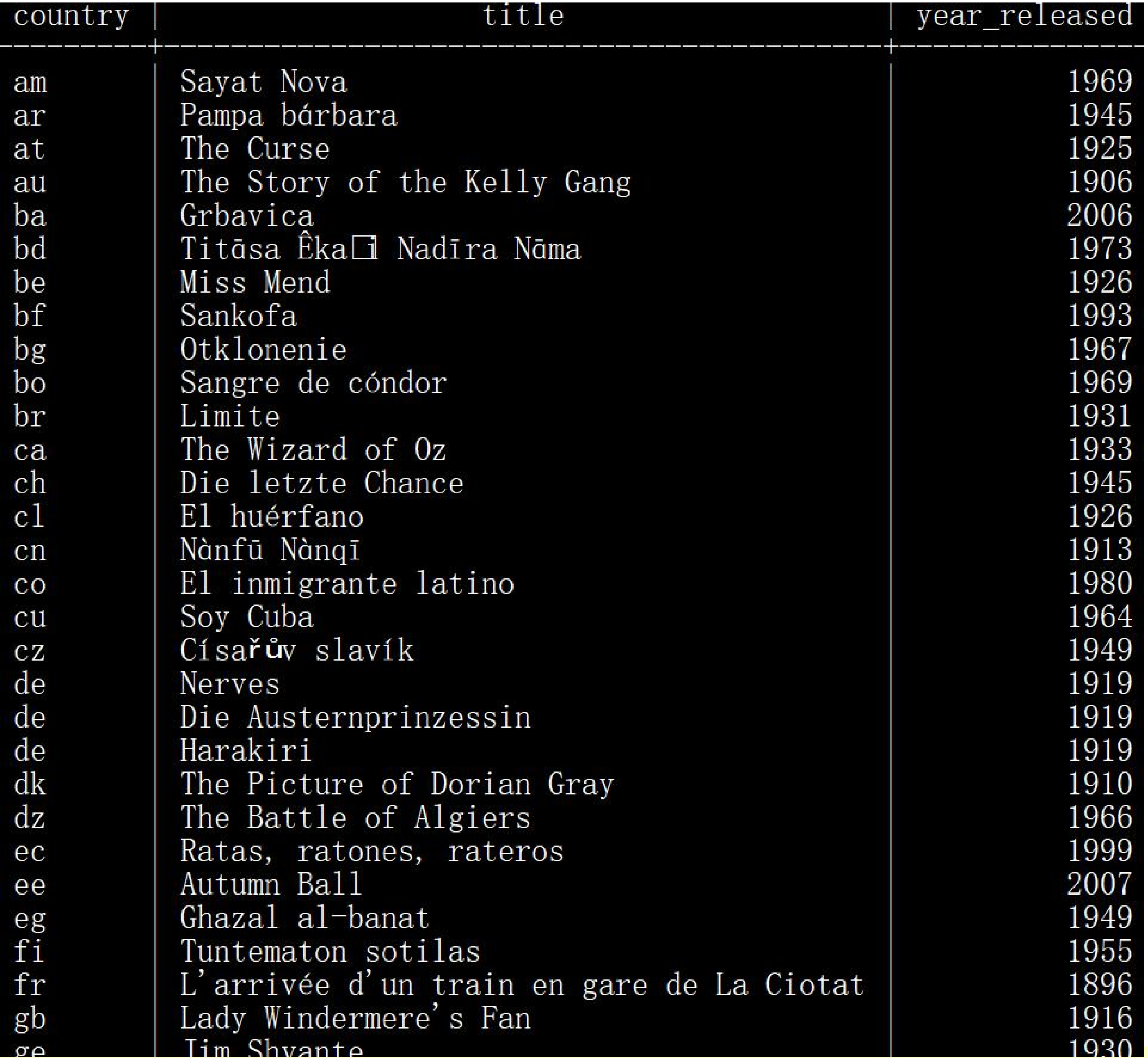
TITLE and year of the earliest movie per country



```
country |                title                 | year_released | earliest_year
--------+--------------------------------------+---------------+--------------
am      | Sayat Nova                           |          1969 |          1969
ar      | La Ciénaga                           |          2001 |          1945
ar      | La bestia debe morir                 |          1952 |          1945
ar      | Truman                               |          2015 |          1945
ar      | Waiting for the Hearse               |          1985 |          1945
ar      | El hombre de al lado                 |          2010 |          1945
ar      | Derecho de familia                   |          2006 |          1945
ar      | Carancho                             |          2010 |          1945
ar      | Savage Pampas                        |          1966 |          1945
ar      | Cama adentro                         |          2004 |          1945
ar      | Un cuento chino                      |          2011 |          1945
ar      | El hijo de la novia                  |          2001 |          1945
ar      | Delirium                             |          2014 |          1945
ar      | Madame Bovary                        |          1947 |          1945
ar      | La hora de los hornos                |          1968 |          1945
ar      | El abrazo partido                    |          2004 |          1945
ar      | Hombre mirando al sudeste            |          1986 |          1945
ar      | Crónica de una fuga                  |          2006 |          1945
ar      | Las aventuras del Capitán Piluso     |          1963 |          1945
ar      | Albéniz                              |          1947 |          1945
```

```
select m.country,
       m.title,
       m.year_released
from (select country,
             title,
             year_released,
             min(year_released)
              over (partition by country)
                    earliest_year
      from movies) m
where m.year_released = m.earliest_year
```

