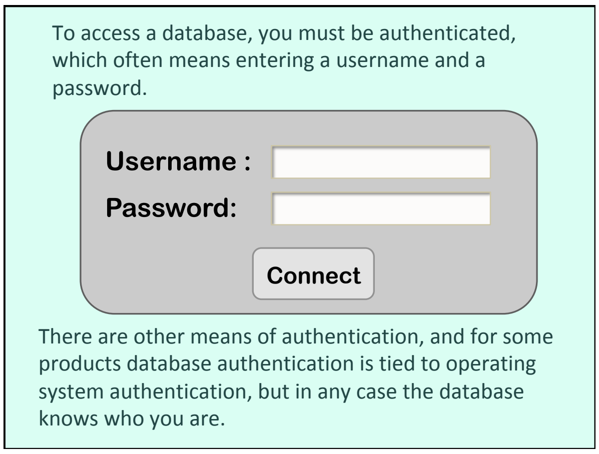
Privileges
and
View update

TOP SECRET

How is security managed?

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



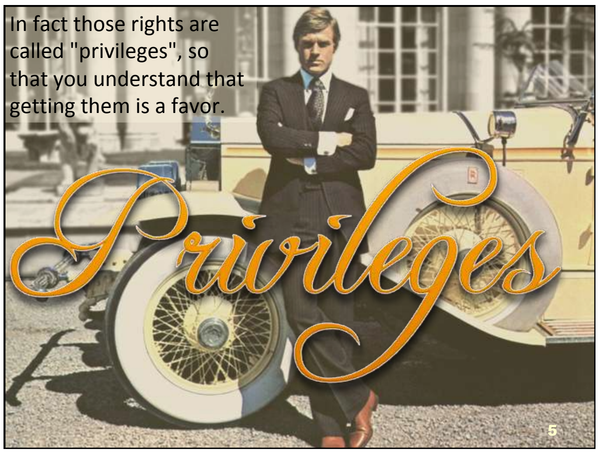
To access a database, you must be authenticated, which often means entering a username and a password.

Username :
Password:
Connect

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.

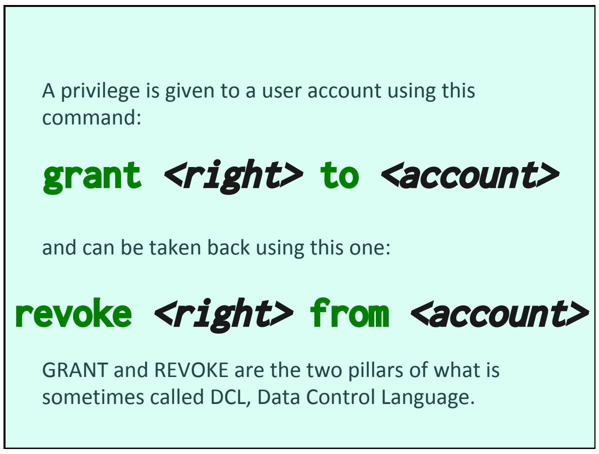
So you end up being connected to a database account, and this account as a set of rights.

Database Account RIGHTS


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

Privileges

5

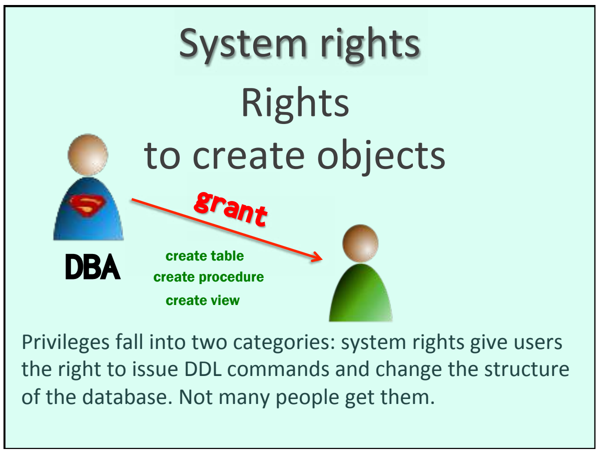

A privilege is given to a user account using this command:

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

and can be taken back using this one:

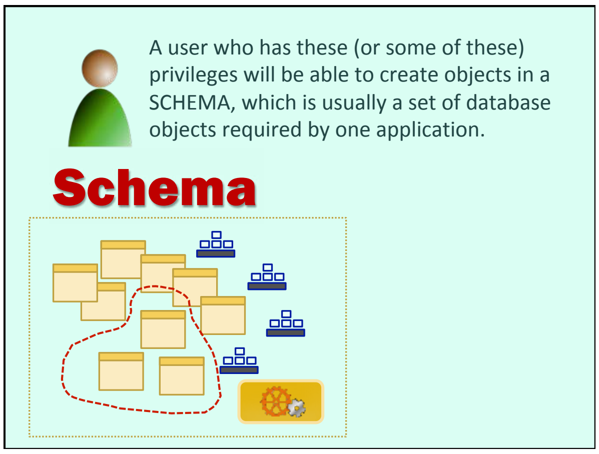
revoke ***<right>*** **from** ***<account>***

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


System rights

Rights to create objects

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.


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.

Schema

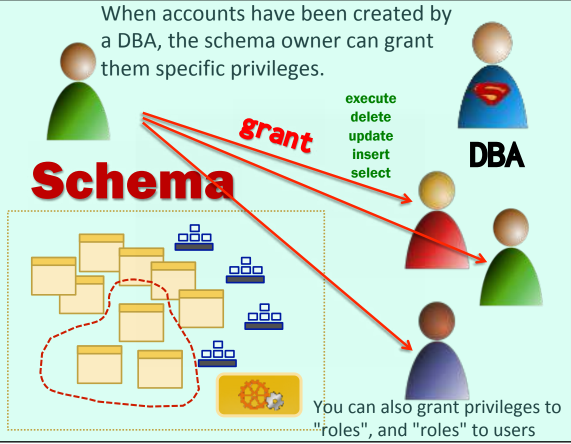
The other category of privileges is composed of privileges to access and change the data. Everybody who accesses the database must have some privileges of that category, otherwise there would be no point in accessing the database ...

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.



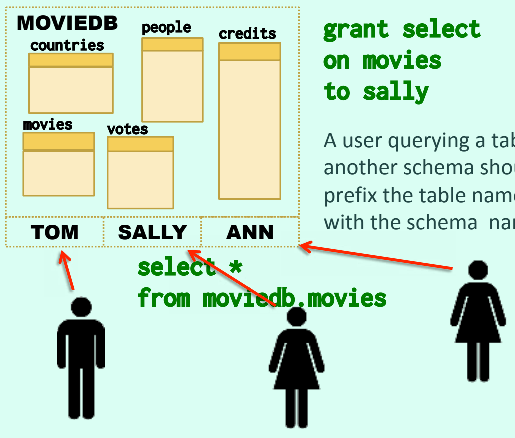
You can also grant privileges to "roles", and "roles" to users

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)

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

And for users who have been naughty:

```
revoke privilege on tablename
from accountname
```



```
select *
from moviedb.movies
```

```
grant select
on movies
to sally
```

A user querying a table in another schema should prefix the table name with the schema name.

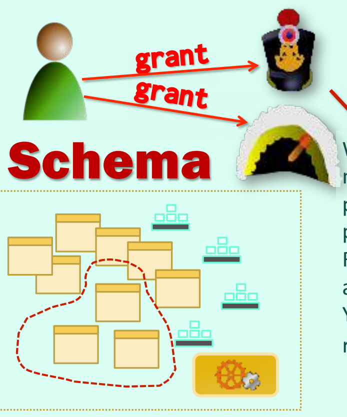
In practice, the full naming is rarely used.

```
select ...
from schema_name.table_name
where ...
```

Very often, in practice:
Set the default schema to what is required
Or define aliases

Not giving the schema name in programs allows to switch easily between schemas that contain the same objects (test/training/production) but not the same data.

ROLES



When there are many users and privileges, sets of privileges called ROLES are created and granted to users. You can even grant a role to a role.

Finally, all products have a way to grant a (low) privilege to everybody, existing as well as future users.

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

= ROLE

(GROUP)

"grant to public" is often used for that.

