BDA 505: BIG DATA MANAGEMENT
01 – INTRO TO RDBMS, SQL AND POSTGRESQL

SERHAT ÇEVİKEL

WHAT IS RDBMS?

RDBMS stands for Relational Database Management System

Basis for SQL, and for all modern database systems like PostgreSQL, MS SQL Server, IBM DB2, Oracle, MySQL, MariaDB and Microsoft Access

WHAT MAKES RDBMS RELATIONAL?

What is the main difference between DBMS and RDBMS?

RDBMS (relational database management system) applications store data in a tabular form, while DBMS applications store data as files

In DBMS, there will be no “relation” between the tables, like in a RDBMS. Data is generally stored in either a hierarchical form or a navigational form

In a RDBMS, the tables will have an identifier called primary key

TABLE

The data in an RDBMS is stored in database objects called as tables

Basically a collection of related data entries and it consists of numerous columns and rows

A table is the most common and simplest form of data storage in a relational database

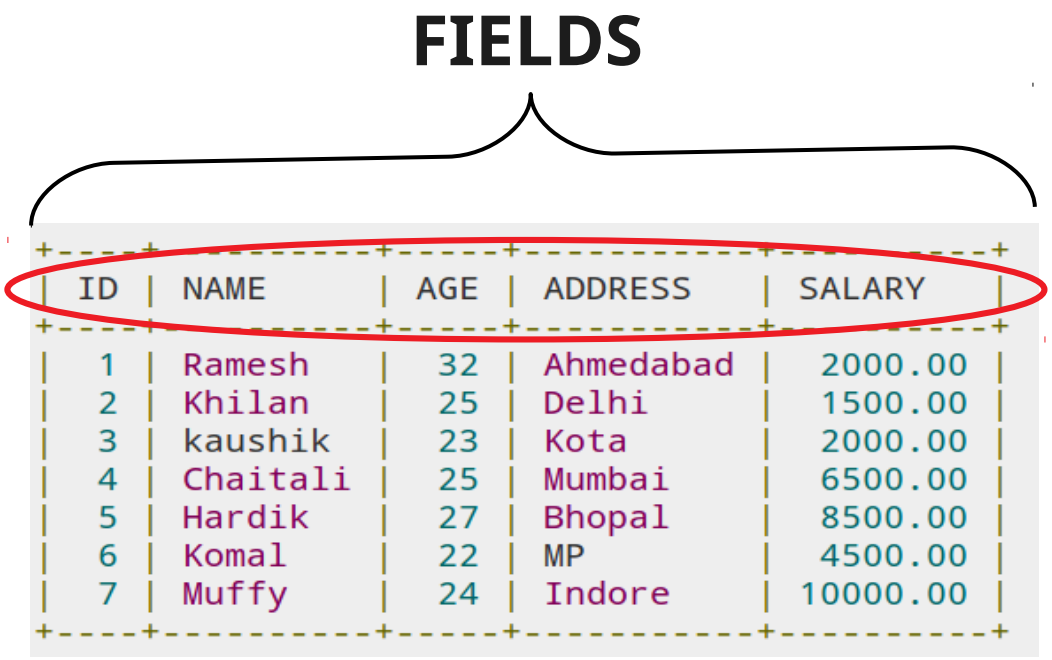
CUSTOMERS:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

FIELDS

Every table is broken up into smaller entities called fields

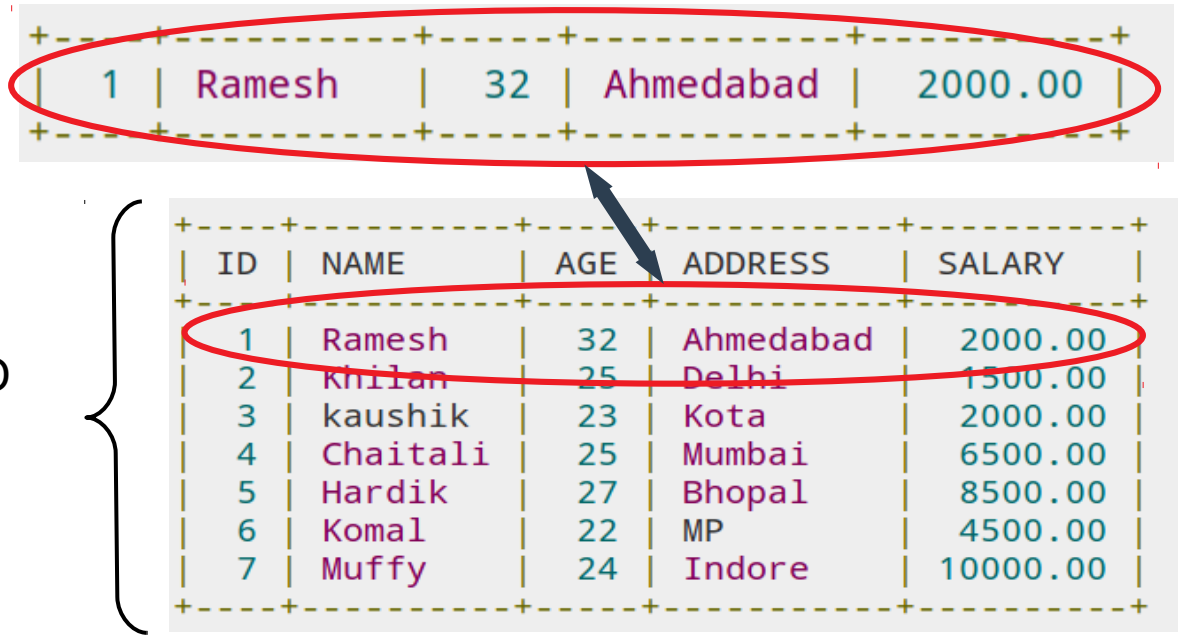
The fields in the CUSTOMERS table: ID, NAME, AGE, ADDRESS and SALARY.



RECORD OR ROW

A record is also called as a row of data is each individual entry that exists in a table

7 records in the CUSTOMERS table

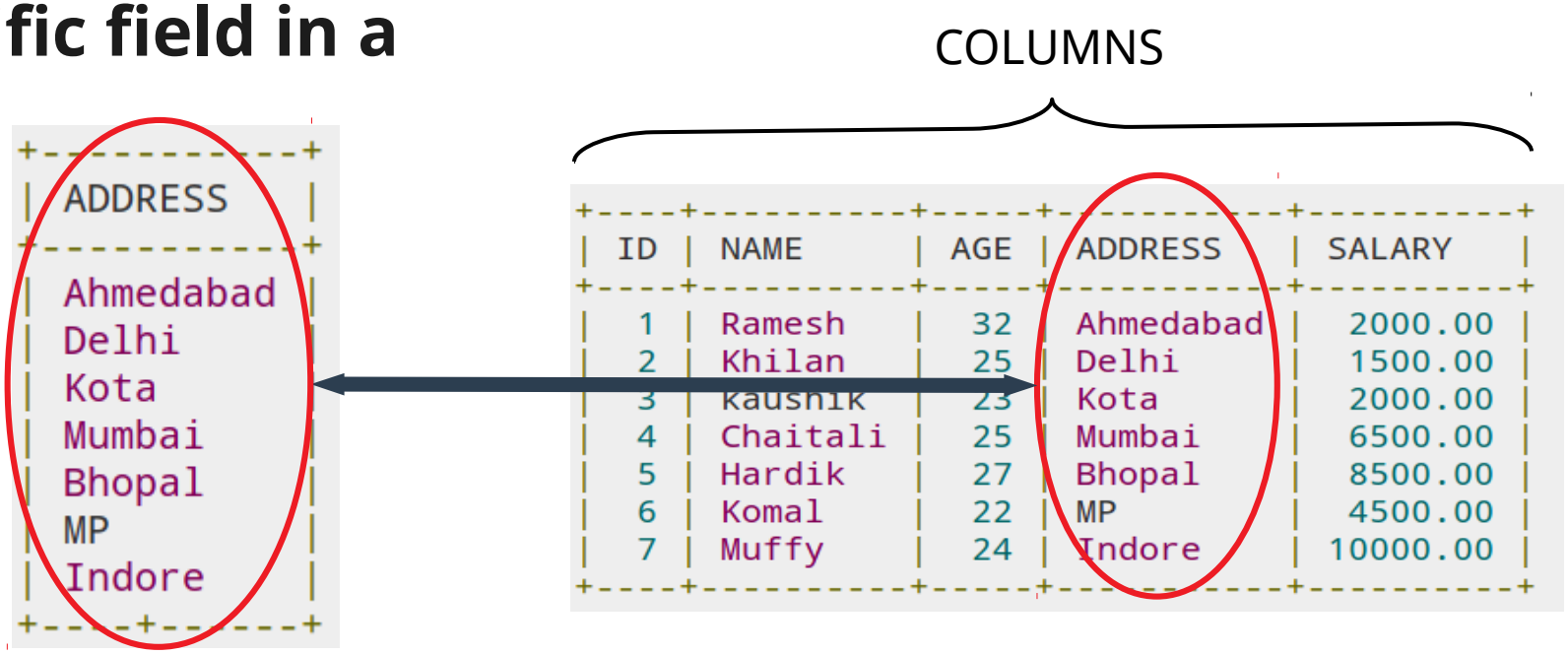


ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

RECORD / ROW

COLUMN

A vertical entity in a table that contains all information associated with a specific field in a table.



NULL VALUE

A NULL value in a table is a value in a field that appears to be blank

NULL value is different than a zero value or a field that contains spaces

A field with a NULL value is the one that has been left blank during a record creation

SQL

Structured Query Language

A computer language for storing, manipulating and retrieving data stored in a relational database

SQL

Allows users to access data in the relational database management systems

Allows users to describe the data

Allows users to define the data in a database and manipulate that data

Allows to embed within other languages using SQL modules, libraries & pre-compilers

Allows users to create and drop databases and tables

Allows users to create view, stored procedure, functions in a database

Allows users to set permissions on tables, procedures and views

SOME SQL COMMANDS

DDL (Data Definition Language) Commands:

CREATE: Creates a new table, a view of a table, or other object in the database

ALTER: Modifies an existing database object, such as a table

DROP: Deletes an entire table, a view of a table or other objects in the database

DML (Data Manipulation Language) Commands:

SELECT: Retrieves certain records from one or more tables

INSERT: Creates a record

UPDATE: Modifies records

DELETE: Deletes records

DCL (Data Control Language) Commands:

GRANT: Gives a privilege to user

REVOKE: Takes back privileges granted from user

INTEGRITY RULES

Entity Integrity: The rows in a relational table should all be distinct (no duplicates)
Domain Integrity: Enforces valid entries for a given column by restricting the type, the format, or the range of values
Referential integrity: Rows cannot be deleted, which are used by other records
Concept of NULL value: A database takes care of situations where data may not be available by using a null value to indicate that a value is missing. It does not equate to a blank or zero

ENTITY INTEGRITY AND PRIMARY KEY

When each row in a table is different, it is possible to use one or more columns to identify a particular row

This unique column or group of columns is called a primary key

Any column that is part of a primary key cannot be null; if it were, the primary key containing it would no longer be a complete identifier

SIMPLE SQL EXAMPLE

Employees table has 5 columns and six rows, one for each employee

Employee_Number	First_name	Last_Name	Date_of_Birth	Car_Number
10001	John	Washington	28-Aug-43	5
10083	Arvid	Sharma	24-Nov-54	null
10120	Jonas	Ginsberg	01-Jan-69	null
10005	Florence	Wojokowski	04-Jul-71	12
10099	Sean	Washington	21-Sep-66	null
10035	Elizabeth	Yamaguchi	24-Dec-59	null

EMPLOYEES TABLE AND PRIMARY KEY

The primary key for this table would generally be the employee number because each one is guaranteed to be different. (A number is also more efficient than a string for making comparisons.)

It would also be possible to use First_Name and Last_Name because the combination of the two also identifies just one row

Using the last name alone would not work because there are two employees with the last name of "Washington."

Employee_Number	First_name	Last_Name	Date_of_Birth	Car_Number
10001	John	Washington	28-Aug-43	5
10083	Arvid	Sharma	24-Nov-54	null
10120	Jonas	Ginsberg	01-Jan-69	null
10005	Florence	Wojokowski	04-Jul-71	12
10099	Sean	Washington	21-Sep-66	null
10035	Elizabeth	Yamaguchi	24-Dec-59	null

SELECT STATEMENT

A SELECT statement, also called a query, is used to get information from a table. It specifies one or more column headings, one or more tables from which to select, and some criteria for selection

The RDBMS returns rows of the column entries that satisfy the stated requirements

SELECT STATEMENT

Employee_Number	First_name	Last_Name	Date_of_Birth	Car_Number
10001	John	Washington	28-Aug-43	5
10083	Arvid	Sharma	24-Nov-54	null
10120	Jonas	Ginsberg	01-Jan-69	null
10005	Florence	Wojokowski	04-Jul-71	12
10099	Sean	Washington	21-Sep-66	null
10035	Elizabeth	Yamaguchi	24-Dec-59	null

FIRST_NAME	LAST_NAME
John	Washington
Florence	Wojokowski

Columns to be returned : SELECT First_Name, Last_Name

Source table name : FROM Employees

Selection criteria : WHERE Car_Number IS NOT NULL

WHERE CLAUSE

Provides the criteria for selecting values in SELECT statement

Can have logical tests (equality, number comparison, text comparison)

Can combine tests with logical operators (and, or, etc)

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Last_Name LIKE 'Washington%'
```

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Car_Number = 12
```

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Employee_Number > 10005
```

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Employee_Number < 10100 and Car_Number IS NULL
```

FROM DBMS TO RDBMS: JOINS

It is possible to get data from more than one table in what is called a join

Suppose one wanted to find out who has which car, including the license plate number, mileage, and year of car

This information is stored in another table, Cars

Car_Number	License_Plate	Mileage	Year
5	ABC123	5000	1996
12	DEF123	7500	1999

JOINS AND FOREIGN KEY

There must be one column that appears in both tables in order to relate them to each other

This column, which must be the primary key in one table, is called the foreign key in the other table

The column that appears in two tables is Car_Number:

- The primary key for the table Cars

- The foreign key in the table Employees

A foreign key must either be null or equal to an existing primary key value of the table to which it refers

JOIN EXAMPLE

Return the first and last names of the employees who have company cars, along with the extra information for cars

JOIN EXAMPLE

Employee_Number	First_name	Last_Name	Date_of_Birth	Car_Number
10001	John	Washington	28-Aug-43	5
10083	Arvid	Sharma	24-Nov-54	null
10120	Jonas	Ginsberg	01-Jan-69	null
10005	Florence	Wojokowski	04-Jul-71	12
10099	Sean	Washington	21-Sep-66	null
10035	Elizabeth	Yamaguchi	24-Dec-59	null

Car_Number	License_Plate	Mileage	Year
5	ABC123	5000	1996
12	DEF123	7500	1999

```
SELECT Employees.First_Name,
Employees.Last_Name, Cars.License_Plate,
Cars.Mileage, Cars.Year

FROM Employees, Cars

WHERE Employees.Car_Number =
Cars.Car_Number
```

FIRST_NAME	LAST_NAME	LICENSE_PLATE	MILEAGE	YEAR
John	Washington	ABC123	5000	1996
Florence	Wojokowski	DEF123	7500	1999

JOIN EXAMPLE

"SELECT" shows the columns to be selected from either of the two tables

Note that, table name is prefixed to avoid confusion

```
SELECT Employees.First_Name, Employees.Last_Name,
Cars.License_Plate, Cars.Mileage, Cars.Year

FROM Employees, Cars

WHERE Employees.Car_Number = Cars.Car_Number
```

"FROM" clause lists both table names to be joined

"WHERE" clause matches common field "car number": the foreign key of "Employees" and primary key of "Cars" tables

POSTGRESQL

Cross-platform object-relational database management system (ORDBMs for short).
Developed and maintained by EnterpriseDB (https://www.enterprisedb.com/)

WHY POSTGRESQL

Multiplatform: Available for Linux, *nixes, Windows, etc.