You just need to limit output to those movies for which the year of release happens to be the same as the earliest one for their country.



```
 country  |                   title                    | year_released
----------+--------------------------------------------+---------------
 am       |  Sayat Nova                                |          1969
 ar       |  Pampa bárbara                             |          1945
 at       |  The Curse                                 |          1925
 au       |  The Story of the Kelly Gang               |          1906
 ba       |  Grbavica                                  |          2006
 bd       |  Titāsa Êka□i Nadīra Nāma                  |          1973
 be       |  Miss Mend                                 |          1926
 bf       |  Sankofa                                   |          1993
 bg       |  Otklonenie                                |          1967
 bo       |  Sangre de cóndor                          |          1969
 br       |  Limite                                    |          1931
 ca       |  The Wizard of Oz                          |          1933
 ch       |  Die letzte Chance                         |          1945
 cl       |  El huérfano                               |          1926
 cn       |  Nànfū Nànqī                               |          1913
 co       |  El inmigrante latino                      |          1980
 cu       |  Soy Cuba                                  |          1964
 cz       |  Císařův slavík                            |          1949
 de       |  Nerves                                    |          1919
 de       |  Die Austernprinzessin                     |          1919
 de       |  Harakiri                                  |          1919
 dk       |  The Picture of Dorian Gray                |          1910
 dz       |  The Battle of Algiers                     |          1966
 ec       |  Ratas, ratones, rateros                   |          1999
 ee       |  Autumn Ball                               |          2007
 eg       |  Ghazal al-banat                           |          1949
 fi       |  Tuntematon sotilas                        |          1955
 fr       |  L’arrivée d’un train en gare de La Ciotat |          1896
 gb       |  Lady Windermere’s Fan                     |          1916
 ge       |  Jim Shvante                               |          1930
```

Oldest movie you like least?

(country with several movies)

```
select m.country, m.title,
     m.year_released
from
  (select country,
       title,
       year_released,
       min(year_released)
       over (partition by country)
                earliest_year
    from movies
    where title <> 'A title here') m
where m.year_released = m.earliest_year
```

If you filter out, with a WHERE condition, one movie, it will be excluded from the window function computation. The earliest year may become the second earliest.

