

# CS213

# Principles of Database Systems(H)

## Chapter 11 View

---

Shiqi YU 于仕琪

yusq@sustech.edu.cn

Most contents are from Stéphane Faroult's slides

# 11.1 View

---

Shiqi Yu 于仕琪

yusq@sustech.edu.cn

Most contents are from Stéphane Faroult's slides

```
main() {
```

```
    my_func()
```

```
    my_func()
```

```
    my_func()
```

```
}
```

```
my_func() {
```

```
}
```

We have seen stored functions, but these functions are just returning numbers, strings or dates. Can we have relational functions, allowing to **reuse relational operations** as we would reuse code in a program?

A close-up photograph of a man's face as he looks through a pair of black binoculars. He is wearing a red baseball cap with the words "HEART W..." visible on the front. He has a mustache and is wearing a dark green long-sleeved shirt. The background is a bright, overexposed sky.

It's possible, and a "recorded  
relational operation" is called a

VIEW

```
create view viewname (col1, ..., coln)
as
select ...
```

In practice (theory is a bit more complicated) there isn't much to a view: it's basically a named query. If the query is correct, it should return a valid relation, so why not consider it as if it were a table? You can optionally rename columns after the view name (if you don't, the view uses column names from the query result)

# View

A view can be as a complicated query as you want, and will usually return something that isn't as normalized as your tables, but easier to understand.

tableA



tableB

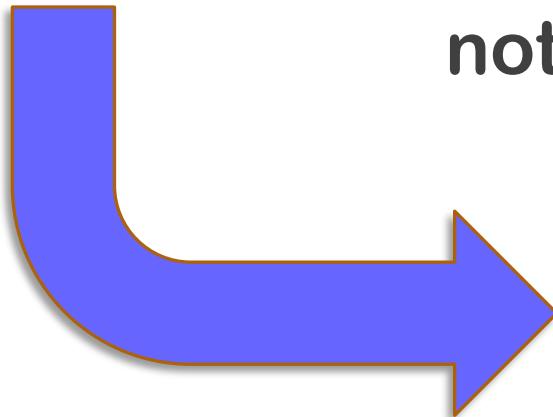


tableC



# Tables = variables

rows



Remember that tables are relational variables, and that the rows they contain are nothing more than their "value"

values

# View changed

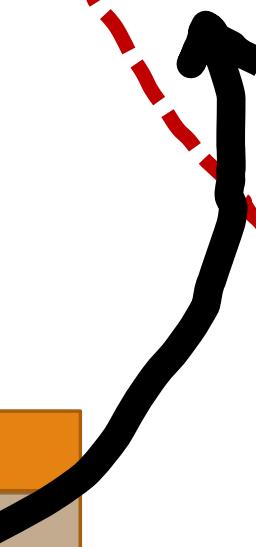
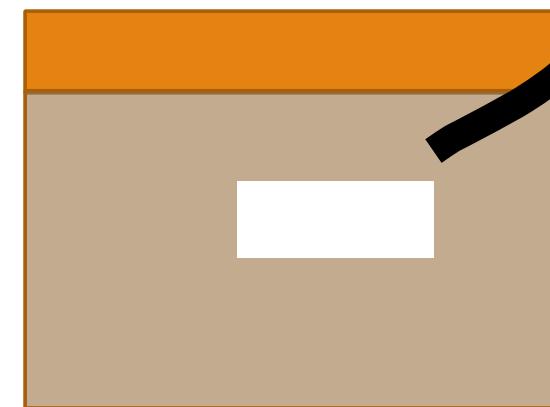
tableA



tableB



tableC



In the same way that you get a different result from a function when you change the values of variables that you pass to it, when you change something in a table you change its value and it's reflected in the view, that takes a different value too.

movies



countries



# View

Let's create a view in the film database that shows the name, rather than the code, of the country.

```
create view vmovies
as select m.movieid,
          m.title,
          m.year_released,
          c.country_name
     from movies m
       inner join countries c
      on c.country_code = m.country
```

```
select *\nfrom vmovies\nwhere country_name = 'Italy'
```

Once the view is created, I can query the view exactly as if it were a table; nothing says that it's a view, except the name that \*I\* have chosen. I like to give a special name to views to make it clear that it's a view (discussion about practical differences between views and tables comes soon) but I could have masked a change in table design to allow old programs to run by having a changed table T renamed T\_V2 and creating a view T rendering the old version.

Result-wise, the previous query is strictly equivalent to this one.

```
select *
from (select m.movieid,
            m.title,
            m.year_released,
            c.country_name
      from movies m
      inner join countries c
              on c.country_code = m.country)
      vmovies
  where country_name = 'Italy'
```

Some optimizers are able to push the condition up into the view.

---

However, there is far more than this to views. As I have said earlier, views are just the relational equivalent of functions: the ability to store (and reuse) a relational expression, in other words something that **returns a relation** and not simply a value like what you usually do with a stored function.

If we step back to design issues, you remember that modelling a database is basically distributing data between normalized tables, and there are often ways of organizing data that are more suitable for a given application. In some respect, views provide a way of **creating an "alternate model"**.

---

What is important is that views are permanent objects in the database - needless to say, their content will change with the data in the underlying tables, but the structure will remain constant and can be described in the same way as the structure of a table can be described: columns are typed.

Beware that columns are the one in tables when the view was created. Columns added later to tables in the view won't be added even if the view was created with **SELECT \*** (bad practice)