Open source: Transparent, no backdoors, active community, passionate developers, quick release cycle and bug fixes and ... free

Scalable: It can work under a low and heavy load in order to meet a users' need

Multiple Language Interfaces: Many interfaces are available for PostgreSQL like Java (JDBC), ODBC, Perl, Python, Ruby, C, C++, PHP, Lisp, Scheme Qt etc

Supports migration from other major proprietary and open source databases: Data migration is very easy to PostgreSQL or from PostgreSQL to any other database

WHY POSTGRESQL

The PostgreSQL project focuses on the following objectives according to its website:

Robust, high-quality software with maintainable, well-commented code

Low maintenance administration for both embedded and enterprise use

Standards-compliant SQL, interoperability, and compatibility

Performance, security, and high availability

IMDB DATASETS

To learn RDBMS and SQL within big data context, we need a relatively large dataset with multiple tables that can be joined through keys

IMDB dataset consists of 6 large tables that have common fields so is suitable for our needs

It is stored on Amazon AWS through S3 bucket service as archive files with .gz extension

Gzipped files total 371 MB, extracted files total 1.4 GB

IMDB DATASETS

title.basics.tsv:

tconst (string) - alphanumeric unique identifier of the title

titleType (string) – the type/format of the title (e.g. movie, short, tvseries, tvepisode, video, etc)

primaryTitle (string) – the more popular title / the title used by the filmmakers on promotional materials at the point of release

originalTitle (string) - original title, in the original language

isAdult (boolean) - 0: non-adult title; 1: adult title.

startYear (YYYY) – represents the release year of a title. In the case of TV Series, it is the series start year.

endYear (YYYY) – TV Sereis end year. '\N' for all other title types

runtimeMinutes – primary runtime of the title, in minutes

genres (string array) – includes up to three genres associated with the title

title.crew.tsv:

tconst (string)

directors (array of nconsts) - director(s) of the given title

writers (array of nconsts) – writer(s) of the given title

title.principals.tsv:

tconst (string)

principalCast (array of nconsts) – title's top-billed cast

title.episode.tsv:

tconst (string) - alphanumeric identifier of episode

parentTconst (string) - alphanumeric identifier of the parent TV Series

seasonNumber (integer) – season number the episode belongs to

episodeNumber (integer) – episode number of the tconst in the TV series.

title.ratings.tsv:

tconst (string)

averageRating – weighted average of all the individual user ratings

numVotes - number of votes the title has received

name.basics.tsv:

nconst (string) - alphanumeric unique identifier of the name/person

primaryName (string)– name by which the person is most often credited

birthYear – in YYYY format

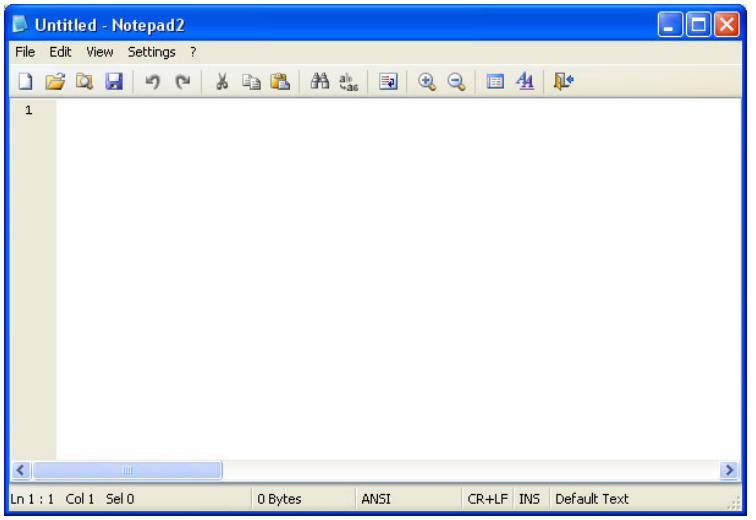
deathYear – in YYYY format if applicable, else '\N'

primaryProfession (array of strings)– the top-3 professions of the person

knownForTitles (array of tconsts) – titles the person is known for

HOW CAN WE OPEN/VIEW/ PROCESS THESE FILES?

?



EXPLORE THE DATA WITHOUT RDBMS

Get some info on the datasets (as files) using standard UNIX command line tools and UNIX pipes

UNIX pipe (“|”) is a powerful tool for UNIX programming that takes in standard output (STDOUT) from a process and redirects it to standard input (STDIN) to another processes

It enables powerful one-liner commands that does much with less

First extract the gz files under “imdb/gz” directory into tsv (tab separated values) files under “imdb/tsv” directory:

```
[s@SS imdb]$ mkdir tsv; gzip -k gz/*.gz; mv gz/*.tsv tsv; cd tsv
```

The -k flag in gzip command keeps the original gz files

SOME STATS

List files (ls) with details (l), human readable sizes (h) and sorted in size (S):

```
[s@SS tsv]$ ls -lSh
total 1.4G
-rw-r--r-- 1 s s 474M Sep 26 08:23 name.basics.tsv
-rw-r--r-- 1 s s 365M Sep 26 08:23 title.basics.tsv
-rw-r--r-- 1 s s 281M Sep 26 08:23 title.principals.tsv
-rw-r--r-- 1 s s 133M Sep 26 08:23 title.crew.tsv
-rw-r--r-- 1 s s  72M Sep 26 08:23 title.episode.tsv
-rw-r--r-- 1 s s  13M Sep 26 08:23 title.ratings.tsv
```

ROW COUNTS

List the files under tsv dir (find), for each file (while), print the name and a tab (printf), get the line count (cat, wc -l) and print the count as human readable numbers (numfmt)

Longest file is name.basics.tsv with 7.8M lines

```
[s@SS tsv]$ find . -mindepth 1 | while read line; do printf "%s\t" $line ;\
 cat $line | wc -l | numfmt --to=iec; done
./title.principals.tsv 3.9M
./title.basics.tsv     4.4M
./title.crew.tsv       4.4M
./title.episode.tsv    2.9M
./name.basics.tsv      7.8M
./title.ratings.tsv    750K
```

COLUMN COUNTS

List the files under tsv dir (find), for each file (while), print the name and a tab (printf), get the word count of the first line (head, wc -w) and print the count as human readable numbers (numfmt)

Widest file is title.basics.tsv with 9 fields

```
[s@SS tsv]$ find . -mindepth 1 | while read line; do printf "%s\t" $line ; head -1 $line | wc -w; done
./title.principals.tsv 2
./title.basics.tsv     9
./title.crew.tsv       3
./title.episode.tsv    4
./name.basics.tsv      6
./title.ratings.tsv    3
```

HEAD OF FILES

List the files under tsv dir (find), for each file (while), print the name and line feed (printf), get the first two lines (head), align the columns (column), print an empty line (echo)

2nd field of title.principals should be molten
Last two fields of name.basics should be split

```
[s@SS tsv]$ find . -mindepth 1 | while read line; do printf "%s\n" $line ; head -2 $line | column -t -s $'\t'; echo ""; done
./title.principals.tsv
tconst    principalCast
tt0000001  nm1588970,nm0374658,nm0005690

./title.basics.tsv
tconst    titleType  primaryTitle  originalTitle  isAdult  startYear  endYear  runtimeMinutes  genres
tt0000001  short     Carmencita    Carmencita     0        1894       \N       1               Documentary,Short

./title.crew.tsv
tconst    directors  writers
tt0000001  nm0005690  \N

./title.episode.tsv
tconst    parentTconst  seasonNumber  episodeNumber
tt0041951  tt0041038    1             9

./name.basics.tsv
nconst    primaryName   birthYear  deathYear  primaryProfession            knownForTitles
nm0000001  Fred Astaire  1899       1987       soundtrack,actor,miscellaneous  tt0120689,tt0027125,tt0028333,tt0050419

./title.ratings.tsv
tconst    averageRating  numVotes
tt0000001  5.8           1306
```

UNIQUE VALUES OF A FIELD

Get the unique values of the endYear field in title.basics.tsv file: Print the 7th column (awk), sort (sort), get unique values (uniq) and print in 8 columns (pr)

Note that the NULL value is \N: If that is not compatible with the default value of PostgreSQL, we should substitute it

```
[s@SS tsv]$ awk 'BEGIN{FS="\t"} {print $7}' title.basics.tsv | sort | uniq | pr -8 -t
1924    1946    1958    1970    1982    1994    2005    2016
1927    1947    1959    1971    1983    1995    2006    2017
1932    1948    1960    1972    1984    1996    2007    2018
1933    1949    1961    1973    1985    1997    2008    2019
1935    1950    1962    1974    1986    1998    2009    2020
1936    1951    1963    1975    1987    1999    2010    2021
1937    1952    1964    1976    1988    2000    2011    2023
1938    1953    1965    1977    1989    2001    2012    2024
1939    1954    1966    1978    1990    2002    2013    2025
1941    1955    1967    1979    1991    2003    2014    endYear
1942    1956    1968    1980    1992    2004    2015    \N
1945    1957    1969    1981    1993
```

COUNT OF UNIQUE VALUES OF A FIELD

Get the counts of unique values of the endYear field in title.basics.tsv file: Print the 7th column (awk), sort (sort), get unique values (uniq) and print in 6 columns (column)

```
[s@SS tsv]$ awk 'BEGIN{FS="\t"}{print $7}' title.basics.tsv | sort | uniq -c | sort -nr | column -c 100
4501613 \N     775 2002     351 1986     214 1977     98 1955      3 2025
1755 2017      749 2001     328 1984     212 1972     93 1953      3 2023
1643 2016      676 2000     323 1985     208 1968     88 1954      3 1941
1302 2015      640 1999     301 1982     208 1967     84 1952      3 1936
1196 2013      633 1998     289 1981     203 1965     81 1951      2 1945
1133 2012      603 1997     273 1980     194 1969     57 1950      2 1942
1131 2014      525 1996     271 1979     193 1961     54 1949      1 endYear
1012 2010      505 1995     266 1978     182 1966     27 2019      1 2024
1011 2011      505 1994     265 1983     180 1960     15 1947      1 2021
998 2009       467 1993     265 1973     176 1959     14 2020      1 1937
938 2008       442 1991     257 1974     164 1964     13 1948      1 1935
931 2004       439 1992     232 1976     163 1962     11 1932      1 1927
904 2005       396 1990     232 1970     156 1963     7 1939       1 1924
850 2006       382 1988     224 2018     156 1958     4 1946
826 2003       374 1989     219 1975     127 1957     4 1938
811 2007       361 1987     216 1971     104 1956     4 1933
```

SELECT AND WHERE USING AWK

Return the id, name and year of titles which include “The Godfather” and “Part” words

```
[s@SS tsv]$ awk 'BEGIN{FS="\t"} $3 ~ /The Godfather.*Part/ { print $1,$3,$6}' title.basics.tsv
tt0071562 The Godfather: Part II 1974
tt0099674 The Godfather: Part III 1990
tt0539894 The Godfather: Part 3 1987
tt4667260 The Godfather Part IV 2015
tt5280194 Storyboards: The Godfather Part III 2001
tt5942862 The Godfather vs. The Godfather Part II 2008
tt6083390 The Godfather Part III/Kindergarten Cop/The Bonfire of the Vanities/The Russia House 1990
tt6620222 The Godfather: Part II 2017
```

JOIN WITH AWK

Match title.basics and title.crew (director/writer) through title id's, report title id, title, year and director id

```
[s@SS tsv]$ awk 'NR==FNR{a[$1]=$2OFS;next}{$10=a[$1];print $1,$3,$6,$10}' OFS='\t' title.crew.tsv title.basics.tsv | head  | column -t
tconst    primaryTitle  startYear  directors
tt0000001  Carmencita    1894       nm0005690
tt0000002  Le           ses        nm0721526
tt0000003  Pauvre       Pierrot    nm0721526
tt0000004  Un           Un         nm0721526
tt0000005  Blacksmith   Scene      nm0005690
tt0000006  Chinese      Chinese    nm0005690
tt0000007  Corbett      Before     nm0005690,nm0374658
tt0000008  Edison       of         nm0005690
tt0000009  Miss         Jerry      nm0085156
```

IN-MEMORY VS DISK-BASED

Our data is read from HDD

Now we create copies on SSD based and in-memory file systems and compare the performance of same query

In not-so-big datasets, the performances do not diverge significantly

```
[s@SS imdb]$ cp -r tsv ~
[s@SS imdb]$ cp -r tsv /dev/shm
[s@SS imdb]$ time awk 'NR==FNR{a[$1]=$2OFS;next}{$10=a[$1];print $1,$3,$6,$10}' OFS='\t' tsv/title.crew.tsv tsv/title.basics.tsv | head  | column -t
real    0m2.569s
user    0m2.284s
sys     0m0.292s
[s@SS imdb]$ time awk 'NR==FNR{a[$1]=$2OFS;next}{$10=a[$1];print $1,$3,$6,$10}' OFS='\t' /home/s/tsv/title.crew.tsv /home/s/tsv/title.basics.tsv |
head  | column -t
real    0m2.632s
user    0m2.367s
sys     0m0.274s
[s@SS imdb]$ time awk 'NR==FNR{a[$1]=$2OFS;next}{$10=a[$1];print $1,$3,$6,$10}' OFS='\t' /dev/shm/tsv/title.crew.tsv /dev/shm/tsv/title.basics.tsv
| head  | column -t
real    0m2.641s
user    0m2.352s
sys     0m0.297s
```

FROM DBMS TO RDBMS

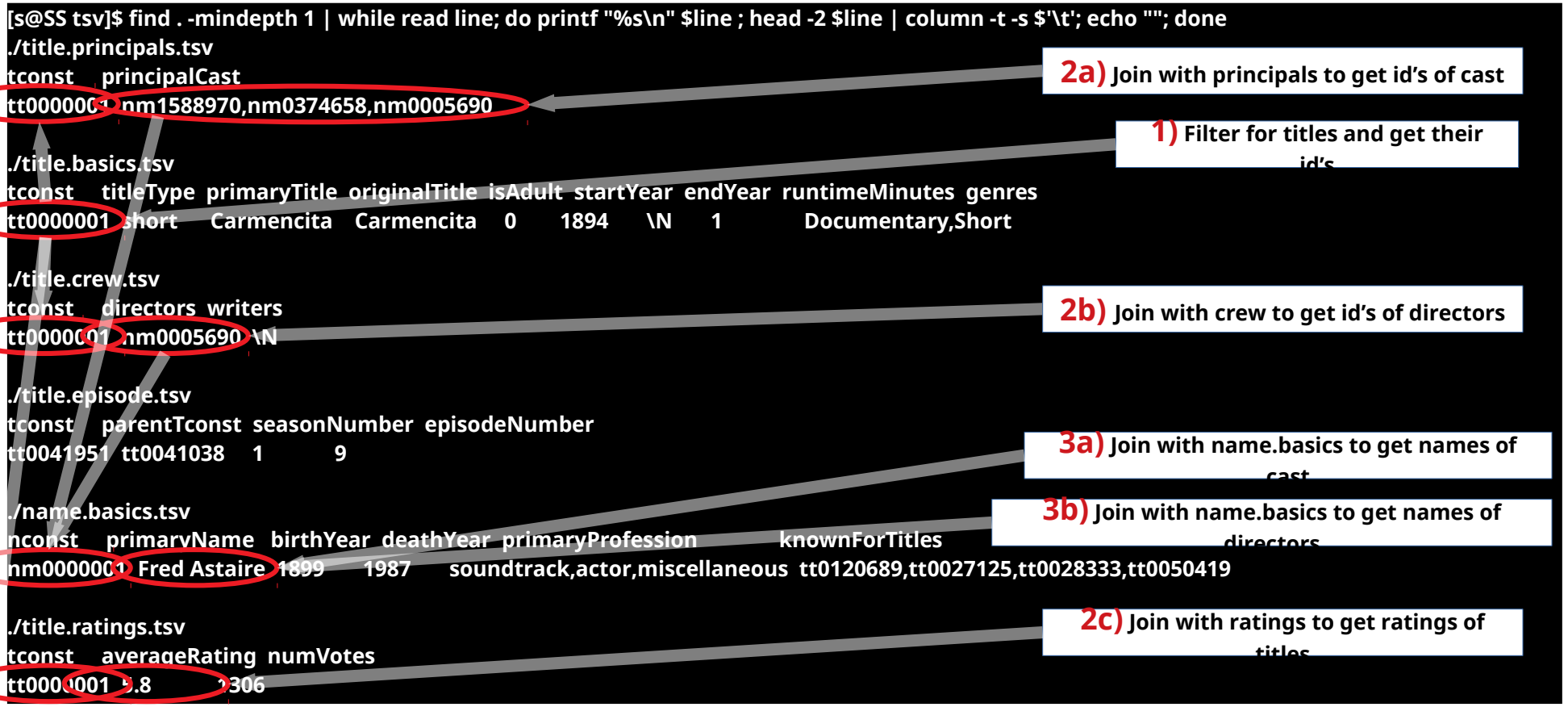
Although some of the query operations available through SQL can be done with standard UNIX tools, the commands may become cryptic and hard to comprehend to a non-coder

SQL simplifies queries with comprehendible commands; it resembles plain english

In order to explore the same data with PostgreSQL:

- Check whether data is suitable for PosgreSQL
- Take necessary actions

WHAT WE CAN DO WITH THE DATA



NULL VALUES

We know from our unique values check that the NULL value identifier in the dataset is \N

The SQL command for external data imports is “COPY”

The official documentation for COPY:

“NULL: Specifies the string that represents a null value. The default is **\N (backslash-N)** in text format, and an unquoted empty string in CSV format. You might prefer an empty string even in text format for cases where you don't want to distinguish nulls from empty strings. This option is not allowed when using binary format.”