"public-the-role" shouldn't be confused with "public-the schema" in PostgreSQL.

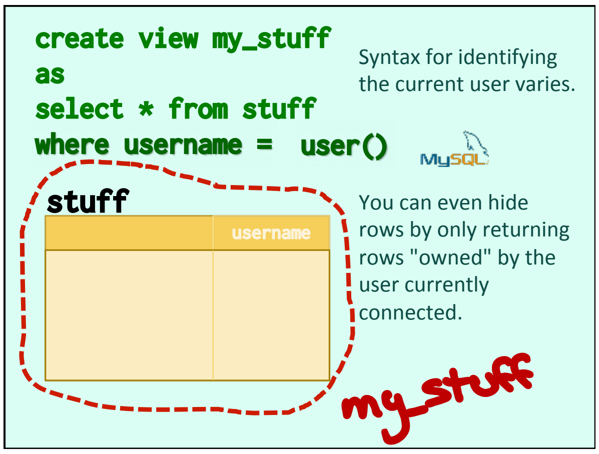
So ...

How can views help with security?

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..

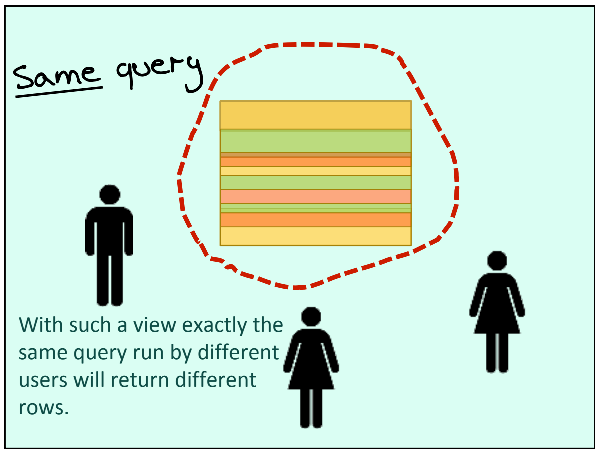
people
grant select on view
Flickr: Nate Steiner

You can hide sensitive columns

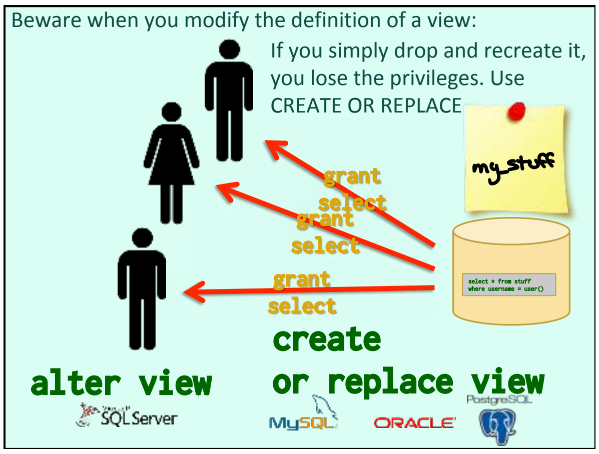

```
create view my_stuff
as
select * from stuff
where username =  user()
```

Syntax for identifying the current user varies.

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



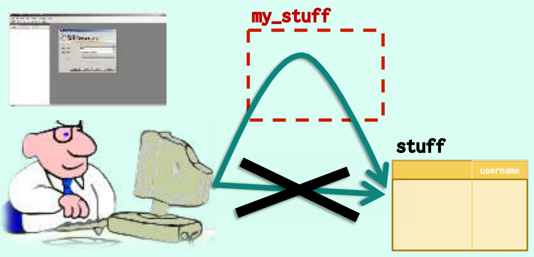
With such a view exactly the same query run by different users will return different rows.


Beware when you modify the definition of a view:

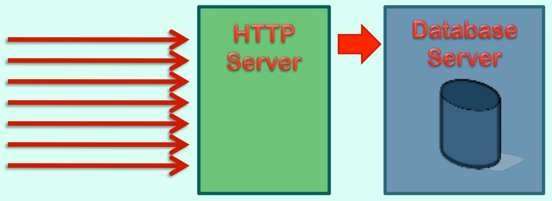
If you simply drop and recreate it, you lose the privileges. Use CREATE OR REPLACE

```
alter view
```

```
create
or replace view
```



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



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.

