Wim Bertels
UC Leuven-Limburg

PGDAY'17 RUSSIA
КОНФЕРЕНЦИЯ ПО БАЗАМ ДАННЫХ

My experience with PostgreSQL and Orange in data mining

UC Leuven Limburg
MOVING MINDS

$ whoami

I’m a lecturer at UC Leuven-Limburg in Belgium teaching database, statistics and data mining courses for professional bachelors in applied IT

Data mining

$ man “data mining”

What is data mining?

$ man “data mining”

Many definitions

- Phrase to put on a CV to get hired

$ man “data mining”

Many definitions

- Phrase to put on a CV to get hired
- Non-trivial extraction of implicit, previously unknown and useful information from data

$ man “data mining”

Many definitions

- Phrase to put on a CV to get hired
- Non-trivial extraction of implicit, previously unknown and useful information from data
- Buzzword used to get money from funding agencies and venture capital firms

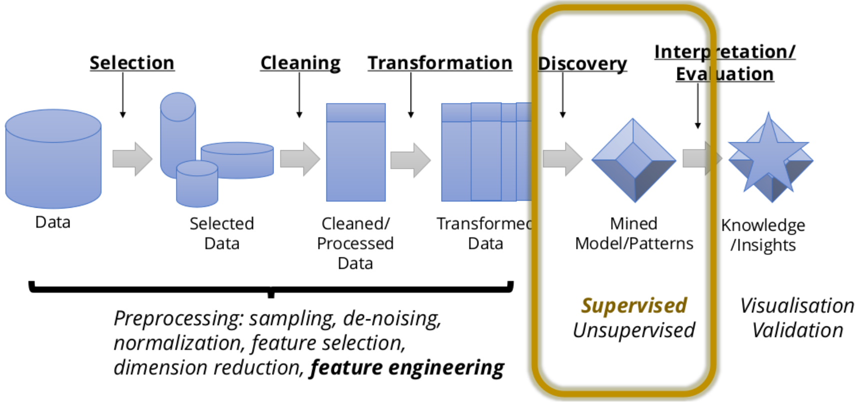
$ man “data mining”

Many definitions

- Phrase to put on a CV to get hired
- Non-trivial extraction of implicit, previously unknown and useful information from data
- Buzzword used to get money from funding agencies and venture capital firms
- (Semi-)automated exploration and analysis of large dataset to discover meaningful patterns

$ data mining -h

- Understand the data
- Extract knowlegde from the data
- Make predictions about the future



$ diff 'big data' 'data mining'

What is the difference?

$ diff 'big data' 'data mining'

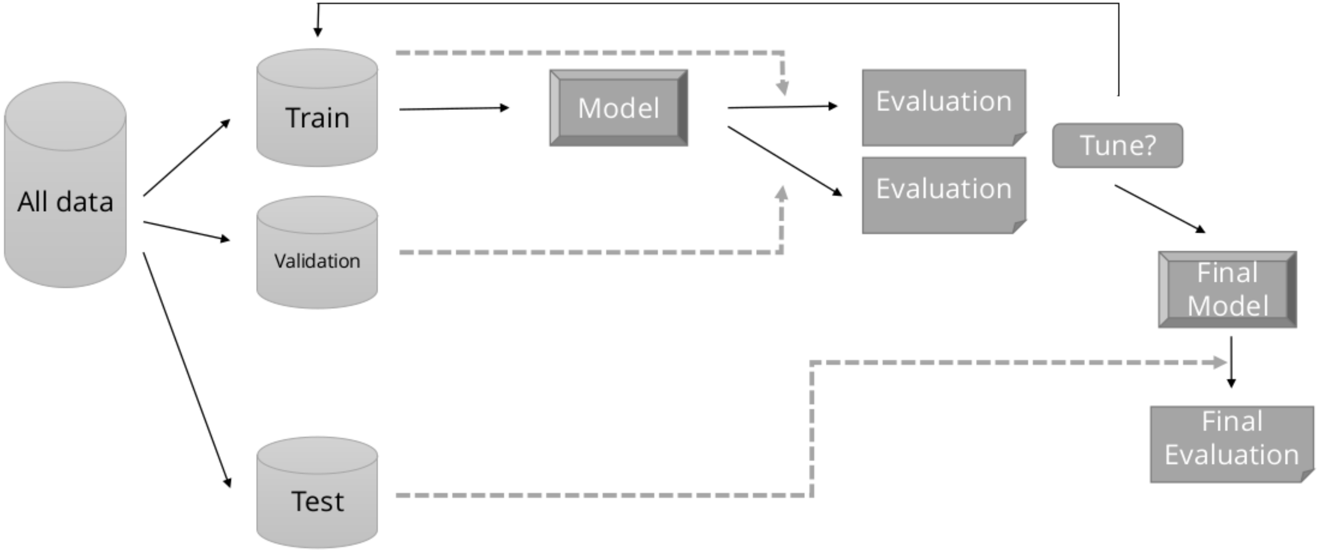
- Also a phrase to put on CV to get hired..
- By some given the same content
  - Big = usefull, novel, .. information
- Size
- Resource

A view on data mining

- Exploration
- Learning
  - Supervised
    - Regression
    - Classification
  - Unsupervised

Supervised: build models

- Training
- Validation
- (Test)



Build models: sampling

- Random
- Stratified if possible

3 common choices

- R
- Python
- Scala

Python: Orange

Build upon

- numpy
- scipy
- scikit-learn

General Storage Guidelines

Different systems

- Operational vs Analytical
- Normalized vs Denormalized

Stars, snowflakes and variants

- Facts and dimensions ..
- DWH

Analytical

- Timestamp
  - Valid
    - From
    - Until
  - Creation
  - ID
  - ..

Denormalized

- Performance
- (Olap)

Constraints

- Not a operational datastore
- If (checks for loading scripts ok), then (drop unused constraints)
  - Integrity of the original data

PostgreSQL as a datastore

PostgreSQL as a datastore

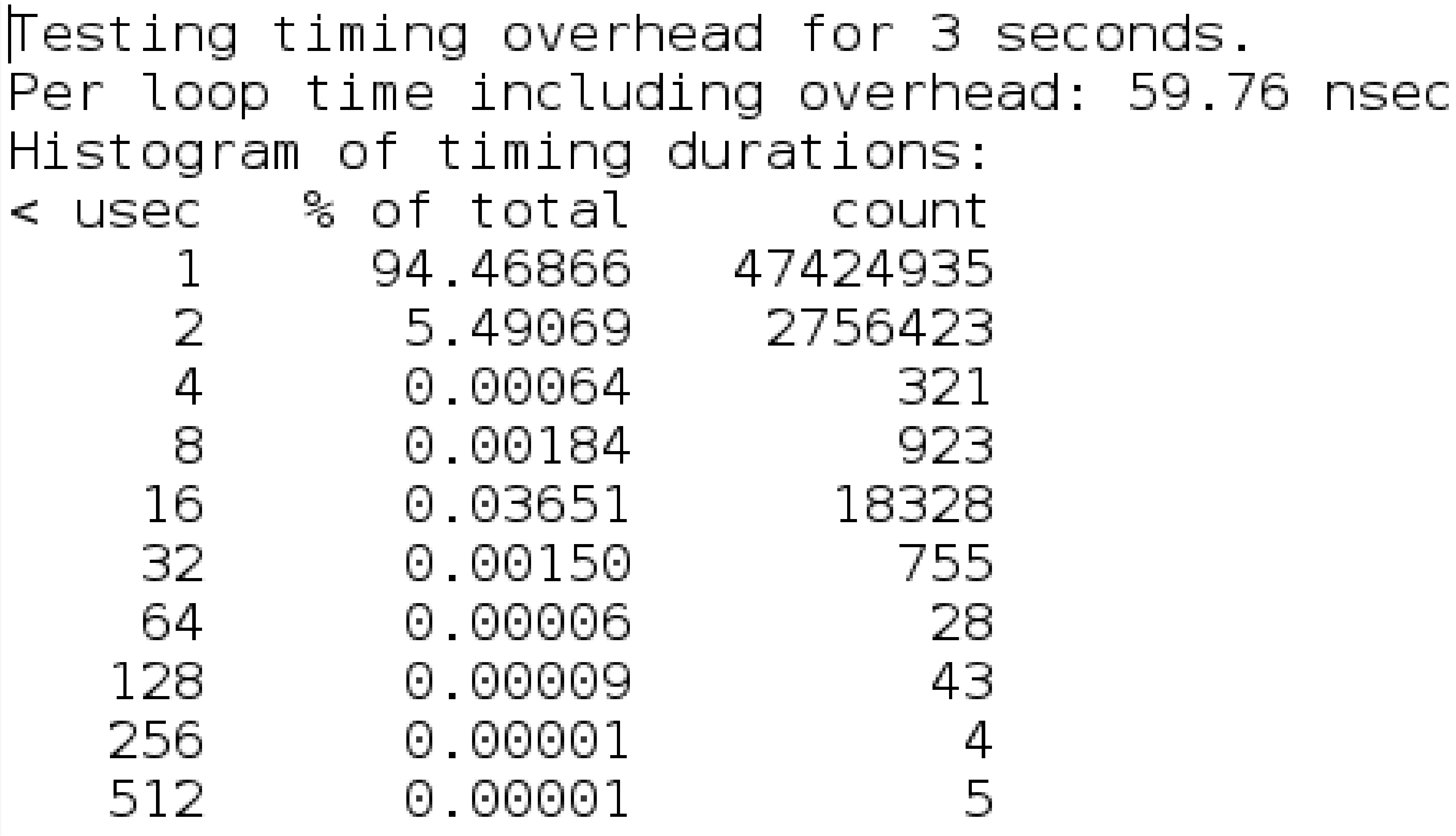
- Setup
- Basic tests
- Basic tuning
- Loading the data
- Space
- Sampling
- Cstore

