

# **CS307**

# **Database Principles**

## **Chapter 12**

---

Shiqi YU 于仕琪

yusq@sustech.edu.cn

Most contents are from Stéphane Faroult's slides

# 12.1 Change data through views

---

Shiqi Yu 于仕琪

yusq@sustech.edu.cn



## 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 views 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.

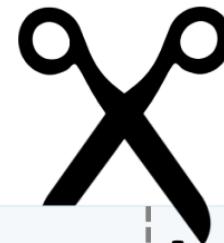
For instance if your view concatenates `first_name` and `surname`, splitting a single string in two parts is tough if you want to insert through the view.

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'.

```
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
```

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

from movies



from countries



movied	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

Wrong!

AUSTRIA, not Australia!

# CORRECTION

```
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
```

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

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

The screenshot shows a MySQL Workbench interface. The top bar includes standard icons for execution, refresh, and transaction control, along with a dropdown for 'Tx: Auto'. The title bar on the right displays the database name 'filmdb'. The main area is a query editor with the following SQL code:

```
1 ! update vmovies
2 set country_name='Australia'
3 where movieid=27;
```

The second line contains a syntax error, indicated by a red exclamation mark icon in the margin.

[55000] ERROR: cannot update view "vmovies"

详细: Views that do not select from a single table or view are not automatically updatable.

建议: To enable updating the view, provide an INSTEAD OF UPDATE trigger or an unconditional ON UPDATE DO INSTEAD rule.

# 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)

### USER\_SCOPE

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

## USER\_SCOPE

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

```
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

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

---

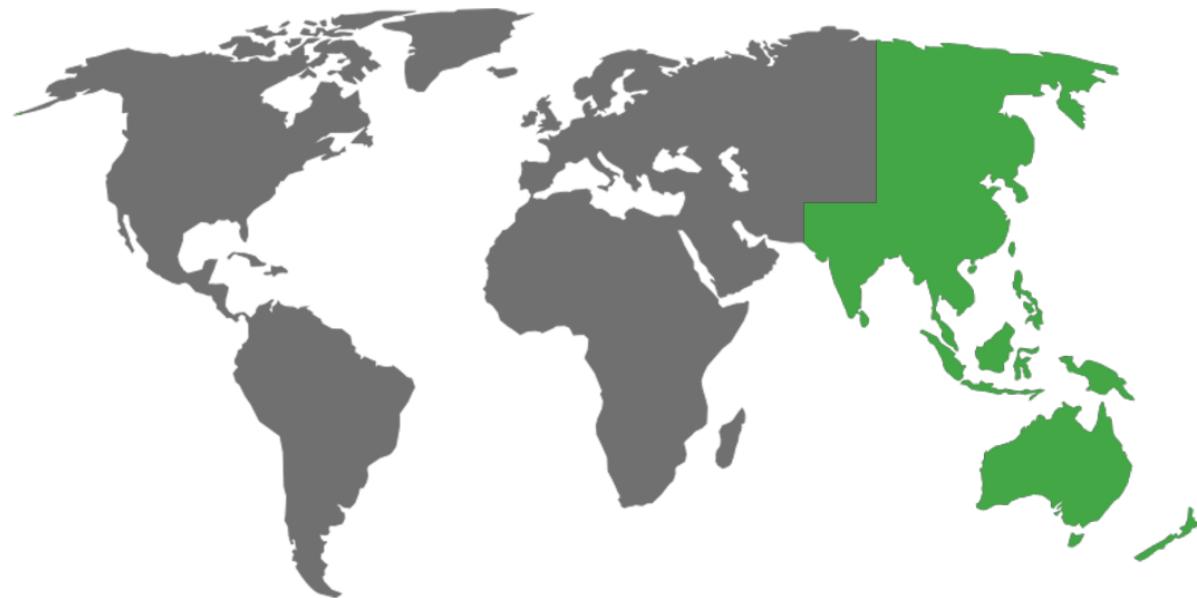
There is no problem because the view update maps to a simple table update.

# 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	Da Nao Tian Gong	1965	cn
26	We Feed the World	2005	au



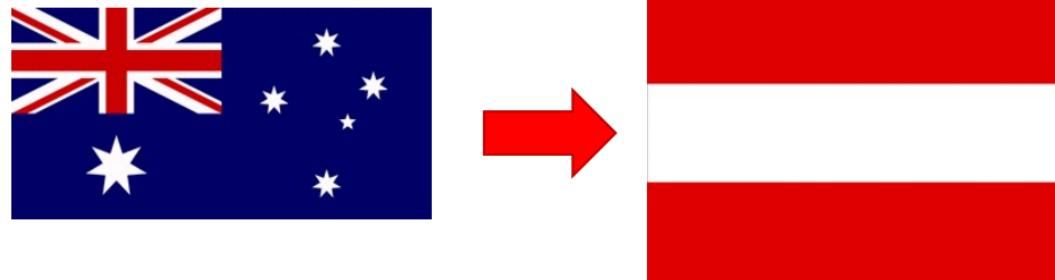
Ooops



Only from  
Asia/Oceania

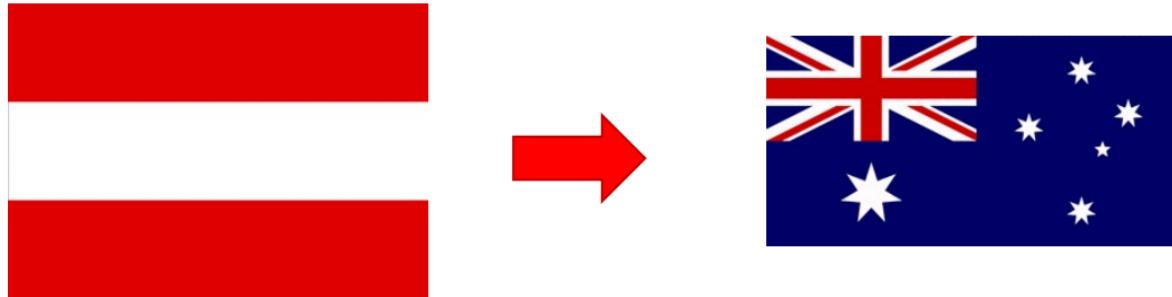
If you change the country from Australia to Austria (in Europe), poof! you no longer see it.

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



The following command cannot bring it back.

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



**Nothing prevents from**