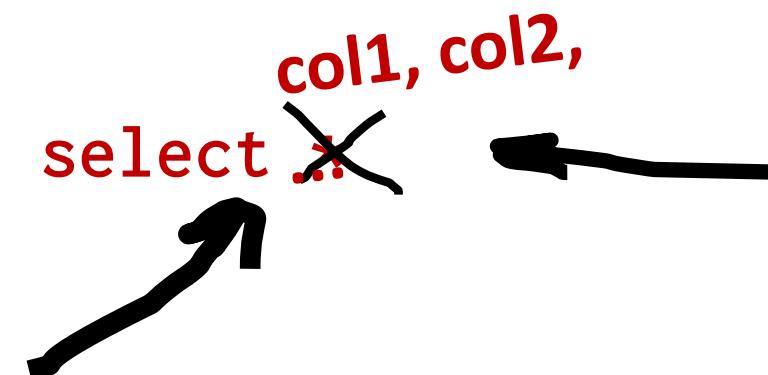
# Permanent object

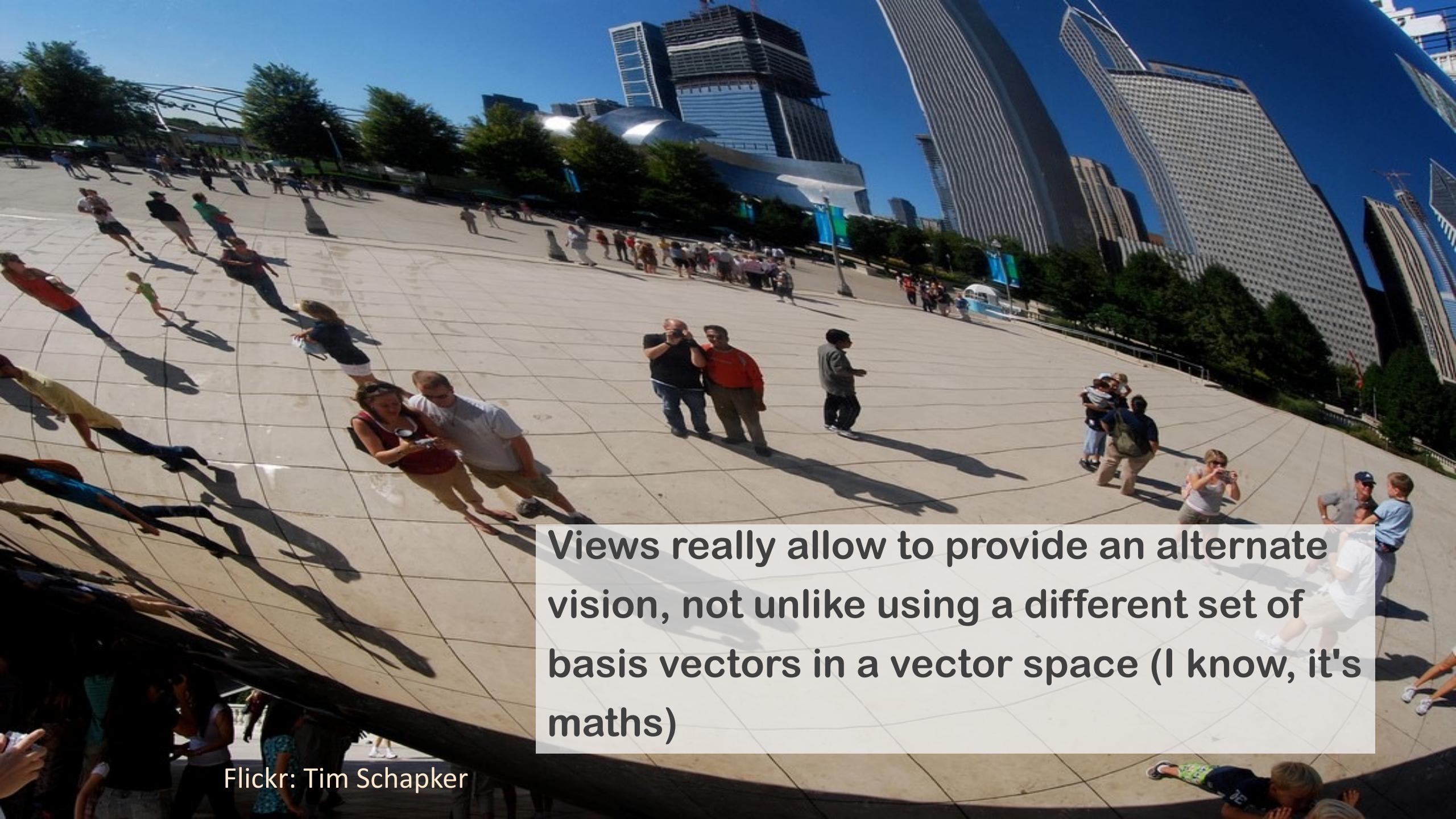
Permanent structure

Describe columns at creation !

Bad anyway

select ~~col1, col2,~~





Views really allow to provide an alternate vision, not unlike using a different set of basis vectors in a vector space (I know, it's maths)

Flickr: Tim Schapker

---

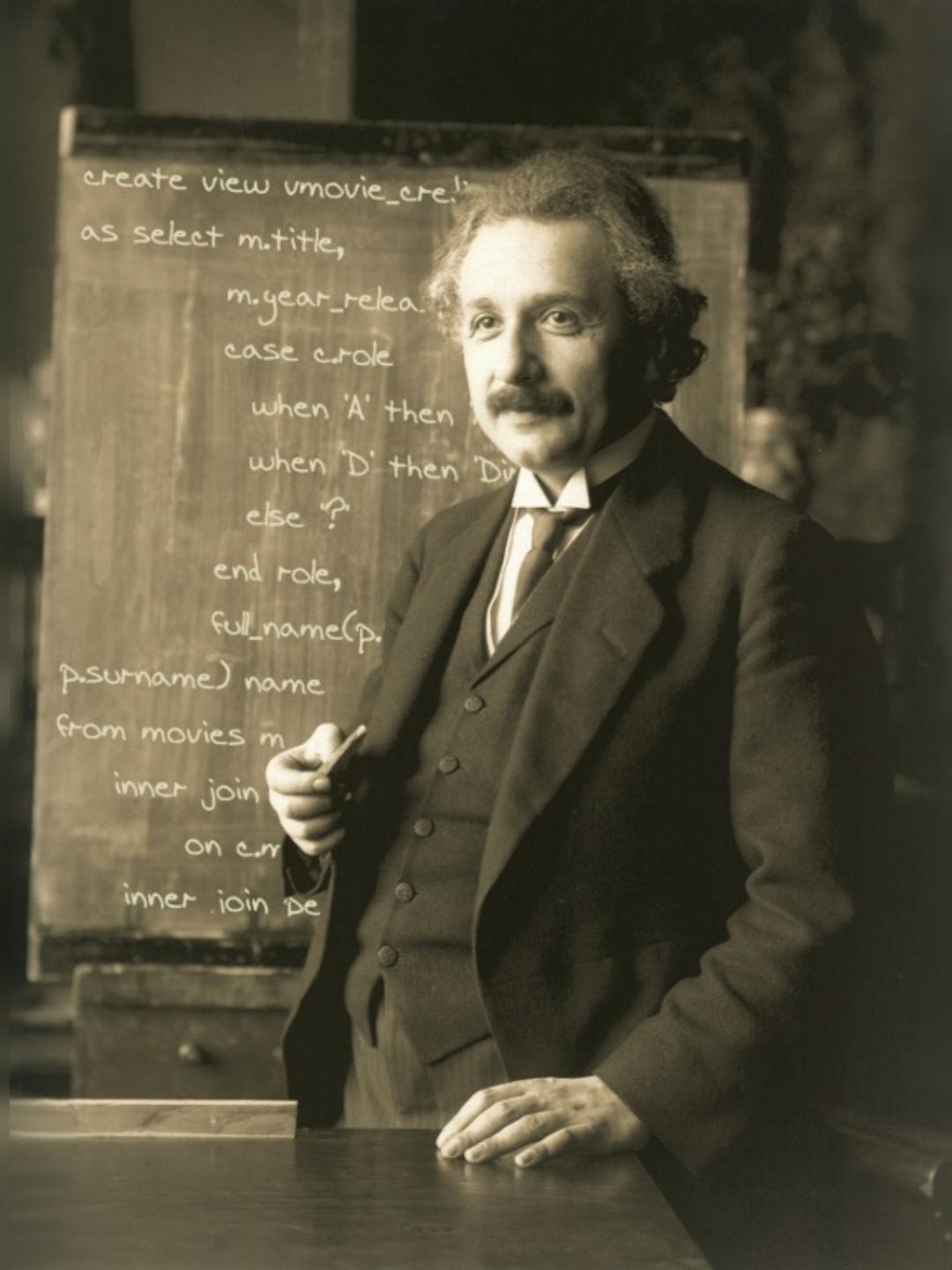
In real life, views are much used for simplifying queries. Many business reports are based on the same set of hairy joins, with just variations on the columns that you aggregate or order by.

Somehow, views allow to come back to that old fancy of the early days of SQL, having something that anybody can query with simple commands, without having to master the intricacies of the querying language.

# Simplify queries

Make the sharp guys write  
the hard stuff, then use  
cheap code-monkeys to  
wrap some basic things  
around... That's more or  
less the idea (but it's rarely  
presented like this).

# Leverage the skills of **THE BEST** **SQL CODERS**



```
select *  
from vmovies  
where country_name = 'Italy'
```

This is something that a cheap beginner  
completely ignorant of databases should be able  
to write after having been briefed for about three  
minutes.

---

**Looks like a table**

**Tastes like a table**

**But**

It's again an area  
where theory is  
bruised by practice.



# 11.2 Deeper in Views

---

Shiqi Yu 于仕琪

yusq@sustech.edu.cn

Most contents are from Stéphane Faroult's slides

```
create view vmovie_credits
as select m.title,
          m.year_released release,
          case c.credited_as
            when 'A' then 'Actor'
            when 'D' then 'Director'
            else '?'
          end duty,
          full_name(p.first_name, p.surname) name
from movies m
  inner join credits c
    on c.movieid = m.movieid
  inner join people p
    on p.peopleid = c.peopleid
```

Let's say that we have this view, which nicely displays film credits, including people names like 'Erich von Stroheim' as they should appear.

# vmovie\_credits

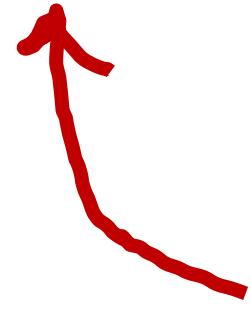
title	release	duty	name
Casablanca	1942	Director	Michael Curtiz
Casablanca	1942	Actor	Humphrey Bogart
Casablanca	1942	Actor	Ingrid Bergman
Casablanca	1942	Actor	Conrad Veidt
Casablanca	1942	Actor	Claude Rains
...	...	...	...