Window functions always operate against rows that belong to a result set. One related characteristics is that they can only appear after the SELECT, not in the WHERE clause, and there is nothing with them similar to HAVING with aggregate functions (it's not a real limitation; you can always work around it by wrapping the query into another one that applies conditions to its output, as shown previously)

Reporting function

SELECTED rows

```
select a.country, a.title, a.year_released
from movies a
     inner join
    (select country,
            min(year_released) earliest_year
     from movies
     where title <> 'A title here'
     group by country) b
        on b.country = a.country
       and b.earliest_year = a.year_released
```

We have seen the functional equivalence with GROUP BY + join, the previous example works like what is above, with the minimum computed on everything but one movie.

```
select m.country, m.title,
         m.year_released
from
  (select country,
             title,
             year_released,
             min(year_released)
             over (partition by country)
                          earliest_year
        from movies) m
where m.year_released = m.earliest_year
  and title <> 'A title here'
```

If the query is nested, then the minimum is computed over everything, then filtered out. One country may disappear out of the picture.

```
select a.country, a.title, a.year_released
from movies a
    inner join
   (select country,
          min(year_released) earliest_year
    from movies
    group by country) b
       on b.country = a.country
      and b.earliest_year = a.year_released
where a.title <> 'A title here'
```

This query is functionally equivalent to the previous one.

min(year_released) over()

In the same way that you can have an aggregate function without a GROUP BY when you want ONE result for the whole table, you can have an empty OVER clause to indicate that you want the result computed over all rows selected. Note that OVER () is still mandatory otherwise the function would be interpreted as a regular aggregate function, not as a window function.

```
select country_name,
       cnt as number_of_movies,
       round(100 * cnt / sum(cnt) over (), 0)
                as percentage
from (   select c.country_name,
                coalesce(m.cnt, 0) cnt
         from countries c
              left outer join (select country,
                                      count(*) cnt
                               from movies
                               group by country) m
                    on m.country = c.country_code
                                          ) q
order by country_name
```

This is frequently used in operations such as computing a value as a percentage of the total.

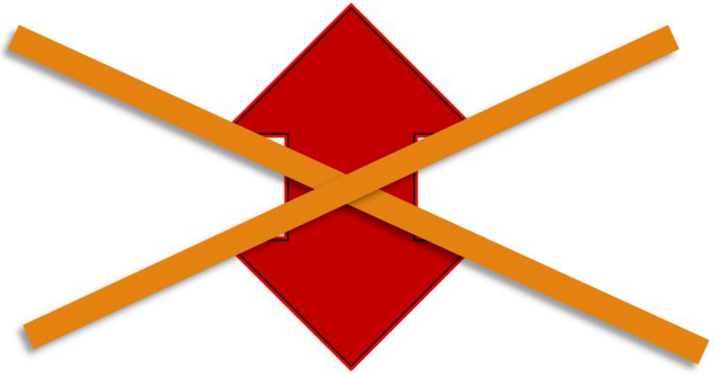
Side note: when there is an ORDER BY you cannot start returning rows before you have seen all of them - so you may count them too when sorting, and the marginal cost of the window function is near zero.

```
select country_name,
     cnt as number_of_movies,
     round(100 * cnt / t.movie_count, 0) percentage
from (   select c.country_name,
              coalesce(m.cnt, 0) cnt
         from countries c
             left outer join (select country,
                                     count(*) cnt
                                from movies
                                group by country) m
                  on m.country = c.country_code
                                   ) q
    cross join (select count(*) movie_count
            from movies) t
order by country_name
```

The same thing can be obtained with a type of join we haven't seen yet, a CROSS JOIN (without any join condition, also called a Cartesian join)

If all aggregate functions can be used as window functions, there are also some window functions that provide ranking capabilities. These functions are original functions and unrelated to either aggregate functions or scalar functions. There are a few of them, we'll only discuss the most important ones.

ranking window function



aggregate function

func() over $\underbrace{(...)}_{}$

When we talk about "ranking", of course, we implicitly talk about "ordering". In the same way as we can put into the OVER clause how we group, we can also say there how we order.

There are three main ranking functions. In many cases, they return identical values. Differences are interesting.

row_number()

rank()

dense_rank()

With a ranking window function you MUST have an ORDER BY clause in the OVER() (you cannot have an empty OVER() clause). You can combine it with a PARTITION BY to order with groups.

