CS307

Database Principles

Chapter 8

Update, Delete, Function, Procedure

Most contents are from Stéphane Faroult's slides

8.1 Update

Most contents are from Stéphane Faroult’s slides

Things change ...

We have talked about inserting data, lets' now see how we can update what is in the database.

Update is the command than changes column values. You can even set a non-mandatory column to NULL. The change is applied to all rows selected by the WHERE.

```
update table_name
set column_name =  , new_value
        other_col = other_val,
        ...
where ...
```

```
update us_movie_info
set title = replace(title, '''', '')
```

Without a WHERE all rows are affected.

A nobiliary particle is used in a surname or family name in many Western cultures to signal the nobility of a family.

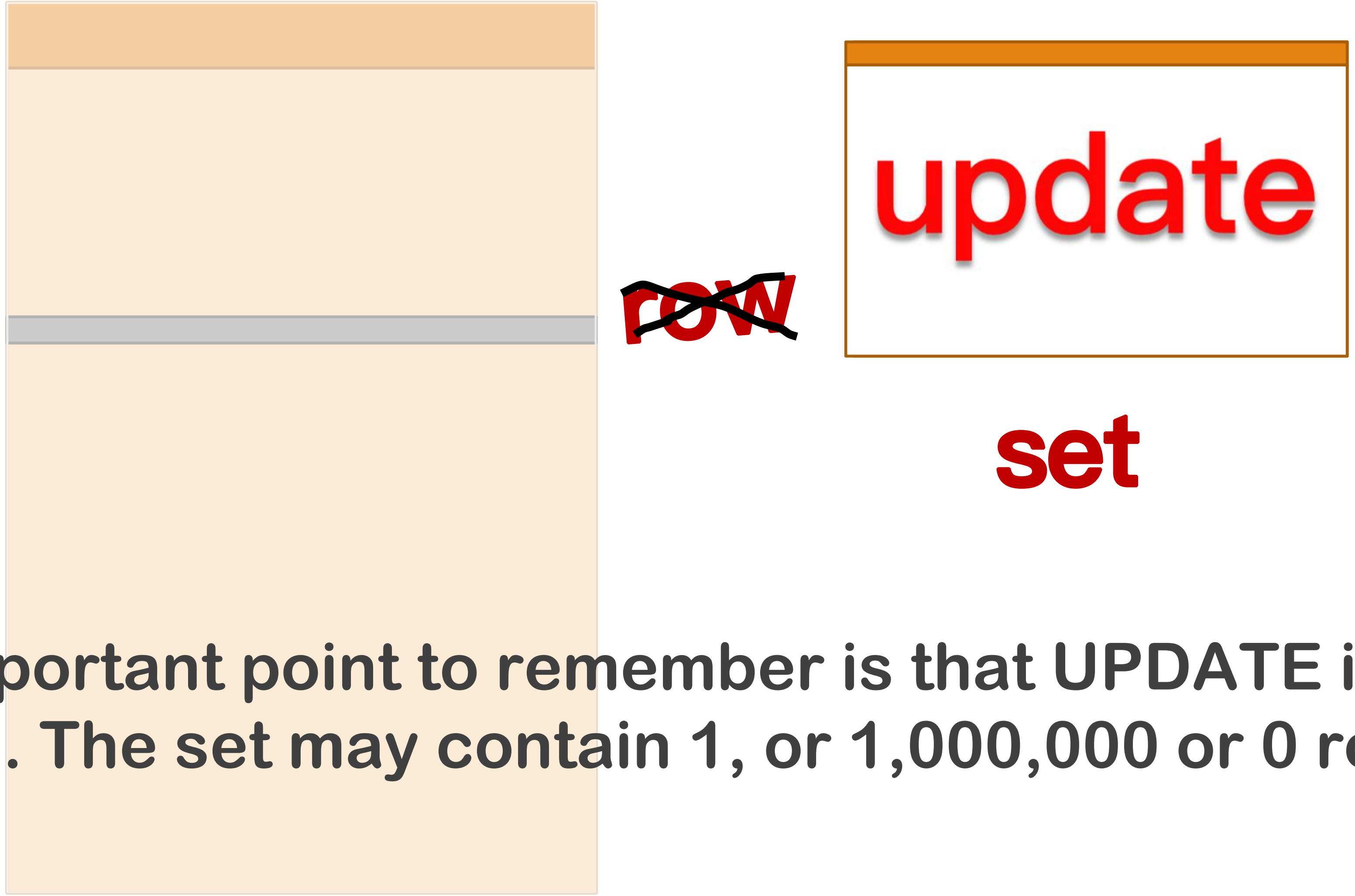
John von Neumann

We may want to modify some names in such a way as they sort as they should.

```
update people
set surname = substr(surname, 4)
               || ' (von)'
where surname like 'von %'
```

This could be used to postfix all surnames starting by 'von' with '(von)' and turn for instance 'von Stroheim' into 'Stroheim (von)'