Setup

- Read performance (olap vs oltp)
- Commodity hardware:
  - 4 cpu cores
  - 8 GB RAM
  - KVM
  - ext4

Basic tests

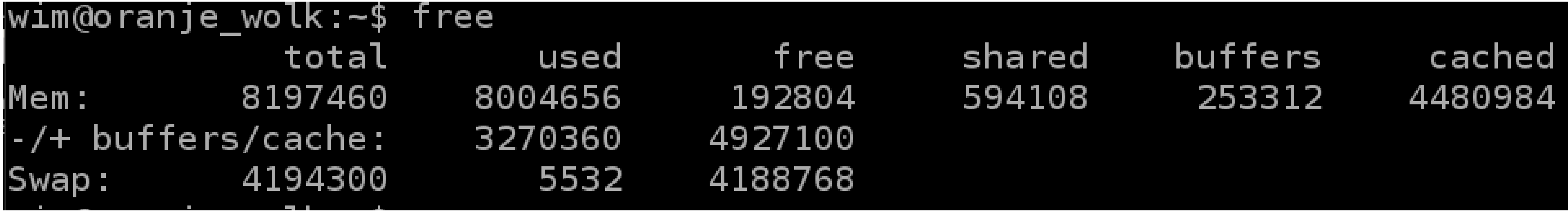
- pg_bench
- pg_test_fsync
- Vm
  - pg_test_timing



```
Testing timing overhead for 3 seconds.
Per loop time including overhead: 59.76 nsec
Histogram of timing durations:
< usec   % of total      count
     1     94.46866   47424935
     2      5.49069    2756423
     4      0.00064        321
     8      0.00184        923
    16      0.03651      18328
    32      0.00150        755
    64      0.00006         28
   128      0.00009         43
   256      0.00001          4
   512      0.00001          5
```

Basic tuning

- $ free



```
wim@oranje_wolk:~$ free
             total       used       free     shared    buffers     cached
Mem:       8197460    8004656     192804     594108     253312    4480984
-/+ buffers/cache:    3270360    4927100
Swap:      4194300       5532    4188768
```

(Read) Tuning

- shared_buffers = '2GB'
- shared_preload_libraries = 'pg_stat_statements,cstore_fdw'
- work_mem = '128MB'
- max_parallel_workers_per_gather = '2'
- effective_cache_size = '4GB' (or 5GB)

Loading the data

- COPY
- https://www.postgresql.org/docs/current/static/populate.html
- maintenance_work_mem in the session/script
  - SET maintenance_work_mem TO '1GB';
  - RESET maintenance_work_mem;
- Analyze
- Avoid single row inserts (single transaction)

“Streaming data”

- Wifi > Radius > Attendance
- Quickly grows over several weeks..
- VACUUM vs VACUUM FULL
- Manage

Tilted device could pinpoint pin number for hackers, study reveals

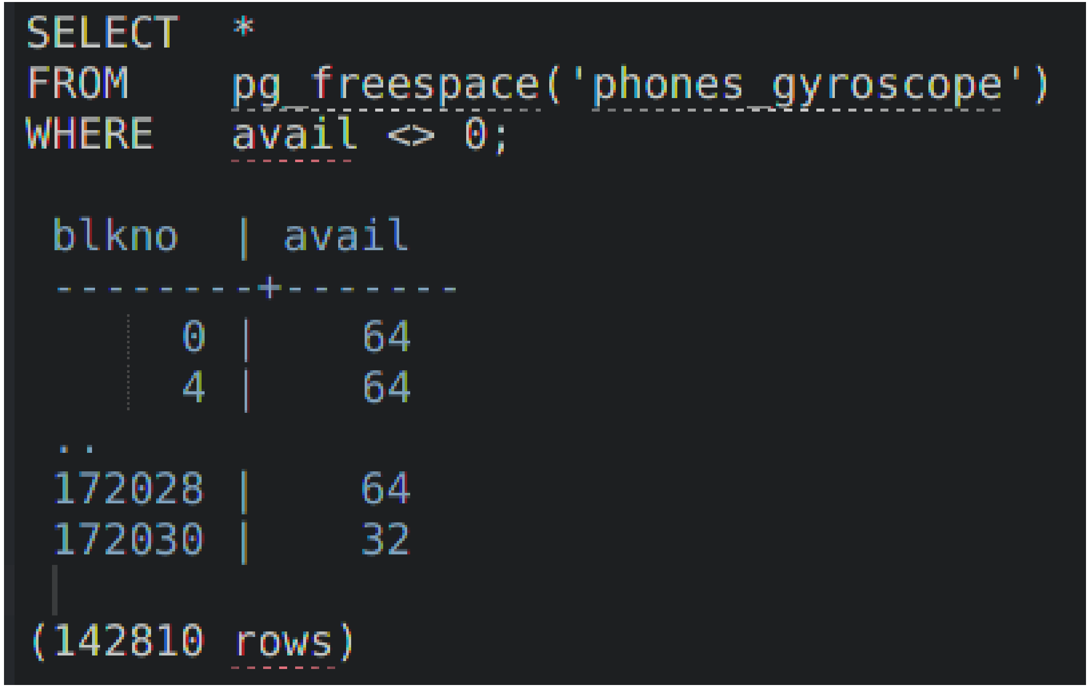
Researchers were able to guess a four-digit code with 70% accuracy at the first attempt and 100% at fifth just from how a device was held.

Hackers could steal mobile phone users' pin numbers from the way their devices tilt as they type on them, researchers have claimed.

Computer scientists at Newcastle University managed to guess a four-digit pin with 70% accuracy at the first attempt by using the gyroscopes built into all modern smartphones. With five attempts, the team was able to correctly guess the pin 100% of the time.

Space: after COPY

- CREATE EXTENSION pg_freespacemap;



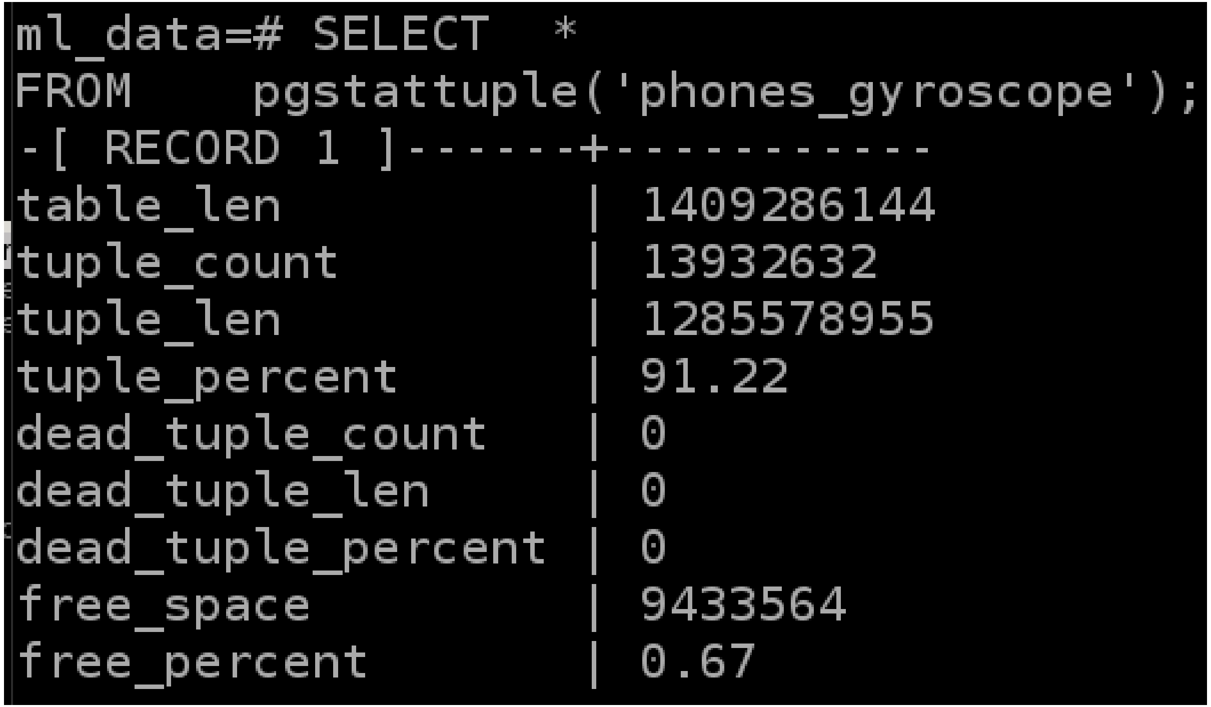
```
SELECT  *
FROM    pg_freespace('phones_gyroscope')
WHERE   avail <> 0;

 blkno  | avail
--------+-------
      0 |    64
      4 |    64
..
 172028 |    64
 172030 |    32

(142810 rows)
```