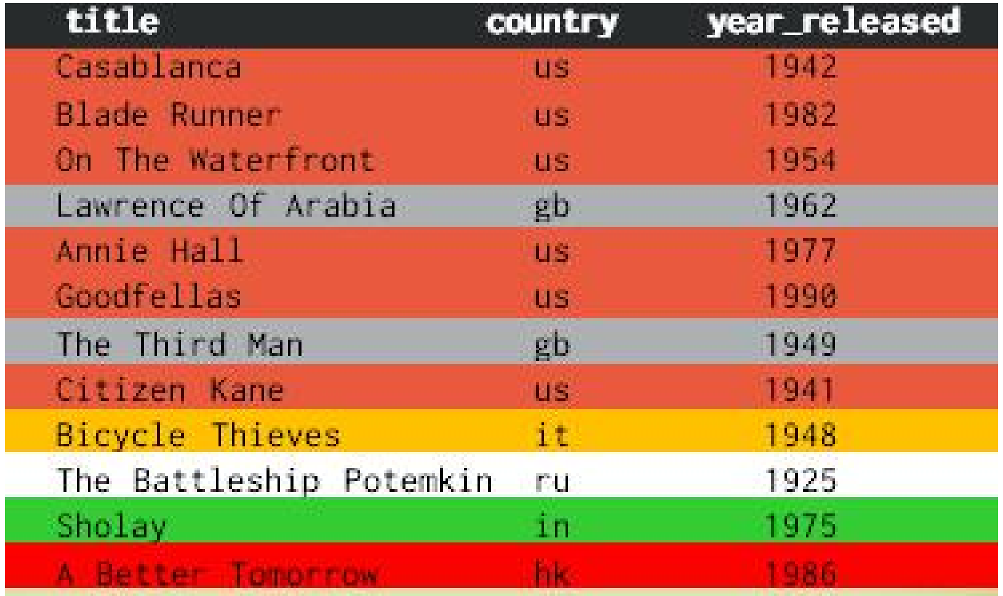
```
over (order by ...)

over (partition by ...
        order by ...)
```

id	title	country	year_released
1	Casablanca	us	1942
2	Blade Runner	us	1982
3	On The Waterfront	us	1954
4	Lawrence Of Arabia	gb	1962
5	Annie Hall	us	1977
6	Goodfellas	us	1990
7	The Third Man	gb	1949
8	Citizen Kane	us	1941
9	Bicycle Thieves	it	1948
10	The Battleship Potemkin	ru	1925
11	Sholay	in	1975
12	A Better Tomorrow	hk	1986

id	title	country	year_released	
1	Casablanca	us	1942	5
2	Blade Runner	us	1982	2
3	On The Waterfront	us	1954	4
4	Lawrence Of Arabia	gb	1962	1
5	Annie Hall	us	1977	3
6	Goodfellas	us	1990	1
7	The Third Man	gb	1949	2
8	Citizen Kane	us	1941	6
9	Bicycle Thieves	it	1948	1
10	The Battleship Potemkin	ru	1925	1
11	Sholay	in	1975	1
12	A Better Tomorrow	hk	1986	1

```
select title,
       country,
       year_released,
       row_number()
        over (      partition by country
                    order by year_released desc
                    ) rn
from movies
```

In this example, movies are grouped by country, and a sequential number is assigned by country to each movie, starting with the most recent movie.