When we query the view, it looks really good and user-friendly. Well, it actually depend on HOW we query it, and on which column. Querying by title will be fine.

```
create view vmovie_credits
as select m.title,
         m.year_released release,
         case c.credited_as
             when 'A' then 'Actor'
             when 'D' then 'Director'
             else '?'
         end duty,
         full_name(p.first_name, p.surname) name
from movies m
    inner join credits c
        on c.movieid = m.movieid
    inner join people p
        on p.peopleid = c.peopleid
```

There are times, though, when all the benevolence of the optimizer cannot do anything for you. You may remember how awful function `full_name()` is

```
select *\nfrom vmovie_credits\nwhere name = 'Humphrey Bogart'
```

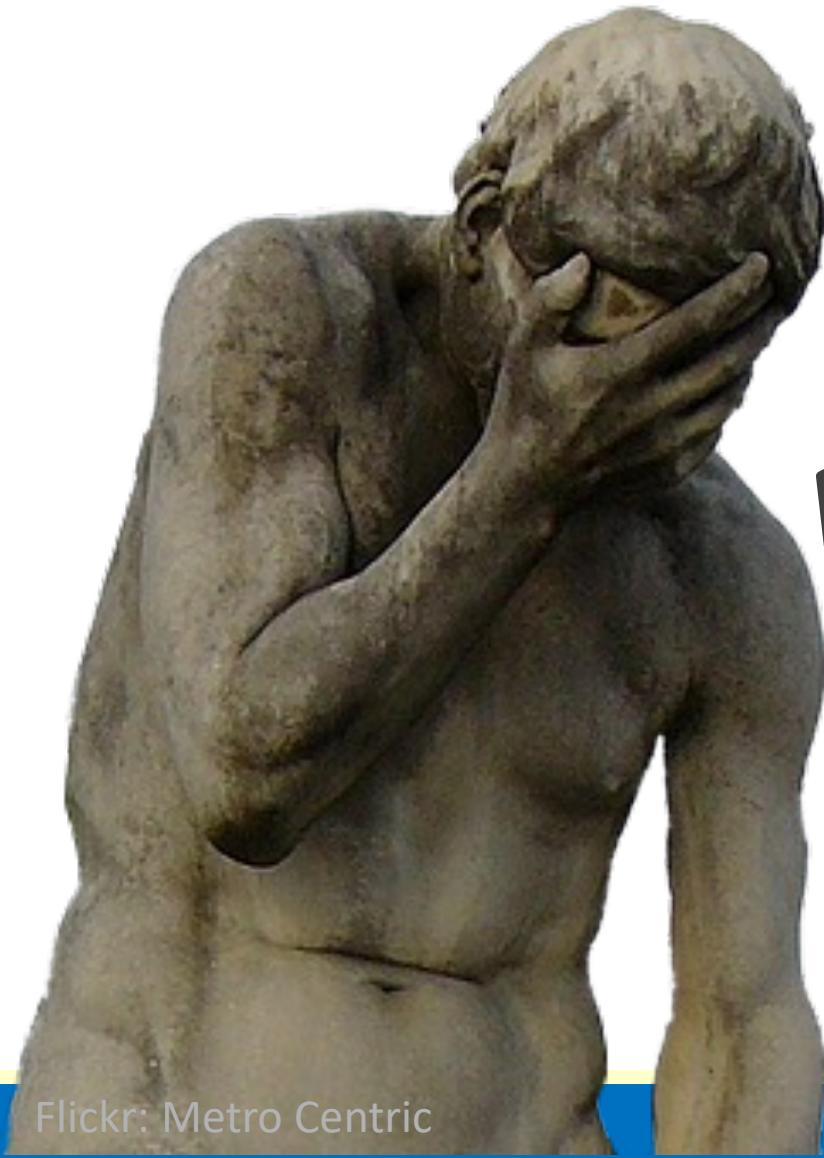


## Dreadful expression

If you are writing something like this, what looks like a column (NAME) is in fact the result of a function. There is no way the index on (SURNAME, FIRST\_NAME) can be used. We'll have to scan the full table, compute the function, and compare its result to the constant. Unless you do some tricky stuff to index in a way or the other the result of the function (not always possible).

```
select *\nfrom vmovie_credits\nwhere name = 'Humphrey Bogart'
```

# Indexes?



# VIEWS

## Hide complexity

The problem with views is that as long as you haven't seen how they have been defined, you have no idea how complex they may be. They may be fairly innocuous, or they may be queries of death (they often are)

```
select distinct title  
from vmovie_credits
```

Difficulties usually increase sharply as a young developer gets with time more confident, not to say bold, with SQL. Being so accustomed to working with this convenient "table", VMOVIES\_CREDITS (it may not bear a name that makes it obvious it's a view), the developer may think of this as a way to return all the different titles in the database. Technically speaking, it will return the desired result, and it may even do it reasonably fast.

```
select distinct title  
from  
(select m.title,  
    m.year_released release,  
    case c.credited_as  
        when 'A' then 'Actor'  
        when 'D' then 'Director'  
        else '?'  
    end duty,  
    full_name(p.first_name, p.last_name) name  
  
from movies m  
inner join credits c  
    on c.movieid = m.movieid  
inner join people p  
on p.peopleid = c.peopleid) vmovie_credits
```

**Lot of useless work  
for what we want**

**Do we really  
want the join?**

# Scalability

And here we are coming to one of the great issues with databases and information systems generally speaking, namely the ability to deliver response times that remain acceptable when the number of users, data volumes, or both, sharply increase.

The computer system of any retailer must survive Black Friday in the US or 11/11 in China.

Computing power is always

LIMITED

And the problem is that it doesn't matter how big and powerful your computers are, computing power will always be a limited resource.

---

**Slower** query to retrieve the same data



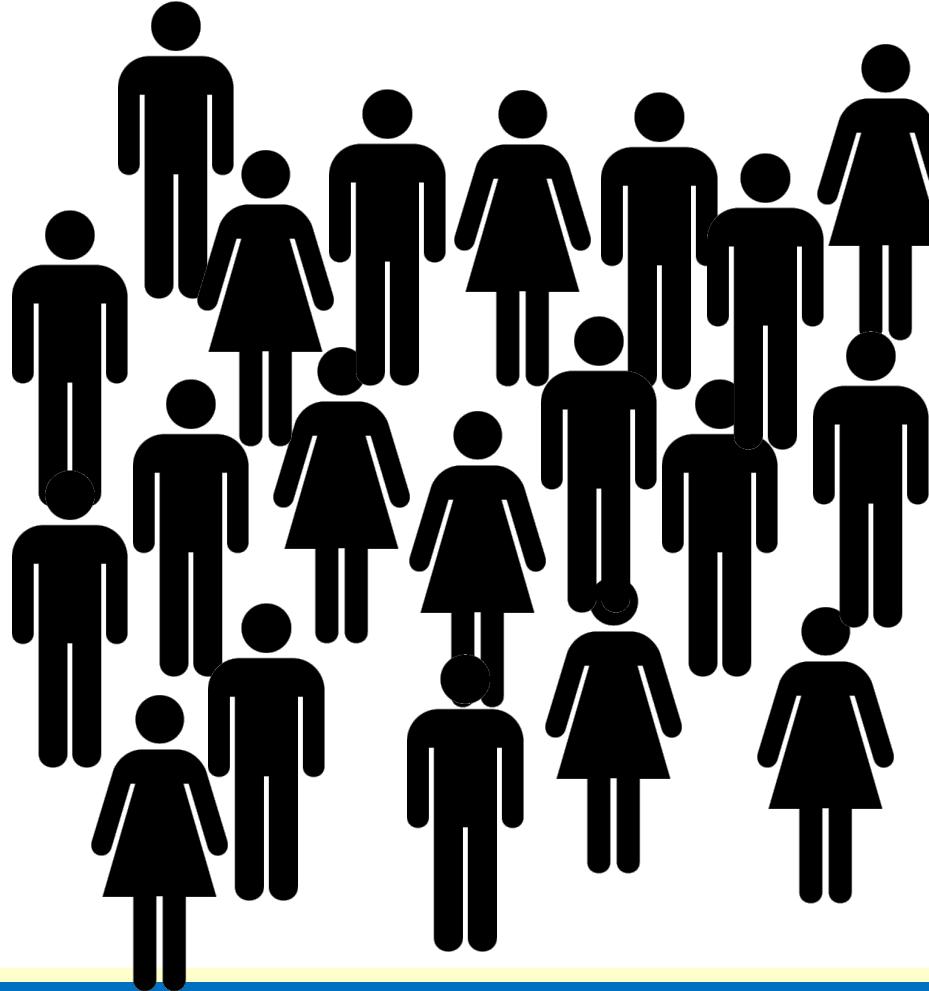
**Fewer** simultaneous users served

You will only be able to serve **\*that\*** many users simultaneously.