```
insert into vmy_movies(movieid, title, year_released, country)  
values (9205, 'NEW MOVIE', 2020, '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
```

```
► | ⏱ | p | Tx: Auto ✓ ↺ | ■  
1 ! insert into vmy_movies(movieid,title, year_released, country)  
2   values (9205, 'NEW MOVIE', 2020, 'us');  
3
```

[44000] ERROR: new row violates check option for view "vmy\_movies"  
详细: Failing row contains (9205, NEW MOVIE, us, 2020, null).

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

But not to a country from another region

cn

in



au



id



jp



at



# Solution in some cases:



insert procedure

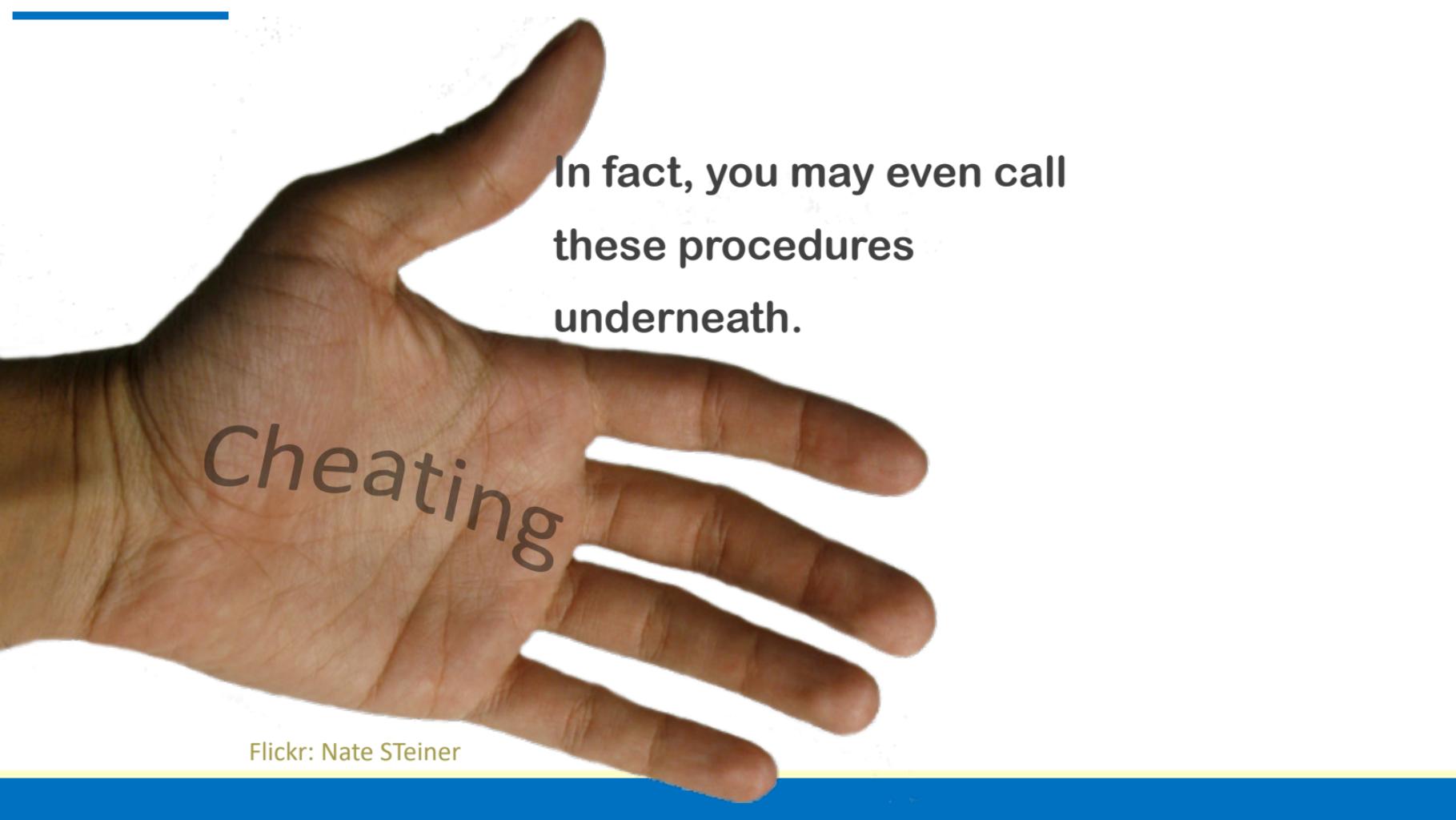


update procedure



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.

A photograph of a person's hand holding another person's hand. The hand being held has white horizontal lines drawn across its fingers, resembling a barcode or a stylized representation of a handprint. The background is plain white.

In fact, you may even call  
these procedures  
underneath.

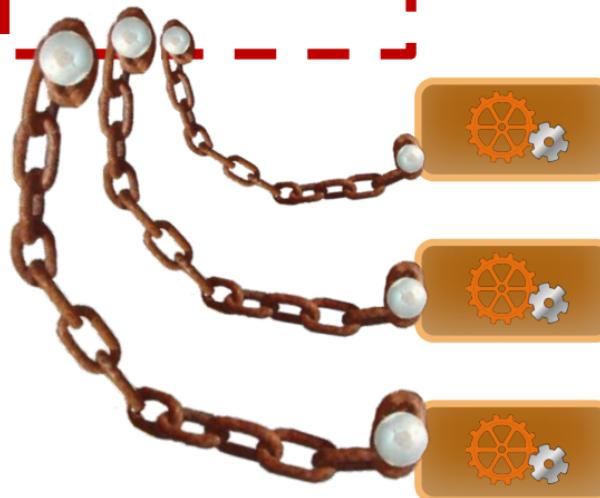
Cheating

# View

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



insert procedure

update procedure

delete procedure

CREATE TRIGGER -- define a new trigger

## Synopsis

---