IBM DB2 ORACLE PostgreSQL SQLite

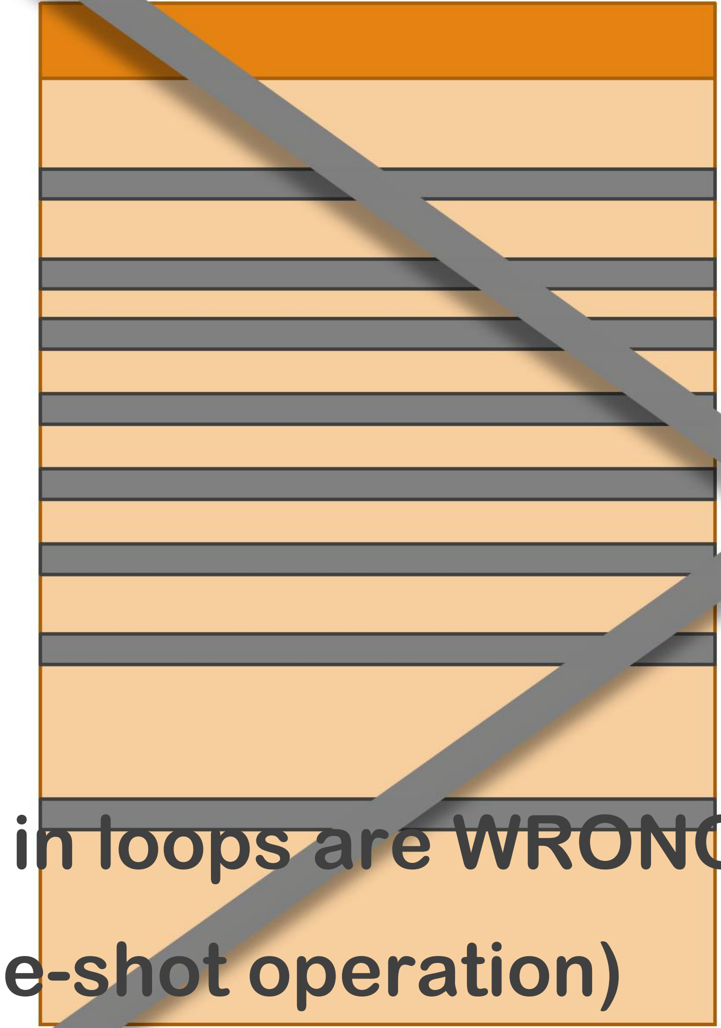


~~row~~

update

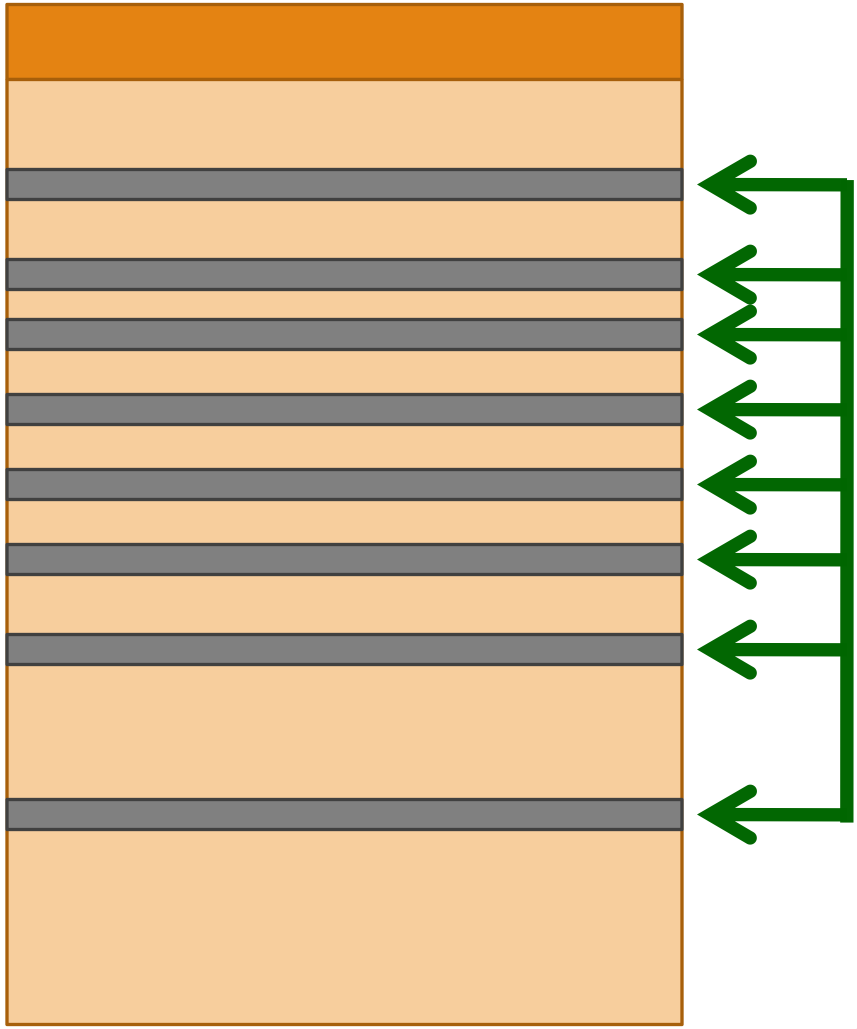
set

A very important point to remember is that UPDATE is a SET operation. The set may contain 1, or 1,000,000 or 0 rows.



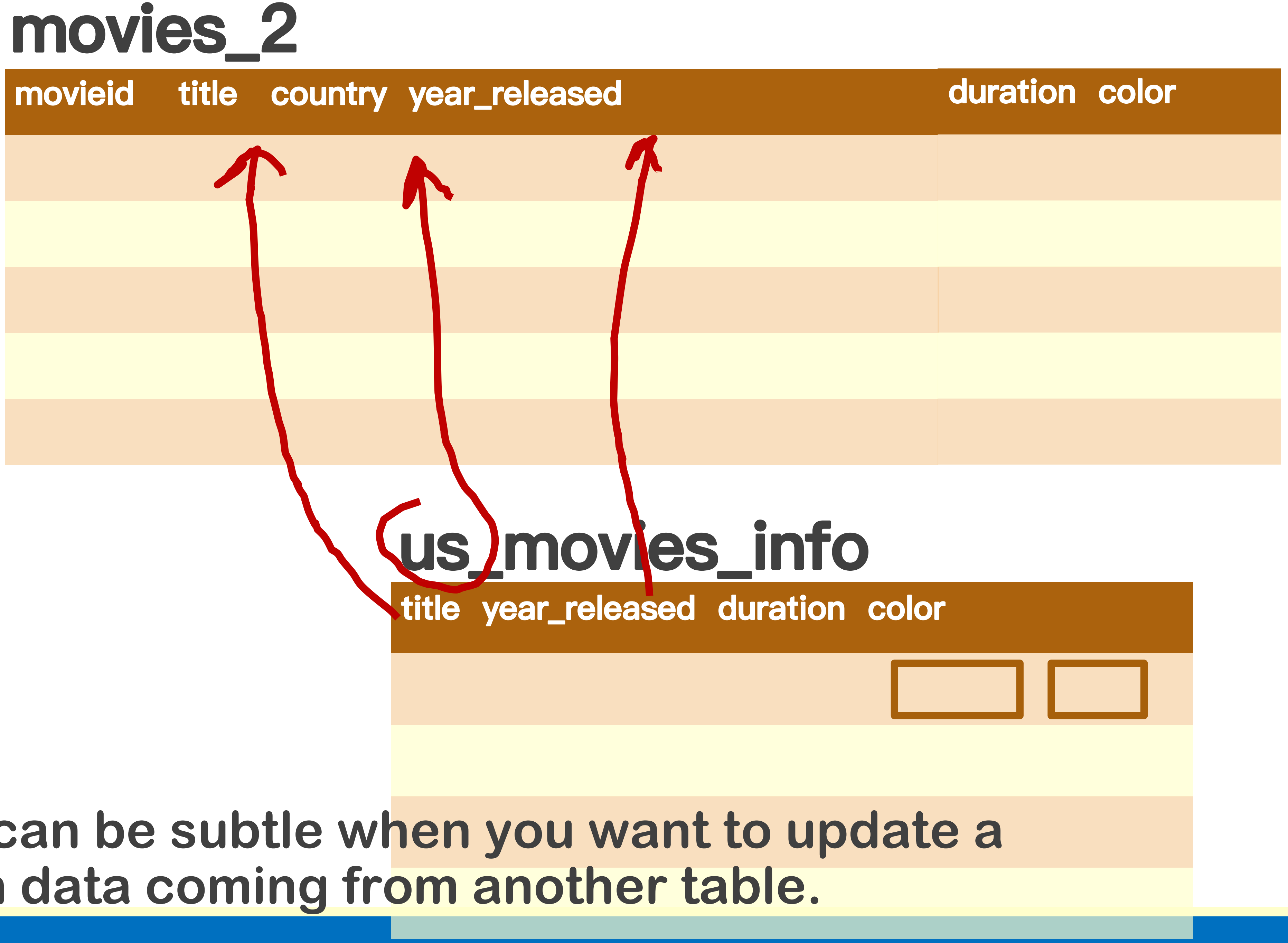
```
Loop on SELECT
    UPDATE T
    SET ...
    WHERE KEY = ...
```

