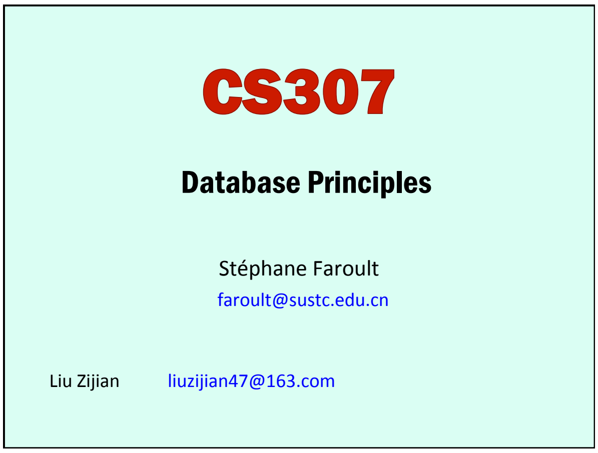
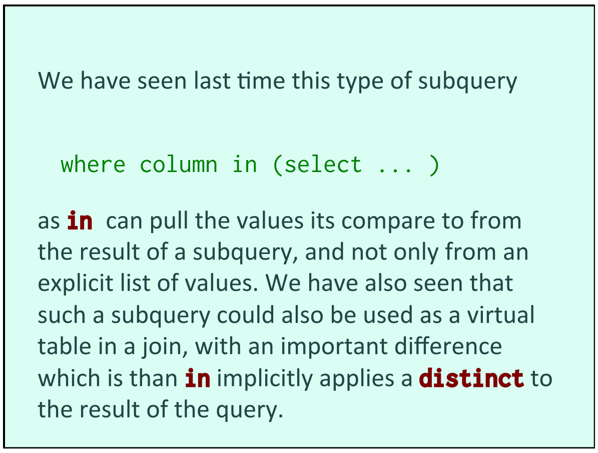


CS307

Database Principles

Stéphane Faroult
faroult@sustc.edu.cn

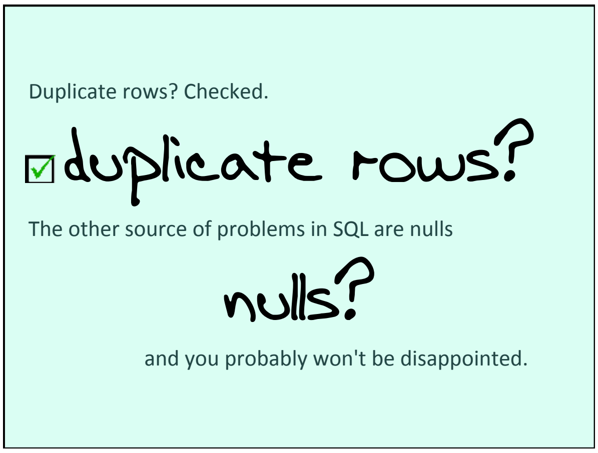
Liu Zijian liuzijian47@163.com



We have seen last time this type of subquery

```
where column in (select ... )
```

as **in** can pull the values its compare to from the result of a subquery, and not only from an explicit list of values. We have also seen that such a subquery could also be used as a virtual table in a join, with an important difference which is than **in** implicitly applies a **distinct** to the result of the query.



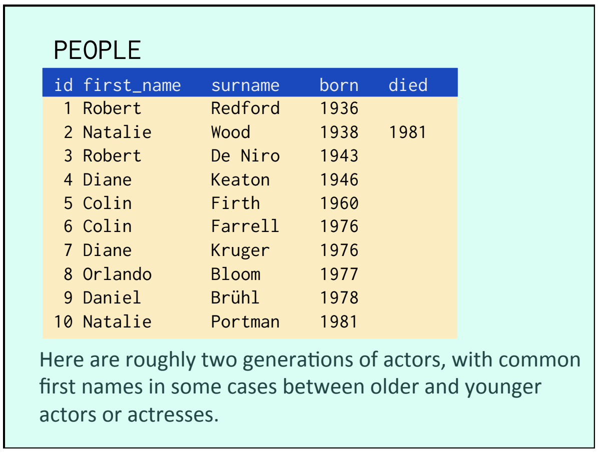
Duplicate rows? Checked.

☑ duplicate rows?

The other source of problems in SQL are nulls

nulls?

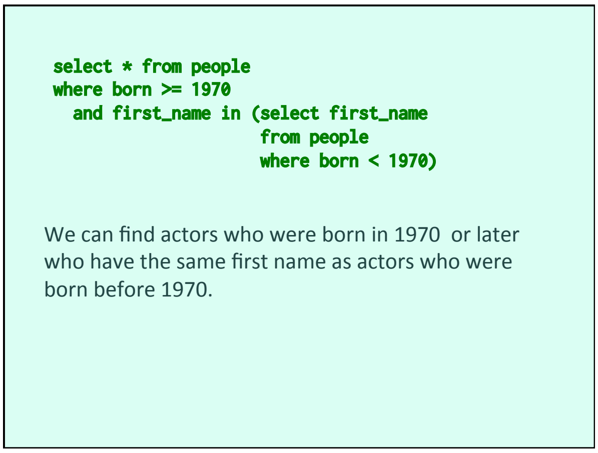
and you probably won't be disappointed.



PEOPLE

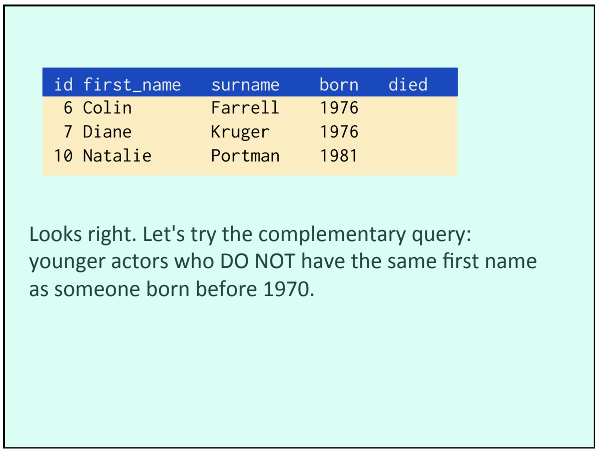
id	first_name	surname	born	died
1	Robert	Redford	1936	
2	Natalie	Wood	1938	1981
3	Robert	De Niro	1943	
4	Diane	Keaton	1946	
5	Colin	Firth	1960	
6	Colin	Farrell	1976	
7	Diane	Kruger	1976	
8	Orlando	Bloom	1977	
9	Daniel	Brühl	1978	
10	Natalie	Portman	1981	

Here are roughly two generations of actors, with common first names in some cases between older and younger actors or actresses.



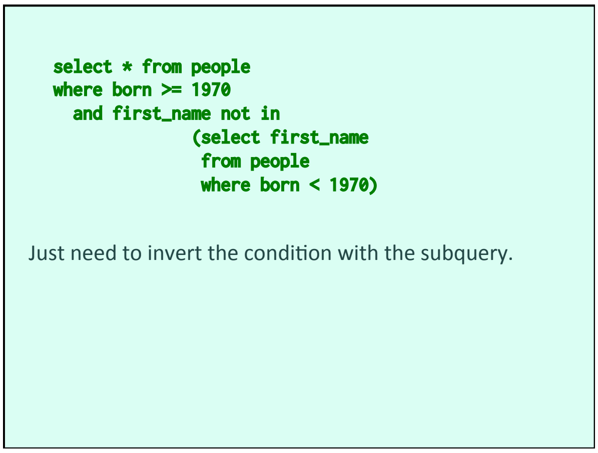
```
select * from people
where born >= 1970
  and first_name in (select first_name
                     from people
                     where born < 1970)
```

We can find actors who were born in 1970 or later who have the same first name as actors who were born before 1970.



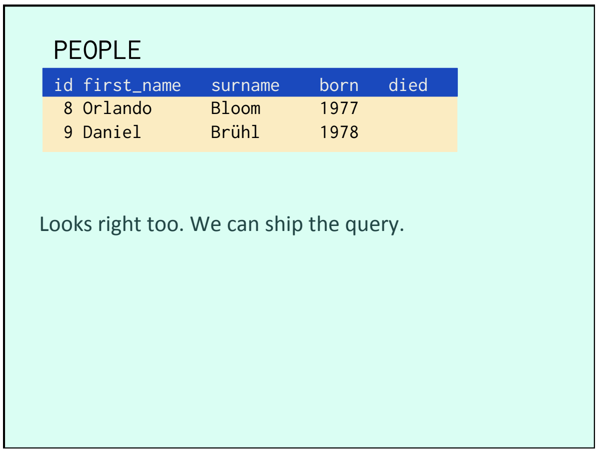
id	first_name	surname	born	died
6	Colin	Farrell	1976	
7	Diane	Kruger	1976	
10	Natalie	Portman	1981	

Looks right. Let's try the complementary query: younger actors who DO NOT have the same first name as someone born before 1970.



```
select * from people
where born >= 1970
  and first_name not in
              (select first_name
               from people
               where born < 1970)
```

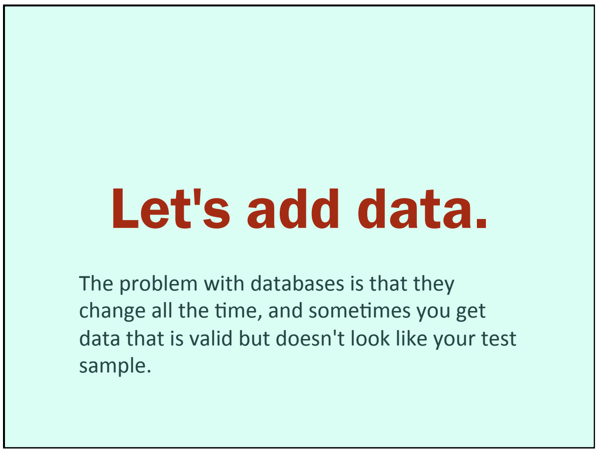
Just need to invert the condition with the subquery.



PEOPLE

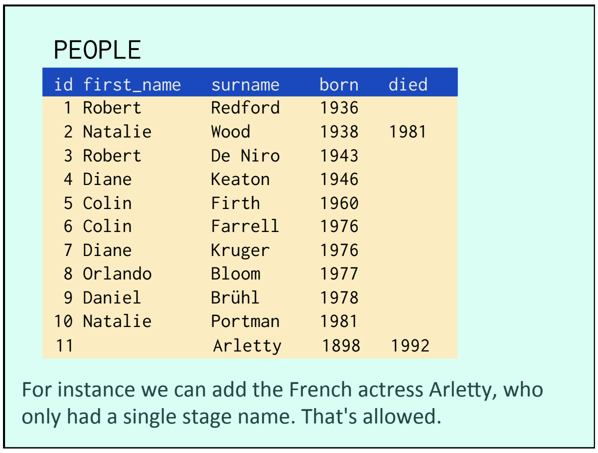
id	first_name	surname	born	died
8	Orlando	Bloom	1977	
9	Daniel	Brühl	1978	

Looks right too. We can ship the query.



Let's add data.

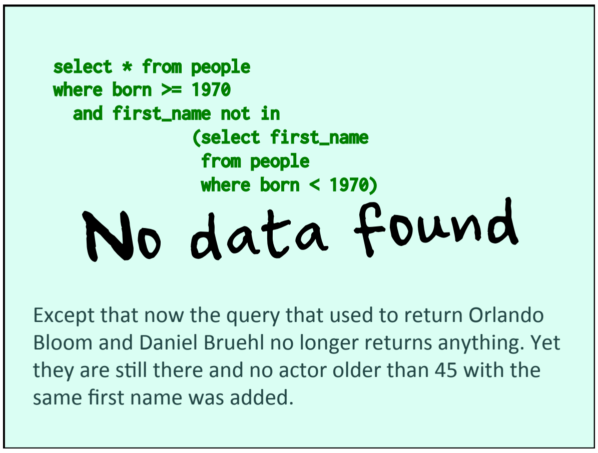
The problem with databases is that they change all the time, and sometimes you get data that is valid but doesn't look like your test sample.