Space: another angle

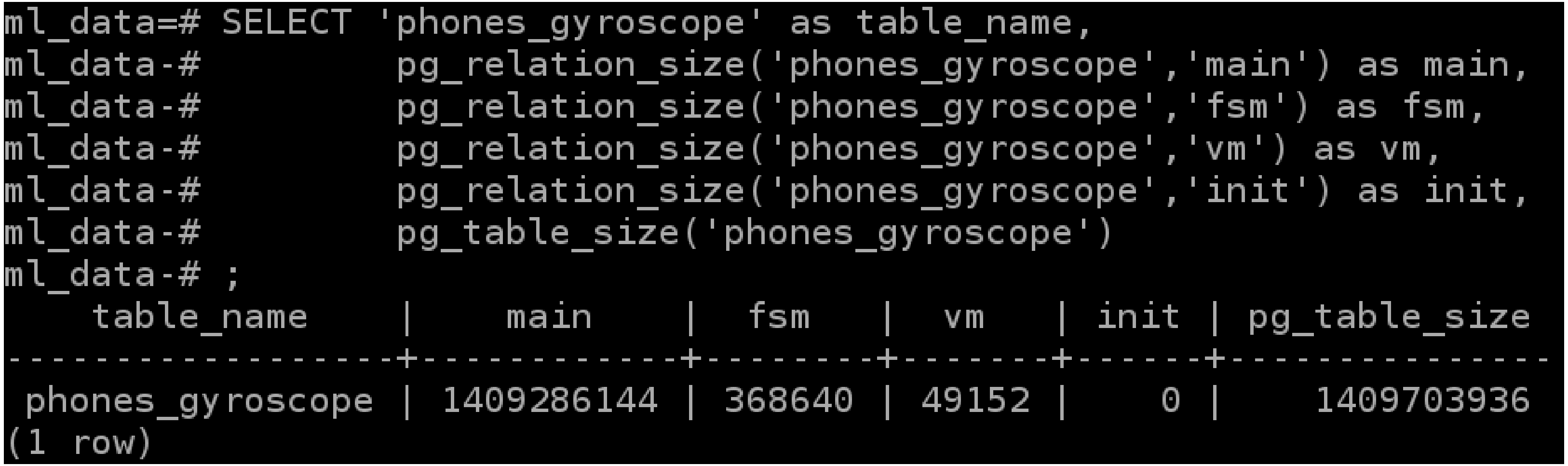
- CREATE EXTENSION pgstattuple;



```
ml_data=# SELECT  *
FROM    pgstattuple('phones_gyroscope');
-[ RECORD 1 ]------+------------
table_len          | 1409286144
tuple_count        | 13932632
tuple_len          | 1285578955
tuple_percent      | 91.22
dead_tuple_count   | 0
dead_tuple_len     | 0
dead_tuple_percent | 0
free_space         | 9433564
free_percent       | 0.67
```

Space: vacuum side effect

- Running vacuum will not change the physical table but add a tiny vm table
  - + 0,0035%



```
ml_data=# SELECT 'phones_gyroscope' as table_name,
ml_data-#         pg_relation_size('phones_gyroscope','main') as main,
ml_data-#         pg_relation_size('phones_gyroscope','fsm') as fsm,
ml_data-#         pg_relation_size('phones_gyroscope','vm') as vm,
ml_data-#         pg_relation_size('phones_gyroscope','init') as init,
ml_data-#         pg_table_size('phones_gyroscope')
ml_data-# ;
    table_name    |    main    |  fsm   |  vm   | init | pg_table_size
------------------+------------+--------+-------+------+---------------
 phones_gyroscope | 1409286144 | 368640 | 49152 |    0 |    1409703936
(1 row)
```

Sampling

- TABLESAMPLE option (since 9.5)
  - SYSTEM or BERNOULLI
- Let's compare them for performance
  - First SYSTEM
  - Then BERNOULLI

```
EXPLAIN (ANALYZE true, BUFFERS true, TIMING true)
SELECT  *
FROM    phones_gyroscope TABLESAMPLE SYSTEM(50);
```

```
                                                             QUERY PLAN
 -----------------------------------------------------------------------
 Sample Scan on phones_gyroscope  (cost=0.00..413741.26 rows=6967726
 width=61) (actual time=4.201..31722.369 rows=6956984 loops=1)
   Sampling: system ('50'::real)
   Buffers: shared read=85900
 Planning time: 7.857 ms
 Execution time: 32351.848 ms
(5 rows)
```

```
EXPLAIN (ANALYZE true, BUFFERS true, TIMING true)
SELECT  *
FROM    phones_gyroscope TABLESAMPLE BERNOULLI(50);
```

```
                                                             QUERY PLAN
 -----------------------------------------------------------------------
 Sample Scan on phones_gyroscope  (cost=0.00..241709.26 rows=6967726
 width=61) (actual time=2.092..5216.084 rows=6969615 loops=1)
   Sampling: bernoulli ('50'::real)
   Buffers: shared hit=32 read=172000
 Planning time: 0.223 ms
 Execution time: 5803.063 ms
```

Sample: Timings

- Bernoulli seems faster
  - 5216.084 ms < 31722.369 ms
- Why?

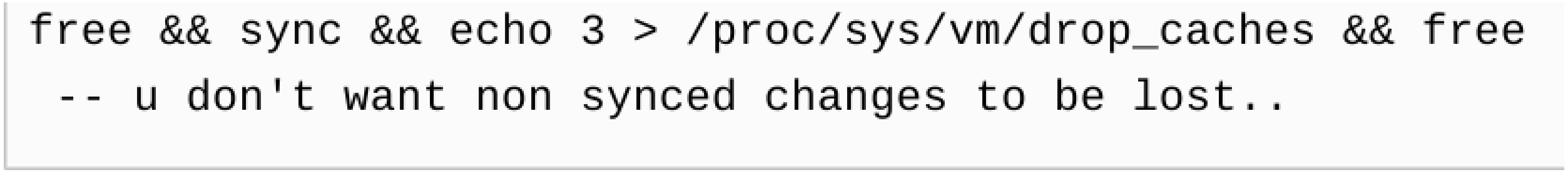
Explain: cost and time

Method	Cost	Time
1. SYSTEM	413741.26	32351.848 ms
2. BERNOULLI	241709.26	5803.063 ms
3. SYSTEM	413741.26	1710.712 ms

Caching

- CREATE extension pg_buffercache;
- After earn statement the cache grew
  https://github.com/postgres/postgres/blob/master/src/backend/storage/buffer/README#L208
- From empty to 3*32 after 3 sample scans with REPEATABLE seed
  - 32 8k buffers / sample scan (=sequential scan)
- The cost of EXPLAIN is misleading in this case

reset OS Cache



```
free && sync && echo 3 > /proc/sys/vm/drop_caches && free
 -- u don't want non synced changes to be lost..
```

- SYSTEM method is faster

Optimizing TABLESAMPLE?