So no need to substitute it

MELT PRINCIPAL CAST

In title.principals.tsv, the id's of all the principal cast of a title appear on a single field as comma separated values

In order to join this table with name.basics table later in PostgreSQL, we have to melt each cast into separated rows

We will process the file using an R script

WHY NOT DO THIS IN SQL INTEAD OF R?

SQL is a domain-specific language: The scope is limited to querying datasets and modifiying them in not-so-complicated ways

There are no looping and conditional branches in SQL, although some extensions to the language such as PL/SQL enable so

More complicated tasks (such as taking a field, splitting it across an identifier, replicating other field as much as the split values and appending them all together) require a "Turing complete" language (meaning it allows all computing tasks that a processor is able to do) such as R or Python

WHAT WE NEED IN ORDER TO MELT

Remember again the data structure of principal.cast:

```
[s@SS tsv]$ find . -mindepth 1 | while read line; do printf "%s\n" $line ; head -2 $line | column -t -s $'\t'; echo ""; done
./title.principals.tsv
tconst     principalCast
tt0000001  nm1588970,nm0374658,nm0005690
```

We have to:

For each row,

- Split the second field across commas
- Count the id’s
- Replicate first field as much as the count
- Combine the split second field and replicated first field into a 2 column matrix

And collect the matrix outputs of each row

EFFECTIVE AND EFFICIENT CODING

We may utilize R-base functions, data types and flow structures to write a “longer” low-level code, that takes more time to develop and may or may not perform better

Or we may follow the “R-way” to find a method faster to implement and faster to execute

Bad vs. good R code may have performance difference of up to 3 orders of magnitude (x1000)

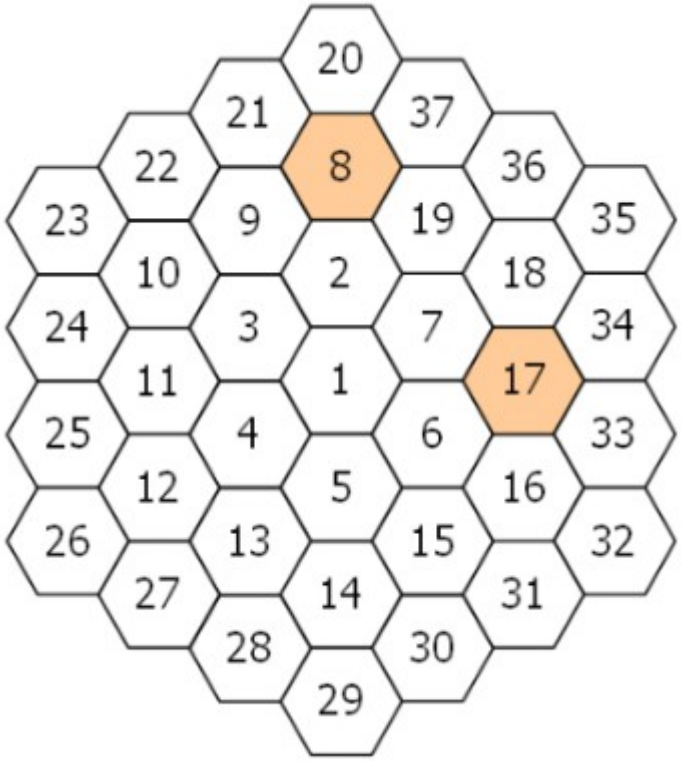
EXAMPLE OF GOOD R CODE: PE 128

Hexagonal tile differences

Problem 128

A hexagonal tile with number 1 is surrounded by a ring of six hexagonal tiles, starting at "12 o'clock" and numbering the tiles 2 to 7 in an anti-clockwise direction.

New rings are added in the same fashion, with the next rings being numbered 8 to 19, 20 to 37, 38 to 61, and so on. The diagram below shows the first three rings.



By finding the difference between tile n and each of its six neighbours we shall define PD(n) to be the number of those differences which are prime.

For example, working clockwise around tile 8 the differences are 12, 29, 11, 6, 1, and 13. So PD(8) = 3.

In the same way, the differences around tile 17 are 1, 17, 16, 1, 11, and 10, hence PD(17) = 2.

It can be shown that the maximum value of PD(n) is 3.

If all of the tiles for which PD(n) = 3 are listed in ascending order to form a sequence, the 10th tile would be 271.

Find the 2000th tile in this sequence.

PERFORMANCE BENCHMARKS ON FORUM

Sun, 5 Jul 2015, 01:31
hacatu
C/C++

It runs in 80 ms.

Sun, 16 Oct 2016, 12:45
st_2605
Python

Took less than 4 secs

Fri, 24 Jul 2015, 18:29
Arancaytar
Haskell

(30 seconds)

Wed, 26 Jul 2017, 22:15
merlinnimue
Java

Runtime: 200 ms

Sat, 20 Aug 2016, 06:16
Efemena
Rust

63ms on laptop

Fri, 8 Sep 2017, 18:00
serhatcevikel
R

Performance is 32.35 ms

EFFECTIVE AND EFFICIENT CODE

When the size of data gets much bigger, performance consequences of any action on the data gets more important

1000x difference in performance:

In small data: 1mcs vs 1 ms

In medium size data: 1 ms vs 1 sec

In large data: 1 sec vs 1000 sec !

So writing effective (faster to develop, does much with less) and efficient code (faster to execude) is critical

CHECK THE CODE CORPUS

Probably the task that you'll perform has been already done by someone else, the solution probably posted on the web

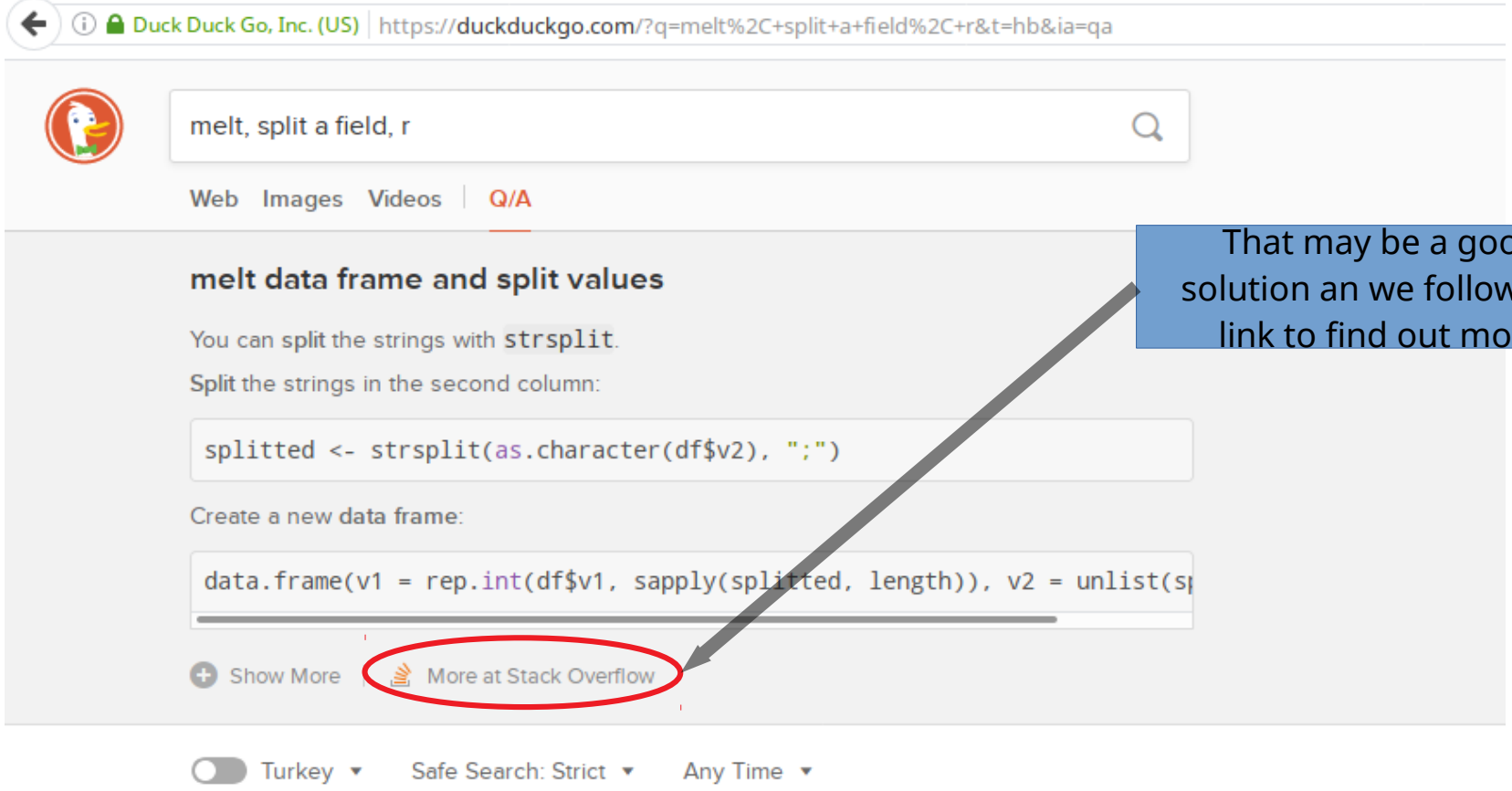
So before starting your own solution, check the web: Don't rediscover America!

We will ask Uncle Duck (http://ggg.dd) if there is such a way. Uncle Google always peeps us when we try to talk to him and this is disturbing!

The key to success is to find the right keywords to ask!

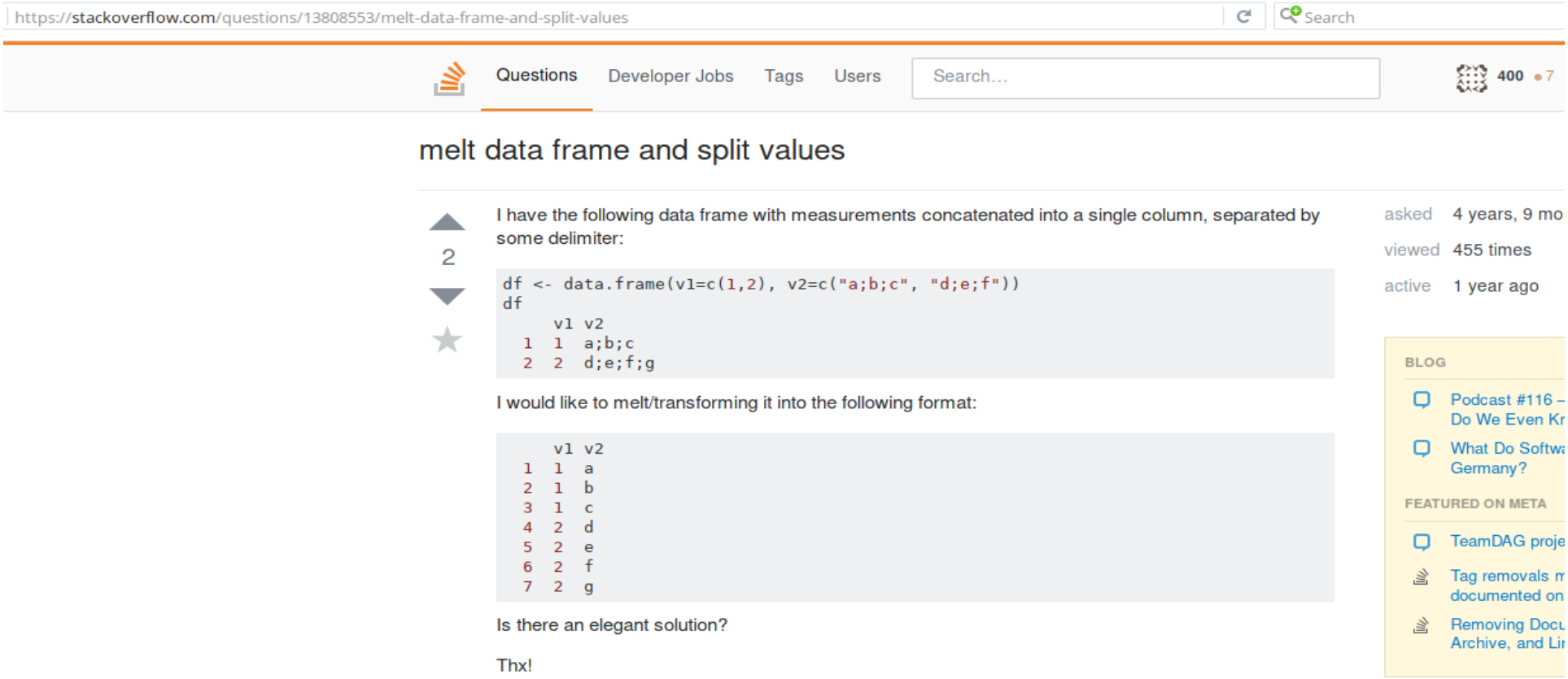
ASK UNCLE DUCK

The keywords we will use are: melt, split a field, r



STACKOVERFLOW SOLUTION W/ DF

The question is similar to ours:


https://stackoverflow.com/questions/13808553/melt-data-frame-and-split-values

Search

Questions Developer Jobs Tags Users Search... 400 • 7

melt data frame and split values

2

I have the following data frame with measurements concatenated into a single column, separated by some delimiter:

```
df <- data.frame(v1=c(1,2), v2=c("a;b;c", "d;e;f"))
df
     v1 v2
  1  1  a;b;c
  2  2  d;e;f;g
```

I would like to melt/transforming it into the following format:

```
    v1 v2
  1  1  a
  2  1  b
  3  1  c
  4  2  d
  5  2  e
  6  2  f
  7  2  g
```

Is there an elegant solution?

Thx!

asked 4 years, 9 mo

viewed 455 times

active 1 year ago

BLOG

Podcast #116 – Do We Even Kr

What Do Softwa Germany?