PEOPLE

id	first_name	surname	born	died
1	Robert	Redford	1936	
2	Natalie	Wood	1938	1981
3	Robert	De Niro	1943	
4	Diane	Keaton	1946	
5	Colin	Firth	1960	
6	Colin	Farrell	1976	
7	Diane	Kruger	1976	
8	Orlando	Bloom	1977	
9	Daniel	Brühl	1978	
10	Natalie	Portman	1981	
11		Arletty	1898	1992

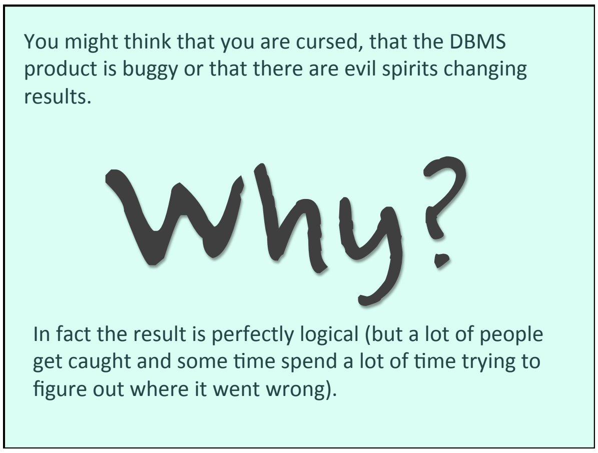
For instance we can add the French actress Arletty, who only had a single stage name. That's allowed.



```
select * from people
where born >= 1970
  and first_name not in
              (select first_name
               from people
               where born < 1970)
```

No data found

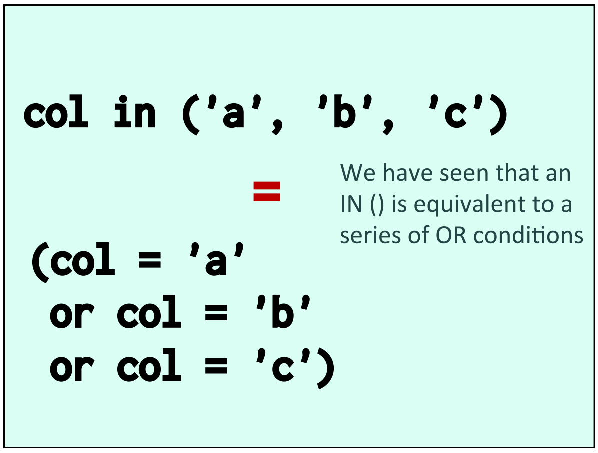
Except that now the query that used to return Orlando Bloom and Daniel Bruehl no longer returns anything. Yet they are still there and no actor older than 45 with the same first name was added.



You might think that you are cursed, that the DBMS product is buggy or that there are evil spirits changing results.

Why?

In fact the result is perfectly logical (but a lot of people get caught and some time spend a lot of time trying to figure out where it went wrong).

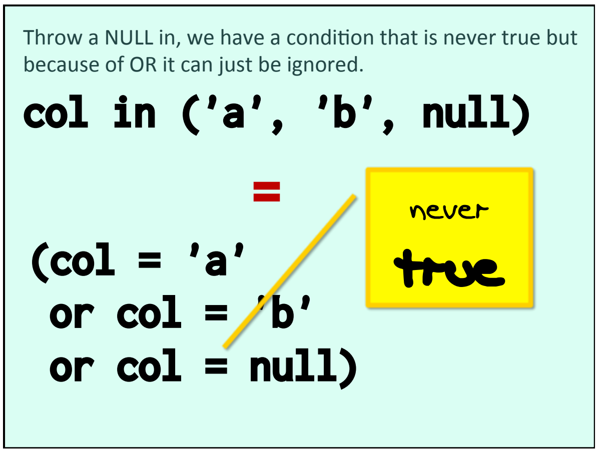


```
col in ('a', 'b', 'c')
```

=

We have seen that an IN () is equivalent to a series of OR conditions

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



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

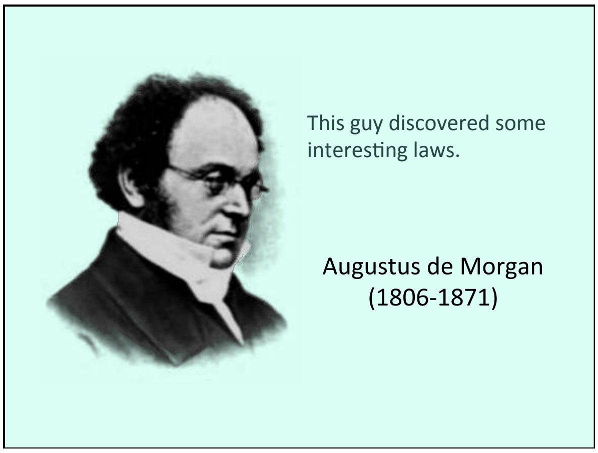
```
col in ('a', 'b', null)
```

=

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

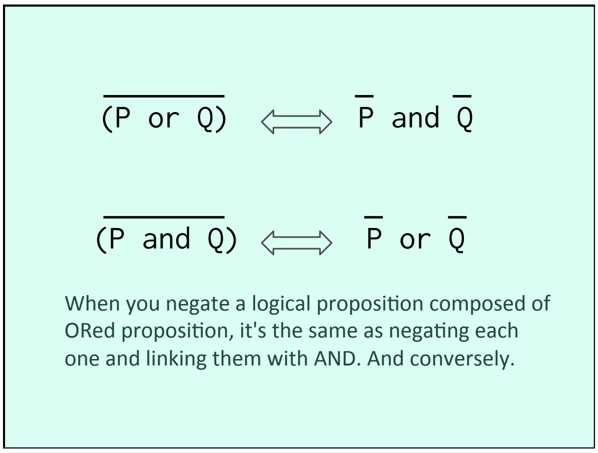
never

true



This guy discovered some interesting laws.

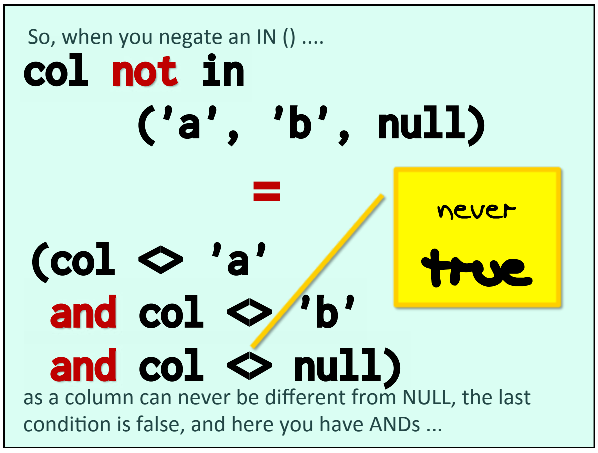
Augustus de Morgan
(1806-1871)



$$\overline{(P \text{ or } Q)} \iff \overline{P} \text{ and } \overline{Q}$$

$$\overline{(P \text{ and } Q)} \iff \overline{P} \text{ or } \overline{Q}$$

When you negate a logical proposition composed of ORed proposition, it's the same as negating each one and linking them with AND. And conversely.

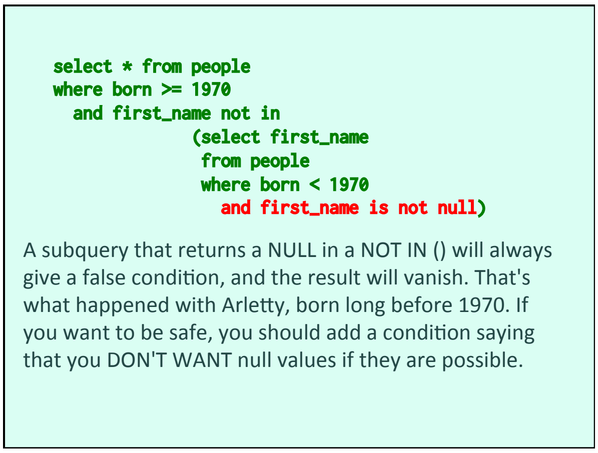


So, when you negate an IN () ....

```
col not in
      ('a', 'b', null)
           =
(col <> 'a'
  and col <> 'b'
  and col <> null)
```

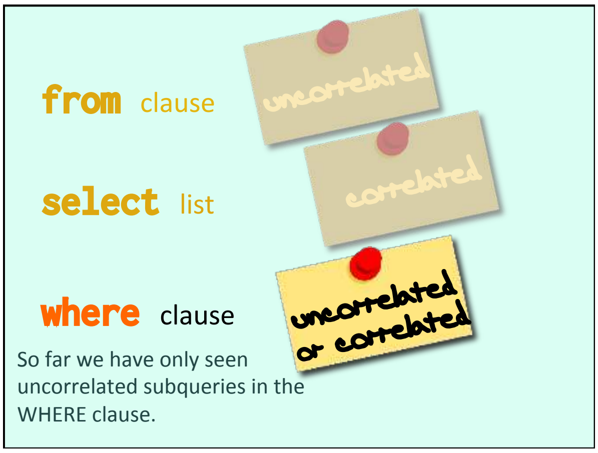
never
~~true~~

as a column can never be different from NULL, the last condition is false, and here you have ANDs ...



```
select * from people
where born >= 1970
  and first_name not in
              (select first_name
               from people
               where born < 1970
                 and first_name is not null)
```

A subquery that returns a NULL in a NOT IN () will always give a false condition, and the result will vanish. That's what happened with Arletty, born long before 1970. If you want to be safe, you should add a condition saying that you DON'T WANT null values if they are possible.



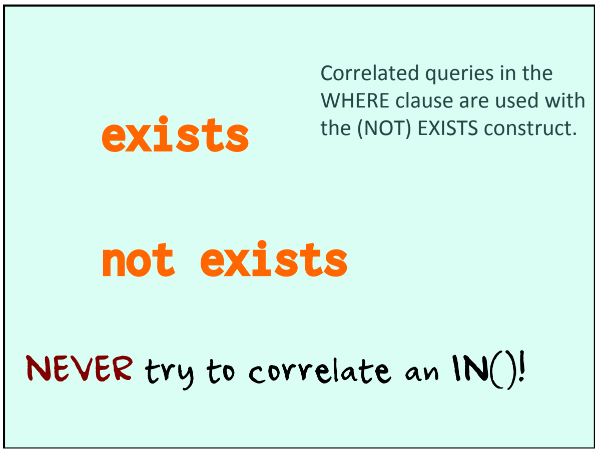
from clause

select list

where clause

uncorrelated
correlated
uncorrelated or correlated

So far we have only seen uncorrelated subqueries in the WHERE clause.



exists

not exists

Correlated queries in the WHERE clause are used with the (NOT) EXISTS construct.

NEVER try to correlate an IN()!

```
and exists
     (select ...
       ...)
```

In an EXISTS I have, as with subqueries after a SELECT, a reference to a value from the current row of the outer query.

```
select distinct m.title
from movies m
where exists
 (select null
  from credits c
       inner join people p
          on p.peopleid = c.peopleid
  where c.credited_as = 'A'
    and p.born >= 1970
    and c.movieid = m.movieid)
```

For instance the films with at least one actor born in 1970 or later.

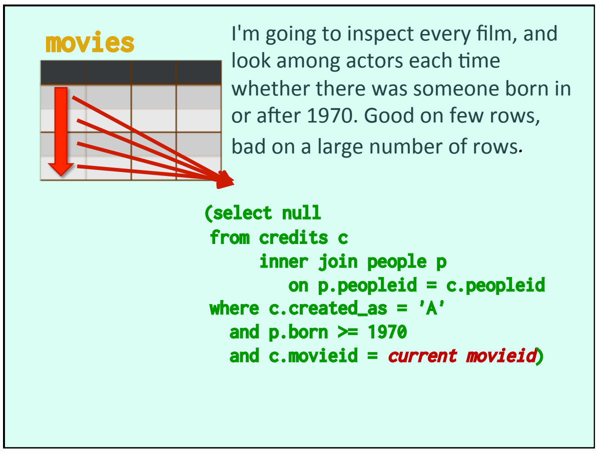
And I've used that scary null.

Flickr: Kevin N. Murphy

```
select m.title
from movies m
where exists
 (select 'Hello World!'
  from credits c
       inner join people p
          on p.peopleid = c.peopleid
  where c.credited_as = 'A'
    and p.born >= 1970
    and c.movieid = m.movieid)
```

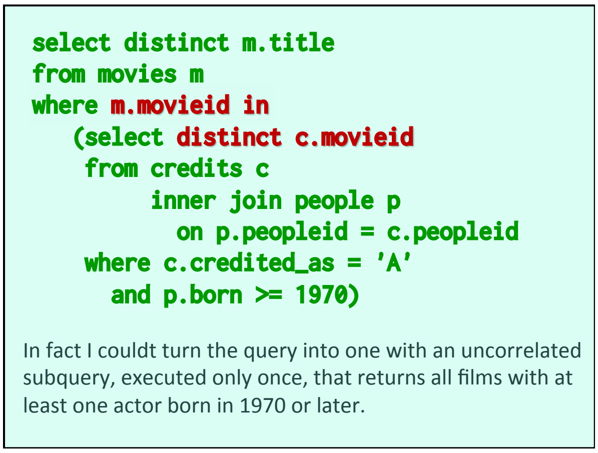
In fact I could have used anything, and NULL emphasizes it.

I'm not interested in whom, or when precisely they were born. I just want to check that there is such a person.