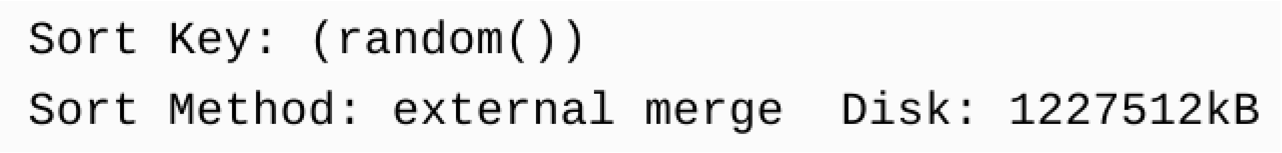
- Index: no benefit
- Parallel querying: no benefit (9.6)

Other sampling methods

- 50% / 30% / 20% sample (TVT)
  - based on random() sort order
  - Repeatable: SELECT setseed(0.17);
    - Between -1 and 1
  - 13932632 rows in total
- ORDER BY OR add Column
- tsm_system_rows and tsm_system_time

random() SORT order

```
SELECT *
FROM phones_gyroscope
ORDER BY random()
FETCH FIRST 6966316 ROWS ONLY;
-- work_mem
```



```
Sort Key: (random())
Sort Method: external merge  Disk: 1227512kB
```

ADD a random() column

- 3 options
  - ADD COLUMN aselect double precision;

    -
      ```
      UPDATE  phones_gyroscope_dm
      SET     aselect = random();
      ```
  - ADD COLUMN aselect double precision DEFAULT random();
  - CREATE UNLOGGED TABLE phones_gyroscope_dm AS
    ```
        SELECT  *, random() AS aselect
        FROM    phones_gyroscope;
    ```

random(): performance and size

- ADD COLUMN +UPDATE is slower than CREATE UNLOGGED TABLE
- ADD COLUMN + UPDATE is in need of VACUUM FULL:

ADD COLUMN	ADD COLUMN + UPDATE	CREATE
1451 MB	2796 MB	1451 MB

Which one to choose?

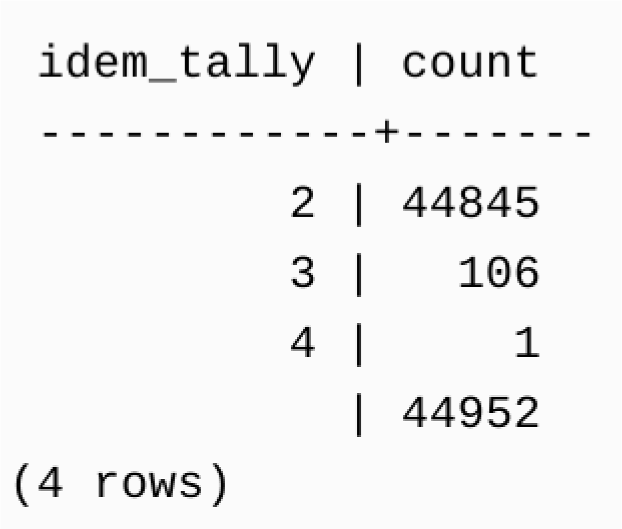
- Don't use ADD COLUMN and UPDATE

Final touch for sample tables

- CREATE INDEX ON phones_gyroscope_dm(aselect);
- CLUSTER VERBOSE phones_gyroscope_dm USING phones_gyroscope_dm_aselect_idx;
  - Remember maintenance_work_mem

Random() =? aselect

```
WITH ac AS (
    SELECT  aselect, count() as idem_tally
    FROM    phones_gyroscope_dm
    GROUP BY aselect
    HAVING  count()>1
    ORDER BY 2)
SELECT  idem_tally, count(*)
FROM    ac
GROUP BY ROLLUP (idem_tally)
ORDER BY 1,2;
```



```
 idem_tally | count
------------+-------
          2 | 44845
          3 |   106
          4 |     1
            | 44952
(4 rows)
```

Collision %

- SELECT 44952.0/13932632*100

  AS collision_percentage;

  0.32%

- Remark: This grows with the table size.

tsm_system_rows

- CREATE EXTENSION tsm_system_rows;
- like the built-in SYSTEM sampling method not completely random (blocklevel), about the same performance, but uses the number of rows as parameter, as such more accurate than the SYSTEM method
- Not repeatable

tsm_system_time

- like the built-in SYSTEM sampling method not completely random (blocklevel)
- u don’t know how many rows will be returned in this case, but you have time limit for reading the table
- not repeatable

sampling Overview	TABLE SAMPLE				RANDOM()	
	BUILT IN		EXTENSIONS			
	BER NOUILLI	**SYS TEM**	**SYSTEM ROWS**	**SYSTEM TIME**	**ORDER BY**	**ADD column + Index**
REPEATABLE	***yes***	***yes***	*no*	*no*	***yes***	***yes***
RANDOMNESS	***good***	***less***	***less***	***less***	***good***	***good***
PERFORMANCE	***3***	***2***	***2***	***1***	*4*	*5**
TIME_LIMIT	*no*	*no*	*no*	***yes***	*no*	*no*
EXACT nr Of ROWS	***almost***	***almost***	***yes***	*no*	***yes***	***yes***

* DML is needed (create) or (create and alter) (> TVT)

TVT setup

- I prefer the ADD COLUMN method
- It allows for a clear TVT
- How would you make a TVT with TABLESAMPLE? (3 separate/disjunct sets)

TVT TABLESAMPLE

- Just using them 3 times will give overlap
- Exclusion?

```
SELECT   *
FROM     phones_gyroscope TABLESAMPLE
             BERNOULLI(30)
WHERE    index NOT IN (:
    SELECT index FROM phones_gyroscope_ts_train:);
```

:) + processing order

```
 Planning time: 0.103 ms
 Execution time: 357532201.668 ms
(12 rows)
```

```
-- 357532201.668/1000/60/60/24 = 4 days
-- What about the number of rows? 2014
-- Why isn't this the same as before, ie 4180027
-- Then why isn't this closer to 4180027/2 = 2090013
-- Don't trust your data..
-- The title of the column index is misleading, it is not unique..
```

Good samples?

- A basic statistics test on comparing the averages to the baseline full table.

```
\set kolom arrival_time

SELECT  'phones_gyroscope' AS tabel ,avg(:kolom), variance(:kolom), count(:kolom)
FROM    phones_gyroscope
UNION
..
SELECT  'rt_phones_gyropscope_system_time_1000_1',avg(:kolom), variance(:kolom), count(:kolom)
FROM    rt_phones_gyropscope_system_time_1000_1
```

Avg	P (1 sided)			Row%	Timing
	two samples	Compared to 'population'			
SYSTEM(0,1%)*	**5,01E-004**	**1,05%**	**4,22E-011**	0,10%	About 5ms
system_time(1s)	11,86%	40,34%	9,09%	3,65%	About 1s
BERNOUILLI(**0,1%**)	49,04%	46,91%	48,28%	0,10%	**About 500ms**
SYSTEM(50%)		10,90%		50,00%	About 2s
BERNOULLI(**50%**)		46,13%		50,00%	About 3s

Cstore

Cstore

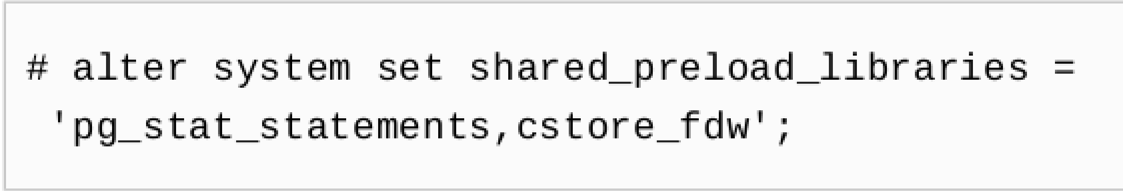
- Debian (install) tips
- Size comparison
- OLAP performance

Debian specific

- $ aptitude install postgresql-server-dev-9.6
- $ pgxn install cstore_fdw

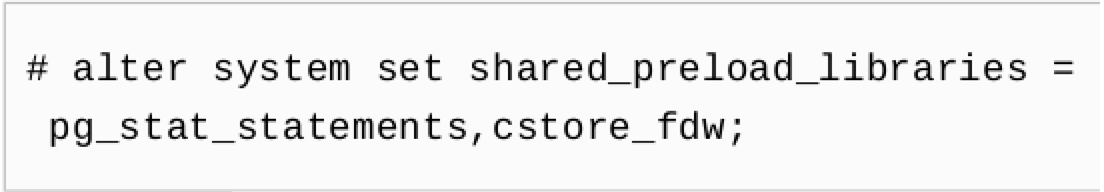
Side note on ALTER SYSTEM

Will result a bad config:

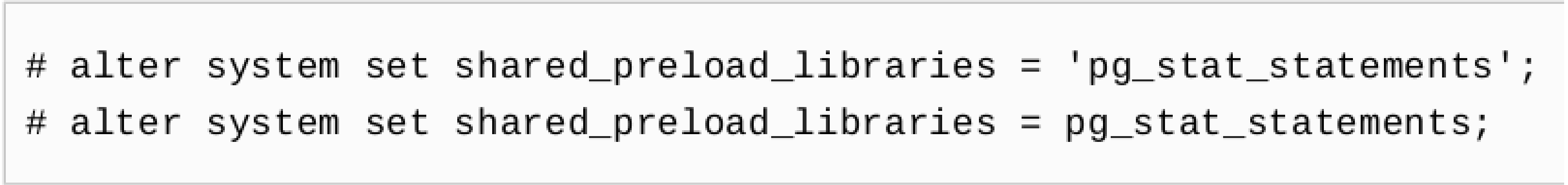


```
# alter system set shared_preload_libraries =
 'pg_stat_statements,cstore_fdw';
```

Will not:

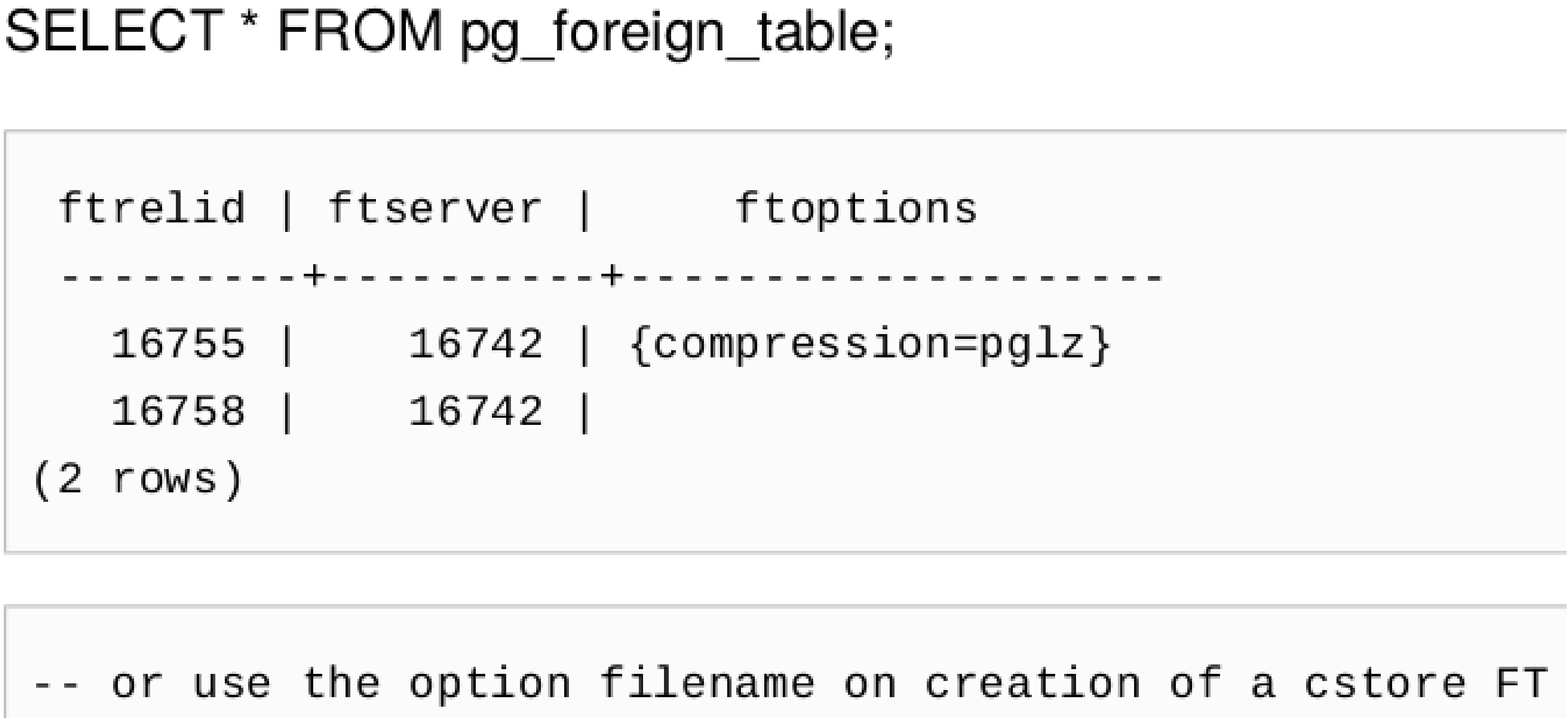


```
# alter system set shared_preload_libraries =
 pg_stat_statements,cstore_fdw;
```



```
# alter system set shared_preload_libraries = 'pg_stat_statements';
# alter system set shared_preload_libraries = pg_stat_statements;
```

Size: relid lookup

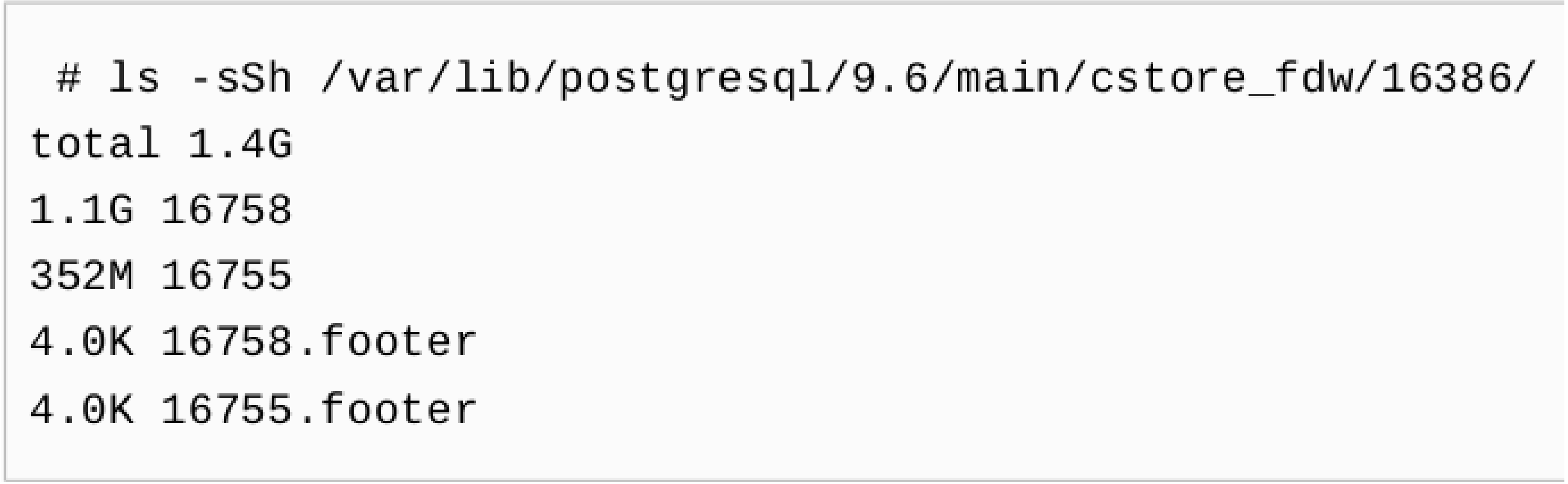


SELECT * FROM pg_foreign_table;

```
 ftrelid | ftserver |      ftoptions
---------+----------+----------------------
   16755 |    16742 | {compression=pglz}
   16758 |    16742 |
(2 rows)
```

```
-- or use the option filename on creation of a cstore FT
```