```
CREATE [ CONSTRAINT ] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [ OR ... ] }
ON table_name
[ FROM referenced_table_name ]
[ NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY DEFERRED } ]
[ FOR [ EACH ] { ROW | STATEMENT } ]
[ WHEN ( condition ) ]
EXECUTE PROCEDURE function_name ( arguments )
```

where *event* can be one of:

INSERT

UPDATE [ OF *column\_name* [, ...] ]

DELETE

TRUNCATE

# 12.2 Data Dictionary

---

Shiqi Yu 于仕琪

yusq@sustech.edu.cn

One very good example of view application is the set of tables that contain information about the objects in the database, collectively known as the

# Data Dictionary

or sometime called the

# Catalog

They are using all the features we have seen (you only have privileges to read views and only see what is relevant to your account)



databases



schemas



tables



columns

# One catalog per database

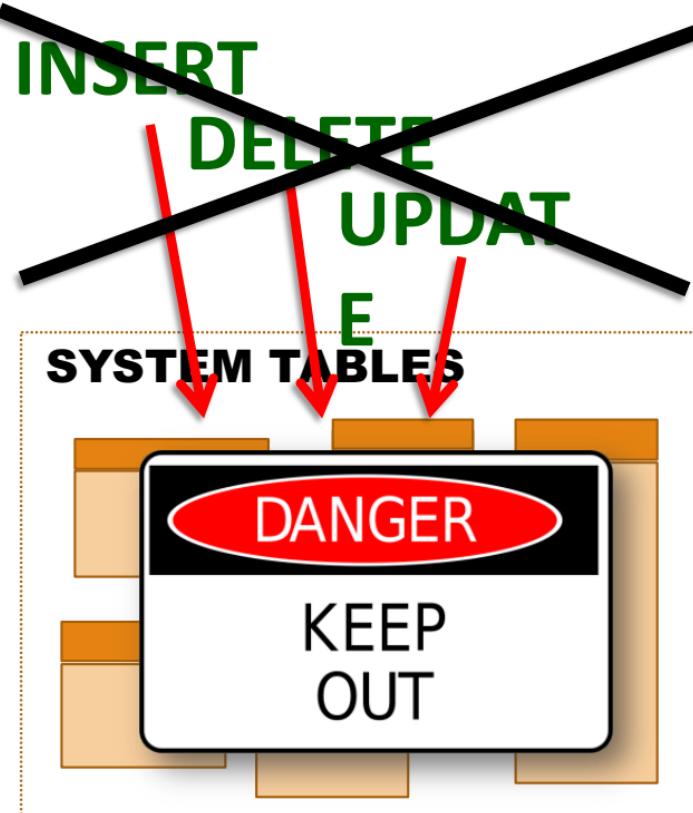
You always have ONE catalog per database. A database is an independent unit and you can have foreign keys only within (inside) one database; however, you can have several schemas in a database, and you can reference tables in another schema. There may also be metadata such as user accounts that is shared among databases. Most DBMS products can manage several databases at once; other than SQLite, the exception is MySQL that only has ONE catalog. What MySQL calls a *database* is actually a schema.

databases 1  
filmdb 3  
schemas 3  
information\_schema  
pg\_catalog  
public  
tables 11  
alt\_titles  
countries  
credits  
films\_francais  
forum\_members  
forum\_posts  
forum\_topics  
merge\_people  
movie title ft index?  
movies  
movieid integer  
title varchar(100)  
country char(2)  
year\_released integer  
runtime integer  
movies\_pkey (movieid)  
movies\_title\_country\_year\_released\_key (title, count)  
movies\_country\_fkey (country) → countries (country\_...  