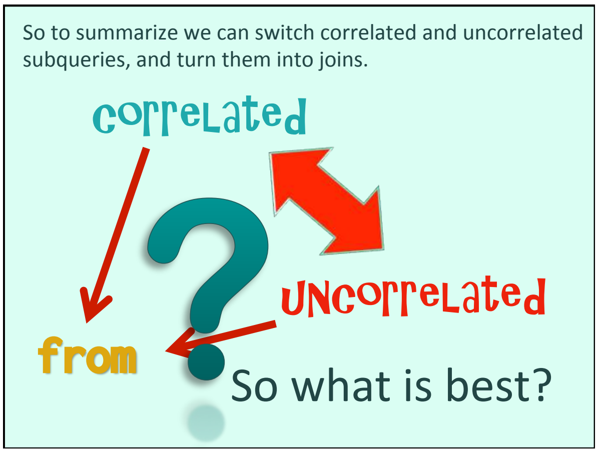
I'm going to inspect every film, and look among actors each time whether there was someone born in or after 1970. Good on few rows, bad on a large number of rows.

```
(select null
 from credits c
      inner join people p
         on p.peopleid = c.peopleid
 where c.created_as = 'A'
   and p.born >= 1970
   and c.movieid = current movieid)
```



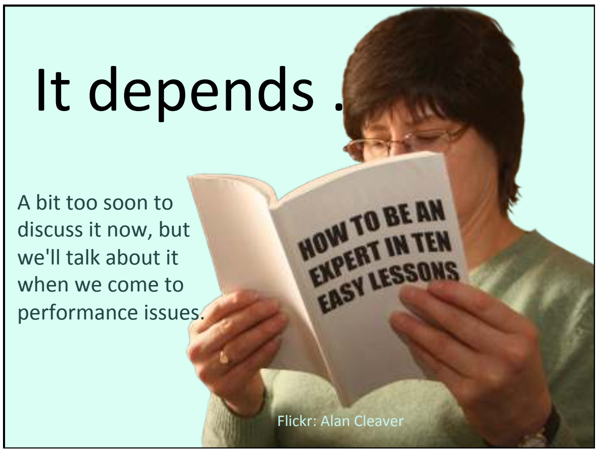
```
select distinct m.title
from movies m
where m.movieid in
   (select distinct c.movieid
    from credits c
         inner join people p
           on p.peopleid = c.peopleid
    where c.credited_as = 'A'
      and p.born >= 1970)
```

In fact I couldt turn the query into one with an uncorrelated subquery, executed only once, that returns all films with at least one actor born in 1970 or later.



So to summarize we can switch correlated and uncorrelated subqueries, and turn them into joins.

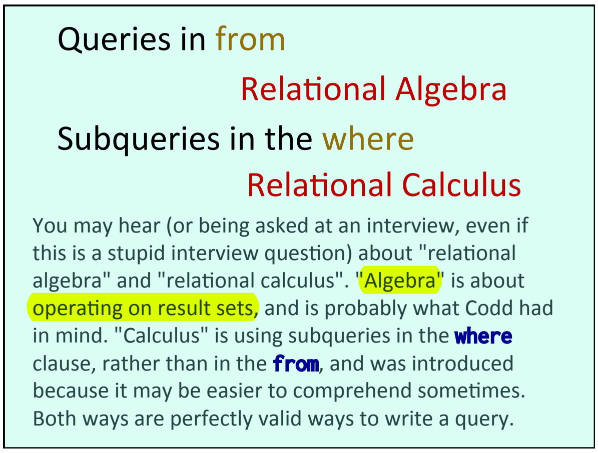
So what is best?



It depends .

A bit too soon to discuss it now, but we'll talk about it when we come to performance issues.

Flickr: Alan Cleaver



Queries in from

Relational Algebra

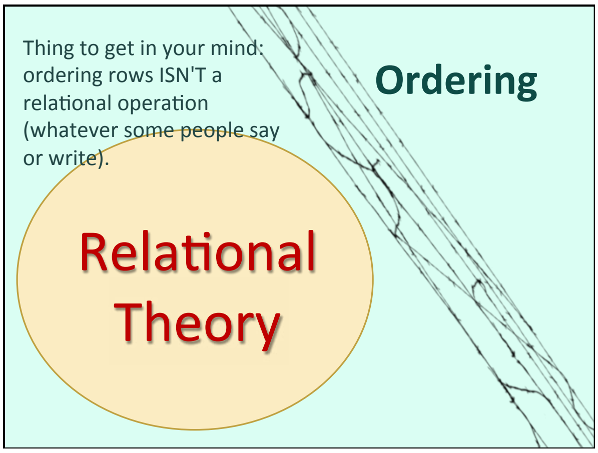
Subqueries in the where

Relational Calculus

You may hear (or being asked at an interview, even if this is a stupid interview question) about "relational algebra" and "relational calculus". "Algebra" is about operating on result sets, and is probably what Codd had in mind. "Calculus" is using subqueries in the **where** clause, rather than in the **from**, and was introduced because it may be easier to comprehend sometimes. Both ways are perfectly valid ways to write a query.

What about sorting data?

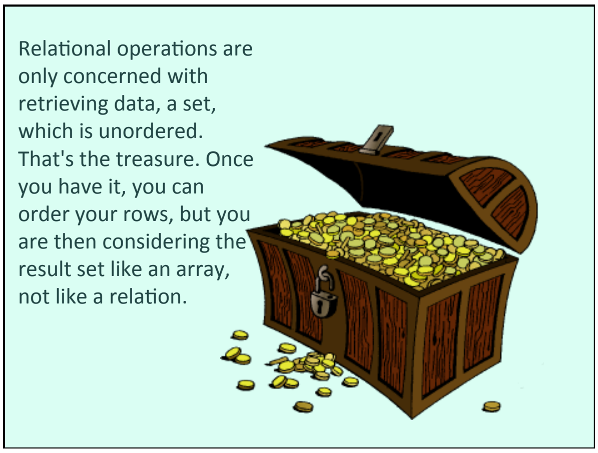
Flickr: emop



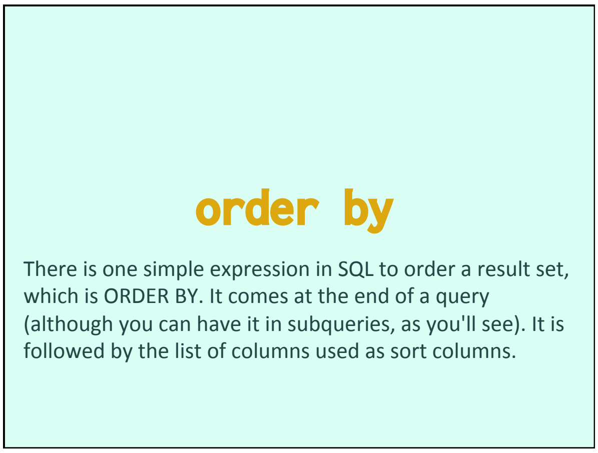
Thing to get in your mind: ordering rows ISN'T a relational operation (whatever some people say or write).

Ordering

Relational Theory

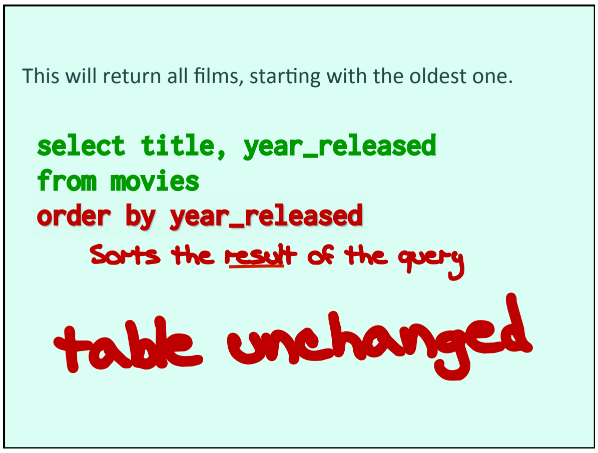


Relational operations are only concerned with retrieving data, a set, which is unordered. That's the treasure. Once you have it, you can order your rows, but you are then considering the result set like an array, not like a relation.



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.

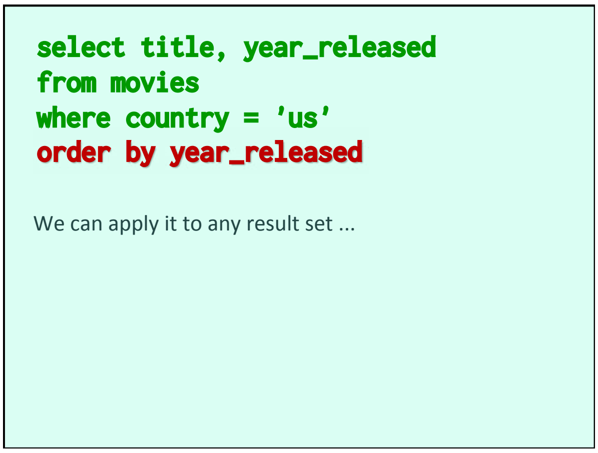


This will return all films, starting with the oldest one.

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

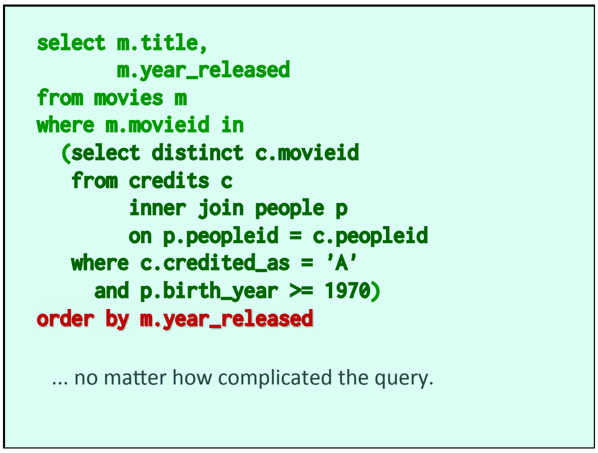
Sorts the result of the query

table unchanged



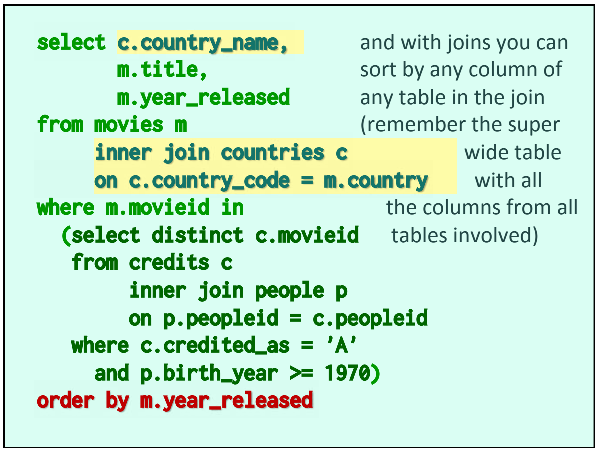
```
select title, year_released
from movies
where country = 'us'
order by year_released
```

We can apply it to any result set ...



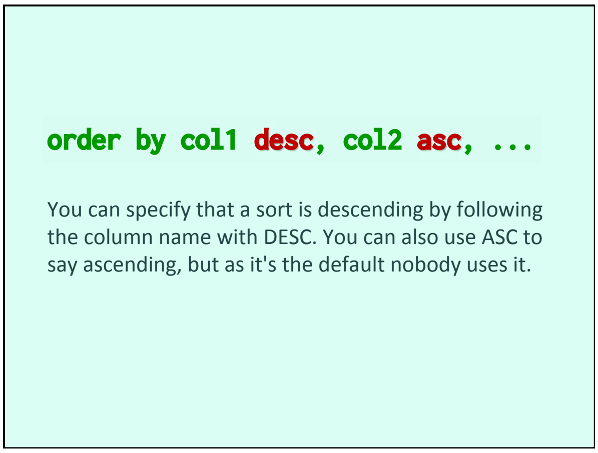
```
select m.title,
       m.year_released
from movies m
where m.movieid in
  (select distinct c.movieid
   from credits c
        inner join people p
        on p.peopleid = c.peopleid
   where c.credited_as = 'A'
     and p.birth_year >= 1970)
order by m.year_released
```

... no matter how complicated the query.



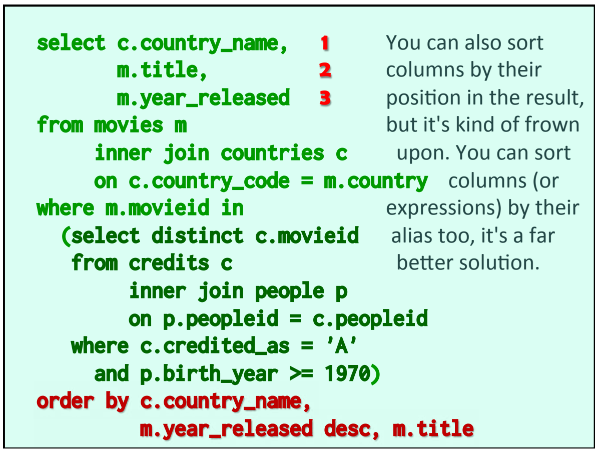
```
select c.country_name,
       m.title,
       m.year_released
from movies m
     inner join countries c
     on c.country_code = m.country
where m.movieid in
  (select distinct c.movieid
   from credits c
        inner join people p
        on p.peopleid = c.peopleid
   where c.credited_as = 'A'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



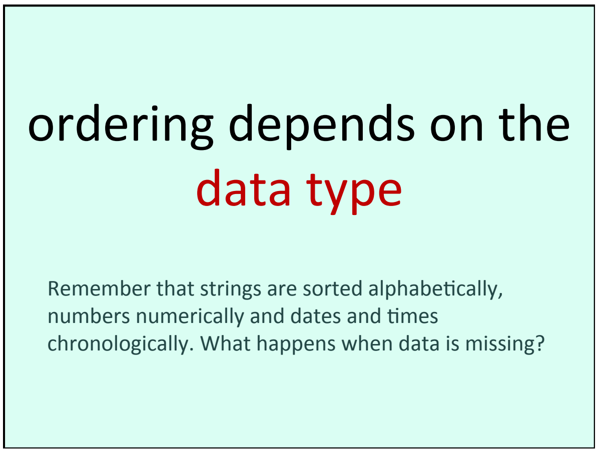
```
order by col1 desc, col2 asc, ...
```

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.



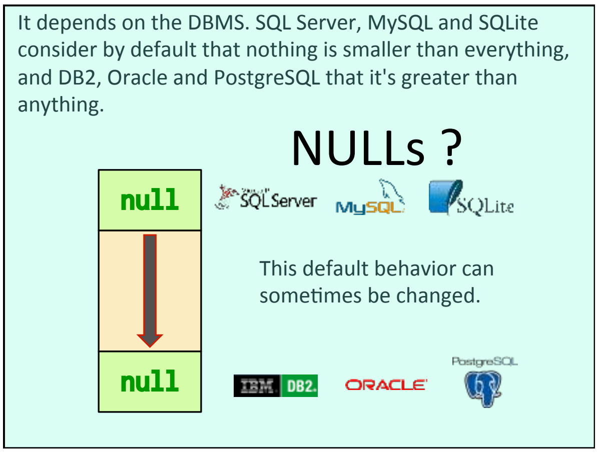
```
select c.country_name,     1
       m.title,            2
       m.year_released     3
from movies m
     inner join countries c
     on c.country_code = m.country
where m.movieid in
  (select distinct c.movieid
   from credits c
        inner join people p
        on p.peopleid = c.peopleid
   where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

You can also sort columns by their position in the result, but it's kind of frown upon. You can sort columns (or expressions) by their alias too, it's a far better solution.