FEATURED ON META

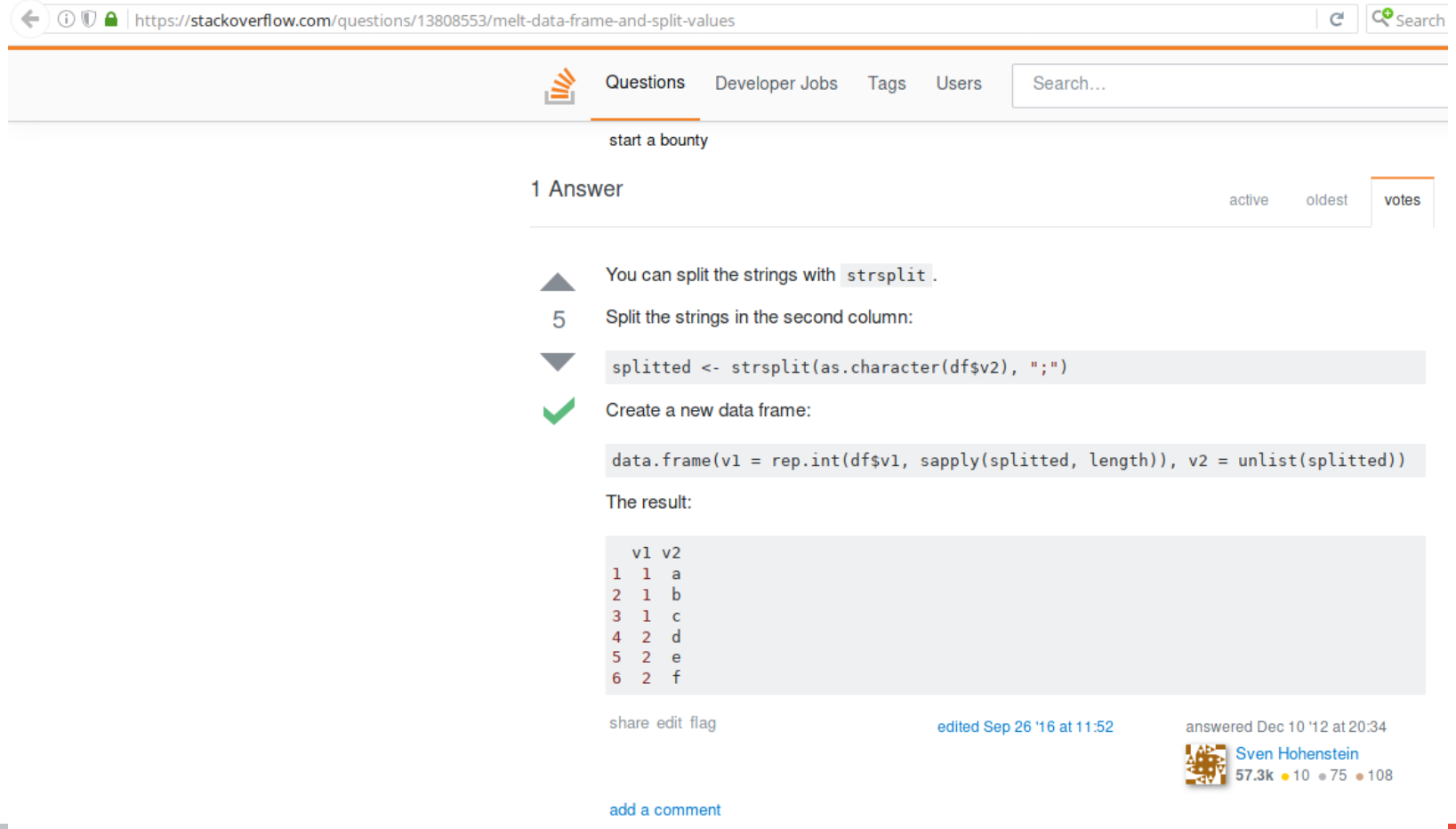
TeamDAG proje

Tag removals m documented on

Removing Docu Archive, and Li

STACKOVERFLOW SOLUTION W/ DF

And the solution is just a two-liner:



https://stackoverflow.com/questions/13808553/melt-data-frame-and-split-values

Search

Questions Developer Jobs Tags Users Search...

start a bounty

1 Answer

active oldest votes

5

You can split the strings with `strsplit`.

Split the strings in the second column:

```
splitted <- strsplit(as.character(df$v2), ";")
```

Create a new data frame:

```
data.frame(v1 = rep.int(df$v1, sapply(splitted, length)), v2 = unlist(splitted))
```

The result:

```
  v1 v2
1  1  a
2  1  b
3  1  c
4  2  d
5  2  e
6  2  f
```

share edit flag

edited Sep 26 '16 at 11:52

answered Dec 10 '12 at 20:34

Sven Hohenstein

57.3k ● 10 ● 75 ● 108

add a comment

DATA.FRAME VS. DATA.TABLE

Main idea is to apply the split function and replicate the data by length (just as we planned)

However the main data structure utilized is data.frame (DF) which is notorious for low performance in large data sets

A newer data structure which is based on data.frame but implemented as an extension to R is data.table

“data.table” is fast even in large data sets, can perform many actions in a concise way and is expected by the R community (including me, SÇ) to replace data.frame in recent future

DATA.TABLE ON CRAN

data.table: Extension of 'data.frame'

Fast aggregation of large data (e.g. 100GB in RAM), fast ordered joins, fast add/modify/delete of columns by group using no copies at all, list columns, a fast friendly file reader and parallel file writer. Offers a natural and flexible syntax, for faster development.

Version:	1.10.4
Depends:	R (≥ 3.0.0)
Imports:	methods
Suggests:	bit64, knitr, nanotime, chron, ggplot2 (≥ 0.9.0), plyr, reshape, reshape2, testthat (≥ 0.4), hexbin, fastmatch, nlme, xts, gdata, GenomicRanges, caret, curl, zoo, plm, rmarkdown, parallel
Published:	2017-02-01
Author:	Matt Dowle [aut, cre], Arun Srinivasan [aut], Jan Gorecki [ctb], Tom Short [ctb], Steve Lianoglou [ctb], Eduard Antonyan [ctb]
Maintainer:	Matt Dowle <mattjdowle at gmail.com>
BugReports:	https://github.com/Rdatatable/data.table/issues
License:	GPL-3 | file LICENSE
URL:	http://r-datatable.com
NeedsCompilation:	yes
Materials:	README NEWS
In views:	Finance, HighPerformanceComputing
CRAN checks:	data.table results

Keep that name in your mind!

Downloads:

Reference manual:	data.table.pdf
Vignettes:	Frequently asked questions Introduction to data.table Keys and fast binary search based subset Reference semantics Efficient reshaping using data.tables Secondary indices and auto indexing
Package source:	data.table_1.10.4.tar.gz
Windows binaries:	r-devel: data.table_1.10.4.zip, r-release: data.table_1.10.4.zip, r-oldrel: data.table_1.10.4.zip
OS X El Capitan binaries:	r-release: data.table_1.10.4.tgz
OS X Mavericks binaries:	r-oldrel: data.table_1.10.4.tgz
Old sources:	data.table archive

DATA.TABLE PROS AND CONS

PROS:

Fast aggregation of large data (e.g. 100GB in RAM),

fast ordered joins,

fast add/modify/delete of columns by group using no copies at all, list columns,

a fast friendly file reader and parallel file writer.

Offers a natural and flexible syntax, for faster development.

CONS:

Learning curve for syntax and features

SEARCH DATA.TABLE CORPUS

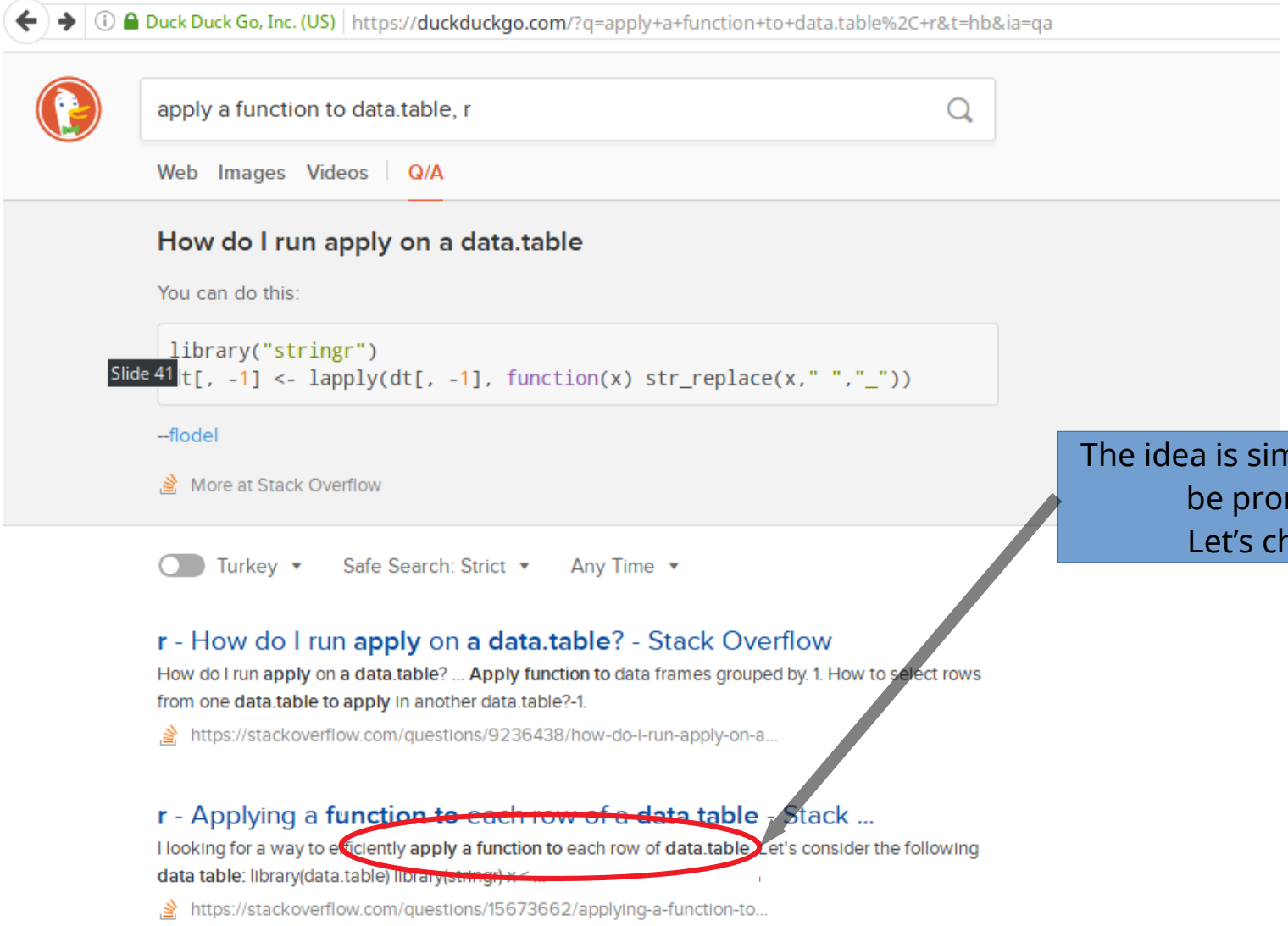
Now, we will modify our search keywords, so that we may find a similar solution with much more efficient data.table structure

We know the key functions are “split” to split id’s by “,” and “rep” for replication of data

Power of data.table comes from the fact that, .ply functions can be run inside the data.table with a significant performance gain

So we should search for how to “apply” a function to a DT

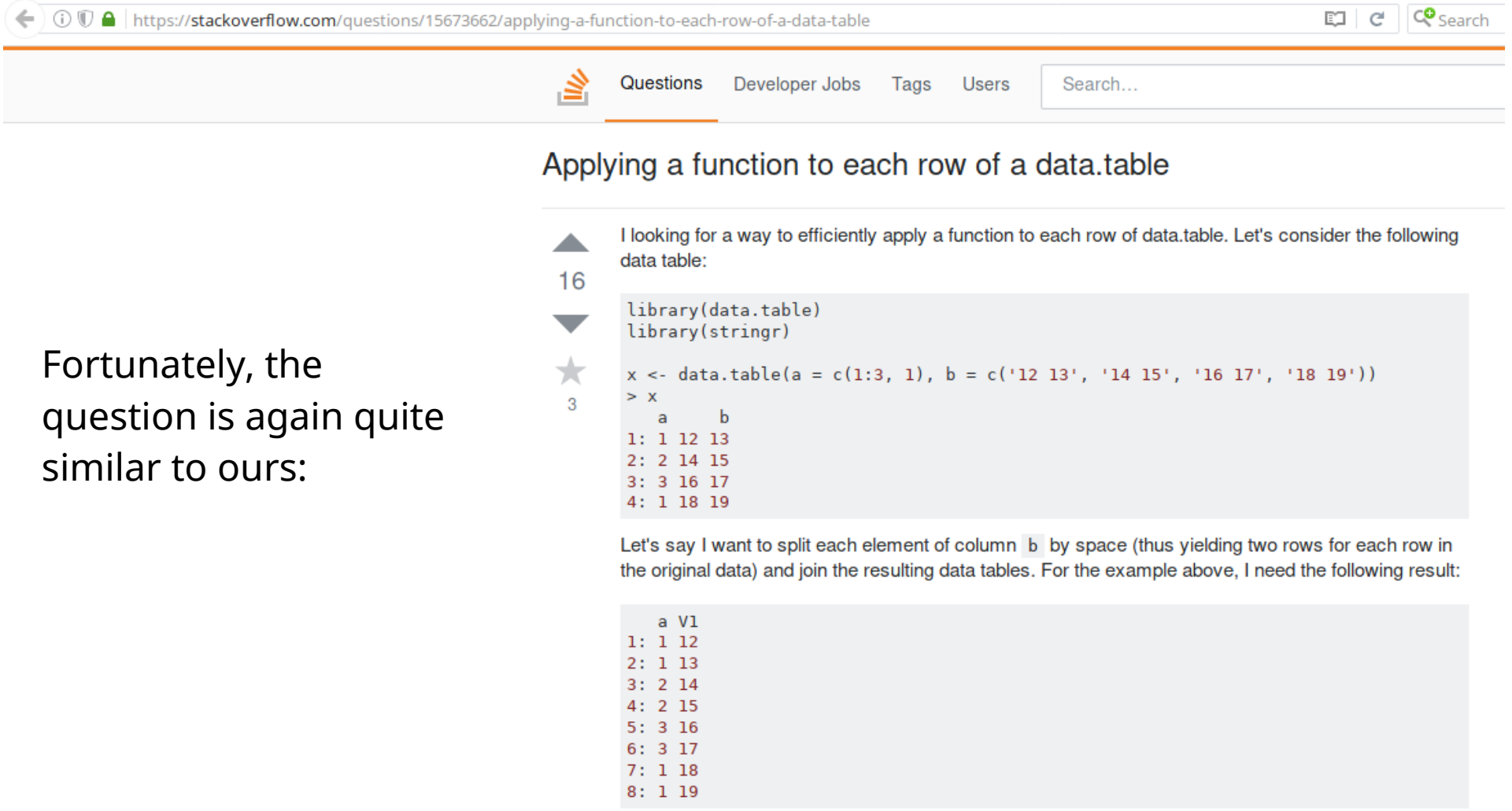
“apply a function to data.table, r”



The idea is similar so it may be promising
Let’s check it!

STACKOVERFLOW SOLUTION W/ DT

Fortunately, the question is again quite similar to ours:



https://stackoverflow.com/questions/15673662/applying-a-function-to-each-row-of-a-data-table

Search

Questions Developer Jobs Tags Users Search...

Applying a function to each row of a data.table

16

3

I looking for a way to efficiently apply a function to each row of data.table. Let's consider the following data table:

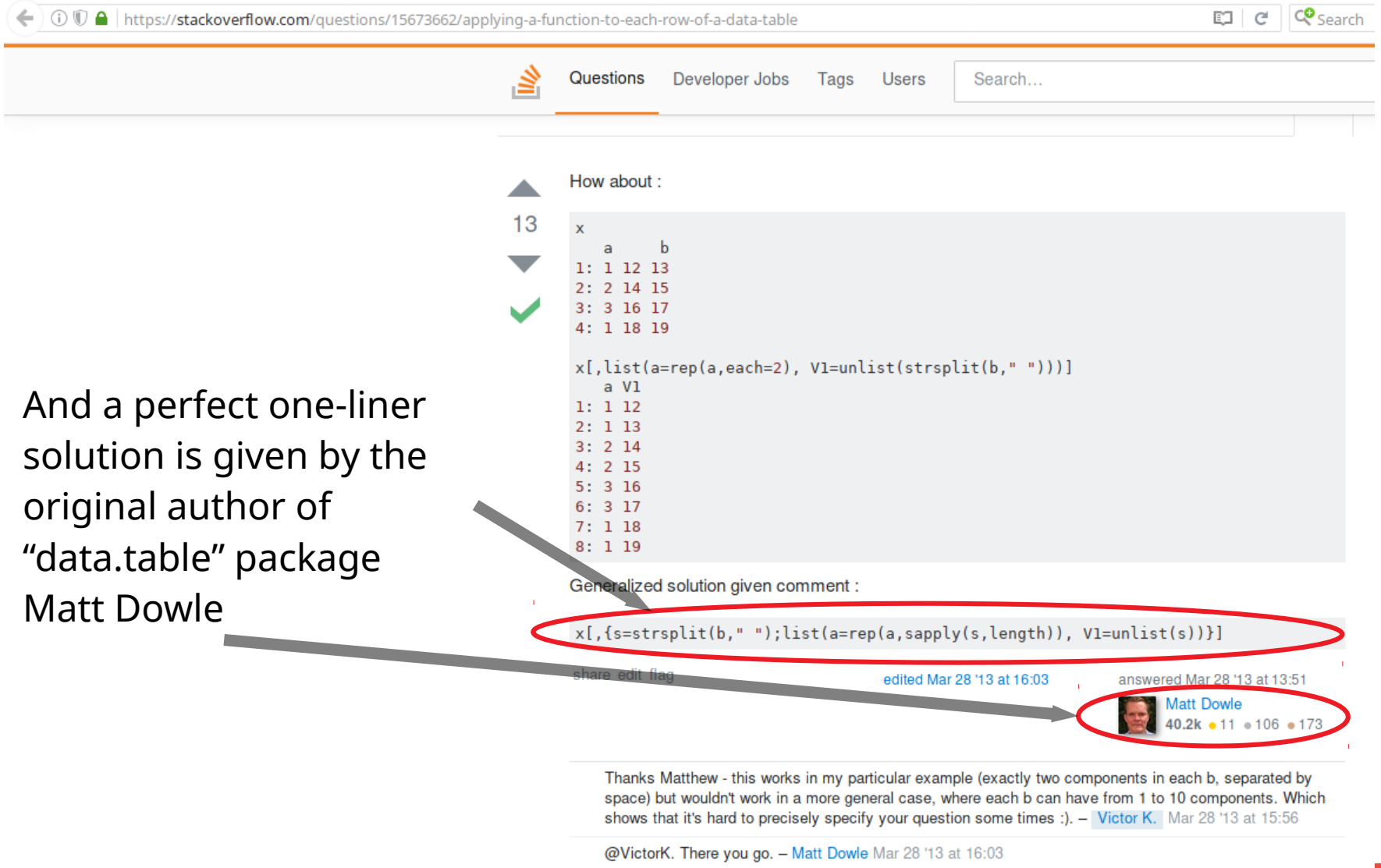
```
library(data.table)
library(stringr)

x <- data.table(a = c(1:3, 1), b = c('12 13', '14 15', '16 17', '18 19'))
> x
   a     b
1: 1 12 13
2: 2 14 15
3: 3 16 17
4: 1 18 19
```

