# CS213
# Principles of Database Systems(H)

# Chapter 9

Shiqi YU 于仕琪

yusq@sustech.edu.cn

Most contents are from Stéphane Faroult's slides

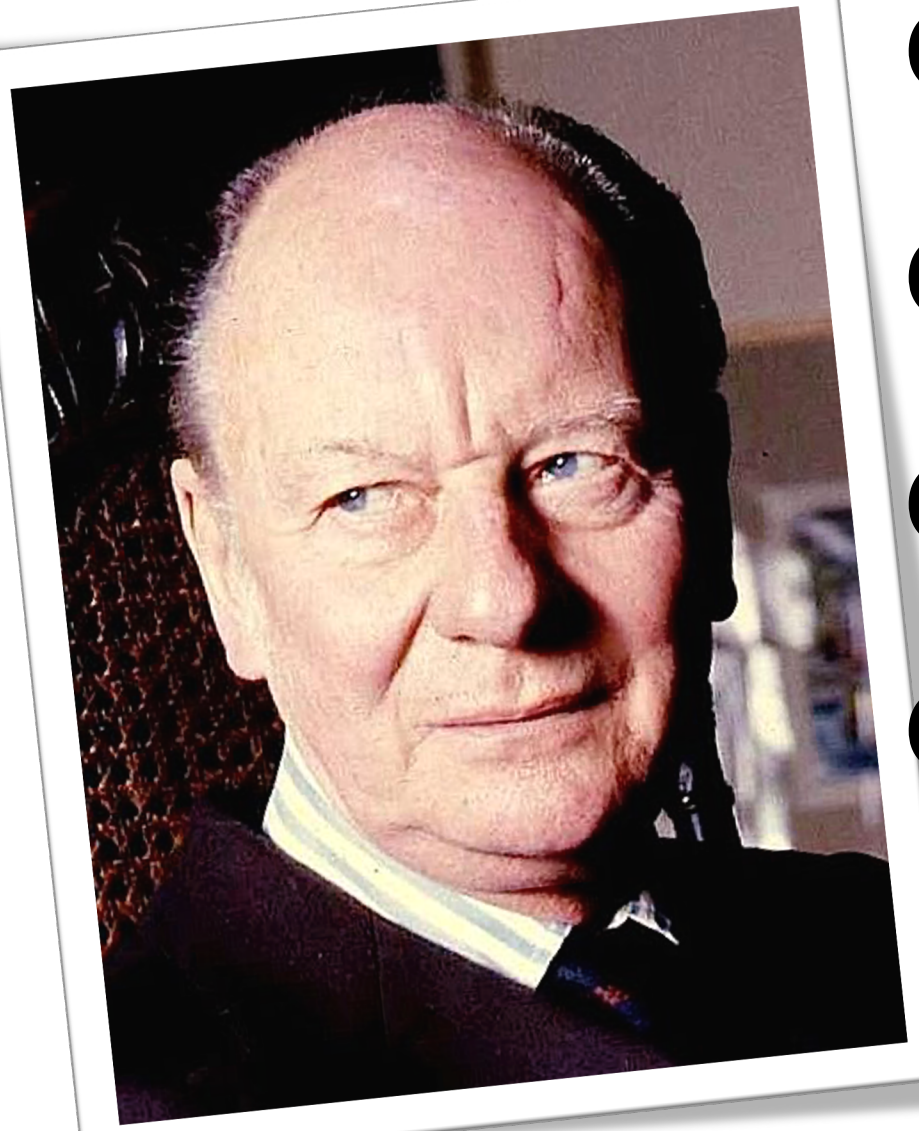
# 9.1 Fuzzy Search

Shiqi Yu 于仕琪

yusq@sustech.edu.cn

Most contents are from Stéphane Faroult's slides


**Life would be easy with databases if we were looking all the time for strict equality of data. Unfortunately, this is rarely the case, especially with text data. Even if we try to normalize and standardize data as much as we can, constraints will do little against typos and some misspellings (surnames are a nightmare for emergency services at hospitals). We'll explore a few problems and possible solutions.**



Picture by Allan Warren

Guilgood

Gillgood

Gielgud

Gilgud

**This famous British Shakespearean actor probably had one of the most misspelt names in his country (Gielgud is correct). His family was of German origin and they never felt a need to make their name look more English (contrary to the von Battenbergs who became Mountbattens, or Sachs-Coburg Gothas who became Windsors during WWI, much to the amusement of Wilhelm II of Prussia, himself a grandson of Queen Victoria)**

Zhou

Chiew

Chiau

Chou

Chu

Jhou

Spelling issues are even worse with name originally written in another script than the Latin alphabet, and for which transcription is merely based on sound (and pronunciation may vary). This common Chinese surname may be turned into many variants.

```
select ...
from ...
where name='Jack Chan'
```

```
select ...
from ...
where name='成龙'
```

```
select ...
from ...
where name='成龍'
```

```
select ...
from ...
where name='Jackie Chan'
```

Title to search: 2001, a space odyssey

# 2001: A Space Odyssey

It's not so easy in comparison of searching by title. The longer the title, the more opportunities for having something wrong, and sometimes it may be as little as punctuation.

What people use is known as "full-text search" and here is a simplified version of how it works. You split a text in (pure) words, eliminate words that are too common, then associate each word with the film identifier.

# FULL-TEXT SEARCH

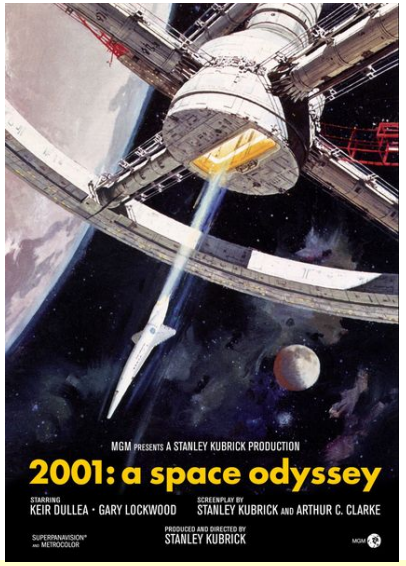
## 2001: a space odyssey

**2001**

~~**a**~~

**SPACE**

**ODYSSEY**



```
create table movie_title_ft_index
       (title_word    varchar(30) not null,
        movieid       int not null,
         primary key(title_word, movieid),
         foreign key (movieid)
             references movies(movieid)
)
```

These words are stored in a special table. Note that a true full-text search engine "stems" words - it will recognize singular and plural as well as infinitive and preterit or past participle as the same word.

Title to search: 2001, a space odyssey

2001

SPACE

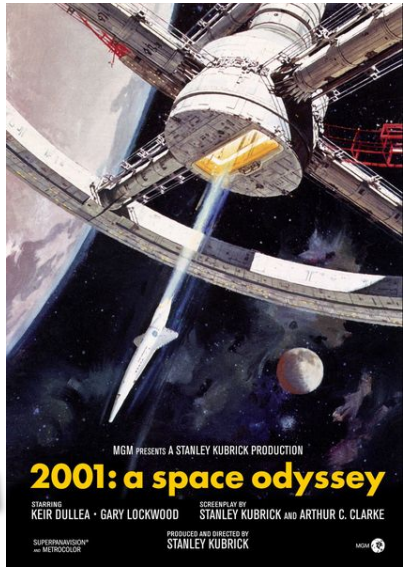
ODYSSEY

How are we going to use this? When people enter a title, we perform the same operation as before of isolating and standardizing words that look important. Then we are going to look for the MOVIEID values associated to these words in the previous table.

2001

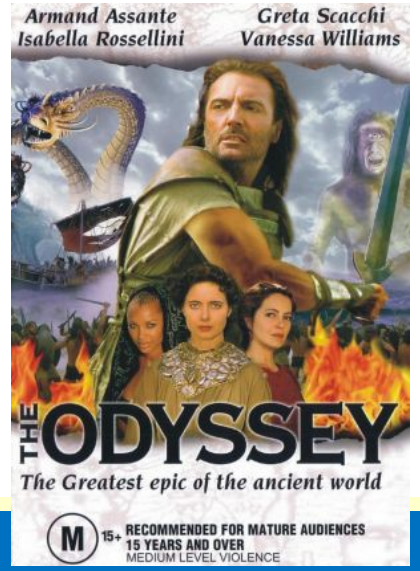
SPACE

ODYSSEY










Trouble is, the same word will probably occur in several titles.

We need a way to qualify matches.