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?



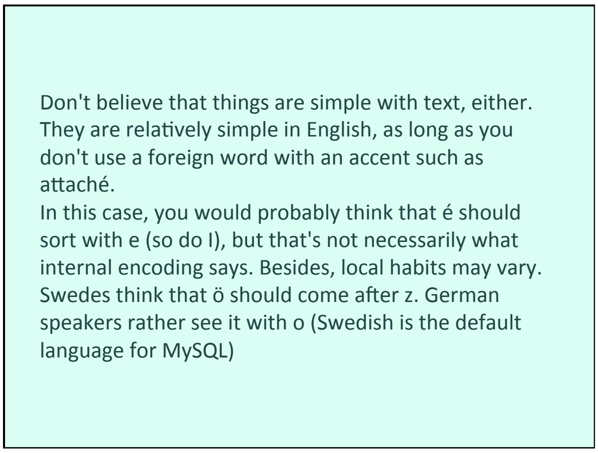
It depends on the DBMS. SQL Server, MySQL and SQLite consider by default that nothing is smaller than everything, and DB2, Oracle and PostgreSQL that it's greater than anything.

NULLs ?

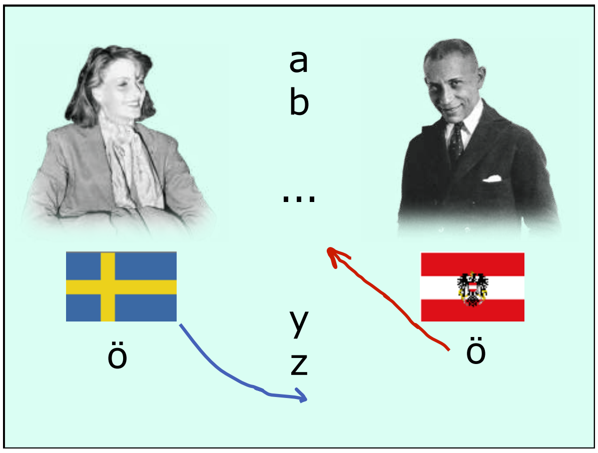
null

null

This default behavior can sometimes be changed.



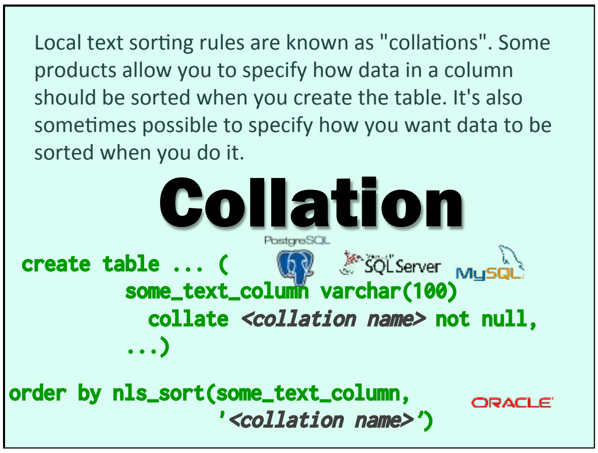
Don't believe that things are simple with text, either. They are relatively simple in English, as long as you don't use a foreign word with an accent such as attaché.
In this case, you would probably think that é should sort with e (so do I), but that's not necessarily what internal encoding says. Besides, 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)



a
b
...
y
z

ö

ö

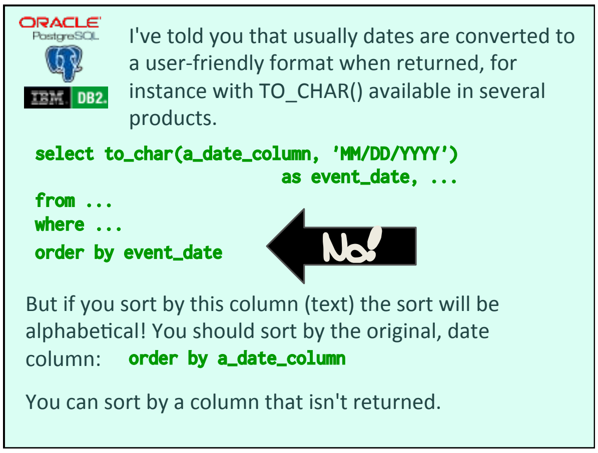


Local text sorting rules are known as "collations". Some products allow you to specify how data in a column should be sorted when you create the table. It's also sometimes possible to specify how you want data to be sorted when you do it.

Collation

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

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



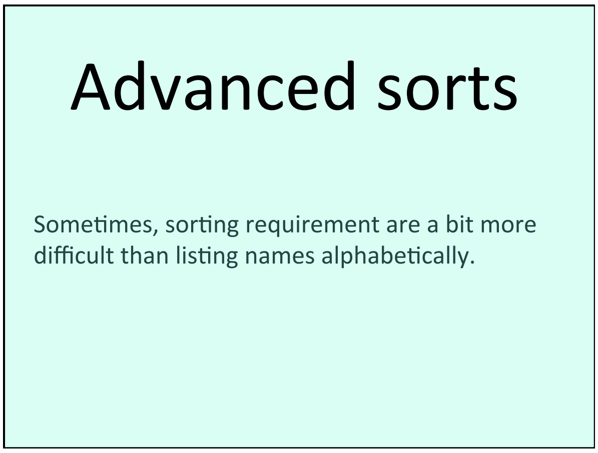
I've told you that usually dates are converted to a user-friendly format when returned, for instance with TO_CHAR() available in several products.

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

ORACLE
PostgreSQL
IBM DB2
No!

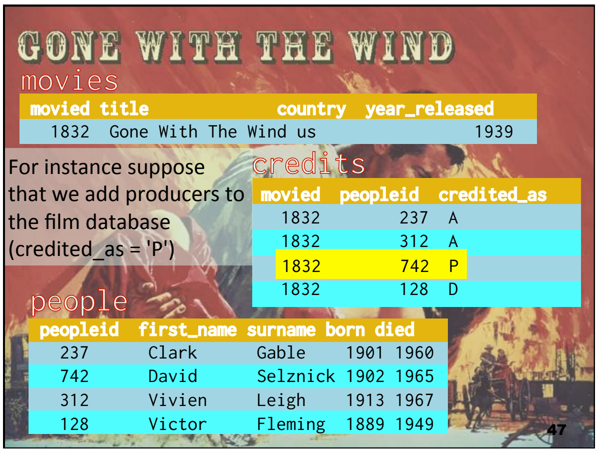
But if you sort by this column (text) the sort will be alphabetical! You should sort by the original, date column: `order by a_date_column`

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



Advanced sorts

Sometimes, sorting requirement are a bit more difficult than listing names alphabetically.



GONE WITH THE WIND

movies

movied	title	country	year_released
1832	Gone With The Wind	us	1939

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

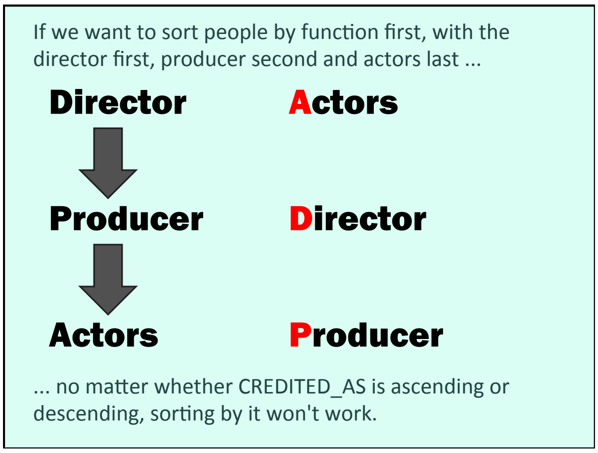
credits

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

people

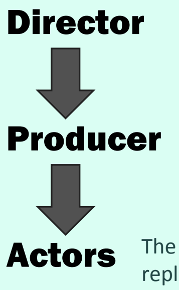
peopleid	first_name	surname	born	died
237	Clark	Gable	1901	1960
742	David	Selznick	1902	1965
312	Vivien	Leigh	1913	1967
128	Victor	Fleming	1889	1949

47



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

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



Director

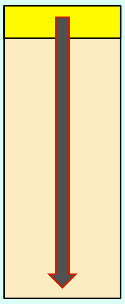
Producer

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".

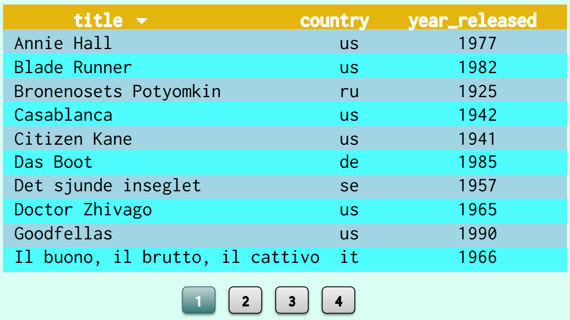
Three oldest ?



```
order by year_released
```

Another problem that isn't so easy is displaying say only a limited number of oldest values, or successive "slices" in a long sorted list. There are now in most products (not MySQL or SQLite) better ways to do it (coming soon), but you may find these things (or similar) in older code.

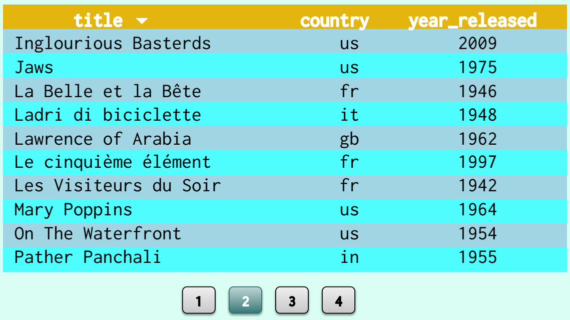
Top 10



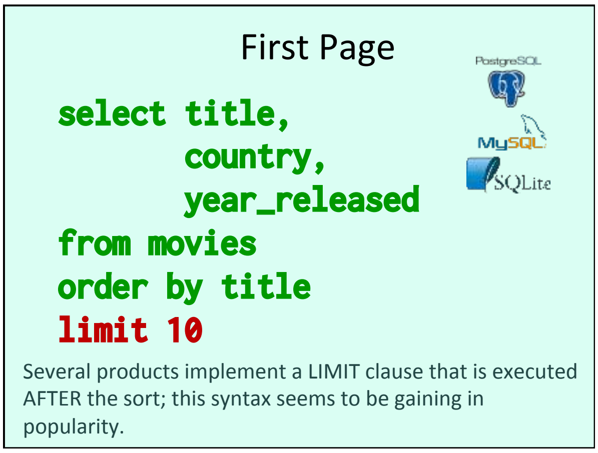
title ▾	country	year_released
Annie Hall	us	1977
Blade Runner	us	1982
Bronenosets Potyomkin	ru	1925
Casablanca	us	1942
Citizen Kane	us	1941
Das Boot	de	1985
Det sjunde inseglet	se	1957
Doctor Zhivago	us	1965
Goodfellas	us	1990
Il buono, il brutto, il cattivo	it	1966

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

Skip 10, Top 10



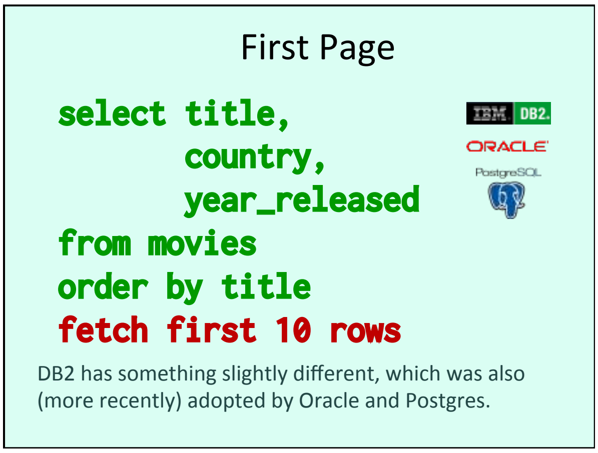
title ▾	country	year_released
Inglourious Basterds	us	2009
Jaws	us	1975
La Belle et la Bête	fr	1946
Ladri di biciclette	it	1948
Lawrence of Arabia	gb	1962
Le cinquième élément	fr	1997
Les Visiteurs du Soir	fr	1942
Mary Poppins	us	1964
On The Waterfront	us	1954
Pather Panchali	in	1955



First Page

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

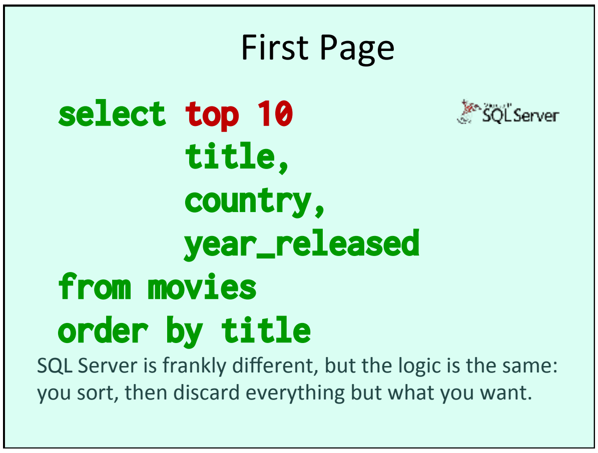
Several products implement a LIMIT clause that is executed AFTER the sort; this syntax seems to be gaining in popularity.