Let's say I want to split each element of column `b` by space (thus yielding two rows for each row in the original data) and join the resulting data tables. For the example above, I need the following result:

```
   a V1
1: 1 12
2: 1 13
3: 2 14
4: 2 15
5: 3 16
6: 3 17
7: 1 18
8: 1 19
```

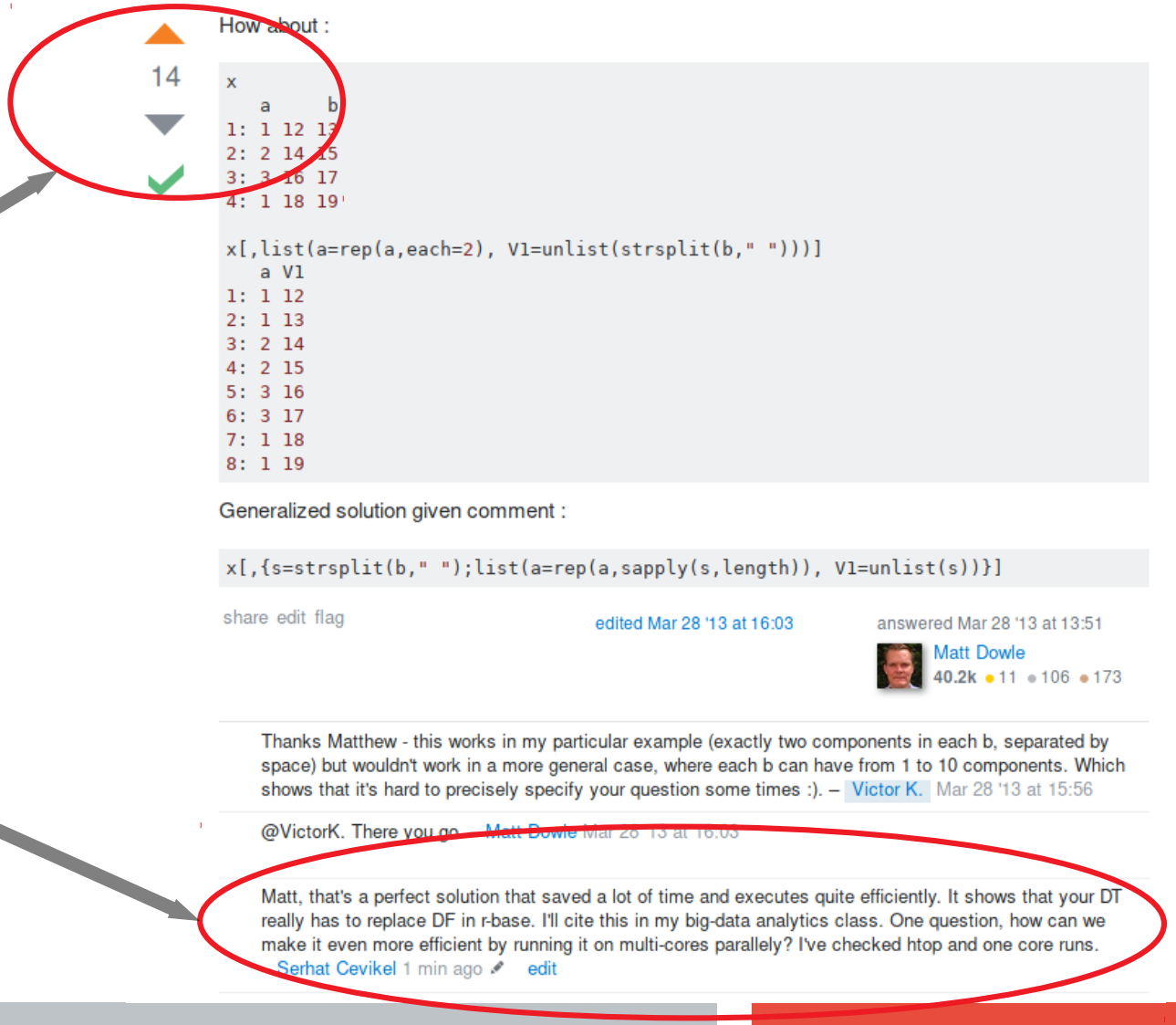
STACKOVERFLOW SOLUTION W/ DT



And a perfect one-liner solution is given by the original author of “data.table” package Matt Dowle

TWO THUMBS UP TO MATT DOWLE

He deserves a big kudo and a very positive comment
(You can do the same if you find time)



14

How about :

```
x
   a      b
1: 1 12 13
2: 2 14 15
3: 3 16 17
4: 1 18 19

x[,list(a=rep(a,each=2), V1=unlist(strsplit(b," "))),]
   a V1
1: 1 12
2: 1 13
3: 2 14
4: 2 15
5: 3 16
6: 3 17
7: 1 18
8: 1 19
```

Generalized solution given comment :

```
x[,{s=strsplit(b," ");list(a=rep(a,sapply(s,length)), V1=unlist(s))},]
```

share edit flag

edited Mar 28 '13 at 16:03

answered Mar 28 '13 at 13:51

Matt Dowle
40.2k ● 11 ● 106 ● 173

Thanks Matthew - this works in my particular example (exactly two components in each b, separated by space) but wouldn't work in a more general case, where each b can have from 1 to 10 components. Which shows that it's hard to precisely specify your question some times :). – Victor K. Mar 28 '13 at 15:56

@VictorK. There you go. – Matt Dowle Mar 28 '13 at 16:03

Matt, that's a perfect solution that saved a lot of time and executes quite efficiently. It shows that your DT really has to replace DF in r-base. I'll cite this in my big-data analytics class. One question, how can we make it even more efficient by running it on multi-cores parallely? I've checked htop and one core runs. – Serhat Cevikel 1 min ago edit

NOW OUR CODE

We will use the idea from the generalized solution from Matt's SO answer:

```
x[,{s=strsplit(b," ");list(a=rep(a,sapply(s,length)),
V1=unlist(s))}]
```

Tailor that line for our data set

And add some code lines for file paths and I/O

CODE ON GITHUB

Branch: **master** ▾ **mef-bigdata** / R / **melt_principals.R**

serhatcevikel commit

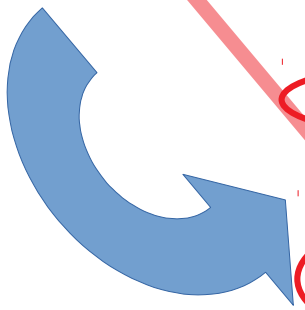
1 contributor

22 lines (14 sloc) | 1.03 KB

```
1  # Melt the table by splitting the second field across commas
2  # Ideas from:
3  # https://stackoverflow.com/questions/13808553/melt-data-frame-and-split-values
4  # https://stackoverflow.com/questions/15673662/applying-a-function-to-each-row-of-a-data-table
5
6  library(data.table) # load data.table package
7
8  prefix <- Sys.getenv("fls") # get the environment variable - a path prefix - "fls"
9  in_path <- sprintf("%s/data/imdb/tsv/title.principals.tsv", prefix) # create the path to input file
10
11 principals <- fread(in_path) # fast read input file into DT
12
13 # split 2nd field, replicate 1st field length of split times
14 principals_molten <- principals[, {principalCast_list = strsplit(principalCast, ",");
15                                    list(tconst = rep(tconst, sapply(principalCast_list, length)),
16                                         principalCast = unlist(principalCast_list))}]
17
18
19 out_path <- sprintf("%s/data/imdb/tsv/title.principals2.tsv", prefix) # create output file path
20
21 fwrite(principals_molten, file = out_path, sep = "\t") # fast write new DT as tsv
```

FROM INPUT TO OUTPUT

```
> dim(principals)
[1] 4008568       2
> head(principals)
     tconst                  principalCast
1: tt0000001         nm1588970,nm0374658,nm0005690
2: tt0000002                nm1335271,nm0721526
3: tt0000003 nm5442194,nm0721526,nm1335271,nm5442200
4: tt0000004                nm0721526,nm1335271
5: tt0000005         nm0443482,nm0653042,nm0005690
6: tt0000006                      nm0005690
```



```
> dim(principals_molten)
[1] 25358895        2
> head(principals_molten)
     tconst principalCast
1: tt0000001    nm1588970
2: tt0000001    nm0374658
3: tt0000001    nm0005690
4: tt0000002    nm1335271
5: tt0000002    nm0721526
6: tt0000003    nm5442194
```

TOOLKIT OF A DATA ANALYST

The code executed in a matter of seconds

The caveat: In order to deal with bigger data, we have to be familiar with light and efficient tools and methods (mostly in the UNIX/Linux environment)

Conventional tool will be choked! (Try opening the files with usual MS tools such as Excel, Access or Notepad)

And we should first refer to the large corpus of solutions on the web before we start to write our own one!

NOW POSTGRESQL ...

Now we can import the data into PostgreSQL

SHORT INTRO TO POSTGRESQL

PostgreSQL follows a server-client model

Even if we work on a single localhost, a server should be initiated an we should connect to that server using a client tool (e.g. psql as cli, or pgadmin3 as gui)

Until initialization of the server, we may enter commands from shell but afterwards for ease of use we will prefer pgadmin3 gui to psql cli (from time to time, we may visit psql)

HOWEVER, ALL ACTIONS REGARDING DATABASES WILL BE EXECUTED THROUGH SQL COMMANDS, NOT GUI CLICKS

WHY?

SQL COMMANDS VS. GUI CLICKS

We prefer SQL commands to GUI (graphical user interface) clicks because it enables:

- Easier reproducablity of all actions (scripting, etc)
- More control over options, arguments
- Sharability with collaborators

STEPS TO INITIALIZE SERVER AND CLIENT

The shell commands will refer to *nix systems:

Install postgresql and pgadmin3

postgres “PC” user (not DB user) is created by default. The user is not a sudoer (like an admin) and need not be

It is recommended to create a password for postgres “PC” user (sudo passwd postgres)

Shift to postgres user (sudo -i -u postgres)

Initialize database (initdb --locale $LANG -E UTF8 -D '/var/lib/postgres/data')

Exit postgres user (exit)

Start (and/or enable to persist after rebopt) postgresql service (sudo systemctl start postgresql.service)

Check whether server listens at default port 5432

```
[s@SS ~]$ sudo netstat -tulpn | grep 5432
[sudo] password for s:
tcp        0      0 127.0.0.1:5432          0.0.0.0:*               LISTEN      4440/postgres
tcp6       0      0 ::1:5432                :::*                    LISTEN      4440/postgres
```

STEPS TO INITIALIZE SERVER AND CLIENT

Shift to postgres user again (sudo -i -u postgres)

Create a “DB” user named “postgres” (createuser –interactive).

Exit postgres “PC user” or not

Create a new database as a DB user (createdb deneme1 -U postgres)

Connect to that new database from psql and as “postgres” user (psql -d deneme1)

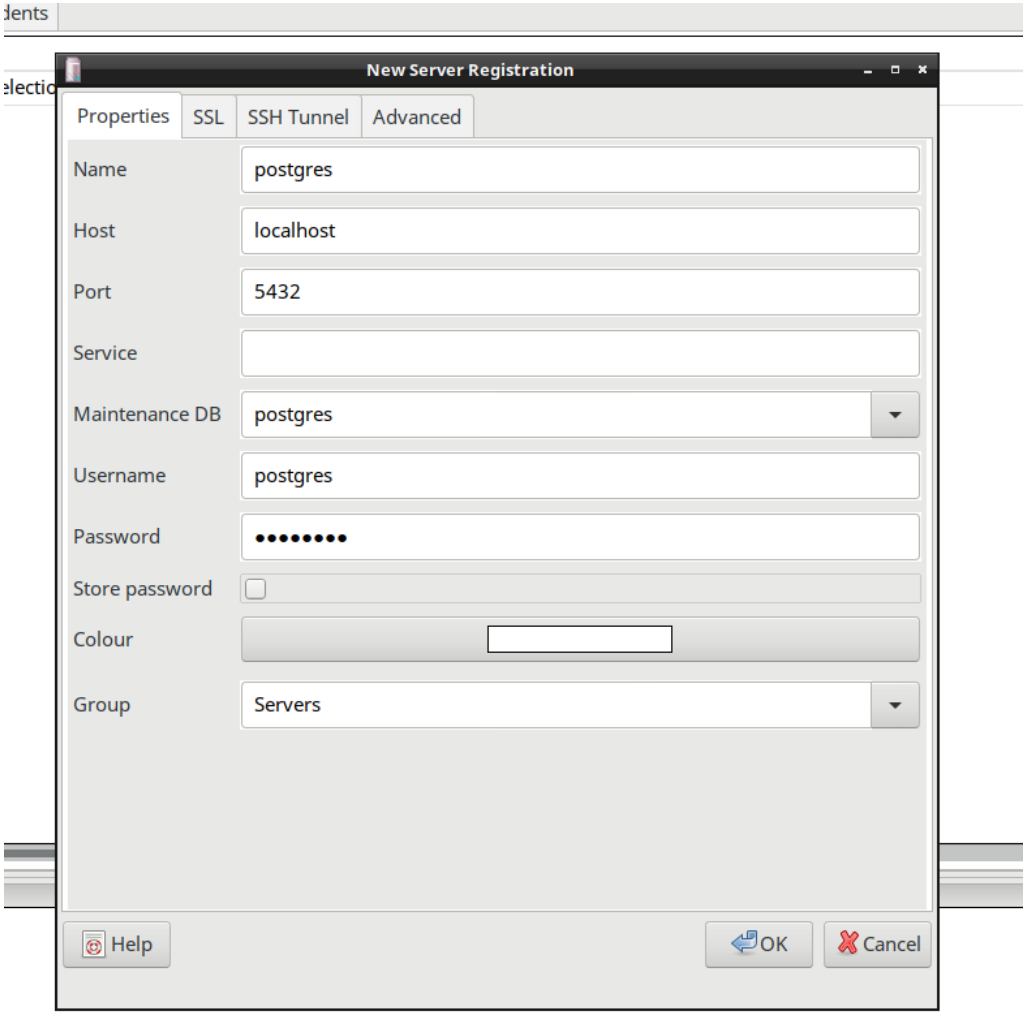
```
[postgres@SS ~]$ psql -d deneme1
psql (9.6.5)
Type "help" for help.

deneme1=#
```

NOW PGADMIN3

The only GUI action we will make from pgadmin3 will be to create a connection to our server from menu