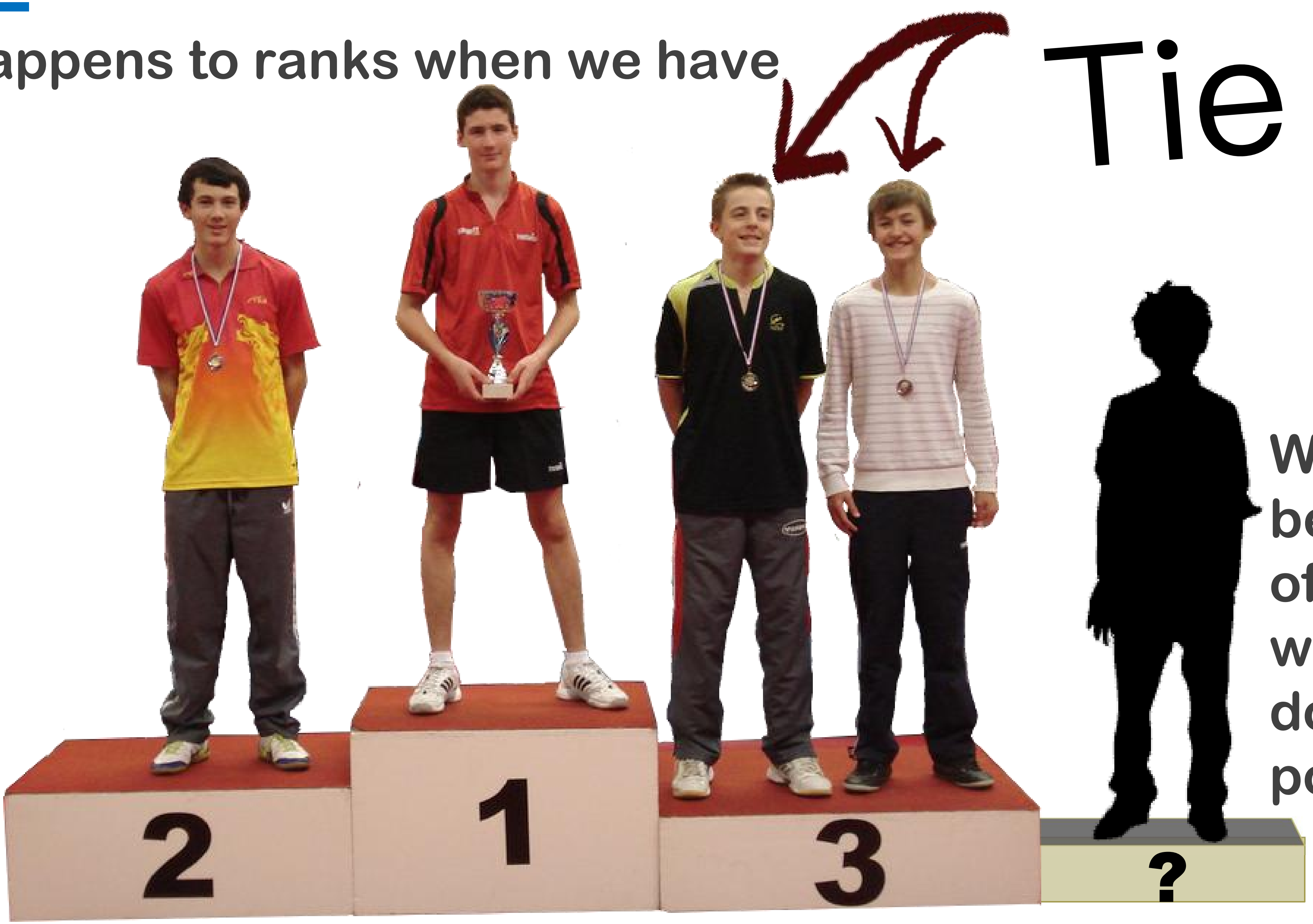


title	country	year_released
Casablanca	us	1942
Blade Runner	us	1982
On The Waterfront	us	1954
Lawrence Of Arabia	gb	1962
Annie Hall	us	1977
Goodfellas	us	1990
The Third Man	gb	1949
Citizen Kane	us	1941
Bicycle Thieves	it	1948
The Battleship Potemkin	ru	1925
Sholay	in	1975
A Better Tomorrow	hk	1986

over (partition by *col1, col2,* ...
order by *col3, col4,* ...)

As with plain GROUP BY and plain ORDER BY, both partitioning and ordering can be applied to several columns (and "columns" can of course be expressions).

What happens to ranks when we have ties?