Updates in loops are WRONG (and very slow compared to the one-shot operation)



```
UPDATE T
SET ...
WHERE ...
```

Think massive operations.



movies_2

movieid	title	country	year_released	duration	color

us_movies_info

title	year_released	duration	color

Updates can be subtle when you want to update a table with data coming from another table.

Like a join in a select ..... same issues with nulls and duplicates

movieid	title	country	year_released	duration	color
1234	Jiong Ma	cn	2020	120	Y

us_movies_info

title	year_released	duration	color

Not found ?

```
update movies_2
set duration = (select duration
          from us_movie_info i
          where i.title = movies_2.title
            and i.year_released = movies_2.year_released)
    color = (select case color
                 when 'C' then 'Y'
                 when 'B' then 'N'
                end color
          from us_movie_info i
          where i.title = movies_2.title
            and i.year_released = movies_2.year_released)
where country = 'us'
  and exists (select null
          from us_movie_info i2
          where i2.title = movies_2.title
            and i2.year_released = movies_2.year_released)
```

NULL

As subqueries can return NULL, you must be certain to only affect rows in your scope.

```
update movies_2
set duration = (select duration
                from us_movie_info i
                where i.title = movies_2.title
                  and i.year_released = movies_2.year_released),
       color = (select case color
                         when 'C' then 'Y'
                         when 'B' then 'N'
                        end color
                from us_movie_info i
                where i.title = movies_2.title
                  and i.year_released = movies_2.year_released)
where country = 'us'
   and exists (select null
               from us_movie_info i2
               where i2.title = movies_2.title
                 and i2.year_released = movies_2.year_released)
```

Three Queries per row processed

Not madly efficient; all subqueries are correlated (for the third query SQLite now supports the same as Oracle - next slide).

ORACLE

IBM DB2

```
update movies_2
set (duration, color) =
         (select duration,
                 case color
                   when 'C' then 'Y'
                   when 'B' then 'N'
                 end color
          from us_movie_info i
          where i.title = movies_2.title
            and i.year_released = movies_2.year_released)
where country = 'us'
   and exists (select null
              from us_movie_info i2
              where i2.title = movies_2.title
                and i2.year_released = movies_2.year_released)
```

run for each retrieved row

Oracle and DB2 both support subqueries returning several columns (SQlite also now).

ORACLE

IBM DB2

```
update movies_2
set (duration, color) =
         (select duration,
                 case color
                   when 'C' then 'Y'
                   when 'B' then 'N'
                 end color
          from us_movie_info i
          where i.title = movies_2.title
            and i.year_released = movies_2.year_released)
where country = 'us'
   and (m.title, m.year_released)
       in (select title, year_released
           from us_movie_info)
```

run for each retrieved row

Once

Oracle and DB2 both support subqueries returning several columns (SQlite also now).

```
update movies_2
set   duration = i.duration,
      color = case i.color
                when 'C' then 'Y'
                when 'B' then 'N'
              end
from us_movie_info i
where i.title = movies_2.title
  and i.year_released = movies_2.year_released
  and movies_2.country = 'us'
```

} Join

SQL Server and PostgreSQL both support the same older-join type of syntax allowing to join the updated table to the one from which we are getting data.

```
update movies_2 m
        inner join us_movie_info i
          on i.title = m.title
         and i.year_released = m.year_released
set m.duration = i.duration,
    m.color = case i.color
                when 'C' then 'Y'
                when 'B' then 'N'
              end
where m.country = 'us'
```