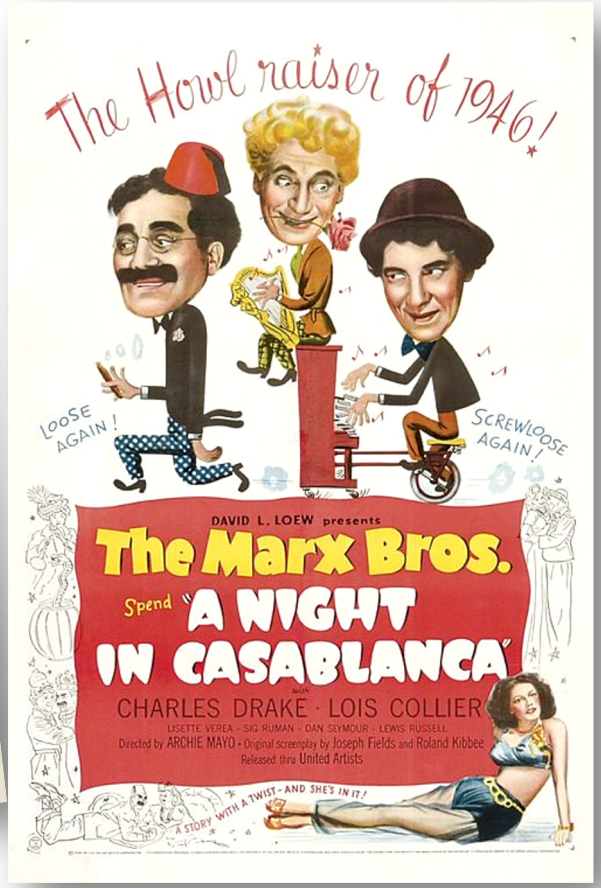
```
select movieid
from (select movieid,
             rank()
             over (order by hits desc) as rnk
      from (select movieid,
                   count(*) as hits
            from movie_title_ft_index
            where title_word in
               ('SPACE', 'ODYSSEY', '2001')
            group by movieid
      ) q1 ) q2
where rnk = 1
```

Such a query might serve the purpose. We count how many of the words we find by film for which at least one is found.

Then we rank. Notice that here we are interested by ties.

# Ties ?

**Title to search:** **Casablanca**



The first one is probably the one that was meant.

# Ties ?

**Title to search:** Casablanca

Different strategies may be adopted. And there is always the option of displaying multiple films in case of doubt (remakes, for instance).

Remove input words from title found

See what remains!

Compare lengths

# Typos ?

**Title to search:** Casabalnca

We may even extend our "index" with common misspellings and typos (what do they do in search engines?)

**CAZABLANCA**

**CASBLANCA**

**CASABALNCA**

# Question

- The Matrix Reloaded
- 黑客帝国2 重装上阵
- 駭客任務：重裝上陣
- Матрица: Перезагрузка
- La matrice recharge
- …


Users want to find it by:

- Matrix Reloaded
- Matrix 2
- 黑客2
- …

# 9.2 Transaction

Shiqi Yu 于仕琪

yusq@sustech.edu.cn

Most contents are from Stéphane Faroult's slides


Flickr:BracketingLife (Clarence)

In real life, a transaction is usually an exchange of goods for money. What is important is that in a transaction, there are several steps, and that is all or nothing.

Flickr: Randen Pederson

Considering the sceneries on drug trade in movies…

Something may happy during the trade …

The classical database example is transfering money from your current account to your saving account. You need to change the balance in two rows, what happens if the system crashes in the middle?

| Account type | Account number | Balance |
|---|---|---|
| CURRENT ACNT | 1234567 | 300.00 |
| SAVINGS ACNT | 8765432 | 1600.00 |

-100

+100

# Single Unit

At worst, balances should remain what they were before you initiated the transfer.

This idea that one business operation may translate into several database operations that must all succeed or fail is of prime importance to a DBMS. Some products require a special command to start a transaction (BEGIN is sometimes START)

```
begin transaction

    insert
      update
        delete
```

Other products such as Oracle or DB2 automatically start a transaction if you aren't already in one when you start modifying data.

```
runoobdb=# BEGIN;
DELETE FROM MOVIES WHERE MOVIEID = 25;
ROLLBACK;
```

```
runoobdb=# BEGIN;
DELETE FROM MOVIES WHERE MOVIEID = 25;
COMMIT;
```

A transaction ends when you issue either COMMIT (which is like an OK button) or ROLLBACK, which cancels everything you have done since the beginning of a transaction (you can sometimes cancel a subpart of a transaction, but it's not much used)

```
commit
```


ROLLBACK automatically undoes everything. You can no longer do it after COMMIT.

```
rollback
```

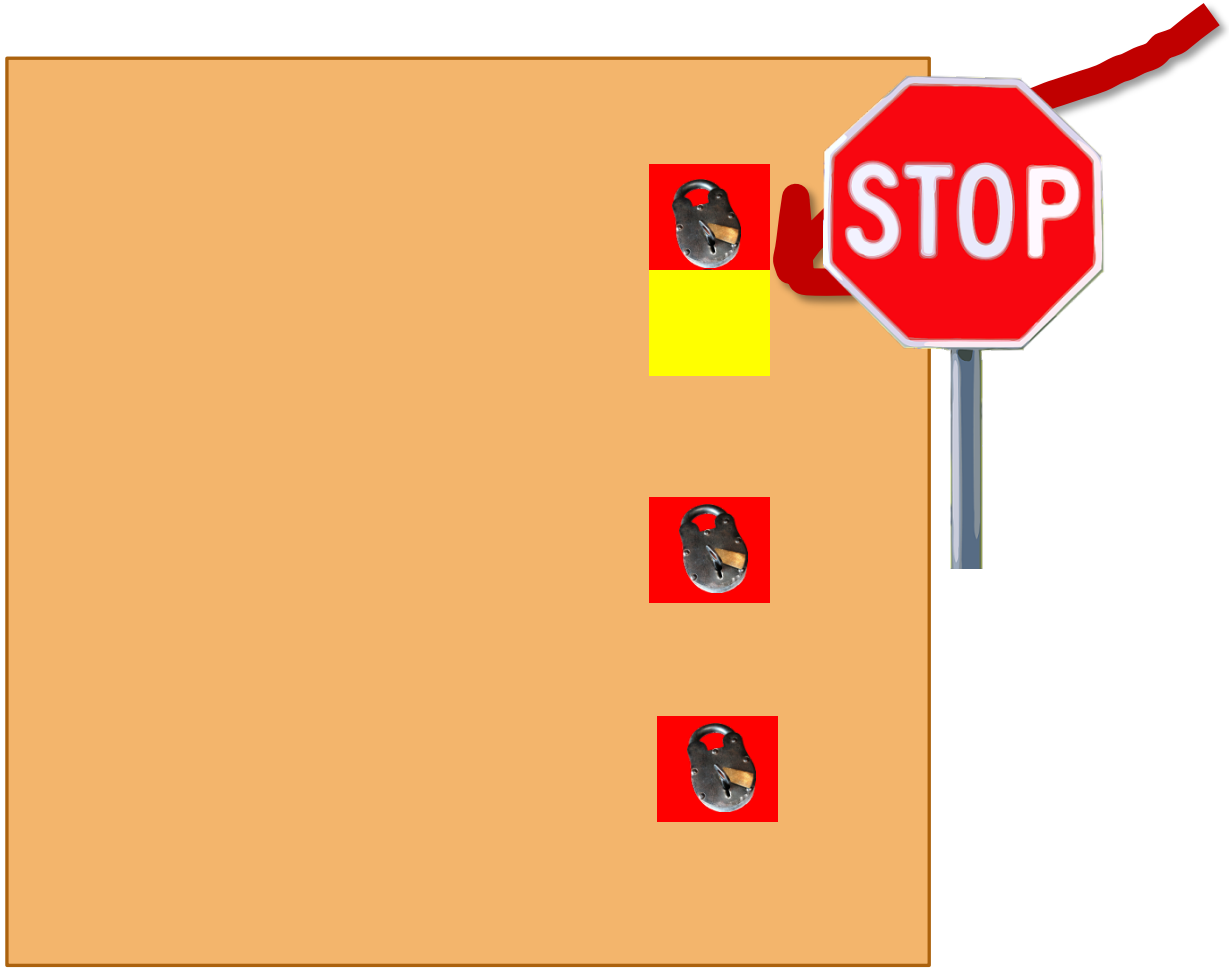
# rollback

| Account type | Account number | Balance |
|---|---|---|
| CURRENT ACNT | 1234567 | 200.00 |
| SAVINGS ACNT | 8765432 | 1600.00 |

If during a transfer the debit from one account went well but you couldn't credit the other one (in some countries, some saving accounts cannot hold more than a given amount, it would be a reason for failure) ROLLBACK will restore the original balance.