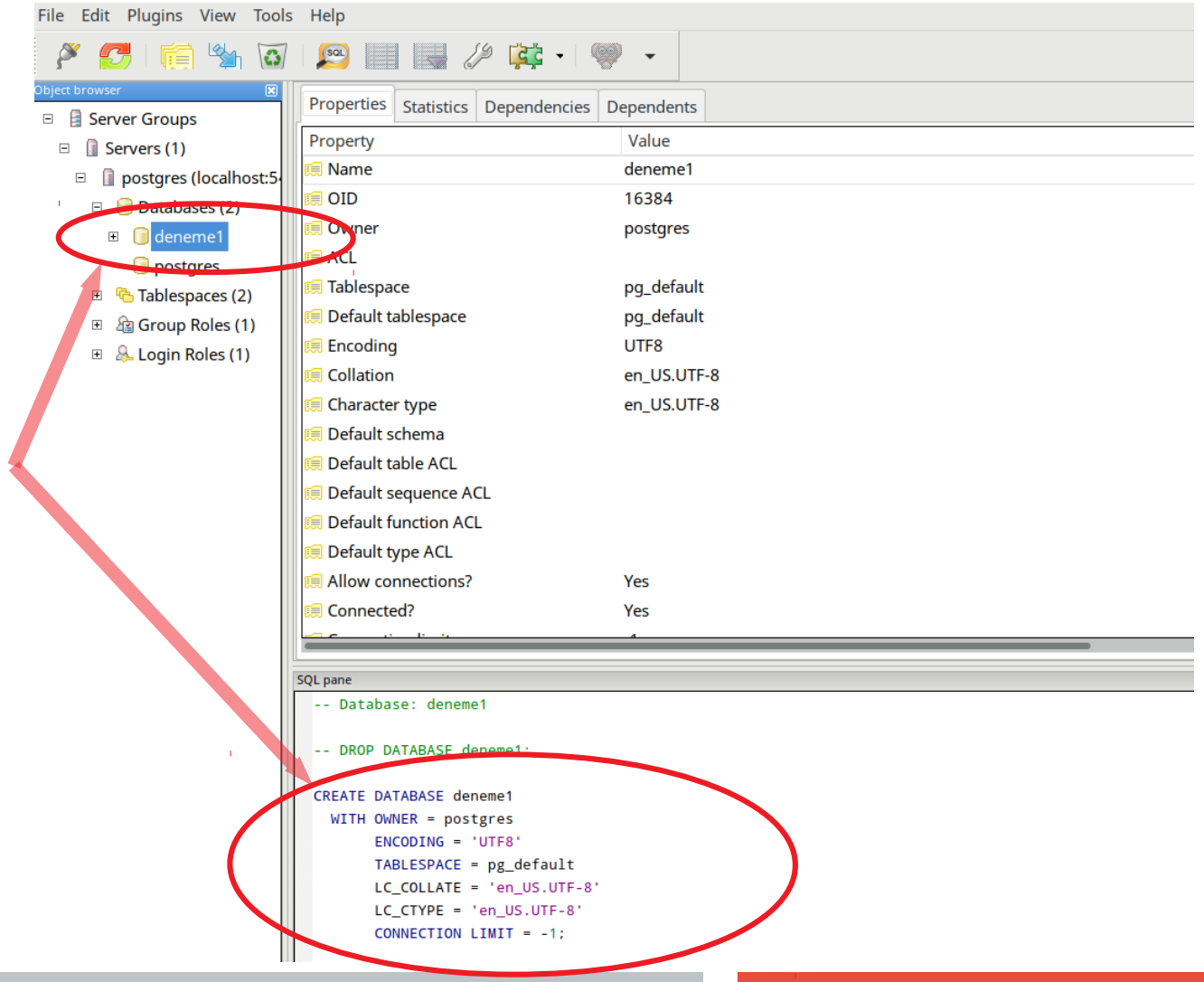
File > Add Server



PGADMIN3 CONNECTS …

Although we created database “deneme1” from shell, we can view the database from pgadmin3 gui front-end

And note that the equivalent SQL command to create the database as is is shown below

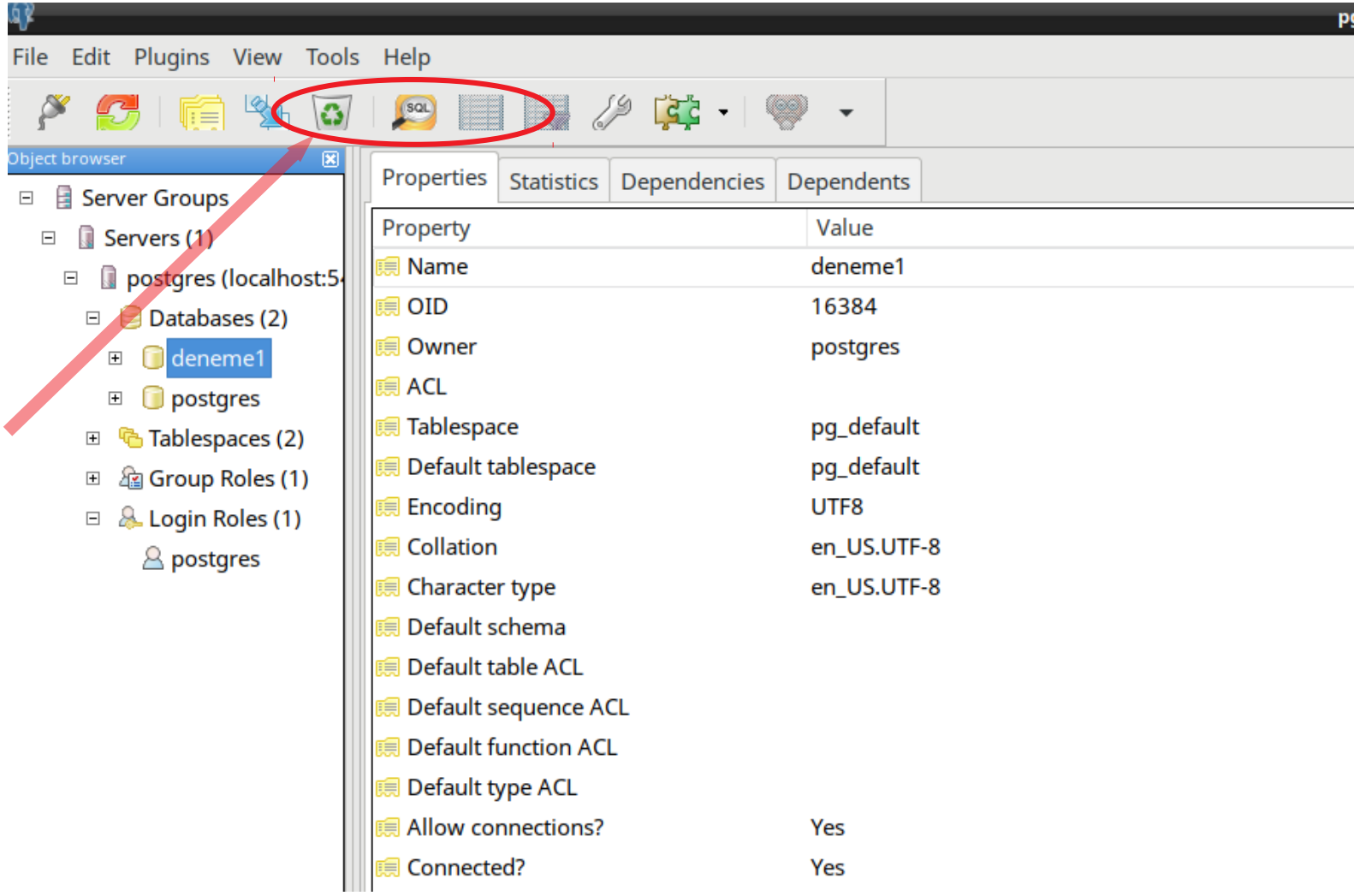


EXECUTE SQL QUERIES

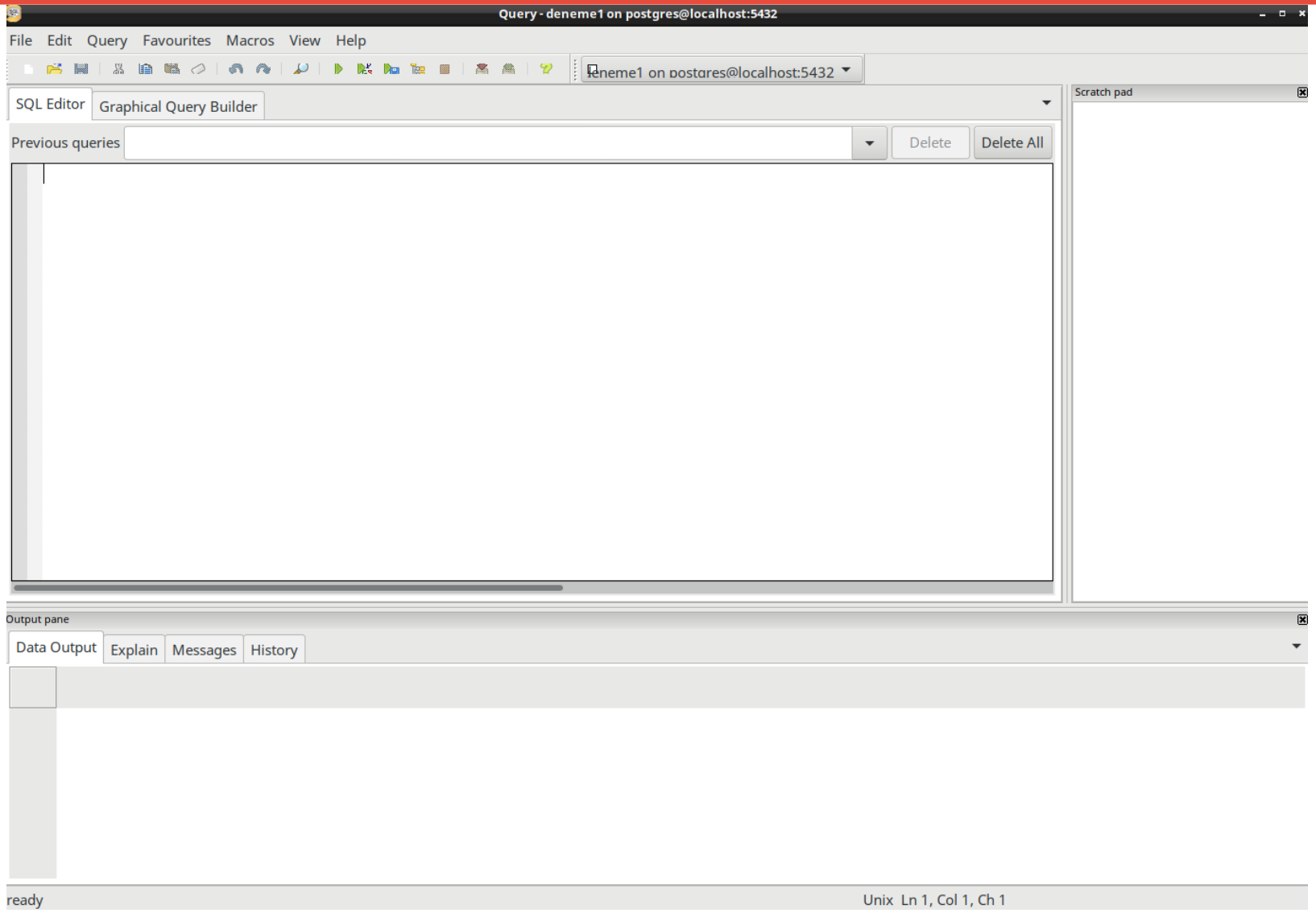
While a database is activated (highlighted on the left pane),

Click on

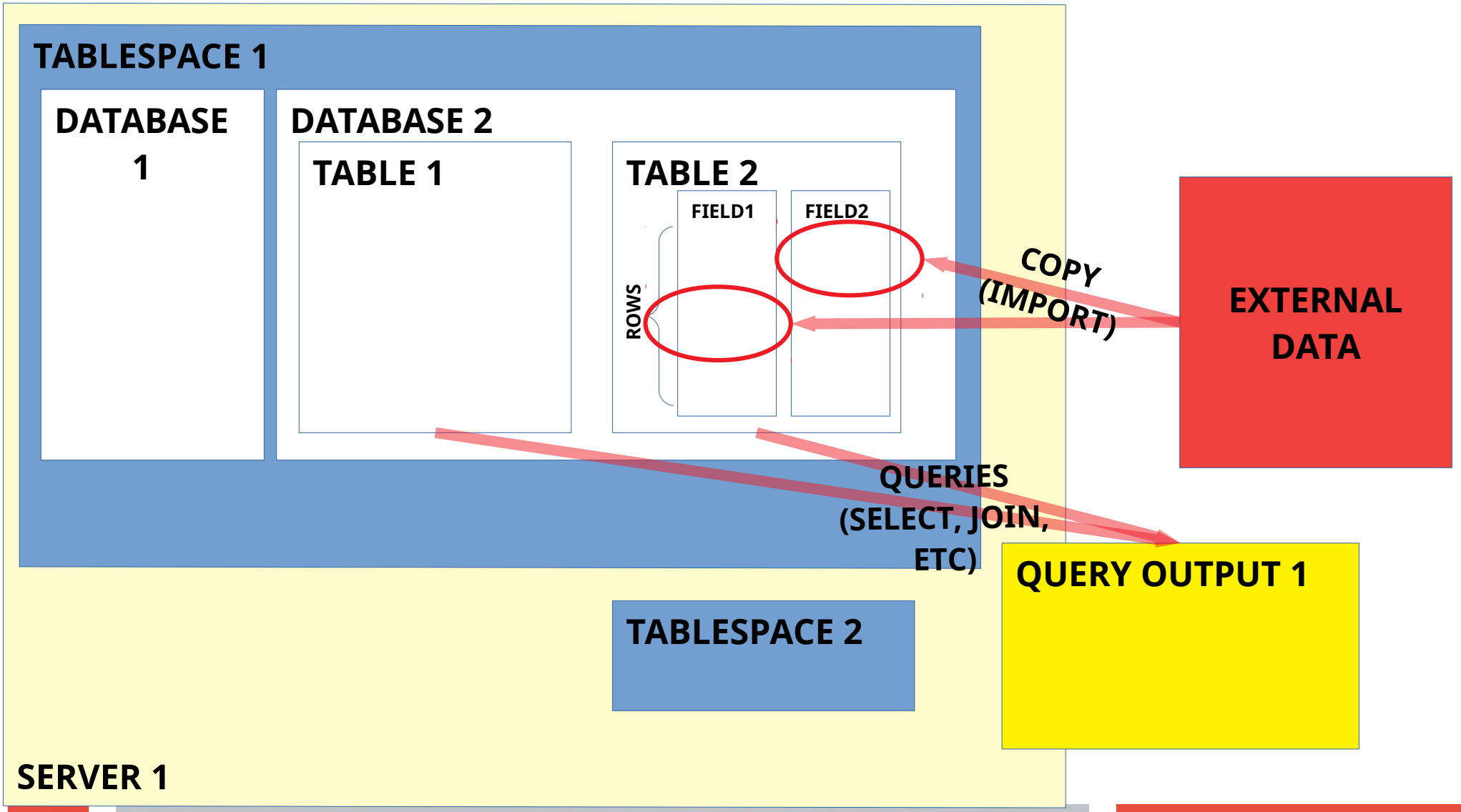
to open SQL query ediyor



SQL QUERY EDITOR



THE STRUCTURE OF OUR DB OBJECTS



THE FIRST SQL COMMAND ON PGADMIN3

Tablespaces: defined locations in the file system where the files representing database objects can be stored

Once created, a tablespace can be referred to by name when creating database objects.