First Page

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

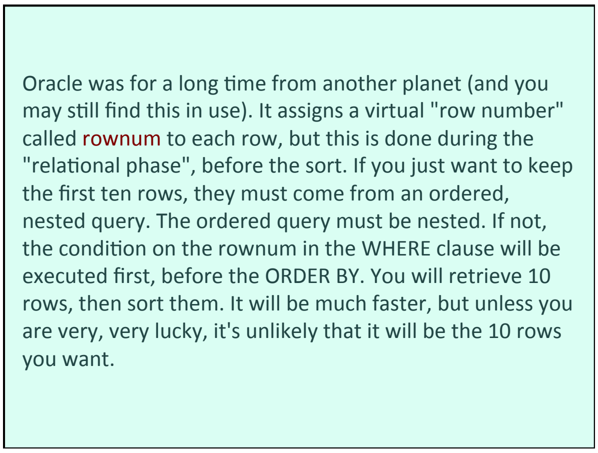
DB2 has something slightly different, which was also (more recently) adopted by Oracle and Postgres.



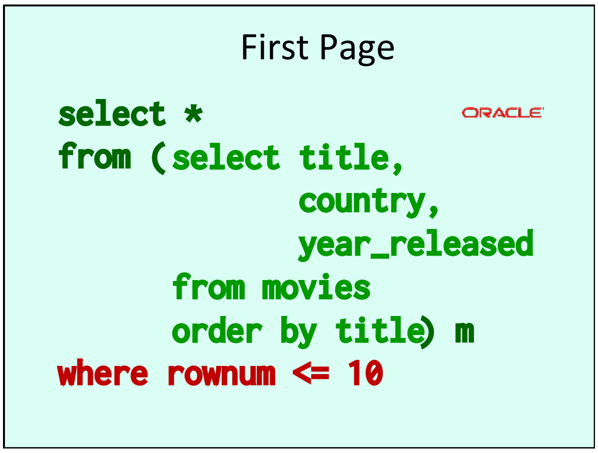
First Page

```
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.

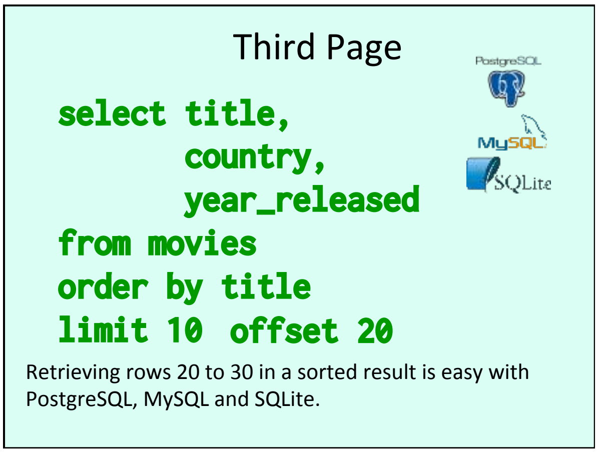


Oracle was for a long time from another planet (and you may still find this in use). It assigns a virtual "row number" called rownum to each row, but this is done during the "relational phase", before the sort. If you just want to keep the first ten rows, they must come from an ordered, nested query. The ordered query must be nested. If not, the condition on the rownum in the WHERE clause will be executed first, before the ORDER BY. You will retrieve 10 rows, then sort them. It will be much faster, but unless you are very, very lucky, it's unlikely that it will be the 10 rows you want.



First Page

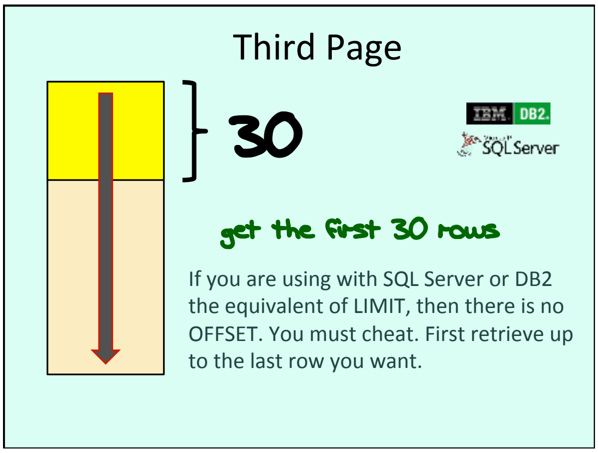
```
select *
from (select title,
             country,
             year_released
      from movies
      order by title) m
where rownum <= 10
```



Third Page

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

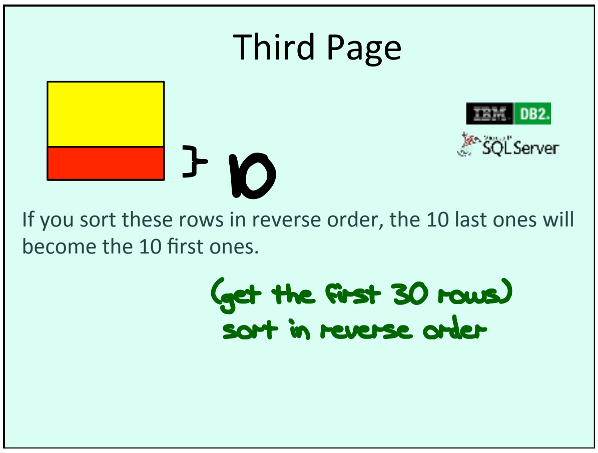
Retrieving rows 20 to 30 in a sorted result is easy with PostgreSQL, MySQL and SQLite.



Third Page

get the first 30 rows

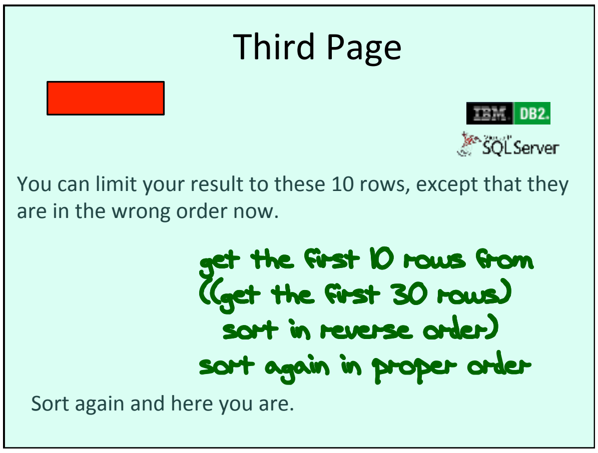
If you are using with SQL Server or DB2 the equivalent of LIMIT, then there is no OFFSET. You must cheat. First retrieve up to the last row you want.



Third Page

If you sort these rows in reverse order, the 10 last ones will become the 10 first ones.

(get the first 30 rows)
sort in reverse order



Third Page

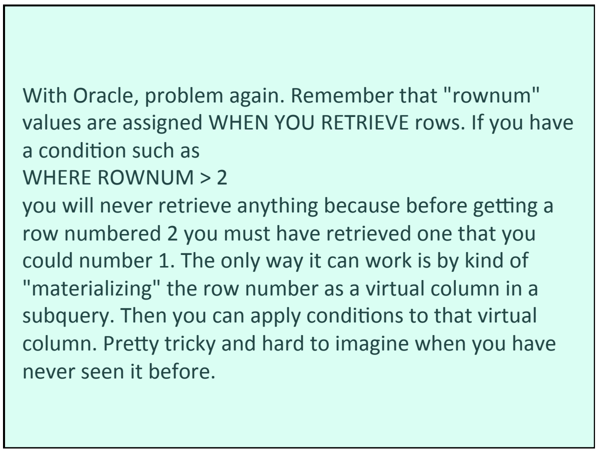
IBM DB2

SQL Server

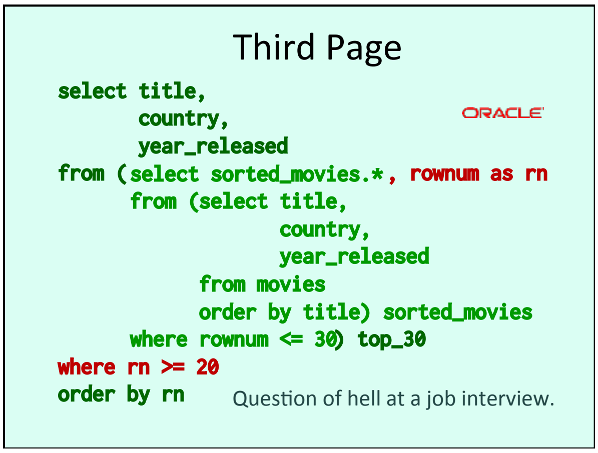
You can limit your result to these 10 rows, except that they are in the wrong order now.

get the first 10 rows from
((get the first 30 rows)
sort in reverse order)
sort again in proper order

Sort again and here you are.



With Oracle, problem again. Remember that "rownum" values are assigned WHEN YOU RETRIEVE rows. If you have a condition such as
WHERE ROWNUM > 2
you will never retrieve anything because before getting a row numbered 2 you must have retrieved one that you could number 1. The only way it can work is by kind of "materializing" the row number as a virtual column in a subquery. Then you can apply conditions to that virtual column. Pretty tricky and hard to imagine when you have never seen it before.

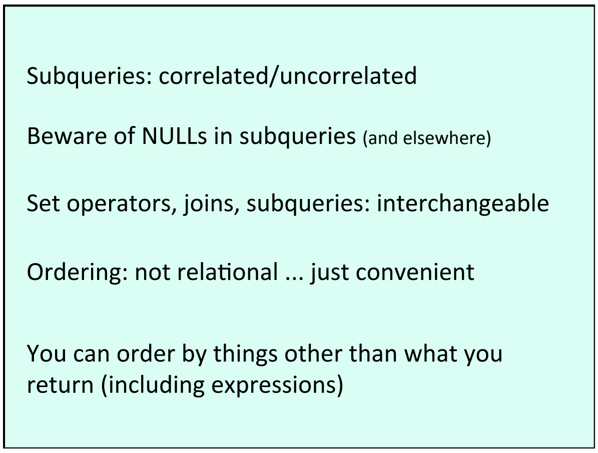


Third Page

ORACLE