movies\_nkey (movieid) UNIQUE  
movies\_title\_country\_year\_released\_lkey (title, count)  
country\_length (length(country) < ...)  
runtime\_numerical (runtime + 0) = ...e  
title\_length (length((title)::text))  
year\_released\_numerical (year\_released + 0...d  
people  
views 2

Any database stores "metadata" that describes the tables in your database (and not only them)

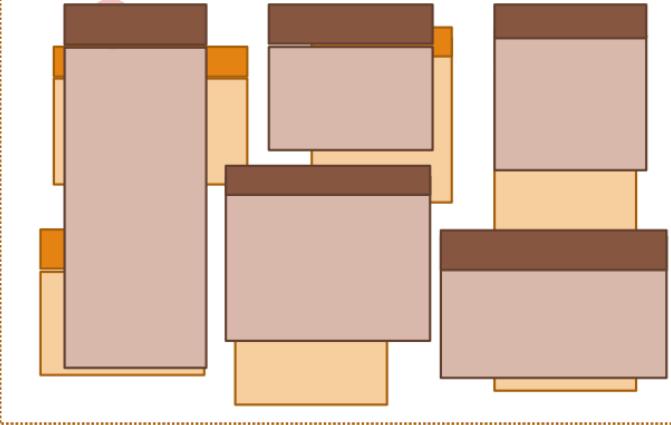
All client tools use this information to let you browse the structure of your tables (here it's SQL Server, Visual Studio)

**CREATE**  
**DROP**  
**ALTER**  
**GRANT**  
**REVOKE**



Whenever you are issuing DDL commands, you are actually modifying system tables. They must NEVER be directly changed.

## SYSTEM VIEWS



SQLite

sqlite\_master

Read access to these tables is provided through system views.

more user

friendly



Microsoft  
SQL Server



MySQL

PostgreSQL



Microsoft  
SQL Server



DB2.



PostgreSQL

+ views in schema **sys**

+ views in schema **syscat**

+ pg\_... views

# INFORMATION\_SCHEMA.TABLES



In these views you only see what YOU are allowed to see. Only administrators see everything.



## SYSCAT.TABLES



## USER\_TABLES / ALL\_TABLES



PostgreSQL - filmdb@localhost 1 of 2

- databases 1
  - filmdb 3
    - schemas 3
      - information\_schema
        - tables 7
        - views 60
  - views 60
    - \_pg\_foreign\_data\_wrappers
    - \_pg\_foreign\_servers
    - \_pg\_foreign\_table\_columns
    - \_pg\_foreign\_tables
    - \_pg\_user\_mappings
    - administrable\_role\_authorizations
    - applicable\_roles
    - attributes
    - character\_sets
    - check\_constraint\_routine\_usage
    - check\_constraints
    - collation\_character\_set\_applicability
    - collations
    - column\_domain\_usage
    - column\_options
    - column\_privileges
    - column\_udt\_usage
    - columns
    - constraint\_column\_usage