Sortly, it is a path on the file system where any database is stored

There can be multiple tablespaces defined and a database can be chosen to reside in any of them

CREATE A NEW TABLESPACE

Why create a new tablespace: The default tablespace may reside on SSD, a much more scarce resource than HDD

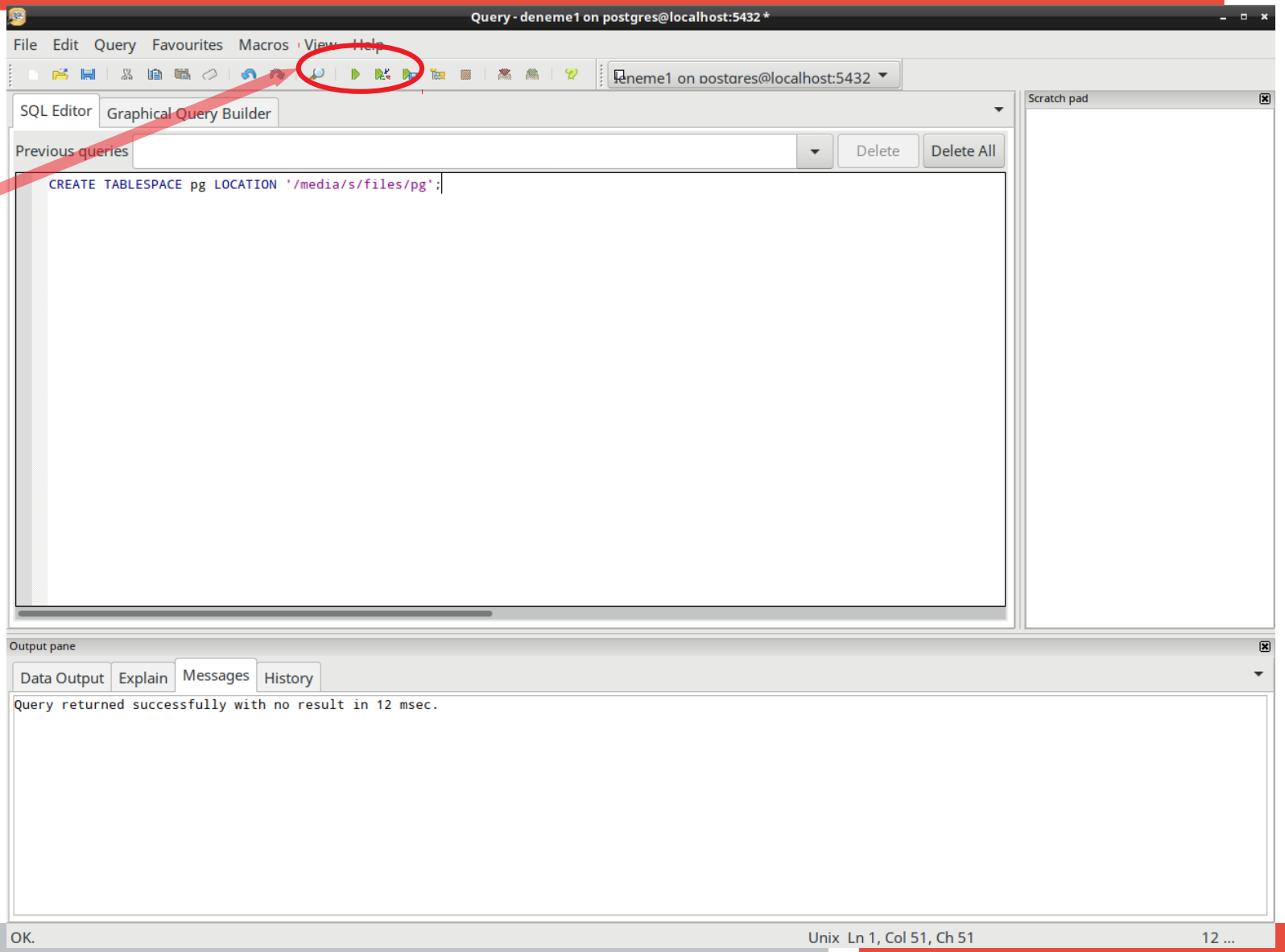
In order to hold a larger dataset, we may want to create a separate tablespace on the HDD

We should first create the directory for the tablespace

```
CREATE TABLESPACE pg LOCATION '/media/s/files/pg';
```

CREATE A NEW TABLESPACE

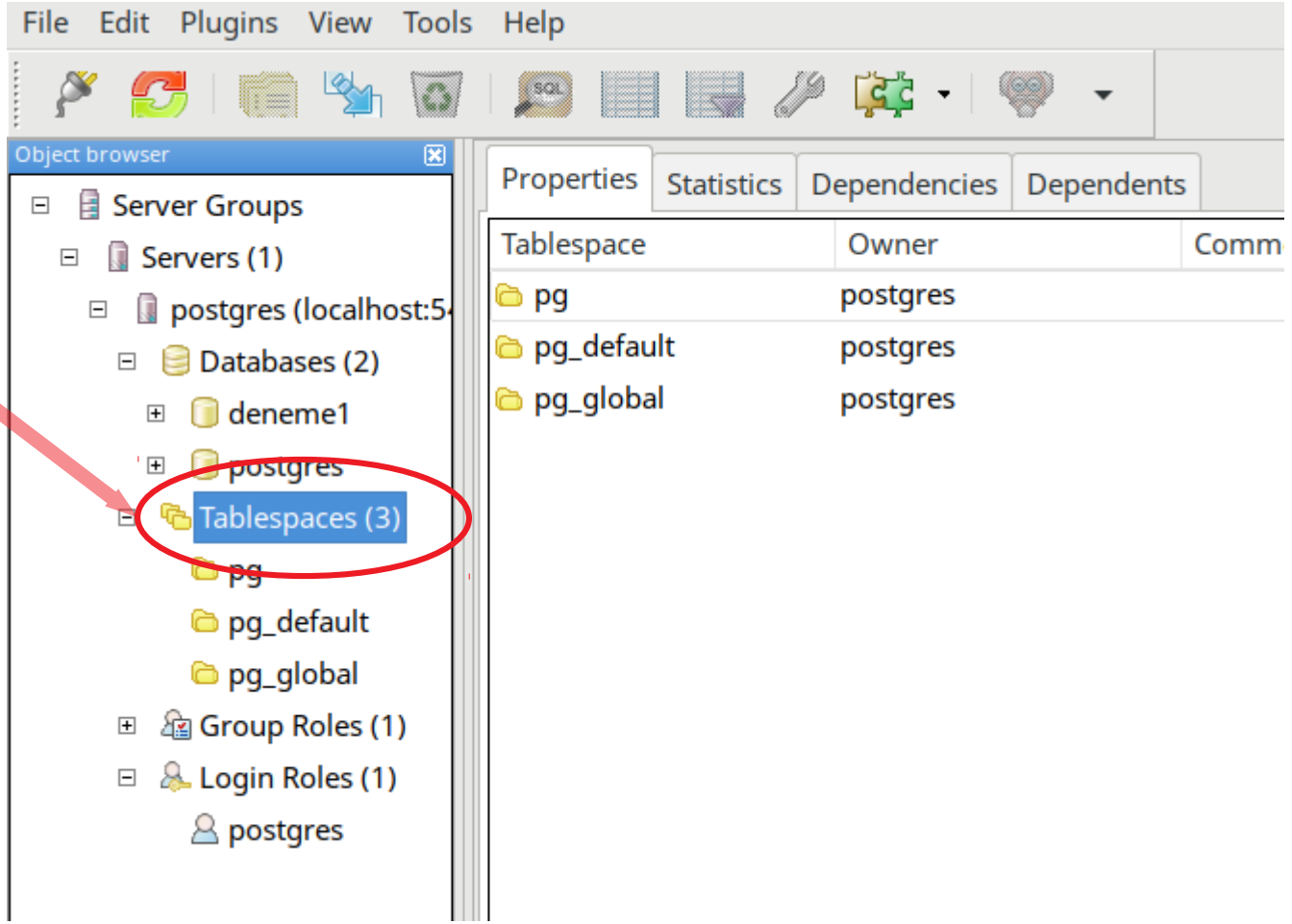
HIT F5 KEY OR HERE TO EXECUTE



CREATE A NEW TABLESPACE

See, now object browser shows the new tablespace “pg”

You should refresh objects to see the changes in the object browser



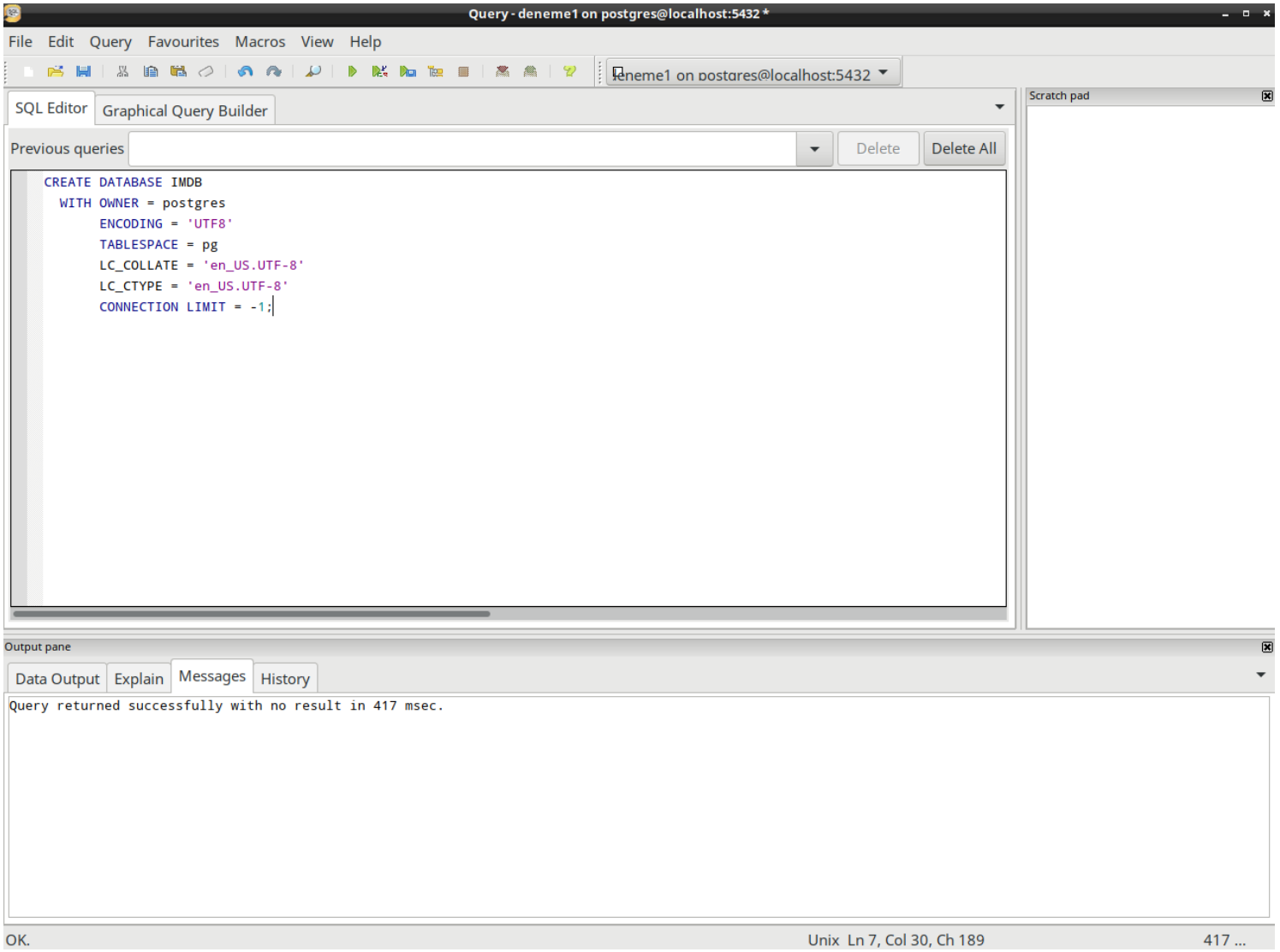
CREATE A NEW DATABASE

Now, we will be creating a new database named “IMDB” in tablespace “pg”

Note that, we can recycle the automatic SQL statement from deneme1

```
CREATE DATABASE IMDB
  WITH OWNER = postgres
      ENCODING = 'UTF8'
      TABLESPACE = pg
      LC_COLLATE = 'en_US.UTF-8'
      LC_CTYPE = 'en_US.UTF-8'
      CONNECTION LIMIT = -1;
```

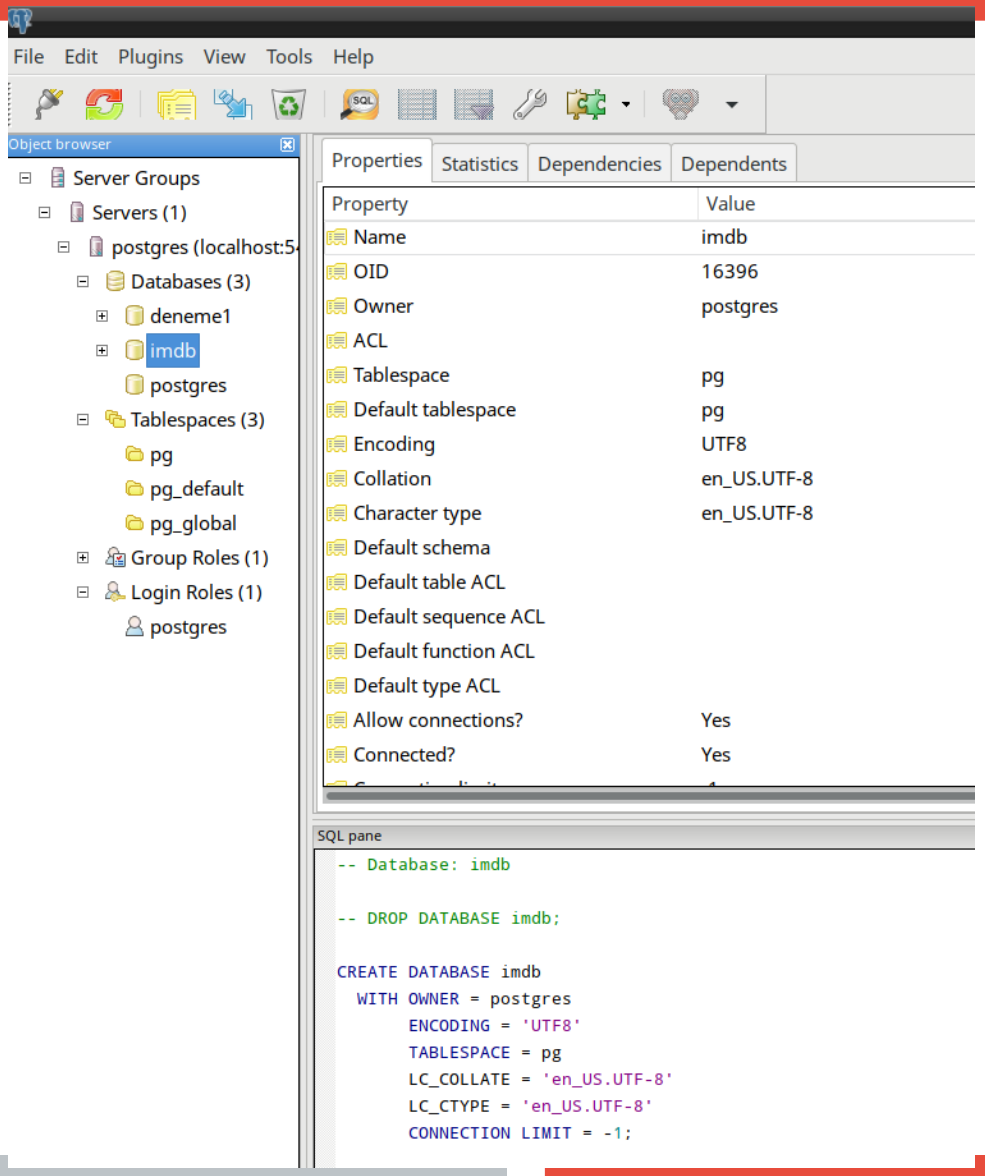
CREATE A NEW DATABASE



CREATE A NEW DATABASE

Now we have our new database called “imdb”

Note that db name is coerced to lower case characters



CREATE A NEW TABLE

Now we create a new empty table so that we can import data from title.basics

Note that the data types must match data of the imdb definition

And the table name should not have a “dot” inside

And you should open the SQL query editor while “imdb” database is highlighted on the object browser

TYPES OF TABLE AN EXTERNAL DATA MUST MATCH!!!