What about **CHANGING DATA** through views?

If views are in theory like tables, why not using them for controlling not only what you SEE, but what you CHANGE?

Lots of things can go wrong

It all depends on the view ... The problem is that most view are designed to provide a more user-friendly view of data: joins transforming codes into more legible values, functions making data prettier (date formatting, for instance). And by doing so you often lose information.

For instance if your view concatenates first_name and surname, splitting a single string in two parts is tough if you want to insert through the view.

```
case
  when p.first_name is null then p.surname
  else p.first_name || ' ' || p.surname
end name
```

Tommy Lee Jones

Benicio Del Toro

Everybody isn't called 'Gary Cooper'.

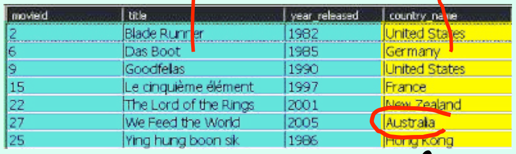
And for updates ... Let's have a view that displays the country name rather than code.

```
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
```

---

from movies

from countries



movieid	title	year_released	country_name
2	Blade Runner	1982	United States
6	Das Boot	1985	Germany
9	Goodfellas	1990	United States
15	Le cinquième élément	1997	France
22	The Lord of the Rings	2001	New Zealand
27	We Feed the World	2005	Australia
25	Ying hung boon sik	1986	Hong Kong

Never understood why people were confusing both.

Wrong! AUSTRIA, not Australia!

CORRECTION

SQL Server would let you update ... and try to change the name in table COUNTRIES.

```
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
```

NOT this

THIS

Most products will express concern and prevent you from doing it.

Abandon all hope, ye who enter here

In many cases, view update is simply impossible.

Most joins

Aggregates

Expressions

Omitted mandatory columns (insert)

Sometimes it works very well

In some cases, view update is quite possible.

This will work fine with Oracle, which would have complained with a join

One table

```
create or replace view vmy_movies
as select m.movieid,
          m.title,
          m.year_released,
          m.country
   from movies m
   where m.country in
     (select c.country_code
      from countries c
           inner join user_scope u
             on u.continent = c.continent
      where u.username = user)
```

Everything else in a subquery

USER_SCOPE

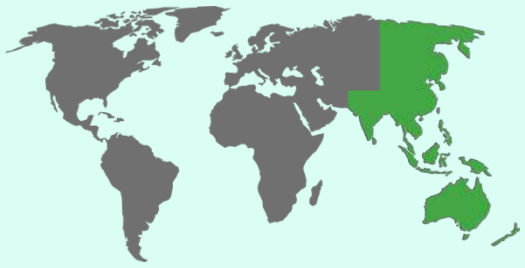
Username	Continent
HUIZHONG	ASIA
PAVEL	EUROPE
IBRAHIM	AFRICA
AMINATA	AFRICA
MICHAEL	EUROPE
JUAN_CARLOS	AMERICA
SANDEEP	ASIA
PATRICIA	AMERICA
PATRICIA	EUROPE

Which proves that in some cases join and subquery aren't exactly equivalent ...

There is no problem because the view update maps to a simple table update.

Plain insert/ update/delete of *movies*

Now, there may STILL be a problem.



Suppose that you are in charge of Asia/Pacific, and only see films from this region.

Consistency Issue

```
select * from vmy_movies;
```

```
movieid  title                           year_released  country
     19  Pather Panchali                          1955       in
     20  Shichinin no Samurai                     1954       jp
     21  Sholay                                   1975       in
     22  The Lord of the Rings                    2001       nz
     25  Ying hung boon sik                       1986       hk
     26  We Feed the World                        2005       au
```

Ooops

Only from Asia/Oceania

If you change the country from Australia to Austria (in Europe), poof! you no longer see it.

```
update vmy_movies
set country = 'at'
where movieid = 26
```

Nothing prevents from

```
insert into vmy_movies(title, year_released, country)
values ('Snow White and the Seven Dwarfs', 1937, 'us')
```