# begin transaction

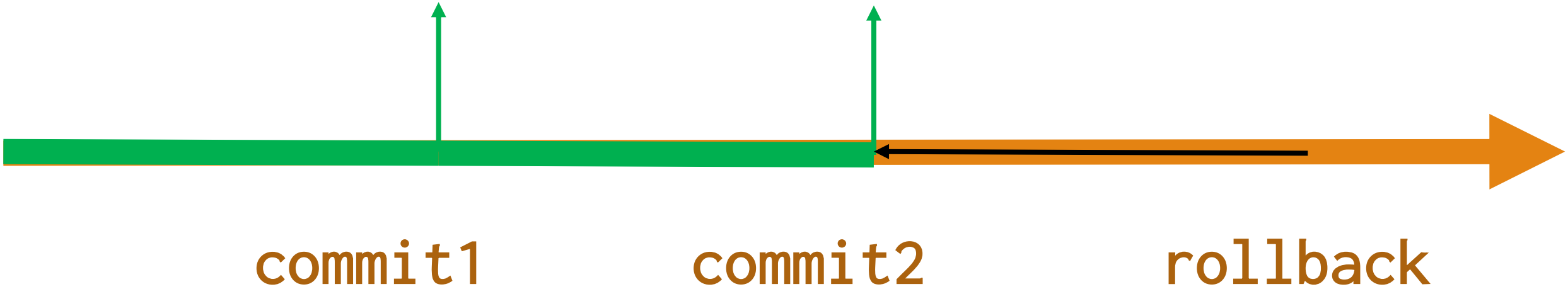
One important thing is concurrency: databases are usually meant to share data between many users. While you change data, and for all the duration of the transaction, other users are prevented by the DBMS from changing the same data.




Flickr: Andrew Magill

# commit

Every commit defines a consistent state of data, which is the "official" state of data at that very moment. A rollback takes you back to the last consistent state known. It all progresses by leaps, and each transaction is a leap.



# Beware of Autocommit

Many products (MySQL is one of them) start in "autocommit" mode, which means that every change is automatically commited and cannot be undone otherwise than by running the reverse operation (not always easy). Some interfaces such as JDBC (Java DataBase Connectivity) always start in autocommit mode, even when accessing products such as Oracle that never natively work in such a mode.

For products such as Oracle and MySQL, any change to the structure of the database (DDL operations) automatically commits all pending changes to data; DDL operations cannot usually be rolled back. This isn't the case with SQL Server or PostgreSQL, for which a transaction can contain DDL statements (they can be rolled back). Switching to another DBMS is a mine field.


`create`

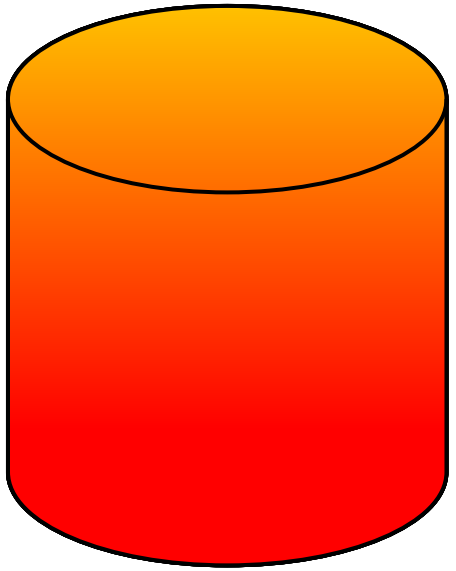
`drop, alter`

=

`commit`

One thing which is important to understand (we'll come back to it) is that transactions are a kind of "blind spot" in which the database switches from a consistent state to another consistent state, but during which internally things may not be consistent. This has a huge impact on a number of highly important practical questions; we have already mentioned that all users may not see the same data, but there is also the question: can we backup the database while it's active?

# Data Change



Begin Transaction

Commit

## ACCOUNTS

| Account type | Number | Balance |
|---|---|---|
| CURRENT ACNT | 1234567 | 200.00 |
| SAVINGS_ACNT | 8765432 | 1600.00 |

After having used the transfer-between-accounts example that you see everywhere, let's haste to say that a bank will never run two updates in one transaction for this kind of operation. Why? If you have ever looked at one of your bank statements, you have seen the list of operations since the last statement. What is stored is operations, and balances are recomputed once in a while.

# ACCOUNTS

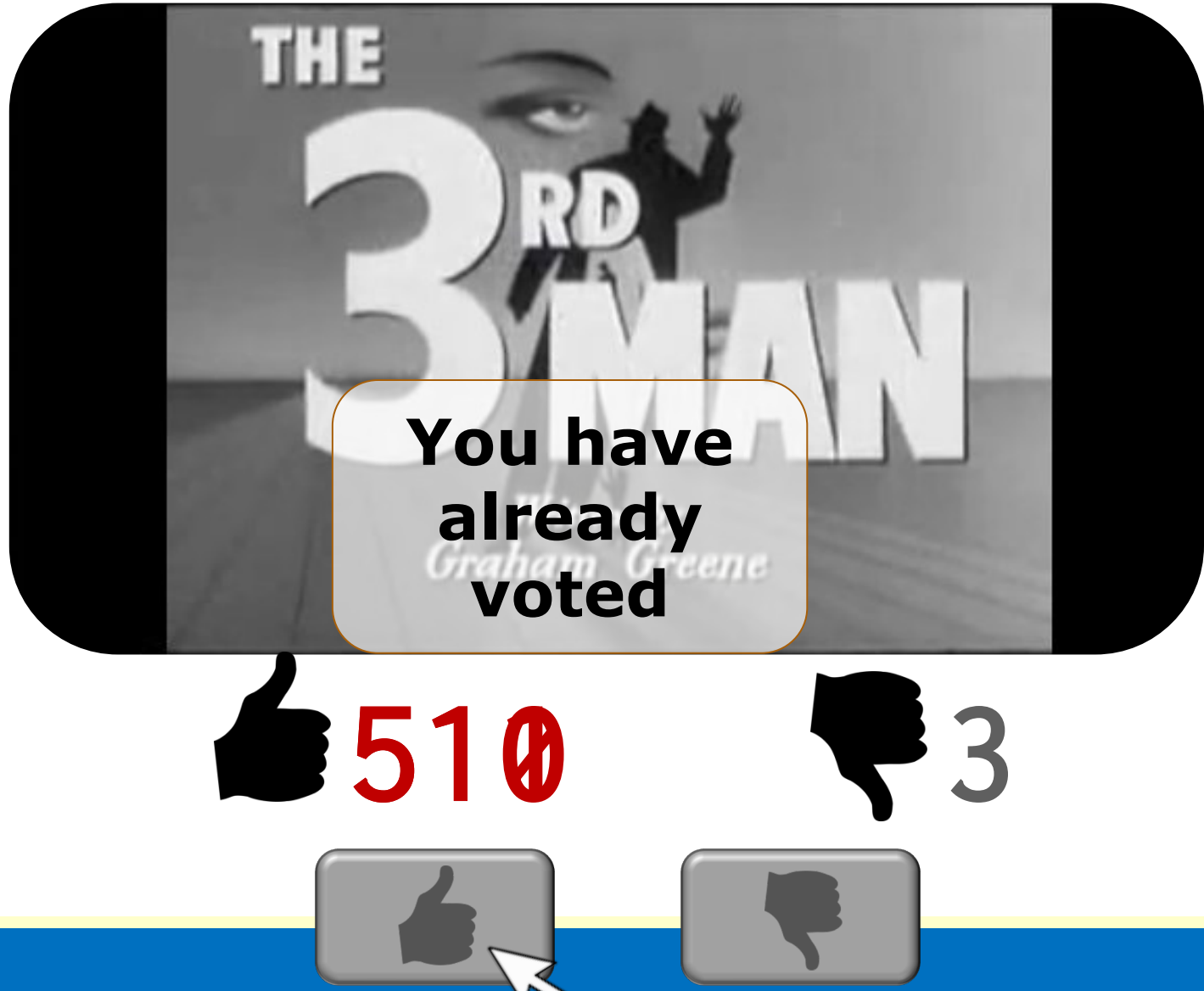
| Account type | Number | Balance | Date |
|---|---|---|---|
| CURRENT ACNT | 1234567 | 300.00 | 1-Sep |
| SAVINGS ACNT | 8765432 | 1600.00 | 1-Sep |

# OPERATIONS

| Account number | Amount | Operation | Date |
|---|---|---|---|
| 1234567 | 100.00 | DEBIT | 3-Sep |
| 8765432 | 100.00 | CREDIT | 3-Sep |

This is what will happen in the real world. A batch program may run daily, weekly or monthly to recompute the new balance. In between, balances can be recomputed from the old balance and by aggregating the latest operations. There are few updates, and a lot of inserts.

In the same way, do you think that when you click on a "Like" button, you simply update a counter? Certainly not, because you could artificially inflate popularity. Usually, on a web site you can only vote once for something from either the same IP address (which is transmitted to the web server), or the same member id if you must be signed-in to be allowed to vote (safer)



# Primary Key

| Videoid | Memberid |
|---|---|
| AGOivHmYbh8TGWzQ | 27097019 |
| BZVGhkMurOZDwqTH | 27097019 |
| FRpohgTYVbDEhkMr | 22109719 |
| AGOivHmYbh8TGWzQ | 78943285 |

A primary key will ensure that everybody can only vote once for a video. Once again, what appears as an update is in fact an insert.

| Id | Label | Active |
|---|---|---|
| 123 | TEST | N |

On the other hand, deletes often are virtual deletes; in fact they are updates of a flag. One good reason is foreign keys, which prevent physical deletion. A web store cannot physically delete an item no longer on sale but which appears in many past orders, you would have to delete much of your sales history.