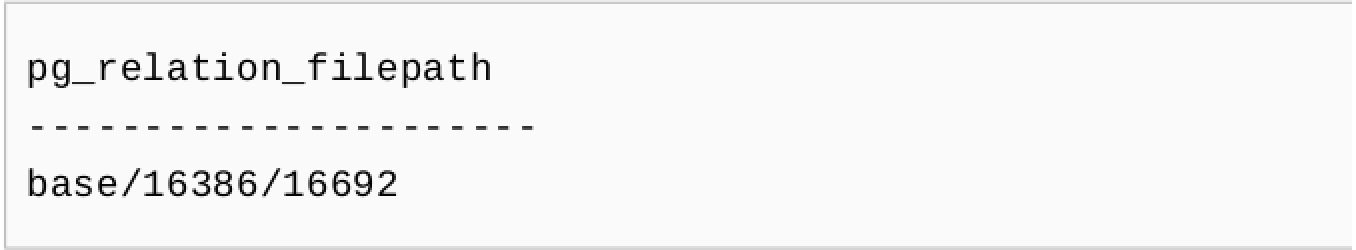
Size cstore tables



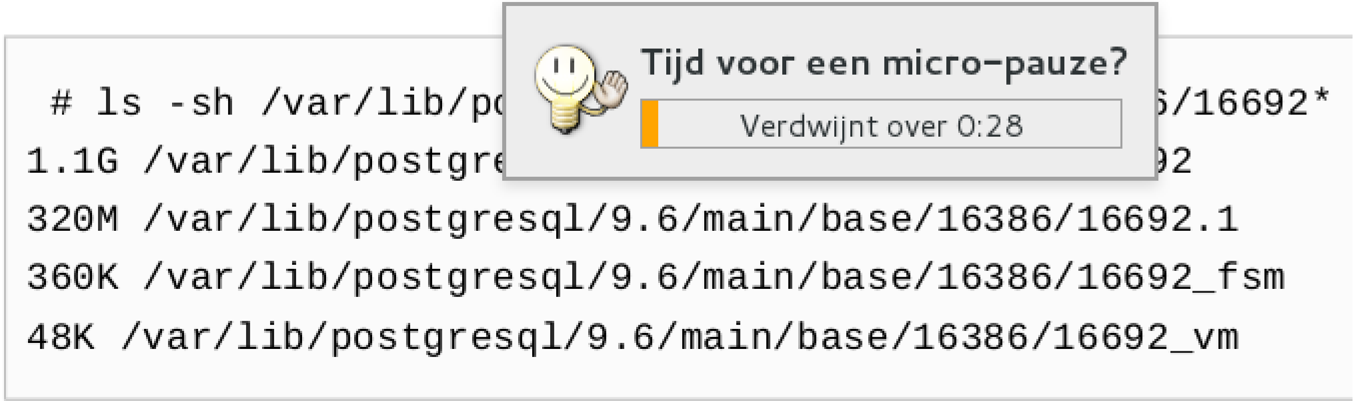
```
 # ls -sSh /var/lib/postgresql/9.6/main/cstore_fdw/16386/
total 1.4G
1.1G 16758
352M 16755
4.0K 16758.footer
4.0K 16755.footer
```

Size postgresql tables

SELECT pg_relation_filepath('phones_gyroscope');



```
pg_relation_filepath
----------------------
base/16386/16692
```



```
 # ls -sh /var/lib/pc                          5/16692*
1.1G /var/lib/postgre                          92
320M /var/lib/postgresql/9.6/main/base/16386/16692.1
360K /var/lib/postgresql/9.6/main/base/16386/16692_fsm
48K /var/lib/postgresql/9.6/main/base/16386/16692_vm
```

Tijd voor een micro-pauze?

Verdwijnt over 0:28

Size comparison

- Compressed is significantly smaller
  - factor 4 in this case
- Not compressed is about 80%

OLAP Performance

- ROLLUP, CUBE, GROUPING SETS
  - “GROUP BY ROLLUP (gt)
  - =
  - GROUP BY gt

    UNION

    GROUP BY ()”

OLAP Performance

- If there is where condition that triggers an index, then this has a bigger impact than the GROUP BY
- Sorted creation is important for Cstore
- Without indexes cstore compressed is a clear winner
- A compressed cstore was about the same size as an index
- Side note: rollup .. vs union's (parallel queries)

On my test tables in general

- slow>faster
- Regular no index>cstore>

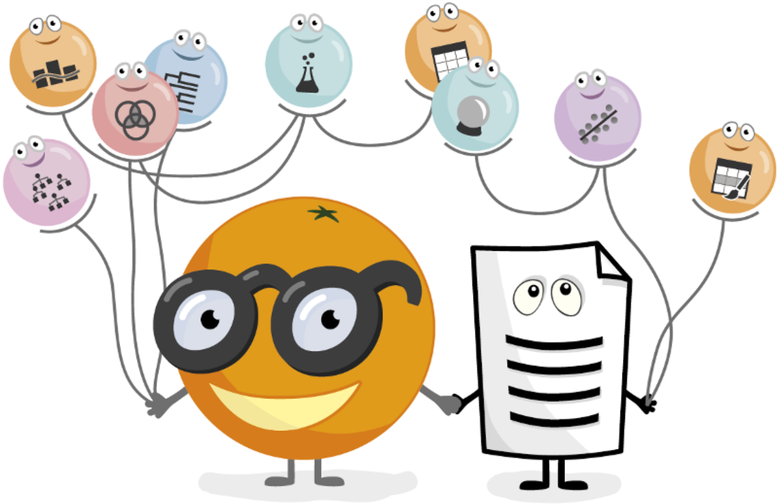
  cstore_good_sorting>regular index used

- c_regular>c_compressed

Notes about cstore

- No update or delete
  - You can append data
- No indexes, but a lightweigt alternative:
- For each block, cstore_fdw keeps track of min and max values
- No TABLESAMPLE on foreign tables
- Usage of random() is comparable
- Within these constraints, especially for limiting space consumption, Cstore compressed is good option

Orange



Orange

- Setup
- Performance
- Gui
- Side effects
- Demo
- Python
- PL/python

Setup

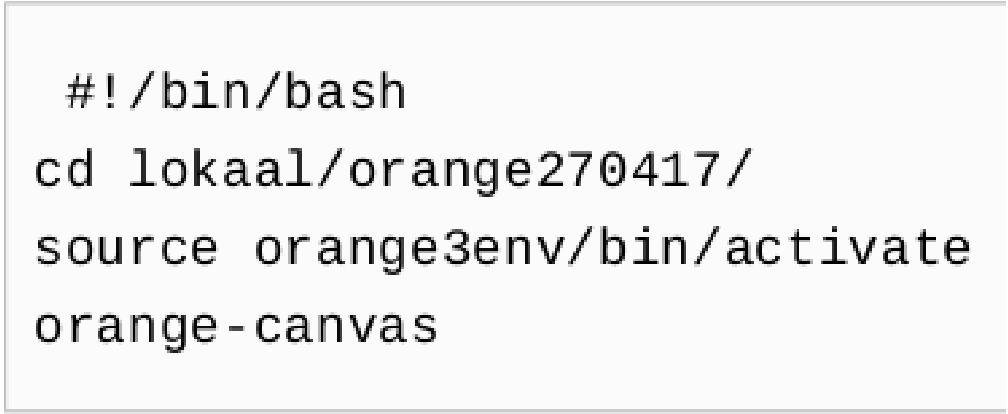
- mac osx: no problems
- windows: no real problems
- linux: can cost you a bit more time, the guidelines are a bit spread out
  - Less stable than PostgreSQL, as probably m... a.. :)

Debian installation tips

- Needed packages: virtualenv git build-essential python3 python3-dev python3-openssl python3-sip-dev aptitude install python3-pyqt4 libqt4-dev python-qt4-dev
- Use anaconda or pip (or install from git repo)
- $ pgxn install quantile
  - Support SQLTable

Virtualenv

- Prefered by user or Latest version



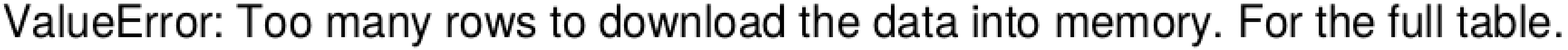
```
 #!/bin/bash
cd lokaal/orange270417/
source orange3env/bin/activate
orange-canvas
```

Performance

- Memory
- CPU

Memory

- Depending on the widgets being used
- Learning: orange needs the data in memory
- The Orange gui seems practically to allow up to 4% of the systems memory for table sizes (tested on 8 and 16GB)