You don't want to see everything crawl during peak time.

# Query table

## Query view

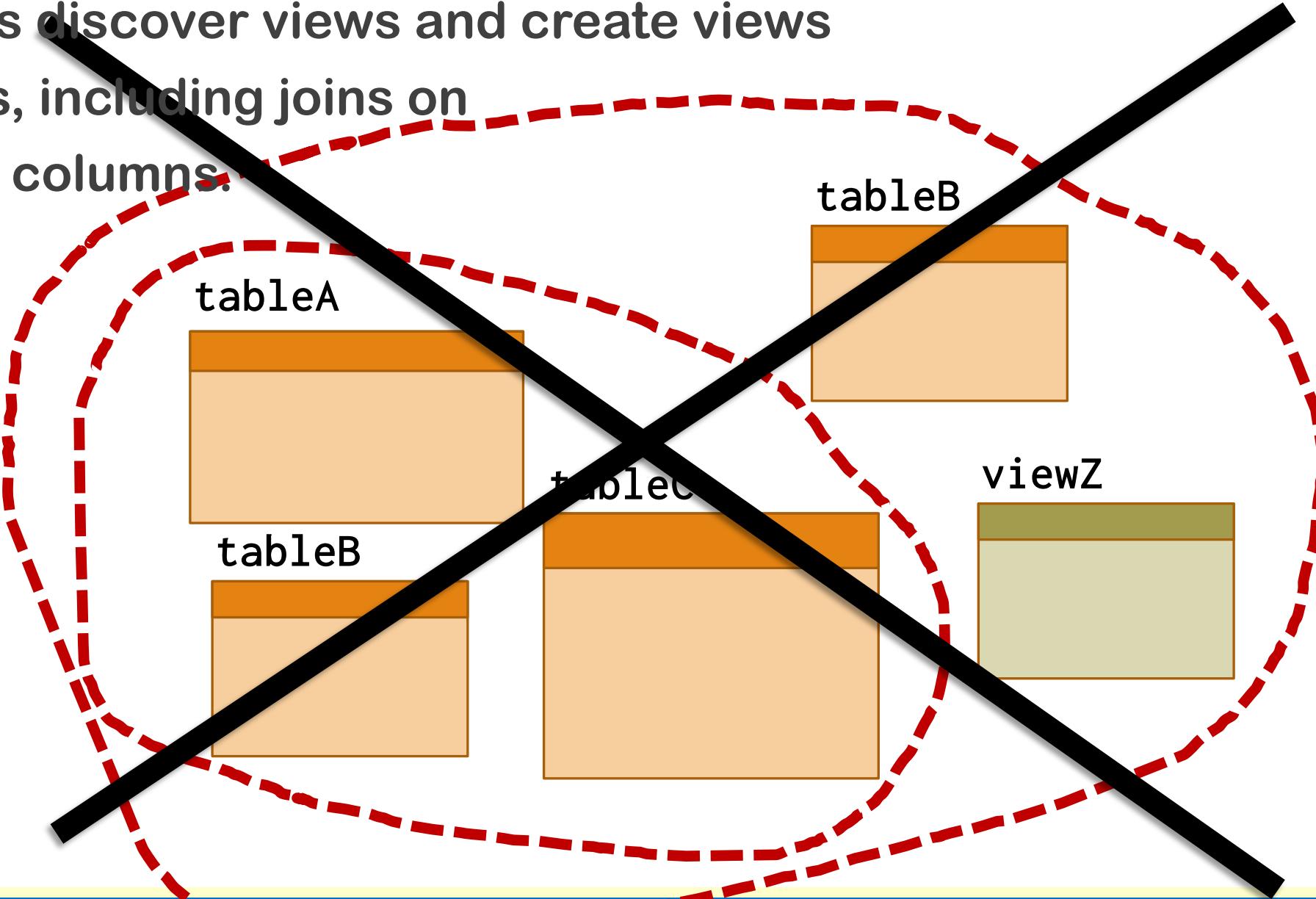


If querying the view takes 4 times as long as querying the table, you'll only be able to serve 25% of users..

Demand, though, isn't something that you control. So as many users will probably query the database at the same time whether you query views or tables. With complex views, more requests will be queued and will wait.



Performance issues are usually compounded when  
developers discover views and create views  
over views, including joins on  
computed columns.



**DON'T**  
create views on  
complex views

# 11.3 Security

---

Shiqi Yu 于仕琪

yusq@sustech.edu.cn

# Good for: Reports User Interface **SECURITY**

Nevertheless, there are three areas where views are very useful.  
the third area is security.

TOP SECRET

# How is security managed?

Before we see how views can help, we need to review how security is managed in a database.

To access a database, you must be authenticated, which often means entering a username and a password.



There are other means of authentication, and for some products database authentication is tied to operating system authentication, but in any case the database knows who you are.

# Database Account **RIGHTS**

So you end up being connected to a database account, and this account as a set of rights.

In fact those rights are called "privileges", so that you understand that getting them is a favor.



Privileges

---

A privilege is given to a user account using this command:

**grant** *<right>* **to** *<account>*

and can be taken back using this one:

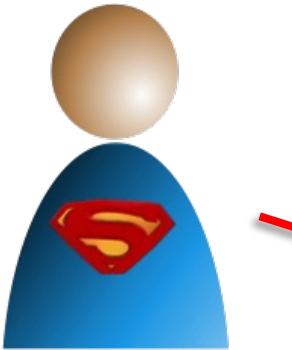
**revoke** *<right>* **to** *<account>*

**GRANT** and **REVOKE** are the two pillars of what is sometimes called **DCL**, Data Control Language.

# System rights

Rights

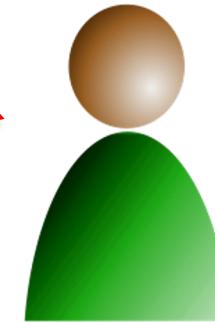
to create objects



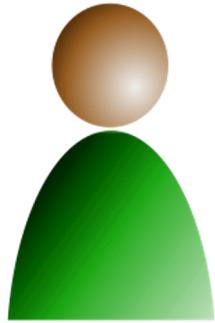
DBA

*grant*

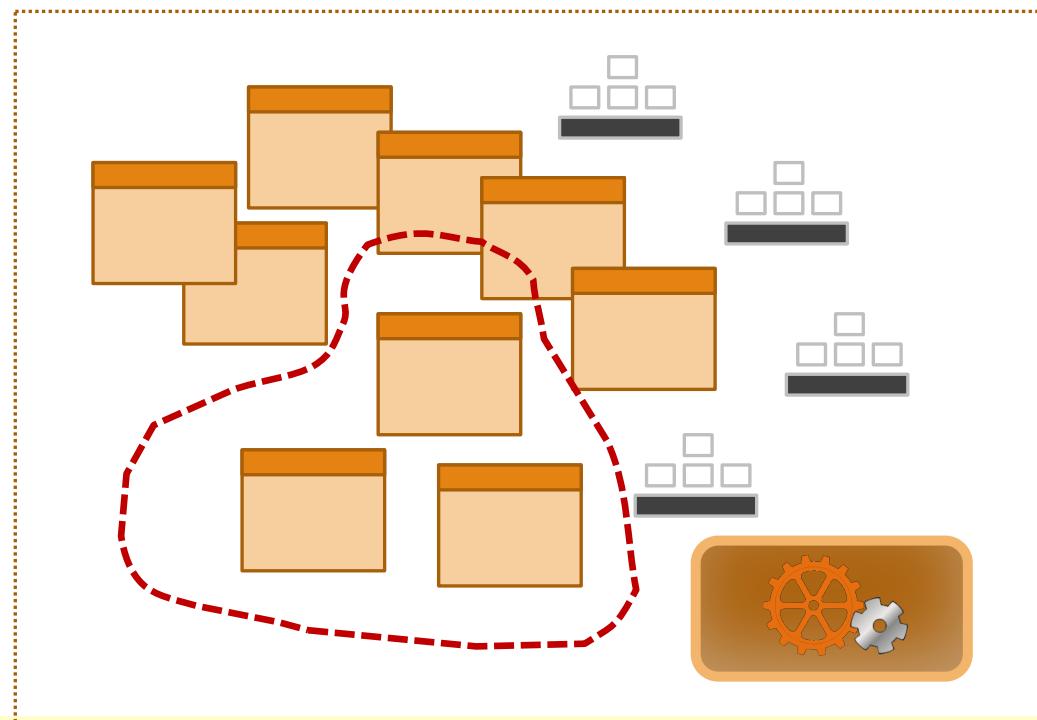
create table  
create procedure  
create view



Privileges fall into two categories: system rights give users the right to issue DDL commands and change the structure of the database. Not many people get them.