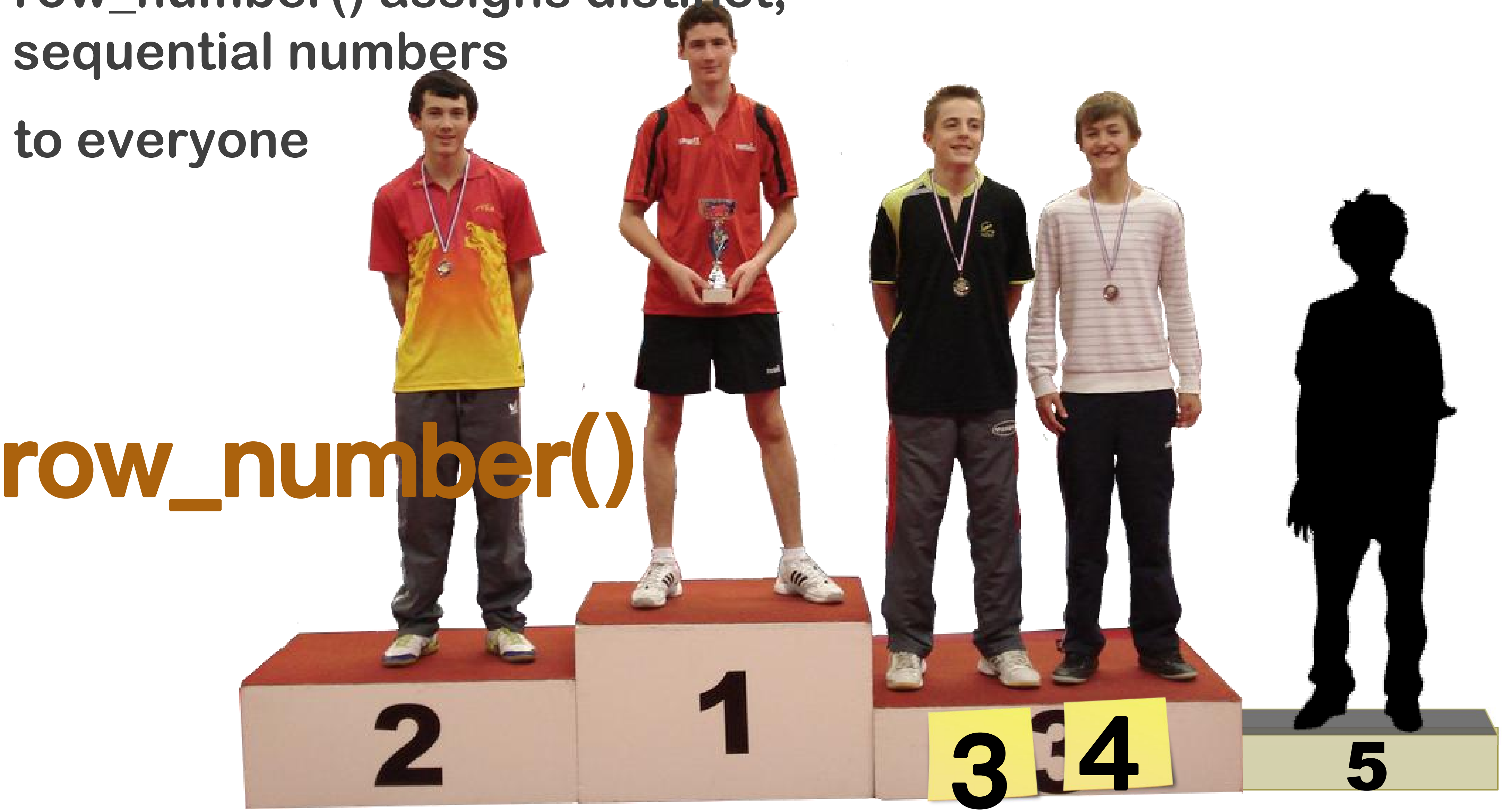


What will be the rank of the one who didn't do it to the podium?