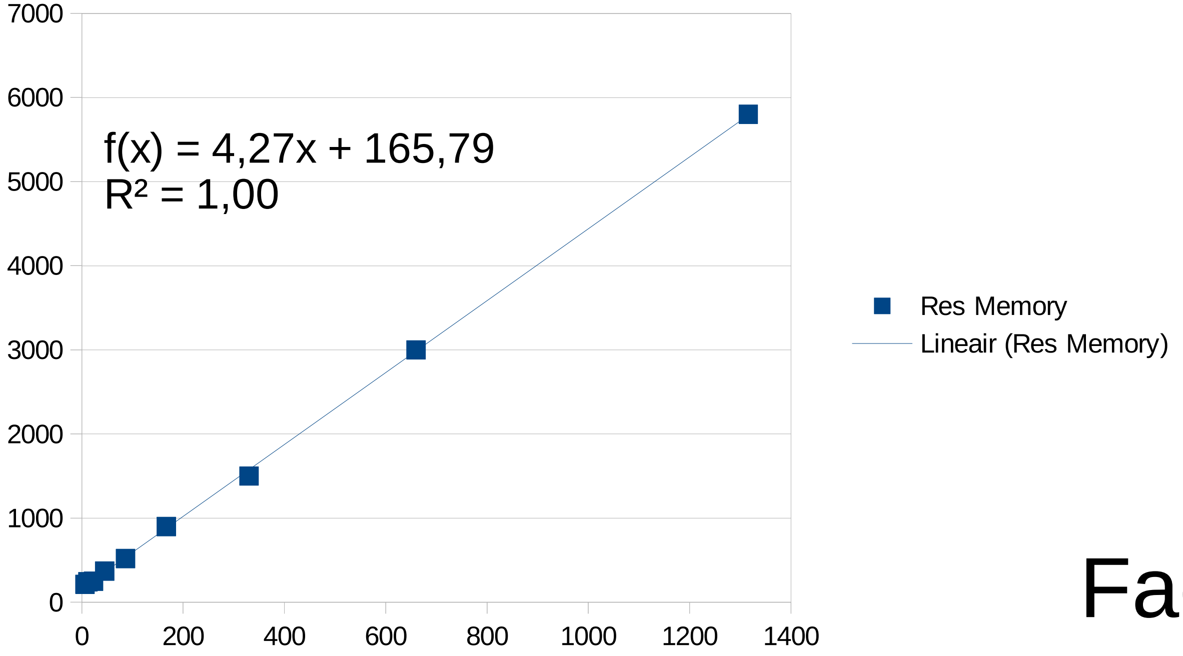
```
ValueError: Too many rows to download the data into memory. For the full table.
```

Orange suggestions

- Luckily you will get suggestions for handling large data
  - Samples are taken using
    - SYSTEM
    - system_time
- Recall that this might be a problem (block based)
  - In my practice this often isn't a problem from samplesize of 5%
  - It largely depends on the table size and the randomness within the blocks

CPU

- Orange does not always use parallel processing, so even if the data fits into memory, the single core being used can become a bottleneck. Hence the <4% suggestion for the gui.
- maximum for the processing before (size)



Factor 4 (to 12)

Side effects

- When loading data from an existing database table Orange creates extra persistent “background” tables.
  - - space
  - + index
- Several widgets will use these “background” sample tables.

Demo

PL/Python

- Access to virtualenv? (Linux)
- plpython3u

```
CREATE OR REPLACE FUNCTION workon(venv text)
  RETURNS void AS
$BODY$
    import os
    import sys

    if sys.platform in ('win32', 'win64', 'cygwin'):
        activate_this = os.path.join(venv, 'Scripts', 'activate_this.py')
    else:
        if 'PATH' not in os.environ:
            import subprocess
            p=subprocess.Popen('echo -n $PATH', stdout=subprocess.PIPE, shell=True)
            (mypath,err) = p.communicate()
            os.environ['PATH'] = mypath.decode("utf8")
            plpy.info(os.environ['PATH'])
        activate_this = os.path.join(venv, 'bin', 'activate_this.py')
    exec(open(activate_this).read(), dict(__file__=activate_this))
    plpy.info(os.environ['PATH'])
$BODY$
LANGUAGE plpython3u VOLATILE;
```

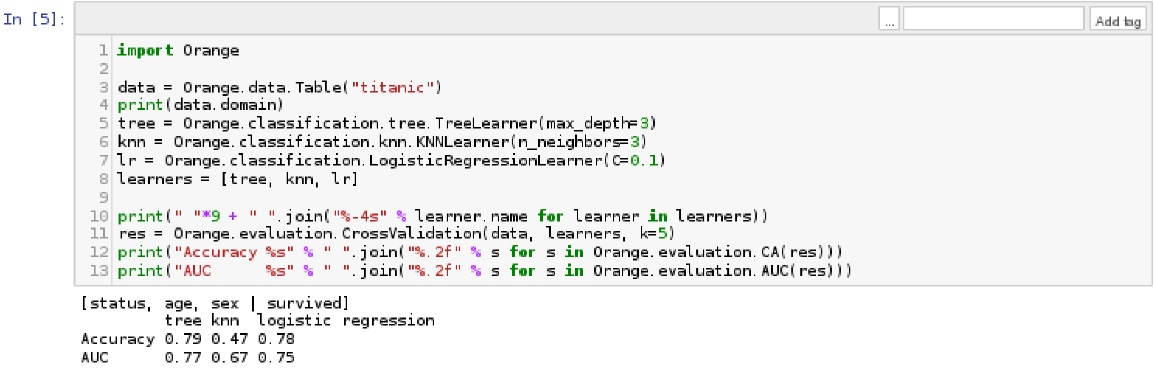
Load the script and continu

```
SELECT workon('/home/wim/lokaal/orange270417/orange3env');

CREATE OR REPLACE FUNCTION use_orange()
  RETURNS text[] AS
$BODY$
    import Orange
    data = Orange.data.Table("voting")
    classifier = Orange.classification.LogisticRegressionLearner(data)
    c_values = data.domain.class_var.values
    for d in data[5:8]:
        c = classifier(d)
        plpy.info("{}, originally {}".format(c_values[int(classifier(d)[0])],d.get_class()))
    return Orange.version.version
$BODY$
LANGUAGE plpython3u VOLATILE;
```

Scripts

- Python or PL/Python
  - A matter of personal choice
  - Eg jupyter notebook



```python
import Orange

data = Orange.data.Table("titanic")
print(data.domain)
tree = Orange.classification.tree.TreeLearner(max_depth=3)
knn = Orange.classification.knn.KNNLearner(n_neighbors=3)
lr = Orange.classification.LogisticRegressionLearner(C=0.1)
learners = [tree, knn, lr]

print(" "*9 + " ".join("%-4s" % learner.name for learner in learners))
res = Orange.evaluation.CrossValidation(data, learners, k=5)
print("Accuracy %s" % " ".join("%.2f" % s for s in Orange.evaluation.CA(res)))
print("AUC      %s" % " ".join("%.2f" % s for s in Orange.evaluation.AUC(res)))
```

```
[status, age, sex | survived]
         tree knn  logistic regression
Accuracy 0.79 0.47 0.78
AUC      0.77 0.67 0.75
```

Pgpredict

Documentation (update)

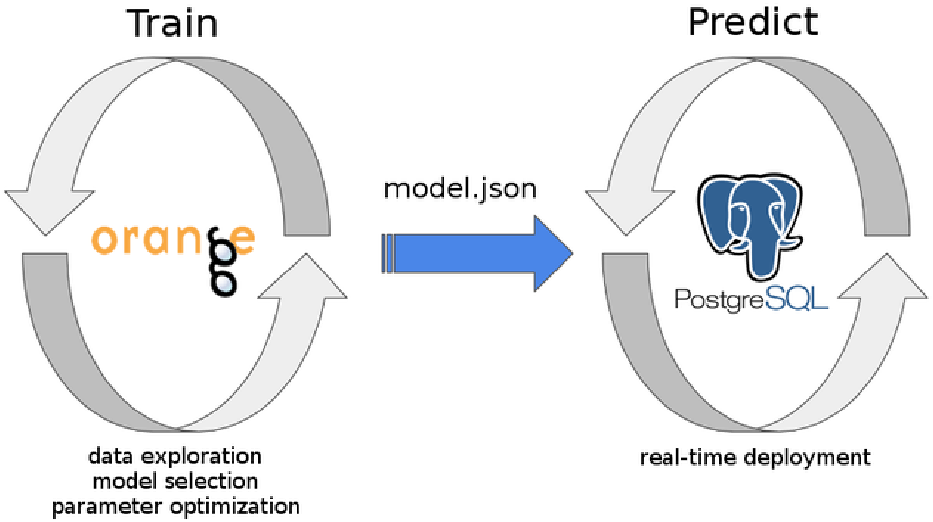
- U need to install the “export model” widget following the instructions inside the pgpredict archive.
- U need to create some functions (.sql script)
- Learner: Mean → Constant
- Requirements: Orange → Orange3

2 learning techniques

- Regression
- Logistic Regression

Workflow

Orange gui → export model → .json file → load file in pgpredict function → make predictions



PL/Python and reading files..