# Schema



A user who has these (or some of these) privileges will be able to create objects in a **SCHEMA**, which is usually a set of database objects required by one application.

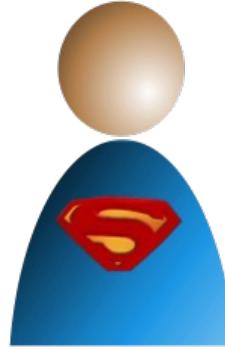
The other category of privileges is composed of privileges to access and change the data. Everybody who accesses the database must have some privileges of that category, otherwise there would be no point in accessing the database ...

## Table rights Rights to access the data

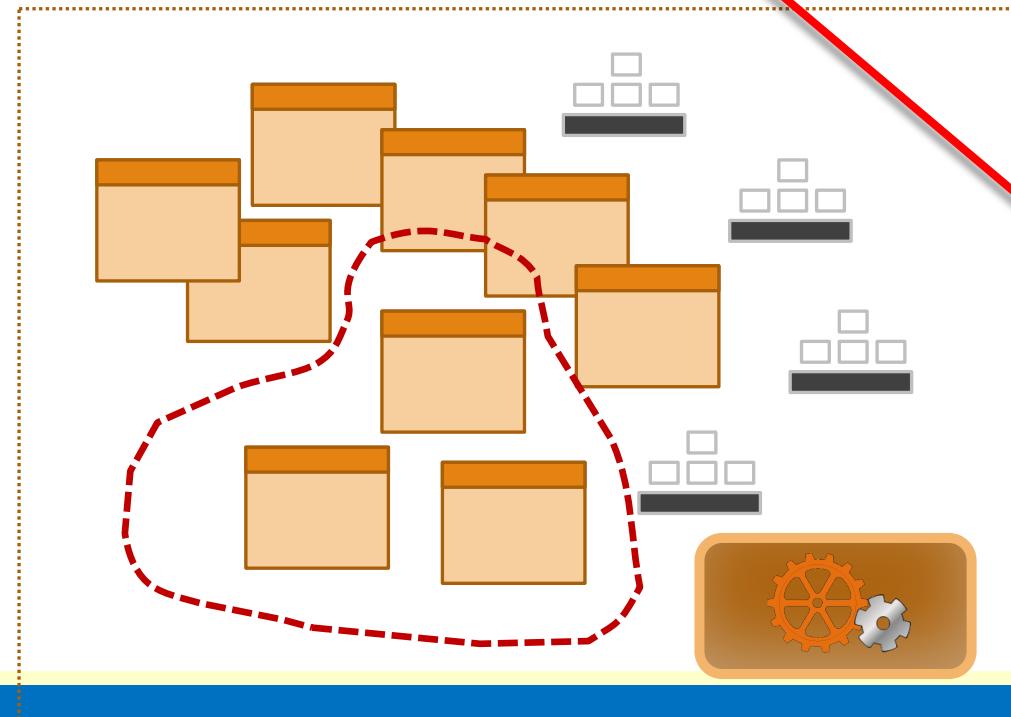
Some people can only access some of the data, some can modify "current" data but not reference tables, some data administrators may have the right to modify any table ... but not necessarily to create even a view!



When accounts have been created by a DBA, the schema owner can grant them specific privileges.

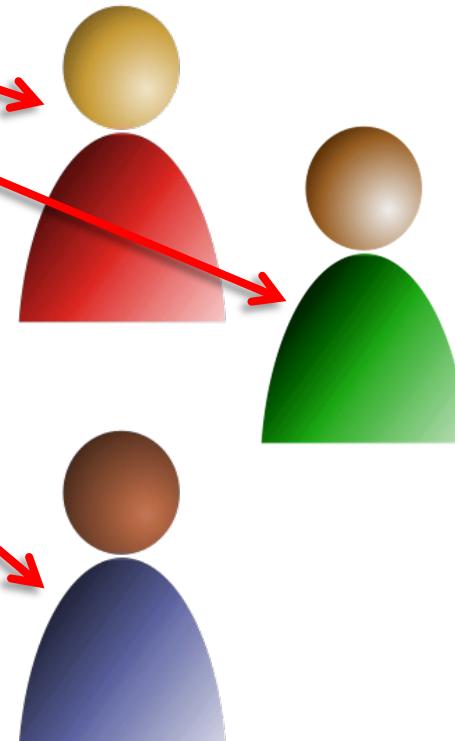


# Schema



*grant*

execute  
delete  
update  
insert  
select



DBA

---

```
grant select on tablename to accountname
```

```
grant insert on tablename to accountname
```

```
grant update on tablename to accountname
```

```
grant delete on tablename to accountname
```

```
grant select, insert on tablename to accountname
```

GRANT commands to give privileges on a table look like this. You can give one or several privileges at once. Sometimes you can give privileges over all the tables in a schema, existing tables and tables still to be created. The UPDATE privilege can also be restricted to some columns only. Some products may require special additional rights (with PostgreSQL "usage" on a schema)

For users who have been naughty:

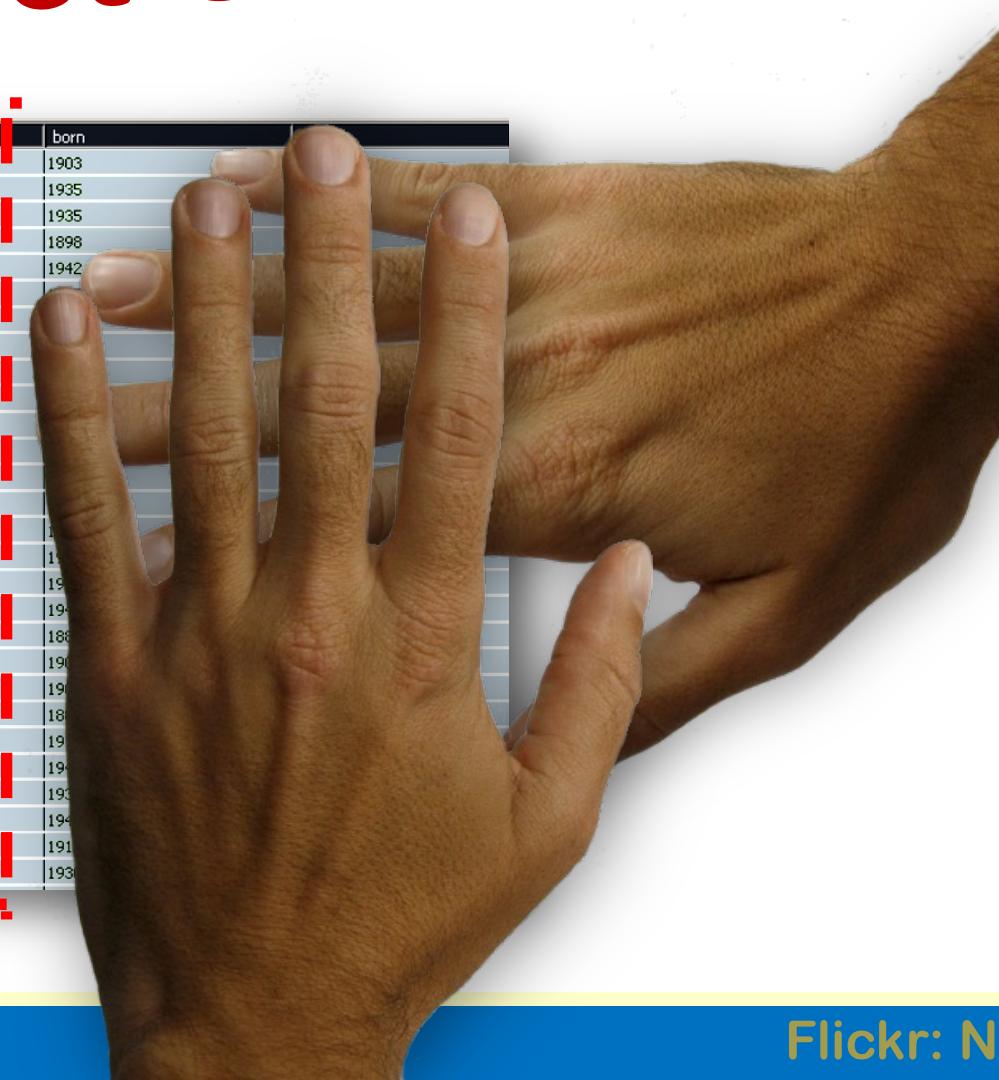
**revoke *privilege* on *tablename*  
from *accountname***

**How can views  
help?**

The trick is to use a view that only shows what people are supposed to see, and grant SELECT on the view and not on the table..