# 9.3 Insert

Shiqi Yu 于仕琪

yusq@sustech.edu.cn

Most contents are from Stéphane Faroult's slides

We have already seen the syntax for inserting one row.

```
insert into table_name
  (column1, column2, ..., columnn)
values (value1, value2, ..., valuen),
       ...
       (valuep, valueq, ..., valuez)
```

Most products (exceptions are Oracle and SQLite) allow inserting several rows in one statement, with a comma separated list of row data between parentheses.

```
insert into table_name
values (value1, value2, ..., valuen)
```

If you don't specify the columns, it's understood as "all the columns, in the same order as they are displayed when running select *"

This is very dangerous in a program, because you can always add a column to a table, and often remove a column or change the default order.

What happens when you omit a column? The value inserted is the default one if defined, otherwise it will be NULL.

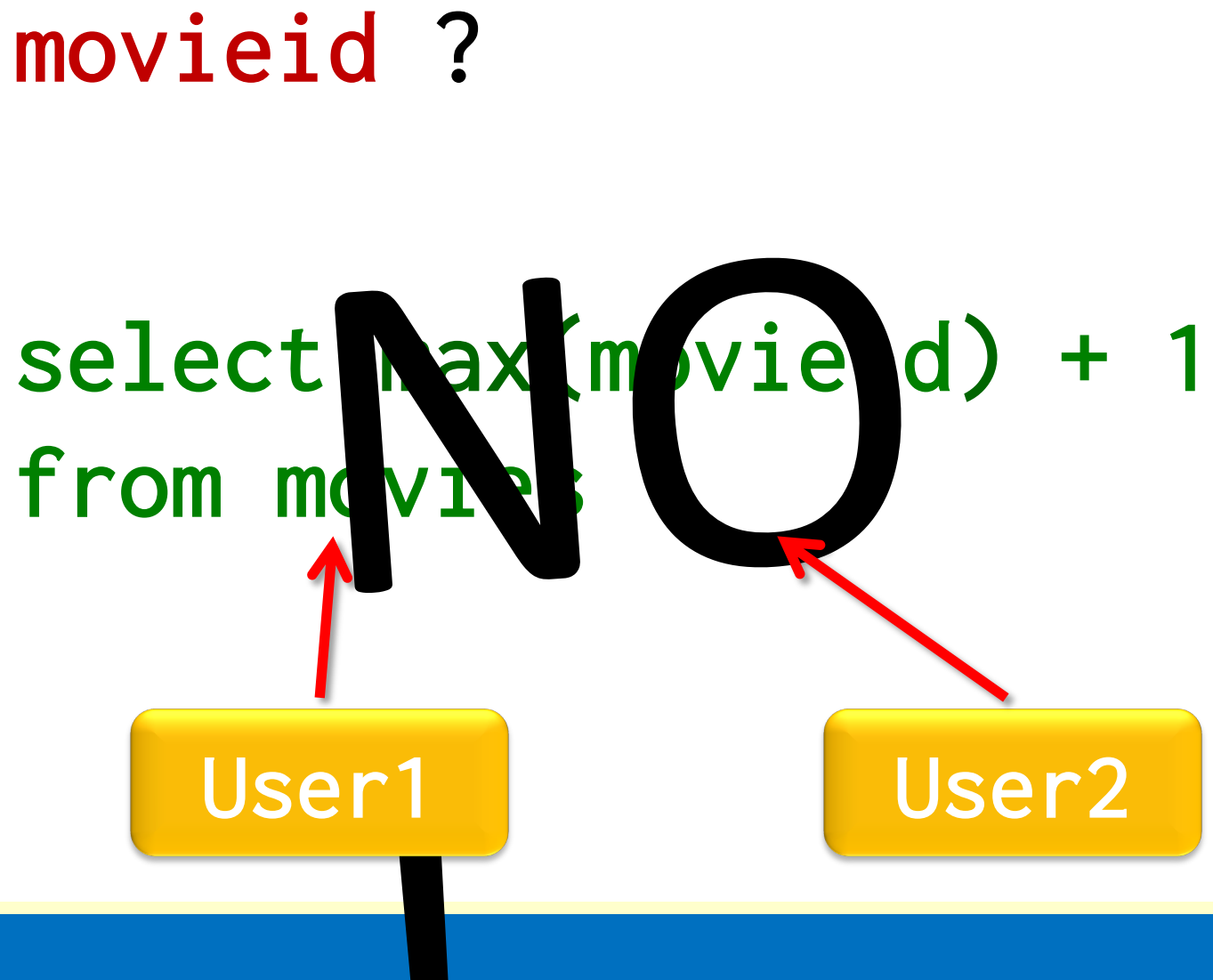
```
insert into table_name
  (col1, col2, col4)
values (value1, value2, value4)
```

```
create table <table_name>
             (...
              <column_name>    <data type>
                    default <default_value> not null,
              ...)
```

If I want to specify a default value, I do it when I create the table. If I have a default value for a mandatory column, it will be OK to omit it in an insert statement. Note that if you have a default value for a nullable column, nothing prevents you from explicitly inserting NULL, and the default value won't be used. Using CURRENT_TIMESTAMP as default for datetime columns is common.

An interesting question is how we populate these numerical identifiers that we are using as primary keys when what identifies a row in real life is a complication combination of columns.

Querying the next value to use is a sure recipe for conflicts. Several users may get the same one.



On a busy database odds are very high that two sessions will read the same value – and try to insert the same value. It will work for one, and the second one will get a constraint violation error. The solution is to let the system manage the generation of new identifiers, and there are two ways of doing it.

# SEQUENCE

```
create sequence movie_seq
```

You can use special database objects called sequences, which are simply number generators. By default they start with 1 and increase by 1 (they can reach values that are very, very big)




ORACLE

```
insert into movies(movieid, ...)
values(movie_seq.nextval, ...)
insert into credits(movieid, ...)
values(movie_seq.currval, ...)
```

Syntax varies, but you can obtain a new (guaranteed to be unique) number, and retrieve the last number you obtained for this sequence and this session.



```
insert into movies(movieid, ...)
values(next value for movie_seq, ...)
insert into credits(movieid, ...)
values(previous value for movie_seq,
       ...)
```


```
insert into movies(movieid, ...)
values(nextval('movie_seq'), ...)
insert into credits(movieid, ...)
values(currval('movie_seq'), ...)
```

# AUTO-NUMBERED COLUMN

```
create table movies
      (movieid
        ...
```





# AUTO-NUMBERED COLUMN

```
create table movies
      (movieid  int not null identity primary key,
       ...
```





# AUTO-NUMBERED COLUMN

```
create table movies
      (movieid  int generated as identity primary key,
       ...
```





# AUTO-NUMBERED COLUMN

```
create table movies
      (movieid  serial primary key,
       ...
```

As usual, syntax differs. PostgreSQL actually creates a sequence behind the scene, which it "attaches" to the table so that dropping the table drops the sequence.

# AUTO-NUMBERED COLUMN

```
create table movies
      (movieid  int not null auto_increment primary key,
       ...
```

# AUTO-NUMBERED COLUMN

```
create table movies
      (movieid  integer primary key,
       ...
```





# AUTO-NUMBERED COLUMN

define

```
movieseq.nextval
```

as default value for `movieid`

Oracle (since version 12,
it wasn't possible before)
can do it PostgreSQL style,
but more explicitly.

ORACLE >= 12c

If you insert a film with an auto-numbered column, you just omit the movieid from the INSERT statement, it will get automatically populated.

```
insert into movies(title, ...)
values('Some Movie Title', ...)
```

To retrieve the last value generated in your session, you use a special variable such as **@@identity** with SQL Server, or functions with other products (eg. **lastval()** with PostgreSQL or **last_insert_id()** with MySQL). This last value is often "across all tables", although sometimes you can retrieve the last value for a specific table.


```
insert into movies(title, ...)
values('Some Movie Title', ...)
insert into credits(movieid, ...)
values(@@identity, ...)
```


```
insert into movies(title, ...)
values('Some Movie Title', ...)
insert into credits(movieid, ...)
values(identity_val_local(), ...)
```


```
insert into movies(title, ...)
values('Some Movie Title', ...)
insert into credits(movieid, ...)
values(lastval(), ...)
```


```
insert into movies(title, ...)
values('Some Movie Title', ...)
insert into credits(movieid, ...)
values(last_insert_id(), ...)
```