```
select title,
       country,
       year_released
from (select sorted_movies.*, rownum as rn
      from (select title,
                   country,
                   year_released
            from movies
            order by title) sorted_movies
      where rownum <= 30) top_30
where rn >= 20
order by rn
```

Question of hell at a job interview.



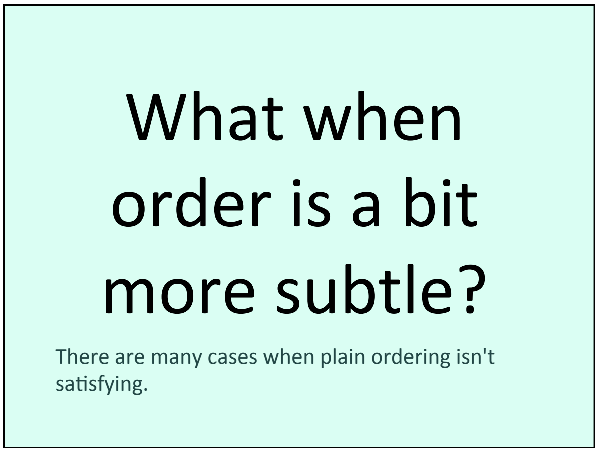
Subqueries: correlated/uncorrelated

Beware of NULLs in subqueries (and elsewhere)

Set operators, joins, subqueries: interchangeable

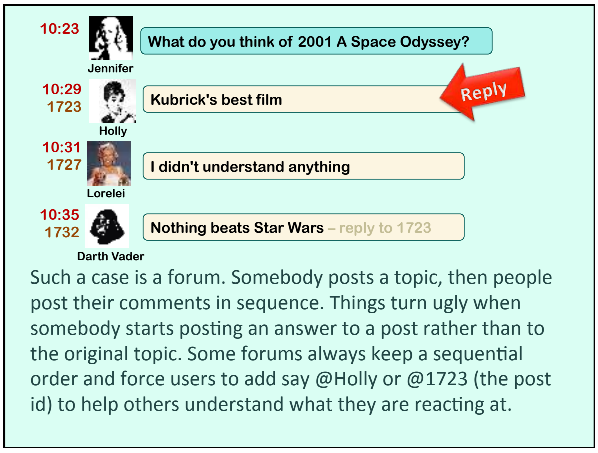
Ordering: not relational ... just convenient

You can order by things other than what you return (including expressions)



What when order is a bit more subtle?

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



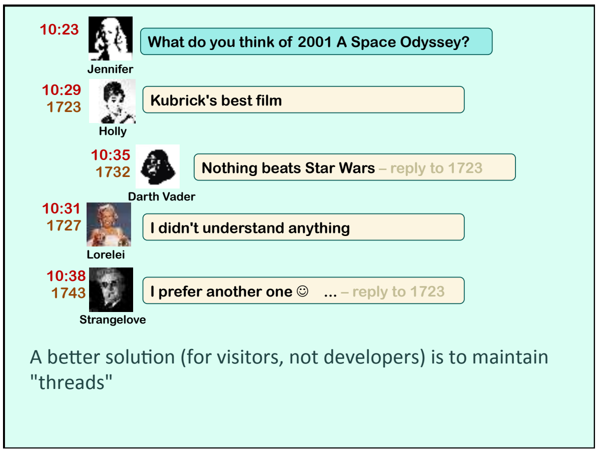
10:23 Jennifer — What do you think of 2001 A Space Odyssey?

10:29 1723 Holly — Kubrick's best film

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 post 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?

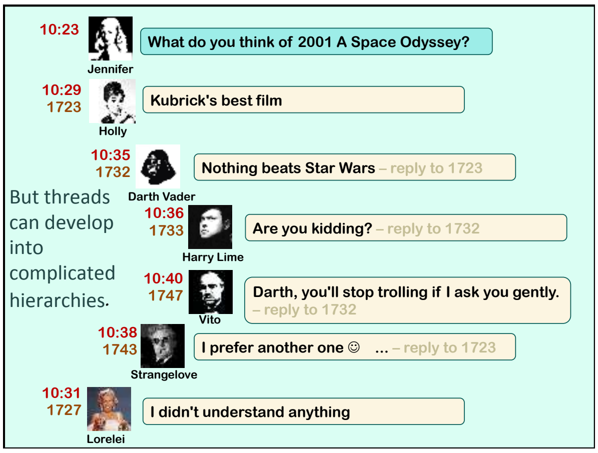
10:29 1723 Holly — Kubrick's best film

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 film

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

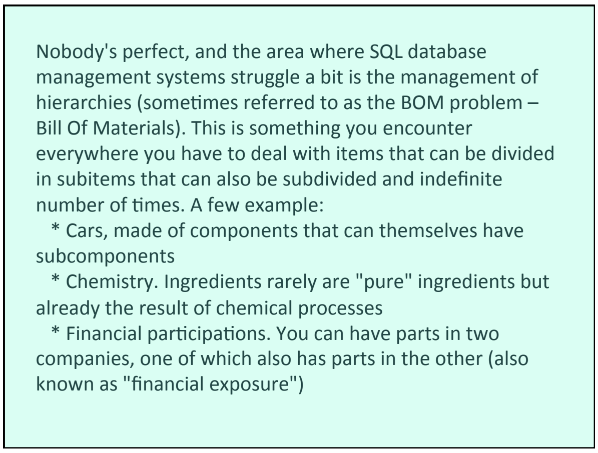
10:36 1733 Harry Lime — Are you kidding? – reply to 1732

10:40 1747 Vito — Darth, you'll stop trolling if I ask you gently. – reply to 1732

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

10:31 1727 Lorelei — I didn't understand anything

But threads can develop into complicated hierarchies.

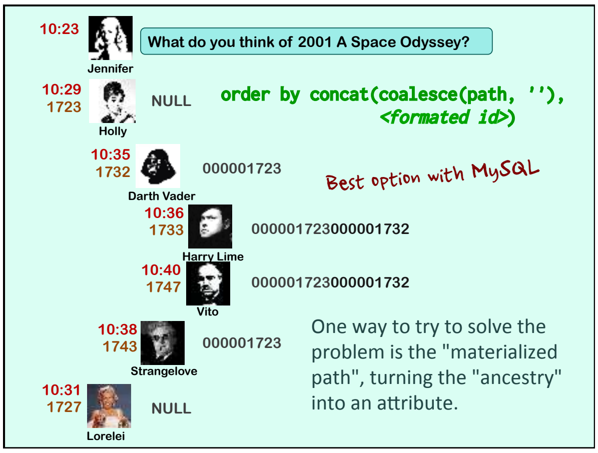


Nobody's perfect, and the area where SQL database management systems struggle a bit is the management of hierarchies (sometimes referred to as the BOM problem – Bill Of Materials). This is something you encounter everywhere you have to deal with items that can be divided in subitems that can also be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

10:35 1732 Darth Vader 000001723

10:36 1733 Harry Lime 000001723000001732

10:40 1747 Vito 000001723000001732

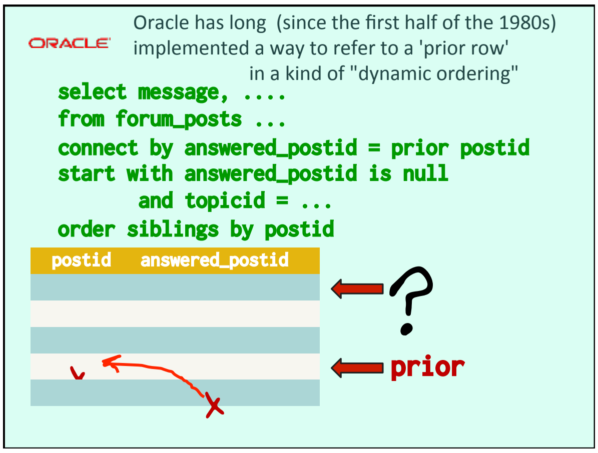
10:38 1743 Strangelove 000001723

10:31 1727 Lorelei NULL

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

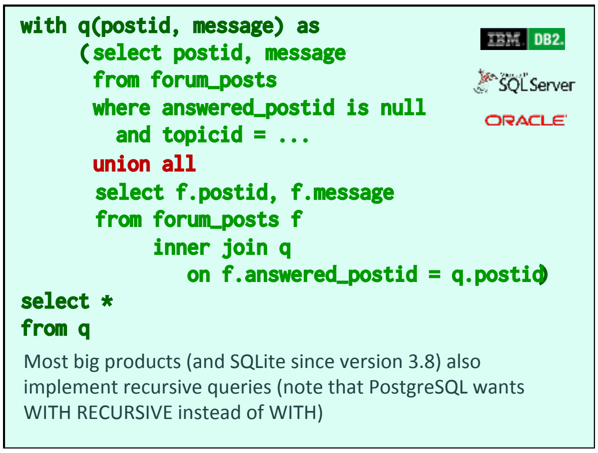
Best option with MySQL

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



ORACLE 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"

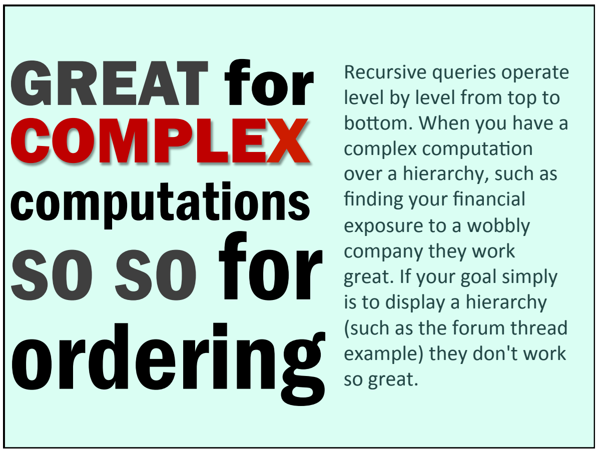
```
select message, ....
from forum_posts ...
connect by answered_postid = prior postid
start with answered_postid is null
          and topicid = ...
order siblings by postid
```



```
with q(postid, message) as
       (select postid, message
        from forum_posts
        where answered_postid is null
          and topicid = ...
        union all
        select f.postid, f.message
        from forum_posts f
             inner join q
                on f.answered_postid = q.postid)
select *
from q
```

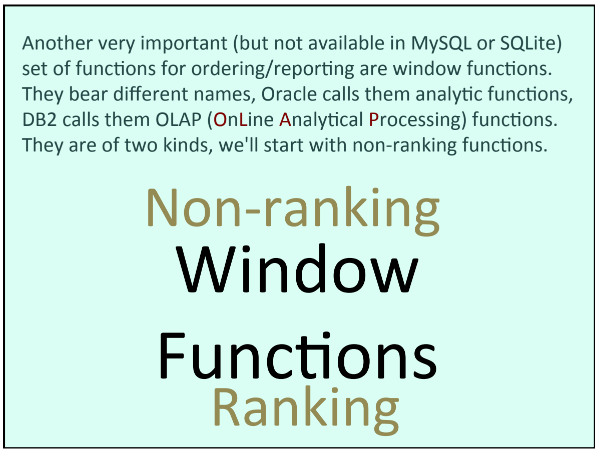
IBM DB2.
SQL Server
ORACLE

Most big products (and SQLite since version 3.8) also implement recursive queries (note that PostgreSQL wants WITH RECURSIVE instead of WITH)



GREAT for COMPLEX computations so so for ordering

Recursive queries operate level by level from top to bottom. When you have a complex computation over a hierarchy, such as finding your financial exposure to a wobbly company they work great. If your goal simply is to display a hierarchy (such as the forum thread example) they don't work so great.



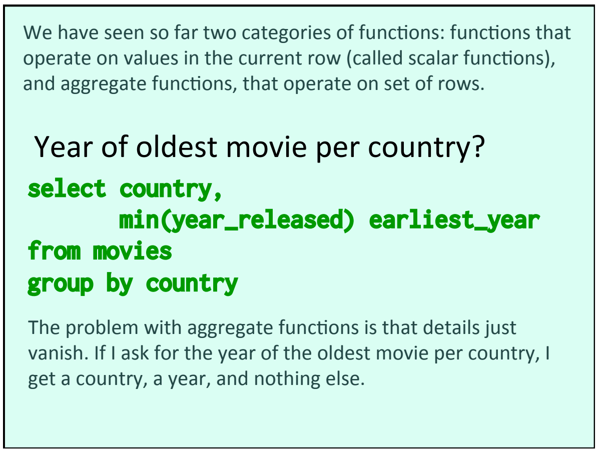
Another very important (but not available in MySQL or SQLite) set of functions for ordering/reporting are window functions. They bear different names, Oracle calls them analytic functions, DB2 calls them OLAP (OnLine Analytical Processing) functions. They are of two kinds, we'll start with non-ranking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

Non-ranking

Window

Functions

Ranking

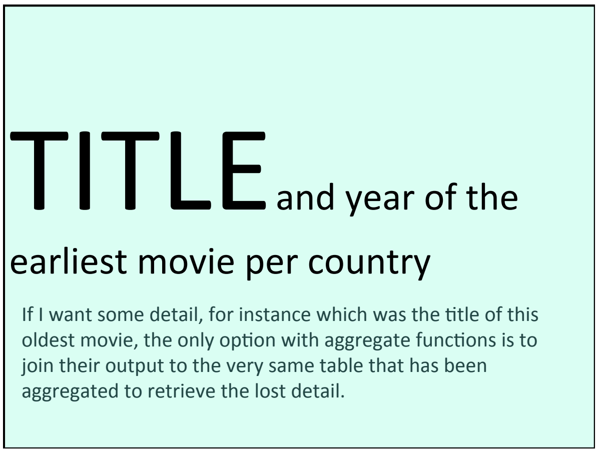


We have seen so far two categories of functions: functions that operate on values in the current row (called scalar functions), and aggregate functions, that operate on set of rows.

Year of oldest movie per country?

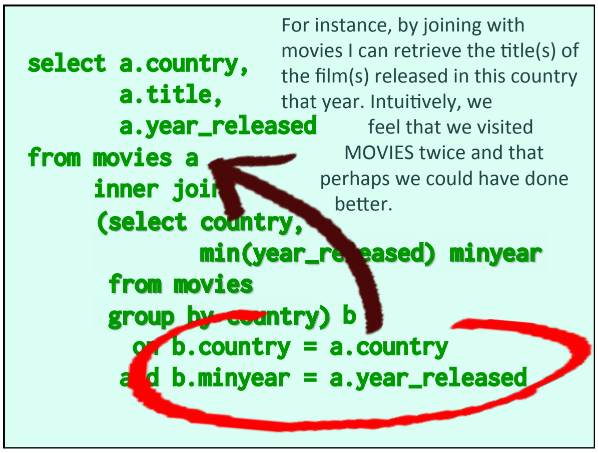
```
select country,
       min(year_released) earliest_year
from movies
group by country
```

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.



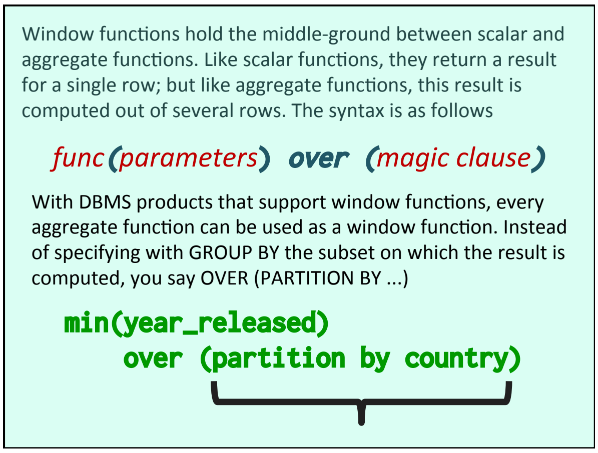
TITLE and year of the earliest movie per country

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.