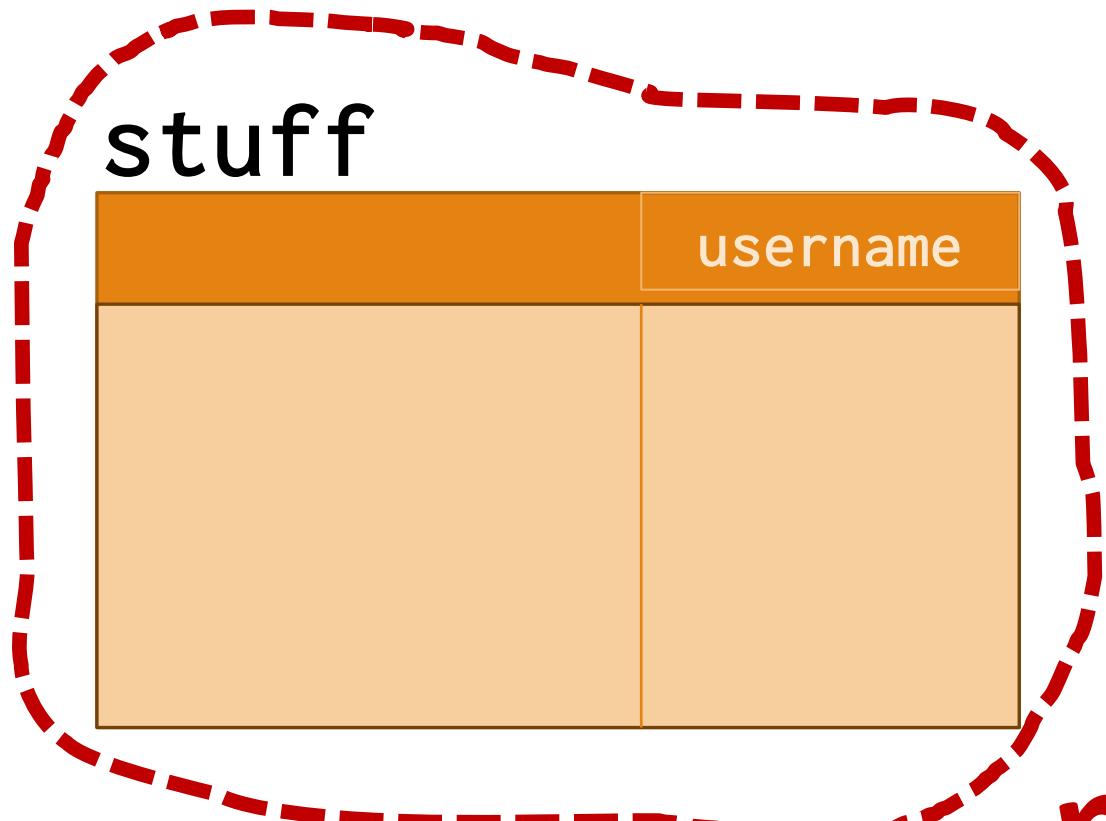
# grant select on view people

peopleid	first_name	surname	born
1	Grigori	Aleksandrov	1903
2	Woody	Allen	1935
3	Julie	Andrews	1935
4		Arletty	1898
5	Amitabh	Bachchan	1942
6	Kanu	Bannerjee	
7	Karuna	Bannerjee	
8	Ingmar	Bergman	
9	Ingrid	Bergman	
10	Luc	Besson	
11	Gunnar	Björnstrand	
12	Orlando	Bloom	
13	Humphrey	Bogart	
14	Marlon	Brando	
15	Marcel	Carné	
16	Leslie	Cheung	
17	Yun-Fat	Chow	
18	Julie	Christie	
19	Jean	Cocteau	
20	Joseph	Cotten	
21	Alain	Cuny	
22	Michael	Curtiz	
23	Josette	Day	
24	Robert	De Niro	
25		Dharmendra	
26	Richard	Dreyfuss	
27	Marie	Déa	1910
28	Clint	Eastwood	1930



You can hide sensitive columns

```
create view my_stuff  
as  
select * from stuff  
where username = user
```



my\_stuff

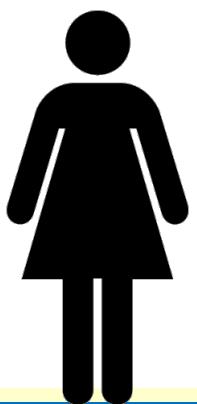
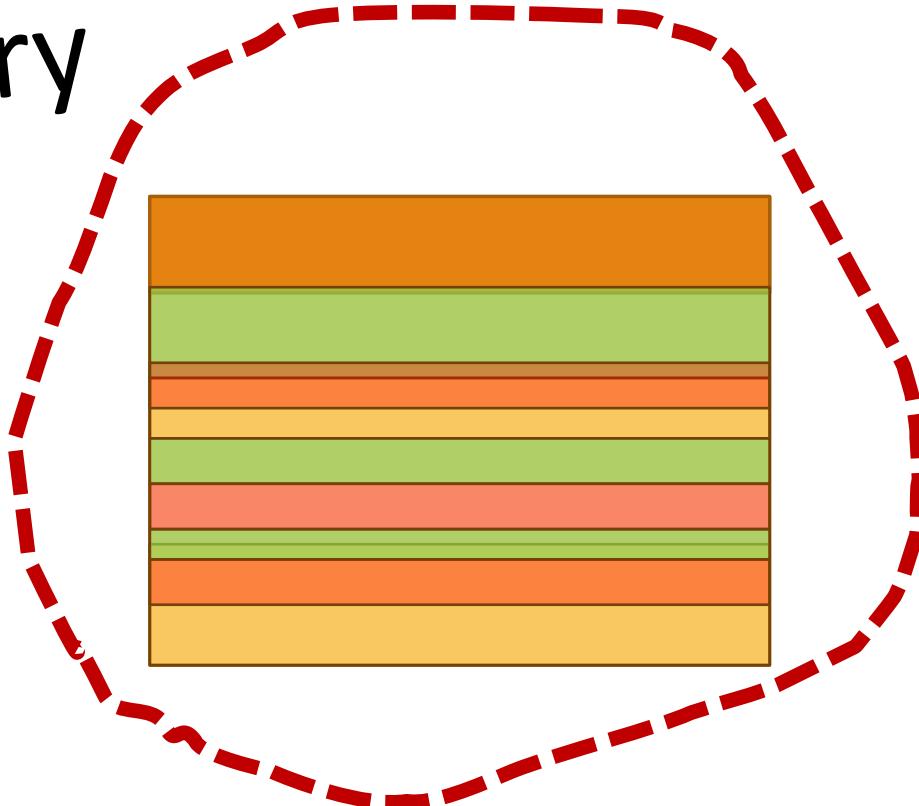
ORACLE®

PostgreSQL



You can even hide rows by only returning rows "owned" by the user currently connected.