- File permissions
- User running PL/Python must be able to access the files
- In most (linux) setups the *postgres* user cannot read your personal files
- Suggestion: link files to postgres readable location

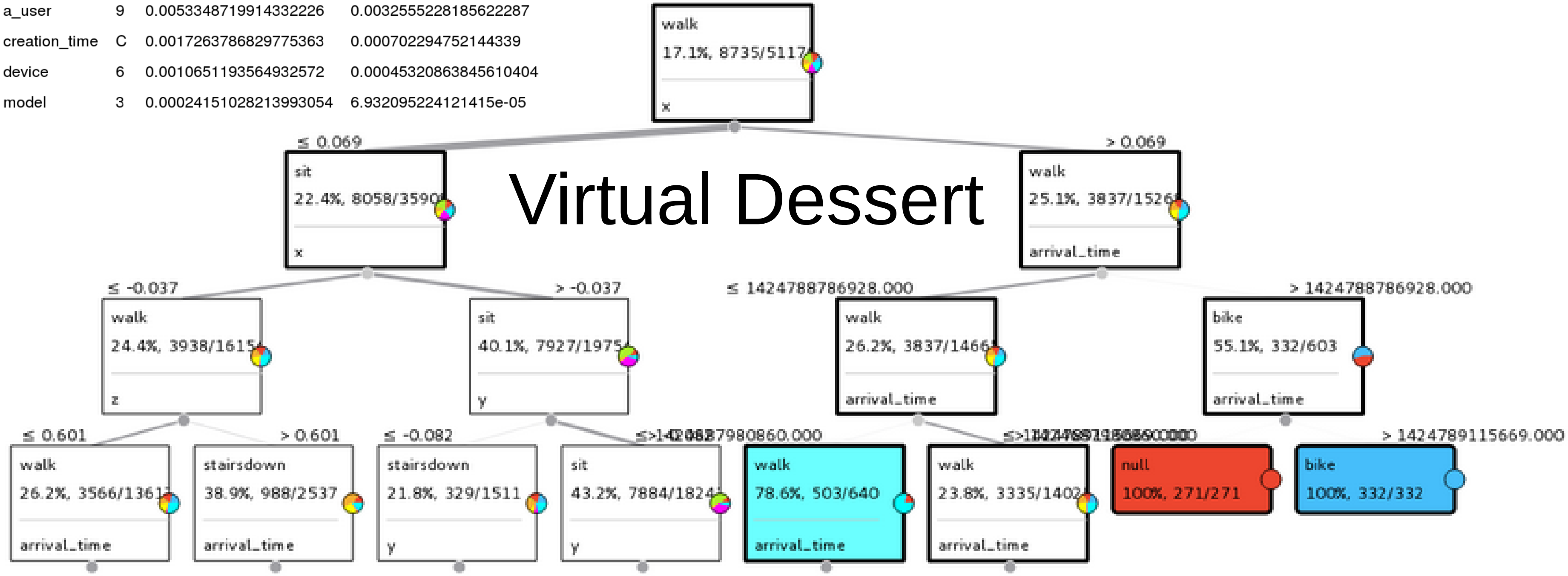
Test metrics

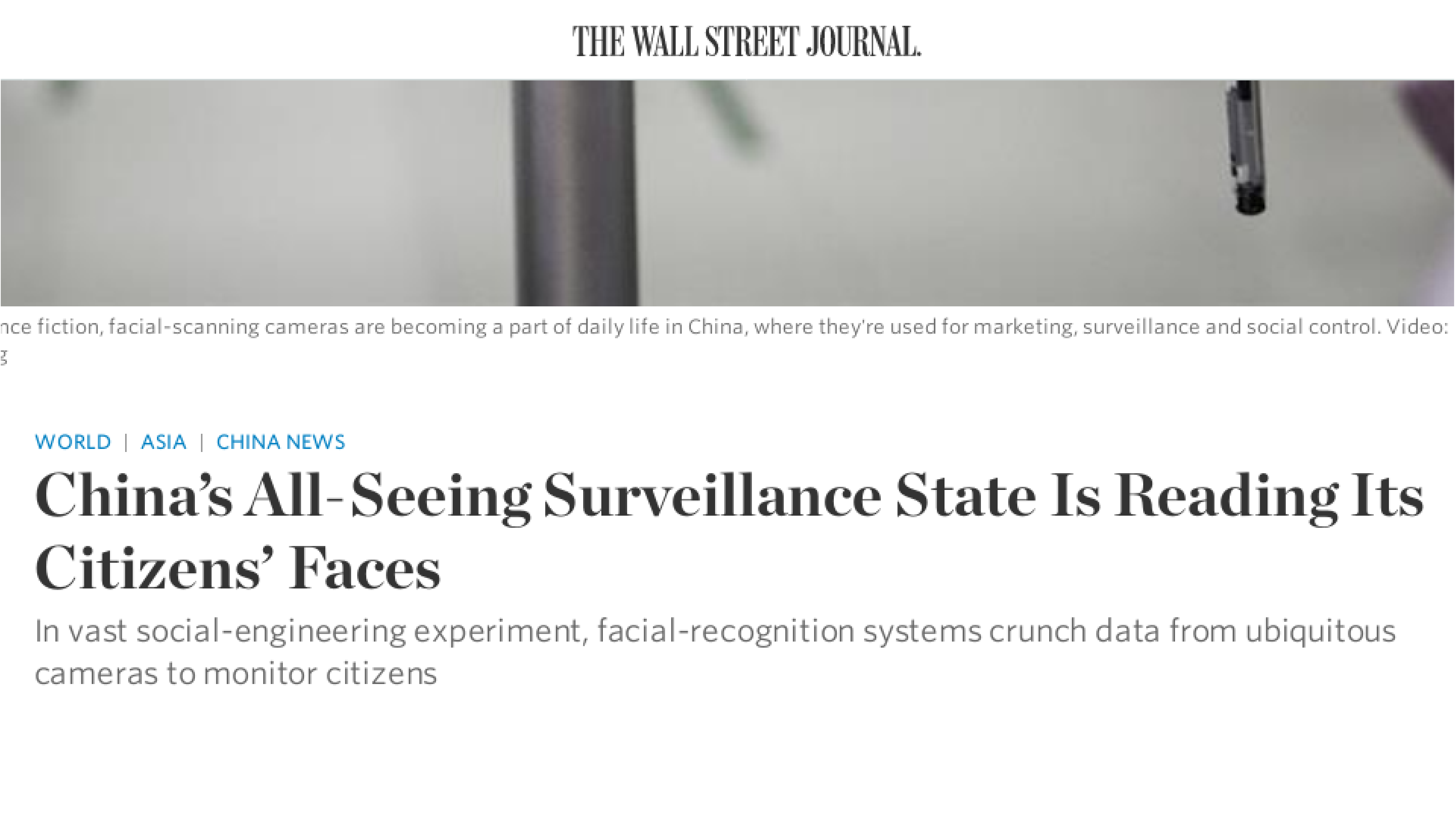
- Since several evaluation metrics can be translated to formulas that fit into SQL

  → U can use them on larger test tables in PostgreSQL than in Orange

Virtual Dessert

	#	Gain Ratio	Gini
x	C	0.2008610816039577	0.06643973736536324
y	C	0.19008980163900854	0.06436997486185836
z	C	0.16244158444581458	0.057384353461334636
arrival_time	C	0.013601482535869014	0.005664211694364152
a_user	9	0.0053348719914332226	0.0032555228185622287
creation_time	C	0.0017263786829775363	0.000702294752144339
device	6	0.0010651193564932572	0.00045320863845610404
model	3	0.00024151028213993054	6.932095224121415e-05





THE WALL STREET JOURNAL.

nce fiction, facial-scanning cameras are becoming a part of daily life in China, where they're used for marketing, surveillance and social control. Video:
g

WORLD | ASIA | CHINA NEWS

China's All-Seeing Surveillance State Is Reading Its Citizens' Faces

In vast social-engineering experiment, facial-recognition systems crunch data from ubiquitous cameras to monitor citizens

Pictures

- Asciidoc notebook and screenshots, W. Bertels
- Knowlegde management and Business Intelligence slides, S. vanden Broucke
- https://nl.wikipedia.org/wiki/Atomium
- http://drbonnie360.com/post/26932618874/words-with-friends-data-mining
- http://bleacherreport.com/articles/707810-nhl-fashion-faux-pas-the-25-worst-alternate-jerseys-in-hockey-history
- https://www.r-project.org/
- https://www.python.org/
- http://scala-lang.org/
- http://www.123rf.com/photo_11384266_funny-snowman-catches-a-snowflake-christmas-background.html
- http://www.ncl.ac.uk/computing/news/item/tilteddevicecouldpinpointpinnumberforhackersstudyreveals.html

References

- https://www.postgresql.org/docs/current/static/index.html
- https://orange.biolab.si/
- https://github.com/citusdata/cstore_fdw
- https://www.2ndquadrant.com/en/resources/pgpredict-predictive-analytics-postgresql/

Wim Bertels