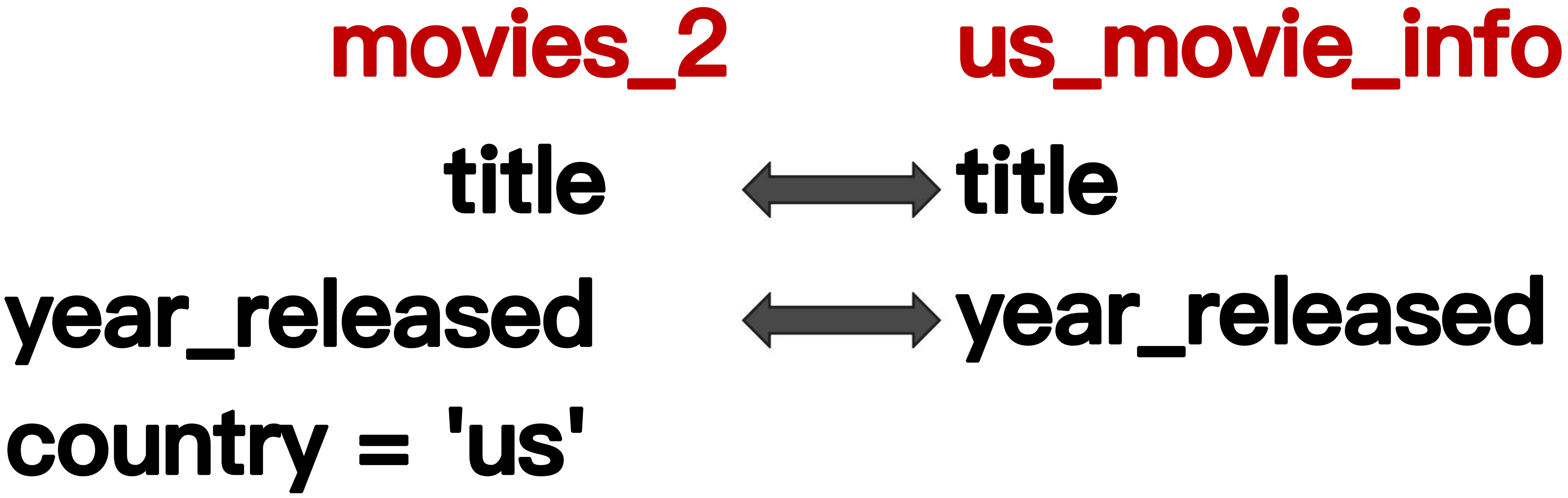
MySQL allows a join with the newer syntax.

What can happen when join conditions are

WRONG?

When you have a SELECT wrong, it only affects your query. When you have an UPDATE wrong, you can corrupt the database and later correct queries on wrong data will return wrong results.

So you really need to be extra careful.



movies_2 us_movie_info

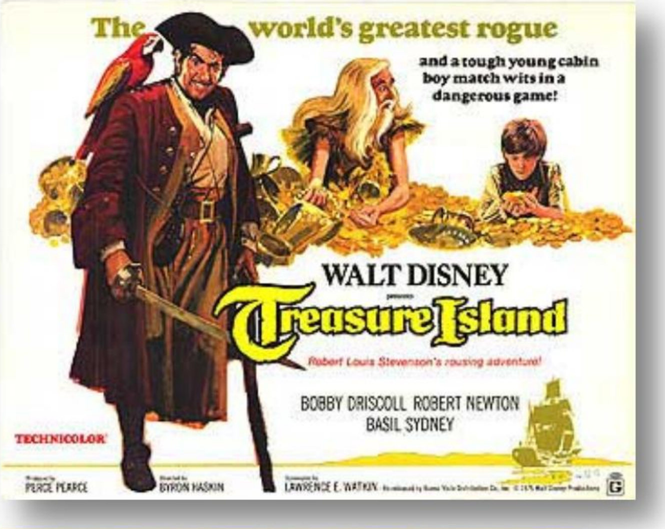
title ⟷ title

year_released ⟷ year_released

country = 'us'

Imagine for instance that we forget the join on the year and that we have remakes. What will happen?

movies_2


The world's greatest rogue
and a tough young cabin boy match wits in a dangerous game!
WALT DISNEY
presents
Treasure Island
Robert Louis Stevenson's rousing adventure!
BOBBY DRISCOLL ROBERT NEWTON
BASIL SYDNEY
TECHNICOLOR
PERCE PEARCE
BYRON HASKIN
LAWRENCE E. WATKIN
G

Wallace BEERY · Jackie COOPER
ROBERT LOUIS STEVENSON'S
TREASURE ISLAND
Lionel BARRYMORE
KRUGER STONE
BRUCE SALE
Metro-Goldwyn-Mayer
VICTOR FLEMING
HUNT STROMBERG

us_movie_info

Wallace BEERY · Jackie COOPER
ROBERT LOUIS STEVENSON'S
TREASURE ISLAND
Lionel BARRYMORE
KRUGER STONE
BRUCE SALE
Metro-Goldwyn-Mayer
VICTOR FLEMING
HUNT STROMBERG

Let's first say that we only have remakes in the table that we update.

Running a SELECT shows what happens.

```
select m.title, m.year_released,
        i.year_released, i.duration, i.color
from movies_2 m
      inner join us_movie_info i
         on i.title = m.title
where m.title like 'Treasure%'
```

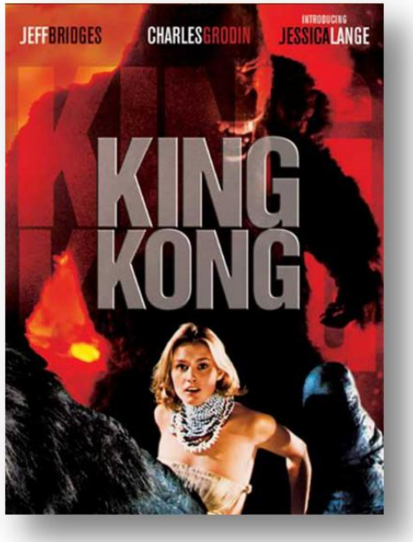
One row from the source table will be associated with both films.

```
title              year_released   year_released   duration   color
------------------ ---------------- ---------------- ----------- ------
Treasure Island              1934            1934        103 B
Treasure Island              1950            1934        103 B
```

movies_2

us_movie_info

movies_2



us_movie_info

JEFF BRIDGES
CHARLES GRODIN
INTRODUCING JESSICA LANGE
KING KONG
RKO Radio
DAVID O. SELZNICK Executive Producer
KING KONG
with
FAY WRAY • ROBT. ARMSTRONG
BRUCE CABOT
A COOPER-SCHOEDSACK PRODUCTION
FROM AN IDEA CONCEIVED BY
EDGAR WALLACE AND MERIAN C. COOPER

Now let's see what happens if we have remakes in the source table, but not in the one that we update

Once again, a SELECT shows what happens.

```
select m.title, m.year_released,
       i.year_released, i.duration, i.color
from movies_2 m
     inner join us_movie_info i
        on i.title = m.title
where m.title = 'King Kong'
```

The same row will be updated twice. What will remain is the last update. Heads or tails?

```
title              year_released    year_released    duration   color
------------------ ---------------- ---------------- ---------- ------
King Kong                    1976             1933        100   B
King Kong                    1976             1976        134   C
```

movies_2

us_movie_info

Subquery

```
update movies_2
set duration =
      (select duration
        from us_movie_info i
        where i.title = movies_2.title)
...
```

2 rows

FAILURE

Note that a subquery returning more than one row would generate an error.