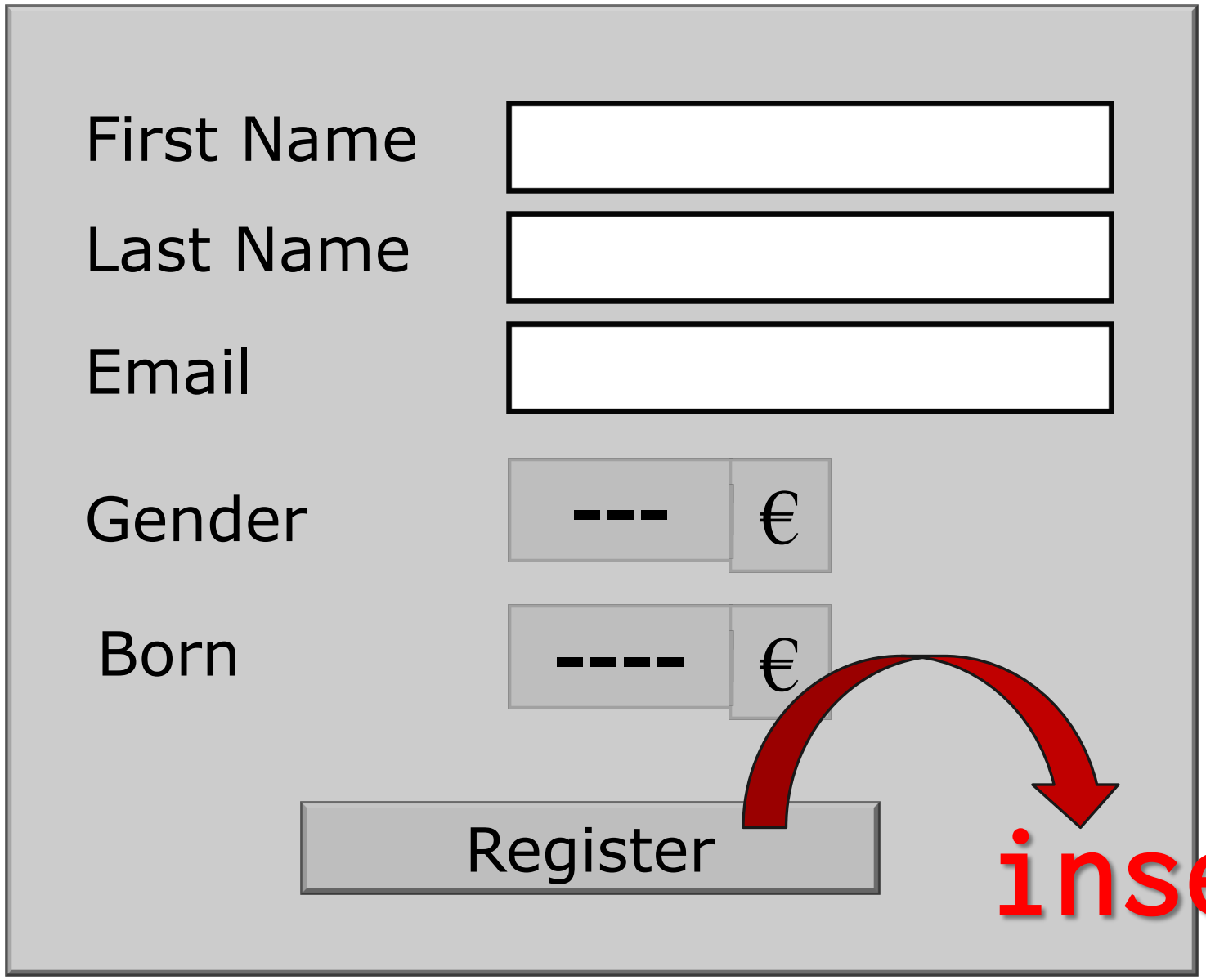

```
insert into movies(title, ...)
values('Some Movie Title', ...)
insert into credits(movieid, ...)
values(last_insert_rowid(), ...)
```

# 9.4 Loading Data from a File

Shiqi Yu 于仕琪

yusq@sustech.edu.cn

Most contents are from Stéphane Faroult's slides



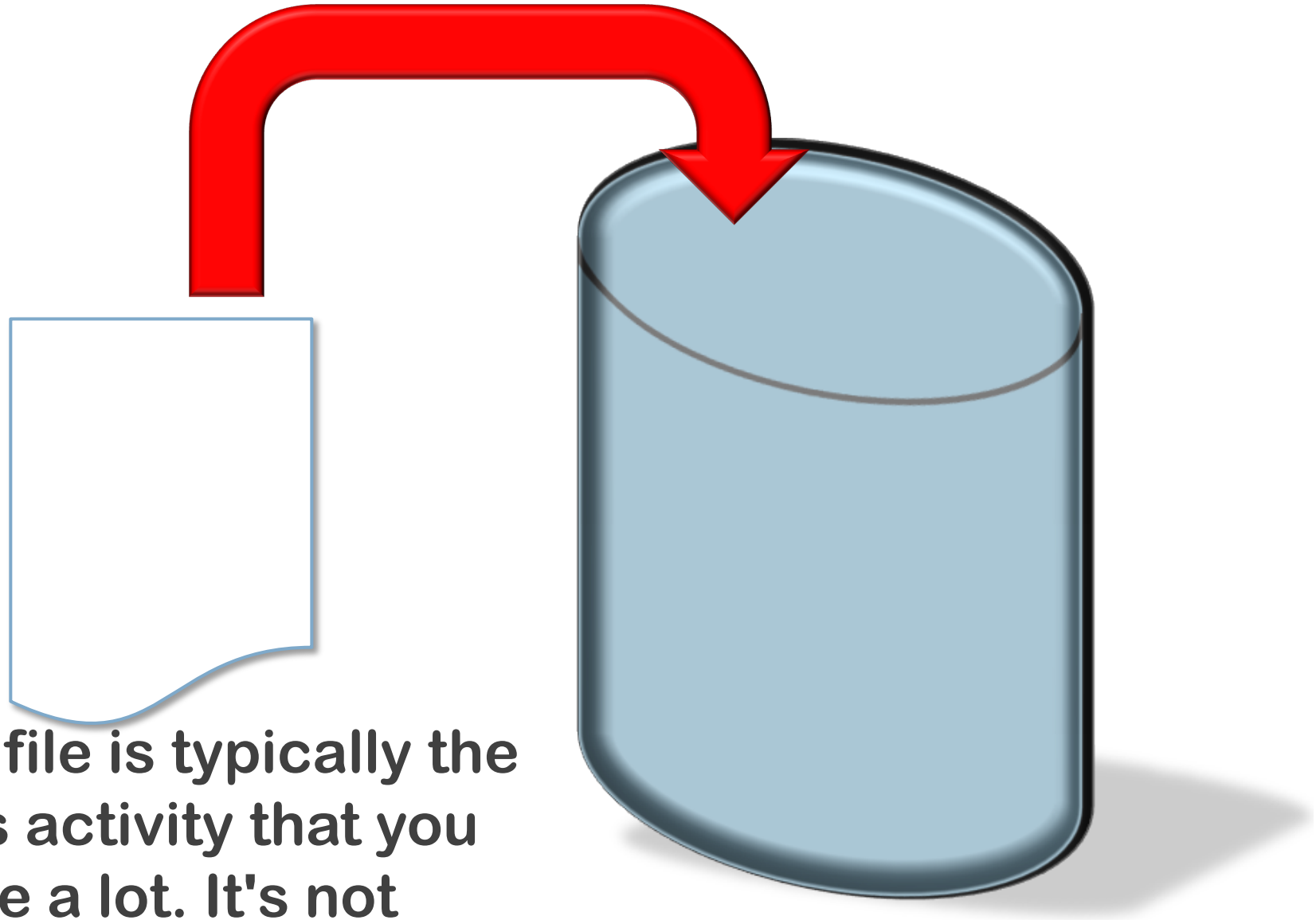

Usually in interactive programs, interfaces collect data for one row and issue the corresponding INSERT statement that adds the row to the right table.

However, it's extremely common that you want to upload data in bulk, either to populate a database initially or because data is exchanged between different systems by using files.


Flickr: Elbfoto

```
insert into table_name
  (column_1, column_2, ..., column_n)
select col_1, col_2, ..., col_n
from ...
```

Another way to massively insert data in a table is by inserting the result of a query. Of course, it assumes that the data is already in the database! What usually happens is that data is loaded "as is" into work tables (staging areas) that are badly normalized (they are hardly more than the table image of a file) then dispatched through INSERT ... SELECT ... statements to the well designed tables.



Loading data from a file is typically the type of unglamorous activity that you have to perform quite a lot. It's not always as easy as it should be.

```
Line 1
Line 2
```

First of all, when you have two lines in a text file, they are separated by one character if the file comes from a Linux system, two characters if it comes from a Windows system. When you transfer data from one to the other you need to be careful.