```
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
```

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

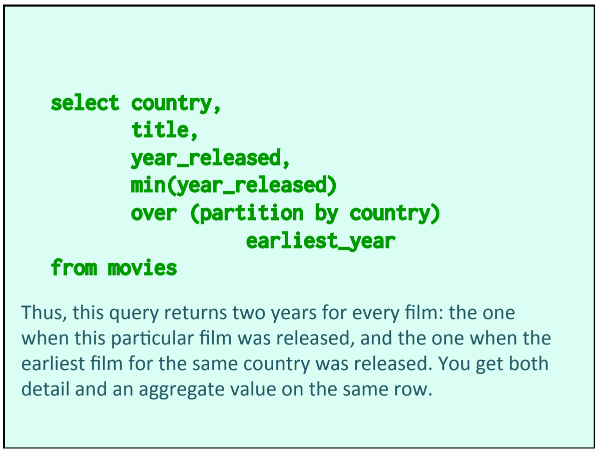


Window functions hold the middle-ground between scalar and aggregate functions. Like scalar functions, they return a result for a single row; but like aggregate functions, this result is computed out of several rows. The syntax is as follows

func(parameters) ***over*** *(magic clause)*

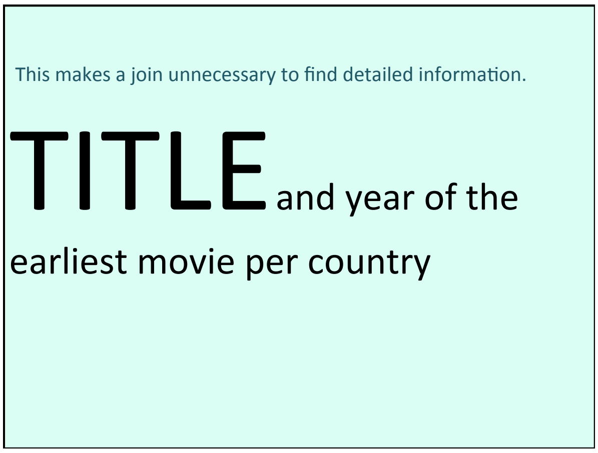
With DBMS products that support window functions, every aggregate function can be used as a window function. Instead of specifying with GROUP BY the subset on which the result is computed, you say OVER (PARTITION BY ...)

```
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 film: the one when this particular film was released, and the one when the earliest film for the same country was released. You get both detail and an aggregate value on the same row.

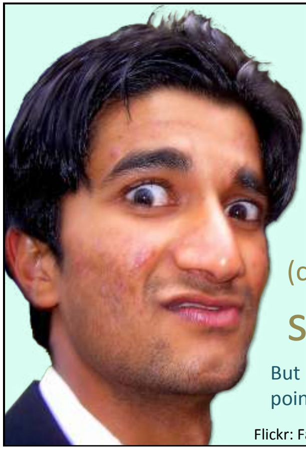


This makes a join unnecessary to find detailed information.

TITLE and year of the earliest movie per country

```
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 films for which the year of release happens to be the same as the earliest one for their country.



Oldest movie you like least?

(country with several movies)

But let's illustrate an important point.

Flickr: Faraz Usmani

```
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
```

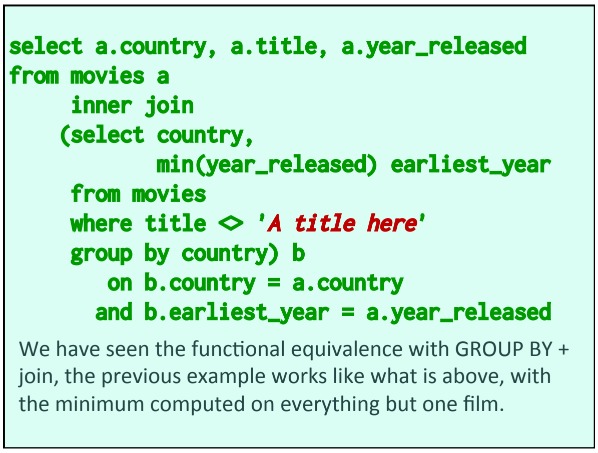
If you filter out, with a WHERE condition, one film, it will be excluded from the window function computation. The earliest year may become the second earliest.

Window functions always operate againts 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

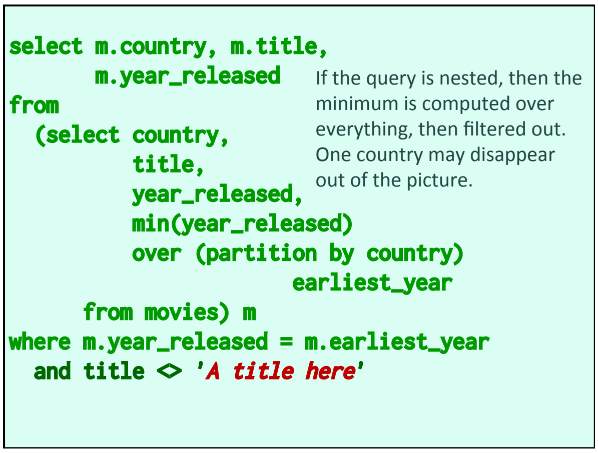
↓

SELECTED rows



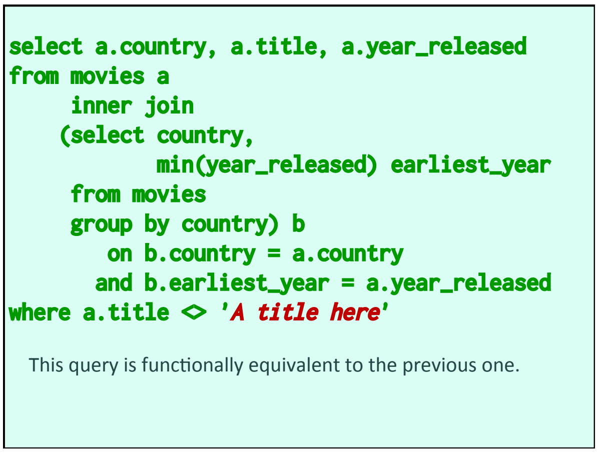
```
select a.country, a.title, a.year_released
from movies a
     inner join
    (select country,
            min(year_released) earliest_year
     from movies
     where title <> 'A title here'
     group by country) b
        on b.country = a.country
      and b.earliest_year = a.year_released
```

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 film.



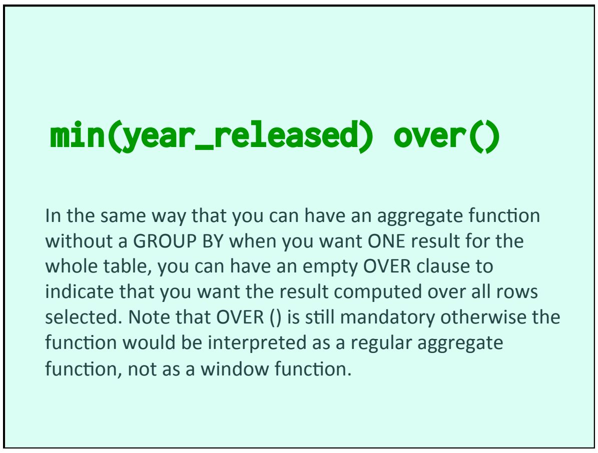
```
select m.country, m.title,
       m.year_released
from
  (select country,
          title,
          year_released,
          min(year_released)
          over (partition by country)
                      earliest_year
      from movies) m
where m.year_released = m.earliest_year
  and title <> 'A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



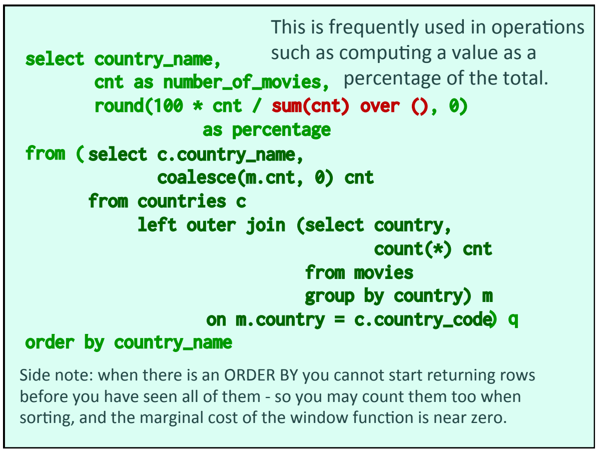
```
select a.country, a.title, a.year_released
from movies a
     inner join
    (select country,
            min(year_released) earliest_year
     from movies
     group by country) b
        on b.country = a.country
      and b.earliest_year = a.year_released
where a.title <> 'A title here'
```

This query is functionally equivalent to the previous one.



```
min(year_released) over()
```

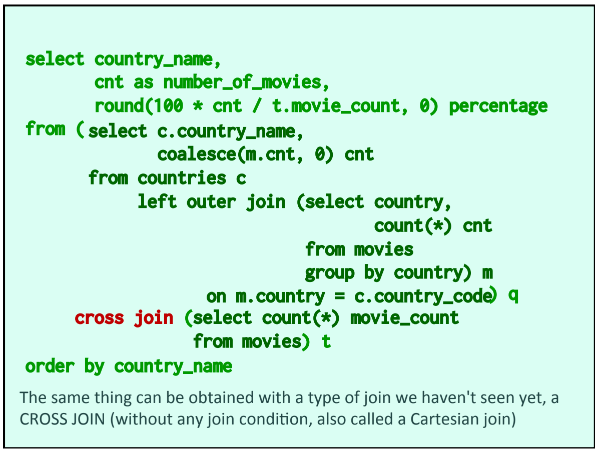
In the same way that you can have an aggregate function without a GROUP BY when you want ONE result for the whole table, you can have an empty OVER clause to indicate that you want the result computed over all rows selected. Note that OVER () is still mandatory otherwise the function would be interpreted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



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

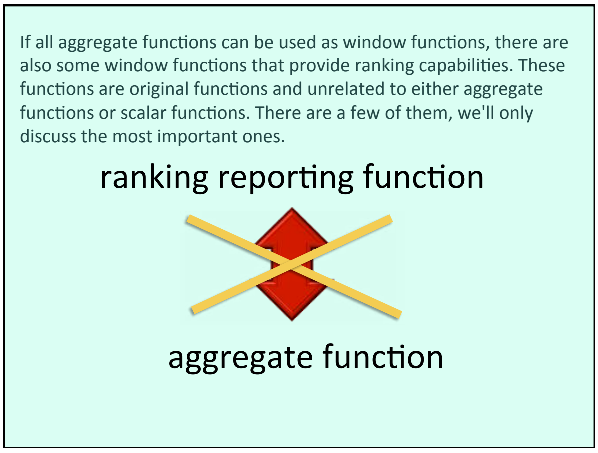
```
select country_name,
       cnt as number_of_movies,
       round(100 * cnt / sum(cnt) over (), 0)
              as percentage
from (select c.country_name,
             coalesce(m.cnt, 0) cnt
        from countries c
             left outer join (select country,
                                 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

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
```

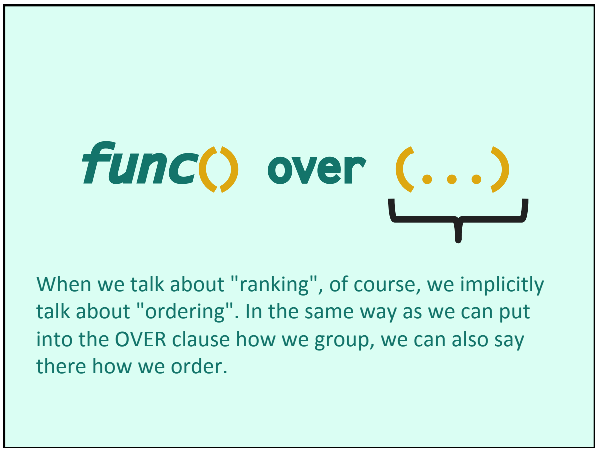
The same thing can be obtained with a type of join we haven't seen yet, a CROSS JOIN (without any join condition, also called a Cartesian join)



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 reporting function

aggregate function



func() over (...)

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.