Same query

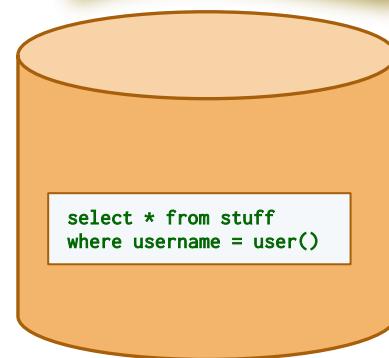


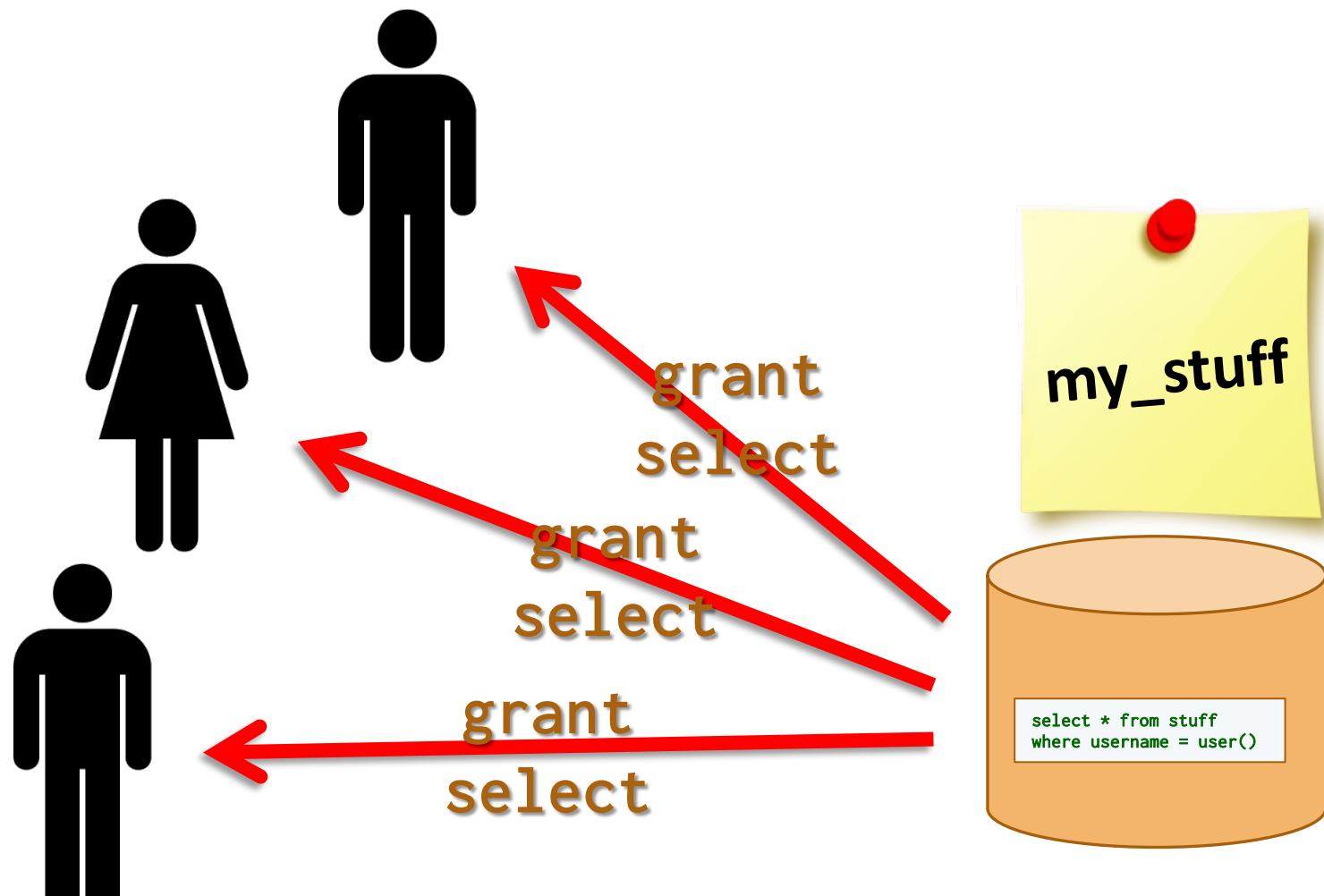
With such a view  
exactly the same query  
run by different users  
will return different  
rows.

# Beware when you **MODIFY** the definition of a view

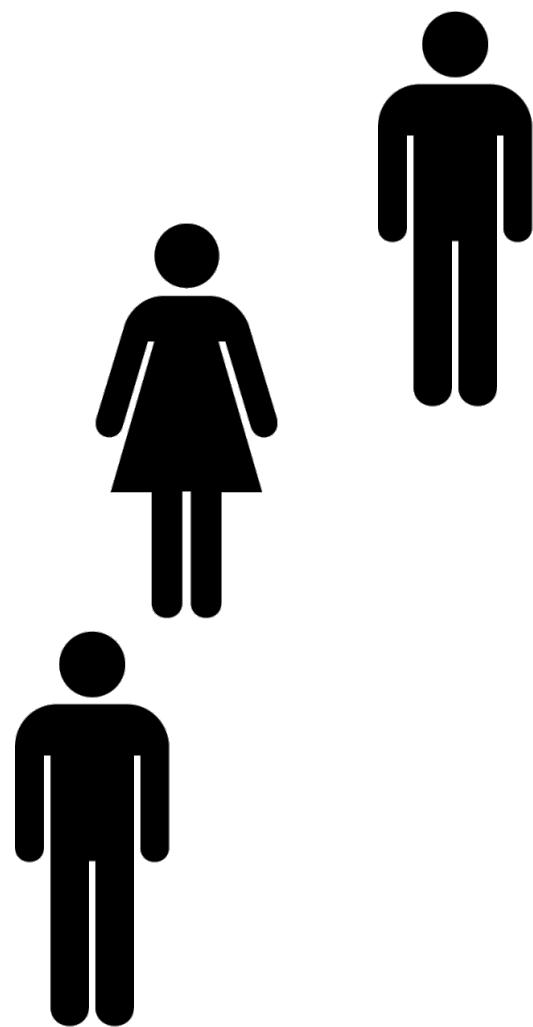
Beware when you modify the definition of a view:  
If you simply drop and recreate it, you lose the privileges.  
Use CREATE OR REPLACE

```
select * from stuff  
where username = user
```