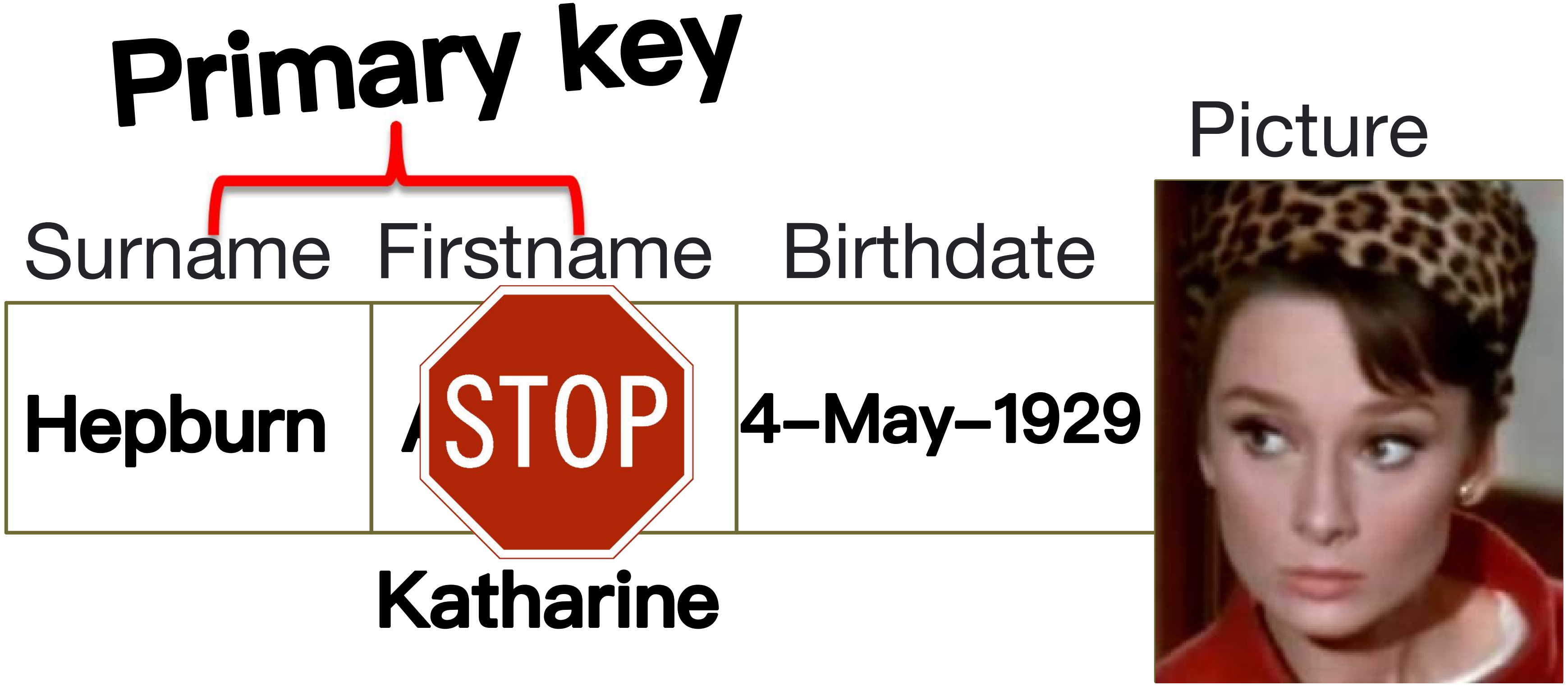
Join

```
title               year_released   year_released   duration   color
------------------- --------------- --------------- ---------- ------
King Kong                      1976            1933        100 B
King Kong                      1976            1976        134 C
```

A join won't fail, and just update randomly.

Same Rules Apply for UPDATE as for SELECT

Except that as already stated, an update can change the data wrongly.



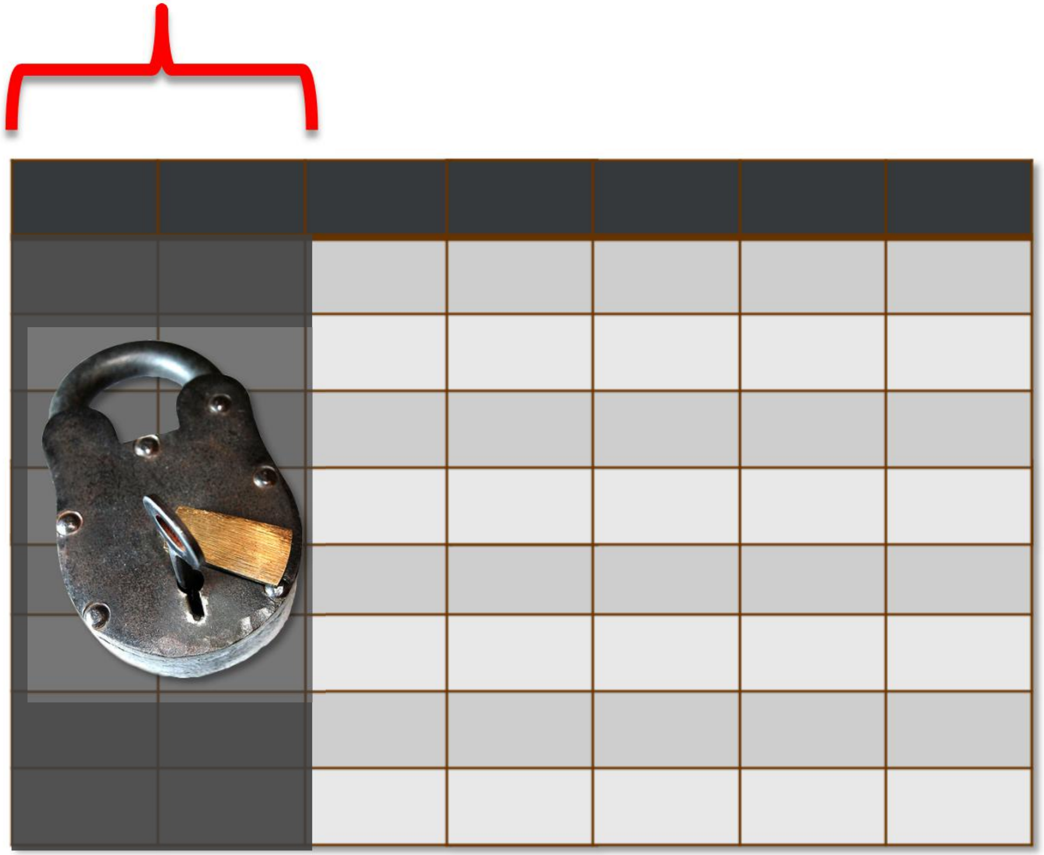
Primary key

Surname	Firstname	Birthdate	Picture
Hepburn	A STOP	4–May–1929	

Katharine

You are reminded that if a regular attribute can be updated, it's usually forbidden to update a key - it's the identifier. You cannot change an identifier. You can only delete the row and insert another.