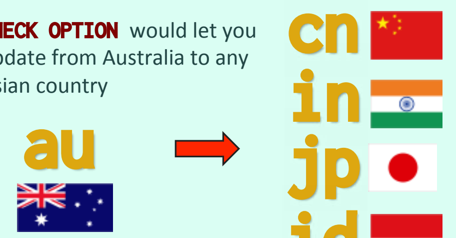
UNLESS

There is one special constraint, though, that exists for views: **WITH CHECK OPTION**.

```
create or replace view vmy_movies
as select m.movieid,
          m.title,
          m.year_released,
          m.country
   from movies m
   where m.country in
     (select c.country_code
      from countries c
           inner join user_scope u
              on u.continent = c.continent
      where u.username = user)
with check option
```

It prevents you from making a change that will make a row disappear from the view (other than a DELETE)

CHECK OPTION would let you update from Australia to any Asian country



But not to a country from another region

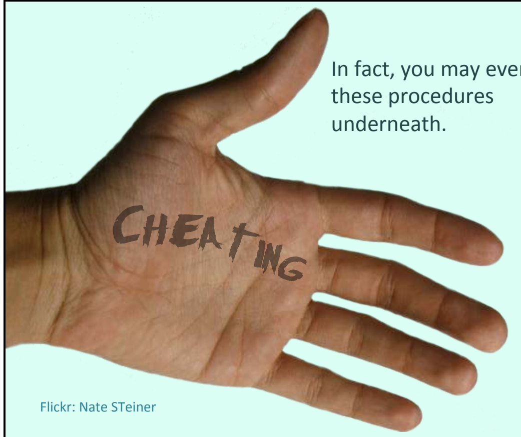
Solution in some cases:

insert procedure

update procedure

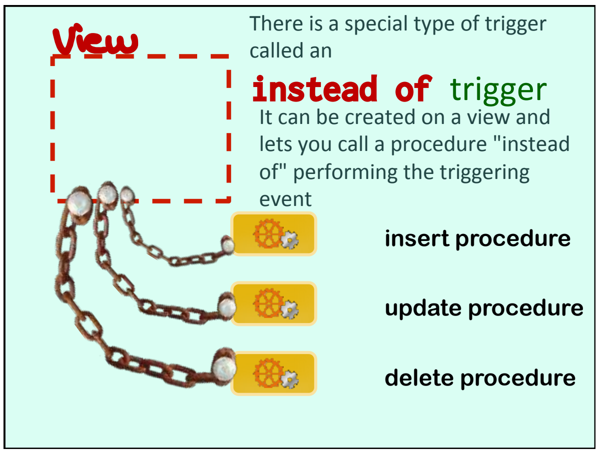
delete procedure

If updating the view directly is impossible, in many cases (remember when we were displaying the country name) what should be applied to base tables is fairly obvious and can be performed by dedicated stored procedures.



Flickr: Nate STeiner

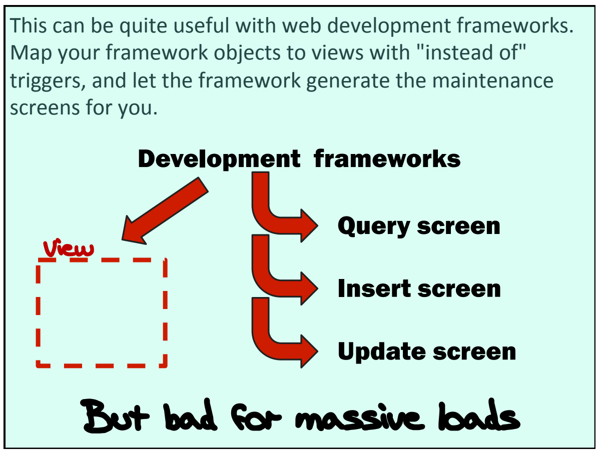
In fact, you may even call these procedures underneath.



There is a special type of trigger called an

instead of trigger

It can be created on a view and lets you call a procedure "instead of" performing the triggering event



This can be quite useful with web development frameworks. Map your framework objects to views with "instead of" triggers, and let the framework generate the maintenance screens for you.

But bad for massive loads