`Line 1\nLine 2`

`Line 1\r\nLine 2`

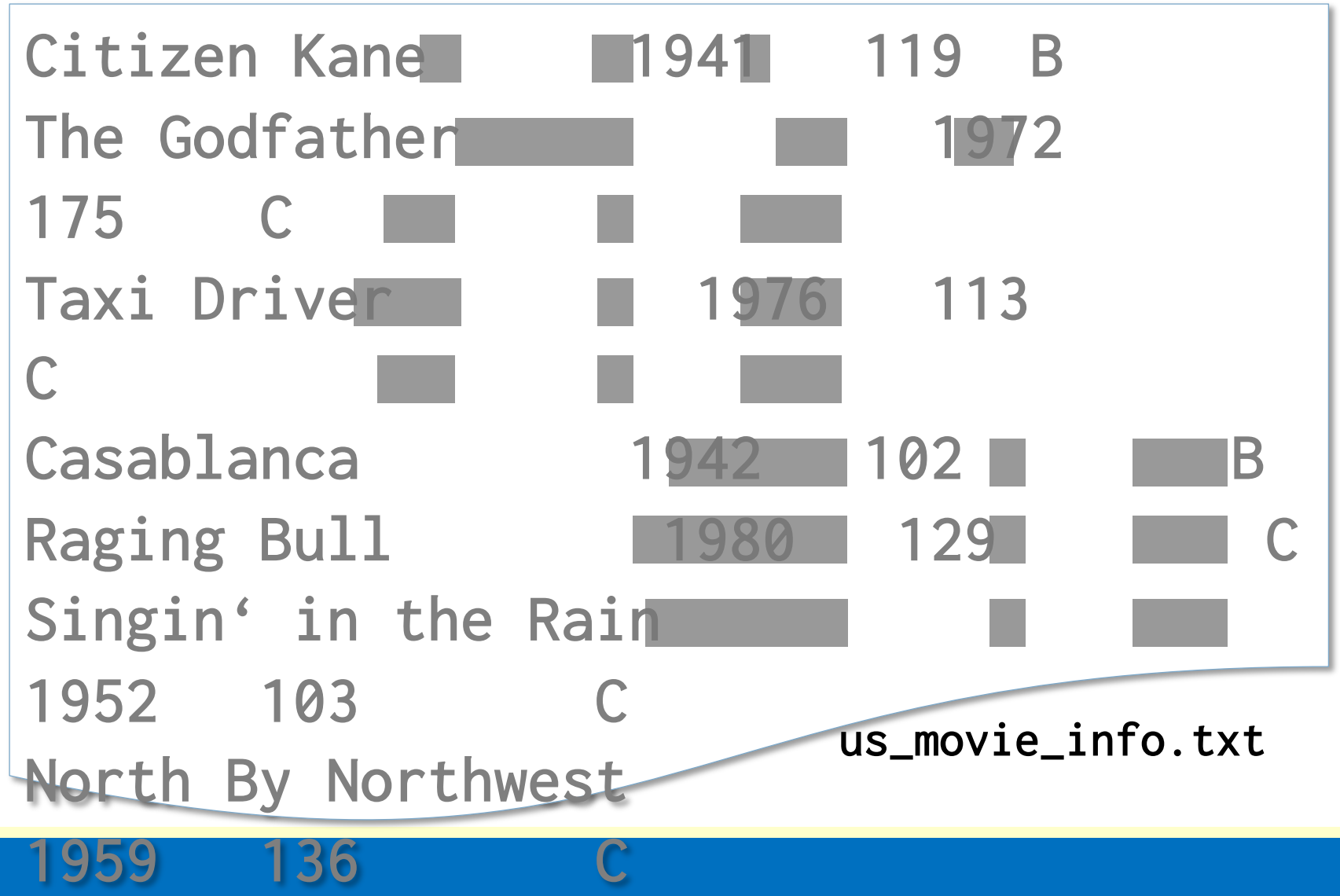
# csv


```
"Citizen Kane",1941,119,B
"The Godfather",1972,175,C
"Taxi Driver",1976,113,C
"Casablanca",1942,102,B
"Raging Bull",1980,129,C
"Singin' in the Rain",1952,103,C
"North By Northwest",1959,136,C
"Gone with the Wind",1939,226,C
```

us_movie_info.csv

A very popular format is the Comma Separated Values format, in which some fields may be enclosed by double quotes.

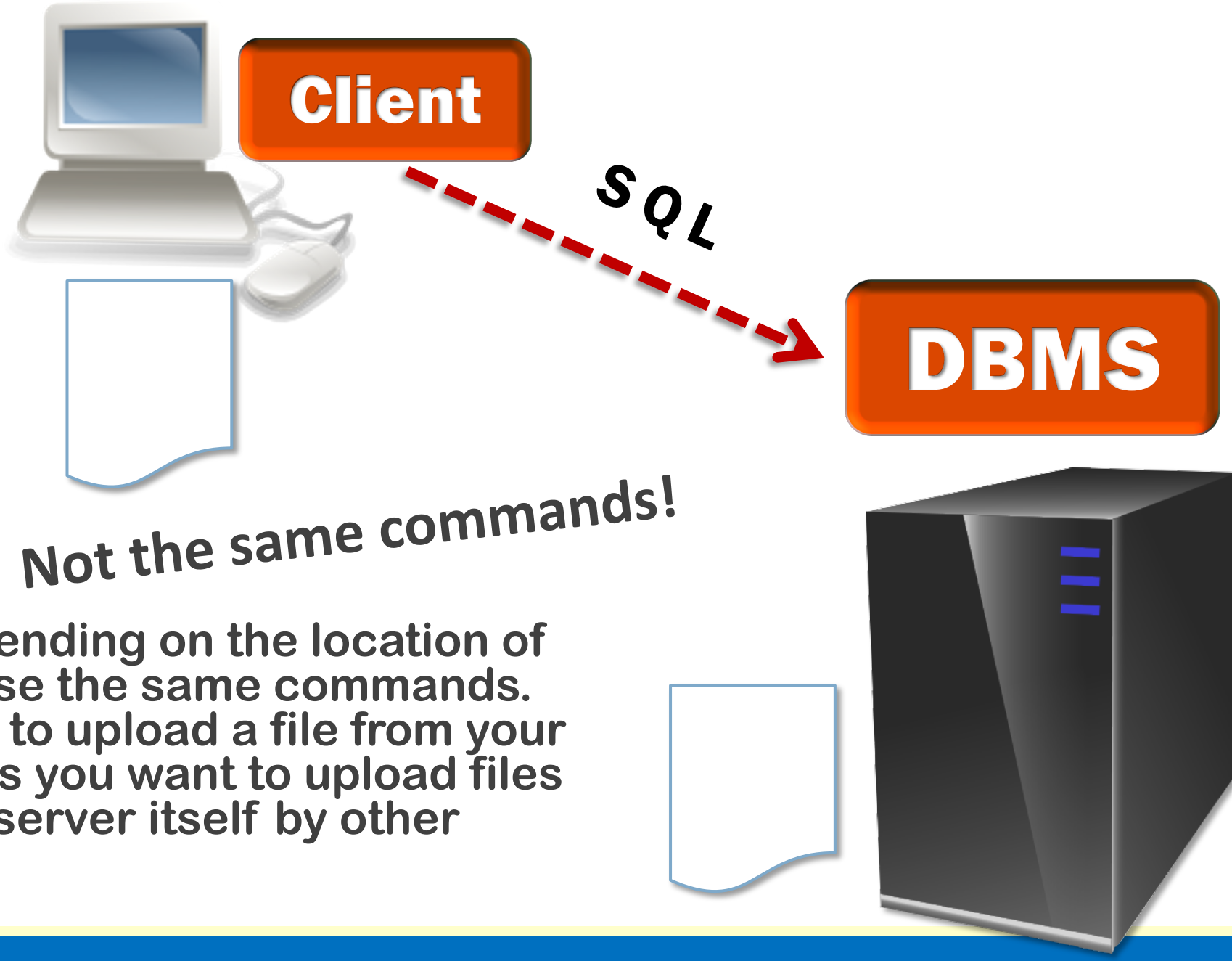
# Tab-separated




Another popular format is to have tab-separated fields.

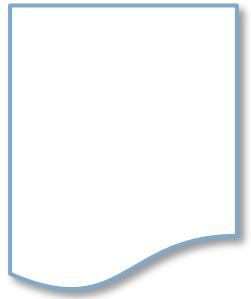
Let's create a "staging table" with one column to receive every field in the file. This syntax wouldn't work with MySQL (wants DECIMAL instead of NUMERIC), nor with Oracle (wants NUMBER).

```
create table us_movie_info
    (title              varchar(100) not null,
     year_released      numeric(4) not null,
     duration           int not null,
     color              char(1) not null,
     primary key(title, year_released))
```




Important point: depending on the location of your file, you won't use the same commands. Sometimes you want to upload a file from your computer. Sometimes you want to upload files stored on the DBMS server itself by other processes.

```
load data infile '/tmp/us_movie_info.txt'
into table us_movie_info
fields terminated by ','
optionally enclosed by '"'
```



With MySQL if the file is on the server you use this command. The default is to load a tab-separated file. For a CSV file you have to be more explicit about the format.