Primary key



Update-wise, a primary key is locked.

Off-limits.

Picture by Andrew Magill

Microsoft SQL Server ORACLE IBM DB2

Update or Insert

```
merge into movies_2 m
using (select 'us' as country,
              title,
              year_released,
              duration,
              case color
                when 'C' then 'Y'
                when 'B' then 'N'
              end as color
       from us_movie_info) i
      on (i.country = m.country
     and i.title = m.title
     and i.year_released = m.year_released)
when matched then
     update
     set m.duration = i.duration,
         m.color = i.color
when not matched then
     insert(title, year_released, country, duration, color)
     values(i.title, i.year_released, i.country, i.duration, i.color)
```

A interesting operation would be to update a film we know, and insert it if we don't. That's the purpose of MERGE.

Update or Insert

```
insert into movies_2(title, year_released,
                     country, duration, color)
select title, year_released, country, duration, color
from (select title,
             year_released,
             'us' as country,
             duration,
             case color
               when 'C' then 'Y'
               when 'B' then 'N'
             end color
      from us_movie_info) i
on duplicate key update
 movies_2.duration = i.duration,
 movies_2.color = i.color
```

MySQL can catch an insert that fails because the row is already here, and turn on the fly the insert into an update.

Update or Insert

```
insert   or replace   into movies_2(title, year_released,
                              country, duration, color)
select title, year_released, country, duration, color
from (select title,
        year_released,
        'us' as country,
        duration,
        case color
          when 'C' then 'Y'
          when 'B' then 'N'
        end color
   from us_movie_info) i
```

SQLite allows something similar with a simpler (but less flexible) syntax.

Beware, because it deletes a row and creates a new one, foreign keys may not like it.

Update ~~or~~ then Insert

Update

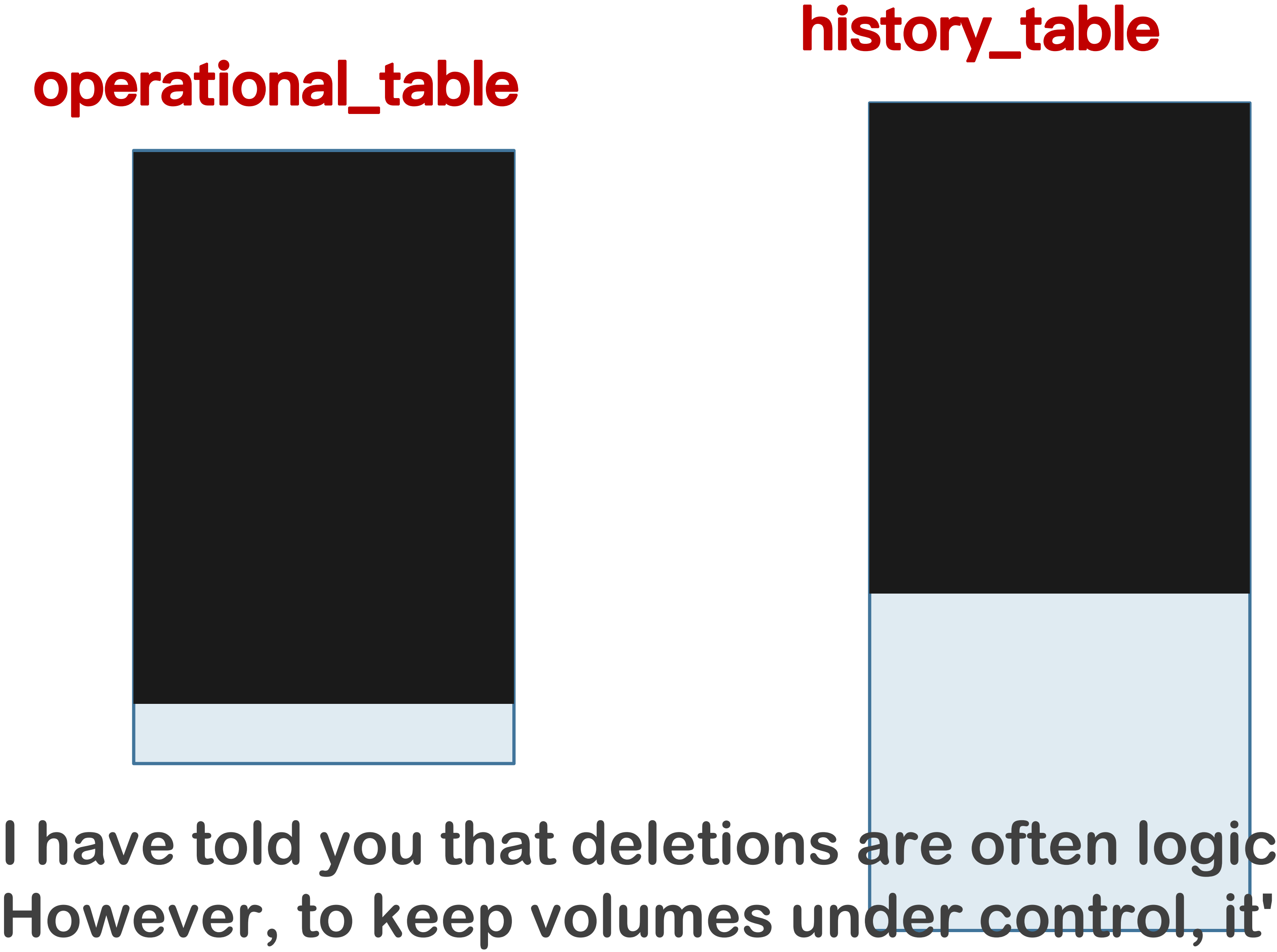
+

```
insert into movies_2(title, year_released, country,
                     duration, color)
select i.title, i.year_released, 'us', i.duration,
       case i.color
         when 'C' then 'Y'
         when 'B' then 'N'
       end
from us_movie_info i
     left outer join movies_2 m
       on m.title = i.title
      and m.year_released = i.year_released
      and m.country = 'us'
where m.movieid is null
```

When none of the above is available, you should try to update, and if nothing is affected insert.