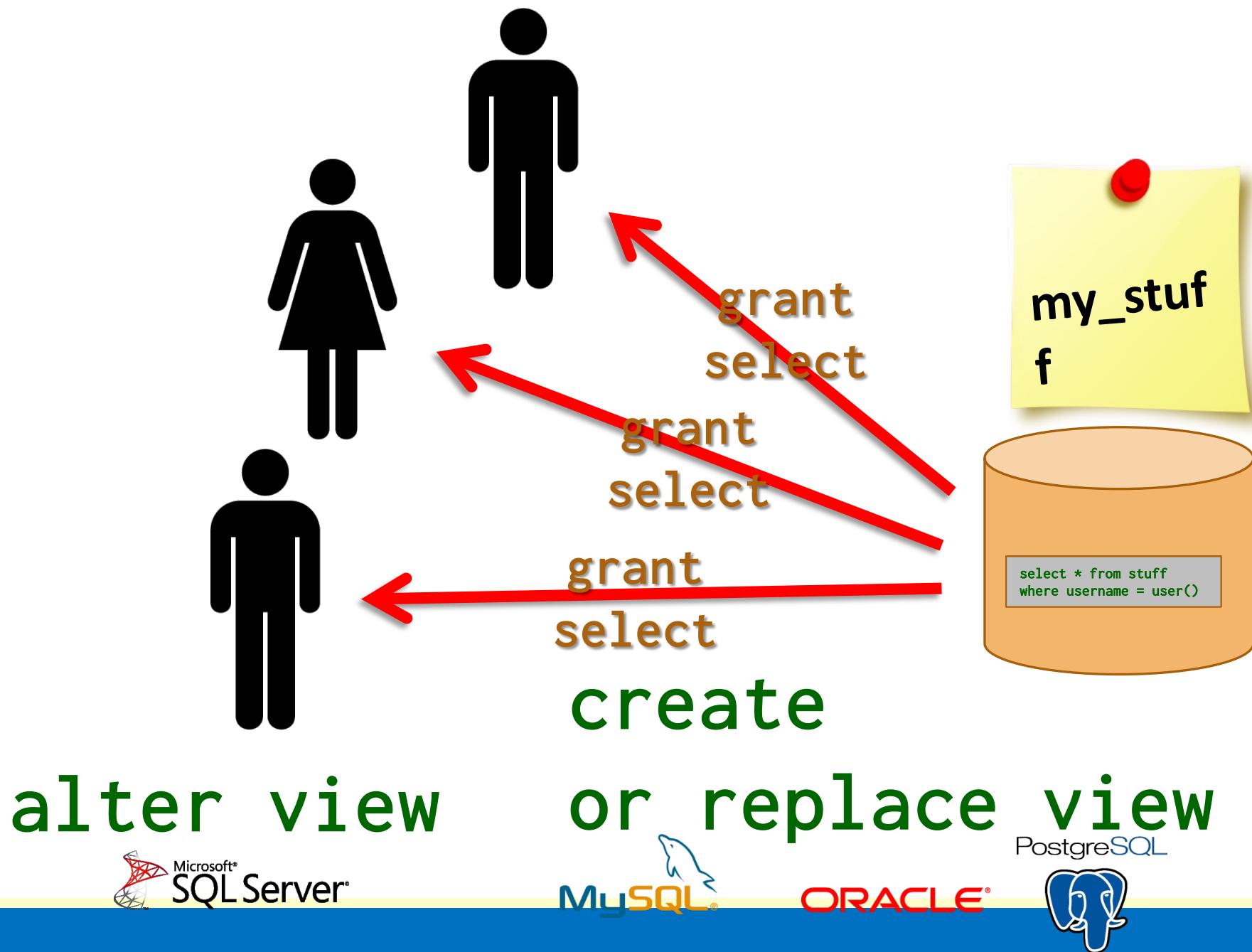


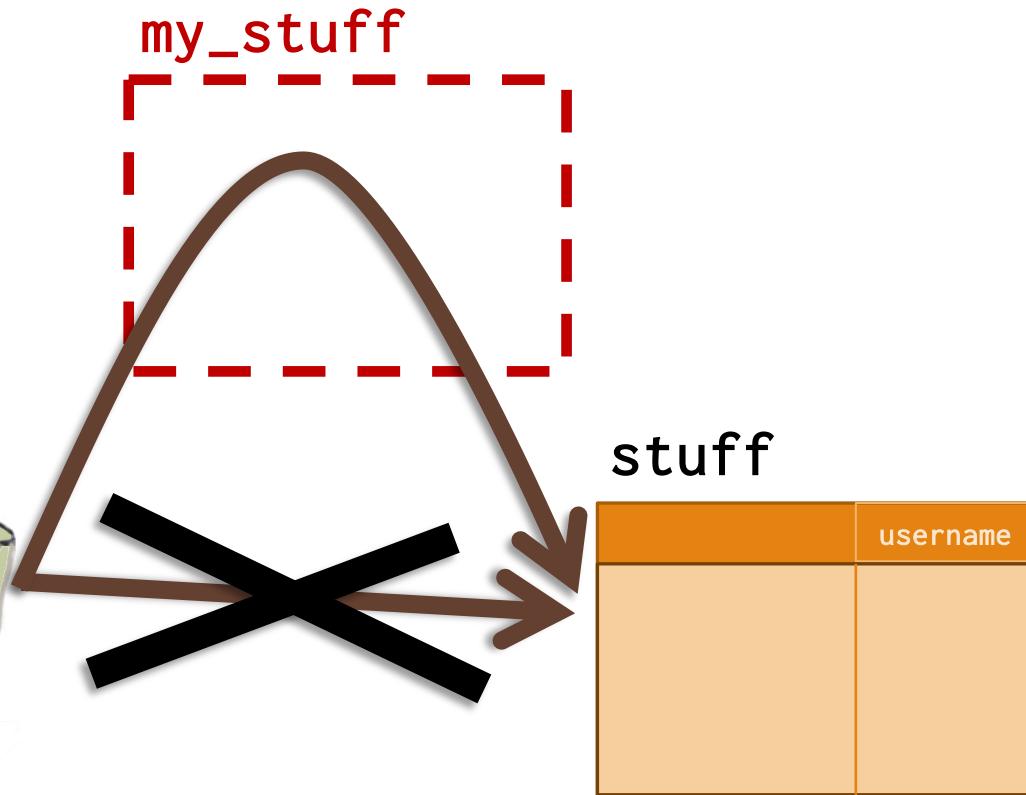
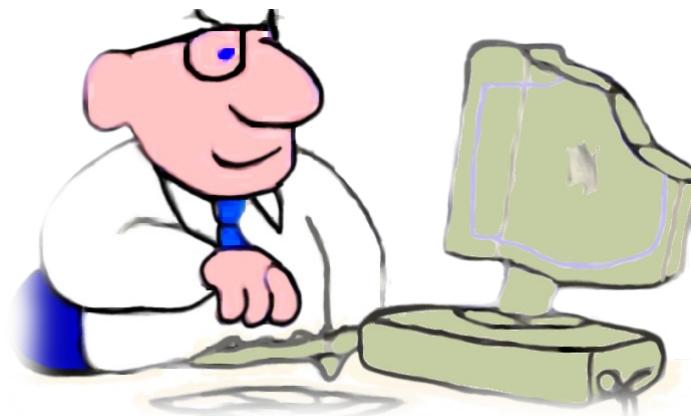
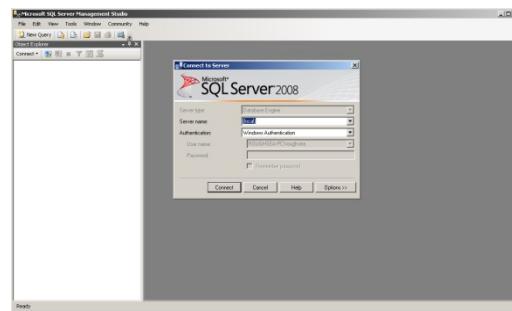


drop view



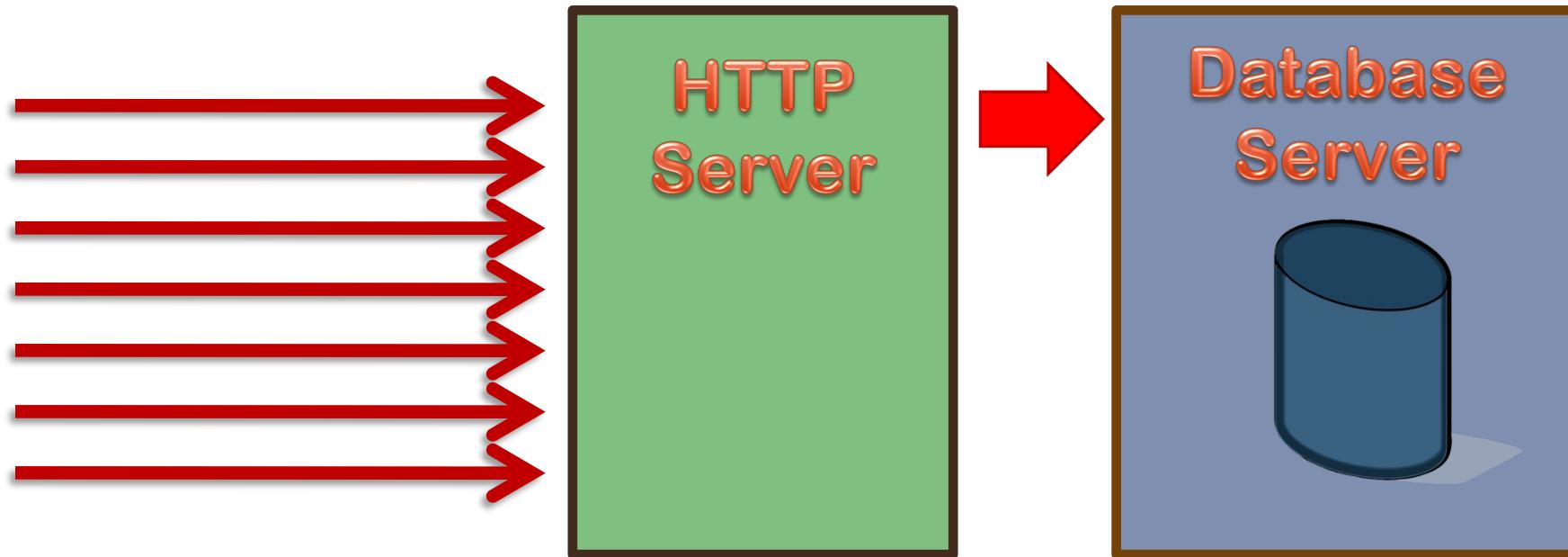
create view





Now, the problem is that for security though a view, users need to be personally authenticated.

**Requires  
PERSONAL  
account**



When accesses are run through a single connection as happens on a web server, it's not really interesting to use views for security. They can, however, be quite interesting for development, as we'll soon see.