```
SELECT table_catalog, table_schema, table_name, table_type  
FROM information schema.tables  
where table_schema='public'
```

	table_catalog	table_schema	table_name	table_type
1	filmdb	public	merge_people	BASE TABLE
2	filmdb	public	countries	BASE TABLE
3	filmdb	public	people	BASE TABLE
4	filmdb	public	credits	BASE TABLE
5	filmdb	public	forum_members	BASE TABLE
6	filmdb	public	forum_topics	BASE TABLE
7	filmdb	public	forum_posts	BASE TABLE
8	filmdb	public	films_francais	BASE TABLE
9	filmdb	public	movies	BASE TABLE
10	filmdb	public	alt_titles	BASE TABLE
11	filmdb	public	movie_title_ft_index2	BASE TABLE
12	filmdb	public	vmovies	VIEW
13	filmdb	public	vmy_movies	VIEW

There are usually simpler commands to display the structure of a table, but these commands execute nothing more than this type of query. Everything is pulled out of the data dictionary.

```
1 ✓  SELECT table_name, column_name, ordinal_position, data_type  
2   FROM information_schema.columns  
3  where table_name='movies'  
4
```

是第几列

	table_name	column_name	ordinal position	data_type
1	movies	movieid	1	integer
2	movies	title	2	character varying
3	movies	country	3	character
4	movies	year_released	4	integer
5	movies	runtime	5	integer

# INFORMATION

As a developer, you can get from the data dictionary some information that is hard to get elsewhere (constraints, for instance). Database administrators use them a lot for scripting, because the data dictionary always reflects the current state of a database.

# SCRIPTING (DBA )



```
select 'drop table ' ||  
       table_name || ';'  
from information_schema.tables  
where table_name like 'TMP%'  
and ...
```



```
drop table TMP_People;  
drop table TMP_Posts;  
drop table TMP_XYZ;  
drop table TMP_dump;
```

DBAs often use queries on the catalog to generate other SQL queries; it can be done in a script, or in a procedure with a cursor (a case when cursors are mandatory). They sometimes generate other commands, such as shell script, for instance for backing up database files (the name of which can be found in some remote corners of the catalog).