Flickr: Vaillante Angers Tennis de Table

row_number() assigns distinct, sequential numbers

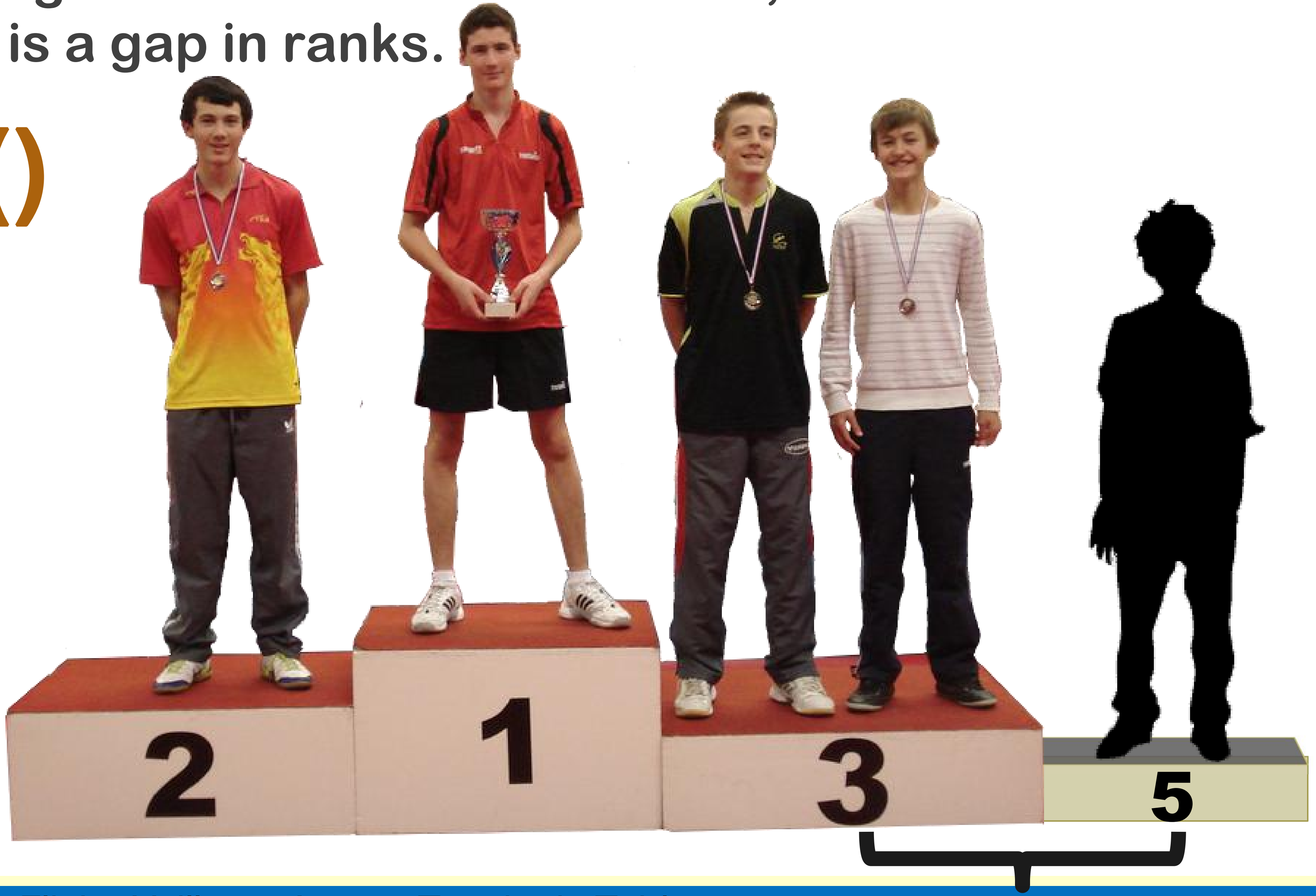
to everyone



Flickr: Vaillante Angers Tennis de Table

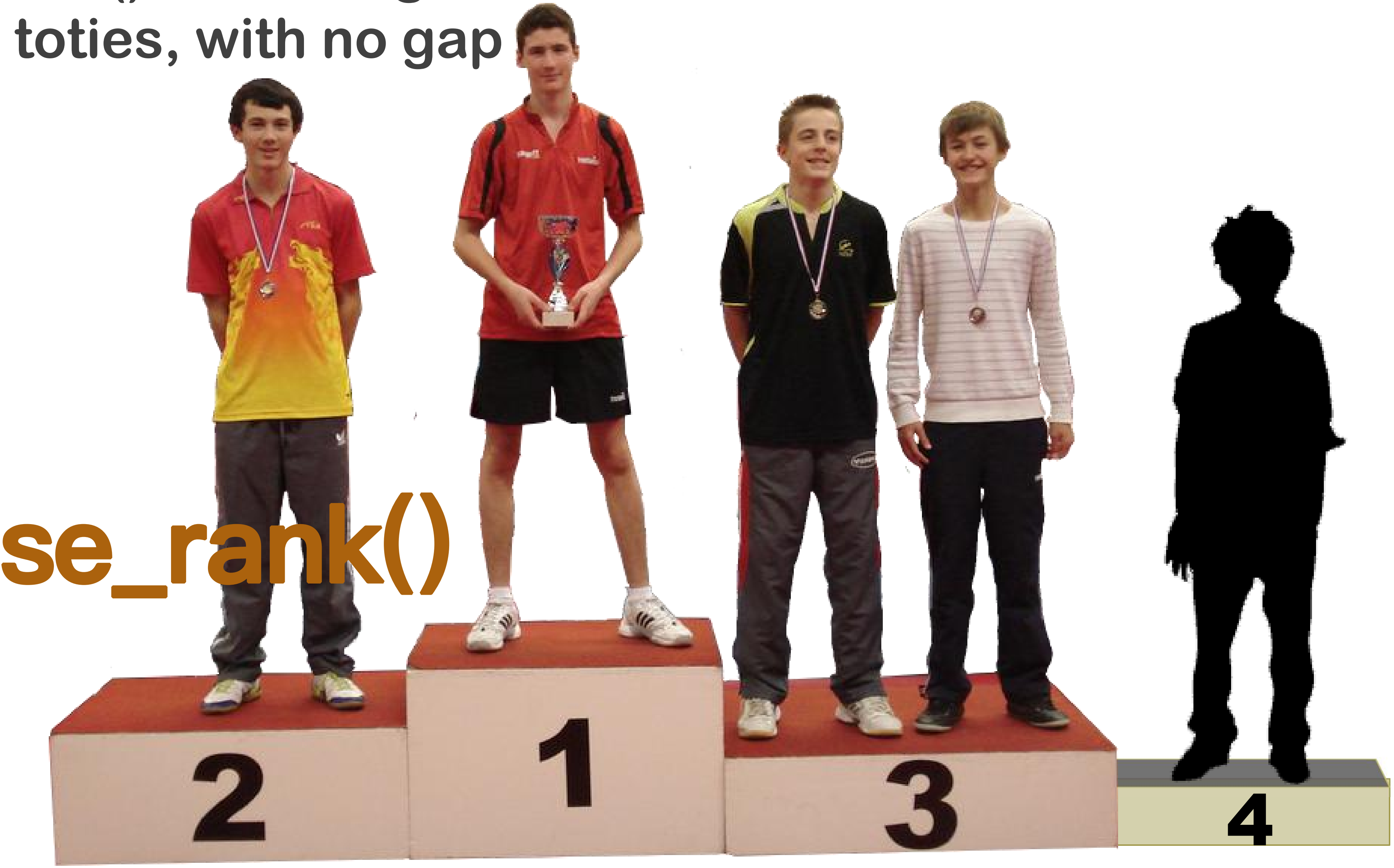
rank() assigns the same number to ties, but there is a gap in ranks.

rank()



Flickr: Vaillante Angers Tennis de Table

dense_rank() also assigns the same number toties, with no gap



Flickr: Vaillante Angers Tennis de Table

Which are the **two most recent** movies for each country?

As an aside, a condition on ROW_NUMBER() works a bit like LIMIT applied to ORDER BY, except that ordering can be by group.

Ranking window functions allow answering easily some really tough questions which are almost impossible to answer efficiently otherwise.

```
select x.country,
       x.title,
       x.year_released
from
(select country,
        title,
        year_released,
        row_number()
        over (partition by country
              order by year_released desc) rn
 from movies
        ) x
where x.rn <= 2
```