8.2 Delete

Most contents are from Stéphane Faroult’s slides



I have told you that deletions are often logical (flagged rows). However, to keep volumes under control, it's frequent to copy old rows to a history table, then delete them from the "active" table.

```
delete from table_name
where ...
```

If you omit the WHERE clause, then (as with UPDATE) the statement affects all rows and you

Empty **table_name** !

But of course you NEVER work in autocommit mode and always execute a big update or delete in a transaction, don't you?

rollback

That's when you feel grateful for some features

Flickr: Giò-S.p.o.t.s.

TRUNCATE

As DELETE saves data for rollback before removing it, it can be slow. There is a TRUNCATE (without a WHERE clause) that cannot be rolled back and is far more efficient. It's better not to use it.

Constraints

= guarantee

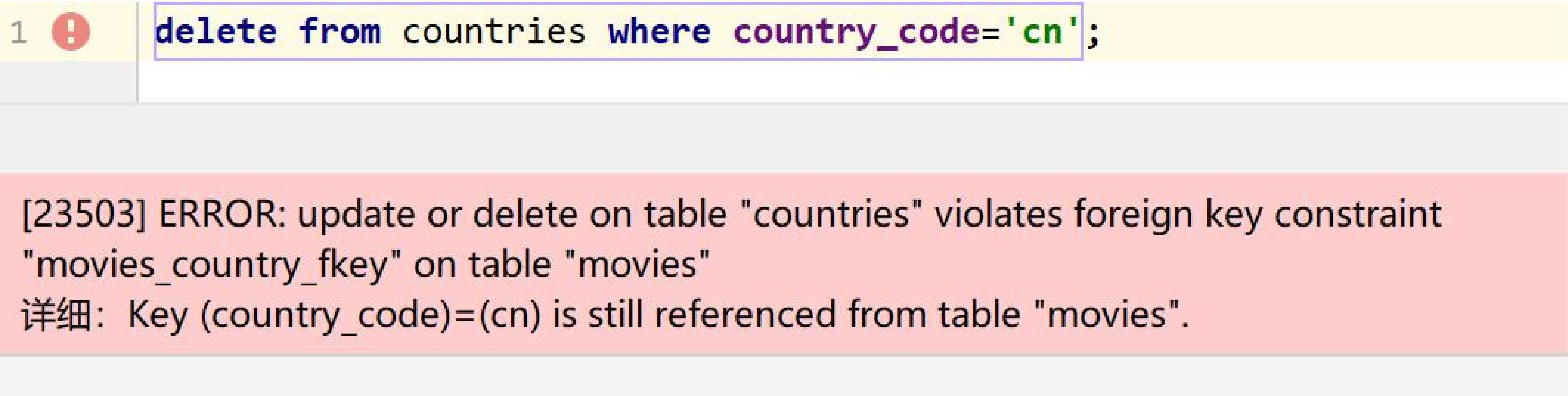
One important point with constraints (foreign keys in particular) is that they guarantee that data remains consistent. They don't only work with INSERT, but with UPDATE and DELETE as well.

Try to delete rows from table **countries**

For instance, you can delete a country for which there are no movies. As soon as you have one movie, you are prevented from deleting the country otherwise the foreign key on table MOVIES would no longer work for films from that country.

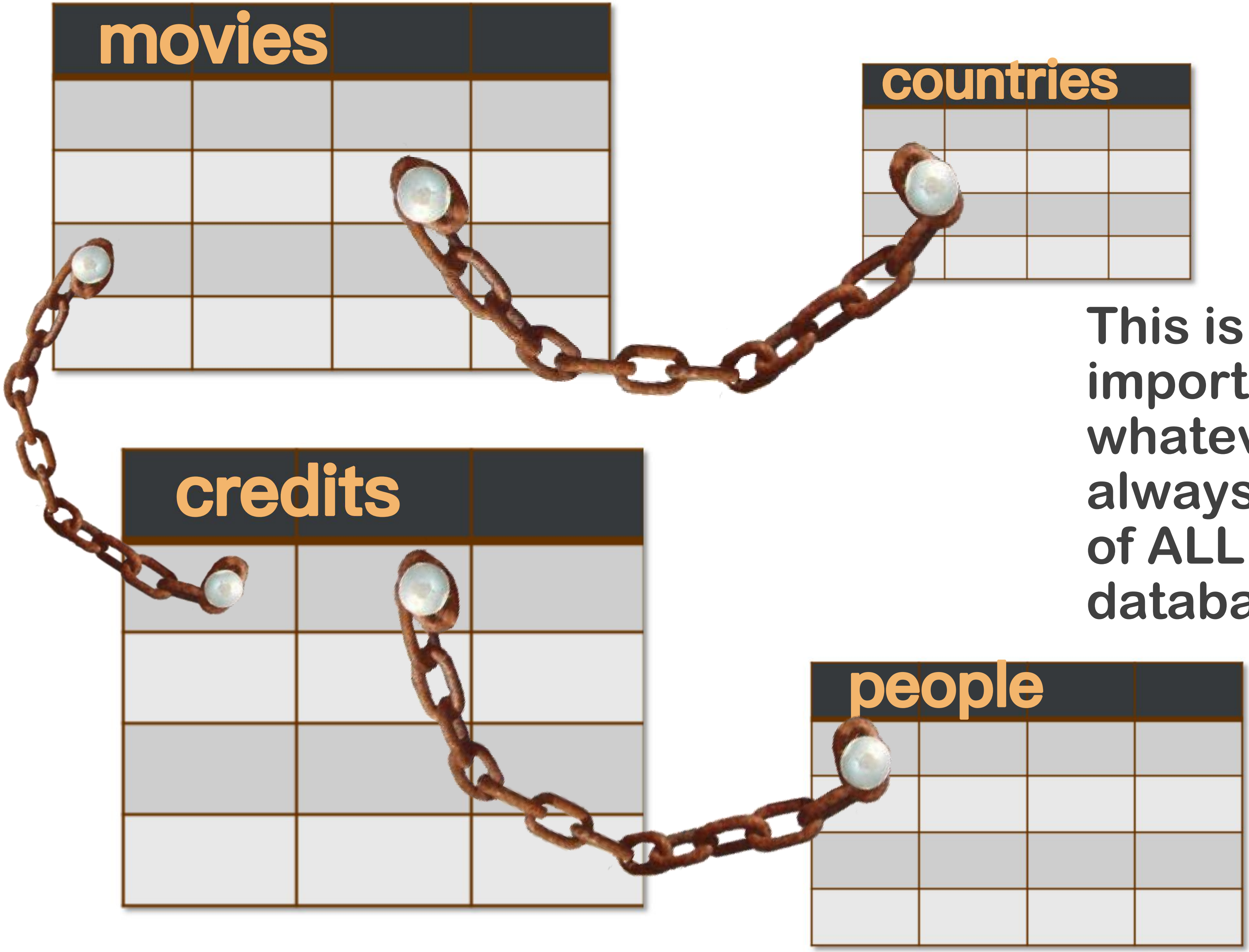
To delete the row for China in table countries.