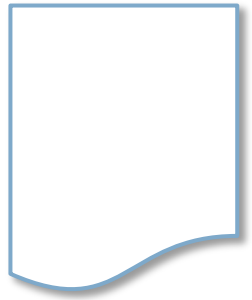
```
load data local infile 'us_movie_info.csv'
into table us_movie_info
fields terminated by ','
optionally enclosed by '"'
```

If the file is on your computer saying LOCAL will transfer it transparently before uploading.

Microsoft® SQL Server®

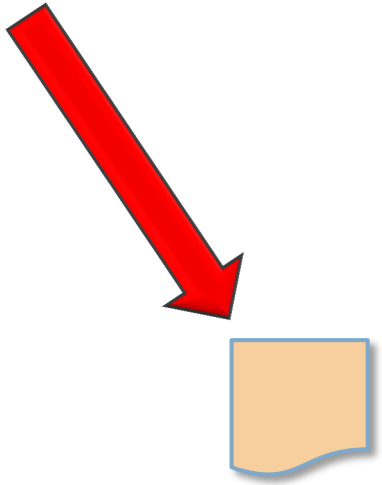
```
bulk insert us_movie_info
from 'C:\temp\us_movie_info.txt'
with (rowterminator='0x0a')
```

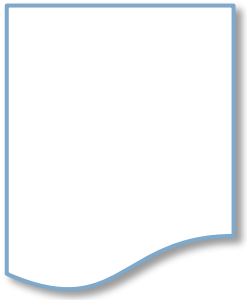
With SQL Server a special command can upload a file already on the server. Beware of Linux files. Default is tabseparated.



Microsoft® SQL Server®

```
bulk insert us_movie_info
from '/tmp/us_movie_info.csv'
with (formatfile='C:\temp\us_movie_info.fmt')
```





**For a CSV file you must use a special file that describes the format.**

# This is what a format file looks line. A (SQL Server) version number and a number of fields.

```
11.0
4
1 SQLCHAR 0 150 "\"," 1 title ""
2 SQLCHAR 0 4 "," 2 year_released ""
3 SQLCHAR 0 3 "," 3 duration ""
4 SQLCHAR 0 1 "\n" 4 color ""
```


# Then the description for each field: number, type, always 0, max length, what terminates it, and to which column it maps.

```
"Citizen Kane",1941,119,B
"The Godfather",1972,175,C
"Taxi Driver",1976,113,C
"Casablanca",1942,102,B
"Raging Bull",1980,129,C
"Singin' in the Rain",1952,103,C
"North By Northwest",1959,136,C
"Gone with the Wind",1939,226,C
```

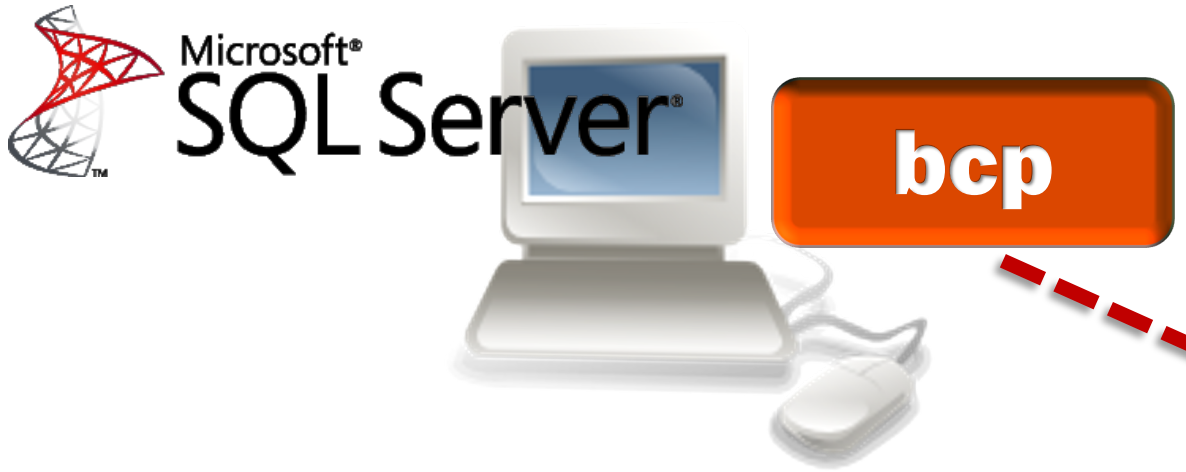
Microsoft® SQL Server®

# Virtual Table

```
select replace(title, '"', '') title,
       year_released,
       duration,
       case color
         when 'B' then 'N'
         when 'C' then 'Y'
       end color
from openrowset(bulk 'C:\temp\us_movie_info.csv',
                formatfile='C:\temp\us_movie_info.fmt') as virtual_table
```

SQL Server can do much better: a very special function called openrowset() allows you to see the file as a table.

It eliminates the need for a staging table: you can run an INSERT SELECT and transform data while loading.




Loading a file from your computer requires using a special utility called BCP (Bulk CoPy)

ORACLE®

* Registered in the database
* Privileges given by DBA

Oracle can do similar things to SQL Server but requires registering a directory from the database, and your account must have special rights given by a Database Administrator (DBA) on this directory.

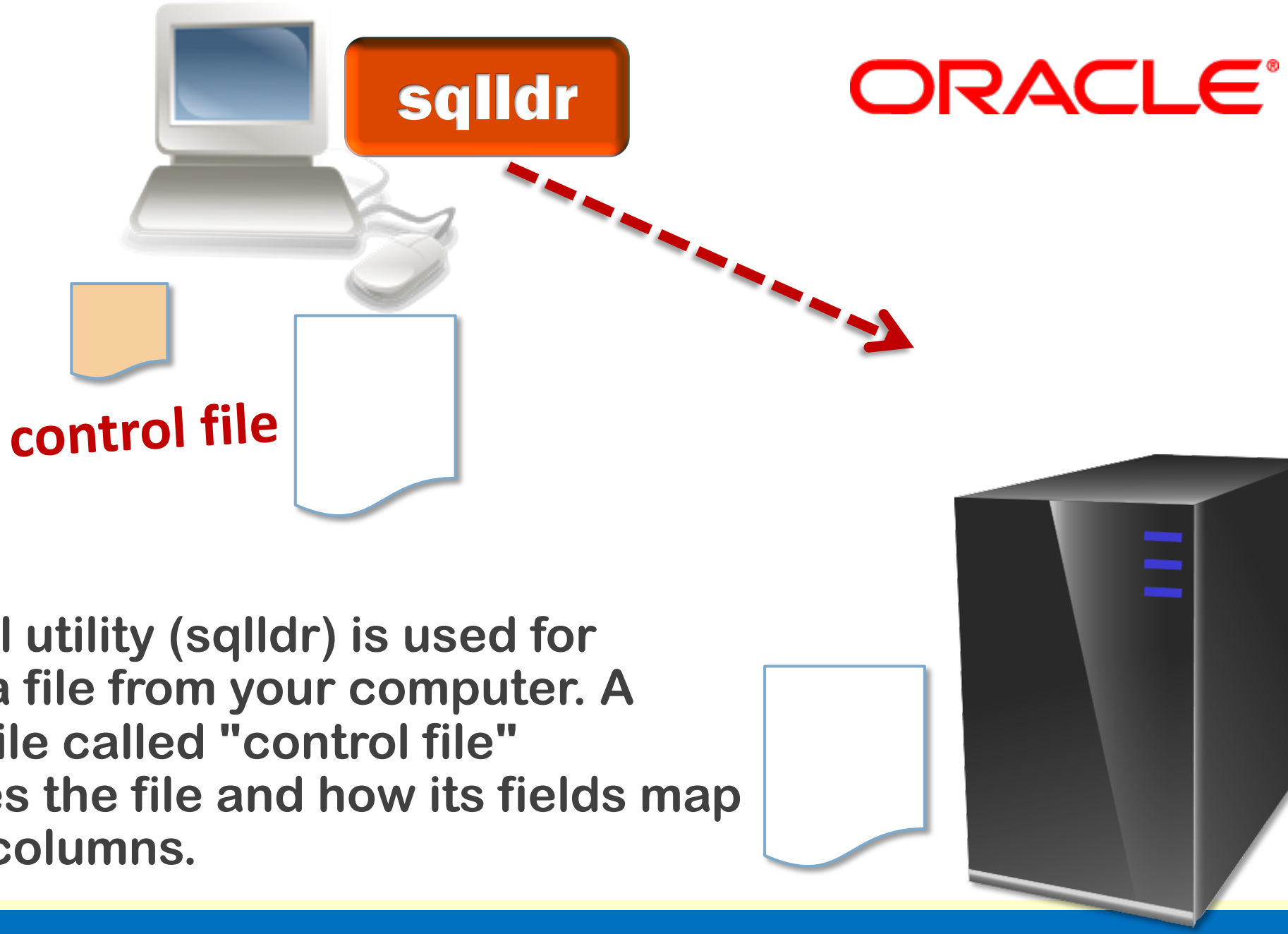
```
create table virtual_us_movie_info
          (title           varchar2(150),
           year_released   number(4),
           duration        number(3),
           color           char)
organization external (default directory input_dir
   access parameters
      (records delimited by '\n'
       fields terminated by '    ')
   location ('us_movie_info.txt'))
```

name given by DBA

Tab

ORACLE®

With Oracle, you can create a permanent virtual table that maps to a file. The default format is CSV. When you query the table, you actually read from the file. You can INSERT .. SELECT ...