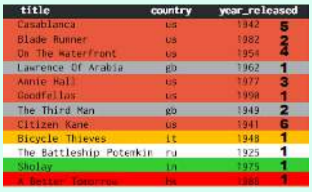
```
row_number()
rank()
dense_rank()
```

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 ...)
```

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

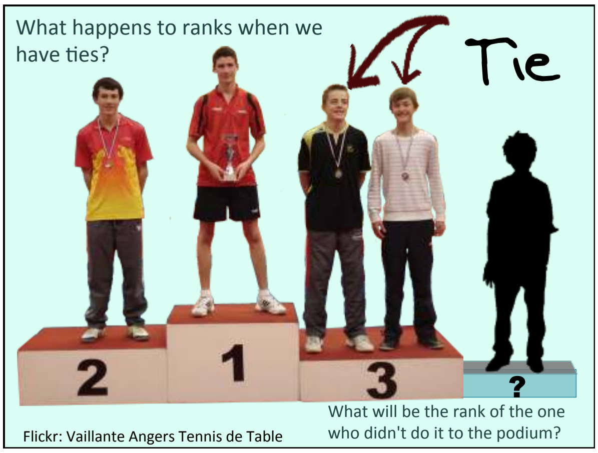
In this example, films are grouped by country, and a sequential number is assigned by country to each film, starting with the most recent film.



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

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

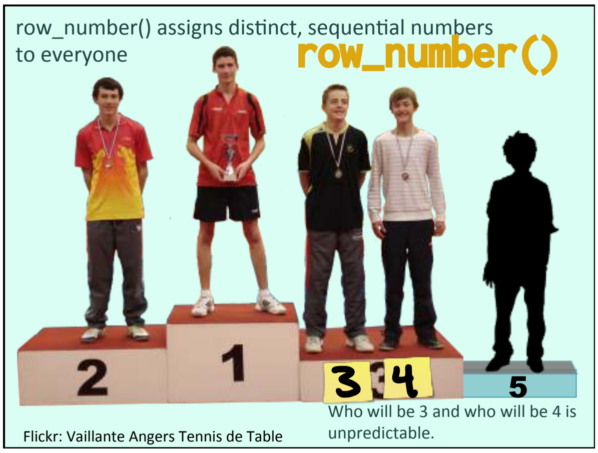
As with plain GROUP BY and plain ORDER BY, both partitioning and ordering can be applied to several columns.



What happens to ranks when we have ties?

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

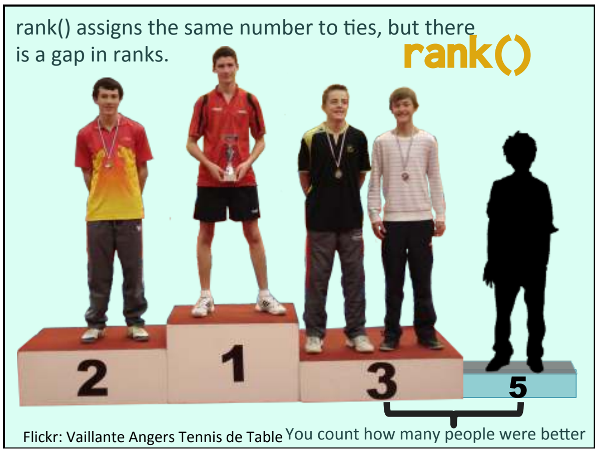
Flickr: Vaillante Angers Tennis de Table



row_number() assigns distinct, sequential numbers to everyone

Who will be 3 and who will be 4 is unpredictable.

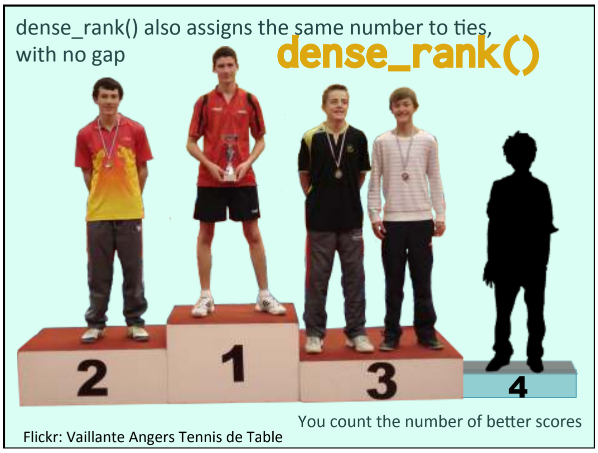
Flickr: Vaillante Angers Tennis de Table



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

You count how many people were better

Flickr: Vaillante Angers Tennis de Table



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

You count the number of better scores

Flickr: Vaillante Angers Tennis de Table

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.

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

```
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
```

An interesting application of Window functions is how you can generate some HTML output straight from SQL. It's certainly stretching SQL a bit, but when you are a consultant, need to generate reports, have nothing but a command line interface and aren't mad about console output into Word and reformatting it, it can help.

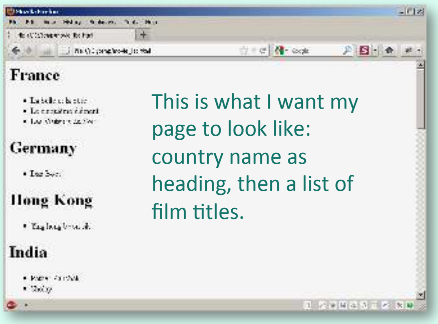
<tag> The only remarkable thing about HTML is that it uses pairs (usually) of tags that can be nested.

O tempora, o mores! Senatus haec intellegit. Consul videt; hic tamen vivit. Vivit? immo vero etiam in senatum venit, fit <tag2>publici consilii</tag2> particeps, notat et designat oculis ad caedem unum quemque nostrum. Nos autem fortes viri satis facere rei publicae videmur, si istius furorem ac tela vitemus. Ad mortem te, Catilina, duci iussu consulis iam pridem oportebat, in te conferri pestem, quam tu in nos machinaris.

</tag>

In this example I'll only use three pair of tags.

`<h1> ... </h1>`	Top-level headings
`<ul> ... </ul>`	Unordered (bullet) list
`<li> ... </li>`	List item



France

- La belle et la bête
- Le cinquième élément
- [illegible]

Germany

- Das Boot

Hong Kong

- [illegible]

India

- Pather Panchali
- Sholay

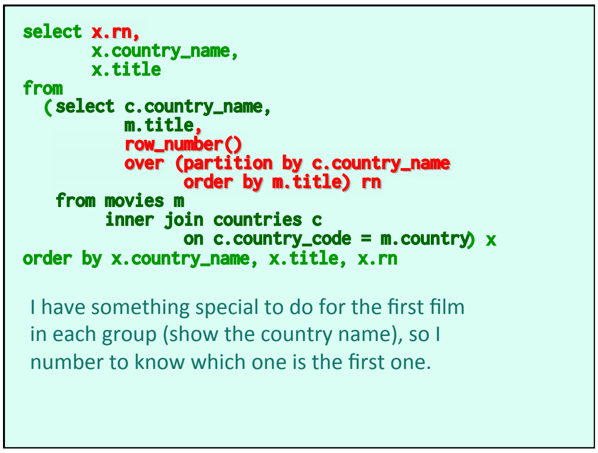
This is what I want my page to look like: country name as heading, then a list of film titles.

```
<h1>India</h1>
<ul>
  <li>Pather Panchali</li>
  <li>Sholay</li>
</ul>
```

As you see, if each title can be plainly output between LI tags, I must display the country name and start the list before the first film from a country, and I must enclose the list after the last one.

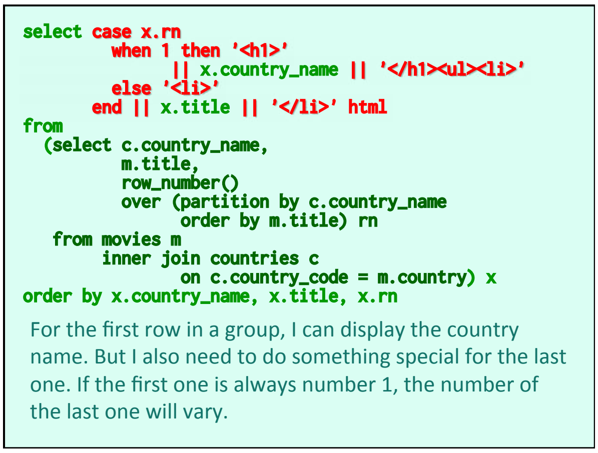
```
select c.country_name,
       m.title
from movies m
     inner join countries c
             on c.country_code = m.country
order by c.country_name, m.title
```

Let's start with the plain query that returns the data I want to display.



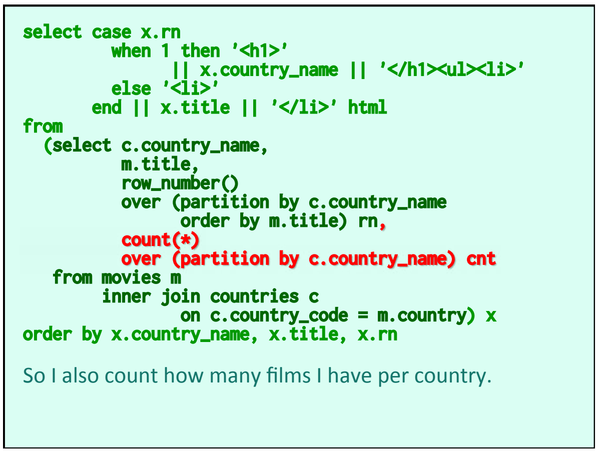
```
select x.rn,
       x.country_name,
       x.title
from
  (select c.country_name,
          m.title,
          row_number()
          over (partition by c.country_name
                order by m.title) rn
   from movies m
        inner join countries c
                on c.country_code = m.country) x
order by x.country_name, x.title, x.rn
```

I have something special to do for the first film in each group (show the country name), so I number to know which one is the first one.



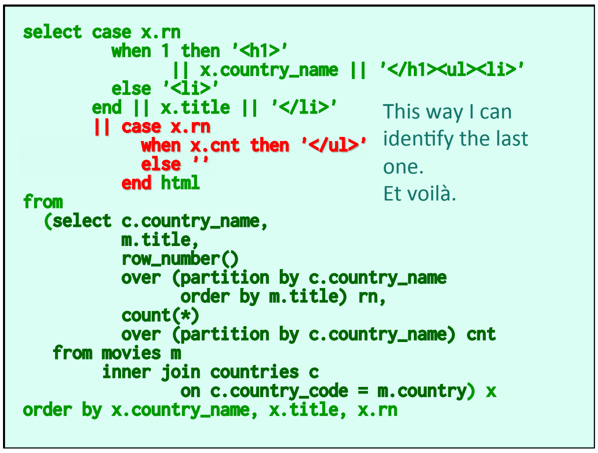
```
select case x.rn
         when 1 then '<h1>'
               || x.country_name || '</h1><ul><li>'
         else '<li>'
       end || x.title || '</li>' html
from
  (select c.country_name,
          m.title,
          row_number()
          over (partition by c.country_name
                order by m.title) rn
   from movies m
        inner join countries c
                on c.country_code = m.country) x
order by x.country_name, x.title, x.rn
```

For the first row in a group, I can display the country name. But I also need to do something special for the last one. If the first one is always number 1, the number of the last one will vary.



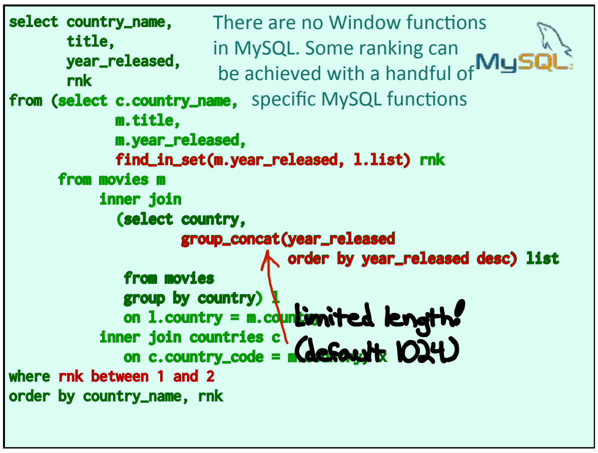
```
select case x.rn
         when 1 then '<h1>'
               || x.country_name || '</h1><ul><li>'
         else '<li>'
       end || x.title || '</li>' html
from
  (select c.country_name,
          m.title,
          row_number()
          over (partition by c.country_name
                order by m.title) rn,
          count(*)
          over (partition by c.country_name) cnt
   from movies m
        inner join countries c
                on c.country_code = m.country) x
order by x.country_name, x.title, x.rn
```

So I also count how many films I have per country.



```
select case x.rn
         when 1 then '<h1>'
               || x.country_name || '</h1><ul><li>'
         else '<li>'
       end || x.title || '</li>'
       || case x.rn
            when x.cnt then '</ul>'
            else ''
          end html
from
  (select c.country_name,
          m.title,
          row_number()
          over (partition by c.country_name
                order by m.title) rn,
          count(*)
          over (partition by c.country_name) cnt
   from movies m
        inner join countries c
                on c.country_code = m.country) x
order by x.country_name, x.title, x.rn
```

This way I can identify the last one.
Et voilà.



There are no Window functions in MySQL. Some ranking can be achieved with a handful of specific MySQL functions

```
select country_name,
       title,
       year_released,
       rnk
from (select c.country_name,
             m.title,
             m.year_released,
             find_in_set(m.year_released, l.list) rnk
      from movies m
           inner join
             (select country,
                     group_concat(year_released
                                  order by year_released desc) list
              from movies
              group by country) l
              on l.country = m.country
           inner join countries c
              on c.country_code = m.country) x
where rnk between 1 and 2
order by country_name, rnk
```

Limited length!
(default 1024)