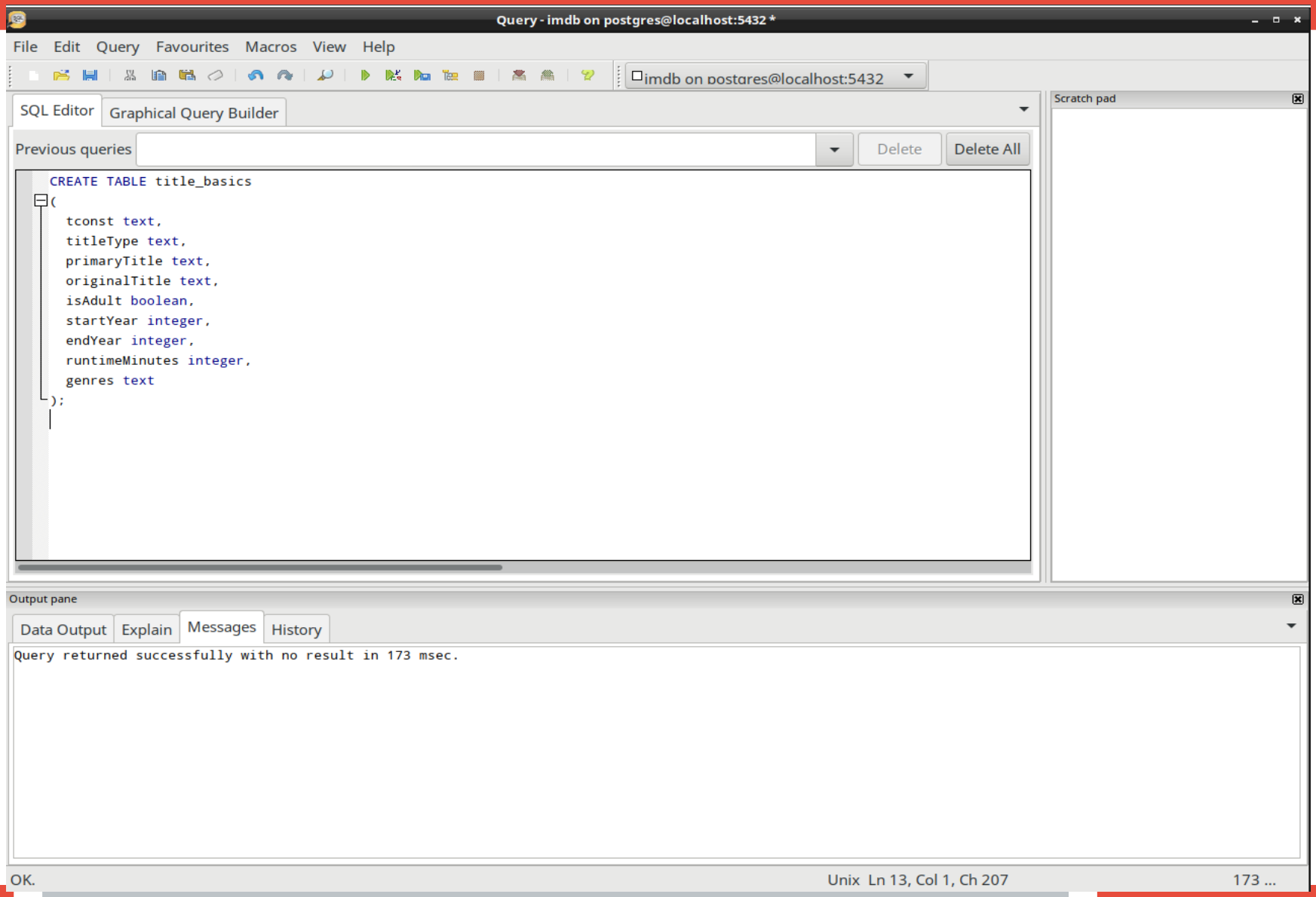
CREATE A NEW TABLE

title.basics.tsv.gz - Contains the following information for titles:

- **tconst (string) - alphanumeric unique identifier of the title**
- **titleType (string) – the type/format of the title (e.g. movie, short, tvseries, tvepisode, video, etc)**
- **primaryTitle (string) – the more popular title / the title used by the filmmakers on promotional materials at the point of release**
- **originalTitle (string) - original title, in the original language**
- **isAdult (boolean) - 0: non-adult title; 1: adult title.**
- **startYear (YYYY) – represents the release year of a title. In the case of TV Series, it is the series start year.**
- **endYear (YYYY) – TV Sereis end year. '\N' for all other title types**
- **runtimeMinutes – primary runtime of the title, in minutes**
- **genres (string array) – includes up to three genres associated with the title**

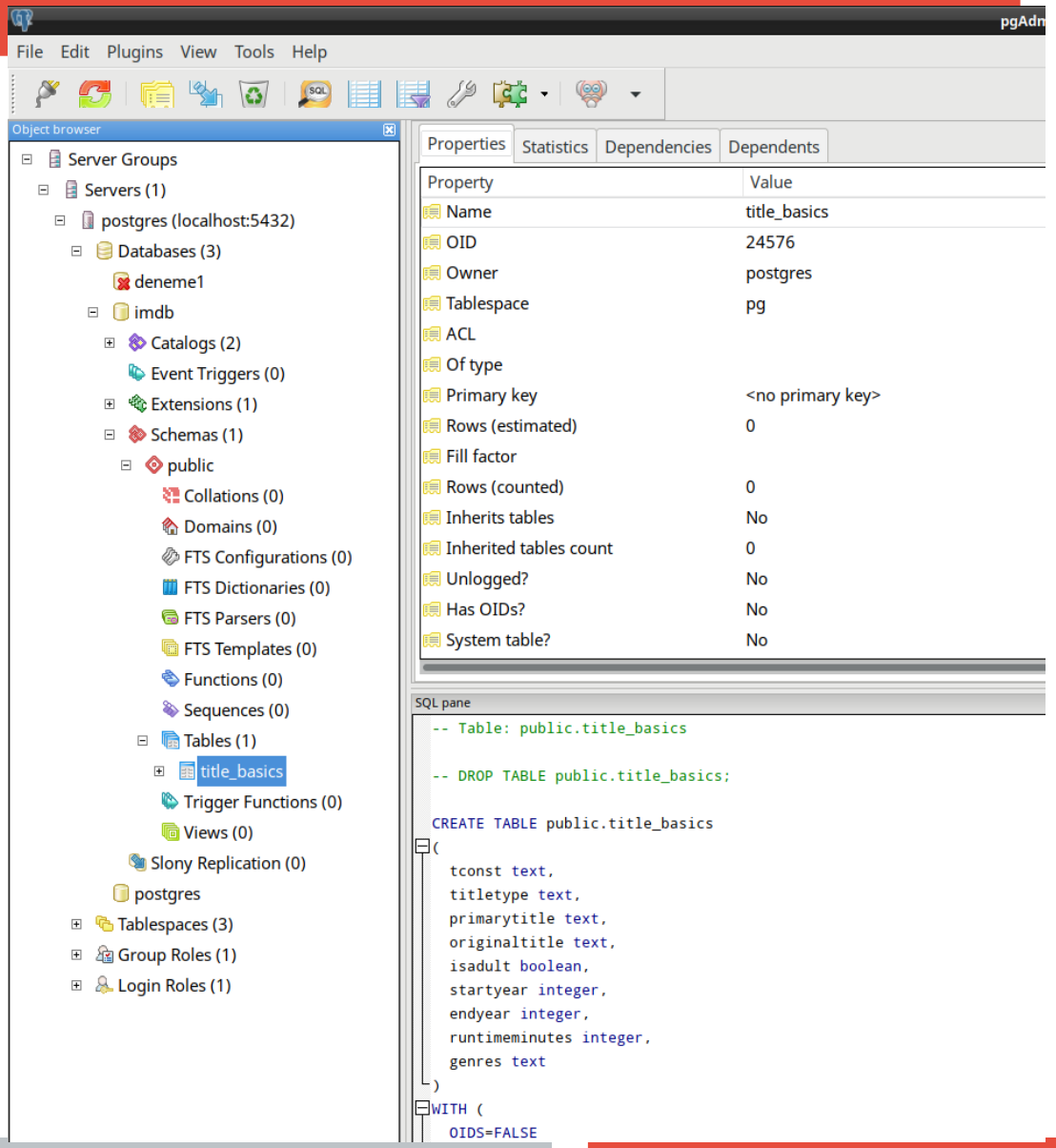
```
CREATE TABLE title_basics
(
  tconst text,
  titleType text,
  primaryTitle text,
  originalTitle text,
  isAdult boolean,
  startYear integer,
  endYear integer,
  runtimeMinutes integer,
  genres text
);
```

CREATE A NEW TABLE



CREATE A NEW TABLE

Now, we have our new table added to the object browser



IMPORT DATA WITH “COPY”

Now, we are ready to import data from external sources into our newly created empty table

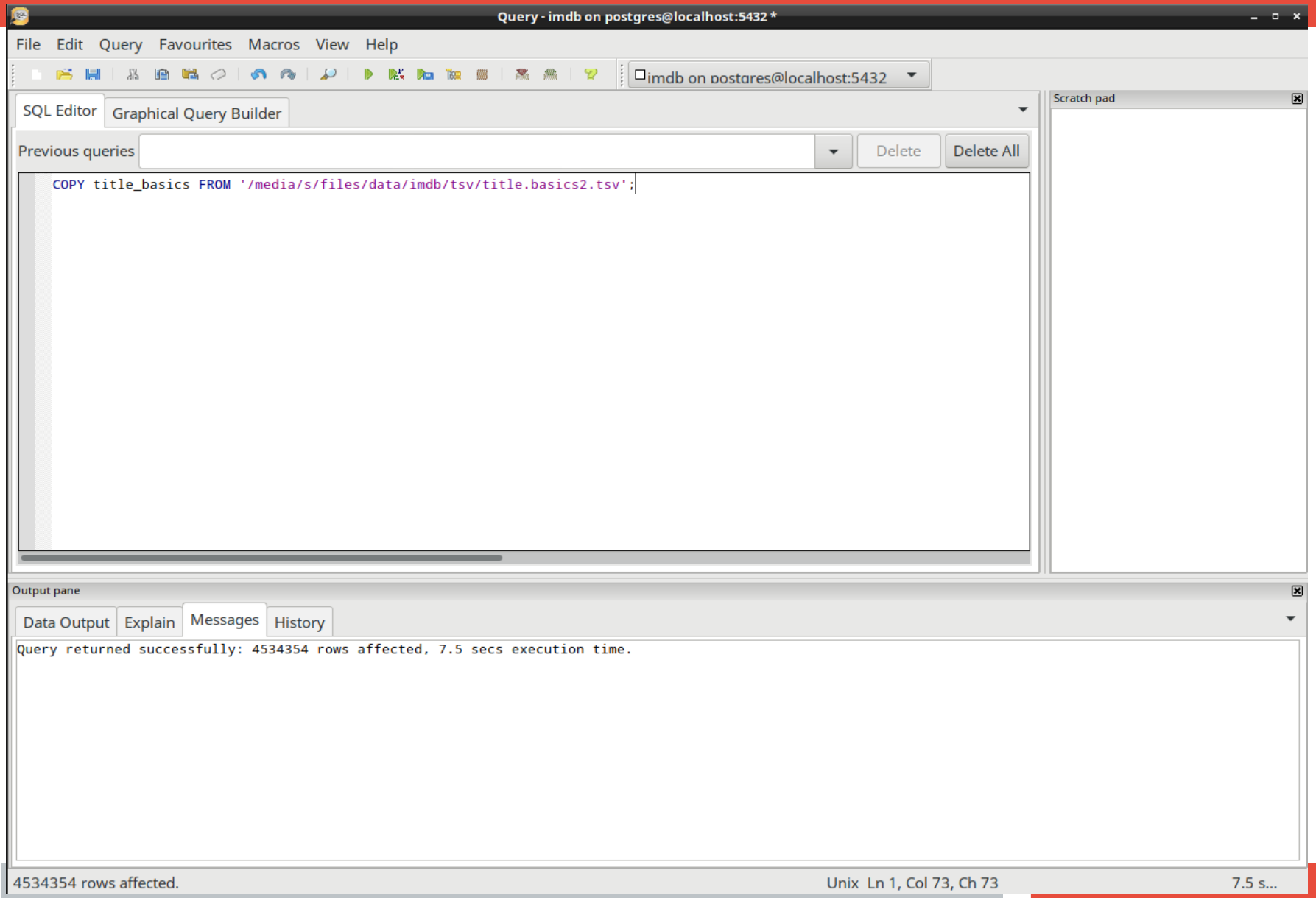
Note that, unless the input format is csv, the header file is also read and header type won’t be the same as the field types

So it is better that you first delete the first line and save it to another file:

```
[s@SS tsv]$ cat title.basics.tsv | tail -n+2 > title.basics2.tsv
[s@SS tsv]$ cat title.basics.tsv | wc -l
4534355
[s@SS tsv]$ cat title.basics2.tsv | wc -l
4534354
```

```
COPY title_basics FROM
'/media/s/files/data/imdb/tsv/title.basics2.tsv';
```

IMPORT DATA WITH “COPY”



IMPORT DATA WITH “COPY”: VIEW DATA

Edit Data - postgres (localhost:5432) - imdb - public.title_basics

File Edit View Tools Help

100 rows

	tconst text	titletype text	primarytitle text	originaltitle text	isadult boolean	startyear integer	endyear integer	runtimeminutes integer	genres text
1	tt0000001	short	Carmencita	Carmencita	FALSE	1894		1	Documentary,Short
2	tt0000002	short	Le clown et ses chiens	Le clown et ses chiens	FALSE	1892		5	Animation,Short
3	tt0000003	short	Pauvre Pierrot	Pauvre Pierrot	FALSE	1892		4	Animation,Comedy,Romance
4	tt0000004	short	Un bon bock	Un bon bock	FALSE	1892			Animation,Short
5	tt0000005	short	Blacksmith Scene	Blacksmith Scene	FALSE	1893		1	Short
6	tt0000006	short	Chinese Opium Den	Chinese Opium Den	FALSE	1894		1	Short
7	tt0000007	short	Corbett and Courtney Before the K	Corbett and Courtney Before the K	FALSE	1894		1	Short,Sport
8	tt0000008	short	Edison Kinetoscopic Record of a S	Edison Kinetoscopic Record of a S	FALSE	1894		1	Documentary,Short
9	tt0000009	movie	Miss Jerry	Miss Jerry	FALSE	1894		45	Romance
10	tt0000010	short	Employees Leaving the Lumière Fac	La sortie de l'usine Lumière à Ly	FALSE	1895		1	Documentary,Short
11	tt0000011	short	Akrobatisches Potpourri	Akrobatisches Potpourri	FALSE	1895		1	Documentary,Short
12	tt0000012	short	The Arrival of a Train	L'arrivée d'un train à La Ciotat	FALSE	1896		1	Documentary,Short
13	tt0000013	short	The Photographical Congress Arriv	Neuville-sur-Saône: Débarquement	FALSE	1895		1	Documentary,Short
14	tt0000014	short	Tables Turned on the Gardener	L'arroseur arrosé	FALSE	1895		1	Comedy,Short
15	tt0000015	short	Autour d'une cabine	Autour d'une cabine	FALSE	1894		2	Animation,Short
16	tt0000016	short	Barque sortant du port	Barque sortant du port	FALSE	1895		1	Documentary,Short
17	tt0000017	short	Italienischer Bauerntanz	Italienischer Bauerntanz	FALSE	1895		1	Documentary,Short
18	tt0000018	short	Das boxende Känguruh	Das boxende Känguruh	FALSE	1895		1	Short
19	tt0000019	short	The Clown Barber	The Clown Barber	FALSE	1898			Comedy,Short
20	tt0000020	short	The Derby 1895	The Derby 1895	FALSE	1895		1	Documentary,Short,Sport
21	tt0000022	short	Blacksmith Scene	Les forgerons	FALSE	1895		1	Documentary,Short
22	tt0000023	short	The Sea	Baignade en mer	FALSE	1895		1	Documentary,Short
23	tt0000024	short	Opening of the Kiel Canal	Opening of the Kiel Canal	FALSE	1895			News,Short
24	tt0000025	short	The Oxford and Cambridge Universi	The Oxford and Cambridge Universi	FALSE	1895			News,Short,Sport
25	tt0000026	short	The Messers. Lumière at Cards	Partie d'écarté	FALSE	1896		1	Documentary,Short
26	tt0000027	short	Cordeliers' Square in Lyon	Place des Cordeliers à Lyon	FALSE	1895		1	Documentary,Short
27	tt0000028	short	Fishing for Goldfish	La pêche aux poissons rouges	FALSE	1895		1	Documentary,Short
28	tt0000029	short	Baby's Dinner	Repas de bébé	FALSE	1895		1	Documentary,Short
29	tt0000030	short	Rough Sea at Dover	Rough Sea at Dover	FALSE	1896		1	Documentary,Short
30	tt0000031	short	Jumping the Blanket	Le saut à la couverture	FALSE	1895		1	Documentary,Short
31	tt0000032	short	Die Serpentintänzerin	Die Serpentintänzerin	FALSE	1895		1	Short
32	tt0000033	short	Trick Riding	La voltige	FALSE	1895		1	Comedy,Documentary,Short
33	tt0000034	short	Arrivée d'un train gare de Vincer	Arrivée d'un train gare de Vincer	FALSE	1896			Documentary,Short
34	tt0000035	short	Watering the Flowers	L'arroseur	FALSE	1896			Short
35	tt0000036	short	Awakening of Rip	Awakening of Rip	FALSE	1896		0	Drama,Short

Scratch pad