A special utility (sqlldr) is used for loading a file from your computer. A special file called "control file" describes the file and how its fields map to table columns.

PostgreSQL

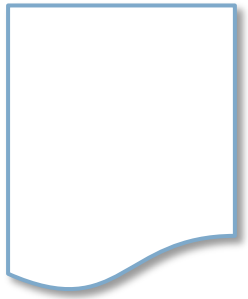
```
copy us_movie_info
from '/tmp/us_movie_info.txt'
```

PostgreSQL uses the (non-standard) SQL COPY command to load a file located on the server. The default format is tabseparated.

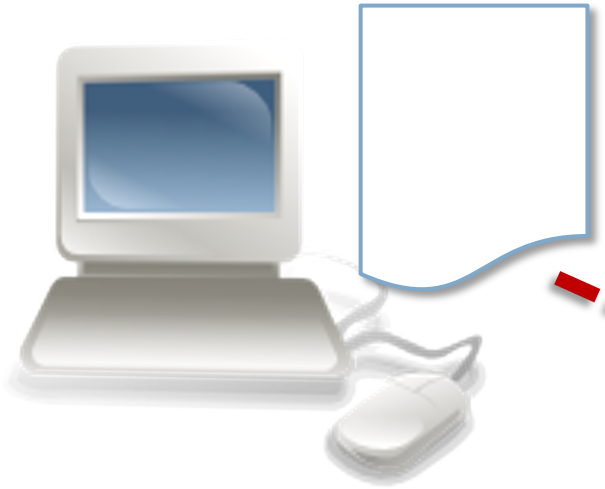
PostgreSQL

```
copy us_movie_info
from '/tmp/us_movie_info.csv'
with (format csv)
```

PostgreSQL uses the (non-standard) SQL COPY command to load a file located on the server. The default format is tabseparated.



PostgreSQL



```
\copy us_movie_info
from '/tmp/us_movie_info.csv'
with (format csv)
```

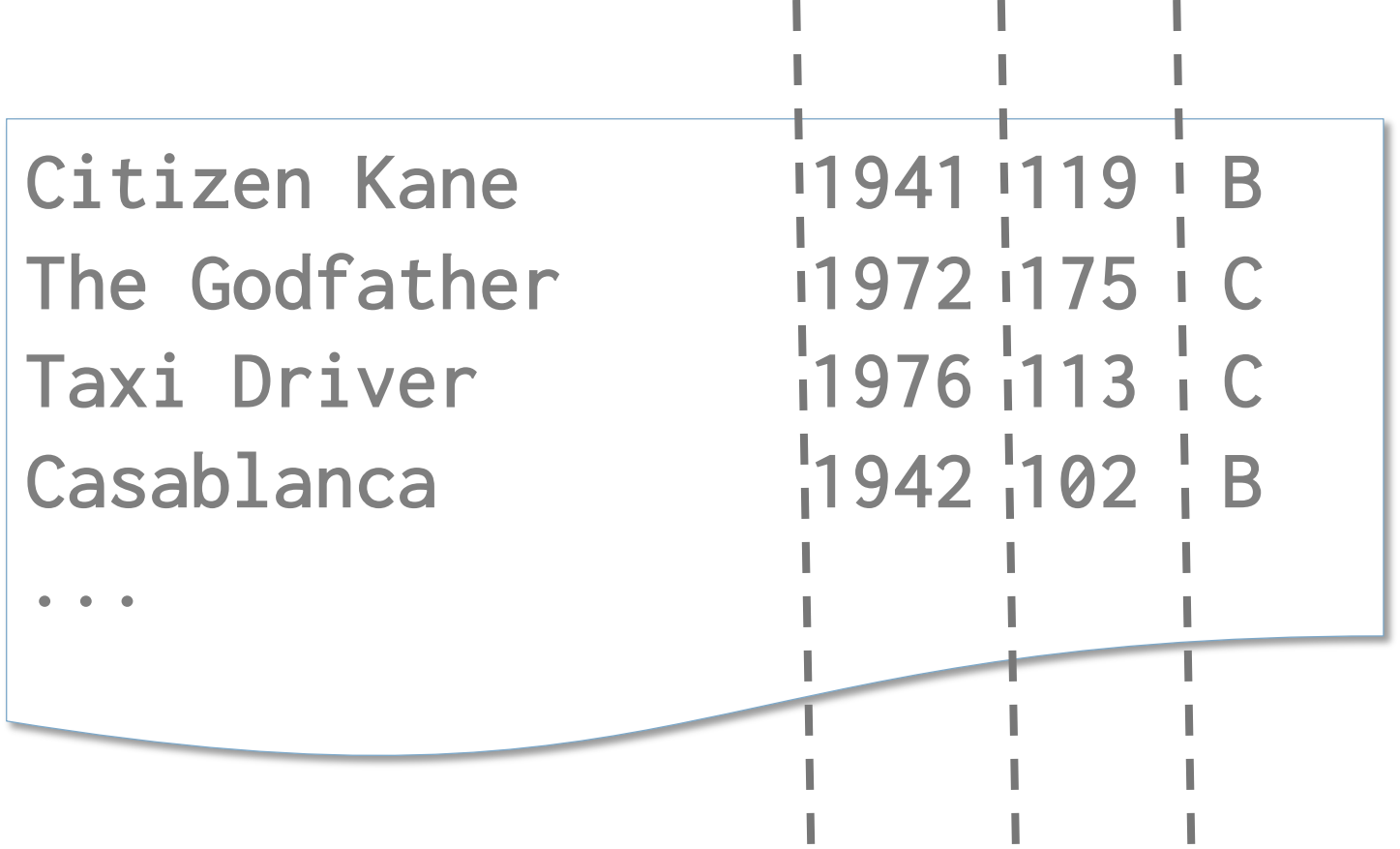
psql command

Almost the same command can be used with psql (the command-line tool) to load a file from your computer.


```
.separator ','
.import 'us_movie_info.csv' us_movie_info

.separator '   '
.import 'us_movie_info.txt' us_movie_info
```

With SQLite, of course client and server are the same machine, so it's far easier.

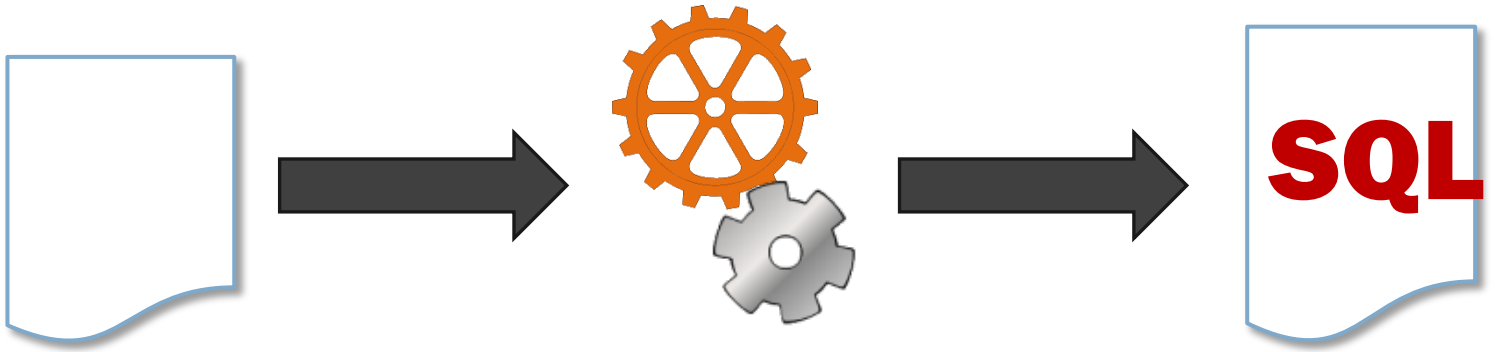



Although CSV and tab-separated files are most common, you can also encounter fixed-field files (fields always start at the same position), which all DBMS load easily.

```
<row><title>Citizen Kane</title><year>1941</year>...</row>
<row><title>The Godfather</title><year>1972</year>...</row>
<row><title>Taxi Driver</title><year>1976</year>...</row>
<row><title>Casablanca</title><year>1942</year>...</row>
...
```

XML is also a popular interchange format. The big products provide utilities for uploading XML files. JSON is also a format that is increasingly popular.

# When everything else fails ...




**awk**

**perl**

**python**

**php**

**...**

**Sometimes you encounter really weird text file formats. Using a scripting language to generate INSERT statements is usually the simplest solution.**