The constraint will prevent you to do that.



```sql
delete from countries where country_code='cn';
```

```
[23503] ERROR: update or delete on table "countries" violates foreign key constraint
"movies_country_fkey" on table "movies"
详细：Key (country_code)=(cn) is still referenced from table "movies".
```



This is why constraints are so important: they ensure that whatever happens, you'll always be able to make sense of ALL pieces of data in your database.

8.3 Functions

Most contents are from Stéphane Faroult's slides

Functions

Build-in functions:
lower()
upper()
substring()
trim()
…

Most DBMS (the exception is SQLite, not a true DBMS) implement a built-in, SQL-based programming language, that can be used when a declarative language is no longer enough. Let's start with the simplest thing, defining functions.

Sort ??

first_name	surname
Erich	von Stroheim

I gave an update example in which I was modifying every name starting with 'von ' so that they sort properly.

```
select first_name || ' ' || surname as full_name
from people;
```

Erich **Stroheim (von)**

Sorting is one thing, but if I ever want to display the full name of a person by concatenating first_name and surname, it will look weird for von Stroheim. What I really want to see is

Erich von Stroheim

first name?

Left parenthesis?

```
case
  when first_name is null then ''
  else   first_name || ' '
end
|| case position('(' in surname)
     when 0 then surname
     else      trim(')' from       substr(surname,
                                          position('(' in surname) + 1)
               || ' '
               || trim(substr(surname, 1,
                              position('(' in surname) – 1))
     end full_name
```

first_name	surname
Erich	Stroheim (von)

Erich von Stroheim

Flickr:Kevin Rawlings

Needless to say, whenever you have painfully written something as complicated, which is pretty generic, you'd rather not copy and paste the code every time you need it.

```
case
  when first_name is null then ''
  else first_name || ' '
 end
 || case position('(' in surname)
      when 0 then surname
      else trim(')' from substr(surname,
                                   position('(' in surname) + 1))
            || ' '
            || trim(substr(surname, 1,
                             position('(' in surname)  – 1))
    end full_name
```

STORE FOR

REUSE

You'd like to store the expression and reuse it in another context. In fact you can.

Here is a PostgreSQL example

```
create function full_name(p_fname varchar, p_sname varchar)
returns varchar
as $$
begin
  return case
           when p_fname is null then ''
           else p_fname || ' '
         end |
           case position('(' in p_sname)
              when 0 then p_sname
              else trim('0' from substr(p_sname,
                                        position('(' in p_sname) + 1))
                   ||' '
                   || trim(substr(p_sname, 1,
                                   position('(' in p_sname) – 1))
           end;
end;
$$ language plpgsql;
```

```
select full_name(first_name, surname) as name,
       born, died
from people
order by surname
```

Once your function is created, you can use it as if it were any built-in function.

Note that you usually have to write your functions in the provided language for safety: a badly coded C function could take down a whole server, corrupt data, etc. The provided language provides a kind of sand-boxed environment.

Procedural extensions to SQL

Microsoft SQL Server	T–SQL
MySQL	(no name)
ORACLE	PL/SQL
PostgreSQL	PL/PGSQL
IBM DB2	SQL PL
SQLite	nothing ...

You can use C or any language with SQLite. If you crash your program, it only affects you.

Procedural ?

variables

conditions

loops

arrays

error management

… TRUE PROGRAMMING LANGUAGE

Procedural extensions provide all the bells and whistles of true programming languages (they were often inspired by programming languages such as PL/I or ADA). They are a mixed blessing, because they often incite programmers to do the wrong things with them.

They also support all DML statements (no always DDL, but you can cheat)

```
select col1, col2, ...
into local_var1, local_var2, ...
from ...
```

+ CURSORS

To retrieve data from the database into your variables, you can use SELECT ... INTO ... if your query returns a single row, or you can use cursors, which are basically "row variables" that are used for iterating over what a query returns.

Cultural mismatch

row–by–row

set processing

And here we have a problem, because there is a big cultural gap between the relational mindset and procedural processing.

Flickr: Jeff Sandquist

Bad example

In the category "never, ever do that even if you encounter it often" there is the infamous "look-up" function that returns for instance the label associated with a value.

Because it's a procedure stored inside the database, many developers believe in good faith that's how things should be done. Definitely no.

```
select country_name(country_code), title, ...
from movies
where ...
```

```
create function country_name(p_code varchar2)
return countries.country_name%type
as
   v_name   countries.country_name%type;
begin
  select country_name
  into v_name
  from countries
  where country_code = p_code;
  return v_name;
end;
```

NO

Нет

禁

Không

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

Why is it bad? You can retrieve the same data with a join. I have hardly talked about the query optimizer so far but there are many ways to perform a join, some of which are particularly efficient on big volumes. A look-up function forces a "one row at a time" join which in most cases will be dreadful. A function shouldn't query the database.

SQL FIRST!

Tom Kyte, who is Senior Technology Architect at Oracle, says that his mantra is:

- You should do it in a single SQL statement if at all possible.

- If you cannot do it in a single SQL statement, then do it in PL/SQL (as little PL/SQL as possible!)

What I suggest:

- You should ask for help from someone more experienced than you, Google, forums, etc.